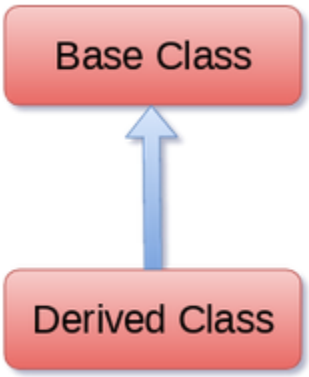
Python Inheritance

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.

In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the following syntax to inherit a base class into the derived class.



Syntax:

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

A class can inherit multiple classes by mentioning all of them inside the bracket. Consider the following syntax.

Syntax:

```
class                                                derive-
class(<base class 1>, <base class 2>, ..... <base class n

   <class - suite>
```

For Example:

```
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()
```

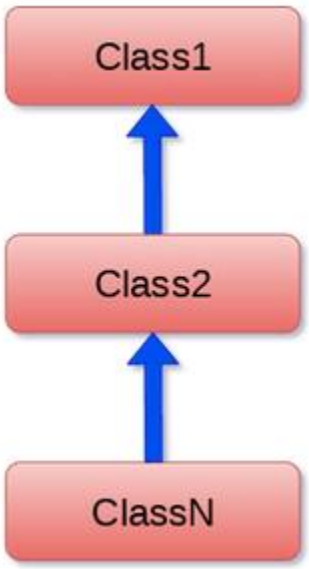
Output:

```
dog barking
Animal Speaking
```

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.



The syntax of multi-level inheritance is given below.

Syntax:

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

For Example:

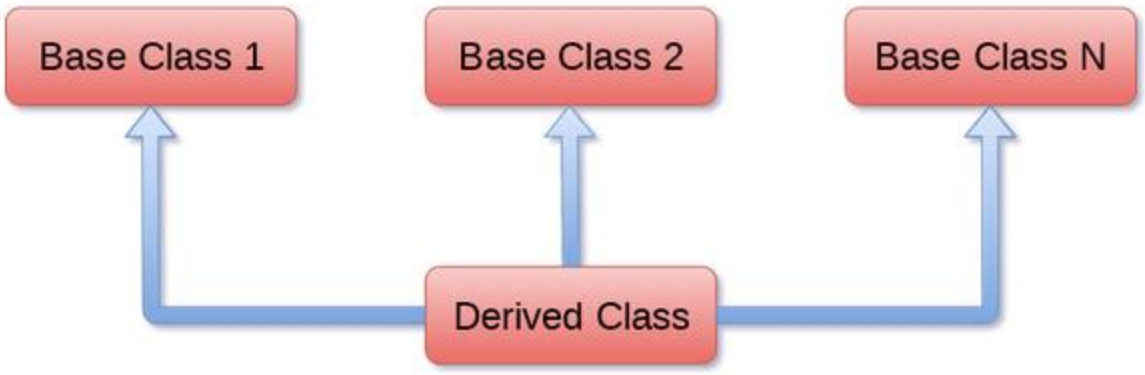
```python
class Animal:
    def speak(self):
        print("Animal Speaking")
    # The child class Dog inherits the base class Animal


class Dog(Animal):
    def bark(self):
        print("dog barking")
    # The child class Dogchild inherits another child class Dog


class DogChild(Dog):
    def eat(self):
        print("Eating bread...")


d = DogChild()
d.bark()
d.speak()
d.eat()
```

Output:

```
dog barking
Animal Speaking
Eating bread...
```

Python Multiple inheritance:

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



The syntax to perform multiple inheritance is given below.

Syntax:

```
class Base1:
   <class-suite>

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

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

For Example:

```
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(5, 10))
print(d.Multiplication(5, 10))
print(d.Divide(5, 10))
```

Output:

```
15
50
0.5
```

The issubclass(sup, sup) method:

The issubclass(sub, sup) method is used to check the relationships between the specified classes. It returns true if the first class is the subclass of the second class, and false otherwise.

Consider the following example.

For Example:

```
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(issubclass(Derived,Calculation2))
print(issubclass(Calculation1,Calculation2))
```

Output:

```
True
False
```

The isinstance (obj, class) method:

The isinstance() method is used to check the relationship between the objects and classes. It returns true if the first parameter, i.e., obj is the instance of the second parameter, i.e., class.

Consider the following example.

For Example:

```
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(isinstance(d, Derived))
```

Output:

```
True
```

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.

Consider the following example to perform method overriding in python.

For Example:

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


class Dog(Animal):
    def speak(self):
        print("Barking")


d = Dog()
d.speak()
```

Output:

```
Barking
```

Another Example:

```
class Bank:
    def getroi(self):
        return 10


class SBI(Bank):
    def getroi(self):
        return 7


class GlobalIME(Bank):
    def getroi(self):
        return 8


b1 = Bank()
b2 = SBI()
b3 = GlobalIME()
print("Bank Rate of interest:", b1.getroi())
print("SBI Rate of interest:", b2.getroi())
print("GlobalIME Rate of interest:", b3.getroi())
```

Output:

```
Bank Rate of interest: 10
SBI Rate of interest: 7
GlobalIME Rate of interest: 8
```

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.

Consider the following example.

For 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()
```

Output:

```
The number of employees 2
Traceback (most recent call last):
  File "/home/nirajan/PycharmProjects/PythonExamples/Dataabstraction.py", line 14, in <module>
    print(emp.__count)
AttributeError: 'Employee' object has no attribute '__count'
```