**Python – Full Stack Assignment**

**Module 8-Advance Python Programming :-**

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

The print() function in Python is used to display output to the console. It can print strings, numbers, variables, or any object. It is commonly used for debugging and interaction.

print("Hello, World!")

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

Python offers f-strings (formatted string literals) and the format() method for string formatting. f-strings are prefixed with 'f' and allow expressions inside curly braces. format() replaces placeholders with values.

name = "Alice"

print(f"Hello, {name}")

print("Hello, {}".format(name))

## 3. Using the input() function to read user input from the keyboard

The input() function is used to take user input from the console. It always returns the input as a string.

name = input("Enter your name: ")

print("Hello", name)

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

User inputs are strings by default. Functions like int(), float(), and str() are used to convert the input into appropriate data types as needed.

age = int(input("Enter your age: ")) # String to int

price = float(input("Enter price: ")) # String to float

## 5. Opening files in different modes ('r', 'w', 'a', 'r+', 'w+')

'r' opens a file for reading, 'w' for writing (overwrites), 'a' for appending, 'r+' for reading and writing, and 'w+' for writing and reading (overwrites).

file = open("data.txt", "r")

## 6. Using the open() function to create and access files

The open() function opens a file and returns a file object. It requires a file name and mode. It is used for reading, writing, or appending to files.

f = open("example.txt", "w")

f.write("Hello!")

f.close()

## 7. Closing files using close()

The close() method is used to close a file after operations are done. It ensures resources are freed and data is saved properly.

f = open("file.txt")

# do something

f.close()

## 8. Reading from a file using read(), readline(), readlines()

read() reads the entire content, readline() reads one line, and readlines() reads all lines into a list.

f = open("file.txt", "r")

print(f.read()) # Reads entire file

print(f.readline()) # Reads one line

print(f.readlines()) # Reads all lines into a list

f.close()

## 9. Writing to a file using write() and writelines()

write() writes a string to the file. writelines() writes a list of strings. Both methods are used for output to files.

f = open("file.txt", "w")

f.write("Hello\nWorld")

f.writelines(["Line1\n", "Line2\n"])

f.close()

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

Exceptions are errors detected during execution. try-except blocks handle these gracefully. finally runs regardless of exceptions.

try:

x = 10 / 0

except ZeroDivisionError:

print("Cannot divide by zero")

finally:

print("This always runs")

## 11. Understanding multiple exceptions and custom exceptions

Multiple exceptions can be handled using multiple except blocks or tuples. Custom exceptions are user-defined by subclassing the Exception class.

try:

x = int("abc")

except (ValueError, TypeError) as e:

print("Error:", e)

# Custom Exception

class MyError(Exception):

pass

raise MyError("Something went wrong")

## 12. Understanding the concepts of classes, objects, attributes, and methods in Python

A class is a blueprint. An object is an instance. Attributes are variables, and methods are functions defined in a class.

class Car:

def \_\_init\_\_(self, brand):

self.brand = brand

def drive(self):

print(f"{self.brand} is driving")

c = Car("Toyota")

c.drive()

## 13. Difference between local and global variables

Local variables are defined within a function. Global variables are defined outside and accessible throughout the module.

x = 10 # global

def show():

x = 5 # local

print("Local:", x)

show()

print("Global:", x)

## 14. Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python

Inheritance allows classes to inherit features from others. Types: Single (one parent), Multilevel (chain), Multiple (many parents), Hierarchical (one parent, many children), and Hybrid (combination).

class A: pass

class B(A): pass # Single

class C(B): pass # Multilevel

class D: pass

class E(A, D): pass # Multiple

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

super() is used to call methods or constructors of a parent class, especially in method overriding scenarios.

class Parent:

def show(self):

print("Parent")

class Child(Parent):

def show(self):

super().show()

print("Child")

## 16. Method overloading: defining multiple methods with the same name but different parameters

Python doesn't support method overloading directly. It can be simulated using default arguments or variable arguments.

class Demo:

def show(self, a=None, b=None):

if a and b:

print(a + b)

elif a:

print(a)

else:

print("No arguments")

## 17. Method overriding: redefining a parent class method in the child class

Overriding occurs when a child class defines a method with the same name as a method in the parent class, replacing its behavior.

class Parent:

def greet(self):

print("Hello from parent")

class Child(Parent):

def greet(self):

print("Hello from child")

## 18. Introduction to SQLite3 and PyMySQL for database connectivity

SQLite3 is a lightweight, built-in database. PyMySQL is a third-party library for connecting to MySQL databases in Python.

## 19. Creating and executing SQL queries from Python using these connectors

SQL queries can be executed using cursor objects after establishing connections with SQLite3 or PyMySQL. Use execute() and commit() for operations.

import sqlite3

conn = sqlite3.connect("test.db")

cursor = conn.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER, name TEXT)")

conn.commit()

conn.close()

import pymysql

conn = pymysql.connect(host="localhost", user="root", password="", db="test")

cursor = conn.cursor()

cursor.execute("SELECT \* FROM users")

conn.close()

## 20. Using re.search() and re.match() functions in Python’s re module for pattern matching

re.search() scans the entire string, while re.match() checks only the beginning. Both are used for regex-based pattern matching.

import re

print(re.search("abc", "123abc")) # Matches anywhere

print(re.match("abc", "abc123")) # Matches at start only

## 21. Difference between search and match

re.search() returns a match if the pattern is found anywhere in the string. re.match() only returns a match if the pattern starts at the beginning.