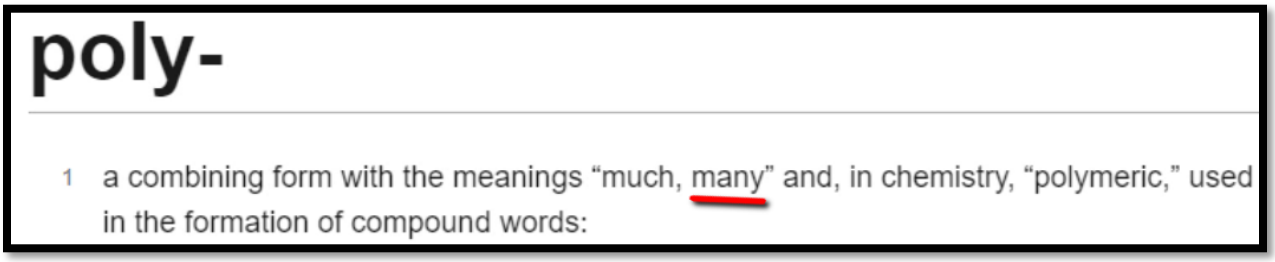
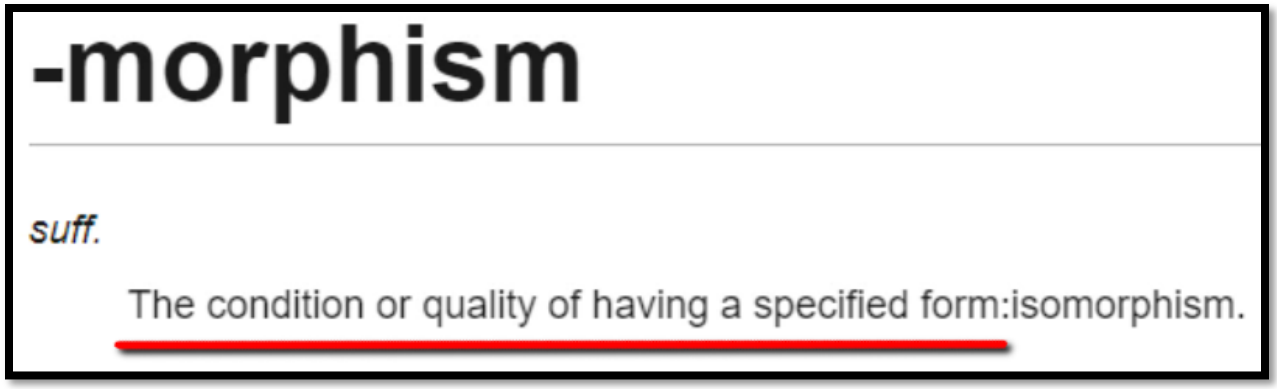
Rex
Jones II

Python OOP
Polymorphism

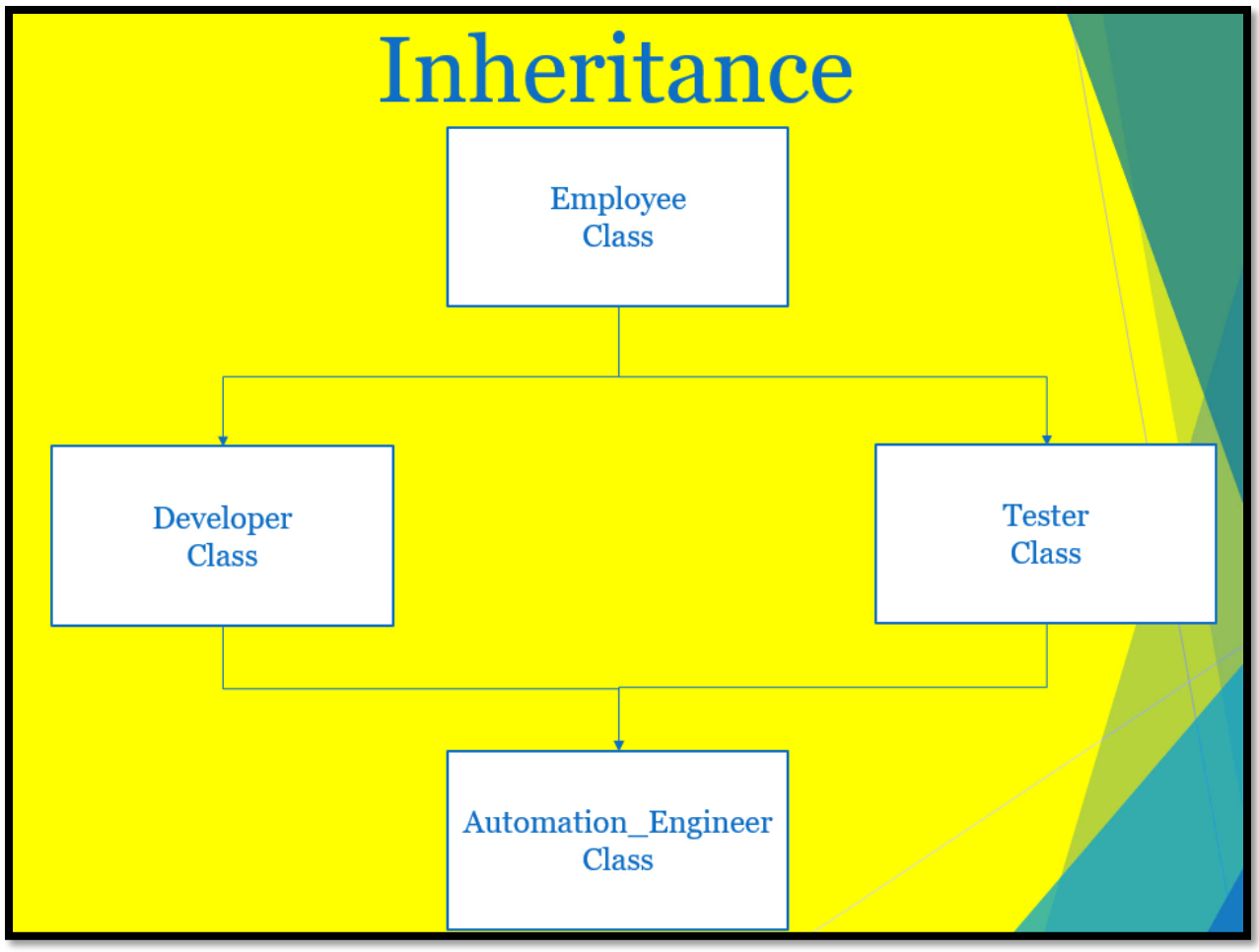
In this session, let's look at Polymorphism in Python. Polymorphism is a compound word that means many specified forms. Dictionary.com shows poly stands for many and morphism is the condition or quality of having a specified form. That's how we get many forms.



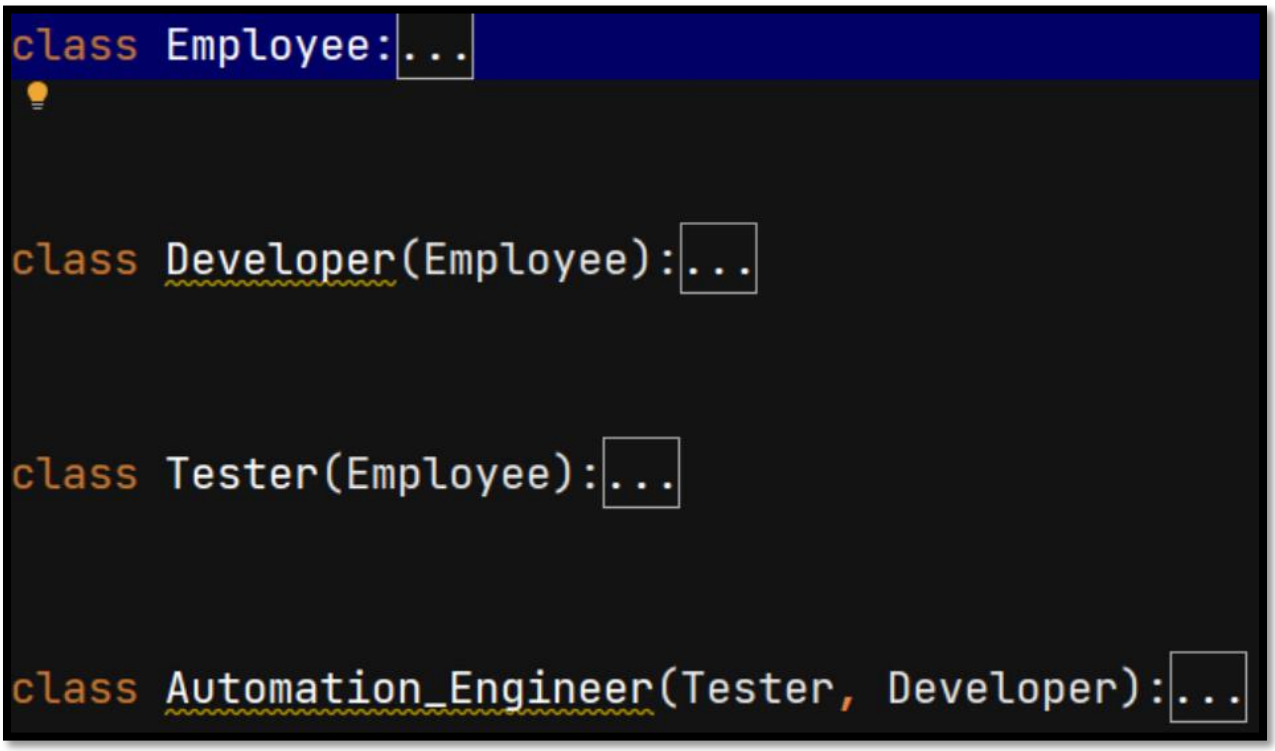


By default, Python supports Method Overriding but does not support Method Overloading. A parent class defines the general method that can be used by all of its child classes.

I will place the transcript PDF document on GitHub. You can follow me on Twitter, connect with me on LinkedIn and Facebook. Also, subscribe to my channel. Polymorphism is related to Inheritance. In this diagram, there are 4 classes.

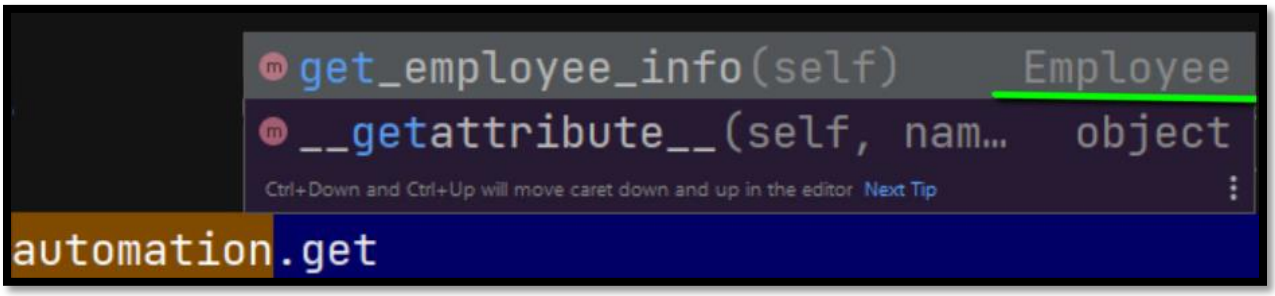


Employee is the parent while Developer and Tester are the child classes. From the previous session, we saw how the Automation_Engineer class inherited 2 classes: Developer and Tester. When it comes to Polymorphism, the Automation_Engineer class and both parent classes maintain the capacity to take on many forms. For example, the Automation_Engineer class can pick up its own behavior plus the behaviors of Employee, Developer, and Tester. In the IDE, we see all 4 classes.



I will show you how the Automation_Engineer class can override the method from its parent class. Right now, there's only 1 method in the Automation_Engineer class and that's the initializer method. Also, from the previous session, I created an instance and wrote 2 print statements. Erase both of these print statements. The reason we override a method from its parent class because the parent class method does not fit what we want to accomplish in the child class.

Let's override the get_employee_info method from Employee. At this point, if I call the method by writing automation.get_employee_info. We see the intellisense shows Employee.



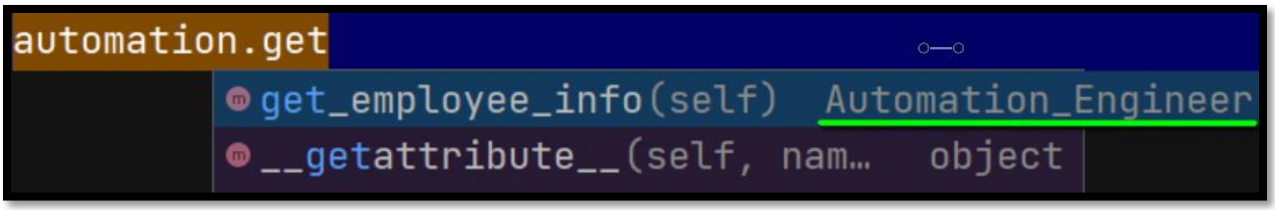
Watch what happens, when I override the method. To override a method in the child class, we define the method with the same name as the method in the parent class. To save time, let's copy and paste the method

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

in the child class Automation_Engineer then return another statement for Salary by writing 'Salary: ' + str(self.salary) + '\n'.

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

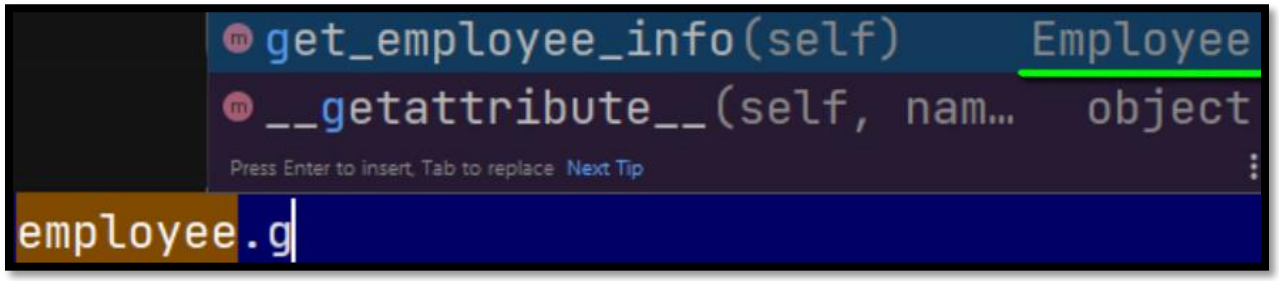
If I use automation. get_employee_info() again to call the method then the intellisense updates to show Automation_Engineer.



Although there are 2 methods with the same name, Python will only focus on the method in the child class because I am using an instance from the child class. Therefore, Automation_Engineer will ignore instructions from the Employee class. However, the Employee class still maintain its own set of instructions if I create an instance by writing employee = Employee('Employee Ed', 5, 100000).

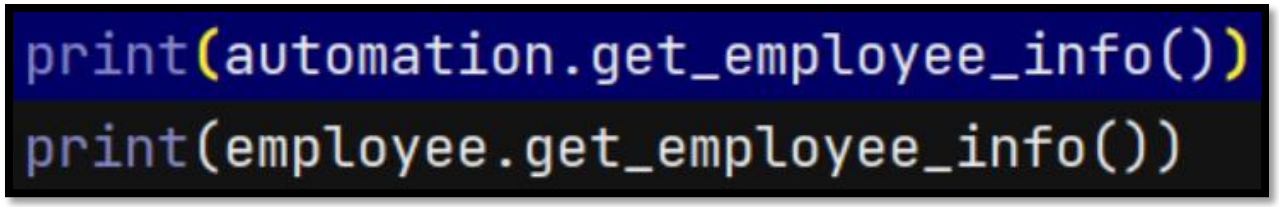
```
automation = Automation_Engineer('Automation Joe', 4, 105000,
                                 'mobile', 'Python')
employee = Employee('Employee Ed', 5, 100000)
```

So, at this point, the Automation_Engineer child class has no effect to the parent class Employee. So, we don't have to worry about breaking anything in the parent class. If I call get_employee_info() using the employee instance, we see intellisense shows Employee.



We can perform this same process for the Developer and Tester classes because they both are parents to the Automation_Engineer class.

I'm going to print employee.get_employee_info() and print automation.get_employee_info()



and when I play. The console shows how it's a difference between the child and parent classes. The child class Automation_Engineer has 3 lines for employee name, employee number, and salary while the parent Employee class has 2 lines for employee name and number. So Polymorphism for Python allows us to override a method. The parent class defines a general method that's common to all child classes then allow the child class to define its own specialized method.

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/