Module 13) Python Fundamentals

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

    Python is a **high-level, interpreted, and general-purpose programming language** created by **Guido van Rossum** and released in **1991**.
    It is known for its **simple syntax** that makes it easy to learn and read, even for beginners. Python emphasizes **code readability** and allows programmers to express concepts in fewer lines of code compared to other languages.
    Features of Python:
    i. Simple and Easy to Learn.
    j. High-Level Language.
    k. Interpreted Language.
    l. Cross platform.
    m. Dynamically Typed.
    n. Open source and free.

    b. History and evolution of Python.

- **Late 1980s** – Guido van Rossum started developing Python as a hobby project during Christmas 1989.
- 1991 – Python 0.9.0 – First ever official version of python was released.
- 1994 – Python 1.0 - Introduced modules, exceptions, functions, and the core standard library.
- 2000 – Python 2.0 - Released with many new features such as **list comprehensions**, **garbage collection**, and **Unicode support.**
- 2008 – Python 3.0 - A major redesign of the language to fix long-standing issues in Python 2.
- 2020 – End of Python 2 Support - Official support for Python 2 ended on **January 1, 2020**, encouraging all developers to move to Python 3.
- Present (Python 3.x Series) - Python continues to evolve with versions like **3.8, 3.9, 3.10, 3.11, and 3.12.**

c. Advantages of using Python over other programming languages.

Advantage of using python:

- Easy to learn and read – Python's syntax is simple and similar to English
- High Productivity - Python allows developers to write fewer lines of code compared to languages like Java or C++, which increases productivity.
- Interpreted Language - Python code is executed line by line, so errors are easier to detect and fix quickly.
- Cross – platform Compatibility - Python programs can run on different operating systems (Windows, macOS, Linux) without needing any changes.
- Extensive Standard Library - Python includes a large collection of built-in modules and packages that help perform tasks like file handling, web development, and data analysis easily.
- Open source and free - Python is completely free to download, use, and distribute, even for commercial purposes.

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

Installing Python Steps**:**

1. Go to the official Python website: https://www.python.org/downloads/
2. Download the latest version of Python suitable for your operating system.
3. Run the installer and **check the box "Add Python to PATH"** before installing.
4. Once installed, open the command prompt (or terminal) and type: python - -version

e. Writing and executing your first Python program.

Write your first python program:

```
print("Hello, World!")
```

Open your terminal and type:

```
py <filename>.py
```

2. Programming style:
    a. Understanding Python's PEP 8 guidelines.

        Key PEP 8 guidelines:
        - Indentation – use 4 spaces per indentation level.
        - Maximum Line Length - Keep each line **under 79 characters** long for better readability.
        - Blank Lines - Use blank lines to separate **functions**, **classes**, and **sections of code** to improve clarity.
        - Imports - Place all import statements at the **top of the file.**
        - Naming Conventions - use lowercase letters with underscores for functions and variables. Use capital words for classes.
        - Whitespace - Use spaces around operators and after commas for better readability.
        - Comments - Write meaningful comments to explain code logic.
        - Function and Variable Naming - Choose descriptive names that clearly describe the purpose.

    b. Indentation, comments, and naming conventions in Python.

        1. Indentation in Python - **Indentation** means adding spaces at the beginning of a line of code. In Python, indentation is **very important** because it defines the **block of code** (such as inside loops, functions, or conditionals). Unlike other languages (like C, C++, or Java) that use curly braces { }, Python uses indentation to separate code blocks.
        2. Comments in Python - Comments are used to explain the code and make it easier to understand.
           They are ignored by the Python interpreter (i.e., they do not affect program execution).
        3. Naming Conventions in Python - Naming conventions are rules for giving names to **variables, functions, classes, and constants** to make the code clean and readable.

    c. Writing readable and maintainable code.

        Writing readable and maintainable code means creating programs that are **easy to understand, modify, and debug** — both for you and for others who

may read your code later. Readable code follows a clear structure, uses meaningful names, and follows consistent formatting rules.

1. Use Meaningful Names for variables and classes.
2. Follow PEP 8 Guidelines.
3. Write Comments and Docstrings.
4. Keep Code Simple and Modular.
5. Handle Errors Properly.
6. Keep Code Consistent.

3. Core Python Concepts:
    a. Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.

    In Python, **data types** define the type of value a variable can store.
    They determine what kind of operations can be performed on that data.
    Python has several **built-in data types**, including numbers, sequences, and collections.

    1. Integers – Used to store whole numbers without decimals.
    2. Floats – Used to store decimal numbers.
    3. Strings – Sequence of characters enclosed in single or double quotes.
    4. Lists – A collection of ordered, changeable elements.
    5. Tuples – Similar to lists, but immutable.
    6. Dictionaries – A collection of key-value pair.
    7. Sets – A collection of unique, unordered elements.

    b. Python variables and memory allocation.

    A **variable** in Python is a **name** that refers to a value stored in the computer's memory.
    It acts as a **container** that holds data which can be changed or used later in the program.
    When you assign a value to a variable, Python:

    - **Creates an object** in memory to store that value.
    - **Assigns a reference** (the variable name) to that memory location.

    c. Python operators: arithmetic, comparison, logical, bitwise.

- Arithmetic Operators: Used to perform **mathematical operations like** Addition, subtraction, multiplication, division, modulo.
- Comparison Operators: Used to **compare two values**. The result is always **True** or **False** (Boolean).
- Logical operator: Used to **combine conditional statements**. They return **True** or **False** based on conditions.
- Bitwise operators: Used to perform operations on **binary numbers (bits).**

4. Conditional Statements:
    a. Introduction to conditionalstatements: if, else, elif.

- The if Statement - The if statement checks a condition. If the condition is **True**, the block of code inside if is executed. Syntax:
  if condition:
    # code to execute if condition is true
- The else Statement - ⍰ The else block runs **if the condition in the if statement is False**. It acts as a fallback option. Syntax:
  if condition:
    # code if true
  else:
    # code if false
- The elif Statement (Else If) - The elif statement allows checking **multiple conditions.** It runs only if the previous if (or elif) conditions are False. Syntax:
  if condition1:
    # code if condition1 is true
  elif condition2:
    # code if condition2 is true
  else:
    # code if none are true

    b. Nested if-else conditions.

A **nested if-else** statement means placing **one if statement inside another**. It is used when you need to check **multiple conditions** one after another — that is, when a decision depends on **another decision.** Syntax:
if condition1:
  if condition2:

```
        # code to execute if both conditions are true
    else:
        # code to execute if condition1 is true but condition2 is false
else:
    # code to execute if condition1 is false
```

5. Looping (For, While):
    a. Introduction to for and while loops.

    Loops are used in Python to **repeat a block of code** multiple times until a certain condition is met.
    They help reduce repetition and make programs shorter and more efficient.
    1. For Loop: The for loop is used to **iterate (loop) through a sequence** such as a list, tuple, string, or range of numbers.
       Syntax:
       ```
       for variable in sequence:
           # code to execute
       ```
    2. While Loop: The while loop keeps executing a block of code **as long as a condition is true**.
       When the condition becomes false, the loop stops.
       Syntax:
       ```
       while condition:
           # code to execute
       ```

    b. How loops work in Python.

    1. For Loop: A for loop in Python is used to **iterate over a sequence** (like a list, string, or range of numbers). In each iteration, the loop picks the **next item** from the sequence and assigns it to a loop variable. The loop body executes once for each item in the sequence.
    2. While Loop: A while loop runs **as long as its condition is True.** The condition is checked before each iteration. If the condition becomes False, the loop stops.

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

    In Python, **collections** like **lists**, **tuples**, **sets**, and **dictionaries** store multiple values in a single variable.

We can use **loops** (mainly the for loop) to easily **iterate** over these collections and perform operations on each element.
Looping through a list:

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(fruit)
```

Looping through a tuple:

```
colors = ("red", "green", "blue")

for color in colors:
    print(color)
```

6. Generators and iterators:
    a. Understanding how generators work in Python.

    A **generator** in Python is a **special type of function** that allows you to **iterate over data without storing it all in memory**.
    Instead of returning all the values at once (like lists), a generator **yields** one value at a time — only when needed.
    A generator is created using a **function with the yield keyword** instead of return.

    b. Difference between yield and return.

        1. Return statement: The return statement **ends** a function and **sends a single value** back to the caller. Once a function executes return, it **stops completely** — you cannot resume it.
        2. Yield statement: The yield statement **turns a function into a generator.** Instead of returning all values at once, it pauses the function and returns one value at a time. The function can **resume** from where it left off when called again.