Python super() Function

One of the key Object-Oriented Concept is Inheritance in Python. When a class inherits some or all properties and methods from another class is known as Inheritance. In this case, the inherited class is the subclass or child class and the latter class is the parent class or super class.

If a child class wants to access Parent class methods or constructor, super() function can be useful. Python super() is an inbuilt function that returns the proxy object that allows you to refer parent class by ‘super.’ The super() function in Python can be used to gain access to inherited methods, which is either from the parent or sibling class.

Python super() function gives you access to methods in a superclass from the subclass that inherits from it. The super() alone returns a temporary object of the superclass that then allows you to call that superclass’s methods.

The super() method can optionally take two parameters:

* Subclass
* An object that is an instance of that subclass.

## Using super() in Subclass Example:

class Simple:  
 def \_\_init\_\_(self):  
 print(**"Simple Parent Class"**)  
  
class SuperSimple(Simple):  
 def \_\_init\_\_(self):  
 super(SuperSimple, self).\_\_init\_\_()  
 print(**"SuperSimple Child Class"**)

This super call is exactly same as:

super().\_\_init\_\_()

## super() in Single Inheritance:

In this example, a Rectangle is a Parent Class, and Square is a Child class. A superclass’s \_\_init\_\_() method is called (Rectangle.\_\_init\_\_()) from that of Square by using a super() keyword.

class Rectangle:  
 def \_\_init\_\_(self, length, width):  
 self.length = length  
 self.width = width  
  
 def area(self):  
 return self.length \* self.width  
  
 def perimeter(self):  
 return 2 \* self.length + 2 \* self.width  
  
class Square(Rectangle):  
 def \_\_init\_\_(self, length):  
 super().\_\_init\_\_(length, length)  
  
  
sqr = Square(4)  
print(**"Area of Square is:"**, sqr.area())  
  
rect = Rectangle(2, 4)  
print(**"Area of Rectangle is:"**, rect.area())

## Benefits of super() function:

Following are some advantages of super function:-

* No Need to remember or specify the parent class name to access its methods. This function can be used both in single and multiple inheritances.
* Modularity is achieved by isolating changes with code reusability as there is no need to rewrite the entire function.
* Super function in Python is called dynamically because Python is a dynamic language unlike other languages.

## super() function in Multiple Inheritance:

Here Address class is Parent class for ValidAddress and ValidNumber class. CustomerAddress is child of both the Classes.

Single super() call is percolate the instructions to both the constructors. However, this method is not applicable when Parent class constructors have different argument list.

class Address:  
 def \_\_init\_\_(self, address):  
 self.address = address  
 print(**"Address provided "**, address)  
  
  
class ValidAddress(Address):  
 def \_\_init\_\_(self, address):  
 super().\_\_init\_\_(address)  
 print(**"Zipcode is provided with "**, self.address)  
  
  
class ValidNumber(Address):  
 def \_\_init\_\_(self, address):  
 super().\_\_init\_\_(address)  
 print(**"house\_number is provided with "**, self.address)  
  
  
class CustomerAddress(ValidAddress, ValidNumber):  
 def \_\_init\_\_(self, address):  
 super().\_\_init\_\_(address)  
 print(**"DONE!"**)  
  
  
cust01 = CustomerAddress(**"1234 Somebody Blvd, FewCity 99887"**)

## Override methods using super()

When you define a parent class method in the child class, then this process is called Overriding. Or We can say that the child class can override methods of its parent or superclass by defining the function with the same name

Rules for overriding:

* The name of the method should be the same and its parameters as well.
* If the superclass method is private (prefixed with double underscores), then you can’t override it.

class Simple:  
 def \_\_init\_\_(self):  
 print(**"Simple Parent Class"**)  
  
 def simple\_method(self):  
 print(**"Method from Simple Class"**)  
  
  
class SuperSimple(Simple):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(**"SuperSimple Child Class"**)  
  
 def simple\_method(self):  
 super(SuperSimple, self).simple\_method()  
 print(**"Method from SuperSimple Class"**)  
  
obj = SuperSimple()  
obj.simple\_method()

Please note that, in order to use the function properly, the following conditions must be met:

* The method being called upon by *super()* must exist
* Both the Super class and child class functions should have a matching argument signature
* Every occurrence of the method must include *super()*after you use it