[Home](https://pynative.com/) » [Python](https://pynative.com/python/) » [Python Object-Oriented Programming (OOP)](https://pynative.com/python/object-oriented-programming/) » Python Class Method Explained With Examples

**Python Class Method Explained With Examples**

Updated on: August 28, 2021 | [3 Comments](https://pynative.com/python-class-method/#llc_comments)

In [Object-oriented programming](https://pynative.com/python/object-oriented-programming/), we use instance methods and class methods. Inside a Class, we can define the following three types of methods.

* [**Instance method**](https://pynative.com/python-instance-methods/): Used to access or modify the object state. If we use [instance variables](https://pynative.com/python-instance-variables/) inside a method, such methods are called instance methods. It must have a self parameter to refer to the current object.
* **Class method**: Used to access or modify the class state. In method implementation, if we use only [class variables](https://pynative.com/python-class-variables/), then such type of methods we should declare as a class method. The class method has a cls parameter which refers to the class.
* [**Static method**](https://pynative.com/python-static-method/): It is a general utility method that performs a task in isolation. Inside this method, we don’t use instance or class variable because this static method doesn’t take any parameters like self and cls.

**Also, read** [Python Class method vs Static method vs Instance method](https://pynative.com/python-class-method-vs-static-method-vs-instance-method/).

**After reading this article, you’ll learn**:

* How to create and use the class methods in Python
* Create class method using the @classmethod decorator and classmethod() function
* how to dynamically add or delete class methods

**Table of contents**

* [What is Class Method in Python](https://pynative.com/python-class-method/#h-what-is-class-method-in-python)
* [Define Class Method](https://pynative.com/python-class-method/#h-define-class-method)
  + [Example 1: Create Class Method Using @classmethod Decorator](https://pynative.com/python-class-method/#h-example-1-create-class-method-using-classmethod-decorator)
  + [Example 2: Create Class Method Using classmethod() function](https://pynative.com/python-class-method/#h-example-2-create-class-method-using-classmethod-function)
  + [Example 3: Access Class Variables in Class Methods](https://pynative.com/python-class-method/#h-example-3-access-class-variables-in-class-methods)
* [Class Method in Inheritance](https://pynative.com/python-class-method/#h-class-method-in-inheritance)
* [Dynamically Add Class Method to a Class](https://pynative.com/python-class-method/#h-dynamically-add-class-method-to-a-class)
* [Dynamically Delete Class Methods](https://pynative.com/python-class-method/#h-dynamically-delete-class-methods)

**What is Class Method in Python**

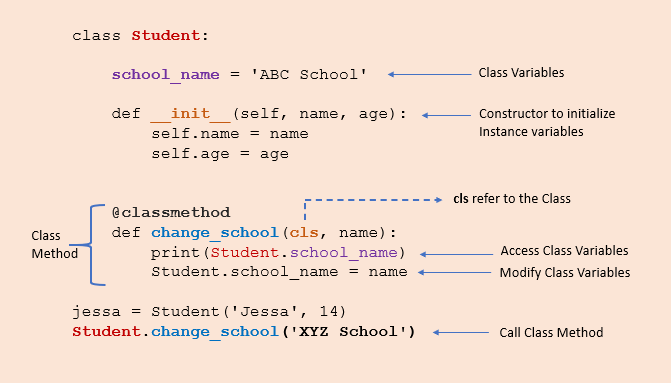
Class methods are methods that are called on the [class](https://pynative.com/python-classes-and-objects/) itself, not on a specific object instance. Therefore, it belongs to a class level, and all class instances share a class method.

* **A class method is bound to the class** and not the object of the class. It can access only class variables.
* It can modify the class state by changing the value of a [class variable](https://pynative.com/python-class-variables/) that would apply across all the class objects.

In method implementation, if we use only class variables, we should declare such methods as class methods. The class method has a cls as the first parameter, which refers to the class.

Class methods are used when we are **dealing with factory methods**. Factory methods are those methods that **return a class object for different use cases**. Thus, factory methods create concrete implementations of a common interface.

The class method can be called using ClassName.method\_name() as well as by using an object of the class.

Define class method

**Define Class Method**

Any method we create in a class will automatically be created as an instance method. We must explicitly tell Python that it is a class method using the @classmethod decorator or classmethod() function.

Class methods are defined inside a class, and it is pretty similar to defining a regular [function](https://pynative.com/python-functions/).

Like, inside an instance method, we use the self keyword to access or modify the instance variables. Same inside the class method, we use the cls keyword as a first parameter to access class variables. Therefore the class method gives us control of changing the class state.

* You may use a [variable](https://pynative.com/python-variables/) named differently for cls, but it is discouraged since self is the recommended convention in Python.
* The class method can only access the class attributes, not the instance attributes

**Example 1: Create Class Method Using @classmethod Decorator**

To make a method as class method, add @classmethod decorator before the method definition, and add cls as the first parameter to the method.

The @classmethod decorator is a built-in function decorator. In Python, we use the @classmethod decorator to declare a method as a class method. The @classmethod decorator is an expression that gets evaluated after our function is defined.

Let’s see how to create a factory method using the class method. In this example, we will create a Student class object using the class method.

**from** datetime **import** date

**class** Student:

**def** \_\_init\_\_(self, name, age):

self.name = name

self.age = age

@classmethod

**def** calculate\_age(cls, name, birth\_year):

# calculate age an set it as a age

# return new object

**return** cls(name, date.today().year - birth\_year)

**def** show(self):

**print**(self.name + "'s age is: " + **str**(self.age))

jessa = Student('Jessa', 20)

jessa.show()

# create new object using the factory method

joy = Student.calculate\_age("Joy", 1995)

joy.show()

**Output**

Jessa's age is: 20  
John's age is: 26

* In the above example, we created two objects, one using the constructor and the second using the calculate\_age() method.
* The [constructor](https://pynative.com/python-constructors/) takes two arguments name and age. On the other hand, class method takes cls, name, and birth\_year and returns a class instance which nothing but a new object.
* The @classmethod decorator is used for converting calculate\_age() method to a class method.
* The calculate\_age() method takes Student class (cls) as a first parameter and returns constructor by calling Student(name, date.today().year - birthYear), which is equivalent to Student(name, age).

**Example 2: Create Class Method Using classmethod() function**

Apart from a decorator, the built-in function classmethod() is used to convert a normal method into a class method. The **classmethod()** is an inbuilt function in Python, which returns a class method for a given function.

**Syntax**:

**classmethod**(function)

* function: It is the name of the method you want to convert as a class method.
* It returns the converted class method.

**Note**: The method you want to convert as a class method must accept class (cls) as the first argument, just like an instance method receives the instance (self).

As we know, the class method is bound to class rather than an object. So we can call the class method both by calling class and object.

A classmethod() function is the older way to create the class method in Python. In a newer version of Python, we should use the @classmethod decorator to create a class method.

**Example**: Create class method using classmethod() function

**class** School:

# class variable

name = 'ABC School'

**def** school\_name(cls):

**print**('School Name is :', cls.name)

# create class method

School.school\_name = **classmethod**(School.school\_name)

# call class method

School.school\_name()

**Output**

School Name is : ABC School

**Example 3: Access Class Variables in Class Methods**

Using the class method, we can only access or modify the [class variables](https://pynative.com/python-class-variables/). Let’s see how to access and modify the class variables in the class method.

Class variables are **shared by all instances of a class**. Using the class method we can modify the class state by changing the value of a class variable that would apply across all the class objects.

**class** Student:

school\_name = 'ABC School'

**def** \_\_init\_\_(self, name, age):

self.name = name

self.age = age

@classmethod

**def** change\_school(cls, school\_name):

# class\_name.class\_variable

cls.school\_name = school\_name

# instance method

**def** show(self):

**print**(self.name, self.age, 'School:', Student.school\_name)

jessa = Student('Jessa', 20)

jessa.show()

# change school\_name

Student.change\_school('XYZ School')

jessa.show()

**Output**:

Jessa 20 School: ABC School  
Jessa 20 School: XYZ School

**Class Method in Inheritance**

In [inheritance](https://pynative.com/python-inheritance/), the class method of a parent class is available to a child class.

Let’s create a Vehicle class that contains a factory class method from\_price() that will return a Vehicle instance from a price. When we call the same method using the child’s class name, it will return the child’s class object.

Whenever we derive a class from a parent class that has a class method then it creates the correct instance of the derived class. The following example shows how the class method works in inheritance.

**Example**

**class** Vehicle:

brand\_name = 'BMW'

**def** \_\_init\_\_(self, name, price):

self.name = name

self.price = price

@classmethod

**def** from\_price(cls, name, price):

# ind\_price = dollar \* 76

# create new Vehicle object

**return** cls(name, (price \* 75))

**def** show(self):

**print**(self.name, self.price)

**class** Car(Vehicle):

**def** average(self, distance, fuel\_used):

mileage = distance / fuel\_used

**print**(self.name, 'Mileage', mileage)

bmw\_us = Car('BMW X5', 65000)

bmw\_us.show()

# class method of parent class is available to child class

# this will return the object of calling class

bmw\_ind = Car.from\_price('BMW X5', 65000)

bmw\_ind.show()

# check type

**print**(**type**(bmw\_ind))

**Output**

BMW X5 65000

BMW X5 4875000

class '\_\_main\_\_.Car'

**Dynamically Add Class Method to a Class**

Typically, we add class methods to a class body when defining a class. However, Python is a dynamic language that allows us to add or delete methods attime. Therefore, it is helpful when you wanted to extend the class functionality without changing its basic structure because many systems use the same structure.

We need to use the classmethod() function to add a new class method to a class.

**Example:**

Let’s see how to add a new class method in the Student class attime.

**class** Student:

school\_name = 'ABC School'

**def** \_\_init\_\_(self, name, age):

self.name = name

self.age = age

**def** show(self):

**print**(self.name, self.age)

# class ended

# method outside class

**def** exercises(cls):

# access class variables

**print**("Below exercises for", cls.school\_name)

# Adding class method attime to class

Student.exercises = **classmethod**(exercises)

jessa = Student("Jessa", 14)

jessa.show()

# call the new method

Student.exercises()

**Output**

Jessa 14

Below exercises for ABC School

**Dynamically Delete Class Methods**

We can dynamically delete the class methods from the class. In Python, there are two ways to do it:

* By using the del operator
* By using delattr() method

**By using the del operator**

The del operator removes the instance method added by class. Use the del class\_name.class\_method syntax to delete the class method.

**Example**:

In this example, we will delete the class method named change\_school() from a Student class. If you try to access it after removing it, you’ll get an Attribute Error.

**class** Student:

school\_name = 'ABC School'

**def** \_\_init\_\_(self, name, age):

self.name = name

self.age = age

@classmethod

**def** change\_school(cls, school\_name):

cls.school\_name = school\_name

jessa = Student('Jessa', 20)

**print**(Student.change\_school('XYZ School'))

**print**(Student.school\_name)

# delete class method

**del** Student.change\_school

# call class method

# it will give error

**print**(Student.change\_school('PQR School'))

**Output**

XYZ School

AttributeError: type object 'Student' has no attribute 'change\_school'

**By using delatt() method**

The delattr() method is used to delete the named attribute and method from the class. The argument to delattr is an object and string. The string must be the name of an attribute or method name.

**Example**

jessa = Student('Jessa', 20)

**print**(Student.change\_school('XYZ School'))

**print**(Student.school\_name)

# delete class method

**delattr**(Student, 'change\_school')

# call class method

# it will give error

**print**(Student.change\_school('PQR School'))