1. Displaying Output on the Screen

What is the `print()` Function in Python?

The `print()` function is used to **show information on the console**. It can display:

- Text (strings)
- Numbers
- Variables
- Results of expressions

By default:

- Items are separated by a **space**.
- The output ends with a **newline character** (moves to the next line).

Formatting Output with f-strings and `format()`

- **f-strings (formatted strings):**
  Added in Python 3.6, f-strings make it easy to include variables and expressions inside a string. You just add an `f` before the quotes and write variables inside `{}`.
- **`format()` method:**
  This is a string method that inserts values into `{}` placeholders. It supports alignment, positioning, and advanced formatting.

2. Accepting Input from the User

Using the `input()` Function

Python uses the `input()` function to **receive data from the user through the keyboard**. Whatever the user enters is **always returned as a string**, even if it looks like a number.

```
name = input("Enter your name: ")
print("Hello", name)
```

Converting Input into Other Data Types

Since `input()` gives data in **string format**, you must convert it to the required type if you want to perform calculations or specific operations:

- Convert to integer → `int()`
- Convert to floating-point number → `float()`
- Convert to boolean → `bool()`

```
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
print("Sum:", num1 + num2)
```

3. Opening and Closing Files in Python

Opening Files in Different Modes

In Python, files can be opened in different **modes** that define how the file will be used:

- `'r'` **(Read mode):**
  Opens an existing file for **reading only**. If the file does not exist, it throws an error.
- `'w'` **(Write mode):**
  Creates a **new file** or overwrites an existing file for **writing**.
- `'a'` **(Append mode):**
  Opens a file for **writing**, but does not delete old content. New data is added at the end of the file.
- `'r+'` **(Read & Write mode):**
  Opens an existing file for both **reading and writing**. The file must exist.
- `'w+'` **(Write & Read mode):**
  Creates a new file (or overwrites existing content) for both reading and writing.

Using `open()` to Create or Access Files

To work with a file, Python provides the `open()` **function**, which returns a **file object**. You need to specify:

1. **File name**
2. **Mode** (`'r'`, `'w'`, etc.)

Example:

```
file = open("data.txt", "w")  # Open file in write mode
file.write("Hello World")
file.close()  # Close the file
```

Closing the File

Once you finish working with a file, always **close it using** `close()`. This ensures that all changes are saved and resources are released.

```
file.close()
```

4. Reading and Writing Files

Reading Methods

- `read()` → Loads the **whole file content** into one string.
- `readline()` → Fetches **only one line** at a time from the file.
- `readlines()` → Gets **all lines** from the file and returns them in a list format.

Writing Methods

- `write()` → Adds a **single text string** to the file.
- `writelines()` → Adds **multiple strings from a list** into the file (no automatic new lines).

5. Exception Handling in Python

What are Exceptions?

Exceptions are **errors that happen while the program is running**. If they are not managed, the program will **stop immediately**.

Handling Exceptions with `try`, `except`, and `finally`

- `try` **block** → Holds the code that **might cause an error**.
- `except` **block** → Runs **only when an error occurs** in the try block.
- `finally` **block** → Executes **every time**, whether there is an error or not (often used for cleanup tasks).

Multiple and Custom Exceptions

- **Multiple Exceptions:** You can have **different except blocks** for **different error types**.
- **Custom Exceptions:** Create your own exception classes by **inheriting from** `Exception` to handle special cases in your program.

6. Classes and Objects in Python (OOP Basics)

Core Concepts

- **Class:** A **template or blueprint** used to create objects. It defines **attributes** (data) and **methods** (functions).
- **Object:** A **real instance** of a class created using that blueprint.
- **Attributes:** Variables that **store data** for the object or class.
- **Methods:** Functions **inside a class** that work on the object's data.

Local vs Global Variables

- **Local Variable:** Created **inside a function or method**, and can only be used there.
- **Global Variable:** Declared **outside all functions**, accessible in the entire program (unless overridden locally).

7. Inheritance in Python

Inheritance allows a class to **use the features of another class**. The class that inherits is called the **child (subclass)**, and the class being inherited from is the **parent (superclass)**.

Types of Inheritance

- **Single Inheritance:** One child class inherits from **one parent class**.
- **Multilevel Inheritance:** A class inherits from a parent, which **itself inherits from another class** (grandparent → parent → child).
- **Multiple Inheritance:** One class inherits from **two or more parent classes**.
- **Hierarchical Inheritance: Several child classes** inherit from **the same parent class**.
- **Hybrid Inheritance:** A **combination of two or more** inheritance types.

Using `super()`

The `super()` **function** is used inside a child class to **call methods or access attributes from the parent class**.

- Helps avoid **hardcoding the parent class name**.
- Supports **method resolution order (MRO)** in multiple inheritance.

8. Method Overloading and Overriding in Python

Method Overloading

- Python does **not support true method overloading** like Java or C++.
- If multiple methods have the **same name**, the **last defined method overrides the previous ones**.
- To mimic overloading, use:
    - **Default Parameters** → Assign default values to parameters.
    - **Variable-Length Arguments** → Use `*args` (for positional arguments) and `**kwargs` (for keyword arguments) to handle different numbers or types of inputs.

Method Overriding

- Occurs when a **child class defines a method with the same name and parameters as in the parent class**.
- The child class version **replaces (overrides)** the parent class method when called from the child object.
- This is useful for **customizing or extending parent class behavior**.

9. SQLite3 and PyMySQL (Database Connectivity in Python)

Introduction

- **SQLite3** → Built-in Python module for **lightweight, file-based databases**. No separate server required.
- **PyMySQL** → External library for connecting to **MySQL databases**. Requires MySQL server and installation via:

  pip install PyMySQL

Steps to Connect and Execute SQL Queries

1. **Import the Connector** → `import sqlite3` or `import pymysql`
2. **Establish a Connection** → Connect to the database file (SQLite) or MySQL server.
3. **Create a Cursor Object** → Used to execute SQL queries.
4. **Execute SQL Commands** → e.g., `CREATE`, `INSERT`, `UPDATE`, `DELETE`, `SELECT`.
5. **Commit Changes** → Use `.commit()` for write operations.
6. **Close the Connection** → Free resources after operations.

10. Search and Match Functions in Python (re Module)

Introduction

The `re` module in Python is used for **regular expression (regex) pattern matching**. Two important functions are:

Functions

- **`re.search(pattern, string)`**
  - Scans the **entire string** to find the **first occurrence** of the pattern.
  - Returns a **match object** if found; otherwise, `None`.
- **`re.match(pattern, string)`**
  - Checks for a match **only at the beginning** of the string.
  - Returns a **match object** if the pattern is at the start; otherwise, `None`.

Key Difference

- **`search()`** → Looks **anywhere** in the string.

- `match()` → Looks **only at the beginning** of the string.