

Theory: Methods and attributes

🕒 39 minutes

8 / 8 problems solved

Start practicing

6095 users solved this topic. Latest completion was about 3 hours ago.

Now that you’ve learned how to create instance methods let’s go even further and learn to use the methods for creating and modifying attributes.

§1. Creating attributes with methods

Instance attributes are the ones defined within methods so by definition we can create new attributes inside our custom methods. Let’s take the class `Ship` as an example.

```
1 class Ship:
2     def __init__(self, name, capacity):
3         self.name = name
4         self.capacity = capacity
5         self.cargo = 0
```

Every ship needs a captain so let’s define a method called `name_captain` for those purposes:

```
1 class Ship:
2     # constructor
3     # ...
4
5     def name_captain(self, cap):
6         self.captain = cap
7         print("{} is the captain of the {}".format(self.captain, self.name))
```

When called for the first time, the `name_captain` method creates a new attribute called `captain` and prints the corresponding message. When used next, it just changes the value of the `self.captain` attribute (and prints the message as well).

To see how it would work, let’s create the ship “Black Pearl”:

```
1 black_pearl = Ship("Black Pearl", 800)
```

If we tried to print the value of the `captain` attribute now, we would get an error:

```
1 print(black_pearl.captain) # AttributeError
```

This is because this attribute is only created within the `name_captain` method. If we call it, we will be able to access the attribute `captain`:

```
1 black_pearl.name_captain("Jack Sparrow")
2 # prints "Jack Sparrow is the captain of the Black Pearl"
3
4 print(black_pearl.captain) # "Jack Sparrow"
```

Note that only those instances that have called this method, will have the `captain` attribute. It’s an important thing to remember! You may get an error if you try to use the attribute and the method hasn’t been called yet.

To avoid these problems, it is recommended to define all possible attributes in the `__init__`. This can not only help you avoid `AttributeError`, but also gives a good understanding of the class and its objects from the get-go. If you do want to create the value for the attribute in a special instance method, then list it in the `__init__` as `None`:

Current topic:

✓ [Methods and attributes](#)

3★ ...

Topic depends on:

✓ [Else statement](#)

15★ ...

✓ [Methods](#)

3★ ...

Topic is required for:

✓ [Magic methods](#)

3★ ...

Table of contents:

[1 Methods and attributes](#)

[§1. Creating attributes with methods](#)

[§2. Modifying attributes with methods](#)

[§3. Summary](#)

[Feedback & Comments](#)

```
1 class Ship:
2     def __init__(self, name, capacity):
3         self.name = name
4         self.capacity = capacity
5         self.cargo = 0
6         self.captain = None
```

Then, in the specific method, you simply modify the default value which is what we'll consider in the next section.

§2. Modifying attributes with methods

Methods can also be used to modify the instance attributes. Take the methods `load_cargo` and `unload_cargo` for example:

```
1 class Ship:
2     def __init__(self, name, capacity):
3         self.name = name
4         self.capacity = capacity
5         self.cargo = 0
6
7     def load_cargo(self, weight):
8         if self.cargo + weight <= self.capacity:
9             self.cargo += weight
10
11         print("Loaded {} tons".format(weight))
12
13     else:
14
15         print("Cannot load that much")
16
17
18     def unload_cargo(self, weight):
19
20         if self.cargo - weight >= 0:
21
22             self.cargo -= weight
23
24             print("Unloaded {} tons".format(weight))
25
26     else:
27
28         print("Cannot unload that much")
29
```

Both these methods are supposed to change the value of the attribute `cargo` if those changes are possible. The `load_cargo` method first checks that the loading of a particular weight will not exceed the capacity of the ship and the `unload_cargo` checks that the unloading will not make the weight of the cargo negative. Then they both make the changes or print a message that those changes are impossible.

```
1 # example
2 black_pearl.load_cargo(600)
3 # "Loaded 600 tons"
4
5 black_pearl.unload_cargo(400)
6 # "Unloaded 400 tons"
7
8 black_pearl.load_cargo(700)
9 # "Cannot load that much"
10
11
12
13 black_pearl.unload_cargo(300)
14
15 # "Cannot unload that much"
16
```

If we wanted to print out the value of `cargo` after all these manipulations, we would see that it would equal 200 (tons). Because of the restrictions that we placed, only the first callings of `load_cargo` and `unload_cargo` made changes to the attribute `cargo`.

So far our methods haven't been returning any values since we only used the `print()` function, but we can make our methods return any type of value that we want. For example, let's create a method that calculates how much more cargo can our ship load.

```
1 class Ship:
2     # all other methods
3
4     def free_space(self):
5         return self.capacity - self.cargo
```

If we were to call this method on our Black Pearl we wouldn't get any messages, because the method doesn't print anything. But instead, we could use the value it returns in our further calculations. We could, for instance, rewrite the `load_cargo` method by using the `free_space` method:

```
1 class Ship:
2     # updated load_cargo method
3     def load_cargo(self, weight):
4         if weight <= self.free_space():
5             self.cargo += weight
6             print("Loaded {} tons".format(weight))
7         else:
8             print("Cannot load that much")
```

In this example, we called a method inside another method and used the values in our calculations. Again, we used `self` to make sure that we only deal with the particular instance of the class `Ship` and that all calculations concern this instance.


§3. Summary

In this topic, we focused on more advanced uses of instance methods in Python.

Methods can be used for creating new attributes and modifying existing ones. You can call methods inside other methods, use the results for calculations or just output messages. Knowing how methods and attributes interact can help you expand the functionality of your classes and make your programs very efficient.

We hope that you'll experiment with instance methods and use them in your projects!

 Report a typo

 Thanks for your feedback!

Start practicing

[Comments \(6\)](#)

[Hints \(0\)](#)

[Useful links \(0\)](#)

[Show discussion](#)