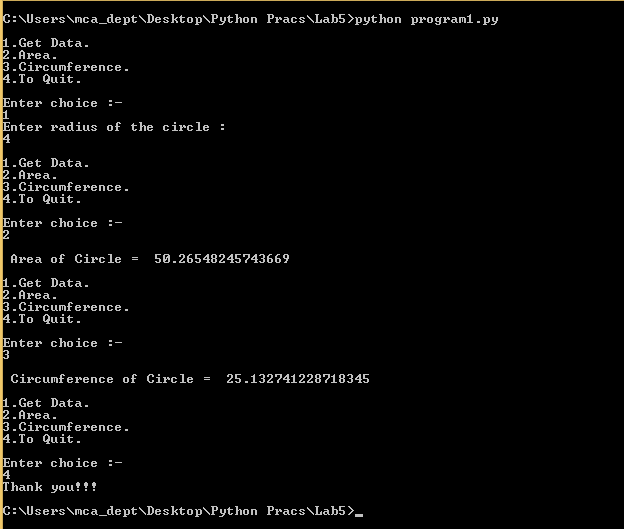
**Practical No. 05**

**1)Python Program to Create a Class and Compute the Area and the Perimeter of the Circle**

**Code:**

import math  
pi = math.pi  
class Circle:  
 r = 0  
 a = 0  
 c = 0  
 def \_\_init\_\_(self):  
 self.r = 0  
 self.a = 0  
 self.p = 0  
  
 def getdata(self):  
 self.r = float(input("Enter radius of the circle :\n"))  
  
 def area(self):  
 self.a = pi\*pow(self.r,2)  
 print("\n Area of Circle = ",self.a)  
   
 def circumference(self):  
 self.c = 2\*pi\*self.r  
 print("\n Circumference of Circle = ",self.c)  
   
obj = Circle()  
while True:  
 print("\n1.Get Data. \n2.Area. \n3.Circumference. \n4.To Quit.\n")  
 choice = int(input("Enter choice :-\n"))  
  
 if choice==1:  
 obj.getdata()  
 elif choice==2:  
 obj.area()  
 elif choice==3:  
 obj.circumference();  
 elif choice==4:  
 break  
 else:  
 print("Invalid choice, please choose again.\n")  
print("Thank you!!!")

**Screenshot**

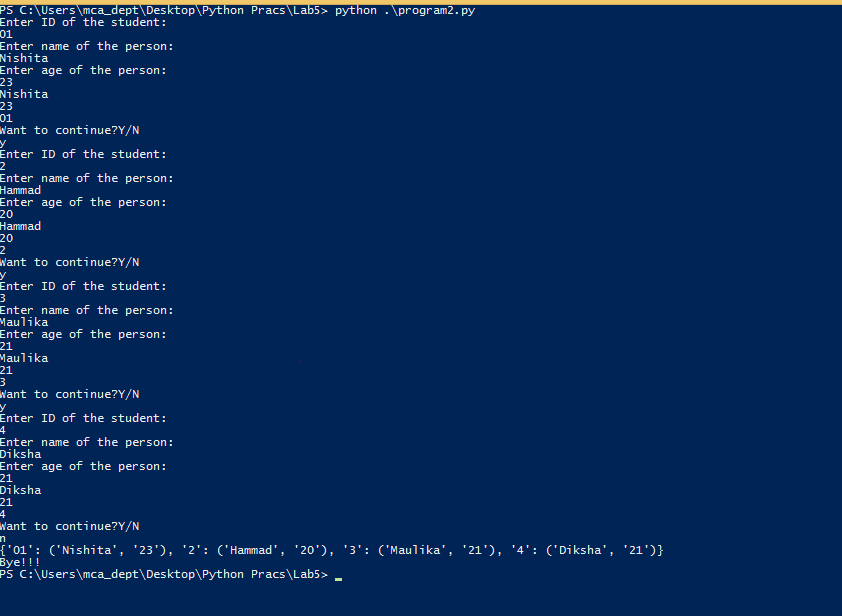
****

**2)To Implement Multiple Inheritance in python**

**Code:**

**class Person:  
 name = None  
 age = 0  
 psDict = {}  
 def setData(self,pName,pAge):  
 self.name = pName  
 self.age = pAge  
  
 def show(self):  
 print(self.name)  
 print(self.age)  
   
class Student:  
 id = 0  
 def setData(self, pId):  
 self.id = pId  
 def getId(self):  
 return self.id  
class Storage:  
 def store(self):  
 self.psDict[self.id] = self.name,self.age  
 def display(self):  
 print(self.psDict)  
  
class Resident(Person, Student, Storage):  
 def getData(self):  
 self.id = input("Enter ID of the student:\n")  
 self.name = input("Enter name of the person:\n")  
 self.age = input("Enter age of the person:\n")  
 def setData(self):  
 Person.setData(self, self.name, self.age)  
 Student.setData(self, self.id)  
  
  
n = 'Y'  
obj = Resident()  
while(True):  
 if n == 'Y':  
 obj.getData()  
 obj.setData()  
 obj.show()  
 print(obj.getId())  
 obj.store()  
 elif n == 'N':  
 break  
 n = input("Want to continue?Y/N\n")  
 n = n.upper()  
obj.display()  
print("Bye!!!")**

**screenshot:**



**3)To implement multilevel inheritance in python**

Code:

class Employee:

def getEmployee(self):

self.name = input("Name:")

self.age = input("Age:")

self.gender = input("Gender:")

class Designation(Employee):

def getDesignation(self):

self.Designation = input("Designation:")

class Salary(Designation):

def getSalary(self):

self.Salary = int(input("Salary:"))

print("\n Name:", self.name)

print("\n Age:",self.age)

print("\n Gender:", self.gender)

print("\n Designation:", self.Designation)

print("\n Salary:", self.Salary)

n = 'Y'

while(True):

if(n=='Y'):

S = Salary()

S.getEmployee()

S.getDesignation()

S.getSalary()

n=input("Do you want to continue?(Y/N)")

n=n.upper();

elif(n=='N'):

break;

Screenshot:

**4) To Implement a program with same method name and multiple arguments**

**Code:**

def add(dtype, \*args):

if dtype == 'int':

answer=0

if dtype == 'str':

answer=''

if dtype == 'float':

answer=0.0

for x in args:

answer = answer+x

print(answer)

#Integer

add('int',10,56,7)

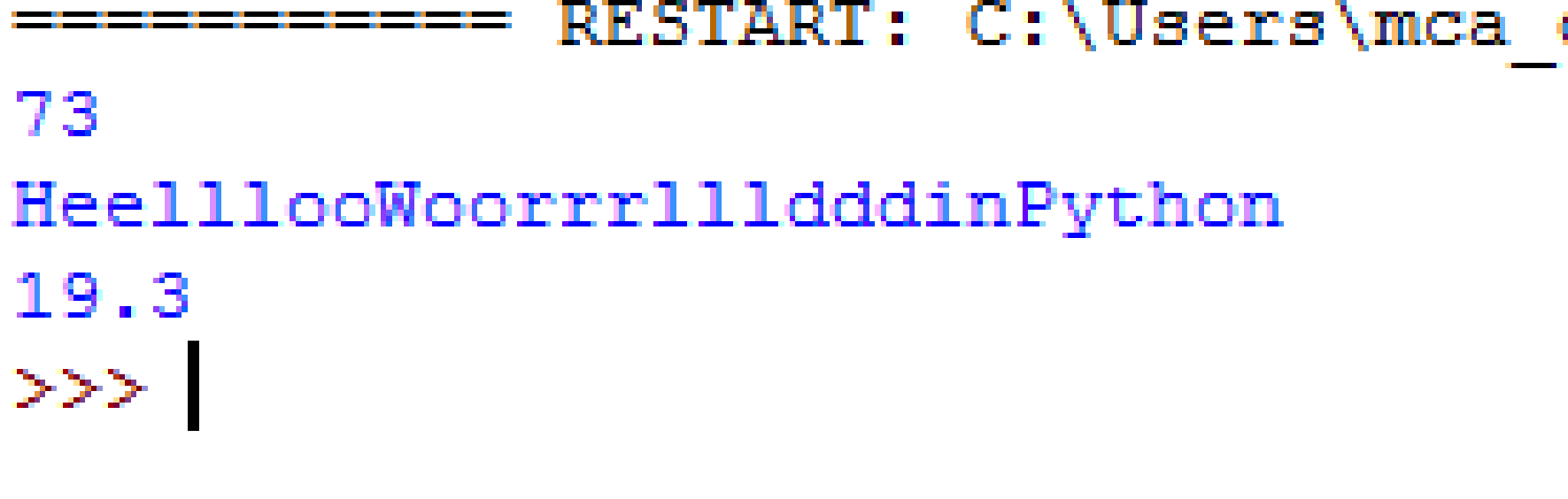
#String

add('str','Heellloo','Woorrrlllddd','in','Python')

#Float

add('float',6.3,9.5,3.5)

**Screenshot:**



**5)To Implement Operator Overloading in python**

**code:**

class complex:

def \_\_init\_\_(self, real, imag):

self.real = real

self.imag = imag

def \_\_add\_\_(self, other):

return complex(self.real + other.real, self.imag + other.imag)

def \_\_sub\_\_(self, other):

return complex(self.real - other.real, self.imag - other.imag)

def \_\_mul\_\_(self, other):

return complex(self.real \* other.real, self.imag \* other.imag)

def \_\_truediv\_\_(self, other):

return complex(self.real / other.real, self.imag / other.imag)

def \_\_floordiv\_\_(self, other):

return complex(self.real // other.real, self.imag // other.imag)

def \_\_mod\_\_(self, other):

return complex(self.real - other.real, self.imag - other.imag)

def \_\_lt\_\_(self, other):

return self.real < other.real, self.imag < other.imag

def \_\_gt\_\_(self, other):

return self.real > other.real, self.imag > other.imag

def \_\_le\_\_(self, other):

return self.real <= other.real, self.imag <= other.imag

def \_\_ge\_\_(self, other):

return self.real >= other.real, self.imag >= other.imag

def \_\_eq\_\_(self, other):

return self.real == other.real, self.imag == other.imag

def \_\_ne\_\_(self, other):

return self.real != other.real, self.imag != other.imag

def \_\_pow\_\_(self, other):

return self.real \*\* other.real, self.imag \*\* other.imag

a1=int(input("Enter the Real part: "))

b1=int(input("Enter the Imaginary part: "))

O1 = complex(a1, b1)

print("First Number is: ",a1,"+",b1,"i")

a2=int(input("Enter the Real part: "))

b2=int(input("Enter the Imaginary part: "))

O2 = complex(a2, b2)

print("Second Number is: ",a2,"+",b2,"i")

O3 = O1 + O2

print("Addition is",O3.real,"+",O3.imag,"i")

O3 = O1 - O2

print("Subtraction is",O3.real,"+",O3.imag,"i")

O3 = O1 \* O2

print("Multipliaction is",O3.real,"+",O3.imag,"i")

O3 = O1 / O2

print("TrueDiv is",O3.real,"+",O3.imag,"i")

O3 = O1 // O2

print("FloorDiv is",O3.real,"+",O3.imag,"i")

O3 = O1 % O2

print("Modulo is",O3.real,"+",O3.imag,"i")

print("Is O1<O2",O1<O2)

print("Is O1<=O2",O1<=O2)

print("Is O1>O2",O1>O2)

print("Is O1>=O2",O1>=O2)

print("Is O1==O2",O1==O2)

print("Is O1!=O2",O1==O2)

print("Is O1\*\*O2",O1\*\*O2)

**Screenshot:**

