Inheritance in Python

**Inheritance** is one of the most important aspects of Object Oriented Programming. While programming, many a times, situations arise where we have to write a few classes with some common features and some unique, class-specific features, which include both variables and methods.

In such situations, as per object oriented programming, we can take out the common part and put it in a separate class, and make all the other classes inherit this class, to use its methods and variables, hence reducing re-writing the common features in every class, again and again.

The class which inherits another class is generally known as the **Child class**, while the class which is inherited by other classes is called as the **Parent class**.

Ofcourse, you must only use this for the related classes only, for example, you can define a class **LivingOrganism** with all the basic features of a living organism defined in it, like breathe, eat etc. Now this class can easily be re-used by another class **Animals** and **HumanBeings**, as both of these shares the features.

Also, at times, Inheritance is used to simplify big classes with a lot of variables and methods, into smaller classes by breaking down the functionality into core features and secondary features. The core features are generally kept in the parent class.

## Syntax for Inheritance

If we have a class Parent and another class Child and we want the class Child to inherit the class Parent, then

# Parent class

class Parent:

# class variable

a = 10;

b = 100;

# some class methods

def doThis()

def doThat()

# Child class inheriting Parent class

class Child(Parent):

# child class variable

x = 1000;

y = -1;

# some child class method

def doWhat()

def doNotDoThat()

By specifying another class's name in parentheses, while declaring a class, we can specify inheritance. In the example above, all the properties of Parent will be inherited to the Child class. With this, all the methods and variables defined in the class Parent becomes part of Child class too.

Time for an Example

Let's take simple example. Animals can be divided into multiple types like reptiles, mammals, amphibians etc. While they all are different physically and biologically, there are many characteristics that are common amongst them. So now, we will define a parent class with name **Animal**, which will have some basic properties and functions related to animals.

And then we will define classes for various other types, and all those classes will also inherit the class **Animal**.

Here we have the Animal class.

class Animal:

# properties

multicellular = True

# Eukaryotic means Cells with Nucleus

eukaryotic = True

# functions

def breath()

def feed()

Now let's define a class for **Mammals**. As mammals are animals with warm blood, who produce milk for their infants etc, hence our Mammal class will look like,

class Mammal(Animal):

# properties

haveMammaryGland = True

def warmBlood = True

# functions

def produceMilk()

In case we want to create another class for Amphibians too, then

class Amphibian(Animal):

# properties

liveInWater = True

# functions

def metamorphosis()

As the classes Mammals and Amphibian both inherit the class Animal, hence they will have the properties and functions defined in the class Animal.

Hence, now any object of Amphibian class, say **Frog**, will have the properties: multicellular(from class Animal), eukaryotic(from class Animal), liveInWater(from class Amphibian), and would be able to breath(), feed() and do metamorphosis().

And how all this can be written as code, if we create an object, say **Frog**:

>>> Amphibian Frog = Amphibian()

>>> Frog.breath() # calling function defined in Animal class

>>> Frog.metamorphosis() # calling function defined in Amphibian class

>>> print Frog.liveInWater

True

**Benefits of using Inheritance**

Here are a few main advantages of using Inheritance in your programs.

1. Less code repeatition, as the code which is common can be placed in the parent class, hence making it available to all the child classes.
2. **Structured Code**: By dividing the code into classes, we can structure our software better by dividing functionality into classes.
3. Make the code more scalable.

Accessing Parent Class Element in Child Class

While working in a child class, at some point you may have to use parent class's properties or functions. In order to access parent class's elements you can use the dot . operator.

Parent.variableName

Mentioned above is how you can access the variable, or in case you need to call parent class's function then,

Parent.functionName()

Where Parent is the name of our parent class, and variableName and functionName() are its variable and function respectively.

Below is an example, we have a simple example to demonstrate this:

class Parent:

var1 = 1

def func1(self):

# do something here

class Child(Parent):

var2 = 2

def func2(self):

# do something here too

# time to use var1 from 'Parent'

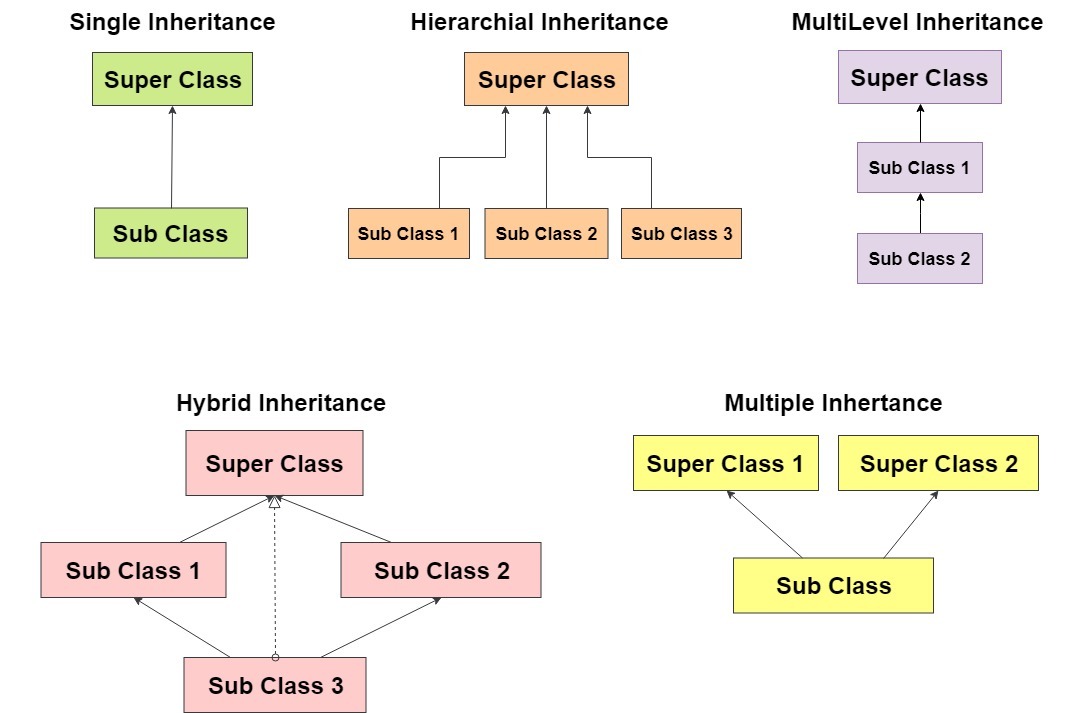
myVar = Parent.var1 + 10

return myVar

Types of Inheritance in Python

In the last tutorial we learned about Inheritance and how a child class can inherit a parent class to utilise its properties and functions.

What if a class want to inherit more than one class? Or it it possible to inherit a class, which already inherits some other class? To answer these questions, lets see the different types of Inheritance.



In Python, there are two types of Inheritance:

1. Multiple Inheritance
2. Multilevel Inheritance

Python - Multiple Inheritance

Multiple Inheritance means that you're inheriting the property of multiple classes into one. In case you have two classes, say A and B, and you want to create a new class which inherits the properties of both A and B, then:

class A:

# variable of class A

# functions of class A

class B:

# variable of class A

# functions of class A

class C(A, B):

# class C inheriting property of both class A and B

# add more properties to class C

So just like a child inherits characteristics from both mother and father, in python, we can inherit multiple classes in a single child class.

As you can see, instead of mentioning one class name in parentheses along with the child class, we have mentioned two class names, separated by comma ,. And just to clear your doubts, yes, you can inherit as many classes you want. Therefore, the syntax should actually be:

class A(A1, A2, A3, ...):

# class A inheriting the properties of A1, A2, A3, etc.

# You can add properties to A class too

Python - Multilevel Inheritance

In multilevel inheritance, we inherit the classes at multiple separate levels. We have three classes A, B and C, where A is the super class, B is its sub(child) class and C is the sub class of B.

Here is a simple example, its just to explain you how this looks in code:

class A:

# properties of class A

class B(A):

# class B inheriting property of class A

# more properties of class B

class C(B):

# class C inheriting property of class B

# thus, class C also inherits properties of class A

# more properties of class C

Using issubclass() method

In python, there is a function which helps us to verify whether a particular class is a sub class of another class, that built-in function is issubclass(paramOne, paramTwo), where paramOne and paramTwo can be either class names or class's object name.

class Parent:

var1 = 1

def func1(self):

# do something

class Child(Parent):

var2 = 2

def func2(self):

# do something else

In order to check if Child class is a child class of Parent class.

>>> issubclass(Child, Parent)

True

Or using the object of the classes,

Parent p = Parent()

Child c = Child()

It's pretty much the same,

>>> issubclass(c, p)

True