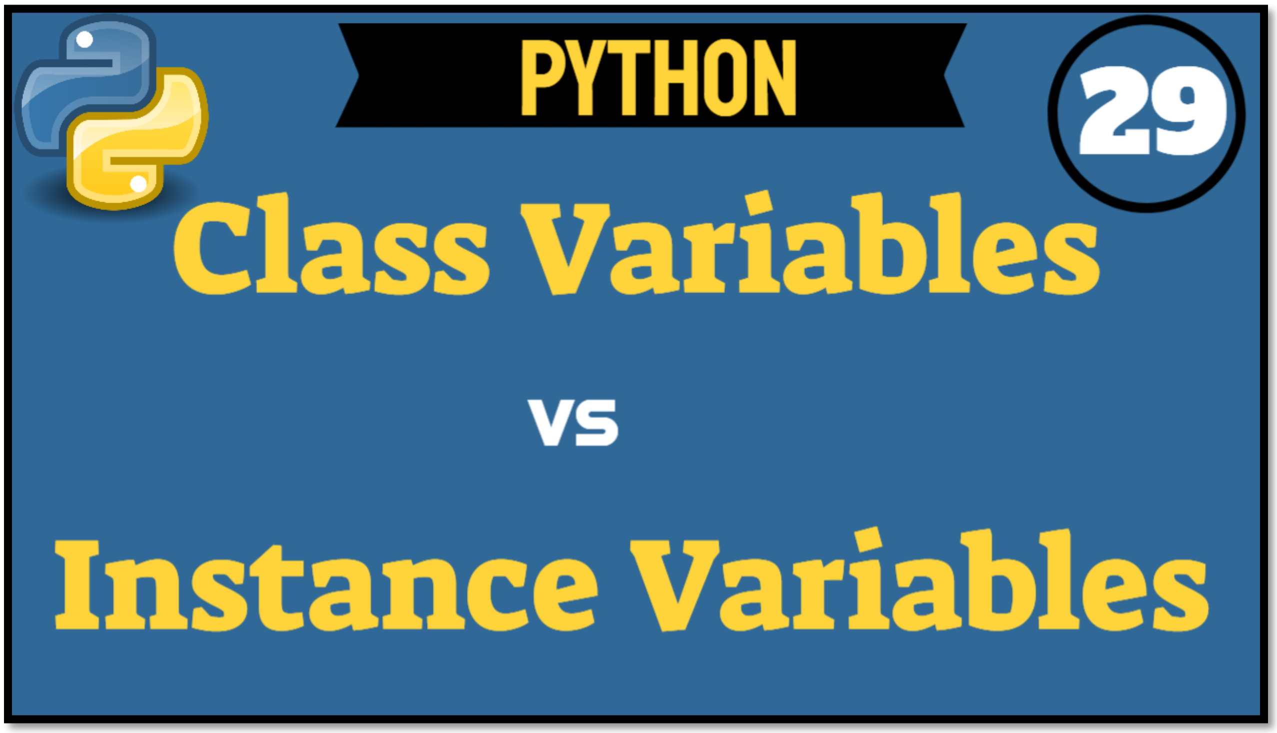
Rex Jones II

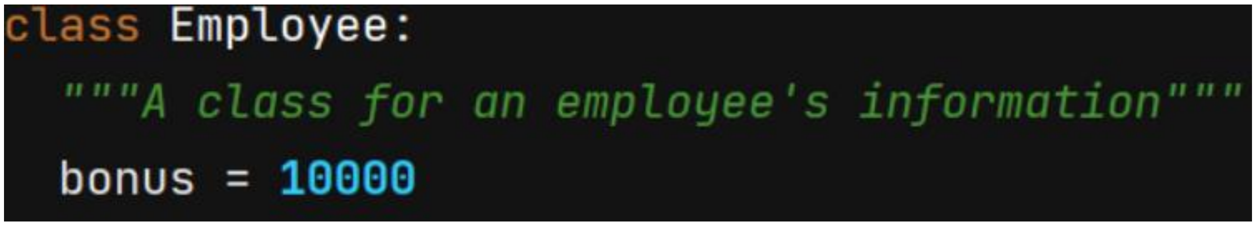
Class Variables vs Instance Variables



Python Video = https://youtu.be/tbqpiCO8SDE

Python Class vs Instance Variables

In this session, let's compare Python Class Variables and Instance Variables. A Class Variable has the same data for all instances of a class. However, an Instance Variable has unique data for all instances in a class. I will build on the previous session using this Employee class. It's not true for all companies but some companies give a bonus = to their employees. This bonus will be 10000.

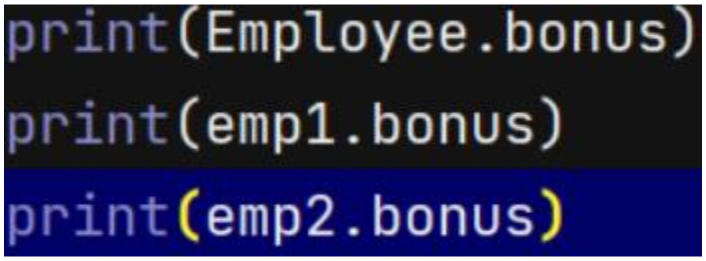


```python
class Employee:
    """A class for an employee's information"""
    bonus = 10000
```

Bonus is a class variable and the instance variables are defined using self: self.name, self.emp_num, and self.salary. Bonus is a class variable because it is defined inside the class but outside each method.

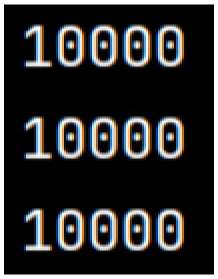
Therefore, the class variable is owned by the class and shared across all members of a class. The members of a class are instances and methods.

Let’s start by sharing a class variable using an instance then I will show you how to share a class variable using a method. We share using an instance by writing each instance emp1.bonus and emp2.bonus. Since bonus is a class variable, we can also access the bonus variable using the class name: Employee.bonus then print() for each one “emp1 and emp2”.



```
print(Employee.bonus)
print(emp1.bonus)
print(emp2.bonus)
```

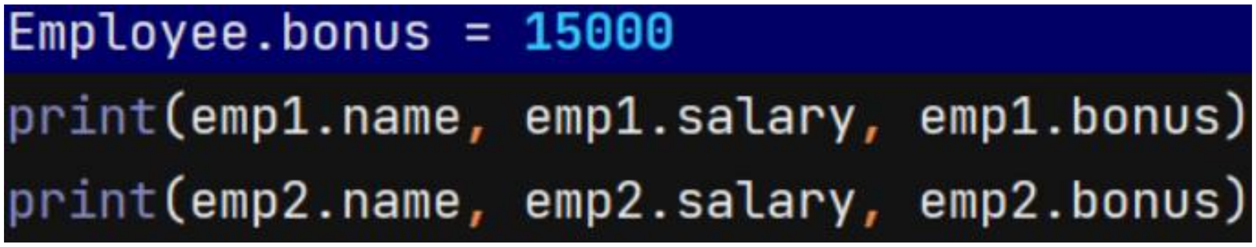
When I run. We see the same value for each instance and Employee.



```
10000
10000
10000
```

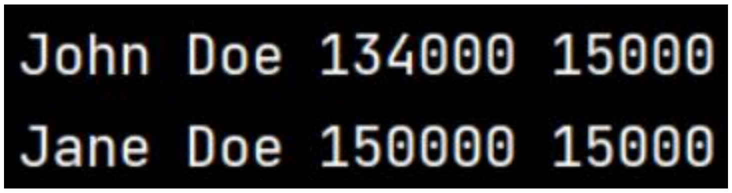
To separate both employees by writing their name and salary. So I’m going to write, emp1.name, emp1.salary – emp2.name, emp2.salary.

Bonus is a good candidate for a class variable because both employees receive a bonus. Anytime we create an employee, they will get a bonus. We can also update the bonus for the entire class after initializing the bonus by writing Employee.bonus = 15000.



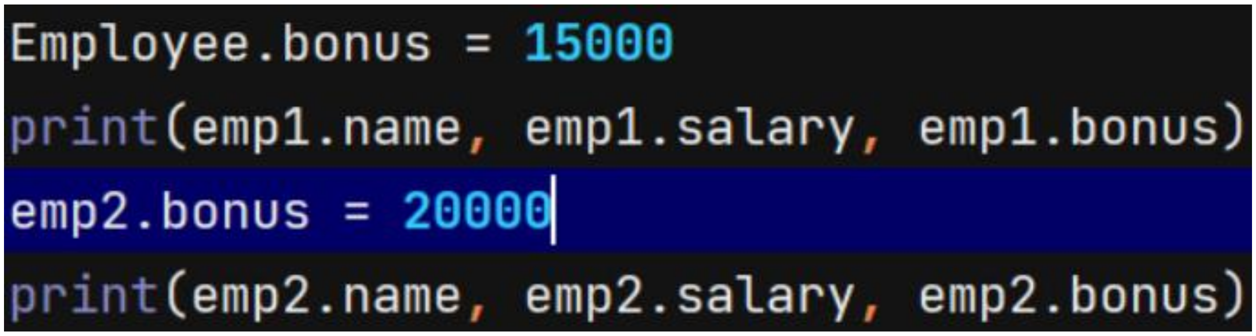
```
Employee.bonus = 15000
print(emp1.name, emp1.salary, emp1.bonus)
print(emp2.name, emp2.salary, emp2.bonus)
```

Run and the console shows 15000 for both employees.

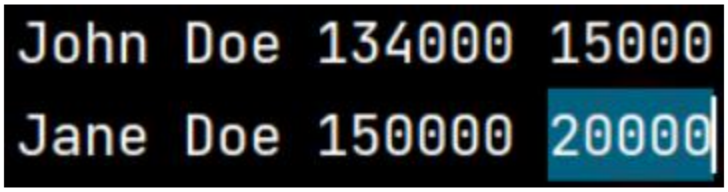


```
John Doe 134000 15000
Jane Doe 150000 15000
```

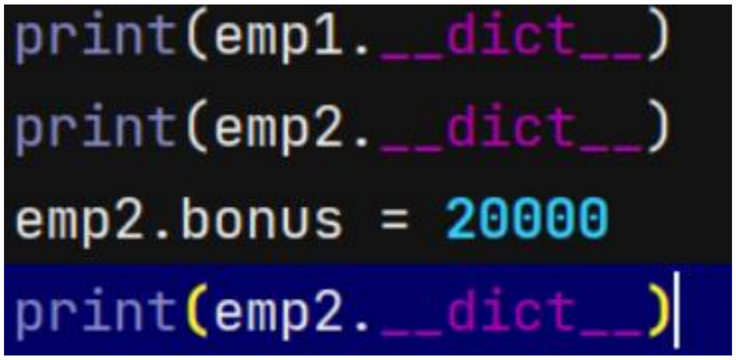
However, we know all employees do not receive the same bonus so I can update the bonus for 1 employee. How about I change employee 2 by writing emp2.bonus then assign 20000?


```
Employee.bonus = 15000
print(emp1.name, emp1.salary, emp1.bonus)
emp2.bonus = 20000
print(emp2.name, emp2.salary, emp2.bonus)
```

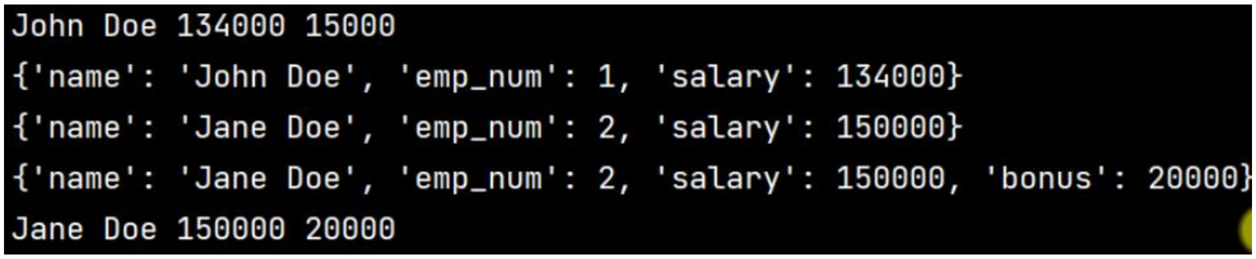
When I run this time. We see a different value for Jane Doe.


```
John Doe 134000 15000
Jane Doe 150000 20000
```

Employee 1 does not have the bonus attribute. However, employee 2 has the bonus attribute but only after updating the value to 20000. Let's print the dictionary by writing emp1. Do you see the 2 underscores before and after dict? That's a way to see the namespaces in Python. Select it for emp1 and also for emp2.__dict__ before updating the value to 20000 and after updating the value to 20000. Let's print the namespaces for employee 1 and employee 2.

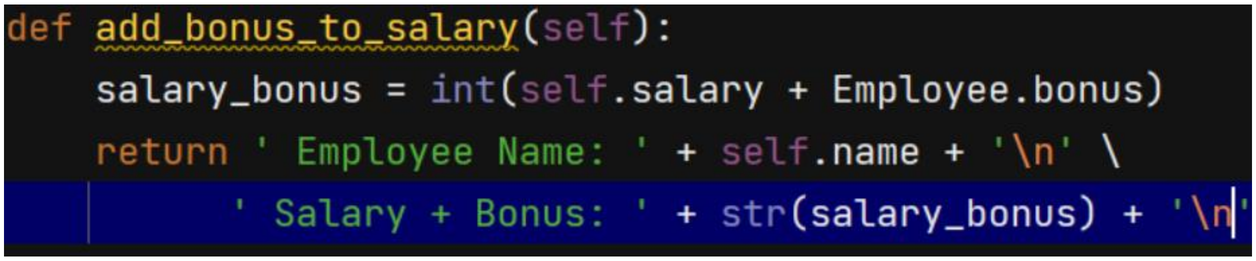

```
print(emp1.__dict__)
print(emp2.__dict__)
emp2.bonus = 20000
print(emp2.__dict__)
```

When I run this time.


```
John Doe 134000 15000
{'name': 'John Doe', 'emp_num': 1, 'salary': 134000}
{'name': 'Jane Doe', 'emp_num': 2, 'salary': 150000}
{'name': 'Jane Doe', 'emp_num': 2, 'salary': 150000, 'bonus': 20000}
Jane Doe 150000 20000
```

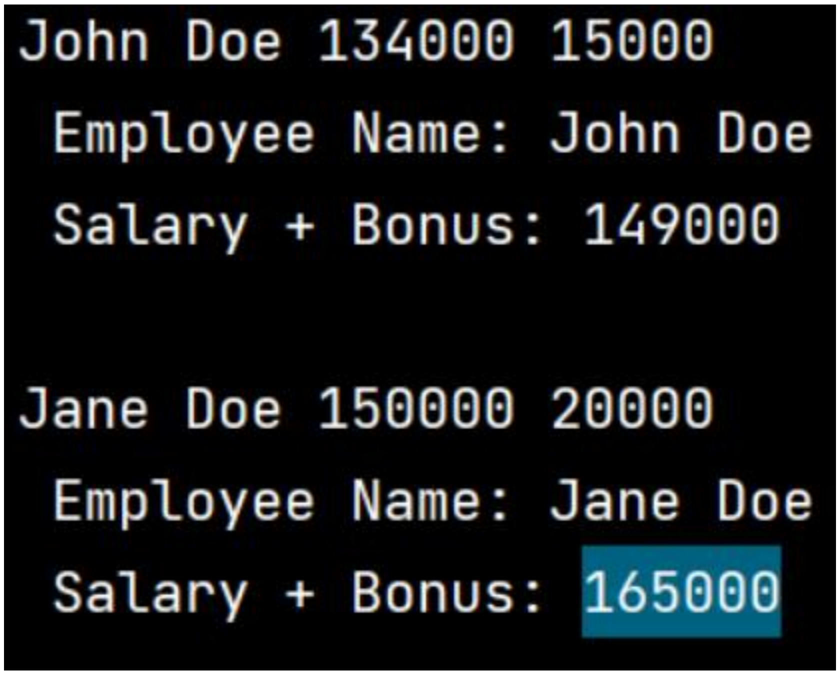
We see the console returns all of the namespaces. However, the namespaces we see are name, emp_num, and salary but bonus only shows up the last time. That's because bonus which is class variable only belongs to the class but was created for employee 2 when updating the value.

This concept is important for the next concept for sharing a class variable using a method. I'm going to show you how to access the class variable using self and the class name Employee. There are times, when we should use the class and other times it's best to use self. Define the method by writing def add_bonus_to_salary(self): The variable name will be salary_bonus =. The value will be an integer so let's write int() then add (self.salary +) At this point, to access the class variable, we must use self or the class name Employee. If I only write bonus then an error shows up that says unresolved reference. Let's start with Employee.bonus and return ' Employee Name: ' + self.name + '\n' + \ ' Salary + Bonus: ' + str(salary_bonus) + '\n'



```
def add_bonus_to_salary(self):
    salary_bonus = int(self.salary + Employee.bonus)
    return ' Employee Name: ' + self.name + '\n' \
           ' Salary + Bonus: ' + str(salary_bonus) + '\n'
```

The next step is to call this method from our 2 instances. We can also use the Employee class but let's focus on employee 1 and employee 2 by writing emp1.add_bonus_to_salary() then print() and print(emp2.add_bonus_to_salary).

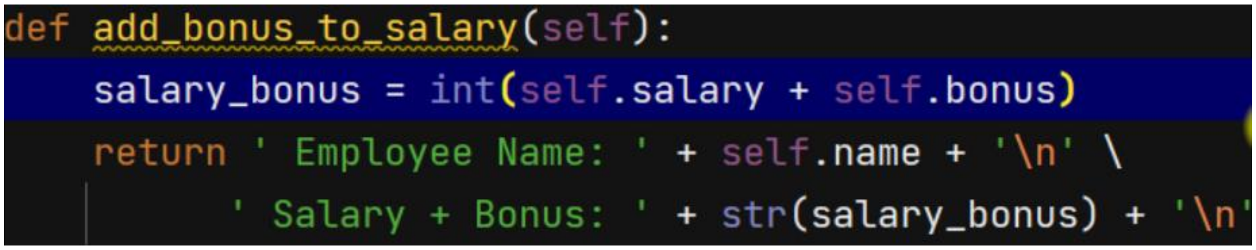


```
John Doe 134000 15000
 Employee Name: John Doe
 Salary + Bonus: 149000

Jane Doe 150000 20000
 Employee Name: Jane Doe
 Salary + Bonus: 165000
```

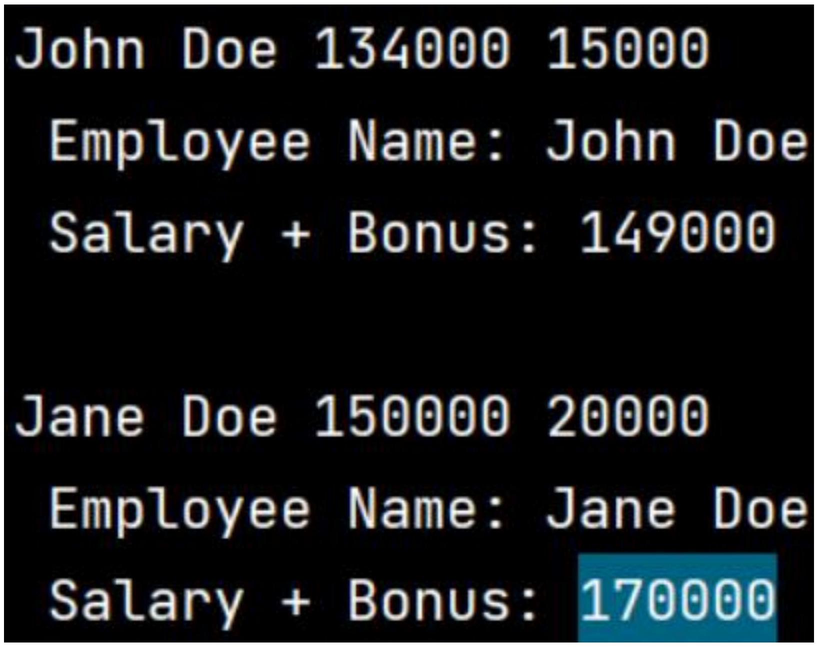
When I run, notice the value for Salary plus Bonus. Employee 1 shows 149,000 because combines 134,000 plus 15,000. But look at Employee 2. It shows 165,000 when combining the salary of 150,000 plus the bonus of 20,000. That total is not correct. It should show 170,000 and not 165,000.

It shows 165,000 because I used Employee.bonus in the add_bonus_to_salary method. For this situation, it's best to use self.bonus because self allows us to override the bonus value for each employee.



```
def add_bonus_to_salary(self):
    salary_bonus = int(self.salary + self.bonus)
    return ' Employee Name: ' + self.name + '\n' \
          ' Salary + Bonus: ' + str(salary_bonus) + '\n'
```

Run and this time we see 170,000.

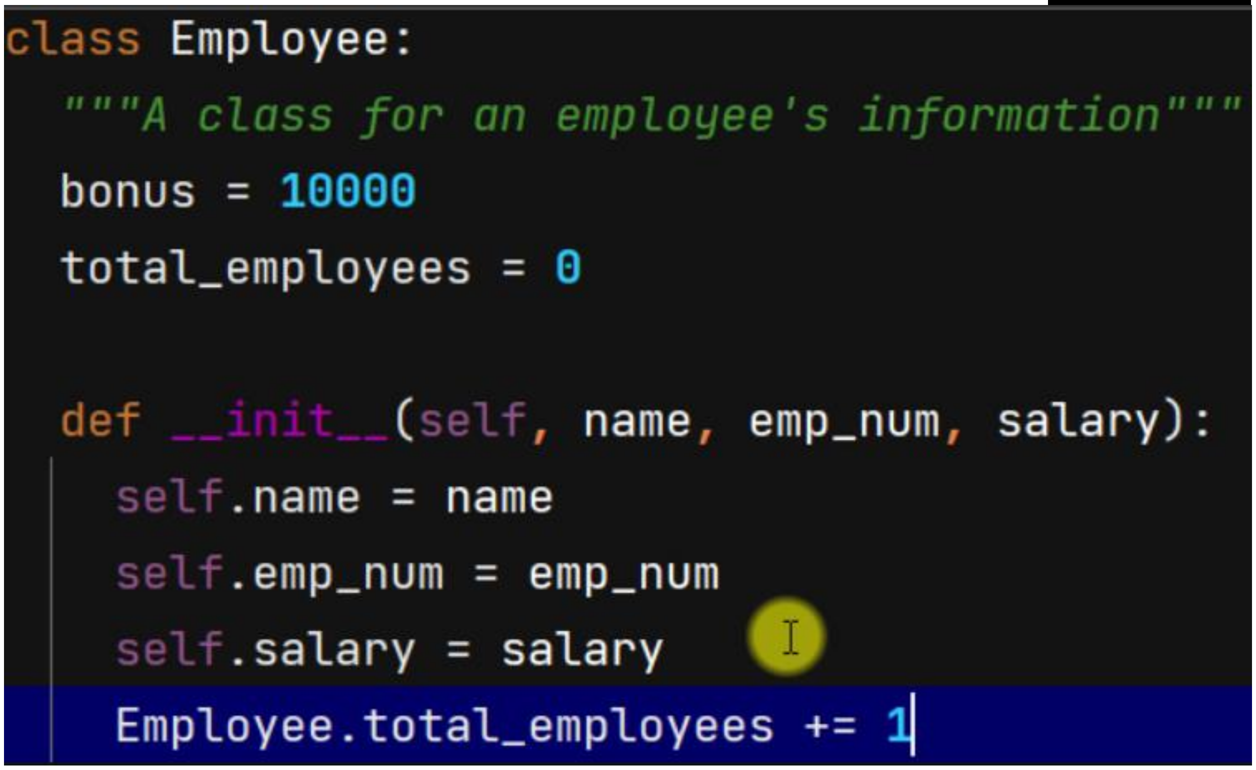


```
John Doe 134000 15000
 Employee Name: John Doe
 Salary + Bonus: 149000


Jane Doe 150000 20000
 Employee Name: Jane Doe
 Salary + Bonus: 170000
```

There are cases when we should not change the value. For example, the number of employees in a company should remain the same for a class. Therefore, As a class variable, we write total_employees = initialized to 0. In the initializer method, we can increase the total number of employees by 1 when creating a new employee. Employee.total_employees += 1.

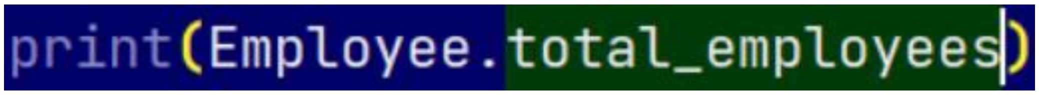
Rex
Jones II



```
class Employee:
  """A class for an employee's information"""
  bonus = 10000
  total_employees = 0

  def __init__(self, name, emp_num, salary):
    self.name = name
    self.emp_num = emp_num
    self.salary = salary
    Employee.total_employees += 1
```

The init method runs automatically every time when creating a new Employee instance. John Doe and Jane Doe. Let's print(Employee.total_employees) and run.



```
print(Employee.total_employees)
```

As expected, we see 2 employees.

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/