Module 6 - Python Fundamentals

Introduction to Python and its Features (simple, high-level, interpreted language).

Python is a simple, high-level, interpreted programming language widely used for web development, data science, automation, and more.
Simple – Easy to read and write with minimal syntax.
High-level – Closer to human language, hides low-level details.
Interpreted – Executes code line by line without needing compilation.

History and evolution of Python.

Python was created by Guido van Rossum in 1991 with a focus on simplicity and readability. In 2000, Python 2.0 was released. Later in 2008, Python 3.0 came as a major upgrade in syntax. Over the years, Python has grown into one of the most popular languages used in areas like AI, data science, web development, and automation.

Advantages of using Python over other programming languages.

Easy to learn and read due to simple syntax
Interpreted language, allows quick testing and debugging
Large standard library with many built-in modules
Strong community support and third-party packages
Widely used in fields like AI, data science, and web development

Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).

Download and install Python from the official website or install Anaconda which comes with Python and useful libraries pre-installed. For coding, you can use IDEs like PyCharm or text editors like VS Code. After installation, set the environment path so Python commands work in the terminal

Writing and executing your first Python program.

Write a simple program like print("Hello, World!")
Save the file with a .py extension, for example hello.py
Run the program.

Understanding Python's PEP 8 guidelines.

PEP 8 is the official style guide for writing Python code. It provides rules for naming, indentation, spacing, and line length to make code clean and readable. Following PEP 8 helps

maintain consistency in projects, improves collaboration, and makes programs easier to understand and debug.

Indentation, comments, and naming conventions in Python.

Indentation in Python is mandatory and defines code blocks instead of using braces.
Comments are written using the # symbol for single-line and triple quotes for multi-line documentation.
Naming conventions follow PEP 8, where variables and functions use lowercase with underscores, constants are written in uppercase, and class names use PascalCase.

Writing readable and maintainable code.

Write clear and descriptive variable, function, and class names
Keep code organized with proper indentation and spacing
Use comments and docstrings to explain complex parts

Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.

Integers are whole numbers without a decimal point. Floats are numbers with a decimal point. Strings are sequences of characters enclosed in quotes. Lists are ordered, mutable collections of items. Tuples are ordered, immutable collections of items. Dictionaries store key-value pairs and are unordered. Sets are unordered collections of unique items.

Python variables and memory allocation.

Variables in Python are used to store data and do not need explicit declaration.
Python automatically allocates memory for a variable when it is created.
The type of data a variable holds can change dynamically. Variable name should follow the identifier rules.

Python operators: arithmetic, comparison, logical, bitwise.

Arithmetic includes: + - * / % ** //
Comparison includes: < > <= >= == !=
Logical include: and or not
Bitwise include: & | ^ ~ << >>

Introduction to conditional statements: if, else, elif.

Conditional statements allow a program to make decisions based on certain conditions:
if statement executes a block of code when a condition is True.
elif statement checks another condition if the previous if was False.
else statement runs a block of code when all previous conditions are False.

Nested if-else conditions.

Nested if-else conditions are if-else statements placed inside another if or else block.
They allow checking multiple conditions in a hierarchical way. The inner if-else executes only when the outer condition is True. This helps handle complex decision-making in programs.

Introduction to for and while loops.

Loops are used to repeat a block of code multiple times.
for loop iterates over a sequence like a list, tuple, string, or range of numbers.
while loop repeats as long as a given condition is True.

How loops work in Python.

In Python, loops repeatedly execute a block of code as long as a condition is met.
A for loop goes through each item in a sequence one by one and runs the code block for each item.

Using loops with collections (lists, tuples, etc.)

Loops can iterate over collections like lists, tuples, dictionaries, and sets. A for loop accesses each element in the collection one by one. For dictionaries, loops can iterate over keys, values, or key-value pairs. This makes it easy to process multiple items without writing repetitive code.

Understanding how generators work in Python.

Generators in Python are special functions that produce values one at a time using yield. They are memory-efficient because they don't store all values at once, making them suitable for large datasets or streams of data.

Difference between yield and return.

The difference between yield and return is that yield pauses the function and remembers its state, so it can produce the next value on next calls, while return exits the function immediately and sends a value once.

Understanding iterators and creating custom iterators.

Iterators are objects that allow traversal over a sequence of elements using iter() and next(). Custom iterators can be created by defining a class in pyhton.

Defining and calling functions in Python.

Functions in Python are defined using the def keyword and executed by calling their name. They help organize code into reusable blocks for better readability and maintainability.

Function arguments (positional, keyword, default).

Function arguments can be positional, where the order matters. keyword, where values are passed using the parameter name. or default, where parameters take a predefined value if no argument is provided.

Scope of variables in Python.

Variable scope in Python determines where a variable can be accessed. Local variables exist only inside the function where they are defined. Global variables are accessible throughout the program. Nonlocal variables allow inner functions to modify variables from their enclosing function.

Built-in methods for strings, lists, etc.

Python provides many built-in methods for strings, lists, and other collections. For example, strings have upper(), lower(), replace(), and split(). Lists have append(), remove(), sort(), and reverse(). These methods make it easier to manipulate data efficiently.

Understanding the role of break, continue, and pass in Python loops

In loops, break is used to exit the loop completely, continue skips the current iteration and moves to the next, and pass does nothing but acts as a placeholder.

Understanding how to access and manipulate strings.

Strings can be accessed using indexing, sliced to extract parts, concatenated with +, repeated with *, and manipulated using methods like upper(), lower(), strip(), and replace().

Basic operations: concatenation, repetition,string methods(upper(), lower(), etc.).

Basic string operations include concatenation with +, repetition with *, and using methods like upper() to convert to uppercase, lower() to convert to lowercase, and strip() to remove extra spaces.

String slicing.

String slicing allows selecting a part of a string using the syntax string[start:end:step], where start is the beginning index, end is the stopping index (exclusive), and step defines the interval between characters.

How functional programming works in Python.

Functional programming in Python treats functions as first-class objects. Functions like map(), filter(), and reduce() allow processing collections without explicit loops.

Using map(), reduce(), and filter() functions for processing data.

The map() function applies a given function to each item of a sequence and returns a new iterable. The filter() function selects items that satisfy a condition. reduce() applies a function cumulatively to the items of a sequence to produce a single value.

Introduction to closures and decorators.

Closures are functions that capture and remember values from their enclosing scope even after that scope has finished execution. Decorators are functions that take another function as input and modify or enhance its behavior without changing its code.