python

Course Id :INT 213

Inheritance

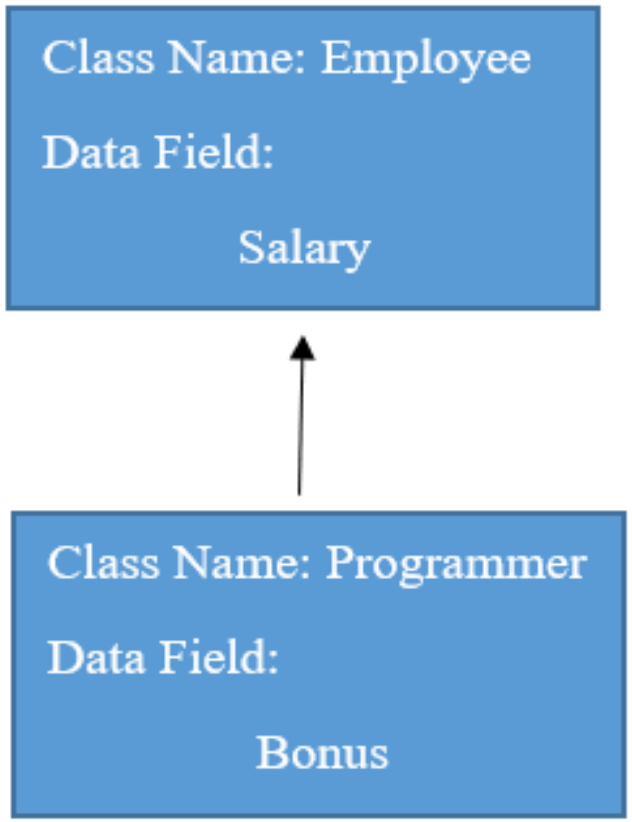
- Creating new classes from existing classes, so that the new classes will acquire all the features of the existing classes is called Inheritance. A good example for Inheritance in nature is parents producing the children and children inheriting the qualities of the parents.

Inheritance

- Let's take a class A with some members i.e., variables and methods. If we feel another class B wants almost same members, then we can derive or create class B from A as: class B(A):
- Now, all the features of A are available to B. If an object to B is created, it contains all the members of class A and also its own members. Thus, the programmer can access and use all the members of both the classes A and B. Thus, class B becomes more useful. This is called inheritance.
- The original class (A) is called the base class or super class and the derived class (B) is called the sub class or derived class.

Inheritance

Inheritance represents the IS – A relationship also known as parent-child relationship.



IS-A Relationship

Advantages of inheritance

- There are three advantages of inheritance.

- First, we can create more useful classes needed by the application (software).
- Next, the process of creating the new classes is very easy, since they are built upon already existing classes.
- The last, but very important advantage is managing the code becomes easy, since the programmer creates several classes in a hierarchical manner, and segregates the code into several modules.

Inheritance

i.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/i.py (3.7.4)

File Edit Format Run Options Window Help

```
class Animal:
    def speak(self):
        print("Animal Speaking")
#child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
d = Dog()
d.bark()
d.speak()
```

Constructors in Inheritance

constr_in_inher.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/constr_in_inher.py (3.7.4)

File Edit Format Run Options Window Help

```
class Father:
    def __init__(self):
        self.property = 800000.00
    def display_property(self):
        print('Father\'s property=', self.property)
class Son(Father):
    pass                                    #we do not want to write anything in the sub class

s = Son()                                   #create sub class instance and display father's property
s.display_property()
```

Overriding Super Class Constructors and Methods

- When the programmer writes a constructor in the sub class, the super class constructor is not available to the sub class.
- In this case, only the sub class constructor is accessible from the sub class object.
- That means the sub class constructor is replacing the super class constructor. This is called constructor overriding.
- Similarly in the sub class, if we write a method with exactly same name as that of super class method, it will override the super class method. This is called method overriding.

Overriding Super Class Constructors and Methods

over.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/over.py (3.7.4)

File Edit Format Run Options Window Help

```
class Father:
    def __init__(self):
        self.property = 800000.00

    def display_property(self):
        print('Father\'s property=', self.property)

class Son(Father):
    def __init__(self):
        self.property = 200000.00

    def display_property(self):
        print('Child\'s property=', self.property)

#create sub class instance and display father's property
s = Son()
s.display_property()
```

- In this Program, in the sub class, we created a constructor and a method with exactly same names as those of super class. When we refer to them, only the sub class constructor and method are executed. The base class constructor and method are not available to the sub class object. That means they are overridden.

Overriding Super Class Constructors and Methods

- Overriding should be done when the programmer wants to modify the existing behavior of a constructor or method in his sub class.
- In this case, how to call the super class constructor so that we can access the father's property from the Son class?
- For this purpose, we should call the constructor of the super class from the constructor of the sub class using the super() method.

The super() Keyword

- super() is a built-in method which is useful to call the super class constructor or methods from the sub class.
- Any constructor written in the super class is not available to the sub class if the sub class has a constructor. Then how can we initialize the super class instance variables and use them in the sub class?
- This is done by calling the super class constructor using the super() method from inside the sub class constructor.
- super() is a built-in method in Python that contains the history of super class methods. Hence, we can use super() to refer to super class constructor and methods from a sub class.

The super() Keyword

super() can be used as:

```
super().__init__()     # call super class constructor
super().__init__(arguments)# call super class constructor and pass
                           # arguments
super().method()   # call super class method
```

When there is a constructor with parameters in the super class, we have to create another constructor with parameters in the sub class and call the super class constructor using super() from the sub class constructor. In the following example, we are calling the super class constructor and passing 'property' value to it from the sub class constructor.

```
# this is sub class constructor
def __init__(self, property1=0, property=0):
    super().__init__(property)  # send property value to superclass
                                # constructor
    self.property1= property1   # store property1 value into subclass
                                # variable
```

A Python program to call the super class constructor in the sub class using super().

sup.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/sup.py (3.7.4)

File Edit Format Run Options Window Help

```
class Father:
    def __init__(self,property=0):
        self.property = property

    def display_property(self):
        print('Father\'s property=', self.property)

class Son(Father):

    def __init__(self, property1=0, property=0):
        super().__init__(property)

        self.property1= property1
    def display_property(self):

        print('Total property of child=', self.property1 + self.property)

 #create sub class instance and display father's property
s = Son(200000.00, 800000.00)
s.display_property()
```

A Python program to access base class constructor and method in the sub class using super().

sup1.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/sup1.py (3.7.4)

File Edit Format Run Options Window Help

```
class Square:
    def __init__(self, x):
        self.x = x
    def area(self):
        print('Area of square=', self.x*self.x)

class Rectangle(Square):
    def __init__(self, x, y):
        super().__init__(x)
        self.y = y
    def area(self):
        super().area()
        print('Area of rectangle=', self.x*self.y)

#find areas of square and rectangle
a, b = [float(x) for x in input("Enter two measurements: ").split()]
r = Rectangle(a,b)
r.area()
```

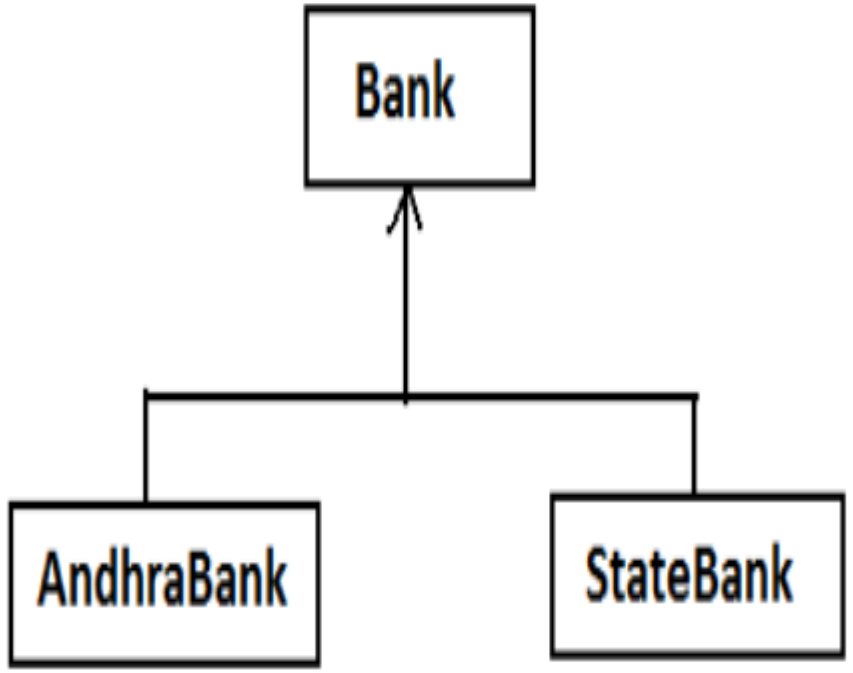
Types of Inheritance

- Inheritance, there are mainly two types of inheritance available.
- They are:
  - Single inheritance
  - Multiple inheritance

Single Inheritance

- Deriving one or more sub classes from a single base class is called ‘single inheritance’.
- **In single inheritance**, we always have only one base class, but there can be n number of sub classes derived from it.
- For example, ‘Bank’ is a single base class from where we derive ‘AndhraBank’ and ‘StateBank’ as sub classes. This is called single inheritance.

.



A Python program showing single inheritance in which two sub classes are derived from a single base class.

single_inhert.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/single_inhert.py (3.7.4)

File Edit Format Run Options Window Help

```
#single inheritance
class Bank(object):
    cash = 100000000
    def available_cash(cls):
        print(cls.cash)

class AndhraBank(Bank):
    pass

class StateBank(Bank):
    cash = 20000000
    def available_cash(cls):
        print(cls.cash + Bank.cash)

a = AndhraBank()
a.available_cash()
s = StateBank()
s.available_cash()
```

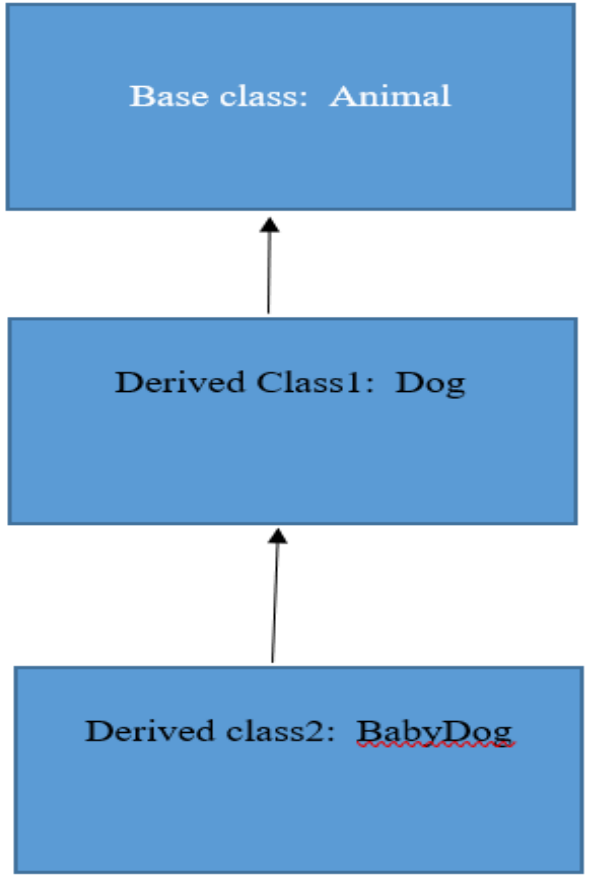
Single Inheritance

```
class person:
    def __init__(self,name,age):

        self.name=name
        self.age=age

    def displayp(self):
        print("Name:",self.name)
        print("Age:",self.age)

class teacher(person):
    def __init__(self,name,age,exp,research_area):

        person.__init__(self,name,age)

        self.exp=exp
        self.research_area=research_area

    def displayt(self):
        person.displayp(self)
        print("Experience:",self.exp,"years")
        print("Research Area:",self.research_area)
```

```
class student(person):
    def __init__(self,name,age,course,marks):
        person.__init__(self,name,age)

        self.course=course
        self.marks=marks


    def displays(self):
        person.displayp(self)
        print("course:",self.course)
        print("marks:",self.marks)

t= teacher("Rahul",43,5,"image processing")
t.displayt()
s= student("Vikas",20,"btech",90)
s.displays()
```

Multilevel Inheritance

Multilevel Inheritance – In multilevel inheritance, you can inherit a derived class from another derived class.



Multilevel Inheritance

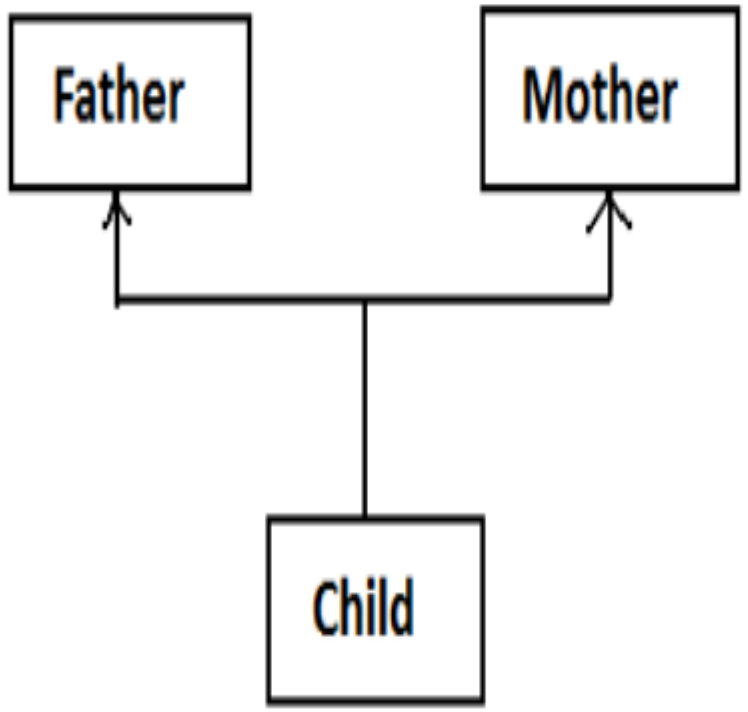
Multiple Inheritance

- Deriving sub classes from multiple (or more than one) base classes is called ‘multiple inheritance’.
- In this type of inheritance, there will be more than one super class and there may be one or more sub classes. All the members of the super classes are by default available to sub classes and the sub classes in turn can have their own members.
- The syntax for multiple inheritance is shown in the following statement:

```
class Subclass(Baseclass1, Baseclass2, …):
```

Multiple Inheritance

- The best example for multiple inheritance is that parents producing the children and the children inheriting the qualities of the parents. Suppose, Father and Mother are two base classes and Child is the sub class derived from these two base classes. Now, whatever the members are found in the base classes are available to the sub class.



A Python program to implement multiple inheritance using two base classes.

multiple_inher.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/multiple_inher.py (3.7.4)

File Edit Format Run Options Window Help

```
class Father:
    def height(self):
        print('Height is 6.0 foot')

class Mother:
    def color(self):
        print('Color is brown')

class Child(Father, Mother):
    pass

c = Child()
print('Child\'s inherited qualities:')
c.height()
c.color()
```

- For example, the Father class has a method that displays his height as 6.0 foot and the Mother class has a method that displays her color as brown.
- To make the Child class acquire both these qualities, we have to make it a sub class for both the Father and Mother class.

Problems in Multiple Inheritance

If the sub class has a constructor, it overrides the super class constructor and hence the super class constructor is not available to the sub class. But writing constructor is very common to initialize the instance variables. In multiple inheritance, let's assume that a sub class 'C' is derived from two super classes 'A' and 'B' having their own constructors. Even the sub class 'C' also has its constructor.

To derive C from A and B, we write: class C(A, B): Also, in class C's constructor, we call the super class super class constructor using super().__init__().

Now, if we create an object of class C, first the class C constructor is called. Then super().__init__() will call the class A's constructor.

A Python program to prove that only one class constructor is available to sub class in multiple inheritance.

multiple.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/multiple.py (3.7.4)

File Edit Format Run Options Window Help

```
class A(object):
    def __init__(self):
        self.a = 'a'
        print(self.a)

class B(object):
    def __init__(self):
        self.b = 'b'
        print(self.b)

class C(A, B):
    def __init__(self):
        self.c = 'c'
        print(self.c)
        super().__init__()  #access the super class instance vars from C
o = C() #o is object of class C
```

- The output of the program indicates that when class C object is created the C's constructor is called. In class C, we used the statement: super().__init__() that calls the class A's constructor only.
- Hence, we can access only class A's instance variables and not that of class B. In this program, we created sub class C,

as: class C(A, B):

A Python program to access all the instance variables of both the base classes in multiple inheritance.

multiple1.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/multiple1.py (3.7.4)

File Edit Format Run Options Window Help

```
class A(object):
    def __init__(self):
        self.a = 'a'
        print(self.a)
        super().__init__()

class B(object):
    def __init__(self):
        self.b = 'b'
        print(self.b)
        super().__init__()

class C(A, B):
    def __init__(self):
        self.c = 'c'
        print(self.c)
        super().__init__()                    #access the super class instance vars from C


o = C()                                       #o is object of class C
```

What is the output of the code?

untitled

File Edit Format Run Options Window Help

```
class Animal:
    def speak(self):
        print("Animal Speaking")


class Dog(Animal):
    def bark(self):
        print("dog barking")

class DogChild(Dog):
    def eat(self):
        print("Eating bread...")
d = DogChild()
d.bark()
d.speak()
d.eat()
```

What is the output of the code?

aq.py - C:/Users/Dipen/AppData/Local/Programs/Python/Python37-32/aq.py (3.7.4)

File Edit Format Run Options Window Help

```
class Calculation1:
    def Summation(self,a,b):
        return a+b
class Calculation2:
    def Multiplication(self,a,b):
        return a*b
class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
        return a/b
d = Derived()
print(d.Summation(10,20))
print(d.Multiplication(10,20))
print(d.Divide(10,20))
```

Analyze the code given below to find the reason behind the error in the program.

```
class A:
    def __init__(self):
        self.P = 10
        self.__Q = 20
    def getY(self):
        return self.__Q
a = A()
print(a.__Q)
```

a. Q is private and cannot be access outside of the class.

b. P is private and cannot be access outside of the class.

c. Both a and b

d. None of the above

What is the output of the code?

s12d.py - C:/Users/a/AppData/Local/Programs/Python/Python38-32/s12d.py (3.8.3)

File Edit Format Run Options Window Help

```
class A:
    def __init__(self, i = 0):
        self.i = i
    def m1(self):
        self.i += 1
class B(A):
    def __init__(self, j = 0):
        super().__init__(3)
        self.j = j
    def m1(self):
        self.i += 1
b = B()
b.m1()
print(b.i)
print(b.j)
```

(a) 0 0

(b) 4 0

(c) 1 3

(d) 0 4