Module 8) Advance Python Programming

1. Printing on Screen

Question-1: Introduction to the print() function in Python.

Answer: The print() function prints the specified message to the screen, or other standard output device.

The message can be a string, or any other object, the object will be converted into a string before written to the screen.

print("Hello", "how are you?")

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

Answer: Python offers a powerful feature called f-strings (formatted string literals) to simplify string formatting and interpolation. f-strings is introduced in Python 3.6 it provides a concise and intuitive way to embed expressions and variables directly into strings. The idea behind f-strings is to make string interpolation simpler.

val = 'Geeks'

print(f"**{**val**}**for**{**val**}** is a portal for **{**val**}**.")

Output

GeeksforGeeks is a portal for Geeks.

2. Reading Data from Keyboard

Question-1: Using the input() function to read user input from the keyboard.

Answer: input () function first takes the input from the user and converts it into a string. The type of the returned object always will be <class 'str'>. It does not evaluate the expression it just returns the complete statement as String.

```
val = input("Enter your value: ")

print(val)
```

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

Answer: There are various function that are used to take as desired input few of them are : –

- int(input())
- float(input())

```
num = int(input("Enter a number: "))

print(num, " ", type(num)

floatNum = float(input("Enter a decimal number: "))

print(floatNum, " ", type(floatNum))
```

3. Opening and Closing Files

Question-1: Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').

Answer:

Read Mode ('r') in Python

This mode allows you to open a file for reading only. If the file does not exist, it will raise a FileNotFoundError.

Write Mode ('w') in Python

This mode allows you to open a file for writing only. If the file already exists, it will truncate the file to zero length. If the file does not exist, it will create a new file.

Append Mode ('a') in Python

This mode allows you to open a file for appending new content. If the file already exists, the new content will be added to the end of the file. If the file does not exist, it will create a new file.

Read and Write Mode ('r+') in Python

This mode allows you to open a file for both reading and writing. The file pointer will be positioned at the beginning of the file. If the file does not exist, it will raise a FileNotFoundError.

Write and Read Mode ('w+') in Python

This mode allows you to open a file for both reading and writing. If the file already exists, it will truncate the file to zero length. If the file does not exist, it will create a new file.

Question-2: Using the open() function to create and access files.

Answer: Opening a file refers to getting the file ready either for reading or for writing. This can be done using the open() function. This function returns a file object and takes two arguments, one that accepts the file name and another that accepts the mode(Access Mode).

Syntax of open() Function

File_object = open("File_Name", "Access_Mode")

Parameters:

- File_Name: This is the name of the file you want to open.
- Access_Mode: This specifies the mode in which the file will be opened.

Question-3: Closing files using close().

Answer: The close() method closes an open file.

You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Syntax

file.close()

4. Reading and Writing Files

Question-1: Reading from a file using read(), readline(), readlines().

Answer: Python provides built-in functions for creating, writing, and reading files. Two types of files can be handled in Python, normal text files and binary files (written in binary language, 0s, and 1s). In this article, we are going to study reading line by line from a file.

```
with open('filename.txt', 'r') as file:
    # Read each line in the file
    for line in file:
        # Print each line
        print(line.strip())
```

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

Answer:

write() function

The write() function will write the content in the file without adding any extra characters.

Syntax:

```
# Writes string content referenced by file object.
file_name.write(content)
file = open("Employees.txt", "w")

for i in range(3):
    name = input("Enter the name of the employee: ")
    file.write(name)
```

```
    file.write("\n")
file.close()
print("Data is written into the file.")
```

Output:

```
Data is written into the file.
```

writelines() function

This function writes the content of a list to a file.

Syntax:

```
# write all the strings present in the list "list_of_lines"
# referenced by file object.
file_name.writelines(list_of_lines)
```

```
file1 = open("Employees.txt", "w")
lst = []
for i in range(3):
    name = input("Enter the name of the employee: ")
    lst.append(name + '\n')

file1.writelines(lst)
file1.close()
print("Data is written into the file.")
```

Output:

```
Data is written into the file.
```

5. Exception Handling

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

Answer: An Exception is an Unexpected Event, which occurs during the execution of the program. It is also known as a run time error. When that error occurs, Python generates an exception during the execution and that can be handled, which prevents your program from interrupting.

- Try: This block will test the excepted error to occur
- Except: Here you can handle the error
- Else: If there is no exception then this block will be executed
- Finally: Finally block always gets executed either exception is generated or not

Question-2: Understanding multiple exceptions and custom exceptions.

Answer: To define a custom exception in Python, you need to create a new class that inherits from the built-in Exception class or one of its subclasses. Here's a basic example:

```
class MyCustomError(Exception):
    """Exception raised for custom error scenarios.
    Attributes:
        message -- explanation of the error
    """
```

```
def __init__(self, message):
    self.message = message
    super().__init__(self.message)
```

6. Class and Object (OOP Concepts)

Question-1: Understanding the concepts of classes, objects, attributes, and methods in Python.

Answer:

- **Class**

A template or blueprint for creating objects. Classes are defined using the class keyword.

- **Object**

A concrete instance of a class. Objects are created based on the blueprint of a class.

- **Attributes**

Variables that store data for a specific instance of a class. Attributes can be accessed and modified by both the class and its objects.

- **Methods**

Functions defined within a class that operate on the attributes of an object. Methods define behaviors

Question-2: Difference between local and global variables.

Answer:

Comparision Basis	**Global Variable**	**Local Variable**
Definition	declared outside the functions	declared within the functions

Lifetime	They are created the execution of the program begins and are lost when the program is ended	They are created when the function starts its execution and are lost when the function ends
Data Sharing	Offers Data Sharing	It doesn’t offers Data Sharing
Scope	Can be access throughout the code	Can access only inside the function
Parameters needed	parameter passing is not necessary	parameter passing is necessary
Storage	A fixed location selected by the compiler	They are kept on the stack
Value	Once the value changes it is reflected throughout the code	once changed the variable don’t affect other functions of the program

7. Inheritance

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

Answer:

Single Inheritance:

Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.

Multiple Inheritance:

When a class can be derived from more than one base class this type of inheritance is called multiple inheritances. In multiple inheritances, all the features of the base classes are inherited into the derived class.

Multilevel Inheritance :

In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and a grandfather.

Hierarchical Inheritance:

When more than one derived class are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.

Hybrid Inheritance:

Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

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

Answer: In Python, the super() function is used to refer to the parent class or superclass. It allows you to call methods defined in the superclass from the subclass, enabling you to extend and customize the functionality inherited from the parent class.

Syntax of super() in Python

Syntax: *super()*

Return : *Return a proxy object which represents the parent's class.*

```
class Emp():
    def __init__(self, id, name, Add):
        self.id = id
        self.name = name
        self.Add = Add
# Class freelancer inherits EMP
class Freelance(Emp):
    def __init__(self, id, name, Add, Emails):
        super().__init__(id, name, Add)
        self.Emails = Emails
Emp_1 = Freelance(103, "Suraj kr gupta", "Noida" , "KKK@gmails")
print('The ID is:', Emp_1.id)
print('The Name is:', Emp_1.name)
print('The Address is:', Emp_1.Add)
print('The Emails is:', Emp_1.Emails)
```

8. Method Overloading and Overriding

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

Answer: Two or more methods have the same name but different numbers of parameters or different types of parameters, or both. These methods are called overloaded methods and this is called method overloading.

Like other languages (for example, method overloading in C++) do, python does not support method overloading by default. But there are different ways to achieve method overloading in Python.

The problem with method overloading in Python is that we may overload the methods but can only use the latest defined method.

```
def product(a, b):
    p = a * b
    print(p)


def product(a, b, c):
    p = a * b*c
    print(p)


product(4, 5, 5)
```

Output

100

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

Answer: Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, the same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to **override** the method in the super-class.

```
# Python program to demonstrate

# Defining parent class

class Parent():


    # Constructor

    def __init__(self):

        self.value = "Inside Parent"


    # Parent's show method

    def show(self):

        print(self.value)


# Defining child class

class Child(Parent):


    # Constructor
```

```
    def __init__(self):
        super().__init__()  # Call parent constructor
        self.value = "Inside Child"

    # Child's show method
    def show(self):
        print(self.value)

# Driver's code
obj1 = Parent()
obj2 = Child()
obj1.show()  # Should print "Inside Parent"
obj2.show()  # Should print "Inside Child"
```

Output:

```
Inside Parent
Inside Child
```

9. SQLite3 and PyMySQL (Database Connectors)

Question-1: Introduction to SQLite3 and PyMySQL for database connectivity.

Answer: Databases offer numerous functionalities by which one can manage large amounts of information easily over the web and high-volume data input and output over a typical file such as a text file. SQL is a query language and is very popular in databases. Many websites use MySQL. SQLite is a “light” version that works over syntax very much similar to SQL. SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. It is the most used database engine on the world wide web. Python has a library to access SQLite databases, called sqlite3, intended for working with this database which has been included with Python package since version 2.5. SQLite has the following features.

1. Serverless
2. Self-Contained
3. Zero-Configuration
4. Transactional
5. Single-Database

Connecting to the Database

Connecting to the SQLite Database can be established using the **connect()** method, passing the name of the database to be accessed as a parameter. If that database does not exist, then it'll be created.

```
sqliteConnection = sqlite3.connect('sql.db')
```

But what if you want to execute some queries after the connection is being made. For that, a cursor has to be created using the **cursor()** method on the connection instance, which will execute our SQL queries.

```
cursor = sqliteConnection.cursor()

print('DB Init')
```

The SQL query to be executed can be written in form of a string, and then executed by calling the **execute()** method on the cursor object. Then, the result can be fetched from the server by using the **fetchall()** method, which in this case, is the SQLite Version Number.

```
query = 'SQL query;'

cursor.execute(query)

result = cursor.fetchall()

print('SQLite Version is {}'.format(result))
```

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

Answer:

SQLite3 Queries – Creating Tables

After connecting to the database and creating the cursor object let’s see how to execute the queries.

- To execute a query in the database, create an object and write the SQL command in it with being commented. Example:- sql_comm = ”SQL statement”
- And executing the command is very easy. Call the cursor method execute() and pass the name of the sql command as a parameter in it. Save a number of commands as the sql_comm and execute them. After you perform all your activities, save the changes in the file by committing those changes and then lose the connection.

```
import sqlite3


# connecting to the database

connection = sqlite3.connect("gfg.db")


# cursor

crsr = connection.cursor()


# SQL command to create a table in the database

sql_command = """CREATE TABLE emp (

staff_number INTEGER PRIMARY KEY,

fname VARCHAR(20),

lname VARCHAR(30),

gender CHAR(1),

joining DATE);"""
```

```
# execute the statement
crsr.execute(sql_command)

# close the connection
connection.close()
```

10. Search and Match Functions

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

Answer:

```
# import re module
import re

Substring ='string'


String1 ='''We are learning regex with geeksforgeeks
        regex is very useful for string matching.
         It is fast too.'''
String2 ='''string We are learning regex with geeksforgeeks
        regex is very useful for string matching.
         It is fast too.'''

# Use of re.search() Method
print(re.search(Substring, String1, re.IGNORECASE))
# Use of re.match() Method
print(re.match(Substring, String1, re.IGNORECASE))

# Use of re.search() Method
print(re.search(Substring, String2, re.IGNORECASE))
```

Use of re.match() Method

print(re.match(Substring, String2, re.IGNORECASE))

Output :

<re.Match object; span=(69, 75), match='string'>
None
<re.Match object; span=(0, 6), match='string'>
<re.Match object; span=(0, 6), match='string'>

Question-2: Difference between search and match.

Answer: The re.search() and re.match() both are functions of re module in python. These functions are very efficient and fast for searching in strings. The function searches for some substring in a string and returns a match object if found, else it returns none.

There is a difference between the use of both functions. Both return the first match of a substring found in the string,
but re.match() searches only from the beginning of the string and return match object if found. But if a match of substring is found somewhere in the middle of the string, it returns none.
While re.search() searches for the whole string even if the string contains multi-lines and tries to find a match of the substring in all the lines of string.