iCEV Fundamentals of Computer Science

Knowledge and Skill Statement	Student Expectation	Breakout	iCEV Citation			
			Narrative/Activity	Type of Citation (New Content/New Citation)	Lesson Title	New Location
(1) Employability. The student identifies various employment opportunities in the computer science field. The student is expected to:	(A) identify job and internship opportunities and accompanying job duties and tasks and contact one or more companies or organizations to explore career opportunities;	(ii) identify internship opportunities	Narrative	New Content	STEM Careers: Fundamentals of Computer Science	Student Handout- Computer Science Career Preparation
(1) Employability. The student identifies various employment opportunities in the computer science field. The student is expected to:	(A) identify job and internship opportunities and accompanying job duties and tasks and contact one or more companies or organizations to explore career opportunities;	(v) contact one or more companies or organizations to explore career opportunities	Narrative	New Content	STEM Careers: Fundamentals of Computer Science	Student Handout- Computer Science Career Preparation
(2) Creativity and innovation. The student develops products and generates new knowledge, understanding, and skills. The student is expected to:	(C) discuss methods and create and publish web pages using a web-based language such as HTML, Java Script, or XML; and	(i) discuss methods [of] using a web-based language	Activity	New Content	Web Publication Basics	Project- Build a Website
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(C) identify a problem's description, purpose, and goals;	(ii) identify a problem's purpose	Narrative	New Content	Problem-Solving in Computer Science	Student Handout- Problem Analysis
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(C) identify a problem's description, purpose, and goals;	(iii) identify a problem's goals	Narrative	New Content	Problem-Solving in Computer Science	Student Handout- Problem Analysis
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(D) demonstrate coding proficiency in a programming language by developing solutions that create stories, games, and animations;	(i) demonstrate coding proficiency in a programming language by developing solutions that create stories	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(D) demonstrate coding proficiency in a programming language by developing solutions that create stories, games, and animations;	(i) demonstrate coding proficiency in a programming language by developing solutions that create stories	Activity	New Content	Programming Languages	Project- Coding a Story
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(D) demonstrate coding proficiency in a programming language by developing solutions that create stories, games, and animations;	(ii) demonstrate coding proficiency in a programming language by developing solutions that create games	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(D) demonstrate coding proficiency in a programming language by developing solutions that create stories, games, and animations;	(iii) demonstrate coding proficiency in a programming language by developing solutions that create animations	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(D) demonstrate coding proficiency in a programming language by developing solutions that create stories, games, and animations;	(iii) demonstrate coding proficiency in a programming language by developing solutions that create animations	Activity	New Content	Programming Languages	Project- Coding a Story
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(F) communicate an understanding of and use variables within a programmed story, game, or animation;	(i) communicate an understanding of variables within a programmed story, game, or animation	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(F) communicate an understanding of and use variables within a programmed story, game, or animation;	(ii) use variables within a programmed story, game, or animation	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(H) communicate an understanding of and use sequence within a programmed story, game, or animation;	(i) communicate an understanding of sequence within a programmed story, game, or animation	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(H) communicate an understanding of and use sequence within a programmed story, game, or animation;	(ii) use sequence within a programmed story, game, or animation	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(I) communicate an understanding of and use conditional statements within a programmed story, game, or animation;	(i) communicate an understanding of conditional statements within a programmed story, game, or animation	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(I) communicate an understanding of and use conditional statements within a programmed story, game, or animation;	(ii) use conditional statements within a programmed story, game, or animation	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(J) communicate an understanding of and use iteration within a programmed story, game, or animation;	(i) communicate an understanding of iteration within a programmed story, game, or animation	Narrative	New Content	Programming Languages	Student Handout- Coding Examples
(4) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:	(J) communicate an understanding of and use iteration within a programmed story, game, or animation;	(ii) use iteration within a programmed story, game, or animation	Narrative	New Content	Programming Languages	Student Handout- Coding Examples

Computer Science Career Preparation

Are you interested in pursuing a career in computer science? This field offers many exciting and challenging opportunities, but choosing the right career path can be daunting. An individual should contact companies for the chance to pursue one of the options below. Companies can be contacted about these opportunities in the following ways:

- Phone calls
- Email
- In-person visits
- Visiting the company website
- Social media
- Recruiters

Job Shadowing

Job shadowing is following a professional to learn what their day-to-day routine looks like in a career of interest. In computer science, this may include observing software development, data analysis, system administration, cybersecurity or web development tasks. Job shadowing helps individuals network with professionals and learn about companies they may be interested in. An individual can participate in job shadowing for a day, a week or longer, depending on the company and an individual's goals. Often, job shadowing is done by students or employees in a company looking to grow into another role.

Mentorships

A mentorship involves a professional relationship between an experienced mentor and a less experienced mentee. Finding a mentor whose career is similar to what an individual would like to obtain is a great way to begin or grow in a career. Mentors share their experiences and knowledge to advise their mentees and help create a clear pathway to reaching their goals. This relationship is often informal and can vary in duration, from a month to years. Mentorships are often pursued by individuals on their own accord and can be developed through networking, conferences, professional organizations or online platforms.

Apprenticeship Training

Apprenticeship programs combine on-the-job learning with formal education. Usually, apprentices attend classes during part of the day, then report to a workplace for the other part of the day. An apprentice learns to master specific skills for a career by working alongside an experienced professional. In the beginning, the apprentice completes tasks and practices skills with supervision. Over time, apprentices learn to do tasks and skills independently. Apprentices are typically paid and are available through community college programs or technical schools.

Internship

An internship allows individuals to gain work experience in a specific field and bridge the gap between academic learning and professional work. These opportunities provide a supervised and structured learning environment where an individual completes tasks to support the company. These opportunities provide individuals with practical, hands-on experience in a professional setting. In computer science, internships can provide supervised experience in software development, programming, data analysis, cybersecurity, artificial intelligence, web and mobile app development, or systems administration. Many colleges or universities require an internship as part of coursework and they may be paid or unpaid opportunities.

Build a Website

Project Overview:
You will create a website which has at least two pages and uses HTML, CSS, JavaScript and a widget.

Directions:
1. First, plan your web page. Draw a sketch, indicating where you would like a widget and where your script will be active.

2. Find a web host. Your instructor may have a suggestion of a specific web host to use. You need to create text files on your own computer.

3. Your web host provider most likely also provided a domain name. If not, ask your instructor how to secure a domain name.

4. Use your HTML code from the **Basic HTML Activity**.

5. Create another file in your text editor and save it as “style.css” or some other name with the same extension. Write your CSS code in this file. Use at least three styles in your HTML file.

6. You can add your JavaScript either directly to your HTML file or as a separate file with a “.js” extension.

7. Add a widget to your HTML file.

8. Upload your files to your web host, including any images you are using. Your web host provider will have information on how to do this. Most likely you will use a file transfer protocol client such as Filezilla™.

9. Test your files including the functionality of your widget and JavaScript. If necessary, fix any problems.

10. When complete, turn in your planning sketch and the URL of your website according to your instructor’s directions.

11. Engage in a class discussion. Based on your knowledge and experience building a website, discuss the methods of using a web-based language. Examine the following:
    - functionality and use
    - pros and cons
    - what you learned about the method you used

Build a Website

Rubric

Description	Possible Points	Your Score
Web Page Plan: • Sketch drawn detailing where the widget and script will be shown • Found a web host provider and domain	10	
Concept & Understanding: • HTML was used from the Basic HTML Activity and a basic understanding of the HTML file is evident • Effective strategies were used to have at least three styles within the web page • Understanding of how to add the widget in the HTML is evident	20	
Web Page: • Host and domain for the web page was found • Basic HTML Activity was used for the HTML file • A .css file was created and saved in the text editor with three styles detailed • JavaScript was implemented within the HTML or with a .js extension • A widget was added • Files were uploaded to the web host • URL of the website was working	60	
Production/Effort: • Class time provided for the web page was used efficiently • Time and effort are evident in the execution of the web page	10	
Total Points	100	

Additional Comments:

Problem Analysis

In computer science, a problem is often a specific process that is difficult, tedious or highly time-consuming for humans to do alone. Programmers may write software applications to solve these kinds of problems. To develop a proper solution, programmers must first identify a problem's description, purpose and goals. This makes it easier for a programmer to develop an effective action plan and allocate resources as needed.

Problem Description

The problem description is a comprehensive explanation of the problem. This description must be as specific as possible for it to be accurately addressed. First, the description should identify the issue. Problems can be related to unoptimized algorithms, system performance, inadequate cybersecurity, software bugs or issues with user-friendly interface. The problem description should also include specific details about how the problem affects the function of the code. This can consist of symptoms of the issues, limitations, constraints or shortcomings.

Problem Purpose

The problem purpose is typically a set of statements that describe why solving the problem identified is important and/or beneficial. This should outline the objectives, desired outcomes and potential impacts. These statements should relate to the issues identified in the problem description. Common focuses of the problem purpose may include improving user experience, increasing data accuracy, reducing security threats, enhancing workflow processes or improving system efficiency.

Problem Goals

The problem goals identify specific, measurable targets needed to solve the problem. The problem goals need to be well-defined since they should serve as milestones to track progress in addressing the issue. These goals can include objective metrics or qualitative improvements needed to reflect the desired end state. Common examples of problem goals include improving relatability to a specific level, decreasing code execution time by a certain percentage or implementing a new feature.

Coding Examples

Programming is the process of writing instructions a computer system can understand and use to perform an operation. A coder might want to develop an operation or solution such as stories, games and animations. The process for creating these solutions may include the following steps, however, the procedure depends on the specific project.

1. **Define**: Describe the idea for the solution. Describe your plans for the story, game or animation by identifying components, such as the concept, roles or characters, storyline, and interactivity features.
2. **Select a programming language**: Select a language you are comfortable using to develop your solution. For example, Python, JavaScript, Java, C++ and C are common languages to choose from.
3. **Plan**: Develop an outline to achieve the solution. Make decisions for the creation of assets and the processes to code them.
4. **Create**: Generate the audio and visual assets to use for the story, game or animation like music, sound effects, images or graphics. Remember to use creativity and align the assets with the conceptual ideas identified in the first step. However, this step can mold the concept based on the assets found or created.
5. **Program**: Using the programming language selected in step two to code specific tasks, such as user inputs, game rules, animating objects, and character interactions. The code may use various functions, such as variables, sequences, conditional statements, operators, iterations and loops. Examples of programming language functions are listed below the process. These examples are for various components of stories, games and animations.
6. **Iterate & Test**: Testing code often, such as after completing each function, can ensure any problems or bugs are more easily identified, and, in turn, solved. Test the solution to check the components of the story, game or animation functionality and quality. Iterate the solution as needed based on the findings.
7. **Incorporate Assets**: Integrate the assets generated in step four to enrich the concept of the story, game or animation.
8. **Disperse**: Prepare the solution for use after coding the necessary elements. Based on the intended use or platform, the solution deployment process will vary.
9. **Improve & Iterate**: Once the solution is deployed, use reactions or commentary to make improvements or updates.
10. **Sustain**: Maintain the solution by incorporating user feedback, patching for errors, or expanding the capabilities of the story, game or animation.

Conditional Statements & Operators

If-Statements

With a single conditional:

Notice you must indent under the if-statement

```
a = 35
b = 70
if a > b:
```

Coding Examples

print (“The condition is True”)

Nothing will be printed on the screen

Steps:

- Each variable is assigned an integer value
  - variable ‘a’ gets a value of 35
  - variable ‘b’ gets a value of 70
- Ask the program to test a conditional statement to see if ‘a’ is greater than ‘b’
  - if True, the if-statement will evaluate to True and print “the condition is True” to the screen
  - if ‘a’ is not greater than ‘b’, the code inside the if-statement will not run and nothing will be printed to the screen

If-Statements

With a single conditional:

Notice you must indent under the if-statement

```
apple = 70
banana = 35
if a > b:
    print (“Congrats, you found the apple and gained 70 points. Go gather the banana next.”)
```

Steps:

- Each variable is assigned an integer value
  - variable ‘a’ gets a value of 70
  - variable ‘b’ gets a value of 35
- Ask the program to test a conditional statement to see if ‘a’ is greater than ‘b’
  - if True, the if-statement will evaluate to True and print “Congrats, you found the apple and gained 70 points. Go gather the banana next.” to the screen
  - if ‘a’ is not greater than ‘b’, the code inside the if-statement will not run and nothing will be printed to the screen

*This is an example of a conditional statement for a game.

Comparing two conditionals in a logical condition with an AND:

```
c = 100
d = 55
e = 100
if (c > d) and (c == e)
    print (“Both conditions are True”)
```

Steps:

- Each variable is assigned an integer value

Coding Examples

- Program tests two conditions to determine if both are True by using the and-operator
  - tests if ‘c’ is greater than ‘d’
  - tests if ‘c’ is exactly equal to ‘e’
    - if the entire if-statement evaluates to True and the code inside runs, printing the message “Both conditionals are True”

Comparing two conditionals in a logical condition with an AND:

step1_completed = True
step2_completed = True

if (step1_completed) ***and*** (step2_comleted)
print (“You have used the clues in the treasure map and located the treasure. You have completed your quest.”)

Steps:

- Each variable is assigned a Boolean value
- Program tests two conditions to determine if both are True by using the and-operator
  - tests if the treasure map clues were followed
  - tests if the treasure was located
    - if the entire if-statement evaluates to True and the code inside runs, printing the message “Both conditionals are True”

*This is an example of a conditional statement for a story.

Comparing two conditionals with an OR:

numberGrade = 85
letterGrade = “A”
if *(numberGrade >= 90)* ***or*** *(letterGrade == “A”):*
print (“Your final grade is A!”)

Steps:

- ‘numberGrade’ is an integer
- ‘letterGrade’ is a string
- Program tests two conditions to determine if at least one is True by using the or-operator
  - tests to see if ‘numberGrade’ is greater than or equal to 90
  - tests to see if ‘letterGrade’ is exactly equal to “A”
    - at least one of the conditions is True, so the entire if-statement evaluates to True

Comparing two conditionals with an OR:

numberGrade = 90

Coding Examples

```
letterGrade = "A"
if (numberGrade >= 90) or (letterGrade == "A"):
    print ("Animation of A+ celebration")
```

Steps:

- 'numberGrade' is an integer
- 'letterGrade' is a string
- Program tests two conditions to determine if at least one is True by using the or-operator
  - tests to see if 'numberGrade' is greater than or equal to 90
  - tests to see if 'letterGrade' is exactly equal to "A"
    - at least one of the conditions is True, so the entire if-statement evaluates to True

*This is an example of a conditional statement for an animation.

<u>If-else statement block:</u>

```
grade = 80
if grade >= 70:
    print ("You passed!")
else:
    print ("Sorry, you failed.")
```

Steps:

- One variable called 'grade' which is assigned to integer value 80
  - tests 'grade' to determine if it is greater than or equal to 70

<u>If-elif-else statement block:</u>

```
grade = 95
if (grade >= 70) and (grade < 80):
    print ("Your grade is a C")
elif (grade >= 80) and (grade < 90):
    print ("Your grade is a B")
elif (grade >= 90):
    print ("Your grade is an A")
else:
    print ("Your grade is an F. Try again.")
```

Steps:

- One variable called 'grade' which is assigned an integer value of 95
- Program will exit from the entire if-elif-else block as soon as a True condition is found

Sequences & Indexing

Coding Examples

List

Multidimensional Array Lists:

myLists = [[i, j, k] , [r, s, t] , [w, x, y, z]]

Steps:

- Spacing added for clarity, Python ignores spaces here
  - set of three lists stored within a larger container
  - each inner list has an index value
  - [i, j, k] has an index of [0]
  - [r, s, t] has an index of [1]
  - [w, x, y, z] has an index of [2]
- Each value in the inner lists also have an index
  - can combine index values to return a specific

In / Not-In Operators

On a list example:

```
fruit1 = “banana”
fruit2 = “honeydew”
myFavoriteFruits = ["banana", "apple", "orange", "pear", "peach", "mango"]

if fruit1 in myFavoriteFruits:
    Print (fruit1, “is in the list”)
if fruit2 not in myFavoriteFruits:
    Print (fruit 2, “is not in the list”)
```

Steps:

- Start with two string variables, fruit1 and fruit2
- List of strings called myFavoriteFruits which contains six different fruit names
- Code contains two if-statements using the in-operator and not-in operator
- First if-statement looks to see if the value of fruit1 is also within the list, myFavoriteFruits
  - banana is in the list, so the if-statement will evaluate as True
    - program will print “banana is in the list”
- Second if-statement looks to see if the value of fruit2 is not included in the list, myFavoriteFruits
  - because the list does not contain “honeydew”, this if-statement will evaluate to True
    - program will print “honeydew is not in the list”

Coding Examples
Iteration & Loops

Loops

Iteration using a loop:

myIntegerNumbers = [34, 45, 55, 92, 843, 22, -88, -92, 0, -635]

→iteration loop (step through each index and double the value in that location) →

Steps:

- Step through each value in a list of integers, multiply each value by two and write the results into a new list
  - original list does not actually change although Python allows list values to be changed if desired

For-Statement

The for-statement declaration requires three values as inputs:

1. Loop counting variable
   - integer called "I" and is initialized to a value of 0
   - semicolon separates from the next part of the statement
   - here is written as int i = 0
2. Loop condition
   - loop should run as long as the counting variable "I" is less than 6
   - written as i < 6
3. Loop stepping increment
   - changes the value of the loop counting variable each time the loop runs
   - value of "i" is to be increased by one after each iteration through the loop
   - increment-by-one operator in C-family languages and others is a double-plus symbol after the variable name
   - can also decrement or decrease the counting variable using the decrement-by-one operator in C-family languages and others, the double minus symbol
     - incrementing 'i' by one, so the double-plus symbol is used, i++

While-Loop:

```
j = 1
while j < 6:
    print (j)
    j += 1
```

Steps:

- Loop counting variable is 'j'
  -variable 'j' is an integer value set to one just before the loop begins
- Loop condition is for the loop to keep running as long as – or while 'j' is less than six

Coding Examples

- Loop body contains the instructions regarding what the program should do during each pass through the loop
  - in this case, the loop body prints the current value of 'j' and then increases the value in 'j' by one using the increment-by-one operator
- Loop stepping increment is one, as given in the last statement, 'j' plus equals one
  - each time add one to the counting variable
  - a colon is used after the while-statement condition and the loop body is indented below

Code:

- 'j' begins at 1, since 1 is less than 6, we enter the loop
  - value in 'j' is printed to the screen → 1
- 'j' is the incremented by one, so 'j' is now 2
  - reevaluate the condition: is 'j' less than 6?
  - 'j' is now 2 and is less than 6
    - enter the loop
    - value in 'j' is printed to the screen → 2
- 'j' is then incremented by one, so 'j' is now 3
  - reevaluate the condition: is 'j' less than 6?
  - 'j' is now 3 and is less than 6
    - enter the loop
    - value in 'j' is printed to the screen → 3
- 'j' is then incremented by one, so 'j' is now 4
  - reevaluate the condition: is 'j' less than 6?
  - 'j' is now 4 and is less than 6
    - enter the loop
    - value in 'j' is printed to the screen → 4
- 'j' is then incremented by one, so 'j' is now 5
  - reevaluate the condition: is 'j' less than 6?
  - 'j' is now 5 and is less than 6
    - enter the loop
    - value in 'j' is printed to the screen → 5
- 'j' is then incremented by one, so 'j' is now 6
  - reevaluate the condition: is 'j' less than 6?
  - 'j' is now 6 but is not less than 6
    - do not enter the loop because the loop condition is now False
    - loop exits and the program moves on to the next – if any – set of instructions after

Coding Examples

For-In-Range Loop

Function (in Python, most similar to general for-loops in many other languages)

```
for j in range (1, 6, 1):
    print (j)
```

- Has the same parameters as the while-loop example
  - start at 1, go to 6, print 'j' each time and increase by 1 during each pass
    - written in a more compact way

Contains all four parts of a general loop:

- Loop counting variable is 'j'
  - variable 'j' is an integer value which is set to one by the first parameter in the parentheses
- Loop condition is for the loop to keep running until the counting variable reaches six, which is the second parameter in the parentheses
- Loop stepping increment is one, as given by the third parameter in the parentheses
  - each time through, add 1 to the counting variable 'j'
- Loop body contains instructions regarding what the program should do during each pass through the loop
  - simply prints the current value of 'j'

For-Loop in Python Using a String Sequence:

```
mySentence = "Programming is fun!"
for eachCharacter in mySentence:
    print(eachCharacter)
```

Steps:

- Start by defining variable, mySentence, and assign it a string value of "Programming is fun!"
- Next enter the for-loop
  - can read this and know exactly what the for-loop will do: "for each character in my sentence, print each character!"
    - do not have to give the for-loop a numeric range or loop counting variable
    - do not have to define the loop condition or loop stepping increment

Coding Examples

- Loop counting variable is the index of each character in the string, starting with an index of [0]
  - python creates its own counting variable in memory to hold this value as the loop is running
  - starts at zero and continues until the last index is reached
  - in this case, the string “Programming is fun!” goes from index [0] to index [18]
    - spaces are indexed like characters
- Loop condition is for the loop to keep running until the last index is reached
  - Python knows how far to go because it can count the number of characters in the string
    - this string has a length of 19, so this loop will run 19 times
  - loop stepping increment is one
    - built into the function by default
    - index value will increase by one during each pass through the loop
- Loop body contains instructions regarding what the program should do during each pass through the loop
  - in this case, the loop body simply prints the character at the current index value
    - a colon is used after the for-statement, and the loop body is indented below that

Character	**P**	**r**	**o**	**g**	**r**	**a**	**m**	**m**	**i**	**n**	**g**	
Index	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
Character	i	s		f	u	n	!					
Index	[12]	[13]	[14]	[15]	[16]	[17]	[18]	Total length of string is 19 characters				

Coding Examples

For-Loop in Python using a List:

```
rainbowColors = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"]
for color in rainbowColors:
    print(color)
```

Steps:

- In this example, we have list, rainbowColors, which contains seven strings
  - one for each of the primary rainbow colors of red to violet
  - "for each color in rainbow colors, print the color name."
  - for-loop iterates through each string in the list and prints that string to the screen until the last index is reached
- Loop counting variable is the index of each string in the list, starting with an index of [0]
  - in this case, the list has seven items, going from index [0] to index [6]
- Loop condition – Python knows how far to go because it can count the number of items in the list, which has a length of seven because there are seven items
  - loop will run seven times
- Loop stepping increment is 1
- Loop body in this case simply prints the string at the current index value
- Will print each string to the screen, one-by-one, like this:
  → red
  → orange
  → yellow
  → green
  → blue
  → indigo
  → violet

Item	"red"	"orange"	"yellow"	"green"	"blue"	"indigo"	"violet"
Index	[0]	[1]	[2]	[3]	[4]	[5]	[6]

Coding Examples

Random Numbers

Modules

random.randint() function:

Simulating the roll of a six-sided game die:

```
import random
rollDie = random.randint(1, 6)
```

Steps:

- In this example, a random integer will be chosen from the values one to six and include one and six
  - only possible choices are 1, 2, 3, 4, 5, or 6

Simulating two rolls of a six-sided game die:

```
import random
die 1 = random.randint(1, 6)
die 2 = radom.randint(1, 6)
rollTotal = die1 + die 2
```

Steps:

- In this example, we generate two random numbers – one for each individual die – and simply add them together to get the total for both

random.random function

```
import random
randomFloat = random.random()
```

Random floating-point greater than one:

```
import random
randomFloat = random.random() * 100
```

Generating integers

```
import random
randomFloat = random.random() * 100
randomInt = round(randomFloat)
```

random.choice function

```
import random
itemsInTreasureChest = [“gems”, “gold coins”, “set of armor”, “spell book”, “magic wand”, “smithing tools”]
RandomItem = random.choice(itemsInTreasureChest)
```

Coding a Story

Project Overview:
You will work with a partner to develop code to create a story with animation for others to enjoy.

Directions:
1. Follow your instructor's directions to identify a partner to work with.

2. Work with your partner to brainstorm the plot and characters for a story. The story should be equivalent to something used in a preschool setting.

3. Consider how the following programming language elements can be used to write a story. Refer to the **Coding Examples Student Handout** as needed.
    - Dialogue between characters
    - Variable
    - Sequence
    - Conditional statements
    - Iteration

4. With your partner, use Python to create your story with at least three animations.

5. Add graphics and sound effects to make your story and animations interesting to viewers.

6. As you develop each piece of the story and animations test, identify and fix any bugs or errors that arise. Testing throughout the process ensures the story will function as intended.

7. Enter the fully developed code into an interpreter and run the code to ensure the outputs are accurate.

8. If there are any errors in your code, identify the trouble spots, address the error and re-test the code.

9. Follow your instructor's directions to view the other stories created by your peers.

10. Once completed, turn in your project according to your instructor's directions.

Coding a Story

Rubric

Description	Possible Points	Your Score
Programming Language Planning & Use: • Animated story includes a variety of programming language elements • Programming language elements function as intended	30	
Concept & Understanding: • Understanding of how programing language can be used to create a story and animation is clear • Effective strategies were used to tell a story using a programming language • Logical thinking was utilized to tell a logical story that corresponds with animations	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	

Additional Comments: