Update to Content Accepted by SRP

Request to Update Content Reviewed and Accepted by the State Review Panel (SRP)
Proposed changes shall be made available for public review on Texas Education Agency's website for a minimum of seven calendar days prior to approval.

Indicate if the changes in the content were reviewed and accepted by the SRP to determine coverage of the Texas Essential Knowledge and Skills (TEKS), English Language Proficiency Standards (ELPS), or Texas Prekindergarten Guidelines (TPG) by selecting a box below. (**Note**: All request to update editions that do not change content reviewed and accepted by the SRP must be entered on the *Update to Content Not Reviewed by SRP* document.)

☒ TEKS ☐ELPS ☐TPG ☐TEKS and ELPS

Proclamation Year: 2024
Publisher: CEV Multimedia
Subject Area/Course: CTE – Science, Technology, Engineering & Mathematics / Computer Science I

Adopted Program Information:

Title: iCEV Computer Science I (Individual Course) - 1-year online access for 25 students and 1 teacher
ISBN: 9798888640036

Enter the identical Program Title of your identical product that will contain the identical updates.
Identical Program Title: iCEV Computer Science I (Individual Course) - 1-year online access for 25 students and 1 teacher
Identical Program ISBN: 9798888640036

Adopted Component Information

Title: iCEV Computer Science I - Student (1-year online access for 1 student)
ISBN: 8888640036001

Enter the identical component title of your identical product that will contain the identical updates.
Identical Component Title: iCEV Computer Science I - Student (1-year online access for 1 student)
Identical Component ISBN: 8888640036001

Publisher's overall rationale for this update

To increase TEKS coverage percentage to 100%.

Publisher's overall description of the change

New content was added to better address TEKS based on State Review Panel identified deficiencies in the content. Additionally, in the new content items were added which include an interactive coding environment to allow students to practice the coding techniques they are learning in the course.

Update to Content Accepted by SRP

Access Information

Enter access information below to the adopted version of the instructional materials and the proposed new content.

Currently Adopted Content URL:
https://login.icevonline.com/mycourses/ADOCOMPU001?aria_label=Computer%20Science%20I%20%28Post%20Adoption%20Sample%29%20
Currently Adopted Content Username: TXPROC24REBID
Currently Adopted Content Password: iCEVREBID

Proposed Updated Content URL:
https://login.icevonline.com/mycourses/ADOCOMPU002?aria_label=Computer%20Science%20I%20-%20UPDATED%20
Proposed Updated Content Username: TXPROC24REBID
Proposed Updated Content Password: iCEVREBID

Update comparison:

Each change in the component on this form should be documented in the update comparison below. You must submit a separate request for **each component**, not each change. (**Note**: Repeat this section as often as needed by copying and pasting the entire area from the (SE)(Breakout(s)) and (Citation Type(s)) to the dividing line for each change.)

(SE)(Breakout(s)) and (Citation Type(s))
(1)(A)(ii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
STEM Careers: Computer Science I (Slides 9-15),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21639

Description of the specific location and hyperlink to the exact location of the proposed new content
STEM Careers: Computer Science I (Slides 6-17),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22287
In the STEM Careers: Computer Science I PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Computer Science Careers

- Can include:
  - software and app developers
  - video game developers
  - systems architects
  - AI engineers
  - web developers
  - robotics software engineer

9

Software & App Developers

- Build computer system software and application software
- Work with software from the beginning to the end, including:
  - designing
  - developing
  - testing
  - maintaining
  - evaluating

10

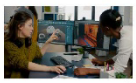
Software & App Developers

- Can start in the workplace with an industry-recognized certification
- Can advance their career with educational degrees

11

Video Game Developers

- Provide code on multiple platforms and systems to run the actual video games with advanced graphics and controls
- Conceptualize with other developers to create playable games
- Can be successful in industry with an industry certification or educational degree



12

Systems Architects

- Design and develop the architecture of a computer system to create the optimal quality user experience
- Analyze systems holistically and create the best possible IT strategy
- Need a bachelor's degree or higher to enter this career path

13

AI Engineers

- Create and test computer systems which demonstrate artificial intelligence (AI)
- Develop systems which intend to mimic human behavior and exhibit the ability to learn
- Need a bachelor's degree or higher to be successful in this career

14

Update to Content Accepted by SRP

Web Developers

- Design and code the layout and all advanced features behind a website
- Review and redesign existing websites to create better user experience and functionality
- Need a bachelor's degree or higher to be successful in this career

15

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Computer Science

- Is the study of computers, including:
  - algorithmic processes
  - hardware
  - software designs
  - computer usage
  - societal impact of computers and technology

6

The Field of Computer Science

- Is suited for students interested in computational processes
- Includes studying computers in theory and application
- Can include areas of interest such as:
  - architecture
  - computational biology
  - programming languages
  - database systems
  - artificial intelligence

7

Computer Science Careers

- Can include:
  - software and app developers
  - video game developers
  - systems architects
  - AI engineers
  - web developers

8

Software & App Developers

- Build computer system software and application software
- Work with software from the beginning to the end, including:
  - designing
  - developing
  - testing
  - maintaining
  - evaluating

9

Update to Content Accepted by SRP

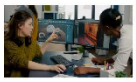
Software & App Developers

- Can start in the workplace with an industry-recognized certification
- Can advance their career with educational degrees

10

Video Game Developers

- Provide code on multiple platforms and systems to run the actual video games with advanced graphics and controls
- Conceptualize with other developers to create playable games
- Can be successful in industry with an industry certification or educational degree



11

Web Developers

- Design and code the layout and all advanced features behind a website
- Review and redesign existing websites to create better user experience and functionality
- Need a bachelor's degree or higher to be successful in this career

14

Other Computer Science Careers

- Include positions which work with computer science professionals, software systems and other technical products, including:
  - technical support specialists
  - data analysts
  - business analysts
- May not require coding, making them ideal entry-level positions
  - internships are another avenue to be introduced to the computer science field

15

Systems Architects

- Design and develop the architecture of a computer system to create the optimal quality user experience
- Analyze systems holistically and create the best possible IT strategy
- Need a bachelor's degree or higher to enter this career path

12

AI Engineers

- Create and test computer systems which demonstrate artificial intelligence (AI)
- Develop systems which intend to mimic human behavior and exhibit the ability to learn
- Need a bachelor's degree or higher to be successful in this career

13

Internships

- Are the position of an individual who works in an organization, sometimes without pay, in order to gain work experience or satisfy requirements for a qualification
  - internships are often offered to students or trainees
- Can be obtained by:
  - determining career goals and interests
  - identifying requirements to meet career goals
  - researching online
  - networking

16

Internships

- In computer science could include:
  - software engineer intern
  - data engineer intern
  - digital and engineering technology intern
  - programming intern
  - software testing intern
  - web and mobile app development intern

17

(SE)(Breakout(s)) and (Citation Type(s))
(1)(H)(ii), Narrative

Update to Content Accepted by SRP

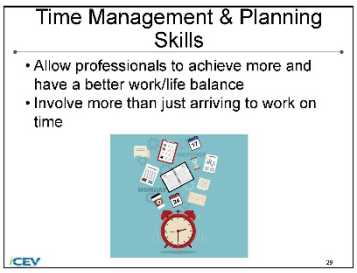
Description of the specific location and hyperlink to the exact location of currently adopted content
Professionalism in the Sciences: Computer Science I (Slides 29-30),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21640

Description of the specific location and hyperlink to the exact location of the proposed new content
Professionalism in the Sciences: Computer Science I (Slides 29-31),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22289
In the Professionalism in the Sciences: Computer Science I PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

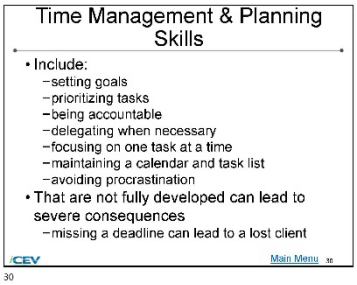
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/20/2024



29

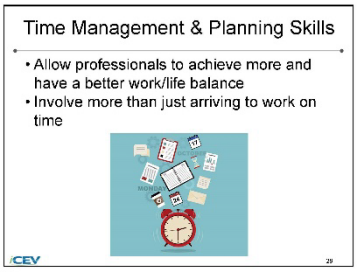


1

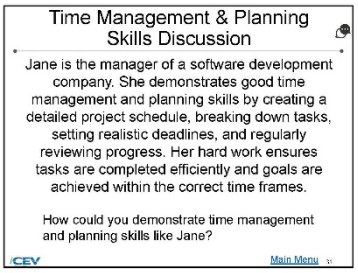
Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

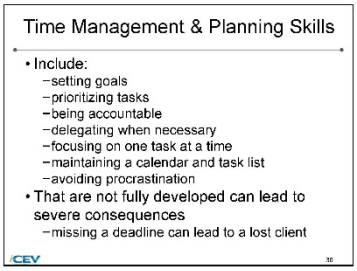
Update to Content Accepted by SRP



29



31



30

(SE)(Breakout(s)) and (Citation Type(s))
(1)(I)(i), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
STEM Careers: Computer Science I (Slides 16-18),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21639
Activity-STEM Careers Exploration,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21639/CEV71506_Activity02

Description of the specific location and hyperlink to the exact location of the proposed new content
STEM Careers: Computer Science I (Slides 18-21),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22287
In the STEM Careers: Computer Science I PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Project-Future STEM Self,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22287/CEV71506_V2_Project01
This Project is found in the STEM Careers: Computer Science I lesson beneath the Interactive Assignments heading. After clicking the link to the Project, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Project.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Robotics Software Engineer

- Design robotic systems, such as drones, industry automation systems, food service machines and more
- Develop algorithms for various robot functions
- Assess needed movement or mobility and test or debug programming as needed
- Need a bachelor's degree or higher to be successful in this career

16

Other Computer Science Careers

- Include positions which work with computer science professionals, software systems and other technical products, including:
  - technical support specialists
  - data analysts
  - business analysts
- May not require coding, making them ideal entry-level positions

17

Computer Science Degrees

- May be required for a career in computer science
- Can include degrees in:
  - information technology
  - information systems
  - cybersecurity
  - software engineering
  - computer engineering
  - electrical engineering
  - networking

18

iCEV

Proclamation 2024 Computer Science I | My Profile | Tutorials | Log Out

LIVE CHAT HELP SCHEDULE ONLINE TRAINING

Computer Science I (Post Adoption Sample)

My Courses / Computer Science I (Post Adoption Sample) / STEM Careers: Computer Science I / Activity - STEM Careers Exploration

Highlight any text to hear text-to-voice speech.

Select Language

SAVE PROGRESS

Activity - STEM Careers Exploration

STEM Careers: Computer Science I

1 of 1

Activity Overview:

You will use the internet to research five careers or internships in computer science to select your favorite. Then you will contact a company or organization to ask questions about your chosen career or internship.

Directions:

1. *Use the internet to research five careers or internships in computer science.*
2. *Fill in the charts for each career or internship you researched including job description, duties or task, education requirements, comparison of relevant post-secondary programs and salary potential.*
3. *Identify your favorite career or internship from the opportunities you researched.*
4. *Contact a company or organization which offers the career or internship you chose. You could contact these individuals through the phone or email. Answer the following questions while you are in contact with the company or organization:*
   - *What are the job duties of your chosen career or internship*
   - *What is the description of your chosen career or internship*
   - *What type of education do they require for your chosen career or internship*
   - *What is the salary or pay for your chosen career or internship*
5. *Once complete, submit your Activity.*

Update to Content Accepted by SRP

Career or Internship 1:

Description	
Job Duties & Tasks	
Education	
Post-Secondary Programs	
Salary or Pay	

Career or Internship 2:

Description	
Job Duties & Tasks	
Education	
Post-Secondary Programs	
Salary or Pay	

Career or Internship 3:

Description	
Job Duties & Tasks	
Education	
Post-Secondary Programs	
Salary or Pay	

Career or Internship 4:

Description	
Job Duties & Tasks	
Education	
Post-Secondary Programs	
Salary or Pay	

Update to Content Accepted by SRP

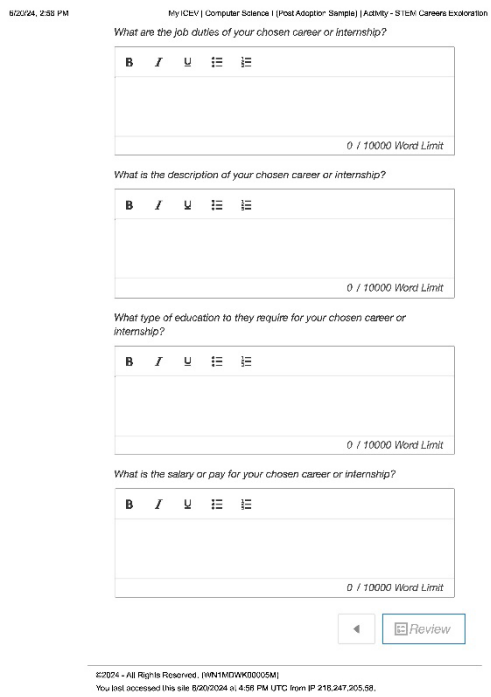
6/20/24, 2:58 PM My ICEV | Computer Science I (Post Adoption Sample) | Activity - STEM Careers Exploration

Career or Internship 5:

Description	
Job Duties & Tasks	
Education	
Post-Secondary Programs	
Salary or Pay	

Identify your favorite career or internship you researched here.

https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21639/CEV71506_Activity02 7/9


6/20/24, 2:58 PM My ICEV | Computer Science I (Post Adoption Sample) | Activity - STEM Careers Exploration

What are the job duties of your chosen career or internship?

0 / 10000 Word Limit

What is the description of your chosen career or internship?

0 / 10000 Word Limit

What type of education to they require for your chosen career or internship?

0 / 10000 Word Limit

What is the salary or pay for your chosen career or internship?

0 / 10000 Word Limit

Review

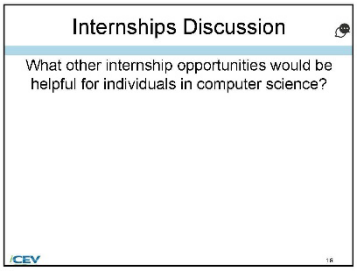
©2024 - All Rights Reserved. [WN1MDWK00005M]
You last accessed this site 6/20/2024 at 4:58 PM UTC from IP 218.247.205.58.

https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21639/CEV71506_Activity02 8/9

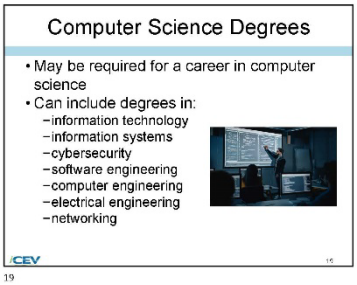
Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

6/20/2024



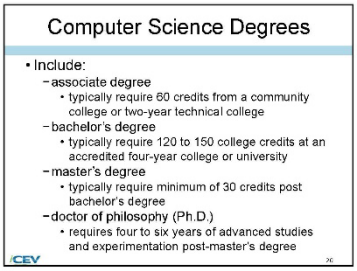
18



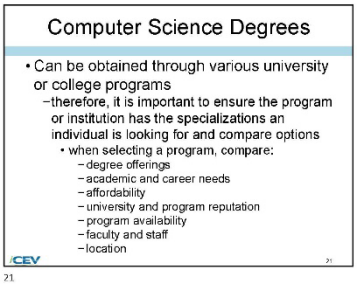
19

1

6/20/2024



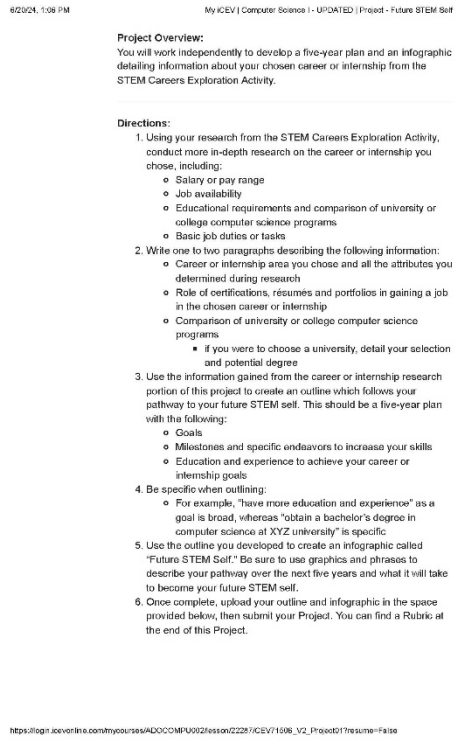
20

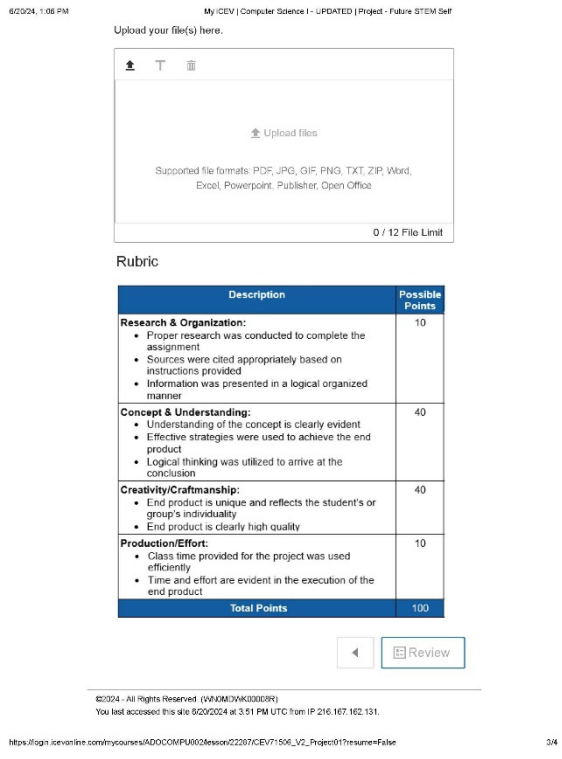


21

2

Update to Content Accepted by SRP





(SE)(Breakout(s)) and (Citation Type(s))

(2)(A)(ii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content

Update to Content Accepted by SRP

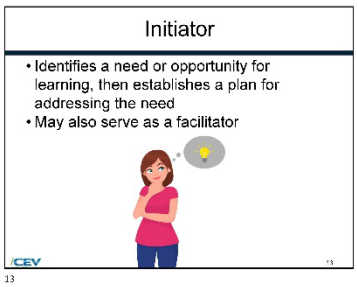
Learning Communities (Slide 13),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21641

Description of the specific location and hyperlink to the exact location of the proposed new content
Learning Communities (Slides 3-13, 18-23),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22291
In the Learning Communities PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/20/2024



13

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Update to Content Accepted by SRP

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Roles

- Are important for a learning community to function effectively
- Include the following:
  - initiator
  - facilitator
  - learner
  - contributor
  - teacher/mentor

Fun Fact: Learning communities do not have to be all work and no play; the topic could be something fun like cooking or Star Wars.

12

Seeking Learning Community Advice

- Begins with submitting a product for review online or in-person to peers, educators or professionals to evaluate quality and accuracy
  - a request to review should be sent or discussed prior
  - reviewers should be qualified
  - can be a product submitted by an individual to the learning community or submitted by the learning community as a group to an outside source

19

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

20

Initiator

- Identifies a need or opportunity for learning, then establishes a plan for addressing the need
- Participates in learning communities by beginning new projects, organizing events or brainstorming new ways of thinking
- May also serve as a facilitator

13

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners
- Can improve a product's quality and accuracy and provide perspective based on the individual's role

18

Responding to a Request for Advice

- Within learning communities can vary depending on the product submitted
- Should be accepted or denied in a timely manner
- Usually begins with an acknowledgment of what was done well and transitions to what could be improved
  - advice should always be respectful and display constructive criticism
  - review should evaluate accuracy and quality of a student project

21

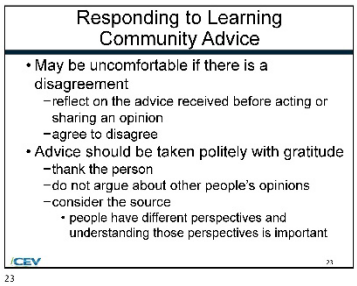
Providing Learning Community Advice

- Tips include:
  - reviewing thoroughly
    - be specific with edits
    - offer solutions when applicable
  - constructive criticism
    - positive framing for improvement is more likely to be received and implemented rather than negative callouts of the flaws
    - highlight any positive aspects completed
  - clear and concise language
    - ensure the feedback is easily understood

22

Update to Content Accepted by SRP

6/20/2024



(SE)(Breakout(s)) and (Citation Type(s))
(2)(A)(iii), Narrative

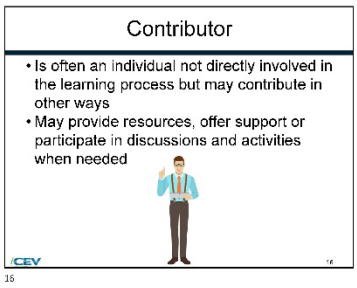
Description of the specific location and hyperlink to the exact location of currently adopted content
Learning Communities (Slide 16),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21641

Description of the specific location and hyperlink to the exact location of the proposed new content
Learning Communities (Slides 3-12, 18-23),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22291
In the Learning Communities PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/20/2024



Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Update to Content Accepted by SRP

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Contributor

- Is often an individual not directly involved in the learning process but may contribute in other ways
- May provide resources, offer support or participate in discussions and activities when needed in a learning community
  - for example, a peer assisting to install a new software and share their expertise, asking questions to initiate discussion, providing feedback, and collaborating on open-source projects

16

Roles

- Are important for a learning community to function effectively
- Include the following:
  - initiator
  - facilitator
  - learner
  - contributor
  - teacher/mentor

Fun Fact: Learning communities do not have to be all work and no play; the topic could be something fun like cooking or Star Wars.

12

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners
- Can improve a product's quality and accuracy and provide perspective based on the individual's role

18

Seeking Learning Community Advice

- Begins with submitting a product for review online or in-person to peers, educators or professionals to evaluate quality and accuracy
  - a request to review should be sent or discussed prior
  - reviewers should be qualified
  - can be a product submitted by an individual to the learning community or submitted by the learning community as a group to an outside source

19

Responding to a Request for Advice

- Within learning communities can vary depending on the product submitted
- Should be accepted or denied in a timely manner
- Usually begins with an acknowledgment of what was done well and transitions to what could be improved
  - advice should always be respectful and display constructive criticism
  - review should evaluate accuracy and quality of a student project

21

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

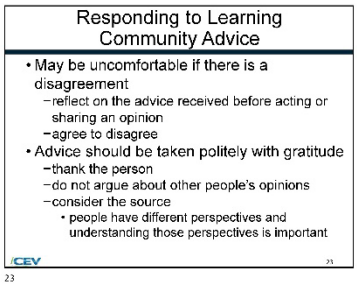
20

Providing Learning Community Advice

- Tips include:
  - reviewing thoroughly
    - be specific with edits
    - offer solutions when applicable
  - constructive criticism
    - positive framing for improvement is more likely to be received and implemented rather than negative callouts of the flaws
    - highlight any positive aspects completed
  - clear and concise language
    - ensure the feedback is easily understood

22

Update to Content Accepted by SRP



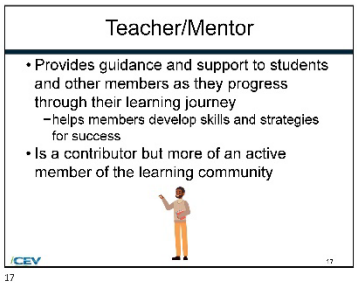
(SE)(Breakout(s)) and (Citation Type(s))
(2)(A)(iv), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Learning Communities (Slide 17),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21641

Description of the specific location and hyperlink to the exact location of the proposed new content
Learning Communities (Slides 3-12, 17-23),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22291
In the Learning Communities PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Update to Content Accepted by SRP

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Teacher/Mentor

- Provides guidance and support to students, other members and teachers as they progress through their learning journey
  - helps members develop skills and strategies for success by:
    - providing expertise to reach a specific goal or project completion
    - networking to discuss industry trends, coding resources, activities and projects, troubleshooting, and effective strategies

- Is a contributor but more of an active member of the learning community

17

Roles

- Are important for a learning community to function effectively
- Include the following:
  - initiator
  - facilitator
  - learner
  - contributor
  - teacher/mentor

Fun Fact: Learning communities do not have to be all work and no play; the topic could be something fun like cooking or Star Wars.

12

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners
- Can improve a product's quality and accuracy and provide perspective based on the individual's role

18

Seeking Learning Community Advice

- Begins with submitting a product for review online or in-person to peers, educators or professionals to evaluate quality and accuracy
  - a request to review should be sent or discussed prior
  - reviewers should be qualified
  - can be a product submitted by an individual to the learning community or submitted by the learning community as a group to an outside source

19

Responding to a Request for Advice

- Within learning communities can vary depending on the product submitted
- Should be accepted or denied in a timely manner
- Usually begins with an acknowledgment of what was done well and transitions to what could be improved
  - advice should always be respectful and display constructive criticism
  - review should evaluate accuracy and quality of a student project

21

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

20

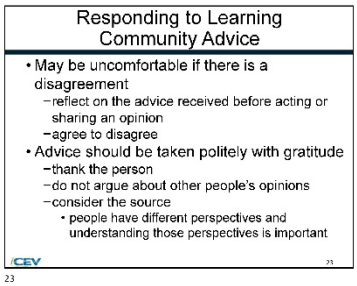
Providing Learning Community Advice

- Tips include:
  - reviewing thoroughly
    - be specific with edits
    - offer solutions when applicable
  - constructive criticism
    - positive framing for improvement is more likely to be received and implemented rather than negative callouts of the flaws
    - highlight any positive aspects completed
  - clear and concise language
    - ensure the feedback is easily understood

22

Update to Content Accepted by SRP

6/20/2024



(SE)(Breakout(s)) and (Citation Type(s))
(2)(A)(v), Narrative

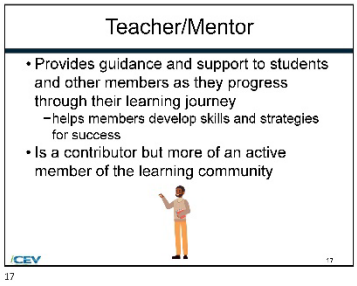
Description of the specific location and hyperlink to the exact location of currently adopted content
Learning Communities (Slide 17),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21641

Description of the specific location and hyperlink to the exact location of the proposed new content
Learning Communities (Slides 3-12, 17-23),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22291
In the Learning Communities PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/20/2024



Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Update to Content Accepted by SRP

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Roles

- Are important for a learning community to function effectively
- Include the following:
  - initiator
  - facilitator
  - learner
  - contributor
  - teacher/mentor

Fun Fact: Learning communities do not have to be all work and no play; the topic could be something fun like cooking or Star Wars.

12

Seeking Learning Community Advice

- Begins with submitting a product for review online or in-person to peers, educators or professionals to evaluate quality and accuracy
  - a request to review should be sent or discussed prior
  - reviewers should be qualified
  - can be a product submitted by an individual to the learning community or submitted by the learning community as a group to an outside source

19

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

20

Teacher/Mentor

- Provides guidance and support to students, other members and teachers as they progress through their learning journey
  - helps members develop skills and strategies for success by:
    - providing expertise to reach a specific goal or project completion
    - networking to discuss industry trends, coding resources, activities and projects, troubleshooting, and effective strategies

- Is a contributor but more of an active member of the learning community

17

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners
- Can improve a product's quality and accuracy and provide perspective based on the individual's role

18

Responding to a Request for Advice

- Within learning communities can vary depending on the product submitted
- Should be accepted or denied in a timely manner
- Usually begins with an acknowledgment of what was done well and transitions to what could be improved
  - advice should always be respectful and display constructive criticism
  - review should evaluate accuracy and quality of a student project

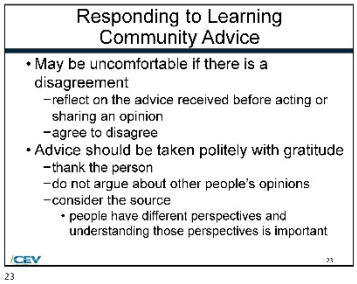
21

Providing Learning Community Advice

- Tips include:
  - reviewing thoroughly
    - be specific with edits
    - offer solutions when applicable
  - constructive criticism
    - positive framing for improvement is more likely to be received and implemented rather than negative callouts of the flaws
    - highlight any positive aspects completed
  - clear and concise language
    - ensure the feedback is easily understood

22

Update to Content Accepted by SRP



(SE)(Breakout(s)) and (Citation Type(s))
(2)(B)(i), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Learning Communities (Slides 3-11, 18-20),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21641

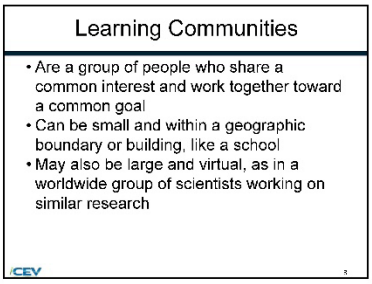
Description of the specific location and hyperlink to the exact location of the proposed new content
Learning Communities (Slides 3-23),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22291
In the Learning Communities PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

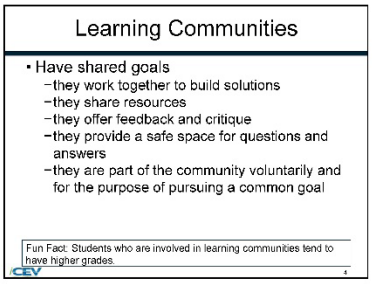
Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

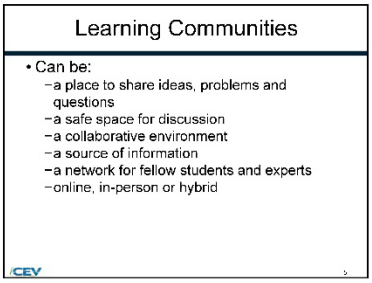
6/20/2024



3

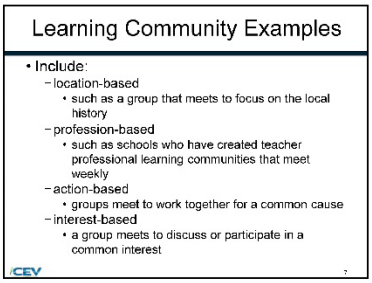


4

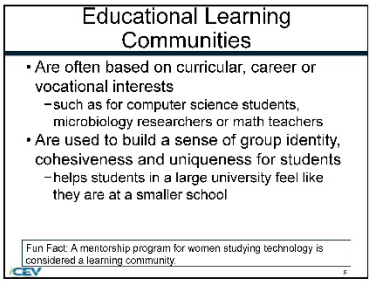


5

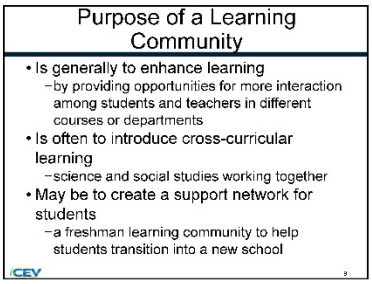
6


Learning Community Examples
• Include:
– location-based
• such as a group that meets to focus on the local history
– profession-based
• such as schools who have created teacher professional learning communities that meet weekly
– action-based
• groups meet to work together for a common cause
– interest-based
• a group meets to discuss or participate in a common interest
CEV
7

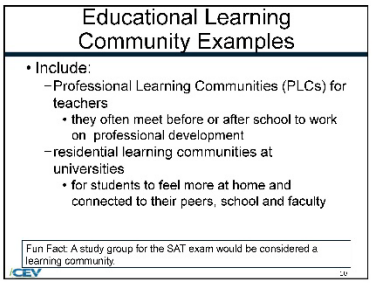
7


Educational Learning Communities
• Are often based on curricular, career or vocational interests
– such as for computer science students, microbiology researchers or math teachers
• Are used to build a sense of group identity, cohesiveness and uniqueness for students
– helps students in a large university feel like they are at a smaller school
Fun Fact: A mentorship program for women studying technology is considered a learning community.
CEV
8

8


Purpose of a Learning Community
• Is generally to enhance learning
– by providing opportunities for more interaction among students and teachers in different courses or departments
• Is often to introduce cross-curricular learning
– science and social studies working together
• May be to create a support network for students
– a freshman learning community to help students transition into a new school
CEV
9

9


Educational Learning Community Examples
• Include:
– Professional Learning Communities (PLCs) for teachers
• they often meet before or after school to work on professional development
– residential learning communities at universities
• for students to feel more at home and connected to their peers, school and faculty
Fun Fact: A study group for the SAT exam would be considered a learning community.
CEV
10

10

Update to Content Accepted by SRP

6/20/2024

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners

18

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

19

Responding to Learning Community Advice

- May be uncomfortable if there is a disagreement
  - reflect on the advice received before acting or sharing an opinion
  - agree to disagree
- Advice should be taken politely with gratitude
  - thank the person
  - do not argue about other people's opinions
  - consider the source
    - people have different perspectives and understanding those perspectives is important

20

3

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Update to Content Accepted by SRP

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Roles

- Are important for a learning community to function effectively
- Include the following:
  - initiator
  - facilitator
  - learner
  - contributor
  - teacher/mentor

Fun Fact: Learning communities do not have to be all work and no play; the topic could be something fun like cooking or Star Wars.

12

Learner

- Is the primary participant in the group
- Is expected to actively engage in the learning process by asking questions and participating in discussions and activities

15

Contributor

- Is often an individual not directly involved in the learning process but may contribute in other ways
- May provide resources, offer support or participate in discussions and activities when needed in a learning community

for example, a peer assisting to install a new software and share their expertise, asking questions to initiate discussion, providing feedback, and collaborating on open-source projects

16

Initiator

- Identifies a need or opportunity for learning, then establishes a plan for addressing the need
- Participates in learning communities by beginning new projects, organizing events or brainstorming new ways of thinking
- May also serve as a facilitator

13

Facilitator

- Guides, or facilitates the learning process
- Is usually an expert or professional on the topic the group is learning about
- Provides support and guidance to the members of the group

Fun Fact: Students involved in learning communities make great contacts and have a valid activity to add to their résumés.

14

Teacher/Mentor

- Provides guidance and support to students, other members and teachers as they progress through their learning journey
  - helps members develop skills and strategies for success by:
    - providing expertise to reach a specific goal or project completion
    - networking to discuss industry trends, coding resources, activities and projects, troubleshooting, and effective strategies

- Is a contributor but more of an active member of the learning community

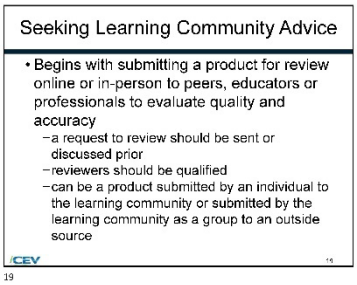
17

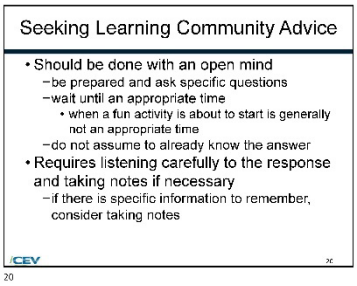
Learning Community Advice

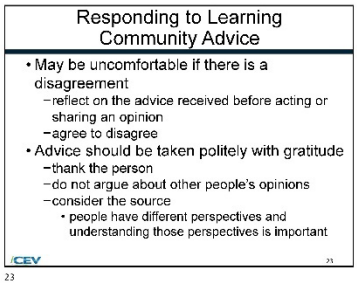
- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners
- Can improve a product's quality and accuracy and provide perspective based on the individual's role

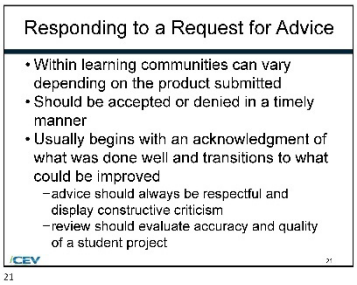
18

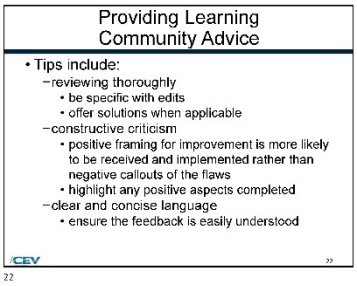
Update to Content Accepted by SRP











(SE)(Breakout(s)) and (Citation Type(s))
(2)(B)(ii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Learning Communities (Slides 3-11, 18-20),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21641

Description of the specific location and hyperlink to the exact location of the proposed new content
Learning Communities (Slides 3-23),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22291
In the Learning Communities PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Update to Content Accepted by SRP

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Update to Content Accepted by SRP

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners

18

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

19

Responding to Learning Community Advice

- May be uncomfortable if there is a disagreement
  - reflect on the advice received before acting or sharing an opinion
  - agree to disagree
- Advice should be taken politely with gratitude
  - thank the person
  - do not argue about other people's opinions
  - consider the source
    - people have different perspectives and understanding those perspectives is important

20

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Update to Content Accepted by SRP

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Roles

- Are important for a learning community to function effectively
- Include the following:
  - initiator
  - facilitator
  - learner
  - contributor
  - teacher/mentor

Fun Fact: Learning communities do not have to be all work and no play; the topic could be something fun like cooking or Star Wars.

12

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Initiator

- Identifies a need or opportunity for learning, then establishes a plan for addressing the need
- Participates in learning communities by beginning new projects, organizing events or brainstorming new ways of thinking
- May also serve as a facilitator

13

Facilitator

- Guides, or facilitates the learning process
- Is usually an expert or professional on the topic the group is learning about
- Provides support and guidance to the members of the group

Fun Fact: Students involved in learning communities make great contacts and have a valid activity to add to their resumés.

14

Update to Content Accepted by SRP

Learner

- Is the primary participant in the group
- Is expected to actively engage in the learning process by asking questions and participating in discussions and activities

15

Contributor

- Is often an individual not directly involved in the learning process but may contribute in other ways
- May provide resources, offer support or participate in discussions and activities when needed in a learning community

for example, a peer assisting to install a new software and share their expertise, asking questions to initiate discussion, providing feedback, and collaborating on open-source projects

16

Seeking Learning Community Advice

- Begins with submitting a product for review online or in-person to peers, educators or professionals to evaluate quality and accuracy
  - a request to review should be sent or discussed prior
  - reviewers should be qualified
  - can be a product submitted by an individual to the learning community or submitted by the learning community as a group to an outside source

19

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

20

Teacher/Mentor

- Provides guidance and support to students, other members and teachers as they progress through their learning journey
  - helps members develop skills and strategies for success by:
    - providing expertise to reach a specific goal or project completion
    - networking to discuss industry trends, coding resources, activities and projects, troubleshooting, and effective strategies

- Is a contributor but more of an active member of the learning community

17

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners
- Can improve a product's quality and accuracy and provide perspective based on the individual's role

18

Responding to a Request for Advice

- Within learning communities can vary depending on the product submitted
- Should be accepted or denied in a timely manner
- Usually begins with an acknowledgment of what was done well and transitions to what could be improved
  - advice should always be respectful and display constructive criticism
  - review should evaluate accuracy and quality of a student project

21

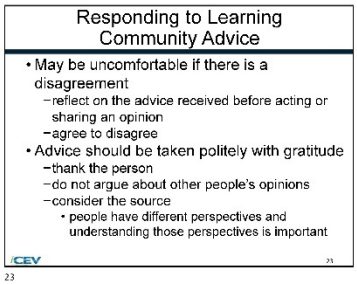
Providing Learning Community Advice

- Tips include:
  - reviewing thoroughly
    - be specific with edits
    - offer solutions when applicable
  - constructive criticism
    - positive framing for improvement is more likely to be received and implemented rather than negative callouts of the flaws
    - highlight any positive aspects completed
  - clear and concise language
    - ensure the feedback is easily understood

22

Update to Content Accepted by SRP

6/20/2024



(SE)(Breakout(s)) and (Citation Type(s))
(2)(B)(iii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Learning Communities (Slides 3-11, 18-20),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21641

Description of the specific location and hyperlink to the exact location of the proposed new content
Learning Communities (Slides 3-23),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22291
In the Learning Communities PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Update to Content Accepted by SRP

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners

18

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

19

Responding to Learning Community Advice

- May be uncomfortable if there is a disagreement
  - reflect on the advice received before acting or sharing an opinion
  - agree to disagree
- Advice should be taken politely with gratitude
  - thank the person
  - do not argue about other people's opinions
  - consider the source
    - people have different perspectives and understanding those perspectives is important

20

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Update to Content Accepted by SRP

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Roles

- Are important for a learning community to function effectively
- Include the following:
  - initiator
  - facilitator
  - learner
  - contributor
  - teacher/mentor

Fun Fact: Learning communities do not have to be all work and no play; the topic could be something fun like cooking or Star Wars.

12

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Initiator

- Identifies a need or opportunity for learning, then establishes a plan for addressing the need
- Participates in learning communities by beginning new projects, organizing events or brainstorming new ways of thinking
- May also serve as a facilitator

13

Facilitator

- Guides, or facilitates the learning process
- Is usually an expert or professional on the topic the group is learning about
- Provides support and guidance to the members of the group

Fun Fact: Students involved in learning communities make great contacts and have a valid activity to add to their resumés.

14

Update to Content Accepted by SRP

Learner

- Is the primary participant in the group
- Is expected to actively engage in the learning process by asking questions and participating in discussions and activities

15

Contributor

- Is often an individual not directly involved in the learning process but may contribute in other ways
- May provide resources, offer support or participate in discussions and activities when needed in a learning community

for example, a peer assisting to install a new software and share their expertise, asking questions to initiate discussion, providing feedback, and collaborating on open-source projects

16

Seeking Learning Community Advice

- Begins with submitting a product for review online or in-person to peers, educators or professionals to evaluate quality and accuracy
  - a request to review should be sent or discussed prior
  - reviewers should be qualified
  - can be a product submitted by an individual to the learning community or submitted by the learning community as a group to an outside source

19

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

20

Teacher/Mentor

- Provides guidance and support to students, other members and teachers as they progress through their learning journey
  - helps members develop skills and strategies for success by:
    - providing expertise to reach a specific goal or project completion
    - networking to discuss industry trends, coding resources, activities and projects, troubleshooting, and effective strategies

- Is a contributor but more of an active member of the learning community

17

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners
- Can improve a product's quality and accuracy and provide perspective based on the individual's role

18

Responding to a Request for Advice

- Within learning communities can vary depending on the product submitted
- Should be accepted or denied in a timely manner
- Usually begins with an acknowledgment of what was done well and transitions to what could be improved
  - advice should always be respectful and display constructive criticism
  - review should evaluate accuracy and quality of a student project

21

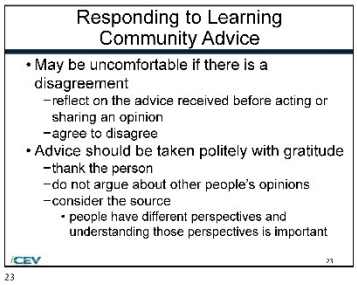
Providing Learning Community Advice

- Tips include:
  - reviewing thoroughly
    - be specific with edits
    - offer solutions when applicable
  - constructive criticism
    - positive framing for improvement is more likely to be received and implemented rather than negative callouts of the flaws
    - highlight any positive aspects completed
  - clear and concise language
    - ensure the feedback is easily understood

22

Update to Content Accepted by SRP

6/20/2024



(SE)(Breakout(s)) and (Citation Type(s))
(2)(B)(iv), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Learning Communities (Slides 3-11, 18-20),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21641

Description of the specific location and hyperlink to the exact location of the proposed new content
Learning Communities (Slides 3-23),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22291
In the Learning Communities PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Update to Content Accepted by SRP

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners

18

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

19

Responding to Learning Community Advice

- May be uncomfortable if there is a disagreement
  - reflect on the advice received before acting or sharing an opinion
  - agree to disagree
- Advice should be taken politely with gratitude
  - thank the person
  - do not argue about other people's opinions
  - consider the source
    - people have different perspectives and understanding those perspectives is important

20

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Learning Communities

- Are a group of people who share a common interest and work together toward a common goal
- Can be small and within a geographic boundary or building, like a school
- May also be large and virtual, as in a worldwide group of scientists working on similar research

3

Learning Communities

- Have shared goals
  - they work together to build solutions
  - they share resources
  - they offer feedback and critique
  - they provide a safe space for questions and answers
  - they are part of the community voluntarily and for the purpose of pursuing a common goal

Fun Fact: Students who are involved in learning communities tend to have higher grades.

4

Learning Communities

- Can be:
  - a place to share ideas, problems and questions
  - a safe space for discussion
  - a collaborative environment
  - a source of information
  - a network for fellow students and experts
  - online, in-person or hybrid

5

Learning Communities

- Are not:
  - a school or professional committee
    - committee members are usually appointed to complete a specific task
  - a social group
    - there is a deeper connection over a shared goal and topic in a learning community
  - a job board or marketing platform
  - a workshop or class
  - a social media platform

Fun Fact: Learning communities are a great way to practice soft skills like collaboration, communication and problem-solving.

6

Update to Content Accepted by SRP

Learning Community Examples

- Include:
  - location-based
    - such as a group that meets to focus on the local history
  - profession-based
    - such as schools who have created teacher professional learning communities that meet weekly
  - action-based
    - groups meet to work together for a common cause
  - interest-based
    - a group meets to discuss or participate in a common interest

7

Educational Learning Communities

- Are often based on curricular, career or vocational interests
  - such as for computer science students, microbiology researchers or math teachers
- Are used to build a sense of group identity, cohesiveness and uniqueness for students
  - helps students in a large university feel like they are at a smaller school

Fun Fact: A mentorship program for women studying technology is considered a learning community.

8

Effective Learning Communities

- Have the following components:
  - a shared sense of purpose
  - active participation
  - support and feedback
  - collaboration and teamwork
  - inclusivity and diversity

11

Roles

- Are important for a learning community to function effectively
- Include the following:
  - initiator
  - facilitator
  - learner
  - contributor
  - teacher/mentor

Fun Fact: Learning communities do not have to be all work and no play; the topic could be something fun like cooking or Star Wars.

12

Purpose of a Learning Community

- Is generally to enhance learning
  - by providing opportunities for more interaction among students and teachers in different courses or departments
- Is often to introduce cross-curricular learning
  - science and social studies working together
- May be to create a support network for students
  - a freshman learning community to help students transition into a new school

9

Educational Learning Community Examples

- Include:
  - Professional Learning Communities (PLCs) for teachers
    - they often meet before or after school to work on professional development
  - residential learning communities at universities
    - for students to feel more at home and connected to their peers, school and faculty

Fun Fact: A study group for the SAT exam would be considered a learning community.

10

Initiator

- Identifies a need or opportunity for learning, then establishes a plan for addressing the need
- Participates in learning communities by beginning new projects, organizing events or brainstorming new ways of thinking
- May also serve as a facilitator

13

Facilitator

- Guides, or facilitates the learning process
- Is usually an expert or professional on the topic the group is learning about
- Provides support and guidance to the members of the group

Fun Fact: Students involved in learning communities make great contacts and have a valid activity to add to their resumés.

14

Update to Content Accepted by SRP

Learner

- Is the primary participant in the group
- Is expected to actively engage in the learning process by asking questions and participating in discussions and activities

15

Contributor

- Is often an individual not directly involved in the learning process but may contribute in other ways
- May provide resources, offer support or participate in discussions and activities when needed in a learning community

for example, a peer assisting to install a new software and share their expertise, asking questions to initiate discussion, providing feedback, and collaborating on open-source projects

16

Seeking Learning Community Advice

- Begins with submitting a product for review online or in-person to peers, educators or professionals to evaluate quality and accuracy
  - a request to review should be sent or discussed prior
  - reviewers should be qualified
  - can be a product submitted by an individual to the learning community or submitted by the learning community as a group to an outside source

19

Seeking Learning Community Advice

- Should be done with an open mind
  - be prepared and ask specific questions
  - wait until an appropriate time
    - when a fun activity is about to start is generally not an appropriate time
  - do not assume to already know the answer
- Requires listening carefully to the response and taking notes if necessary
  - if there is specific information to remember, consider taking notes

20

Teacher/Mentor

- Provides guidance and support to students, other members and teachers as they progress through their learning journey
  - helps members develop skills and strategies for success by:
    - providing expertise to reach a specific goal or project completion
    - networking to discuss industry trends, coding resources, activities and projects, troubleshooting, and effective strategies

- Is a contributor but more of an active member of the learning community

17

Learning Community Advice

- Is one advantage of being part of a group of people with similar interests
- May come from professionals who are contributors or facilitators
- May come from educators who are teachers and mentors
- May come from peers or other learners
- Can improve a product's quality and accuracy and provide perspective based on the individual's role

18

Responding to a Request for Advice

- Within learning communities can vary depending on the product submitted
- Should be accepted or denied in a timely manner
- Usually begins with an acknowledgment of what was done well and transitions to what could be improved
  - advice should always be respectful and display constructive criticism
  - review should evaluate accuracy and quality of a student project

21

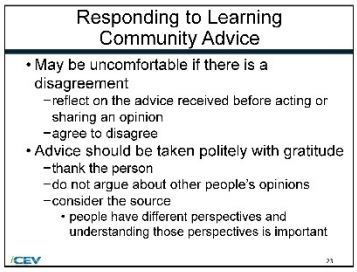
Providing Learning Community Advice

- Tips include:
  - reviewing thoroughly
    - be specific with edits
    - offer solutions when applicable
  - constructive criticism
    - positive framing for improvement is more likely to be received and implemented rather than negative callouts of the flaws
    - highlight any positive aspects completed
  - clear and concise language
    - ensure the feedback is easily understood

22

Update to Content Accepted by SRP

6/20/2024



(SE)(Breakout(s)) and (Citation Type(s))
(3)(A)(i), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Visual Presentation (Slides 7-18, 21-28),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21653

Description of the specific location and hyperlink to the exact location of the proposed new content
Visual Presentation Student Handout-Inputs, Outputs and Data Displays Examples,
https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm
This Student Handout is found in the Visual Presentation lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Escape Sequences

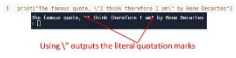
- Are a sequence of characters used to output special characters
  - forced return, tab and single or double quotes
- Contain string data, a sequence of characters
  - it might be the case the program will need to force a key press from the keyboard
    - the programmer must include a command that instructs the computer to perform the operation

Escape Sequence Example

- May include:
  - if the print statement is too long and needs to be broken up when displayed, \n would be used to indicate 'go to the next line' or force a return
  - to display "Hello" on a separate line from "World" use print("Hello\nWorld")

Quotation Marks

- Which need to appear in the output can be done by using \"
  - when there is no backslash before the quotation mark, they do not show up in the output



Character Sequences

- Can include:

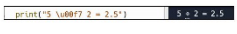
Tag	Description
\'	Single quote. Example: print('She said,\'Hello\' today")
\"	Double quote. Example: print("She said, \"Hello \" today")
\n	New line. Example: print("Hello\nWorld")
\\	Backslash. Example: print("__/\\") will print /__/

Unicode Character Encoding

- Is an extension of ASCII (American Standard Code for Information Interchange)
  - allows programmers to include a multitude of symbols and characters in all languages
  - the tag is generally \u followed by the Unicode value

Unicode Character Example

- Can include:
  - \u00f7, which is a symbol that denotes the divided by character
  - in the code editor, type print("5 \u00f7 2 = 2.5")



Main Menu

Dialog Boxes

- Are Graphic User Interfaces like tkinter class
  - tkinter is a common Python graphics package
  - to display text as output in a dialog box in Python, a class called tkinter.messagebox must be imported
    - this is a separate program

Tkinter Package

- Once imported can use the showinfo method
  - takes two arguments, the title of the box and the text that will be displayed in the box
  - the code would start with the keyword import followed tkinter.messagebox

Coding Corner: If the command is followed with the keyword 'as' the class is given a nickname like tk, that you can use.

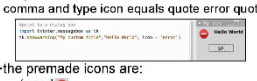
Update to Content Accepted by SRP

The Showinfo Method

- Will provide a basic dialog box, a title on the top of the box, a lightbulb icon on the left and an OK button
  - to achieve this box type in the command tk.showinfo() in all lowercase letters
    - two arguments can be added separated by a comma and both written with quotation marks surrounding each of the string arguments
    - the first argument is the title on the box and the second argument is the text in the box
      - tk.showinfo("My pop up box" , "Hello World")

The Showinfo Method

- Can include a third argument to change the special icon
  - for example, adding a third parameter inside of the parentheses after "Hello World" insert a comma and type icon equals quote error quote



  - the premade icons are:
    - 'error'
    - 'info'
    - 'question'
    - 'warning'

Printing Text to a File

- Includes:
  - adding a comma before entering the second argument
  - the second argument specifies the type of file as read or write in quotation marks
    - lowercase letter w denotes a write file and the lowercase letter r indicates a read only file
      - the file can be named for example 'myfile'

Printing Text to a File

- Includes:
  - hit enter after typing this line
  - the file is now ready to be printed
    - use the print command, but add a second argument specifying the text wanted in the file
      - print("Hello World", file = myfile)

Printing Text to a File

- Can be done by importing the sys package, opening a 'write' file and printing directly to the file
  - the process involves the commands 'with', 'open' and file

Printing Text to a File

- Starts by typing import sys, then hit enter to advance to the next line
  - the next line of code will start with the word 'with' followed by a space and the word open
  - open is a command that takes two arguments so after the word open, type the opening parenthesis
    - the first argument is the full name of the file and must be in quotations

Programing Languages

- Provide commands allowing programmers to print words and phrases to the computer monitor
  - text can be delivered from a computer to the world in different ways and differ depending on the programing languages

Printing Display Text

- Can be done by using the programing language Python
  - must have the Python software downloaded and a developmental environment
    - examples of integrated development environments are jGrasp, Eclipse and Visual Studio Code
    - can also use an online environment which does not need to be downloaded such as repl.it

Coding Corner: There are around 700 separate programing languages.

Update to Content Accepted by SRP

Printing Display Text

- Can be done using repl.it
  - users must create an account to save programs
    - can be done by clicking the button in the upper right corner that reads, "Sign up"
    - to begin a new program, click on the 'Create' button on the left of the page which will display the Template and Title dialog boxes

9

An Integrated Development Environment

- Should be displayed on the screen
  - file browser
  - code editor
  - console where output will display

11

Printing Display Text

- Can be done using repl.it
  - users must create an account to save programs
    - select "Python" from the Template drop down menu and type in a title in the textbox under the word, "Title"
    - click on the button at the bottom right of the screen that reads, "Create Repl"

10

Commands

- Display words to the screen in the console first
  - parenthesis are used to indicate what text the programmer wishes to have displayed
  - the text must have quotation marks around it
    - for example, if a programmer types the command print("Hello World"), the words 'Hello World' will appear in the console when the program is run
      - to run the program, click the button that reads 'Run'

12

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Inputs, Outputs & Data Displays Examples

Creating Outputs

To create a simple text output, the 'print' function is most often used, but it can have several variations to change how the data is displayed. Below are examples of how to create a text output with the print function for several different scenarios. Keep in mind when creating text output, the code in green is what will be displayed for the user.

```
# Basic text output using the print function
print("Hello World")

# Specific information with text explain what the information shows
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)

# Formatting specific results of a function
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Multiple items in a text display
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)
```

Properly Labeling and Displaying Outputs

When creating text output, it is important to make sure the user will understand what is being displayed. Properly labeling output is an important part of visual representation and can make code more user-friendly. Below are a few examples of labels to make the code easier to read and understand.

```
# The labels 'Name' and 'Favorite Food' make the display more clear
and help define what the terms relate to
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)
```



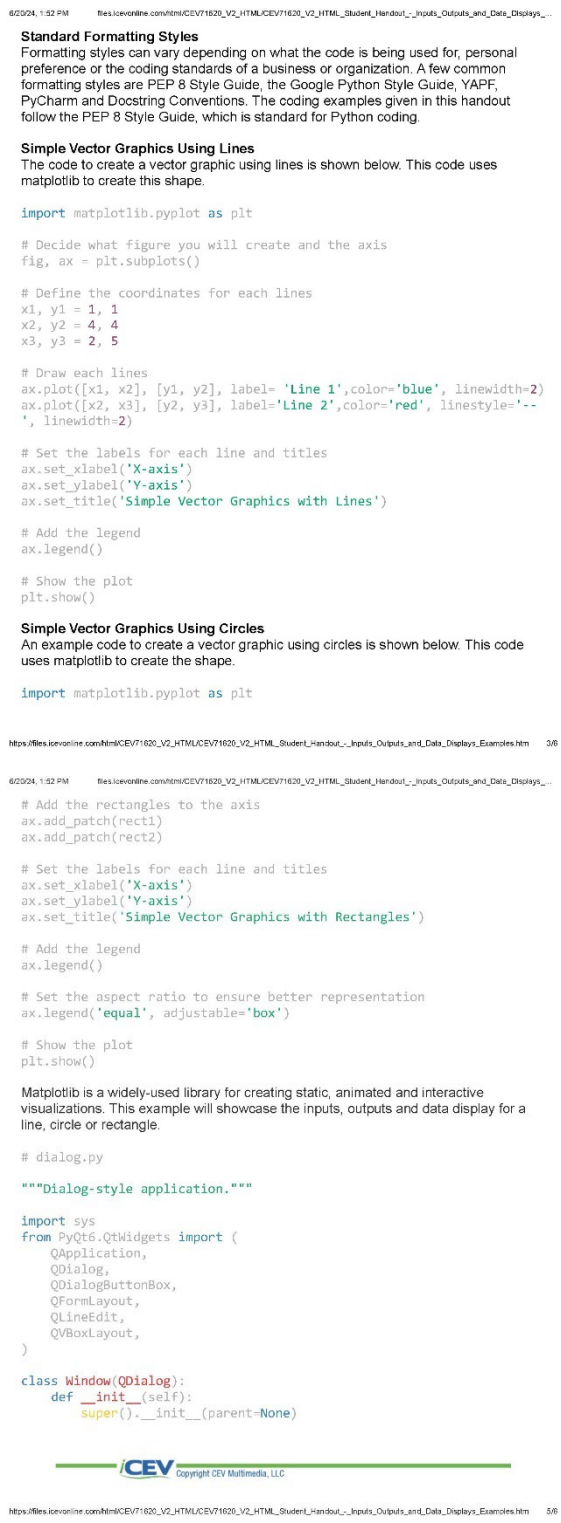
```
# In this example, putting the information that is calculated in a
sentence helps convey what function was performed
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Adding labels that show what x, y, and z correlate makes the display
easier to read
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)

# Adding units when displaying specific information can also make text
displays more clear
num1 = 7
num2 = 4
print(f"Loaded Weight: {num1}lbs Unloaded Weight: {num2}lbs")

# When there are several types of complex information being displayed,
descriptive labels can help clear any confusion
name = "Benjamin"
number = 4
transactions = 13
print(f"Customer: {name}")
print(f"Customer Number for December: {number}")
print(f"Number of Transaction for November: {transactions}")
```

Interactive Input Interfaces

When creating interactive input interfaces with relevant user prompts to acquire data from a user, it is important the code is readable and easy to understand. This code requires information from the user to display a certain message or action. An example of a code that requires user input is shown below.

```
# The prompt below asks for inputs from the user to display a welcome
message
name = input("Enter your name:")

# Now the text will be displayed with the information that the user
added
print(f"Welcome {name}!")
```

Update to Content Accepted by SRP



6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

Standard Formatting Styles
Formatting styles can vary depending on what the code is being used for, personal preference or the coding standards of a business or organization. A few common formatting styles are PEP 8 Style Guide, the Google Python Style Guide, YAPF, PyCharm and Docstring Conventions. The coding examples given in this handout follow the PEP 8 Style Guide, which is standard for Python coding.

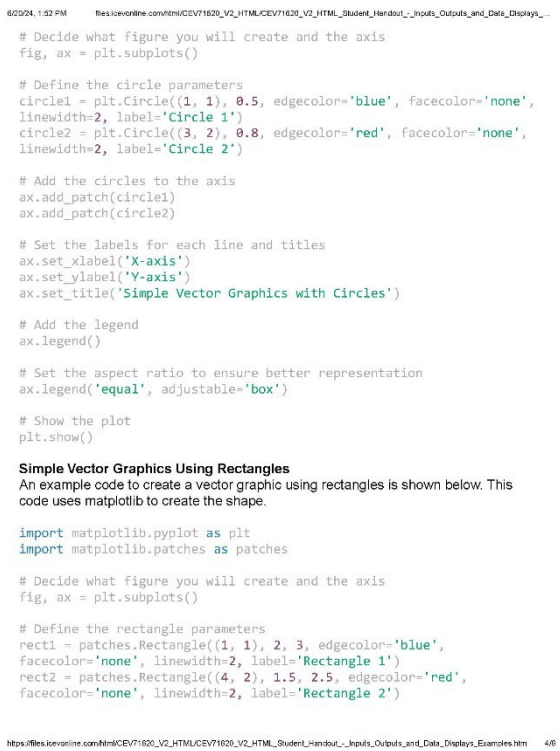
Simple Vector Graphics Using Lines
The code to create a vector graphic using lines is shown below. This code uses matplotlib to create this shape.

```
import matplotlib.pyplot as plt

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the coordinates for each lines
x1, y1 = 1, 1
x2, y2 = 4, 4
x3, y3 = 2, 5

# Draw each lines
ax.plot([x1, x2], [y1, y2], label= 'Line 1',color='blue', linewidth=2)
ax.plot([x2, x3], [y2, y3], label='Line 2',color='red', linestyle='--
', linewidth=2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Lines')

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

Simple Vector Graphics Using Circles
An example code to create a vector graphic using circles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 3/6



6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

```
# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blue', facecolor='none',
linewidth=2, label='Circle 1')
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red', facecolor='none',
linewidth=2, label='Circle 2')

# Add the circles to the axis
ax.add_patch(circle1)
ax.add_patch(circle2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Circles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Simple Vector Graphics Using Rectangles
An example code to create a vector graphic using rectangles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolor='blue',
facecolor='none', linewidth=2, label='Rectangle 1')
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edgecolor='red',
facecolor='none', linewidth=2, label='Rectangle 2')
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 4/6

6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

```
# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Rectangles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Matplotlib is a widely-used library for creating static, animated and interactive visualizations. This example will showcase the inputs, outputs and data display for a line, circle or rectangle.

```
# dialog.py

"""Dialog-style application."""

import sys
from PyQt6.QtWidgets import (
    QApplication,
    QDialog,
    QDialogButtonBox,
    QFormLayout,
    QLineEdit,
    QVBoxLayout,
)

class Window(QDialog):
    def __init__(self):
        super().__init__(parent=None)
```

iCEV Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 5/6

(SE)(Breakout(s)) and (Citation Type(s))
(3)(A)(ii), Narrative and Activity

Description of the specific location and hyperlink to the exact location of currently adopted content

Update to Content Accepted by SRP

Visual Presentation (Slides 7-18, 21-28),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21653

Description of the specific location and hyperlink to the exact location of the proposed new content
Visual Presentation Student Handout-Inputs, Outputs and Data Displays Examples,
https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm
This Student Handout is found in the Visual Presentation lesson beneath the Instructional Materials heading.

Activity-Character Art, https://files.icevonline.com/html/CEV71620_TXP24/CEV71620_TXP24_Activity_-_Character_Art.htm
This Activity is found in the Visual Presentation lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Escape Sequences

- Are a sequence of characters used to output special characters
  - forced return, tab and single or double quotes
- Contain string data, a sequence of characters
  - it might be the case the program will need to force a key press from the keyboard
    - the programmer must include a command that instructs the computer to perform the operation

13

Escape Sequence Example

- May include:
  - if the print statement is too long and needs to be broken up when displayed, \n would be used to indicate 'go to the next line' or force a return
  - to display "Hello" on a separate line from "World" use print("Hello\nWorld")

14

Quotation Marks

- Which need to appear in the output can be done by using \"
  - when there is no backslash before the quotation mark, they do not show up in the output

15

Character Sequences

- Can include:

Tag	Description
\'	Single quote. Example: print('She said,\'Hello\' today")
\"	Double quote. Example: print("She said, \"Hello \" today")
\n	New line. Example: print("Hello\nWorld")
\\	Backslash. Example: print("__/\") will print __/

16

Update to Content Accepted by SRP

Unicode Character Encoding

- Is an extension of ASCII (American Standard Code for Information Interchange)
  - allows programmers to include a multitude of symbols and characters in all languages
  - the tag is generally \u followed by the Unicode value

Unicode Character Example

- Can include:
  - \u00f7, which is a symbol that denotes the divided by character
  - in the code editor, type print("5 \u00f7 2 = 2.5")

```
print("5 \u00f7 2 = 2.5")
```

```
5 ÷ 2 = 2.5
```

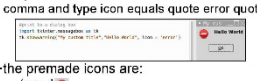
Main Menu

The Showinfo Method

- Will provide a basic dialog box, a title on the top of the box, a lightbulb icon on the left and an OK button
  - to achieve this box type in the command tk.showinfo() in all lowercase letters
    - two arguments can be added separated by a comma and both written with quotation marks surrounding each of the string arguments
    - the first argument is the title on the box and the second argument is the text in the box
      - tk.showinfo("My pop up box" , "Hello World")

The Showinfo Method

- Can include a third argument to change the special icon
  - for example, adding a third parameter inside of the parentheses after "Hello World" insert a comma and type icon equals quote error quote



  - the premade icons are:
    - 'error'
    - 'info'
    - 'question'
    - 'warning'

Dialog Boxes

- Are Graphic User Interfaces like tkinter class
  - tkinter is a common Python graphics package
  - to display text as output in a dialog box in Python, a class called tkinter.messagebox must be imported
    - this is a separate program

Tkinter Package

- Once imported can use the showinfo method
  - takes two arguments, the title of the box and the text that will be displayed in the box
  - the code would start with the keyword import followed tkinter.messagebox

Coding Corner: If the command is followed with the keyword 'as' the class is given a nickname like tk, that you can use.

Printing Text to a File

- Can be done by importing the sys package, opening a 'write' file and printing directly to the file
  - the process involves the commands 'with', 'open' and file

Printing Text to a File

- Starts by typing import sys, then hit enter to advance to the next line
  - the next line of code will start with the word 'with' followed by a space and the word open
  - open is a command that takes two arguments so after the word open, type the opening parenthesis
    - the first argument is the full name of the file and must be in quotations

Update to Content Accepted by SRP

Printing Text to a File

- Includes:
  - adding a comma before entering the second argument
  - the second argument specifies the type of file as read or write in quotation marks
    - lowercase letter w denotes a write file and the lowercase letter r indicates a read only file
      - the file can be named for example 'myfile'

27

Programing Languages

- Provide commands allowing programmers to print words and phrases to the computer monitor
  - text can be delivered from a computer to the world in different ways and differ depending on the programing languages

7

Printing Text to a File

- Includes:
  - hit enter after typing this line
  - the file is now ready to be printed
    - use the print command, but add a second argument specifying the text wanted in the file
      - print("Hello World", file = myfile)

Main Menu

28

10

Printing Display Text

- Can be done by using the programing language Python
  - must have the Python software downloaded and a developmental environment
    - examples of integrated development environments are jGrasp, Eclipse and Visual Studio Code
    - can also use an online environment which does not need to be downloaded such as repl.it

Coding Corner: There are around 700 separate programing languages.

8

1

Printing Display Text

- Can be done using repl.it
  - users must create an account to save programs
    - can be done by clicking the button in the upper right corner that reads, "Sign up"
    - to begin a new program, click on the 'Create' button on the left of the page which will display the Template and Title dialog boxes

9

An Integrated Development Environment

- Should be displayed on the screen
  - file browser
  - code editor
  - console where output will display

11

Printing Display Text

- Can be done using repl.it
  - users must create an account to save programs
    - select "Python" from the Template drop down menu and type in a title in the textbox under the word, "Title"
    - click on the button at the bottom right of the screen that reads, "Create Repl"

10

Commands

- Display words to the screen in the console first
  - parenthesis are used to indicate what text the programmer wishes to have displayed
  - the text must have quotation marks around it
    - for example, if a programmer types the command print("Hello World"), the words 'Hello World' will appear in the console when the program is run
      - to run the program, click the button that reads 'Run'

12

2

3

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Inputs, Outputs & Data Displays Examples

Creating Outputs
To create a simple text output, the 'print' function is most often used, but it can have several variations to change how the data is displayed. Below are examples of how to create a text output with the print function for several different scenarios. Keep in mind when creating text output, the code in green is what will be displayed for the user.

```
# Basic text output using the print function
print("Hello World")

# Specific information with text explain what the information shows
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)

# Formatting specific results of a function
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Multiple items in a text display
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)
```

Properly Labeling and Displaying Outputs
When creating text output, it is important to make sure the user will understand what is being displayed. Properly labeling output is an important part of visual representation and can make code more user-friendly. Below are a few examples of labels to make the code easier to read and understand.

```
# The labels 'Name' and 'Favorite Food' make the display more clear
and help define what the terms relate to
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)
```



```
# In this example, putting the information that is calculated in a
sentence helps convey what function was performed
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Adding labels that show what x, y, and z correlate makes the display
easier to read
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)

# Adding units when displaying specific information can also make text
displays more clear
num1 = 7
num2 = 4
print(f"Loaded Weight: {num1}lbs Unloaded Weight: {num2}lbs")

# When there are several types of complex information being displayed,
descriptive labels can help clear any confusion
name = "Benjamin"
number = 4
transactions = 13
print(f"Customer: {name}")
print(f"Customer Number for December: {number}")
print(f"Number of Transaction for November: {transactions}")
```

Interactive Input Interfaces
When creating interactive input interfaces with relevant user prompts to acquire data from a user, it is important the code is readable and easy to understand. This code requires information from the user to display a certain message or action. An example of a code that requires user input is shown below.

```
# The prompt below asks for inputs from the user to display a welcome
message
name = input("Enter your name:")

# Now the text will be displayed with the information that the user
added
print(f"Welcome {name}!")
```


Standard Formatting Styles
Formatting styles can vary depending on what the code is being used for, personal preference or the coding standards of a business or organization. A few common formatting styles are PEP 8 Style Guide, the Google Python Style Guide, YAPF, PyCharm and Docstring Conventions. The coding examples given in this handout follow the PEP 8 Style Guide, which is standard for Python coding.

Simple Vector Graphics Using Lines
The code to create a vector graphic using lines is shown below. This code uses matplotlib to create this shape.

```
import matplotlib.pyplot as plt

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the coordinates for each lines
x1, y1 = 1, 1
x2, y2 = 4, 4
x3, y3 = 2, 5

# Draw each lines
ax.plot([x1, x2], [y1, y2], label= 'Line 1',color='blue', linewidth=2)
ax.plot([x2, x3], [y2, y3], label='Line 2',color='red', linestyle='--
', linewidth=2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Lines')

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

Simple Vector Graphics Using Circles
An example code to create a vector graphic using circles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
```

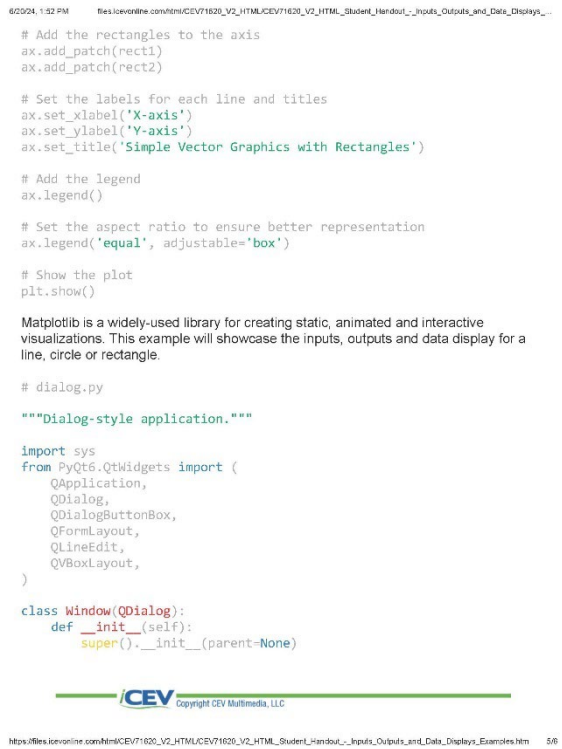


```
# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blue', facecolor='none',
linewidth=2, label='Circle 1')
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red', facecolor='none',
linewidth=2, label='Circle 2')

# Add the circles to the axis
ax.add_patch(circle1)
ax.add_patch(circle2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Circles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Simple Vector Graphics Using Rectangles
An example code to create a vector graphic using rectangles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolor='blue',
facecolor='none', linewidth=2, label='Rectangle 1')
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edgecolor='red',
facecolor='none', linewidth=2, label='Rectangle 2')
```

Update to Content Accepted by SRP

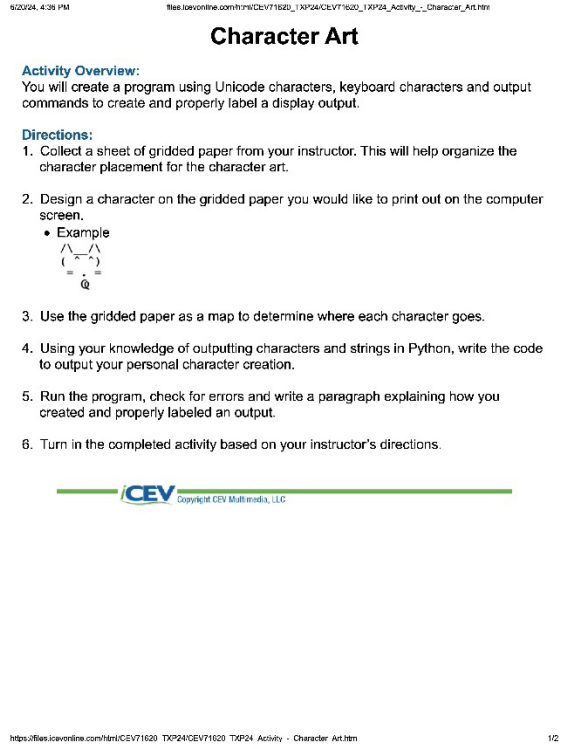



```
# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Rectangles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Matplotlib is a widely-used library for creating static, animated and interactive visualizations. This example will showcase the inputs, outputs and data display for a line, circle or rectangle.

```
# dialog.py

"""Dialog-style application."""

import sys
from PyQt6.QtWidgets import (
    QApplication,
    QDialog,
    QDialogButtonBox,
    QFormLayout,
    QLineEdit,
    QVBoxLayout,
)

class Window(QDialog):
    def __init__(self):
        super().__init__(parent=None)
```

iCEV Copyright CEV Multimedia, LLC



Character Art

Activity Overview:
You will create a program using Unicode characters, keyboard characters and output commands to create and properly label a display output.

Directions:

1. Collect a sheet of gridded paper from your instructor. This will help organize the character placement for the character art.
2. Design a character on the gridded paper you would like to print out on the computer screen.
   - Example
     ```
     /\__/\
     ( ^ ^)
      = . =
        Q
     ```
3. Use the gridded paper as a map to determine where each character goes.
4. Using your knowledge of outputting characters and strings in Python, write the code to output your personal character creation.
5. Run the program, check for errors and write a paragraph explaining how you created and properly labeled an output.
6. Turn in the completed activity based on your instructor's directions.

iCEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(3)(A)(iii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Visual Presentation (Slides 7-18, 21-28),

Update to Content Accepted by SRP

https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21653

Description of the specific location and hyperlink to the exact location of the proposed new content
Visual Presentation Student Handout-Inputs, Outputs and Data Displays Examples, https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm
This Student Handout is found in the Visual Presentation lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Escape Sequences

- Are a sequence of characters used to output special characters
  - forced return, tab and single or double quotes
- Contain string data, a sequence of characters
  - it might be the case the program will need to force a key press from the keyboard
    - the programmer must include a command that instructs the computer to perform the operation

13

Escape Sequence Example

- May include:
  - if the print statement is too long and needs to be broken up when displayed, \n would be used to indicate 'go to the next line' or force a return
  - to display "Hello" on a separate line from "World" use print("Hello\nWorld")

14

Quotation Marks

- Which need to appear in the output can be done by using \"
  - when there is no backslash before the quotation mark, they do not show up in the output

15

Character Sequences

- Can include:

Tag	Description
\'	Single quote. Example: print('She said,\'Hello\' today")
\"	Double quote. Example: print("She said, \"Hello \" today")
\n	New line. Example: print("Hello\nWorld")
\\	Backslash. Example: print("__/\\") will print __/

16

Update to Content Accepted by SRP

Unicode Character Encoding

- Is an extension of ASCII (American Standard Code for Information Interchange)
  - allows programmers to include a multitude of symbols and characters in all languages
  - the tag is generally \u followed by the Unicode value

Unicode Character Example

- Can include:
  - \u00f7, which is a symbol that denotes the divided by character
  - in the code editor, type print("5 \u00f7 2 = 2.5")

```
print("5 \u00f7 2 = 2.5")
```

```
5 ÷ 2 = 2.5
```

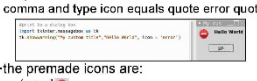
Main Menu

The Showinfo Method

- Will provide a basic dialog box, a title on the top of the box, a lightbulb icon on the left and an OK button
  - to achieve this box type in the command tk.showinfo() in all lowercase letters
    - two arguments can be added separated by a comma and both written with quotation marks surrounding each of the string arguments
    - the first argument is the title on the box and the second argument is the text in the box
      - tk.showinfo("My pop up box" , "Hello World")

The Showinfo Method

- Can include a third argument to change the special icon
  - for example, adding a third parameter inside of the parentheses after "Hello World" insert a comma and type icon equals quote error quote



  - the premade icons are:
    - 'error'
    - 'info'
    - 'question'
    - 'warning'

Dialog Boxes

- Are Graphic User Interfaces like tkinter class
  - tkinter is a common Python graphics package
  - to display text as output in a dialog box in Python, a class called tkinter.messagebox must be imported
    - this is a separate program

Tkinter Package

- Once imported can use the showinfo method
  - takes two arguments, the title of the box and the text that will be displayed in the box
  - the code would start with the keyword import followed tkinter.messagebox

Coding Corner: If the command is followed with the keyword 'as' the class is given a nickname like tk, that you can use.

Printing Text to a File

- Can be done by importing the sys package, opening a 'write' file and printing directly to the file
  - the process involves the commands 'with', 'open' and file

Printing Text to a File

- Starts by typing import sys, then hit enter to advance to the next line
  - the next line of code will start with the word 'with' followed by a space and the word open
  - open is a command that takes two arguments so after the word open, type the opening parenthesis
    - the first argument is the full name of the file and must be in quotations

Update to Content Accepted by SRP

Printing Text to a File

- Includes:
  - adding a comma before entering the second argument
  - the second argument specifies the type of file as read or write in quotation marks
    - lowercase letter w denotes a write file and the lowercase letter r indicates a read only file
      - the file can be named for example 'myfile'

27

Programing Languages

- Provide commands allowing programmers to print words and phrases to the computer monitor
  - text can be delivered from a computer to the world in different ways and differ depending on the programing languages

7

Printing Text to a File

- Includes:
  - hit enter after typing this line
  - the file is now ready to be printed
    - use the print command, but add a second argument specifying the text wanted in the file
      - print("Hello World", file = myfile)

Main Menu

28

Printing Display Text

- Can be done by using the programing language Python
  - must have the Python software downloaded and a developmental environment
    - examples of integrated development environments are jGrasp, Eclipse and Visual Studio Code
    - can also use an online environment which does not need to be downloaded such as repl.it

Coding Corner: There are around 700 separate programing languages.

8

6/20/2024

6/20/2024

Printing Display Text

- Can be done using repl.it
  - users must create an account to save programs
    - can be done by clicking the button in the upper right corner that reads, "Sign up"
    - to begin a new program, click on the 'Create' button on the left of the page which will display the Template and Title dialog boxes

9

An Integrated Development Environment

- Should be displayed on the screen
  - file browser
  - code editor
  - console where output will display

11

Printing Display Text

- Can be done using repl.it
  - users must create an account to save programs
    - select "Python" from the Template drop down menu and type in a title in the textbox under the word, "Title"
    - click on the button at the bottom right of the screen that reads, "Create Repl"

10

Commands

- Display words to the screen in the console first
  - parenthesis are used to indicate what text the programmer wishes to have displayed
  - the text must have quotation marks around it
    - for example, if a programmer types the command print("Hello World"), the words 'Hello World' will appear in the console when the program is run
      - to run the program, click the button that reads 'Run'

12

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Inputs, Outputs & Data Displays Examples

Creating Outputs
To create a simple text output, the 'print' function is most often used, but it can have several variations to change how the data is displayed. Below are examples of how to create a text output with the print function for several different scenarios. Keep in mind when creating text output, the code in green is what will be displayed for the user.

```
# Basic text output using the print function
print("Hello World")

# Specific information with text explain what the information shows
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)

# Formatting specific results of a function
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Multiple items in a text display
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)
```

Properly Labeling and Displaying Outputs
When creating text output, it is important to make sure the user will understand what is being displayed. Properly labeling output is an important part of visual representation and can make code more user-friendly. Below are a few examples of labels to make the code easier to read and understand.

```
# The labels 'Name' and 'Favorite Food' make the display more clear
and help define what the terms relate to
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)
```



```
# In this example, putting the information that is calculated in a
sentence helps convey what function was performed
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Adding labels that show what x, y, and z correlate makes the display
easier to read
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)

# Adding units when displaying specific information can also make text
displays more clear
num1 = 7
num2 = 4
print(f"Loaded Weight: {num1}lbs Unloaded Weight: {num2}lbs")

# When there are several types of complex information being displayed,
descriptive labels can help clear any confusion
name = "Benjamin"
number = 4
transactions = 13
print(f"Customer: {name}")
print(f"Customer Number for December: {number}")
print(f"Number of Transaction for November: {transactions}")
```

Interactive Input Interfaces
When creating interactive input interfaces with relevant user prompts to acquire data from a user, it is important the code is readable and easy to understand. This code requires information from the user to display a certain message or action. An example of a code that requires user input is shown below.

```
# The prompt below asks for inputs from the user to display a welcome
message
name = input("Enter your name:")

# Now the text will be displayed with the information that the user
added
print(f"Welcome {name}!")
```


Standard Formatting Styles
Formatting styles can vary depending on what the code is being used for, personal preference or the coding standards of a business or organization. A few common formatting styles are PEP 8 Style Guide, the Google Python Style Guide, YAPF, PyCharm and Docstring Conventions. The coding examples given in this handout follow the PEP 8 Style Guide, which is standard for Python coding.

Simple Vector Graphics Using Lines
The code to create a vector graphic using lines is shown below. This code uses matplotlib to create this shape.

```
import matplotlib.pyplot as plt

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the coordinates for each lines
x1, y1 = 1, 1
x2, y2 = 4, 4
x3, y3 = 2, 5

# Draw each lines
ax.plot([x1, x2], [y1, y2], label= 'Line 1',color='blue', linewidth=2)
ax.plot([x2, x3], [y2, y3], label='Line 2',color='red', linestyle='--
', linewidth=2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Lines')

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

Simple Vector Graphics Using Circles
An example code to create a vector graphic using circles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
```

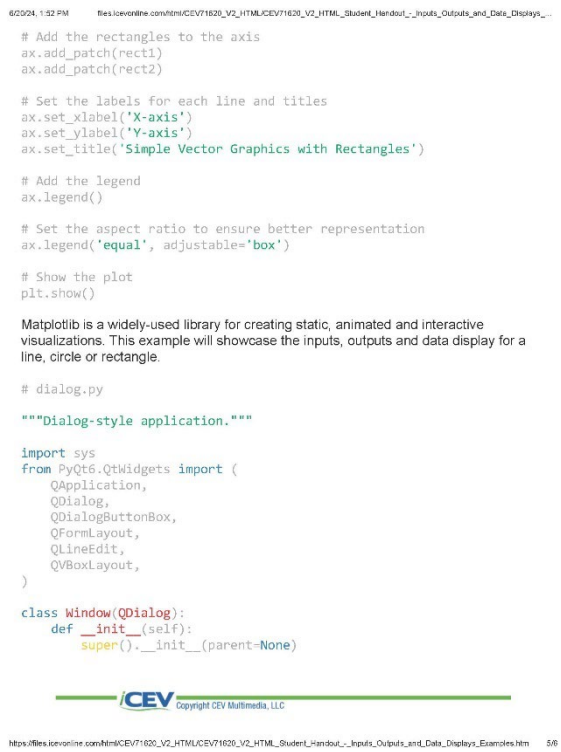


```
# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blue', facecolor='none',
linewidth=2, label='Circle 1')
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red', facecolor='none',
linewidth=2, label='Circle 2')

# Add the circles to the axis
ax.add_patch(circle1)
ax.add_patch(circle2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Circles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Simple Vector Graphics Using Rectangles
An example code to create a vector graphic using rectangles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolor='blue',
facecolor='none', linewidth=2, label='Rectangle 1')
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edgecolor='red',
facecolor='none', linewidth=2, label='Rectangle 2')
```

Update to Content Accepted by SRP



6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

```
# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Rectangles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Matplotlib is a widely-used library for creating static, animated and interactive visualizations. This example will showcase the inputs, outputs and data display for a line, circle or rectangle.

```
# dialog.py

"""Dialog-style application."""

import sys
from PyQt6.QtWidgets import (
    QApplication,
    QDialog,
    QDialogButtonBox,
    QFormLayout,
    QLineEdit,
    QVBoxLayout,
)

class Window(QDialog):
    def __init__(self):
        super().__init__(parent=None)
```

Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 5/8

(SE)(Breakout(s)) and (Citation Type(s))
(3)(B)(i), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Visual Presentation (Slides 31-40),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21653

Description of the specific location and hyperlink to the exact location of the proposed new content
Visual Presentation Student Handout-Inputs, Outputs and Data Displays Examples,
https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm
This Student Handout is found in the Visual Presentation lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Hardware Inputs

- Are the physical components of a computer
- Are used to input data, commands or both into a computer
- Can include:
  - keyboard
  - mouse
  - microphone
  - webcam
  - touchpad
  - touch screen
  - copier/scanner
  - buttons

Software Inputs

- Are programs and operating systems running on a computer
  - provide the interface through which a user can give commands and perform tasks
- Can also take input in the form of data flow
  - moving sequence of information that is directed to a specific action
- Can include software applications such as:
  - Windows 10™
  - Adobe Photoshop®
  - Google Chrome™

Widget

- Examples include:
  - Python and the PyQt framework versus Java and the Java Swing toolkit
  - each will give the UI's created a distinct appearance

Python

Java

PyQt Framework

- Can develop two types of GUI desktop applications
  - main window-style application (QMainWindow)
  - dialog-style application (QDialog)

Computer Inputs

- Differ based on the computer language being used
  - languages can include
    - Python
    - Java

Coding Corner: The first programming language was called FORTRAN.

Graphical User Interfaces (GUI's)

- Are a form of software input
- Allows users to interact with digital devices through graphical elements such as icons, buttons and windows
- May differ depending on the programming language being used
  - a widget set provided by the programing language or framework will change the appearance
    - a widget is a basic building block for a UI such as a button or a text field

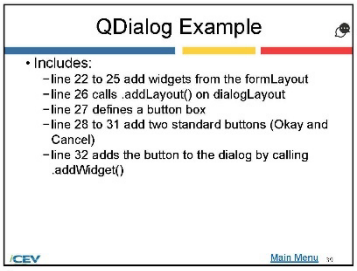
QDialog

- Is created by a dialog window or a stand-alone window
  - always an independent window
  - a dialog with a parent will display centered on top of the parent widget
    - will share the parent's task bar entry
  - dialog windows are commonly used in main window-style applications for brief communication and interaction with the user

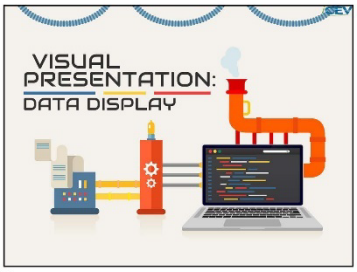
QDialog Example

- Includes:
  - line 16 defining a Window class for the app's GUI by inheriting from QDialog
  - line 18 calls the parent classes
  - line 19 sets the window title
  - line 20 assigns a QVBoxLayout object to dialogLayout
  - line 21 assigns a QFormLayout object to formLayout

Update to Content Accepted by SRP



39



40

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Inputs, Outputs & Data Displays Examples

Creating Outputs

To create a simple text output, the 'print' function is most often used, but it can have several variations to change how the data is displayed. Below are examples of how to create a text output with the print function for several different scenarios. Keep in mind when creating text output, the code in green is what will be displayed for the user.

```
# Basic text output using the print function
print("Hello World")

# Specific information with text explain what the information shows
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)

# Formatting specific results of a function
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Multiple items in a text display
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)
```

Properly Labeling and Displaying Outputs

When creating text output, it is important to make sure the user will understand what is being displayed. Properly labeling output is an important part of visual representation and can make code more user-friendly. Below are a few examples of labels to make the code easier to read and understand.

```
# The labels 'Name' and 'Favorite Food' make the display more clear
and help define what the terms relate to
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)
```



```
# In this example, putting the information that is calculated in a
sentence helps convey what function was performed
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Adding labels that show what x, y, and z correlate makes the display
easier to read
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)

# Adding units when displaying specific information can also make text
displays more clear
num1 = 7
num2 = 4
print(f"Loaded Weight: {num1}lbs Unloaded Weight: {num2}lbs")

# When there are several types of complex information being displayed,
descriptive labels can help clear any confusion
name = "Benjamin"
number = 4
transactions = 13
print(f"Customer: {name}")
print(f"Customer Number for December: {number}")
print(f"Number of Transaction for November: {transactions}")
```

Interactive Input Interfaces

When creating interactive input interfaces with relevant user prompts to acquire data from a user, it is important the code is readable and easy to understand. This code requires information from the user to display a certain message or action. An example of a code that requires user input is shown below.

```
# The prompt below asks for inputs from the user to display a welcome
message
name = input("Enter your name:")

# Now the text will be displayed with the information that the user
added
print(f"Welcome {name}!")
```

Update to Content Accepted by SRP

Standard Formatting Styles
Formatting styles can vary depending on what the code is being used for, personal preference or the coding standards of a business or organization. A few common formatting styles are PEP 8 Style Guide, the Google Python Style Guide, YAPF, PyCharm and Docstring Conventions. The coding examples given in this handout follow the PEP 8 Style Guide, which is standard for Python coding.

Simple Vector Graphics Using Lines
The code to create a vector graphic using lines is shown below. This code uses matplotlib to create this shape.

```
import matplotlib.pyplot as plt

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the coordinates for each lines
x1, y1 = 1, 1
x2, y2 = 4, 4
x3, y3 = 2, 5

# Draw each lines
ax.plot([x1, x2], [y1, y2], label= 'Line 1',color='blue', linewidth=2)
ax.plot([x2, x3], [y2, y3], label='Line 2',color='red', linestyle='--
', linewidth=2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Lines')

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

Simple Vector Graphics Using Circles
An example code to create a vector graphic using circles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
```



```
# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blue', facecolor='none',
linewidth=2, label='Circle 1')
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red', facecolor='none',
linewidth=2, label='Circle 2')

# Add the circles to the axis
ax.add_patch(circle1)
ax.add_patch(circle2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Circles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Simple Vector Graphics Using Rectangles
An example code to create a vector graphic using rectangles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolor='blue',
facecolor='none', linewidth=2, label='Rectangle 1')
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edgecolor='red',
facecolor='none', linewidth=2, label='Rectangle 2')
```



```
# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Rectangles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Matplotlib is a widely-used library for creating static, animated and interactive visualizations. This example will showcase the inputs, outputs and data display for a line, circle or rectangle.

```
# dialog.py

"""Dialog-style application."""

import sys
from PyQt6.QtWidgets import (
    QApplication,
    QDialog,
    QDialogButtonBox,
    QFormLayout,
    QLineEdit,
    QVBoxLayout,
)

class Window(QDialog):
    def __init__(self):
        super().__init__(parent=None)
```

iCEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(3)(C)(i), Narrative

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming with Proper Format and Style (Slides 4-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21642

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming with Proper Format and Style (Slides 4-25),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22294
In the Programming with Proper Format & Style PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Programming with Proper Format & Style lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Programming Format

- Refers to the way in which the source code of a program is written
  - this example is written in Python and includes indentation, spacing, line breaks and white space

4

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

5

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

6

Descriptive Identifiers

- Are names used for variables, functions and other elements of the code
- Are chosen to accurately and concisely describe their purpose or meaning
- Can make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future

7

Update to Content Accepted by SRP

Descriptive Identifiers

- Example:
  - in this code the identifiers "customer_name", "customer_age", "calculate_discount" and "Customer" are all descriptive

Internal Comments

- Are comments placed within the body of a function or other block of code
- Are used to explain the purpose or function of specific lines or blocks of code
- Are typically used to provide more detailed explanations or clarifications of the code
- Can be especially useful when the code is complex or does something which is not obvious

Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```


White Space

- Refers to the blank space between code
- Helps format the code in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Is used to group statements together and to indicate which statements belong to a particular block of code, such as a function, loop or conditional statement

White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

Update to Content Accepted by SRP

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

16

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Examples include:
  - snake_case
  - discount_amount

17

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

18

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")

19

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

20

Example of Proper Formatting & Style

- Include:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

21

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Programming Format

- Refers to the way in which the source code of a program is written
  - coding example is written in Python and includes indentation, spacing, line breaks and white space

NOTE: Use the Coding Examples Student Handout for references.

4

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

5

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

NOTE: Use the Coding Examples Student Handout for references.

6

Descriptive Identifiers

- Enhance the readability of the code by:
  - allowing other users to understand what the creator intended
  - increasing understanding of code
  - describing the purpose behind a code
- Enhance the functionality of the code by:
  - providing descriptions of the coding process
  - adding necessary details to the code to allow future users to adapt a code for a different use

7

Proper Use of Descriptive Identifiers

- Examples include:
  - "customer_name", "customer_age", "calculate_discount" and "Customer"

NOTE: Use the Coding Examples Student Handout for references.

8

Improper Use of Descriptive Identifiers

- Makes the descriptive identifiers hard to understand and creates confusion
- Examples include:
  - single letter identifiers
  - unclear abbreviations
  - inconsistent naming
  - misleading names
  - non-descriptive names

NOTE: Use the Coding Examples Student Handout for references.

9

Internal Comments

- Are comments placed within the body of a function or other block of code
- Enhance the readability of the code by:
  - explaining the purpose or function of specific lines or blocks of code
  - providing more detailed explanations or clarifications of the code
- Enhance the functionality of the code by:
  - providing descriptions when the code is complex or performs a function that is not obvious

10

Proper Use of Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

11

Update to Content Accepted by SRP

Improper Use of Internal Comments

- Makes code harder to understand and maintain
- Include internal comments which are:
  - redundant
  - misleading
  - outdated
  - excessive
  - unclear

6/20/2024

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Enhances the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

White Space

- Refers to the blank space between code
- Enhance the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhance the functionality of the code by:
  - providing a standard format to make code run properly in a program

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Enhances the readability of the code by:
  - grouping statements together
  - indicating which statements belong to a particular block of code, such as a function, loop or conditional statement
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

6

6/20/2024

Proper Use of White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

NOTE: Use the Coding Examples Student Handout for references.

6/20/2024

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

Improper Use of White Space, Spacing & Indentation

- Can lead to errors when editing or changing code because of confusion of which lines of code belong to which function
- Examples include:
  - extra whitespaces
  - extra blank lines
  - extra or inconsistent indentation
  - inconsistent spacing

NOTE: Use the Coding Examples Student Handout for references.

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

8

Update to Content Accepted by SRP

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Works by the first letter of each word being written in lowercase
- Examples include:
  - snake_case
  - discount_amount

20

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

21

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")
- Begins with a capital letter and ends with a period

22

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

23

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

NOTE: Use the Coding Examples Student Handout for references.

24

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```

25

Coding Examples

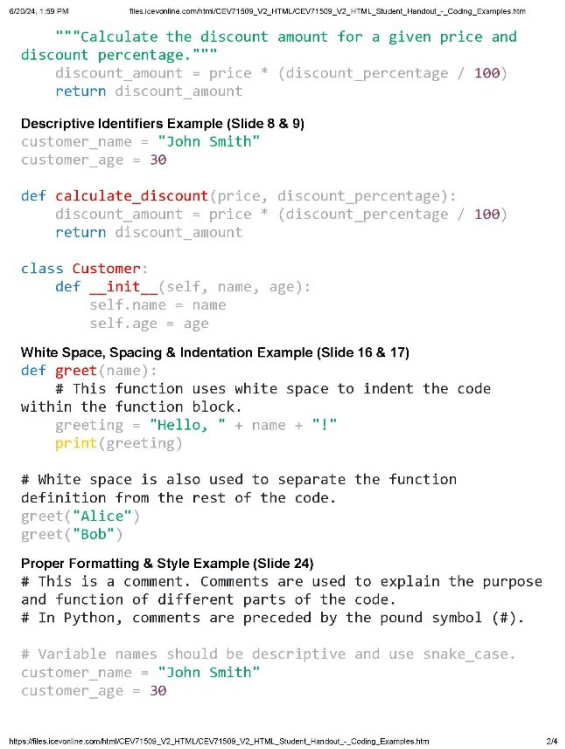
Programming Format Example (Slide 4)

```
import tokenize
import io
import sys

def check_formatting(filename):
    with open(filename, "rb") as f:
        try:
            tokens = tokenize(f.readline)
        except tokenize.TokenError as e:
            print("Formatting error:", e)
            return False
    return True

if len(sys.argv) != 2:
    print("Usage: python check_formatting.py you_file.py")
else:
    result = check_formatting(sys.argv[1])
    if result:
        print("Formatting is correct!")
    else:
        print("Formatting is incorrect.")
```

Standardized Programming Style Example (Slide 6)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30

# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
```

Update to Content Accepted by SRP



6/20/24, 1:59 PM files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

Descriptive Identifiers Example (Slide 8 & 9)

```
customer_name = "John Smith"
customer_age = 30

def calculate_discount(price, discount_percentage):
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

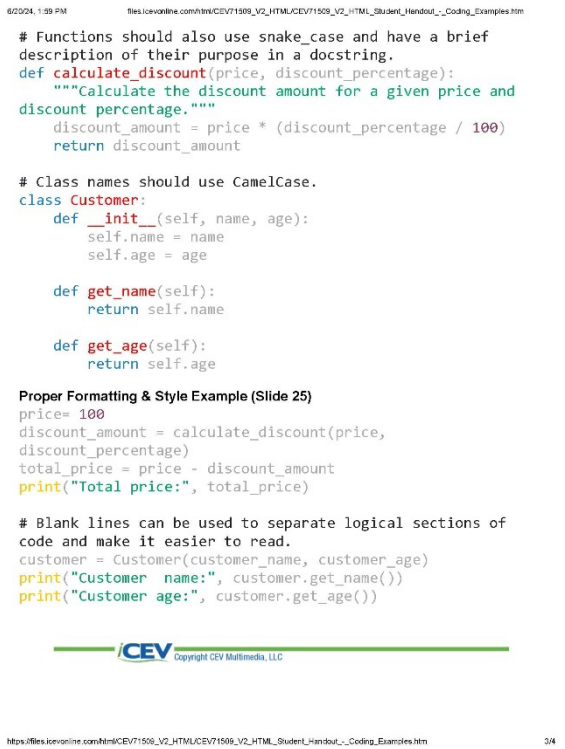
White Space, Spacing & Indentation Example (Slide 16 & 17)

```
def greet(name):
    # This function uses white space to indent the code
within the function block.
    greeting = "Hello, " + name + "!"
    print(greeting)

# White space is also used to separate the function
definition from the rest of the code.
greet("Alice")
greet("Bob")
```

Proper Formatting & Style Example (Slide 24)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30
```

https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm 2/4



6/20/24, 1:59 PM files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

# Class names should use CamelCase.
class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

Proper Formatting & Style Example (Slide 25)

```
price= 100
discount_amount = calculate_discount(price,
discount_percentage)
total_price = price - discount_amount
print("Total price:", total_price)

# Blank lines can be used to separate logical sections of
code and make it easier to read.
customer = Customer(customer_name, customer_age)
print("Customer  name:", customer.get_name())
print("Customer age:", customer.get_age())
```

iCEV Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm 3/4

(SE)(Breakout(s)) and (Citation Type(s))
(3)(C)(ii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming with Proper Format and Style (Slides 4-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21642

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming with Proper Format and Style (Slides 4-25),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22294
In the Programming with Proper Format & Style PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Programming with Proper Format & Style lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Programming Format

- Refers to the way in which the source code of a program is written
  - this example is written in Python and includes indentation, spacing, line breaks and white space

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

Descriptive Identifiers

- Are names used for variables, functions and other elements of the code
- Are chosen to accurately and concisely describe their purpose or meaning
- Can make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future

Descriptive Identifiers

- Example:
  - in this code the identifiers "customer_name", "customer_age", "calculate_discount" and "Customer" are all descriptive

Internal Comments

- Are comments placed within the body of a function or other block of code
- Are used to explain the purpose or function of specific lines or blocks of code
- Are typically used to provide more detailed explanations or clarifications of the code
- Can be especially useful when the code is complex or does something which is not obvious

Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```


White Space

- Refers to the blank space between code
- Helps format the code in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

Update to Content Accepted by SRP

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

12

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Is used to group statements together and to indicate which statements belong to a particular block of code, such as a function, loop or conditional statement

13

White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

14

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

15

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

16

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Examples include:
  - snake_case
  - discount_amount

17

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

18

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")

19

Update to Content Accepted by SRP

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

Example of Proper Formatting & Style

- Include:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```


Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Programming Format

- Refers to the way in which the source code of a program is written
  - coding example is written in Python and includes indentation, spacing, line breaks and white space

NOTE: Use the Coding Examples Student Handout for references.

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

NOTE: Use the Coding Examples Student Handout for references.

Descriptive Identifiers

- Enhance the readability of the code by:
  - allowing other users to understand what the creator intended
  - increasing understanding of code
  - describing the purpose behind a code
- Enhance the functionality of the code by:
  - providing descriptions of the coding process
  - adding necessary details to the code to allow future users to adapt a code for a different use

Update to Content Accepted by SRP

Proper Use of Descriptive Identifiers

- Examples include:
  - "customer_name", "customer_age", "calculate_discount" and "Customer"

NOTE: Use the Coding Examples Student Handout for references.

8

Improper Use of Descriptive Identifiers

- Makes the descriptive identifiers hard to understand and creates confusion
- Examples include:
  - single letter identifiers
  - unclear abbreviations
  - inconsistent naming
  - misleading names
  - non-descriptive names

NOTE: Use the Coding Examples Student Handout for references.

9

Improper Use of Internal Comments

- Makes code harder to understand and maintain
- Include internal comments which are:
  - redundant
  - misleading
  - outdated
  - excessive
  - unclear

12

White Space

- Refers to the blank space between code
- Enhance the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhance the functionality of the code by:
  - providing a standard format to make code run properly in a program

13

Internal Comments

- Are comments placed within the body of a function or other block of code
- Enhance the readability of the code by:
  - explaining the purpose or function of specific lines or blocks of code
  - providing more detailed explanations or clarifications of the code
- Enhance the functionality of the code by:
  - providing descriptions when the code is complex or performs a function that is not obvious

10

Proper Use of Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

11

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Enhances the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

14

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Enhances the readability of the code by:
  - grouping statements together
  - indicating which statements belong to a particular block of code, such as a function, loop or conditional statement
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

15

Update to Content Accepted by SRP

Proper Use of White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

NOTE: Use the Coding Examples Student Handout for references.

16

Improper Use of White Space, Spacing & Indentation

- Can lead to errors when editing or changing code because of confusion of which lines of code belong to which function
- Examples include:
  - extra whitespaces
  - extra blank lines
  - extra or inconsistent indentation
  - inconsistent spacing

NOTE: Use the Coding Examples Student Handout for references.

17

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Works by the first letter of each word being written in lowercase
- Examples include:
  - snake_case
  - discount_amount

20

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

21

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

18

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

19

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")
- Begins with a capital letter and ends with a period

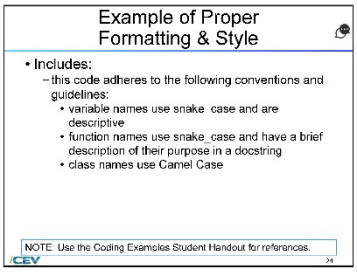
22

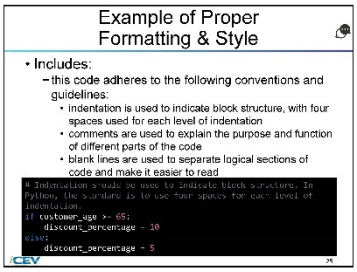
Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

23

Update to Content Accepted by SRP





Coding Examples

Programming Format Example (Slide 4)

```
import tokenize
import io
import sys

def check_formatting(filename):
    with open(filename, "rb") as f:
        try:
            tokens = tokenize(f.readline)
        except tokenize.TokenError as e:
            print("Formatting error:", e)
            return False
    return True

if len(sys.argv) != 2:
    print("Usage: python check_formatting.py you_file.py")
else:
    result = check_formatting(sys.argv[1])
    if result:
        print("Formatting is correct!")
    else:
        print("Formatting is incorrect.")
```

Standardized Programming Style Example (Slide 6)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30

# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
```



```
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

Descriptive Identifiers Example (Slide 8 & 9)

```
customer_name = "John Smith"
customer_age = 30

def calculate_discount(price, discount_percentage):
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

White Space, Spacing & Indentation Example (Slide 16 & 17)

```
def greet(name):
    # This function uses white space to indent the code
within the function block.
    greeting = "Hello, " + name + "!"
    print(greeting)

# White space is also used to separate the function
definition from the rest of the code.
greet("Alice")
greet("Bob")
```

Proper Formatting & Style Example (Slide 24)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30
```



```
# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

# Class names should use CamelCase.
class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

Proper Formatting & Style Example (Slide 25)

```
price= 100
discount_amount = calculate_discount(price,
discount_percentage)
total_price = price - discount_amount
print("Total price:", total_price)

# Blank lines can be used to separate logical sections of
code and make it easier to read.
customer = Customer(customer_name, customer_age)
print("Customer  name:", customer.get_name())
print("Customer age:", customer.get_age())
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(3)(C)(iii), Narrative

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming with Proper Format and Style (Slides 4-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21642

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming with Proper Format and Style (Slides 4-25),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22294
In the Programming with Proper Format & Style PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Programming with Proper Format & Style lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Programming Format

- Refers to the way in which the source code of a program is written
  - this example is written in Python and includes indentation, spacing, line breaks and white space

4

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

5

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

6

Descriptive Identifiers

- Are names used for variables, functions and other elements of the code
- Are chosen to accurately and concisely describe their purpose or meaning
- Can make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future

7

Update to Content Accepted by SRP

Descriptive Identifiers

- Example:
  - in this code the identifiers "customer_name", "customer_age", "calculate_discount" and "Customer" are all descriptive

Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```


Internal Comments

- Are comments placed within the body of a function or other block of code
- Are used to explain the purpose or function of specific lines or blocks of code
- Are typically used to provide more detailed explanations or clarifications of the code
- Can be especially useful when the code is complex or does something which is not obvious

White Space

- Refers to the blank space between code
- Helps format the code in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Is used to group statements together and to indicate which statements belong to a particular block of code, such as a function, loop or conditional statement

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

Update to Content Accepted by SRP

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

16

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

18

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Examples include:
  - snake_case
  - discount_amount

17

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")

19

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

20

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```

22

Example of Proper Formatting & Style

- Include:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

21

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Programming Format

- Refers to the way in which the source code of a program is written
  - coding example is written in Python and includes indentation, spacing, line breaks and white space

NOTE: Use the Coding Examples Student Handout for references.

4

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

5

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

NOTE: Use the Coding Examples Student Handout for references.

6

Descriptive Identifiers

- Enhance the readability of the code by:
  - allowing other users to understand what the creator intended
  - increasing understanding of code
  - describing the purpose behind a code
- Enhance the functionality of the code by:
  - providing descriptions of the coding process
  - adding necessary details to the code to allow future users to adapt a code for a different use

7

Proper Use of Descriptive Identifiers

- Examples include:
  - "customer_name", "customer_age", "calculate_discount" and "Customer"

NOTE: Use the Coding Examples Student Handout for references.

8

Improper Use of Descriptive Identifiers

- Makes the descriptive identifiers hard to understand and creates confusion
- Examples include:
  - single letter identifiers
  - unclear abbreviations
  - inconsistent naming
  - misleading names
  - non-descriptive names

NOTE: Use the Coding Examples Student Handout for references.

9

Internal Comments

- Are comments placed within the body of a function or other block of code
- Enhance the readability of the code by:
  - explaining the purpose or function of specific lines or blocks of code
  - providing more detailed explanations or clarifications of the code
- Enhance the functionality of the code by:
  - providing descriptions when the code is complex or performs a function that is not obvious

10

Proper Use of Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

11

Update to Content Accepted by SRP

Improper Use of Internal Comments

- Makes code harder to understand and maintain
- Include internal comments which are:
  - redundant
  - misleading
  - outdated
  - excessive
  - unclear

White Space

- Refers to the blank space between code
- Enhance the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhance the functionality of the code by:
  - providing a standard format to make code run properly in a program

Proper Use of White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

NOTE: Use the Coding Examples Student Handout for references.

Improper Use of White Space, Spacing & Indentation

- Can lead to errors when editing or changing code because of confusion of which lines of code belong to which function
- Examples include:
  - extra whitespaces
  - extra blank lines
  - extra or inconsistent indentation
  - inconsistent spacing

NOTE: Use the Coding Examples Student Handout for references.

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Enhances the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Enhances the readability of the code by:
  - grouping statements together
  - indicating which statements belong to a particular block of code, such as a function, loop or conditional statement
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

Update to Content Accepted by SRP

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Works by the first letter of each word being written in lowercase
- Examples include:
  - snake_case
  - discount_amount

20

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

21

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")
- Begins with a capital letter and ends with a period

22

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

23

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

NOTE: Use the Coding Examples Student Handout for references.

24

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```

25

Coding Examples

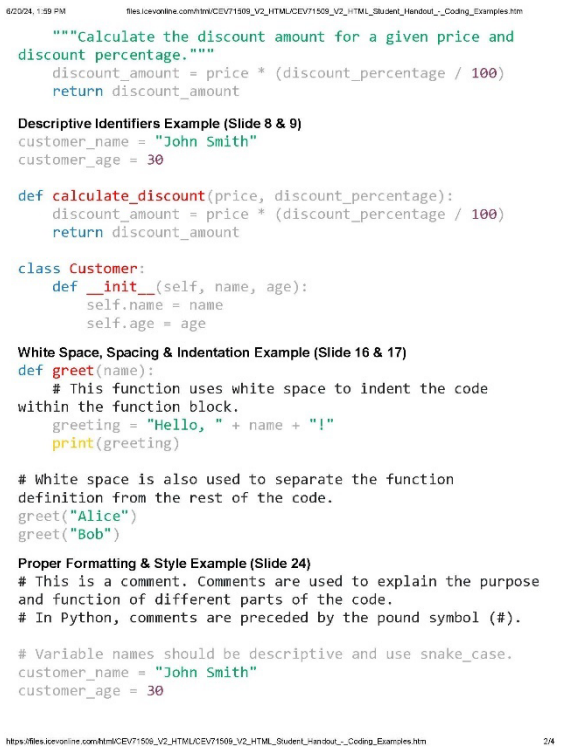
Programming Format Example (Slide 4)

```
import tokenize
import io
import sys

def check_formatting(filename):
    with open(filename, "rb") as f:
        try:
            tokens = tokenize(f.readline)
        except tokenize.TokenError as e:
            print("Formatting error:", e)
            return False
    return True

if len(sys.argv) != 2:
    print("Usage: python check_formatting.py you_file.py")
else:
    result = check_formatting(sys.argv[1])
    if result:
        print("Formatting is correct!")
    else:
        print("Formatting is incorrect.")
```

Standardized Programming Style Example (Slide 6)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30

# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
```

Update to Content Accepted by SRP



6/20/24, 1:59 PM files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

Descriptive Identifiers Example (Slide 8 & 9)

```
customer_name = "John Smith"
customer_age = 30

def calculate_discount(price, discount_percentage):
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

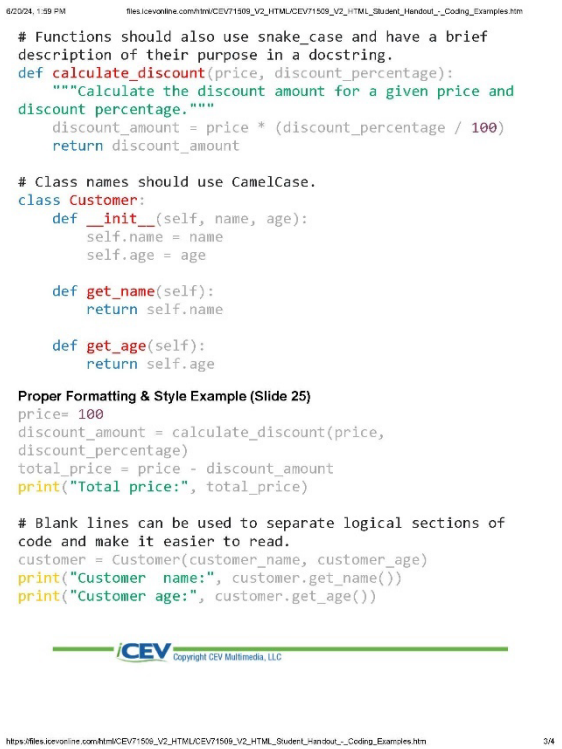
White Space, Spacing & Indentation Example (Slide 16 & 17)

```
def greet(name):
    # This function uses white space to indent the code
within the function block.
    greeting = "Hello, " + name + "!"
    print(greeting)

# White space is also used to separate the function
definition from the rest of the code.
greet("Alice")
greet("Bob")
```

Proper Formatting & Style Example (Slide 24)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30
```

https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm 2/4



6/20/24, 1:59 PM files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

# Class names should use CamelCase.
class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

Proper Formatting & Style Example (Slide 25)

```
price= 100
discount_amount = calculate_discount(price,
discount_percentage)
total_price = price - discount_amount
print("Total price:", total_price)

# Blank lines can be used to separate logical sections of
code and make it easier to read.
customer = Customer(customer_name, customer_age)
print("Customer  name:", customer.get_name())
print("Customer age:", customer.get_age())
```

iCEV Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm 3/4

(SE)(Breakout(s)) and (Citation Type(s))
(3)(C)(iv), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming with Proper Format and Style (Slides 4-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21642

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming with Proper Format and Style (Slides 4-25),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22294
In the Programming with Proper Format & Style PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Programming with Proper Format & Style lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Programming Format

- Refers to the way in which the source code of a program is written
  - this example is written in Python and includes indentation, spacing, line breaks and white space

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

Descriptive Identifiers

- Are names used for variables, functions and other elements of the code
- Are chosen to accurately and concisely describe their purpose or meaning
- Can make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future

Descriptive Identifiers

- Example:
  - in this code the identifiers "customer_name", "customer_age", "calculate_discount" and "Customer" are all descriptive

Internal Comments

- Are comments placed within the body of a function or other block of code
- Are used to explain the purpose or function of specific lines or blocks of code
- Are typically used to provide more detailed explanations or clarifications of the code
- Can be especially useful when the code is complex or does something which is not obvious

Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```


White Space

- Refers to the blank space between code
- Helps format the code in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

Update to Content Accepted by SRP

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

12

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Is used to group statements together and to indicate which statements belong to a particular block of code, such as a function, loop or conditional statement

13

White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

14

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

15

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

16

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Examples include:
  - snake_case
  - discount_amount

17

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

18

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")

19

Update to Content Accepted by SRP

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```


Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Programming Format

- Refers to the way in which the source code of a program is written
  - coding example is written in Python and includes indentation, spacing, line breaks and white space

NOTE: Use the Coding Examples Student Handout for references.

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

NOTE: Use the Coding Examples Student Handout for references.

Descriptive Identifiers

- Enhance the readability of the code by:
  - allowing other users to understand what the creator intended
  - increasing understanding of code
  - describing the purpose behind a code
- Enhance the functionality of the code by:
  - providing descriptions of the coding process
  - adding necessary details to the code to allow future users to adapt a code for a different use

Update to Content Accepted by SRP

Proper Use of Descriptive Identifiers

- Examples include:
  - "customer_name", "customer_age", "calculate_discount" and "Customer"

NOTE: Use the Coding Examples Student Handout for references.

8

Improper Use of Descriptive Identifiers

- Makes the descriptive identifiers hard to understand and creates confusion
- Examples include:
  - single letter identifiers
  - unclear abbreviations
  - inconsistent naming
  - misleading names
  - non-descriptive names

NOTE: Use the Coding Examples Student Handout for references.

9

Improper Use of Internal Comments

- Makes code harder to understand and maintain
- Include internal comments which are:
  - redundant
  - misleading
  - outdated
  - excessive
  - unclear

12

White Space

- Refers to the blank space between code
- Enhance the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhance the functionality of the code by:
  - providing a standard format to make code run properly in a program

13

Internal Comments

- Are comments placed within the body of a function or other block of code
- Enhance the readability of the code by:
  - explaining the purpose or function of specific lines or blocks of code
  - providing more detailed explanations or clarifications of the code
- Enhance the functionality of the code by:
  - providing descriptions when the code is complex or performs a function that is not obvious

10

Proper Use of Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

11

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Enhances the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

14

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Enhances the readability of the code by:
  - grouping statements together
  - indicating which statements belong to a particular block of code, such as a function, loop or conditional statement
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

15

Update to Content Accepted by SRP

Proper Use of White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

NOTE: Use the Coding Examples Student Handout for references.

16

Improper Use of White Space, Spacing & Indentation

- Can lead to errors when editing or changing code because of confusion of which lines of code belong to which function
- Examples include:
  - extra whitespaces
  - extra blank lines
  - extra or inconsistent indentation
  - inconsistent spacing

NOTE: Use the Coding Examples Student Handout for references.

17

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Works by the first letter of each word being written in lowercase
- Examples include:
  - snake_case
  - discount_amount

20

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

21

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

18

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

19

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")
- Begins with a capital letter and ends with a period

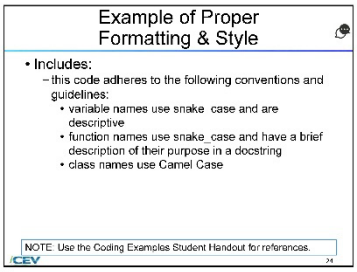
22

Block Structure

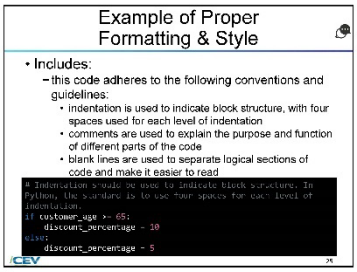
- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

23

Update to Content Accepted by SRP



24



25

Coding Examples

Programming Format Example (Slide 4)

```
import tokenize
import io
import sys

def check_formatting(filename):
    with open(filename, "rb") as f:
        try:
            tokens = tokenize(f.readline)
        except tokenize.TokenError as e:
            print("Formatting error:", e)
            return False
    return True

if len(sys.argv) != 2:
    print("Usage: python check_formatting.py you_file.py")
else:
    result = check_formatting(sys.argv[1])
    if result:
        print("Formatting is correct!")
    else:
        print("Formatting is incorrect.")
```

Standardized Programming Style Example (Slide 6)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30

# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
```



```
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

Descriptive Identifiers Example (Slide 8 & 9)

```
customer_name = "John Smith"
customer_age = 30

def calculate_discount(price, discount_percentage):
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

White Space, Spacing & Indentation Example (Slide 16 & 17)

```
def greet(name):
    # This function uses white space to indent the code
within the function block.
    greeting = "Hello, " + name + "!"
    print(greeting)

# White space is also used to separate the function
definition from the rest of the code.
greet("Alice")
greet("Bob")
```

Proper Formatting & Style Example (Slide 24)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30
```



```
# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

# Class names should use CamelCase.
class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

Proper Formatting & Style Example (Slide 25)

```
price= 100
discount_amount = calculate_discount(price,
discount_percentage)
total_price = price - discount_amount
print("Total price:", total_price)

# Blank lines can be used to separate logical sections of
code and make it easier to read.
customer = Customer(customer_name, customer_age)
print("Customer  name:", customer.get_name())
print("Customer age:", customer.get_age())
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(3)(C)(v), Narrative

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming with Proper Format and Style (Slides 4-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21642

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming with Proper Format and Style (Slides 4-25),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22294
In the Programming with Proper Format & Style PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Programming with Proper Format & Style lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Programming Format

- Refers to the way in which the source code of a program is written
  - this example is written in Python and includes indentation, spacing, line breaks and white space

4

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

5

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

6

Descriptive Identifiers

- Are names used for variables, functions and other elements of the code
- Are chosen to accurately and concisely describe their purpose or meaning
- Can make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future

7

Update to Content Accepted by SRP

Descriptive Identifiers

- Example:
  - in this code the identifiers "customer_name", "customer_age", "calculate_discount" and "Customer" are all descriptive

Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```


Internal Comments

- Are comments placed within the body of a function or other block of code
- Are used to explain the purpose or function of specific lines or blocks of code
- Are typically used to provide more detailed explanations or clarifications of the code
- Can be especially useful when the code is complex or does something which is not obvious

White Space

- Refers to the blank space between code
- Helps format the code in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Is used to group statements together and to indicate which statements belong to a particular block of code, such as a function, loop or conditional statement

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

Update to Content Accepted by SRP

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

16

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

18

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Examples include:
  - snake_case
  - discount_amount

17

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")

19

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

20

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```

22

Example of Proper Formatting & Style

- Include:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

21

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Programming Format

- Refers to the way in which the source code of a program is written
  - coding example is written in Python and includes indentation, spacing, line breaks and white space

NOTE: Use the Coding Examples Student Handout for references.

4

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

5

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

NOTE: Use the Coding Examples Student Handout for references.

6

Descriptive Identifiers

- Enhance the readability of the code by:
  - allowing other users to understand what the creator intended
  - increasing understanding of code
  - describing the purpose behind a code
- Enhance the functionality of the code by:
  - providing descriptions of the coding process
  - adding necessary details to the code to allow future users to adapt a code for a different use

7

Proper Use of Descriptive Identifiers

- Examples include:
  - "customer_name", "customer_age", "calculate_discount" and "Customer"

NOTE: Use the Coding Examples Student Handout for references.

8

Improper Use of Descriptive Identifiers

- Makes the descriptive identifiers hard to understand and creates confusion
- Examples include:
  - single letter identifiers
  - unclear abbreviations
  - inconsistent naming
  - misleading names
  - non-descriptive names

NOTE: Use the Coding Examples Student Handout for references.

9

Internal Comments

- Are comments placed within the body of a function or other block of code
- Enhance the readability of the code by:
  - explaining the purpose or function of specific lines or blocks of code
  - providing more detailed explanations or clarifications of the code
- Enhance the functionality of the code by:
  - providing descriptions when the code is complex or performs a function that is not obvious

10

Proper Use of Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

11

Update to Content Accepted by SRP

Improper Use of Internal Comments

- Makes code harder to understand and maintain
- Include internal comments which are:
  - redundant
  - misleading
  - outdated
  - excessive
  - unclear

White Space

- Refers to the blank space between code
- Enhance the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhance the functionality of the code by:
  - providing a standard format to make code run properly in a program

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Enhances the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Enhances the readability of the code by:
  - grouping statements together
  - indicating which statements belong to a particular block of code, such as a function, loop or conditional statement
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

Proper Use of White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

NOTE: Use the Coding Examples Student Handout for references.

Improper Use of White Space, Spacing & Indentation

- Can lead to errors when editing or changing code because of confusion of which lines of code belong to which function
- Examples include:
  - extra whitespaces
  - extra blank lines
  - extra or inconsistent indentation
  - inconsistent spacing

NOTE: Use the Coding Examples Student Handout for references.

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

Update to Content Accepted by SRP

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Works by the first letter of each word being written in lowercase
- Examples include:
  - snake_case
  - discount_amount

20

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

21

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")
- Begins with a capital letter and ends with a period

22

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

23

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

NOTE: Use the Coding Examples Student Handout for references.

24

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```

25

Coding Examples

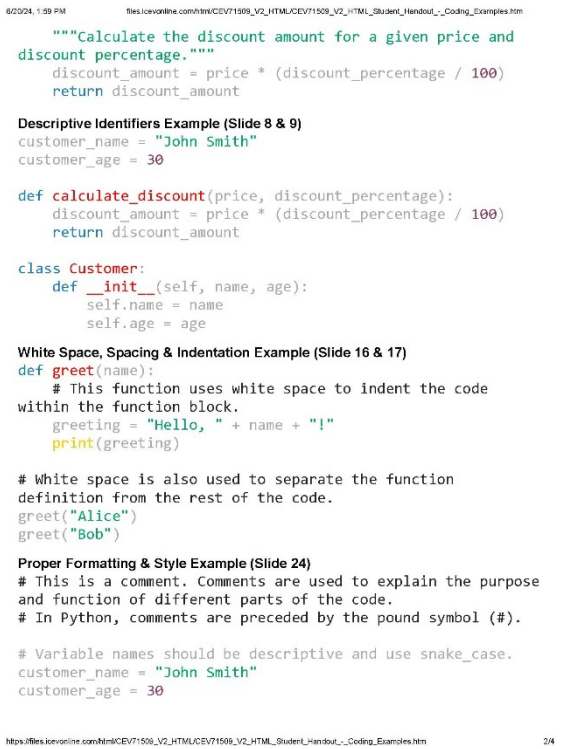
Programming Format Example (Slide 4)

```
import tokenize
import io
import sys

def check_formatting(filename):
    with open(filename, "rb") as f:
        try:
            tokens = tokenize(f.readline)
        except tokenize.TokenError as e:
            print("Formatting error:", e)
            return False
    return True

if len(sys.argv) != 2:
    print("Usage: python check_formatting.py you_file.py")
else:
    result = check_formatting(sys.argv[1])
    if result:
        print("Formatting is correct!")
    else:
        print("Formatting is incorrect.")
```

Standardized Programming Style Example (Slide 6)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30

# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
```

Update to Content Accepted by SRP



6/20/24, 1:59 PM files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

Descriptive Identifiers Example (Slide 8 & 9)

```
customer_name = "John Smith"
customer_age = 30

def calculate_discount(price, discount_percentage):
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

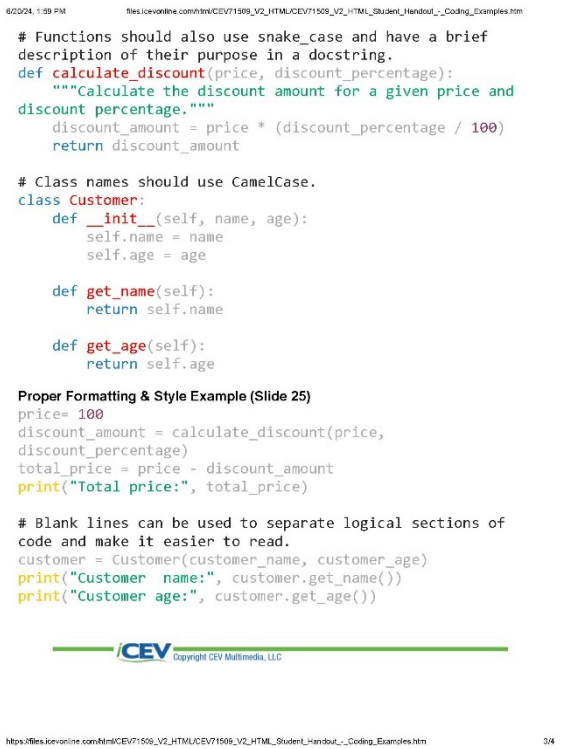
White Space, Spacing & Indentation Example (Slide 16 & 17)

```
def greet(name):
    # This function uses white space to indent the code
within the function block.
    greeting = "Hello, " + name + "!"
    print(greeting)

# White space is also used to separate the function
definition from the rest of the code.
greet("Alice")
greet("Bob")
```

Proper Formatting & Style Example (Slide 24)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30
```

https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm 2/4



6/20/24, 1:59 PM files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

# Class names should use CamelCase.
class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

Proper Formatting & Style Example (Slide 25)

```
price= 100
discount_amount = calculate_discount(price,
discount_percentage)
total_price = price - discount_amount
print("Total price:", total_price)

# Blank lines can be used to separate logical sections of
code and make it easier to read.
customer = Customer(customer_name, customer_age)
print("Customer  name:", customer.get_name())
print("Customer age:", customer.get_age())
```

iCEV Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm 3/4

(SE)(Breakout(s)) and (Citation Type(s))
(3)(C)(vi), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming with Proper Format and Style (Slides 4-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21642

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming with Proper Format and Style (Slides 4-25),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22294
In the Programming with Proper Format & Style PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Programming with Proper Format & Style lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Programming Format

- Refers to the way in which the source code of a program is written
  - this example is written in Python and includes indentation, spacing, line breaks and white space

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

Descriptive Identifiers

- Are names used for variables, functions and other elements of the code
- Are chosen to accurately and concisely describe their purpose or meaning
- Can make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future

Descriptive Identifiers

- Example:
  - in this code the identifiers "customer_name", "customer_age", "calculate_discount" and "Customer" are all descriptive

Internal Comments

- Are comments placed within the body of a function or other block of code
- Are used to explain the purpose or function of specific lines or blocks of code
- Are typically used to provide more detailed explanations or clarifications of the code
- Can be especially useful when the code is complex or does something which is not obvious

Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```


White Space

- Refers to the blank space between code
- Helps format the code in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

Update to Content Accepted by SRP

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

12

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Is used to group statements together and to indicate which statements belong to a particular block of code, such as a function, loop or conditional statement

13

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

16

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Examples include:
  - snake_case
  - discount_amount

17

White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

14

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

15

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

18

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")

19

Update to Content Accepted by SRP

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

Example of Proper Formatting & Style

- Include:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```


Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Programming Format

- Refers to the way in which the source code of a program is written
  - coding example is written in Python and includes indentation, spacing, line breaks and white space

NOTE: Use the Coding Examples Student Handout for references.

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

NOTE: Use the Coding Examples Student Handout for references.

Descriptive Identifiers

- Enhance the readability of the code by:
  - allowing other users to understand what the creator intended
  - increasing understanding of code
  - describing the purpose behind a code
- Enhance the functionality of the code by:
  - providing descriptions of the coding process
  - adding necessary details to the code to allow future users to adapt a code for a different use

Update to Content Accepted by SRP

Proper Use of Descriptive Identifiers

- Examples include:
  - "customer_name", "customer_age", "calculate_discount" and "Customer"

NOTE: Use the Coding Examples Student Handout for references.

8

Improper Use of Descriptive Identifiers

- Makes the descriptive identifiers hard to understand and creates confusion
- Examples include:
  - single letter identifiers
  - unclear abbreviations
  - inconsistent naming
  - misleading names
  - non-descriptive names

NOTE: Use the Coding Examples Student Handout for references.

9

Improper Use of Internal Comments

- Makes code harder to understand and maintain
- Include internal comments which are:
  - redundant
  - misleading
  - outdated
  - excessive
  - unclear

12

White Space

- Refers to the blank space between code
- Enhance the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhance the functionality of the code by:
  - providing a standard format to make code run properly in a program

13

Internal Comments

- Are comments placed within the body of a function or other block of code
- Enhance the readability of the code by:
  - explaining the purpose or function of specific lines or blocks of code
  - providing more detailed explanations or clarifications of the code
- Enhance the functionality of the code by:
  - providing descriptions when the code is complex or performs a function that is not obvious

10

Proper Use of Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

11

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Enhances the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

14

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Enhances the readability of the code by:
  - grouping statements together
  - indicating which statements belong to a particular block of code, such as a function, loop or conditional statement
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

15

Update to Content Accepted by SRP

Proper Use of White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

NOTE: Use the Coding Examples Student Handout for references.

Improper Use of White Space, Spacing & Indentation

- Can lead to errors when editing or changing code because of confusion of which lines of code belong to which function
- Examples include:
  - extra whitespaces
  - extra blank lines
  - extra or inconsistent indentation
  - inconsistent spacing

NOTE: Use the Coding Examples Student Handout for references.

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Works by the first letter of each word being written in lowercase
- Examples include:
  - snake_case
  - discount_amount

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

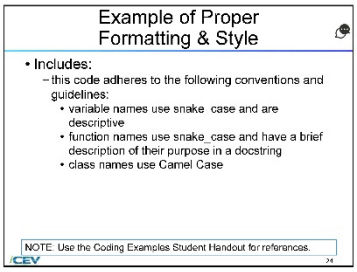
Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")
- Begins with a capital letter and ends with a period

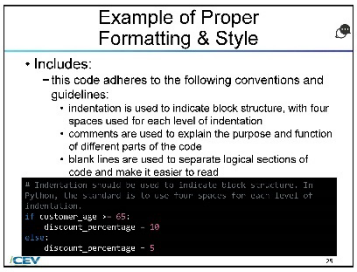
Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

Update to Content Accepted by SRP



24



25

Coding Examples

Programming Format Example (Slide 4)

```
import tokenize
import io
import sys

def check_formatting(filename):
    with open(filename, "rb") as f:
        try:
            tokens = tokenize(f.readline)
        except tokenize.TokenError as e:
            print("Formatting error:", e)
            return False
    return True

if len(sys.argv) != 2:
    print("Usage: python check_formatting.py you_file.py")
else:
    result = check_formatting(sys.argv[1])
    if result:
        print("Formatting is correct!")
    else:
        print("Formatting is incorrect.")
```

Standardized Programming Style Example (Slide 6)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30

# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
```



```
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

Descriptive Identifiers Example (Slide 8 & 9)

```
customer_name = "John Smith"
customer_age = 30

def calculate_discount(price, discount_percentage):
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

White Space, Spacing & Indentation Example (Slide 16 & 17)

```
def greet(name):
    # This function uses white space to indent the code
within the function block.
    greeting = "Hello, " + name + "!"
    print(greeting)

# White space is also used to separate the function
definition from the rest of the code.
greet("Alice")
greet("Bob")
```

Proper Formatting & Style Example (Slide 24)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30
```



```
# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

# Class names should use CamelCase.
class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

Proper Formatting & Style Example (Slide 25)

```
price= 100
discount_amount = calculate_discount(price,
discount_percentage)
total_price = price - discount_amount
print("Total price:", total_price)

# Blank lines can be used to separate logical sections of
code and make it easier to read.
customer = Customer(customer_name, customer_age)
print("Customer  name:", customer.get_name())
print("Customer age:", customer.get_age())
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(3)(C)(vii), Narrative

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming with Proper Format and Style (Slides 4-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21642

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming with Proper Format and Style (Slides 4-25),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22294
In the Programming with Proper Format & Style PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Programming with Proper Format & Style lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Programming Format

- Refers to the way in which the source code of a program is written
  - this example is written in Python and includes indentation, spacing, line breaks and white space

4

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

5

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

6

Descriptive Identifiers

- Are names used for variables, functions and other elements of the code
- Are chosen to accurately and concisely describe their purpose or meaning
- Can make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future

7

Update to Content Accepted by SRP

Descriptive Identifiers

- Example:
  - in this code the identifiers "customer_name", "customer_age", "calculate_discount" and "Customer" are all descriptive

8

Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

10

Internal Comments

- Are comments placed within the body of a function or other block of code
- Are used to explain the purpose or function of specific lines or blocks of code
- Are typically used to provide more detailed explanations or clarifications of the code
- Can be especially useful when the code is complex or does something which is not obvious

9

White Space

- Refers to the blank space between code
- Helps format the code in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

11

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

12

White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

14

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Is used to group statements together and to indicate which statements belong to a particular block of code, such as a function, loop or conditional statement

13

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

15

Update to Content Accepted by SRP

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

16

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

18

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Examples include:
  - snake_case
  - discount_amount

17

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")

19

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

20

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```

22

Example of Proper Formatting & Style

- Include:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

21

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Programming Format

- Refers to the way in which the source code of a program is written
  - coding example is written in Python and includes indentation, spacing, line breaks and white space

NOTE: Use the Coding Examples Student Handout for references.

4

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

5

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

NOTE: Use the Coding Examples Student Handout for references.

6

Descriptive Identifiers

- Enhance the readability of the code by:
  - allowing other users to understand what the creator intended
  - increasing understanding of code
  - describing the purpose behind a code
- Enhance the functionality of the code by:
  - providing descriptions of the coding process
  - adding necessary details to the code to allow future users to adapt a code for a different use

7

Proper Use of Descriptive Identifiers

- Examples include:
  - "customer_name", "customer_age", "calculate_discount" and "Customer"

NOTE: Use the Coding Examples Student Handout for references.

8

Improper Use of Descriptive Identifiers

- Makes the descriptive identifiers hard to understand and creates confusion
- Examples include:
  - single letter identifiers
  - unclear abbreviations
  - inconsistent naming
  - misleading names
  - non-descriptive names

NOTE: Use the Coding Examples Student Handout for references.

9

Internal Comments

- Are comments placed within the body of a function or other block of code
- Enhance the readability of the code by:
  - explaining the purpose or function of specific lines or blocks of code
  - providing more detailed explanations or clarifications of the code
- Enhance the functionality of the code by:
  - providing descriptions when the code is complex or performs a function that is not obvious

10

Proper Use of Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

11

Update to Content Accepted by SRP

Improper Use of Internal Comments

- Makes code harder to understand and maintain
- Include internal comments which are:
  - redundant
  - misleading
  - outdated
  - excessive
  - unclear

White Space

- Refers to the blank space between code
- Enhance the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhance the functionality of the code by:
  - providing a standard format to make code run properly in a program

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Enhances the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Enhances the readability of the code by:
  - grouping statements together
  - indicating which statements belong to a particular block of code, such as a function, loop or conditional statement
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

Proper Use of White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

NOTE: Use the Coding Examples Student Handout for references.

Improper Use of White Space, Spacing & Indentation

- Can lead to errors when editing or changing code because of confusion of which lines of code belong to which function
- Examples include:
  - extra whitespaces
  - extra blank lines
  - extra or inconsistent indentation
  - inconsistent spacing

NOTE: Use the Coding Examples Student Handout for references.

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

Update to Content Accepted by SRP

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Works by the first letter of each word being written in lowercase
- Examples include:
  - snake_case
  - discount_amount

20

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

21

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")
- Begins with a capital letter and ends with a period

22

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

23

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

NOTE: Use the Coding Examples Student Handout for references.

24

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```

25

Coding Examples

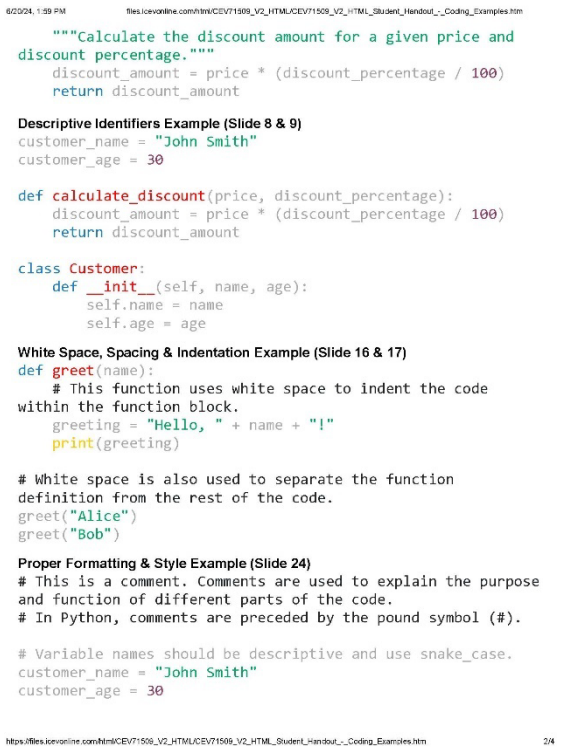
Programming Format Example (Slide 4)

```
import tokenize
import io
import sys

def check_formatting(filename):
    with open(filename, "rb") as f:
        try:
            tokens = tokenize(f.readline)
        except tokenize.TokenError as e:
            print("Formatting error:", e)
            return False
    return True

if len(sys.argv) != 2:
    print("Usage: python check_formatting.py you_file.py")
else:
    result = check_formatting(sys.argv[1])
    if result:
        print("Formatting is correct!")
    else:
        print("Formatting is incorrect.")
```

Standardized Programming Style Example (Slide 6)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30

# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
```

Update to Content Accepted by SRP



6/20/24, 1:59 PM files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

Descriptive Identifiers Example (Slide 8 & 9)

```
customer_name = "John Smith"
customer_age = 30

def calculate_discount(price, discount_percentage):
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

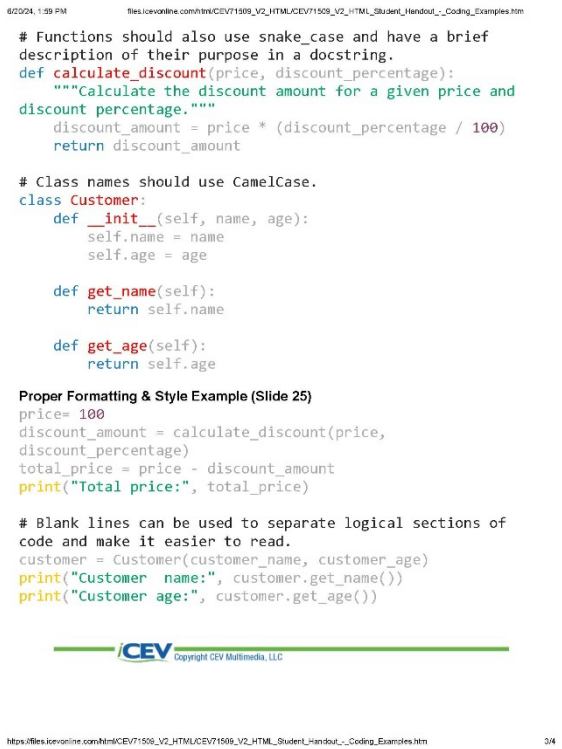
White Space, Spacing & Indentation Example (Slide 16 & 17)

```
def greet(name):
    # This function uses white space to indent the code
within the function block.
    greeting = "Hello, " + name + "!"
    print(greeting)

# White space is also used to separate the function
definition from the rest of the code.
greet("Alice")
greet("Bob")
```

Proper Formatting & Style Example (Slide 24)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30
```

https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm 2/4



6/20/24, 1:59 PM files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

# Class names should use CamelCase.
class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

Proper Formatting & Style Example (Slide 25)

```
price= 100
discount_amount = calculate_discount(price,
discount_percentage)
total_price = price - discount_amount
print("Total price:", total_price)

# Blank lines can be used to separate logical sections of
code and make it easier to read.
customer = Customer(customer_name, customer_age)
print("Customer  name:", customer.get_name())
print("Customer age:", customer.get_age())
```

iCEV Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm 3/4

(SE)(Breakout(s)) and (Citation Type(s))
(3)(C)(viii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming with Proper Format and Style (Slides 4-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21642

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming with Proper Format and Style (Slides 4-25),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22294
In the Programming with Proper Format & Style PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71509_V2_HTML/CEV71509_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Programming with Proper Format & Style lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Programming Format

- Refers to the way in which the source code of a program is written
  - this example is written in Python and includes indentation, spacing, line breaks and white space

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

Descriptive Identifiers

- Are names used for variables, functions and other elements of the code
- Are chosen to accurately and concisely describe their purpose or meaning
- Can make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future

Descriptive Identifiers

- Example:
  - in this code the identifiers "customer_name", "customer_age", "calculate_discount" and "Customer" are all descriptive

Internal Comments

- Are comments placed within the body of a function or other block of code
- Are used to explain the purpose or function of specific lines or blocks of code
- Are typically used to provide more detailed explanations or clarifications of the code
- Can be especially useful when the code is complex or does something which is not obvious

Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```


White Space

- Refers to the blank space between code
- Helps format the code in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

Update to Content Accepted by SRP

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Is used to indicate the structure of the code and to group statements together

12

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Is used to group statements together and to indicate which statements belong to a particular block of code, such as a function, loop or conditional statement

13

White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

14

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

15

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

16

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Examples include:
  - snake_case
  - discount_amount

17

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

18

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")

19

Update to Content Accepted by SRP

Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - variable names use snake_case and are descriptive
    - function names use snake_case and have a brief description of their purpose in a docstring
    - class names use Camel Case

Example of Proper Formatting & Style

- Includes:
  - this code adheres to the following conventions and guidelines:
    - indentation is used to indicate block structure, with four spaces used for each level of indentation
    - comments are used to explain the purpose and function of different parts of the code
    - blank lines are used to separate logical sections of code and make it easier to read

```
# Indentation should be used to indicate block structure. In
Python, the standard is to use four spaces for each level of
indentation.
if customer_age >= 65:
    discount_percentage = 10
else:
    discount_percentage = 5
```


Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Programming Format

- Refers to the way in which the source code of a program is written
  - coding example is written in Python and includes indentation, spacing, line breaks and white space

NOTE: Use the Coding Examples Student Handout for references.

Standardized Programming Style

- Refers to a set of conventions and guidelines for writing and formatting source code in a consistent manner
- Helps to make code more readable and easier to understand
- Ensures the code is consistent and follows established best practices

Standardized Programming Style

- Example:
  - this Python code provides comments, descriptive identifiers, white space, spacing and indentation

NOTE: Use the Coding Examples Student Handout for references.

Descriptive Identifiers

- Enhance the readability of the code by:
  - allowing other users to understand what the creator intended
  - increasing understanding of code
  - describing the purpose behind a code
- Enhance the functionality of the code by:
  - providing descriptions of the coding process
  - adding necessary details to the code to allow future users to adapt a code for a different use

Update to Content Accepted by SRP

Proper Use of Descriptive Identifiers

- Examples include:
  - "customer_name", "customer_age", "calculate_discount" and "Customer"

NOTE: Use the Coding Examples Student Handout for references.

8

Improper Use of Descriptive Identifiers

- Makes the descriptive identifiers hard to understand and creates confusion
- Examples include:
  - single letter identifiers
  - unclear abbreviations
  - inconsistent naming
  - misleading names
  - non-descriptive names

NOTE: Use the Coding Examples Student Handout for references.

9

Improper Use of Internal Comments

- Makes code harder to understand and maintain
- Include internal comments which are:
  - redundant
  - misleading
  - outdated
  - excessive
  - unclear

12

White Space

- Refers to the blank space between code
- Enhance the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhance the functionality of the code by:
  - providing a standard format to make code run properly in a program

13

Internal Comments

- Are comments placed within the body of a function or other block of code
- Enhance the readability of the code by:
  - explaining the purpose or function of specific lines or blocks of code
  - providing more detailed explanations or clarifications of the code
- Enhance the functionality of the code by:
  - providing descriptions when the code is complex or performs a function that is not obvious

10

Proper Use of Internal Comments

- Example:
  - the internal comment explains the purpose of the line of code which calculates the discount amount, which can be found after the # symbol within the code

```
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price
and discount percentage."""
    # Calculate the discount amount by multiplying the
price by the discount percentage.
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

11

Spacing

- Refers to the use of spaces, tabs and blank lines to separate elements of the code and to format it in a way which is easy to read and understand
- Enhances the readability of the code by:
  - indicating the structure of the code
  - grouping statements together
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

14

Indentation

- Refers to the use of spaces or tabs at the beginning of a line of code to indicate the block structure of the code
- Enhances the readability of the code by:
  - grouping statements together
  - indicating which statements belong to a particular block of code, such as a function, loop or conditional statement
- Enhances the functionality of the code by:
  - providing a standard format to make code run properly in a program

15

Update to Content Accepted by SRP

Proper Use of White Space, Spacing & Indentation

- Make it clear which lines of code belong to the function and which do not
  - the code within the greet function is indented by four spaces, which indicates it is part of the function block
  - the lines of code which call the greet function are not indented, which indicates they are not part of the function block

NOTE: Use the Coding Examples Student Handout for references.

16

Improper Use of White Space, Spacing & Indentation

- Can lead to errors when editing or changing code because of confusion of which lines of code belong to which function
- Examples include:
  - extra whitespaces
  - extra blank lines
  - extra or inconsistent indentation
  - inconsistent spacing

NOTE: Use the Coding Examples Student Handout for references.

17

Proper Formatting & Style

- Helps to make the code easier to read and understand, both for the person who wrote the code and for others who might need to read and work with the code in the future
- Helps to ensure the code is consistent and follows established conventions, making it easier to maintain and update
- Can help to prevent syntax errors and other issues which can arise from incorrect formatting

18

Syntax Error

- Occurs when code violates the rules of a programming language's syntax
- Refers to an error which occurs when the code is written in a way the compiler cannot understand
  - causes the code to fail to compile

Syntax: rules defining the structure of a coding language including the punctuation, words and symbols

19

Snake Case

- Refers to the convention of writing compound words by separating them with underscores
- Works by the first letter of each word being written in lowercase
- Examples include:
  - snake_case
  - discount_amount

20

Camel Case

- Is a naming convention used for variables, functions and other identifiers in code
- Works by capitalizing the first letter of each word, except the first word
- Examples include:
  - camelCase
  - employeeFirstName

21

Docstring

- Is a string literal used to define a module, function, class or method
- Is the first string in the definition of the object
- Is enclosed in triple quotes (""")
- Begins with a capital letter and ends with a period

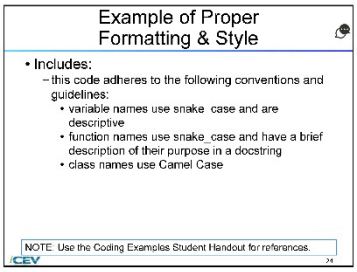
22

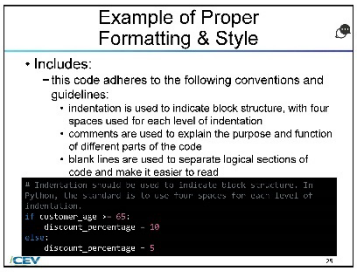
Block Structure

- Groups together a set of statements which belong together
- Executes grouped statements as a single unit

23

Update to Content Accepted by SRP





Coding Examples

Programming Format Example (Slide 4)

```
import tokenize
import io
import sys

def check_formatting(filename):
    with open(filename, "rb") as f:
        try:
            tokens = tokenize(f.readline)
        except tokenize.TokenError as e:
            print("Formatting error:", e)
            return False
    return True

if len(sys.argv) != 2:
    print("Usage: python check_formatting.py you_file.py")
else:
    result = check_formatting(sys.argv[1])
    if result:
        print("Formatting is correct!")
    else:
        print("Formatting is incorrect.")
```

Standardized Programming Style Example (Slide 6)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30

# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
```



```
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount
```

Descriptive Identifiers Example (Slide 8 & 9)

```
customer_name = "John Smith"
customer_age = 30

def calculate_discount(price, discount_percentage):
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

White Space, Spacing & Indentation Example (Slide 16 & 17)

```
def greet(name):
    # This function uses white space to indent the code
within the function block.
    greeting = "Hello, " + name + "!"
    print(greeting)

# White space is also used to separate the function
definition from the rest of the code.
greet("Alice")
greet("Bob")
```

Proper Formatting & Style Example (Slide 24)

```
# This is a comment. Comments are used to explain the purpose
and function of different parts of the code.
# In Python, comments are preceded by the pound symbol (#).

# Variable names should be descriptive and use snake_case.
customer_name = "John Smith"
customer_age = 30
```



```
# Functions should also use snake_case and have a brief
description of their purpose in a docstring.
def calculate_discount(price, discount_percentage):
    """Calculate the discount amount for a given price and
discount percentage."""
    discount_amount = price * (discount_percentage / 100)
    return discount_amount

# Class names should use CamelCase.
class Customer:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

Proper Formatting & Style Example (Slide 25)

```
price= 100
discount_amount = calculate_discount(price,
discount_percentage)
total_price = price - discount_amount
print("Total price:", total_price)

# Blank lines can be used to separate logical sections of
code and make it easier to read.
customer = Customer(customer_name, customer_age)
print("Customer  name:", customer.get_name())
print("Customer age:", customer.get_age())
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))

(3)(D)(i), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content

Update to Content Accepted by SRP

Visual Presentation (Slides 43-48),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21653

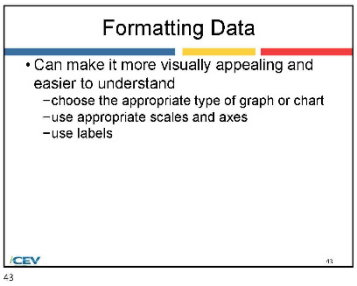
Project-Data Visualization Program, https://login.icevonline.com/download/cef8f65d-3b5b-495f-93aa-806de17f41d6

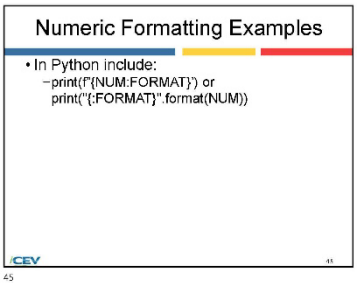
Description of the specific location and hyperlink to the exact location of the proposed new content
Visual Presentation Student Handout-Inputs, Outputs and Data Displays Examples,
https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm
This Student Handout is found in the Visual Presentation lesson beneath the Instructional Materials heading.

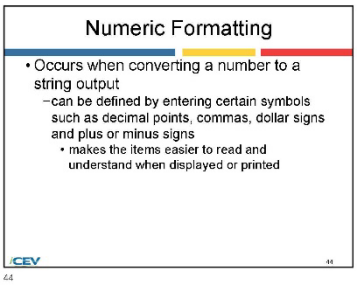
Coding Challenge: Simple Linear Regression,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22046/CEV71809_SIM01
Access to the interactive coding environment can be located beneath the Interactive Assignments heading by clicking the link to the Coding Challenge. Once clicked, the link will take you to a page prompting you to click Start. Select Start to view the Coding Challenge in the interactive environment.

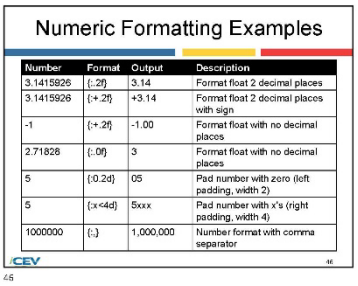
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.









Update to Content Accepted by SRP

6/20/2024

Numeric Formatting Examples

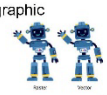
Number	Format	Output	Description
0.25	{:.2%}	25.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
13	{:10d}	13	Right aligned (default, width 10)
13	{:<10d}	13	Left aligned (width 10)
13	{:^10d}	13	Center aligned (width 10)

ICEV 47

47

Vector Graphics

- Are a type of digital image made up of mathematical formulas describing lines and shapes
- Can be resized without losing quality
  - are not made up of pixels like raster graphics (JPEG or PNG)
- Are typically created using graphic software like Adobe Illustrator® and Inkscape™



ICEV 48

48

3

Project - Data Visualization Program
Visual Presentation

1 of 1

6/20/24, 11:54 PM My ICEV | Computer Science I (Post Adoption Sample) | Project - Data Visualization Program

Project Overview:
You will create a program which displays a data set in a visually appealing way using appropriate formatting styles and graphics.

Directions:

1. ***On your computer, access the matplotlib library. The matplotlib library is not in the Standard Python Library. You can find an online compiler by entering "online matplotlib compiler" in a search engine.***
2. ***Enter and run the following code.***

```
import matplotlib.pyplot as plt

# Sample Data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a line chart
plt.plot(x, y)

# Add Labels and title
plt.xlabel('X-axis label')
plt.xlabel('Y-axis label')
plt.xlabel('Title of the chart')

# Display the chart
plt.show()
```

3. ***This code will create a line graph with the given x and y data, then add labels and a title to the chart. Change the data and customize the chart by adding meaningful axis labels and a meaningful title. Adjust the colors, markers and other properties of the chart elements. Feel free to increase the quantity of data points. Format your code for readability. Watching a video on matplotlib plots for adding these extra features may be helpful.***
4. ***Analyze the resulting graph and answer the following questions in a paragraph:***
   - ***How do the changes you made impact the message the graph imparts***
   - ***Does the formatting of the code and the numeric formatting impact the output***
   - ***How did formatting the data improve the numeric display***
5. ***Once complete, submit your Project. You can find a Rubric at the end of this Project.***

https://login.icevonline.com/mycourses/ADCCOMPU001/lesson/21653/CEV71620_ProjectD1 2/3

Update to Content Accepted by SRP

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

Inputs, Outputs & Data Displays Examples

Creating Outputs

To create a simple text output, the 'print' function is most often used, but it can have several variations to change how the data is displayed. Below are examples of how to create a text output with the print function for several different scenarios. Keep in mind when creating text output, the code in green is what will be displayed for the user.

```
# Basic text output using the print function
print("Hello World")

# Specific information with text explain what the information shows
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)

# Formatting specific results of a function
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Multiple items in a text display
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)
```

Properly Labeling and Displaying Outputs

When creating text output, it is important to make sure the user will understand what is being displayed. Properly labeling output is an important part of visual representation and can make code more user-friendly. Below are a few examples of labels to make the code easier to read and understand.

```
# The labels 'Name' and 'Favorite Food' make the display more clear
and help define what the terms relate to
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 1/6

6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

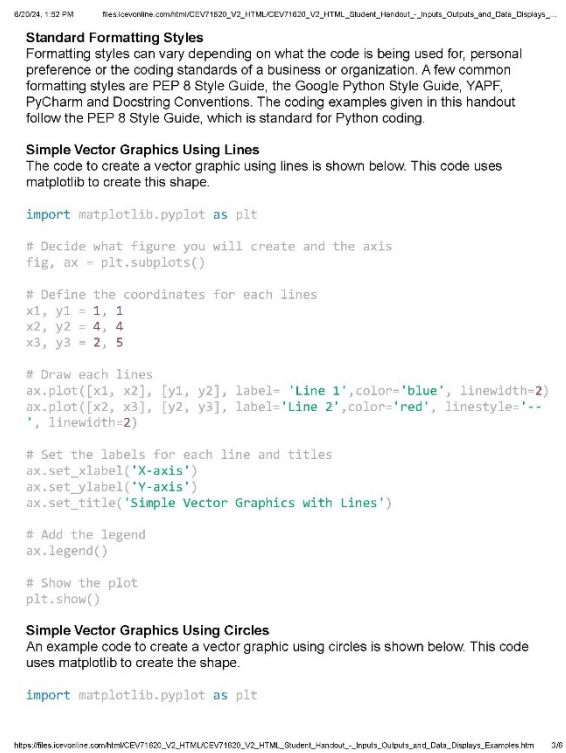
```
# In this example, putting the information that is calculated in a
sentence helps convey what function was performed
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Adding labels that show what x, y, and z correlate makes the display
easier to read
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)

# Adding units when displaying specific information can also make text
displays more clear
num1 = 7
num2 = 4
print(f"Loaded Weight: {num1}lbs Unloaded Weight: {num2}lbs")

# When there are several types of complex information being displayed,
descriptive labels can help clear any confusion
name = "Benjamin"
number = 4
transactions = 13
print(f"Customer: {name}")
print(f"Customer Number for December: {number}")
print(f"Number of Transaction for November: {transactions}")
```

Interactive Input Interfaces

When creating interactive input interfaces with relevant user prompts to acquire data from a user, it is important the code is readable and easy to understand. This code requires information from the user to display a certain message or action. An example of a code that requires user input is shown below.

```
# The prompt below asks for inputs from the user to display a welcome
message
name = input("Enter your name:")

# Now the text will be displayed with the information that the user
added
print(f"Welcome {name}!")
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 2/6



6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

Standard Formatting Styles

Formatting styles can vary depending on what the code is being used for, personal preference or the coding standards of a business or organization. A few common formatting styles are PEP 8 Style Guide, the Google Python Style Guide, YAPF, PyCharm and Docstring Conventions. The coding examples given in this handout follow the PEP 8 Style Guide, which is standard for Python coding.

Simple Vector Graphics Using Lines

The code to create a vector graphic using lines is shown below. This code uses matplotlib to create this shape.

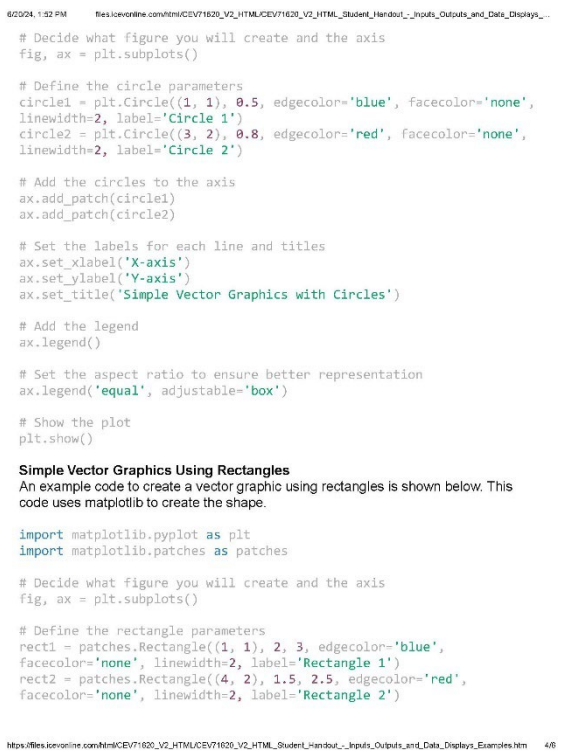
```
import matplotlib.pyplot as plt

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the coordinates for each lines
x1, y1 = 1, 1
x2, y2 = 4, 4
x3, y3 = 2, 5

# Draw each lines
ax.plot([x1, x2], [y1, y2], label= 'Line 1',color='blue', linewidth=2)
ax.plot([x2, x3], [y2, y3], label='Line 2',color='red', linestyle='--
', linewidth=2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Lines')

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

Simple Vector Graphics Using Circles

An example code to create a vector graphic using circles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 3/6



6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

```
# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blue', facecolor='none',
linewidth=2, label='Circle 1')
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red', facecolor='none',
linewidth=2, label='Circle 2')

# Add the circles to the axis
ax.add_patch(circle1)
ax.add_patch(circle2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Circles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Simple Vector Graphics Using Rectangles

An example code to create a vector graphic using rectangles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolor='blue',
facecolor='none', linewidth=2, label='Rectangle 1')
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edgecolor='red',
facecolor='none', linewidth=2, label='Rectangle 2')
```

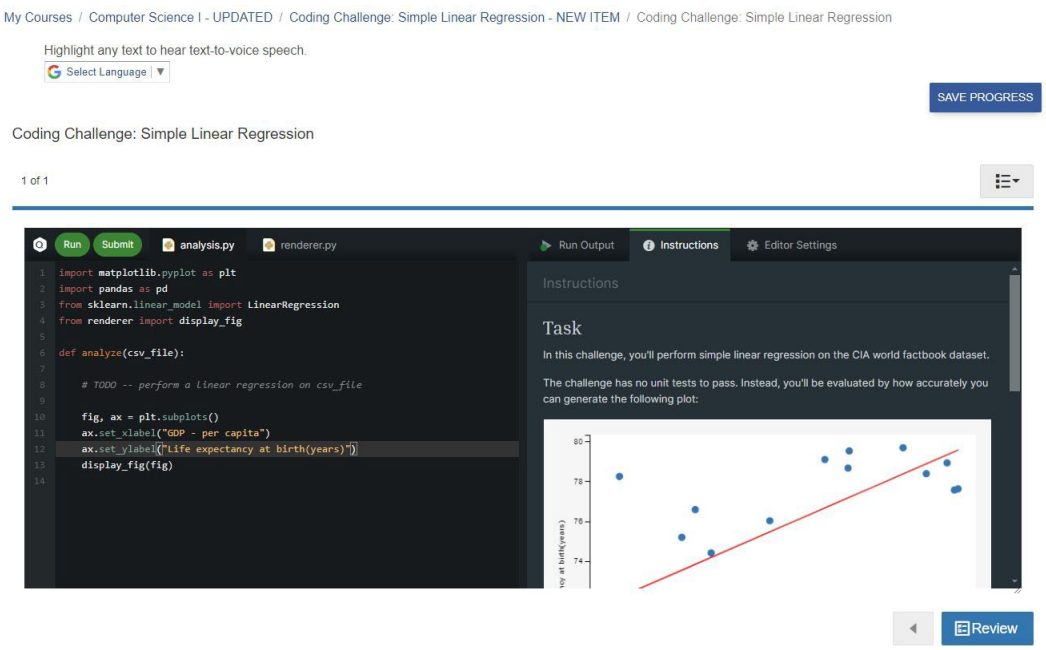
https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 4/6

Update to Content Accepted by SRP

```
# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Rectangles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Matplotlib is a widely-used library for creating static, animated and interactive visualizations. This example will showcase the inputs, outputs and data display for a line, circle or rectangle.

```
# dialog.py

"""Dialog-style application."""

import sys
from PyQt6.QtWidgets import (
    QApplication,
    QDialog,
    QDialogButtonBox,
    QFormLayout,
    QLineEdit,
    QVBoxLayout,
)

class Window(QDialog):
    def __init__(self):
        super().__init__(parent=None)
```



(SE)(Breakout(s)) and (Citation Type(s))
(3)(E)(i), Narrative

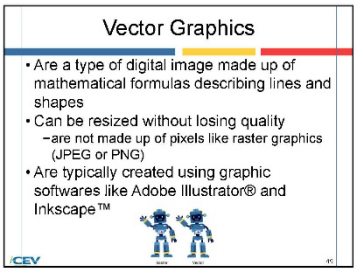
Description of the specific location and hyperlink to the exact location of currently adopted content
Visual Presentation (Slides 49-54),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21653

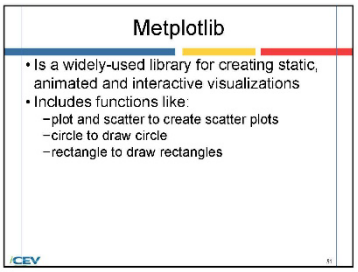
Update to Content Accepted by SRP

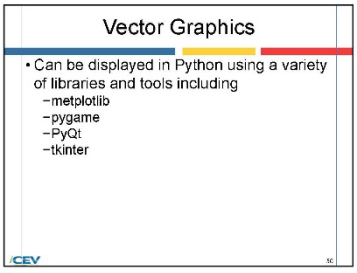
Description of the specific location and hyperlink to the exact location of the proposed new content
Visual Presentation Student Handout-Inputs, Outputs and Data Displays Examples, https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm
This Student Handout is found in the Visual Presentation lesson beneath the Instructional Materials heading.

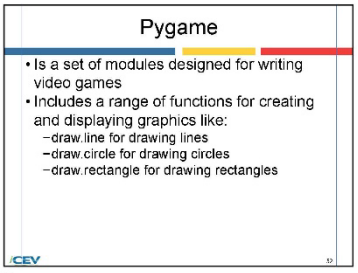
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



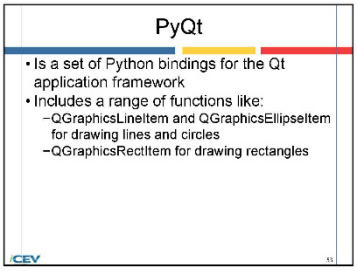




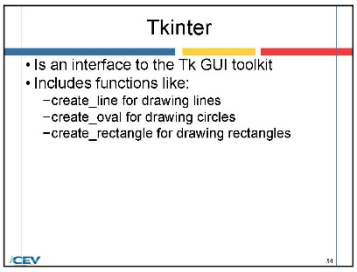


Update to Content Accepted by SRP

6/21/2024



53



54

3

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Inputs, Outputs & Data Displays Examples

Creating Outputs

To create a simple text output, the 'print' function is most often used, but it can have several variations to change how the data is displayed. Below are examples of how to create a text output with the print function for several different scenarios. Keep in mind when creating text output, the code in green is what will be displayed for the user.

```
# Basic text output using the print function
print("Hello World")

# Specific information with text explain what the information shows
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)

# Formatting specific results of a function
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Multiple items in a text display
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)
```

Properly Labeling and Displaying Outputs

When creating text output, it is important to make sure the user will understand what is being displayed. Properly labeling output is an important part of visual representation and can make code more user-friendly. Below are a few examples of labels to make the code easier to read and understand.

```
# The labels 'Name' and 'Favorite Food' make the display more clear
and help define what the terms relate to
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)
```



```
# In this example, putting the information that is calculated in a
sentence helps convey what function was performed
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Adding labels that show what x, y, and z correlate makes the display
easier to read
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)

# Adding units when displaying specific information can also make text
displays more clear
num1 = 7
num2 = 4
print(f"Loaded Weight: {num1}lbs Unloaded Weight: {num2}lbs")

# When there are several types of complex information being displayed,
descriptive labels can help clear any confusion
name = "Benjamin"
number = 4
transactions = 13
print(f"Customer: {name}")
print(f"Customer Number for December: {number}")
print(f"Number of Transaction for November: {transactions}")
```

Interactive Input Interfaces

When creating interactive input interfaces with relevant user prompts to acquire data from a user, it is important the code is readable and easy to understand. This code requires information from the user to display a certain message or action. An example of a code that requires user input is shown below.

```
# The prompt below asks for inputs from the user to display a welcome
message
name = input("Enter your name:")

# Now the text will be displayed with the information that the user
added
print(f"Welcome {name}!")
```

Update to Content Accepted by SRP

Standard Formatting Styles
Formatting styles can vary depending on what the code is being used for, personal preference or the coding standards of a business or organization. A few common formatting styles are PEP 8 Style Guide, the Google Python Style Guide, YAPF, PyCharm and Docstring Conventions. The coding examples given in this handout follow the PEP 8 Style Guide, which is standard for Python coding.

Simple Vector Graphics Using Lines
The code to create a vector graphic using lines is shown below. This code uses matplotlib to create this shape.

```
import matplotlib.pyplot as plt

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the coordinates for each lines
x1, y1 = 1, 1
x2, y2 = 4, 4
x3, y3 = 2, 5

# Draw each lines
ax.plot([x1, x2], [y1, y2], label= 'Line 1',color='blue', linewidth=2)
ax.plot([x2, x3], [y2, y3], label='Line 2',color='red', linestyle='--
', linewidth=2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Lines')

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

Simple Vector Graphics Using Circles
An example code to create a vector graphic using circles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
```



```
# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blue', facecolor='none',
linewidth=2, label='Circle 1')
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red', facecolor='none',
linewidth=2, label='Circle 2')

# Add the circles to the axis
ax.add_patch(circle1)
ax.add_patch(circle2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Circles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Simple Vector Graphics Using Rectangles
An example code to create a vector graphic using rectangles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolor='blue',
facecolor='none', linewidth=2, label='Rectangle 1')
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edgecolor='red',
facecolor='none', linewidth=2, label='Rectangle 2')
```



```
# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Rectangles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Matplotlib is a widely-used library for creating static, animated and interactive visualizations. This example will showcase the inputs, outputs and data display for a line, circle or rectangle.

```
# dialog.py

"""Dialog-style application."""

import sys
from PyQt6.QtWidgets import (
    QApplication,
    QDialog,
    QDialogButtonBox,
    QFormLayout,
    QLineEdit,
    QVBoxLayout,
)

class Window(QDialog):
    def __init__(self):
        super().__init__(parent=None)
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(3)(E)(ii), Narrative & Activity

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Visual Presentation (Slides 49-54),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21653

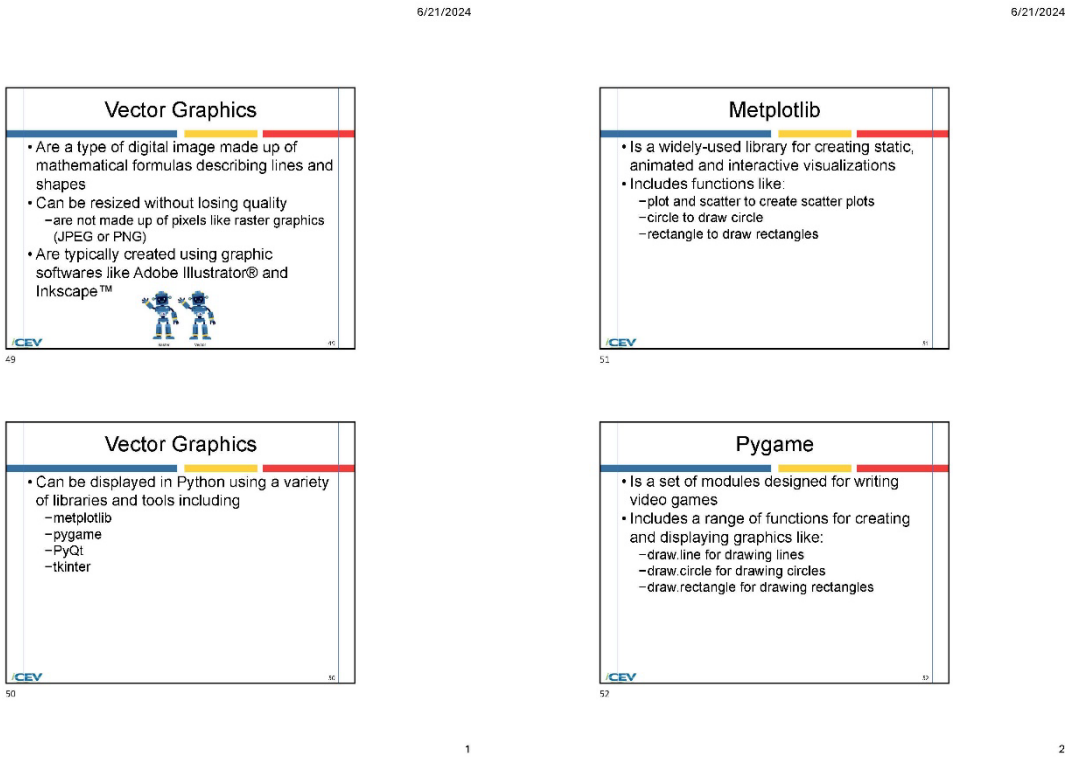
Project- Data Visualization Program,
https://files.icevonline.com/html/CEV71620_TXP24/CEV71620_TXP24_Project_-_Data_Visualization_Program.htm

Description of the specific location and hyperlink to the exact location of the proposed new content
Visual Presentation Student Handout-Inputs, Outputs and Data Displays Examples,
https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm
This Student Handout is found in the Visual Presentation lesson beneath the Instructional Materials heading.

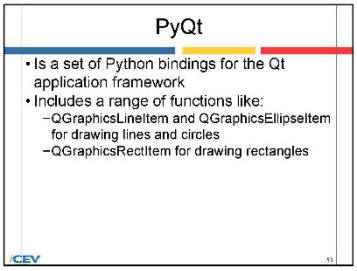
Project- Data Visualization Program,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22314/CEV71620_V2_Project01
This Project is found in the Visual Presentation lesson beneath the Interactive Assignments heading. After clicking the link to the Project, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Project.

Screenshot of Currently Adopted Content

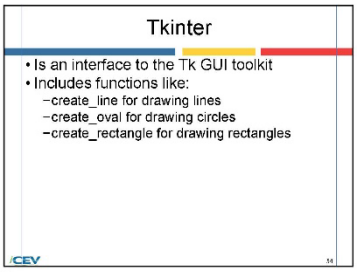
Insert a screenshot of your currently adopted content.



Update to Content Accepted by SRP



53



54

Project - Data Visualization Program

Visual Presentation

1 of 1

Directions:

1. On your computer, access the matplotlib library. The matplotlib library is not in the Standard Python Library. You can find an online compiler by entering "online matplotlib compiler" in a search engine.
2. Enter and run the following code.

```
import matplotlib.pyplot as plt

# Sample Data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a line chart
plt.plot(x, y)

# Add Labels and title
plt.xlabel('X-axis label')
plt.xlabel('Y-axis label')
plt.xlabel('Title of the chart')

# Display the chart
plt.show()
```

3. This code will create a line graph with the given x and y data, then add labels and a title to the chart. Change the data and customize the chart by adding meaningful axis labels and a meaningful title. Adjust the colors, markers and other properties of the chart elements. Feel free to increase the quantity of data points. Format your code for readability. Watching a video on matplotlib plots for adding these extra features may be helpful.
4. Analyze the resulting graph and answer the following questions in a paragraph:
   - How do the changes you made impact the message the graph imparts
   - Does the formatting of the code and the numeric formatting impact the output
   - How did formatting the data improve the numeric display
5. Once complete, submit your Project. You can find a *Rubric* at the end of this Project.

Rubric

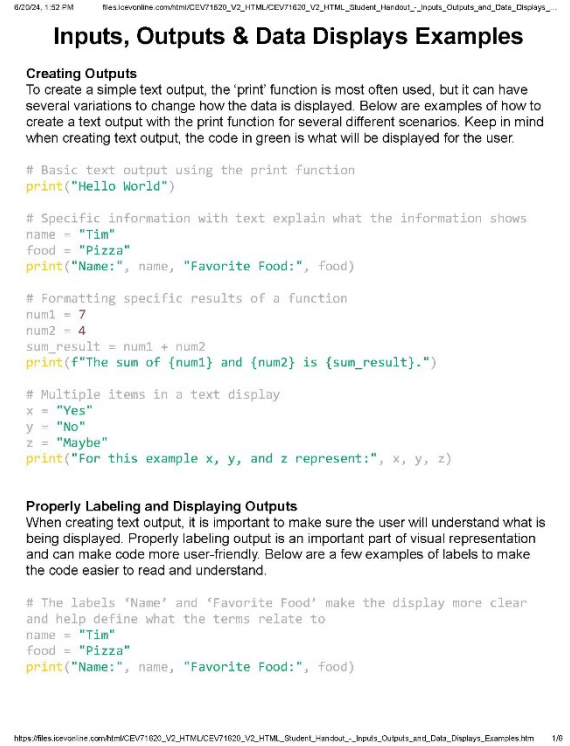
Description	Possible Points
Concept & Understanding: • Understanding of the concept is clearly evident • Effective formatting techniques were applied to the code • Understanding of formatting data to improve numeric displays was apparent • Code was clearly wrote after formatting was applied	90
Craftmanship: • Paragraph was well written using proper grammar	5
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	5
Total Points	100

Review

Update to Content Accepted by SRP

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.



6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

Inputs, Outputs & Data Displays Examples

Creating Outputs
To create a simple text output, the 'print' function is most often used, but it can have several variations to change how the data is displayed. Below are examples of how to create a text output with the print function for several different scenarios. Keep in mind when creating text output, the code in green is what will be displayed for the user.

```
# Basic text output using the print function
print("Hello World")

# Specific information with text explain what the information shows
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)

# Formatting specific results of a function
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Multiple items in a text display
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)
```

Properly Labeling and Displaying Outputs
When creating text output, it is important to make sure the user will understand what is being displayed. Properly labeling output is an important part of visual representation and can make code more user-friendly. Below are a few examples of labels to make the code easier to read and understand.

```
# The labels 'Name' and 'Favorite Food' make the display more clear
and help define what the terms relate to
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 1/6

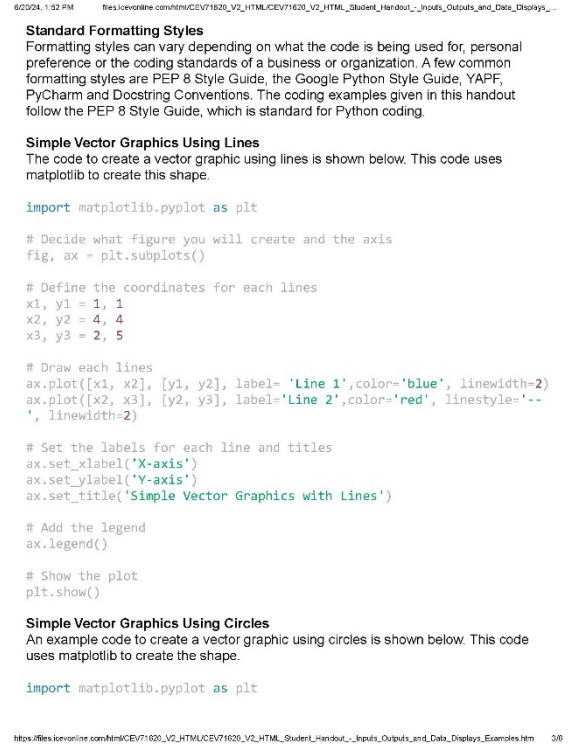
6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

```
# In this example, putting the information that is calculated in a
sentence helps convey what function was performed
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Adding labels that show what x, y, and z correlate makes the display
easier to read
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)

# Adding units when displaying specific information can also make text
displays more clear
num1 = 7
num2 = 4
print(f"Loaded Weight: {num1}lbs Unloaded Weight: {num2}lbs")

# When there are several types of complex information being displayed,
descriptive labels can help clear any confusion
name = "Benjamin"
number = 4
transactions = 13
print(f"Customer: {name}")
print(f"Customer Number for December: {number}")
print(f"Number of Transaction for November: {transactions}")
```

Interactive Input Interfaces
When creating interactive input interfaces with relevant user prompts to acquire data from a user, it is important the code is readable and easy to understand. This code requires information from the user to display a certain message or action. An example of a code that requires user input is shown below.

```
# The prompt below asks for inputs from the user to display a welcome
message
name = input("Enter your name:")

# Now the text will be displayed with the information that the user
added
print(f"Welcome {name}!")
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 2/6



6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

Standard Formatting Styles
Formatting styles can vary depending on what the code is being used for, personal preference or the coding standards of a business or organization. A few common formatting styles are PEP 8 Style Guide, the Google Python Style Guide, YAPF, PyCharm and Docstring Conventions. The coding examples given in this handout follow the PEP 8 Style Guide, which is standard for Python coding.

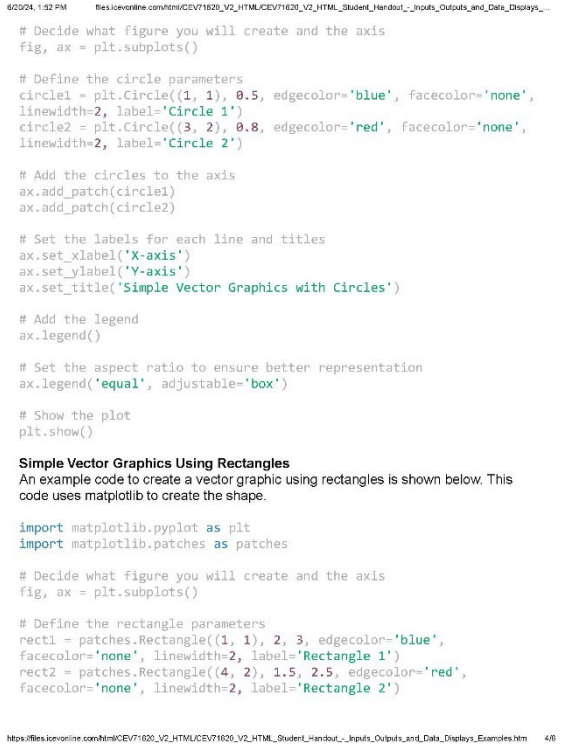
Simple Vector Graphics Using Lines
The code to create a vector graphic using lines is shown below. This code uses matplotlib to create this shape.

```
import matplotlib.pyplot as plt

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the coordinates for each lines
x1, y1 = 1, 1
x2, y2 = 4, 4
x3, y3 = 2, 5

# Draw each lines
ax.plot([x1, x2], [y1, y2], label= 'Line 1',color='blue', linewidth=2)
ax.plot([x2, x3], [y2, y3], label='Line 2',color='red', linestyle='--
', linewidth=2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Lines')

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

Simple Vector Graphics Using Circles
An example code to create a vector graphic using circles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 3/6



6/20/24, 1:52 PM files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_...

```
# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blue', facecolor='none',
linewidth=2, label='Circle 1')
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red', facecolor='none',
linewidth=2, label='Circle 2')

# Add the circles to the axis
ax.add_patch(circle1)
ax.add_patch(circle2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Circles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Simple Vector Graphics Using Rectangles
An example code to create a vector graphic using rectangles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolor='blue',
facecolor='none', linewidth=2, label='Rectangle 1')
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edgecolor='red',
facecolor='none', linewidth=2, label='Rectangle 2')
```

https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm 4/6

Update to Content Accepted by SRP


```
# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Rectangles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Matplotlib is a widely-used library for creating static, animated and interactive visualizations. This example will showcase the inputs, outputs and data display for a line, circle or rectangle.

```
# dialog.py

"""Dialog-style application."""

import sys
from PyQt6.QtWidgets import (
    QApplication,
    QDialog,
    QDialogButtonBox,
    QFormLayout,
    QLineEdit,
    QVBoxLayout,
)

class Window(QDialog):
    def __init__(self):
        super().__init__(parent=None)
```

CEV Copyright CEV Multimedia, LLC

SAVE PROGRESS

Project - Data Visualization Program

Visual Presentation

1 of 1

Project Overview:

You will create a program which displays a data set in a visually appealing way using appropriate formatting styles and graphics.

Directions:

1. On your computer, access the matplotlib library. The matplotlib library is not in the Standard Python Library. You can find an online compiler by entering "online matplotlib compiler" in a search engine.
2. Enter and run each of the following codes. The codes will create a line graph, circle graph and rectangle graph with the given x and y data. Add labels and a title to the chart. Change the data and customize each chart by adding meaningful axis labels and a meaningful title. Adjust the colors, markers and other properties of the chart elements. Feel free to increase the quantity of data points. Format your code for readability. Watching a video on matplotlib plots for adding these extra features may be helpful.
3. Line Graph

```
import matplotlib.pyplot as plt

# Sample Data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a line chart
plt.plot(x, y)

# Add Labels and title
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title of the chart')

# Display the chart
plt.show()
```

4. Circle Graph

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Define the figure and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blu
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red

# Add the circles to the axis
ax.add_patch(circle1)
```



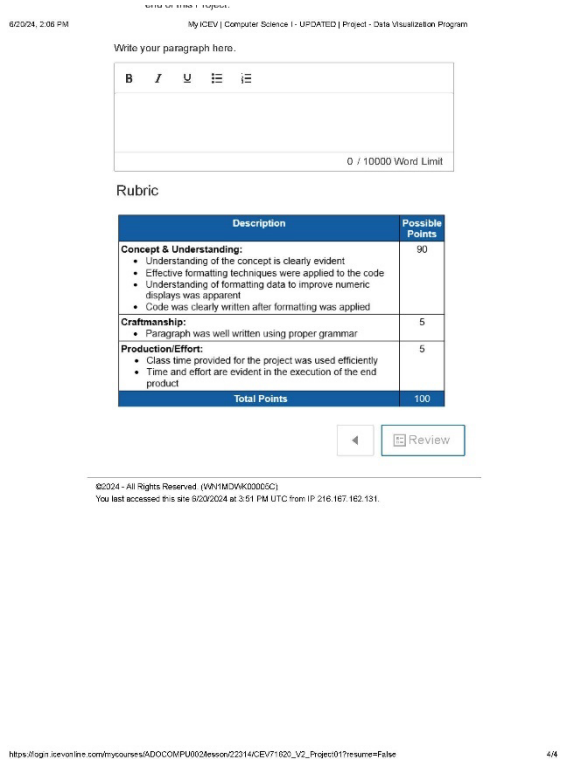
```
# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Title of chart')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better represen
ax.legend('equal', adjustable='box')

# Show the plot plt.show()
```

5. Rectangle Graph

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axi
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolo
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edge

# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Title of chart')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better represen
ax.legend('equal', adjustable='box')

# Show the plot plt.show()
```

6. Analyze the resulting graphs and answer the following questions in a paragraph:
   - How do the changes you made impact the message the graph imparts
   - Does the formatting of the code and the numeric formatting impact the output
   - How did formatting the data improve the numeric display

Update to Content Accepted by SRP



(SE)(Breakout(s)) and (Citation Type(s))
(3)(E)(iii), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Visual Presentation (Slides 49-54),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21653

Project- Data Visualization Program,
https://files.icevonline.com/html/CEV71620_TXP24/CEV71620_TXP24_Project_-_Data_Visualization_Program.htm

Description of the specific location and hyperlink to the exact location of the proposed new content
Visual Presentation Student Handout-Inputs, Outputs and Data Displays Examples,
https://files.icevonline.com/html/CEV71620_V2_HTML/CEV71620_V2_HTML_Student_Handout_-_Inputs_Outputs_and_Data_Displays_Examples.htm
This Student Handout is found in the Visual Presentation lesson beneath the Instructional Materials heading.

Project- Data Visualization Program,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22314/CEV71620_V2_Project01
This Project is found in the Visual Presentation lesson beneath the Interactive Assignments heading. After clicking the link to the Project, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Project.

Update to Content Accepted by SRP

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Vector Graphics

- Are a type of digital image made up of mathematical formulas describing lines and shapes
- Can be resized without losing quality
  - are not made up of pixels like raster graphics (JPEG or PNG)
- Are typically created using graphic softwares like Adobe Illustrator® and Inkscape™

CEV

49

Vector Graphics

- Can be displayed in Python using a variety of libraries and tools including
  - metplotlib
  - pygame
  - PyQt
  - tkinter

CEV

50

PyQt

- Is a set of Python bindings for the Qt application framework
- Includes a range of functions like:
  - QGraphicsLineItem and QGraphicsEllipseItem for drawing lines and circles
  - QGraphicsRectItem for drawing rectangles

CEV

53

Tkinter

- Is an interface to the Tk GUI toolkit
- Includes functions like:
  - create_line for drawing lines
  - create_oval for drawing circles
  - create_rectangle for drawing rectangles

CEV

54

Metplotlib

- Is a widely-used library for creating static, animated and interactive visualizations
- Includes functions like:
  - plot and scatter to create scatter plots
  - circle to draw circle
  - rectangle to draw rectangles

CEV

51

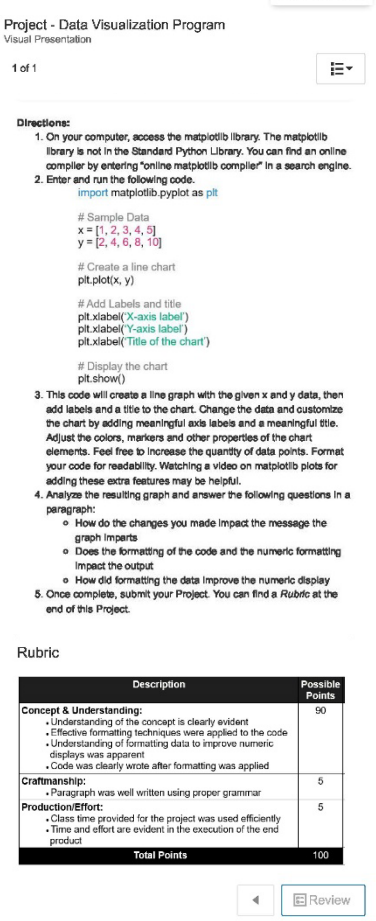
Pygame

- Is a set of modules designed for writing video games
- Includes a range of functions for creating and displaying graphics like:
  - draw.line for drawing lines
  - draw.circle for drawing circles
  - draw.rectangle for drawing rectangles

CEV

52

Update to Content Accepted by SRP



Project - Data Visualization Program
Visual Presentation

1 of 1

Directions:

1. On your computer, access the matplotlib library. The matplotlib library is not in the Standard Python Library. You can find an online compiler by entering "online matplotlib compiler" in a search engine.
2. Enter and run the following code.

```
import matplotlib.pyplot as plt

# Sample Data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a line chart
plt.plot(x, y)

# Add Labels and title
plt.xlabel('X-axis label')
plt.xlabel('Y-axis label')
plt.xlabel('Title of the chart')

# Display the chart
plt.show()
```

3. This code will create a line graph with the given x and y data, then add labels and a title to the chart. Change the data and customize the chart by adding meaningful axis labels and a meaningful title. Adjust the colors, markers and other properties of the chart elements. Feel free to increase the quantity of data points. Format your code for readability. Watching a video on matplotlib plots for adding these extra features may be helpful.
4. Analyze the resulting graph and answer the following questions in a paragraph:
   - How do the changes you made impact the message the graph imparts
   - Does the formatting of the code and the numeric formatting impact the output
   - How did formatting the data improve the numeric display
5. Once complete, submit your Project. You can find a *Rubric* at the end of this Project.

Rubric

Description	Possible Points
Concept & Understanding: • Understanding of the concept is clearly evident • Effective formatting techniques were applied to the code • Understanding of formatting data to improve numeric displays was apparent • Code was clearly wrote after formatting was applied	90
Craftmanship: • Paragraph was well written using proper grammar	5
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	5
Total Points	100

Review

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Inputs, Outputs & Data Displays Examples

Creating Outputs
To create a simple text output, the 'print' function is most often used, but it can have several variations to change how the data is displayed. Below are examples of how to create a text output with the print function for several different scenarios. Keep in mind when creating text output, the code in green is what will be displayed for the user.

```
# Basic text output using the print function
print("Hello World")

# Specific information with text explain what the information shows
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)

# Formatting specific results of a function
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Multiple items in a text display
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)
```

Properly Labeling and Displaying Outputs
When creating text output, it is important to make sure the user will understand what is being displayed. Properly labeling output is an important part of visual representation and can make code more user-friendly. Below are a few examples of labels to make the code easier to read and understand.

```
# The labels 'Name' and 'Favorite Food' make the display more clear
and help define what the terms relate to
name = "Tim"
food = "Pizza"
print("Name:", name, "Favorite Food:", food)
```



```
# In this example, putting the information that is calculated in a
sentence helps convey what function was performed
num1 = 7
num2 = 4
sum_result = num1 + num2
print(f"The sum of {num1} and {num2} is {sum_result}.")

# Adding labels that show what x, y, and z correlate makes the display
easier to read
x = "Yes"
y = "No"
z = "Maybe"
print("For this example x, y, and z represent:", x, y, z)

# Adding units when displaying specific information can also make text
displays more clear
num1 = 7
num2 = 4
print(f"Loaded Weight: {num1}lbs Unloaded Weight: {num2}lbs")

# When there are several types of complex information being displayed,
descriptive labels can help clear any confusion
name = "Benjamin"
number = 4
transactions = 13
print(f"Customer: {name}")
print(f"Customer Number for December: {number}")
print(f"Number of Transaction for November: {transactions}")
```

Interactive Input Interfaces
When creating interactive input interfaces with relevant user prompts to acquire data from a user, it is important the code is readable and easy to understand. This code requires information from the user to display a certain message or action. An example of a code that requires user input is shown below.

```
# The prompt below asks for inputs from the user to display a welcome
message
name = input("Enter your name:")

# Now the text will be displayed with the information that the user
added
print(f"Welcome {name}!")
```


Standard Formatting Styles
Formatting styles can vary depending on what the code is being used for, personal preference or the coding standards of a business or organization. A few common formatting styles are PEP 8 Style Guide, the Google Python Style Guide, YAPF, PyCharm and Docstring Conventions. The coding examples given in this handout follow the PEP 8 Style Guide, which is standard for Python coding.

Simple Vector Graphics Using Lines
The code to create a vector graphic using lines is shown below. This code uses matplotlib to create this shape.

```
import matplotlib.pyplot as plt

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the coordinates for each lines
x1, y1 = 1, 1
x2, y2 = 4, 4
x3, y3 = 2, 5

# Draw each lines
ax.plot([x1, x2], [y1, y2], label= 'Line 1',color='blue', linewidth=2)
ax.plot([x2, x3], [y2, y3], label='Line 2',color='red', linestyle='--
', linewidth=2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Lines')

# Add the legend
ax.legend()

# Show the plot
plt.show()
```

Simple Vector Graphics Using Circles
An example code to create a vector graphic using circles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
```



```
# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blue', facecolor='none',
linewidth=2, label='Circle 1')
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red', facecolor='none',
linewidth=2, label='Circle 2')

# Add the circles to the axis
ax.add_patch(circle1)
ax.add_patch(circle2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Circles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Simple Vector Graphics Using Rectangles
An example code to create a vector graphic using rectangles is shown below. This code uses matplotlib to create the shape.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axis
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolor='blue',
facecolor='none', linewidth=2, label='Rectangle 1')
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edgecolor='red',
facecolor='none', linewidth=2, label='Rectangle 2')
```

Update to Content Accepted by SRP


```
# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Vector Graphics with Rectangles')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better representation
ax.legend('equal', adjustable='box')

# Show the plot
plt.show()
```

Matplotlib is a widely-used library for creating static, animated and interactive visualizations. This example will showcase the inputs, outputs and data display for a line, circle or rectangle.

```
# dialog.py

"""Dialog-style application."""

import sys
from PyQt6.QtWidgets import (
    QApplication,
    QDialog,
    QDialogButtonBox,
    QFormLayout,
    QLineEdit,
    QVBoxLayout,
)

class Window(QDialog):
    def __init__(self):
        super().__init__(parent=None)
```

Copyright CEV Multimedia, LLC

Project - Data Visualization Program
Visual Presentation

1 of 1

You will create a program which displays a data set in a visually appealing way using appropriate formatting styles and graphics.

Directions:

1. On your computer, access the matplotlib library. The matplotlib library is not in the Standard Python Library. You can find an online compiler by entering "online matplotlib compiler" in a search engine.
2. Enter and run each of the following codes. The codes will create a line graph, circle graph and rectangle graph with the given x and y data. Add labels and a title to the chart. Change the data and customize each chart by adding meaningful axis labels and a meaningful title. Adjust the colors, markers and other properties of the chart elements. Feel free to increase the quantity of data points. Format your code for readability. Watching a video on matplotlib plots for adding these extra features may be helpful.
3. Line Graph

```
import matplotlib.pyplot as plt

# Sample Data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a line chart
plt.plot(x, y)

# Add Labels and title
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title of the chart')

# Display the chart
plt.show()
```

4. Circle Graph

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Define the figure and the axis
fig, ax = plt.subplots()

# Define the circle parameters
circle1 = plt.Circle((1, 1), 0.5, edgecolor='blu
circle2 = plt.Circle((3, 2), 0.8, edgecolor='red

# Add the circles to the axis
ax.add_patch(circle1)
```



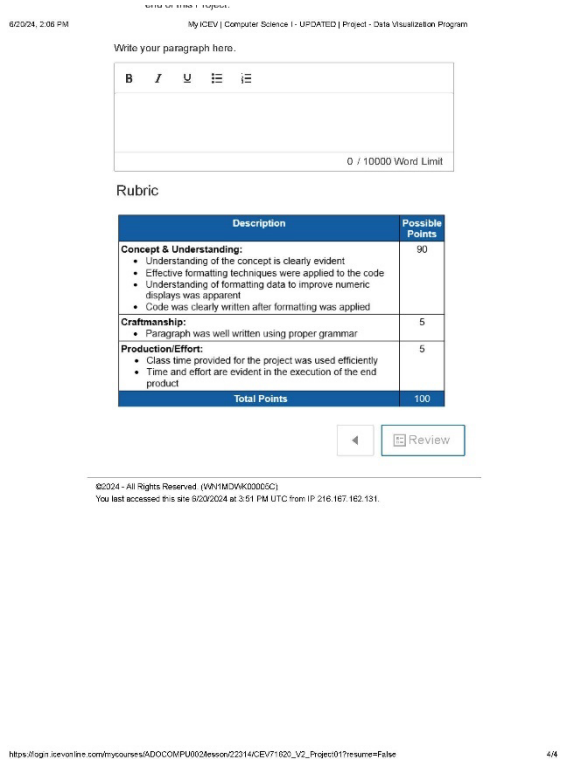
```
# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Title of chart')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better represen
ax.legend('equal', adjustable='box')

# Show the plot plt.show()
```

5. Rectangle Graph

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Decide what figure you will create and the axi
fig, ax = plt.subplots()

# Define the rectangle parameters
rect1 = patches.Rectangle((1, 1), 2, 3, edgecolo
rect2 = patches.Rectangle((4, 2), 1.5, 2.5, edge

# Add the rectangles to the axis
ax.add_patch(rect1)
ax.add_patch(rect2)

# Set the labels for each line and titles
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Title of chart')

# Add the legend
ax.legend()

# Set the aspect ratio to ensure better represen
ax.legend('equal', adjustable='box')

# Show the plot plt.show()
```

6. Analyze the resulting graphs and answer the following questions in a paragraph:
   - How do the changes you made impact the message the graph imparts
   - Does the formatting of the code and the numeric formatting impact the output
   - How did formatting the data improve the numeric display

Update to Content Accepted by SRP



(SE)(Breakout(s)) and (Citation Type(s))
(4)(A)(i), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Programming Problem-Solving Processes-Overview of Solving Processes (00:15-3:12), https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21654

Description of the specific location and hyperlink to the exact location of the proposed new content
Programming Problem-Solving Processes Student Handout-Creating Program Solutions, https://files.icevonline.com/html/CEV81114_V2_HTML/CEV81114_V2_HTML_Student_Handout_-_Creating_Program_Solutions.htm
This Student Handout is found in the Programming Problem-Solving Processes lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Programming Problem-Solving Processes

The content within this transcript has been created utilizing a third-party software company which complies with all federal accessibility laws and international standards for web accessibility, providing a measured accuracy rate of 99.6 percent.

1. Overview of the Solving Processes

TEXT ON SCREEN
Programming Problem-Solving Processes
Overview of Solving Processes

Computer programming is the process used to write code that instructs how a computer, application or software program performs. Computer programming's core function is to solve or create solutions to problems or needs. The process of making a program or a set of instructions for a computer to execute includes the following steps-- understand the problem, design a solution, translate the solution into a program, and test and debug the program.

The first step of the programming problem-solving process is understanding the problem. This can be done through research, questioning, and identifying key components of the problem. Most of the time, this information will come from the customer or the key identifier of the problem.

Once the data has been collected and the problem is understood, the next step is to design a solution. Charting and working through ideas will create a solution. We'll use the example of making a peanut butter and jelly sandwich to better visualize this process.

Look at the diagram. It shows the potential choices needed to make a peanut butter and jelly sandwich. The first decision answers the question, what kind of peanut butter? Chunky or creamy? The next decision involves the type of jelly to use. This process continues building on this concept until all possibilities in making
IMAGE ON SCREEN- A diagram showing the choices to make a peanut butter and jelly sandwich. At the top it starts with the question, what kind of peanut butter would you like. Then it has two arrows going down, on the left it says creamy and then on the right it says chunky. Then there are two more arrows going down to the question what kind of jelly would you like.

the sandwich are explored-- bread type,
IMAGE ON SCREEN- A diagram showing the choices of bread type. At the top it starts with the question, what type of bread is used. Then it has two arrows going down, on the left it says white and on the right it says wheat.

how to cut the sandwich,
IMAGE ON SCREEN- A diagram showing the choices of cutting a sandwich. At the top it starts with the question, how should the sandwich be cut. Then it has four arrows going down, the one on the left says keep as is, the next one over says halved, the next one says squares and the one on the far left says triangle.

crusts on or off.
IMAGE ON SCREEN- A diagram showing the choices of crusts being on or off. At the top it starts with the question, should the crusts be on or off. Then it has two arrows going down, the one on the left says one and the one on the right says off.

The process of making a sandwich is simple, but the same process is used to design a new phone app for a customer or a computer program for a major corporation.

The next step in the programming problem-solving process is translating the gathered information and solution into a program. Programs are built on a step-by-step process called an algorithm.

Algorithms are used in mathematics when following the order of operations to solve equations.
IMAGE ON SCREEN- PEMDAS is titled on screen to show how mathematics is used to follow the order of operations. P- parentheses, E- exponents, M- mutiple, D- dived, A- add, and S- subtract. The following equation explains PEMDAS: 4 x 5 (6+4) = 200

Let us explore some of the steps for making a program using the peanut butter and jelly sandwich Example we know from the first program what people want on their sandwich before it is made.

We want to use the same questions and answers from our explored solutions.
IMAGE ON SCREEN- IMAGE ON SCREEN- A diagram showing the choices to make a peanut butter and jelly sandwich. At the top it starts with the question, what kind of peanut butter would you like. Then it has two arrows going down, on the left it says creamy and then on the right it says chunky. The next question is what kind of jelly would you like. The question about jelly has three arrows going down from it. The arrow on the left says grape, the arrow in the middle says apricot and the arrow on the right says strawberry. Then the next question is asked, what type of bread is used. Then it has two arrows going down, on the left it says white and on the right it says wheat. Then the next question is asked, how should the sandwich be cut. Then it has four arrows going down, the one on the left says keep as is, the next one over says halved, the next one says

squares and the one on the far left says triangle. Then the next question is asked, should the crusts be on or off. Then it has two arrows going down, the one on the left says one and the one on the right says off.

In this case, the first question that would be seen by the user is, what kind of peanut butter would you like? Chunky or creamy?
IMAGE ON SCREEN- A graphic of a computer screen is detailed with a diagram showing the choices to make a peanut butter and jelly sandwich. At the top it starts with the question, what kind of peanut butter would you like. Then it has two arrows going down, on the left it says creamy and then on the right it says chunky.

The program prompts the user to provide an input to the question before moving on to the next step.
IMAGE ON SCREEN- A graphic of a computer screen is detailed with a diagram showing the choices to make a peanut butter and jelly sandwich. At the top it starts with the question, what kind of jelly would you like. Then it has three arrows going down, on the left it says grape, in the middle it says apricot and then on the right it says strawberry.

It goes through these steps until the end. Then the program gives an output of every selection and requests a confirmation of the options that the customer chose before finalizing the order.
IMAGE ON SCREEN- A graphic of a computer screen is detailed with various questions sliding on and off the screen. The title of the computer is order summary. Question one- what kind of peanut butter would you like? Creamy. Question two- what kind of jelly would you like? Grape. Question three- What type of bread is used? White. Question four- How should the sandwich be cut? Keep as is. Question five- should the crusts be on or off? On. The last slide on the computer screen states thank you for your order.

Finally, after the program is created, it is tested and refined.
TEXT ON SCREEN
Test and debug he program

This process resets and starts again at step one to continuously improve and refine the original program version.
TEXT ON SCREEN- Steps are listed to make a program or a set of instructions for a computer to execute. Step 1: understand the problem, step 2: design a solution, step 3: translate a solution into a program, step 4: test and debug the program.

2. Tasks & Subtasks for Solving Problems

TEXT ON SCREEN
Programming Problem-Solving Processes
Tasks and Subtasks for Solving Problems

OK. We understand the steps for solving programming design problems. Now, let us break them into tasks and subtasks. Tasks are the steps needed to solve the problem or deliverable. Subtasks are smaller tasks associated with a larger and more complex task, which can be broken into three types-- sequential, conditional, and iterative.

Sequential subtasks are items that execute in order. Sequential subtasks are used in programs that have a linear flow. These subtasks are used in programs that do not offer choices to repeat a process.
IMAGE ON SCREEN- A graphic with a man standing and looking at a phone. On the phone is a timer with the words start and stop. The timer is enlarged on the left side of the screen as well. The timer is started and goes until 60 seconds.

Asking the user for two numbers followed by a program, adding the numbers together, is an example of a sequential subtask.
IMAGE ON SCREEN- **IMAGE ON SCREEN- A graphic with a man standing and looking at a phone. On the phone is a dial consisting of numbers. On the far left it starts with 1, 2, 3, going down on the left it starts with 4, 5, 6, going down on the far left it starts with 7, 8, 9, and going down at the very bottom starting on the left is an X, 0, check mark. The dial is also enlarged on the left side of the screen. The man taps 3 and 6. Then 3 + 6 = 9 pops up enlarged on the left side in replace of the dial.**

Conditional subtasks offer choices in the form of true/false statements.

Conditional subtasks are used when programs need to make decisions about what to do based on user input or the results of other actions in the program.
IMAGE ON SCREEN- A graphic with a man standing and looking at a phone. On the phone is the choices A and B. The choices A and B are also enlarged on the left side of the screen as Choice "A" and Choice "B". The man standing at the phone chooses A or Choice "A". The Choice "A" on the left side of the screen funnels down to Choice "A" and Choice "B". Choice "A" and Choice "B" are connected. The man selects Choice "B" then from Choice "B" it funnels down to Choce "A" and Choice "B". They are connected. Choice "A" is selected.

Asking a user if they would like to play a game again is a conditional subtask because the game will either start or exit depending on the answer.
IMAGE ON SCREEN- A graphic with a man standing and looking at a phone. On the phone is the choices yes or no. The choices yes or no are also enlarged on

Update to Content Accepted by SRP

the left side of the screen. The choice yes is selected by the man.

Finally, iterative subtasks create a loop that continues until certain conditions are met. This specific subtasks can create infinite loops if not used correctly.
IMAGE ON SCREEN- A graphic with a man standing and looking at a phone. On the phone is the choices stop and start with a coin. The coin is also enlarged on the left side of the screen. The enlarged coin flips continuously as the man presses start on the phone.

Iterative subtasks repeat until a condition is met. A timer that counts down to 0 and ends once it hits 0 is an iterative subtask.
IMAGE ON SCREEN- A graphic with a man standing and looking at a phone. On the phone is the choices start and stop with a timer of 0:03.00. The timer is also enlarged on the left side of the screen. The man presses start, and the timer goes from 0:02.60 to 0:00.00

3. Data Types & Objects Needed

TEXT ON SCREEN
Programming Problem-Solving Processes
Date Types & Objects Needed

There will be different data types associated with whatever specific computer programming language is used. In Python, the data types are numeric, int, integer, that holds signed integers of non-limited length. Long hold long integers. Float holds floating precision numbers and is accurate up to 15 decimal places. Complex holds complex numbers.

String is a sequence of characters which can be letters, numbers, or special. And lists are used to store a set of information. For example, 0, 1 2, or purple, blue, red. The problem must be analyzed, and the solution must be explored to understand the data types and objects needed to create the program.

In most cases, it will need multiple data types in the same program to make it work. Remember the sandwich program discussed earlier. We would use the string data type to store the user's answers to the questions crunchy or creamy. We would use the int data type to store how many sandwiches they would like to order.

A list can be used to print out all the options that the user has entered for the entire program before they confirm that they are done with their order.

4. Applying the Process

TEXT ON SCREEN
Programming Problem-Solving Processes
Applying the Process

Feathered Paws, a new pet store, hired computer programmers to help their new application be more secure when users enter data. Currently, users need to input a four-digit code to gain access to their mobile application. Let us look at creating a program that can help users protect their data.

Step one, understanding the problem. Through discussions with the client, when users select their four-digit code, it is transferred to Feathered Paws's database in plain text. This means whatever the user enters is sent exactly as selected.

There are ways that hackers can collect information being sent over the internet. Feathered Paws needs additional security for their data transfer to protect the information. If a hacker steals the information, they can access the user's account, and this causes possible legal actions against Feathered Paws. That is the problem.

Now move on to step number two, design a solution that researches and creates ideas to solve Feathered Paws's problem. After some research into basic cybersecurity, you've discovered some previously used techniques to protect information. As the team lead on the project and after discussing the options with your colleagues, you decide to reverse the order of the four-digit code to help protect the data.

Step three, translate the solution into a program using the iterative subtasks to create the code.

In lines number one and number two, this is the basic language that is used when starting programs. Most programming software will atomically have lines number one and number two written for you.
IMAGE ON SCREEN- Coding is displayed.
Line 1 class Main { with class in blue
Line 2 public static void main(String [ ] args) { with public static void and String in blue

In line number four, we are creating an int named num that is storing a four-digit number, 1234, and reversed that is storing 0.
IMAGE ON SCREEN- Coding is displayed.
Line 4 int num = 1234, reversed = 0; with int in blue, = signs in dark blue, and 1234 and 0 in green

In line number six, system.out.println original number plus num, it creates an output to the screen that the user can see. Original number, 1234.
IMAGE ON SCREEN- Coding is displayed on screen.
Line 6 System.out.println("Original Number : " + num); with println in brown, "Original Number : " in orange and + in dark blue

Line number eight is a note that only programmers see and helps identify what the lines of code are being used for.
IMAGE ON SCREEN- Coding is displayed on screen.
Line 8 // run loop until num becomes 0 with everything in gray

Line number nine, while integer num is not equal to 0, the program will repeat.
IMAGE ON SCREEN- Coding is displayed on screen.
Line 9 while (num != 0) { with while, ! and = in blue, and 0 in green

Line number 11 is another note created by the programmer to identify the next section of code will get the last number from the int named num.
IMAGE ON SCREEN- Coding is displayed on screen.
Line 11 // get last digit from num with everything in gray

Line number 12 is creating an int named digit, which is equal to num modulo 10. Modulo returns the remainder from division.
IMAGE ON SCREEN- Coding is displayed on screen.
Line 12 int digit = num % 10; with int in blue, = and in dark blue, and 10 in green

Line number 13 is now making reversed equal reversed multiply 10 plus digit.
IMAGE ON SCREEN- Coding is displayed on screen.
Line 13 reversed = reversed * 10 + digit; with = and * in dark blue, 10 in green, and + in blue

Line number 15 is the final programming note that line number 16 will remove the last digit from number.
IMAGE ON SCREEN- Coding is displayed on screen.
Line 15 // remove the last digit from num with everything gray

Line number 16, num divide and assignment operator 10.
IMAGE ON SCREEN- Coding is displayed on screen.
Line 16 num /= 10; with / and = in dark blue and 10 in green
Line 17 }

Finally, line number 19, system.out.println reversed number plus reversed, will send a message to the screen for the user to read, stating, reversed number– 4321.
IMAGE ON SCREEN- Coding is displayed on screen.

Line 19 System.out.println("Reversed Number: " + reversed); with println in brown, "Reversed Number : " in red, and + in blue
Line 20 icon of a light bulb }
Line 21 }

When programming, always look for ways to reuse previously created lines of code or create processes that reduce the lines of code needed to make the program function. This saves time for the programmer, uses less space on the device's storage, and allows any other programmers that are working on the project have an easier time reading the code.

As seen in the example, system.out.println was used in two different sections of the program. The line was copied for reuse and updated to produce the new required output.
IMAGE ON SCREEN- Coding is displayed.
Line 1 class Main { with class in blue
Line 2 public static void main(String [ ] args) { with public static void and String in blue
Line 4 int num = 1234, reversed = 0; with int in blue, = signs in dark blue, and 1234 and 0 in green
Line 6 System.out.println("Original Number : " + num); with println in brown, "Original Number : " in orange and + in dark blue
Line 8 // run loop until num becomes 0 with everything in gray
Line 9 while (num != 0) { with while, ! and = in blue, and 0 in green
Line 11 // get last digit from num with everything in gray
Line 12 int digit = num % 10; with int in blue, = and in dark blue, and 10 in green
Line 13 reversed = reversed * 10 + digit; with = and * in dark blue, 10 in green, and + in blue
Line 15 // remove the last digit from num with everything gray
Line 16 num /= 10; with / and = in dark blue and 10 in green
Line 17 }
Line 19 System.out.println("Reversed Number: " + reversed); with println in brown, "Reversed Number : " in red, and + in blue
Line 20 icon of a light bulb }
Line 21 }

Line 6 and 19 are highlighted to show how the line was copied, reused and updated to produce the new required output.

Step four, test and debug the solution...
TEXT ON SCREEN
...Program

Update to Content Accepted by SRP

When testing this program, a four-digit number can be entered, and the output is the number reversed. If it does not have the requested outcome of reversing the order of the number, refine your program and retest.

Copyright CEV Multimedia, LLC

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Creating Program Solutions

Task and subtasks (sequential, conditional and iterative) are demonstrated below. Using the steps of program design problem-solving strategies, assess the following code for any errors.

```
# Main program
sequential_subtask() #Execute the sequential subtask

while conditional_subtask(): # Keep repeating as long as user wants to
play
     sequential_subtask() #Execute the sequential subtask
     again

iterative_subtask() #Execute iterative subtask

def sequential_subtask():
     # This is a sequential subtask. It asks the users for two
     numbers
     num1 = float(input("Enter the first number: "))
      num1 = float(input("Enter the second number: "))

     # Then this adds them together
     result = num1 + num2
      print(f"The sum of {num1} and {num2} is: {result}")

def conditional_subtask():
     # This is a conditional subtask. It asks the users if
     they would like to play the game again
     play_again = input("Do you want to play again? (yes/no):
     ").lower()

     # Checking the users choice
     if play_again == "yes":
          print ("Let's play again!")
          return True
     elif play_again == "no":
          print (Goodbye!)
          return False
     else:
          print ("Invalid choice. Please enter 'yes' or 'no'.)
          return condition_subtask()
```



```
          # Recursive call to handle an invalid response

def iterative_subtask():
     # Iterative subtask: Creating a loop that will continue
     until a certain condition is met. This loop is a timer
     timer = 5 # This is the initial timer value

     while timer > 0:
          print(f"Timer: {timer}")
          timer -= 1

     print("Iterative subtask is complete. Timer has reached
     0.")
```

One example of a possible error for the code above is shown below.

```
     play_again = input"Do you want to play again? (yes/no):
     ").lower()
```

To find the error, use the steps of program design problem-solving strategies. The steps are listed below, as well as the solution to debug the code.

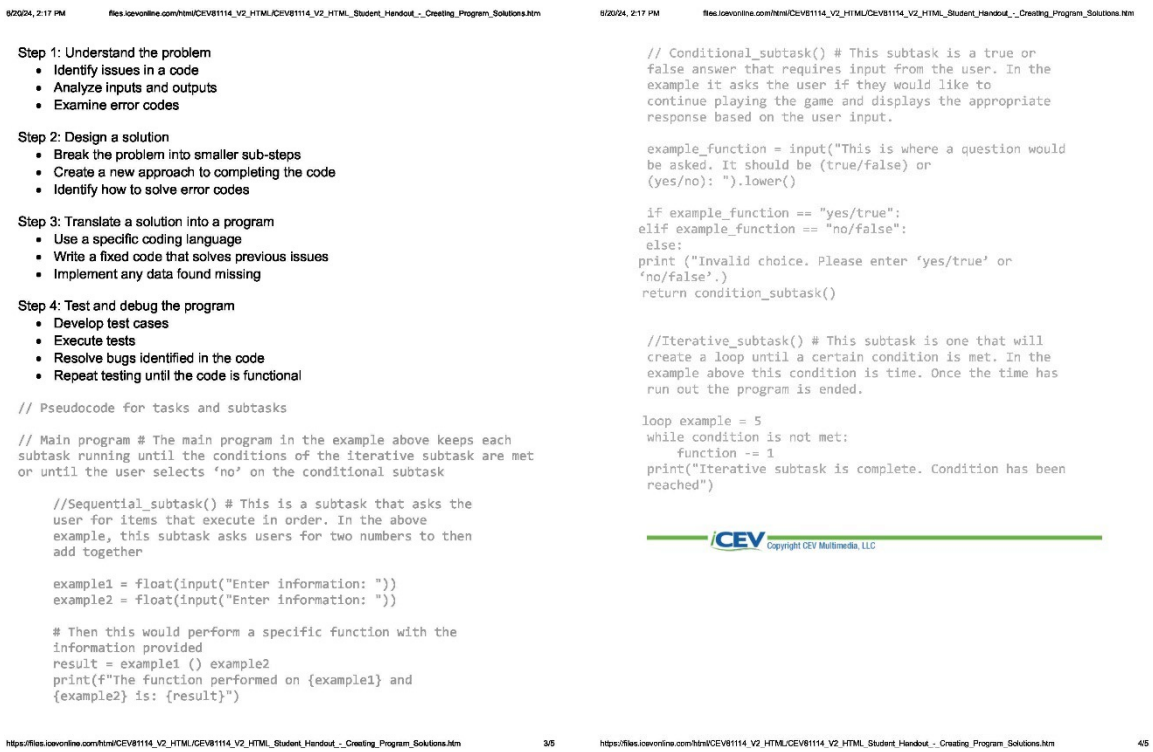
Step 1: Understand the problem
Step 2: Design a solution
Step 3: Translate a solution into a program
Step 4: Test and debug the program

Step 1: By examining the code, it is apparent there is a syntax error. There is an open parenthesis missing before "Do you want to play again?. This is the problem with the code.
Step 2: Adding the parenthesis will ensure proper spacing and code formatting. The code should be functional.
Step 3: To fix this error, an additional parenthesis should be placed in the correct position.
Step 4: After adding the parenthesis, the code appears to be fixed as shown below. Running the program shows there are no errors, so the code is now debugged.

```
     play_again = input("Do you want to play again? (yes/no):
     ").lower()
```

Using the example above, make pseudocode for problem-solving processes within a program.

Update to Content Accepted by SRP



6/20/24, 2:17 PM files.icevonline.com/html/CEV81114_V2_HTML/CEV81114_V2_HTML_Student_Handout_-_Creating_Program_Solutions.htm

Step 1: Understand the problem
- Identify issues in a code
- Analyze inputs and outputs
- Examine error codes

Step 2: Design a solution
- Break the problem into smaller sub-steps
- Create a new approach to completing the code
- Identify how to solve error codes

Step 3: Translate a solution into a program
- Use a specific coding language
- Write a fixed code that solves previous issues
- Implement any data found missing

Step 4: Test and debug the program
- Develop test cases
- Execute tests
- Resolve bugs identified in the code
- Repeat testing until the code is functional

```
// Pseudocode for tasks and subtasks

// Main program # The main program in the example above keeps each
subtask running until the conditions of the iterative subtask are met
or until the user selects 'no' on the conditional subtask

    //Sequential_subtask() # This is a subtask that asks the
    user for items that execute in order. In the above
    example, this subtask asks users for two numbers to then
    add together

    example1 = float(input("Enter information: "))
    example2 = float(input("Enter information: "))

    # Then this would perform a specific function with the
    information provided
    result = example1 () example2
    print(f"The function performed on {example1} and
    {example2} is: {result}")
```

https://files.icevonline.com/html/CEV81114_V2_HTML/CEV81114_V2_HTML_Student_Handout_-_Creating_Program_Solutions.htm 3/5

6/20/24, 2:17 PM files.icevonline.com/html/CEV81114_V2_HTML/CEV81114_V2_HTML_Student_Handout_-_Creating_Program_Solutions.htm

```
// Conditional_subtask() # This subtask is a true or
false answer that requires input from the user. In the
example it asks the user if they would like to
continue playing the game and displays the appropriate
response based on the user input.

example_function = input("This is where a question would
be asked. It should be (true/false) or
(yes/no): ").lower()

if example_function == "yes/true":
elif example_function == "no/false":
else:
print ("Invalid choice. Please enter 'yes/true' or
'no/false'.)
return condition_subtask()


//Iterative_subtask() # This subtask is one that will
create a loop until a certain condition is met. In the
example above this condition is time. Once the time has
run out the program is ended.

loop example = 5
while condition is not met:
    function -= 1
print("Iterative subtask is complete. Condition has been
reached")
```

iCEV Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV81114_V2_HTML/CEV81114_V2_HTML_Student_Handout_-_Creating_Program_Solutions.htm 4/5

(SE)(Breakout(s)) and (Citation Type(s))
(4)(I)(i), Narrative and Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Error Types and Debugging Strategies (Slide 18),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644

Activity-Debugging, Project-Error Types and Debugging,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644/CEV71511_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Error Types and Debugging Strategies Student Handout-Testing Program Solutions,
https://login.icevonline.com/download/c46dac71-1b97-4a29-b4b4-8f45c0d2c910
This Student Handout is found in the Error Types & Debugging Strategies lesson beneath the Instructional Materials heading.

Project-Error Types and Debugging,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22298/CEV71511_V2_Project01
This Project is found in the Error Types & Debugging Strategies lesson beneath the Interactive Assignments heading. After clicking the link to the Project, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Project.

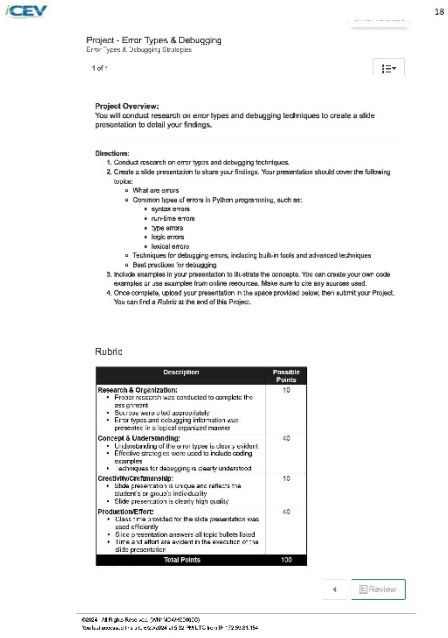
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Valid & Invalid Test Data

- Include ways to test the program solutions and analyze the resulting behavior
  - write test cases which include valid and invalid input data
  - run the program with the test data
  - observe the behavior of the program and compare it to the expected behavior
    - if the program does not produce the expected behavior, debug the program



CEV

18

Project - Error Types & Debugging

Error Types & Debugging Strategies

1 of 1

Project Overview:
You will conduct research on error types and debugging techniques to create a slide presentation to detail your findings.

Directions:

1. Conduct research on error types and debugging techniques.
2. Create a slide presentation to share your findings. Your presentation should cover the following topics:
   - What are errors
   - Common types of errors in Python programming, such as:
     - syntax errors
     - run-time errors
     - type errors
     - logic errors
     - lexical errors
   - Techniques for debugging errors, including built-in tools and advanced techniques
   - Best practices for debugging
3. Include examples in your presentation to illustrate the concepts. You can create your own code examples or use examples from online resources. Make sure to cite any sources used.
4. Once complete, upload your presentation in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted to complete the assignment • Sources were cited appropriately • Error types and debugging information was presented in a logical organized manner	10
Concept & Understanding: • Understanding of the error types is clearly evident • Effective strategies were used to include coding examples • techniques for debugging is clearly understood	40
Creativity/Ownership: • Slide presentation is unique and reflects the student's or group's individuality • Slide presentation is clearly high quality	10
Production/Effort: • Class time provided for the slide presentation was used efficiently • Slide presentation answers all topic bullets listed • Time and effort are evident in the execution of the slide presentation	40
Total Points	**100**

Review

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Testing Program Solutions

Testing Program Solutions with Valid and Invalid Test Data
Test data is what a user inputs that allows a function to be performed to test a system of programs. Valid test data is when the information added is in a recognizable range and format. When test data is invalid, it is not recognized by the program and will produce errors. The code below is an example of a program with invalid test data.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: 8 "))
    num2 = float(input("Enter the second number: none "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When examining the code, the num2 input is not a number, but a word inserted instead. In this scenario, the code would not recognize a term other than a number and would not produce an answer. This would result in a ValueError code if tested. To make this test data valid, the word 'none' would need to be replaced with a number. Analyzing the resulting behavior and comparing it to the expected behavior is an important step in debugging any errors.

Solving Problems
The following task is an example of a program that is correctly coded and will display a correct solution.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

The following task is an example of the same code as above, but with an error that will cause the program to display an error message.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "

    print(f"The sum of {num1} and {num2} is: {result}")
```

Another example of a code with errors is shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of num1 and num2 is: result")
```

In this code, there are no brackets around the 'num1' and num2' or the 'result' displays in the print function. This would cause a runtime error, possibly causing the program to crash. To resolve this issue, look at language documentation such as Python documentation. This programming language code documentation can help explain what is required for a code to be functional. The corrected code is shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

CEV Copyright CEV Multimedia, LLC

```
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When this code is inputted in Python, the following error message will occur.

```
File "example.py", line 1
    num1 = float(input("Enter the first number: "
                                                 ^
SyntaxError: unexpected EOF while parsing
```

After reading the error message, it is noted there is a syntax error near the end of line four. To solve this problem, the code must be fixed and the correct syntax should be used. Once the error has been resolved the code will match the first example given.

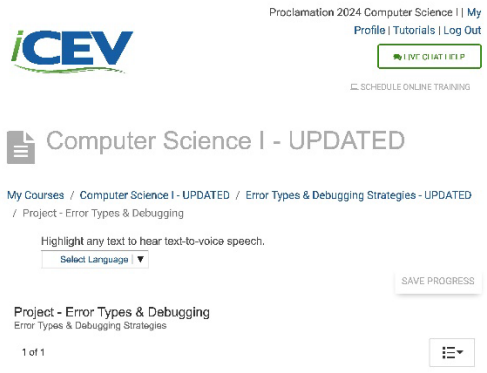
Another example of a potential problem and solution is shown in the example below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(("Enter the first number: "))
    num2 = float(("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

In this example, the code was copied over from a reference material, but the programmer missed putting in the input function after the float function. Without the input function, the numbers added by the user would not become a string, and then into a float. This would display an error. One way to fix this issue would be to use a search engine to find the error or double-check the source of the code for missing information. This issue could also be resolved by analyzing the reference material and double checking for errors. The corrected code will appear as shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
```

Update to Content Accepted by SRP



Project Overview:
You will conduct research on error types and debugging techniques to create a slide presentation to detail your findings.

Directions:

1. Conduct research on error types and debugging techniques.
2. Create a slide presentation to share your findings. Your presentation should cover the following topics:
   - What are errors
   - How to test program solutions with valid and invalid test data
     - Include coding examples for both data types
   - Common types of errors with invalid test data in Python programming, such as:
     - syntax errors
     - run-time errors
     - type errors
     - logic errors
     - lexical errors
   - Resulting behavior after using valid and invalid data in Python programming
   - Techniques for debugging errors while working with invalid test data, including built-in tools and advanced techniques
   - Best practices for debugging
3. Include examples in your presentation to illustrate the concepts. You can create your own code examples or use examples from online resources. Make sure to cite any sources used.
   - Use your IDE to write a program that has at least three errors, take a screenshot of those errors and insert into your presentation
   - Make the necessary changes so the program runs correctly, take a screenshot and insert into your presentation
4. Once complete, upload your presentation in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted to complete the assignment • Sources were cited appropriately • Error types and debugging information was presented in a logical organized manner	10
Concept & Understanding: • Understanding of the error types is clearly evident • Effective strategies were used to include coding examples • Techniques for debugging is clearly understood	40
Creativity/Craftmanship: • Slide presentation is unique and reflects the student's or group's individuality • Slide presentation is clearly high quality	10
Production/Effort: • Class time provided for the slide presentation was used efficiently • Slide presentation answers all topic bullets listed • Time and effort are evident in the execution of the slide presentation	40
Total Points	100

Update to Content Accepted by SRP

(SE)(Breakout(s)) and (Citation Type(s))
(4)(I)(ii), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Error Types and Debugging Strategies (Slide 18),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644

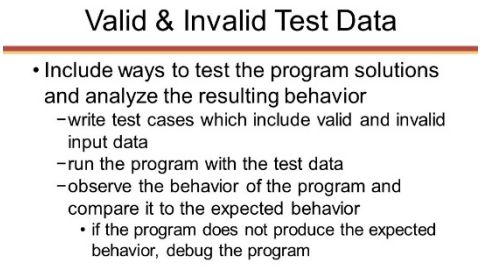
Activity-Debugging, Project-Error Types and Debugging,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644/CEV71511_Activity01

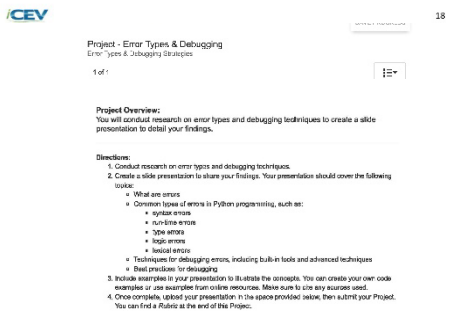
Description of the specific location and hyperlink to the exact location of the proposed new content
Error Types and Debugging Strategies Student Handout-Testing Program Solutions,
https://login.icevonline.com/download/c46dac71-1b97-4a29-b4b4-8f45c0d2c910
This Student Handout is found in the Error Types & Debugging Strategies lesson beneath the Instructional Materials heading.

Project-Error Types and Debugging,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22298/CEV71511_V2_Project01
This Project is found in the Error Types & Debugging Strategies lesson beneath the Interactive Assignments heading. After clicking the link to the Project, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Project.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.





Update to Content Accepted by SRP

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted to complete the assignment • Sources were cited appropriately • Error types and debugging information was presented in a logical organized manner	10
Concept & Understanding: • Understanding of the error types is clear y evident • Effective strategies were used to include coding examples • techniques for debugging is clearly understood	40
Creativity/Craftsmanship: • Slide presentation is unique and reflects the student's or group's individuality • Slide presentation is clearly high quality	10
Production/Effort: • Class time provided for the slide presentation was used efficiently • Slide presentation answers all topic bullets listed • Time & effort are evident in the execution of the slide presentation	40
Total Points	100

Review

©2024 - All Rights Reserved - WGU

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Testing Program Solutions

Testing Program Solutions with Valid and Invalid Test Data
Test data is what a user inputs that allows a function to be performed to test a system of programs. Valid test data is when the information added is in a recognizable range and format. When test data is invalid, it is not recognized by the program and will produce errors. The code below is an example of a program with invalid test data.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: 8 "))
    num2 = float(input("Enter the second number: none "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When examining the code, the num2 input is not a number, but a word inserted instead. In this scenario, the code would not recognize a term other than a number and would not produce an answer. This would result in a ValueError code if tested. To make this test data valid, the word 'none' would need to be replaced with a number. Analyzing the resulting behavior and comparing it to the expected behavior is an important step in debugging any errors.

Solving Problems
The following task is an example of a program that is correctly coded and will display a correct solution.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

The following task is an example of the same code as above, but with an error that will cause the program to display an error message.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When this code is inputted in Python, the following error message will occur.

```
File "example.py", line 1
    num1 = float(input("Enter the first number: "
                                               ^
SyntaxError: unexpected EOF while parsing
```

After reading the error message, it is noted there is a syntax error near the end of line four. To solve this problem, the code must be fixed and the correct syntax should be used. Once the error has been resolved the code will match the first example given.

Another example of a potential problem and solution is shown in the example below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(("Enter the first number: "))
    num2 = float(("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

In this example, the code was copied over from a reference material, but the programmer missed putting in the input function after the float function. Without the input function, the numbers added by the user would not become a string, and then into a float. This would display an error. One way to fix this issue would be to use a search engine to find the error or double-check the source of the code for missing information. This issue could also be resolved by analyzing the reference material and double checking for errors. The corrected code will appear as shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
```

Update to Content Accepted by SRP

```
    print(f"The sum of {num1} and {num2} is: {result}")
```

Another example of a code with errors is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of num1 and num2 is: result")
```

In this code, there are no brackets around the 'num1' and num2' or the 'result' displays in the print function. This would cause a runtime error, possibly causing the program to crash. To resolve this issue, look at language documentation such as Python documentation. This programming language code documentation can help explain what is required for a code to be functional. The corrected code is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

CEV Copyright CEV Multimedia, LLC

Project - Error Types & Debugging

Error Types & Debugging Strategies

1 of 1

Project Overview:
You will conduct research on error types and debugging techniques to create a slide presentation to detail your findings.

Directions:

1. Conduct research on error types and debugging techniques.
2. Create a slide presentation to share your findings. Your presentation should cover the following topics:
    - What are errors
    - How to test program solutions with valid and invalid test data
        - Include coding examples for both data types
    - Common types of errors with invalid test data in Python programming, such as:
        - syntax errors
        - run-time errors
        - type errors
        - logic errors
        - lexical errors
    - Resulting behavior after using valid and invalid data in Python programming
    - Techniques for debugging errors while working with invalid test data, including built-in tools and advanced techniques
    - Best practices for debugging
3. Include examples in your presentation to illustrate the concepts. You can create your own code examples or use examples from online resources. Make sure to cite any sources used.
    - Use your IDE to write a program that has at least three errors, take a screenshot of those errors and insert into your presentation
    - Make the necessary changes so the program runs correctly, take a screenshot and insert into your presentation
4. Once complete, upload your presentation in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Update to Content Accepted by SRP

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted to complete the assignment • Sources were cited appropriately • Error types and debugging information was presented in a logical organized manner	10
Concept & Understanding: • Understanding of the error types is clearly evident • Effective strategies were used to include coding examples • Techniques for debugging is clearly understood	40
Creativity/Craftmanship: • Slide presentation is unique and reflects the student's or group's individuality • Slide presentation is clearly high quality	10
Production/Effort: • Class time provided for the slide presentation was used efficiently • Slide presentation answers all topic bullets listed • Time and effort are evident in the execution of the slide presentation	40
Total Points	100

(SE)(Breakout(s)) and (Citation Type(s))
(4)(I)(iii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Error Types and Debugging Strategies (Slide 18),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644

Activity-Debugging, Project-Error Types and Debugging,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644/CEV71511_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Error Types and Debugging Strategies Student Handout-Testing Program Solutions,
https://login.icevonline.com/download/c46dac71-1b97-4a29-b4b4-8f45c0d2c910
This Student Handout is found in the Error Types & Debugging Strategies lesson beneath the Instructional Materials heading.

Project-Error Types and Debugging,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22298/CEV71511_V2_Project01
This Project is found in the Error Types & Debugging Strategies lesson beneath the Interactive Assignments heading. After clicking the link to the Project, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Project.

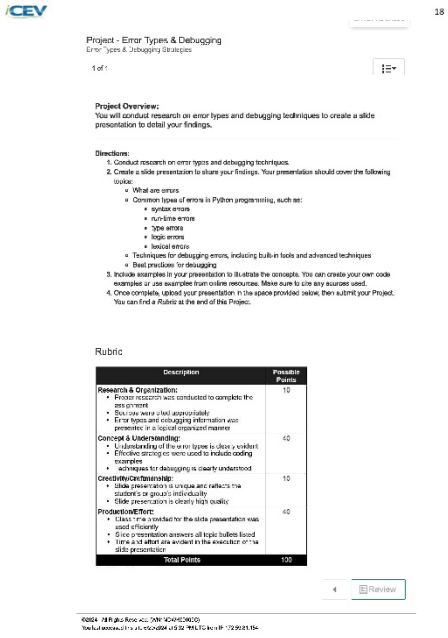
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Valid & Invalid Test Data

- Include ways to test the program solutions and analyze the resulting behavior
  - write test cases which include valid and invalid input data
  - run the program with the test data
  - observe the behavior of the program and compare it to the expected behavior
    - if the program does not produce the expected behavior, debug the program



CEV

18

Project - Error Types & Debugging

Error Types & Debugging Strategies

1 of 1

Project Overview:
You will conduct research on error types and debugging techniques to create a slide presentation to detail your findings.

Directions:

1. Conduct research on error types and debugging techniques.
2. Create a slide presentation to share your findings. Your presentation should cover the following topics:
   - What are errors
   - Common types of errors in Python programming, such as:
     - syntax errors
     - run-time errors
     - type errors
     - logic errors
     - lexical errors
   - Techniques for debugging errors, including built-in tools and advanced techniques
   - Best practices for debugging
3. Include examples in your presentation to illustrate the concepts. You can create your own code examples or use examples from online resources. Make sure to cite any sources used.
4. Once complete, upload your presentation in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted to complete the assignment • Sources were cited appropriately • Error types and debugging information was presented in a logical organized manner	10
Concept & Understanding: • Understanding of the error types is clearly evident • Effective strategies were used to include coding examples • techniques for debugging is clearly understood	40
Creativity/Ownership: • Slide presentation is unique and reflects the student's or group's individuality • Slide presentation is clearly high quality	10
Production/Effort: • Class time provided for the slide presentation was used efficiently • Slide presentation answers all topic bullets listed • time and effort are evident in the execution of the slide presentation	40
Total Points	**100**

Review

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Testing Program Solutions

Testing Program Solutions with Valid and Invalid Test Data
Test data is what a user inputs that allows a function to be performed to test a system of programs. Valid test data is when the information added is in a recognizable range and format. When test data is invalid, it is not recognized by the program and will produce errors. The code below is an example of a program with invalid test data.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: 8 "))
    num2 = float(input("Enter the second number: none "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When examining the code, the num2 input is not a number, but a word inserted instead. In this scenario, the code would not recognize a term other than a number and would not produce an answer. This would result in a ValueError code if tested. To make this test data valid, the word 'none' would need to be replaced with a number. Analyzing the resulting behavior and comparing it to the expected behavior is an important step in debugging any errors.

Solving Problems
The following task is an example of a program that is correctly coded and will display a correct solution.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

The following task is an example of the same code as above, but with an error that will cause the program to display an error message.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "

    print(f"The sum of {num1} and {num2} is: {result}")
```

Another example of a code with errors is shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of num1 and num2 is: result")
```

In this code, there are no brackets around the 'num1' and num2' or the 'result' displays in the print function. This would cause a runtime error, possibly causing the program to crash. To resolve this issue, look at language documentation such as Python documentation. This programming language code documentation can help explain what is required for a code to be functional. The corrected code is shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

CEV Copyright CEV Multimedia, LLC

```
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When this code is inputted in Python, the following error message will occur.

```
File "example.py", line 1
    num1 = float(input("Enter the first number: "
                                                 ^
SyntaxError: unexpected EOF while parsing
```

After reading the error message, it is noted there is a syntax error near the end of line four. To solve this problem, the code must be fixed and the correct syntax should be used. Once the error has been resolved the code will match the first example given.

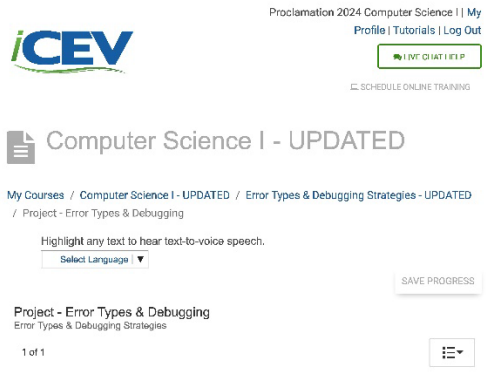
Another example of a potential problem and solution is shown in the example below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(("Enter the first number: "))
    num2 = float(("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

In this example, the code was copied over from a reference material, but the programmer missed putting in the input function after the float function. Without the input function, the numbers added by the user would not become a string, and then into a float. This would display an error. One way to fix this issue would be to use a search engine to find the error or double-check the source of the code for missing information. This issue could also be resolved by analyzing the reference material and double checking for errors. The corrected code will appear as shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
```

Update to Content Accepted by SRP



Project Overview:
You will conduct research on error types and debugging techniques to create a slide presentation to detail your findings.

Directions:

1. Conduct research on error types and debugging techniques.
2. Create a slide presentation to share your findings. Your presentation should cover the following topics:
   - What are errors
   - How to test program solutions with valid and invalid test data
     - Include coding examples for both data types
   - Common types of errors with invalid test data in Python programming, such as:
     - syntax errors
     - run-time errors
     - type errors
     - logic errors
     - lexical errors
   - Resulting behavior after using valid and invalid data in Python programming
   - Techniques for debugging errors while working with invalid test data, including built-in tools and advanced techniques
   - Best practices for debugging
3. Include examples in your presentation to illustrate the concepts. You can create your own code examples or use examples from online resources. Make sure to cite any sources used.
   - Use your IDE to write a program that has at least three errors, take a screenshot of those errors and insert into your presentation
   - Make the necessary changes so the program runs correctly, take a screenshot and insert into your presentation
4. Once complete, upload your presentation in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted to complete the assignment • Sources were cited appropriately • Error types and debugging information was presented in a logical organized manner	10
Concept & Understanding: • Understanding of the error types is clearly evident • Effective strategies were used to include coding examples • Techniques for debugging is clearly understood	40
Creativity/Craftmanship: • Slide presentation is unique and reflects the student's or group's individuality • Slide presentation is clearly high quality	10
Production/Effort: • Class time provided for the slide presentation was used efficiently • Slide presentation answers all topic bullets listed • Time and effort are evident in the execution of the slide presentation	40
Total Points	100

Update to Content Accepted by SRP

(SE)(Breakout(s)) and (Citation Type(s))
(4)(I)(iv), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Error Types and Debugging Strategies (Slide 18),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644

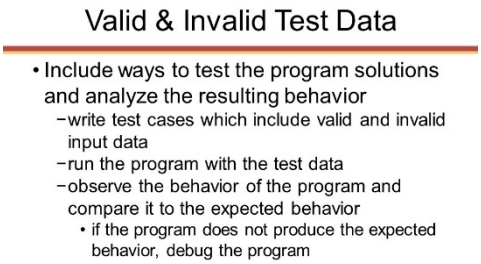
Activity-Debugging, Project-Error Types and Debugging,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644/CEV71511_Activity01

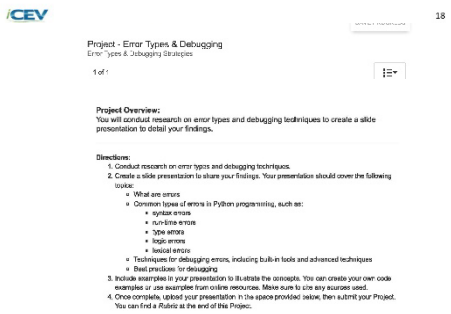
Description of the specific location and hyperlink to the exact location of the proposed new content
Error Types and Debugging Strategies Student Handout-Testing Program Solutions,
https://login.icevonline.com/download/c46dac71-1b97-4a29-b4b4-8f45c0d2c910
This Student Handout is found in the Error Types & Debugging Strategies lesson beneath the Instructional Materials heading.

Project-Error Types and Debugging,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22298/CEV71511_V2_Project01
This Project is found in the Error Types & Debugging Strategies lesson beneath the Interactive Assignments heading. After clicking the link to the Project, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Project.

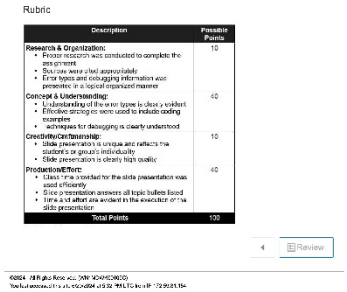
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.





Update to Content Accepted by SRP



Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted to complete the assignment • Sources were cited appropriately • Error types and debugging information was presented in a logical organized manner	10
Concept & Understanding: • Understanding of the error types is clear y evident • Effective strategies were used to include coding examples • techniques for debugging is clearly understood	40
Creativity/Craftsmanship: • Slide presentation is unique and reflects the student's or group's individuality • Slide presentation is clearly high quality	10
Production/Effort: • Class time provided for the slide presentation was used efficiently • Slide presentation answers all topic bullets listed • Time and effort are evident in the execution of the slide presentation	40
Total Points	100

Review

©2024 All Rights Reserved. WGU

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Testing Program Solutions

Testing Program Solutions with Valid and Invalid Test Data
Test data is what a user inputs that allows a function to be performed to test a system of programs. Valid test data is when the information added is in a recognizable range and format. When test data is invalid, it is not recognized by the program and will produce errors. The code below is an example of a program with invalid test data.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: 8 "))
    num2 = float(input("Enter the second number: none "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When examining the code, the num2 input is not a number, but a word inserted instead. In this scenario, the code would not recognize a term other than a number and would not produce an answer. This would result in a ValueError code if tested. To make this test data valid, the word 'none' would need to be replaced with a number. Analyzing the resulting behavior and comparing it to the expected behavior is an important step in debugging any errors.

Solving Problems
The following task is an example of a program that is correctly coded and will display a correct solution.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

The following task is an example of the same code as above, but with an error that will cause the program to display an error message.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When this code is inputted in Python, the following error message will occur.

```
File "example.py", line 1
    num1 = float(input("Enter the first number: "
                                                 ^
SyntaxError: unexpected EOF while parsing
```

After reading the error message, it is noted there is a syntax error near the end of line four. To solve this problem, the code must be fixed and the correct syntax should be used. Once the error has been resolved the code will match the first example given.

Another example of a potential problem and solution is shown in the example below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(("Enter the first number: "))
    num2 = float(("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

In this example, the code was copied over from a reference material, but the programmer missed putting in the input function after the float function. Without the input function, the numbers added by the user would not become a string, and then into a float. This would display an error. One way to fix this issue would be to use a search engine to find the error or double-check the source of the code for missing information. This issue could also be resolved by analyzing the reference material and double checking for errors. The corrected code will appear as shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
```

```
    print(f"The sum of {num1} and {num2} is: {result}")
```

Another example of a code with errors is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of num1 and num2 is: result")
```

In this code, there are no brackets around the 'num1' and num2' or the 'result' displays in the print function. This would cause a runtime error, possibly causing the program to crash. To resolve this issue, look at language documentation such as Python documentation. This programming language code documentation can help explain what is required for a code to be functional. The corrected code is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

CEV Copyright CEV Multimedia, LLC

Project - Error Types & Debugging

Error Types & Debugging Strategies

1 of 1

Project Overview:
You will conduct research on error types and debugging techniques to create a slide presentation to detail your findings.

Directions:

1. Conduct research on error types and debugging techniques.
2. Create a slide presentation to share your findings. Your presentation should cover the following topics:
    - What are errors
    - How to test program solutions with valid and invalid test data
        - Include coding examples for both data types
    - Common types of errors with invalid test data in Python programming, such as:
        - syntax errors
        - run-time errors
        - type errors
        - logic errors
        - lexical errors
    - Resulting behavior after using valid and invalid data in Python programming
    - Techniques for debugging errors while working with invalid test data, including built-in tools and advanced techniques
    - Best practices for debugging
3. Include examples in your presentation to illustrate the concepts. You can create your own code examples or use examples from online resources. Make sure to cite any sources used.
    - Use your IDE to write a program that has at least three errors, take a screenshot of those errors and insert into your presentation
    - Make the necessary changes so the program runs correctly, take a screenshot and insert into your presentation
4. Once complete, upload your presentation in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Update to Content Accepted by SRP

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted to complete the assignment • Sources were cited appropriately • Error types and debugging information was presented in a logical organized manner	10
Concept & Understanding: • Understanding of the error types is clearly evident • Effective strategies were used to include coding examples • Techniques for debugging is clearly understood	40
Creativity/Craftmanship: • Slide presentation is unique and reflects the student's or group's individuality • Slide presentation is clearly high quality	10
Production/Effort: • Class time provided for the slide presentation was used efficiently • Slide presentation answers all topic bullets listed • Time and effort are evident in the execution of the slide presentation	40
Total Points	100

(SE)(Breakout(s)) and (Citation Type(s))
(4)(J)(v), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Error Types and Debugging Strategies (Slides 10-20),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644

Description of the specific location and hyperlink to the exact location of the proposed new content
Error Types and Debugging Strategies Student Handout-Testing Program Solutions,
https://login.icevonline.com/download/c46dac71-1b97-4a29-b4b4-8f45c0d2c910
This Student Handout is found in the Error Types & Debugging Strategies lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Debugging

- Is the process of identifying and fixing errors or bugs in a software program
- Is a crucial step in the software development process
  - helps to ensure the final product is reliable and functional
- Requires careful language documentation to keep track of progress and methods

Critical Thinking

- Is often the first step in debugging
- Requires utilization of problem-solving skills, such as:
  - identifying the problem
  - brainstorming possible causes of the problem
  - planning strategies and methods to solve the problem

The Assert Statement

- Is used to check if a certain condition is true, and if it is not, an *AssertionError* is raised

```
def divide(a, b):
    assert b != 0, "division by zero is not allowed"
    return a / b

print(divide(4, 2))
print(divide(4, 0))

2.0
Traceback (most recent call last):
  File "<stdin>", line 8, in <module>
  File "<stdin>", line 2, in divide
AssertionError: division by zero is not allowed
```


Try & Except Statements

- Are made up of:
  - try block
    - contains the code which may cause an error
  - except block
    - contains the code which will be executed in case of an error

Print Statements

- Is a technique involving adding print statements to the code to display the values of variables and check the flow of the program
  - displays the values of variables
  - checks the flow of the program
  - helps identify where the error is occurring and which values are being processed at the point

Interactive Debugging

- Is a technique involving a debugger tool
  - allows to step through the code, pause it at any point, inspect variables and their values, and evaluate expressions
- Tools are available in most integrated development environments (IDEs) for Python

Error-Checking Functions

- Are built-in functions, such as the built-in function *isinstance()*
  - can be used to check for errors

```
Value = 123
if isinstance(value, int):
    print("integer")
else:
    print("not an integer")
```


Code Instrumentation

- Involves adding extra code to the program to help diagnose and isolate errors
- Can include adding logging statements to:
  - log messages
  - performance metrics
  - adding assertions
  - check for certain conditions which must be true at a certain point in the code

Update to Content Accepted by SRP

Valid & Invalid Test Data

- Include ways to test the program solutions and analyze the resulting behavior
  - write test cases which include valid and invalid input data
  - run the program with the test data
  - observe the behavior of the program and compare it to the expected behavior
    - if the program does not produce the expected behavior, debug the program

CEV

18

Error Messages

- Can be debugged and solved using the following steps:
  - reading the error message and trying to understand what it is saying
  - looking up the error message in the Python documentation or on a search engine to find explanations and solutions

CEV

19

Error Messages

- Can be debugged and solved using the following steps:
  - checking the code and comparing it to the documentation to see if there are any syntax errors or other mistakes
  - experimenting with different solutions and using the print statements and interactive debugging techniques to help diagnose the problem

CEV

20

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Testing Program Solutions

Testing Program Solutions with Valid and Invalid Test Data

Test data is what a user inputs that allows a function to be performed to test a system of programs. Valid test data is when the information added is in a recognizable range and format. When test data is invalid, it is not recognized by the program and will produce errors. The code below is an example of a program with invalid test data.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: 8 "))
    num2 = float(input("Enter the second number: none "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When examining the code, the num2 input is not a number, but a word inserted instead. In this scenario, the code would not recognize a term other than a number and would not produce an answer. This would result in a ValueError code if tested. To make this test data valid, the word 'none' would need to be replaced with a number. Analyzing the resulting behavior and comparing it to the expected behavior is an important step in debugging any errors.

Solving Problems

The following task is an example of a program that is correctly coded and will display a correct solution.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

The following task is an example of the same code as above, but with an error that will cause the program to display an error message.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When this code is inputted in Python, the following error message will occur.

```
File "example.py", line 1
    num1 = float(input("Enter the first number: "
                                                ^
SyntaxError: unexpected EOF while parsing
```

After reading the error message, it is noted there is a syntax error near the end of line four. To solve this problem, the code must be fixed and the correct syntax should be used. Once the error has been resolved the code will match the first example given.

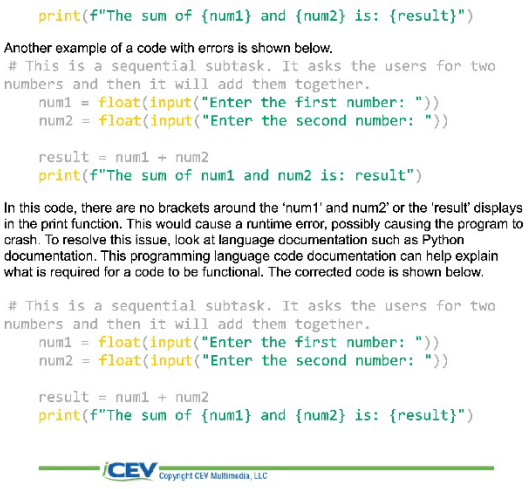
Another example of a potential problem and solution is shown in the example below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(("Enter the first number: "))
    num2 = float(("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

In this example, the code was copied over from a reference material, but the programmer missed putting in the input function after the float function. Without the input function, the numbers added by the user would not become a string, and then into a float. This would display an error. One way to fix this issue would be to use a search engine to find the error or double-check the source of the code for missing information. This issue could also be resolved by analyzing the reference material and double checking for errors. The corrected code will appear as shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
```

Update to Content Accepted by SRP



```
    print(f"The sum of {num1} and {num2} is: {result}")
```

Another example of a code with errors is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of num1 and num2 is: result")
```

In this code, there are no brackets around the 'num1' and num2' or the 'result' displays in the print function. This would cause a runtime error, possibly causing the program to crash. To resolve this issue, look at language documentation such as Python documentation. This programming language code documentation can help explain what is required for a code to be functional. The corrected code is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(4)(J)(vi), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Error Types and Debugging Strategies (Slides 10-20),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644

Description of the specific location and hyperlink to the exact location of the proposed new content
Error Types and Debugging Strategies Student Handout-Testing Program Solutions,
https://login.icevonline.com/download/c46dac71-1b97-4a29-b4b4-8f45c0d2c910
This Student Handout is found in the Error Types & Debugging Strategies lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Debugging

- Is the process of identifying and fixing errors or bugs in a software program
- Is a crucial step in the software development process
  - helps to ensure the final product is reliable and functional
- Requires careful language documentation to keep track of progress and methods

Critical Thinking

- Is often the first step in debugging
- Requires utilization of problem-solving skills, such as:
  - identifying the problem
  - brainstorming possible causes of the problem
  - planning strategies and methods to solve the problem

The Assert Statement

- Is used to check if a certain condition is true, and if it is not, an *AssertionError* is raised

```
def divide(a, b):
    assert b != 0, "division by zero is not allowed"
    return a / b

print(divide(4, 2))
print(divide(4, 0))

2.0
Traceback (most recent call last):
  File "<stdin>", line 8, in <module>
  File "<stdin>", line 2, in divide
AssertionError: division by zero is not allowed
```


Try & Except Statements

- Are made up of:
  - try block
    - contains the code which may cause an error
  - except block
    - contains the code which will be executed in case of an error

Print Statements

- Is a technique involving adding print statements to the code to display the values of variables and check the flow of the program
  - displays the values of variables
  - checks the flow of the program
  - helps identify where the error is occurring and which values are being processed at the point

Interactive Debugging

- Is a technique involving a debugger tool
  - allows to step through the code, pause it at any point, inspect variables and their values, and evaluate expressions
- Tools are available in most integrated development environments (IDEs) for Python

Error-Checking Functions

- Are built-in functions, such as the built-in function *isinstance()*
  - can be used to check for errors

```
Value = 123
if isinstance(value, int):
    print("integer")
else:
    print("not an integer")
```


Code Instrumentation

- Involves adding extra code to the program to help diagnose and isolate errors
- Can include adding logging statements to:
  - log messages
  - performance metrics
  - adding assertions
  - check for certain conditions which must be true at a certain point in the code

Update to Content Accepted by SRP

Valid & Invalid Test Data

- Include ways to test the program solutions and analyze the resulting behavior
  - write test cases which include valid and invalid input data
  - run the program with the test data
  - observe the behavior of the program and compare it to the expected behavior
    - if the program does not produce the expected behavior, debug the program

CEV

18

Error Messages

- Can be debugged and solved using the following steps:
  - reading the error message and trying to understand what it is saying
  - looking up the error message in the Python documentation or on a search engine to find explanations and solutions

CEV

19

Error Messages

- Can be debugged and solved using the following steps:
  - checking the code and comparing it to the documentation to see if there are any syntax errors or other mistakes
  - experimenting with different solutions and using the print statements and interactive debugging techniques to help diagnose the problem

CEV

20

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Testing Program Solutions

Testing Program Solutions with Valid and Invalid Test Data
Test data is what a user inputs that allows a function to be performed to test a system of programs. Valid test data is when the information added is in a recognizable range and format. When test data is invalid, it is not recognized by the program and will produce errors. The code below is an example of a program with invalid test data.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: 8 "))
    num2 = float(input("Enter the second number: none "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When examining the code, the num2 input is not a number, but a word inserted instead. In this scenario, the code would not recognize a term other than a number and would not produce an answer. This would result in a ValueError code if tested. To make this test data valid, the word 'none' would need to be replaced with a number. Analyzing the resulting behavior and comparing it to the expected behavior is an important step in debugging any errors.

Solving Problems
The following task is an example of a program that is correctly coded and will display a correct solution.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

The following task is an example of the same code as above, but with an error that will cause the program to display an error message.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When this code is inputted in Python, the following error message will occur.

```
File "example.py", line 1
    num1 = float(input("Enter the first number: "
                                                ^
SyntaxError: unexpected EOF while parsing
```

After reading the error message, it is noted there is a syntax error near the end of line four. To solve this problem, the code must be fixed and the correct syntax should be used. Once the error has been resolved the code will match the first example given.

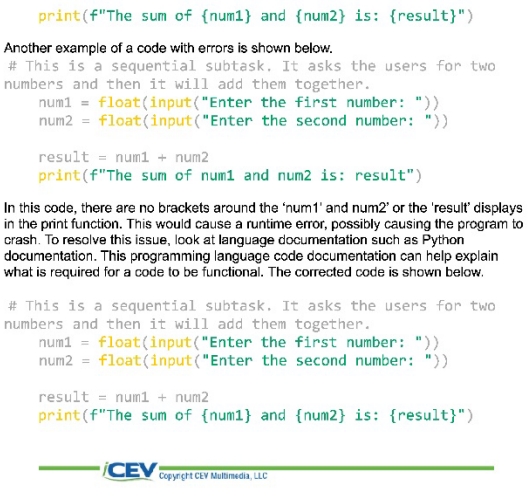
Another example of a potential problem and solution is shown in the example below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(("Enter the first number: "))
    num2 = float(("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

In this example, the code was copied over from a reference material, but the programmer missed putting in the input function after the float function. Without the input function, the numbers added by the user would not become a string, and then into a float. This would display an error. One way to fix this issue would be to use a search engine to find the error or double-check the source of the code for missing information. This issue could also be resolved by analyzing the reference material and double checking for errors. The corrected code will appear as shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
```

Update to Content Accepted by SRP



```
    print(f"The sum of {num1} and {num2} is: {result}")
```

Another example of a code with errors is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of num1 and num2 is: result")
```

In this code, there are no brackets around the 'num1' and num2' or the 'result' displays in the print function. This would cause a runtime error, possibly causing the program to crash. To resolve this issue, look at language documentation such as Python documentation. This programming language code documentation can help explain what is required for a code to be functional. The corrected code is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(4)(J)(vii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Error Types and Debugging Strategies (Slides 10-20),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21644

Description of the specific location and hyperlink to the exact location of the proposed new content
Error Types and Debugging Strategies Student Handout-Testing Program Solutions,
https://login.icevonline.com/download/c46dac71-1b97-4a29-b4b4-8f45c0d2c910
This Student Handout is found in the Error Types & Debugging Strategies lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Debugging

- Is the process of identifying and fixing errors or bugs in a software program
- Is a crucial step in the software development process
  - helps to ensure the final product is reliable and functional
- Requires careful language documentation to keep track of progress and methods

Critical Thinking

- Is often the first step in debugging
- Requires utilization of problem-solving skills, such as:
  - identifying the problem
  - brainstorming possible causes of the problem
  - planning strategies and methods to solve the problem

The Assert Statement

- Is used to check if a certain condition is true, and if it is not, an *AssertionError* is raised

```
def divide(a, b):
    assert b != 0, "division by zero is not allowed"
    return a / b

print(divide(4, 2))
print(divide(4, 0))

2.0
Traceback (most recent call last):
  File "<stdin>", line 8, in <module>
  File "<stdin>", line 2, in divide
AssertionError: division by zero is not allowed
```


Try & Except Statements

- Are made up of:
  - try block
    - contains the code which may cause an error
  - except block
    - contains the code which will be executed in case of an error

Print Statements

- Is a technique involving adding print statements to the code to display the values of variables and check the flow of the program
  - displays the values of variables
  - checks the flow of the program
  - helps identify where the error is occurring and which values are being processed at the point

Interactive Debugging

- Is a technique involving a debugger tool
  - allows to step through the code, pause it at any point, inspect variables and their values, and evaluate expressions
- Tools are available in most integrated development environments (IDEs) for Python

Error-Checking Functions

- Are built-in functions, such as the built-in function *isinstance()*
  - can be used to check for errors

```
Value = 123
if isinstance(value, int):
    print("integer")
else:
    print("not an integer")
```


Code Instrumentation

- Involves adding extra code to the program to help diagnose and isolate errors
- Can include adding logging statements to:
  - log messages
  - performance metrics
  - adding assertions
  - check for certain conditions which must be true at a certain point in the code

Update to Content Accepted by SRP

Valid & Invalid Test Data

- Include ways to test the program solutions and analyze the resulting behavior
  - write test cases which include valid and invalid input data
  - run the program with the test data
  - observe the behavior of the program and compare it to the expected behavior
    - if the program does not produce the expected behavior, debug the program

CEV

18

Error Messages

- Can be debugged and solved using the following steps:
  - reading the error message and trying to understand what it is saying
  - looking up the error message in the Python documentation or on a search engine to find explanations and solutions

CEV

19

Error Messages

- Can be debugged and solved using the following steps:
  - checking the code and comparing it to the documentation to see if there are any syntax errors or other mistakes
  - experimenting with different solutions and using the print statements and interactive debugging techniques to help diagnose the problem

CEV

20

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Testing Program Solutions

Testing Program Solutions with Valid and Invalid Test Data
Test data is what a user inputs that allows a function to be performed to test a system of programs. Valid test data is when the information added is in a recognizable range and format. When test data is invalid, it is not recognized by the program and will produce errors. The code below is an example of a program with invalid test data.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: 8 "))
    num2 = float(input("Enter the second number: none "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When examining the code, the num2 input is not a number, but a word inserted instead. In this scenario, the code would not recognize a term other than a number and would not produce an answer. This would result in a ValueError code if tested. To make this test data valid, the word 'none' would need to be replaced with a number. Analyzing the resulting behavior and comparing it to the expected behavior is an important step in debugging any errors.

Solving Problems
The following task is an example of a program that is correctly coded and will display a correct solution.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

The following task is an example of the same code as above, but with an error that will cause the program to display an error message.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

When this code is inputted in Python, the following error message will occur.

```
File "example.py", line 1
    num1 = float(input("Enter the first number: "
                                                ^
SyntaxError: unexpected EOF while parsing
```

After reading the error message, it is noted there is a syntax error near the end of line four. To solve this problem, the code must be fixed and the correct syntax should be used. Once the error has been resolved the code will match the first example given.

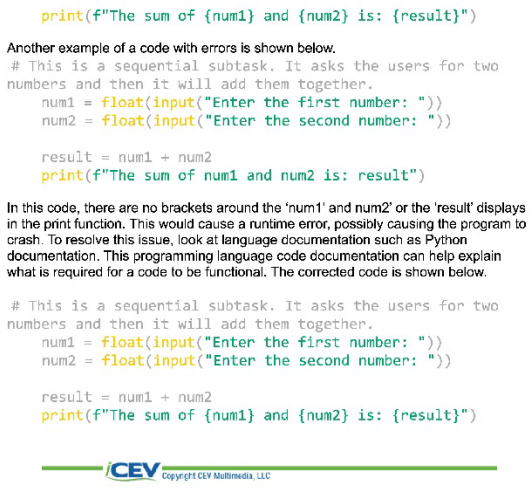
Another example of a potential problem and solution is shown in the example below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(("Enter the first number: "))
    num2 = float(("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

In this example, the code was copied over from a reference material, but the programmer missed putting in the input function after the float function. Without the input function, the numbers added by the user would not become a string, and then into a float. This would display an error. One way to fix this issue would be to use a search engine to find the error or double-check the source of the code for missing information. This issue could also be resolved by analyzing the reference material and double checking for errors. The corrected code will appear as shown below.

```
# This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
```

Update to Content Accepted by SRP



```
    print(f"The sum of {num1} and {num2} is: {result}")
```

Another example of a code with errors is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of num1 and num2 is: result")
```

In this code, there are no brackets around the 'num1' and num2' or the 'result' displays in the print function. This would cause a runtime error, possibly causing the program to crash. To resolve this issue, look at language documentation such as Python documentation. This programming language code documentation can help explain what is required for a code to be functional. The corrected code is shown below.

```
 # This is a sequential subtask. It asks the users for two
numbers and then it will add them together.
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    result = num1 + num2
    print(f"The sum of {num1} and {num2} is: {result}")
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(4)(Q)(i), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Problem Solving with Algorithms Programming Concepts (00:50 - 1:41),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21655

Description of the specific location and hyperlink to the exact location of the proposed new content
Problem Solving with Algorithms Student Handout-Problem Solving with Algorithms,
https://files.icevonline.com/html/CEV81115_V2_HTML/CEV81115_V2_HTML_Student_Handout_-_Problem_Solving_with_Algorithms.htm
This Student Handout is found in the Problem Solving with Algorithms lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Problem Solving with Algorithms

The content within this transcript has been created utilizing a third-party software company which complies with all federal accessibility laws and international standards for web accessibility, providing a measured accuracy rate of 99.6 percent.

1. Common Algorithms

TEXT ON SCREEN
Problem Solving with Algorithms
Common Algorithms

What are algorithms? An algorithm is a set of well-defined logical steps to follow when performing a task. For example, the procedure or algorithm for boiling water would be fill a pot with water. Place the pot on a burner on the stove. Turn on the burner to high. Observe the water until it starts bubbling. When the water is bubbling, the water is boiling.

When are algorithms used? Algorithms are used in every part of computer science. They form the field's backbone. In computer science, an algorithm gives the computer a specific set of instructions, which allows the computer to do everything, be it running a calculator or running a rocket.

Computer programs are at their core algorithms written in programming languages that the computer can understand. Computer algorithms play a big role in how social media works, which posts show up, which ads are seen, and so on. These decisions are all made by algorithms.

Programmers who build search engines use algorithms to optimize searches, predict what users are going to type, and more. Much of computer programming involves using algorithms to solve problems, such as increasing the relevance of search results or making websites load faster. Programming has almost unlimited uses and applications.

Smartphones are just one example of devices utilizing programming. Although each app is programmed differently for different functions, all programming uses similar logic to process data and produce results. Another example where programming is especially useful is the field of data science and analytics. Many companies collect user data to make product decisions. These data collections are very large and need to be organized, sorted, and manipulated before analysis can begin. The programming language Python is used extensively for data analysis.

Common algorithms-- when writing an algorithm, it's often helpful to have a good understanding of some common, well-known algorithms. This helps save time because programmers can re-use or adapt the algorithm to fit their needs.

For example, to sort the items of a list, often referred to as an array, into a particular order, it is helpful to refer to sorting algorithms.
IMAGE ON SCREEN- A graphic displays four books and they are labeled A, B, C, and D, from left to right. A and D are then switched, so D is on the far left and A is on the far right. Then B and C are switched, so C is on the left in the middle and B is on the right in the middle. Therefore, the books end up in the order D, C, B, and A from left to right.

Similarly, searching algorithms are used to find and retrieve an element from wherever it is stored in a data structure.
IMAGE ON SCREEN- A computer screen is displayed with an algorithm: array [duck, duck, goose]. Then a magnifying glass is seen at the bottom by the word goose. This means the searching algorithm is looking for goose. The magnifying glass then goes up to the algorithm to find the word goose.

Finding the Greatest Common Factor, GCF. The greatest common factor, sometimes called the greatest common divisor, is the largest number that divides two or more numbers without a remainder. For example, the greatest common factor of 4 and 10 is 2 because 2 is the largest number divisible into 4 and 10 without a remainder.
IMAGE ON SCREEN- The numbers 4 and 10 are displayed. Under 4 are two lines each pointing to the number 2. Under 10 are two lines, one pointing to the number 5 and the other pointing to the number 2. Number 2 is circled under both 4 and 10.

Computing a greatest common factor is important because it allows us to reduce fractions efficiently. On paper, it may seem simple to explain the process or algorithm of finding the greatest common factor.

One way is to simply divide two numbers by all the numbers less than the smaller number until the largest number that divides into both is found. This strategy will work. However, there is a better way to reduce the number of total calculations, which increases overall efficiency.
IMAGE ON SCREEN- The title computing the greatest common factor is at the top. The numbers 4 and 10 are on screen. Underneath 4 are the following equations: 4 ÷ 2 = 1.333… marked out and 4 ÷ 2 = 2 with the 2 in the equation circled. Underneath 10 are the following equations: 10 ÷ 4 = 2.5 marked out, 10 ÷ 3 = 3.333… marked out, and 10 ÷ 2 = 5 with the 2 in the equation circled.

The Euclidean algorithm is a better way of finding the greatest common factor. It works as follows. Given two numbers, where a is the larger value and b the smaller, divide a by b and get the remainder r. Then replace a with b and replace b with r. Repeat this process until r equal 0. When the remainder equal 0, b is the greatest common factor of the two numbers.
IMAGE ON SCREEN- A graphic of a computer screen with the following text:
Find the GCF of a and b (where a > b):
Repeat until r = 0
a + b - ? r = ?
a ÷ b = ? r = ?
a ÷ b = ? r = 0

Here's an example. Find the greatest common factor of 30 and 12. First, divide the larger number 30 by the smaller number 12. The result is 2 with a remainder of 6.

Now continue and divide 12 by 6, the remainder in the previous step. 12 divided by 6 equals 2 with no remainder. Because there is no remainder, the task is complete. And the last number used to divide is the greatest common factor, which in this case is 6.

Let's do another example. Find the greatest common factor of 123 and 36. 123 divided by 36 is 3 with a remainder of 15. Continue to divide 36 by 15 because 15 was the remainder of the previous step.

36 divided by 15 equals 2 with a remainder of 6. Continue to divide 15 by 6, which equals 2 with a remainder of 3. And, finally, 6 divided by 3 equals 2 with no remainder. Therefore, our greatest common factor is 3.

TEXT ON SCREEN
Common Algorithms:…
…Finding the biggest number-- here is another type of calculation.

Given three integer numbers, called num1, num2, and num3, respectively find the largest number using an algorithm like this. Check if num1 is greater than num2. If true, check if num1 is greater than num3. If true, then show num1 is the largest number. If false, then show num3 as the largest number. If num1 is not greater than num2, check if num2 is greater than num3. If true, then show num2 to as the largest number. If false, then show num3 as the largest number. This algorithm or set of steps will always work given any three numbers.
IMAGE ON SCREEN- A graphic organizer is displayed.
The very top box: find the largest number [num1], [num2], [num3]
The next box branching down from the top box: is [num1] > [num2] ?
Branching from the above box it splits into two boxes: on the left is true and on the right is false.

Boxes underneath true:
Is [num1] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num1] is the largest number
Box underneath False: [num3 is the largest number
Boxes underneath False:
Is [num2] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num2] is the largest number
Box underneath False: [num3] is the largest number

Let's work through an example. Suppose num1 equals 6, num2 equals 10, and num3 equals 25. Now use the algorithm above to determine the largest number. Is num1 greater than num2? In other words, is 6 greater than 10? No. Therefore, the next question is, is num2 to greater than num3? In other words, is 10 greater than 25? The answer is no. So the output states num3 or 25 must be the largest number.
IMAGE ON SCREEN- A graphic organizer is displayed.
The very top box: find the largest number num = 6, num2 = 10, num3 = 25
The next box branching down from the top box: is [num1] > [num2] ? 6 > 10 ?
Branching from the above box it splits into two boxes: on the left is true and on the right is false.
Boxes underneath true:
is [num1] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num1] is the largest number
Box underneath False: [num3 is the largest number
Boxes underneath False:
is [num2] > [num3] ? 10 > 25 ?
Branching into two: True on the left or False on the right
Box underneath True: [num2] is the largest number
Box underneath False: [num3] is the largest number, 25 is the largest number

TEXT ON SCREEN
Common Algorithms:…
…Finding primes-- a prime number is a positive integer greater than 1 that can only be divided by itself and 1. The numbers 2, 3, 5, 7, et cetera are prime numbers, as they do not have any other factors.
IMAGE ON SCREEN- A chart of prime numbers is displayed. The chart includes numbers:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379,

Update to Content Accepted by SRP

383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 967, 971, 977, 983, 991, 997, 953

To test if a number is prime, check every number greater than 1 to see if the number divides the potential prime number.

One of the most widely used applications of prime numbers in computing is an encryption system. In 1978, Ron Rivest, Adi Shamir, and Leonhard Adleman combined some simple known facts about numbers to create RSA encryption, named after the inventors. The system they developed allows for the secure transmission of information, such as credit card numbers, online. The first ingredient required for the algorithm are two large prime numbers. The larger the numbers, the safer the encryption.

TEXT ON SCREEN
Common Algorithms:...
...Making change-- when making a purchase with cash, the amount of change due is calculated using another common algorithm. Find the sum of all the purchases to store in a variable named totalCost. Input the cash paid in a different variable named amountPaid. Subtract total cost from amountPaid to find the change stored in a third variable named change. change equals amount paid minus totalCost.

TEXT ON SCREEN
Common Algorithms:...
...Finding the average-- to find the arithmetic average or mean, add up a list of numbers. And then divide by the quantity of numbers summed up. Here is a set of eight numbers-- 4, 5, 1, 2, 9, 7, 10, 8. The sum of the numbers is 4 plus 5 plus 1 plus 2 plus 9 plus 7 plus 10 plus 8, which equals 46. So the average is 46 divided by 8, which comes to 5.75.

2. Programming Concepts

TEXT ON SCREEN
Problem Solving with Algorithms
Programming Concepts

Part of learning how to program is learning what tools and functions can be leveraged to solve problems and write custom algorithms. When it comes to solving problems, the first step is to fully understand the problem at hand. Without proper understanding, it's

easy to make assumptions about problems and potential solutions without really getting a clear picture of what's going on.

Once a problem has been identified, take time to consider a solution and focus on writing clear instructions one step at a time. Test the code to ensure it works as expected. Then continue to the next step.

Importance of order when using algorithms. Sequential algorithms follow a step-by-step process, and the order of those steps are crucial to ensuring the correctness of an algorithm.

Here's an algorithm for translating a word into Pig Latin.
TEXT ON SCREEN
PIG

One, append a hyphen.
TEXT ON SCREEN
PIG-

Two, append the first letter.
TEXT ON SCREEN
PIG-P

Three, append A, Y.
TEXT ON SCREEN
PIG-PAY

Four, remove the first letter.
TEXT ON SCREEN
IG-PAY

If the steps are completed out of order, the result would be different and incorrect.
TEXT ON SCREEN
PIG=IG-PAY in Pig Latin

Sequential algorithms must follow the same order of the steps to be effective.
TEXT ON SCREEN
1
2
3
4
5
...

For example, if a contact form prompts a user to enter data into a database, the program must first ask the user for the data before it can be saved to the database. Otherwise, there is no data to save.
IMAGE ON SCREEN- A graphic of website. On the website is the text Join the mailing list and there is a place to enter an email address. Then there is a button to submit.

Conditional logic. Decision-making problems involve conditional statements or branching. Conditional statements are features of programming languages that tell the computer to execute certain actions provided certain user-defined conditions are met.
IMAGE ON SCREEN- Conditional statement or branching graphic.
Begins at the word START
Branches to an IF (condition) box
The IF statement branches into false on the left or true on the right
If false it branches to an ELSE (code) statement
If true it branches to a THEN (code) statement
Both else and then statements branch into the END statement

For example, a program used to enter student grades may ask a teacher to input grades until the entire class has a grade. Once that condition is met, the program will stop asking the teacher for grades.

Conditional statements are used through the various programming languages to instruct the computer on the decision to make when given some conditions. These decisions are made if and only if the pre-stated conditions are either true or false, depending on the functions the programmer has in mind.
IMAGE ON SCREEN- Conditional statement or branching graphic.
Begins at the word START
Branches to an IF (condition) box
The IF statement branches into false on the left or true on the right
If false it branches to an ELSE (code) statement
If true it branches to a THEN (code) statement
Both else and then statements branch into the END statement

All programming languages have conditional expression syntax, although the syntax slightly differs from one programming language to the others.

Branching versus non-branching. When writing algorithms, it's often helpful to use a series of conditional statements to test complex criteria and execute appropriate actions, which is often referred to as branching. Using branching allows programmers to give precise instructions based on a variety of conditions and then output a relevant result. As opposed to branching, non-branching programming simply refers to

programs that don't use conditional statements a real world example would be furniture assembly instructions.

A program that doesn't use conditionals might resemble an input program that asks the user for data input to populate a database.

Think of an email sign-up form on a website. You enter your information and click Submit. There are not different paths or conditions that need to be accounted for.
IMAGE ON SCREEN- A graphic of a website contact us page. There are four areas for a user to fill out: Name- Jonah Torres, Email- jonahpt@email.com, Subject- Hours, and Message- What are your updated store hours?

Examples of branching programming. Conditional statements have a simple job-- check whether a condition has been met and return True if yes, and False if not. Then based on those outputs, programmers can automate other decisions or actions and so on.

Suppose a programmer is working on a program for teachers to enter student grades into a grade book. The programmer should consider how the data will be entered and saved into a database and create some conditional expressions to help make the teacher's job easier and minimize errors. To help avoid typos, the program could be written to check that values entered as grades are numeric or that the scores are no greater than a maximum value.

If a wrong number was entered, the program could display the grade in bright red or reject the input until an acceptable value is entered.

Then a variable could be created to capture whether or not all scores have been entered. It would initially be set to False. And then once all grades are entered, it would be set to True.
IMAGE ON SCREEN- Numbers are counted from 1 to 100.

This type of variable is called a Boolean variable. These only have two states: True or False.

Iteration. Iteration is another way to say repetition. Programming that uses iteration will have some kind of loop that repeats a block of code. The repetition will terminate after a predetermined number of repetitions or when a certain condition is met. If there is a process needing to be run over and over again, it's best to write the code so it can loop repeatedly, cycling through inputs or adjusting variables incrementally. This saves time programming and makes the process more efficient. Any program without a need to repeat a section of code could be considered non-iterative.

Update to Content Accepted by SRP

Using loops. Iteration is implemented using the loop in programming. The two most common types of loops are for-loop and the while-loop. The for-loop is used when a loop needs to be run a certain number of times. The other type of loop, the while-loop repeats until a specific condition is true. This is particularly useful when it's difficult or impossible to predict how many times the loop will execute.

Sequential search using loops. In computer science, linear search or sequential search is a method for finding a particular value in a list by checking each element in sequence until the desired element is found or the list is exhausted. The list doesn't have to be in order.

Here is an example. The process starts with setting up variables.
TEXT ON SCREEN
GOAL: find "6"

The first variable captures the search position and is initially set to a value of 0.
TEXT ON SCREEN
var position = 0

The search position is the current position or current value being tested.
TEXT ON SCREEN
var position = 0

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

A second Boolean variable stores one of two possible values, True or False.
TEXT ON SCREEN
var position = 0
var isFound =

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

At the start, this should be set to False.
TEXT ON SCREEN
var position = 0
var isFound = false

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

Once the condition is met, the variable will be instead set to True, and the code will stop running since the desired output has been generated.
TEXT ON SCREEN
var position = 2
var isFound = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

The code works by looking at search position 0 and testing its condition.
TEXT ON SCREEN
var position = 0
var isFound = false

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

If the condition is met, the second variable is changed from False to True and the process is over.
TEXT ON SCREEN
var position = 0
var isFound = false to true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "2" equal to "6" ?

If the condition is not met, and the value of the position variable is less than the length of the list, the code is set up to increment the position variable by 1.
TEXT ON SCREEN
var position = 0 + 1
var isFound = false

0 < 3 = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "2" equal to "6" ?

This means the code will now check the contents of the second position and repeat the process. The code terminates when found equals True or when position equals the total number of positions, which means the condition was never met and found equals False still.
TEXT ON SCREEN
var position = 2
var isFound = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "6" equal to "6" ?

The power of iterative programming. Suppose a teacher wants to give their class 10 extra points on an exam. While the teacher could manually adjust each grade and do the calculation for every student, iterative program is useful because the calculation of adding 10 to each grade is repeated.

This example demonstrates the calculation using Python but the logic for "for" and "while" loops is similar across languages.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

for score in listOfTwentyScores: with for and in in dark blue and listOfTwentyScores in light blue
newScores.append(score + 10) with newScores in light blue and score, .append and 10 in yellow

print(newScores) with print(in yellow and newScores in light blue

#Output: in green
#[95, 100, 80, 85, 93, 92, 91, 55, 88, 100, 66, 87, 55, 77, 90, 100, 80, 60, 85, 100] in green

The code starts with a variable storing an array of the grades of the 20 students in the class separated with commas. This variable is called list of 20 scores.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow

A new variable is created to hold the final values. Right now, it's blank.

IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

Then for-loop starts. In plain language, this section accesses one score at a time within the list of 20 scores variable, adds 10 to it and appends it to the new score's variable. Appending the new score allows the variable to hold multiple values and not get overwritten. This repeats until all 20 scores have been adjusted and stored in the new scores variable.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

for score in listOfTwentyScores: with for and in in dark blue and listOfTwentyScores in light blue
newScores.append(score + 10) with newScores in light blue and score, .append and 10 in yellow

print(newScores) with print(in yellow and newScores in light blue

#Output: in green
#[95, 100, 80, 85, 93, 92, 91, 55, 88, 100, 66, 87, 55, 77, 90, 100, 80, 60, 85, 100] in green

Knowing how to choose the best tool. Many problems in programming can be related back to one of the algorithms previously discussed. For example, an iterative looking program should be used if certain code can be reused, while a one-time calculation or action would use a non-iterative sequential algorithm.

Computer programmers constantly use algorithms, whether it's an existing algorithm for a common problem like sorting a list of values, or a completely new algorithm unique to the program. By understanding algorithms, programmers can make better decisions about which existing algorithms to use and learn how to make new algorithms that are correct and efficient.

CEV Copyright CEV Multimedia, LLC

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Problem Solving with Algorithims

Sequential Algorithm for Non-Branching Problems:
In computer programming and algorithm design, the term "non-branching" refers to a situation or code structure where there are no decision points or branches based on conditional statements.

- In a non-branching scenario, the code proceeds sequentially without any alternative paths based on conditions
- Sequential algorithms for non-branching problems often involve iteration, calculation or manipulation of data without the need to make choices based on conditions
- Examples:
  - summing an array
  - finding the maximum element in a list
  - reversing the order of elements in a sequence

Example: Sequential algorithm for calculating the sum of an array
Problem: Given an array of numbers, calculate the sum of all the elements
Algorithm:

```
1. Initialize a variable sum to 0.
2. For each element in the array:
   a. Add the element to the sum.
3. Display or return the sum.
```

Sequential Algorithm for Non-Iterative Problems:
Non-iterative problems refer to computational or algorithmic problems that do not require the use of iteration or repetitive structures to arrive at a solution.

- Iteration typically involves using loops or recursive functions to perform a set of instructions repeatedly until a specific condition is met
- Examples:
  - mathematical computations
    - for example: calculating the square root of a number
  - string manipulations
    - for example: checking if a string is a palindrome
  - direct conversions or computations that can be accomplished in a single pass through the data
    - for example: calculating the average of a list of numbers

Example: Algorithm to convert Celsius to Fahrenheit
Problem: Convert a temperature from Celsius to Fahrenheit
Algorithm:

```
1. Accept a temperature in Celsius as input.
2. Calculate the equivalent temperature in Fahrenheit using the
formula: Fahrenheit = (Celsius * 9/5) + 32.
3. Display or return the temperature in Fahrenheit.
```

Sequential Algorithm for Branching Control Statements:
Branching control statements, also known as conditional statements, are programming constructs that enable the execution of different sets of instructions based on certain conditions or criteria. These statements allow the flow of a program to branch into different paths, depending on whether a specified condition is true or false. The two primary branching control statements are the "if" statement and the "switch" statement.

- The "if" statement is a fundamental branching control statement that allows a program to make decisions based on a given condition
  - The "if" statement can be extended with an optional "else" clause to specify an alternative set of instructions to be executed when the condition is false
- The "switch" statement is used when there are multiple possible conditions to be checked against a single expression
  - The "switch" statement compares the expression against each case, and if a match is found, the corresponding block of code is executed. The "break" statement is used to exit the switch block

Example: Algorithm to determine if a number is even or odd
Problem: Determine if a given integer is even or odd
Algorithm:

```
1. Accept an integer as input.
2. If the input modulo 2 equals 0:
   a. Display "The number is even."
3. Else:
   a. Display "The number is odd."
```

Including a switch:

```
1. Accept an integer as input.
2. Switch on the result of the input modulo 2:
   a. Case 0:
      i. Display "The number is even."
      ii. Break.
   b. Case 1:
      i. Display "The number is odd."
      ii. Break.
   c. Default:
      i. Display "Invalid input." (optional, in case the input is not
an integer)
```

Random Numbers:

Unpredictable and are often used in various algorithms and applications

The following are examples of random numbers in different languages:
C++: 'rand()'
Python: 'random()'
JavaScript: 'Math.random()'

Loops:
Structures that repeat a specific block of code until a certain condition is met

The following are examples of loops in different languages:

Python:

```python
# Problem: Print numbers from 1 to 5

# Solution using a for loop
for i in range(1, 6):
    print(i, end=' ')

print()  # Move to the next line
```

Java:

```java
public class LoopExample {
    public static void main(String[] args) {
        // Problem: Print numbers from 1 to 5

        // Solution using a for loop
        for (int i = 1; i <= 5; ++i) {
            System.out.print(i + " ");
        }

        System.out.println(); // Move to the next line
    }
}
```

C++:

```cpp
#include <iostream>

int main() {
    // Problem: Print numbers from 1 to 5

    // Solution using a for loop
    for (int i = 1; i <= 5; ++i) {
        std::cout << i << " ";
    }

    std::cout << std::endl; // Move to the next line

    return 0;
}
```

Conditionals:
Allow the execution of different code blocks based on whether a specified condition evaluates to true or false

The following are examples of conditionals in different languages:

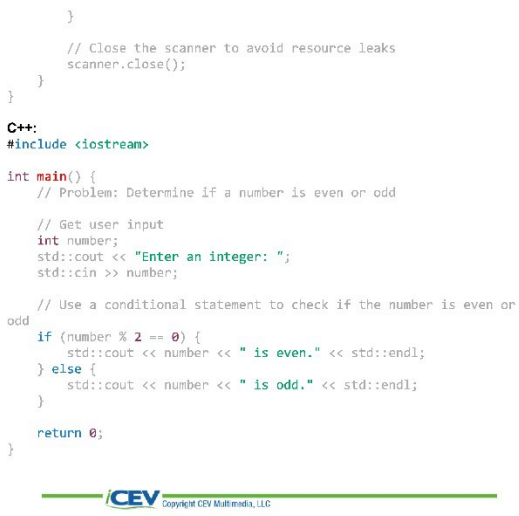
Python:

```python
# Problem: Determine if a number is even or odd

# Get user input
number = int(input("Enter an integer: "))

# Use a conditional statement to check if the number is even or odd
if number % 2 == 0:
    print(f"{number} is even.")
else:
    print(f"{number} is odd.")
```

Java:

```java
import java.util.Scanner;

public class ConditionalExample {
    public static void main(String[] args) {
        // Problem: Determine if a number is even or odd

        // Get user input
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Use a conditional statement to check if the number is even
or odd
        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
```

Update to Content Accepted by SRP



```
        }

        // Close the scanner to avoid resource leaks
        scanner.close();
    }
}
```

C++:

```
#include <iostream>

int main() {
    // Problem: Determine if a number is even or odd

    // Get user input
    int number;
    std::cout << "Enter an integer: ";
    std::cin >> number;

    // Use a conditional statement to check if the number is even or
odd
    if (number % 2 == 0) {
        std::cout << number << " is even." << std::endl;
    } else {
        std::cout << number << " is odd." << std::endl;
    }

    return 0;
}
```

iCEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(4)(Q)(ii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Problem Solving with Algorithms Programming Concepts (00:50 - 1:41),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21655

Description of the specific location and hyperlink to the exact location of the proposed new content
Problem Solving with Algorithms Student Handout-Problem Solving with Algorithms,
https://files.icevonline.com/html/CEV81115_V2_HTML/CEV81115_V2_HTML_Student_Handout_-_Problem_Solving_with_Algorithms.htm
This Student Handout is found in the Problem Solving with Algorithms lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Problem Solving with Algorithms

The content within this transcript has been created utilizing a third-party software company which complies with all federal accessibility laws and international standards for web accessibility, providing a measured accuracy rate of 99.6 percent.

1. Common Algorithms

TEXT ON SCREEN
Problem Solving with Algorithms
Common Algorithms

What are algorithms? An algorithm is a set of well-defined logical steps to follow when performing a task. For example, the procedure or algorithm for boiling water would be fill a pot with water. Place the pot on a burner on the stove. Turn on the burner to high. Observe the water until it starts bubbling. When the water is bubbling, the water is boiling.

When are algorithms used? Algorithms are used in every part of computer science. They form the field's backbone. In computer science, an algorithm gives the computer a specific set of instructions, which allows the computer to do everything, be it running a calculator or running a rocket.

Computer programs are at their core algorithms written in programming languages that the computer can understand. Computer algorithms play a big role in how social media works, which posts show up, which ads are seen, and so on. These decisions are all made by algorithms.

Programmers who build search engines use algorithms to optimize searches, predict what users are going to type, and more. Much of computer programming involves using algorithms to solve problems, such as increasing the relevance of search results or making websites load faster. Programming has almost unlimited uses and applications.

Smartphones are just one example of devices utilizing programming. Although each app is programmed differently for different functions, all programming uses similar logic to process data and produce results. Another example where programming is especially useful is the field of data science and analytics. Many companies collect user data to make product decisions. These data collections are very large and need to be organized, sorted, and manipulated before analysis can begin. The programming language Python is used extensively for data analysis.

Common algorithms-- when writing an algorithm, it's often helpful to have a good understanding of some common, well-known algorithms. This helps save time because programmers can re-use or adapt the algorithm to fit their needs.

For example, to sort the items of a list, often referred to as an array, into a particular order, it is helpful to refer to sorting algorithms.
IMAGE ON SCREEN- A graphic displays four books and they are labeled A, B, C, and D, from left to right. A and D are then switched, so D is on the far left and A is on the far right. Then B and C are switched, so C is on the left in the middle and B is on the right in the middle. Therefore, the books end up in the order D, C, B, and A from left to right.

Similarly, searching algorithms are used to find and retrieve an element from wherever it is stored in a data structure.
IMAGE ON SCREEN- A computer screen is displayed with an algorithm: array [duck, duck, goose]. Then a magnifying glass is seen at the bottom by the word goose. This means the searching algorithm is looking for goose. The magnifying glass then goes up to the algorithm to find the word goose.

Finding the Greatest Common Factor, GCF. The greatest common factor, sometimes called the greatest common divisor, is the largest number that divides two or more numbers without a remainder. For example, the greatest common factor of 4 and 10 is 2 because 2 is the largest number divisible into 4 and 10 without a remainder.
IMAGE ON SCREEN- The numbers 4 and 10 are displayed. Under 4 are two lines each pointing to the number 2. Under 10 are two lines, one pointing to the number 5 and the other pointing to the number 2. Number 2 is circled under both 4 and 10.

Computing a greatest common factor is important because it allows us to reduce fractions efficiently. On paper, it may seem simple to explain the process or algorithm of finding the greatest common factor.

One way is to simply divide two numbers by all the numbers less than the smaller number until the largest number that divides into both is found. This strategy will work. However, there is a better way to reduce the number of total calculations, which increases overall efficiency.
IMAGE ON SCREEN- The title computing the greatest common factor is at the top. The numbers 4 and 10 are on screen. Underneath 4 are the following equations: 4 ÷ 2 = 1.333… marked out and 4 ÷ 2 = 2 with the 2 in the equation circled. Underneath 10 are the following equations: 10 ÷ 4 = 2.5 marked out, 10 ÷ 3 = 3.333… marked out, and 10 ÷ 2 = 5 with the 2 in the equation circled.

The Euclidean algorithm is a better way of finding the greatest common factor. It works as follows. Given two numbers, where a is the larger value and b the smaller, divide a by b and get the remainder r. Then replace a with b and replace b with r. Repeat this process until r equal 0. When the remainder equal 0, b is the greatest common factor of the two numbers.
IMAGE ON SCREEN- A graphic of a computer screen with the following text:
Find the GCF of a and b (where a > b):
Repeat until r = 0
a + b - ? r = ?
a ÷ b = ? r = ?
a ÷ b = ? r = 0

Here's an example. Find the greatest common factor of 30 and 12. First, divide the larger number 30 by the smaller number 12. The result is 2 with a remainder of 6.

Now continue and divide 12 by 6, the remainder in the previous step. 12 divided by 6 equals 2 with no remainder. Because there is no remainder, the task is complete. And the last number used to divide is the greatest common factor, which in this case is 6.

Let's do another example. Find the greatest common factor of 123 and 36. 123 divided by 36 is 3 with a remainder of 15. Continue to divide 36 by 15 because 15 was the remainder of the previous step.

36 divided by 15 equals 2 with a remainder of 6. Continue to divide 15 by 6, which equals 2 with a remainder of 3. And, finally, 6 divided by 3 equals 2 with no remainder. Therefore, our greatest common factor is 3.

TEXT ON SCREEN
Common Algorithms:…
…Finding the biggest number-- here is another type of calculation.

Given three integer numbers, called num1, num2, and num3, respectively find the largest number using an algorithm like this. Check if num1 is greater than num2. If true, check if num1 is greater than num3. If true, then show num1 is the largest number. If false, then show num3 as the largest number. If num1 is not greater than num2, check if num2 is greater than num3. If true, then show num2 to as the largest number. If false, then show num3 as the largest number. This algorithm or set of steps will always work given any three numbers.
IMAGE ON SCREEN- A graphic organizer is displayed.
The very top box: find the largest number [num1], [num2], [num3]
The next box branching down from the top box: is [num1] > [num2] ?
Branching from the above box it splits into two boxes: on the left is true and on the right is false.

Boxes underneath true:
Is [num1] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num1] is the largest number
Box underneath False: [num3 is the largest number
Boxes underneath False:
Is [num2] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num2] is the largest number
Box underneath False: [num3] is the largest number

Let's work through an example. Suppose num1 equals 6, num2 equals 10, and num3 equals 25. Now use the algorithm above to determine the largest number. Is num1 greater than num2? In other words, is 6 greater than 10? No. Therefore, the next question is, is num2 to greater than num3? In other words, is 10 greater than 25? The answer is no. So the output states num3 or 25 must be the largest number.
IMAGE ON SCREEN- A graphic organizer is displayed.
The very top box: find the largest number num = 6, num2 = 10, num3 = 25
The next box branching down from the top box: is [num1] > [num2] ? 6 > 10 ?
Branching from the above box it splits into two boxes: on the left is true and on the right is false.
Boxes underneath true:
is [num1] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num1] is the largest number
Box underneath False: [num3 is the largest number
Boxes underneath False:
is [num2] > [num3] ? 10 > 25 ?
Branching into two: True on the left or False on the right
Box underneath True: [num2] is the largest number
Box underneath False: [num3] is the largest number, 25 is the largest number

TEXT ON SCREEN
Common Algorithms:…
…Finding primes-- a prime number is a positive integer greater than 1 that can only be divided by itself and 1. The numbers 2, 3, 5, 7, et cetera are prime numbers, as they do not have any other factors.
IMAGE ON SCREEN- A chart of prime numbers is displayed. The chart includes numbers:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379,

Update to Content Accepted by SRP

383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 967, 971, 977, 983, 991, 997, 953

To test if a number is prime, check every number greater than 1 to see if the number divides the potential prime number.

One of the most widely used applications of prime numbers in computing is an encryption system. In 1978, Ron Rivest, Adi Shamir, and Leonhard Adleman combined some simple known facts about numbers to create RSA encryption, named after the inventors. The system they developed allows for the secure transmission of information, such as credit card numbers, online. The first ingredient required for the algorithm are two large prime numbers. The larger the numbers, the safer the encryption.

TEXT ON SCREEN
Common Algorithms:...
...Making change-- when making a purchase with cash, the amount of change due is calculated using another common algorithm. Find the sum of all the purchases to store in a variable named totalCost. Input the cash paid in a different variable named amountPaid. Subtract total cost from amountPaid to find the change stored in a third variable named change. change equals amount paid minus totalCost.

TEXT ON SCREEN
Common Algorithms:...
...Finding the average-- to find the arithmetic average or mean, add up a list of numbers. And then divide by the quantity of numbers summed up. Here is a set of eight numbers-- 4, 5, 1, 2, 9, 7, 10, 8. The sum of the numbers is 4 plus 5 plus 1 plus 2 plus 9 plus 7 plus 10 plus 8, which equals 46. So the average is 46 divided by 8, which comes to 5.75.

2. Programming Concepts

TEXT ON SCREEN
Problem Solving with Algorithms
Programming Concepts

Part of learning how to program is learning what tools and functions can be leveraged to solve problems and write custom algorithms. When it comes to solving problems, the first step is to fully understand the problem at hand. Without proper understanding, it's

easy to make assumptions about problems and potential solutions without really getting a clear picture of what's going on.

Once a problem has been identified, take time to consider a solution and focus on writing clear instructions one step at a time. Test the code to ensure it works as expected. Then continue to the next step.

Importance of order when using algorithms. Sequential algorithms follow a step-by-step process, and the order of those steps are crucial to ensuring the correctness of an algorithm.

Here's an algorithm for translating a word into Pig Latin.
TEXT ON SCREEN
PIG

One, append a hyphen.
TEXT ON SCREEN
PIG-

Two, append the first letter.
TEXT ON SCREEN
PIG-P

Three, append A, Y.
TEXT ON SCREEN
PIG-PAY

Four, remove the first letter.
TEXT ON SCREEN
IG-PAY

If the steps are completed out of order, the result would be different and incorrect.
TEXT ON SCREEN
PIG=IG-PAY in Pig Latin

Sequential algorithms must follow the same order of the steps to be effective.
TEXT ON SCREEN
1
2
3
4
5
...

For example, if a contact form prompts a user to enter data into a database, the program must first ask the user for the data before it can be saved to the database. Otherwise, there is no data to save.
IMAGE ON SCREEN- A graphic of website. On the website is the text Join the mailing list and there is a place to enter an email address. Then there is a button to submit.

Conditional logic. Decision-making problems involve conditional statements or branching. Conditional statements are features of programming languages that tell the computer to execute certain actions provided certain user-defined conditions are met.
IMAGE ON SCREEN- Conditional statement or branching graphic.
Begins at the word START
Branches to an IF (condition) box
The IF statement branches into false on the left or true on the right
If false it branches to an ELSE (code) statement
If true it branches to a THEN (code) statement
Both else and then statements branch into the END statement

For example, a program used to enter student grades may ask a teacher to input grades until the entire class has a grade. Once that condition is met, the program will stop asking the teacher for grades.

Conditional statements are used through the various programming languages to instruct the computer on the decision to make when given some conditions. These decisions are made if and only if the pre-stated conditions are either true or false, depending on the functions the programmer has in mind.
IMAGE ON SCREEN- Conditional statement or branching graphic.
Begins at the word START
Branches to an IF (condition) box
The IF statement branches into false on the left or true on the right
If false it branches to an ELSE (code) statement
If true it branches to a THEN (code) statement
Both else and then statements branch into the END statement

All programming languages have conditional expression syntax, although the syntax slightly differs from one programming language to the others.

Branching versus non-branching. When writing algorithms, it's often helpful to use a series of conditional statements to test complex criteria and execute appropriate actions, which is often referred to as branching. Using branching allows programmers to give precise instructions based on a variety of conditions and then output a relevant result. As opposed to branching, non-branching programming simply refers to

programs that don't use conditional statements a real world example would be furniture assembly instructions.

A program that doesn't use conditionals might resemble an input program that asks the user for data input to populate a database.

Think of an email sign-up form on a website. You enter your information and click Submit. There are not different paths or conditions that need to be accounted for.
IMAGE ON SCREEN- A graphic of a website contact us page. There are four areas for a user to fill out: Name- Jonah Torres, Email- jonahpt@email.com, Subject- Hours, and Message- What are your updated store hours?

Examples of branching programming. Conditional statements have a simple job-- check whether a condition has been met and return True if yes, and False if not. Then based on those outputs, programmers can automate other decisions or actions and so on.

Suppose a programmer is working on a program for teachers to enter student grades into a grade book. The programmer should consider how the data will be entered and saved into a database and create some conditional expressions to help make the teacher's job easier and minimize errors. To help avoid typos, the program could be written to check that values entered as grades are numeric or that the scores are no greater than a maximum value.

If a wrong number was entered, the program could display the grade in bright red or reject the input until an acceptable value is entered.

Then a variable could be created to capture whether or not all scores have been entered. It would initially be set to False. And then once all grades are entered, it would be set to True.
IMAGE ON SCREEN- Numbers are counted from 1 to 100.

This type of variable is called a Boolean variable. These only have two states: True or False.

Iteration. Iteration is another way to say repetition. Programming that uses iteration will have some kind of loop that repeats a block of code. The repetition will terminate after a predetermined number of repetitions or when a certain condition is met. If there is a process needing to be run over and over again, it's best to write the code so it can loop repeatedly, cycling through inputs or adjusting variables incrementally. This saves time programming and makes the process more efficient. Any program without a need to repeat a section of code could be considered non-iterative.

Update to Content Accepted by SRP

Using loops. Iteration is implemented using the loop in programming. The two most common types of loops are for-loop and the while-loop. The for-loop is used when a loop needs to be run a certain number of times. The other type of loop, the while-loop repeats until a specific condition is true. This is particularly useful when it's difficult or impossible to predict how many times the loop will execute.

Sequential search using loops. In computer science, linear search or sequential search is a method for finding a particular value in a list by checking each element in sequence until the desired element is found or the list is exhausted. The list doesn't have to be in order.

Here is an example. The process starts with setting up variables.
TEXT ON SCREEN
GOAL: find "6"

The first variable captures the search position and is initially set to a value of 0.
TEXT ON SCREEN
var position = 0

The search position is the current position or current value being tested.
TEXT ON SCREEN
var position = 0

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

A second Boolean variable stores one of two possible values, True or False.
TEXT ON SCREEN
var position = 0
var isFound =

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

At the start, this should be set to False.
TEXT ON SCREEN
var position = 0
var isFound = false

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

Once the condition is met, the variable will be instead set to True, and the code will stop running since the desired output has been generated.
TEXT ON SCREEN
var position = 2
var isFound = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

The code works by looking at search position 0 and testing its condition.
TEXT ON SCREEN
var position = 0
var isFound = false

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

If the condition is met, the second variable is changed from False to True and the process is over.
TEXT ON SCREEN
var position = 0
var isFound = false to true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "2" equal to "6" ?

If the condition is not met, and the value of the position variable is less than the length of the list, the code is set up to increment the position variable by 1.
TEXT ON SCREEN
var position = 0 + 1
var isFound = false

0 < 3 = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "2" equal to "6" ?

This means the code will now check the contents of the second position and repeat the process. The code terminates when found equals True or when position equals the total number of positions, which means the condition was never met and found equals False still.
TEXT ON SCREEN
var position = 2
var isFound = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "6" equal to "6" ?

The power of iterative programming. Suppose a teacher wants to give their class 10 extra points on an exam. While the teacher could manually adjust each grade and do the calculation for every student, iterative program is useful because the calculation of adding 10 to each grade is repeated.

This example demonstrates the calculation using Python but the logic for "for" and "while" loops is similar across languages.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

for score in listOfTwentyScores: with for and in in dark blue and listOfTwentyScores in light blue
newScores.append(score + 10) with newScores in light blue and score, .append and 10 in yellow

print(newScores) with print(in yellow and newScores in light blue

#Output: in green
#[95, 100, 80, 85, 93, 92, 91, 55, 88, 100, 66, 87, 55, 77, 90, 100, 80, 60, 85, 100] in green

The code starts with a variable storing an array of the grades of the 20 students in the class separated with commas. This variable is called list of 20 scores.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow

A new variable is created to hold the final values. Right now, it's blank.

IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

Then for-loop starts. In plain language, this section accesses one score at a time within the list of 20 scores variable, adds 10 to it and appends it to the new score's variable. Appending the new score allows the variable to hold multiple values and not get overwritten. This repeats until all 20 scores have been adjusted and stored in the new scores variable.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

for score in listOfTwentyScores: with for and in in dark blue and listOfTwentyScores in light blue
newScores.append(score + 10) with newScores in light blue and score, .append and 10 in yellow

print(newScores) with print(in yellow and newScores in light blue

#Output: in green
#[95, 100, 80, 85, 93, 92, 91, 55, 88, 100, 66, 87, 55, 77, 90, 100, 80, 60, 85, 100] in green

Knowing how to choose the best tool. Many problems in programming can be related back to one of the algorithms previously discussed. For example, an iterative looking program should be used if certain code can be reused, while a one-time calculation or action would use a non-iterative sequential algorithm.

Computer programmers constantly use algorithms, whether it's an existing algorithm for a common problem like sorting a list of values, or a completely new algorithm unique to the program. By understanding algorithms, programmers can make better decisions about which existing algorithms to use and learn how to make new algorithms that are correct and efficient.

CEV Copyright CEV Multimedia, LLC

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Problem Solving with Algorithims

Sequential Algorithm for Non-Branching Problems:
In computer programming and algorithm design, the term "non-branching" refers to a situation or code structure where there are no decision points or branches based on conditional statements.

- In a non-branching scenario, the code proceeds sequentially without any alternative paths based on conditions
- Sequential algorithms for non-branching problems often involve iteration, calculation or manipulation of data without the need to make choices based on conditions
- Examples:
  - summing an array
  - finding the maximum element in a list
  - reversing the order of elements in a sequence

Example: Sequential algorithm for calculating the sum of an array
Problem: Given an array of numbers, calculate the sum of all the elements
Algorithm:

```
1. Initialize a variable sum to 0.
2. For each element in the array:
   a. Add the element to the sum.
3. Display or return the sum.
```

Sequential Algorithm for Non-Iterative Problems:
Non-iterative problems refer to computational or algorithmic problems that do not require the use of iteration or repetitive structures to arrive at a solution.

- Iteration typically involves using loops or recursive functions to perform a set of instructions repeatedly until a specific condition is met
- Examples:
  - mathematical computations
    - for example: calculating the square root of a number
  - string manipulations
    - for example: checking if a string is a palindrome
  - direct conversions or computations that can be accomplished in a single pass through the data
    - for example: calculating the average of a list of numbers

Example: Algorithm to convert Celsius to Fahrenheit
Problem: Convert a temperature from Celsius to Fahrenheit
Algorithm:

```
1. Accept a temperature in Celsius as input.
2. Calculate the equivalent temperature in Fahrenheit using the
formula: Fahrenheit = (Celsius * 9/5) + 32.
3. Display or return the temperature in Fahrenheit.
```

Sequential Algorithm for Branching Control Statements:
Branching control statements, also known as conditional statements, are programming constructs that enable the execution of different sets of instructions based on certain conditions or criteria. These statements allow the flow of a program to branch into different paths, depending on whether a specified condition is true or false. The two primary branching control statements are the "if" statement and the "switch" statement.

- The "if" statement is a fundamental branching control statement that allows a program to make decisions based on a given condition
  - The "if" statement can be extended with an optional "else" clause to specify an alternative set of instructions to be executed when the condition is false
- The "switch" statement is used when there are multiple possible conditions to be checked against a single expression
  - The "switch" statement compares the expression against each case, and if a match is found, the corresponding block of code is executed. The "break" statement is used to exit the switch block

Example: Algorithm to determine if a number is even or odd
Problem: Determine if a given integer is even or odd
Algorithm:

```
1. Accept an integer as input.
2. If the input modulo 2 equals 0:
   a. Display "The number is even."
3. Else:
   a. Display "The number is odd."
```

Including a switch:

```
1. Accept an integer as input.
2. Switch on the result of the input modulo 2:
   a. Case 0:
      i. Display "The number is even."
      ii. Break.
   b. Case 1:
      i. Display "The number is odd."
      ii. Break.
   c. Default:
      i. Display "Invalid input." (optional, in case the input is not
an integer)
```

Random Numbers:

Unpredictable and are often used in various algorithms and applications

The following are examples of random numbers in different languages:
C++: 'rand()'
Python: 'random()'
JavaScript: 'Math.random()'

Loops:
Structures that repeat a specific block of code until a certain condition is met

The following are examples of loops in different languages:

Python:

```python
# Problem: Print numbers from 1 to 5

# Solution using a for loop
for i in range(1, 6):
    print(i, end=' ')

print()  # Move to the next line
```

Java:

```java
public class LoopExample {
    public static void main(String[] args) {
        // Problem: Print numbers from 1 to 5

        // Solution using a for loop
        for (int i = 1; i <= 5; ++i) {
            System.out.print(i + " ");
        }

        System.out.println(); // Move to the next line
    }
}
```

C++:

```cpp
#include <iostream>

int main() {
    // Problem: Print numbers from 1 to 5

    // Solution using a for loop
    for (int i = 1; i <= 5; ++i) {
        std::cout << i << " ";
    }

    std::cout << std::endl; // Move to the next line

    return 0;
}
```

Conditionals:
Allow the execution of different code blocks based on whether a specified condition evaluates to true or false

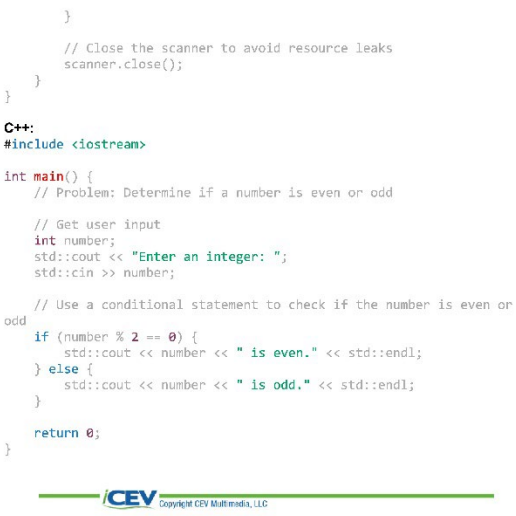
The following are examples of conditionals in different languages:

Python:

```python
# Problem: Determine if a number is even or odd

# Get user input
number = int(input("Enter an integer: "))

# Use a conditional statement to check if the number is even or odd
if number % 2 == 0:
    print(f"{number} is even.")
else:
    print(f"{number} is odd.")
```

Java:

```java
import java.util.Scanner;

public class ConditionalExample {
    public static void main(String[] args) {
        // Problem: Determine if a number is even or odd

        // Get user input
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Use a conditional statement to check if the number is even
or odd
        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
```



```
        }

        // Close the scanner to avoid resource leaks
        scanner.close();
    }
}
```

C++:

```
#include <iostream>

int main() {
    // Problem: Determine if a number is even or odd

    // Get user input
    int number;
    std::cout << "Enter an integer: ";
    std::cin >> number;

    // Use a conditional statement to check if the number is even or
odd
    if (number % 2 == 0) {
        std::cout << number << " is even." << std::endl;
    } else {
        std::cout << number << " is odd." << std::endl;
    }

    return 0;
}
```

Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(4)(R)(i), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Problem Solving with Algorithms Programming Concepts (1:42 - 2:59),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21655

Description of the specific location and hyperlink to the exact location of the proposed new content
Problem Solving with Algorithms Student Handout-Problem Solving with Algorithms,
https://files.icevonline.com/html/CEV81115_V2_HTML/CEV81115_V2_HTML_Student_Handout_-_Problem_Solving_with_Algorithms.htm
This Student Handout is found in the Problem Solving with Algorithms lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Problem Solving with Algorithms

The content within this transcript has been created utilizing a third-party software company which complies with all federal accessibility laws and international standards for web accessibility, providing a measured accuracy rate of 99.6 percent.

1. Common Algorithms

TEXT ON SCREEN
Problem Solving with Algorithms
Common Algorithms

What are algorithms? An algorithm is a set of well-defined logical steps to follow when performing a task. For example, the procedure or algorithm for boiling water would be fill a pot with water. Place the pot on a burner on the stove. Turn on the burner to high. Observe the water until it starts bubbling. When the water is bubbling, the water is boiling.

When are algorithms used? Algorithms are used in every part of computer science. They form the field's backbone. In computer science, an algorithm gives the computer a specific set of instructions, which allows the computer to do everything, be it running a calculator or running a rocket.

Computer programs are at their core algorithms written in programming languages that the computer can understand. Computer algorithms play a big role in how social media works, which posts show up, which ads are seen, and so on. These decisions are all made by algorithms.

Programmers who build search engines use algorithms to optimize searches, predict what users are going to type, and more. Much of computer programming involves using algorithms to solve problems, such as increasing the relevance of search results or making websites load faster. Programming has almost unlimited uses and applications.

Smartphones are just one example of devices utilizing programming. Although each app is programmed differently for different functions, all programming uses similar logic to process data and produce results. Another example where programming is especially useful is the field of data science and analytics. Many companies collect user data to make product decisions. These data collections are very large and need to be organized, sorted, and manipulated before analysis can begin. The programming language Python is used extensively for data analysis.

Common algorithms-- when writing an algorithm, it's often helpful to have a good understanding of some common, well-known algorithms. This helps save time because programmers can re-use or adapt the algorithm to fit their needs.

For example, to sort the items of a list, often referred to as an array, into a particular order, it is helpful to refer to sorting algorithms.
IMAGE ON SCREEN- A graphic displays four books and they are labeled A, B, C, and D, from left to right. A and D are then switched, so D is on the far left and A is on the far right. Then B and C are switched, so C is on the left in the middle and B is on the right in the middle. Therefore, the books end up in the order D, C, B, and A from left to right.

Similarly, searching algorithms are used to find and retrieve an element from wherever it is stored in a data structure.
IMAGE ON SCREEN- A computer screen is displayed with an algorithm: array [duck, duck, goose]. Then a magnifying glass is seen at the bottom by the word goose. This means the searching algorithm is looking for goose. The magnifying glass then goes up to the algorithm to find the word goose.

Finding the Greatest Common Factor, GCF. The greatest common factor, sometimes called the greatest common divisor, is the largest number that divides two or more numbers without a remainder. For example, the greatest common factor of 4 and 10 is 2 because 2 is the largest number divisible into 4 and 10 without a remainder.
IMAGE ON SCREEN- The numbers 4 and 10 are displayed. Under 4 are two lines each pointing to the number 2. Under 10 are two lines, one pointing to the number 5 and the other pointing to the number 2. Number 2 is circled under both 4 and 10.

Computing a greatest common factor is important because it allows us to reduce fractions efficiently. On paper, it may seem simple to explain the process or algorithm of finding the greatest common factor.

One way is to simply divide two numbers by all the numbers less than the smaller number until the largest number that divides into both is found. This strategy will work. However, there is a better way to reduce the number of total calculations, which increases overall efficiency.
IMAGE ON SCREEN- The title computing the greatest common factor is at the top. The numbers 4 and 10 are on screen. Underneath 4 are the following equations: 4 ÷ 2 = 1.333… marked out and 4 ÷ 2 = 2 with the 2 in the equation circled. Underneath 10 are the following equations: 10 ÷ 4 = 2.5 marked out, 10 ÷ 3 = 3.333… marked out, and 10 ÷ 2 = 5 with the 2 in the equation circled.

The Euclidean algorithm is a better way of finding the greatest common factor. It works as follows. Given two numbers, where a is the larger value and b the smaller, divide a by b and get the remainder r. Then replace a with b and replace b with r. Repeat this process until r equal 0. When the remainder equal 0, b is the greatest common factor of the two numbers.
IMAGE ON SCREEN- A graphic of a computer screen with the following text:
Find the GCF of a and b (where a > b):
Repeat until r = 0
a ÷ b - ? r = ?
a ÷ b = ? r = ?
a ÷ b = ? r = 0

Here's an example. Find the greatest common factor of 30 and 12. First, divide the larger number 30 by the smaller number 12. The result is 2 with a remainder of 6.

Now continue and divide 12 by 6, the remainder in the previous step. 12 divided by 6 equals 2 with no remainder. Because there is no remainder, the task is complete. And the last number used to divide is the greatest common factor, which in this case is 6.

Let's do another example. Find the greatest common factor of 123 and 36. 123 divided by 36 is 3 with a remainder of 15. Continue to divide 36 by 15 because 15 was the remainder of the previous step.

36 divided by 15 equals 2 with a remainder of 6. Continue to divide 15 by 6, which equals 2 with a remainder of 3. And, finally, 6 divided by 3 equals 2 with no remainder. Therefore, our greatest common factor is 3.

TEXT ON SCREEN
Common Algorithms:…
…Finding the biggest number-- here is another type of calculation.

Given three integer numbers, called num1, num2, and num3, respectively find the largest number using an algorithm like this. Check if num1 is greater than num2. If true, check if num1 is greater than num3. If true, then show num1 is the largest number. If false, then show num3 as the largest number. If num1 is not greater than num2, check if num2 is greater than num3. If true, then show num2 to as the largest number. If false, then show num3 as the largest number. This algorithm or set of steps will always work given any three numbers.
IMAGE ON SCREEN- A graphic organizer is displayed.
The very top box: find the largest number [num1], [num2], [num3]
The next box branching down from the top box: is [num1] > [num2] ?
Branching from the above box it splits into two boxes: on the left is true and on the right is false.

Boxes underneath true:
Is [num1] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num1] is the largest number
Box underneath False: [num3 is the largest number
Boxes underneath False:
Is [num2] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num2] is the largest number
Box underneath False: [num3] is the largest number

Let's work through an example. Suppose num1 equals 6, num2 equals 10, and num3 equals 25. Now use the algorithm above to determine the largest number. Is num1 greater than num2? In other words, is 6 greater than 10? No. Therefore, the next question is, is num2 to greater than num3? In other words, is 10 greater than 25? The answer is no. So the output states num3 or 25 must be the largest number.
IMAGE ON SCREEN- A graphic organizer is displayed.
The very top box: find the largest number num = 6, num2 = 10, num3 = 25
The next box branching down from the top box: is [num1] > [num2] ? 6 > 10 ?
Branching from the above box it splits into two boxes: on the left is true and on the right is false.
Boxes underneath true:
is [num1] > [num3] ?
Branching into two: True on the left or False on the right
Box underneath True: [num1] is the largest number
Box underneath False: [num3 is the largest number
Boxes underneath False:
is [num2] > [num3] ? 10 > 25 ?
Branching into two: True on the left or False on the right
Box underneath True: [num2] is the largest number
Box underneath False: [num3] is the largest number, 25 is the largest number

TEXT ON SCREEN
Common Algorithms:…
…Finding primes-- a prime number is a positive integer greater than 1 that can only be divided by itself and 1. The numbers 2, 3, 5, 7, et cetera are prime numbers, as they do not have any other factors.
IMAGE ON SCREEN- A chart of prime numbers is displayed. The chart includes numbers:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379,

Update to Content Accepted by SRP

383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 967, 971, 977, 983, 991, 997, 953

To test if a number is prime, check every number greater than 1 to see if the number divides the potential prime number.

One of the most widely used applications of prime numbers in computing is an encryption system. In 1978, Ron Rivest, Adi Shamir, and Leonhard Adleman combined some simple known facts about numbers to create RSA encryption, named after the inventors. The system they developed allows for the secure transmission of information, such as credit card numbers, online. The first ingredient required for the algorithm are two large prime numbers. The larger the numbers, the safer the encryption.

TEXT ON SCREEN
Common Algorithms:...
...Making change-- when making a purchase with cash, the amount of change due is calculated using another common algorithm. Find the sum of all the purchases to store in a variable named totalCost. Input the cash paid in a different variable named amountPaid. Subtract total cost from amountPaid to find the change stored in a third variable named change. change equals amount paid minus totalCost.

TEXT ON SCREEN
Common Algorithms:...
...Finding the average-- to find the arithmetic average or mean, add up a list of numbers. And then divide by the quantity of numbers summed up. Here is a set of eight numbers-- 4, 5, 1, 2, 9, 7, 10, 8. The sum of the numbers is 4 plus 5 plus 1 plus 2 plus 9 plus 7 plus 10 plus 8, which equals 46. So the average is 46 divided by 8, which comes to 5.75.

2. Programming Concepts

TEXT ON SCREEN
Problem Solving with Algorithms
Programming Concepts

Part of learning how to program is learning what tools and functions can be leveraged to solve problems and write custom algorithms. When it comes to solving problems, the first step is to fully understand the problem at hand. Without proper understanding, it's

easy to make assumptions about problems and potential solutions without really getting a clear picture of what's going on.

Once a problem has been identified, take time to consider a solution and focus on writing clear instructions one step at a time. Test the code to ensure it works as expected. Then continue to the next step.

Importance of order when using algorithms. Sequential algorithms follow a step-by-step process, and the order of those steps are crucial to ensuring the correctness of an algorithm.

Here's an algorithm for translating a word into Pig Latin.
TEXT ON SCREEN
PIG

One, append a hyphen.
TEXT ON SCREEN
PIG-

Two, append the first letter.
TEXT ON SCREEN
PIG-P

Three, append A, Y.
TEXT ON SCREEN
PIG-PAY

Four, remove the first letter.
TEXT ON SCREEN
IG-PAY

If the steps are completed out of order, the result would be different and incorrect.
TEXT ON SCREEN
PIG=IG-PAY in Pig Latin

Sequential algorithms must follow the same order of the steps to be effective.
TEXT ON SCREEN
1
2
3
4
5
...

For example, if a contact form prompts a user to enter data into a database, the program must first ask the user for the data before it can be saved to the database. Otherwise, there is no data to save.
IMAGE ON SCREEN- A graphic of website. On the website is the text Join the mailing list and there is a place to enter an email address. Then there is a button to submit.

Conditional logic. Decision-making problems involve conditional statements or branching. Conditional statements are features of programming languages that tell the computer to execute certain actions provided certain user-defined conditions are met.
IMAGE ON SCREEN- Conditional statement or branching graphic.
Begins at the word START
Branches to an IF (condition) box
The IF statement branches into false on the left or true on the right
If false it branches to an ELSE (code) statement
If true it branches to a THEN (code) statement
Both else and then statements branch into the END statement

For example, a program used to enter student grades may ask a teacher to input grades until the entire class has a grade. Once that condition is met, the program will stop asking the teacher for grades.

Conditional statements are used through the various programming languages to instruct the computer on the decision to make when given some conditions. These decisions are made if and only if the pre-stated conditions are either true or false, depending on the functions the programmer has in mind.
IMAGE ON SCREEN- Conditional statement or branching graphic.
Begins at the word START
Branches to an IF (condition) box
The IF statement branches into false on the left or true on the right
If false it branches to an ELSE (code) statement
If true it branches to a THEN (code) statement
Both else and then statements branch into the END statement

All programming languages have conditional expression syntax, although the syntax slightly differs from one programming language to the others.

Branching versus non-branching. When writing algorithms, it's often helpful to use a series of conditional statements to test complex criteria and execute appropriate actions, which is often referred to as branching. Using branching allows programmers to give precise instructions based on a variety of conditions and then output a relevant result. As opposed to branching, non-branching programming simply refers to

programs that don't use conditional statements a real world example would be furniture assembly instructions.

A program that doesn't use conditionals might resemble an input program that asks the user for data input to populate a database.

Think of an email sign-up form on a website. You enter your information and click Submit. There are not different paths or conditions that need to be accounted for.
IMAGE ON SCREEN- A graphic of a website contact us page. There are four areas for a user to fill out: Name- Jonah Torres, Email- jonahpt@email.com, Subject- Hours, and Message- What are your updated store hours?

Examples of branching programming. Conditional statements have a simple job-- check whether a condition has been met and return True if yes, and False if not. Then based on those outputs, programmers can automate other decisions or actions and so on.

Suppose a programmer is working on a program for teachers to enter student grades into a grade book. The programmer should consider how the data will be entered and saved into a database and create some conditional expressions to help make the teacher's job easier and minimize errors. To help avoid typos, the program could be written to check that values entered as grades are numeric or that the scores are no greater than a maximum value.

If a wrong number was entered, the program could display the grade in bright red or reject the input until an acceptable value is entered.

Then a variable could be created to capture whether or not all scores have been entered. It would initially be set to False. And then once all grades are entered, it would be set to True.
IMAGE ON SCREEN- Numbers are counted from 1 to 100.

This type of variable is called a Boolean variable. These only have two states: True or False.

Iteration. Iteration is another way to say repetition. Programming that uses iteration will have some kind of loop that repeats a block of code. The repetition will terminate after a predetermined number of repetitions or when a certain condition is met. If there is a process needing to be run over and over again, it's best to write the code so it can loop repeatedly, cycling through inputs or adjusting variables incrementally. This saves time programming and makes the process more efficient. Any program without a need to repeat a section of code could be considered non-iterative.

Update to Content Accepted by SRP

Using loops. Iteration is implemented using the loop in programming. The two most common types of loops are for-loop and the while-loop. The for-loop is used when a loop needs to be run a certain number of times. The other type of loop, the while-loop repeats until a specific condition is true. This is particularly useful when it's difficult or impossible to predict how many times the loop will execute.

Sequential search using loops. In computer science, linear search or sequential search is a method for finding a particular value in a list by checking each element in sequence until the desired element is found or the list is exhausted. The list doesn't have to be in order.

Here is an example. The process starts with setting up variables.
TEXT ON SCREEN
GOAL: find "6"

The first variable captures the search position and is initially set to a value of 0.
TEXT ON SCREEN
var position = 0

The search position is the current position or current value being tested.
TEXT ON SCREEN
var position = 0

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

A second Boolean variable stores one of two possible values, True or False.
TEXT ON SCREEN
var position = 0
var isFound =

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

At the start, this should be set to False.
TEXT ON SCREEN
var position = 0
var isFound = false

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

Once the condition is met, the variable will be instead set to True, and the code will stop running since the desired output has been generated.
TEXT ON SCREEN
var position = 2
var isFound = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

The code works by looking at search position 0 and testing its condition.
TEXT ON SCREEN
var position = 0
var isFound = false

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

If the condition is met, the second variable is changed from False to True and the process is over.
TEXT ON SCREEN
var position = 0
var isFound = false to true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "2" equal to "6" ?

If the condition is not met, and the value of the position variable is less than the length of the list, the code is set up to increment the position variable by 1.
TEXT ON SCREEN
var position = 0 + 1
var isFound = false

0 < 3 = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "2" equal to "6" ?

This means the code will now check the contents of the second position and repeat the process. The code terminates when found equals True or when position equals the total number of positions, which means the condition was never met and found equals False still.
TEXT ON SCREEN
var position = 2
var isFound = true

array [2, 4, 6, 8]
2 is in position 0, 4 is in position 1, 6 is in position 2, 8 is in position 3

is "6" equal to "6" ?

The power of iterative programming. Suppose a teacher wants to give their class 10 extra points on an exam. While the teacher could manually adjust each grade and do the calculation for every student, iterative program is useful because the calculation of adding 10 to each grade is repeated.

This example demonstrates the calculation using Python but the logic for "for" and "while" loops is similar across languages.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

for score in listOfTwentyScores: with for and in in dark blue and listOfTwentyScores in light blue
newScores.append(score + 10) with newScores in light blue and score, .append and 10 in yellow

print(newScores) with print(in yellow and newScores in light blue

#Output: in green
#[95, 100, 80, 85, 93, 92, 91, 55, 88, 100, 66, 87, 55, 77, 90, 100, 80, 60, 85, 100] in green

The code starts with a variable storing an array of the grades of the 20 students in the class separated with commas. This variable is called list of 20 scores.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow

A new variable is created to hold the final values. Right now, it's blank.

IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

Then for-loop starts. In plain language, this section accesses one score at a time within the list of 20 scores variable, adds 10 to it and appends it to the new score's variable. Appending the new score allows the variable to hold multiple values and not get overwritten. This repeats until all 20 scores have been adjusted and stored in the new scores variable.
IMAGE ON SCREEN- Coding is shown on screen as follows:
listOfTwentyScores = [85, 90, 70, 75, 83, 82, 81, 45, 78, 90, 56, 77, 45, 67, 80, 90, 70, 50, 75, 90] with listOfTwentyScores in light blue and all numbers in yellow
newScores = [ ] with newScores in light blue

for score in listOfTwentyScores: with for and in in dark blue and listOfTwentyScores in light blue
newScores.append(score + 10) with newScores in light blue and score, .append and 10 in yellow

print(newScores) with print(in yellow and newScores in light blue

#Output: in green
#[95, 100, 80, 85, 93, 92, 91, 55, 88, 100, 66, 87, 55, 77, 90, 100, 80, 60, 85, 100] in green

Knowing how to choose the best tool. Many problems in programming can be related back to one of the algorithms previously discussed. For example, an iterative looking program should be used if certain code can be reused, while a one-time calculation or action would use a non-iterative sequential algorithm.

Computer programmers constantly use algorithms, whether it's an existing algorithm for a common problem like sorting a list of values, or a completely new algorithm unique to the program. By understanding algorithms, programmers can make better decisions about which existing algorithms to use and learn how to make new algorithms that are correct and efficient.

CEV Copyright CEV Multimedia, LLC

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Problem Solving with Algorithims

Sequential Algorithm for Non-Branching Problems:
In computer programming and algorithm design, the term "non-branching" refers to a situation or code structure where there are no decision points or branches based on conditional statements.

- In a non-branching scenario, the code proceeds sequentially without any alternative paths based on conditions
- Sequential algorithms for non-branching problems often involve iteration, calculation or manipulation of data without the need to make choices based on conditions
- Examples:
  - summing an array
  - finding the maximum element in a list
  - reversing the order of elements in a sequence

Example: Sequential algorithm for calculating the sum of an array
Problem: Given an array of numbers, calculate the sum of all the elements
Algorithm:

```
1. Initialize a variable sum to 0.
2. For each element in the array:
   a. Add the element to the sum.
3. Display or return the sum.
```

Sequential Algorithm for Non-Iterative Problems:
Non-iterative problems refer to computational or algorithmic problems that do not require the use of iteration or repetitive structures to arrive at a solution.

- Iteration typically involves using loops or recursive functions to perform a set of instructions repeatedly until a specific condition is met
- Examples:
  - mathematical computations
    - for example: calculating the square root of a number
  - string manipulations
    - for example: checking if a string is a palindrome
  - direct conversions or computations that can be accomplished in a single pass through the data
    - for example: calculating the average of a list of numbers

Example: Algorithm to convert Celsius to Fahrenheit
Problem: Convert a temperature from Celsius to Fahrenheit
Algorithm:

```
1. Accept a temperature in Celsius as input.
2. Calculate the equivalent temperature in Fahrenheit using the
formula: Fahrenheit = (Celsius * 9/5) + 32.
3. Display or return the temperature in Fahrenheit.
```

Sequential Algorithm for Branching Control Statements:
Branching control statements, also known as conditional statements, are programming constructs that enable the execution of different sets of instructions based on certain conditions or criteria. These statements allow the flow of a program to branch into different paths, depending on whether a specified condition is true or false. The two primary branching control statements are the "if" statement and the "switch" statement.

- The "if" statement is a fundamental branching control statement that allows a program to make decisions based on a given condition
  - The "if" statement can be extended with an optional "else" clause to specify an alternative set of instructions to be executed when the condition is false
- The "switch" statement is used when there are multiple possible conditions to be checked against a single expression
  - The "switch" statement compares the expression against each case, and if a match is found, the corresponding block of code is executed. The "break" statement is used to exit the switch block

Example: Algorithm to determine if a number is even or odd
Problem: Determine if a given integer is even or odd
Algorithm:

```
1. Accept an integer as input.
2. If the input modulo 2 equals 0:
   a. Display "The number is even."
3. Else:
   a. Display "The number is odd."
```

Including a switch:

```
1. Accept an integer as input.
2. Switch on the result of the input modulo 2:
   a. Case 0:
      i. Display "The number is even."
      ii. Break.
   b. Case 1:
      i. Display "The number is odd."
      ii. Break.
   c. Default:
      i. Display "Invalid input." (optional, in case the input is not
an integer)
```

Random Numbers:

Unpredictable and are often used in various algorithms and applications

The following are examples of random numbers in different languages:
C++: 'rand()'
Python: 'random()'
JavaScript: 'Math.random()'

Loops:
Structures that repeat a specific block of code until a certain condition is met

The following are examples of loops in different languages:

Python:

```python
# Problem: Print numbers from 1 to 5

# Solution using a for loop
for i in range(1, 6):
    print(i, end=' ')

print()  # Move to the next line
```

Java:

```java
public class LoopExample {
    public static void main(String[] args) {
        // Problem: Print numbers from 1 to 5

        // Solution using a for loop
        for (int i = 1; i <= 5; ++i) {
            System.out.print(i + " ");
        }

        System.out.println(); // Move to the next line
    }
}
```

C++:

```cpp
#include <iostream>

int main() {
    // Problem: Print numbers from 1 to 5

    // Solution using a for loop
    for (int i = 1; i <= 5; ++i) {
        std::cout << i << " ";
    }

    std::cout << std::endl; // Move to the next line

    return 0;
}
```

Conditionals:
Allow the execution of different code blocks based on whether a specified condition evaluates to true or false

The following are examples of conditionals in different languages:

Python:

```python
# Problem: Determine if a number is even or odd

# Get user input
number = int(input("Enter an integer: "))

# Use a conditional statement to check if the number is even or odd
if number % 2 == 0:
    print(f"{number} is even.")
else:
    print(f"{number} is odd.")
```

Java:

```java
import java.util.Scanner;

public class ConditionalExample {
    public static void main(String[] args) {
        // Problem: Determine if a number is even or odd

        // Get user input
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Use a conditional statement to check if the number is even
or odd
        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
```

Update to Content Accepted by SRP

```
        }

        // Close the scanner to avoid resource leaks
        scanner.close();
    }
}
```

C++:

```
#include <iostream>

int main() {
    // Problem: Determine if a number is even or odd

    // Get user input
    int number;
    std::cout << "Enter an integer: ";
    std::cin >> number;

    // Use a conditional statement to check if the number is even or
odd
    if (number % 2 == 0) {
        std::cout << number << " is even." << std::endl;
    } else {
        std::cout << number << " is odd." << std::endl;
    }

    return 0;
}
```

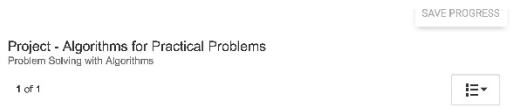
(SE)(Breakout(s)) and (Citation Type(s))
(4)(S)(ii), Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Project - Algorithms for Practical Problems,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21655/CEV81115_Project01

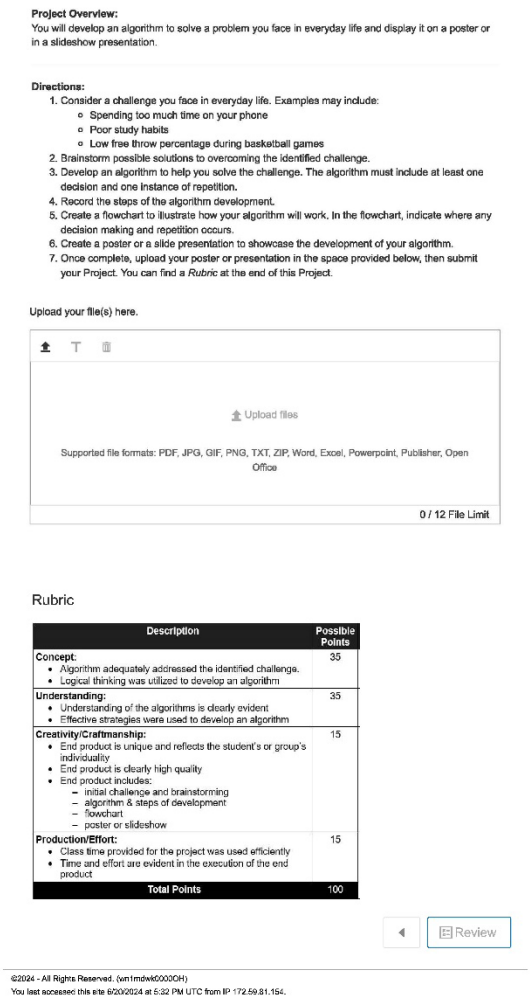
Description of the specific location and hyperlink to the exact location of the proposed new content
Coding Challenge: Find Open Days at Time (Debug),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22043/CEV71806_SIM01
Access to the interactive coding environment can be located beneath the Interactive Assignments heading by clicking the link to the Coding Challenge. Once clicked, the link will take you to a page prompting you to click Start. Select Start to view the Coding Challenge in the interactive environment.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



Update to Content Accepted by SRP



Project Overview:
You will develop an algorithm to solve a problem you face in everyday life and display it on a poster or in a slideshow presentation.

Directions:

1. Consider a challenge you face in everyday life. Examples may include:
   - Spending too much time on your phone
   - Poor study habits
   - Low free throw percentage during basketball games
2. Brainstorm possible solutions to overcoming the identified challenge.
3. Develop an algorithm to help you solve the challenge. The algorithm must include at least one decision and one instance of repetition.
4. Record the steps of the algorithm development.
5. Create a flowchart to illustrate how your algorithm will work. In the flowchart, indicate where any decision making and repetition occurs.
6. Create a poster or a slide presentation to showcase the development of your algorithm.
7. Once complete, upload your poster or presentation in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Concept: • Algorithm adequately addressed the identified challenge. • Logical thinking was utilized to develop an algorithm	35
Understanding: • Understanding of the algorithms is clearly evident • Effective strategies were used to develop an algorithm	35
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality • End product includes: – initial challenge and brainstorming – algorithm & steps of development – flowchart – poster or slideshow	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

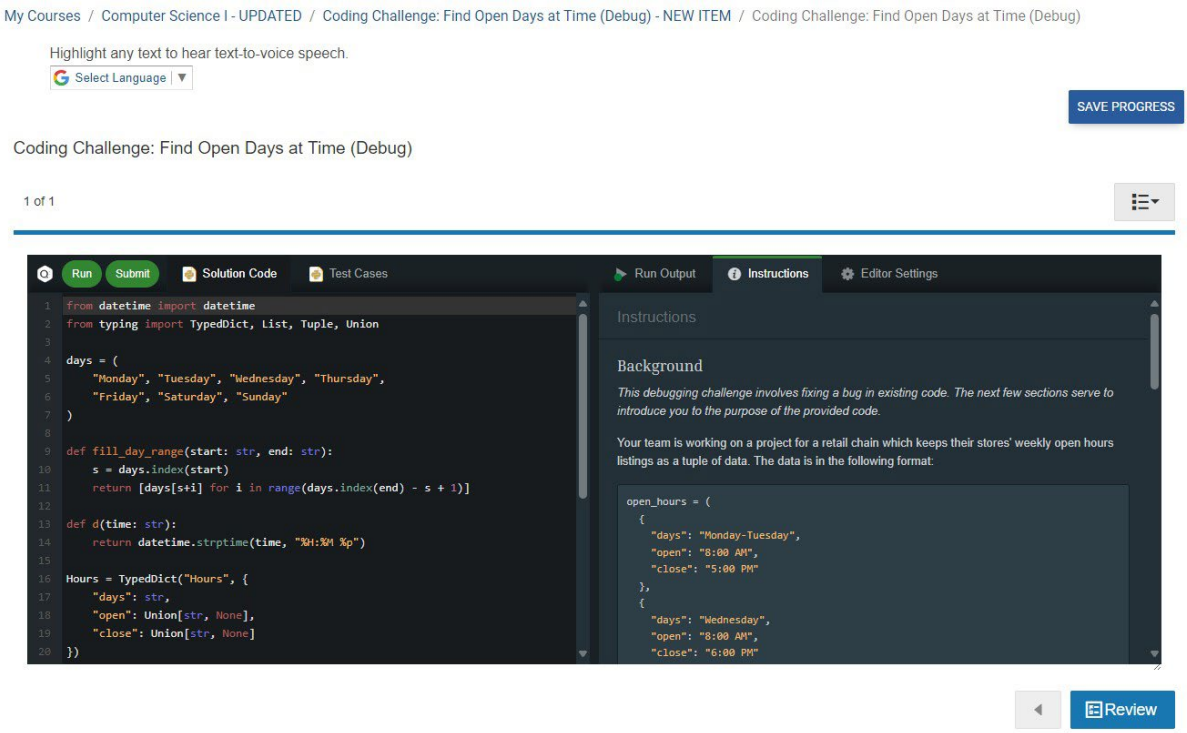
Review

©2024 - All Rights Reserved. (wn1mdwk0003OH)
You last accessed this site 6/20/2024 at 5:22 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP



(SE)(Breakout(s)) and (Citation Type(s))
(4)(V)(i), Activity

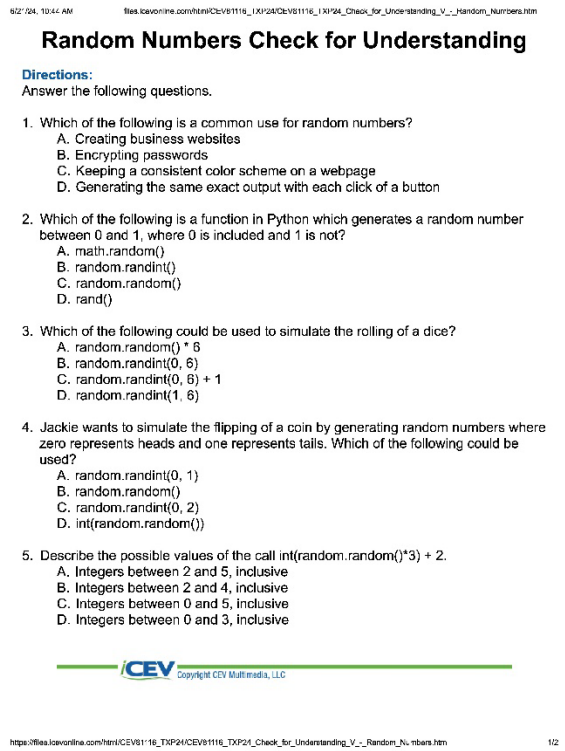
Description of the specific location and hyperlink to the exact location of currently adopted content
Check for Understanding V - Random Numbers #2-4,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21656/CEV81116_Assess05

Description of the specific location and hyperlink to the exact location of the proposed new content
Problem Solving with Functions Activity-Generating and Using Random Numbers,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22280/CEV81116_V2_Activity05
This Activity is found in the Problem Solving with Functions lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP



6/27/24, 10:44 AM files.icevonline.com/html/CEV81116_TXP24/CEV81116_TXP24_Check_for_Understanding_V_-_Random_Numbers.htm

Random Numbers Check for Understanding

Directions:
Answer the following questions.

1. Which of the following is a common use for random numbers?
   A. Creating business websites
   B. Encrypting passwords
   C. Keeping a consistent color scheme on a webpage
   D. Generating the same exact output with each click of a button

2. Which of the following is a function in Python which generates a random number between 0 and 1, where 0 is included and 1 is not?
   A. math.random()
   B. random.randint()
   C. random.random()
   D. rand()

3. Which of the following could be used to simulate the rolling of a dice?
   A. random.random() * 6
   B. random.randint(0, 6)
   C. random.randint(0, 6) + 1
   D. random.randint(1, 6)

4. Jackie wants to simulate the flipping of a coin by generating random numbers where zero represents heads and one represents tails. Which of the following could be used?
   A. random.randint(0, 1)
   B. random.random()
   C. random.randint(0, 2)
   D. int(random.random())

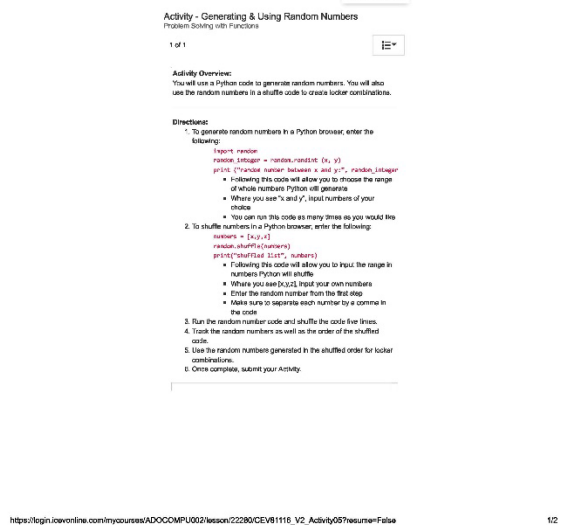
5. Describe the possible values of the call int(random.random()*3) + 2.
   A. Integers between 2 and 5, inclusive
   B. Integers between 2 and 4, inclusive
   C. Integers between 0 and 5, inclusive
   D. Integers between 0 and 3, inclusive

Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV81116_TXP24/CEV81116_TXP24_Check_for_Understanding_V_-_Random_Numbers.htm 1/2

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.



Activity - Generating & Using Random Numbers
Problem Solving with Functions

1 of 1

Activity Overview:
You will use a Python code to generate random numbers. You will also use the random numbers in a shuffle code to create locker combinations.

Directions:
1. To generate random numbers in a Python browser, enter the following:

```
import random
random_integer = random.randint (x, y)
print ("random number between x and y:", random_integer
```

   - Following this code will allow you to choose the range of whole numbers Python will generate
   - Where you see "x and y", input numbers of your choice
   - You can run this code as many times as you would like
2. To shuffle numbers in a Python browser, enter the following:

```
numbers = [x,y,z]
random.shuffle(numbers)
print("shuffled list", numbers)
```

   - Following this code will allow you to input the range in numbers Python will shuffle
   - Where you see [x,y,z], input your own numbers
   - Enter the random number from the first step
   - Make sure to separate each number by a comma in the code
3. Run the random number code and shuffle the code five times.
4. Track the random numbers as well as the order of the shuffled code.
5. Use the random numbers generated in the shuffled order for locker combinations.
6. Once complete, submit your Activity.

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22280/CEV81116_V2_Activity05?resume=False 1/2

(SE)(Breakout(s)) and (Citation Type(s))
(4)(V)(ii), Activity

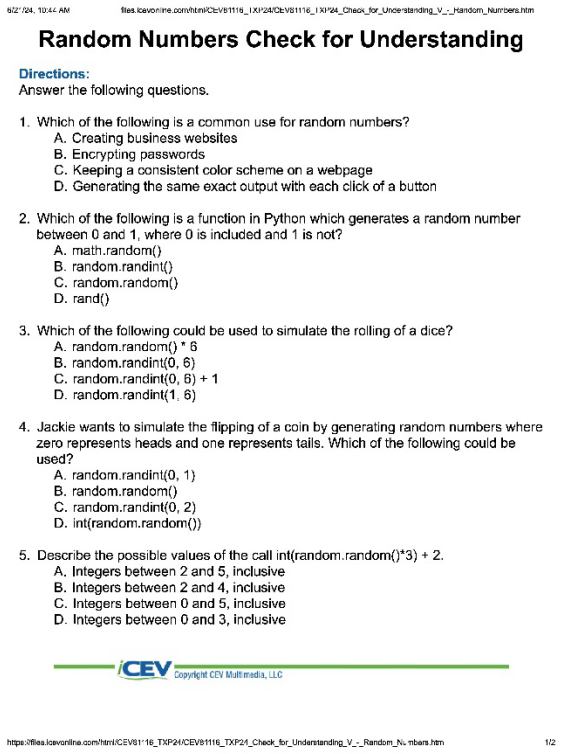
Description of the specific location and hyperlink to the exact location of currently adopted content
Check for Understanding V - Random Numbers #1, 5,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21656/CEV81116_Assess05

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of the proposed new content
Problem Solving with Functions Activity-Generating and Using Random Numbers,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22280/CEV81116_V2_Activity05
This Activity is found in the Problem Solving with Functions lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



6/27/24, 10:44 AM files.icevonline.com/html/CEV81116_TXP24/CEV81116_TXP24_Check_for_Understanding_V_-_Random_Numbers.htm

Random Numbers Check for Understanding

Directions:
Answer the following questions.

1. Which of the following is a common use for random numbers?
   A. Creating business websites
   B. Encrypting passwords
   C. Keeping a consistent color scheme on a webpage
   D. Generating the same exact output with each click of a button

2. Which of the following is a function in Python which generates a random number between 0 and 1, where 0 is included and 1 is not?
   A. math.random()
   B. random.randint()
   C. random.random()
   D. rand()

3. Which of the following could be used to simulate the rolling of a dice?
   A. random.random() * 6
   B. random.randint(0, 6)
   C. random.randint(0, 6) + 1
   D. random.randint(1, 6)

4. Jackie wants to simulate the flipping of a coin by generating random numbers where zero represents heads and one represents tails. Which of the following could be used?
   A. random.randint(0, 1)
   B. random.random()
   C. random.randint(0, 2)
   D. int(random.random())

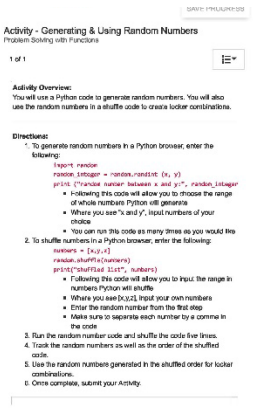
5. Describe the possible values of the call int(random.random()*3) + 2.
   A. Integers between 2 and 5, inclusive
   B. Integers between 2 and 4, inclusive
   C. Integers between 0 and 5, inclusive
   D. Integers between 0 and 3, inclusive

Copyright CEV Multimedia, LLC

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP



Activity - Generating & Using Random Numbers

Problem Solving with Functions

1 of 1

Activity Overview:

You will use a Python code to generate random numbers. You will also use the random numbers in a shuffle code to create locker combinations.

Directions:

1. To generate random numbers in a Python browser, enter the following:

```
import random
random_integer = random.randint (x, y)
print ("random number between x and y:", random_integer
```

   - Following this code will allow you to choose the range of whole numbers Python will generate
   - Where you see "x and y", input numbers of your choice
   - You can run this code as many times as you would like

2. To shuffle numbers in a Python browser, enter the following:

```
numbers = [x,y,z]
random.shuffle(numbers)
print("shuffled list", numbers)
```

   - Following this code will allow you to input the range in numbers Python will shuffle
   - Where you see [x,y,z], input your own numbers
   - Enter the random number from the first step
   - Make sure to separate each number by a comma in the code

3. Run the random number code and shuffle the code five times.
4. Track the random numbers as well as the order of the shuffled code.
5. Use the random numbers generated in the shuffled order for locker combinations.
6. Once complete, submit your Activity.

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22280/CEV81116_V2_Activity05?resume=False 1/2

(SE)(Breakout(s)) and (Citation Type(s))

(5)(A)(i), Activity

Description of the specific location and hyperlink to the exact location of currently adopted content

Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645/CEV71512_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content

Intellectual Property and Software Development Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300/CEV71512_V2_Activity01
This Activity is found in the Intellectual Property & Software Development lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

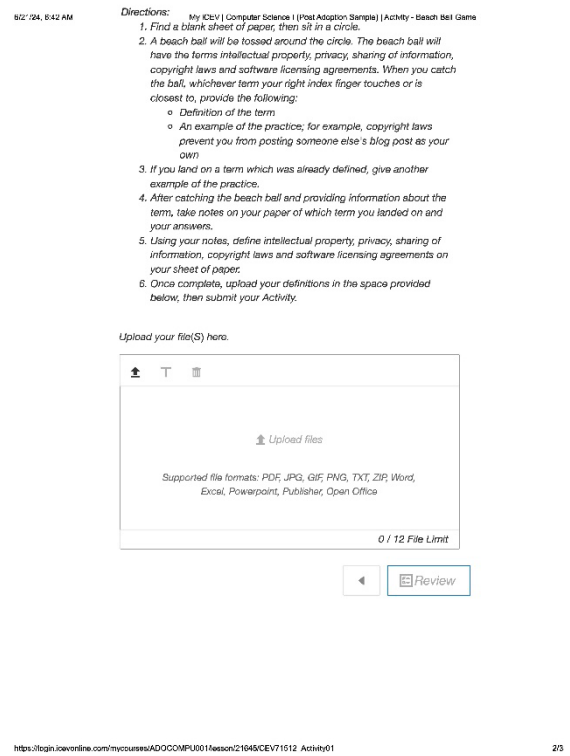
Insert a screenshot of your currently adopted content.

Activity - Beach Ball Game

Intellectual Property & Software Development

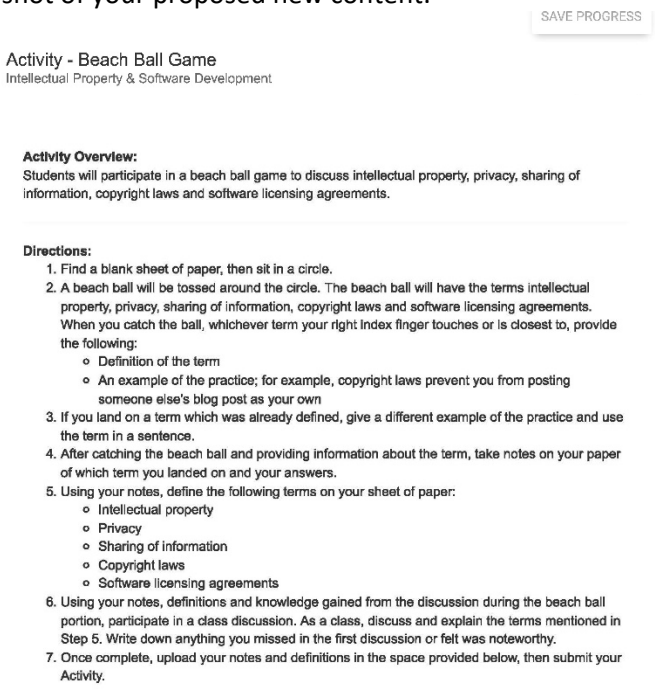
1 of 1

Update to Content Accepted by SRP



Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.



(SE)(Breakout(s)) and (Citation Type(s))

Update to Content Accepted by SRP

(5)(A)(ii), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Intellectual Property and Software Development (Slides 11-15),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645

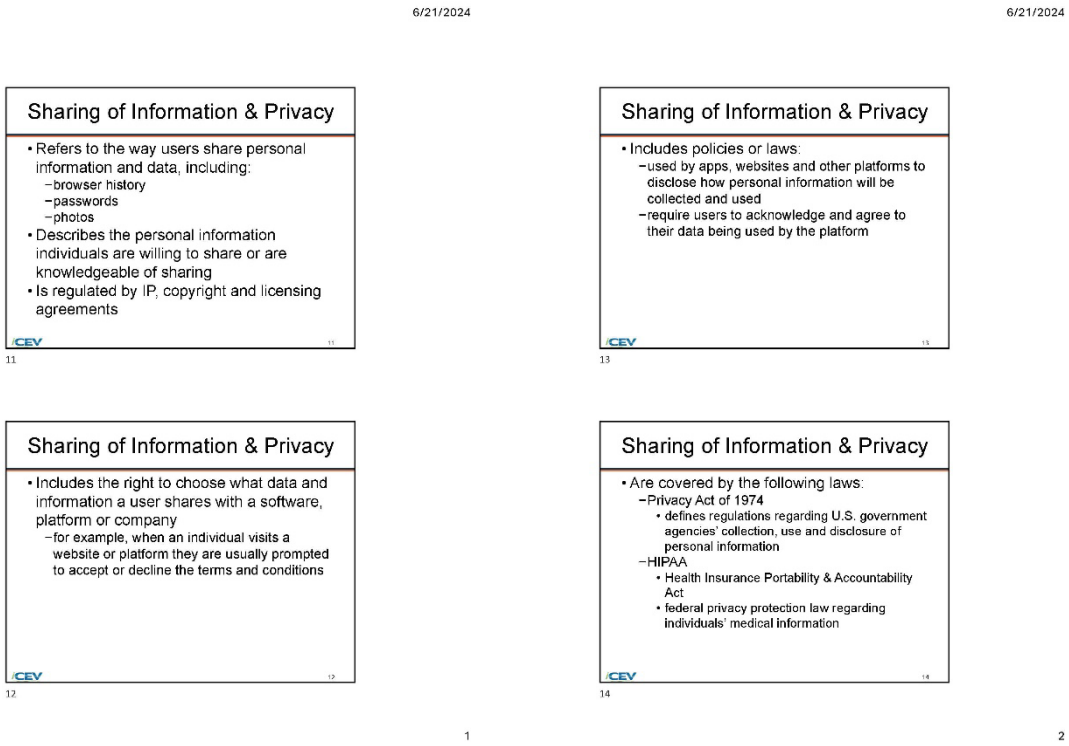
Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645/CEV71512_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Intellectual Property and Software Development (Slides 12-18),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300
In the Intellectual Property & Software Development PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

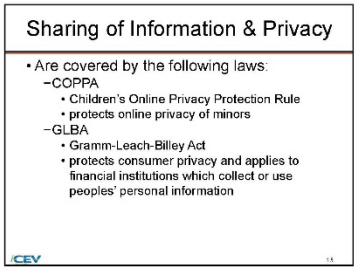
Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300/CEV71512_V2_Activity01
This Activity is found in the Intellectual Property & Software Development lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

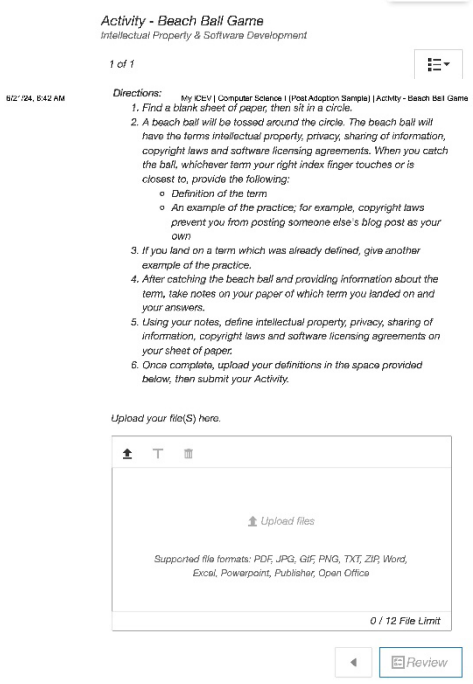
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



Update to Content Accepted by SRP





Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Sharing of Information & Privacy

- Includes policies or laws:
  - used by apps, websites and other platforms to disclose how personal information will be collected and used
  - require users to acknowledge and agree to their data being used by the platform

16

Sharing of Information & Privacy

- Refers to the way users share personal information and data, including:
  - browser history
  - passwords
  - photos
- Describes the personal information individuals are willing to share or are knowledgeable of sharing
- Is regulated by IP, copyright and licensing agreements

14

Sharing of Information & Privacy Discussion

What are examples of proper and improper sharing of information or privacy?

17

Sharing of Information & Privacy

- Includes the right to choose what data and information a user shares with a software, platform or company
  - for example, when an individual visits a website or platform they are usually prompted to accept or decline the terms and conditions

15

Software Licensing Agreements

- Are legal contracts between a licensor and purchaser of software
- Are necessary when using copyrighted software
- Protects IP and copyrights by:
  - preventing misuse of the software
  - reducing the liability from the software
  - securing ownership rights of the software
  - guaranteeing correct timing provisions for use of the software

12

Privacy

- In terms of intellectual property and software development addresses the proper storage, access, retention and security of data
- Is typically associated with the proper handling of personal data or identifiable information

18

Software Licensing Agreements Discussion

Why are software licensing agreements necessary?

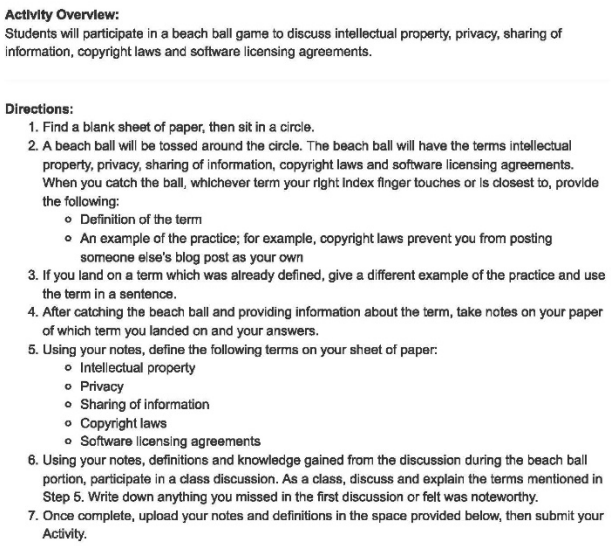
13

SAVE PROGRESS

Activity - Beach Ball Game

Intellectual Property & Software Development

Update to Content Accepted by SRP



Activity Overview:
Students will participate in a beach ball game to discuss intellectual property, privacy, sharing of information, copyright laws and software licensing agreements.

Directions:

1. Find a blank sheet of paper, then sit in a circle.
2. A beach ball will be tossed around the circle. The beach ball will have the terms intellectual property, privacy, sharing of information, copyright laws and software licensing agreements. When you catch the ball, whichever term your right index finger touches or is closest to, provide the following:
    - Definition of the term
    - An example of the practice; for example, copyright laws prevent you from posting someone else's blog post as your own
3. If you land on a term which was already defined, give a different example of the practice and use the term in a sentence.
4. After catching the beach ball and providing information about the term, take notes on your paper of which term you landed on and your answers.
5. Using your notes, define the following terms on your sheet of paper:
    - Intellectual property
    - Privacy
    - Sharing of information
    - Copyright laws
    - Software licensing agreements
6. Using your notes, definitions and knowledge gained from the discussion during the beach ball portion, participate in a class discussion. As a class, discuss and explain the terms mentioned in Step 5. Write down anything you missed in the first discussion or felt was noteworthy.
7. Once complete, upload your notes and definitions in the space provided below, then submit your Activity.

(SE)(Breakout(s)) and (Citation Type(s))
(5)(A)(iii), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Intellectual Property and Software Development (Slides 11-15),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645

Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645/CEV71512_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Intellectual Property and Software Development (Slides 12-18),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300
In the Intellectual Property & Software Development PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300/CEV71512_V2_Activity01
This Activity is found in the Intellectual Property & Software Development lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Update to Content Accepted by SRP

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Sharing of Information & Privacy

- Refers to the way users share personal information and data, including:
  - browser history
  - passwords
  - photos
- Describes the personal information individuals are willing to share or are knowledgeable of sharing
- Is regulated by IP, copyright and licensing agreements

11

Sharing of Information & Privacy

- Includes the right to choose what data and information a user shares with a software, platform or company
  - for example, when an individual visits a website or platform they are usually prompted to accept or decline the terms and conditions

12

Sharing of Information & Privacy

- Includes policies or laws:
  - used by apps, websites and other platforms to disclose how personal information will be collected and used
  - require users to acknowledge and agree to their data being used by the platform

13

Sharing of Information & Privacy

- Are covered by the following laws:
  - Privacy Act of 1974
    - defines regulations regarding U.S. government agencies' collection, use and disclosure of personal information
  - HIPAA
    - Health Insurance Portability & Accountability Act
    - federal privacy protection law regarding individuals' medical information

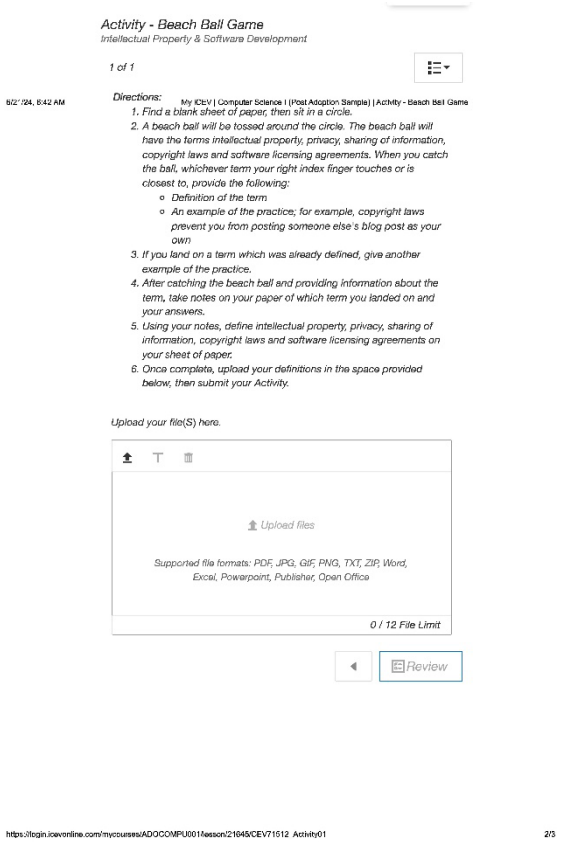
14

Sharing of Information & Privacy

- Are covered by the following laws:
  - COPPA
    - Children's Online Privacy Protection Rule
    - protects online privacy of minors
  - GLBA
    - Gramm-Leach-Bliley Act
    - protects consumer privacy and applies to financial institutions which collect or use peoples' personal information

15

Update to Content Accepted by SRP



Activity - Beach Ball Game
Intellectual Property & Software Development

1 of 1

8/2/24, 8:42 AM My ICEV | Computer Science I (Post Adoption Sample) | Activity - Beach Ball Game

Directions:

1. Find a blank sheet of paper, then sit in a circle.
2. A beach ball will be tossed around the circle. The beach ball will have the terms intellectual property, privacy, sharing of information, copyright laws and software licensing agreements. When you catch the ball, whichever term your right index finger touches or is closest to, provide the following:
   - Definition of the term
   - An example of the practice; for example, copyright laws prevent you from posting someone else's blog post as your own
3. If you land on a term which was already defined, give another example of the practice.
4. After catching the beach ball and providing information about the term, take notes on your paper of which term you landed on and your answers.
5. Using your notes, define intellectual property, privacy, sharing of information, copyright laws and software licensing agreements on your sheet of paper.
6. Once complete, upload your definitions in the space provided below, then submit your Activity.

Upload your file(S) here.

Upload files

Supported file formats: PDF, JPG, Gif, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Review

https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21045/CEV71512_Activity01 2/3

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Sharing of Information & Privacy

- Includes policies or laws:
  - used by apps, websites and other platforms to disclose how personal information will be collected and used
  - require users to acknowledge and agree to their data being used by the platform

16

Sharing of Information & Privacy

- Refers to the way users share personal information and data, including:
  - browser history
  - passwords
  - photos
- Describes the personal information individuals are willing to share or are knowledgeable of sharing
- Is regulated by IP, copyright and licensing agreements

14

Sharing of Information & Privacy Discussion

What are examples of proper and improper sharing of information or privacy?

17

Sharing of Information & Privacy

- Includes the right to choose what data and information a user shares with a software, platform or company
  - for example, when an individual visits a website or platform they are usually prompted to accept or decline the terms and conditions

15

Software Licensing Agreements

- Are legal contracts between a licensor and purchaser of software
- Are necessary when using copyrighted software
- Protects IP and copyrights by:
  - preventing misuse of the software
  - reducing the liability from the software
  - securing ownership rights of the software
  - guaranteeing correct timing provisions for use of the software

12

Privacy

- In terms of intellectual property and software development addresses the proper storage, access, retention and security of data
- Is typically associated with the proper handling of personal data or identifiable information

18

Software Licensing Agreements Discussion

Why are software licensing agreements necessary?

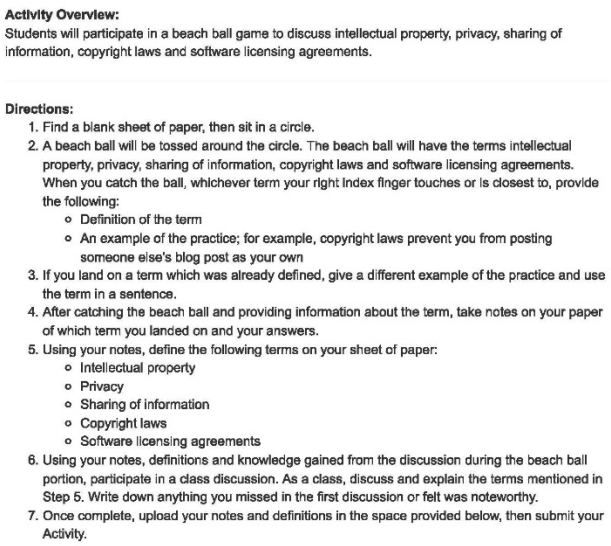
13

SAVE PROGRESS

Activity - Beach Ball Game

Intellectual Property & Software Development

Update to Content Accepted by SRP



Activity Overview:
Students will participate in a beach ball game to discuss intellectual property, privacy, sharing of information, copyright laws and software licensing agreements.

Directions:

1. Find a blank sheet of paper, then sit in a circle.
2. A beach ball will be tossed around the circle. The beach ball will have the terms intellectual property, privacy, sharing of information, copyright laws and software licensing agreements. When you catch the ball, whichever term your right index finger touches or is closest to, provide the following:
   - Definition of the term
   - An example of the practice; for example, copyright laws prevent you from posting someone else's blog post as your own
3. If you land on a term which was already defined, give a different example of the practice and use the term in a sentence.
4. After catching the beach ball and providing information about the term, take notes on your paper of which term you landed on and your answers.
5. Using your notes, define the following terms on your sheet of paper:
   - Intellectual property
   - Privacy
   - Sharing of information
   - Copyright laws
   - Software licensing agreements
6. Using your notes, definitions and knowledge gained from the discussion during the beach ball portion, participate in a class discussion. As a class, discuss and explain the terms mentioned in Step 5. Write down anything you missed in the first discussion or felt was noteworthy.
7. Once complete, upload your notes and definitions in the space provided below, then submit your Activity.

(SE)(Breakout(s)) and (Citation Type(s))
(5)(A)(iv), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Intellectual Property and Software Development (Slides 9-15),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645

Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645/CEV71512_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Intellectual Property and Software Development (Slides 6-11),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300
In the Intellectual Property & Software Development PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300/CEV71512_V2_Activity01
This Activity is found in the Intellectual Property & Software Development lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Copyright

- Is a legally binding type of IP which protects original works of authorship
  - authorship includes writing but also includes
    - coding
    - blogging
    - programming
- Was first defined by the Copyright Act of 1790
- Is usually owned by the creator of the IP but might be sold to another party

9

Software Licensing Agreements

- Are legal contracts between a licensor and purchaser of a software
- Are necessary when using a copyrighted software
- Protects IP and copyrights by:
  - preventing misuse of the software
  - reducing the liability from the software
  - securing ownership rights of the software
  - guaranteeing correct timing provisions for use of the software

10

Sharing of Information & Privacy

- Refers to the way users share personal information and data, including:
  - browser history
  - passwords
  - photos
- Describes the personal information individuals are willing to share or are knowledgeable of sharing
- Is regulated by IP, copyright and licensing agreements

11

Sharing of Information & Privacy

- Includes the right to choose what data and information a user shares with a software, platform or company
  - for example, when an individual visits a website or platform they are usually prompted to accept or decline the terms and conditions

12

Sharing of Information & Privacy

- Includes policies or laws:
  - used by apps, websites and other platforms to disclose how personal information will be collected and used
  - require users to acknowledge and agree to their data being used by the platform

13

Sharing of Information & Privacy

- Are covered by the following laws:
  - Privacy Act of 1974
    - defines regulations regarding U.S. government agencies' collection, use and disclosure of personal information
  - HIPAA
    - Health Insurance Portability & Accountability Act
    - federal privacy protection law regarding individuals' medical information

14

Sharing of Information & Privacy

- Are covered by the following laws:
  - COPPA
    - Children's Online Privacy Protection Rule
    - protects online privacy of minors
  - GLBA
    - Gramm-Leach-Bliley Act
    - protects consumer privacy and applies to financial institutions which collect or use peoples' personal information

15

Activity - Beach Ball Game
Intellectual Property & Software Development

1 of 1

Update to Content Accepted by SRP

Directions:

1. *Find a blank sheet of paper, then sit in a circle.*
2. *A beach ball will be tossed around the circle. The beach ball will have the terms intellectual property, privacy, sharing of information, copyright laws and software licensing agreements. When you catch the ball, whichever term your right index finger touches or is closest to, provide the following:*
   - *Definition of the term*
   - *An example of the practice; for example, copyright laws prevent you from posting someone else's blog post as your own*
3. *If you land on a term which was already defined, give another example of the practice.*
4. *After catching the beach ball and providing information about the term, take notes on your paper of which term you landed on and your answers.*
5. *Using your notes, define intellectual property, privacy, sharing of information, copyright laws and software licensing agreements on your sheet of paper.*
6. *Once complete, upload your definitions in the space provided below, then submit your Activity.*

Upload your file(S) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Review

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Ethics

- Is the idea individuals should try to do the right thing, morally, professionally and legally
- Can refer to computer science standards, including:
  - not using technology to harm others
  - not looking at others' data without permission
  - not taking information without permission
  - not contributing to or spreading misinformation

CEV 6

6

Ethics

- Can refer to computer science standards, including:
  - not purchasing pirated or copy software
  - not using others' resources unless authorized to do so
  - not claiming ownership of someone else's ideas

CEV 7

7

Intellectual Property

- Refers to original ideas, creative works or inventions protected by law
- Is an intangible form of property
- Includes software
- Is often shortened to "IP"

CEV 8

8

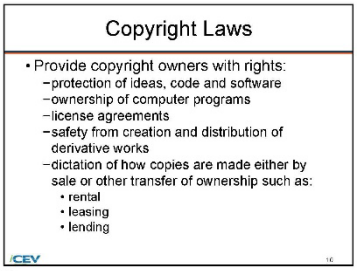
Copyright

- Is a legally binding type of IP which protects original works of authorship
  - authorship includes writing but also includes
    - coding
    - blogging
    - programming
- Was first defined by the Copyright Act of 1790
- Is usually owned by the creator of the IP but might be sold to another party

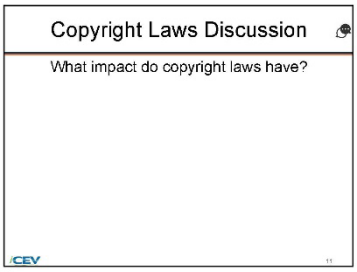
CEV 9

9

Update to Content Accepted by SRP



10



11

SAVE PROGRESS

Activity - Beach Ball Game

Intellectual Property & Software Development

Activity Overview:

Students will participate in a beach ball game to discuss intellectual property, privacy, sharing of information, copyright laws and software licensing agreements.

Directions:

1. Find a blank sheet of paper, then sit in a circle.
2. A beach ball will be tossed around the circle. The beach ball will have the terms intellectual property, privacy, sharing of information, copyright laws and software licensing agreements. When you catch the ball, whichever term your right index finger touches or is closest to, provide the following:
   - Definition of the term
   - An example of the practice; for example, copyright laws prevent you from posting someone else's blog post as your own
3. If you land on a term which was already defined, give a different example of the practice and use the term in a sentence.
4. After catching the beach ball and providing information about the term, take notes on your paper of which term you landed on and your answers.
5. Using your notes, define the following terms on your sheet of paper:
   - Intellectual property
   - Privacy
   - Sharing of information
   - Copyright laws
   - Software licensing agreements
6. Using your notes, definitions and knowledge gained from the discussion during the beach ball portion, participate in a class discussion. As a class, discuss and explain the terms mentioned in Step 5. Write down anything you missed in the first discussion or felt was noteworthy.
7. Once complete, upload your notes and definitions in the space provided below, then submit your Activity.

(SE)(Breakout(s)) and (Citation Type(s))

(5)(A)(v), Narrative & Activity

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Intellectual Property and Software Development (Slide 10),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645

Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645/CEV71512_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Intellectual Property and Software Development (Slides 6-11),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300
In the Intellectual Property & Software Development PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Activity-Beach Ball Game,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300/CEV71512_V2_Activity01
This Activity is found in the Intellectual Property & Software Development lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

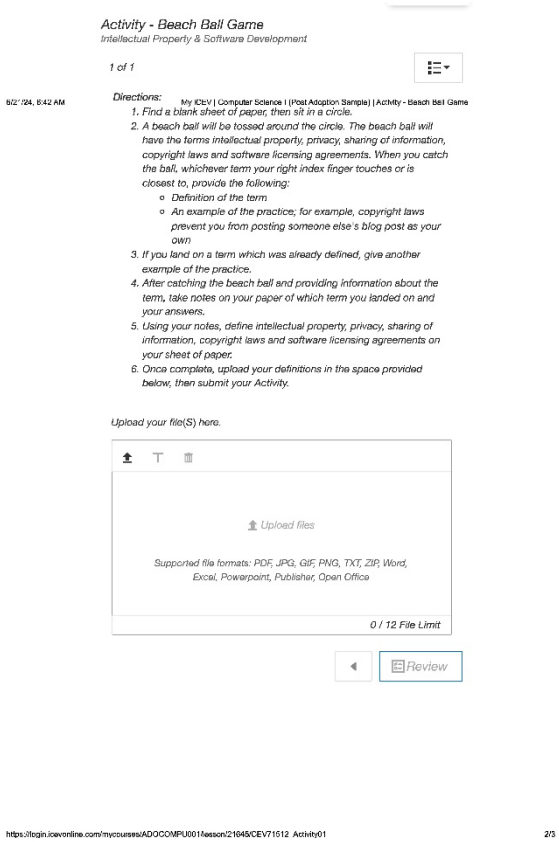
Insert a screenshot of your currently adopted content.

6/21/2024

10

1

Update to Content Accepted by SRP



8/2/24, 8:42 AM My ICEV | Computer Science I (Post Adoption Sample) | Activity - Beach Ball Game

Activity - Beach Ball Game

Intellectual Property & Software Development

1 of 1

Directions:

1. Find a blank sheet of paper, then sit in a circle.
2. A beach ball will be tossed around the circle. The beach ball will have the terms intellectual property, privacy, sharing of information, copyright laws and software licensing agreements. When you catch the ball, whichever term your right index finger touches or is closest to, provide the following:
   - Definition of the term
   - An example of the practice; for example, copyright laws prevent you from posting someone else's blog post as your own
3. If you land on a term which was already defined, give another example of the practice.
4. After catching the beach ball and providing information about the term, take notes on your paper of which term you landed on and your answers.
5. Using your notes, define intellectual property, privacy, sharing of information, copyright laws and software licensing agreements on your sheet of paper.
6. Once complete, upload your definitions in the space provided below, then submit your Activity.

Upload your file(S) here.

Upload files

Supported file formats: PDF, JPG, Gif, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Review

https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21045/CEV71512_Activity01 2/3

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Ethics

- Is the idea individuals should try to do the right thing, morally, professionally and legally
- Can refer to computer science standards, including:
  - not using technology to harm others
  - not looking at others' data without permission
  - not taking information without permission
  - not contributing to or spreading misinformation

6

Ethics

- Can refer to computer science standards, including:
  - not purchasing pirated or copy software
  - not using others' resources unless authorized to do so
  - not claiming ownership of someone else's ideas

7

Intellectual Property

- Refers to original ideas, creative works or inventions protected by law
- Is an intangible form of property
- Includes software
- Is often shortened to "IP"

8

Copyright

- Is a legally binding type of IP which protects original works of authorship
  - authorship includes writing but also includes
    - coding
    - blogging
    - programming
- Was first defined by the Copyright Act of 1790
- Is usually owned by the creator of the IP but might be sold to another party

9

Copyright Laws

- Provide copyright owners with rights:
  - protection of ideas, code and software
  - ownership of computer programs
  - license agreements
  - safety from creation and distribution of derivative works
  - dictation of how copies are made either by sale or other transfer of ownership such as:
    - rental
    - leasing
    - lending

10

Copyright Laws Discussion

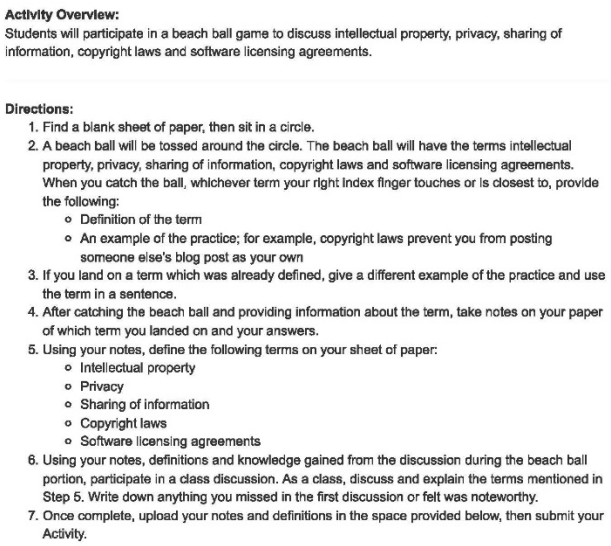
What impact do copyright laws have?

11

SAVE PROGRESS

Activity - Beach Ball Game
Intellectual Property & Software Development

Update to Content Accepted by SRP



Activity Overview:
Students will participate in a beach ball game to discuss intellectual property, privacy, sharing of information, copyright laws and software licensing agreements.

Directions:

1. Find a blank sheet of paper, then sit in a circle.
2. A beach ball will be tossed around the circle. The beach ball will have the terms intellectual property, privacy, sharing of information, copyright laws and software licensing agreements. When you catch the ball, whichever term your right index finger touches or is closest to, provide the following:
    - Definition of the term
    - An example of the practice; for example, copyright laws prevent you from posting someone else's blog post as your own
3. If you land on a term which was already defined, give a different example of the practice and use the term in a sentence.
4. After catching the beach ball and providing information about the term, take notes on your paper of which term you landed on and your answers.
5. Using your notes, define the following terms on your sheet of paper:
    - Intellectual property
    - Privacy
    - Sharing of information
    - Copyright laws
    - Software licensing agreements
6. Using your notes, definitions and knowledge gained from the discussion during the beach ball portion, participate in a class discussion. As a class, discuss and explain the terms mentioned in Step 5. Write down anything you missed in the first discussion or felt was noteworthy.
7. Once complete, upload your notes and definitions in the space provided below, then submit your Activity.

(SE)(Breakout(s)) and (Citation Type(s))
(5)(B)(i), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Intellectual Property and Software Development (Slides 16-17),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645

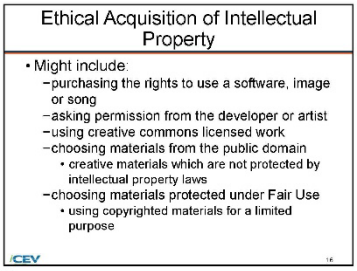
Description of the specific location and hyperlink to the exact location of the proposed new content
Intellectual Property and Software Development (Slides 19-20),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300
In the Intellectual Property & Software Development PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

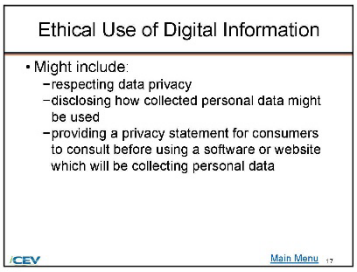
Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

6/21/2024



16



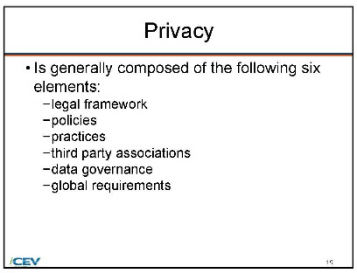
17

1

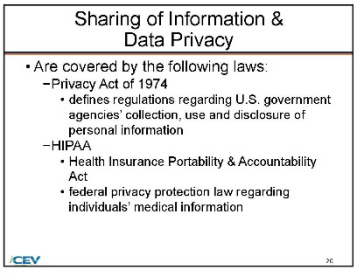
Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

6/20/2024



19



20

1

(SE)(Breakout(s)) and (Citation Type(s))

Update to Content Accepted by SRP

(5)(B)(ii), Narrative

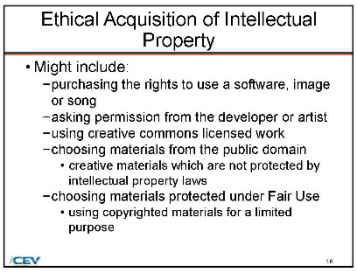
Description of the specific location and hyperlink to the exact location of currently adopted content
Intellectual Property and Software Development (Slides 16-17),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21645

Description of the specific location and hyperlink to the exact location of the proposed new content
Intellectual Property and Software Development (Slides 19-20),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22300
In the Intellectual Property & Software Development PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

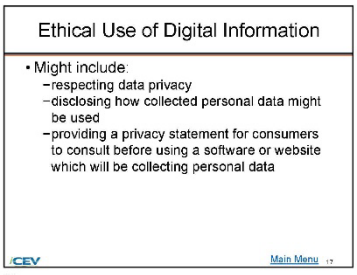
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/21/2024



16



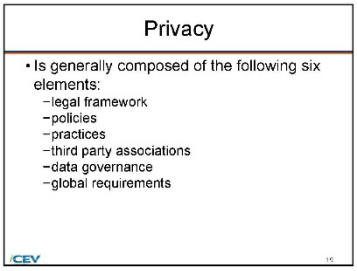
17

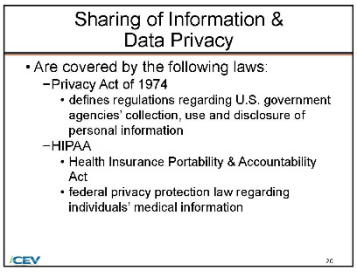
1

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP





(SE)(Breakout(s)) and (Citation Type(s))
(5)(C)(i), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Digital Etiquette and Security (Slides 27-30),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21646

Activity-Acceptable Use Scenarios,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21646/CEV71513_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Digital Etiquette and Security (Slides 27-31),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22302
In the Digital Etiquette & Security PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Activity-Acceptable Use Scenarios,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22302/CEV71513_V2_Activity01
This Activity is found in the Digital Etiquette & Security lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Digital Etiquette

- Refers to the way electronic communications should be conducted
- Is the behavior rules for using technology devices or interacting with others on the internet

27

Digital Etiquette

- Can include:
  - use of appropriate language
  - respect for other's privacy
  - avoiding inflammatory or offensive posts and comments
  - not spamming people
  - staying on topic in discussion forums and posts
  - double-checking messages and posts

28

Digital Responsibility

- Is using technology in an appropriate and constructive way
- Can involve a variety of ethical situations related to privacy and transparency

29

Digital Responsibility

- Can include:
  - obeying intellectual property laws
  - following rules of conduct for every internet site
  - reporting inappropriate or harmful content

30

Activity - Acceptable Use Scenarios

Digital Etiquette & Security

1 of 1

Activity Overview:
You will match each scenario with the policy it aligns best with. You will then analyze an article that explores fair use.

Directions:

1. Match the following scenarios to the policy they best align with.
2. Using the internet, locate a news article related to fair use and answer the questions listed below.
3. Once complete, submit your Activity.

NOTE:
To complete this question without using the drag-and-drop feature, first click on a single answer choice from the answer choice box, then click in the response container you wish to answer. This will "drop" the answer choice into the response container.

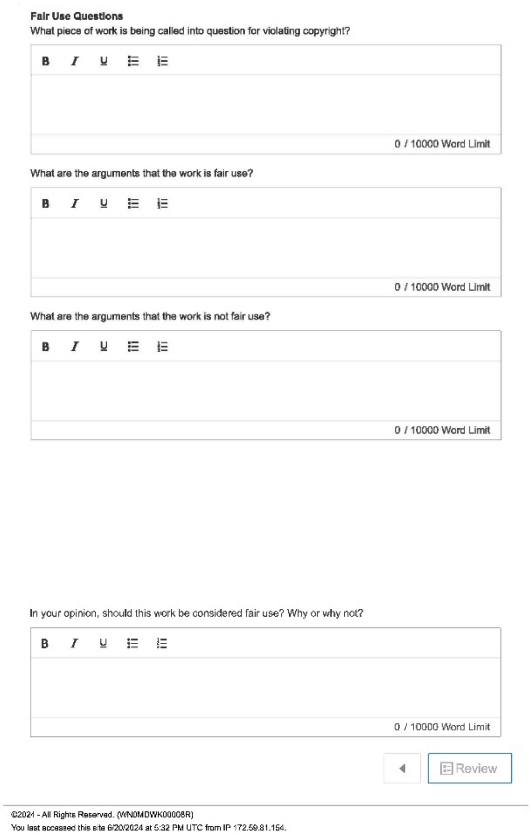
To complete this question while utilizing a screen reader, use the Tab key to navigate to an answer choice. Answer choices can be selected and inserted using the Enter key, Spacebar, left mouse button or touchpad. Using any of these keys, select your answer choice. Use the up and down arrow keys to navigate to the response container you wish to place the selected answer into. Press the key again to "drop" the answer choice into the response container.

Matching

Jae posts his code on GitHub with the intention of creating a project that many people can contribute to	
Priyanka creates a parody video of a popular song	
Jose creates a photography project showcasing people in different professions and wants the photos to be able to be shared freely by the public	
Kiara posts her artwork to social media and does not want anyone to copy it	

Fair Use | Copyright | Open Source | Creative Commons

Update to Content Accepted by SRP



Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Digital Etiquette

- Refers to the way electronic communications should be conducted
- Is the behavior rules for using technology devices or interacting with others on the internet
- Are not enforced laws, but rather widely accepted social norms technology users follow and abide by

27

Digital Etiquette

- Should be demonstrated by:
  - using appropriate language
  - respecting other's privacy
  - avoiding inflammatory or offensive posts and comments
  - not spamming people
  - staying on topic in discussion forums and posts
  - double-checking messages and posts

28

Digital Etiquette Discussion

What are other examples of how to demonstrate proper digital etiquette?

29

Digital Responsibility

- Is using technology in an appropriate and constructive way
- Can involve a variety of ethical situations related to privacy and transparency

30

Digital Responsibility

- Can include:
  - obeying intellectual property laws
  - following rules of conduct for every internet site
  - reporting inappropriate or harmful content
  - following acceptable use policies
  - adhering to community guidelines

31

Update to Content Accepted by SRP

Activity - Acceptable Use Scenarios

Digital Etiquette & Security

1 of 1

Activity Overview:
You will match each scenario with the policy it aligns best with. You will then analyze an article that explores fair use.

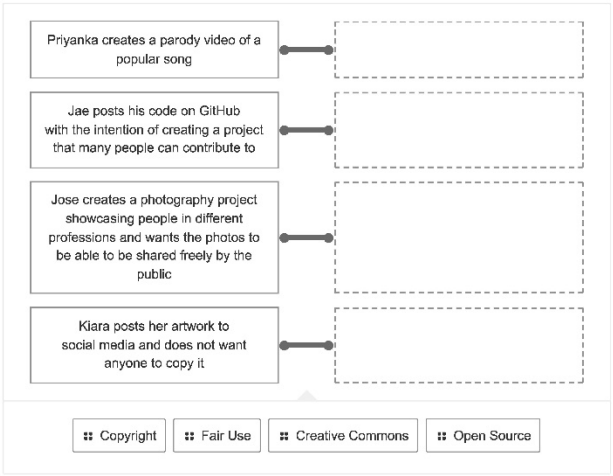
Directions:

1. Match the following scenarios to the policy they best align with.
2. Using the internet, locate a news article related to fair use and answer the questions listed below.
3. Once complete, submit your Activity.

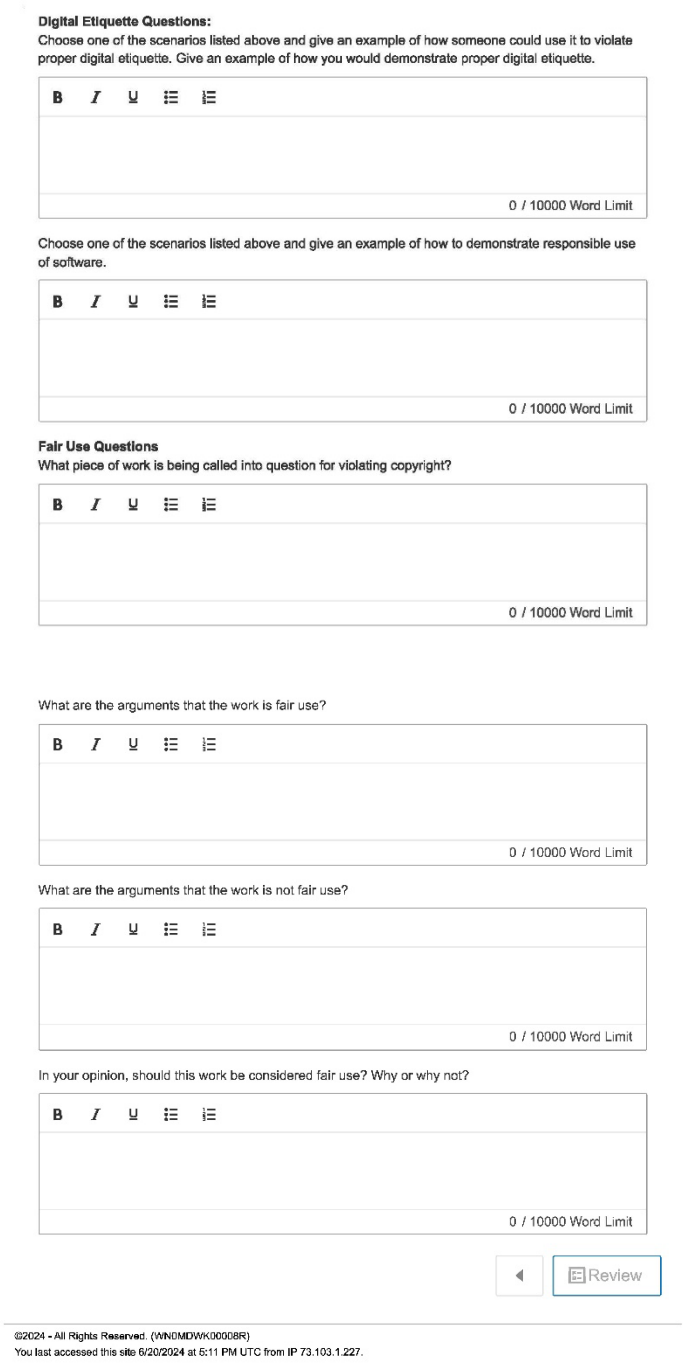
NOTE:
To complete this question without using the drag-and-drop feature, first click on a single answer choice from the answer choice box, then click in the response container you wish to answer. This will "drop" the answer choice into the response container.

To complete this question while utilizing a screen reader, use the Tab key to navigate to an answer choice. Answer choices can be selected and inserted using the Enter key, Spacebar, left mouse button or touchpad. Using any of these keys, select your answer choice. Use the up and down arrow keys to navigate to the response container you wish to place the selected answer into. Press the key again to "drop" the answer choice into the response container.

Matching



Update to Content Accepted by SRP



Digital Etiquette Questions:
Choose one of the scenarios listed above and give an example of how someone could use it to violate proper digital etiquette. Give an example of how you would demonstrate proper digital etiquette.

0 / 10000 Word Limit

Choose one of the scenarios listed above and give an example of how to demonstrate responsible use of software.

0 / 10000 Word Limit

Fair Use Questions
What piece of work is being called into question for violating copyright?

0 / 10000 Word Limit

What are the arguments that the work is fair use?

0 / 10000 Word Limit

What are the arguments that the work is not fair use?

0 / 10000 Word Limit

In your opinion, should this work be considered fair use? Why or why not?

0 / 10000 Word Limit

Review

©2024 - All Rights Reserved. (WN0MDWK00008R)
You last accessed this site 6/20/2024 at 5:11 PM UTC from IP 73.103.1.227.

(SE)(Breakout(s)) and (Citation Type(s))
(5)(C)(ii), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Digital Etiquette and Security (Slides 27-30, 33),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21646

Update to Content Accepted by SRP

Activity-Acceptable Use Scenarios,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21646/CEV71513_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Digital Etiquette and Security (Slides 27-34),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22302
In the Digital Etiquette & Security PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Activity-Acceptable Use Scenarios,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22302/CEV71513_V2_Activity01
This Activity is found in the Digital Etiquette & Security lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Digital Etiquette

- Refers to the way electronic communications should be conducted
- Is the behavior rules for using technology devices or interacting with others on the internet

27

Digital Etiquette

- Can include:
  - use of appropriate language
  - respect for other's privacy
  - avoiding inflammatory or offensive posts and comments
  - not spamming people
  - staying on topic in discussion forums and posts
  - double-checking messages and posts

28

Digital Responsibility

- Is using technology in an appropriate and constructive way
- Can involve a variety of ethical situations related to privacy and transparency

29

Digital Responsibility

- Can include:
  - obeying intellectual property laws
  - following rules of conduct for every internet site
  - reporting inappropriate or harmful content

30

Update to Content Accepted by SRP

6/21/2024

Copyright in Software

- Applies to code programmers write
- Infringement can be:
  - the plagiarism of someone else's code
  - continuing to use software after a subscription has ended
  - hackers selling counterfeit versions of a software
    - make sure to use legal copyrighted software and not pirated versions

CEV

33

3

Activity - Acceptable Use Scenarios

Digital Etiquette & Security

1 of 1

Activity Overview:
You will match each scenario with the policy it aligns best with. You will then analyze an article that explores fair use.

Directions:
1. Match the following scenarios to the policy they best align with.
2. Using the internet, locate a news article related to fair use and answer the questions listed below.
3. Once complete, submit your Activity.

NOTE:
To complete this question without using the drag-and-drop feature, first click on a single answer choice from the answer choice box, then click in the response container you wish to answer. This will "drop" the answer choice into the response container.

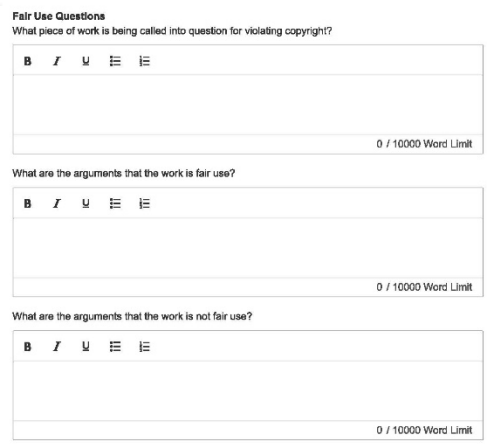
To complete this question while utilizing a screen reader, use the Tab key to navigate to an answer choice. Answer choices can be selected and inserted using the Enter key, Spacebar, left mouse button or touchpad. Using any of these keys, select your answer choice. Use the up and down arrow keys to navigate to the response container you wish to place the selected answer into. Press the key again to "drop" the answer choice into the response container.

Matching

Jae posts his code on GitHub with the intention of creating a project that many people can contribute to	
Priyanka creates a parody video of a popular song	
Jose creates a photography project showcasing people in different professions and wants the photos to be able to be shared freely by the public	
Kiara posts her artwork to social media and does not want anyone to copy it	

Fair Use | Copyright | Open Source | Creative Commons

Update to Content Accepted by SRP



Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

6/20/2024

Digital Etiquette

- Refers to the way electronic communications should be conducted
- Is the behavior rules for using technology devices or interacting with others on the internet
- Are not enforced laws, but rather widely accepted social norms technology users follow and abide by

27

Digital Etiquette

- Should be demonstrated by:
  - using appropriate language
  - respecting other's privacy
  - avoiding inflammatory or offensive posts and comments
  - not spamming people
  - staying on topic in discussion forums and posts
  - double-checking messages and posts

28

1

6/20/2024

Digital Etiquette Discussion

What are other examples of how to demonstrate proper digital etiquette?

29

Digital Responsibility

- Is using technology in an appropriate and constructive way
- Can involve a variety of ethical situations related to privacy and transparency

30

2

Update to Content Accepted by SRP

Digital Responsibility

- Can include:
  - obeying intellectual property laws
  - following rules of conduct for every internet site
  - reporting inappropriate or harmful content
  - following acceptable use policies
  - adhering to community guidelines

31

Digital Responsibility

- Can include:
  - being responsible with software utilization
    - not using or developing software to harass other users
    - not using software to infiltrate or cause damage to a network
    - avoiding plagiarism
    - maintain security by using strong passwords, scanning for and patching vulnerabilities, and updating software

32

Digital Responsibility Discussion

What are other examples of how to demonstrate responsible use of software?

33

Copyright

- Is the exclusive right over a created material
- Protects creators from others publishing, selling or distributing their intellectual property

34

Activity - Acceptable Use Scenarios

Digital Etiquette & Security

1 of 1

Activity Overview:
You will match each scenario with the policy it aligns best with. You will then analyze an article that explores fair use.

Directions:

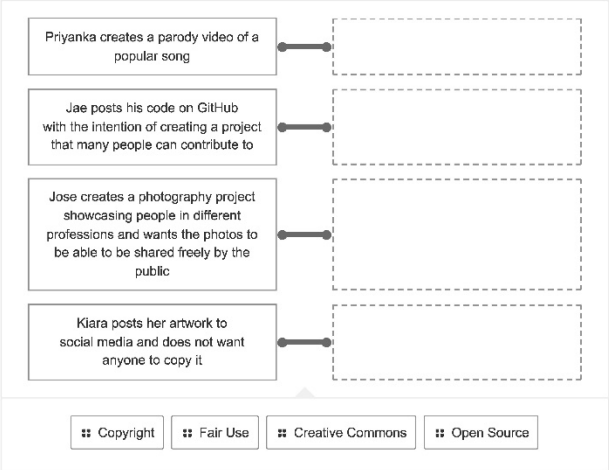
1. Match the following scenarios to the policy they best align with.
2. Using the internet, locate a news article related to fair use and answer the questions listed below.
3. Once complete, submit your Activity.

NOTE:
To complete this question without using the drag-and-drop feature, first click on a single answer choice from the answer choice box, then click in the response container you wish to answer. This will "drop" the answer choice into the response container.

To complete this question while utilizing a screen reader, use the Tab key to navigate to an answer choice. Answer choices can be selected and inserted using the Enter key, Spacebar, left mouse button or touchpad. Using any of these keys, select your answer choice. Use the up and down arrow keys to navigate to the response container you wish to place the selected answer into. Press the key again to "drop" the answer choice into the response container.

Update to Content Accepted by SRP

Matching



Digital Etiquette Questions:

Choose one of the scenarios listed above and give an example of how someone could use it to violate proper digital etiquette. Give an example of how you would demonstrate proper digital etiquette.

0 / 10000 Word Limit

Choose one of the scenarios listed above and give an example of how to demonstrate responsible use of software.

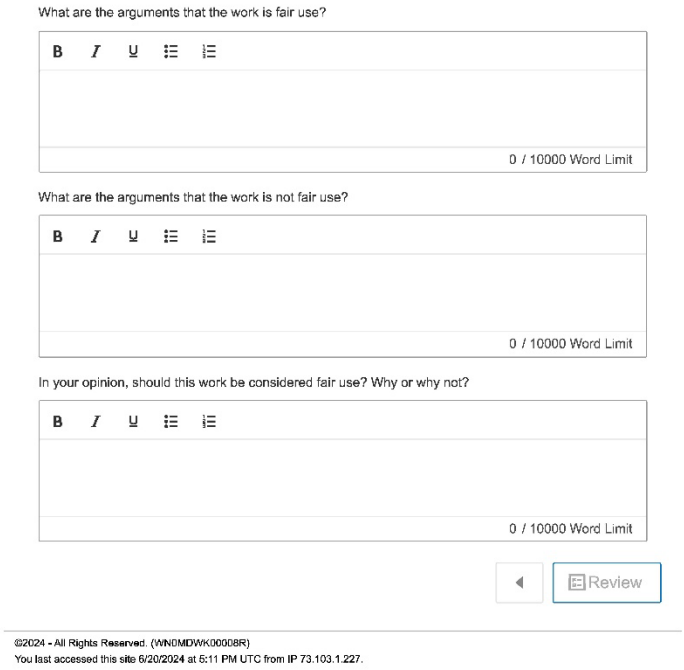
0 / 10000 Word Limit

Fair Use Questions

What piece of work is being called into question for violating copyright?

0 / 10000 Word Limit

Update to Content Accepted by SRP



(SE)(Breakout(s)) and (Citation Type(s))
(5)(C)(iii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Digital Etiquette and Security (Slides 27-30, 39),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21646

Description of the specific location and hyperlink to the exact location of the proposed new content
Digital Etiquette and Security (Slides 27-31, 40-41),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22302
In the Digital Etiquette & Security PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Digital Etiquette

- Refers to the way electronic communications should be conducted
- Is the behavior rules for using technology devices or interacting with others on the internet

27

Digital Etiquette

- Can include:
  - use of appropriate language
  - respect for other's privacy
  - avoiding inflammatory or offensive posts and comments
  - not spamming people
  - staying on topic in discussion forums and posts
  - double-checking messages and posts

28

Digital Responsibility

- Is using technology in an appropriate and constructive way
- Can involve a variety of ethical situations related to privacy and transparency

29

Digital Responsibility

- Can include:
  - obeying intellectual property laws
  - following rules of conduct for every internet site
  - reporting inappropriate or harmful content

30

Digital Responsibility

- Includes understanding acceptable use of digital materials
  - checking the license of work can help to avoid issues of copyright infringement
  - understanding copyright can help when developing your own work to be shared with others

Main Menu

39

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Digital Etiquette

- Refers to the way electronic communications should be conducted
- Is the behavior rules for using technology devices or interacting with others on the internet
- Are not enforced laws, but rather widely accepted social norms technology users follow and abide by

27

Digital Etiquette

- Should be demonstrated by:
  - using appropriate language
  - respecting other's privacy
  - avoiding inflammatory or offensive posts and comments
  - not spamming people
  - staying on topic in discussion forums and posts
  - double-checking messages and posts

28

Digital Etiquette Discussion

What are other examples of how to demonstrate proper digital etiquette?

29

Digital Responsibility

- Is using technology in an appropriate and constructive way
- Can involve a variety of ethical situations related to privacy and transparency

30

Digital Responsibility

- Can include:
  - obeying intellectual property laws
  - following rules of conduct for every internet site
  - reporting inappropriate or harmful content
  - following acceptable use policies
  - adhering to community guidelines

31

Open Source

- Is code accessible to the public
- Allows programmers to post their code so others can modify and distribute it
  - can be helpful for community-built projects
  - leads to peer review as programmers are frequently accessing and improving on the code
- Is helpful for people learning to program

41

Creative Commons

- Allows creators to license their work with different specifications for the general public to use
  - creators can specify whether adaptations of their work will be allowed
  - creators can specify whether their work can be used for commercial purposes

40

(SE)(Breakout(s)) and (Citation Type(s))
(5)(D)(iv), Activity

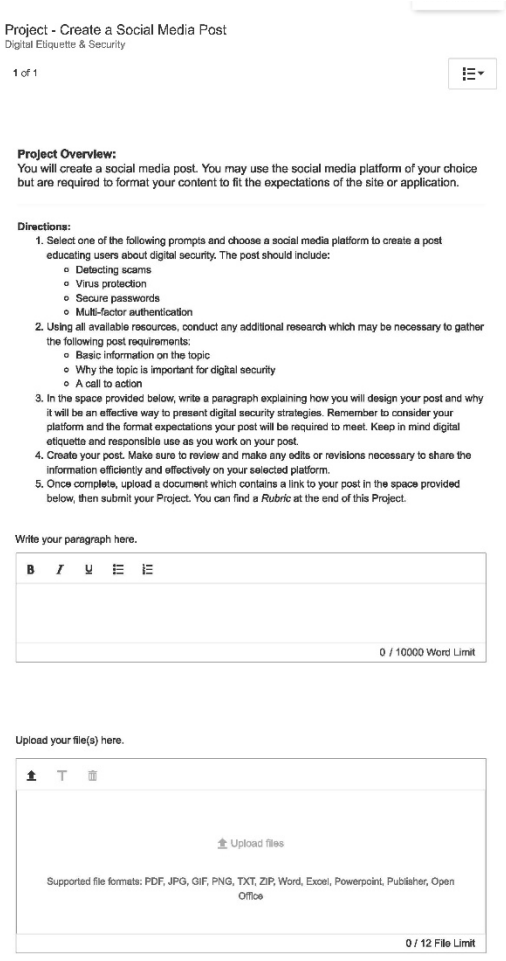
Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Project-Create a Social Media Post,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21646/CEV71513_Project01

Description of the specific location and hyperlink to the exact location of the proposed new content
Digital Etiquette and Security Project-Create a Social Media Post,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22302/CEV71513_V2_Project01
This Project is found in the Digital Etiquette & Security lesson beneath the Interactive Assignments heading. After clicking the link to the Project, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Project.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



Update to Content Accepted by SRP

Rubric

Description	Possible Points
Research & Organization: • The social media post was organized effectively and efficiently • All required topics were researched thoroughly • Information was presented in a logical organized manner	35
Concept & Understanding: • Student clearly understands how to detect scams, virus protection, creating secure passwords, and multi-factor authentication • Student created a social media post which effectively portrayed the content	45
Creativity/Craftmanship: • Social media post contains creativity in the way it is written • Social media post contains relative imagery (if included)	10
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	10
Total Points	100

◂ Review

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

SAVE PROGRESS

Project - Create a Social Media Post

Digital Etiquette & Security

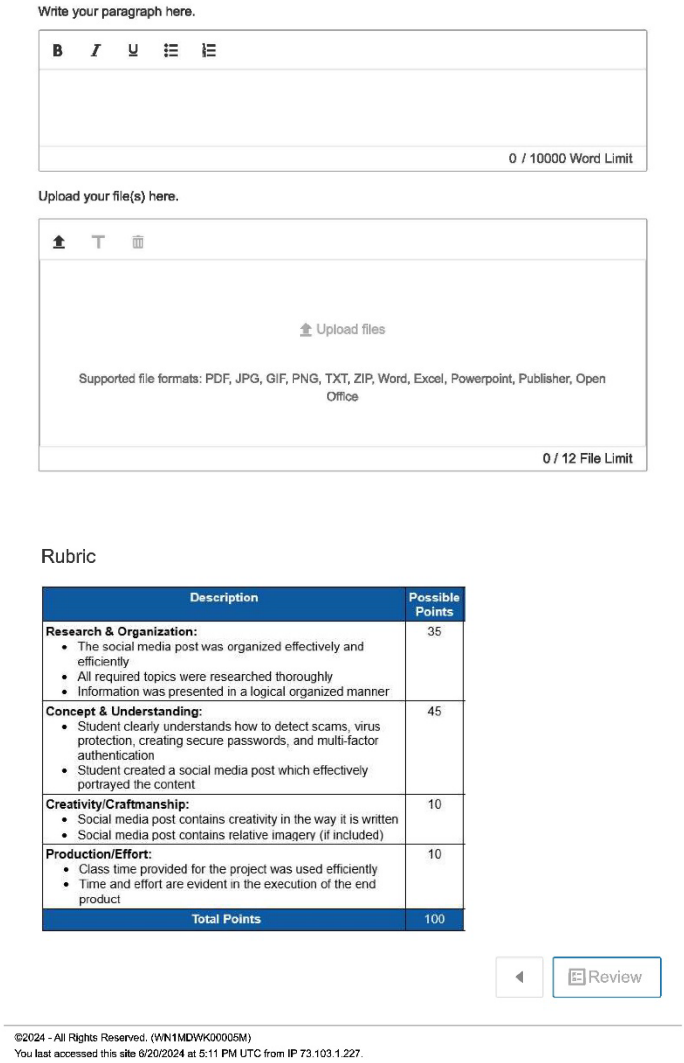
1 of 1

Project Overview:
You will create a social media post. You may use the social media platform of your choice but are required to format your content to fit the expectations of the site or application.

Directions:

1. Select one of the following prompts and choose a social media platform to create a post educating users about digital security. The post should include:
   - Detecting scams
   - Privacy and security measures, including virus protection and detection
   - Virus protection
   - Secure passwords
   - Multi-factor authentication
2. Using all available resources, conduct any additional research which may be necessary to gather the following post requirements:
   - Basic information on the topic
   - Why the topic is important for digital security
   - A call to action
3. In the space provided below, write a paragraph explaining how you will design your post and why it will be an effective way to present digital security strategies. Remember to consider your platform and the format expectations your post will be required to meet. Keep in mind digital etiquette and responsible use as you work on your post.
4. Create your post. Make sure to review and make any edits or revisions necessary to share the information efficiently and effectively on your selected platform.
5. Once complete, upload a document which contains a link to your post in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Update to Content Accepted by SRP



Write your paragraph here.

0 / 10000 Word Limit

Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Research & Organization: • The social media post was organized effectively and efficiently • All required topics were researched thoroughly • Information was presented in a logical organized manner	35
Concept & Understanding: • Student clearly understands how to detect scams, virus protection, creating secure passwords, and multi-factor authentication • Student created a social media post which effectively portrayed the content	45
Creativity/Craftmanship: • Social media post contains creativity in the way it is written • Social media post contains relative imagery (if included)	10
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	10
Total Points	100

Review

©2024 - All Rights Reserved. (WN1MDWK00005M)
You last accessed this site 6/20/2024 at 5:11 PM UTC from IP 73.103.1.227.

(SE)(Breakout(s)) and (Citation Type(s))
(6)(B)(ii), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Defining Programming Languages (Slides 4-16),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21648
Student Handout-History of Programming,
https://files.icevonline.com/html/CEV71515_TXP24/CEV71515_TXP24_Student_Handout_-_History_of_Programming.htm
Activity-KWL Chart Bell Ringer/Exit Ticket,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21648/CEV71515_Activity01?resume=False
Activity-Programming Languages,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21648/CEV71515_Activity02

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of the proposed new content
Defining Programming Languages (Slides 4-17),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22304
In the Defining Programming Languages PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.

Student Handout-History of Programming,
https://files.icevonline.com/html/CEV71515_V2_HTML/CEV71515_V2_HTML_Student_Handout_-_History_of_Programming.htm
This Student Handout is found in the Defining Programming Languages lesson beneath the Instructional Materials heading.

Activity-Programming Languages,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22304/CEV71515_V2_Activity02
This Activity is found in the Defining Programming Languages lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Activity-KWL Chart Bell Ringer,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22304/CEV71515_V2_Activity01?resume=False
This Activity is found in the Defining Programming Languages lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Programming Languages

- Have evolved in complexity and functionality over time
  - for example, in the 20th century, punch code was a common mechanism used for machines to read and interpret data as holes were punched in specific locations to signify characters or data
  - this method and many others have advanced programming and data processing

4

Machine Language

- Is the language computers understand
  - binary language consisting of digits or bits
    - a bit is one piece of information, either a 0 or 1
- Is considered low-level language

5

Low-Level Programming Languages

- Are closer to machine code, or binary
- Are harder to learn and understand
- Are machine-dependent
- Need an assembler for translation
- Include assembly language and machine language

6

Programming Language

- Is written in English so humans can write and understand it when coding
  - humans write in programming language, then it is translated into machine language
    - for example,
      - machine language for the text "Hello World" is 01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010 01101100 01100100
- Can include Python, Java or C++
- Is considered high-level language

7

High-Level Programming Languages

- Use English words and mathematical symbols
- Include C, C++, Java and Python
- Are easy to learn, maintain and debug
- Can generally run on any platform but need a compiler or interpreter for translation

8

Compiling & Interpreting

- Are required to translate high-level source code into machine code
- Are determined by the programming language used, the type of application and the target platform
- May be done by the same programming language, depending on the job at hand

9

Compiled Languages

- Convert the entire source code into machine code, saves it as an executable file, then runs it independently of the source code
- Are efficient and suitable for larger, more complex projects or for applications needing to run on more than one platform
- Include C, C++ and Java

10

Interpreted Languages

- Read and execute the source code line-by-line, as it is encountered
  - require the interpreter be present on the computer where the program is being run
- Are easier to write and debug because the source code is being executed directly and errors can be identified right away
- Are suitable for small, relatively simple projects
- Include Python, Ruby and JavaScript

11

Update to Content Accepted by SRP

Popular Programming Languages

- Are often multi-purpose and used in more than one area of computer science
- Include :
  - Python
  - Java
  - JavaScript
  - C++
- Are used to create the software and applications used every day

Python

- Is one of the most popular multi-purpose languages used today
- Is considered strong, dynamically typed
- Is used in the following areas of computer science:
  - cybersecurity
  - data science
  - game development
  - cloud computing
  - application and software development

C++

- Is an untyped language
- Is used in the following areas of computer science:
  - application and software development, especially with a large number of graphics

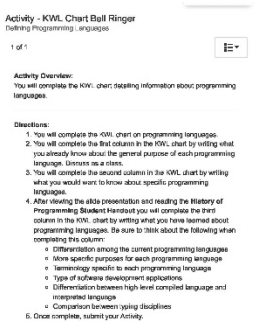
Java

- Is considered a statically typed language
- Is used in the following areas of computer science:
  - game development
  - cloud computing
  - application and software development

JavaScript

- Is an untyped language
- Is used in the following area of computer science:
  - web-based applications

Update to Content Accepted by SRP



Activity - KWL Chart Bell Ringer

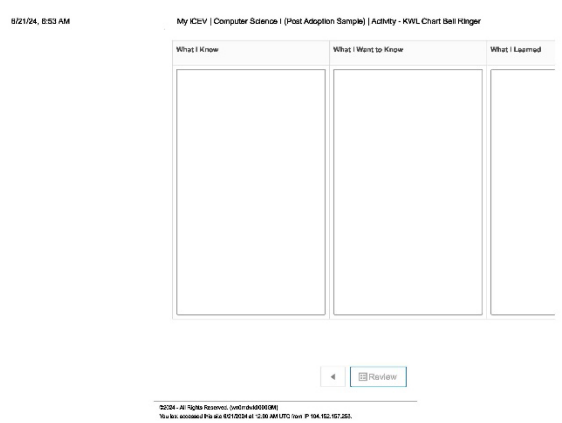
Defining Programming Languages

1 of 1

Activity Overview:
You will complete the KWL chart detailing information about programming languages.

Directions:

1. You will complete the KWL chart on programming languages.
2. You will complete the first column in the KWL chart by writing what you already know about the general purpose of each programming language. Discuss as a class.
3. You will complete the second column in the KWL chart by writing what you would want to know about specific programming languages.
4. After viewing the slide presentation and reading the **History of Programming Student Handout** you will complete the third column in the KWL chart by writing what you have learned about programming languages. Be sure to think about the following when completing this column:
   - Differentiation among the current programming languages
   - More specific purposes for each programming language
   - Terminology specific to each programming language
   - Type of software development applications
   - Differentiation between high level compiled language and interpreted language
   - Comparison between typing disciplines
5. Once complete, submit your Activity.



8/21/24, 8:53 AM My ICEV | Computer Science I (Post Adoption Sample) | Activity - KWL Chart Bell Ringer

What I Know	What I Want to Know	What I Learned

Review

©2024 - All Rights Reserved. [redacted]
You last accessed this site 8/21/2024 at 12:00 AM UTC from IP 104.152.167.253.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Programming Languages

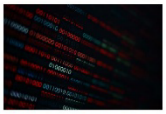
- Have evolved in complexity and functionality over time
  - for example, in the 20th century, punch code was a common mechanism used for machines to read and interpret data as holes were punched in specific locations to signify characters or data
  - this method and many others have advanced programming and data processing

ICEV 4

4

Machine Language

- Is the language computers understand
  - binary language consisting of digits or bits
    - a bit is the smallest unit of data containing one of two values, 0 or 1
- Is considered low-level language



ICEV 5

5

Low-Level Programming Languages

- Are closer to machine code, or binary
- Are harder to learn and understand
- Are machine-dependent
- Need an assembler for translation
- Include assembly language and machine language

ICEV 6

6

Programming Language

- Is written in English so humans can write and understand it when coding
  - humans write in programming language, then it is translated into machine language
    - for example,
      - machine language for the text "Hello World" is 01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010 01101100 01100100
- Can include Python, Java or C++
- Is considered high-level language

ICEV 7

7

Update to Content Accepted by SRP

High-Level Programming Languages

- Use English words and mathematical symbols
- Include C#, C++, Java and Python
- Are easy to learn, maintain and debug
- Can generally run on any platform but need a compiler or interpreter for translation

8

Compiling & Interpreting

- Are required to translate high-level source code into machine code
- Are determined by the programming language used, the type of application and the target platform
- May be done by the same programming language, depending on the job at hand

9

Popular Programming Languages

- Are often multi-purpose and used in more than one area of computer science
- Include :
  - Python
  - Java
  - JavaScript
  - C++
- Are used to create the software and applications used every day

12

Python

- Is one of the most popular languages used today
- Provides a general programming language which emphasizes code readability and simplicity, while still being powerful enough to develop advanced programming

13

Compiled Languages

- Convert the entire source code into machine code, saves it as an executable file, then runs it independently of the source code
- Are efficient and suitable for larger, more complex projects or for applications needing to run on more than one platform
- Include C, C++ and Java

10

Interpreted Languages

- Read and execute the source code line-by-line, as it is encountered
  - require the interpreter be present on the computer where the program is being run
- Are easier to write and debug because the source code is being executed directly and errors can be identified right away
- Are suitable for small, relatively simple projects
- Include Python, Ruby and JavaScript

11

Python

- Is considered a strong, dynamically typed language
- Is used in the following areas of computer science:
  - cybersecurity
  - data science
  - game development
  - cloud computing
  - application and software development

14

Java

- Is considered a statically typed language
- Provides a language which is versatile, portable and has a wide variety of applications in many different scales of development
- Is used in the following areas of computer science:
  - game development
  - cloud computing
  - application and software development

15

Update to Content Accepted by SRP

JavaScript

- Is an untyped language
- Is generally used in web development to enhance web pages to make them more dynamic and responsive
- Is used in the following area of computer science:
  - web-based applications

iCEV

16

16

C++

- Is an untyped language
- Is generally used as a powerful programming development tool in graphics, software and operating systems that is capable of high-level and low-level system application
- Is used in the following areas of computer science:
  - application and software development, especially with a large number of graphics

iCEV

17

17

LIVE CHAT HELP SCHEDULE ONLINE TRAINING

Computer Science I - UPDATED

My Courses / Computer Science I - UPDATED / Defining Programming Languages - UPDATED / Activity - KWL Chart Bell Ringer

Highlight any text to hear text-to-voice speech.
Select Language

SAVE PROGRESS

Activity - KWL Chart Bell Ringer
Defining Programming Languages

1 of 1

Activity Overview:
You will complete the KWL chart detailing information about programming languages.

Directions:

1. You will complete the KWL chart on programming languages.
2. You will complete the first column in the KWL chart by writing what you already know about the general purpose of each programming language. Discuss as a class.
3. You will complete the second column in the KWL chart by writing what you would want to know about specific programming languages.
4. After viewing the slide presentation and reading the **History of Programming Student Handout** you will complete the third column in the KWL chart by writing what you have learned about programming languages. Be sure to think about the following when completing this column:
   - Differentiation among the current programming languages
   - General purpose for each programming language
   - Terminology specific to each programming language
   - Type of software development applications
   - Differentiation between high-level compiled language and interpreted language
   - Comparison between typing disciplines
5. Once complete, submit your Activity.

(SE)(Breakout(s)) and (Citation Type(s))
(6)(B)(iv), Activity

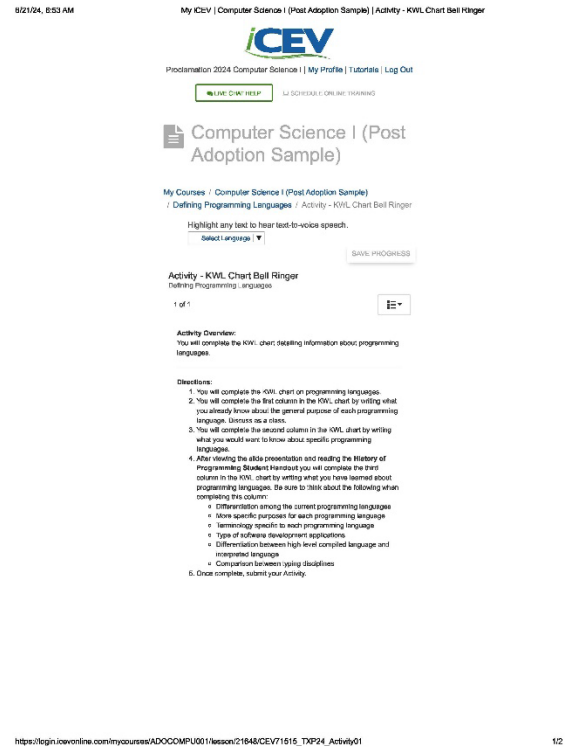
Update to Content Accepted by SRP

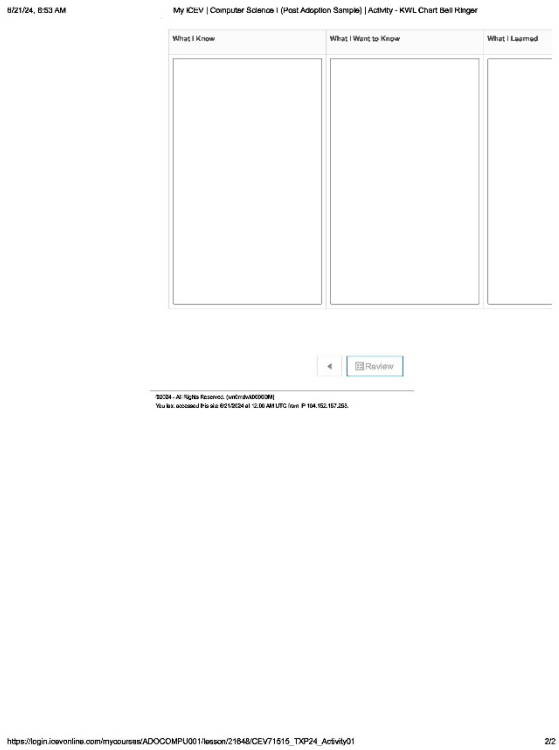
Description of the specific location and hyperlink to the exact location of currently adopted content
Activity-KWL Chart Bell Ringer/Exit Ticket,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21648/CEV71515_Activity01?resume=False
Activity-Programming Languages,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21648/CEV71515_Activity02

Description of the specific location and hyperlink to the exact location of the proposed new content
Coding Challenge: Town Population Growth,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22048/CEV71811_SIM01
Access to the interactive coding environment can be located beneath the Interactive Assignments heading by clicking the link to the Coding Challenge. Once clicked, the link will take you to a page prompting you to click Start. Select Start to view the Coding Challenge in the interactive environment.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.





Update to Content Accepted by SRP

Highlight any text to hear text-to-voice speech.
Select Language
Powered by Google Translate

Programming Languages

Activity Overview:
You will identify programming languages which can be used in different scenarios.

Directions:
1. Read each scenario.

2. Using the lesson content and any outside research, decide which programming language you would use and why for each scenario. Then, provide an example of a similar application with which you are familiar.

3. Turn in your completed activity according to your instructor's directions.

Scenario #1:
You are developing the newest mobile app for storing and sorting music for users.
Language/program choice and reasoning:

Example:

Scenario #2:
Your program requires statically typed variables and will be used for an app for connecting users with services in the area.
Language/program choice and reasoning:

Example:

Scenario #3:
You are developing a web-based application that does not necessarily need to follow any typing discipline.
Language/program choice and reasoning:

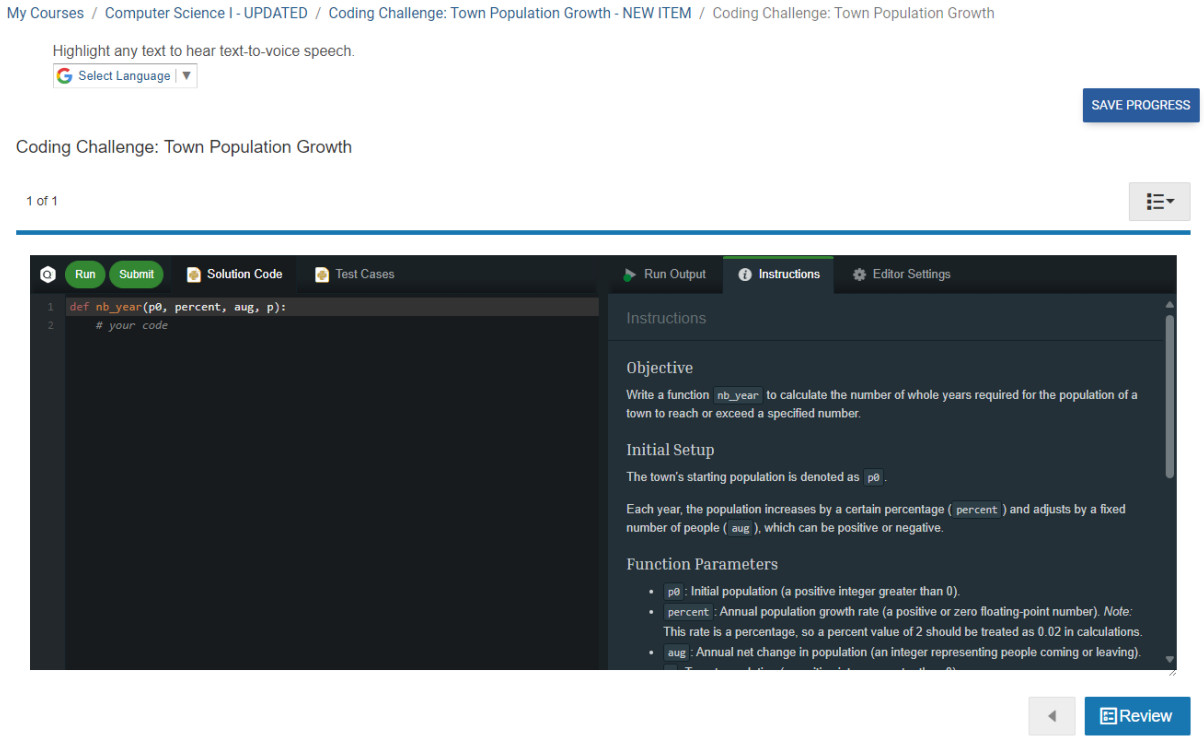
Example:

Scenario #4:
You are aspiring to develop the next suite of desktop programs and they will use many graphics.
Language/program choice and reasoning:

Example:

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.



(SE)(Breakout(s)) and (Citation Type(s))
(6)(H)(iii), Narrative

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Subroutines and Data (Slides 3-6, 24-28),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21650

Description of the specific location and hyperlink to the exact location of the proposed new content
Subroutines and Data Student Handout-Subroutines,
https://files.icevonline.com/html/CEV71517_V2_HTML/CEV71517_V2_HTML_Student_Handout_-_Subroutines.htm

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Subroutines

- Are named blocks of code that can be called multiple times from different parts of a program
- Are also known as functions or methods
- Are an essential part of modular programming
- Can help to simplify the design and organization of a program

3

Subroutines Example

- Includes:
  - defined subroutine 'fahrenheit_to_celsius' takes a single argument 'tfhrenheit' (the temperature in Fahrenheit) and returns the temperature in Celsius
    - formula for converting Fahrenheit (F) to Celsius (C) is '(F-32)*5/9

4

Subroutines Example

- Includes:
  - next, prompt the user to enter a temperature in F using the 'input' function and convert it to a floating-point number using the 'float' function
  - then, call the 'Fahrenheit_to_Celsius' subroutine to convert the temperature to C and store the result in the variable 'Celsius'

5

Subroutines Example

- Includes:
  - lastly, use the 'print' function to display the original temperature in F and the converted temperature in C to the user

6

Update to Content Accepted by SRP

Parameters

- Are the variables defined in the function definition
  - for example, the 'sum()' function has a parameter "*numbers"

```
def sum(*numbers):
    total = 0
    for n in numbers:
        total += n
    return total

# We can call the function with any number of arguments
print(sum(1, 2, 3, 4)) # Output: 10
print(sum(10))         # Output: 10 (sum of a single number)
print(sum(2, 4, 6, 8)) # Output: 20
```

24

Arguments

- Are the values passed into the function when it is called
  - for example, parameter, '*numbers", means it can accept any numbers or arguments

```
def sum(*numbers):
    total = 0
    for n in numbers:
        total += n
    return total

# We can call the function with any number of arguments
print(sum(1, 2, 3, 4)) # Output: 10
print(sum(10))         # Output: 10 (sum of a single number)
print(sum(2, 4, 6, 8)) # Output: 20
```

25

Subroutines Without Return Values Example

- Includes:
  - the 'say_hello()' subroutine does not take any arguments or parameters and does not return a value
  - instead, it prints the string '"Hello, world!"' to the console when it is called

```
def say_hello():
    print("Hello, world!")

# Call the subroutine
say_hello()  # Output: Hello, world!
```

28

Function

- Executes to add up all the numbers passed into the arguments and returns the total as an output
- Can be called with any number of arguments, and it will always return the sum of all the arguments

```
# We can call the function with any number of arguments
print(sum(1, 2, 3, 4)) # Output: 10
print(sum(10))         # Output: 10 (sum of a single number)
print(sum(2, 4, 6, 8)) # Output: 20
```

26

Creating Subroutines Without Return Values

- Is similar to creating subroutines that return values
  - instead of using the 'return' statement to return a value, the function can be used to perform some of the actions

27

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Subroutines

Subroutines are a way for a user to organize their code by breaking the code down into smaller pieces. The pieces of code are written once and then can be called mutiple times from various parts of a program. Subroutines enhance a code's readability, maintainability and reusability.

Parameters are the variables defined in the function definition. For example, the 'sum()' function has a parameter "numbers". Arguments are the values passed into the function when it is called. For example, parameter, "*numbers", means it can accept any numbers or arguments.

```
def sum(*numbers):
      total = 0
      for n in numbers:
         total += n
      return total

# We can call the function with any number of arguments
print(sum(1, 2, 3, 4)) # Output: 10
print(sum(10))         # Output: 10 (sum of a single number)
print(sum(2, 4, 6, 8)) # Output: 20
```

Return Typed Values Without the Use of Arguments
Subroutines can be created to return typed values without the use of arguments. This can be referred to as parameterless functions. Parameterless functions do not take any parameters, but they do perform a task and then return a result. The result of this function can be a data type.

The following is an example of a subroutine created to return a typed value without the use of an argument. In this example, the generate_random_number function does not have an argument but will return a random number between 1 and 50.

```
import random

def generate_random_number():
    return random.randint(1, 50)

# Call the function and get the result
random_number = generate_random_number()
print("Random Number:", random_number)
```

Return Typed Values With the Use of Arguments

```
# Call the function with arguments and get the result
result = add_numbers(7, 9)
print("Result:", result)
```

CEV Copyright CEV Multimedia, LLC

Subroutines can be created to return typed values with the use of arguments. This is often referred to as methods or functions in programming languages. The function will accept an input parameter and perform the operation based on the input. Then, after the operation is performed, a specific result will occur.

The following is an example of a subroutine created to return a typed value with the use of an argument. In this example, the subtract_numbers function takes the two arguments x and y, subtracts them, and then returns the result.

```
def subtract_numbers(x, y):
    return x - y

# Call the function with arguments and get the result
result = subtract_numbers(3, 8)
print("Result:", result)
```

Return Typed Values Without the Use of Parameters
Subroutines can be created to return typed values without the use of parameters. These subroutines are functions or methods that do not take input arguments but will produce a result.

The following is an example of a subroutine created to return a typed value without the use of a parameter. In this example, the get_current_year function does not have an argument but still returns the year with the datetime module.

```
import datetime

def get_current_year():
    return datetime.datetime.now().year

# Call the function and get the result
current_year = get_current_year()
print("Current Year:", current_year)
```

Return Typed Values With the Use of Parameters
Subroutines can be created to return typed values with the use of parameters. These can be referred to as methods or functions in programming languages. These functions or methods accept an input parameter and perform the operation based on the input. Then, after the operation is performed, a specific result will occur.

The following is an example of a subroutine created to return a typed value with the use of a parameter. In this example, the add_numbers function takes the two arguments x and y, adds them, and then returns the result.

```
def add_numbers(x, y):
    return x + y
```

(SE)(Breakout(s)) and (Citation Type(s))
(6)(H)(iv), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content

Update to Content Accepted by SRP

Subroutines and Data (Slides 3-6, 24-28),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21650

Description of the specific location and hyperlink to the exact location of the proposed new content
Subroutines and Data Student Handout-Subroutines,
https://files.icevonline.com/html/CEV71517_V2_HTML/CEV71517_V2_HTML_Student_Handout_-_Subroutines.htm

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Subroutines

- Are named blocks of code that can be called multiple times from different parts of a program
- Are also known as functions or methods
- Are an essential part of modular programming
- Can help to simplify the design and organization of a program

3

Subroutines Example

- Includes:
  - defined subroutine 'fahrenheit_to_celsius' takes a single argument 'tfhrenheit' (the temperature in Fahrenheit) and returns the temperature in Celsius
    - formula for converting Fahrenheit (F) to Celsius (C) is '(F-32)*5/9

4

Subroutines Example

- Includes:
  - next, prompt the user to enter a temperature in F using the 'input' function and convert it to a floating-point number using the 'float' function
  - then, call the 'Fahrenheit_to_Celsius' subroutine to convert the temperature to C and store the result in the variable 'Celsius'

5

Subroutines Example

- Includes:
  - lastly, use the 'print' function to display the original temperature in F and the converted temperature in C to the user

6

Update to Content Accepted by SRP

Parameters

- Are the variables defined in the function definition
  - for example, the 'sum()' function has a parameter "*numbers"

```
def sum(*numbers):
    total = 0
    for n in numbers:
        total += n
    return total

# We can call the function with any number of arguments
print(sum(1, 2, 3, 4)) # Output: 10
print(sum(10))         # Output: 10 (sum of a single number)
print(sum(2, 4, 6, 8)) # Output: 20
```

24

Arguments

- Are the values passed into the function when it is called
  - for example, parameter, '*numbers", means it can accept any numbers or arguments

```
def sum(*numbers):
    total = 0
    for n in numbers:
        total += n
    return total

# We can call the function with any number of arguments
print(sum(1, 2, 3, 4)) # Output: 10
print(sum(10))         # Output: 10 (sum of a single number)
print(sum(2, 4, 6, 8)) # Output: 20
```

25

Subroutines Without Return Values Example

- Includes:
  - the 'say_hello()' subroutine does not take any arguments or parameters and does not return a value
  - instead, it prints the string '"Hello, world!"' to the console when it is called

```
def say_hello():
    print("Hello, world!")

# Call the subroutine
say_hello()  # Output: Hello, world!
```

28

Function

- Executes to add up all the numbers passed into the arguments and returns the total as an output
- Can be called with any number of arguments, and it will always return the sum of all the arguments

```
# We can call the function with any number of arguments
print(sum(1, 2, 3, 4)) # Output: 10
print(sum(10))         # Output: 10 (sum of a single number)
print(sum(2, 4, 6, 8)) # Output: 20
```

26

Creating Subroutines Without Return Values

- Is similar to creating subroutines that return values
  - instead of using the 'return' statement to return a value, the function can be used to perform some of the actions

27

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Subroutines

Subroutines are a way for a user to organize their code by breaking the code down into smaller pieces. The pieces of code are written once and then can be called mutiple times from various parts of a program. Subroutines enhance a code's readability, maintainability and reusability.

Parameters are the variables defined in the function definition. For example, the 'sum()' function has a parameter "numbers". Arguments are the values passed into the function when it is called. For example, parameter, "*numbers", means it can accept any numbers or arguments.

```
def sum(*numbers):
      total = 0
      for n in numbers:
         total += n
      return total

# We can call the function with any number of arguments
print(sum(1, 2, 3, 4)) # Output: 10
print(sum(10))         # Output: 10 (sum of a single number)
print(sum(2, 4, 6, 8)) # Output: 20
```

Return Typed Values Without the Use of Arguments
Subroutines can be created to return typed values without the use of arguments. This can be referred to as parameterless functions. Parameterless functions do not take any parameters, but they do perform a task and then return a result. The result of this function can be a data type.

The following is an example of a subroutine created to return a typed value without the use of an argument. In this example, the generate_random_number function does not have an argument but will return a random number between 1 and 50.

```
import random

def generate_random_number():
    return random.randint(1, 50)

# Call the function and get the result
random_number = generate_random_number()
print("Random Number:", random_number)
```

Return Typed Values With the Use of Arguments

Subroutines can be created to return typed values with the use of arguments. This is often referred to as methods or functions in programming languages. The function will accept an input parameter and perform the operation based on the input. Then, after the operation is performed, a specific result will occur.

The following is an example of a subroutine created to return a typed value with the use of an argument. In this example, the subtract_numbers function takes the two arguments x and y, subtracts them, and then returns the result.

```
def subtract_numbers(x, y):
    return x - y

# Call the function with arguments and get the result
result = subtract_numbers(3, 8)
print("Result:", result)
```

Return Typed Values Without the Use of Parameters
Subroutines can be created to return typed values without the use of parameters. These subroutines are functions or methods that do not take input arguments but will produce a result.

The following is an example of a subroutine created to return a typed value without the use of a parameter. In this example, the get_current_year function does not have an argument but still returns the year with the datetime module.

```
import datetime

def get_current_year():
    return datetime.datetime.now().year

# Call the function and get the result
current_year = get_current_year()
print("Current Year:", current_year)
```

Return Typed Values With the Use of Parameters
Subroutines can be created to return typed values with the use of parameters. These can be referred to as methods or functions in programming languages. These functions or methods accept an input parameter and perform the operation based on the input. Then, after the operation is performed, a specific result will occur.

The following is an example of a subroutine created to return a typed value with the use of a parameter. In this example, the add_numbers function takes the two arguments x and y, adds them, and then returns the result.

```
def add_numbers(x, y):
    return x + y

# Call the function with arguments and get the result
result = add_numbers(7, 9)
print("Result:", result)
```

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(6)(K)(iii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Numeric and Nonnumeric Data (Slides 4-10),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Student Handout-ASCII Conversion Chart,
https://files.icevonline.com/html/CEV71519_TXP24/CEV71519_TXP24_Student_Handout_-_ASCII_Conversion_Chart.htm

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-ASCII Translation and Binary Conversions,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_ASCII_Translation_and_Binary_Conversions.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

Update to Content Accepted by SRP

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Numeric Data

- Is comprised of data points which are quantifiable, meaning mathematical calculations can be performed
- Is numerical (quantitative)
  - discrete which are only particular numbers
  - continuous which are any numeric value
- May include:
  - age
  - height
  - weight
  - test scores

CEV 4

4

Numeric Data

- Is based on meanings derived from numbers
- Is a result of collection, numerical and standardized data available
- Is an analysis of data conducted through mathematical operations, statistics and visualization diagrams

CEV 5

5

Nonnumeric Data

- Is data which cannot be manipulated using basic mathematical operations, but rather making use of symbols and letters
- Is categorical (qualitative)
  - nominal, which are named categories
  - ordinal, which are categories with an implied order
- May include:
  - name
  - degree type
  - type of pet

CEV 6

6

Nonnumeric Data

- Is based on meanings expressed through words
- Is a result of collection, non-standardized data which must be classified into categories
- Includes analysis which is conducted through the use of conceptualization

CEV 7

7

ASCII

- Stands for the "American Standard Code for Information Interchange"
- Is used to represent character data
- Requires seven bits for each character
  - extended ASCII is a superset of ASCII and requires eight bits for each character
    - provides codes for 256 characters, double of ASCII

CEV 8

8

Unicode

- Is an extension of ASCII
  - allows programmers to include a multitude of symbols and characters in all languages
- Requires 16 bits for each character
  - enables Unicode to provide codes for 65,000 characters
- Uses hexadecimal (hex) rather than binary in order to produce many combinations

Data Details: The word "Hello" has a Unicode of U+0048U+0065U+006CU+006CU+0065.

CEV 9

9

Representing Numeric Data

- Stores numbers in memory by finding a way to represent the character sequence
- Bases strategy used to store numbers on type of numerical data (decimal and binary)
- Encounters problems due to limited memory capabilities including:
  - overflow
    - magnitude of a number exceeds the range allowed by the size
  - numeric value precision
    - number of digits in a number

CEV 10

10

Update to Content Accepted by SRP

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

ASCII Translation & Binary Conversions

ASCII (American Standard Code for Information Interchange) and Unicode are character encoding standards used to represent text in computers. They assign numeric values to characters, allowing computers to store and communicate text in a standardized way.

In ASCII, each character is represented by a 7-bit or 8-bit binary number. Unicode extends this concept by providing a unique numeric code for every character, regardless of the platform, program or language. Unicode typically uses 16 bits (or more) for each character.

Here is a basic explanation of how binary representation can be converted to ASCII and Unicode:

Numeric Conversions:
Numeric data is typically represented in binary using ASCII or Unicode by encoding the individual digits.

ASCII:
Each digit has a unique ASCII code. For instance:

- ASCII code for '0' is 48 (in decimal), which is 00110000 in binary.
- ASCII code for '1' is 49 (in decimal), which is 00110001 in binary.

The binary representation of the numeric digit '0' in ASCII is 00110000, and for '1', it's 00110001, and so on.

Unicode:
Unicode represents numeric characters using code points. For example:

- Unicode code point for '0' is 48 (decimal), which is 0000000000110000 in binary.
- Unicode code point for '1' is 49 (decimal), which is 0000000000110001 in binary.

The binary representation of the numeric digit '0' in Unicode is 0000000000110000, and for '1', it's 0000000000110001.

Nonnumeric Conversions:
ASCII:
ASCII represents characters using 7 or 8 bits. This is an example of using the ASCII representation of the letter 'A':

- ASCII code for 'A' is 65.
- In binary, 65 is represented as 01000001 (8 bits).

The binary representation of the nonnumeric character 'A' in ASCII is 01000001.

Unicode:
Unicode can use various bit lengths, considering a common representation with 16 bits. Using the Unicode code point for 'A':

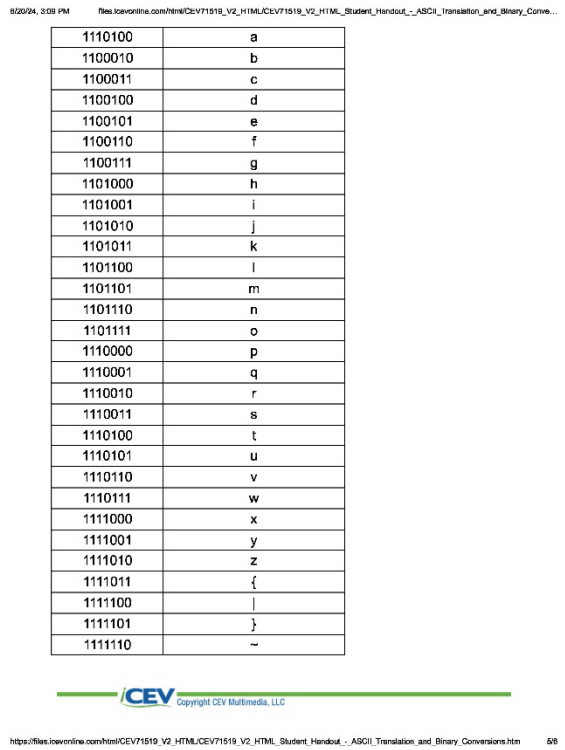
- Unicode code point for 'A' is 65.
- In binary, 65 (in 16 bits) is represented as 0000000001000001.

The binary representation of the nonnumeric character 'A' in Unicode (using 16 bits) is 0000000001000001.

Binary Code	Character Represented
100000	Space
100001	!
100010	"
100011	#
100100	$
100101	%
100110	&
100111	'
101000	(
101001	)
101010	*
101011	+
101100	,
101101	-
101110	.
101111	/
110000	0
110001	1
110010	2
110011	3
110100	4
110101	5
110110	6
110111	7
111000	8
111001	9
111010	:
111011	;
111100	<
111101	=
111110	>
111111	?

1000000	@
1000001	A
1000010	B
1000011	C
1000100	D
1000101	E
1000110	F
1000111	G
1001000	H
1001001	I
1001010	J
1001011	K
1001100	L
1001101	M
1001110	N
1001111	O
1010000	P
1010001	Q
1010010	R
1010011	S
1010100	T
1010101	U
1010110	V
1010111	W
1011000	X
1011001	Y
1011010	Z
1011011	[
1011100	\
1011101	]
1011110	^
1011111	_
1100000	`

Update to Content Accepted by SRP



6/20/24, 3:09 PM files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_ASCII_Translation_and_Binary_Conve...

1110100	a
1100010	b
1100011	c
1100100	d
1100101	e
1100110	f
1100111	g
1101000	h
1101001	i
1101010	j
1101011	k
1101100	l
1101101	m
1101110	n
1101111	o
1110000	p
1110001	q
1110010	r
1110011	s
1110100	t
1110101	u
1110110	v
1110111	w
1111000	x
1111001	y
1111010	z
1111011	{
1111100	|
1111101	}
1111110	~

(SE)(Breakout(s)) and (Citation Type(s))
(6)(K)(iv), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content
Numeric and Nonnumeric Data (Slides 4-10),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Student Handout-ASCII Conversion Chart,
https://files.icevonline.com/html/CEV71519_TXP24/CEV71519_TXP24_Student_Handout_-_ASCII_Conversion_Chart.htm

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-ASCII Translation and Binary Conversions,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_ASCII_Translation_and_Binary_Conversions.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Numeric Data

- Is comprised of data points which are quantifiable, meaning mathematical calculations can be performed
- Is numerical (quantitative)
  - discrete which are only particular numbers
  - continuous which are any numeric value
- May include:
  - age
  - height
  - weight
  - test scores

CEV

4

Numeric Data

- Is based on meanings derived from numbers
- Is a result of collection, numerical and standardized data available
- Is an analysis of data conducted through mathematical operations, statistics and visualization diagrams

CEV

5

Nonnumeric Data

- Is data which cannot be manipulated using basic mathematical operations, but rather making use of symbols and letters
- Is categorical (qualitative)
  - nominal, which are named categories
  - ordinal, which are categories with an implied order
- May include:
  - name
  - degree type
  - type of pet

CEV

6

Nonnumeric Data

- Is based on meanings expressed through words
- Is a result of collection, non-standardized data which must be classified into categories
- Includes analysis which is conducted through the use of conceptualization

CEV

7

ASCII

- Stands for the "American Standard Code for Information Interchange"
- Is used to represent character data
- Requires seven bits for each character
  - extended ASCII is a superset of ASCII and requires eight bits for each character
    - provides codes for 256 characters, double of ASCII

CEV

8

Unicode

- Is an extension of ASCII
  - allows programmers to include a multitude of symbols and characters in all languages
- Requires 16 bits for each character
  - enables Unicode to provide codes for 65,000 characters
- Uses hexadecimal (hex) rather than binary in order to produce many combinations

Data Details: The word "Hello" has a Unicode of U+0048U+0065U+006CU+006CU+0065.

CEV

9

Representing Numeric Data

- Stores numbers in memory by finding a way to represent the character sequence
- Bases strategy used to store numbers on type of numerical data (decimal and binary)
- Encounters problems due to limited memory capabilities including:
  - overflow
    - magnitude of a number exceeds the range allowed by the size
  - numeric value precision
    - number of digits in a number

CEV

10

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

ASCII Translation & Binary Conversions

ASCII (American Standard Code for Information Interchange) and Unicode are character encoding standards used to represent text in computers. They assign numeric values to characters, allowing computers to store and communicate text in a standardized way.

In ASCII, each character is represented by a 7-bit or 8-bit binary number. Unicode extends this concept by providing a unique numeric code for every character, regardless of the platform, program or language. Unicode typically uses 16 bits (or more) for each character.

Here is a basic explanation of how binary representation can be converted to ASCII and Unicode:

Numeric Conversions:
Numeric data is typically represented in binary using ASCII or Unicode by encoding the individual digits.

ASCII:
Each digit has a unique ASCII code. For instance:

- ASCII code for '0' is 48 (in decimal), which is 00110000 in binary.
- ASCII code for '1' is 49 (in decimal), which is 00110001 in binary.

The binary representation of the numeric digit '0' in ASCII is 00110000, and for '1', it's 00110001, and so on.

Unicode:
Unicode represents numeric characters using code points. For example:

- Unicode code point for '0' is 48 (decimal), which is 0000000000110000 in binary.
- Unicode code point for '1' is 49 (decimal), which is 0000000000110001 in binary.

The binary representation of the numeric digit '0' in Unicode is 0000000000110000, and for '1', it's 0000000000110001.

Nonnumeric Conversions:
ASCII:
ASCII represents characters using 7 or 8 bits. This is an example of using the ASCII representation of the letter 'A':

- ASCII code for 'A' is 65.
- In binary, 65 is represented as 01000001 (8 bits).

The binary representation of the nonnumeric character 'A' in ASCII is 01000001.

Unicode:
Unicode can use various bit lengths, considering a common representation with 16 bits. Using the Unicode code point for 'A':

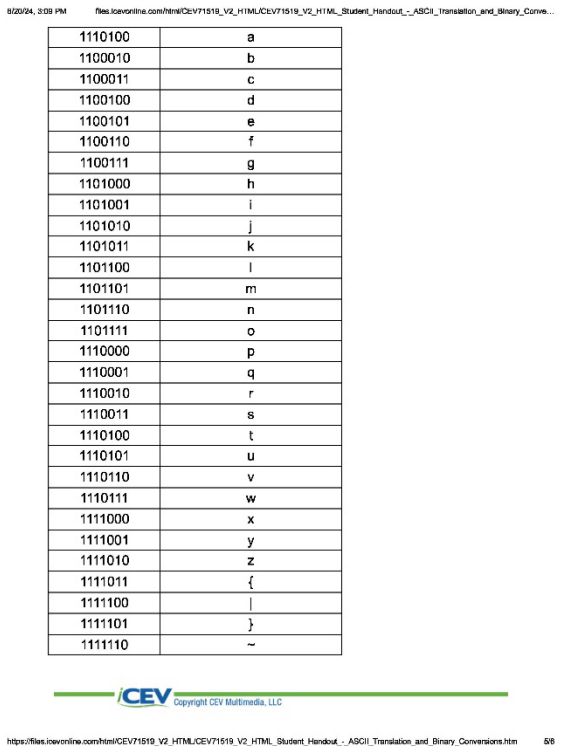
- Unicode code point for 'A' is 65.
- In binary, 65 (in 16 bits) is represented as 0000000001000001.

The binary representation of the nonnumeric character 'A' in Unicode (using 16 bits) is 0000000001000001.

Binary Code	Character Represented
100000	Space
100001	!
100010	"
100011	#
100100	$
100101	%
100110	&
100111	'
101000	(
101001	)
101010	*
101011	+
101100	,
101101	-
101110	.
101111	/
110000	0
110001	1
110010	2
110011	3
110100	4
110101	5
110110	6
110111	7
111000	8
111001	9
111010	:
111011	;
111100	<
111101	=
111110	>
111111	?

1000000	@
1000001	A
1000010	B
1000011	C
1000100	D
1000101	E
1000110	F
1000111	G
1001000	H
1001001	I
1001010	J
1001011	K
1001100	L
1001101	M
1001110	N
1001111	O
1010000	P
1010001	Q
1010010	R
1010011	S
1010100	T
1010101	U
1010110	V
1010111	W
1011000	X
1011001	Y
1011010	Z
1011011	[
1011100	\
1011101	]
1011110	^
1011111	_
1100000	`

Update to Content Accepted by SRP



8/20/24, 3:09 PM files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_ASCII_Translation_and_Binary_Conve...

1110100	a
1100010	b
1100011	c
1100100	d
1100101	e
1100110	f
1100111	g
1101000	h
1101001	i
1101010	j
1101011	k
1101100	l
1101101	m
1101110	n
1101111	o
1110000	p
1110001	q
1110010	r
1110011	s
1110100	t
1110101	u
1110110	v
1110111	w
1111000	x
1111001	y
1111010	z
1111011	{
1111100	|
1111101	}
1111110	~

(SE)(Breakout(s)) and (Citation Type(s))
(6)(M)(ii), Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Activity-ASCII Translation and Binary Conversion,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Activity01

Description of the specific location and hyperlink to the exact location of the proposed new content
Activity-ASCII Translation and Binary Conversion,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity01
This Activity is found in the Numeric & Nonnumeric Data lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Activity - ASCII Translation & Binary Conversions

Numeric & Nonnumeric Data

1 of 1

Activity Overview:

You will use the ASCII Conversion Chart Student Handout to translate words and characters into their binary counterparts, then convert a variety of numbers from binary to decimal form and decimal numbers back to their binary form.

Directions:

1. In the first section, using the ASCII Table, translate the following characters and words to binary.
2. In the second section, convert each binary number to its decimal form.
3. In the third section, convert each decimal number to its binary form.
4. Once complete, submit your Activity.

Section 1:

Translate the characters and words to binary.

1. Q

2. q

3. R

4. S

5. a

6. To

7. In

8. ASCII

9. Binary

Section 2:

Convert the binary number to decimal form.

1. 10101

2. 110110

3. 1011001

4. 10100101

Update to Content Accepted by SRP

Section 3:
Convert the decimal number to binary form.

1. 17

2. 45

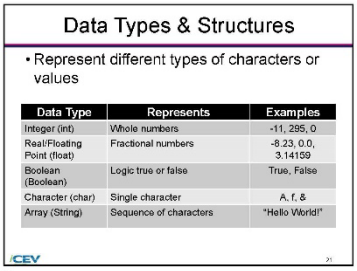
3. 112

4. 191

Review

©2024 - All Rights Reserved. (WN1MDWK00005C)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

6/21/2024



Data Types & Structures

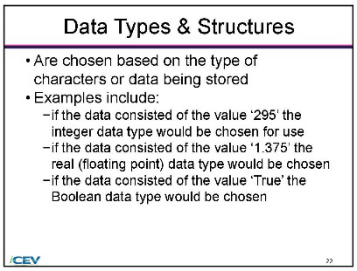
- Represent different types of characters or values

Data Type	Represents	Examples
Integer (int)	Whole numbers	-11, 295, 0
Real/Floating Point (float)	Fractional numbers	-8.23, 0.0, 3.14159
Boolean (Boolean)	Logic true or false	True, False
Character (char)	Single character	A, f, &
Array (String)	Sequence of characters	"Hello World!"

CEV

21

21



Data Types & Structures

- Are chosen based on the type of characters or data being stored
- Examples include:
  - if the data consisted of the value '295' the integer data type would be chosen for use
  - if the data consisted of the value '1.375' the real (floating point) data type would be chosen
  - if the data consisted of the value 'True' the Boolean data type would be chosen

CEV

22

22

1

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

ASCII Translation & Binary Conversions

Activity Overview:
You will use the **ASCII Translation & Binary Conversions Student Handout** to translate words and characters into their binary counterparts, then convert a variety of numbers from binary to decimal form and decimal numbers back to their binary form.

Directions:

1. In the first section, using the ASCII Table, translate the following characters and words to binary.
2. In the second section, convert each binary number to its decimal form.
3. In the third section, convert each decimal number to its binary form.
4. In the fourth section, fill in the chart with the correct binary numbers in order from 0 to 15.
5. Turn in your activity according to your instructor's directions.

Section 1:
Translate the characters and words to binary.

1. Q
2. q
3. R
4. S
5. a
6. To
7. In
8. ASCII
9. Binary

Section 2:
Convert the binary number to decimal form.

1. 10101
2. 110110
3. 1011001
4. 10100101

Section 3:
Convert the decimal number to binary form.

1. 17
2. 45
3. 112
4. 191

Section 4:
Fill in the chart with the correct binary numbers in order from 0 to 15.

Decimal	Binary
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

CEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(6)(N)(vii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content

Update to Content Accepted by SRP

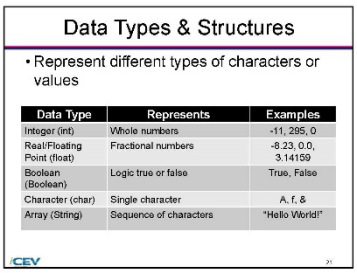
Numeric and Nonnumeric Data (Slides 21-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

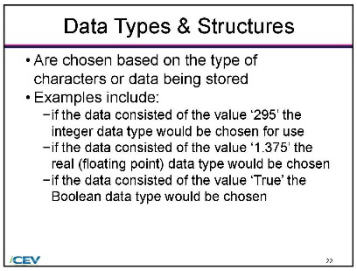
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/21/2024



21



22

1

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85,
92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95,
89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(6)(N)(viii), Narrative

Description of the specific location and hyperlink to the exact location of currently adopted content

Update to Content Accepted by SRP

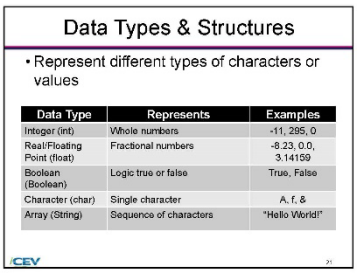
Numeric and Nonnumeric Data (Slides 21-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

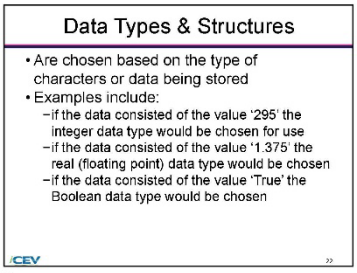
Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/21/2024





1

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

iCEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(6)(N)(ix), Narrative

Update to Content Accepted by SRP

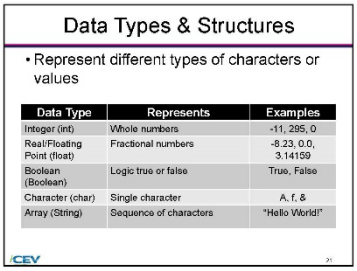
Description of the specific location and hyperlink to the exact location of currently adopted content
Numeric and Nonnumeric Data (Slides 21-22),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

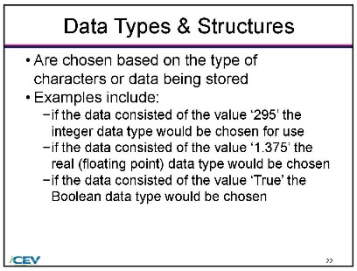
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/21/2024



21



22

1

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

iCEV Copyright CEV Multimedia, LLC

(SE)(Breakout(s)) and (Citation Type(s))
(6)(O)(i), Narrative & Activity

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Objects and Variables (Slides 21-22, 28-29),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21649

Project-Objects and Scope,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21649/CEV71516_Project01

Description of the specific location and hyperlink to the exact location of the proposed new content
Objects and Variables (Slides 21-22, 28-29),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22306
In the Objects & Variables PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.
Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Objects & Variables lesson beneath the Instructional Materials heading.

Activity-Primitives and Objects,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22306/CEV71516_V2_Activity02
This Activity is found in the Objects & Variables lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Variables

- Are named locations in memory used to store a value
- Can be used to store any type of data, such as numbers, strings, lists and more
- Are assigned using the "=" operator, and their value can be accessed by referencing their name

21

Variables Examples

- Can include:
  - "x" is assigned the value "10"
  - "y" is assigned the value "hello"
  - "z" is assigned the value "[1,2,3]"

```
x = 10
y = "hello"
z = [1, 2, 3]
```

22

Primitive

- Types in Python are simple data types that hold a single value
- Are defined without any special methods and are usually the basic building blocks of more complex data structures

```
x = 10  # Integer
y = 10.5  # Floating-point number
z = "hello"  # String
```

28

Objects

- Are more complex data structures that can have multiple values and methods
- Have their own state, which is represented by its instance variables
- Have their own behavior, which is represented by its method

29

Project - Objects & Scope

Objects & Variables

1 of 1

Project Overview:
You will write a program which uses at least two classes and examples of both local and global scope.

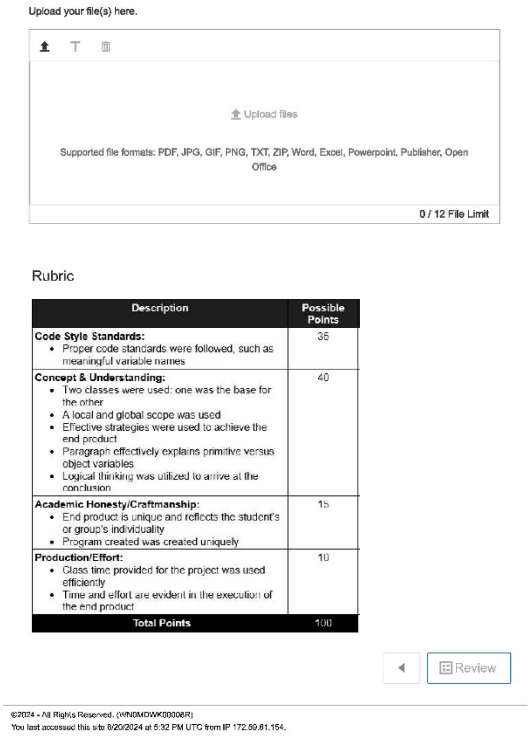
Directions:

1. Plan a program using object-oriented design. You will use at least two classes, and one class must be the base for another class. You may have more than two classes if you wish. Consider what you would like to organize.
2. Open a Python interpreter. Write a comment outline of your program.
3. Write the definition of your two classes. Remember, one class must use another class as its base.
4. Write the main program which creates at least one instance of the class which has a base class.
5. Make sure your program has an example of both local and global scope. Add a comment to the line to identify a local variable and global variable. You only need to do this once for each type of scope.
6. In three to five sentences, explain if the variables used in the program contain primitives, objects or both.
7. Run your program and fix any errors, if necessary.
8. Once complete, upload a document which contains your program in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Write your paragraph here.

0 / 10000 Word Limit

Update to Content Accepted by SRP



Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Code Style Standards: • Proper code standards were followed, such as meaningful variable names	35
Concept & Understanding: • Two classes were used: one was the base for the other • A local and global scope was used • Effective strategies were used to achieve the end product • Paragraph effectively explains primitive versus object variables • Logical thinking was utilized to arrive at the conclusion	40
Academic Honesty/Craftmanship: • End product is unique and reflects the student's or group's individuality • Program created was created uniquely	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	10
Total Points	100

Review

©2024 - All Rights Reserved. (WN0MDWK00006R)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.61.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Variables

- Are named locations in memory used to store a value
- Can be used to store any type of data, such as numbers, strings, lists and more
- Are assigned using the "=" operator, and their value can be accessed by referencing their name

21

Variables Examples

- Can include:
  - "x" is assigned the value "10"
  - "y" is assigned the value "hello"
  - "z" is assigned the value "[1,2,3]"

```
x = 10
y = "hello"
z = [1, 2, 3]
```

22

Primitive

- Are basic data types not derived from any other data type
- Data types include:
  - integers
    - age (17)
  - float
    - radius of a circle (6.5)
  - strings
    - name
  - Boolean
    - True or False

NOTE: Use the Coding Examples Student Handout for references.

28

Primitive

- Types in Python are simple data types that hold a single value
- Are defined without any special methods and are usually the basic building blocks of more complex data structures

```
x = 10  # Integer
y = 10.5  # Floating-point number
z = "hello"  # String
```

NOTE: Use the Coding Examples Student Handout for references.

29

Coding Examples

Object Examples:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def description (self):
        return f"{self.year} {self.make} {self.model}"

# Create an object of the Car class
my_car = Car("Tesla", "Model S", 2022)

#Access the attributes of the object
print(my_car.make) # Tesla
print(my_car.model) # Model S
print(my_car.year) # 2022

# Call the method of the object
print(my_car.description()) # 2022 Tesla Model S
```

Object-Oriented Design Example 1:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.speed = 0

    def accelerate(self, rate):
        self.speed += rate

    def brake(self, rate):
        self.speed -= rate

    def honk(self):
```

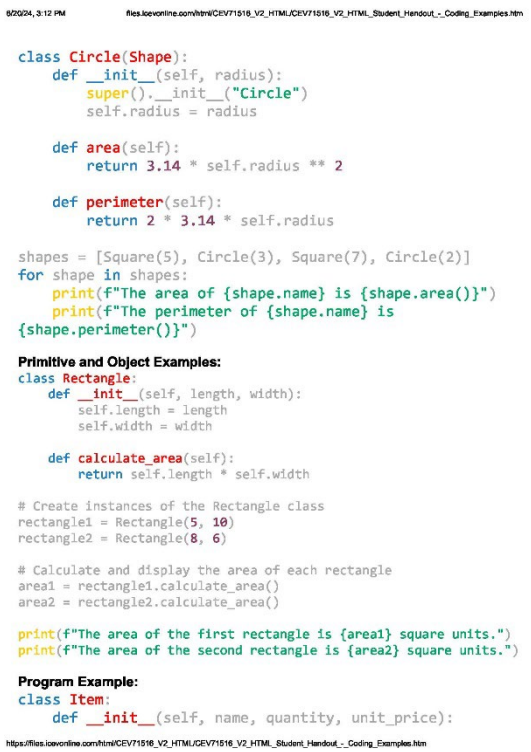


```
        print(f"{self.make} {self.model} honks!")

class ElectricCar(Car):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.battery_level = 100

    def honk(self):
        print(f"{self.make} {self.model} beeps!")

my_car = Car("Toyota", "Camry", 2020)
my_car.accelerate(10)
my_car.honk() # Output: Toyota Camry honks!

my_electric_car = ElectricCar("Tesla", "Model S", 2021)
my_electric_car.honk() # Output: Tesla Model S beeps!
```

Object-Oriented Design Example 2:

```
class Shape:
    def __init__(self, name):
        self.name = name

    def area(self):
        pass

    def perimeter(self):
        pass

class Square(Shape):
    def __init__(self, length):
        super().__init__("Square")
        self.length = length

    def area(self):
        return self.length ** 2

    def perimeter(self):
        return 4 * self.length
```

Update to Content Accepted by SRP



6/20/24, 3:12 PM files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
class Circle(Shape):
    def __init__(self, radius):
        super().__init__("Circle")
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

    def perimeter(self):
        return 2 * 3.14 * self.radius

shapes = [Square(5), Circle(3), Square(7), Circle(2)]
for shape in shapes:
    print(f"The area of {shape.name} is {shape.area()}")
    print(f"The perimeter of {shape.name} is
{shape.perimeter()}")
```

Primitive and Object Examples:

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

# Create instances of the Rectangle class
rectangle1 = Rectangle(5, 10)
rectangle2 = Rectangle(8, 6)

# Calculate and display the area of each rectangle
area1 = rectangle1.calculate_area()
area2 = rectangle2.calculate_area()

print(f"The area of the first rectangle is {area1} square units.")
print(f"The area of the second rectangle is {area2} square units.")
```

Program Example:

```
class Item:
    def __init__(self, name, quantity, unit_price):
```

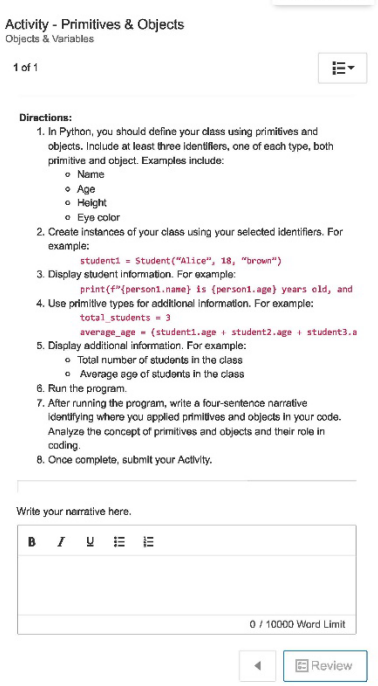
https://files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm 3/5

6/20/24, 3:12 PM files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
        self.name = name
        self.quantity = quantity
        self.unit_price = unit_price

    def total_price(self):
        return self.quantity * self.unit_price

class GroceryInventory:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)

    def remove_item(self, item_name):
        for item in self.items:
            if item.name == item_name:
                self.items.remove(item)
                break

    def show_inventory(self):
        total = 0
        print("Name\tQuantity\tUnit Price\tTotal Price")
        for item in self.items:
            print(f"
{item.name}\t{item.quantity}\t\t${item.unit_price}\t\t${item.
total_price()}")
            total += item.total_price()
        print(f"\nTotal: ${total}")

inventory = GroceryInventory()

item1 = Item("Bananas", 10, 0.5)
item2 = Item("Apples", 5, 1.0)
item3 = Item("Oranges", 15, 0.75)
```

CEV Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm 4/5



Activity - Primitives & Objects

Objects & Variables

1 of 1

Directions:

1. In Python, you should define your class using primitives and objects. Include at least three identifiers, one of each type, both primitive and object. Examples include:
   - Name
   - Age
   - Height
   - Eye color
2. Create instances of your class using your selected identifiers. For example:
   ```
   student1 = Student("Alice", 18, "brown")
   ```
3. Display student information. For example:
   ```
   print(f"{person1.name} is {person1.age} years old, and
   ```
4. Use primitive types for additional information. For example:
   ```
   total_students = 3
   average_age = {student1.age + student2.age + student3.a
   ```
5. Display additional information. For example:
   - Total number of students in the class
   - Average age of students in the class
6. Run the program.
7. After running the program, write a four-sentence narrative identifying where you applied primitives and objects in your code. Analyze the concept of primitives and objects and their role in coding.
8. Once complete, submit your Activity.

Write your narrative here.

B I U

0 / 10000 Word Limit

Review

(SE)(Breakout(s)) and (Citation Type(s))

(6)(O)(ii), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content

Objects and Variables (Slides 21-22, 28-29),

Update to Content Accepted by SRP

https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21649

Project-Objects and Scope,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21649/CEV71516_Project01

Description of the specific location and hyperlink to the exact location of the proposed new content
Objects and Variables (Slides 21-22, 28-29),
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22306
In the Objects & Variables PowerPoint, go to the slides suggested in the Page Number(s). When the PowerPoint opens, if a menu appears asking "Would you like to resume the presentation from the last slide viewed?" select No.
Student Handout-Coding Examples,
https://files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm
This Student Handout is found in the Objects & Variables lesson beneath the Instructional Materials heading.

Activity-Primitives and Objects,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22306/CEV71516_V2_Activity02
This Activity is found in the Objects & Variables lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Variables

- Are named locations in memory used to store a value
- Can be used to store any type of data, such as numbers, strings, lists and more
- Are assigned using the "=" operator, and their value can be accessed by referencing their name

CEV

21

Variables Examples

- Can include:
  - "x" is assigned the value "10"
  - "y" is assigned the value "hello"
  - "z" is assigned the value "[1,2,3]"

```
x = 10
y = "hello"
z = [1, 2, 3]
```

CEV

22

Primitive

- Types in Python are simple data types that hold a single value
- Are defined without any special methods and are usually the basic building blocks of more complex data structures

```
x = 10  # Integer
y = 10.5  # Floating-point number
z = "hello"  # String
```

CEV

28

Objects

- Are more complex data structures that can have multiple values and methods
- Have their own state, which is represented by its instance variables
- Have their own behavior, which is represented by its method

CEV

29

Update to Content Accepted by SRP

Project - Objects & Scope

Objects & Variables

1 of 1

Project Overview:
You will write a program which uses at least two classes and examples of both local and global scope.

Directions:

1. Plan a program using object-oriented design. You will use at least two classes, and one class must be the base for another class. You may have more than two classes if you wish. Consider what you would like to organize.
2. Open a Python interpreter. Write a comment outline of your program.
3. Write the definition of your two classes. Remember, one class must use another class as its base.
4. Write the main program which creates at least one instance of the class which has a base class.
5. Make sure your program has an example of both local and global scope. Add a comment to the line to identify a local variable and global variable. You only need to do this once for each type of scope.
6. In three to five sentences, explain if the variables used in the program contain primitives, objects or both.
7. Run your program and fix any errors, if necessary.
8. Once complete, upload a document which contains your program in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Write your paragraph here.

0 / 10000 Word Limit

Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Code Style Standards: • Proper code standards were followed, such as meaningful variable names	35
Concept & Understanding: • Two classes were used; one was the base for the other • A local and global scope was used • Effective strategies were used to achieve the end product • Paragraph effectively explains primitive versus object variables • Logical thinking was utilized to arrive at the conclusion	40
Academic Honesty/Craftmanship: • End product is unique and reflects the student's or group's individuality • Program created was created uniquely	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	10
Total Points	100

Review

©2024 - All Rights Reserved. (WN0MDWK00006R)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.61.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

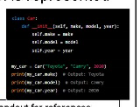
Variables

- Are named locations in memory used to store a value
- Can be used to store any type of data, such as numbers, strings, lists and more
- Are assigned using the "=" operator, and their value can be accessed by referencing their name

21

Objects

- Are more complex data structures that can have multiple values and methods
  - examples: area of a shape (length x width), make, model, year of a vehicle
- Have their own state, which is represented by its instance variables
- Have their own behavior, which is represented by its method



NOTE: Use the Coding Examples Student Handout for references.

30

Variables Examples

- Can include:
  - "x" is assigned the value "10"
  - "y" is assigned the value "hello"
  - "z" is assigned the value "[1,2,3]"

```
x = 10
y = "hello"
z = [1, 2, 3]
```

22

Coding Examples

Object Examples:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def description (self):
        return f"{self.year} {self.make} {self.model}"

# Create an object of the Car class
my_car = Car("Tesla", "Model S", 2022)

#Access the attributes of the object
print(my_car.make) # Tesla
print(my_car.model) # Model S
print(my_car.year) # 2022

# Call the method of the object
print(my_car.description()) # 2022 Tesla Model S
```

Object-Oriented Design Example 1:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.speed = 0

    def accelerate(self, rate):
        self.speed += rate

    def brake(self, rate):
        self.speed -= rate

    def honk(self):
```

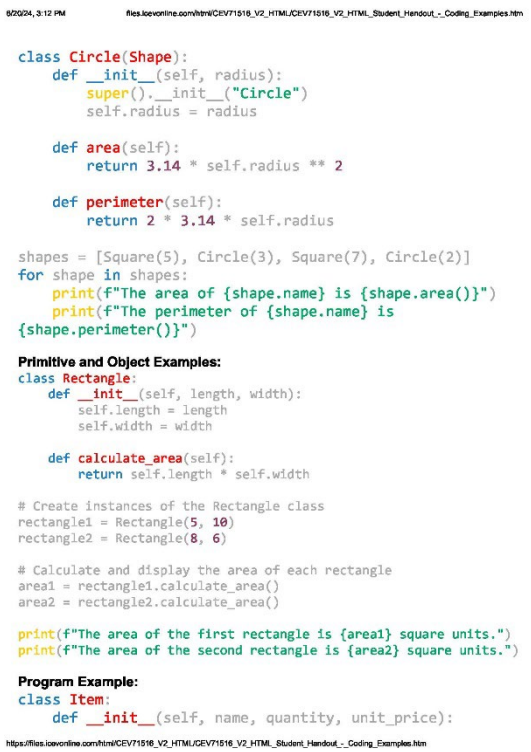


```
        print(f"{self.make} {self.model} honks!")

class ElectricCar(Car):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.battery_level = 100

    def honk(self):
        print(f"{self.make} {self.model} beeps!")

my_car = Car("Toyota", "Camry", 2020)
my_car.accelerate(10)
my_car.honk() # Output: Toyota Camry honks!

my_electric_car = ElectricCar("Tesla", "Model S", 2021)
my_electric_car.honk() # Output: Tesla Model S beeps!
```

Object-Oriented Design Example 2:

```
class Shape:
    def __init__(self, name):
        self.name = name

    def area(self):
        pass

    def perimeter(self):
        pass

class Square(Shape):
    def __init__(self, length):
        super().__init__("Square")
        self.length = length

    def area(self):
        return self.length ** 2

    def perimeter(self):
        return 4 * self.length
```

Update to Content Accepted by SRP



6/20/24, 3:12 PM files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
class Circle(Shape):
    def __init__(self, radius):
        super().__init__("Circle")
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

    def perimeter(self):
        return 2 * 3.14 * self.radius

shapes = [Square(5), Circle(3), Square(7), Circle(2)]
for shape in shapes:
    print(f"The area of {shape.name} is {shape.area()}")
    print(f"The perimeter of {shape.name} is
{shape.perimeter()}")
```

Primitive and Object Examples:

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

# Create instances of the Rectangle class
rectangle1 = Rectangle(5, 10)
rectangle2 = Rectangle(8, 6)

# Calculate and display the area of each rectangle
area1 = rectangle1.calculate_area()
area2 = rectangle2.calculate_area()

print(f"The area of the first rectangle is {area1} square units.")
print(f"The area of the second rectangle is {area2} square units.")
```

Program Example:

```
class Item:
    def __init__(self, name, quantity, unit_price):
```

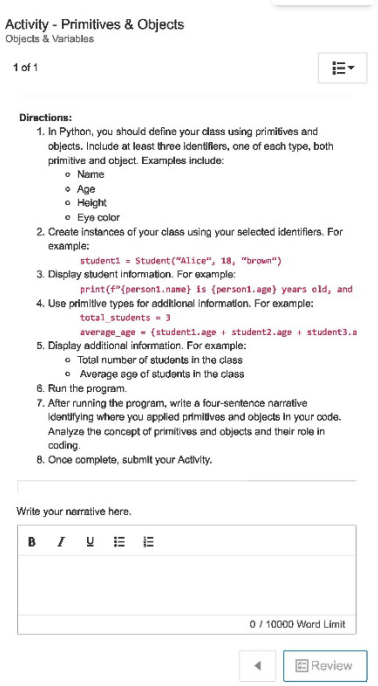
https://files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm 3/5

6/20/24, 3:12 PM files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm

```
        self.name = name
        self.quantity = quantity
        self.unit_price = unit_price

    def total_price(self):
        return self.quantity * self.unit_price

class GroceryInventory:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)

    def remove_item(self, item_name):
        for item in self.items:
            if item.name == item_name:
                self.items.remove(item)
                break

    def show_inventory(self):
        total = 0
        print("Name\tQuantity\tUnit Price\tTotal Price")
        for item in self.items:
            print(f"
{item.name}\t{item.quantity}\t\t${item.unit_price}\t\t${item.
total_price()}")
            total += item.total_price()
        print(f"\nTotal: ${total}")

inventory = GroceryInventory()

item1 = Item("Bananas", 10, 0.5)
item2 = Item("Apples", 5, 1.0)
item3 = Item("Oranges", 15, 0.75)
```

CEV Copyright CEV Multimedia, LLC

https://files.icevonline.com/html/CEV71516_V2_HTML/CEV71516_V2_HTML_Student_Handout_-_Coding_Examples.htm 4/5



Activity - Primitives & Objects

Objects & Variables

1 of 1

Directions:

1. In Python, you should define your class using primitives and objects. Include at least three identifiers, one of each type, both primitive and object. Examples include:
   - Name
   - Age
   - Height
   - Eye color
2. Create instances of your class using your selected identifiers. For example:
   `student1 = Student("Alice", 18, "brown")`
3. Display student information. For example:
   `print(f"{person1.name} is {person1.age} years old, and`
4. Use primitive types for additional information. For example:
   `total_students = 3`
   `average_age = {student1.age + student2.age + student3.a`
5. Display additional information. For example:
   - Total number of students in the class
   - Average age of students in the class
6. Run the program.
7. After running the program, write a four-sentence narrative identifying where you applied primitives and objects in your code. Analyze the concept of primitives and objects and their role in coding.
8. Once complete, submit your Activity.

Write your narrative here.

0 / 10000 Word Limit

Review

(SE)(Breakout(s)) and (Citation Type(s))

(6)(P)(i), Activity

Description of the specific location and hyperlink to the exact location of currently adopted content

Project-Data Types and Structures Handbook,

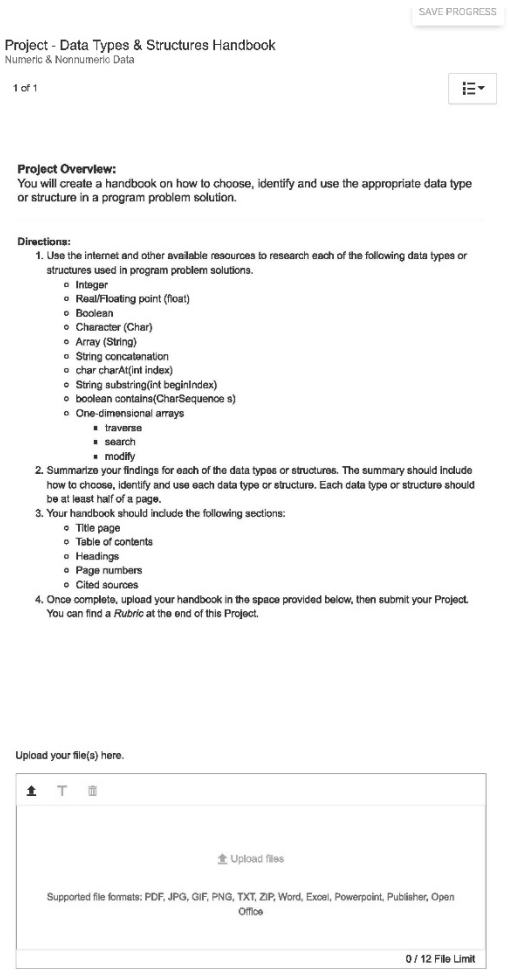
Update to Content Accepted by SRP

https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01

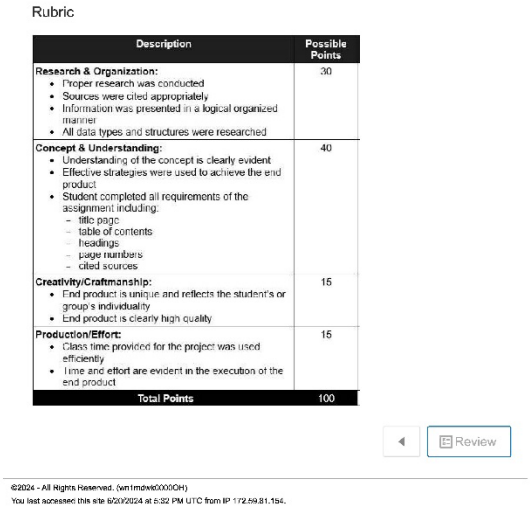
Description of the specific location and hyperlink to the exact location of the proposed new content
Coding Challenge: Calculate String Rotation,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22045/CEV71808_SIM01
Access to the interactive coding environment can be located beneath the Interactive Assignments heading by clicking the link to the Coding Challenge. Once clicked, the link will take you to a page prompting you to click Start. Select Start to view the Coding Challenge in the interactive environment.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



Update to Content Accepted by SRP



Rubric

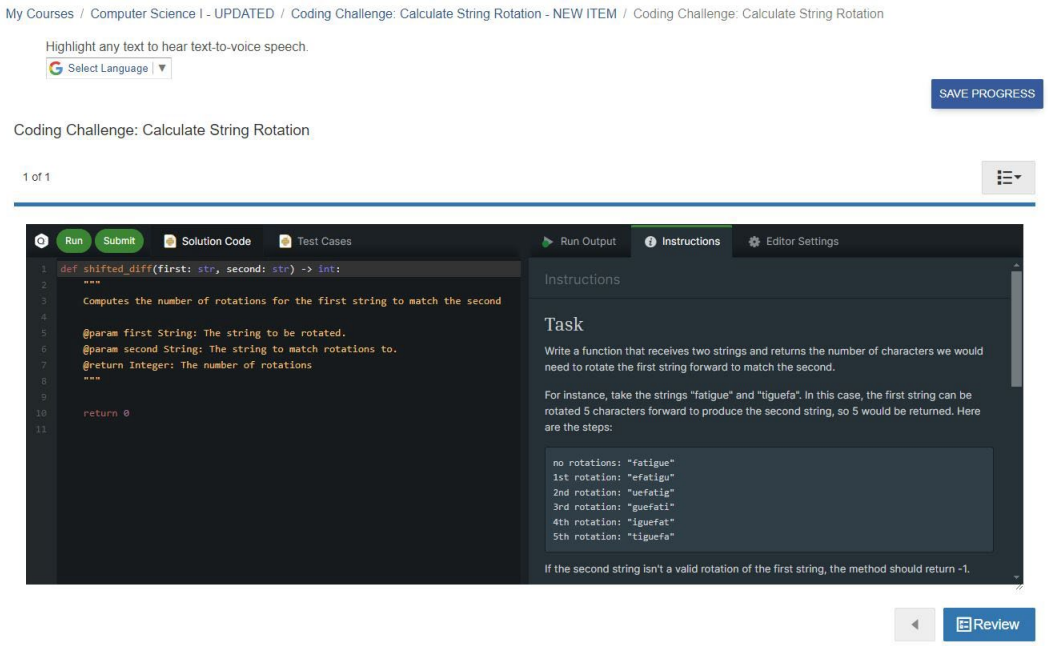
Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwle0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.



(SE)(Breakout(s)) and (Citation Type(s))
(6)(P)(ii), Activity

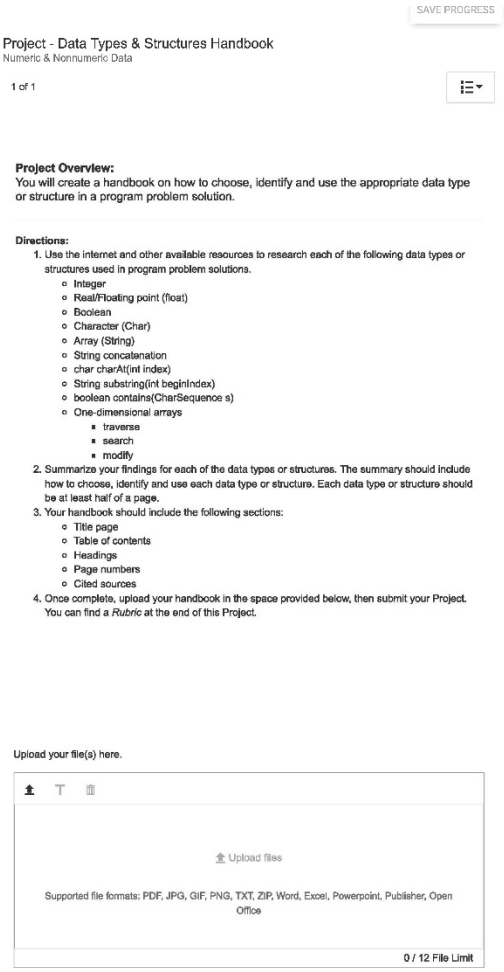
Description of the specific location and hyperlink to the exact location of currently adopted content
Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01

Update to Content Accepted by SRP

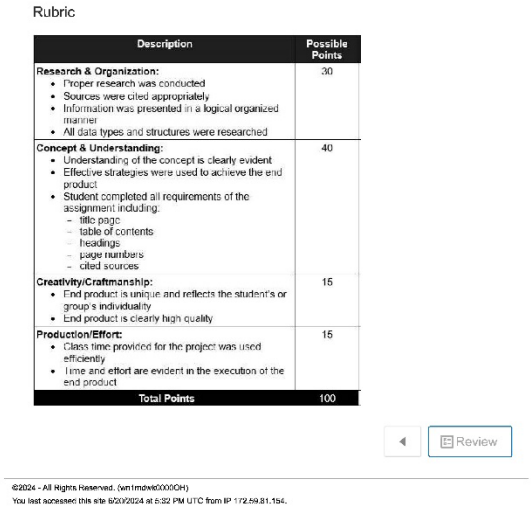
Description of the specific location and hyperlink to the exact location of the proposed new content
Coding Challenge: Capitalize Words,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22050/CEV71813_SIM01
Access to the interactive coding environment can be located beneath the Interactive Assignments heading by clicking the link to the Coding Challenge. Once clicked, the link will take you to a page prompting you to click Start. Select Start to view the Coding Challenge in the interactive environment.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



Update to Content Accepted by SRP



Rubric

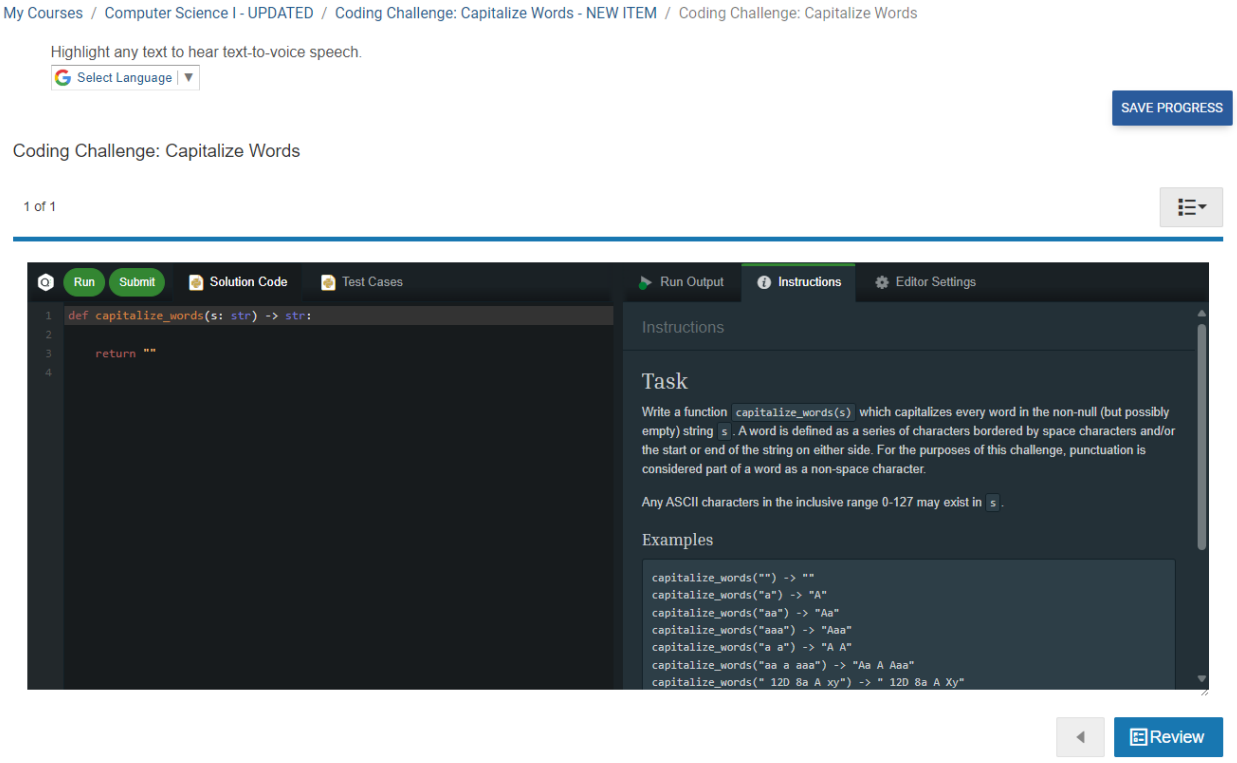
Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwk0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.



(SE)(Breakout(s)) and (Citation Type(s))

(6)(P)(iii), Activity

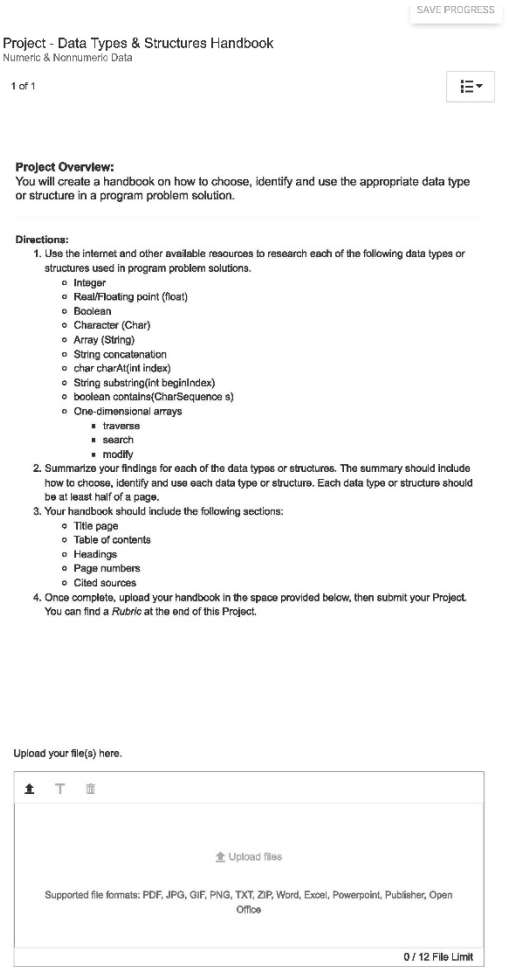
Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01

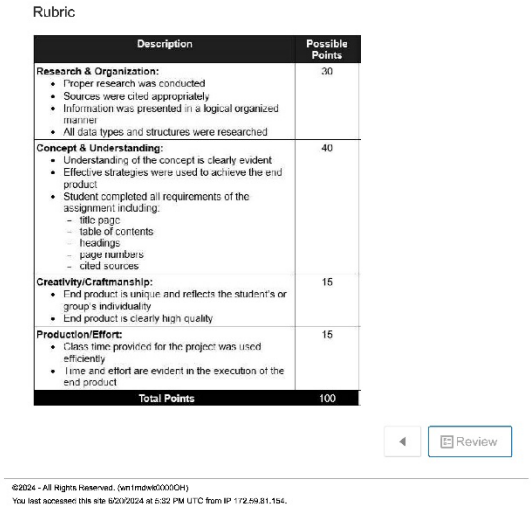
Description of the specific location and hyperlink to the exact location of the proposed new content
Coding Challenge: Calculate String Rotation,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22045/CEV71808_SIM01
Access to the interactive coding environment can be located beneath the Interactive Assignments heading by clicking the link to the Coding Challenge. Once clicked, the link will take you to a page prompting you to click Start. Select Start to view the Coding Challenge in the interactive environment.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



Update to Content Accepted by SRP



Rubric

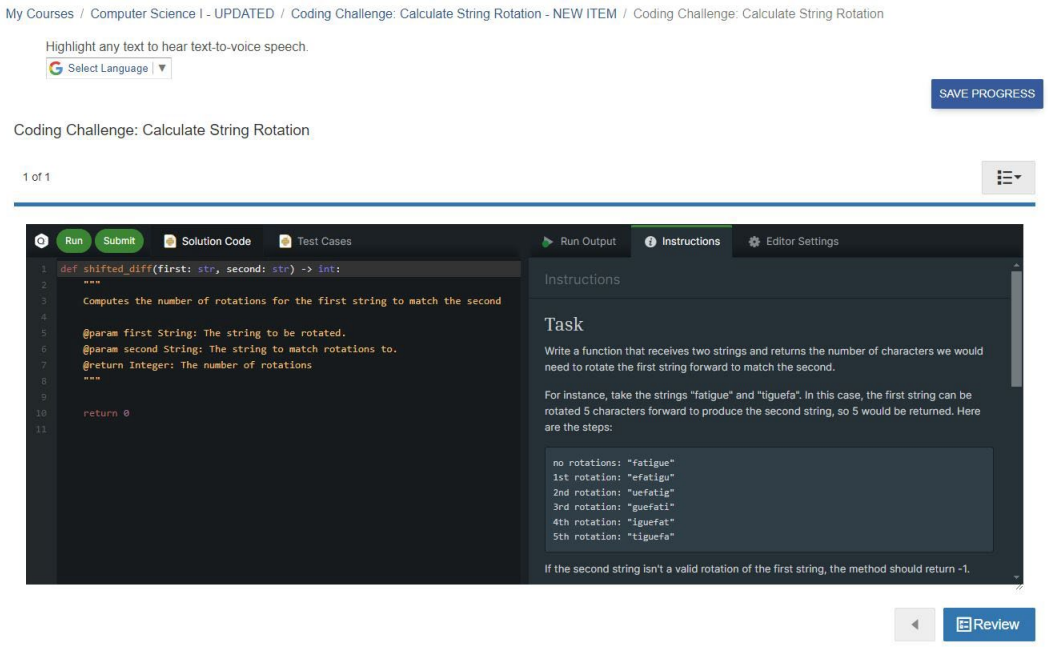
Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdvle0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.



(SE)(Breakout(s)) and (Citation Type(s))

(6)(P)(iv), Activity

Description of the specific location and hyperlink to the exact location of currently adopted content

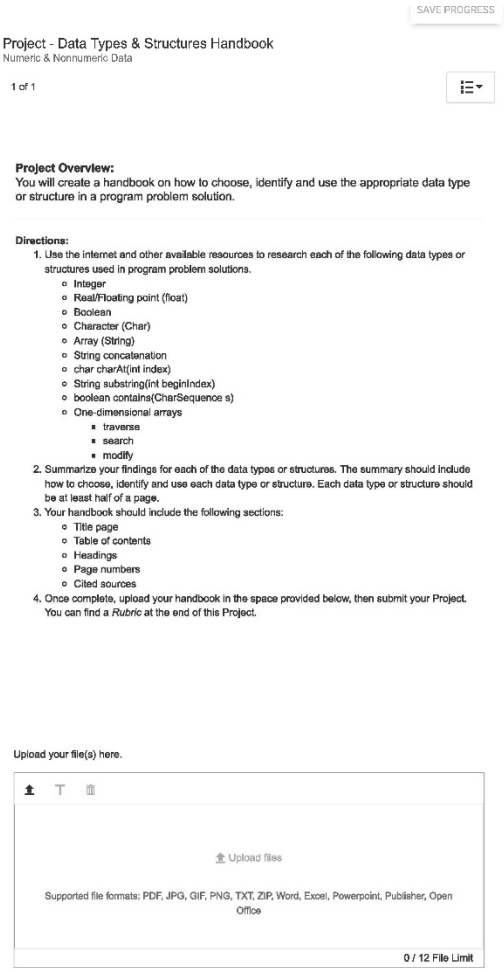
Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of the proposed new content
Coding Challenge: Capitalize Words,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22050/CEV71813_SIM01
Access to the interactive coding environment can be located beneath the Interactive Assignments heading by clicking the link to the Coding Challenge. Once clicked, the link will take you to a page prompting you to click Start. Select Start to view the Coding Challenge in the interactive environment.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.



Update to Content Accepted by SRP

Rubric

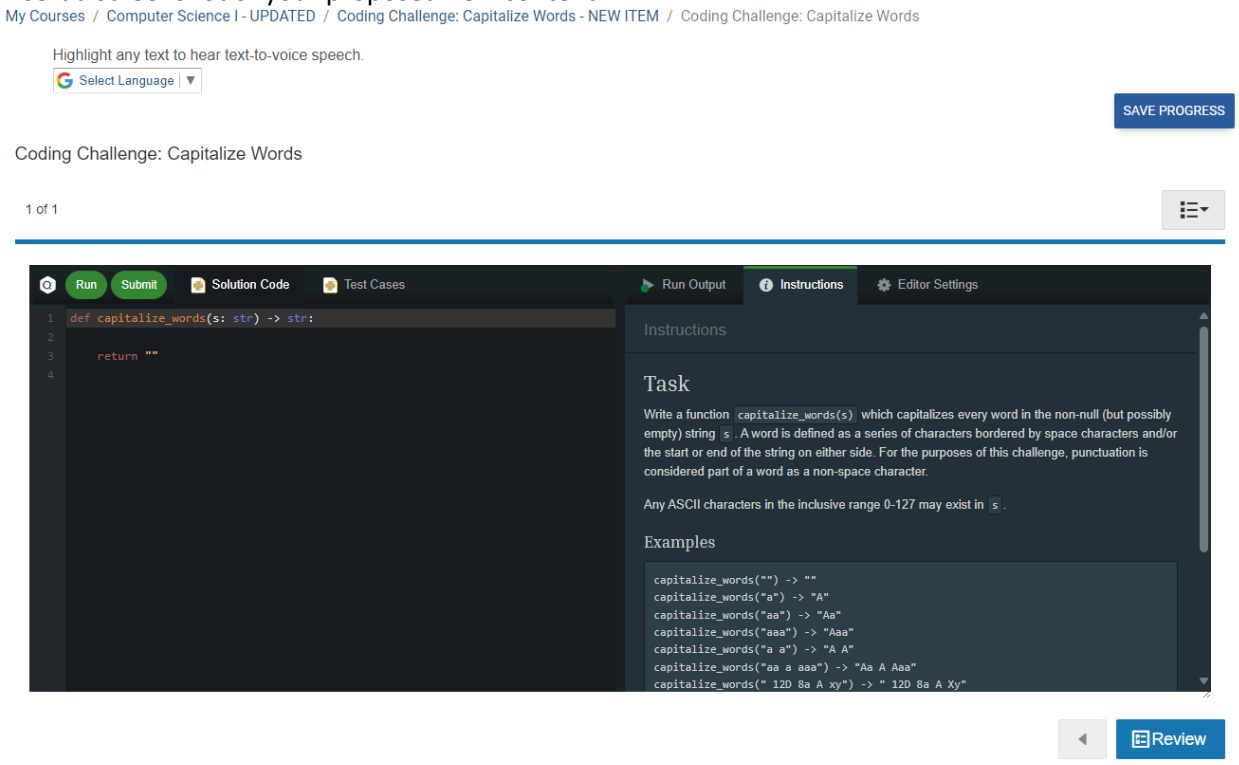
Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwk0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.



(SE)(Breakout(s)) and (Citation Type(s))

(6)(Q)(iv), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content

Update to Content Accepted by SRP

Numeric and Nonnumeric Data (Slides 26-27),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01?resume=False

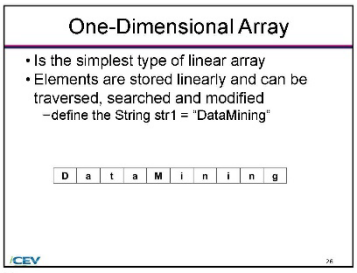
Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

Activity-Using Structured Data Types,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity04
This Activity is found in the Numeric & Nonnumeric lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

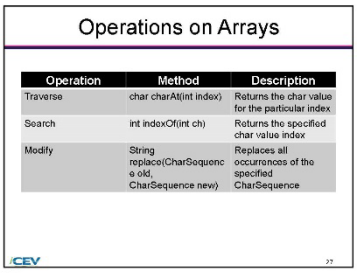
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/21/2024



26



27

1

Update to Content Accepted by SRP

SAVE PROGRESS

Project - Data Types & Structures Handbook

Numeric & Nonnumeric Data

1 of 1

Project Overview:
You will create a handbook on how to choose, identify and use the appropriate data type or structure in a program problem solution.

Directions:

1. Use the internet and other available resources to research each of the following data types or structures used in program problem solutions.
   - Integer
   - Real/Floating point (float)
   - Boolean
   - Character (Char)
   - Array (String)
   - String concatenation
   - char charAt(int index)
   - String substring(int beginIndex)
   - boolean contains(CharSequence s)
   - One-dimensional arrays
     - traverse
     - search
     - modify
2. Summarize your findings for each of the data types or structures. The summary should include how to choose, identify and use each data type or structure. Each data type or structure should be at least half of a page.
3. Your handbook should include the following sections:
   - Title page
   - Table of contents
   - Headings
   - Page numbers
   - Cited sources
4. Once complete, upload your handbook in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwle0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Update to Content Accepted by SRP

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

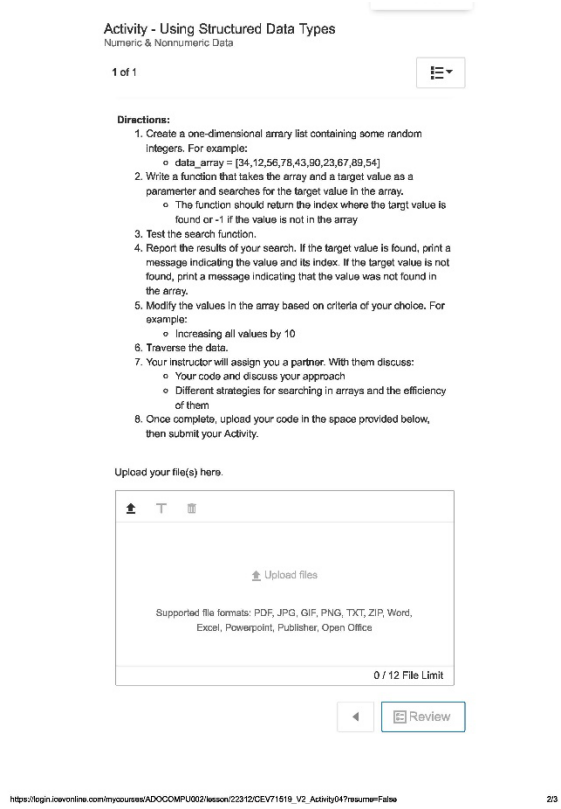
Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

CEV Copyright CEV Multimedia, LLC

Update to Content Accepted by SRP



Activity - Using Structured Data Types
Numeric & Nonnumeric Data

1 of 1

Directions:

1. Create a one-dimensional array list containing some random integers. For example:
   - data_array = [34,12,56,78,43,90,23,67,89,54]
2. Write a function that takes the array and a target value as a parameter and searches for the target value in the array.
   - The function should return the index where the targt value is found or -1 if the value is not in the array
3. Test the search function.
4. Report the results of your search. If the target value is found, print a message indicating the value and its index. If the target value is not found, print a message indicating that the value was not found in the array.
5. Modify the values in the array based on criteria of your choice. For example:
   - Increasing all values by 10
6. Traverse the data.
7. Your instructor will assign you a partner. With them discuss:
   - Your code and discuss your approach
   - Different strategies for searching in arrays and the efficiency of them
8. Once complete, upload your code in the space provided below, then submit your Activity.

Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Review

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity04?resume=False 2/3

(SE)(Breakout(s)) and (Citation Type(s))
(6)(Q)(v), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Numeric and Nonnumeric Data (Slides 26-27),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01?resume=False

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

Activity-Using Structured Data Types,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity04
This Activity is found in the Numeric & Nonnumeric lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Update to Content Accepted by SRP

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

One-Dimensional Array

- Is the simplest type of linear array
- Elements are stored linearly and can be traversed, searched and modified
  - define the String str1 = "DataMining"

D	a	t	a	M	i	n	i	n	g

CEV 26

26

Operations on Arrays

Operation	Method	Description
Traverse	char charAt(int index)	Returns the char value for the particular index
Search	int indexOf(int ch)	Returns the specified char value index
Modify	String replace(CharSequence old, CharSequence new)	Replaces all occurrences of the specified CharSequence

CEV 27

27

SAVE PROGRESS

Project - Data Types & Structures Handbook

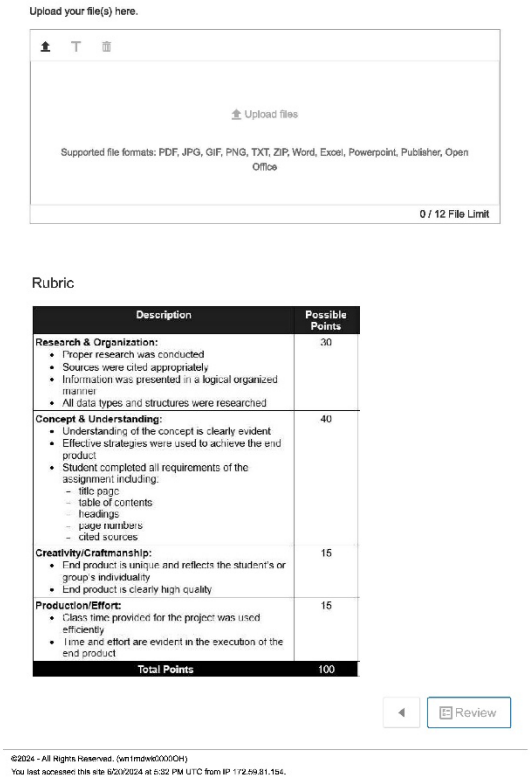
Numeric & Nonnumeric Data

Project Overview:
You will create a handbook on how to choose, identify and use the appropriate data type or structure in a program problem solution.

Directions:

1. Use the internet and other available resources to research each of the following data types or structures used in program problem solutions.
   - Integer
   - Real/Floating point (float)
   - Boolean
   - Character (Char)
   - Array (String)
   - String concatenation
   - char charAt(int index)
   - String substring(int beginIndex)
   - boolean contains(CharSequence s)
   - One-dimensional arrays
     - traverse
     - search
     - modify
2. Summarize your findings for each of the data types or structures. The summary should include how to choose, identify and use each data type or structure. Each data type or structure should be at least half of a page.
3. Your handbook should include the following sections:
   - Title page
   - Table of contents
   - Headings
   - Page numbers
   - Cited sources
4. Once complete, upload your handbook in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Update to Content Accepted by SRP



Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwle0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

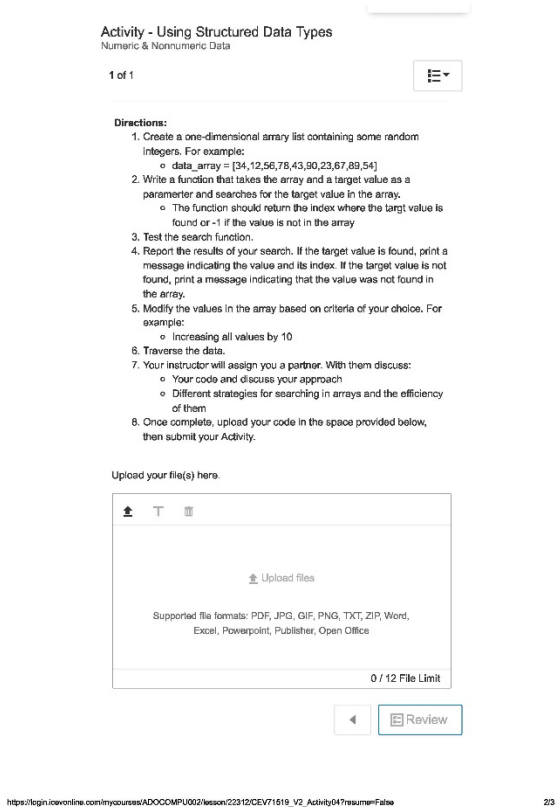
Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85,
92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95,
89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

iCEV Copyright CEV Multimedia, LLC

Update to Content Accepted by SRP



Activity - Using Structured Data Types

Numeric & Nonnumeric Data

1 of 1

Directions:

1. Create a one-dimensional array list containing some random integers. For example:
   - data_array = [34,12,56,78,43,90,23,67,89,54]
2. Write a function that takes the array and a target value as a parameter and searches for the target value in the array.
   - The function should return the index where the targt value is found or -1 if the value is not in the array
3. Test the search function.
4. Report the results of your search. If the target value is found, print a message indicating the value and its index. If the target value is not found, print a message indicating that the value was not found in the array.
5. Modify the values in the array based on criteria of your choice. For example:
   - Increasing all values by 10
6. Traverse the data.
7. Your instructor will assign you a partner. With them discuss:
   - Your code and discuss your approach
   - Different strategies for searching in arrays and the efficiency of them
8. Once complete, upload your code in the space provided below, then submit your Activity.

Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Review

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity04?resume=False 2/3

(SE)(Breakout(s)) and (Citation Type(s))
(6)(Q)(vi), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Numeric and Nonnumeric Data (Slides 26-27),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01?resume=False

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

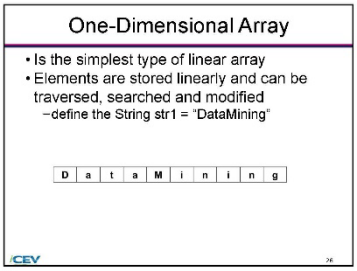
Activity-Using Structured Data Types,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity04
This Activity is found in the Numeric & Nonnumeric lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off

or start over, select Start Over to view the Activity.

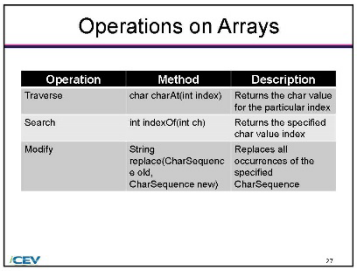
Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

6/21/2024



26



1

SAVE PROGRESS

Project - Data Types & Structures Handbook

Numeric & Nonnumeric Data

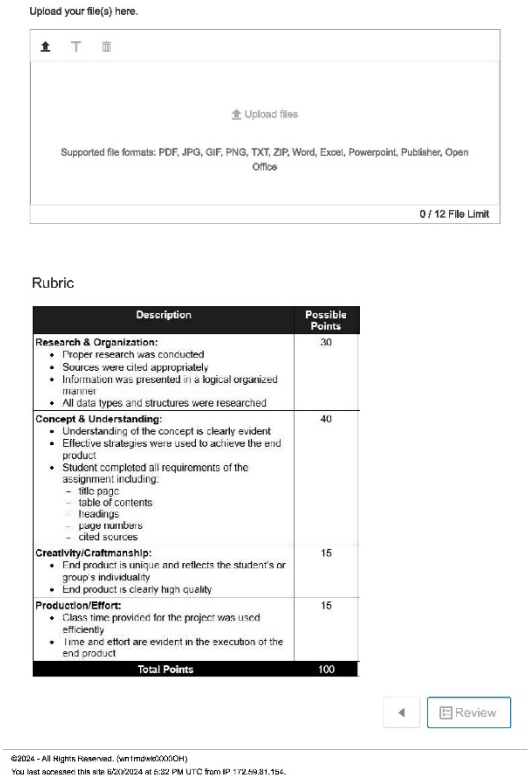
1 of 1

Project Overview:
You will create a handbook on how to choose, identify and use the appropriate data type or structure in a program problem solution.

Directions:

1. Use the internet and other available resources to research each of the following data types or structures used in program problem solutions.
   - Integer
   - Real/Floating point (float)
   - Boolean
   - Character (Char)
   - Array (String)
   - String concatenation
   - char charAt(int index)
   - String substring(int beginIndex)
   - boolean contains(CharSequence s)
   - One-dimensional arrays
     - traverse
     - search
     - modify
2. Summarize your findings for each of the data types or structures. The summary should include how to choose, identify and use each data type or structure. Each data type or structure should be at least half of a page.
3. Your handbook should include the following sections:
   - Title page
   - Table of contents
   - Headings
   - Page numbers
   - Cited sources
4. Once complete, upload your handbook in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Update to Content Accepted by SRP



Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwle0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

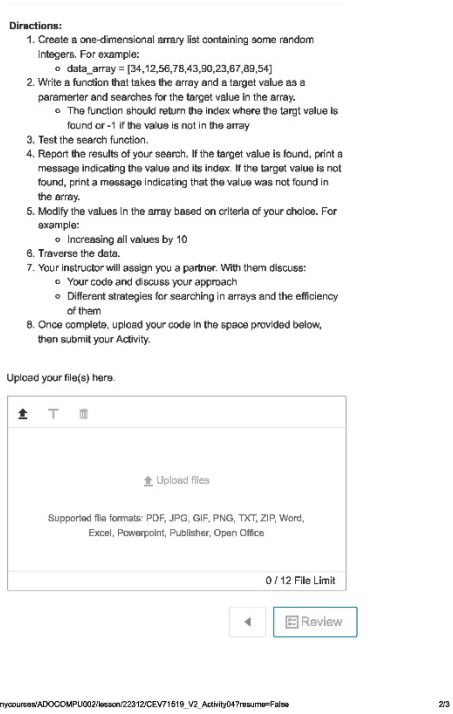
This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

iCEV Copyright CEV Multimedia, LLC

Activity - Using Structured Data Types
Numeric & Nonnumeric Data

1 of 1

Update to Content Accepted by SRP



Directions:

1. Create a one-dimensional array list containing some random integers. For example:
   - data_array = [34,12,56,78,43,90,23,67,89,54]
2. Write a function that takes the array and a target value as a parameter and searches for the target value in the array.
   - The function should return the index where the targt value is found or -1 if the value is not in the array
3. Test the search function.
4. Report the results of your search. If the target value is found, print a message indicating the value and its index. If the target value is not found, print a message indicating that the value was not found in the array.
5. Modify the values in the array based on criteria of your choice. For example:
   - Increasing all values by 10
6. Traverse the data.
7. Your instructor will assign you a partner. With them discuss:
   - Your code and discuss your approach
   - Different strategies for searching in arrays and the efficiency of them
8. Once complete, upload your code in the space provided below, then submit your Activity.

Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Review

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity04?resume=False 2/3

(SE)(Breakout(s)) and (Citation Type(s))
(6)(R)(i), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content
Numeric and Nonnumeric Data (Slides 21-27),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01?resume=False

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

Activity-Data Representation and Selection,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03
This Activity is found in the Numeric & Nonnumeric lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Update to Content Accepted by SRP

Data Types & Structures

- Represent different types of characters or values

Data Type	Represents	Examples
Integer (int)	Whole numbers	-11, 295, 0
Real/Floating Point (float)	Fractional numbers	-8.23, 0.0, 3.14159
Boolean (Boolean)	Logic true or false	True, False
Character (char)	Single character	A, f, &
Array (String)	Sequence of characters	"Hello World!"

21

Data Types & Structures

- Are chosen based on the type of characters or data being stored
- Examples include:
  - if the data consisted of the value '295' the integer data type would be chosen for use
  - if the data consisted of the value '1.375' the real (floating point) data type would be chosen
  - if the data consisted of the value 'True' the Boolean data type would be chosen

22

String Data

- Is an array of characters
- Examples include:
  - "Hello World!"
  - "How are you?"
  - "Yes"
    - define a variable called strSample that represents the string "Hello World!"
    - String strSample = "Hello World!"
    - //representation of string

23

String Concatenation

- Is joining two or more strings together into one
- Looks like the task being performed below:
  - String str1="Rock"; String str2 = "Star"; String str3 = Str1+str2; System.out.println(str3)

24

Other String Functions

Method	Description
char charAt(int index)	Returns the character value for the particular index
String substring(int beginIndex)	Returns the substring for a given beginning index
boolean contains(CharSequence s)	Returns True or False after matching the sequence of char values

25

Operations on Arrays

Operation	Method	Description
Traverse	char charAt(int index)	Returns the char value for the particular index
Search	int indexOf(int ch)	Returns the specified char value index
Modify	String replace(CharSequence old, CharSequence new)	Replaces all occurrences of the specified CharSequence

27

One-Dimensional Array

- Is the simplest type of linear array
- Elements are stored linearly and can be traversed, searched and modified
  - define the String str1 = "DataMining"

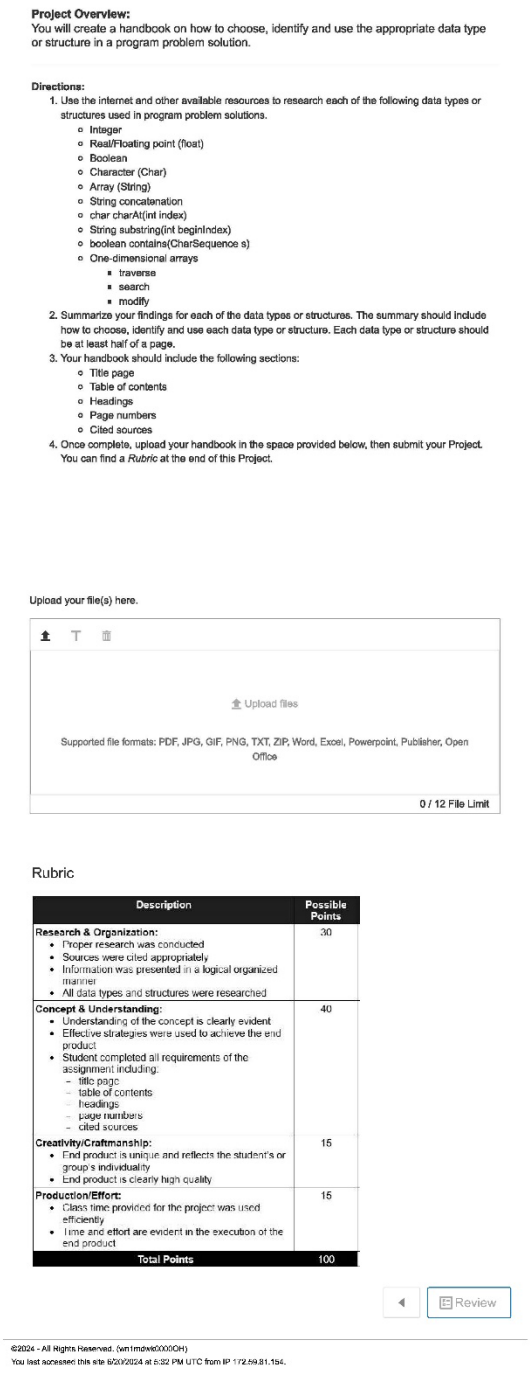
D	a	t	a	M	i	n	i	n	g

26

SAVE PROGRESS

Project - Data Types & Structures Handbook
Numeric & Nonnumeric Data

Update to Content Accepted by SRP



Project Overview:
You will create a handbook on how to choose, identify and use the appropriate data type or structure in a program problem solution.

Directions:

1. Use the internet and other available resources to research each of the following data types or structures used in program problem solutions.
   - Integer
   - Real/Floating point (float)
   - Boolean
   - Character (Char)
   - Array (String)
   - String concatenation
   - char charAt(int index)
   - String substring(int beginIndex)
   - boolean contains(CharSequence s)
   - One-dimensional arrays
     - traverse
     - search
     - modify
2. Summarize your findings for each of the data types or structures. The summary should include how to choose, identify and use each data type or structure. Each data type or structure should be at least half of a page.
3. Your handbook should include the following sections:
   - Title page
   - Table of contents
   - Headings
   - Page numbers
   - Cited sources
4. Once complete, upload your handbook in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwld0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

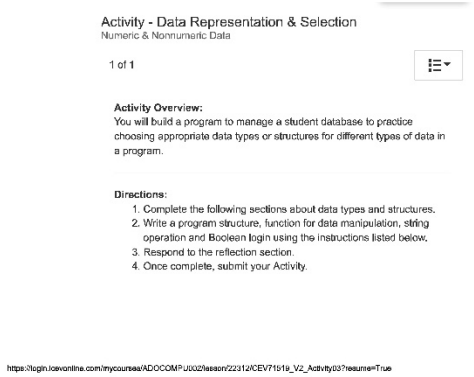
Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85,
92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95,
89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

iCEV Copyright CEV Multimedia, LLC

Update to Content Accepted by SRP



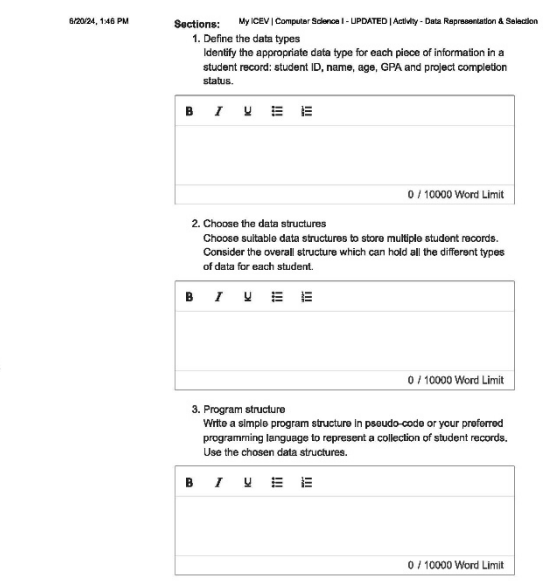
Activity - Data Representation & Selection
Numeric & Nonnumeric Data

1 of 1

Activity Overview:
You will build a program to manage a student database to practice choosing appropriate data types or structures for different types of data in a program.

Directions:
1. Complete the following sections about data types and structures.
2. Write a program structure, function for data manipulation, string operation and Boolean login using the instructions listed below.
3. Respond to the reflection section.
4. Once complete, submit your Activity.

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03?resume=True 1/4



6/20/24, 1:46 PM My ICEV | Computer Science I - UPDATED | Activity - Data Representation & Selection

Sections:

1. Define the data types
   Identify the appropriate data type for each piece of information in a student record: student ID, name, age, GPA and project completion status.

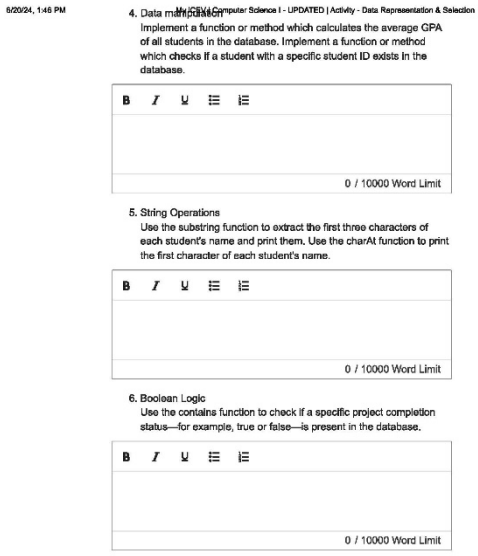
0 / 10000 Word Limit

2. Choose the data structures
   Choose suitable data structures to store multiple student records. Consider the overall structure which can hold all the different types of data for each student.

0 / 10000 Word Limit

3. Program structure
   Write a simple program structure in pseudo-code or your preferred programming language to represent a collection of student records. Use the chosen data structures.

0 / 10000 Word Limit



6/20/24, 1:46 PM My ICEV | Computer Science I - UPDATED | Activity - Data Representation & Selection

4. Data manipulation
   Implement a function or method which calculates the average GPA of all students in the database. Implement a function or method which checks if a student with a specific student ID exists in the database.

0 / 10000 Word Limit

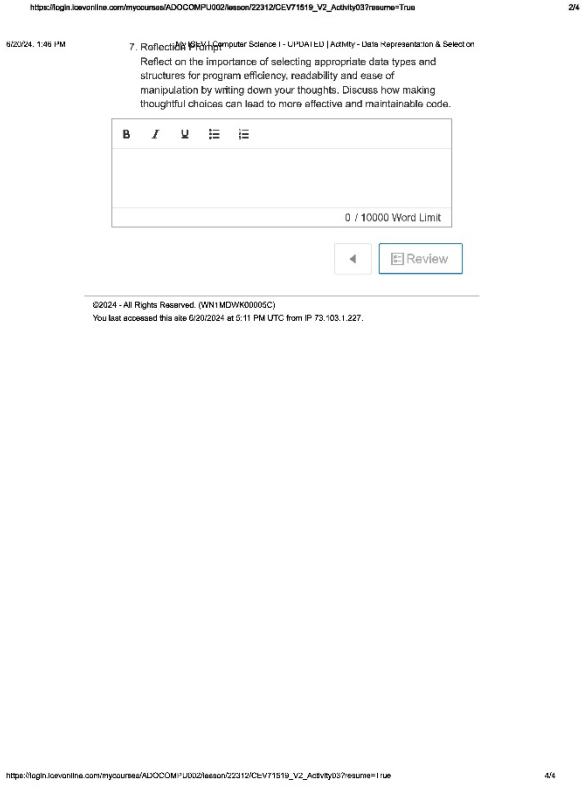
5. String Operations
   Use the substring function to extract the first three characters of each student's name and print them. Use the charAt function to print the first character of each student's name.

0 / 10000 Word Limit

6. Boolean Logic
   Use the contains function to check if a specific project completion status—for example, true or false—is present in the database.

0 / 10000 Word Limit

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03?resume=True 3/4



https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03?resume=True 2/4

6/20/24, 1:46 PM My ICEV | Computer Science I - UPDATED | Activity - Data Representation & Selection

7. Reflection Prompt
   Reflect on the importance of selecting appropriate data types and structures for program efficiency, readability and ease of manipulation by writing down your thoughts. Discuss how making thoughtful choices can lead to more effective and maintainable code.

0 / 10000 Word Limit

Review

©2024 - All Rights Reserved. (WN1MDWK00005C)
You last accessed this site 6/20/2024 at 5:11 PM UTC from IP 73.103.1.227.

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03?resume=True 4/4

(SE)(Breakout(s)) and (Citation Type(s))
(6)(R)(ii), Narrative & Activity

Description of the specific location and hyperlink to the exact location of currently adopted content

Update to Content Accepted by SRP

Numeric and Nonnumeric Data (Slides 21-27),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01?resume=False

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

Activity-Data Representation and Selection,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03
This Activity is found in the Numeric & Nonnumeric lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Data Types & Structures

- Represent different types of characters or values

Data Type	Represents	Examples
Integer (int)	Whole numbers	-11, 295, 0
Real/Floating Point (float)	Fractional numbers	-8.23, 0.0, 3.14159
Boolean (Boolean)	Logic true or false	True, False
Character (char)	Single character	A, f, &
Array (String)	Sequence of characters	"Hello World!"

CEV 21

Data Types & Structures

- Are chosen based on the type of characters or data being stored
- Examples include:
  - if the data consisted of the value '295' the integer data type would be chosen for use
  - if the data consisted of the value '1.375' the real (floating point) data type would be chosen
  - if the data consisted of the value 'True' the Boolean data type would be chosen

CEV 22

String Data

- Is an array of characters
- Examples include:
  - "Hello World!"
  - "How are you?"
  - "Yes"
    - define a variable called strSample that represents the string "Hello World!"
    - String strSample = "Hello World!"
    - //representation of string

CEV 23

String Concatenation

- Is joining two or more strings together into one
- Looks like the task being performed below:
  - String str1="Rock"; String str2 = "Star"; String str3 = Str1+str2; System.out.println(str3)

CEV 24

Update to Content Accepted by SRP

Other String Functions

Method	Description
char charAt(int index)	Returns the character value for the particular index
String substring(int beginIndex)	Returns the substring for a given beginning index
boolean contains(CharSequence s)	Returns True or False after matching the sequence of char values

25

One-Dimensional Array

- Is the simplest type of linear array
- Elements are stored linearly and can be traversed, searched and modified
  - define the String str1 = "DataMining"

D	a	t	a	M	i	n	i	n	g

26

Operations on Arrays

Operation	Method	Description
Traverse	char charAt(int index)	Returns the char value for the particular index
Search	int indexOf(int ch)	Returns the specified char value index
Modify	String replace(CharSequence old, CharSequence new)	Replaces all occurrances of the specified CharSequence

27

SAVE PROGRESS

Project - Data Types & Structures Handbook

Numeric & Nonnumeric Data

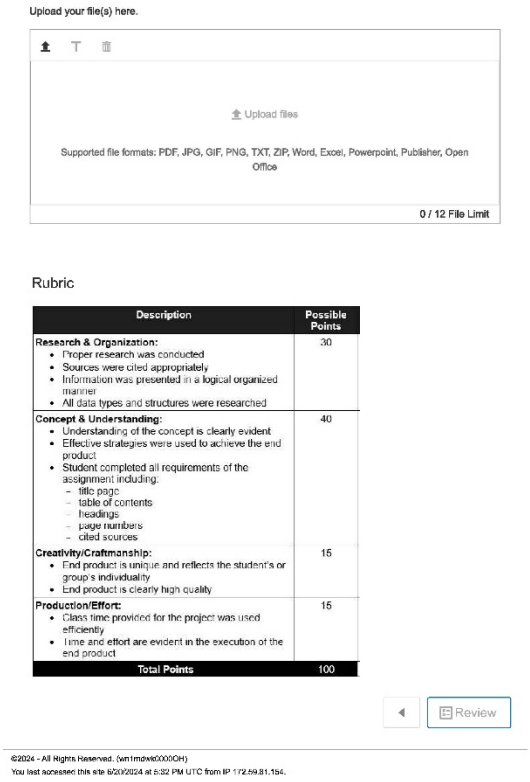
1 of 1

Project Overview:
You will create a handbook on how to choose, identify and use the appropriate data type or structure in a program problem solution.

Directions:

1. Use the internet and other available resources to research each of the following data types or structures used in program problem solutions.
   - Integer
   - Real/Floating point (float)
   - Boolean
   - Character (Char)
   - Array (String)
   - String concatenation
   - char charAt(int index)
   - String substring(int beginIndex)
   - boolean contains(CharSequence s)
   - One-dimensional arrays
     - traverse
     - search
     - modify
2. Summarize your findings for each of the data types or structures. The summary should include how to choose, identify and use each data type or structure. Each data type or structure should be at least half of a page.
3. Your handbook should include the following sections:
   - Title page
   - Table of contents
   - Headings
   - Page numbers
   - Cited sources
4. Once complete, upload your handbook in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Update to Content Accepted by SRP



Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwle0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

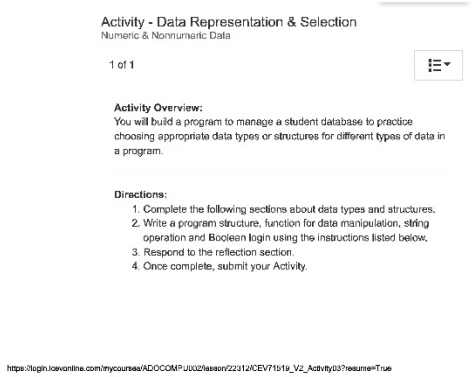
Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85,
92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95,
89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

iCEV Copyright CEV Multimedia, LLC

Update to Content Accepted by SRP



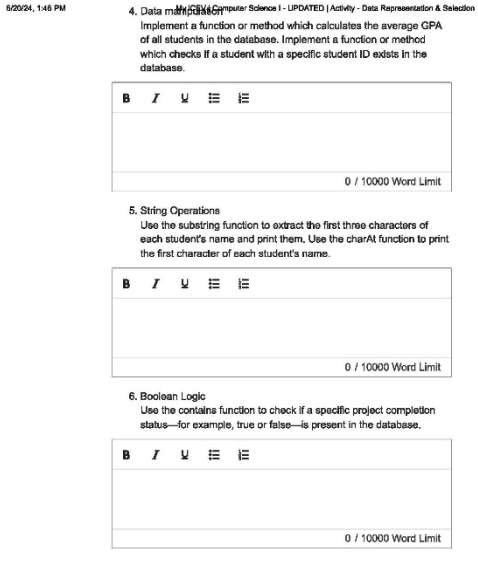
Activity - Data Representation & Selection
Numeric & Nonnumeric Data

1 of 1

Activity Overview:
You will build a program to manage a student database to practice choosing appropriate data types or structures for different types of data in a program.

Directions:
1. Complete the following sections about data types and structures.
2. Write a program structure, function for data manipulation, string operation and Boolean login using the instructions listed below.
3. Respond to the reflection section.
4. Once complete, submit your Activity.

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03?resume=True 1/4



6/20/24, 1:46 PM My ICEV | Computer Science I - UPDATED | Activity - Data Representation & Selection

Sections:

1. Define the data types
   Identify the appropriate data type for each piece of information in a student record: student ID, name, age, GPA and project completion status.

0 / 10000 Word Limit

2. Choose the data structures
   Choose suitable data structures to store multiple student records. Consider the overall structure which can hold all the different types of data for each student.

0 / 10000 Word Limit

3. Program structure
   Write a simple program structure in pseudo-code or your preferred programming language to represent a collection of student records. Use the chosen data structures.

0 / 10000 Word Limit

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03?resume=True 2/4

6/20/24, 1:46 PM My ICEV | Computer Science I - UPDATED | Activity - Data Representation & Selection

4. Data manipulation
   Implement a function or method which calculates the average GPA of all students in the database. Implement a function or method which checks if a student with a specific student ID exists in the database.

0 / 10000 Word Limit

5. String Operations
   Use the substring function to extract the first three characters of each student's name and print them. Use the charAt function to print the first character of each student's name.

0 / 10000 Word Limit

6. Boolean Logic
   Use the contains function to check if a specific project completion status—for example, true or false—is present in the database.

0 / 10000 Word Limit

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03?resume=True 3/4

6/20/24, 1:46 PM My ICEV | Computer Science I - UPDATED | Activity - Data Representation & Selection

7. Reflection Prompt
   Reflect on the importance of selecting appropriate data types and structures for program efficiency, readability and ease of manipulation by writing down your thoughts. Discuss how making thoughtful choices can lead to more effective and maintainable code.

0 / 10000 Word Limit

Review

©2024 - All Rights Reserved. (WN1MDWK00005C)
You last accessed this site 6/20/2024 at 5:11 PM UTC from IP 73.103.1.227.

https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03?resume=True 4/4

(SE)(Breakout(s)) and (Citation Type(s))
(6)(R)(iii), Narrative & Activity

Update to Content Accepted by SRP

Description of the specific location and hyperlink to the exact location of currently adopted content
Numeric and Nonnumeric Data (Slides 21-27),
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652

Project-Data Types and Structures Handbook,
https://login.icevonline.com/mycourses/ADOCOMPU001/lesson/21652/CEV71519_Project01?resume=False

Description of the specific location and hyperlink to the exact location of the proposed new content
Numeric and Nonnumeric Data Student Handout-Data Types and Structures,
https://files.icevonline.com/html/CEV71519_V2_HTML/CEV71519_V2_HTML_Student_Handout_-_Data_Types_and_Structures.htm
This Student Handout is found in the Numeric & Nonnumeric Data lesson beneath the Instructional Materials heading.

Activity-Data Representation and Selection,
https://login.icevonline.com/mycourses/ADOCOMPU002/lesson/22312/CEV71519_V2_Activity03
This Activity is found in the Numeric & Nonnumeric lesson beneath the Interactive Assignments heading. After clicking the link to the Activity, if a page appears asking if you want to continue where you left off or start over, select Start Over to view the Activity.

Screenshot of Currently Adopted Content

Insert a screenshot of your currently adopted content.

Data Types & Structures

- Represent different types of characters or values

Data Type	Represents	Examples
Integer (int)	Whole numbers	-11, 295, 0
Real/Floating Point (float)	Fractional numbers	-8.23, 0.0, 3.14159
Boolean (Boolean)	Logic true or false	True, False
Character (char)	Single character	A, f, &
Array (String)	Sequence of characters	"Hello World!"

21

Data Types & Structures

- Are chosen based on the type of characters or data being stored
- Examples include:
  - if the data consisted of the value '295' the integer data type would be chosen for use
  - if the data consisted of the value '1.375' the real (floating point) data type would be chosen
  - if the data consisted of the value 'True' the Boolean data type would be chosen

22

String Data

- Is an array of characters
- Examples include:
  - "Hello World!"
  - "How are you?"
  - "Yes"
    - define a variable called strSample that represents the string "Hello World!"
    - String strSample = "Hello World!"
    - //representation of string

23

String Concatenation

- Is joining two or more strings together into one
- Looks like the task being performed below:
  - String str1="Rock"; String str2 = "Star"; String str3 = Str1+str2; System.out.println(str3)

24

Update to Content Accepted by SRP

Other String Functions

Method	Description
char charAt(int index)	Returns the character value for the particular index
String substring(int beginIndex)	Returns the substring for a given beginning index
boolean contains(CharSequence s)	Returns True or False after matching the sequence of char values

25

One-Dimensional Array

- Is the simplest type of linear array
- Elements are stored linearly and can be traversed, searched and modified
  - define the String str1 = "DataMining"

D	a	t	a	M	i	n	i	n	g

26

Operations on Arrays

Operation	Method	Description
Traverse	char charAt(int index)	Returns the char value for the particular index
Search	int indexOf(int ch)	Returns the specified char value index
Modify	String replace(CharSequence old, CharSequence new)	Replaces all occurrences of the specified CharSequence

27

SAVE PROGRESS

Project - Data Types & Structures Handbook

Numeric & Nonnumeric Data

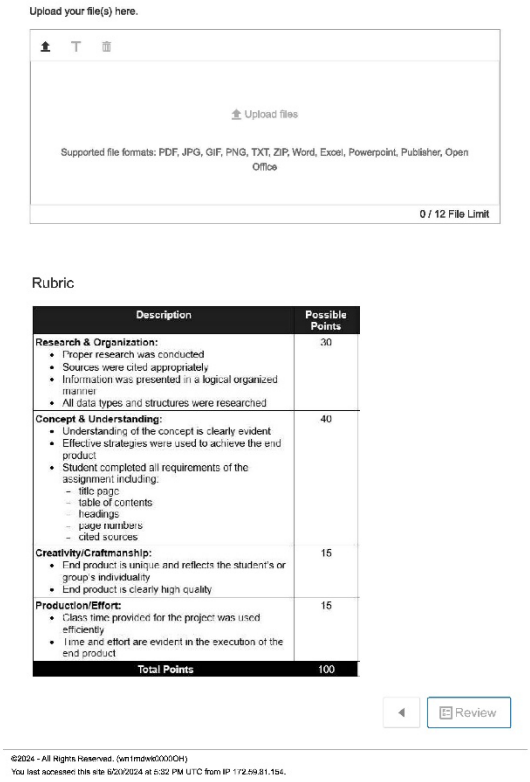
1 of 1

Project Overview:
You will create a handbook on how to choose, identify and use the appropriate data type or structure in a program problem solution.

Directions:

1. Use the internet and other available resources to research each of the following data types or structures used in program problem solutions.
   - Integer
   - Real/Floating point (float)
   - Boolean
   - Character (Char)
   - Array (String)
   - String concatenation
   - char charAt(int index)
   - String substring(int beginIndex)
   - boolean contains(CharSequence s)
   - One-dimensional arrays
     - traverse
     - search
     - modify
2. Summarize your findings for each of the data types or structures. The summary should include how to choose, identify and use each data type or structure. Each data type or structure should be at least half of a page.
3. Your handbook should include the following sections:
   - Title page
   - Table of contents
   - Headings
   - Page numbers
   - Cited sources
4. Once complete, upload your handbook in the space provided below, then submit your Project. You can find a *Rubric* at the end of this Project.

Update to Content Accepted by SRP



Upload your file(s) here.

Upload files

Supported file formats: PDF, JPG, GIF, PNG, TXT, ZIP, Word, Excel, Powerpoint, Publisher, Open Office

0 / 12 File Limit

Rubric

Description	Possible Points
Research & Organization: • Proper research was conducted • Sources were cited appropriately • Information was presented in a logical organized manner • All data types and structures were researched	30
Concept & Understanding: • Understanding of the concept is clearly evident • Effective strategies were used to achieve the end product • Student completed all requirements of the assignment including: – title page – table of contents – headings – page numbers – cited sources	40
Creativity/Craftmanship: • End product is unique and reflects the student's or group's individuality • End product is clearly high quality	15
Production/Effort: • Class time provided for the project was used efficiently • Time and effort are evident in the execution of the end product	15
Total Points	100

Review

©2024 - All Rights Reserved. (wn1mdwle0000OH)
You last accessed this site 6/20/2024 at 5:32 PM UTC from IP 172.59.81.154.

Screenshot of Proposed New Content

Insert a screenshot of your proposed new content.

Update to Content Accepted by SRP

Data Types & Structures

Choosing Data Types for Integer Data:
When writing program solutions, it is important to choose the appropriate data types for integer data based on the requirements of the program and the range of values one needs to represent. Common integer data types include:

- short int
- int
- long int or long
- fixed-width integer types like intX_t and uintX_t (from <stdint.h>), where X represents the number of bits

Choosing Data Types for Real Data:
When working with real data (floating-point numbers) run in program solutions, it is crucial to choose the appropriate data types based on the precision and range required for the application. Common real data types include:

- float (32-bit)
- double (64-bit)
- long double (extended precision, typically 80 or 128 bits)

Choosing Data Types for Boolean Data:
When working with Boolean data in program solutions, the appropriate data type is typically a Boolean type or its equivalent in the programming language being used. Common Boolean types include:

- bool
- Boolean(java)
- bool(C#):
- int(C):

The choice between these types depends on factors such as memory usage, precision needs, and the specific requirements of the calculations. Always ensure the chosen data types can accurately represent the values the program manages without causing overflow, loss of precision, or other issues.

Using One-Dimensional Arrays to Traverse Data:
One-dimensional arrays are a structured data type that allows storage and access to a collection of elements of the same data type. One can traverse the elements of a one-dimensional array using loops, typically a 'for' or 'while' loop.

Example:

```
# Define a one-dimensional array (list)
numbers = [1, 2, 3, 4, 5]

# Traverse the array using a for loop
print("Traversing the array using a for loop:")
for num in numbers:
print(num)
# Alternatively, traverse the array using a while loop
print("\nTraversing the array using a while loop:")
index = 0
while index < len(numbers):
print(numbers[index])
index += 1
```

In this example, there is a one-dimensional array named numbers containing integers. The array is then traversed using both a for loop and a while loop.

The output would be:

```
Traversing the array using a for loop:
1
2
3
4
5

Traversing the array using a while loop:
1
2
3
4
5
```

Using One-Dimensional Arrays to Search Data:
One-dimensional arrays are commonly used for searching data because they provide a structured and efficient way to organize and access elements.

Example:

```
# Sample data in an array
data_array = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Function to search for a value in the array
def search_data(array, target):
    for index, value in enumerate(array):
        if value == target:
            return index  # Return the index if found
    return -1  # Return -1 if not found

# Example usage
target_value = 40
result = search_data(data_array, target_value)

if result != -1:
    print(f"Value {target_value} found at index {result}.")
else:
    print(f"Value {target_value} not found in the array.")
```

Using One-Dimensional Arrays to Modify Data:
Modifying data in programming is a common and essential operation. Some reasons include updating values and correcting errors.

Example:

```
# Creating a one-dimensional array (list in Python)
my_array = [1, 2, 3, 4, 5]

# Accessing and modifying elements
print("Original array:", my_array)

# Modify the element at index 2
my_array[2] = 10

# Add a new element to the end of the array
my_array.append(6)

print("Modified array:", my_array)
```

Identifying and choosing data structures and functions in program problem solution :
Following a checklist when identifying which data structures or functions you should use will help narrow down the choices. Your checklist may include:

- the nature of the data
- what operations will be performed
- problems/solutions
- precision requirements
- insertion and deletion efficiency
- search and sorting requirements
- etc.

Example:

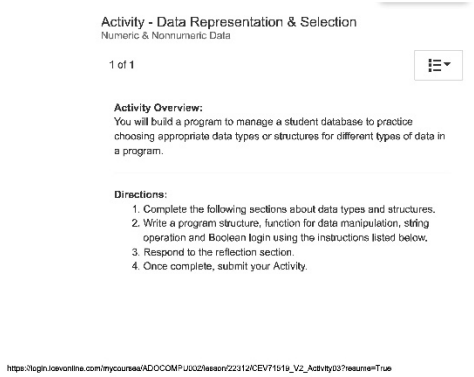
Problem: Keeping Track of Students and Grades

```
student1 = {"name": "John Doe", "student_id": 1, "grades": [90, 85,
92]}
student2 = {"name": "Jane Smith", "student_id": 2, "grades": [88, 95,
89]}
students = [
    {"name": "John Doe", "student_id": 1, "grades": [90, 85, 92]},
    {"name": "Jane Smith", "student_id": 2, "grades": [88, 95, 89]}
]
grades_john = [90, 85, 92]
grades_jane = [88, 95, 89]
average_grade_john = sum(grades_john) / len(grades_john)
average_grade_jane = sum(grades_jane) / len(grades_jane)
student_ids = {1, 2}
```

This example demonstrates the use of dictionaries, lists, sets, and basic arithmetic operations to represent and manage data related to students and their grades in a class. The choice of data types and structures depends on the specific requirements of your program and the operations you need to perform.

iCEV Copyright CEV Multimedia, LLC

Update to Content Accepted by SRP

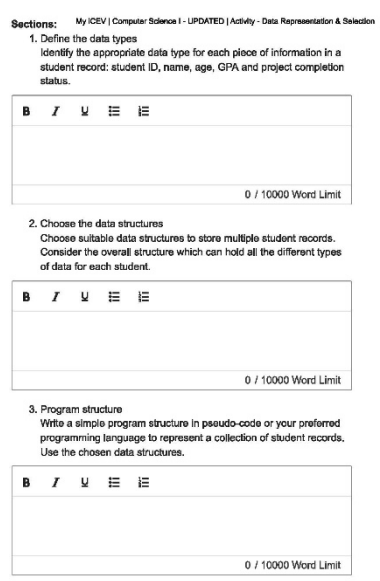


Activity - Data Representation & Selection
Numeric & Nonnumeric Data

1 of 1

Activity Overview:
You will build a program to manage a student database to practice choosing appropriate data types or structures for different types of data in a program.

Directions:
1. Complete the following sections about data types and structures.
2. Write a program structure, function for data manipulation, string operation and Boolean login using the instructions listed below.
3. Respond to the reflection section.
4. Once complete, submit your Activity.



Sections:

1. Define the data types
   Identify the appropriate data type for each piece of information in a student record: student ID, name, age, GPA and project completion status.

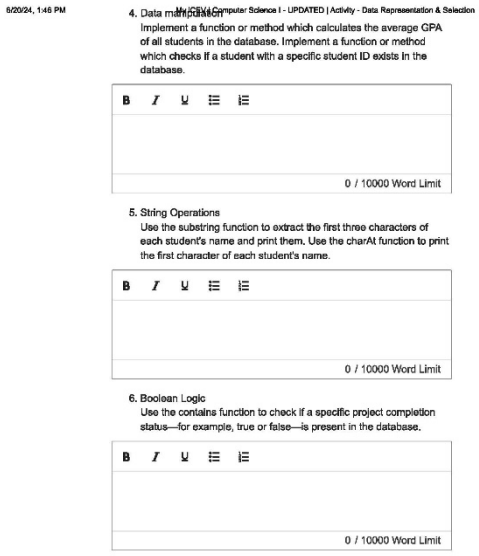
0 / 10000 Word Limit

2. Choose the data structures
   Choose suitable data structures to store multiple student records. Consider the overall structure which can hold all the different types of data for each student.

0 / 10000 Word Limit

3. Program structure
   Write a simple program structure in pseudo-code or your preferred programming language to represent a collection of student records. Use the chosen data structures.

0 / 10000 Word Limit



4. Data manipulation
   Implement a function or method which calculates the average GPA of all students in the database. Implement a function or method which checks if a student with a specific student ID exists in the database.

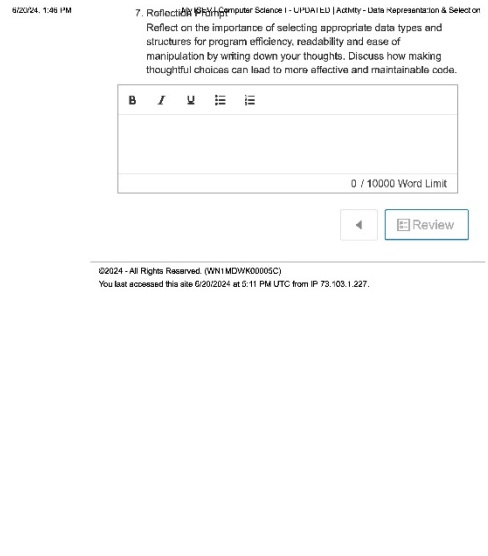
0 / 10000 Word Limit

5. String Operations
   Use the substring function to extract the first three characters of each student's name and print them. Use the charAt function to print the first character of each student's name.

0 / 10000 Word Limit

6. Boolean Logic
   Use the contains function to check if a specific project completion status—for example, true or false—is present in the database.

0 / 10000 Word Limit



7. Reflection Prompt
   Reflect on the importance of selecting appropriate data types and structures for program efficiency, readability and ease of manipulation by writing down your thoughts. Discuss how making thoughtful choices can lead to more effective and maintainable code.

0 / 10000 Word Limit

Review

©2024 - All Rights Reserved. (WN1MDWK00005C)
You last accessed this site 6/20/2024 at 5:11 PM UTC from IP 73.103.1.227.

Update to Content Accepted by SRP

Assurances

These assurances apply to all material submitted to update content in state-adopted instructional materials.

Publisher acknowledges that:

- There will be no additional cost to the state;
- The new material meets the applicable Texas Essential Knowledge and Skills (TEKS), English Language Proficiency Standards (ElPS), or Texas Prekindergarten Guidelines (TPG) and is free from factual errors; and
- The updates in the new edition do not affect the product's coverage of Texas Education Code (TEC), §28.002(h), as it relates to that specific subject and grade level or course(s), understanding the importance of patriotism and functioning productively in a free-enterprise society with appreciation for the basic democratic values of our state and national heritage.

Signature: By entering your name below, you are confirming the above assurances, and signing this document electronically. You agree that your electronic signature is the equivalent of your manual signature.

Clayton Franklin

Date Submitted: 6/21/2024