Python Methods

Summary: in this tutorial, you'll learn about Python methods and the differences between functions and methods.

Introduction to the Python methods

By definition, a method is a function (https://www.pythontutorial.net/python-basics/python-functions/) that is bound to an instance of a class (https://www.pythontutorial.net/python-oop/python-class/) . This tutorial helps you understand how it works under the hood.

The following defines a `Request` class that contains a function `send()` :

```
class Request:
    def send():
        print('Sent')
```

And you can call the `send()` function via the `Request` class like this:

```
Request.send() # Sent
```

The `send()` is a function object, which is an instance of the `function` class as shown in the following output:

```
print(Request.send)
```

Output:

```
<function Request.send at 0x00000276F9E00310>
```

The type of the `send` is `function`:

```
print(type(Request.send))
```

Output:

```
<class 'function'>
```

The following creates a new instance of the `Request` class:

```
http_request = Request()
```

If you display the `http_request.send`, it'll return a bound method object:

```
print(http_request.send)
```

Output:

```
<bound method Request.send of <__main__.Request object at 0x00000104B6C3D580>>
```

So the `http_request.send` is not a function like `Request.send`. The following checks if the `Request.send` is the same object as `http_request.send`. It'll returns `False` as expected:

```
print(type(Request.send) is type(http_request.send))
```

The reason is that the type of the `Request.send` is `function` while the type of the `http_request.send` is `method`, as shown below:

```
print(type(http_request.send))  # <class 'method'>
print(type(Request.send))  # <class 'function'>
```

So when you define a function inside a class, it's purely a function. However, when you access that function via an object, the function becomes a method.

Therefore, a method is a function that is bound to an instance of a class.

If you call the `send()` function via the `http_request` object, you'll get a `TypeError` as follows:

```
http_request.send()
```

Error:

```
TypeError: send() takes 0 positional arguments but 1 was given
```

Because the `http_request.send` is a method that is bound to the `http_request` object, Python always implicitly passes the object to the method as the first argument.

The following redefines the `Request` class where the `send` function accepts a list of arguments:

```
class Request:
    def send(*args):
        print('Sent', args)
```

The following calls the `send` function from the `Request` class:

```
Request.send()
```

Output:

```
Sent ()
```

The `send()` function doesn't receive any arguments.

However, if you call the `send()` function from an instance of the `Request` class, the `args` is not empty:

```
http_request.send()
```

Output:

```
Sent (<__main__.Request object at 0x000001374AF4D580>,)
```

In this case, the `send()` method receives an object which is the `http_request`, which is the object that it is bound to.

The following illustrates that the object that calls the `send()` method is the one that Python implicitly passes into the method as the first argument:

```
print(hex(id(http_request)))
http_request.send()
```

Output:

```
0x1ee3a74d580
Sent (<__main__.Request object at 0x000001EE3A74D580>,)
```

The `http_request` object is the same as the one Python passes to the `send()` method as the first argument because they have the same memory address. In other words, you can access the instance of the class as the first argument inside the `send()` method:

The following method call:

```
http_request.send()
```

is equivalent to the following function call:

```
Request.send(http_request)
```

For this reason, a method of an object always has the object as the first argument. By convention, it is called `self`:

```
class Request:
    def send(self):
        print('Sent', self)
```

If you have worked with other programming languages such as Java or C#, the `self` is the same as the `this` object.

Summary

- When you define a function inside a class, it's purely a function. However, when you call the function via an instance of a class, the function becomes a method. Therefore, a method is a function that is bound to an instance of a class.
- A method is an instance of the `method` class.
- A method has the first argument (`self`) as the object to which it is bound.
- Python automatically passes the bound object to the method as the first argument. By convention, its name is `self`.