1. Printing on Screen

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

Ans. print() function in Python is a built-in function used to display output on the console. It is one of the most commonly used functions.

Ex.

Print("Hello Python")

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

Ans. f-strings and the format() method allow flexible and readable string formatting by embedding variables or expressions directly into strings.

Ex.

Print(f "Name:,{name}")

2. Reading Data from Keyboard

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

Ans. input() function in Python is a built-in function used to read user input from the keyboard as a string. It's commonly used for interactive programs that require input from the user during runtime.

Ex.

Id=int(input("Enter your id: ")

Name=input("Enter your name: ")

Print("ID:", Id)

Print("Hello", Name)

Q.4 Converting user input into different data types.

Ans. To convert user input into different data types in Python, you can use type conversion functions such as int(), float(), bool(), etc. This is necessary because the input() function always returns the input as a string**.**

Ex.

```
Number=int(input("Enter Any Number: ")

Value=float(input("Enter Any Number: ")

Print(f "This number is integer: ,{Number}")

Print(f "This number is float : ,{Value}")
```

3. Opening and Closing Files

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

Ans. Python provides various modes to open files using the open() function. The mode determines how the file is accessed (e.g., reading, writing, appending). Here's a breakdown of the most common file modes:

Syntax: file = open("filename", mode)

Ex.

- **'w' (Write)**

```
fl=open("file_operations.txt","w")

print(fl.write("Welcome to the python programming\nhello students"))
```

- **'r' (Read)**

```
fl=open("file_operations.txt","r")

print(fl.read())
```

- **'a' (Append)**

```
fl=open("file_operations.txt","a")

print(fl.write("\nPython is easy to learn"))
```

❖ **'r+' (Read and Write)**

```
fl=open("file_operations.txt","r+")
print(fl.read())
print(fl.write("\nAdd data after read file"))
```

❖ **'w+' (Write and Read)**

```
fl=open("file_operations.txt","w+")
print(fl.write("\nall file operations done"))
print(fl.read())
```

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

Ans. open() function in Python is used to create, read, write, and manipulate files. Depending on the **mode** specified, you can create and access files for different purposes like reading, writing, appending, etc.

Ex.

```
Open(“demo.txt”,“w”)
Fl= Open(“demo.txt”,“w”)
Print(fl.write(“Hello World”)
```

Q.7 Closing files using close().

Ans. close() method is used to close a file after it has been opened using the open() function.

Ex.

```
Fl=open(“demo.txt” ,“r”)
Print(fl.read( ))
Fl.close( )
```

4. Reading and Writing Files

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

Ans. **Ex.**

```
fl=open("temp.txt","r")
print(fl.read())
print(fl.readline())
print(fl.readlines())
```

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

Ans.

Ex.

```
fl=open("temp.txt",'w')
print(fl.write("\n it's very easy to learn"))
print(fl.writelines("\nit is heigh level programming language.\npython does not support method overloading"))
```

5. Exception Handling

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

Ans. Exceptions are errors that happen during a program's execution, like dividing by zero or trying to open a file that doesn't exist. Python lets you handle these errors so your program doesn’t crash.

How to Handle Exceptions?

❖ Use try, except, and finally blocks:

1. **try**: Use try to test for errors.
2. **except**: Use except to handle specific errors.
3. **finally**: Code that always runs, even if there's an error.

Ex.

```
try:
   a=int(input("Enter A: "))
   b=int(input("Enter B: "))
   print("Sum: ",a+b)

except Exception as e:
   print(e)

finally:
   print("This is finally block")
```

Q.11 Understanding multiple exceptions and custom exceptions.

Ans. Python allows you to handle multiple exceptions using multiple except blocks or by combining them into a single block.

Sometimes, different types of errors can occur in a program, and you may want to handle them separately.

➢ multiple exceptions

Ex.

```
try:

   num1 = int(input("Enter the first number: "))
   num2 = int(input("Enter the second number: "))

   result = num1 / num2
   print("The result is:", result)

except ZeroDivisionError:
   print("Division by zero is not allowed!")

except Exception as e:
```

```
    print("An unexpected error occurred:", e)

print("Program continues...")
```

- **custom exceptions.**

Ex.

```
try:
    a=int(input("Enter A:"))
    b=int(input("Enter B:"))
    print("Sum:",a+b)
except:
    print("Error!")
```

6. Class and Object (OOP Concepts)

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

Ans.

1. Classes

- A class is a blueprint or template for creating objects.
- It defines the structure and behavior (attributes and methods) that the objects created from the class will have.
- Use the class keyword to define a class.

2. Objects

- An object is an instance of a class.
- When a class is instantiated, it creates an object.

Objects have attributes (data) and methods (functions) associated with them.

Ex.

```
class data:
    id=7
    name= "devangi"

    def print_data(self):
        print("id: ", self.id)
        print("name: ",self.name)

dt=data()
dt.print_data()
```

3. Attributes

- Attributes are variables that belong to an object.
- They store data or properties related to the object.
- You define attributes inside the class and initialize them using the __init__ method.

4. Methods

- Methods are functions defined inside a class.
- They describe the behavior of the objects.
- Methods often operate on the object's attributes and can perform actions.

Ex.

```
class stud:
    def __init__(self, id,name):
        self.id = id
        self.name = name

    def print_data(self):
        print(f"ID:{self.id} \nName:{self.name}")

s = stud("21", "Krishna")
s.print_data()
```

Q.13 Difference between local and global variables.

Ans.

Local Variable	Global Variable
A variable declared inside a function or block.	A variable declared outside any function or block.
Accessible only within the function or block.	Accessible throughout entire program.
It can be modified only within the function or block.	It can be accessed and modified anywhere in the program.
Used for temporary storage specific to a function's operation.	Used for values that need to be shared across functions.

7. Inheritance

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

Ans.

1. Single Inheritance

In single inheritance, a child class inherits from a single parent class.

Ex.

```
class father:
    def get_data(self):
        print("This Is Parent Class")

class daughter(father):
    def get_child_data(self):
        print("This Is Child Class")

d=daughter()
d.get_data()
d.get_child_data()
```

2. Multilevel Inheritance

In multilevel inheritance, a class inherits from a parent class, and another class inherits from this child class, forming a chain.

Ex.

```
class grandparent:
    def getdata(self):
        print("This Is GrandParent Class")

class parent(grandparent):
    def get_data(self):
        print("This Is Parent Class")

class child(parent):
    def get_child_data(self):
        print("This Is Child Class")

c=child()
c.getdata()
c.get_data()
c.get_child_data()
```

3. Multiple Inheritance

In multiple inheritance, a child class inherits from two or more parent classes. This allows the child class to access properties and methods of all its parents.

Ex.

```
class Daughters:
    def getdata1(self):
        print("Hello From Daughters")

class Son:
    def getdata2(self):
        print("Hello From Son")

class Parents(Daughters,Son):
    def getdata(self):
        print("Hello From Parents")

p=Parents()
p.getdata()
```

```
p.getdata1()
p.getdata2()
```

4. Hierarchical Inheritance

In hierarchical inheritance, multiple child classes inherit from a single parent class. This is useful when several classes share the same base functionality.

Ex.

```
class Parents:
    def getdata(self):
        print("This is parents class!")

class Daughters(Parents):
    def getdata1(self):
        print("This is daughters class!!!")

class Son(Parents):
    def getdata2(self):
        print("This is son's class!!!")

obj1=Son()
obj1.getdata()
obj1.getdata2()

obj2=Daughters()
obj2.getdata()
obj2.getdata1()
```

5. Hybrid Inheritance

Hybrid inheritance combines two or more types of inheritance to form a more complex hierarchy.

Ex.

```
class Parents:
    def getdata(self):
        print("Hello from parents")

class Child1(Parents):
    def getdata1(self):
        print("Hello from Child1")

class Child2(Parents):
    def getdata2(self):
```

```
        print("Hello from Child2")

class GrandChild(Child1,Child2):
    def getdata3(self):
        print("Hello from Grandchild")

g=GrandChild()
g.getdata()
g.getdata1()
g.getdata2()
g.getdata3()
```

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

Ans. **super()** Function**:** Used to call methods of the parent class, especially in cases of multiple inheritance.

Ex.

```
class master:
    def signin(self,username,password):
        print("Username:",username)
        print("Password:",password)

class home(master):
    def signin(self,username,password):
        return super().signin(username,password)

class about(master):
    def signin(self,username,password):
        return super().signin(username,password)

a=about()
a.signin('vivek',1234)

h=home()
h.signin('jeet',9876)
```

8. Method Overloading and Overriding

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

Ans. Python does not support method overloading.

Ex. It's return error

```
class studinfo:
    def getdata(self,id,name):
        print("ID:",id)
        print("Name:",name)

    def getdata(self,sal):
        print("Salary:",sal)

st=studinfo()
st.getdata(101,'Sanket')
st.getdata(457.34)
```

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

Ans. Method overriding occurs when a child class provides a specific implementation for a method that is already defined in its parent class.

The overriding method in the child class must have the same name, parameters, and return type as the method in the parent class.

Ex.

```
class Parent:
    def getdata(self):
        print("Hello from parent's class")

class Child(Parent):
    def getdata(self):
        print("Hello from child's class")

c=Child()
c.getdata()
p=Parent()
p.getdata()
```

10. Search and Match Functions

Q.18 Using re.search() and re.match() functions in Python's re module for pattern matching.

Ans.

re.search(): Searches the entire string for the pattern and returns the first match.

Ex.

```
import re

mystr="This is Python!"
x=re.search('Python',mystr)
print(x)
if x:
   print("Match done!")
else:
   print("Error!")
```

re.match(): Checks for a match only at the beginning of the string.

Ex.

```
import re
mystr="This is Python!"
x=re.match('This',mystr)
print(x)
if x:
   print("Match done!")
else:
   print("Error!")
```

Q.19 Difference between search and match.

Ans.

search	match
Searches for the pattern anywhere in the string.	Matches the pattern only at the beginning of the string.
Returns a match object if the pattern is found anywhere; otherwise, returns none.	Returns a match object if the pattern matches at the beginning; otherwise, returns none.
Use when the pattern can appear anywhere in the sting.	Use when you need to ensure the string starts with the pattern
Slightly slower as it scans the entire string.	Faster as it checks only the beginning of the string.