**All About “Class Method” in Python**

OOPs in Python — Extension

[[Isha Choudhary](https://medium.com/@shahooda637?source=post_page-----e278d394551b--------------------------------)](https://medium.com/@shahooda637?source=post_page-----e278d394551b--------------------------------)

[Isha Choudhary](https://medium.com/@shahooda637?source=post_page-----e278d394551b--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fa0449bd47c69&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40shahooda637%2Fall-about-class-method-in-python-e278d394551b&user=Isha+Choudhary&userId=a0449bd47c69&source=post_page-a0449bd47c69----e278d394551b---------------------post_header-----------)

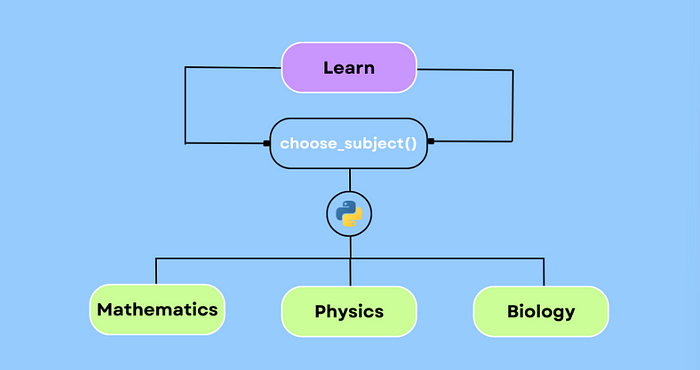
5 min read

·

Sep 15

4

1



We have already discussed about 4 pillars of python OOPs in the article titled “[*The Python OOPs Moments (Part — 01*](https://medium.com/@shahooda637/the-python-oops-moments-part-01-9e16bfd4cc09)*)”* , and about the Decorators in Python in the one titled as*“*[*All About “Decorators” in Python*](https://medium.com/@shahooda637/all-about-decorators-python-e7260adb9ce7)*” .*

Moving forward in the series, we will now discuss another concept in OOPs in Python which is the Class Method.

In Python, a class method is a method that is bound to the class and not the instance of the class (object). Class methods are defined using the “@classmethod” decorator and take the class itself as their first argument, typically named ‘cls’.

Class methods are often used for operations that are related to the class as a whole rather than to a specific instance of the class.

However, this is not the official definition of class methods. The detailed explaination is provided further in the notebook.

**Definition of Class Method**

To define a class method, we use the ‘@classmethod’ decorator, as discussed earlier, within a class definition. The method must take the ‘cls’ argument as its first parameter in order to create the method a class method in addition to using the ‘@classmethod’ decorator.

We can access the class methods directly through class name and can assign them to variables whenever required and that variable is able to access other methods defined in the class.

While defining a class method, we always use ‘cls’ instead of ‘self’ as it would bing the method directly to the class unlike the **\_\_init\_\_** method.

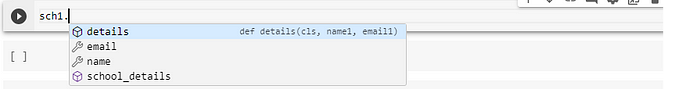
The following cells of code explains the concept through code.

# defining a class  
  
class school:  
  
 # defining the constructor \_\_init\_\_() method of the class to enable the class to take the data  
 def \_\_init\_\_(self, name, email):  
 self.name = name  
 self.email = email  
   
 # defining a class method which overloads the \_\_init\_\_() method  
 @classmethod  
 def details(cls, name1, email1):  
 return cls(name1, email1)  
   
 # defining an instance method inside the class  
 def school\_details(self):  
 print("This School was established in 1950s")

The instance method and the class method both are accessible through the class name and can also be assigned to a variable.



# using the class method to add data into the class through constructor   
  
sch1 = school.details('Central School', 'centralschool@gmail.com')



sch1.email  
  
Output:  
centralschool@gmail.com

sch1.name  
  
Output:  
Central School

sch1.school\_details()  
  
Output:  
This School was established in 1950s

As observed in the code above, we can perform function overloading using the class method in python. An example can be seen in the code above where we overloaded the ***\_\_init\_\_*** function. But, what do we exactly mean by function overloading?

**Function Overloading**

Function overloading is a feature that allows us to define multiple functions with the same name in the same scope, but with different parameter lists. The choice of which function to call is determined by the number or types of arguments passed to the function.

Function overloading is a form of polymorphism, where the different functions with the same name can behave differently based on their parameters.

**Static variable and Class Method**

We can also access a static variable defined inside the class through the class method variable. The example is as below.

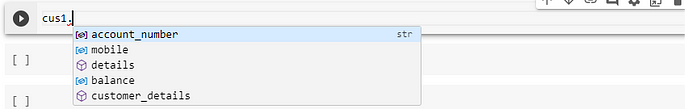
class bank:  
  
 def \_\_init\_\_(self, account\_number, balance):  
 self.account\_number = account\_number  
 self.balance = balance  
   
 mobile = 7089251232  
  
 @classmethod  
 def change\_mobile(cls, mobile\_num):  
 bank.mobile = mobile\_num  
  
 @classmethod  
 def details(cls, account\_number, balance):  
 return cls(account\_number, balance)  
   
 def customer\_details(self):  
 print('account number: ' , self.account\_number)  
 print('balance: ', self.balance)  
 print('Mobile: ', bank.mobile)

Calling the static variable inside the instance method is one way to use the static variable inside a class and through methods. We can also access them through objects and class method variables.

Also, we can use static method to change the value of static method defined inside the class by defining the class method as follows.

@classmethod  
def change\_mobile(cls, mobile\_num):  
 bank.mobile = mobile\_num

cus1 = bank.details('1287655825665', '1153225')



cus1.customer\_details()  
  
Output:  
account number: 1287655825665  
balance: 1153225  
Mobile: 7089251232

cus1.mobile  
  
Output:  
7089251232

cus1.change\_mobile(4535786624)

cus1.mobile  
  
Output:  
4535786624

**Uses and Benefits of Class Methods in OOP in Python**

**01. Accessing and Modifying Class-Level Data:**

Class methods are particularly useful for accessing and modifying class-level data or attributes. They can manipulate or manage data shared among all instances of a class.

**02. Factory Methods:**

Class methods can be used to create and return instances of a class, serving as factory methods. This is helpful when we want to customize the creation of objects, enforce certain constraints, or return cached instances.

**03. Alternate Constructor:**

Class methods can be used to provide an alternative constructors for a class. For example, we can create a class method to initialize instances from a different data format or source.

**04. Singleton Pattern:**

Class methods can be used to implement the Singleton pattern, ensuring that only one instance of a class exists throughout the program.

**05. Namespace Organization:**

Class methods can help organize the namespace by grouping related methods together within a class.

**06. Enhancing Code Readability:**

Class methods can improve code readability by explicitly indicating that a method operates at the class level, making it clear that the method does not depend on instance specific state.

**07. Testing and Mocking:**

In unit testing, class methods can be easier to mock and test because they do not rely on instance- specific state.

**Summary**

In Summary, Class methods in python are valuable for operations related to the class itself, class level data, or providing alternative constructors. The class methods help organize code, improve readibility, and offer felxibility in managing class specific behavior and attributes within an object-oriented programming paradigm.