Module – 3 : Advance Python Programming

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

- **print()** ---> Whatever You Say.. You want to print out on your console window.
  Ex :
  print("Hello, Python!!") **#** output :: Hello, Python!!

- **end = " "** ---> This Parameter is used to add the string at the end of print().
  Ex :
  print("Hello, Python!!" , end = "--")
  print("Easy to Learn") **#** output :: Hello, Python!!—Easy to Learn

- **sep = " "** ---> This Parameter is used to specified separator between passed value and print().
  Ex :
  print("Hello", "Python!!" , sep = "--")
  # output :: Hello--Python!!

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

- **format() :** This Parameter is used to format specified value and insert inside the string placeholder.
  Ex :
  name = "Vivek"
  age = 21

  print("My name is {} and my age is {}.".format(name, age))
  print(f"My name is {name} and my age is {age}.")

  # output :: My name is Vivek and my age is 21.

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

- **input() ::** This Function is used to take input directly from the user through the keyboard.
- **Ex** : **num = int(input("Enter Value :: "))**
  print(num) # output --> Enter Value :: 2 ---> print : 2

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

- **Integer**

  Ex :

  ```
  num = int(input("Enter Value :: "))
  print(num, type(num))
  # output --> Enter Value :: 15 --> print : 15 <class 'int'>
  ```

- **Float**

  Ex :

  ```
  num = float(input("Enter Value :: "))
  print(num, type(num))
  # output --> Enter Value :: 15.75 --> print : 15.75 <class 'float'>
  ```

- **String**

  Ex :

  ```
  my_str = input("Enter String :: "))
  print(my_str, type(my_str))
  # output --> Enter String:: Python --> print : Python <class 'str'>
  ```

- **Boolean**

  Ex :

  ```
  x = True
  print(x, type(x))
  # output --> print : True <class 'bool'>
  ```

- **List**

  Ex :

  ```
  x = [1, 2, 3]
  print(x, type(x))
  # output --> print : [1, 2, 3] <class 'list'>
  ```

- **Tuple**

  Ex :

  ```
  x = (1, 2, 3)
  print(x, type(x))
  # output --> print : (1, 2, 3) <class 'tuple'>
  ```

- **Set**

  Ex :

  ```
  x = {1, 2, 3}
  ```

print(x, type(x))
 # output --> print : {1, 2, 3} <class 'set'>

- **Dictionary**
  Ex :

```
x = {"id": 101, "name" : "Vivek"}
print(x, type(x))
# output --> print : {'id': 101, 'name' : 'Vivek'} <class 'dict'>
```

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

- **'r' –> Read Mode :** Opens the file for reading only.
  Ex :

```
x = open('Demo.txt', 'r')
print(x.read())
```

- **'w' –> Write Mode :** Opens the file for writing and create both.
  Ex :

```
x = open('Demo.txt', 'w')
x.write("Hello, Python!!")
```

- **'a' –> Append Mode :** Opens the file for writing but adds to the end instead of overwriting and create if doesn't exists.
  Ex :

```
x = open('Demo.txt', 'a')
x.write("It's very easy to learn.")
```

- **'r+' –> Read and Write :** Opens the file for both reading and writing. File must already exist, also overwrite existing content.
  Ex :

```
x = open('Demo.txt', 'r+')
print(x.read())
x.write("It's high level programming language..")
```

- **'w+' –> Write and Read :** Opens the file for both writing and reading. If the file exists, it is erased first. If it doesn't exist, a new file is created.
  Ex :

```
x = open('Demo.txt', 'w+')
x.write("It's interpreted Languages..")
print(x.read())
```

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

- x = open(“Task.txt”, “a”)

```
n = int(input(“Enter Number of Student Info : ”))

for i in range(n):
   stu_name = input(“Enter Student Name : ”)
   stu_sub = input(“Enter Subject Name : ”)
   stu_city = input(“Enter City Name : ”)
   print(“----------------------”)

x.write(f“Timestemp : {dt.datetime.now()}\n”)
x.write(f“ID : {random.randint(1000,9999)}\n”)
x.write(f“Student Name : {stu_name}\n”)
x.write(f“Subject Name : {stu_sub}\n”)
x.write(f“City Name : {stu_city}\n”)
x.write(f“------------------------------\n”)

print(“Done....!!”)
```

7) Closing files using close().

- When you open a file with open(), it stays open in memory until you close it. Closing a file is important because it..

- Frees up system resources
- Ensures all data is properly written to disk

- **x = open(“Temp.txt”, “w”)**
  x.write(“Hello!!, I am learning Python!!”)

  x.close()

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

- **read() :** reads the whole file.

Ex :

```
x = open(“Demo.txt”, “r”)
print(x.read())          # reads entire file
```

- **readline() :** reads one line at a time.

Ex : x = open("Demo.txt", "r")

print(x.readline()) # first line

- **readlines() :** reads all lines into a list.

Ex : x = open("Demo.txt", "r")

print(x.readlines()) # each line becomes an item in the list

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

- **write() :** writes a single string.

Ex : x = open("Demo.txt", "w")

x.write("Hello! This is file handling…")

- **writelines() :** writes a list of strings

Ex : list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

x = open("Demo.txt", "w")

x.writelines(list)

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

- **Exceptions** : Errors that happen while a program is running. Instead of crashing the whole program when something goes wrong, Python lets you catch and handle these errors gracefully.
- **Ex** :

```
try:
    result = 10 / 0                          # Code that might cause an error
except ZeroDivisionError:
    print("You can't divide by zero!")       # Code that runs if an error happens
finally:
    print("Execution complete.")             # Code that always runs
```

11) Understanding multiple exceptions and custom exceptions.

- **Multiple Exception :**

  Ex :

```
try:
   x = int("abc")
   y = 10 / 0
except ValueError:
   print("Invalid conversion.")
except ZeroDivisionError:
   print("Division by zero.")
except Exception as e:
   print("Error :: ",e)
```

- **Custom Exception :**

  Ex :

```
class AgeRestriction(Exception):    # Define a custom exception
        pass

def check_age(age):
        if age < 18:
                raise AgeRestriction("You must be at least 18 years old.")
        else:
                print("Done…!!")

try:                        # Using a custom exception
        check_age(15)
except AgeRestriction as e:
        print("Error :: ", e)
```

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

- **Class :**
  - Class is collection of data member and member of function.
  - Class is blueprint of an object.
  - **Ex** :

```
class Student:
        pass
```

- **Object :**
  - Object is an instance of a class.
  - It's Real World Entity.
  - **Ex** :

```
stu = Student()
```

- **Attributes :**
  - Attributes are variables that belong to a class or object.
  - They store information about the object.
  - **Ex** :

```
class Student:          # empty
    pass

stu = Student()         # Object

stu.name = "Vivek"
stu.age = 21
stu.grade = "A"

print(stu.name)         # Vivek
print (stu.age)         # 21
print(stu.grade)        # A
```

- **Methods :**
  - Methods are functions defined inside a class...
  - They describe the behavior of the object.
  - **Ex** :

```
class Calculator:
    def add(self, a, b):
        return a + b

    def mul(self, a, b):
        return a * b

calc = Calculator()

print(calc.add(10, 5))          # 15
print(calc.mul(5, 5))           # 25
```

13) Difference between local and global variables.

Feature	Local Variable	Global Variable
Defined	Inside a function	Outside all functions
Scope	Only within the function	Anywhere in the program
Lifetime	Exists only during function execution	Exists throughout the program
Access	Not accessible outside the function	Accessible anywhere
Ex :	def add(): **a = 10** **b = 20** print("Sum :: ",a+b) add() # 30	**a = 10** **b = 20** def add(): print("Sum :: ",a+b) add() # 30

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

- **Single :** When a child class inherits from only one parent class.
  Ex :

```
class Parent:
        def display(self):
        print("This is the parent class.")

class Child(Parent):
        def show(self):
        print("This is the child class.")

obj = Child()
obj.display()
obj.show()
```

- **Multilevel :** When a class is derived from another derived class (grandparent --> parent --> child).
  Ex :

```
class Animal:
        def eat(self):
        print("Animals can eat.")

class Dog(Animal):
        def bark(self):
        print("Dogs can bark.")

class Puppy(Dog):
```

```
        def weep(self):
            print("Puppies can weep.")

    p = Puppy()
    p.eat()
    p.bark()
    p.weep()
```

- **Multiple :** When a class inherits from more than one parent class.
  Ex :

```
    class Bird:
        def fly(self):
            print("This animal can fly.")

    class Fish:
        def swim(self):
            print("This animal can swim.")

    class Duck(Bird, Fish):
        def sound(self):
            print("Duck quacks.")

    d = Duck()
    d.fly()
    d.swim()
    d.sound()
```

- **Hierarchical :** When multiple child classes inherit from a single parent.
  Ex :

```
    class Animal:
        def eat(self):
            print("Animals can eat.")

    class Dog(Animal):
        def bark(self):
            print("Dogs bark.")

    class Cat(Animal):
        def meow(self):
            print("Cats meow.")
```

```
d = Dog()
d.eat()
d.bark()

c = Cat()
c.eat()
c.meow()
```

- **Hybrid :** A combination of two or more types of inheritance.
  Ex :

```
class A:
        def featureA(self):
        print("Feature from A.")

class B(A):
        def featureB(self):
        print("Feature from B.")

class C(A):
        def featureC(self):
        print("Feature from C.")

class D(B, C):                  # Multiple + hierarchical
        def featureD(self):
         print("Feature from D.")

obj = D()
obj.featureA()
obj.featureB()
obj.featureC()
obj.featureD()
```

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

- **Ex :**

```
class master:
   def header(self, home, about, contact):
      print("Home ::",home)
      print("About ::",about)
      print("Contact ::",contact)
```

```
class index(master):
   def header(self, home, about, contact):
      return super().header(home, about, contact)

class profile(master):
   def header(self, home, about, contact):
      return super().header(home, about, contact)

ind = index()
ind.header("Home", "About", "Contact")
```

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

- **Method Overloading :** Same Name and Different Parameter
  Ex :

```
class stud_info:
   def getdata(self, id):
      print("Id : ",id)

   def getdata(self, name):
      print("Name : ",name)

st = stud_info()
st.getdata("Vivek")
st.getdata(12)
```

- **Use Default Argument :**

```
class Calculator:
   def add(self, a, b=0, c=0):
      return a + b + c

calc = Calculator()
print(calc.add(5))
print(calc.add(5, 10))
print(calc.add(5, 10, 15))
```

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

- **Method Overriding : Same Name and Same Parameter but Class are Different.**

  Ex :

```
class Vehicle:
        def move(self):
        print("The vehicle moves.")

class Car(Vehicle):
        def move(self):
        print("The car drives on the road.")

class Boat(Vehicle):
        def move(self):
        print("The boat sails on water.")

v = Vehicle()
c = Car()
b = Boat()

print(v.move())
print(c.move())
print(b.move())
```

18) Introduction to SQLite3 and PyMySQL for database connectivity.

- **SQLite3 :**
  - Built-in with Python, no setup.
  - Stores data in a single file, good for small apps.
  - Use sqlite3 module.

  Ex :

```
import sqlite3
conn = sqlite3.connect("test.db")
cur = conn.cursor()
cur.execute("CREATE TABLE IF NOT EXISTS users(id INTEGER, name
TEXT)")
cur.execute("INSERT INTO users VALUES(1, 'Vivek')")
conn.commit()
conn.close()
```

- **PyMySQL :**
  - Needs pip install pymysql and MySQL server.
  - Good for large, multi-user apps.
  - Use pymysql library.

  Ex :

```
import pymysql

conn = pymysql.connect(host="localhost", user="root",
password="pass", database="testdb")

cur = conn.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS users(id INT, name
VARCHAR(50))")

cur.execute("INSERT INTO users VALUES(1, 'Meet')")

conn.commit()
conn.close()
```

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

- **SQLite3 :**

```
import sqlite3
conn = sqlite3.connect("mydb.db")
cur = conn.cursor()

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

cur.execute("INSERT INTO users (name) VALUES (?)", ("Vivek",))

cur.execute("SELECT * FROM users")
print(cur.fetchall())

cur.execute("UPDATE users SET name=? WHERE id=?", ("Meet", 1))

cur.execute("DELETE FROM users WHERE id=?", (1,))
conn.commit()
conn.close()
```

- **PyMySQL :**

```
import pymysql
conn = pymysql.connect(
   host="localhost", user="root", password="yourpass",
database= "testdb"
)
cur = conn.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS users(id INT
AUTO_INCREMENT PRIMARY KEY, name VARCHAR(50))")

cur.execute("INSERT INTO users (name) VALUES (%s)", ("Meet",))

cur.execute("SELECT * FROM users")
print(cur.fetchall())

cur.execute("UPDATE users SET name=%s WHERE id=%s" ("Kashyap",
1))

cur.execute("DELETE FROM users WHERE id=%s", (1,))
conn.commit()
conn.close()
```

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

- **re.search() :**
  - Scans the entire string.
  - returns the first occurrence of the pattern.

Ex :

```
import re

my_str = input("Enter String :: ")

x = re.search("Python", my_str)

print(x)

if x:

        print("Match Successfully..!!")

else:

        print("Error..!!")
```

- **re.match() :**
  - Tries to match the pattern only at the beginning of the string.
  - Returns a match object if found, otherwise None.

Ex :

```
import re
my_str = input("Enter String :: ")
x = re.match("Python", my_str)
print(x)
if x:
    print("Match Successfully..!!")
else:
    print("Error..!!")
```

21) Difference between search and match.

Feature	re.match()	re.search()
Scope	Checks only at the beginning	Scans the entire string
Return	Match object if pattern at start only	Match object if pattern found anywhere
Use case	Validate if string starts with	Find pattern anywhere in text
Example	re.match("Hi", "Hi there")	re.search("there", "Hi there")