***Super(type, obj)***

**Type**: is the start point in MRO (method resolution order, like hierarchy) from where super () searches parent classes

**Obj**: is the object or subtype of the **type,** it is where the returned method binds to

**EX:**

class A(object):  
 def \_\_init\_\_(self):  
 self.word = "This is A !"  
  
 def print(self):  
 print(self.word)  
  
  
class B(A):  
 def \_\_init\_\_(self):  
 self.word = "This is B"  
  
 def print(self):  
 print(self.word)  
  
  
class C(B):  
 def \_\_init\_\_(self):  
 self.word = "This is C"  
  
 def print(self):  
 super(C, c).print()  
  
  
  
a = A()  
b = B()  
c = C()  
c.print()

**Result:**

>>> This is C

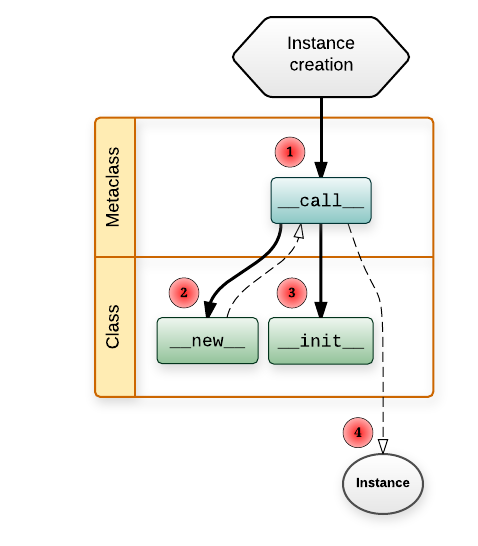
What happens here is that, super () will traverse the Class C parent classes (Class A and Class B) until an attribute print() is found, and bind this print() method to object c ( an instance of Class C). In this case, it binds print() in b to c, we can see in this way: c pass c’ self to b.print(self), that is why printing out “This is C”

If we change Class c into:

class C(B):  
 def \_\_init\_\_(self):  
 self.word = "This is C"  
  
 def print(self):  
 super(B, b).print()

Result would be : >>> This is B

Because, it found print() in A, and bind the print() to b, and use self from b.



***Singleton\_Metaclass***

