Object Methods

by Sophia

WHAT'S COVERED

In this lesson, we will explore how to create and use an object's method in a larger program. Specifically, this lesson covers:

1. Creating and Using Methods Without Parameters

We'll start by using the class with the attributes that we already had in place from the prior lesson as a starting point:

EXAMPLE

```
import datetime

class User:
  def __init__(self, uname, pword):
    self.username = uname
    self.password = pword

    self.activeUser = True
    self.numOfLogins = 0
    self.dateJoined = datetime.date.today()
```

Remember, a class can have any number of attributes and methods. We already covered how we can set and change attributes, which are properties like facts. We can also define methods, other than the initializing `__init__` method, that state behaviors of an object rather than the facts about the object. For example, the behaviors for a car would be that it can start up, move, turn, and stop. Or like our example at the beginning of this lesson, our dog can bark and fetch. A method is really just a function. However, the key difference between a function and a method is that a method is associated with a specific class and with each object (instance) that we create from that class.

For us to define what the methods are in the class, we use the same syntax as defining a function.

```
def <method_name>(self, <parameter(s)>):
```

As a best practice, we should use all lower cases with no spaces in the method name. Note that similar to the —init— method, we need to use `self` as the first parameter to indicate that it's referencing the object that was created. We can also pass in additional parameters after `self` but they are not required.

New method: show_num_logins()

Let's go ahead and create a method that we will name `show_num_logins()` which will show the username and let us know how many times they have logged in. To define this method, it will look like the following.

EXAMPLE

```
  #display number of logins
  def show_num_logins(self):
    return self.username + " logged in " + str(self.numOfLogins) + " times."
```

In our definition line we see the `def` keyword to define this method. Next, our method name is followed by the `self` parameter inside the parentheses. Again, remember from the past lessons that this first parameter is typically called `self`. This `self` is a variable that represents the instance of the class (the object that's being created at that moment.). Even though we are not expecting to pass any arguments, we need this first parameter so the instance call is bound to this class's method. In the next section, we will create a method with parameters. The last item needed to complete the creation of our method is the colon (:).

This method is part of the `User` class so now our entire class would look like the following.

```
import datetime
class User:
  def __init__(self, uname, pword):
    self.username = uname
    self.password = pword

    self.activeUser = True
    self.numOfLogins = 0
    self.dateJoined = datetime.date.today()

  #display number of logins
  def show_num_logins(self):
    return self.username + " logged in " + str(self.numOfLogins) + " times."
```

TRY IT

Directions: Go ahead and add this new method to your `User` class in the IDE. We will continue to use this method in this lesson.

Notice that the indentation is the same as the definition of the `__init__` method that we had created previously. In the body of our `show_num_logins()` method, the indentation is again to associate it as part of the method; we are returning a long string that has the `username` concatenated with the string “logged in “, then concatenated with the `numOfLogins`, and then finally the string “times.” Two items to note:

- The `username` and the `numOfLogins` have the prefix of `self`. Since they are referencing the existing object’s attributes, this is necessary for it to work correctly.
- Since the output will be a concatenated string, we need to convert the numeric `numOfLogins` value to a string using the `str()` function.

Now, for us to call this method from code, we would use this syntax:

```
<object_name>.<method_name()>
```
In this case, we replace the `<object_name>` with the actual name of the object we’re referring to and the `<method_name>` would be changed to the name of the method. Being that these must be used from an instance, we will need to create the instance first. We will use the same instance that we used previously in the last lesson account.

```
import datetime
class User:
  def __init__(self, uname, pword):
    self.username = uname
    self.password = pword

    self.activeUser = True
    self.numOfLogins = 0
    self.dateJoined = datetime.date.today()

  #display number of logins
  def show_num_logins(self):
    return self.username + " logged in " + str(self.numOfLogins) + " times."

account = User('sophia','mypass') #creating the instance from class User
print(account.show_num_logins()) #here is our method call
```
Now, with the instance/object code added, we’ll be able to execute our account object and the `show_num_logins()` method. Notice that we have to place it within the `print()` function since the `show_num_logins()` method returns a string.

```
sophia logged in 0 times.
```

TRY IT

Directions: Go ahead and add this new instance call to your program. Try running the program and see if you get the output.

New method: logged_in()

Let’s now create a method that increases the number of times that the user logs in one at a time. Let's call it `logged_in()`.

```
  #increase number of logins
  def logged_in(self):
    self.numOfLogins = self.numOfLogins + 1
```

As part of this method, we will be incrementing `numOfLogins` by 1 for the object. The entire class will look like:

```
import datetime
class User:
  def __init__(self, uname, pword):
    self.username = uname
    self.password = pword

    self.activeUser = True
    self.numOfLogins = 0
    self.dateJoined = datetime.date.today()

  #display number of logins
  def show_num_logins(self):
    return self.username + " logged in " + str(self.numOfLogins) + " times."

  #increase number of logins
  def logged_in(self):
    self.numOfLogins = self.numOfLogins + 1
```

TRY IT

Directions: Go ahead and add this new method to our `User` class. We will use this new method in the next section.

Given what we have, we typically won’t be calling the `logged_in()` method directly except for testing purposes. This is because once we log in, the `logged_in()` method should automatically be called after the user has correctly entered in the username and password. We can verify that it works correctly by calling the method a couple of times after the account object is created and then calling the code>show_num_logins() method afterward.

```
import datetime
class User:
  def __init__(self, uname, pword):
    self.username = uname
    self.password = pword

    self.activeUser = True
    self.numOfLogins = 0
    self.dateJoined = datetime.date.today()

  #display number of logins
  def show_num_logins(self):
    return self.username + " logged in " + str(self.numOfLogins) + " times."

  #increase number of logins
  def logged_in(self):
    self.numOfLogins = self.numOfLogins + 1

account = User('sophia','mypass')
account.logged_in()
account.logged_in()
print(account.show_num_logins())
account.logged_in()
print(account.show_num_logins())
```

Let’s follow the program code below the `User` class. First we initialized the `account` sending our arguments of `sophia` and `mypass`. We then call the `logged_in()` method twice, and use the `print()` function to show the value of the `show_num_logins()` method. Then we call the `logged_in()` method again and use the `print()` function to show the `show_num_logins()` method’s value. The results show the following.

```
sophia logged in 2 times.
sophia logged in 3 times.
```

TRY IT

Directions: Go ahead and add the program calls. Did you get the same output?

2. Using Methods With Parameters

We can also pass data into methods in the same way that we do functions by using parameter names inside of the parentheses. The `self` parameter is always the first parameter after the method name; however, we never pass data to the `self` parameter. This variable binds the class methods to the arguments that it is receiving from this new instance. Let's go ahead and create a method that passes in the username and password to login. If they are valid to the object that was created, we'll output an indication that the login was successful and call the `logged_in()` method to increase the `numOfLogins` value. If it was not successful, we'll output an error message.

```
  #logging into the user account
  def login(self, uname, pword):
    if (self.username == uname and self.password == pword):
      print("Login successful")
      self.logged_in()
    else:
      print("Incorrect username and password combination.")
```

Within the entire program, it will look like the following:

```
import datetime
class User:
  def __init__(self, uname, pword):
    self.username = uname
    self.password = pword

    self.activeUser = True
    self.numOfLogins = 0
    self.dateJoined = datetime.date.today()

  #display number of logins
  def show_num_logins(self):
    return self.username + " logged in " + str(self.numOfLogins) + " times."

  #increase number of logins
  def logged_in(self):
    self.numOfLogins = self.numOfLogins + 1

  #logging into the user account
  def login(self, uname, pword):
    if (self.username == uname and self.password == pword):
      print("Login successful")
      self.logged_in()
    else:
      print("Incorrect username and password combination.")
```

Let's break down what this method is doing in detail. First we define our new method and add the parameters `uname` and `pword` in addition to `self`. Note our parameter names are the same as they were in our `__init__` method to be able to use them within the method. However, these names are just used within this method and are not attributes of the class. Then we have a conditional statement within the login method's body that looks to see if the arguments passed into it are equal to the variable values of `username` and `password` which were assigned from the __init__ method. If they are the same, the `print()` function will output "Login successful" and call the `logged_in` method to add 1 to the `numOfLogins` variable. If the arguments passed are not equal then the `else` statement will trigger the `print()` function to output "Incorrect username and password combination".

TRY IT

Directions: Go ahead and add this new method with parameters to your `User` class. We will use this new method with the next calls to the `User` class.

Let's test this code by calling the `login()` method using the correct username and password.

```
account = User('sophia','mypass')
account.login('sophia','mypass')
print(account.show_num_logins())
```

Again the first line is initializing the `account` and sending the arguments of `sophia` and `mypass`. We then call the `login()` method, this time adding arguments of `sophia` and `mypass` for the method's parameters. Then we use the `print()` function to show the `show_num_logins()` method's value.

```
Login successful
sophia logged in 1 times.
```

The results are as expected; since the username and password were correct, we were able to see that the user logged in correctly.

TRY IT

Directions: Go ahead and add the program calls. Did you get the same successful output?

What if we had entered the wrong username or password? Let's see what the results of that look like.

```
account = User('sophia','mypass')
account.login('sophia','wrongpassword')
print(account.show_num_logins())
```

That new password argument did not match up.

```
Incorrect username and password combination.
sophia logged in 0 times.
```

TRY IT

Directions: Your turn. Try changing the `login()` method's arguments to something other than what the __ `init`__ method set them up as.

What happens if the user had multiple login attempts with some that were successful and some that were not? Looking at the following code, how many times do you think the user should have logged in correctly?

```
account = User('sophia','mypass')

account.login('sophia','mypass')
account.login('sophia','wrongpassword')
account.login('sophia','mypass')
account.login('sophia','mypass')
account.login('sophia','wrongpassword')

print(account.show_num_logins())
```

TRY IT

Directions: Try adding these new program calls and see if you can determine what the expected output will be.

```
Login successful
Incorrect username and password combination.
Login successful
Login successful
Incorrect username and password combination.
sophia logged in 3 times.
```

In looking at the results, were they what you expected to see?

SUMMARY

In this lesson, we learned how to **create methods with and without parameters**. We learned that for methods, we need to use the `self` parameter as the first method parameter to ensure that it reflects the instance calling the class. We also discovered that any variables declared in the class's method(s) will need the prefix self in order to use those method(s) by the object calling it. Lastly, we had a chance to call the methods of a class.

Best of luck in your learning!

Source: THIS CONTENT AND SUPPLEMENTAL MATERIAL HAS BEEN ADAPTED FROM "PYTHON FOR EVERYBODY" BY DR. CHARLES R. SEVERANCE ACCESS FOR FREE AT www.py4e.com/html3/ LICENSE: