

# Multi-Level Inheritance

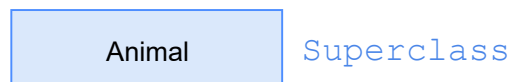
This lesson talks about a hierarchy of classes when one class inherits from its parent class, and that parent class inherits from its parent class, and so on.

## WE'LL COVER THE FOLLOWING ^

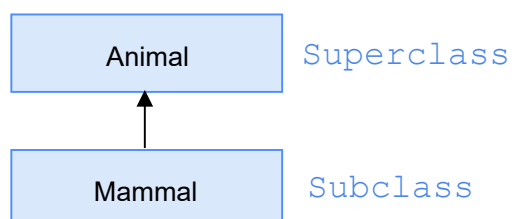
- Introduction
- Implementation

## Introduction #

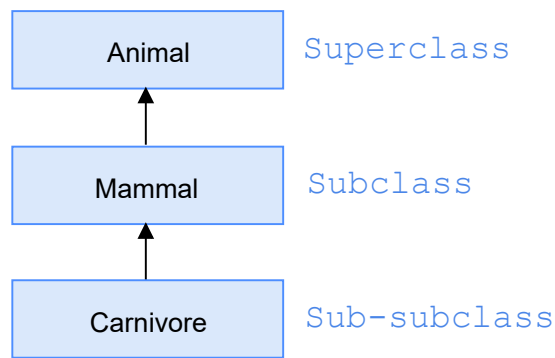
In addition to single-level inheritance, Python also supports multi-level inheritance. This means that you can create a hierarchy of classes, each inheriting from its superclass. The following figure illustrates an example of multi-level inheritance.



1 of 3



2 of 3



3 of 3



The following hierarchy is clear from the above figure:

Class	Superclass	Relation
Carnivore	Mammal	Carnivore <i>is a</i> Mammal
Mammal	Animal	Mammal <i>is an</i> Animal
Animal	-	-

## Implementation #

Syntactically, multi-level inheritance in Python is quite similar to single-level inheritance. Consider the following basic code snippet:

```
class Animal():
    def __init__(self, name, food, characteristic): # Animal's constructor
        self.name = name # Animal's attribute
        self.characteristic = characteristic # Animal's attribute
        self.food = food # Animal's attribute
        print("I am a " + str(self.name) + ".")

class Mammal(Animal): # Mammal inherits from Animal
    def __init__(self, name, food): # Mammal's constructor
        Animal.__init__(self, name, food, "warm blooded") # Animal's constructor
        print("I am warm blooded.")
```



```
class Carnivore (Mammal): # Carnivore inherits from Mammal
    def __init__(self, name): # Carnivore's constructor
        Mammal.__init__(self, name, "meat") # Mammal's constructor

        print ("I eat meat.")

lion = Carnivore("lion") # lion is an instance of Carnivore
```



However, if we move the `print` method in a separate method, the subclass can not only access the method of its parent class—but also override it if needed. This behavior is shown in the highlighted lines of the following code:

```
class Animal ():
    def __init__(self, name, food, characteristic):
        self.name = name
        self.characteristic = characteristic
        self.food = food
    def printer(self):
        print ("I am a " + str(self.name) + ".")

class Mammal (Animal):
    def __init__(self, name, food):
        Animal.__init__(self, name, food, "warm blooded")
    def printer(self):
        print ("I am warm blooded.")

class Carnivore (Mammal):
    def __init__(self, name):
        Mammal.__init__(self, name, "meat")
    def printer(self):
        print ("I eat meat.")

lion = Carnivore("lion")
lion.printer()
```



Now if you remove the `printer()` method of the `Carnivore` class in lines 18, 19, the `lion` object will be able to access the `printer()` method of its super/parent class `Mammal`. The output will then become:

```
I am warm blooded.
```

If you also remove the `printer()` method in 12, 13 of the `Mammal` class, the `lion` object will access the `printer()` method of its superclass `Animal`. It will then print:

I am a lion.

In the next lesson, we will discuss multiple inheritance; it's slightly different from multi-level inheritance.