Rex Jones II

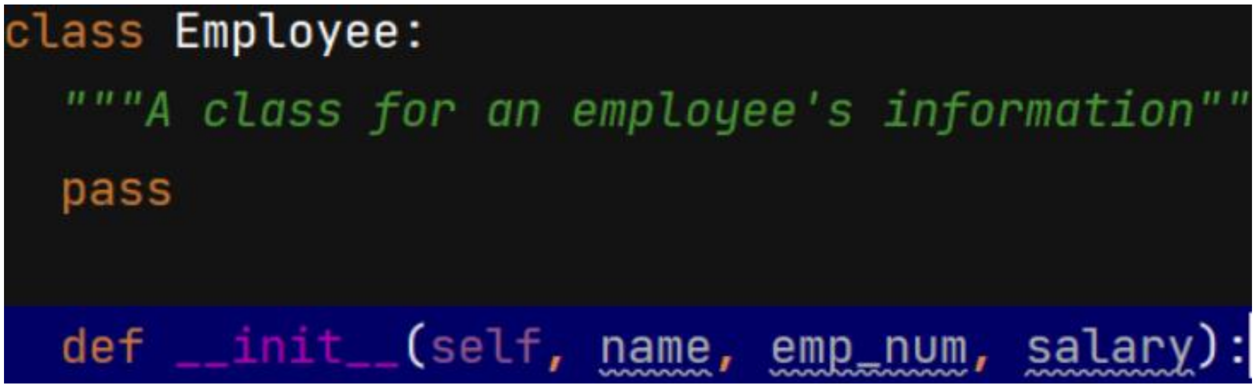
How To Work With Classes, Objects, Methods, & Instances

Python Video = https://youtu.be/Z2_Bo4Ywj0M

In this session, I will continue with classes and objects but show you how to create attributes and methods so an instance can access that information. employee1 and employee2 are unique instances of the Employee class so they will have their own unique data. From the last session, we covered the difference between classes, objects, and instances.

A class is a set of directions for how to create an instance. The Employee class instructs Python to create a unique instance representing a particular employee. Therefore, inside the Employee class, we define attributes and we define methods. An attribute is a quality or characteristic about a person, place, or thing. A method performs an action on the person, place, or thing. In Python, there is a difference between a method and a function. You will hear some people call a method – a function and call a function – a method. The difference is a method is connected to a class. A variable plus the dot operator are required to call the method. However, a function is not connected to a class and does not require a variable. input() and print() are examples of a function.

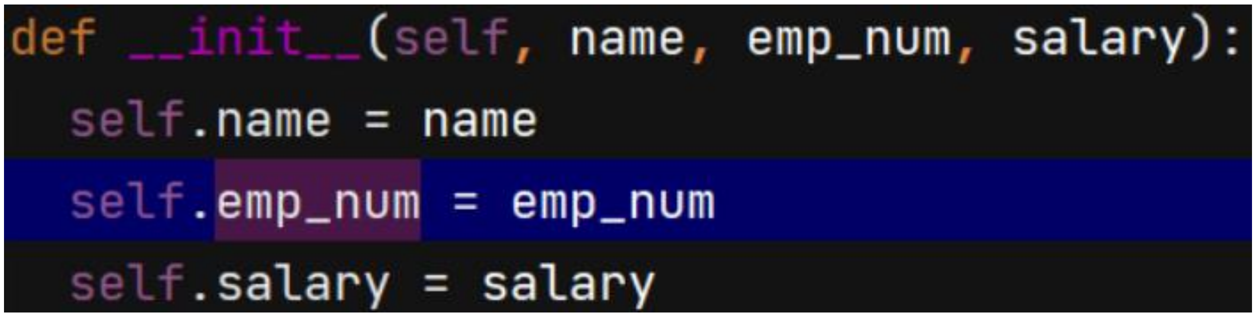
For our class, let's start with an initializer by writing def __init__(self): The initializer is a special method that operates like a constructor. Python automatically runs this method when creating a new Employee instance. The self parameter can be any name like this. However, in Python we use self as the convention. It’s required in the method definition and comes before all parameters. The other parameters will be our attributes which provides information about the employee. All employees have a name, emp_num, and salary.



```
class Employee:
  """A class for an employee's information""
  pass


  def __init__(self, name, emp_num, salary):
```

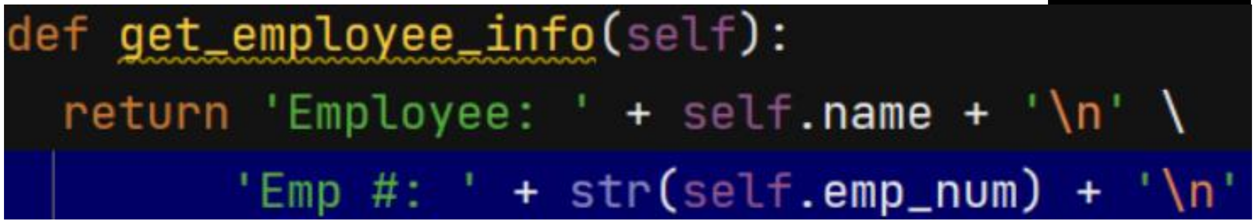
All 3 attributes represent an employee and describe what each employee has in our program. Each piece of information is data about an employee. Therefore, that makes up our Employee data type. Now, let’s assign each attribute to a variable by writing self.name = name, self.emp_num = emp_num, self.salary = salary.



```
def __init__(self, name, emp_num, salary):
  self.name = name
  self.emp_num = emp_num
  self.salary = salary
```

When it comes to the attributes and variables do not have to match each but it’s convention. We can write self.emp_number = emp_num but I prefer them to match each other because it’s convention. We have to create an employee, the value for salary is stored in the parameter attribute then assigned to the variable self.salary. It’s the same process for employee number and name. It’s stored in the parameters and assigned to the variables.

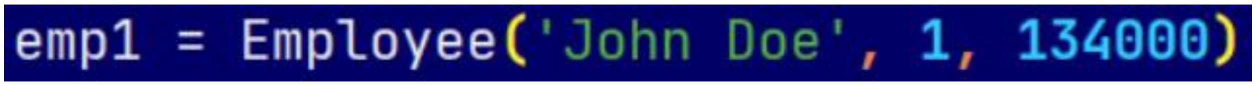
When it comes to a method, it determines what action an employee can do and what we can do with the employee. For example, we can get the employee’s information by writing def to define the method. The name will be get_employee_info(self): Let’s return the ‘Employee: ’ + self.name + ‘\n’ \
‘Emp #: ’ + str(self.emp_num) + ‘\n’.



```python
def get_employee_info(self):
  return 'Employee: ' + self.name + '\n' \
         'Emp #: ' + str(self.emp_num) + '\n'
```

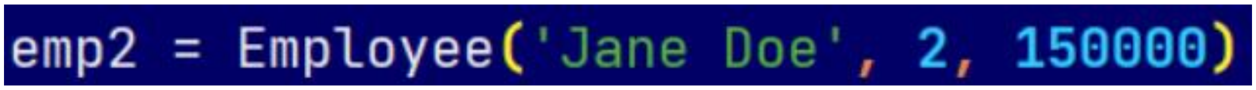
This method has 1 action and that's to return the Employee Name and Number. We see self.name and self.emp_num are the variables. Any variable that starts with self is available to every method in this class. self is the only parameter because the method does not require no more information.

We are finished creating the attributes and methods. Now, we are going to access the attributes and call the method. First step is to create an employee with a name like ('John Doe',) with an Employee Number of 1 and salary of 134000.



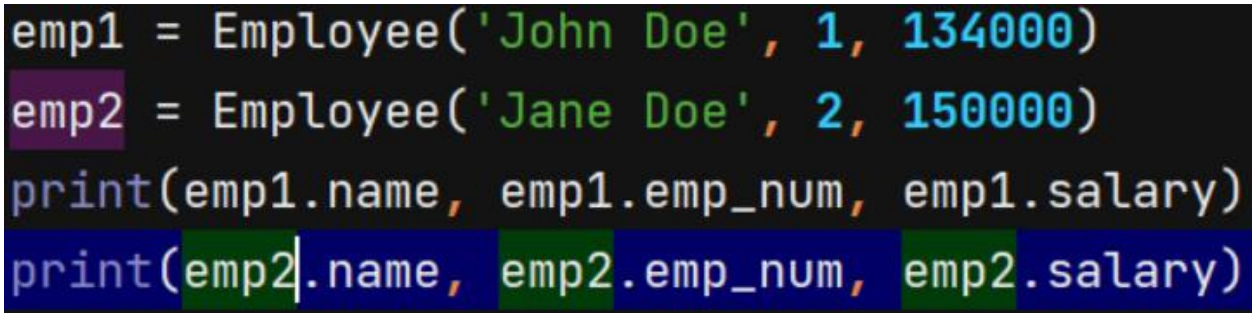
```python
emp1 = Employee('John Doe', 1, 134000)
```

When Python reads this code line, it will call the initializer method. The purpose of this init method is to create an instance representing a unique employee then set the attributes according to the values passed from the arguments. When the arguments John Doe, 1, 134000 are sent to the attributes then the values are returned automatically and stored in the instance employee1. It's the same for employee2. Name will be ('Jane Doe',) employee number will be 2, and salary is 150000. employee1 and employee2 are separate with its own set of attributes and methods.



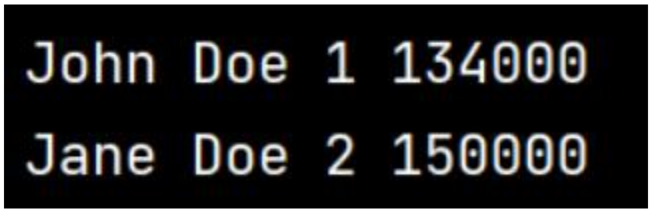
```python
emp2 = Employee('Jane Doe', 2, 150000)
```

The dot operator allows Python to access each attribute and call each method. We write emp1. and we see all of the information from the Employee class for this instance. Attributes employee number, name salary, and both methods get_employee_info and the initializer. It's the same for emp2. We see the same information. Let's print the attributes by writing print(emp1.name, emp1.emp_num, emp1.salary). Copy and Paste this same information then change 1 to 2.



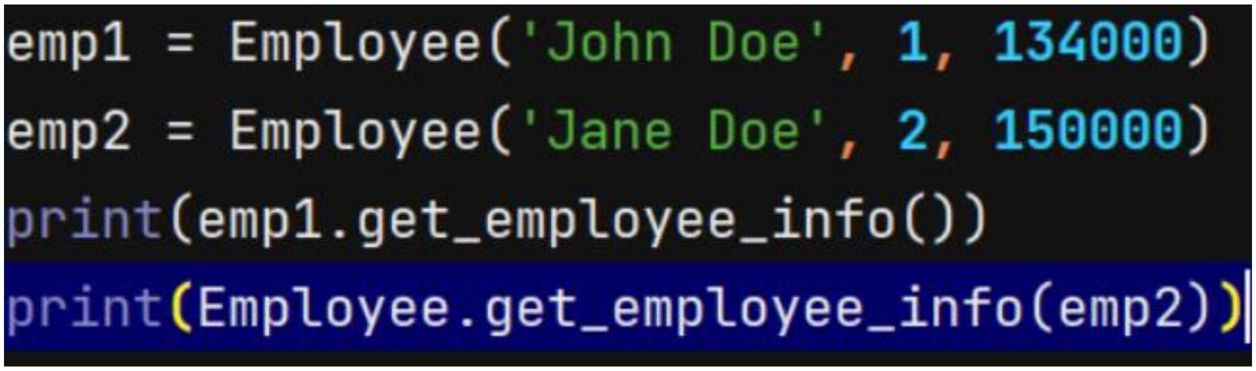
```python
emp1 = Employee('John Doe', 1, 134000)
emp2 = Employee('Jane Doe', 2, 150000)
print(emp1.name, emp1.emp_num, emp1.salary)
print(emp2.name, emp2.emp_num, emp2.salary)
```

Run and the console shows each value (John Doe, 1, 134000) (Jane Doe, 2, 150000).



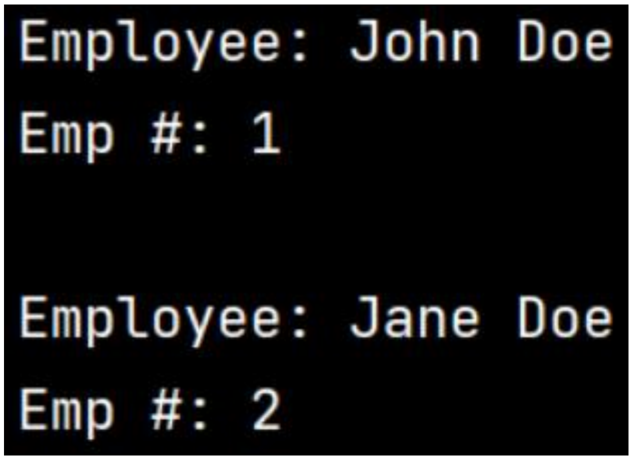
```
John Doe 1 134000
Jane Doe 2 150000
```

When it comes to calling a method, there are 2 ways. We can use the instance or the class name. The instant variable emp1. then call the get_employee_info() method. For the class name, we write Employee. then call the method get_employee_info(). However, with the class name, we must also pass the instance as an argument emp2 because Python does not know which instance we want to run. Let's print both by writing the print() statement before we call the method.



```python
emp1 = Employee('John Doe', 1, 134000)
emp2 = Employee('Jane Doe', 2, 150000)
print(emp1.get_employee_info())
print(Employee.get_employee_info(emp2))
```

I'm going to run.



```
Employee: John Doe
Emp #: 1


Employee: Jane Doe
Emp #: 2
```

The console shows both employee's information. Their name and number. That's it for Python's Classes, Objects, Methods, and Instances.

Contact Info

- ✔ Email Rex.Jones@Test4Success.org
- ✔ YouTube https://www.youtube.com/c/RexJonesII/videos

- ✓ Facebook https://facebook.com/JonesRexII
- ✓ Twitter https://twitter.com/RexJonesII
- ✓ GitHub https://github.com/RexJonesII/Free-Videos
- ✓ LinkedIn https://www.linkedin.com/in/rexjones34/