Python – Short Term Assignment

Module 13

Introduction to Python

- Python is one of the world's most popular programming languages because of its simplicity and power. It is an **interpreted, high-level, and general-purpose** language designed by Guido van Rossum.

Features of Python

- **Simple and Easy to Learn:** Python has a clean syntax that reads much like English, making it accessible for beginners.
- **High-Level Language:** You don't have to manage complex tasks like memory management; Python handles that in the background.
- **Interpreted:** Code is executed line-by-line, which makes debugging much faster.
- **Extensive Libraries:** It comes with a "batteries included" philosophy, offering huge libraries for data science, web development, and AI.

History and Evolution

- **Origin:** Created by **Guido van Rossum** at CWI in the Netherlands; first released in **1991**.
- **Named After:** Not the snake, but the British comedy troupe *Monty Python's Flying Circus*.
- **Major Versions:** * **Python 1.0 (1994):** Introduced functional programming tools.
    - **Python 2.0 (2000):** Introduced list comprehensions and garbage collection.
    - **Python 3.0 (2008):** A major revision that fixed consistency issues (not backward compatible with 2.x).

Advantages Over Other Languages

- Python is one of the most popular programming languages because it is simple and easy to learn.
- Its syntax is very clear and readable, which makes it suitable for beginners.
- Python is a high-level language, so it automatically manages memory. It is an interpreted language, which helps in easy debugging and testing of programs.
- Python provides a large standard library that supports many programming tasks. It is platform independent, so the same program can run on different operating systems.
- Python supports object-oriented, procedural, and functional programming styles. Less code is required in Python compared to other languages like C or Java. This makes program development faster and more efficient. Python is widely used in web development, data science, and artificial intelligence.

Setting Up the Environment

To start coding, you need the Python interpreter and an Integrated Development Environment (IDE):

1. **Download Python:** Visit python.org to install the latest version.
2. **Choose an IDE/Editor:**
    - **VS Code:** Lightweight and versatile (highly recommended).
    - **PyCharm:** A powerful IDE specifically for professional Python development.
    - **Anaconda:** Best for Data Science, as it comes pre-packaged with libraries like NumPy and pandas.

Your First Python Program

Python's simplicity is best seen in its "Hello World." Unlike other languages that require multiple lines of setup, Python requires only one.

Step 1: Write the code Open your editor and type:

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

Step 2: Execute the program

- **In a terminal:** Type python hello.py and press Enter.
- **In an IDE:** Simply click the **"Run"** button (usually a green play icon).

Output:

```
Hello, World!
```

Writing and executing your first Python program.

- **Install Python:** Download and install Python from the official website www.python.org.
- **Open Editor:** Open a Python editor such as IDLE, VS Code, or Notepad.
- **Write Code:** Type a simple Python program, for example:
  ```
  print("Hello, World!")
  ```
- **Save File:** Save the file with a .py extension, for example first.py.
- **Open Command Prompt:** Press Windows + R, type cmd, and press Enter.
- **Go to File Location:** Use the cd command to go to the folder where the file is saved.
- **Run Program:** Type python first.py and press Enter.
- **View Output:** The output will be displayed on the screen.

2. Programming Style

Understanding Python's PEP 8 Guidelines

- PEP 8 stands for *Python Enhancement Proposal 8*. It is the official style guide for writing Python code. It provides rules and recommendations on how to format Python programs so that the code looks clean, consistent, and easy to read. PEP 8 covers things like indentation, line length, naming conventions, spacing, and comments. Following PEP 8 helps programmers understand each other's code and makes programs easier to maintain.

Indentation, Comments, and Naming Conventions in Python

- **Indentation:** In Python, indentation is used to define blocks of code. All statements inside loops, functions, and conditionals must be properly indented. Usually, four spaces are used for one level of indentation.

- **Comments:** Comments are used to explain the code. They start with the `#` symbol and are ignored by the Python interpreter. Comments make the program easier to understand.
- **Naming Conventions:** Naming conventions are rules for naming variables, functions, and classes. Variable and function names should be written in lowercase with underscores (for example, `total_marks`), and class names should start with a capital letter (for example, `Student`).

Writing Readable and Maintainable Code

- Writing readable and maintainable code means writing programs that are easy to read, understand, and modify in the future. This can be achieved by using proper indentation, meaningful variable and function names, writing comments where necessary, and following PEP 8 guidelines. Readable code helps reduce errors and makes it easier for other programmers to work on the same program.

3.Core Python Concepts

Understanding Data Types in Python

Data types define the type of data that a variable can store in Python. Python supports several built-in data types such as integers, floats, strings, lists, tuples, dictionaries, and sets. These data types help in organizing and managing data efficiently in a program.

- **Integer (int):** Stores whole numbers, for example 10, -5, 0.
- **Float (float):** Stores decimal numbers, for example 3.14, -2.5.
- **String (str):** Stores a sequence of characters enclosed in quotes, for example `"Hello"`.
- **List:** Stores an ordered and changeable collection of items, for example `[1, 2, 3]`.
- **Tuple:** Stores an ordered but unchangeable collection of items, for example `(1, 2, 3)`.
- **Dictionary (dict):** Stores data in key-value pairs, for example `{"name": "Ram", "age": 20}`.
- **Set:** Stores an unordered collection of unique elements, for example `{1, 2, 3}`.

Python Variables and Memory Allocation

- A variable in Python is a name that refers to a memory location where data is stored. When a value is assigned to a variable, Python automatically allocates memory for that value. Python is a dynamically typed language, which means the type of variable is decided at runtime. The same variable can store different types of values at different times. Python uses automatic memory management, so the programmer does not need to manually allocate or free memory.

Python Operators

Operators are special symbols used to perform operations on variables and values. Python supports different types of operators such as arithmetic, comparison, logical, and bitwise operators.

- **Arithmetic Operators:** Used to perform mathematical operations like addition, subtraction, multiplication, division, modulus, and exponentiation. Examples: `+`, `-`, `*`, `/`, `%`, `**`.
- **Comparison Operators:** Used to compare two values and return True or False. Examples: `==`, `!=`, `>`, `<`, `>=`, `<=`.
- **Logical Operators:** Used to combine conditional statements. Examples: `and`, `or`, `not`.
- **Bitwise Operators:** Used to perform operations on binary numbers. Examples: `&`, `|`, `^`, `~`, `<<`, `>>`.

4. Conditional Statements

Introduction to conditional statements: if, else, elif.

Conditional statements are used to **make decisions** in a program.
They allow the program to **execute different blocks of code** based on whether a condition is **True or False**.

if Statement

- The `if` statement checks a condition.
  If the condition is **True**, the block of code inside `if` will run.
- Syntax:

```
if condition:
statement
```

if –else Statement

- The `if-else` statement is used when we want to execute **one block if the condition is True** and **another block if it is False**.
- Syntax:

```
if condition:
    statement
else:
    statement
```

elif Statement

- `elif` means **else if**.
  It is used to check **multiple conditions** in a program.
- Syntax:

```
if condition1:
    statement
elif condition2:
    statement
else:
    statement
```

5. Looping (For, While)

Introduction to for and while loops.

- A **for loop** is used when we **know how many times** we want to repeat the code.

- A **while loop** is used when we **do not know the number of times** in advance.
  It runs **as long as the condition is true**.

How loops work in Python.

- Python checks the **condition**
- If condition is **True** → code inside loop runs
- After running, Python goes back and checks the condition again
- This continues until condition becomes **False**
- When condition is False → loop stops

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

- **List:** for item in list_name: → Accesses each element one by one.
- **Tuple:** for item in tuple_name: → Iterates through all tuple values.
- **Set:** for item in set_name: → Loops through set elements in random order.
- **Dictionary:** for key in dict_name: → Accesses keys and values using dict_name[key].
- **While Loop:** while i < len(collection): → Uses index to access each element.

6. Generators and Iterators

Understanding how generators work in Python.

- A generator in Python is a special function that uses the **yield** keyword to return values one at a time. When called, it returns a generator object and pauses execution after each yield. The function resumes from where it stopped when the next value is requested. Generators are memory efficient and useful for handling large data.

Difference between yield and return.

Yield

- The yield keyword is used in generator functions to produce values one at a time. When a function executes a yield statement, it returns a value and pauses its execution. The function remembers its current state and resumes execution from the same point when the next value is requested. Yield is memory efficient and is mainly used when working with large data sets or sequences.

Return

- The return keyword is used in normal functions to return a value. When a return statement is executed, the function immediately stops its execution and exits. The function does not remember its state after returning a value. Return is commonly used when a function needs to produce a single result.

Understanding Iterators and Creating Custom Iterators

- An **iterator** in Python is an object that is used to iterate over a sequence of values, such as lists, tuples, or strings, one element at a time. An iterator follows the **iterator protocol**, which means it must implement two special methods: **__iter__()** and **__next__()**.
- The __iter__() method returns the iterator object itself, and the __next__() method returns the next value from the sequence. When there are no more values to return, the __next__() method raises a **StopIteration** exception to stop the iteration.
- A **custom iterator** is created by defining a class that implements the iterator protocol. This allows the programmer to control how the values are generated and iterated. Custom iterators are useful when working with user-defined objects or when a specific iteration logic is required.

7. Functions and Methods

Defining and calling functions in Python.

A **function** in Python is a block of reusable code that performs a specific task. Functions help in reducing code repetition and make programs easier to understand and maintain.

Defining a Function

- A function is defined using the `def` keyword followed by the function name and parentheses. Parameters can be passed inside the parentheses. The body of the function contains the code to be executed and is written with proper indentation.

Calling a Function

- A function is called by using its name followed by parentheses. If the function has parameters, appropriate arguments are passed while calling the function. When a function is called, the control of the program moves to the function body, executes the statements, and returns the result if a return statement is used.

Function Arguments in Python (Positional, Keyword, Default)

Function arguments are the values that are passed to a function when it is called. Python supports different types of function arguments to provide flexibility while calling functions.

- ### Positional Arguments
  Positional arguments are passed to a function in the **same order** as defined in the function definition. The position of each argument matters while calling the function.

- ### Keyword Arguments

  Keyword arguments are passed using the **parameter name** followed by its value. The order of arguments does not matter when using keyword arguments.

- **Default Arguments**
  Default arguments are parameters that are assigned a **default value** in the function definition. If a value is not provided during the function call, the default value is used automatically.

Scope of Variables in Python

The **scope of a variable** in Python refers to the area of the program where the variable can be accessed. Python determines the scope of variables based on where they are declared.

- **Local Scope**
  A variable declared inside a function has a local scope. It can be accessed only within that function.
- **Global Scope**
  A variable declared outside all functions has a global scope. It can be accessed anywhere in the program.
- **Enclosed (Non-local) Scope**
  A variable declared in an outer function but used inside an inner function has an enclosed scope.
- **Built-in Scope**
  Built-in scope includes predefined names provided by Python, such as print(), len(), and range().

Built-in Methods for Strings, Lists, etc.

Python provides many **built-in methods** for commonly used data types such as strings, lists, tuples, and dictionaries. These methods help in performing operations easily and efficiently.

- **String Methods**
  String methods are used to manipulate text data. Common string methods include upper(), lower(), capitalize(), strip(), split(), and replace(). These methods are used to change the case of characters, remove spaces, split strings, and replace text.
- **List Methods**
  List methods are used to add, remove, and modify list elements. Common list methods include append(), insert(), remove(), pop(), sort(), and reverse(). These methods help in managing list data dynamically.
- **Tuple Methods**
  Tuples have limited built-in methods because they are immutable. Common tuple methods include count() and index(), which are used to count elements and find their position.
- **Dictionary Methods**
  Dictionary methods are used to work with key-value pairs. Common dictionary methods include keys(), values(), items(), get(), and update().

8. Control Statements (Break, Continue, Pass)

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

- **break** – Stops the loop immediately and comes out of it when a condition is met.
- **continue** – Skips the current loop iteration and moves to the next one.
- **pass** – Does nothing; used as a placeholder where a statement is required but no action is needed.

9. String Manipulation

Understanding how to access and manipulate strings.

- Strings can be accessed using index positions and manipulated using built-in string methods.

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

- Basic operations on strings include concatenation, which joins two or more strings using the + operator, repetition, which repeats a string multiple times using the * operator, and string methods such as upper(), lower(), capitalize(), title(), replace(), and split() that are used to modify, format, and manipulate the content of a string in different ways.

String slicing

- String slicing is used to extract a specific part of a string by specifying start, end, and step values inside square brackets [start:end:step].

10. Advanced Python (map(), reduce(), filter(), Closures and Decorators)

How functional programming works in Python.

Functional programming in Python is a programming approach where programs are written using functions and expressions rather than changing data or using loops. In Python, functions are treated as first-class objects, which means they can be passed as arguments, returned from other functions, and stored in variables. This style helps in writing clean, modular, and reusable code.

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

The map() function applies a function to each element of an iterable and returns the result. The filter() function selects elements from an iterable that satisfy a given condition. The reduce() function applies a function repeatedly to reduce the iterable to a single value and is

available in the functools module. These functions are commonly used for data processing in functional programming.

Introduction to closures and decorators.

A closure is a function that remembers the variables from its outer function even after the outer function has finished execution. A decorator is a function that modifies the behavior of another function without changing its code. Closures and decorators are used to enhance functionality and maintain clean code structure.