NAME: **Meet Raut**
DIV: **S21**
ROLL NO: **2201084**

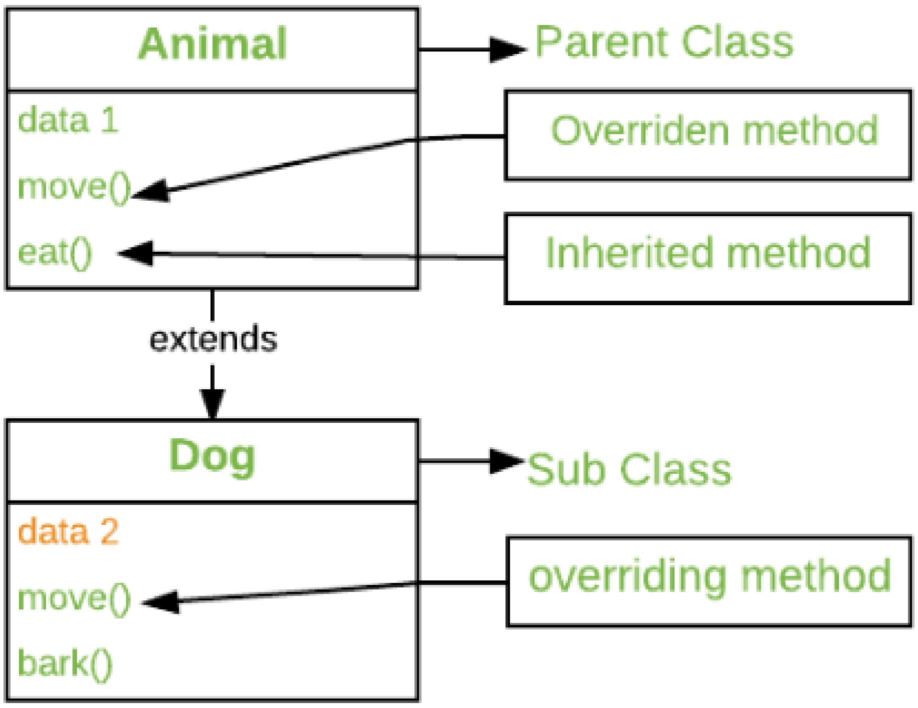
EXPERIMENT – 2:

- ***AIM:*** **To study and implement program on the concept of method overriding.**

- ***THEORY:***

Method Overriding in Python:

Any object-oriented programming language can allow a subclass or child class to offer a customized implementation of a method already supplied by one of its superclasses or parent classes. This capability is known as method overriding. The term "override" refers to a method in a subclass that replaces a method in a superclass when both methods share the same name, parameters, signature, and return type (or sub-type).



The object that calls a method will determine which version of the method is executed. When a method is called from an object of a parent class, the method's parent class version is executed; however, when a method is called from an object of a subclass, the child class version is executed. In other words, the version of an overridden method depends on the object being referenced, not the type of the reference variable.

- **Benefits of Method Overriding in Python:**

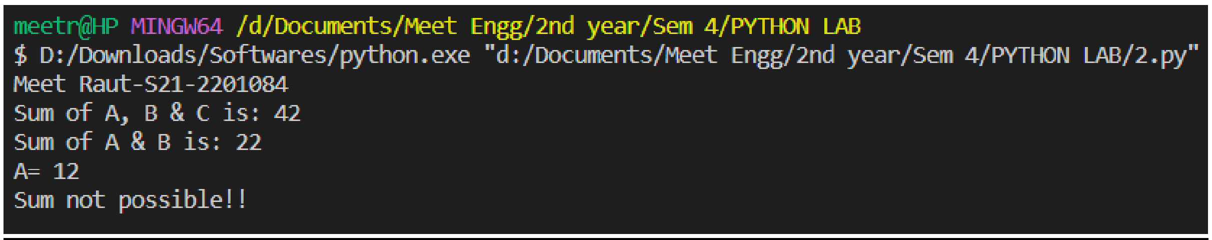
Both **overriding and overloading in Python** provide unique benefits. Let's go through the benefits of method overriding in python.

- Overriding methods permit a user to change the existing methods' behaviour. For its implementation, a minimum of two classes are needed.
- The key benefit is that it allows the main class to declare those methods shared by all. Also, it allows subclasses to define their implementations of any or all such methods.
- When overriding a method, you need to use inheritance.
- The parent class function and child class function must have the same signature. They must have the same number of parameters.
- It allows a child class to adapt the execution of any method provided by any one of its parent classes.

➢ PROGRAM 1:

```
print("Meet Raut-S21-2201084")

class A:
   def sum(self, A=None, B=None, C=None):
      if (A is not None and B is not None and C is not None):
         print("Sum of A, B & C is:", A+B+C)
      elif (A is not None and B is not None and C is None):
         print("Sum of A & B is:",A+B)
      elif (A is not None and B is None and C is None):
         print("A=",A)
      else:
         print("Sum not possible!!")

ob=A()
ob.sum(12,10,20)
ob.sum(12,10)
ob.sum(12)
ob.sum()
```

- **<u>OUTPUT:</u>**



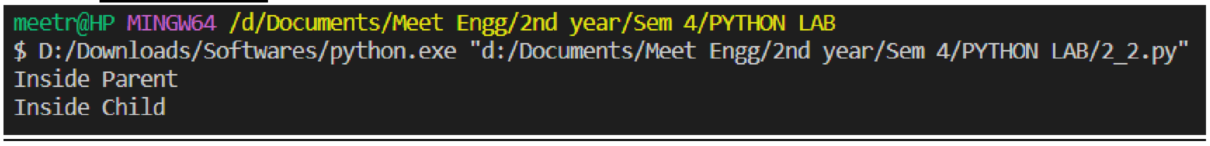
```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2.py"
Meet Raut-S21-2201084
Sum of A, B & C is: 42
Sum of A & B is: 22
A= 12
Sum not possible!!
```

➢ PROGRAM 2:

```
class Parent():

   def __init__(self):
      self.value = "Inside Parent"
```

```
    def show(self):
        print(self.value)

class Child(Parent):

    def __init__(self):
        self.value = "Inside Child"

    def show(self):
        print(self.value)

obj1 = Parent()
obj2 = Child()
obj1.show()
obj2.show()
```

- **OUTPUT:**

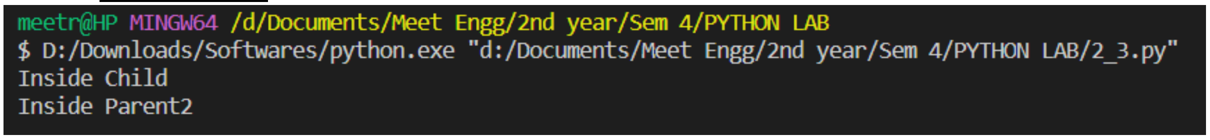


```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2_2.py"
Inside Parent
Inside Child
```

➢ **PROGRAM 3:**

```
class Parent1():

    def show(self):
        print ("Inside Parent1")

class Parent2():

    def display(self):
        print ("Inside Parent2")

class Child (Parent1, Parent2):

    def show(self):
        print ("Inside Child")
obj = Child ()
obj. show ()
obj. display ()
```

- **OUTPUT:**

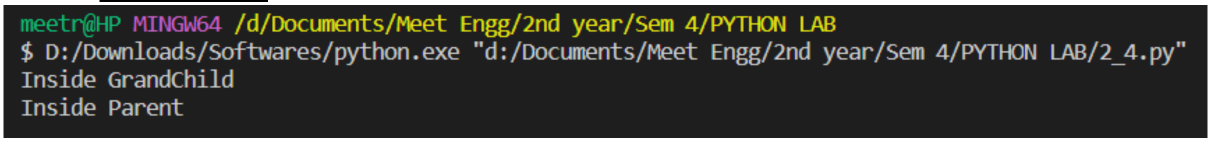


```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2_3.py"
Inside Child
Inside Parent2
```

➢ PROGRAM 4:

```
class Parent ():

   def display(self):
      print ("Inside Parent")

class Child (Parent):

   def show(self):
      print ("Inside Child")

class GrandChild (Child):

   def show(self):
      print ("Inside GrandChild")


g = GrandChild ()
g.show()
g.display()
```

- **OUTPUT:**



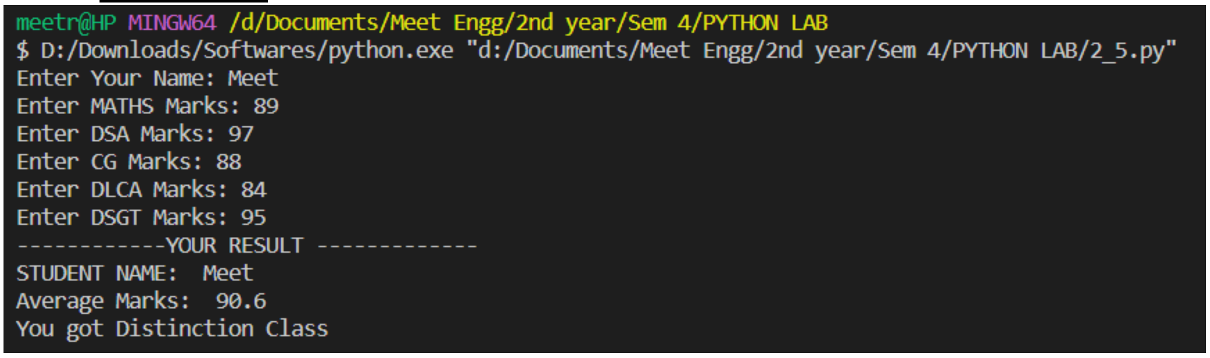
```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2_4.py"
Inside GrandChild
Inside Parent
```

➢ PROGRAM 5:

```
class student:
   def __init__ (self, name, mmarks=0, dmarks=0, cmarks=0, dlmarks=0,
dsmarks=0, amarks=0):
      self.name=input("Enter Your Name: ")
      self.mmarks=int(input("Enter MATHS Marks: "))
      self.dmarks=int(input("Enter DSA Marks: "))
```

```
        self.cmarks=int(input("Enter CG Marks: "))
        self.dlmarks=int(input("Enter DLCA Marks: "))
        self.dsmarks=int(input("Enter DSGT Marks: "))

self.amarks=float(self.mmarks+self.dmarks+self.cmarks+self.dlmarks+self.dsm
arks)/5

    def result_display (self):
        print("STUDENT NAME: ",self.name)
        print("Average Marks: ",self.amarks)

        if (self.amarks >= 80):
              print("You got Distinction Class")
        elif (self.amarks >= 60):
            print("You got First Class")
        elif (self.amarks >=40):
            print("You have Passes")
        else:
            print("You are Failed")
name=""
mmarks=0
dmarks=0
cmarks=0
dlmarks=0
dsmarks=0
amarks=0
ob=student(name, mmarks, dmarks, cmarks, dlmarks, dsmarks, amarks)
print("------------YOUR RESULT -------------")
ob.result_display()
```

- **OUTPUT:**



```
meetr@HP MINGW64 /d/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB
$ D:/Downloads/Softwares/python.exe "d:/Documents/Meet Engg/2nd year/Sem 4/PYTHON LAB/2_5.py"
Enter Your Name: Meet
Enter MATHS Marks: 89
Enter DSA Marks: 97
Enter CG Marks: 88
Enter DLCA Marks: 84
Enter DSGT Marks: 95
------------YOUR RESULT -------------
STUDENT NAME:  Meet
Average Marks:  90.6
You got Distinction Class
```

- ***CONCLUSION:*** Hence, we have successfully implemented program on the concept of method overriding.