Object Oriented Programing with Python

05/03/2024

Contents

1 Creating an object with a class (basic):

```python
class Item:
    pass

item1 = 'Phone'
item1_price = 100
item1_quantity = 5
item1_price_total = item1_price * item1_quantity
```

2 Creating the first method

Methods are functions inside classes.

How we can go ahead and design some methods, which are going to be allowed to be executed on our instances?

The answer is, the methods should be created inside our class.

```python
class Item:
    def calculate_total_price(self):
        pass


item1 = Item()
item1_price = 100
item1_quantity = 5
item1_price_total = item1_price * item1_quantity
```

3 Self

Python passes the object itself as the first argument everytime. So we are not allowed to create methods without the self.

```python
class Item:
    def calculate_total_price(self, x, y): # x, and y are parameters
        return x * y


item1 = Item()
item1.name = "Phone"
item1.price = 100
item1.quantity = 5
```

```python
print(item1.calculate_total_price(item1.price, item1.quantity)) # When I
    call the method calculate_total_price, Python passes the variables
    as mehtods.
```

4 Magic Methods:

__init__

Instance Attributes:
An instance attribute is a Python variable belonging to one, and only one, object. This variable is only accessible in the scope of this object, and it's defined inside the constructor function, __init__(self,..) of the class.

```python
class Item:
    def __init__(self): # Python executes the init function
        automatically.
        print("I AM CREATED")


item1 = Item()
item1_name = "Phone"
item1.price = 100
item1_quantity = 5


item1 = Item()
item_name = "Laptop"
item1_price = 1000
item1_quantity = 3


Output:
    I AM CREATED
    I AM CREATED
```

How to avoid creating the whole attribute, adding more parameters into the class:Assign the attributes dinamicaly.

```python
class Item:
    def __init__(self, name):
        self.name = name
        print(f"An instance created from instance: {name}")


item1 = Item("Phone")
#item1.name = "Phone"
item1.price = 100
```

```
item1_quantity = 5

item2 = Item("Laptop")
#item2.name = "Laptop("
item2_price = 1000
item2.quantity = 3

An instance created from instance: Phone
An instance created from instance: Laptop
```

```
class Item:
    def __init__(self, name):
        self.name = name
        print(f"An instance created from instance: {name}")


item1 = Item("Phone")
#item1.name = "Phone"
item1.price = 100
item1_quantity = 5

item2 = Item("Laptop")
#item2.name = "Laptop("
item2_price = 1000
item2.quantity = 3

Output:
An instance created from instance: Phone
An instance created from instance: Laptop
```

5   Simplification / Improvement of objects:

Here is important to observe, that we can simplify the creation of objects adding
the attributes to the class. Self is always mandatory.

```
class Item:
    def __init__(self, name, price, quantity = 0):
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_total_price(self):
        return self.price * self.quantity



item1 = Item("Phone", 10, 2)
```

```
item2 = Item('Laptop', 50, 15)

print(item1.calculate_total_price())
print(item2.calculate_total_price())

Output:
20
750
```

6   Delimiting data types in the class attributes.

```
class Item:
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
        assert price >= 0
        assert quantity >= 0

        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_revenues(self):
        return self.price * self.quantity



item1 = Item("Phone", "ten", 2)
item2 = Item('Laptop', 50, 15)

print(item1.calculate_revenues())
print(item2.calculate_revenues())

Output:
Traceback (most recent call last):
File "/home/lucas/workspace/OOP/test.py", line 16, in <module>
item1 = Item("Phone", "ten", 2)
File "/home/lucas/workspace/OOP/test.py", line 4, in __init__
assert price >= 0
TypeError: '>=' not supported between instances of 'str' and 'int'
```

7 Assertion Error: Technick to find quicklier the errors in the objects input

Assert statement allows us to validate the data types of the objects and to identify errors quicklier.

```python
class Item:
    def __init__(self, name: str, price: float, quantity = 0):
      #Run validations to the received arguments
      assert price >= 0, f"Price is lower than 0"
      assert quantity >= 0, f"Quantity is lower than 0"

      self.name = name
      self.price = price
      self.quantity = quantity

  def calculate_revenues(self):
      return self.price * self.quantity



item1 = Item("Phone", 2, -2)
item2 = Item('Laptop', 50, 15)

print(item1.calculate_revenues())
print(item2.calculate_revenues())

Output:

/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
Traceback (most recent call last):
File "/home/lucas/workspace/OOP/test.py", line 16, in <module>
item1 = Item("Phone", 2, -2)
File "/home/lucas/workspace/OOP/test.py", line 5, in __init__
assert quantity >= 0, f"Quantity is lower than 0"
AssertionError: Quantity is lower than 0

Process finished with exit code 1
```

8 Class Attributes

Class attributes are like global attributes. They belong to the class, but they can also be reached from the instance aswell.

For this example I used pay_rate as a class attribute.

```python
class Item:
  pay_rate = 0.8
```

```python
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
        assert price >= 0, f"Price is lower than 0"
        assert quantity >= 0, f"Quantity is lower than 0"

        #Instance Level
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_revenues(self):
        return self.price * self.quantity



item1 = Item("Phone", 2, 2)
item2 = Item('Laptop', 50, 15)

print(item1.pay_rate)
print(item2.pay_rate)

Output:
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
0.8
0.8

Process finished with exit code 0
```

9 Magic Attriute:

To see all the existent attributes: __dict__ This is used to see all the attributes belonging to an object.

```python
class Item:
    #Class Level
    pay_rate = 0.8
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
        assert price >= 0, f"Price is lower than 0"
        assert quantity >= 0, f"Quantity is lower than 0"

        #Instance Level
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_revenues(self):
        return self.price * self.quantity
```

```python
item1 = Item("Phone", 2, 2)
item2 = Item('Laptop', 50, 15)

print(f"These are all the class level attributes: {Item.__dict__}")
print(f"These are all the instance level attributes: {item1.__dict__}")
```

```
Output:
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
These are all the class level attributes: {'__module__': '__main__',
    'pay_rate': 0.8, '__init__': <function Item.__init__ at
    0x7ff8de22a0e0>, 'calculate_revenues': <function
    Item.calculate_revenues at 0x7ff8de22a440>, '__dict__': <attribute
    '__dict__' of 'Item' objects>, '__weakref__': <attribute
    '__weakref__' of 'Item' objects>, '__doc__': None}
These are all the instance level attributes: {'name': 'Phone', 'price':
    2, 'quantity': 2}

Process finished with exit code 0
```

10 Accessing a class attribute from a method

```python
class Item:
  #Class Level
  pay_rate = 0.8
  def __init__(self, name: str, price: float, quantity = 0):
    #Run validations to the received arguments
    assert price >= 0, f"Price is lower than 0"
    assert quantity >= 0, f"Quantity is lower than 0"

    #Instance Level
    self.name = name
    self.price = price
    self.quantity = quantity

  def calculate_revenues(self):
    return self.price * self.quantity

  def apply_discount(self):
    self.price = self.price * Item.pay_rate



item1 = Item("Phone", 15000, 2)
item1.apply_discount()
```

```python
print(item1.price)
```

Output:
```
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
12000.0

Process finished with exit code 0
```

11 Modify a class attribute for an specific instance

:

```python
class Item:
    #Class Level
    pay_rate = 0.8
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
        assert price >= 0, f"Price is lower than 0"
        assert quantity >= 0, f"Quantity is lower than 0"

        #Instance Level
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_revenues(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price * self.pay_rate # It is important to
            change Item with self. If not, the pay rate will not be read
            from the instance, rather from the class. Without this
            modification, it will be changed on the instance.


item1 = Item("Phone", 100, 2)
item1.apply_discount()
print(f'The price of {item1.name} is {item1.price}')

item2 = Item("Laptop", 1500, 1)
item2.pay_rate = 0.9
item2.apply_discount()
print(f'The price of {item2.name} is {item2.price}')
```

Output:

```
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
The price of Phone is 80.0
The price of Laptop is 1350.0

Process finished with exit code 0
```

12   Working with multiple instances:

We create a list, and we append all items from Item to the list all.

```python
class Item:
    #Class Level
    pay_rate = 0.8
    all = []
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
        assert price >= 0, f"Price is lower than 0"
        assert quantity >= 0, f"Quantity is lower than 0"

        #Assign to self object
        self.name = name
        self.price = price
        self.quantity = quantity

        #Actions to execute
        Item.all.append(self)

    def calculate_revenues(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price * self.pay_rate



item1 = Item("Phone", 100, 1)
item2 = Item("Laptop", 1000, 3)
item3 = Item("Cable", 10, 5)
item4 = Item("Mouse", 50, 5)
item5 = Item("Keyboard", 75, 5)

for instance in Item.all:
    print(instance.name)

Output:
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
Phone
Laptop
```

```
Cable
Mouse
Keyboard

Process finished with exit code 0
```

13 Magic Method: __repr__

repr: means representing your objects It's a good tool to handle the different objects.

```python
class Item:
    #Class Level
    pay_rate = 0.8
    all = []
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
        assert price >= 0, f"Price is lower than 0"
        assert quantity >= 0, f"Quantity is lower than 0"

        #Assign to self object
        self.name = name
        self.price = price
        self.quantity = quantity

        #Actions to execute
        Item.all.append(self)

    def calculate_revenues(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price * self.pay_rate

    def __repr__(self):
        return f"Item name:{self.name}, price: {self.price}, and
            quantity: {self.quantity}"



item1 = Item("Phone", 100, 1)
item2 = Item("Laptop", 1000, 3)
item3 = Item("Cable", 10, 5)
item4 = Item("Mouse", 50, 5)
item5 = Item("Keyboard", 75, 5)
```

```
    print(Item.all)
```

```
Output:
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
[Item name:Phone, price: 100, and quantity: 1, Item name:Laptop, price:
    1000, and quantity: 3, Item name:Cable, price: 10, and quantity: 5,
    Item name:Mouse, price: 50, and quantity: 5, Item name:Keyboard,
    price: 75, and quantity: 5]

Process finished with exit code 0
```

The __repr__ magic method is also useful to show the objects in a form directly usable to transfer them to another python user. This is part of the best practices according to python document.

```python
class Item:
    #Class Level
    pay_rate = 0.8
    all = []
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
        assert price >= 0, f"Price is lower than 0"
        assert quantity >= 0, f"Quantity is lower than 0"

        #Assign to self object
        self.name = name
        self.price = price
        self.quantity = quantity

        #Actions to execute
        Item.all.append(self)

    def calculate_revenues(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price * self.pay_rate

    def __repr__(self):
        return f"Item({self.name}, {self.price}, {self.quantity})"




item1 = Item("Phone", 100, 1)
item2 = Item("Laptop", 1000, 3)
item3 = Item("Cable", 10, 5)
item4 = Item("Mouse", 50, 5)
item5 = Item("Keyboard", 75, 5)
```

12

```
    print(Item.all)
```

```
Output:
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
[Item(Phone, 100, 1), Item(Laptop, 1000, 3), Item(Cable, 10, 5),
    Item(Mouse, 50, 5), Item(Keyboard, 75, 5)] #We recibe a list, which
    is more friendly.
The first element is equal to the first object, and so on.

  Process finished with exit code 0
```

14   Class Method

Definition:
A class method is a method that is bound to the class and not the object of the class. They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance. It can modify a class state that would apply across all the instances of the class.

  For this I created a csv file named "csv.csv". The content of the file is as follows:

<div align="center">

name, price, quantity
"Phone", 100, 1
"Laptop", 1000, 3
"Cable", 10, 5
"Mouse", 50, 5
"Keyboard", 75, 5

</div>

```python
import csv

class Item:
    #Class Level
    pay_rate = 0.8
    all = []
    def __init__(self, name: str, price: float, quantity = 0):
    #Run validations to the received arguments
    assert price >= 0, f"Price is lower than 0"
    assert quantity >= 0, f"Quantity is lower than 0"

    #Assign to self object
    self.name = name
    self.price = price
```

```python
        self.quantity = quantity

        #Actions to execute
        Item.all.append(self)

    def calculate_revenues(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price * self.pay_rate

    @classmethod
    def instantiate_from_csv(cls):
        with open('csv.csv', 'r') as f:
        reader = csv.DictReader(f)
        items = list(reader)

        for item in items:
        print(item)

    def __repr__(self):
        return f"Item({self.name}, {self.price}, {self.quantity})"


Item.instantiate_from_csv()
```

```
Output:
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
{'name': 'Phone', ' price': ' 100', ' quantity': ' 1'}
{'name': 'Laptop', ' price': ' 1000', ' quantity': ' 3'}
{'name': 'Cable', ' price': ' 10', ' quantity': ' 5'}
{'name': 'Mouse', ' price': ' 50', ' quantity': ' 5'}
{'name': 'Keyboard', ' price': ' 75', ' quantity': ' 5'}

Process finished with exit code 0
```

15 Creating instances using the class method:

```python
import csv

class Item:
    #Class Level
    pay_rate = 0.8
    all = []
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
```

```python
        assert price >= 0, f"Price is lower than 0"
        assert quantity >= 0, f"Quantity is lower than 0"

        #Assign to self object
        self.name = name
        self.price = price
        self.quantity = quantity

        #Actions to execute
        Item.all.append(self)

    def calculate_revenues(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price * self.pay_rate

    @classmethod
    def instantiate_from_csv(cls):
        with open('csv.csv', 'r') as f:
        reader = csv.DictReader(f)
        items = list(reader)

        for item in items:
        Item(
        name = item.get('name'),
        price = float(item.get('price')),
        quantity = int(item.get('quantity'))
        )

    def __repr__(self):
        return f"Item({self.name}, {self.price}, {self.quantity})"


Item.instantiate_from_csv()
print(Item.all)
```

```
Output:
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
[Item(Phone, 100.0, 1), Item(Laptop, 1000.0, 3), Item(Cable, 10.0, 5),
    Item(Mouse, 50.0, 5), Item(Keyboard, 74.5, 5)]

Process finished with exit code 0
```

16   Static Method

The static method never sends the object as the first argument.

```python
import csv

class Item:
    #Class Level
    pay_rate = 0.8
    all = []
    def __init__(self, name: str, price: float, quantity = 0):
        #Run validations to the received arguments
        assert price >= 0, f"Price is lower than 0"
        assert quantity >= 0, f"Quantity is lower than 0"

        #Assign to self object
        self.name = name
        self.price = price
        self.quantity = quantity

        #Actions to execute
        Item.all.append(self)

    def calculate_revenues(self):
        return self.price * self.quantity

    def apply_discount(self):
        self.price = self.price * self.pay_rate

    @classmethod
    def instantiate_from_csv(cls):
        with open('csv.csv', 'r') as f:
        reader = csv.DictReader(f)
        items = list(reader)

        for item in items:
        Item(
        name = item.get('name'),
        price = float(item.get('price')),
        quantity = int(item.get('quantity'))
        )

    @staticmethod
    def is_integer(num):
        #We will count out the floats that are point zero.
        #For i.e.: 5.0, 10.0
        if isinstance(num, float):
        #Count out the floats that are point zero
        return num.is_integer()
        elif isinstance(num, int):
        return True
        else:
```

```python
            return False

    def __repr__(self):
        return f"Item({self.name}, {self.price}, {self.quantity})"



print(Item.is_integer(7.5))
print(Item.is_integer(8.0))
```

```
Output:
/usr/bin/python3.10 /home/lucas/workspace/OOP/test.py
False
True

Process finished with exit code 0
```