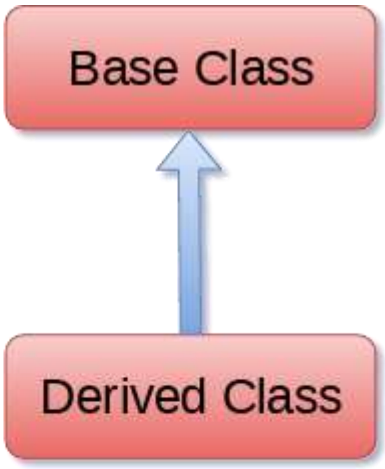
Python Inheritance

Inheritance is an important aspect of the object-oriented paradigm. Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch.

In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class

Single Inheritance



Syntax

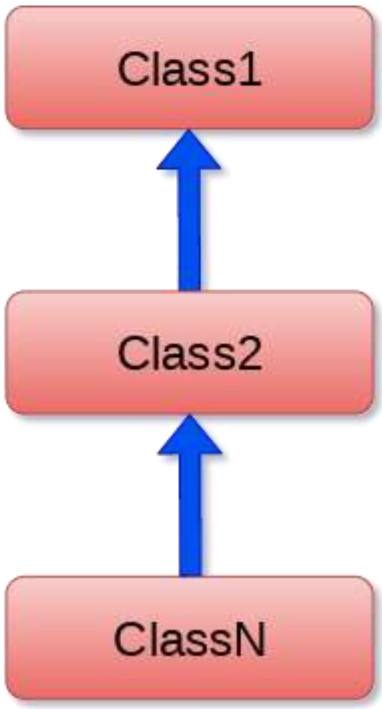
```
class derived-class(base class):
    <class-suite>
```

Example 1

```
class A:
    def show(self):
        print("This is class A")
classB(A):
    def disp(self):
        print("dog barking")
b = B()
b.disp()
b.show()
```

Python Multi-Level inheritance

Multi-Level inheritance is possible in python like other object-oriented languages. Multi-level inheritance is archived when a derived class inherits another derived class. There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.



Syntax

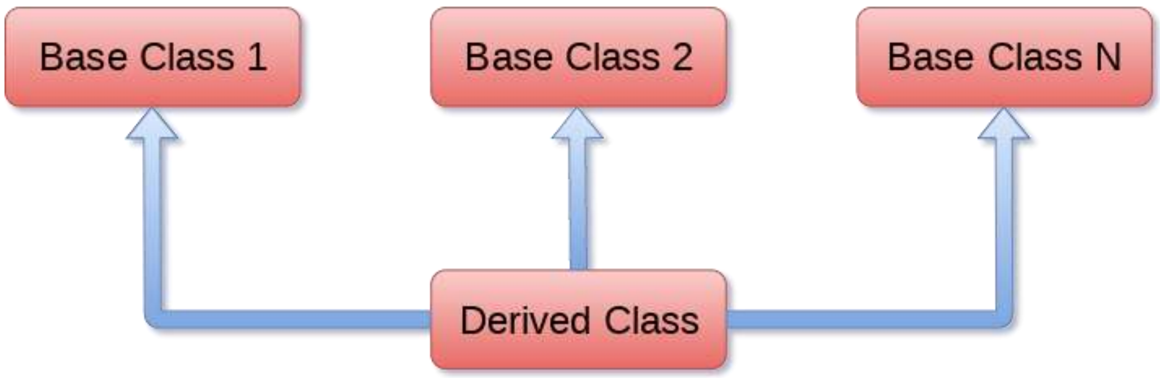
```
class class1:
   <class-suite>
class class2(class1):
   <class suite>
class class3(class2):
   <class suite>
```

Example

```
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()
```

Python Multiple inheritance

Python provides us the flexibility to inherit multiple base classes in the child class.



Syntax

```
class Base1:
    <class-suite>

class Base2:
    <class-suite>
.
.
.
class BaseN:
    <class-suite>

class Derived(Base1, Base2, ...... BaseN):
    <class-suite>
```

Example

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

Method Overriding

We can provide some specific implementation of the parent class method in our child class. When the parent class method is defined in the child class with some specific implementation, then the concept is called method overriding. We may need to perform method overriding in the scenario where the different definition of a parent class method is needed in the child class.

Example

```
class Animal:
    def speak(self):
        print("speaking")
class Dog(Animal):
    def speak(self):
        print("Barking")
d = Dog()
d.speak()
```

Real Life Example of method overriding

```
class Bank:
    def getroi(self):
        return 10;
class SBI(Bank):
    def getroi(self):
        return 7;

class ICICI(Bank):
    def getroi(self):
        return 8;
b1 = Bank()
b2 = SBI()
b3 = ICICI()
print("Bank Rate of interest:",b1.getroi());
print("SBI Rate of interest:",b2.getroi());
print("ICICI Rate of interest:",b3.getroi());
```

Data abstraction in python

Abstraction is an important aspect of object-oriented programming. In python, we can also perform data hiding by adding the double underscore (___) as a prefix to the attribute which is to be hidden. After this, the attribute will not be visible outside of the class through the object.

Example

```
class Employee:
    __count = 0;
    def __init__(self):
        Employee.__count = Employee.__count+1
    def display(self):
        print("The number of employees",Employee.__count)
emp = Employee()
emp2 = Employee()
try:
    print(emp.__count)
finally:
    emp.display()
```

S