1.Printing on screen

Q.1 introduction to the print() function in python.

Ans: It outputs strings, numbers, and other data types to the screen.

Ex:

```
Print(“HELLO  WORLD”)
```

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:

Ex:

```
id = int(input(“Enter your id: “))
name=input(“Enter your name: ”)
Print(“id: ”, id)
Print(“name: ”, name)
```

Q.4 converting user input into different data types

Ans:

```
id = int(input(“Enter your number: “))
print(f “This number is integer{number}”)
```

3.opening and closing files

Q.5 opening files in different modes (‘r’, ‘w’, ‘a’, 'r+', 'w+')

Ans:

Ex:

```
fl= open("info.txt", 'x')
fl = open("info.txt", 'w')
fl.write("Hello python!")
fl = open("info.txt" , 'r')
print(fl.read())
fl=open("info.txt",'a')
fl.write("\nThis programming language")
fl=open("info.txt", 'r+')
print(fl.write("\nThis is high level programming language"))
print(fl.read())
fl=open("info.txt", 'w+')
print(fl.write("\nThis is high level programming language"))
print(fl.read())
```

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

Ans:

Ex:

```
fl=open("info.txt", 'r')
print(fl.read())
```

Q.7 Closing files using close().

Ans:

Ex:

```
fl=open("info.txt", 'r')
print(fl.read())
fl.close()
```

<u>4.Reading and Writing files</u>

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

Ans:

Ex:

```
fl=open("info.txt","r")
print(fl.read())
print(fl.readline())
print(fl.readlines())
print(fl.readlines()[3])
```

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

Ans:

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())
```

<u>**5.Exception handling**</u>

Q.10 introduction to exception and how to handle them using try, except and finally

Ans:

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 exception:

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 exception:

Ex:

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

<u>6. Class and Object (OOP Concepts)</u>

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= "priyanshi"
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.

```
class stud_info:

    def info(self,id,name):
        print("id: ", id)
        print("name: ", name)

class data(stud_info):
    def info(self, id, name):
        return super().info(id, name)



dt=data ()
dt.info(7, "piyu")
```

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 math function

Q.18 using re.search() and re.match() functions in python’s re module for pattern matching

Ans:

re.search()

Ex:

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

re.match()

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.