**PRACTICAL 9**

**AIM :Write a program to implement concepts of object oriented Programming such as classes, inheritance and polymorphism.**

**Class:**

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

Compared with other programming languages, Python’s class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.

**Objects:**

Objects have individuality, and multiple names (in multiple scopes) can be bound to the same object. This is known as aliasing in other languages. This is usually not appreciated on a first glance at Python, and can be safely ignored when dealing with immutable basic types (numbers, strings, tuples). However, aliasing has a possibly surprising effect on the semantics of Python code involving mutable objects such as lists, dictionaries, and most other types. This is usually used to the benefit of the program, since aliases behave like pointers in some respects.

**Inheritance**:

Inheritance is the capability of one class to derive or inherit the properties from some another class. The benefits of inheritance are:

1. It represents real-world relationships well.
2. It provides reusability of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

**OOPs support the six different types of inheritance as given below :**

* Single inheritance.
* Multi-level inheritance.
* Multiple inheritance.
* Hierarchical Inheritance.
* Hybrid Inheritance.

**Polymorphism:**

 The word polymorphism means having many forms. In programming, polymorphism means same function name (but different signatures) being uses for different types.

**#PROGRAM:**

**#class**

class a:

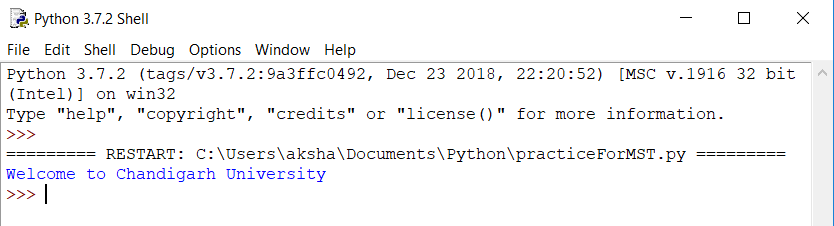
def root(self):

print("Welcome to Chandigarh University");

obj=a()

obj.root();

**output:**

****

**#inheritance**

**#single**

class abc:

def sum(self):

self.a=10;

self.b=20;

result=self.a+self.b

print(result);

class xyz(abc):

def mul(self):

self.p=10;

self.q=12;

result=self.p\*self.q

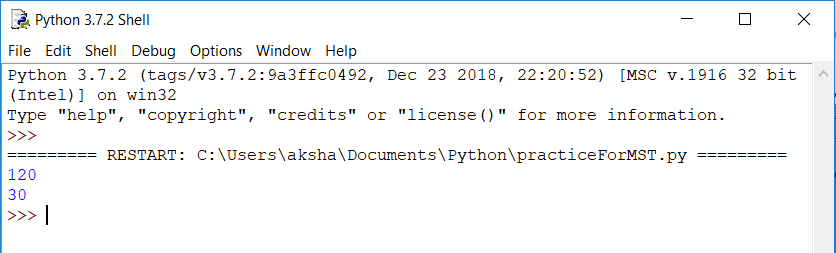
print(result);

obj1=xyz()

obj1.mul()

obj1.sum()

**output:**



**#single with user input**

class abc:

def sum(self):

self.a=int(input('Enter a: '))

self.b=int(input('Enter b: '))

result=self.a+self.b

print(result);

class xyz(abc):

def mul(self):

self.p=int(input('Enter p: '))

self.q=int(input('Enter q: '))

result=self.p\*self.q

print(result);

obj1=xyz()

obj1.mul()

obj1.sum()

**output:**

A screenshot of a social media post

Description automatically generated

**#Multiple**

class abc:

def sum(self):

self.a=10;

self.b=20;

result=self.a+self.b

print(result);

class xyz:

def mul(self):

self.p=10;

self.q=12;

result=self.p\*self.q

print(result);

class r(abc,xyz):

def display(self):

print("Hello");

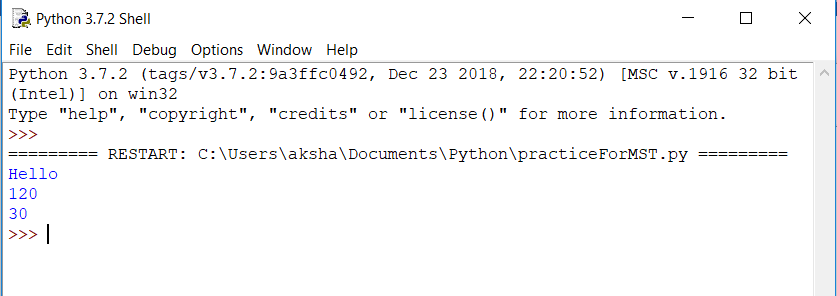
obj1=r()

obj1.display()

obj1.mul()

obj1.sum()

**output:**



**#Multilevel**

class abc:

def sum(self):

self.a=10;

self.b=20;

result=self.a+self.b

print(result);

class xyz(abc):

def mul(self):

self.p=10;

self.q=12;

result=self.p\*self.q

print(result);

class r(xyz):

def display(self):

print("Hello");

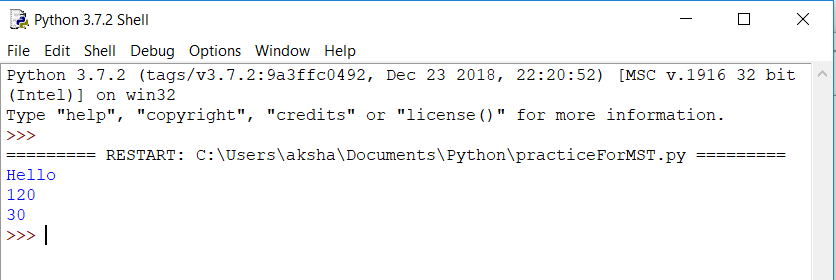
obj1=r()

obj1.display()

obj1.mul()

obj1.sum()

**output:**



**#Hierarchical**

class abc:

def sum(self):

self.a=10;

self.b=20;

result=self.a+self.b

print(result);

class xyz(abc):

def mul(self):

self.p=10;

self.q=12;

result=self.p\*self.q

print(result);

class r(abc):

def display(self):

print("Hello");

obj1=r()

obj2=xyz()

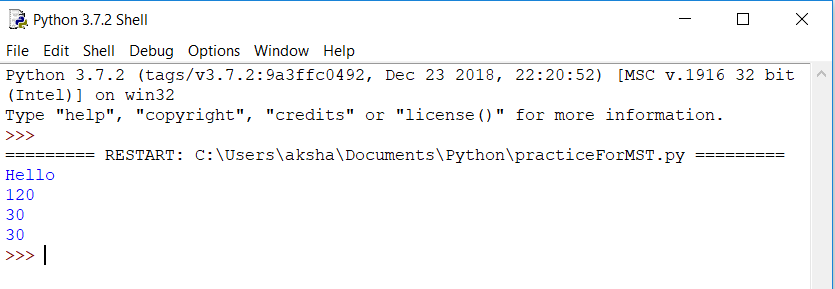
obj1.display()

obj2.mul()

obj1.sum()

obj2.sum()

**output:**



**#Hybrid**

class abc:

def sum(self):

self.a=10;

self.b=20;

result=self.a+self.b

print(result);

class xyz:

def mul(self):

self.p=10;

self.q=12;

result=self.p\*self.q

print(result);

class r(abc,xyz):

def display(self):

print("Hello");

class d(r):

def tool(self):

print("Hybrid");

obj=d()

obj.tool()

obj1=r()

obj1.display()

obj1.mul()

obj1.sum()

**output:**

