**What is Static Methods in Python**

**A static method is a general utility method that performs a task in isolation**. Static methods in Python are similar to those found in Java or C++.

A static method is bound to the [class](https://pynative.com/python-classes-and-objects/) and not the object of the class. Therefore, we can call it using the class name.

A static method doesn’t have access to the class and [instance variables](https://pynative.com/python-instance-variables/) because it does not receive an implicit first argument like self and cls. Therefore **it cannot modify the state of the object or class**.

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

**class** Employee:

@staticmethod

**def** sample(x):

**print**('Inside static method', x)

# call static method

Employee.sample(10)

# can be called using object

emp = Employee()

emp.sample(10)

**Define Static Method in Python**

Any method we create in a class will automatically be created as an [instance method](https://pynative.com/python-instance-methods/). We must explicitly tell Python that it is a static method using the **@staticmethod decorator** or **staticmethod() function**.

Static methods are defined inside a class, and it is pretty similar to defining a regular [function](https://pynative.com/python-functions/). To declare a static method, use this idiom:

**class** C:

@staticmethod

**def** f(arg1, arg2, ...): ...

**Example: Create Static Method Using @staticmethod Decorator**

To make a method a static method, add @staticmethod decorator before the method definition.

The @staticmethod decorator is a built-in function decorator in Python to declare a method as a static method. It is an expression that gets evaluated after our function is defined.

In this example, we will create a static method gather\_requirement() that accepts the project name and returns all requirements to complete under this project.

Static methods are a special case of methods. Sometimes, you’ll write code that belongs to a class, but that doesn’t use the object itself at all. It is a utility method and doesn’t need an object (self parameter) to complete its operation. So we declare it as a static method. Also, we can call it from another method of a class.

**class** Employee(**object**):

**def** \_\_init\_\_(self, name, salary, project\_name):

self.name = name

self.salary = salary

self.project\_name = project\_name

@staticmethod

**def** gather\_requirement(project\_name):

**if** project\_name == 'ABC Project':

requirement = ['task\_1', 'task\_2', 'task\_3']

**else**:

requirement = ['task\_1']

**return** requirement

# instance method

**def** work(self):

# call static method from instance method

requirement = self.gather\_requirement(self.project\_name)

**for** task **in** requirement:

**print**('Completed', task)

emp = Employee('Kelly', 12000, 'ABC Project')

emp.work()

**Output**:

Completed task\_1

Completed task\_2

Completed task\_3

**Advantages of a Static Method**

Here, the static method has the following advantages

* **Consume Less memory**: Instance methods are object too, and creating them has a cost. Having a static method avoids that. Let’s assume you have ten employee objects and if you create gather\_requirement() as a [instance method](https://pynative.com/python-instance-methods/) then Python have to create a ten copies of this method (seperate for each object) which will consume more memeory. On the other hand static method has only one copy per class.

kelly = Employee('Kelly', 12000, 'ABC Project')

jessa = Employee('Jessa', 7000, 'XYZ Project')

# false

# because seperate copy of instance method is created for each object

**print**(kelly.work **is** jessa.work)

# True

# because only one copy is created

# kelly and jess objects share the same methods

**print**(kelly.gather\_requirement **is** jessa.gather\_requirement)

# True

**print**(kelly.gather\_requirement **is** Employee.gather\_requirement)

* **To Write Utility functions**: Static methods have limited use because they don’t have access to the attributes of an object ([instance variables](https://pynative.com/python-instance-variables/)) and class attributes ([class variables](https://pynative.com/python-class-variables/)). However, they can be helpful in utility such as conversion form one type to another. The parameters provided are enough to operate.
* **Readabiltity**: Seeing the @staticmethod at the top of the method, we know that the method does not depend on the object’s state or the class state.

**The staticmethod() function**

Some code might use the old method of defining a static method, using staticmethod() as a function rather than a decorator.

You should only use staticmethod() function to define static method if you have to support older versions of Python (2.2 and 2.3). Otherwise, it is recommended to use the @staticmethod decorator.

**Syntax**:

**staticmethod**(function)

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

**Example**:

**class** Employee:

**def** sample(x):

**print**('Inside static method', x)

# convert to static method

Employee.sample = **staticmethod**(Employee.sample)

# call static method

Employee.sample(10)

The staticmethod() approach is helpful when you need a reference to a function from a class body and you want to avoid the automatic transformation to the instance method.

**Call Static Method from Another Method**

Let’s see how to call a static method from another static method of the same class. Here we will class a static method from a class method.

**class** Test :

@staticmethod

**def** static\_method\_1():

**print**('static method 1')

@staticmethod

**def** static\_method\_2() :

Test.static\_method\_1()

@classmethod

**def** class\_method\_1(cls) :

cls.static\_method\_2()

# call class method

Test.class\_method\_1()