

Operator Overloading

In this lesson, we will be learning about operator overloading in Python.

We'll cover the following



- Overloading Operators in Python
- Overloading Operators for a User-Defined Class
- Explanation
- Special Functions for Some Common Operators

Overloading Operators in Python

Operators in Python can be overloaded to operate in a certain user-defined way. Whenever an operator is used in Python, its corresponding method is invoked to perform its *predefined* function. For example, when the `+` operator is called, it invokes the special function, `__add__`, in Python, but this operator acts differently for different data types. For example, the `+` operator **adds** the numbers when it is used between two `int` data types and **merges** two strings when it is used between `string` data types.

Run the code below for the implementation of the `+` operator for integers and strings:

```
print(5 + 3) # adding integers using '+'  
print("money" + "maker") # merging strings using '+'
```



Overloading Operators for a User-Defined Class

When a class is defined, its objects can interact with each other through the operators, **BUT** it is necessary to define the behavior of these operators

through operator overloading.

We are going to implement a class that represents a complex number, which has a real part and an imaginary part.

$$\begin{array}{c} \text{Z} \\ \text{┌───┐} \\ \text{└───┘} \\ \text{complex} \\ \text{number} \end{array} = \begin{array}{c} \text{X} \\ \text{┌───┐} \\ \text{└───┘} \\ \text{real} \\ \text{part} \end{array} + i \begin{array}{c} \text{y} \\ \text{┌───┐} \\ \text{└───┘} \\ \text{imaginary} \\ \text{part} \end{array}$$

When we *add* a complex number, the real part is added to the real part and the imaginary part is added to the imaginary part.

Similarly, when we *subtract* a complex number, the real part is subtracted from the real part and the imaginary part is subtracted from the imaginary part.

An example of this is shown below:

$$a = 3 + 7i$$

$$b = 2 + 5i$$

$$a + b = (3 + 2) + (7 + 5)i = 5 + 12i$$

$$a - b = (3 - 2) + (7 - 5)i = 1 + 2i$$

Now let's implement the complex number class and overload the **+** and **-** operators below:

```
class Com:
    def __init__(self, real=0, imag=0):
        self.real = real
        self.imag = imag

    def __add__(self, other): # overloading the '+' operator
        temp = Com(self.real + other.real, self.imag + other.imag)
        return temp

    def __sub__(self, other): # overloading the '-' operator
        temp = Com(self.real - other.real, self.imag - other.imag)
        return temp
```

```
obj1 = Com(3, 7)
```

```
obj2 = Com(2, 5)

obj3 = obj1 + obj2

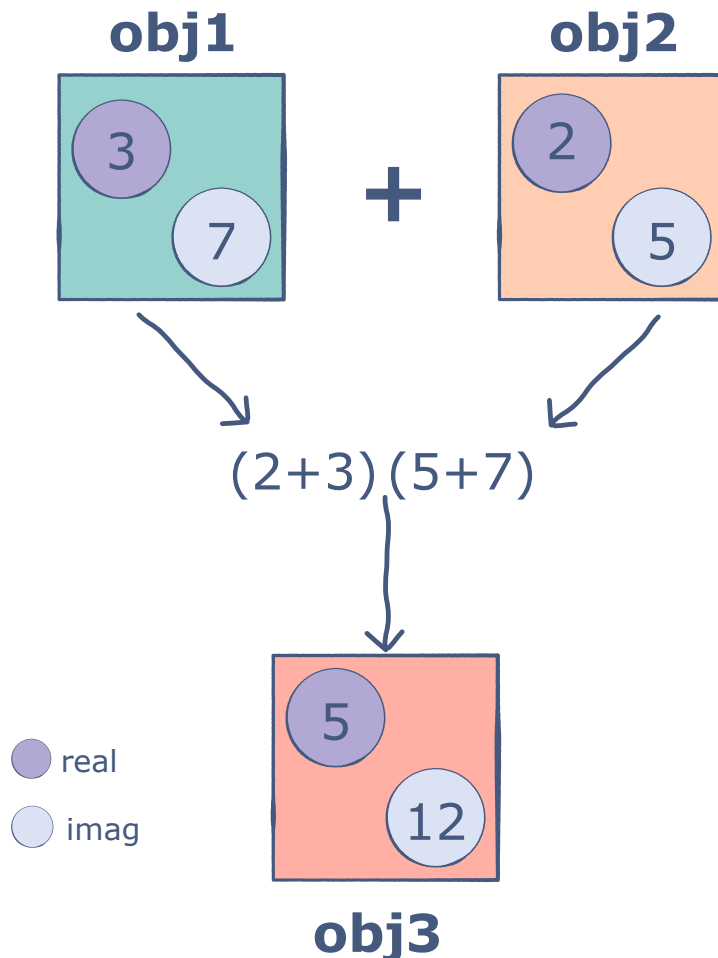
obj4 = obj1 - obj2

print("real of obj3:", obj3.real)
print("imag of obj3:", obj3.imag)
print("real of obj4:", obj4.real)
print("imag of obj3:", obj4.imag)
```



Explanation

- In the above code, we have overloaded the built-in method `__add__` (**line 6**) and `__sub__` (**line 10**) that are invoked when the `+` and the `-` operators are used.
- Whenever two objects of class `Com` are added using the `+` operator, the overloaded `__add__` method is called.
- This method adds the `real` property separately and the `imag` property separately and then returns a new `Com` class object that is initialized by these sums.
- Note that `__add__` and `__sub__` methods have *two* input parameters. The first one is `self`, which we know is the reference to the class itself. The second parameter is `other`. `other` is a reference to the *other* objects that are interacting with the class object.
- In **line 18**, `obj2` will be considered the other object, the operator will be called on the `obj1` object and the returned object will be stored in `obj3`.
- In **line 19**, `obj2` will be considered the other object, the operator will be called on the `obj1` object and the returned object will be stored in `obj4`.
- Other has `Com` class attributes and thus, it has the `real` and the `imag` properties.



You can name the second argument to be anything, but as per convention, we will be using the word **other** to reference the *other* object.

Similarly, whenever two objects of class **Com** are subtracted using the **-** operator, the overloaded **__sub__** method is called. This method subtracts the **real** property separately and the **imag** property separately and then returns a new **Com** class object that is initialized by these differences.

Special Functions for Some Common Operators

Below are some common special functions that can be overloaded while implementing operators for objects of a class.

Operator	Method
+	<code>__add__(self, other)</code>

-	<code>__sub__(self, other)</code>
/	<code>__truediv__(self, other)</code>
*	<code>__mul__(self, other)</code>
<	<code>__lt__(self, other)</code>
>	<code>__gt__(self, other)</code>
==	<code>__eq__(self, other)</code>

It is for the user to decide how they want the objects to interact when an operator *operates* on them, but they usually make sure that these operations make sense. For example, the `+` is not going to be used for finding products of different properties of a class.

This is it for operator overloading in Python. Now let's learn how to implement polymorphism using duck typing in Python.