The Employee Class Program

by Sophia

WHAT'S COVERED

In this lesson, we will explore the creation of a completed class that contains attributes and methods. Specifically, this lesson covers:

1. The Employee Class

Since Python is one of the computer languages that follows the object-oriented programming (OOP) model, you've basically been working with classes and objects since the beginning of all of the tutorials. When it comes to classes and objects, it's important to note that an object should have everything about itself encapsulated in a class. We have looked at some examples already but we want to make sure that we have defined the `__init__` method, attributes, and the methods associated with the class.

Let's first start by looking at an employee of an organization. For an employee, we should think about what attributes we may need to describe them. We would have their first name, last name, title, salary, hire date, and employee ID as their basic attributes.

Let's start to define our class with the attributes within the `__init__` method. We will need to have each of those instance variables (first name, last name, etc.) set as part of the parameters of the `__init__` method. The only item that won't be passed in will be the hire date, which will use the current day's date for when the object representing the employee is created. Remember, we need to import the `datetime` module for that.

```
import datetime
class Employee:
  def __init__(self, fname, lname, empid, title, sal):
    self.firstname = fname
    self.lastname = lname
    self.employeeid = empid
    self.jobtitle = title
    self.salary = sal
    self.hiredate = datetime.date.today()
```

TRY IT

Directions: Create our `Employee` class with the `__init__` method parameters and attributes. Also, import the `datetime` module and create the `hiredate` attribute.

This gives us a starting point with the employee's details. Notice that we needed to import the `datetime` module to make use of the `datetime.date.today()` method which gives us the current date. Without the import of that module, we would not be able to call that method. Once we have attributes set, we typically won't be accessing these variables directly. Rather, we'll use additional methods to update and access these variables so we can add some error checking on the object's variables.

For example, for the first name, returning it may not have anything special or unique to check, but when we try to set the first name, we'll make sure that the length of the value that's being passed in isn't empty. So let's add a method called `get_firstname()` that returns the `firstname` variable when it is called by the new instance. And we will create another method called `set_firstname` with an additional parameter of `fname` to check that the argument passed is not empty. In order to use it, we must call it directly to set the attribute. This would be done after the object has been defined and the `__init__` method has set the values already. Using this method, we'll be able to update the attribute value afterward.

```
  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
    if len(fname) > 0:
      self.firstname = fname
```

TRY IT

Directions: Go ahead and add these two new methods.

This of course doesn't check if the value passed into the `__init__` method for the first name was empty, meaning there was no argument for this parameter added during the initial instance call. We would need to handle that separately.

The last name and job title can use the same format for their respective methods to set and get those values.

```
  #returns last name
  def get_lastname(self):
    return self.lastname

  #sets lastname if lname isn't an empty string
  def set_lastname(self,lname):
    if len(lname) > 0:
      self.lastname = lname

  #returns job title
  def get_jobtitle(self):
    return self.jobtitle

  #sets job title if job title isn't an empty string
  def set_jobtitle(self,title):
    if len(title) > 0:
      self.jobtitle = title
```

TRY IT

Directions: Go ahead and add the four new methods for `lastname` and `jobtitle` get and set.

Let’s see if you have what is currently created so far.

```
import datetime
class Employee:
  def __init__(self, fname, lname, empid, title, sal):
    self.firstname = fname
    self.lastname = lname
    self.employeeid = empid
    self.jobtitle = title
    self.salary = sal

    self.hiredate = datetime.date.today()

  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
    if len(fname) > 0:
      self.firstname = fname

  #returns last name
  def get_lastname(self):
    return self.lastname
  #sets lastname if lname isn't an empty string
  def set_lastname(self,lname):
    if len(lname) > 0:
      self.lastname = lname

  #returns job title
  def get_jobtitle(self):
    return self.jobtitle

  #sets job title if job title isn't an empty string
  def set_jobtitle(self,title):
    if len(title) > 0:
      self.jobtitle = title
```

If your class looks different than what is displayed above, check line by line to see what may be different. Make sure it looks identical before moving on.

Now we have to set up the `salary` and `employeeid`.

For the `employeeid`, we will not permit the value to be updated (once an employee has an employee id, it is typically set) so we only need to create the get method to return the value. We will not need a set method for this variable. For this get method, we'll concatenate a string in front of it. Since `employeeid` will likely be a number, we will need to convert it to string using the `str()` function first to allow the string concatenation.

```
  #return employee id
  def get_employeeid(self):
    return "Employee ID: " + str(self.employeeid)
```

TRY IT

Directions: Add the `get_employeeid()` method to your class.

For the salary, we will permit any value that's larger than 0 when we set it. When we return it, we'll want to format it as currency. As such, we'll use the `.format()` method to make the conversion. The `.format()` method takes in a parameter to format it but it can be a bit confusing with the way that it takes in the data with its parameters. The `.2f` specifically stands for a conversion of the number to two decimal places as a `float`. The ":," (colon followed by a comma) ensures that you have a comma every third digit and we precede this with a $ since this is currency.

KEY CONCEPT

There are many formatting types that are included in Python. We have used a few in previous tutorials and now during this program. If you want to see a listing of the formatting types here is a good reference:

www.w3schools.com/python/ref_string_format.asp

```
  #returns salary
  def get_salary(self):
    return "${:,.2f}".format(self.salary)

  #sets salary if salary isn't an empty string
  def set_salary(self,sal):
    if sal > 0:
      self.salary = sal
```

TRY IT

Directions: Now add the two new methods for salary, the `get` method (for returning), and the `set` method to make sure the value is greater than 0 (not empty).

Altogether our code should look like the following.

```
import datetime
class Employee:
  def __init__(self, fname, lname, empid, title, sal):
    self.firstname = fname
    self.lastname = lname
    self.employeeid = empid
    self.jobtitle = title
    self.salary = sal

    self.hiredate = datetime.date.today()

  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
    if len(fname) > 0:
      self.firstname = fname

  #returns last name
  def get_lastname(self):
    return self.lastname

  #sets lastname if lname isn't an empty string
  def set_lastname(self,lname):
    if len(lname) > 0:
      self.lastname = lname

  #returns job title
  def get_jobtitle(self):
    return self.jobtitle

  #sets job title if job title isn't an empty string
  def set_jobtitle(self,title):
    if len(title) > 0:
      self.jobtitle = title

  #return employee id
  def get_employeeid(self):
    return "Employee ID: " + str(self.employeeid)

  #returns salary
  def get_salary(self):
    return "${:,.2f}".format(self.salary)

  #sets salary if salary isn't an empty string
  def set_salary(self,sal):
    if sal > 0:
      self.salary = sal
```

Now to create the instances

Now that our class is set up, it's time to create instances (objects) of the `Employee` class.

TRY IT

Directions: Let's fill each of those items in with the specific order. You can use the same arguments as seen in the code below, or create your own employee. After entering the arguments, make sure to add the `print()` functions with each variable so you can see that the instance created, `sophia` in this case, did include our arguments.

```
sophia = Employee("Jack","Krichen", 1000, "Manager", 50000)
print(sophia.get_firstname())
print(sophia.get_lastname())
print(sophia.get_employeeid())
print(sophia.get_jobtitle())
print(sophia.get_salary())
```

TRY IT

Directions: Go ahead and run the program to see if you get the same output, or the expected output if you used different employee values.

```
Jack
Krichen
Employee ID: 1000
Manager
$50,000.00
```

This would be a good foundation for our `Employee` class. However, we could add some more elements such as allowing for increases in the employee's salary. We could pass in a percentage and the `salary` would automatically be increased by that amount.

EXAMPLE

```
  #increase salary
  def increase_salary(self,percent):
    if percent > 0:
      self.set_salary(self.salary + self.salary * percent)
```

Notice that as part of the calculation of this new `increase_salary()` method, we check if the percentage is greater than 0 or not. If it is, we'll call the `set_salary` method passing in the increase of salary.

TRY IT

Directions: Let's give it a shot by adding a 2% increase in salary. First add this new method to the `Employee` class and then add the increase of salary code below to the program. Finally, run the program.

```
import datetime
class Employee:
  def __init__(self, fname, lname, empid, title, sal):
    self.firstname = fname
    self.lastname = lname
    self.employeeid = empid
    self.jobtitle = title
    self.salary = sal

    self.hiredate = datetime.date.today()

  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
    if len(fname) > 0:
      self.firstname = fname

  #returns last name
  def get_lastname(self):
    return self.lastname

  #sets lastname if lname isn't an empty string
  def set_lastname(self,lname):
    if len(lname) > 0:
      self.lastname = lname

  #returns job title
  def get_jobtitle(self):
    return self.jobtitle

  #sets job title if job title isn't an empty string
  def set_jobtitle(self,title):
    if len(title) > 0:
      self.jobtitle = title

  #return employee id
  def get_employeeid(self):
    return "Employee ID: " + str(self.employeeid)

  #returns salary
  def get_salary(self):
    return "${:,.2f}".format(self.salary)

  #sets salary if salary isn't an empty string
  def set_salary(self,sal):
    if sal > 0:
      self.salary = sal

  #increase salary
  def increase_salary(self,percent):
    if percent > 0:
      self.set_salary(self.salary + self.salary * percent)

sophia = Employee("Jack","Krichen", 1000, "Manager", 50000)
print(sophia.get_firstname())
print(sophia.get_lastname())
print(sophia.get_employeeid())
print(sophia.get_jobtitle())
print(sophia.get_salary())

#increase of salary
sophia.increase_salary(0.02)
print("After increase: " + sophia.get_salary())
```

Did you get the following as expected? Now our employee will get a bump of $1000.00.

```
Jack
Krichen
Employee ID: 1000
Manager
$50,000.00
After increase: $51,000.00
```

What happens in the case when the salary increase is less than 0?

```
#increase of salary
sophia.increase_salary(-0.02)
print("After increase: " + sophia.get_salary())
```

In that case, there is no change. But this was probably an error on the part of the user entering the values.

```
Jack
Krichen
Employee ID: 1000
Manager
$50,000.00
After increase: $50,000.00
```

TRY IT

Directions: Go ahead and try a negative salary increase.

This is not an ideal result as we most likely would want to inform the user that the value was not accepted. We can add that message to the user to inform them of the error.

```
  #increase salary
  def increase_salary(self,percent):
    if percent > 0:
      self.set_salary(self.salary + self.salary * percent)
    else:
      print("Increase of salary must be greater than 0.")
```

Here we added a conditional statement to look to see if the value entered (passed as an argument) is greater than 0. If it is not, it will present an error.

TRY IT

Directions: Add the conditional statement lines to our `increase_salary()` method and try running it again. Your program should look like this:

```
import datetime
class Employee:
  def __init__(self, fname, lname, empid, title, sal):
    self.firstname = fname
    self.lastname = lname
    self.employeeid = empid
    self.jobtitle = title
    self.salary = sal

    self.hiredate = datetime.date.today()

  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
    if len(fname) > 0:
      self.firstname = fname

  #returns last name
  def get_lastname(self):
    return self.lastname

  #sets lastname if lname isn't an empty string
  def set_lastname(self,lname):
    if len(lname) > 0:
      self.lastname = lname

  #returns job title
  def get_jobtitle(self):
    return self.jobtitle
  #sets job title if job title isn't an empty string
  def set_jobtitle(self,title):
    if len(title) > 0:
      self.jobtitle = title

  #return employee id
  def get_employeeid(self):
    return "Employee ID: " + str(self.employeeid)

  #returns salary
  def get_salary(self):
    return "${:,.2f}".format(self.salary)

  #sets salary if salary isn't an empty string
  def set_salary(self,sal):
    if sal > 0:
      self.salary = sal

  #increase salary
  def increase_salary(self,percent):
    if percent > 0:
      self.set_salary(self.salary + self.salary * percent)
    else:
      print("Increase of salary must be greater than 0.")

sophia = Employee("Jack","Krichen", 1000, "Manager", 50000)
print(sophia.get_firstname())
print(sophia.get_lastname())
print(sophia.get_employeeid())
print(sophia.get_jobtitle())
print(sophia.get_salary())

#increase of salary
sophia.increase_salary(-0.02)
print("After increase: " + sophia.get_salary())
```

We should now see the following:

```
Jack
Krichen
Employee ID: 1000
Manager
$50,000.00
```

```
Increase of salary must be greater than 0.
After increase: $50,000.00
```

TRY IT

Directions: This marks the end of this program for now. Try making some additional changes on your own. See if you can add any additional parameters to our `Employee` class.

To see the final version of this program visit Sophia's Python code page

SUMMARY

In this lesson, we went through a program that creates an `Employee` **class** with all of the attributes and methods associated with it. We've included validation and verification of the object attributes to ensure that valid values are being set when they are being updated.

Best of luck in your learning!