Python staticmethod()

programiz.com/python-programming/methods/built-in/staticmethod

Join our newsletter for the latest updates.

The staticmethod() built-in function returns a static method for a given function.

The syntax of `staticmethod()` is:

```
staticmethod(function)
```

Using `staticmethod()` is considered a un-Pythonic way of creating a static function.

Hence, in newer versions of Python, you can use the `@staticmethod` decorator.

The syntax of `@staticmethod` is:

```
@staticmethod
def func(args, ...)
```

---

staticmethod() Parameters

The `staticmethod()` method takes a single parameter:

function - function that needs to be converted to a static method

---

Return value from staticmethod()

The `staticmethod()` returns a static method for a function passed as the parameter.

What is a static method?

Static methods, much like class methods, are methods that are bound to a class rather than its object.

They do not require a class instance creation. So, they are not dependent on the state of the object.

The difference between a static method and a class method is:

- Static method knows nothing about the class and just deals with the parameters.
- Class method works with the class since its parameter is always the class itself.

They can be called both by the class and its object.

```
Class.staticmethodFunc()
or even
Class().staticmethodFunc()
```

Example 1: Create a static method using staticmethod()

```
class Mathematics:

    def addNumbers(x, y):
        return x + y

# create addNumbers static method
Mathematics.addNumbers = staticmethod(Mathematics.addNumbers)

print('The sum is:', Mathematics.addNumbers(5, 10))
```

Output

```
The sum is: 15
```

When do you use static methods?

1. Grouping utility function to a class

Static methods have a limited use case because, like class methods or any other methods within a class, they cannot access the properties of the class itself.

However, when you need a utility function that doesn't access any properties of a class but makes sense that it belongs to the class, we use static functions.

Example 2: Create a utility function as a static method

```
class Dates:
    def __init__(self, date):
        self.date = date

    def getDate(self):
        return self.date

    @staticmethod
    def toDashDate(date):
        return date.replace("/", "-")

date = Dates("15-12-2016")
dateFromDB = "15/12/2016"
dateWithDash = Dates.toDashDate(dateFromDB)

if(date.getDate() == dateWithDash):
    print("Equal")
else:
    print("Unequal")
```

Output

```
Equal
```

Here, we have a `Dates` class that only works with dates with dashes. However, in our previous database, all dates were present in slashes.

In order to convert the slash-dates to dash-dates, we have created a utility function `toDashDate` within `Dates`.

It is a static method because it doesn't need to access any properties of `Dates` itself and only requires the parameters.

We can also create `toDashDate` outside the class, but since it works only for dates, it's logical to keep it inside the `Dates` class.

---

2. Having a single implementation

Static methods are used when we don't want subclasses of a class change/override a specific implementation of a method.

Example 3: How inheritance works with static method?

```
class Dates:
    def __init__(self, date):
        self.date = date

    def getDate(self):
        return self.date

    @staticmethod
    def toDashDate(date):
        return date.replace("/", "-")

class DatesWithSlashes(Dates):
    def getDate(self):
        return Dates.toDashDate(self.date)

date = Dates("15-12-2016")
dateFromDB = DatesWithSlashes("15/12/2016")

if(date.getDate() == dateFromDB.getDate()):
    print("Equal")
else:
    print("Unequal")
```

Output

```
Equal
```

Here, we wouldn't want the subclass `DatesWithSlashes` to override the static utility method `toDashDate` because it only has a single use, i.e. change date to dash-dates.

We could easily use the static method to our advantage by overriding `getDate()` method in the subclass so that it works well with the `DatesWithSlashes` class.