1. Printing on Screen

Introduction to the `print()` Function in Python

The `print()` function in Python is used to display output on the screen. It can output text, numbers, variables, or results of expressions. By default, it prints each item separated by a space and ends with a newline.

Formatting Outputs using f-strings and `format()`

- **f-strings**: Introduced in Python 3.6, they allow you to embed variables and expressions directly inside string literals by prefixing the string with `f` or `F`.

- **`format()` method**: A string method that replaces placeholders (`{}`) inside a string with specified values. It allows positioning, alignment, and formatting of output.

2. Reading Data from Keyboard

Using the `input()` Function to Read User Input from the Keyboard

The `input()` function in Python is used to take input from the user via the keyboard. It always returns the entered value as a string, even if the user types numbers.

Converting User Input into Different Data Types (e.g., `int`, `float`, etc.)

Since `input()` returns a string, you can convert it to other data types using type conversion functions like `int()` for integers, `float()` for floating-point numbers, or `bool()` for boolean values. This allows mathematical operations or type-specific processing on the entered data.

3. Opening and Closing Files

Opening Files in Different Modes (`'r'`, `'w'`, `'a'`, `'r+'`, `'w+'`)

In Python, file modes specify how a file is opened:

- `'r'` → Read mode (default). Opens file for reading; error if file doesn't exist.
- `'w'` → Write mode. Creates a new file or overwrites an existing file.
- `'a'` → Append mode. Opens file for writing without erasing existing content; adds new data at the end.
- `'r+'` → Read and write mode. File must exist. Allows both reading and writing.
- `'w+'` → Write and read mode. Creates a new file or overwrites existing content.

Using the `open()` Function to Create and Access Files

The `open()` function is used to open a file and returns a file object. You pass the file name and mode as arguments, for example:

```
file = open("data.txt", "w")
```

Closing Files using `close()`

After performing file operations, the `close()` method is used to free system resources and ensure all changes are saved:

```
file.close()
```

4. Reading and Writing Files

Reading from a File using `read()`, `readline()`, `readlines()`

- **`read()`** → Reads the entire file content as a single string.
- **`readline()`** → Reads one line from the file at a time.
- **`readlines()`** → Reads all lines from the file and returns them as a list of strings.

Writing to a File using `write()` and `writelines()`

- **`write()`** → Writes a single string to the file.
- **`writelines()`** → Writes a list of strings to the file (no newlines are added automatically).

5. Exception Handling

Introduction to Exceptions and How to Handle Them using `try`, `except`, and `finally`

Exceptions are runtime errors that occur during program execution, which can stop the program if not handled.

- **`try` block** → Contains the code that might cause an exception.
- **`except` block** → Contains the code that runs if an exception occurs.
- **`finally` block** → Contains code that always runs, whether an exception occurs or not (commonly used for cleanup).

Understanding Multiple Exceptions and Custom Exceptions

- **Multiple exceptions** → You can handle different types of errors separately by using multiple `except` blocks, each for a specific exception type.
- **Custom exceptions** → You can define your own exception classes (by inheriting from `Exception`) to handle application-specific error conditions.

6. Class and Object (OOP Concepts)

Understanding the Concepts of Classes, Objects, Attributes, and Methods in Python

- **Class** → A blueprint for creating objects. It defines the structure and behavior (attributes and methods).
- **Object** → An instance of a class, created using the class blueprint.
- **Attributes** → Variables that belong to an object or class, used to store data.
- **Methods** → Functions defined inside a class that operate on the object's data.

Difference between Local and Global Variables

- **Local Variable** → Declared inside a function or method and accessible only within that scope.
- **Global Variable** → Declared outside all functions and accessible throughout the program, unless shadowed by a local variable with the same name.

7. Inheritance

Single, Multilevel, Multiple, Hierarchical, and Hybrid Inheritance in Python

- **Single Inheritance** → A child class inherits from a single parent class.
- **Multilevel Inheritance** → A child class inherits from a parent class, and that parent class inherits from another parent class.
- **Multiple Inheritance** → A child class inherits from more than one parent class.
- **Hierarchical Inheritance** → Multiple child classes inherit from the same parent class.
- **Hybrid Inheritance** → A combination of two or more types of inheritance.

Using the `super()` Function to Access Properties of the Parent Class

The `super()` function is used in a child class to call methods or access properties of its parent class. It helps avoid directly naming the parent class and supports multiple inheritance resolution.

8. Method Overloading and Overriding

Method Overloading
In Python, *true* method overloading (like in Java or C++) is not directly supported. If multiple methods with the same name are defined, the last one overrides the previous ones. However, you can achieve similar behavior by using **default parameters** or **variable-length arguments** (`*args`, `**kwargs`) to allow different numbers or types of arguments in a single method.

Method Overriding
Method overriding occurs when a child class defines a method with the same name and parameters as a method in its parent class. The child class's method replaces (overrides) the parent's version when called from a child object, allowing customized behavior.

9. SQLite3 and PyMySQL (Database Connectors)

Introduction to SQLite3 and PyMySQL for Database Connectivity

- **SQLite3** → A built-in Python module for working with SQLite databases (lightweight, file-based, requires no separate server).
- **PyMySQL** → A Python library for connecting to MySQL databases (requires a running MySQL server). It must be installed separately using `pip install PyMySQL`.

Creating and Executing SQL Queries from Python Using These Connectors

1. **Import the connector** (`sqlite3` or `pymysql`).
2. **Establish a connection** to the database.
3. **Create a cursor object** to execute SQL queries.
4. **Execute SQL queries** such as `CREATE`, `INSERT`, `UPDATE`, `DELETE`, `SELECT`.
5. **Commit changes** (for write operations) using `commit()`.
6. **Close the connection** after operations are done.

10. Search and Match Functions

Using `re.search()` and `re.match()` Functions in Python's `re` Module for Pattern Matching

- **`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 found at the start, otherwise `None`.

Difference Between `search` and `match`

- `search()` → Looks anywhere in the string for the pattern.
- `match()` → Looks only at the beginning of the string.