Theory: Methods

23 minutes 8 / 9 problems solved

Start practicing

If attributes define the data that the objects of a particular class have, the **methods** define their behavior. Python has several types of methods that you can create within a class but, in this topic, we will focus on the instance methods.

§1. Method syntax

Methods define the functionality of the objects that belong to the particular class. The basic syntax looks like this:

```
# basic method syntax
class MyClass:
    # the constructor
    def __init__(self, arg1):
        self.att = arg1

    # custom method
    def do_smt(self):
        # does something
```

You can see that declaring a method resembles declaring a function: we have keyword `def` followed by the name of the method. The parameters of the method are written inside the parentheses.

The first parameter of the method should always be `self`. You may remember that `self` represents the particular instance of the class. When it comes to instance methods, the first parameter that is passed to the method is the instance that called it. Let's create an instance of **MyClass** and see how this works:

```
my_object = MyClass(some_value)
# calling the instance method
my_object.do_smt()
# my_object does something
```

In this example, the **my_object** instance is passed *implicitly* so we do not write the parameter in the code. We can, however, pass the instance explicitly:

```
MyClass.do_smt(my_object)
# my_object does the same thing
```

These examples clearly illustrate why `self` has to be the first argument of the instance methods. If you want your method to have other parameters, just write them after the `self` keyword!

§2. Methods vs functions

Though they are quite similar, Python does make a distinction between methods and functions. To quote the official documentation, *"a method is a function that 'belongs to' an object."* Since we're interested in OOP, we'll specifically be looking at methods associated with class instances.

Let's consider an example:

Current topic:
✓ Methods 3★ (Stage 1)

Topic depends on:
✓ String formatting 7★ (Stage 1)
✓ Class instances 3★ (Stage 1)

Topic is required for:
✓ Methods and attributes 3★
Docstrings
XML in Python
Unit testing in Python
Processing requests (Stage 1)
Text normalization

```
# class and its methods
class Ship:
    def __init__(self, name, capacity):
        self.name = name
        self.capacity = capacity
        self.cargo = 0

    def sail(self):
        print("{} has sailed!".format(self.name))




# function

def sail_function(name):

    print("{} has sailed!".format(name))
```

What is of interest to us here is the method `sail` of the class `Ship` and the function `sail_function`. Let's call them:

```
# creating an instance of the class Ship
# and calling the method sail
black_pearl = Ship("Black Pearl", 800)
black_pearl.sail()
# prints "Black Pearl has sailed!"


# calling the function sail_function
sail_function(black_pearl.name)

# also prints "Black Pearl has sailed!"
```

The way that we've defined them, both our method and our function produce the same results but in a different way. A method is connected to an object of the class, it is not independent the way a function is. Sure they are both called by their names, but to call a method we need to invoke the class that this method belongs to.

§3. Return

So far the method hasn't returned any values since we only used the `print()` function. Obviously, just as with functions, we can define what type of data the method can return with the `return` statement. For example, let's create a method that calculates how many kilograms of cargo the ship has (initially, the weight of the cargo is given in tons):

```
class Ship:
    # other methods

    def convert_cargo(self):
        return self.cargo * 1000
```

The method is simple: it converts the tons into kilograms (by multiplying it by 1000) and then returns the calculated value. If we were to call it, we wouldn't get any messages unless we explicitly printed the result of the function:

```
print(black_pearl.convert_cargo())  # 0
```

Since we haven't changed the default value of the `cargo` attribute, the method would return 0 multiplied by 1000, which is also 0.

§4. Summary

Methods within classes specify the behavior of class or its objects. They are similar to functions with the exception that they are strongly connected to the class and cannot be called independently from it or its instances.

Table of contents:

The first parameter of instance methods is the keyword `self` that represents the particular instance of the class. That particular instance of the class is the first argument that is passed to the method. Methods can return values or simply print messages (i.e. return nothing).

Methods allow you to add any functionality to your classes. This is how you can manipulate your objects and create complex programs, so we encourage you to explore methods in your projects!

Report a typo

469 users liked this theory. **16** didn't like it. **What about you?**

Start practicing

Comments (12) Hints (0) Useful links (0) Show discussion