Rex Jones II

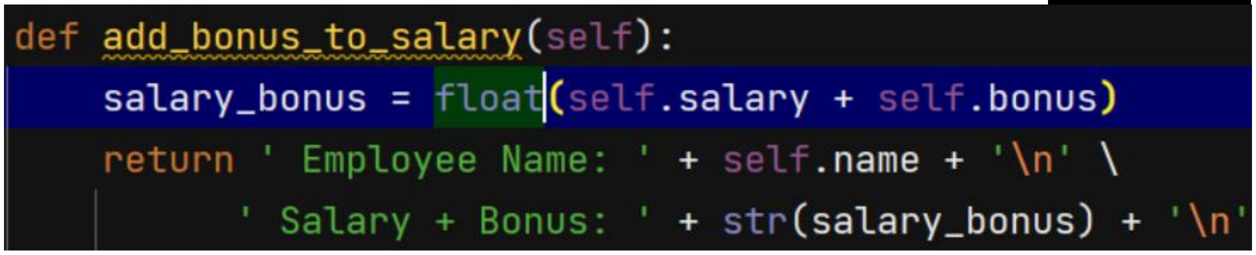
Inheritance (OOP) Object-Oriented Programming

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

In this session, we are going to look at Inheritance. The purpose of Inheritance is to let one class pass everything to a different class. This concept helps with the DRY principle which stands for Don't Repeat Yourself. It helps to reduce code duplication and supports code maintenance. Therefore, we do not have to start from scratch when creating a new class.

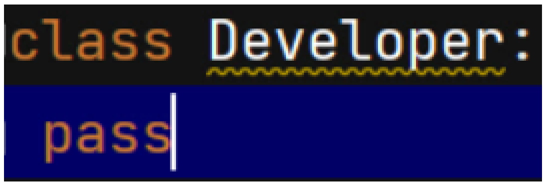
For example, right now, I have a class called Employee class. The Employee class will be our Parent class also known as Super class. It has information that will be inherited by a Child class also known as subclass. The information are class variables, attributes, and methods. The methods are initializer which is the init method. Also get_employee_info() method plus add_bonus_to_salary() method. A Child class is a specialized version of a Parent class. Therefore, good candidates for child classes are developers and testers because they are specialized employees with unique skills to write code.

When it comes to the DRY principle, an engineer not respecting that principle would copy and paste all of this information into a new class called Developer. But this defeats the DRY principle because we should not repeat the same code for a few reasons but 1 reason involves a possible update to our code. A requirement may change to make a function such as int() update to float().



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

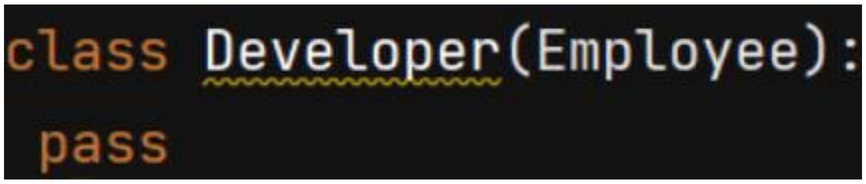
As a result, the code will change in 2 locations and not 1 location. We would have to go back and make a change everywhere we repeated this code. Run and the console shows a decimal for Salary + Bonus for John Doe and Jane Doe. I'm going to change it back to int() so the value is a whole number. Plus removed this duplicated code in the Developer class and only write pass.



```
class Developer:
  pass
```

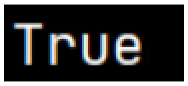
In a previous session, I mentioned pass is a placeholder for future code.

Let's say, we want to create a specialized version of the Employee class. To inherit, we write the name of our Parent class in parenthesis after specifying the Child class. The Developer inherits the (Employee) class.



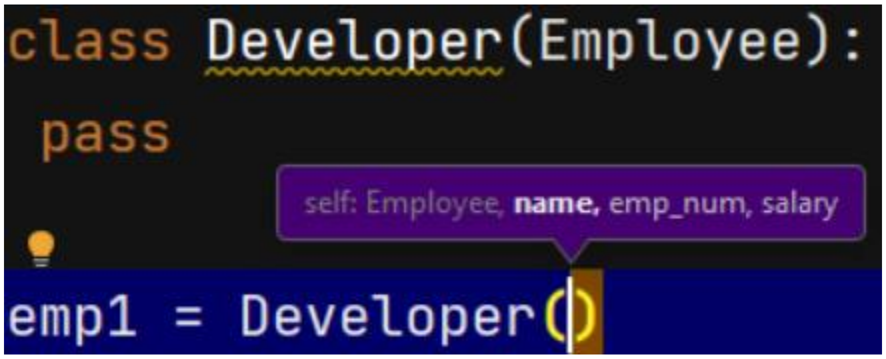
```
class Developer(Employee):
  pass
```

To verify the inheritance, Python provides a built-in Python function called issubclass(). We pass in the Child class which is (Developer,) followed by the Parent class which is (Employee). This statement will return True or False. But it will return True because the Developer class is a subclass of the Employee class.



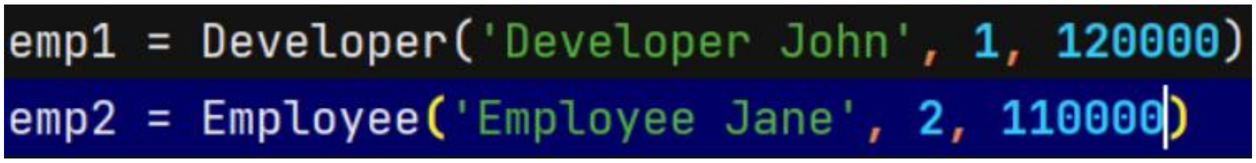
```
True
```

We can write as many subclasses we need to complete our program. At this point, the Developer Child class automatically inherit all of the values such as the attributes, variables, and methods from the Employee class. All of the information. Let me demo the inheritance by writing emp1 = Developer().



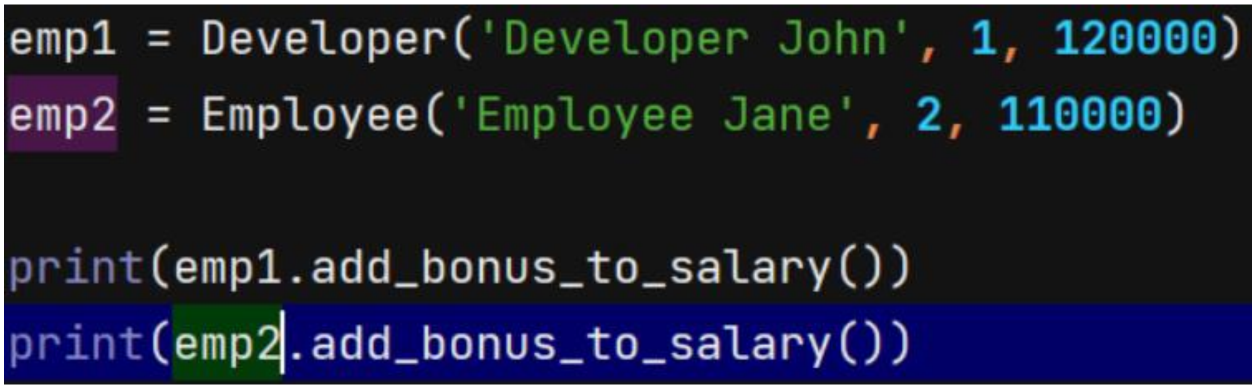
```
class Developer(Employee):
 pass

emp1 = Developer()
```

Do you see self; Employee, name, emp_num, salary? That shows the relationship and which arguments to pass. Name is ‘Developer John’, Employee Number is ‘1’, and Salary is ‘120000’. Also, instantiate emp2 = Employee(‘Employee Jane, 2, 110000).



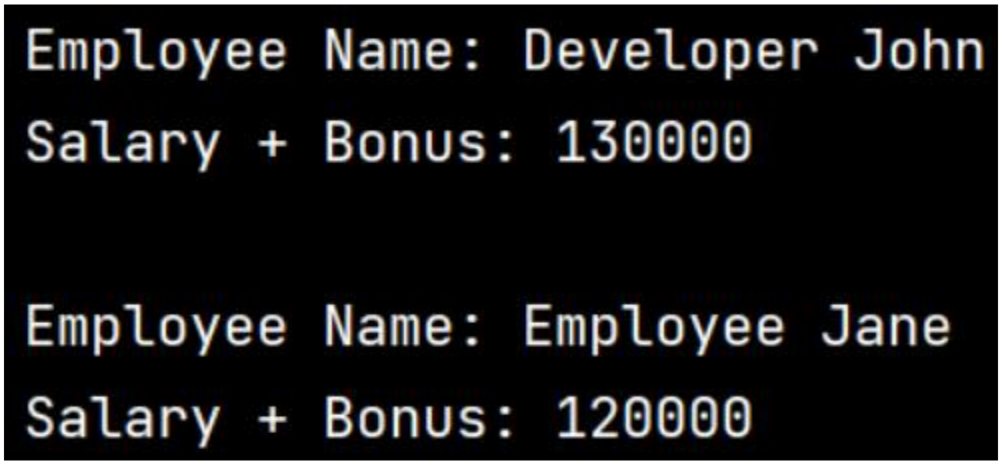
```
emp1 = Developer('Developer John', 1, 120000)
emp2 = Employee('Employee Jane', 2, 110000)
```

Recall the Developer class is empty and only have pass as a placeholder. However, Developer has access to the Employee class by writing emp1. The dot operator provides access to the variables, attributes, and methods. Notice, we see emp_num, name, salary, bonus, add_bonus_to_salary, the initializer, get_employee_info, and total_employees. How about we call the add_bonus_to_salary_method? print(emp1.add_bonus_to_salary) and print(emp2.add_bonus_to_salary).



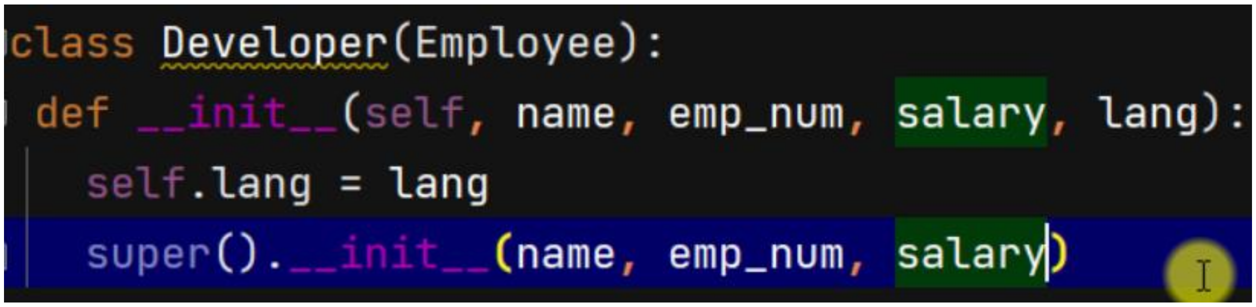
```
emp1 = Developer('Developer John', 1, 120000)
emp2 = Employee('Employee Jane', 2, 110000)

print(emp1.add_bonus_to_salary())
print(emp2.add_bonus_to_salary())
```

Run the program and we see 130000 for Developer John and 120000 for Employee Jane.



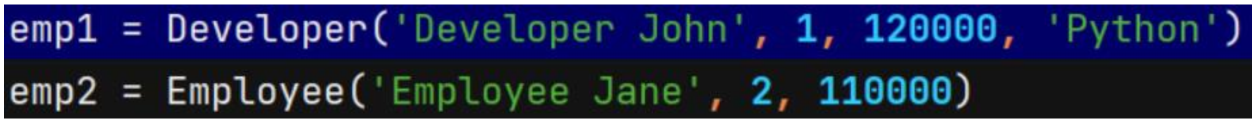
```
Employee Name: Developer John
Salary + Bonus: 130000

Employee Name: Employee Jane
Salary + Bonus: 120000
```

With Inheritance, Python allows us to customize the Child classes with more information than the Parent class. We initialize the Developer subclass with its own init method. Go to the Parent class which is Employee and only copy the init method because there is no need to copy the instant variables. So when I go back to the Developer class, I'm going to paste the init method. At this point, we can add another attribute. The extra attribute will be lang which is short for programming language. Now, create the instanct variable by writing self.lang = lang. Next, is to call the init method from our Parent class. There are 2 ways to call the Parent class. One way it to use super and the other way is to not use super. Let's start by using keyword super().__init__(name, emp_num_salary).



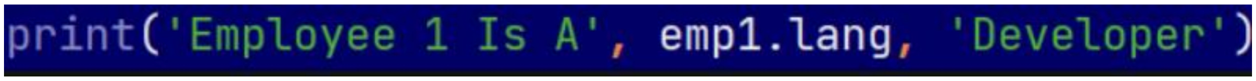
```
class Developer(Employee):
  def __init__(self, name, emp_num, salary, lang):
    self.lang = lang
    super().__init__(name, emp_num, salary)
```

This statement connects the Employee class to the Developer class. Notice the background for our Developer instance is brown. This is an error because we must include another argument since we added the lang attribute. Let's write 'Python' and the error goes away.



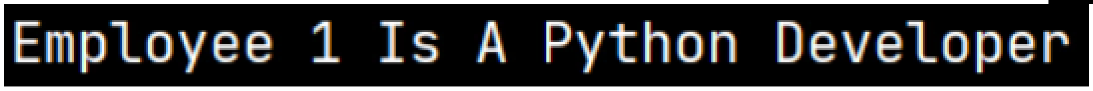
```
emp1 = Developer('Developer John', 1, 120000, 'Python')
emp2 = Employee('Employee Jane', 2, 110000)
```

We can print() Python developer by writing ('Employee 1 Is A', emp1.lang, 'Developer').



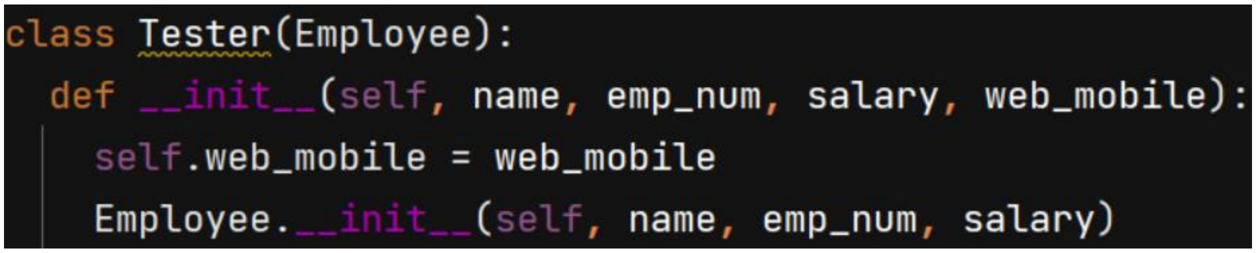
```
print('Employee 1 Is A', emp1.lang, 'Developer')
```

Run and the console shows 'Python for Employee 1 Is A Python Developer.



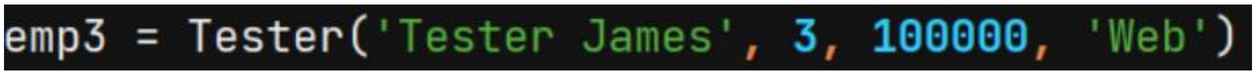
```
Employee 1 Is A Python Developer
```

The next class, let's call it Tester() and it will also inherit the (Employee): class. Define this init method by writing def __init__(self, name, emp_num_salary, web_mobile). Web_mobile is the customized attribute for our Tester class. Create an instance variable by writing self.web_mobile = web_mobile. The other way to call the Parent class init method is by writing the class name Employee.__init__() and use (self, name, emp_num, salary). Using super does not include self but Employee must use self. Then we can write the attributes.



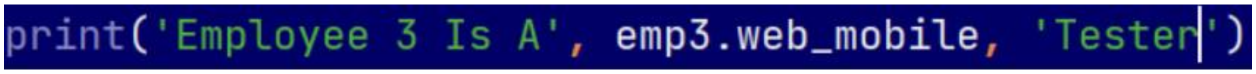
```
class Tester(Employee):
  def __init__(self, name, emp_num, salary, web_mobile):
    self.web_mobile = web_mobile
    Employee.__init__(self, name, emp_num, salary)
```

We generate an instance of the Tester class by writing emp3 = Tester('Tester James', 3, 100000, 'Web').



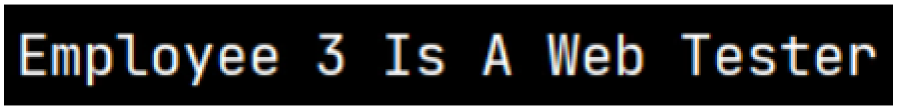
```
emp3 = Tester('Tester James', 3, 100000, 'Web')
```

print() the value by writing ('Employee 3 Is A', emp3.web_mobile, 'Tester').



```
print('Employee 3 Is A', emp3.web_mobile, 'Tester')
```

Run and the console shows Employee 3 Is A Web Tester.



```
Employee 3 Is A Web Tester
```

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/