Python

Project Workbook

Student Edition

Presented By

LearnKey®

Python

First Edition

LearnKey creates signature multimedia courseware. LearnKey provides expert instruction for popular computer software, technical certifications, and application development with dynamic video-based courseware and effective learning management systems. For a complete list of courses, visit http://www.learnkey.com.

www.learnkey.com

LearnKey®

Table of Contents

Domain 4

Domain 5

Domain 6

Appendix

Using This Workbook

This project workbook contains practice projects and exercises to reinforce the knowledge you have gained through the video portion of the **Python** course. The purpose of this workbook is twofold. First, get you further prepared to pass the Python exam, and second, to get you job-ready skills and increase your employability in the area of application development using Python.

The projects within this workbook follow the order of the video portion of this course. To save your answers in this workbook, you must first download a copy to your computer. You will not be able to save your answers in the web version. You can complete the workbook exercises as you go through each section of the course, complete several of these at the end of each domain, or complete them after viewing the entire course. The key is to go through these projects to strengthen your knowledge of this course.

Each project is based upon a specific video (or videos) in the course and specific test objectives. The materials you will need for this course include:

- A code editing tool that can build and run Python projects. This course uses Visual Studio Community with the Python tools installed.
- The practice files for these projects.

For Teachers

LearnKey is proud to provide extra support to instructors upon request. For your benefit as an instructor, we also provide an instructor support .zip file containing answer keys, completed versions of the workbook project files, and other teacher resources. This .zip file is available within your learning platform's admin portal.

Notes

- Extra teacher notes, when applicable, are in the Project Details box within each exercise.
- Exam objectives are aligned with the course objectives listed in each project, and project file names correspond with these numbers.
- The Finished folder in each domain has reference versions of each project. These can help you grade projects.
- Short answers may vary but should be similar to those provided in this workbook.
- Teachers may consider asking students to add their initials, student ID, or other personal identifiers at the end of each saved project.
- Refer to your course representatives for further support.

We value your feedback about our courses. If you have any questions, comments, or concerns, please let us know by visiting https://about.learnkey.com.

Best Practices Using LearnKey's Online Training

LearnKey offers video-based training solutions that are flexible enough to accommodate private students and educational facilities and organizations.

Our course content is presented by top experts in their respective fields and provides clear and comprehensive information. The full line of LearnKey products has been extensively reviewed to meet superior standards of quality. Our course content has also been endorsed by organizations such as Certiport, CompTIA®, Cisco, and Microsoft. However, it is the testimonials given by countless satisfied customers that truly set us apart as leaders in the information training world.

LearnKey experts are highly qualified professionals who offer years of job and project experience in their subjects. Each expert has been certified at the highest level available for their field of expertise. This expertise provides the student with the knowledge necessary to obtain top-level certifications in their chosen field.

Our accomplished instructors have a rich understanding of the content they present. Effective teaching encompasses presenting the basic principles of a subject and understanding and appreciating organization, real-world application, and links to other related disciplines. Each instructor represents the collective wisdom of their field and within our industry.

Our Instructional Technology

Each course is independently created based on the manufacturer's standard objectives for which the course was developed.

We ensure that the subject matter is up-to-date and relevant. We examine the needs of each student and create training that is both interesting and effective. LearnKey training provides auditory, visual, and kinesthetic learning materials to fit diverse learning styles.

Course Training Model

The course training model allows students to undergo basic training, building upon primary knowledge and concepts to more advanced application and implementation. In this method, students will use the following toolset:

Pre-assessment: The pre-assessment is used to determine the student's prior knowledge of the subject matter. It will also identify a student's strengths and weaknesses, allowing the student to focus on the specific subject matter he/she needs to improve the most. Students should not necessarily expect a passing score on the pre-assessment as it is a test of prior knowledge.

Video training sessions: Each training course is divided into sessions or domains and lessons with topics and subtopics. LearnKey recommends incorporating all available external resources into your training, such as student workbooks, glossaries, course support files, and additional customized instructional material. These resources are located in the folder icon at the top of the page.

Exercise labs: Labs are interactive activities that simulate situations presented in the training videos. Step-by-step instructions and live demonstrations are provided.

Post-assessment: The post-assessment is used to determine the student's knowledge gained from interacting with the training. In taking the post-assessment, students should not consult the training or any other materials. A passing score is 80 percent or higher. If the individual does not pass the post-assessment the first time, LearnKey recommends incorporating external resources, such as the workbook and additional customized instructional material.

Workbook: The workbook has various activities, such as fill-in-the-blank worksheets, short answer questions, practice exam questions, and group and individual projects that allow the student to study and apply concepts presented in the training videos.

Skills Assessment

Instructions: Rate your skills on the following tasks from 1-5 (1 being needs improvement, 5 being excellent).

Skills	1	2	3	4	5
Evaluate expressions to identify the data types Python assigns to variables.	☐	☐	☐	☐	☒
Perform data and data type operations.	☐	☐	☐	☐	☒
Determine the sequence of execution based on operator precedence.	☐	☐	☐	☒	☐
Select operators to achieve the intended results.	☐	☐	☐	☒	☐
Construct and analyze code segments that use branching statements.	☐	☐	☐	☐	☐
Construct and analyze code segments that perform iteration.	☐	☐	☐	☐	☐
Construct and analyze code segments that perform file input and output operations.	☐	☐	☐	☐	☐
Construct and analyze code segments that perform console input and output operations.	☐	☐	☐	☐	☐
Document code segments.	☐	☐	☐	☐	☐
Construct and analyze code segments that include function definitions.	☐	☐	☐	☐	☐
Analyze, detect, and fix code segments that have errors.	☐	☐	☐	☐	☐
Analyze and construct code segments that handle exceptions.	☐	☐	☐	☐	☐
Perform basic operations by using built-in modules.	☐	☐	☐	☐	☐
Solve complex computing problems by using built-in modules.	☐	☐	☐	☐	☐

Python Video Times

Domain 1	Actual Time
Course Opener	00:02:31
Evaluate Data Types	00:06:36
Convert and Work with Data Types	00:06:56
Operator Sequence and Selection	00:14.33
Total Time	00:30:36

Domain 2	Actual Time
Branching Statements	00:10:32
Iterations	00:15:03
Total Time	00:25:35

Domain 3	Actual Time
File Input and Output Operations	00:14:21
Console Input and Output Operations	00:08:04
Total Time	00:22:25

Domain 4	Actual Time
Document Code Segments	00:06:21
Functions	00:11:44
Total Time	00:18:05

Domain 5	Actual Time
Analyze, Detect, and Fix Errors	00:07:31
Exception Handlers	00:13:14
Total Time	00:20:45

Domain 6	Actual Time
Built-in Modules for Operations	00:06:33
Solve Problems with Built-in Modules	00:12:23
Domain 6 and Final Recaps	00:02:14
Total Time	00:21:14

Python

Domain 1

Presented By

LearnKey®

Fill-in-the-Blanks Domain 1

Instructions: While watching Domain 1, fill in the missing words according to the information presented by the instructor. [References are found in the brackets.]

Evaluate Data Types

1. Variables are containers that store information. [Variables]
2. In Python, the print() command is used to return information. [String Data Type]
3. A floating variable is a variable with a decimal. [Integer and Float Data Types]
4. A Boolean variable only has two possible values: true or false. [Boolean Data Types]

Convert and Work with Data Types

5. Python is a loosely typed language. [Type Casting]
6. Lists are variables which store multiple values. [Constructing Data Structures]
7. Python, like many other languages, is a zero-based language. [Indexing and Slicing Operations]

Operator Sequence and Selection

8. A number inside of quotes is actually a string. [Assignment]
9. A = equal sign is an assignment, while a == equal sign is a comparison. [Comparison]
10. Three logical keywords that are important to Python are and, or, and not. [Logical]
11. A % is a remainder after a division. [Arithmetic]
12. The identity operator is used to determine if two is have the same ID. [Identity]
13. The keyword, in, basically asks, does the variable contain this? [Containment]

Using String Variables and the Print Command

In Python, just like in any other programming language, variables are containers that store information for later use. In Python, variables are loosely typed, meaning that you do not need to declare a variable type (string, number, Boolean, and others) when declaring a variable, as Python will figure out what the variable type should be automatically.

A best practice for variables is to use camel casing; that is, if a variable has two or more words, the first word is lowercase, and the remaining words are uppercase. For example, firstName is a camel casing; Firstname is not.

In simple console apps, which we are building in this course, the **print** statement prints out to the screen whatever is inside the parentheses with the print command. For example, print(firstName prints the contents of the firstName variable.

In this project, you will use what you have learned to declare and then display two string variables. String variables can have any type of data: numbers, letters, symbols, and special characters. The contents of a string variable need to be in double quotes. This tells Python that you are indeed using a string for a variable.

Purpose:

After completing this project, you will be able to declare, assign, and display string variables. Note that for every project in this course, you will want to open the practice files used in your code editor unless told otherwise.

Steps for Completion:

1. Name three rules for naming variables in Python.
   Do not use spaces, capitalize letters, or numbers
2. Open Visual Studio Community or the code editor you are using in this course.
3. From your Session 1 Student folder, open the **Python11.sln** solution file.
4. Double-click the **_11a_Strings.py** file to open it.
5. On the first two lines of the file, declare two variables, firstName and lastName, and assign your firstName and lastName to the two variables, respectively.
6. On the next line, use a print statement and a concatenating symbol to print the lastName, a comma, and the firstName.
7. Run the code. You should see your last name, a comma, and your first name.
8. Close the output window.

Project Details

Project Files:

Python11.sln
_11a_Strings.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 1

Topic: Evaluate Data Types

Subtopics: Variables; String Data Type

Objectives Covered:

1 Operations using Data Types and Operators

1.1 Evaluate expressions to identify the data types Python assigns to variables

1.1.a Str, int, float, and bool

Integer and Float Data Types

In Python, numbers are stored and displayed as an integer or float variable. An integer is a whole number that is positive or negative. A float is a variable with a decimal.

Numbers that will be, or could be, part of a calculation should not be entered as a string in a variable. A string cannot multiply numbers or create or return a calculation. Only strings can be concatenated or linked, so strings and integers cannot be concatenated or linked together in a print statement. When concatenation is used it will combine or link multiple strings to create a new string.

To add an integer to a string in a print statement, a comma needs to be used to add a space in the original string where the integers can be a placed. This will make the integers in the variables changeable within the string.

Purpose:

Upon completing this project, you will be able to add an integer variable and a float variable to a string.

Steps for Completion:

1. If necessary, open the **Python11.sln** solution file from the Domain 1 Student folder in your code editor. Open the **_11a_Numbers.py** file.
2. Above the two print statements, declare these two variables:
   a. price set to **3.95**
   b. widgets set to **5**
3. On the last line of the code, change the concatenating symbols so that the number of widgets will print along with the print statement.
4. Run the code. You should see the following:
   The price of the widget is 3.95
   We have 5 in stock.
5. Close the output window.

Project Details

Project Files:

Python11.sln
_11a_Numbers.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 1
Topic: Evaluate Data Types
Subtopic: Integer and Float Data Types

Objectives Covered:

1 Operations using Data Types and Operators
1.1 Evaluate expressions to identify the data types Python assigns to variables
1.1.a Str, int, float, and bool

Boolean Data Types

A Boolean variable will only return True or False. Python is case-sensitive, so the return word will start with an uppercase letter. The Boolean variables are normally set as a flag to start an action within a program.

Purpose:

Upon completing this project, you will be able to create a Boolean variable and print the return on separate lines.

Steps for Completion:

1. If necessary, open the **Python11.sln** solution file from the Domain 1 Student folder. Open the **_11a_Boolean.py** file.
2. Under the north and south variables, create a Boolean variable named **northwins** that will check to see if north is greater than south.
3. On the next line, create a Boolean variable named **southwins** that will check to see if south is greater than north.
4. Create a print statement that will print if the Boolean variables just created are True or False and have the answers print on separate lines.
5. Run the code. You should see the following:

   Northwins = False

   Southwins = True
6. Close the output window.

Project Details

Project Files:

Python11.sln

_11a_Boolean.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 1

Topic: Evaluate Data Types

Subtopic: Boolean Data Types

Objectives Covered:

1 Operations using Data Types and Operators

1.1 Evaluate expressions to identify the data types Python assigns to variables

1.1.a Str, int, float, and bool

Type Casting

Type casting is casting one type of data as another type of data, using built-in functions. For example, the int function casts a variable as an integer. When converting a float variable to an integer, the numbers after the decimal will not be considered and the numbers before the decimal point will become the integer.

Purpose:

Upon completing this project, you will be able to use type casting to change data types of variables.

Steps for Completion:

1. Open the **Python12.sln** solution file from your Domain 1 Student folder.
2. From the solution, open the **_12a_Type_Casting.py** file.
3. Use type casting to change the widgets variable in the second print statement to a string.
4. On the next line, create a print statement to multiply the **price** variable and the **widgets** variable.
5. On the next line, create a print statement to print the **price** variable as an integer and the **widgets** variable as a float, using the correct concatenating symbol in between the two.
6. Run the code. You should see the following:

   The price of the widget is 3.95

   We have 5 in stock.

   19.75
7. Close the output window.

Project Details

Project Files:

Python12.sln

_12a_Type_Casting.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 1

Topic: Convert and Work with Data Types

Subtopic: Type Casting

Objectives Covered:

1 Operations using Data Types and Operators

1.2 Perform data and data type operations

1.2.a Type casting

Constructing Data Structures

Constructing data structures is the act of building lists which store multiple values. It is not recommended to mix data types in a list. A list can be created and left empty until it is needed. Items in the list can be added or removed as needed. A for loop, covered later, iterates through the items in the list. Often, a print statement is added within the loop with code added to print each value in the list.

Purpose:

Upon completing this project, you will be able to add or remove items from a data structure and sort the items in the list alphabetically.

Steps for Completion:

1. If necessary, open the **Python12.sln** solution file from your Domain 1 Student folder. From the solution, open the **_12b_constructing_data_structures.py** file.
2. Complete the variable declaration for employees through adding the values **Alice**, **Vera**, **Flo**, and **Mel**
3. Run the code. You should see the following:

   Alice
   Vera
   Flo
   Mel
4. Close the output window.
5. Below the for loop, add the code to add an employee named **Belle**
6. On the next line, add the code needed to remove the employee named **Flo**
7. On the next line, add code to sort the employee list.
8. Cut the two lines in the for loop and paste them at the end of the code.
9. Run the code. You should see the following:

   Alice
   Belle
   Mel
   Vera
10. Close the output window.

Project Details

Project Files:

Python12.sln
_12b_constructing_data_structures.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 1

Topic: Convert and Work with Data Types

Subtopic: Constructing Data Structures

Objectives Covered:

1 Operations using Data Types and Operators

1.2 Perform data and data type operations

1.2.c Constructing data structures

Indexing and Slicing Operations

Python is a zero-based list, so the first item in a list is always item zero. Slicing will pull a piece of a list without looking up the index number. A shortcut to slicing from the end of a list is to use -1 to pull the last item from a list or -2 for the second to last item in a list.

Purpose:

Upon completing this project, you will be able to print a statement that will pull the first and last index item in a list.

Steps for Completion:

1. If necessary, open the **Python12.sln** solution file from your Domain 1 Student folder. From the solution, open the **_12c_Indexing_and_Slicing_Operations.py** file.
2. In the existing print statement, replace the comment, #firstregion, with the first region in the regions list.
3. In the same print statement, replace the comment, #firstsales, with the first value in the sales list.
4. On the line below the existing print statement, create a print statement that mimics the previous print statement but prints both the last region and the last sales amount.
5. Run the code. You should see the following:

   Region: North Sales: 30000

   Region: West Sales: 35000
6. Close the output window.

Project Details

Project Files:

Python12.sln

_12c_Indexing_and_Slicing_Operations.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 1

Topic: Convert and Work with Data Types

Subtopic: Indexing and Slicing Operations

Objectives Covered:

1 Operations using Data Types and Operators

1.2 Perform data and data type operations

1.2.b Indexing and slicing operations

Assignment

An assignment is the act of setting a variable to be equal to a value. The assigned value can be a string, number, date, time, Boolean, or list. If two assignments have the same name, the last assignment made will cancel the previous assignments. An assignment must take place before any other operator type can be used.

Purpose:

Upon completing this project, you will understand what creates an assignment and the assignment order.

Steps for Completion:

1. Open the **Python13and14.sln** solution file from your Domain 1 Student folder.
2. From the solution, open the **_13a_14a_Assignments.py** file.
3. Below the current code, assign a value of **42000** to the sales variable.
4. On the next line, create a print statement for the **sales** variable.
5. Run the code.
6. What sales variable was printed?

42000

7. Why did Python print this and not the assigned value earlier in the code?

8. Close the output window.

Project Details

Project Files:

Python13and14.sln,
_13a_14a_Assignments.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 1

Topic: Operator Sequence and Selection

Subtopic: Assignment

Objectives Covered:

1 Operations using Data Types and Operators

1.3 Determine the sequence of execution based on operator precedence

1.3.a Assignment

1.4 Select the appropriate operator to achieve the intended result

1.4.a Assignment

Comparison

A comparison operator uses the same symbols found in mathematics, but any Python symbol that compares items and creates a return is a comparison operator. Python uses a double equal sign (==) to ask if a variable is equal to a variable. An exclamation point (!) means not, so when it is added to a comparison symbol such as !< it will mean that a variable is not less than a variable.

Purpose:

Upon completing this project, you will be able to write comparison print statements using variables.

Steps for Completion:

1. If necessary, open the **Python13and14.sln** solution file from your Domain 1 Student folder. From the solution, open the **_13b_14b_Comparisons.py** file.
2. Under the variables a, b, and c, create comparison print statements:
   a. Is **a** equal to **b**
   b. Is **a** equal to **c**
   c. Is **a** greater than **c**
   d. Is **a** greater than or equal to **c**
   e. Is **a** less than or equal to **b**
   f. Is **a** not equal to **c**
3. Run the code. You should see the following:
   False
   True
   False
   True
   True
   False
4. Close the output window.

Project Details

Project Files:

Python13and14.sln
_13b_14b_Comparisons.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 1

Topic: Operator Sequence and Selection

Subtopic: Comparison

Objectives Covered:

1 Operations using Data Types and Operators

1.3 Determine the sequence of execution based on operator precedence

1.3.b Comparison

1.4 Select operators to achieve the intended results

1.4.b Comparison

Logical

The logical keywords that Python uses are and, or, and not. For a statement to be True when using an and keyword, both sides of the and statement must be True. For a statement to return True when using an or keyword, only one side of the or statement needs to be True. With the not keyword, the condition is checked for the opposite to return True or False.

Purpose:

Upon completing this project, you will be able to write logical operator print statements using variables.

Steps for Completion:

1. If necessary, open the **Python13and14.sln** solution file from your Domain 1 Student folder. From the solution, open the **_13c_14c_Logical.py** file.
2. Under the variables a, b, c, and d, create these logical operator print statements.
   a. Is **a** equal to **c** and **b** not equal to **c**
   b. Is **a** equal to **b** and **b** equal to **c**
   c. Is **a** equal to **b** or **b** equal to **c**
   d. Is **a not** equal to **b** or **b** equal to **c**
3. Run the code. You should see the following:
   True
   False
   False
   True
4. Close the output window.

Project Details

Project Files:

Python13and14.sln
_13c_14c_Logical.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 1

Topic: Operator Sequence and Selection

Subtopic: Logical

Objectives Covered:

1 Operations using Data Types and Operators

1.3 Determine the sequence of execution based on operator precedence

1.3.c Logical

1.4 Select operators to achieve the intended results

1.4.c Logical

Arithmetic

Arithmetic comparison operators use the same symbols found in mathematics, but Python has some additional arithmetic symbols. Python uses a double asterisk (**) as the symbol for an exponent, which will raise a variable to the power of a variable. A percentage symbol (%) is the symbol for a modulus, which is the remainder after a division is executed.

Purpose:

Upon completing this project, you will be able to write arithmetic print statements using variables.

Steps for Completion:

1. If necessary, open the **Python13and14.sln** solution file from your Domain 1 Student folder. From the solution, open the **_13d_14d_Arithmetic.py** file.
2. Under the variables a and b, create these arithmetic print statements
   a. **a** plus **b**
   b. **a** minus **b**
   c. **a** multiplied by **b**
   d. **a** divided by **b**
   e. **a** to the **b** power
   f. **a** divided by **b**
3. Run the code. You should see the following results:
   11
   5
   24
   2.6666666666666665
   512
   2
4. Close the output window.

Project Details

Project Files:

Python13and14.sln
_13d_14d_Arithmetic.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 1

Topic: Operator Sequence and Selection

Subtopic: Arithmetic

Objectives Covered:

1 Operations using Data Types and Operators

1.3 Determine the sequence of execution based on operator precedence

1.3.d Arithmetic

1.4 Select operators to achieve the intended results

1.4.d Arithmetic

Identity

The identity operator is used to determine if two variables have the same ID, but this operator is not commonly used. The keywords is and is not are used in identity operators. If one variable is used to set another, they are considered to have the same identity. If the original variable changes values, the two variables are no longer considered to have the same identity.

Purpose:

Upon completing this project, you will understand the difference between equal and the identity operator.

Steps for Completion:

1. If necessary, open the **Python13and14.sln** solution file from your Domain 1 Student folder. From the solution, open the **_13e_14e_Identity.py** file.
2. Under the existing variables, create these print statements to compare identity operators and equal statements.
   a. Is **a** equal to **b**
   b. Is **a** **b**
   c. Is **northItems** equal to **eastItems**
   d. Is **northItems** **eastItems**
3. Run the code. You should see the following results:
   True
   True
   True
   False
4. Return to the code.
5. Below the existing variables, add code to change the value of a to **5**
6. Run the code. You should now see these results:
   False
   False
   True
   False
7. Close the output window.

Project Details

Project Files:

Python13and14.sln
_13e_14e_Identity.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 1

Topic: Operator Sequence and Selection

Subtopic: Identity

Objectives Covered:

1 Operations using Data Types and Operators

1.3 Determine the sequence of execution based on operator precedence

1.3.e Identity

1.4 Select operators to achieve the intended results

1.4.e Identity

Containment

The containment operator determines if a variable contains specific content. A containment operator is typically used on lists or variables that have a lot of content. The keyword used in a containment operator is the word, in. Containment operators ask, is an item found within a variable?

Purpose:

Upon completing this project, you will be able to write containment operator print statements using variables.

Steps for Completion:

1. If necessary, open the **Python13and14.sln** solution file from your Domain 1 Student folder. From the solution, open the **_13f_14f_Containment.py** file.
2. Under the existing variables, create containment operator print statements to answer these questions:
   a. Is the 25000 amount in the sales list?
   b. Is the word, do, found in the quote string?
   c. Is the word, did, found in the quote string?
3. Run the code. You should see the following results:
   True
   True
   False
4. Close the output window.

Project Details

Project Files:

Python13and14.sln
_13f_14f_Containment.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 1
Topic: Operator Sequence and Selection
Subtopic: Containment

Objectives Covered:

1 Operations using Data Types and Operators
1.3 Determine the sequence of execution based on operator precedence
1.3.f Containment
1.4 Select operators to achieve the intended results
1.4.f Containment

Python

Domain 2

Presented By

LearnKey®

Fill-in-the-Blanks Domain 2

Instructions: While watching Domain 2, fill in the missing words according to the information presented by the instructor. [References are found in the brackets]

Branching Statements

1. Statements to run after an if condition need to be true or false [If]
2. Python has an elif statement, which is short for else. [Elif]
3. If/elif statements always stop after the first match. [Order of If and Elif Statements]
4. When typing an else, each of these statements needs a colon at the end. [Else]
5. A nested if statement is an if statement inside of an if statement. [Nested Conditionals]
6. In a compound conditional statement, multiple conditions are being checked. [Compound Conditionals]

Iterations

7. The most important thing to do with a while loop is to make sure you don't create an endlees loop as that will crash your program. [While]
8. Python is a zero-based language. [For]
9. A break statement exits the loop and runs the code following the loop. [Break]
10. A continue statement skip a turn in a loop. [Continue]
11. The pass keyword is used when a condition is defined or a function is built. [Pass]
12. Conditional compounds involve the keywords, and, or, or not. [Nested Loops and Conditional Compounds]

If

If statements are used in most programming languages to check to see if a condition is true and then perform one or more actions if that condition is indeed true. When writing an if statement, any actions or messages need to be indented to be assigned to the if statement. Without the indent it will be its own statement and not part of a condition.

Purpose:

Upon completing this project, you will understand how an if statement works and use an if statement with a code example.

Steps for Completion:

1. Open the **Python21.sln** solution file from your Domain 2 Student folder.
2. From the solution, open the **_21a_If.py** file.
3. Below the variable, create an if statement that will return the message **Gold Customer** if annual sales are greater than or equal to **500000**
4. Create a message that says **Thank you for your business** that will display no matter what the amount is for the annual sales.
5. Run the code. You should see the following:

   Gold Customer

   Thank you for your business
6. Close the output window.
7. Adjust the **annualSales** variable to **300000**
8. Run the code again. Only the thank-you message should be displayed.
9. Close the output window.

Project Details

Project Files:

Python21.sln

_21a_If.py

Estimated Completion Time:

10-15 minutes

Video Reference:

Domain 2

Topic: Branching Statements

Subtopic: If

Objectives Covered:

2 Flow Control with Decisions and Loops

2.1 Construct and analyze code segments that use branching statements

2.1.a If

Elif

Elif statements in Python are the equivalent of an else if statement in most other programming languages. The else if statement is used to check for a condition when the original if condition returns false. It is important to consider the order of elif statements as an if block is done as soon as a condition finds a match.

As is the case with an if statement, a colon is needed at the end of an elif statement and the actions to take place under each condition need to be indented.

Purpose:

Upon completing this project, you will be able to create two elif statements and determine which statement will be returned.

Steps for Completion:

1. If necessary, open the **Python21.sln** solution file from your Domain 2 Student folder. From the solution, open the **_21b_Elif.py** file.
2. Below the if statement, create an elif statement that will return the message **Silver Customer** if annual sales are greater than or equal to **300000**
3. Create another elif statement that will return the message **Bronze Customer** if annual sales are greater than or equal to **100000**
4. Run the code. You should see the following:

   Silver Customer

   Thank you for your business
5. Close the output window.

Project Details

Project Files:

Python21.sln
_21b_Elif.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 2
Topic: Branching Statements
Subtopic: Elif

Objectives Covered:

2 Flow Control with Decisions and Loops
2.1 Construct and analyze code segments that use branching statements
2.1.b Elif

Order of If and Elif Statements

When working with number ranges greater than or equal to, the order of statements must be the largest number to the smallest number. When working with number ranges less than or equal to, the order of statements should be the smallest number to the largest number. An if/elif statement will run the first condition and if the condition is True the If block will exit. If the if/elif statements are not ordered correctly the user will receive the wrong return.

Purpose:

Upon completing this project, you will understand the required order of if/elif statements.

Steps for Completion:

1. If necessary, open the **Python21.sln** solution file from your Domain 2 Student folder. From the solution, open the **_21b_Elif_bad.py** file.
2. Look through the code. What type of customer should display?
   Bronze Coustomer
3. Run the code. Which customer displays? Bronze Coustomer
4. Why?
   compares the specified value in the variable whether .... Is the given value greater than or equal to the value specified in the comparison attributes
5. Close the output window.
6. Rearrange the code to where the correct customer will display based on the annualSales amount.
7. Run your code. You should see the following:
   Gold Customer
   Thank you for your business
8. Close the output window.

Project Details

Project Files:

Python21.sln
_21b_elif_bad.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 2

Topic: Branching Statements

Subtopic: Order of If and Elif Statements

Objectives Covered:

2 Flow Control with Decisions and Loops

2.1 Construct and analyze code segments that use branching statements

2.1.b Elif

Else

The else statement runs if none of the if/elif statements return True. The else, like the if and elif statements, needs a colon at the end of it.

Purpose:

Upon completing this project, you will be able to create a else statement that will run when an if/elif statement's conditions are not met.

Steps for Completion:

1. If necessary, open the **Python21.sln** solution file from your Domain 2 Student folder. From the solution, open the **_21c_Else.py** file.
2. Under the last elif statement, create an **else** statement for annual sales under 100000 that returns the message: **Up and Coming Customer**
3. Change the annual sales to **50000**
4. Run the code. You should see the following:

   Up and Coming Customer

   Thank you for your business
5. Close the output window.

Project Details

Project Files:

Python21.sln

_21c_Else.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 2

Topic: Branching Statements

Subtopic: Else

Objectives Covered:

2 Flow Control with Decisions and Loops

2.1 Construct and analyze code segments that use branching statements

2.1.c Else

Nested Conditionals

A nested condition is an if statement within another if statement. This is often necessary for when one condition, if true, then triggers another set of conditions that need to be checked.

Purpose:

Upon completing this project, you will be able to nest an if statement with two different return messages.

Steps for Completion:

1. If necessary, open the **Python21.sln** solution file from your Domain 2 Student folder. From the solution, open the **_21d_Nested.py** file.
2. Within the elif statement that prints a silver customer, create an if statement with these conditions:
   a. If the region is in the North, a message to **Send a snowboard** will print.
   b. For any other region, a message to **Send a baseball bat** will print.
3. Run the code. You should see the following:
   Silver Customer
   Send a snowboard
   Thank you for your business
4. Return to your code editor.
5. Change the region to **West**
6. Run the code. You should see the following:
   Silver Customer
   Send a baseball bat
   Thank you for your business
7. Close all output windows.

Project Details

Project Files:

Python21.sln
_21d_Nested.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 2
Topic: Branching Statements
Subtopic: Nested Conditionals

Objectives Covered:

2 Flow Control with Decisions and Loops
2.1 Construct and analyze code segments that use branching statements
2.1.d Nested and compound conditionals

Compound Conditionals

A compound conditional statement is used when multiple conditions may need to be met. The and/or portions of the statement are used to match those conditions. Whereas a nested condition is a condition inside of another, a compound condition does not need to be nested.

Purpose:

Upon completing this project, you will be able to create a compound conditional print statement using variables.

Steps for Completion:

1. If necessary, open the **Python21.sln** solution file from your Domain 2 Student folder. From the solution, open the **_21d_Compound.py** file.
2. Replace the comment, # add compound here, with a compound conditional statement that will print the message: **Bronze Customer and first-time prize winner** when the annual sales are over 100000 and the customer is new.
3. Set the annualSales variable to **200000** and the newCustomer variable to **True**
4. Run the code. You should see the following:

   Bronze Customer and first-time prize winner

   Thank you for your business
5. Close the output window.

Project Details

Project Files:

Python21.sln

_21d_Compound.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 2

Topic: Branching Statements

Subtopic: Compound Conditionals

Objectives Covered:

2 Flow Control with Decisions and Loops

2.1 Construct and analyze code segments that use branching statements

2.1.d Nested and compound conditionals

While

A while loop will run code while the condition used in a while statement is true. An input statement, often used in a while loop, will capture a user's input information and make it a variable. It is important to make sure the while loop will end at some point as failing to do so will cause an endless loop and eventually crash your program.

As is the case with an if statement, the while condition must end with a colon and the loop (the actions) to take place must be indented under the while condition.

Purpose:

Upon completing this project, you will be able to create a while loop that will allow a user to guess an answer until the answer is right.

Steps for Completion:

1. Open the **Python22.sln** solution file from your Domain 2 Student folder.
2. Within the solution, open the **_22a_While.py** file.
3. Under the variables, create a while statement with a condition that the capitalGuess variable does not equal Riga.
4. For the loop, add the following:
   a. Add 1 to the numberOfGuesses variable.
   b. Set the capitalGuess variable to be an input with the message **Guess again** and a space.
5. Under the while loop, add a print statement with this text: **You guessed it. Riga is the capital of Latvia.**
6. Add another print statement with the message **It took you** plus the number of guesses plus **guesses**.
7. Run the code.
8. Type **Vilnius** and press the Enter key.
9. Type **Riga** and press the Enter key. You should see the following display:
   What is the capital of Latvia? Vilnius
   Guess again. Riga
   You guessed it. Riga is the capital of Latvia.
   It took you 2 guesses.
10. Close the output window.

Project Details

Project Files:

Python22.sln
_22a_While.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 2
Topic: Iterations
Subtopic: While

Objectives Covered:

2 Flow Control with Decisions and Loops
2.2 Construct and analyze code segments that perform iteration
2.2.a While

For

Unlike a while loop, which is a conditional loop, a for loop is set to run a specific number of times. For example, you may want code to run to represent each month of a year. For that, you would want to create a loop that runs 12 times. The amount of times a loop will run is specified within a range.

The one caveat with a for loop within Python is that the last number in the range does not run. Thus, if you want a loop to run 12 times, 13 needs to be the last number in the range.

Purpose:

Upon completing this project, you will be able to create a for loop that will create a monthly increase sales goal to display.

Steps for Completion:

1. If necessary, open the **Python22.sln** solution file from your Domain 2 Student folder. From the solution, open the **_22b_For.py** file.
2. Below the variables, replace the comment, # add for loop and logic, with a for loop that does the following:
   a. Uses a variable named **monthlyGoal** and set the range to be large enough for all 12 months.
   b. Within the loop, a variable named monthlySalesGoal is defined and set to the initialSalesGoal plus the total of the monthlyGoal multiplied by the multiplier.
   c. Within the loop, a message with the text **Your sales goal for month** and the monthly goal plus the word **is** and the monthlySalesGoal is displayed.
3. Run the code. You should see the following:

   Your sales goal for month 1 is 20100
   Your sales goal for month 2 is 20200
   Your sales goal for month 3 is 20300
   Your sales goal for month 4 is 20400
   Your sales goal for month 5 is 20500
   Your sales goal for month 6 is 20600
   Your sales goal for month 7 is 20700
   Your sales goal for month 8 is 20800
   Your sales goal for month 9 is 20900
   Your sales goal for month 10 is 21000
   Your sales goal for month 11 is 21100
   Your sales goal for month 12 is 21200
   Good luck with your goals.
4. Close the output window.

Project Details

Project Files:

Python22.sln
_22b_For.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 2
Topic: Iterations
Subtopic: For

Objectives Covered:

2 Flow Control with Decisions and Loops
2.2 Construct and analyze code segments that perform iteration
2.2.b For

Break

A break statement will exit the loop when a set condition is met and can be used for any type of loop. The break statement will keep a loop from becoming an endless loop.

Purpose:

Upon completing this project, you will be able to use a break statement to end an if loop.

Steps for Completion:

1. If necessary, open the **Python22.sln** solution file from your Domain 2 Student folder. From the solution, open the **_22c_Break.py** file.
2. Within the while loop, add a break statement in the appropriate place to break the loop should the numberOfGuesses variable be greater than three.
3. Adjust the last print statement to print only when the numberOfGuesses variable is less than or equal to three.
4. Run the code.
5. Enter three incorrect guesses (not Riga). After three incorrect guesses, you should see the following:
   You guessed incorrectly three times. Game over.
6. Close the output window.

Project Details

Project Files:

Python22.sln,
_22c_Break.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 2
Topic: Iterations
Subtopic: Break

Objectives Covered:

2 Flow Control with Decisions and Loops
2.2 Construct and analyze code segments that perform iteration
2.2.c Break

Continue

Whereas a break statement stops a loop, the continue statement allows a turn in a loop to be skipped when a condition is met during the loop. The loop then continues with the next iteration.

Purpose:

Upon completing this project, you will be able to use a continue statement to skip a section of a for loop.

Steps for Completion:

1. If necessary, open the **Python22.sln** solution file from your Domain 2 Student folder. From the solution, open the **_22d_Continue.py** file.
2. Within the if statement, add the following:
   a. A **continue** statement that will allow month 6 of the sales goal to be skipped.
   b. A print statement with the text: **No goal for month 6** when the month is the sixth month in the for loop.
3. Run the code. You should see the following:

   Your sales goal for month 1 is 20100
   Your sales goal for month 2 is 20200
   Your sales goal for month 3 is 20300
   Your sales goal for month 4 is 20400
   Your sales goal for month 5 is 20500
   No goal for month 6
   Your sales goal for month 7 is 20700
   Your sales goal for month 8 is 20800
   Your sales goal for month 9 is 20900
   Your sales goal for month 10 is 21000
   Your sales goal for month 11 is 21100
   Your sales goal for month 12 is 21200
   Good luck with your goals.
4. Close the output window.

Project Details

Project Files:

Python22.sln
_22d_Continue.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 2
Topic: Iterations
Subtopic: Continue

Objectives Covered:

2 Flow Control with Decisions and Loops
2.2 Construct and analyze code segments that perform iteration
2.2.d Continue

Pass

The pass keyword is a placeholder for an action to take place based on a condition but the action is not known. This is more of a strategic keyword to remind a developer that an action is needed once it is determined. Pass is not used often in Python but you do need to know this for the exam.

Purpose:

Upon completing this project, you will be able to use the pass keyword to hold a place for future actions to be added to a condition within an if loop.

Steps for Completion:

1. If necessary, open the **Python22.sln** solution file from your Domain 2 Student folder. From the solution, open the **_22e_Pass.py** file.
2. Remove the print statement for the bronze customer setting.
3. In place of the bronze customer statement, add a keyword to serve as a placeholder action for the condition of annualSales >= 100000.
4. Run the code. You should see the following:

   Thank you for your business
5. Close the output window.

Project Details

Project Files:

Python22.sln
_22e_Pass.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 2
Topic: Iterations
Subtopic: Pass

Objectives Covered:

2 Flow Control with Decisions and Loops
2.2 Construct and analyze code segments that perform iteration
2.2.e Pass

Nested Loops and Conditional Compounds

A nested loop is a loop within a loop. A conditional compound uses the keywords and, or, or not to add multiple criteria to a condition within a loop.

Purpose:

Upon completing this project, you will be able to create nested loops and a conditional compound statement.

Steps for Completion:

1. If necessary, open the **Python22.sln** solution file from your Domain 2 Student folder. From the solution, open the **_22f_Nested_and_Conditional_loops.py** file.
2. In the existing if condition, add a criterion to include offMonth being true.
3. After the print statement for the sales goal, add a nested for loop that does the following:
   a. Uses weeklyGoal as a variable and loops four times.
   b. Prints a statement that says **Your goal for week** plus the weeklyGoal value plus **is** plus the monthlySalesGoal divided by 4.
4. Run the code. The first few lines of the output should look like this:

   Your sales goal for month 1 is 20100
   Your goal for week 1 is 5025.0
   Your goal for week 2 is 5025.0
   Your goal for week 3 is 5025.0
   Your goal for week 4 is 5025.0
5. Close the output window.

Project Details

Project Files:

Python22.sln
_22f_Nested_and_Conditional_loops.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 2
Topic: Iterations
Subtopic: Nested Loops and Conditional Compounds

Objectives Covered:

2 Flow Control with Decisions and Loops
2.2 Construct and analyze code segments that perform iteration
2.2.f Nested loops and loops that include conditional compounds

Python

Domain 3

Presented By

LearnKey®

Fill-in-the-Blanks Domain 3

Instructions: While watching Domain 3, fill in the missing words according to the information presented by the instructor. [References are found in the brackets.]

File Input and Output Operations

1. The open function takes two arguments: the _____, and the _____. [Open]
2. The moment you are done with a _____ you should close it. [Close]
3. The _____ command reads one line at a time. [Read]
4. When writing to an existing file, the default behavior is that the file will be _____. [Write]
5. Python has a library called _____, short for operating system. [Check Existence]
6. Once a file you are using becomes _____, you will want to delete it. [Delete]
7. The way to avoid the problem of a file not _____ properly is to use a with statement in conjunction with working with files. [With Statement]

Console Input and Output Operations

8. To keep an app running until the user enters a correct value, use a _____ loop. [Read Input from Console]
9. Adding the _____ in front of quotation marks tells Python the text within the quotes will be formatted. [Print Formatted Text]
10. Python itself runs in a _____ environment. [Command Line Arguments]

Open

Python data is stored in text or comma separated value files. The open statement requires a file and mode argument. The mode arguments are: r for read, w for write, a for append, and b for binary.

Purpose:

Upon completing this project, you will be able to create code that will open a file in Python.

Steps for Completion:

1. Open the **Python31.sln** solution file from your Domain 3 Student folder.
2. From the solution, open the **_31a_Open.py** file.
3. Set the workFile variable to an open method that opens the **jackets.txt** file in read mode.
4. Set the workFileContents variable to the read method on the workFile variable.
5. Run the code. You should see the following:

   Product, Price
   Big Down Jacket, 100
   Medium Down Jacket, 80
   Regular Jacket, 40
6. Close the output window.

Project Details

Project Files:

Python31.sln
31a_Open.py
jackets.txt

Estimated Completion Time:

5 minutes

Video Reference:

Domain 3

Topic: File Input and Output Operations

Subtopic: Open

Objectives Covered:

3 Input and Output Operations

3.1 Construct and analyze code segments that perform file input and output operations

3.1.a Open

Close

Any open file or program will use CPU and memory, so it is good practice to close a file when it is completed with its task. Open, read, and close are built-in functions or actions. An action at times needs an argument added to create an action. The argument is placed in parentheses and the parentheses can be empty but still show an argument can be assigned.

Purpose:

Upon completing this project, you will be able to close a file after it performs a function.

Steps for Completion:

1. If necessary, open the **Python31.sln** solution file from your Domain 3 Student folder. From the solution, open the **_31b_Close.py** file.
2. Below the current code, add a statement to close the file represented by the workFile variable.
3. Below that code, add a print statement indicating that the file has been closed.
4. Run the code. The file should have opened, been read, printed, and then the close message printed.
5. Close the output window.

Project Details

Project Files:

Python31.sln
_31b_Close.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 3

Topic: File Input and Output Operations

Subtopic: Close

Objectives Covered:

3 Input and Output Operations

3.1 Construct and analyze code segments that perform file input and output operations

3.1.b Close

Read

The read command will read an entire file, while the readline command reads a line at a time, starting with the first line and reading subsequent lines each time the readline command is used.

Purpose:

Upon completing this project, you will be able to write code to have a file read and printed, both for an entire file and for a line at a time.

Steps for Completion:

1. If necessary, open the **Python31.sln** solution file from your Domain 3 Student folder. From the solution, open the **_31c_Read.py** file.
2. Replace the text, read file, with a command that will read the entire jackets.txt file.
3. Remove the read command and replace it with the read line command that will read just the first line of the jackets file.
4. Run the code. You should see the following:

   Product, Price
   Big Down Jacket, 100
   Medium Down Jacket, 80
   Regular Jacket, 40
5. Close the output window.
6. Comment out the two lines after the line containing the open command.
7. Remove the comments from the last four lines of the code.
8. Replace the two instances of the text, read line, with a command that will read a single line from the jackets.txt file.
9. Run the code. You should see the following:

   Product, Price

   Big Down Jacket, 100
10. Add a line of code to close the file using the workFile variable.
11. Close the output window.
12. Save your changes.

Project Details

Project Files:

Python31.sln,
_31c_Read.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 3

Topic: File Input and Output Operations

Subtopic: Read

Objectives Covered:

3 Input and Output Operations

3.1 Construct and analyze code segments that perform file input and output operations

3.1.c Read

Write

The write command, which writes to a file, will create a file if the file has not already been created. Be careful with the write command, as it will overwrite information already in a file and the previous information will be lost.

Purpose:

Upon completing this project, you will be able to write to and create a new file using the write command.

Steps for Completion:

1. If necessary, open the **Python31.sln** solution file from your Domain 3 Student folder. From the solution, open the **_31d_Write.py** file.
2. Set the writeFile variable to open the **log.txt** file in write mode.
3. Set the toLog variable to an input method with the question: **What do you want to write to the log?**
4. On the line following the input, add code to invoke the write method and have it write the contents of the toLog variable.
5. Run the code.
6. When prompted to type something, type **Happy Trails** and press the Enter key.
7. Close the output window.

Project Details

Project Files:

Python31.sln
_31d_Write.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 3

Topic: File Input and Output Operations

Subtopic: Write

Objectives Covered:

3 Input and Output Operations

3.1 Construct and analyze code segments that perform file input and output operations

3.1.d Write

Check Existence

You have seen that Python, when writing to a file, by default will overwrite a file if it exists. This is true if a file is opened with a 'w' attribute. However, a file opened with an 'a' attribute will have any text added to it append to the existing text within a file. To check to see whether a file is in existence, the isfile method is used. In order to use the method, the operating system library, named os, needs to be imported.

Purpose:

Upon completing this project, you will be able to write a code that will check to see if a file exists and have different actions if a file exists or does not exist.

Steps for Completion:

1. If necessary, open the **Python31.sln** solution file from your Domain 3 Student folder. From the solution, open the **_31e_Check_Existence.py** file.
2. Replace the comment, need library**,** with code to import the os library.
3. Replace the comment, if the log file exists, append to it**,** with an if statement that will check to see if the **log.txt** file exists and if so, open the file in append mode.
4. Replace the comment, otherwise, open a new log, with an else statement that will open the **log.txt** file in write mode.
5. Run the code.
6. Type: **Show me** and press the Enter key.
7. Close the output window.
8. Run the code.
9. Type: **the money!!** and press the Enter key. You should see this at the end of the output:
   Show me the money!!
10. Close the output window.

Project Details

Project Files:

Python31.sln,
_31e_Check_Existence.py
log.txt

Estimated Completion Time:

10 minutes

Video Reference:

Domain 3

Topic: File Input and Output Operations

Subtopic: Check Existence

Objectives Covered:

3 Input and Output Operations

3.1 Construct and analyze code segments that perform file input and output operations

3.1.e Check existence

Delete

Obsolete files should be deleted in order to regain space on one's hard drive. The os library needs to be imported in order to delete a file.

Purpose:

Upon completing this project, you will be able to write code to delete files using the delete command.

Steps for Completion:

1. If necessary, open the **Python31.sln** solution file from your Domain 3 Student folder. From the solution, open the **_31f_Delete.py** file.
2. Replace the comment with an if statemtent that will do the following:
   a. Check to see if the **log_old.txt** file exists.
   b. If it exists, delete it.
3. Run the code. You should see a message indicating that the log_old file has been removed.
4. Close the output window.
5. Run the code again. You should see a message that there was no log_old file to remove.
6. Close the output window.

Project Details

Project Files:

Python31.sln
_31f_Delete.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 3

Topic: File Input and Output Operations

Subtopic: Delete

Objectives Covered:

3 Input and Output Operations

3.1 Construct and analyze code segments that perform file input and output operations

3.1.f Delete

With Statement

When working with files, the with statement will close a file if there is a problem with the file or when the close statement has not been written. Under normal use of the with statement, a file will close once the indented portion of the code has been run.

It is important for files to be closed and removed from memory when the file work is complete. The with statement written into the code is the safeguard to ensuring a file is closed properly.

Purpose:

Upon completing this project, you will be able to write the with statement into the code to ensure a file is closed.

Steps for Completion:

1. If necessary, open the **Python31.sln** solution file from your Domain 3 Student folder. From the solution, open the **_31g_With_statement.py** file.
2. Replace the existing comment with a with statement to open the **log.txt** file in write mode.
3. Run the code.
4. Type: **Happy Trails** and press the Enter key.
5. Close the output window.
6. Check the log.txt file in the project folder. The message should appear.

Project Details

Project Files:

Python31.sln
_31g_With_statement.py
log.txt

Estimated Completion Time:

5 minutes

Video Reference:

Domain 3

Topic: File Input and Output Operations

Subtopic: With Statement

Objectives Covered:

3 Input and Output Operations

3.1 Construct and analyze code segments that perform file input and output operations

3.1.g With statement

Read Input from Console

Apps can need user input. This type of app is easy to test through a console. A blank variable needs to be created to allow the user input to print.

Purpose:

Upon completing this project, you will be able to read a user's input that meets the conditions of a loop from a console.

Steps for Completion:

1. In the Domain 3 Student folder, open the **Python32.sln** solution file.
2. Double-click the **_32a_ReadInput.py** file to open it.
3. Below the existing comment, create a variable named **location** that is empty.
4. Create a loop that does the following:
   a. Sets the location variable to an input with the statement, **Enter HQ, North, or South for a location**
   b. Set the loop to repeat if the user does not enter a proper choice for a location.
5. Run the code.
6. Type: **West** and press the Enter key. You should be prompted to enter a location.
7. Type: **North** and press the Enter key. You should see the North location displayed.
8. Close the output window.

Project Details

Project Files:

Python32.sln
_32a_ReadInput.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 3

Topic: Console Input and Output Operations

Subtopic: Read Input from Console

Objectives Covered:

3 Input and Output Operations

3.2 Construct and analyze code segments that perform console input and output operations

3.2.a Read input from console

Print Formatted Text

When using the print formatted text feature, it is easier to combine numbers and strings into one line of text. Python will know printed text will be formatted text when the letter f is set in front of the quotation marks and any variables to be printed are in curly brackets. This allows pieces of text with different formats to display in a single print statement.

Purpose:

Upon completing this project, you will be able to write code that will create print-formatted text and display numbers and strings in one line of text.

Steps for Completion:

1. If necessary, open the **Python32.sln** solution file from your Domain 3 Student folder. From the solution, open the **_32b_Print_Formatted_Text.py** file.
2. Change the print statement on line 3 to a variable named **msg** with the price printed as a fixed decimal to two places.
3. Change the print statement on the last line of the code to use the print formatted text feature to display the number of widgets in stock in the string. There should be no commas left when this is complete.
4. Add a line of code to print the msg variable.
5. Run the code. The message and number will print in a single print statement.
6. Close the output window.

Project Details

Project Files:

Python32.sln

_32b_Print_Formatted_Text.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 3

Topic: Console Input and Output Operations

Subtopic: Print Formatted Text

Objectives Covered:

3 Input and Output Operations

3.2 Construct and analyze code segments that perform console input and output operations

3.2.b Print formatted text

Use of Command-Line Arguments

Python runs in a command line environment. Python can be worked with directly by typing Python, and help() is an interactive help section that will give help with any Python command, keyword, module, or topic. Pressing the Enter key will move through the information one line at a time, while pressing the Spacebar key will move through several lines of text at a time. To exit the help section, type: quit, and to exit Python completely type: quit().

Note: This project will require your student folder.

Purpose:

Upon completing this project, you will be able to use the command line to access help information in Python.

Steps for Completion:

1. Launch a command prompt
2. Navigate to the folder containing the **31d-Write** project.
3. Run the **Python _31d_Write.py** command.
4. At the prompt, type **testing123** and press the Enter key.
5. In File Explorer, navigate to the folder containing the **31d-Write** project. A log file should be present.
6. Open the log file. It should have the text you added earlier in this project.
7. Return to the command prompt.
8. Run the **Python** command.
9. Open the help utility.
10. Get help on the keyword **with**
11. Quit the help.
12. Close the command prompt window.

Project Details

Project Files:

_31d_Write.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 3

Topic: Console Input and Output Operations

Subtopic: Command Line Arguments

Objectives Covered:

3 Input and Output Operations

3.2 Construct and analyze code segments that perform console input and output operations

3.2.c Use of command-line arguments

Python

Domain 4

Presented By

LearnKey®

Fill-in-the-Blanks Domain 4

Instructions: While watching Domain 4, fill in the missing words according to the information presented by the instructor. [References are found in the brackets.]

Document Code Segments

1. In Python, _____ is far more important than _____. [Indentation and White Space]

2. Comments in Python start with a pound symbol, or a hash symbol, or _____ symbol. [Comments and Document Settings]

3. The Pydoc feature automatically generates _____ on Python modules. [Pydoc]

Functions

4. The _____ symbol in Python turns a number into a format with two decimal places. [Call Signatures]

5. Default values can be used as function _____. [Default Values]

6. A return keyword defines the end of a _____. [Return]

7. The def keyword in Python is analogous to the _____ keyword seen in other programming languages. [Def]

8. The pass keyword is merely a _____ for something to define later. [Pass]

Indentation and White Space

Indented statements are statements attached to the last non-indented statement, like a condition, for example. If the statement is not indented it is a separate statement. Indents in Python are analogous to text inside of curly brackets in most other programming languages.

A backslash after quotations will move the code to the next line in the code window without making changes to the data displayed to the user. The code is still grouped together but easier to read in the code window.

White space is a blank line between code types. It makes the code easier to read and find conditions.

Purpose:

Upon completing this project, you will be able to use indentation to attach statements to conditions and white space to organize code.

Steps for Completion:

1. Open the **Python41.sln** solution file from your Domain 4 Student folder.
2. From the solution, open the **_41a_Indentation_White_Space.py** file.
3. Fix the if-elif-elif statement using indents so that the statement works properly.
4. For the last print statement in the code, split the print up to print over three lines with line breaks after **{firstName}** and **region,"**
5. Run the code. You should see the following:

   Gold Customer
   Thank you for your business
   Your sales representative is Tony,
   you are in the West region,
   and you had 500000 in sales last year. Thanks!
6. Close the output window.

Project Details

Project Files:

Python41.sln
_41a_Indentation_White_Space.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 4
Topic: Document Code Segments
Subtopic: Indentation and White Space

Objectives Covered:

4 Code Documentation and Structure
4.1 Document code segments

4.1.a Use of indentation and white space

Comments, Documentation Strings, and Pydoc

Comments allow an explanation of code and can tell Python to ignore code temporarily when troubleshooting or testing the code. Comments will start with a pound/hashtag symbol and will not print on the display screen when the code is run. The shortcut Ctrl+K followed by Ctrl+C will create a code comment, while Ctrl+K followed by Ctrl+U will uncomment the code.

Pydoc is a Python library module that automatically generates documentation on Python keywords and is run from the Python command prompt. The Pydoc is added to your Python path in Windows so you can run the Python app from any folder.

Purpose:

Upon completing this project, you will be able to use comments to document and test the code. You will also be able to use Pydoc to get information on a keyword.

Steps for Completion:

1. If necessary, open the **Python41.sln** solution file from your Domain 4 Student folder. From the solution, open the **_41b_Comments_and_Documentation_Strings.py** file.
2. Add a comment above the variables with the text: **set test variables**
3. Add a comment for the if statement with the message: **sales logic**
4. Comment out the two print statements at the bottom of the code.
5. Run the code. The Gold Customer status should be the only message.
6. Close the output window.
7. Open a command prompt.
8. Run the Pydoc command that will show information on the pass statement in Python.
9. Close the command prompt window.

Project Details

Project Files:

Python41.sln

_41b_Comments_and_Documentation_Strings.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 4

Topic: Document Code Segments

Subtopic: Comments and Documentation Strings

Objectives Covered:

4 Code Documentation and Structure

4.1 Document code segments

4.1.b Comments and documentation strings

Call Signatures

A call signature can reference a user function. A function is a sequence of actions written once that can be referred to multiple times without repeating the written code. A function that has arguments uses the arguments to create an action. A variable is then created to call/reference the function adding data to the arguments of the function. The function must be written before the variables to call the function actions/arguments to the variable. The function can then be called using its name and any arguments needed are typed in parentheses.

Purpose:

Upon completing this project, you will be able to create a variable that will call a function to process data and create a return.

Steps for Completion:

1. Open the **Python42.sln** solution file from your Domain 4 Student folder.
2. From the solution, open the **_42a_Call_Signatures.py** file.
3. Replace the comment for creating the first order total with a variable named **firstOrderTotal** that will use the subtotal function with an order amount of **300** and a sales tax rate of **.08**
4. Replace the comment for creating the second order total with a variable named **secondOrderTotal** that will use the subtotal function with an order amount of **400** and a sales tax rate of **.06**
5. Replace the comment for creating the third order total with a variable named **thirdOrderTotal** that will use the subtotal function with an order amount of the thirdOrderAmount variable and a sales tax rate of the thirdTax variable.
6. Run the code.
7. Enter **250** for the amount and **.07** for the sales tax. The output should return the following:
   Your subtotal for the first order is 324.00
   Your subtotal for the second order is 424.00
   Your subtotal for the third order is 267.50
8. Close the output window.

Project Details

Project Files:
Python42.sln,
_42a_Call_Signatures.

Estimated Completion Time:
10 minutes

Video Reference:
Domain 4
Topic: Functions
Subtopic: Call Signatures

Objectives Covered:
4 Code Documentation and Structure
4.2 Construct and analyze code segments that include function definitions
4.2.a Call signatures

Default Values

A default value for a variable will allow a variable value to be set automatically, Though it is a default value, that same variable can be overriden with another value.

Purpose:

Upon completing this project, you will be able to create a default value that can be adjusted by the user.

Steps for Completion:

1. If necessary, open the **Python42.sln** solution file from your Domain 4 Student folder. From the solution, open the **_42b_Default_Values.py** file.
2. Adjust the salesTax variable to have a default value of **.08**
3. Remove the sales tax rate from the firstOrderTotal variable. This will force the variable to use the default rate.
4. Replace the comment for the fourth order total with a variable called **fourthOrderTotal** that calls the subtotal function with a order amount of **800** and no sales tax assigned.
5. Run the code.
6. Enter **250** for the amount and **.07** for the sales tax rate. You should see the following output:

   Your subtotal for the first order is 324.00

   Your subtotal for the second order is 424.00

   Your subtotal for the third order is 267.50

   Your subtotal for the fourth order is 864.00
7. Close the output window.

Project Details

Project Files:

Python42.sln

_42b_Default_Values.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 4

Topic: Functions

Subtopic: Default Values

Objectives Covered:

4 Code Documentation and Structure

4.2 Construct and analyze code segments that include function definitions

4.2.b Default values

Return

The return keyword is used to take a value from a function and return it to code outside of a function for later use. For example, a function could be told to return a subtotal which is then used later in a print statement. A return keyword without a variable is typically used to return a print statement at the end of a function.

Purpose:

Upon completing this project, you will understand and use code to create two different return types.

Steps for Completion:

1. If necessary, open the **Python42.sln** solution file from your Domain 4 Student folder. From the solution, open the **_42c_Return.py** file.
2. Add a return statement to the subtotal function. Set the return to be the answer to the **subtotal** calculation.
3. Replace the comment to add an orderMsg function with a function that will return a print statement with the text **Thank you for your order(s)**
4. Below the existing code, add a call to the orderMsg function you built.
5. Run the code.
6. Enter **250** for the amount and **.07** for the sales tax. You should see the following output:

   Your subtotal for the first order is 324.00
   Your subtotal for the second order is 424.00
   Your subtotal for the third order is 267.50
   Your subtotal for the fourth order is 864.00
   Thank you for your order(s)
7. Close the output window.

Project Details

Project Files:

Python42.sln
_42c_Return.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 4
Topic: Functions
Subtopic: Return

Objectives Covered:

4 Code Documentation and Structure
4.2 Construct and analyze code segments that include function definitions
4.2.c Return

Def

Def is a keyword that allows a function to be defined. A function is a sequence of actions written once that can be referred to multiple times without repeating the written code. A function that has arguments uses the arguments to create an action. The arguments are set in parentheses.

Purpose:

Upon completing this project, you will be able to create/define a function.

Steps for Completion:

1. If necessary, open the **Python42.sln** solution file from your Domain 4 Student folder. From the solution, open the **_42d_Def.py** file.
2. Define a function named **fullName** that does the following:
   a. Takes the arguments **firstName** and **lastName**
   b. Prints the lastName, a comma and a space, and the firstName.
   c. Prints the term **Initials:**, a space, and then the first initial and the last initial.
3. Below the function, make two calls to the function, using **Mighty Mouse** for a name in the first call and **Tom Servo** for a name in the second call.
4. Run the code. You should see the following output:
   Mouse, Mighty
   Initials: MM
   Servo, Tom
   Initials: TS
5. Close the output window.

Project Details

Project Files:

Python42.sln
_42d_def.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 4
Topic: Functions
Subtopic: Def

Objectives Covered:

4 Code Documentation and Structure
4.2 Construct and analyze code segments that include function definitions
4.2.d Def

Pass

The pass keyword is a placeholder, in a defined function, for actions that will be added later. It is good practice to attach a comment to a pass keyword so that anyone looking at the code will know why the pass keyword is there and what needs to be done.

Purpose:

Upon completing this project, you will understand how the pass keyword works in a function.

Steps for Completion:

1. If necessary, open the **Python42.sln** solution file from your Domain 4 Student folder. From the solution, open the **_42e_Pass.py** file.
2. Below the fullName function, create a function named **personInfo**
3. Within a function, add a pass statement and a comment with the text **need to know where to get this info**
4. Run the code. You should see the same output as you did in the previous project.
5. Close the output window.

Project Details

Project Files:

Python42.sln
_42e_Pass.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 4
Topic: Functions
Subtopic: Pass

Objectives Covered:

4 Code Documentation and Structure
4.2 Construct and analyze code segments that include function definitions
4.2.e Pass

Python

Domain 5

Presented By

LearnKey®

Fill-in-the-Blanks Domain 5

Instructions: While watching Domain 5, fill in the missing words according to the information presented by the instructor. [References are found in the brackets.]

Analyze, Detect, and Fix Errors

1. Any time you see a _____ squiggly line in code there is a syntax error. [Syntax Errors]
2. Most programmers agree logic errors are harder to catch than _____ errors. [Logic Errors]
3. One common runtime error occurs when an app tries to read a file that does not _____. [Runtime Errors]

Exception Handlers

4. In exception handling, an except statement is _____. [Try and Except]
5. Else has to come right after the _____ part of the try/except block. [Else]
6. To display a message to the user while in a try/except block, go to the _____ of the code, type the keyword, finally, and a colon. [Finally]
7. Raise will tell Python to raise the _____ error if one tries to divide by 0. [Raise]

Syntax Errors

Syntax errors in Python, and for that matter in any other language, are the equivalent of grammar errors in English. The common causes of syntax errors are typing mistakes, punctuation mistakes, or entering items in the wrong places. Some errors are designated by a red squiggly line under the text. Other errors are designated on the display after the code is run.

One correction can fix many other errors, so a best practice is to correct errors from the top of the code working down. Hovering over the start of an error will give you information on what the program was expecting and why it was designated as an error.

Purpose:

Upon completing this project, you will be able to fix syntax errors.

Steps for Completion:

1. Open the **Python51.sln** solution file from your Domain 5 Student folder.
2. From the solution, open the **_51a_Syntax_Errors.py** file.
3. Correct the syntax errors in the code (there are four).
4. Run the code. You should see the following:

   Bronze Customer

   Thank you for your business
5. Briefly describe the four syntax errors you found and fixed:
6. Close the output window.

Project Details

Project Files:

Python51.sln

_51a_Syntax_Errors.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 5

Topic: Analyze, Detect, and Fix Errors

Subtopic: Syntax Errors

Objectives Covered:

5 Troubleshooting and Error Handling

5.1 Analyze, detect, and fix code segments that have errors

5.1.a Syntax errors

Logic Errors

Logic errors have no alert to make the error stand out and can be difficult to detect. Testing for scenarios and considering how a user will use the app is important in reducing logic errors.

Purpose:

Upon completing this project, you will be able to correct a logic error in a while loop.

Steps for Completion:

1. If necessary, open the **Python51.sln** solution file from your Domain 5 Student folder. From the solution, open the **_51b_Logic_Errors.py** file.
2. Run the code.
3. Type **riga** and press the Enter key. Even though the answer is correct, you get notified of an incorrect guess.
4. Close the output window.
5. Change the logic of the code so that the user input is converted to uppercase and the condition checks for **RIGA** as the correct answer.
6. Run the code.
7. Type **riga** and press the Enter key. You should be told that the answer is correct.
8. Close the output window.

Project Details

Project Files:

Python51.sln
_51b_Logic_Errors.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 5
Topic: Analyze, Detect, and Fix Errors
Subtopic: Logic Errors

Objectives Covered:

5 Troubleshooting and Error Handling
5.1 Analyze, detect, and fix code segments that have errors
5.1.b Logic errors

Runtime Errors

Runtime errors occur when an app runs, and they can crash the program. A common runtime error occurs when an app tries to read a file that has not been created, or a set calculation cannot be calculated.

Purpose:

Upon completing this project, you will understand runtime errors.

Steps for Completion:

1. If necessary, open the **Python51.sln** solution file from your Domain 5 Student folder. From the solution, open the **_51c_Runtime_Errors.py** file.
2. Run the code and enter **5** for the first number and **0** for the next number.
3. Why did the calculation create a runtime error?

4. Can the runtime error be fixed at runtime?

5. Close the output window.

Project Details

Project Files:

Python51.sl
_51c_Runtime_Errors.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 5
Topic: Analyze, Detect, and Fix Errors
Subtopic: Runtime Errors

Objectives Covered:

5 Troubleshooting and Error Handling
5.1 Analyze, detect, and fix code segments that have errors
5.1.c Runtime errors

Try and Except

Python has a mechanism that will catch errors and allow the programmer to handle errors and what the user will see. The try/except statement tells Python to try an action and, if an error occurs, perform a different action. This can prevent an app from crashing should a runtime error occur.

Purpose:

Upon completing this project, you will be able to write a try/except statement that will control an error code the user will see.

Steps for Completion:

1. Open the **Python52.sln** solution file from the Domain 5 Student folder.
2. Double-click the **_52ab_try_except.py** file to open it.
3. Write a try/except statement that will try to perform the existing print statement within the code. Make the except statement print this: **This did not work. You cannot divide by zero**
4. Run the code.
5. Enter **5** for the first number and **0** for the next number. A message indicating that the calculation did not work should display.
6. Close the output window.
7. Run the code.
8. Enter **10** for the first number and **2** for the second number. You should get an answer of **5**
9. Close the output window.

Project Details

Project Files:

Python52.sln
_52ab_try_except.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 5
Topic: Exception Handlers
Subtopic: Try and Except

Objectives Covered:

5 Troubleshooting and Error Handling
5.2 Analyze and construct code segments that handle exceptions
5.2.a Try
5.2.b Except

Else

The else statement is added to the try/except block to add code to run if the try statement runs successfully. The else statement must be written in the code right after the except statement or it will not run in the try/except block. An else statement will not run if the try/except block fails.

Purpose:

Upon completing this project, you will be able to write an else statement into a try/except block.

Steps for Completion:

1. If necessary, open the **Python52.sln** solution file from your Domain 5 Student folder. From the solution, open the **_52c_Else.py** file.
2. Add an else statement to the existing try/except block with the print message: **You successfully used the division feature in Python**
3. Run the code.
4. Enter the numbers **5** and **2**
5. Press the Enter key. You should see the answer (2.5) and a message that the division feature was used successfully.
6. Close the output window.
7. Run the code again, this time using the numbers **5** and **0.** You should see the except message.
8. Close the output window.

Project Details

Project Files:

Python52.sln,
_52c_Else

Estimated Completion Time:

10 minutes

Video Reference:

Domain 5

Topic: Exception Handlers

Subtopic: Else

Objectives Covered:

5 Troubleshooting and Error Handling

5.2 Analyze and construct code segments that handle exceptions

5.2.c Else

Finally

The finally keyword will run code regardless of the try/except block outcome. This is a contrast to the else keyword, which only runs if the try of a try/catch block is successful.

Purpose:

Upon completing this project, you will be able to use the finally keyword in a try/except block to display a message regardless of the try/except block outcome.

Steps for Completion:

1. If necessary, open the **Python52.sln** solution file from your Domain 5 Student folder. From the solution, open the **_52d_Finally.py** file.
2. Add a finally message: **Thank you for playing**
3. Run the code.
4. Enter the numbers **5** and **2**
5. Press the Enter key. The answer should display along with both the else and finally statements.
6. Close the output window.

Project Details

Project Files:

Python52.sln
_52d_Finally.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 5
Topic: Exception Handlers
Subtopic: Finally

Objectives Covered:

5 Troubleshooting and Error Handling
5.2 Analyze and construct code segments that handle exceptions
5.2.d Finally

Raise

The raise keyword will allow a programmer to raise or show an error message. This is often done for training and testing purposes. The raise keyword will stop an app and the rest of the code from running, including any keywords as part of a try/catch/else/finally block.

Unit testing allows users to take sections of Python source code and test its functionality. Python has a built-in module called unittest that allows one to take functions and test them in an environment that is isolated from the source code within an application.

Purpose:

Upon completing this project, you will be able to use the raise keyword in a try/except block to allow an error to display.

Steps for Completion:

1. If necessary, open the **Python52.sln** solution file from your Domain 5 Student folder. From the solution, open the **_52e_Raise.py** file.
2. Add the following code to the except statement:
   a. Check to see if b = 0
   b. If b = 0, raise an error and display this print statement: **This did not work**
3. Run the code.
4. Enter **5** and **0** for the two numbers.
5. Press the Enter key. The error should display and none of the messages should print after the error.
6. Close the output window.

Project Details

Project Files:

Python52.sln
52e_Raise.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 5
Topic: Exception Handlers
Subtopic: Raise; Unit Test

Objectives Covered:

5 Troubleshooting and Error Handling
5.2 Analyze and construct code segments that handle exceptions
5.2.e Raise
5.3 Perform unit testing
5.3.a Unittest, functions, and methods

Python

Domain 6

Presented By

LearnKey®

Fill-in-the-Blanks Domain 6

Instructions: While watching Domain 6, fill in the missing words according to the information presented by the instructor. [References are found in the brackets.]

Built-in Modules for Operation

1. In Python, built-in math _____ such as pi, sine, cosine, or tangent can be found in the math module. [Math Module]

2. The strftime function is a built-in function that takes two _____: what to convert, and how to convert it. [Datetime]

3. You can write _____ code without specifically importing the io module. [IO]

4. The sys module gives you access to _____ and _____ system information. [Sys]

5. The os module allows us to use Python to perform _____tasks. [Os]

6. The os.path, in the os module, is used to help find files and folders on _____ paths. [Os.path]

7. Python has a built-in random library that warehouses random number _____. [Random Library]

Solve Problems with Built-in Modules

8. The operator symbol for raising a number to a power is a _____. [Math]

9. The function, strptime takes a _____ and converts it to a _____. [Datetime Class]

10. For a game of guess the number, use the _____ module in the random library. [Random]

IO

Python IO is short for input/output. The IO module is the default module for code input and output, so it does not always need to be imported. The IO module is a built-in module and default essential to the functioning of Python. The IO module's main use is to allow files to be opened, read, and written to, often using user input.

Purpose:

Upon completing this project, you will understand the IO module and how to use its functions.

Steps for Completion:

1. If necessary, open the **Python61.sln** solution file from your Domain 6 Student folder. From the solution, open the **_61a_io.py** file.
2. Replace the comment starting with **use with statement** with a with statement that opens the log.txt file in write mode to a variable named writeFile.
3. Replace the comment starting with **write user input** with a method that will write the user input from the previous line to the writeFile variable.
4. Run your code.
5. When you see the input message, type **This works without the io module** and select the Enter key on the keyboard.
6. Close the output window.
7. Open the containing folder and log file. The typed message should be there in a file called log.txt.

Project Details

Project Files:

Python61.sln,
_61a_io.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 6

Topic: Built-in Modules for Operations

Subtopic: IO

Objectives Covered:

6 Operations using Modules and Tools
6.1 Perform basic operations by using built-in modules
6.1.a IO

Os

The built-in os (operating system) module allows Python to be used in performing operating system tasks, many of which have a syntax similar to what you would find in a command prompt in Windows.

Purpose:

Upon completing this project, you will be able to understand and use the built-in os module.

Steps for Completion:

1. If necessary, open the **Python61.sln** solution file from your Domain 6 Student folder. From the solution, open the **_61b_os.py** file.
2. Replace the comment **add the os module** with a statement to import the **os** module.
3. Replace the comment **add a directory with the name the user entered** with a command to create a directory with the name being the text to user input from the previous line.
4. Run the code.
5. Enter your first name as the folder name. When you press the Enter key, you should see a message stating that the directory has been created.
6. Open the containing folder, and the file created with your first name should be in the folder.
7. Close the output window.

Project Details

Project Files:

Python61.sln

_61b_os.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 6

Topic: Built-in Modules for Operations

Subtopic: Os

Objectives Covered:

6 Operations using Modules and Tools

6.1 Perform basic operations by using built-in modules

6.1.b Os

Os.path

The os.path is a specific piece of the os module that is used to find files and folders on specific paths.

Purpose:

Upon completing this project, you will be able to use the os.path to find files and folders and write or append the folder through code.

Steps for Completion:

1. If necessary, open the **Python61.sln** solution file from your Domain 6 Student folder. From the solution, open the **_61c_ospath.py** file.
2. Replace the two existing comments with an if statement that will check for the **log.txt** file and do the following:
   a. If there is a file, it will append the file.
   b. If there is not a file, it will write to the file.
3. Run the code.
4. Type **Success is** and press the Enter key.
5. Open the containing folder and the log file. The success message should be in the file.
6. Close the output window.
7. Re-run the code.
8. Type **fleeting** and press the Enter key.
9. Open the containing folder and the log file. You should see the text: Success is fleeting.
10. Close the output window.

Project Details

Project Files:

Python61.sln
_61c_ospath.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 6
Topic: Built-in Modules for Operations
Subtopic: Os.path

Objectives Covered:

6 Operations using Modules and Tools
6.1 Perform basic operations by using built-in modules
6.1.c Os.path

Sys

The built-in sys module is used to import built-in system functions. The sys module gives access to display and process system information. The sys module will display the information, allowing the code to keep running, but it will not stop the app like the raise keyword does.

Purpose:

Upon completing this project, you will be able to use the sys module to read a file.

Steps for Completion:

1. If necessary, open the **Python61.sln** solution file from your Domain 6 Student folder. From the solution, open the **_61d_sys.py** file.
2. Replace the comment **add the sys module** with a statement to import the sys module.
3. Run the code. You should see the output of a text file.

Project Details

Project Files:

Python61.sln

_61d_sys.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 6

Topic: Built-in Modules for Operations

Subtopic: Sys

Objectives Covered:

6 Operations using Modules and Tools

6.1 Perform basic operations by using built-in modules

6.1.d Sys

Using a Command Prompt

In Python, many commands that use the sys module can also be run in the command prompt. To use the command prompt to run Python commands, you first need to make sure Python is installed, and then, you need to be in the directory containing the files you wish to run.

Purpose:

Upon completing this project, you will be able to use a command prompt to run Python commands.

Steps for Completion:

1. If necessary, open the **Python61.sln** solution file from your Domain 6 Student folder. From the solution, open the **_61d_sys.py** file.
2. Open a command prompt.
3. Navigate to the folder containing the _61d_sys.py file you are using in this project.
4. Run the Python command to open the interactive Python app.
5. Run each command within the _61d_sys.py file in the command prompt window.
6. Close the command prompt window.

Project Details

Project Files:

Python61.sln
_61d_sys.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 6

Topic: Built-in Modules for Operations

Subtopic: Sys

Objectives Covered:

6 Operations using Modules and Tools
6.1 Perform basic operations by using built-in modules
6.1.d Sys

Math

The Python math module has built-in math functions. For math functions to be available, the math module must be imported. To become familiar with built-in modules, go through the lists of available built-in functions. The Intellisense information box will display as you use the built-in math functions.

Some examples of math functions include the pi function, which returns pi, the ceil function, which returns a round up to the next integer from a decimal number, and the floor function, which returns a round down to an integer from a decimal number.

Purpose:

Upon completing this project, you will be able to use the math module to set basic math functions in a variable.

Steps for Completion:

1. Open the **Python62.sln** solution file from your Domain 6 Student folder.
2. From the solution, open the **_62a_Math.py** file.
3. Run the code that will show the floating absolute for the variable y. You should see this: **-12.0**
4. Run the code that will show the ceiling for x. You should see this: **4**
5. Run the code that will show the floor for x. You should see this: **3**
6. Change y to **-12.5**.
7. Run the code that will show the truncated values of x and y. You should see **3** and **-13** for values.
8. Change y to **-13**
9. Run the code that will show a floating modulus when y is divided by x. You should see **-1** for the answer.
10. Run the code that will show the mantissa and exponent of x. The result should be **-2** and **0.75**
11. If possible, leave the file open for the next project.

Project Details

Project Files:

Python62.sln,
_62a_Math.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 6

Topic: Solve Problems with Built-in Modules

Subtopic: Math

Objectives Covered:

6 Operations using Modules and Tools
6.2 Solve complex computing problems by using built-in modules
6.2.a Math

Math Problems

Part of working with the math module is to use the module and its functions to get calculations on data. For example, the math module has the functions necessary to calculate the area of a circle, which is pi * radius * radius.

Purpose:

Upon completing this project, you will be able to use the Python math module to calculate answers to math problems.

Steps for Completion:

1. If necessary, open the **Python62.sln** solution file from your Domain 6 Student folder.
2. If necessary, from the solution, open the **_62a_Math.py** file.
3. If necessary, set x to **3** and y to **-13**.
4. Run the code that will show the result of x * y and whether the total is not a number. The answers should read **False** and **-39**.
5. Run the code that will show both the square root of x and the integer of the square root of x. The answers should be **1.7** and **1**, respectively.
6. Run the code that will show y raised to the power of x. The answer should be **-2197**.
7. Run the code that will show pi and the area of a circle with a radius of x. The results should be **3.14159** and **28.27**, respectively.

Project Details

Project Files:

Python62.sln
_62a_Math.py

Estimated Completion Time:

10 minutes

Video Reference:

Domain 6

Topic: Solve Problems with Built-in Modules

Subtopic: Math

Objectives Covered:

6 Operations using Modules and Tools
6.2 Solve complex computing problems by using built-in modules
6.2.a Math

Datetime

In Python, to work with date and time possibilities, the datetime needs to be imported. The strftime function allows you to style the date to month, date, and year; the time is styled to hours, minutes, and seconds.

Purpose:

Upon completing this project, you will understand how Python's language displays dates and times, and you will be able to use the datetime module.

Steps for Completion:

1. If necessary, open the **Python62.sln** solution file from your Domain 6 Student folder. From the solution, open the **_62b_DateTime.py** file.
2. Replace the comment **add the function to return the current date and time** with a call to function that will return the current day and time.
3. Replace the comment **todayWithTime in hh:mm:ss format** with the todayWithTime variable, formatted in hh:mm:ss format.
4. Replace the comment **add the function to return the day of the week** with the function that returns the day of the week.
5. Run the code. The output should look similar to the following:

   2019-06-30 10:33:06.782696

   The current time is 10:33:06

   1
6. Close the output window.

Project Details

Project Files:

Python62.sln

_62b_DateTime.py

Estimated Completion Time:

5-10 minutes

Video Reference:

Domain 6

Topic: Built-in Modules for Operations

Subtopic: Datetime

Objectives Covered:

6 Operations using Modules and Tools

6.2 Solve complex computing problems by using built-in modules

6.2.b Datetime

Random

Python has a built-in library with multiple types of random number modules. A random number library is used heavily in programming and playing games.

Purpose:

Upon completing this project, you will be able to use the random library to generate a set group of random numbers.

Steps for Completion:

1. If necessary, open the **Python62.sln** solution file from your Domain 6 Student folder. From the solution, open the **_612g_Random.py** file.
2. Make sure the for loop is commented out.
3. Run the code that will show an odd number between 1 and 10.
4. Run the code that will display five random numbers between 0 and 100.
5. Run the code that will show a single car from a list of cars.
6. Run the code that will rearrange an existing list of cars.
7. Ensure that all the code is commented out.
8. Save the file.
9. Close the output window.

Project Details

Project Files:

Python62.sln,
_62c_Random.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 6

Topic: Built-in Modules for Operations

Subtopic: Random

Objectives Covered:

6 Operations using Modules and Tools
6.2 Solve complex computing problems by using built-in modules
6.2.c Random

Random Problems

Python has a built-in library with multiple types of random number modules. A random number library is used heavily in programming and playing games.

Purpose:

Upon completing this project, you will be able to build a guessing game that will use the random numbers module.

Steps for Completion:

1. If necessary, open the **Python62.sln** solution file from your Domain 6 Student folder. From the solution, open the **_62c_Random.py** file.
2. Replace the comment **run this five times** with code that will end the loop after five iterations.
3. Replace the code **a random number between 1 and 10** with the method used to generate a random number between 1 and 10.
4. Uncomment the for loop.
5. If necessary, comment out the code after the for loop.
6. Run the code to play the game.
7. Close the output window.
8. Close all open files and your development tool.

Project Details

Project Files:

Python62.sln
_62c_Random.py

Estimated Completion Time:

5 minutes

Video Reference:

Domain 6

Topic: Solve Problems with Built-in Modules

Subtopic: Random

Objectives Covered:

6 Operations using Modules and Tools
6.2 Solve complex computing problems by using built-in modules
6.2.c Random

Python

Appendix

Presented By

LearnKey®

Glossary

Term	Definition
AND	A logical keyword where both sides of the comparison must be true in order for the statement to be true.
Arithmetic Operator	A tool used inside Python to perform basic mathematical operations.
Assignment	An event which takes place when the user sets a variable to be equal to a string, number, Boolean operator, list, or date.
Boolean Variable	A type of variable with two possible values: true or false. A Boolean variable is often used to determine a course of action within a program.
Break Statement	A statement used to stop or get out of a loop.
Class	A reference type which is used to encapsulate data and can contain nested types.
Comment	A tool used in Python to explain what code does and to tell Python to ignore code temporarily.
Comparison Operator	An operator which is used inside Python to compare arguments.
Continue Keyword	A keyword which is used to skip a turn in a loop where a user is displaying a message or running an action a set number of times.
Datetime Module	The module used in Python to import dates and times.
Def Keyword	A keyword used in Python to begin and define a function.
Default Value	A value or item in a function which is set by the user to be constant.
Elif Statement	An if statement which can check for more than one condition. It is designed to stop after the first match.
Else Statement	A statement used in conjunction with if and elif statements. An else statement provides an option in case if all other options fail.
Finally Keyword	A keyword which is used to deliver a final message to a user regardless of whether an argument was successful or not.
Floating Variable	A type of number variable with a decimal. A floating variable is used for storage purposes.
For Loop	A loop which will act a set number of times before canceling.
Function	A routine which is built by the user, that allows a user not to be required to repeat code up and down through a program.
Identity Operator	An operator in Python which is used to determine if two variables have the same ID.
If Statement	A tool used in Python to instigate action. In the event that something is true, it will perform a designated action.
Indentation	The process of moving code to the right in order to make it easier to read, and also to complete if statements.
Io Module	The input/output (io) module is the default module for input and output stream code in Python 3.0.
List	A versatile method for storing data. A list does not need to have a defined length at the start.
Logic Error	A type of error in Python which occurs when an argument is set incorrectly. The only way around this error is to test the code thoroughly.
Loosely Typed Language	A coding language where the user does not need to specify a data type when declaring variables. Python is a loosely typed language.
Math Function	A function in Python which is used to perform mathematical calculations.
Nested Loop	A loop which is placed inside of another loop.
Nested Statement	An if statement that is placed inside of another if statement.
NOT	The logical keyword which takes a condition set by the user and checks for its opposite.
Object	An item that can be named by a variable. It is the part of code that receives an action.
OR	A logical keyword where only one of the two possibilities need to be true in an argument.
Os Module	The operating system (os) module allows the user to perform operating system tasks, such as create a folder.
Os.path	A piece of the os module, the os.path is used to help users to find files and folders on specific paths.
Pass Keyword	A keyword which is used as a placeholder for a condition or a function until the user knows what they wish to do with the condition or function.
Pydoc	A feature in Python which acts as a library module. It automatically generates documentation on Python modules or keywords.

Term	Definition
Raise Keyword	A type of keyword which is used to display an error message when an exception is caused.
Random Library	A tool in Python which is used to warehouse all kinds of random number modules.
Return Keyword	A keyword which defines the end of a function with an order to return back to regular set code.
Runtime Error	A error type which only occurs when an app runs and often results in the app crashing.
String Variable	A type of variable in Python which is used to store items and information.
Syntax Error	An error in programming code which is caused either by a typo or by trying to use an item incorrectly.
Sys Module	A module used to import built-in system functions which can be used in Python.
Try/Except Statement	A statement used to deal with exception handling in Python.
Variable	Containers which store information that can be used later.
While Loop	A loop which proposes an argument where if something is true or false it will run a designated code.
White Space	The empty lines of code found in Python. It can be utilized to make code easier to read.

Python Lesson Plan

Approximately 20 hours of videos and projects.

Domain 1 - Operations using Data Types and Operators [approximately 4.25 hours of videos and projects]			
Lesson	**Lesson Topic and Subtopics**	**Objectives**	**Workbook Projects and Files**
Pre-Assessment Assessment time - 00:16:00	Operations using Data Types and Operators: Pre-Assessment		
Lesson 1 Video time - 00:02:47 Workbook time - 00:00:00	Course Opener Introduction How to Study for This Exam		
Lesson 2 Video time - 00:06:36 Workbook time - 00:25:00	Evaluate Data Types Variables String Data Type Integer and Float Data Types Boolean Data Types	1.1 Evaluate expressions to identify the data types Python assigns to variables 1.1a Str, int, float, and bool data types	Using String Variables and the Print Command – pg. 11 Python11.sln _11a_Strings.py Integer and Float Data Types – pg. 12 Python11.sln _11a_Numbers.py Boolean Data Types – pg. 13 Python11.sln _11a_Boolean.py
Lesson 3 Video time - 00:06:56 Workbook time - 00:25:00	Convert and Work With Data Types Type Casting Constructing Data Structures Indexing and Slicing Operations	1.2 Perform data and data type operations 1.2a Type casting 1.2b Constructing data structures 1.2c Indexing and slicing operations	Type Casting – pg. 14 Python12.sln _12a_Type_Casting.py Constructing Data Structures – pg. 15 Python12.sln _12b_constructing_data_structures.py Indexing and Slicing Operations – pg. 16 Python12.sln _12c_Indexing_and_Slicing_Operations.py
Lesson 4 Video time - 00:13:37 Workbook time - 00:50:00	Operator Sequence and Selection Assignment Comparison Logical Arithmetic Identity Containment Domain 1 Exam Tips	1.3 Determine the sequence of execution based on operator precedence 1.3a Assignment 1.3b Comparison 1.3c Logical 1.3d Arithmetic 1.3e Identity 1.3f Containment 1.4 Select operators to achieve the intended results 1.4a Assignment 1.4b Comparison 1.4c Logical 1.4d Arithmetic 1.4e Identity 1.4f Containment	Assignment – pg. 17 Python13and14.sln _13a_14a_Assignments.py Comparison – pg. 18 Python13and14.sln _13b_14b_Comparisons.py Logical – pg. 19 Python13and14.sln _13c_14c_Logical.py Arithmetic – pg. 20 Python13and14.sln _13a_14a_Assignments.py Identity – pg. 21 Python13and14.sln _13a_14a_Identity.py Containment – pg. 22 Python13and14.sln _13a_14a_Containment.py
Post-Assessment Assessment time - 01:08:00	Operations using Data Types and Operators: Post-Assessment		

Domain 2 - Flow Control with Decisions and Loops [approximately 3.5 hours of videos and projects]			
Lesson	**Lesson Topic and Subtopics**	**Objectives**	**Workbook Projects and Files**
Pre-Assessment Assessment time - 00:16:00	Flow Control with Decisions and Loops: Pre-Assessment		
Lesson 1 Video time - 00:10:32 Workbook time - 00:50:00	Branching Statements If Elif Order of If and Elif Statements Else Nested Conditionals Compound Conditionals	2.1 Construct and analyze code segments that use branching statements 2.1a If 2.1b Elif 2.1c Else 2.1d Nested and compound conditionals	If – pg. 25 Python21.sln _21a_If.py Elif – pg. 26 Python21.sln _21b_Elif.py Order of If and Elif Statements – pg. 27 Python21.sln _21b_elif_bad.py Else – pg. 28 Python21.sln _21c_Else.py Nested Conditionals – pg. 29 Python21.sln _21d_Nested.py Compound Conditionals – pg. 30 Python21.sln _21d_Compound.py
Lesson 2 Video time - 00:12:53 Workbook time - 00:45:00	Iterations While For Break Continue Pass Nested Loops and Conditional Compounds Domain 2 Exam Tips	2.2 Construct and analyze code segments that perform iteration 2.2a While 2.2b For 2.2c Break 2.2d Continue 2.2e Pass 2.2f Nested loops and loops that include conditional compounds	While – pg. 31 Python22.sln _22a_While.py For – pg. 32 Python22.sln _22b_For.py Break – pg. 33 Python22.sln _22c_Break.py Continue – pg. 34 Python22.sln _22d_Continue.py Pass – pg. 35 Python22.sln _22e_Pass.py Nested Loops and Conditional Compounds – pg. 36 Python22.sln _22f_Nested_and_Conditional_loops
Post-Assessment Assessment time - 00:48:00	Flow Control with Decisions and Loops: Post Assessment		

Domain 3 - Input and Output Operations [approximately 3.25 hours of videos and projects]			
Lesson	**Lesson Topic and Subtopics**	**Objectives**	**Workbook Projects and Files**
Pre-Assessment Assessment time - 00:12:00	Input and Output Operations: Pre-Assessment		
Lesson 1 Video time - 00:14:21 Workbook time - 00:50:00	File Input and Output Operations Open Close Read Write Check Existence Delete With Statement	3.1 Construct and analyze code segments that perform file input and output operations 3.1a Open 3.1b Close 3.1c Read 3.1d Write 3.1e Check existence 3.1f Delete 3.1g With statement	Open – pg. 39 Python31.sln _31a_Open.py Close – pg. 40 Python31.sln _31b_Close.py Read – pg. 41 Python31.sln _31c_Read.py Write – pg. 42 Python31.sln _31d_Write.py Check Existence – pg. 43 Python31.sln log.txt _31e_Check_Existence.py Delete – pg. 44 Python31.sln _31f_Delete.py With Statement – pg. 45 Python31.sln _31g_With_statement.py log.txt
Lesson 2 Video time - 00:08:04 Workbook time - 00:25:00	Console Input and Output Operations Read Input from Console Print Formatted Text Command Line Arguments Domain 3 Exam Tips	3.2 Construct and analyze code segments that perform console input and output operations 3.2a Read input from console 3.2b Print formatted text (string.format() method, f-String method) 3.2c Use of command-line arguments	Read Input from Console – pg. 46 Python32.sln _32a_ReadInput.py Print Formatted Text – pg. 47 Python32.sln _32b_Print_Formatted_Text.py Use of Command Line Arguments – pg. 48 Python31.sln _31-dWrite.py
Post-Assessment Assessment time - 00:40:00	Input and Output Operations: Post-Assessment		

Domain 4 - Code Documentation and Structure [approximately 2.75 hours of videos and projects]			
Lesson	**Lesson Topic and Subtopics**	**Objectives**	**Workbook Projects and Files**
Pre-Assessment Assessment time - 00:12:00	Code Documentation and Structure: Pre-Assessment		
Lesson 1 Video time - 00:06:21 Workbook time - 00:10:00	Document Code Segments Indentation and White Space Comments and Documentation Strings Pydoc	4.1 Document code segments 4.1a Use of indentation and white space 4.1b Comments and documentation strings 4.1c Pydoc	Indentation and White Space – pg. 51 Python41.sln _41a_Indentation_White_Space.py Comments and Documentation Strings, and Pydoc – pg. 52 Python41.sln _41b_Comments_and_Documentation_Strings.py
Lesson 2 Video time - 00:11:44 Workbook time - 00:45:00	Functions Call Signatures Default Values Return Def Pass Domain 4 Exam Tips	4.2 Construct and analyze code segments that include function definitions 4.2a Call signatures 4.2b Default values 4.2c Return 4.2d Def 4.2e Pass	Call Signatures – pg. 53 Python42.sln _42a_Call_Signatures.py Default Values – pg. 54 Python42.sln _42b_Default_Values.py Return – pg. 55 Python42.sln _42c_Return.py Def – pg. 56 Python42.sln _42d_def.py Pass – pg. 57 Python42.sln _42e_Pass.py
Post-Assessment Assessment time - 00:40:00	Code Documentation and Structure: Post-Assessment		

Domain 5 - Troubleshooting and Error Handling [approximately 2.75 hours of videos and projects]			
Lesson	**Lesson Topic and Subtopics**	**Objectives**	**Workbook Projects and Files**
Pre-Assessment Assessment time - 00:16:00	Troubleshooting and Error Handling: Pre-Assessment		
Lesson 1 Video time - 00:07:31 Workbook time - 00:20:00	Analyze, Detect, and Fix Errors Syntax Errors Logic Errors Runtime Errors	5.1 Analyze, detect, and fix code segments that have errors 5.1a Syntax errors 5.1b Logic errors 5.1c Runtime errors	Syntax Errors – pg. 60 Python51.sln _51a_Syntax_Errors.py Logic Errors – pg. 61 Python51.sln _51a_Syntax_Errors.py Runtime Errors – pg. 62 Python51.sln _51c_Runtime_Errors.py
Lesson 2 Video time - 00:13:14 Workbook time - 00:30:00	Exception Handlers Try and Except Else Finally Raise Domain 5 Exam Tips	5.2 Analyze and construct code segments that handle exceptions 5.2a Try 5.2b Except 5.2c Else 5.2d Finally 5.2e Raise 5.3 Perform unit testing 5.3a Unittest, functions, and methods	Try and Except – pg. 63 Python52.sln _52ab_try_except.py Else – pg. 64 Python52.sln _52ab_try_except.py Finally – pg. 65 Python52.sln _52d_Finally.py Raise – pg. 66 Python52.sln 52e_Raise.py
Post-Assessment Assessment time - 00:32:00	Troubleshooting and Error Handling: Post-Assessment		

Domain 6 - Operations using Modules and Tools [approximately 3.25 hours of videos and projects]			
Lesson	**Lesson Topic and Subtopics**	**Objectives**	**Workbook Projects and Files**
Pre-Assessment Assessment time - 00:12:00	Operations using Modules and Tools: Pre-Assessment		
Lesson 1 Video time - 00:06:33 Workbook time - 01:00:00	Built-in Modules for Operations Built-in Modules IO Sys Os Os.path	6.1 Perform basic operations using built-in modules 6.1a IO 6.1b Os 6.1c Os.path 6.1d Sys (importing modules, opening, reading and writing files, command-line arguments)	IO – pg. 69 Python61.sln _61a_io.py Os – pg. 70 Python61.sln _61b_os.py Os.path – pg. 71 Python61.sln _61c_ospath.py Sys – pg. 72 Python61.sln _61d_sys.py Using a Command Prompt – pg. 73 Python61.sln _61d_sys.py
Lesson 2 Video time - 00:12:23 Workbook time - 00:20:00	Solve Problems with Built-in Modules Math Datetime Class Random	6.2 Solve complex computing problems by using built-in modules 6.2a Math (fabs, ceil, floor, trunc, fmod, frexp, nan, isnan, sqrt, isqrt, pow, pi) 6.2b Datetime (now, strftime, weekday) 6.2c Random (randrange, randint, random, shuffle, choice, sample)	Math– pg. 74 Python62.sln _62a_Math.py Math Problems – pg. 75 Python62.sln _62a_Math.py Datetime – pg. 76 Python62.sln _62b_DateTime.py Random – pg. 77 Python62.sln _62c_Random.py Random Problems – pg. 78 Python62.sln _62c_Random.py
Lesson 3 Video time - 00:02:14 Workbook time - 00:00:00	Domain 6 and Final Recaps Domain 6 Exam Tips Final Exam Tips Conclusion		
Post-Assessment Assessment time - 00:40:00	Operations using Modules and Tools: Post-Assessment		

Python Objectives

Domain 1 Operations using Data Types and Operators	Domain 2 Flow Control with Decisions and Loops	Domain 3 Input and Output Operations
1.1 Evaluate expressions to identify the data types Python assigns to variables 1.1a Str, int, float, and bool data types	2.1 Construct and analyze code segments that use branching statements 2.1a If 2.1b Elif 2.1c Else 2.1d Nested and compound conditionals	3.1 Construct and analyze code segments that perform file input and output operations 3.1a Open 3.1b Close 3.1c Read 3.1d Write 3.1e Check existence 3.1f Delete 3.1g With statement
1.2 Perform data and data type operations 1.2a Type casting 1.2b Constructing data structures 1.2c Indexing and slicing operations	2.2 Construct and analyze code segments that perform iteration 2.2a While 2.2b For 2.2c Break 2.2d Continue 2.2e Pass 2.2f Nested loops and loops that include conditional compounds	3.2 Construct and analyze code segments that perform console input and output operations 3.2a Read input from console 3.2b Print formatted text (string.format() method, f-String method) 3.2c Use of command-line arguments
1.3 Determine the sequence of execution based on operator precedence 1.3a Assignment 1.3b Comparison 1.3c Logical 1.3d Arithmetic 1.3e Identity 1.3f Containment		
1.4 Select operators to achieve the intended results 1.4a Assignment 1.4b Comparison 1.4c Logical 1.4d Arithmetic 1.4e Identity 1.4f Containment		

Python Objectives

Domain 4 Code Documentation and Structure	Domain 5 Troubleshooting and Error Handling	Domain 6 Operations using Modules and Tools
4.1 Document code segmenta 4.1a Use of indentation and white space 4.1b Comments and documentation strings 4.1c Pydoc	5.1 Analyze, detect, and fix code segments that have errors 5.1a Syntax errors 5.1b Logic errors 5.1c Runtime errors	6.1 Perform basic operations by using built-in modules 6.1a IO 6.1b Os 6.1c Os.path 6.1d sys (importing modules, opening, reading and writing files, command-line arguments)
4.2 Construct and analyze code segments that include function definitions 4.2a Call signatures 4.2b Default values 4.2c Return 4.2d Def 4.2e Pass	5.2 Analyze and construct code segments that handle exceptions 5.2a Try 5.2b Except 5.2c Else 5.2d Finally 5.2e Raise	6.2 Solve complex computing problems by using built-in modules 6.2a Math (fabs, ceil, floor, trunc, fmod, frexp, nan, isnan, sqrt, isqrt, pow, pi) 6.2b Datetime (now, strftime, weekday) 6.2c Random (randrange, randint, random, shuffle, choice, sample)
	5.3 Perform unit testing 5.3a Unittest, functions, and methods	