Module 8) Advance Python Programming

1. Printing on Screen

Theory:

1. **Introduction to the print() function in Python.**

Ans: The print() function in Python is used to display output on the screen.
It is one of the most commonly used built-in functions and helps show text, numbers, variables, or results of expressions.

Syntax:

print(object(s), sep=' ', end='\n')

Parameters:

- object(s) → The value(s) to be printed. Multiple objects can be separated by commas.
- sep → Defines the separator between multiple objects (default is a single space).
- end → Specifies what to print at the end (default is a newline \n).

2. **Formatting outputs using f-strings and format().**

Ans: Python provides several ways to format text output, allowing variables or expressions to be inserted inside strings neatly.

1. f-Strings (Formatted String Literals)

Introduced in Python 3.6, f-strings make formatting easier and more readable.

Syntax:

print(f"Text {variable_name}")

Advantages:

- Easier to read and write.
- Allows direct expression evaluation inside {}.

2. Using the format() Method

Before f-strings, Python used the str.format() method for string formatting.

Syntax:

```
print("Text {} more text {}".format(value1, value2))
```

Method	Introduced In	Syntax Example	Features
print()	Python 1.x	print("Hello")	Displays output
f-String	Python 3.6+	f"Hello {name}"	Simple and fast
format()	Python 2.6+	"Hello {}".format(name)	Flexible and backward-compatible

2. Reading Data from Keyboard

1. **Using the input() function to read user input from the keyboard.**

   Ans: Using the input() Function to Read User Input from the Keyboard In Python, the input() function is used to take input from the user during program execution. It reads data as a string by default, even if the user enters a number.

   Syntax:

   ```
   variable = input("Enter something: ")
   ```

   Example:

   ```
   name = input("Enter your name: ")

   print("Hello,", name)
   ```

   Output:

   ```
   Enter your name: Aayushi

   Hello, Aayushi
   ```

2. **Converting user input into different data types (e.g., int, float, etc.).**

   Ans: Since input() always returns a string, you need to convert it into the appropriate type when working with numbers.

   Example:

   ```
   # Taking integer input

      age = int(input("Enter your age: "))

   # Taking float input

      marks = float(input("Enter your marks: "))


   # Displaying results

      print(f"You are {age} years old and scored {marks} marks.")
   ```

   Output:

   ```
   Enter your age: 21

   Enter your marks: 89.5

   You are 21 years old and scored 89.5 marks.
   ```

3. Opening and Closing Files

Theory:

1. **Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').**

   Ans: Opening Files in Different Modes

   Python provides several modes for opening files using the open() function. Each mode specifies how the file will be accessed — for reading, writing, or appending.

Mode	Description
'r'	Read mode – opens the file for reading (default). Error if the file doesn't exist.
'w'	Write mode – creates a new file or overwrites an existing file.
'a'	Append mode – adds data to the end of the file without erasing existing content.
'r+'	Read and Write mode – opens file for both reading and writing.
'w+'	Write and Read mode – overwrites the file and allows reading.

2. **Using the open() function to create and access files.**

   Ans: Using the open() Function to Create and Access Files

   Syntax:

```
file_object = open("filename", "mode")
```

   Example:

```
# Writing to a file
file = open("example.txt", "w")
file.write("Hello, Python file handling!")
file.close()
```

   Explanation:

   - "example.txt" → Name of the file.
   - "w" → Write mode.
   - write() → Writes text to the file.

3. **Closing files using close().**

Ans: It is important to close a file after completing operations to free up system resources.

Example:

```
file = open("example.txt", "r")

content = file.read()

print(content)

file.close()
```

Output:

```
Hello, Python file handling!
```

Operation	**Function Used**	**Description**
Read input	input()	Reads data from keyboard as string
Type conversion	int(), float(), str()	Converts input to required type
Open file	open(filename, mode)	Opens file in specified mode
Write to file	write()	Writes data to a file
Read file	read(), readline()	Reads content from file
Close file	close()	Closes file manually
Auto close	with open()	Automatically closes after use

4. Reading and Writing Files

Theory:

1. **Reading from a file using read(), readline(), readlines().**

Ans: **read()**

- Reads the entire content of the file as a single string.
- You can also specify the number of characters to read.

Example:

```
file = open("example.txt", "r")

content = file.read()

print(content)

file.close()
```

readline()

- Reads one line at a time from the file.
- Useful for processing large files line by line.

Example:

```
file = open("example.txt", "r")

line1 = file.readline()

line2 = file.readline()

print(line1)

print(line2)

file.close()
```

readlines()

- Reads all lines of a file and returns them as a list of strings.

Example:

```
file = open("example.txt", "r")

lines = file.readlines()

print(lines)

file.close()
```

2. Writing to a file using write() and writelines().

Ans: Files must be opened in write ('w'), append ('a'), or write+ ('w+') mode to write data.

◆ **write()**

- Writes a single string to a file.

Example:

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

file.write("Hello, Python!\n")

file.close()
```

◆ **writelines()**

- Writes a list of strings to a file.

Example:

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

lines = ["Python\n", "is\n", "awesome!\n"]

file.writelines(lines)

file.close()
```

5. Exception Handling

Theory:

1. Introduction to exceptions and how to handle them using try, except, and finally.

Ans: Exceptions are errors that occur during program execution and disrupt the normal flow.

- Why Handle Exceptions?

To prevent the program from crashing and handle errors gracefully.

Exception	**Description**
ZeroDivisionError	Division by zero
ValueError	Invalid value for a function
FileNotFoundError	File not found
TypeError	Invalid data type operation

2. Understanding multiple exceptions and custom exceptions.

Ans: Multiple Exceptions

You can handle different errors separately.

Example:

```
try:
   a = int(input("Enter a number: "))
   b = int(input("Enter another number: "))
   print(a / b)
except ZeroDivisionError:
   print("Cannot divide by zero.")
except ValueError:
   print("Please enter valid integers.")
```

- Custom Exceptions

You can create your own exception using class definitions.

Example:

```
class InvalidAgeError(Exception):
  pass
age = int(input("Enter your age: "))
if age < 18:
  raise InvalidAgeError("Age must be 18 or above.")
else:
  print("You are eligible!")
```

6. Class and Object (OOP Concepts)

Theory:

1. **Understanding the concepts of classes, objects, attributes, and methods in Python.**

Ans: Python is an object-oriented language, and everything is based on classes and objects.

- Class

A blueprint that defines attributes (variables) and methods (functions).

- Object

An instance of a class that can access its attributes and methods.

Example:

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print(f"Name: {self.name}, Age: {self.age}")
# Creating object
s1 = Student("Aayushi", 21)
s1.display()
```

2. Difference between local and global variables.

Ans:

Type	**Scope**	**Declared Inside**	**Accessible Where**
Local	**Inside function**	**Function**	**Only inside that function**
Global	**Outside function**	**Main program**	**Anywhere in the program**

```
x = 10  # Global variable
def show():
   y = 5  # Local variable
   print("Local:", y)
   print("Global:", x)
show()
```

7. Inheritance

Theory:

1. Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python.

Ans: Inheritance allows a class (child) to acquire properties and methods of another class (parent).

Type	Description	Example
Single Inheritance	One parent, one child	Child(Parent)
Multilevel Inheritance	Child inherits from parent, and another child inherits from that child	C(B(A))
Multiple Inheritance	One class inherits from multiple parents	Child(Parent1, Parent2)
Hierarchical Inheritance	Multiple children inherit from the same parent	Multiple Child(Parent)
Hybrid Inheritance	Combination of two or more types	Mixed structure

2. Using the super() function to access properties of the parent class.

Ans: The super() function allows a child class to access parent class methods or constructors.

```
class Parent:
  def __init__(self):
    print("Parent constructor")
class Child(Parent):
  def __init__(self):
    super().__init__()  # Access parent constructor
    print("Child constructor")
obj = Child()
```

7. Method Overloading and Overriding

1. **Method overloading: defining multiple methods with the same name but different parameters.**

Ans: Python does not support true method overloading (like Java or C++),
but you can achieve similar behavior using default arguments.

Example:

```
class Example:
  def display(self, a=None, b=None):
    if a != None and b != None:
      print(a + b)
    elif a != None:
      print(a)
    else:
      print("No arguments")
obj = Example()
obj.display()
obj.display(10)
obj.display(10, 20)
```

2. **Method overriding: redefining a parent class method in the child class.**

Ans: When a child class redefines a method from its parent class, it is called *overriding*.

Example:

```
class Parent:
  def show(self):
    print("Parent method")
class Child(Parent):
  def show(self):
    print("Child method")
obj = Child()
obj.show()
```

8. SQLite3 and PyMySQL (Database Connectors)

Theory:

1. Introduction to SQLite3 and PyMySQL for database connectivity.

Ans: Python supports databases using built-in and external connectors.

- SQLite3
  - A lightweight, file-based database built into Python.
  - Does not require a server.

Example:

```
import sqlite3

con = sqlite3.connect("student.db")

cur = con.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS student(name TEXT, age INT)")

cur.execute("INSERT INTO student VALUES ('Aayushi', 21)")

con.commit()

con.close()
```

- PyMySQL
  - A third-party module for connecting Python with MySQL database.

Example:

```
import pymysql

con = pymysql.connect(host='localhost', user='root', password='',
database='school')

cur = con.cursor()

cur.execute("INSERT INTO student VALUES ('Aayushi', 21)")

con.commit()

con.close()
```

2. Creating and executing SQL queries from Python using these connectors.

Ans: Python provides database connectivity through modules such as sqlite3 (for SQLite) and PyMySQL (for MySQL).
These connectors allow Python programs to interact with databases and perform SQL operations such as creating tables, inserting data, updating, deleting, and retrieving records.

- Steps:
  1. Import the connector (e.g., import sqlite3 or import pymysql)
  2. Establish a connection to the database using connect()
  3. Create a cursor object using cursor()
  4. Execute SQL queries with execute() method
  5. Commit changes using commit() (for insert, update, delete)
  6. Close the connection using close()
- Common SQL Queries:
  - CREATE TABLE – To create a new table
  - INSERT INTO – To insert data
  - SELECT – To retrieve data
  - UPDATE – To modify existing data
  - DELETE – To remove data
- Example Queries:

```
cur.execute("CREATE TABLE student(name TEXT, age INT)")

cur.execute("INSERT INTO student VALUES('Aayushi', 21)")

cur.execute("SELECT * FROM student")
```

9. Search and Match Functions

Theory:

1. **Using re.search() and re.match() functions in Python's re module for pattern matching.**

Ans: 🔹 re.search()

- Searches the entire string for the first occurrence of a pattern.
- Returns a match object if found, else None.

Example:

```
import re

text = "Python is fun"

result = re.search("fun", text)

print(result)
```

🔹 re.match()

- Checks only at the beginning of the string for a match.

Example:

```
import re

text = "Python is fun"

result = re.match("Python", text)

print(result)
```

2. **Difference between search and match.**

Ans:

Feature	re.search()	re.match()
Scope	Searches entire string	Checks only at the start
Return	Match object if found anywhere	Match object if found at beginning
Example	re.search("fun", "Python is fun") → Match	re.match("fun", "Python is fun") → No match