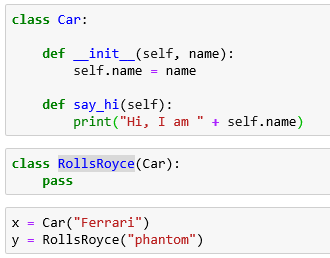
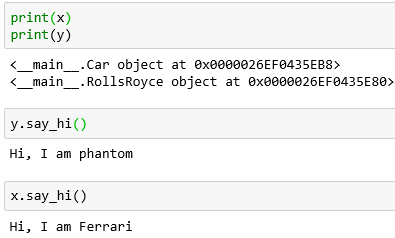
**Inheritance: it is the mechanism of deriving new classes from existing ones**

It means child class can acquires all the properties and methods from parent class

Advantages: code reusability

Syntax:



As the class RollsRoyce is a subclass of Car, it inherits in this case both the method \_\_init\_\_ and say\_hi.

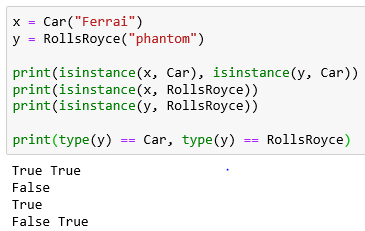
**Diff b/w type and Instance**

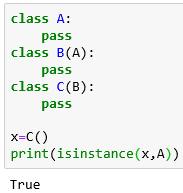
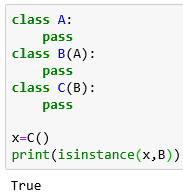
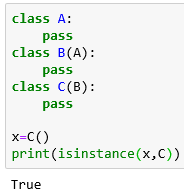
isinstance returns True if we compare an object either with the class it belongs to or with the superclass.

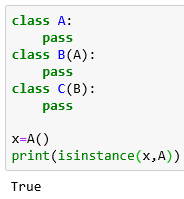
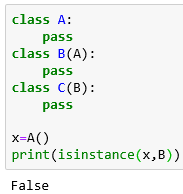
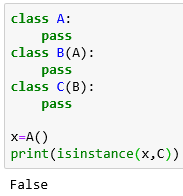
Whereas == operator only returns True, if we compare an object with its own class.

**Isinstance:**

Return whether an object is an instance of a class or of a subclass thereof







**Overriding: if we override the say\_hi function inside the sub class**

A method of a parent class gets overridden

by simply defining in the child class a

method with the same name.

**Note**:- If a method is overridden in a class,

the original method can still be .

accessed, but we have to do it

by calling the method directly with

the class name,

i.e. vehicle.say\_hi(y).



super is not realls necessary in this case.

One could argue that it makes the

code more maintainable, because

we could change the name of the

parent class, but this is seldom done

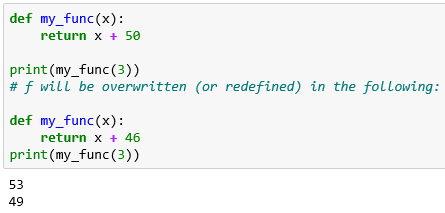
anyway in existing classes. The real

benefit of super shows when we use

it with multiple inheritance.

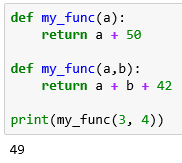
**Overwriting:**

If we overwrite a function, the original function will be gone. The function will be redefined. This process has nothing to do with object orientation or inheritance.



**Overriding:**

Within the class Writing two or more functions with same name and different parameters



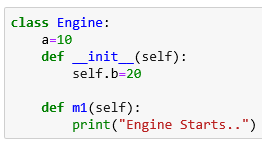
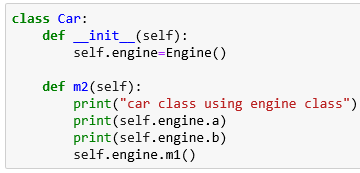
**Using members of one class inside another class:**

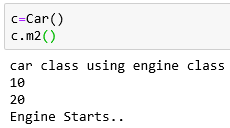
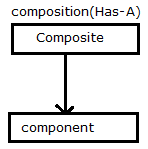
* By Composition( Has-A relationship )
* By inheritance( IS-A relationship )

**By Composition( Has-A relationship ):-**

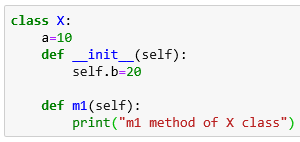
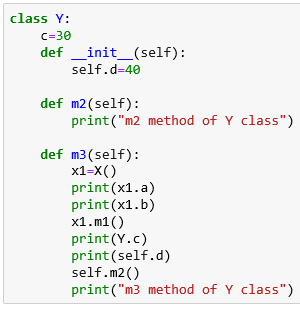
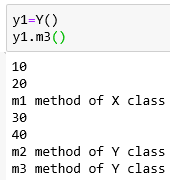
By using Class Name or by creating object we can access members of one class inside another class.

**Advantage**: code reusability







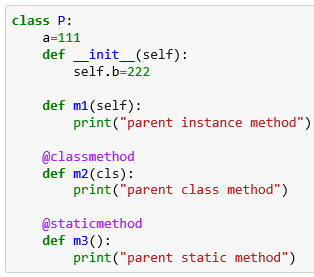
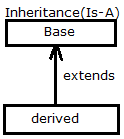
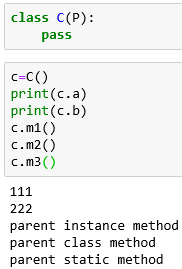


**By inheritance( IS-A relationship ):-**

Whatever variables, functions, constructors present in parent class are available to child class.

Advantages: code reusability

We can extend existing functionality with some extra functionality

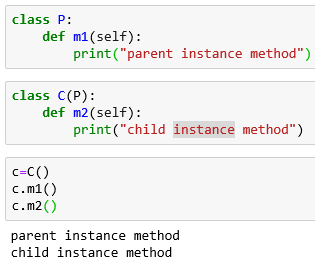


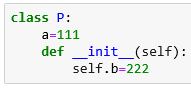
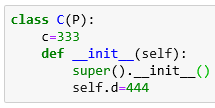
Demo2: In below example while creating object to child class, parent

class object also created automatically

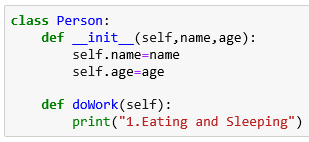
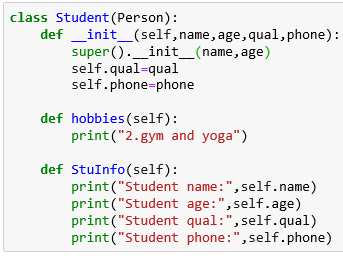
Demo3: if we want to call parent class constructor from child class

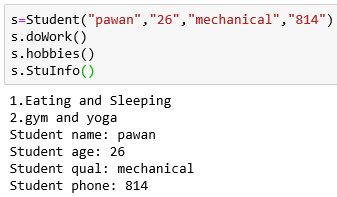
just use super () in child class constructor



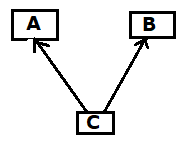
 





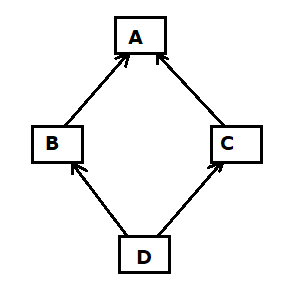


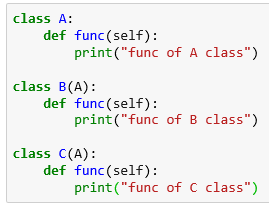
**Single Inheritance: Multiple Inheritance:**

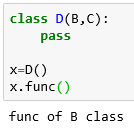


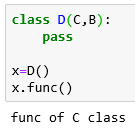
Parent class

Child class

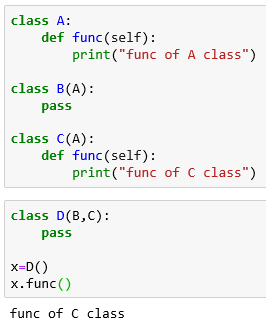
**Diamond problem:**





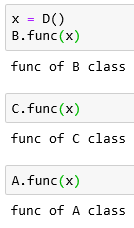
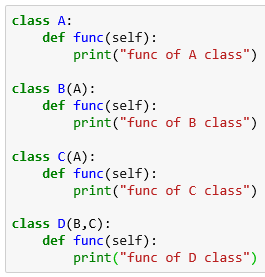


**The case in which func will be overridden only in one of the classes B or C, Ex: in C**

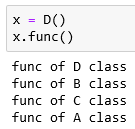
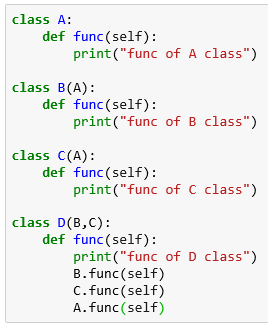


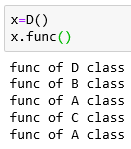
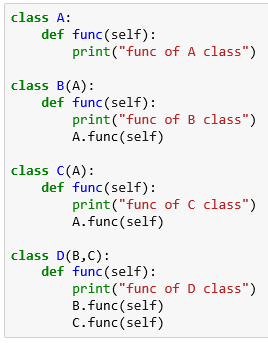
**Method Resolution Order(MRO) and super**

If we create instance for “D class” then we can see that only the code of “Func of D class” will be executed. We can also explicitly call the functions func of the other classes via the class name.



Let me assume that the function “func of D” should be execute the code of “func of B,C and A” as well, when it is called. We could implement it like this:





The bug is that the method “func of A class” will be called twice:

Non-pythonic pythonic way:

