



Python Class

#Python Notes

OOP's Concept



Class

- A class is a blueprint for the object.
- Classes are defined by the "**Class**" keyword.
- Class name in python is preceded with class keyword followed by colon (:).
- Classes commonly contains data field to store the data and methods for defining behaviours.
- **Syntax :-**
class MyClass:

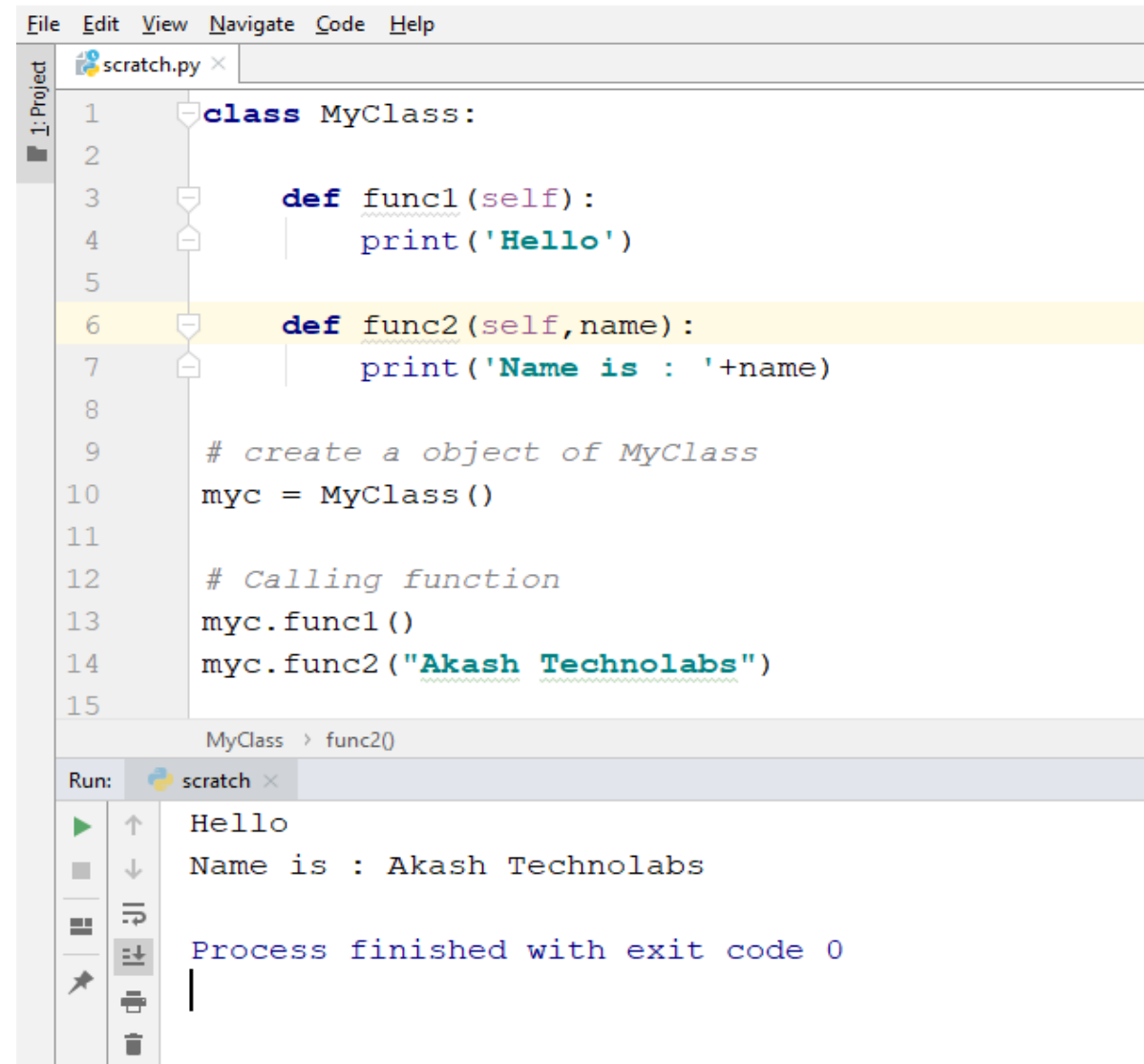


Object

- Object is an instance of a class.
- **Syntax :-**
`myc= MyClass()`
- Here, **myc** is object of class **MyClass**.



Example



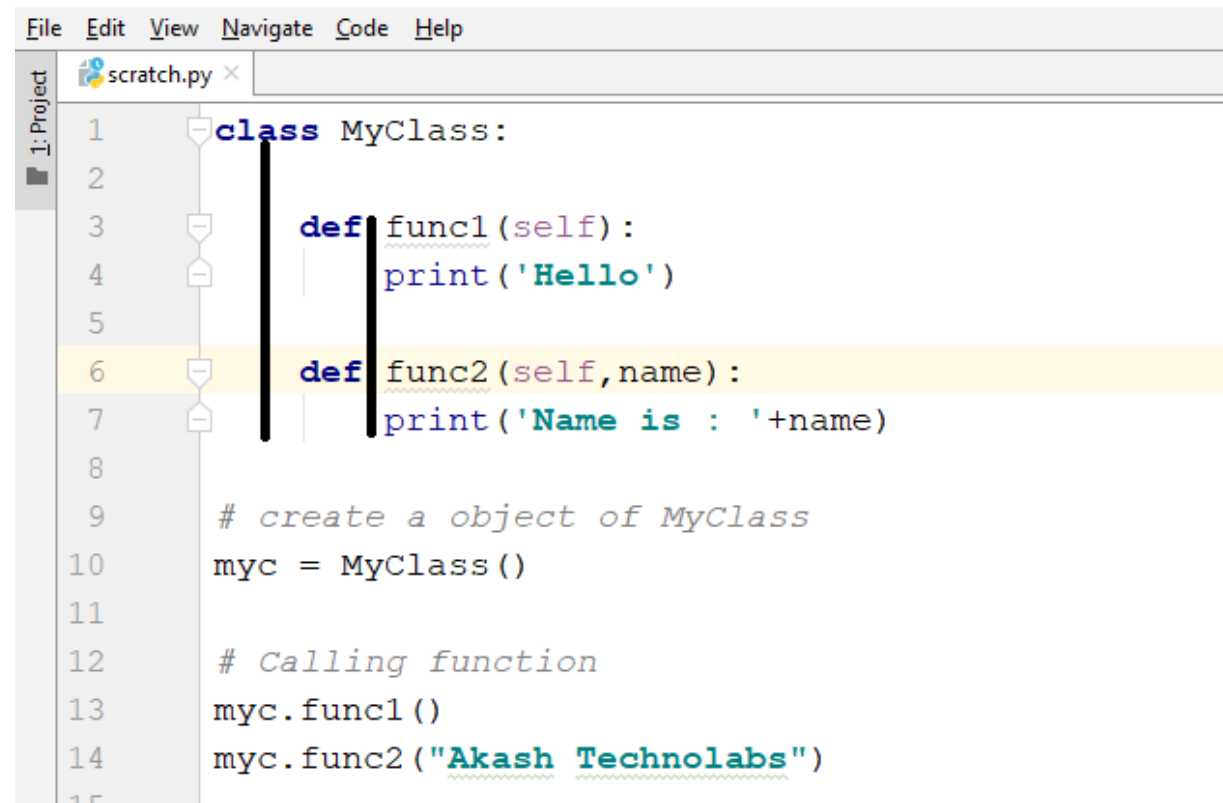
The screenshot shows an IDE window with a file named 'scratch.py'. The code defines a class 'MyClass' with two methods: 'func1' which prints 'Hello', and 'func2' which prints 'Name is : ' followed by a variable 'name'. Below the class definition, an object 'myc' is created, 'func1' is called, and 'func2' is called with the argument 'Akash Technolabs'. The output window at the bottom shows the execution results: 'Hello', 'Name is : Akash Technolabs', and 'Process finished with exit code 0'.

```
File Edit View Navigate Code Help
scratch.py x
1 class MyClass:
2
3     def func1(self):
4         print('Hello')
5
6     def func2(self, name):
7         print('Name is : '+name)
8
9     # create a object of MyClass
10    myc = MyClass()
11
12    # Calling function
13    myc.func1()
14    myc.func2("Akash Technolabs")
15

MyClass > func2()
Run: scratch x
Hello
Name is : Akash Technolabs
Process finished with exit code 0
```

Note

- Everything in a class is indented, just like the code in the function, loop, if statement, etc. Anything not indented is not in the class

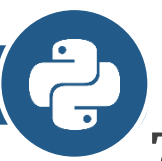


```
File Edit View Navigate Code Help
scratch.py x
1 class MyClass:
2
3     def func1(self):
4         print('Hello')
5
6     def func2(self, name):
7         print('Name is : '+name)
8
9     # create a object of MyClass
10    myc = MyClass()
11
12    # Calling function
13    myc.func1()
14    myc.func2("Akash Technolabs")
15
```



Difference between Method and Function

Method	Functions
Method is called by its name, but it is associated to an object (dependent).	Function is block of code that is also called by its name. (independent)
It may or may not return any data.	It may or may not return any data.
A method can operate on the data (instance variables) that is contained by the corresponding class	Function does not deal with Class and its instance concept.



What is self?

- All methods in python including some special methods like initializer have first parameter self .
- This parameter refers to the object which invokes the method.
- When you create new object the self parameter in the `__init__` method is automatically set to reference the object you have just created.



Sum Of 2 Numbers Using Class

```
scratch.py x
1 class MyClass:
2
3     def func1(self,n1,n2):
4         ans=n1+n2
5         print('Ans is : ',ans)
6
7     # create a object of MyClass
8     myc = MyClass()
9
10    # Calling function
11    myc.func1(10,20)
12
```

Run: scratch x

```
Ans is : 30

Process finished with exit code 0
```

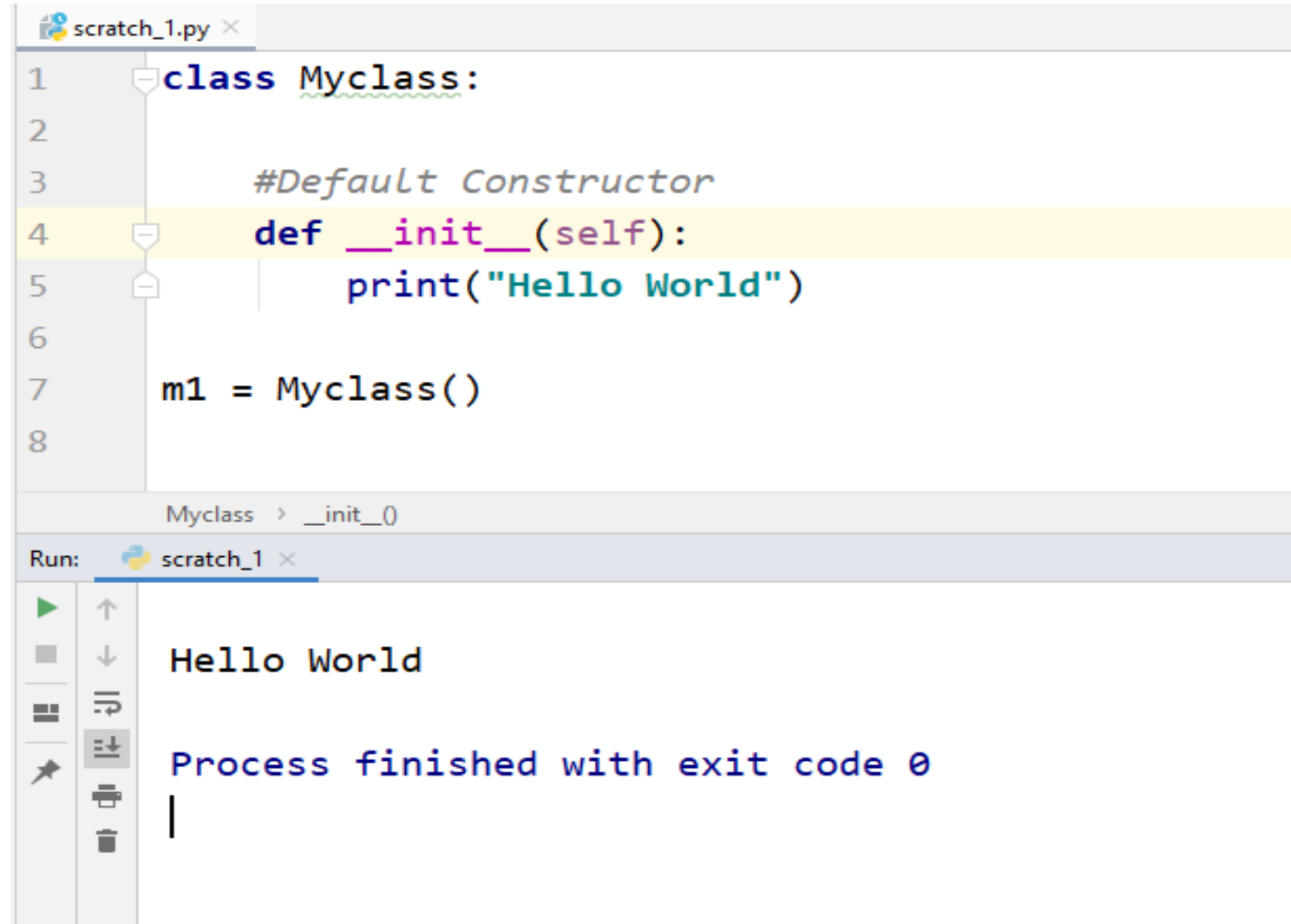


Python Constructors

- A constructor is a class function that instantiates an object to predefined values.
- It begins with a double underscore (__). It `__init__()` method.
- There are two types of Constructor.
 - Default Constructor
 - Parameterized Constructor



Default Constructor Example



The screenshot shows a Python IDE with a file named `scratch_1.py`. The code defines a class `Myclass` with a default constructor `__init__` that prints "Hello World". An instance `m1` of the class is created. The output window shows the execution result: "Hello World" and "Process finished with exit code 0".

```
1 class Myclass:
2
3     #Default Constructor
4     def __init__(self):
5         print("Hello World")
6
7 m1 = Myclass()
8
```

Myclass > __init__()

Run: scratch_1

Hello World

Process finished with exit code 0

Parameterized Constructor Example

```
scratch_1.py x
1 class MyClass:
2
3     #Parameterized Constructor
4     def __init__(self,name):
5         print("Value is ",name)
6
7     m1 = MyClass("Akash Technolabs")
8
Myclass > __init__()
Run: scratch_1 x
Value is Akash Technolabs
Process finished with exit code 0
```

Assign String Value to Class Variable Using Method

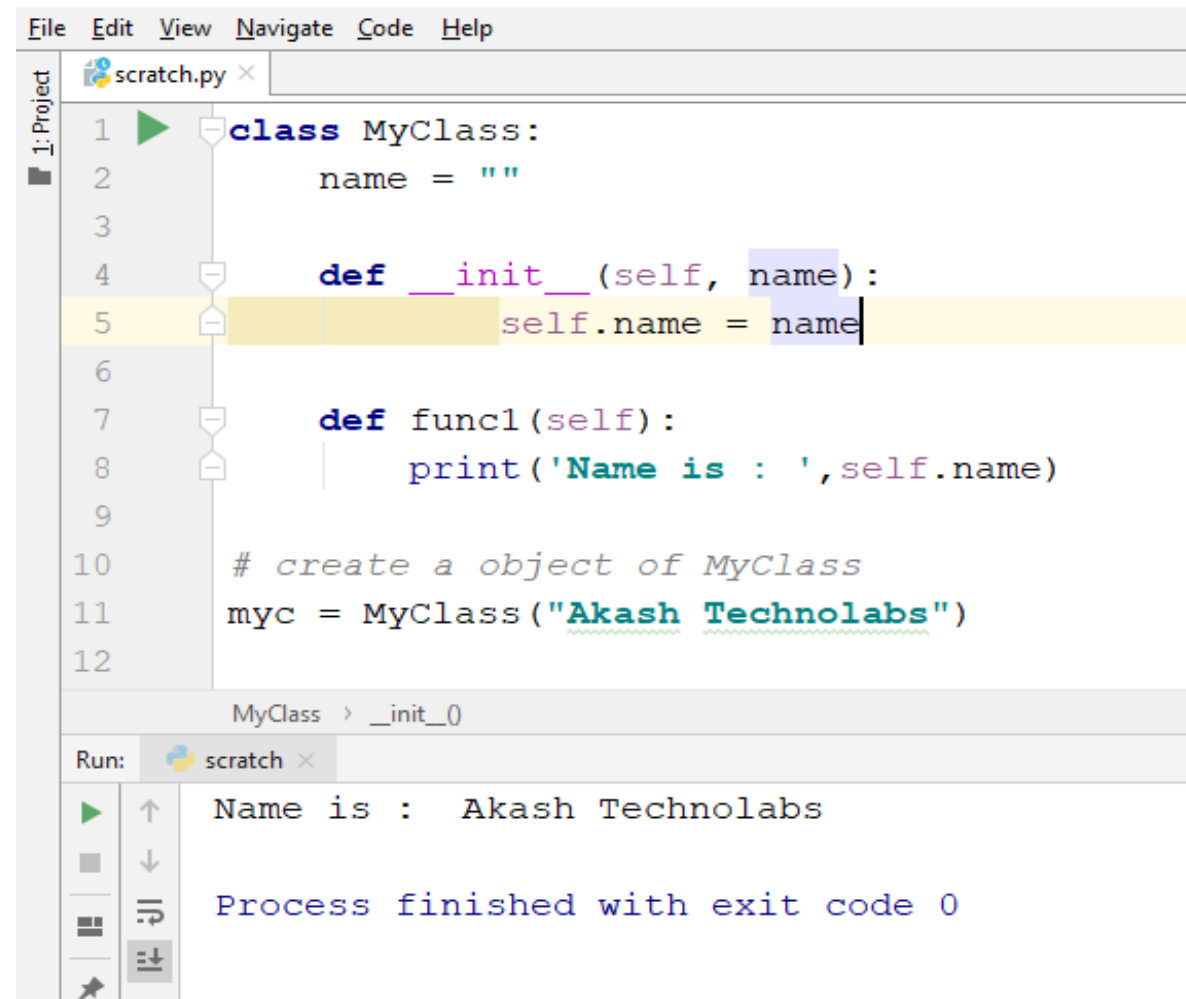
```
scratch_1.py ×
1 class MyClass:
2     name=""
3
4     def func1(self):
5         print("Hello Function1")
6
7     def func2(self,name):
8         self.name=name
9
10    def func3(self):
11        print("Value is ", self.name)
12
13 m1 = MyClass()
14 m1.func1()
15 m1.func2("Akash Technolabs")
16 m1.func3()
```

```
Run: scratch_1 ×
C:\Users\Devanshi\PycharmProjects\demo\venv\Scr
Hello Function1
Value is  Akash Technolabs

Process finished with exit code 0
|
```

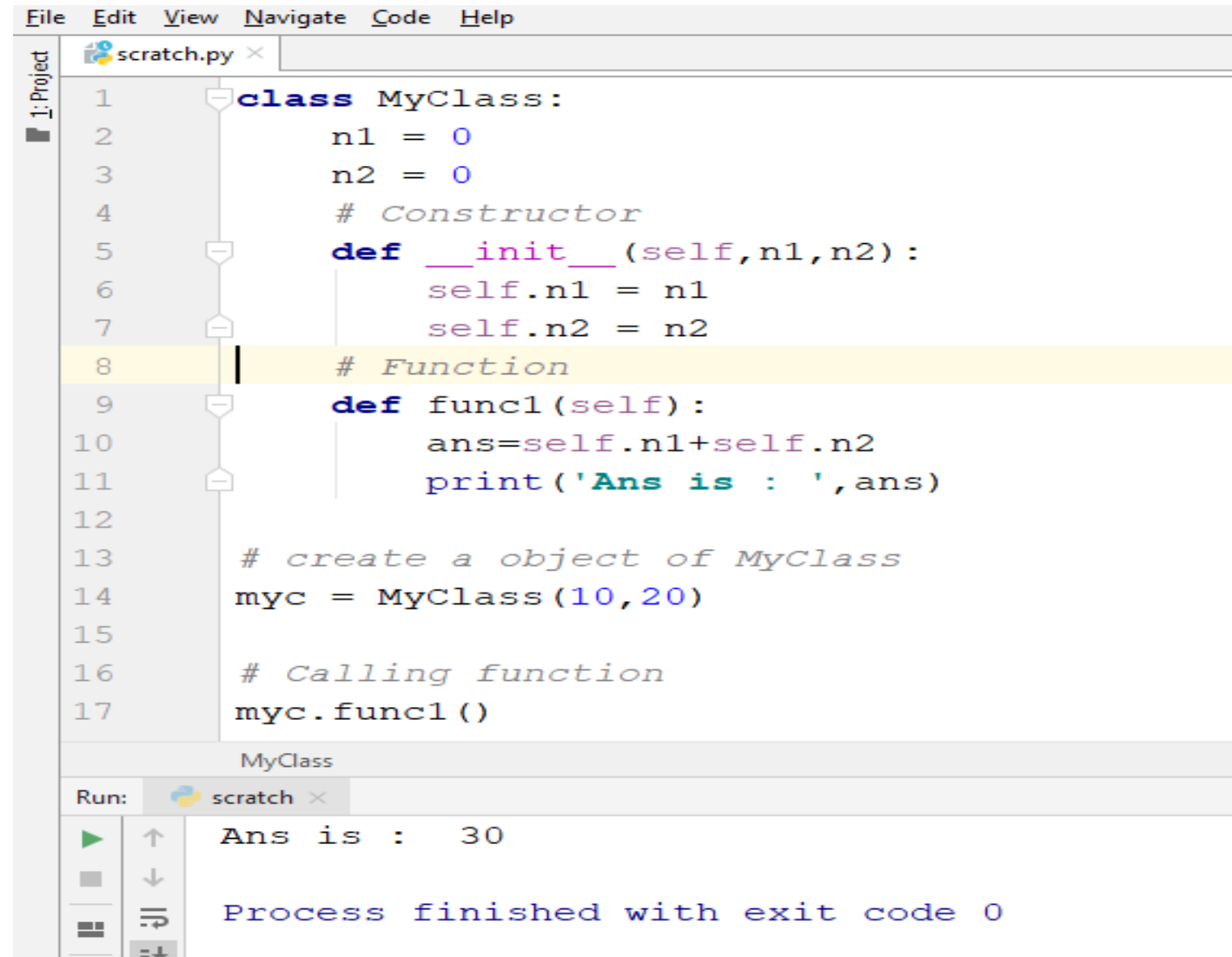


Assign String Value to Class Variable Using Constructor



```
File Edit View Navigate Code Help
scratch.py x
1: Project
1 class MyClass:
2     name = ""
3
4     def __init__(self, name):
5         self.name = name
6
7     def func1(self):
8         print('Name is : ', self.name)
9
10 # create a object of MyClass
11 myc = MyClass("Akash Technolabs")
12
MyClass > __init__
Run: scratch x
Name is : Akash Technolabs
Process finished with exit code 0
```

Example 2



```
File Edit View Navigate Code Help
scratch.py x
1 class MyClass:
2     n1 = 0
3     n2 = 0
4     # Constructor
5     def __init__(self,n1,n2):
6         self.n1 = n1
7         self.n2 = n2
8     # Function
9     def func1(self):
10         ans=self.n1+self.n2
11         print('Ans is : ',ans)
12
13     # create a object of MyClass
14     myc = MyClass(10,20)
15
16     # Calling function
17     myc.func1()
```

MyClass

Run: scratch x

Ans is : 30

Process finished with exit code 0

Inheritance

- Inheritance allows programmer to create a general class first then later extend it to more specialized class.
- Using inheritance you can inherit all access data fields and methods, plus you can add your own methods and fields, thus inheritance provide a way to organize code, rather than rewriting it from scratch.



Syntax to create a subclass is:

```
■ class SubClass(SuperClass):  
    # data fields  
    # instance methods
```



Types of Inheritance

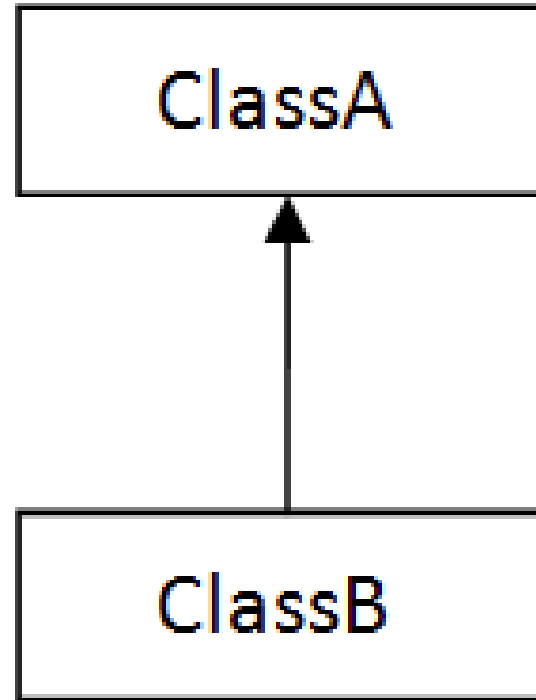
- Single-Level inheritance
- Multi-Level inheritance
- Multiple inheritance
- Hierarchical Inheritance
- Hybrid Inheritance



Single level Inheritance

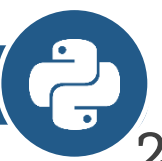
- When one class inherits another class, it is known as single level inheritance.





Example

```
File Edit View Navigate Code Help
scratch.py x
1: Project
1 # define parent class
2 class Parent:
3
4     def __init__(self):
5         print("Calling parent constructor")
6
7     def parentMethod(self):
8         print("Calling parent method")
9
10 # define child class
11 class Child(Parent):
12     def __init__(self):
13         print("Calling child constructor")
14
15     def childMethod(self):
16         print("Calling child method")
17
18 c = Child()           # object of child
19 c.childMethod()      # child class method
20 c.parentMethod()     # parent class method
21
```



Output

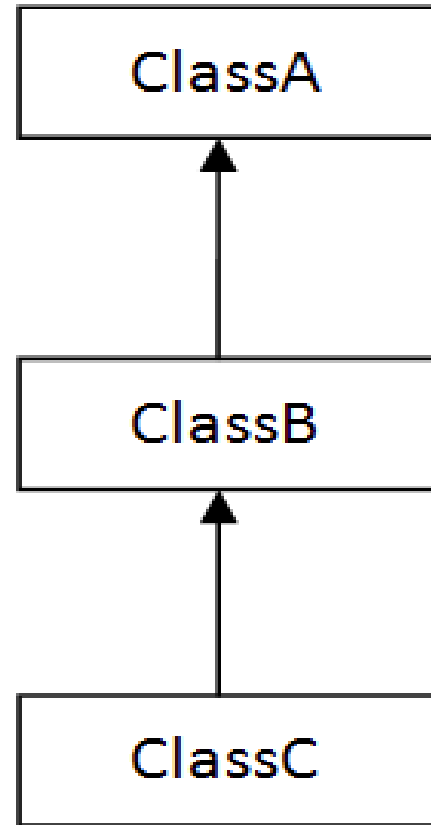
```
Run: scratch x
Calling child constructor
Calling child method
Calling parent method
Process finished with exit code 0
```



Multi level Inheritance

- When one class inherits another class which is further inherited by another class, it is known as multi level inheritance.





Example

```
File Edit View Navigate Code Help
scratch.py x
1 # define parent class
2 class Parent:
3     def __init__(self):
4         print("Calling parent constructor")
5
6     def parentMethod(self):
7         print("Calling parent method")
8
9 # define child class
10 class Child(Parent):
11     def __init__(self):
12         print("Calling child constructor")
13
14     def childMethod(self):
15         print("Calling child method")
16
```



Cont...

```
File Edit View Navigate Code Help
scratch.py x
12 print ("Calling child constructor")
13
14 def childMethod(self):
15     print ("Calling child method")
16
17 # define sub child class
18 class SubChild(Child):
19     def __init__(self):
20         print ("Calling sub child constructor")
21
22     def subchildMethod(self):
23         print ("Calling sub child method")
24
25
26 sc = SubChild()           # object of sub child
27 sc.subchildMethod()       # sub child class method
28 sc.childMethod()          # child class method
29 sc.parentMethod()         # parent class method
30
31
32
33
```



Output

```
Calling sub child constructor
Calling sub child method
Calling child method
Calling parent method

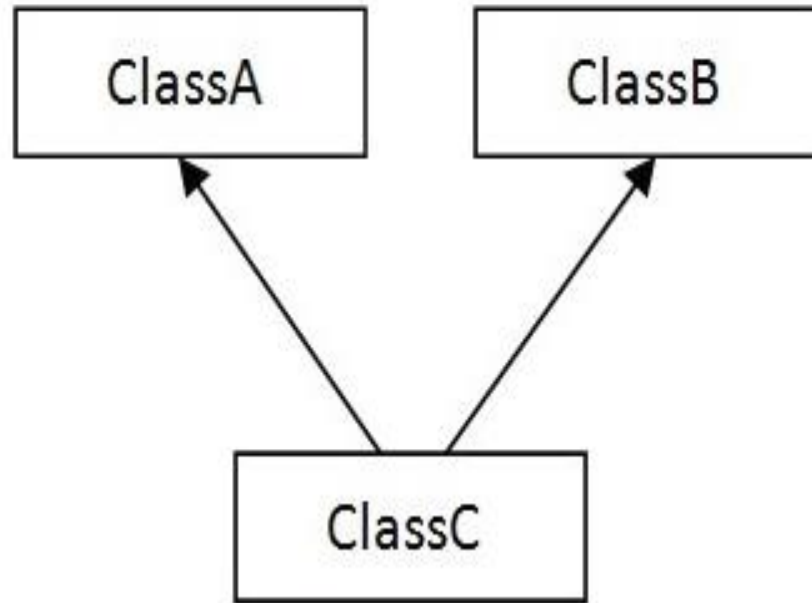
Process finished with exit code 0
|
```



Multiple Inheritance

- If a class can inherit members from more than one base class is known as Multiple Inheritance.





Syntax

- Syntax :-

```
class A:      # define your class A
```

```
.....
```

```
class B:      # define your class B
```

```
.....
```

```
class C(A, B): # subclass of A and B
```

```
.....
```



Example

```
File Edit View Navigate Code Help
scratch.py x
1: Project
1 # define parent class1
2 class MyParentClass1():
3
4     def method_Parent1(self):
5         print("Parent1 method called")
6
7 # define parent class2
8 class MyParentClass2():
9
10    def method_Parent2(self):
11        print("Parent2 method called")
12
13 # define Child class
14 class ChildClass(MyParentClass1, MyParentClass2):
15
16    def child_method(self):
17        print("child method")
18
19 c = ChildClass()          # object of child
20 c.method_Parent1()        # parent class1 method
21 c.method_Parent2()        # parent class2 method
22 c.child_method()          # child class method
23
```



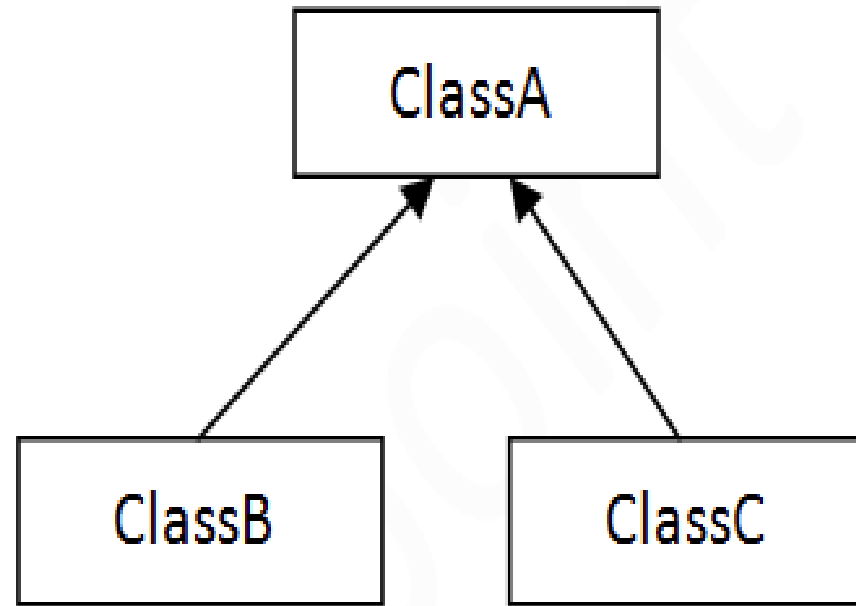
Output

```
Run: scratch x
Parent1 method called
Parent2 method called
child method
Process finished with exit code 0
```


Hierarchical Inheritance

- If more than one class can inherit members from one class is known as Hierarchical Inheritance.





Example

```
File Edit View Navigate Code Help
scratch.py x
1: Project
1  # define parent class
2  class Parent:
3      def __init__(self):
4          print("Calling parent constructor")
5
6      def parentMethod(self):
7          print("Calling parent method")
8
9  # define child class
10 class Child(Parent):
11     def __init__(self):
12         print("Calling child constructor")
13
14     def childMethod(self):
15         print("Calling child method")
16
```



Cont...

```
File Edit View Navigate Code Help
scratch.py x
1: Project
13
14     def childMethod(self):
15         print ("Calling child method")
16
17     # define sub child class
18     class Child2(Parent):
19         def __init__(self):
20             print ("Calling child2 constructor")
21
22         def childMethod2(self):
23             print ("Calling child2 method")
24
25
26     sc = Child2()           # object of child
27     sc.childMethod2()      # Child 2 class method
28     sc.parentMethod()      # parent class method
29
30
31
32
```



Output

```
Calling child2 constructor  
Calling child2 method  
Calling parent method
```

```
Process finished with exit code 0
```

```
|
```



Hybrid Inheritance

- Hybrid Inheritance is implemented by combining more than one type of inheritance.



Example

```
File Edit View Navigate Code Help
scratch.py x
1: Project
1  # define parent class1
2  class MyParentClass1():
3
4      def method_Parent1(self):
5          print("Parent1 method called")
6
7  # define parent class2
8  class MyParentClass2():
9
10     def method_Parent2(self):
11         print("Parent2 method called")
12
13     # define Child class
14     class ChildClass(MyParentClass1, MyParentClass2): #Multiple Inheritance
15
16     def child_method(self):
17         print("child method")
18
```



Cont...

```
File Edit View Navigate Code Help
scratch.py x
1: Project
12
13 # define Child class
14 class ChildClass(MyParentClass1, MyParentClass2): #Multiple Inheritance
15
16     def child_method(self):
17         print("child method")
18
19 # define Child class2
20 class ChildClass2(MyParentClass1): #Hierarchical Inheritance
21
22     def child_method2(self):
23         print("child method2")
24
25 c = ChildClass() # object of child
26 c.method_Parent1() # parent class1 method
27 c.method_Parent2() # parent class2 method
28 c.child_method() # child class method
29
30 c2=ChildClass2() # object of child class 2
31 c2.child_method2() # child class2 method
32 c2.method_Parent1() # parent class1 method
33
```



Output

```
Parent1 method called  
Parent2 method called  
child method  
child method2  
Parent1 method called  
  
Process finished with exit code 0  
|
```



Polymorphism

- Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).
- Types of Polymorphism :-
 - Overloading Methods
 - Overriding Methods

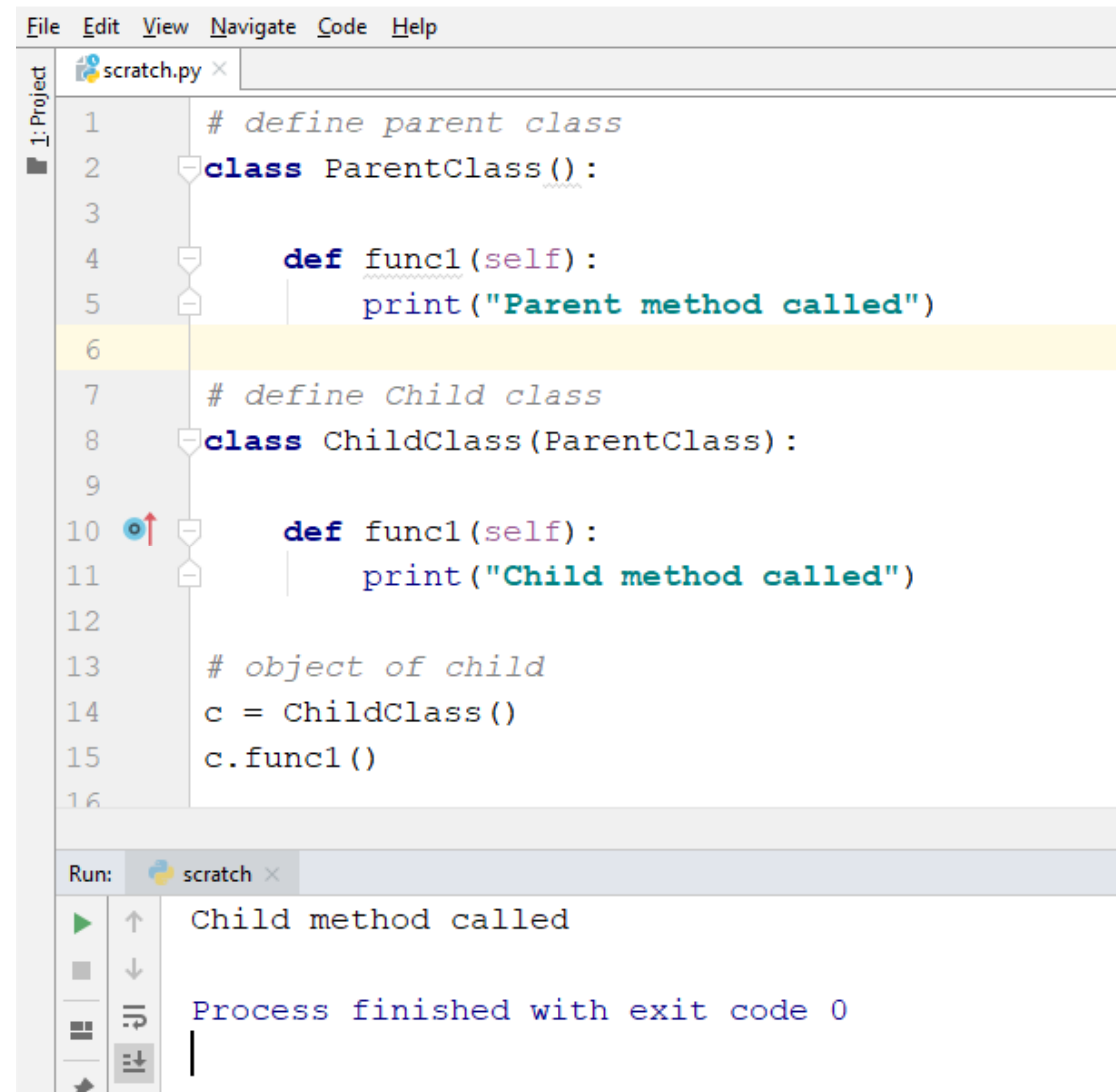


Overriding Methods

- **Override** means having two methods with the same name but doing different tasks. It means that one of the methods overrides the other.
- If there is any method in the superclass and a method with the same name in a subclass, then by executing the method, the method of the corresponding class will be executed.



Example 1



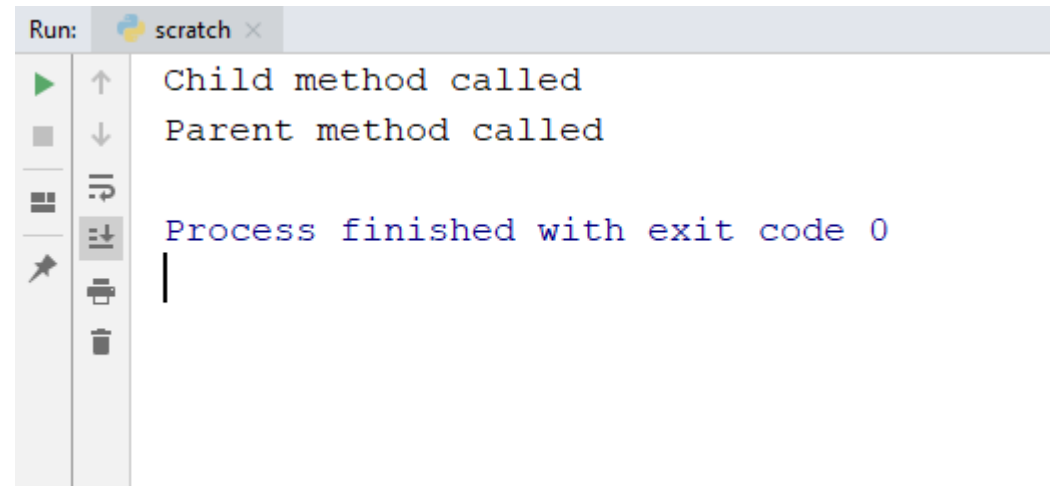
```
File Edit View Navigate Code Help
scratch.py x
1: Project
1 # define parent class
2 class ParentClass():
3
4     def func1(self):
5         print("Parent method called")
6
7 # define Child class
8 class ChildClass(ParentClass):
9
10    def func1(self):
11        print("Child method called")
12
13 # object of child
14 c = ChildClass()
15 c.func1()
16
Run: scratch x
Child method called
Process finished with exit code 0
```

Example 2

```
File Edit View Navigate Code Help
scratch.py x
1: Project
1  # define parent class
2  class ParentClass():
3
4      def func1(self):
5          print("Parent method called")
6
7  # define Child class
8  class ChildClass(ParentClass):
9
10     def func1(self):
11         print("Child method called")
12
13 # object of child
14 c = ChildClass()
15 c.func1()
16
17 # object of parent
18 p=ParentClass()
19 p.func1()
20
```



Output



A screenshot of a terminal window titled "Run: scratch". The terminal displays the following output:

```
Child method called
Parent method called

Process finished with exit code 0
|
```

The terminal interface includes a vertical toolbar on the left with icons for running, stepping through code, and other debugging actions.

Overloading Methods

- Python does not supports method overloading.
- We may overload the methods but can only use the latest defined method.



Example

```
File Edit View Navigate Code Help
scratch.py x
1: Project
1 # define class
2 class MyClass():
3
4     def sum(self,n1,n2):
5         ans=n1+n2
6         print("ans is : ",ans)
7
8     def sum(self,n1,n2,n3):
9         ans=n1+n2+n3
10        print("ans is : ",ans)
11
12    # object of class
13    p=MyClass()
14
15    # p.sum(10,20)
16
17    p.sum(10,20,30)
Run: scratch x
ans is : 60
Process finished with exit code 0
```


Explanation

- In the previous example, we have defined two sum method, but we can only use the second sum method, as python does not supports method overloading.
- We may define many method of same name and different argument but we can only use the latest defined method.
- Calling the other method will produce an error.



Get Exclusive Video Tutorials



www.apptutorials.com

<https://www.youtube.com/user/Akashtips>





Get More Details



www.akashsir.com



If You Liked It !

Rating Us Now



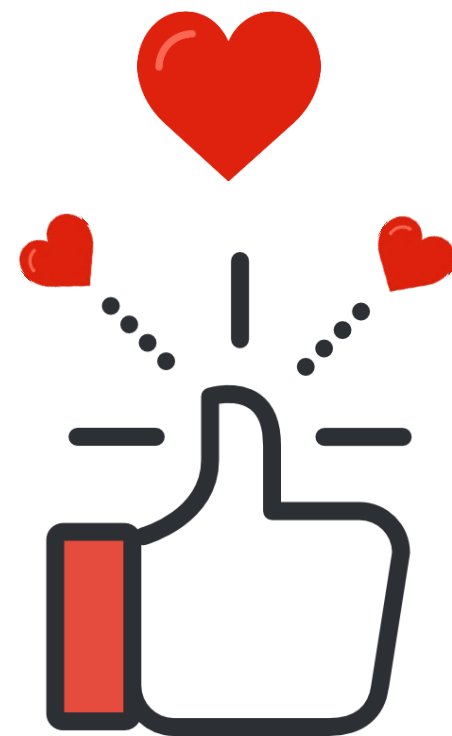
Just Dial

https://www.justdial.com/Ahmedabad/Akash-Technolabs-Navrangpura-Bus-Stop-Navrangpura/079PXX79-XX79-170615221520-S5C4_BZDET



Sulekha

<https://www.sulekha.com/akash-technolabs-navrangpura-ahmedabad-contact-address/ahmedabad>



Connect With Me



Akash Padhiyar
#AkashSir

www.akashsir.com

www.akashtechlabs.com

www.akashpadhiyar.com

www.aptutorials.com

Social Info



Akash.padhiyar



Akashpadhiyar



Akash_padhiyar



+91 99786-21654



#Akashpadhiyar
#aptutorials