Object Attributes

by Sophia

WHAT'S COVERED

In this lesson, we will explore how to set and change values of object attributes and create default values. Specifically, this lesson covers:

1. Setting and Changing Attribute Values

2. Setting Default Attribute Values

1. Setting and Changing Attribute Values

In the prior lesson, we created a `User` class with two attributes/properties. Once we used the `__init__` method, we didn't make any changes to the attributes directly. Using the `__init__` method, we initially set the values of the attributes. Let's revisit that class.

EXAMPLE

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

TRY IT

Directions: If you haven't already, go ahead and add the `User` class to the IDE.

Continuing to use the example from the last lesson, we created `account` as the instance of `User` class. Here is that full code again from the previous lesson.

EXAMPLE

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

```
account = User('sophia','mypass') #here is the instance created from the User class

print(account)
print(account.username)
print(account.password)
print(type(account))
```

TRY IT

Directions: Add the instance and the `print()` functions and run the program.

```
<__main__.User object at 0x7fca2528cf70>
sophia
mypass
<class '__main__.User'>
```

Remember, with our new instance `account`, we are passing the arguments of `sophia` as the username and `mypass` as the password. This instance assigns these values to the parameters set in the attributes of the `User` class. We just set the attribute values right here.

Changing attribute values

THINK ABOUT IT

If you remember back to us using sets and tuples, we defined initial values when we first defined the data collection type.

Recall that sets and tuples are immutable, meaning that once they are defined, the program can’t change anything about their values. However, with classes that we define, we can change the value of any attribute as long as the class allows it. We will explore the prevention of attributes being set in a later lesson.

The syntax to do so would look like this:

```
<objectName>.<attributeName> = value
```

Doesn’t that look familiar already? It should! Let’s take a look at how we can make use of this after we’ve already created an instance of the `User` class.

In our example line of code above, we can replace “`<objectName>`” with the name of the object, which is `account`. We can also replace the `attributeName` with the name of the attribute whose value we want to change. So, let’s say that we want to change the value of the username (which is currently `sophia`) to `python`. We can do so by adding one line of code to our prior example.

EXAMPLE

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

account = User('sophia','mypass')
print(account.username)
print(account.password)
account.username = 'python'
print(account.username)
print(account.password)
```

Here is that output.

```
sophia
mypass
python
mypass
```

TRY IT

Directions: Add the changes to the code and try running it.

Let’s follow the code. First we initialized the `account`, then we output the username (`sophia`) and password (`mypass`). We then changed the value of the username attribute to `python` and output each value again. We can see that the username has actually changed. The password hasn’t changed but that’s to be expected as we didn’t do anything to it in our code yet.

2. Setting Default Attribute Values

Up to this point, we’ve set all variables through their parameters. However, we don’t have to pass in the value for every attribute when we create a new instance of an object. If we want to always set a default value when an object is created, we can just use the following in the `__init__` method.

```
self.<attributeName> = value
```

Remember, the `__init__` method is the first thing a class does when called to create an instance; the `__init__` method initializes everything.

So, using this line of code, the “`<attributeName>`” is the attribute’s name that we want to set and the value is the specific data that we want to set. Typically for most cases, this would just be initializing numeric attributes to 0 or strings to an empty string.

```
self.<attributeNameNum> = 0
self.<attributeNameString> = ""
```

Let’s expand on our `User` class to see how we can use default values.

Perhaps now we want to:

- Determine if the user’s account is active
- Track the number of times that a user has logged in
- Determine when the user joined

This gives us three more attributes that we want to set. However, in all of these cases, there’s no need for us to manually pass these values in, since:

- The account should automatically have the activity flag set to True.
- The number of times that a user has logged in should be set to 0.
- When a user is created, the created date should simply be set to the current date.

Our last attribute deals with time. Anytime that we will do anything with dates and times, we have to import the `datetime` module.

KEY CONCEPT

If you can remember back to some previous examples we used in Unit 1 and 2, we imported a module called `math` to do some calculations for us in our code. Here we need to import the `datetime` module to obtain this attribute’s value. We will go into more details of importing modules in a later lesson. Just know for now that importing of modules allows you to incorporate additional functionality outside of what’s available by default in Python. To import the module, we’ll use the `import` command at the top of the program before the class is defined. The module we need this time is the `datetime` one.

EXAMPLE

```
import datetime
```

Next, we can add code to the `_init_` method that initializes our new variables and the default values we want.

EXAMPLE

```
import datetime

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

    self.activeUser = True #new variable with True as default value
    self.numOfLogins = 0 #new variable with 0 as default value
    self.dateJoined = datetime.date.today() #new variable using the datetime module
```

TRY IT

Directions: In the IDE, add the new variable and default values to your User class.

Setting the values for the new variables activeUser and numOfLogins attributes should be familiar. We are simply setting the activeUser variable to True for every instance of the User class that is created and setting the variable numOfLogins to 0 by default.

The dateJoined attribute is using the datetime module to get today's date. It is set using the .today() method of the date class within the datetime module. Again, these are all objects and methods within the datetime module. Once we import the module, we have access to use anything that module has prebuilt in it.

TRY IT

Directions: Let's add an output for each of these new attributes to our program and run it.

EXAMPLE

```
account = User('sophia','mypass')
print(account.username)
print(account.password)
print(account.activeUser)
print(account.numOfLogins)
print(account.dateJoined)
```

If we output each attribute of the account instance, this is what we should see:

```
sophia
mypass
True
0
2022-02-24
```

Notice that in creating our instance, nothing has changed as we're just passing in the username and password. All of the other attributes are automatically set by the __init__ method which is called when we create an instance.

From there, though, we can change the value from that default value in the same way that we would change any other attribute. For example, if we want to disable the account and set the variable activeUser to False, we can do so after we've created the object or anytime after that.

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()

account = User('sophia','mypass')
account.activeUser = False
print(account.username)
print(account.password)
print(account.activeUser)
print(account.numOfLogins)
print(account.dateJoined)
```

TRY IT

Directions: Try making the change to flag the user is no longer active and see if you get the same output below. Try making some other changes.

```
sophia
mypass
False
0
2022-02-24
```

SUMMARY

In this lesson, we learned how to **set values to attributes** and **change** those values after the instances have been created. We also learned how to **set default values for attributes** and change them afterwards if needed. We also quickly looked at importing the `datetime` module so that we can set the default value using the current date. Importing will be the topic of a future lesson.

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: CREATIVE COMMONS ATTRIBUTION 3.0 UNPORTED.