More on Constructors

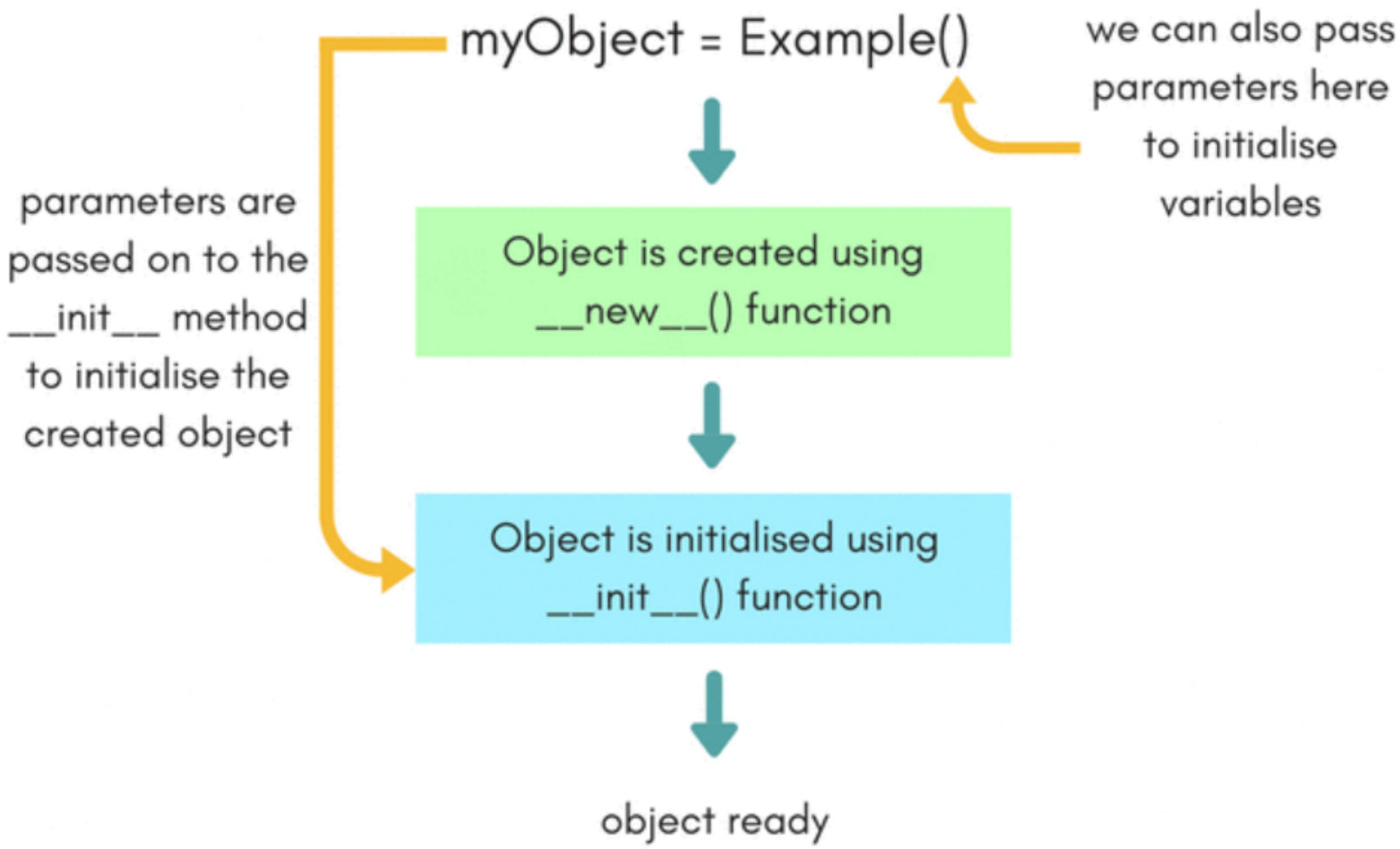
- In python, the object creation part is divided into two parts:

  Object Creation

  Object Initialization

- Object creation is controlled by a static class method with the name **__new__**. Hence when you call Example(), to create an object of the class Example, then the __new__ method of this class is called.
- Python defines this function for every class by default, although you can always do that explicitly too, to play around with object creation.

More on Constructors



Destructors

- Although in python we do have **garbage collector** to clean up the memory, but its not just memory which has to be freed when an object is dereferenced or destroyed, it can be a lot of other resources as well, like closing open files, closing database connections, cleaning up the buffer or cache etc.
- The way we had defined __init__ to initialize our object, we can define the __del__ method to act as a destructor.

Destructors

```
class Example:
    def __init__(self):
            print ("Object created");

    # destructor
    def __del__(self):
        print ("Object destroyed");

# creating an object
myObj = Example();
# to delete the object explicitly
del myObj;
```

```
Object created
Object destroyed
>>>
```

- we have defined the __del__ method to act as a destructor.

Destructors

- Like Destructor is counter-part of a Constructor, function __del__ is the counter-part of function __new__. Because __new__ is the function which creates the object.
- __del__ method is called for any object when the reference count for that object becomes zero.

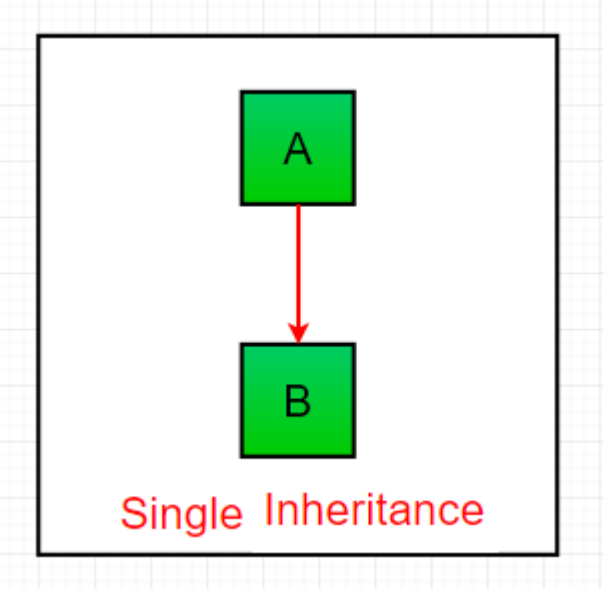
Inheritance in Python

- 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>
```

Types of Inheritance in Python

- **Single Inheritance:** Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.



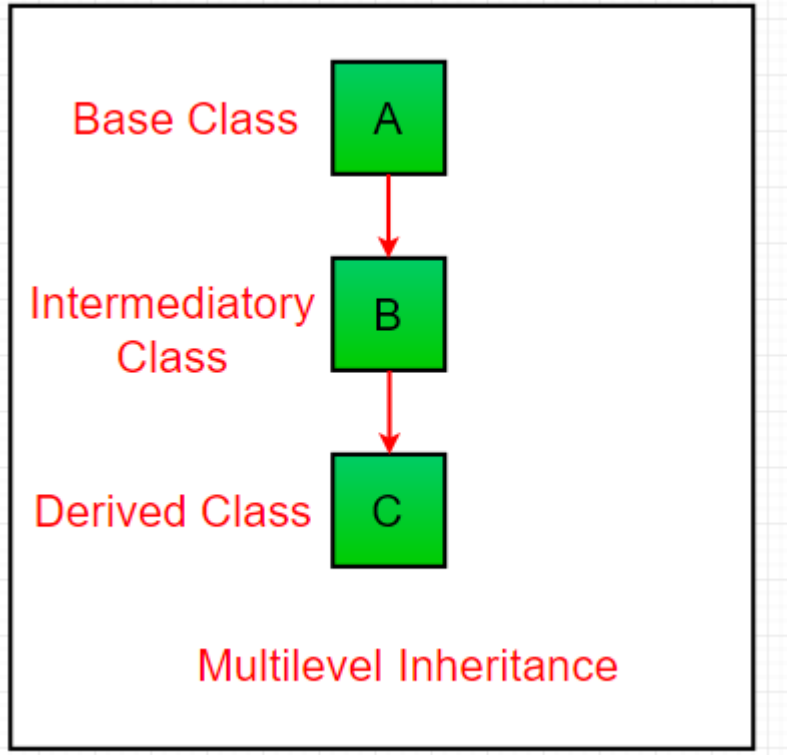
Ex: Inheritance in Python

```
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()
```

```
dog barking
Animal Speaking
>>>
```

- Dog class inherits base Animal class here and acquires all of its properties

- **Multilevel Inheritance**
  In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and grandfather.



Multilevel Inheritance in Python

- 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>
```

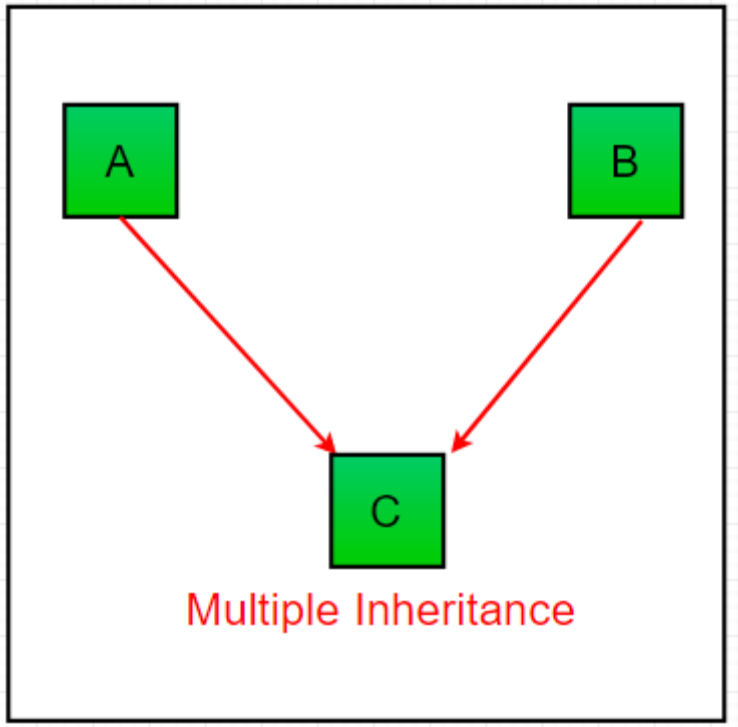
Multilevel Inheritance in Python

```
class Animal:
    def run(self):
        print("Animal running")
#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 Puppy(Dog):
    def eat(self):
        print("Eating bread...")
p = Puppy()
p.bark()
p.run()
p.eat()
```

```
dog barking
Animal Running
Eating bread...
>>>
```

- Here inheritance goes like Animal← Dog ← Puppy. Puppy object can access all the properties of its parents.

- **Multiple Inheritance:** When a class can be derived from more than one base class this type of inheritance is called multiple inheritance. In multiple inheritance, all the features of the base classes are inherited into the derived class.



Multiple Inheritance in Python

- Python provides 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>
```

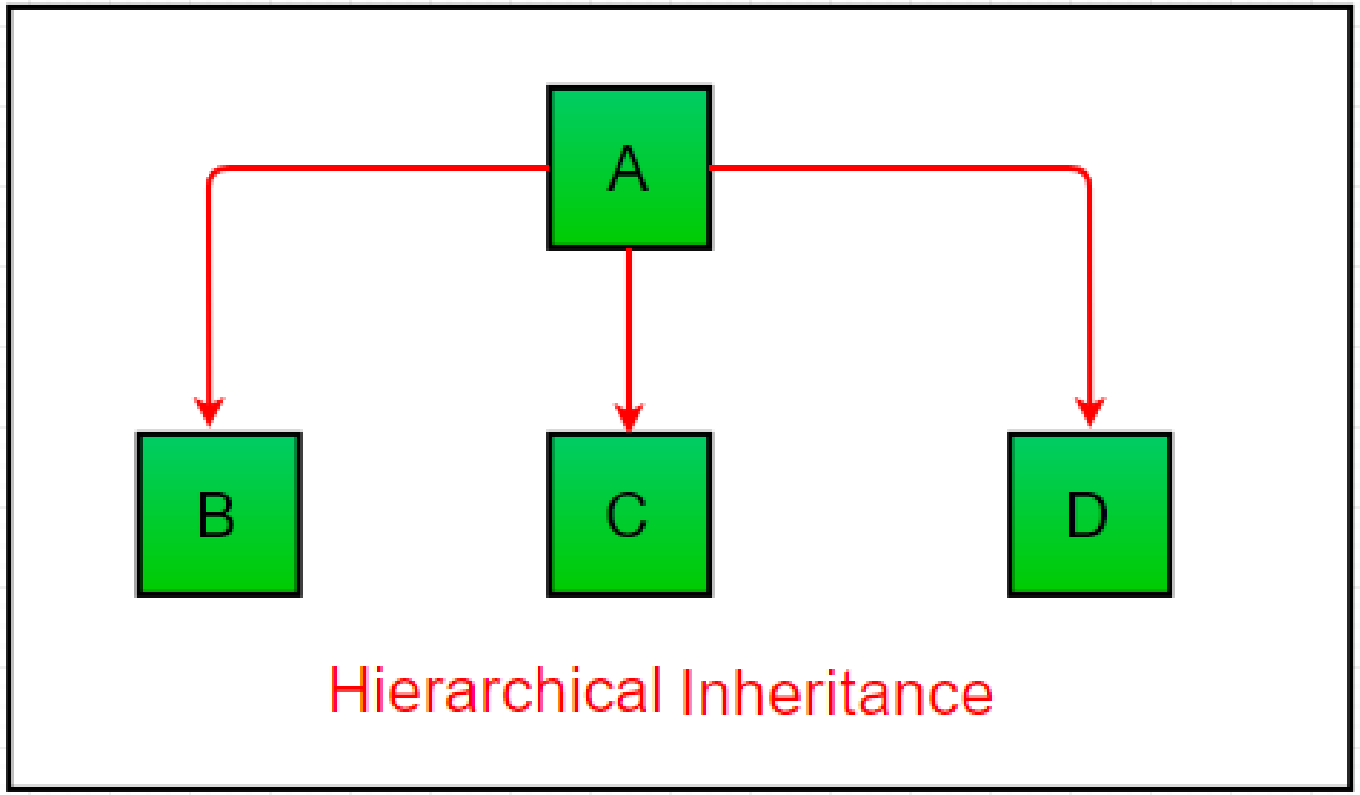
Multiple Inheritance in Python

```
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))
```

```
30
200
0.5
>>>
```

- Here derived/child class inherits both Calculation1 and Calculation2 classes and has access to their properties.

- **Hierarchical Inheritance:** When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.



```
# Python program to demonstrate
# Hierarchical inheritance
# Base class
class Parent:
      def func1(self):
          print("This function is in parent class.")
# Derived class1
class Child1(Parent):
      def func2(self):
          print("This function is in child 1.")
# Derivied class2
class Child2(Parent):
      def func3(self):
          print("This function is in child 2.")

# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

Checking for Inheritance

```
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(isinstance(d,Calculation1))
```

- The **issubclass(sub, sup)** method returns true if the first class is the subclass of the second class, and false otherwise.
- The **isinstance(obj, class)** method returns true if the first parameter, i.e., obj is the instance of the second parameter, i.e., class.