Python Overriding Method

Summary: in this tutorial, you'll learn how to use Python overriding method to allow a child class to provide a specific implementation of a method that is provided by one of its parent classes.

Introduction to Python overridding method

The overriding method allows a child class to provide a specific implementation of a method that is already provided by one of its parent classes.

Let's take an example to understand the overriding method better.

First, define the `Employee` class:

```
class Employee:
    def __init__(self, name, base_pay):
        self.name = name
        self.base_pay = base_pay

    def get_pay(self):
        return self.base_pay
```

The `Employee` class has two instance variables (https://www.pythontutorial.net/python-oop/python-instance-variables/) `name` and `base_pay` . It also has the `get_pay()` method that returns the `base_pay` .

Second, define the `SalesEmployee` that inherits from the `Employee` class:

```
class SalesEmployee(Employee):
    def __init__(self, name, base_pay, sales_incentive):
        self.name = name
        self.base_pay = base_pay
        self.sales_incentive = sales_incentive
```

The `SalesEmployee` class has three instance attributes: `name` , `base_pay` , and `sales_incentive` .

Third, create a new instance of the `SalesEmployee` class and display the pay:

```
john = SalesEmployee('John', 5000, 1500)
print(john.get_pay())
```

Output:

```
5000
```

The `get_pay()` method returns only the `base_pay` , not the sum of the `base_pay` and `sales_incentive` .

When you call the `get_pay()` from the instance of the `SalesEmployee` class, Python executes the `get_pay()` method of the `Employee` class, which returns the `base_pay` .

To include the sales incentive in the pay, you need to redefine the `get_pay()` method in the `SalesEmployee` class as follows:

```
class SalesEmployee(Employee):
    def __init__(self, name, base_pay, sales_incentive):
        self.name = name
        self.base_pay = base_pay
        self.sales_incentive = sales_incentive

    def get_pay(self):
        return self.base_pay + self.sales_incentive
```

In this case, we say that the `get_pay()` method in the `SalesEmployee` class overrides the `get_pay()` method in the `Employee` class.

When you call the `get_pay()` method of the `SalesEmployee` ‘s object, Python will call the `get_pay()` method in the `SalesEmployee` class:

```
john = SalesEmployee('John', 5000, 1500)
print(john.get_pay())
```

Output:

```
6500
```

If you create an instance of the `Employee` class, Python will call the `get_pay()` method of the `Employee` class, not the `get_pay()` method of the `SalesEmployee` class. For example:

```
jane = Employee('Jane', 5000)
print(jane.get_pay())
```

Put it all together.

```
class Employee:
    def __init__(self, name, base_pay):
        self.name = name
        self.base_pay = base_pay

    def get_pay(self):
        return self.base_pay
```

```
class SalesEmployee(Employee):
    def __init__(self, name, base_pay, sales_incentive):
        self.name = name
        self.base_pay = base_pay
        self.sales_incentive = sales_incentive

    def get_pay(self):
        return self.base_pay + self.sales_incentive


if __name__ == '__main__':
    john = SalesEmployee('John', 5000, 1500)
    print(john.get_pay())

    jane = Employee('Jane', 5000)
    print(jane.get_pay())
```

Advanced method overriding example

The following defines the `Parser` class:

```
class Parser:
    def __init__(self, text):
        self.text = text

    def email(self):
        match = re.search(r'[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+', self.text)
        if match:
            return match.group(0)
        return None
    def phone(self):
        match = re.search(r'\d{3}-\d{3}-\d{4}', self.text)
        if match:
            return match.group(0)
        return None

    def parse(self):
        return {
            'email': self.email(),
            'phone': self.phone()
        }
```

The `Parser` class has an attribute `text` which specifies a piece of text to be parsed. Also, the `Parser` class has three methods:

- The `email()` method parses a text and returns the email.
- The `phone()` method parses a text and returns a phone number in the format `nnn-nnnn-nnnn` where `n` is a number from 0 to 9 e.g., `408-205-5663` .
- The `parse()` method returns a dictionary that contains two elements `email` and `phone` . It calls the `email()` and `phone()` method to extract the email and phone from the `text` attribute.

The following uses the `Parser` class to extract email and phone:

```
s = 'Contact us via 408-205-5663 or email@test.com'
parser = Parser(s)
print(parser.parse())
```

Output:

```
{'email': 'email@test.com', 'phone': '408-205-5663'}
```

Suppose you need to extract phone numbers in the format `n-nnn-nnn-nnnn` , which is the UK phone number format. Also, you want to use extract email like the `Parser` class

To do it, you can define a new class called `UkParser` that inherits from the `Parser` class. In the `UkParser` class, you override the `phone()` method as follows:

```
class UkParser(Parser):
    def phone(self):
        match = re.search(r'(\+\d{1}-\d{3}-\d{3}-\d{4})', self.text)
        if match:
            return match.group(0)
        return None
```

The following use the `UkParser` class to extract a phone number (in UK format) and email from a text:

```
s2 = 'Contact me via +1-650-453-3456 or email@test.co.uk'
parser = UkParser(s2)
print(parser.parse())
```

Output:

```
{'email': 'email@test.co.uk', 'phone': '+1-650-453-3456'}
```

In this example, the `parser` calls the `parse()` method from the parent class which is the Parser class. In turn, the `parse()` method calls the `email()` and `phone()` methods.

However, the `parser()` doesn't call the `phone()` method of the `Parser` class but the `phone()` method of the `UkParser` class:

```
parser.parse()
```

The reason is that inside the `parse()` method, the `self` is the `parser` which is an instance of the `UkParser` class.

Therefore, when you call `self.phone()` method inside the `parse()` method, Python will look for the `phone()` method that is bound to the instance of the `UkParser` .

Put it all together.

```python
import re



class Parser:
    def __init__(self, text):
        self.text = text

    def email(self):
        match = re.search(r'[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+', self.text)
        if match:
            return match.group(0)
        return None

    def phone(self):
        match = re.search(r'\d{3}-\d{3}-\d{4}', self.text)
        if match:
            return match.group(0)
        return None

    def parse(self):
        return {
            'email': self.email(),
            'phone': self.phone()
        }



class UkParser(Parser):
    def phone(self):
        match = re.search(r'(\+\d{1}-\d{3}-\d{3}-\d{4})', self.text)
        if match:
            return match.group(0)
        return None


if __name__ == '__main__':
    s = 'Contact us via 408-205-5663 or email@test.com'
    parser = Parser(s)
    print(parser.parse())

    s2 = 'Contact me via +1-650-453-3456 or email@test.co.uk'
    parser = UkParser(s2)
    print(parser.parse())
```

Overriding attributes

The following shows how to implement the Parser and UkParser classes by overriding attributes:

```
import re


class Parser:
    phone_pattern = r'\d{3}-\d{3}-\d{4}'

    def __init__(self, text):
        self.text = text

    def email(self):
        match = re.search(r'[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+', self.text)
        if match:
            return match.group(0)
        return None
    def phone(self):
        match = re.search(self.phone_pattern, self.text)
        if match:
            return match.group(0)
        return None

    def parse(self):
        return {
            'email': self.email(),
            'phone': self.phone()
        }


class UkParser(Parser):
    phone_pattern = r'(\+\d{1}-\d{3}-\d{3}-\d{4})'


if __name__ == '__main__':
    s = 'Contact us via 408-205-5663 or email@test.com'
    parser = Parser(s)
    print(parser.parse())

    s2 = 'Contact me via +1-650-453-3456 or email@test.co.uk'
    parser = UkParser(s2)
    print(parser.parse())
```

```

```

In this example, the `Parser` has a class variable `phone_pattern`. The `phone()` method in the `Parser` class uses the `phone_pattern` to extract a phone number.

The `UkParser` child class redefines (or overrides) the `phone_pattern` class attribute.

If you call the `parse()` method from the `UkParser` ‘s instance, the `parse()` method calls the `phone()` method that uses the `phone_pattern` defined in the `UkParser` class.

Summary

- Method overrding allows a child class to provide a specific implementation of a method that is already provided by one of its parent class.