**Python Programming Fundamentals**

This module discusses Python fundamentals and begins with the concepts of conditions and branching. Continue through the module and learn how to implement loops to iterate over sequences, create functions to perform a specific task, perform exception handling to catch errors, and how classes are needed to create objects.

**Learning Objectives**

* Classify conditions and branching by identifying structured scenarios with outputs.
* Work with objects and classes.
* Explain objects and classes by identifying data types and creating a class.
* Use exception handling in Python.
* Explain what functions do.
* Build a function using inputs and outputs.
* Explain how for loops and while loops work.
* Work with condition statements in Python, including operators and branching.
* Create and use loop statements in Python.

# **Conditions and Branching**

## **Conditions and Branching**

**conditions and branching** in programming, specifically using Python. Here’s a simplified explanation and summary:

Simplified Explanation:

1. **Comparison Operations**:
   * You can compare values (like numbers) to see if they are equal, greater, or less than each other.
   * For example, checking if 6 is equal to 7 gives a result of **false**, while checking if 6 is equal to 6 gives **true**.
2. **Branching with If Statements**:
   * An **if statement** lets your program make decisions. If a condition is true, it runs a specific block of code; if false, it skips that code.
   * Example: If someone is 18 or older, they can enter a concert; if not, they cannot.
3. **Else and Elif Statements**:
   * **Else** provides an alternative action if the if condition is false.
   * **Elif** (else if) checks another condition if the first one is false.
4. **Logical Operators**:
   * **Not** reverses the truth value (true becomes false and vice versa).
   * **And** requires both conditions to be true for the result to be true.
   * **Or** requires at least one condition to be true for the result to be true.

Summary:

The video explains how to use comparison operations to evaluate conditions in Python. It introduces **if statements** for decision-making, along with **else** and **elif** for alternative conditions. It also covers logical operators like **not**, **and**, and **or** to combine conditions. This knowledge is essential for controlling the flow of a program based on different inputs.

# **Loops**

Simple Explanation:

1. **Loops**: Loops are used in programming to repeat a task multiple times without having to write the same code over and over.
2. **Range Function**:
   * The **range function** creates a list of numbers.
   * For example, range(3) gives you [0, 1, 2].
   * If you use two numbers, like range(10, 15), it gives you [10, 11, 12, 13, 14].
3. **For Loops**:
   * A **for loop** goes through each item in a list and performs an action.
   * For example, if you have a list of colored squares, you can use a for loop to change each square to white.
4. **While Loops**:
   * A **while loop** continues to run as long as a certain condition is true.
   * For example, if you want to copy orange squares from one list to another, you keep copying until you find a square that is not orange.

Summary:

* The video explains **loops** in Python, focusing on **for loops** and **while loops**.
* It introduces the **range function** to generate sequences of numbers.
* **For loops** are used to iterate through lists and perform actions on each item.
* **While loops** run as long as a condition is met, allowing for more flexible iterations.

# **Functions**

Simple Explanation:

* **Functions**: Think of a function as a small machine that takes some input, does something with it, and gives you an output. You can use it whenever you need to perform that task without rewriting the code.
* **Built-in Functions**: Python has many ready-made functions, like len() to find out how many items are in a list, or sum() to add up numbers in a list.
* **Creating Your Own Functions**: You can also create your own functions. To do this, you start with the keyword def, give your function a name, and then write the code that it will run. For example, a function that adds 1 to a number.
* **Calling Functions**: When you want to use a function, you "call" it by its name and give it the input it needs. The function runs and gives you back a result.
* **Parameters and Return Values**: Functions can take multiple inputs (called parameters) and can return a value. If a function doesn’t return anything, it gives back a special value called None.
* **Scope**: Variables can be global (accessible anywhere) or local (only accessible within the function). This means you can have a variable with the same name in different places without conflict.

Summary:

* The video explains how to use and create functions in Python.
* Functions are reusable pieces of code that take inputs, perform tasks, and return outputs.
* It covers built-in functions like len() and sum(), and how to define your own functions using def.
* It discusses the concept of parameters, return values, and the difference between global and local variables.

# **Exception Handling**

**exception handling** in programming, specifically using Python. Here’s a simplified explanation and summary:

Simple Explanation:

* **What is Exception Handling?**: It's a way for a program to deal with errors that happen when it runs. Instead of crashing, the program can handle the error and continue working.
* **Try...Except Statement**:
  + The program tries to run some code in a "try" block.
  + If an error happens, it jumps to the "except" block to handle the error.
* **Example**: If you try to read a file that doesn't exist, the program will show an error message instead of stopping completely.
* **Else Statement**: If the code in the "try" block runs without errors, the program can do something else, like print a success message.
* **Finally Statement**: This part runs no matter what happens, like closing a file after you're done with it.

Summary:

* The video explains how to use exception handling in Python to manage errors.
* It covers the **try...except** structure, the importance of specifying errors, and how to use **else** and **finally** statements.
* By using these tools, programmers can create more robust and user-friendly applications that handle errors gracefully.

# **Objects and Classes**

**objects and classes in Python** in simpler terms.

Simple Explanation:

* **Objects and Classes**: In Python, everything is an object. An object is a specific instance of a class, which is like a blueprint. For example, if a class is a blueprint for a house, an object is the actual house built from that blueprint.
* **Data Types**: Python has different types of data like numbers (integers), decimal numbers (floats), text (strings), lists (collections of items), dictionaries (key-value pairs), and true/false values (booleans). Each of these is an object.
* **Methods**: Methods are functions that belong to an object. They allow you to interact with the data in the object. For example, a list has a method called sort that organizes the items in the list.
* **Creating Classes**: You can create your own classes in Python. For example, you can create a class for a circle that has attributes like radius and color. You can also create methods to change the radius or draw the circle.
* **Constructors**: When you create a new object from a class, you use a special function called a constructor. This function initializes the object's attributes.

Summary:

* The video explains how Python uses objects and classes to organize data and functionality.
* It covers different data types in Python, how to create your own classes, and how to use methods to interact with objects.
* It emphasizes that understanding how to create and use classes and objects is essential for programming in Python.

# **Module 3 Summary:**

**Module 3 Summary: Python Programming Fundamentals**

Congratulations! You have completed this module. At this point, you know that:

* Python conditions use “if” statements to execute code based on true/false conditions created by comparisons and Boolean expressions.
* Comparison operations require using comparison operators equal to "=", greater than ">", less than "<".
* An exclamation mark "!" is used to define inequalities of a variable.
* You can compare integers, strings, and floats.
* Python branching directs program flow by using conditional statements (for example, if, else, elif) to execute different code blocks based on conditions or tests.
* You can use the "if" statement with conditions to define actions if true.
* To perform actions based on true or false output, you can use the "else" statement with conditions.
* The elif statement allows for additional checks only if the initial condition is false.
* To execute various operations on Boolean values, we use Boolean logic operators.
* Python loops are control structures that automate repetitive tasks and iterate over data structures like lists or dictionaries.
* The range() function generates a sequence of numbers with a specified start, stop, and step value for loops in Python.
* A for loop in Python iterates over a sequence, such as a list, tuple, or string, and executes a block of code for each item in the sequence.
* A while loop in Python executes a block of code as long as a specified condition remains true.
* Python functions are reusable code blocks that perform specific tasks, take input parameters, and often return results, enhancing code modularity and reusability.
* You may or may not have written the codes that are often included in functions.
* Python has a set of built-in functions such as "len" to find the length of a sequence or "sum" to find the total sum of a sequence.
* The "sorted" function creates a new sorted list, while "sort" sorts items in the original list.
* You can also create your own functions in Python.
* To ensure clarity and organization and facilitate understanding and maintenance of the code, developers must document functions using a documentation string enclosed in three quotes.
* The help command will return the documentation defined for a particular function.
* A function can have multiple parameters.
* “No return” statement in the function means that the function will return nothing.
* The "No work" function does not execute any task. You can use the "pass" keyword to meet the requirement of a non-empty body.
* A function will usually perform more than one task.
* In Python, the scope of a variable determines where you can access or modify that variable. Global scope allows access from anywhere, while local scope restricts it to a block or function.
* In Python, a programmer defines a local variable within a specific block or function, which can only be accessed or modified within that block or function.
* In Python, a global variable is a variable defined at the top level of a program that any part of the code can access or modify.
* Exception handling in Python is a mechanism for managing and responding to errors and exceptions that may occur during program execution, preventing them from crashing the program.
* In Python, you use the "try-except" statement to attempt a block of code and specify alternative actions to execute if an error occurs, allowing you to handle exceptions.
* In Python, you use the "try-except-else" statement to attempt a block of code, handle exceptions in the "except" block, and execute code in the "else" block when no exceptions occur.
* Python developers use the "try-except-else-finally" statement to attempt a block of code, catch exceptions in the "except" block, execute code in the "else" block when no exceptions occur, and ensure that the "finally" block always runs, regardless of whether an exception raised or not.
* In Python, objects are instances of classes that encapsulate data and behavior, serving as the foundation for creating and working with various data types and custom data structures.
* To determine the type of an object in Python, you can use the `type()` command.
* Any changes made within the method of the object may result in a change in object type.
* Classes in Python are blueprints for creating objects, defining their attributes and methods, enabling code organization, and object-oriented programming.
* Function "init" is a special method used to initialize data attributes.
* We can create instances of a class in Python.
* Data attributes consist of the data defining the objects.
* Methods are functions that interact and change the data attributes.
* The method has a function that requires the self as well as other parameters.

# **Glossary**

**Glossary: Python Programming Fundamentals**

Welcome! This alphabetized glossary contains many of the terms you'll find within this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are important for you to recognize when working in the industry, participating in user groups, and participating in other certificate programs.

| **Term** | **Definition** |
| --- | --- |
| Analogy | Refers to a concept or comparison outside the scope of the programming language itself, used to explain or relate one concept to another in a more understandable way. |
| Attributes | Attributes in Python refer to the characteristics or properties of an object, and they can be accessed using dot notation. |
| Branching | Branching in Python is a process of altering the flow of a program based on conditions, typically using if, elif, and else statements. |
| Comparison operators | Comparison operators in Python are used to compare values and return Boolean results (True or False), including operators like == (equal),!= (not equal), < (less than), > (greater than), <= (less than or equal to), and >= (greater than or equal to). |
| Conditions | Conditions in Python are used to make decisions in code, executing specific blocks of code based on whether a given expression evaluates to True or False. |
| Enumerate | In Python, "enumerate" is a built-in function that adds a counter to an iterable, allowing you to loop through both the elements and their corresponding indices. |
| Exception handling | Exception handling in Python is a mechanism for gracefully managing and responding to errors or exceptional conditions that may occur during program execution. |
| Explicitly | In Python, the term "explicitly" refers to performing an action or specifying something in a clear, unambiguous, and direct manner. |
| For loops | For loops in Python are used for iterating over a sequence (such as a list, tuple, or string) or other iterable objects, executing a set of statements for each item in the sequence. |
| Global variable | Global variables in Python are variables defined outside of any function or block and can be accessed and modified from any part of the code. |
| Incremented | "Incremented" in Python means to increase the value of a variable by a specified amount, typically done using the += operator or by adding a fixed value. |
| Indent | In Python, "indent" refers to the use of whitespace at the beginning of a line to signify the structure and scope of code blocks, such as loops and functions. |
| Indices | In Python, "indices" refer to the position or location of elements in a sequence, like a string, list, or tuple, starting with 0 for the first element. |
| Iterate | In Python, "iterate" means to repeatedly perform a set of operations or steps on each item in a collection, such as a list, tuple, or dictionary, typically using loops or iterators. |
| Local variables | Local variables in Python are variables defined within a specific function or block of code and are only accessible within that function or block. |
| Logic operators | Logic operators in Python are used to perform logical operations on Boolean values, including operators like and (logical AND), or (logical OR), and not (logical NOT). |
| Loops | Loops in Python are constructs for repeating a block of code, enabling the execution of the same code multiple times. |
| Parameters | Parameters in Python are placeholders in a function definition, used to accept and work with values provided to the function when it is called. |
| Programming Fundamentals | Programming fundamentals in Python involve variables, control structures, functions, data structures, input/output, and error handling for building software. |
| Range function | The range function in Python generates a sequence of numbers that can be used for iterating in a loop and is typically used as range (start, stop, step), where it creates numbers from start to stop-1 with the given step increment. |
| Scope of function | The "scope of a function" in Python refers to the region of code where a variable defined within that function is accessible or visible. |
| Sequences | Sequences in Python are ordered collections of items that can include data types like strings, lists, and tuples, allowing for indexing and iteration. |
| Syntax | In Python, "explicitly" means to state something clearly and directly, leaving no room for ambiguity or implicit interpretation. |
| While loops | While loops in Python are used to repeatedly execute a block of code as long as a specified condition is true. |