DataScientest • com

# Object Oriented Programming

## Classes

### Introduction and prerequisites

In Python and many other programming languages, *object-oriented programming* consists in creating *classes* of *objects* that contain specific information and tools suitable for their handling.

All the tools we use for data science (*DataFrames*, *scikit-learn models*, *matplotlib*, ...) are built in this way. Understanding the mechanics of Python objects and knowing how to use them is essential to exploit all the features of these very useful tools.

Furthermore, object-oriented programming gives the developer the flexibility to readapt an object to his needs thanks to the *inheritance* that we will see in the second part. Indeed, this technique is widely used to develop packages such as **scikit-learn** which allow a user to easily develop and evaluate the models he needs.

To approach these modules in the best possible conditions, it is important to have completed the *Introduction to Python Programming* module.

### Introduction to Classes

In Python, a class is defined as follows:

```
    class Vehicle: # definition of the class Vehicle
       def __init__ (self, a, b = []):
           self.seats = a                # number of seats in the vehicle
           self.passengers = b           # list containing passenger names

       def print_passengers (self):
           for i in range (len (self.passengers)):
               print (self.passengers [i])
               ```
``` py
car1 = Vehicle(4, ['Pierre', 'Adrian']) # instantiation of an object from the Vehicle class
```

The code above corresponds to the definition of a class named `Vehicle` which contains 2 pieces of information: the number of seats of the `Vehicle` in the variable **`seats`** and the names of the passengers onboard the `Vehicle` in the variable **`passengers`** .

This class contains a `print_passengers` *method* which displays the names of the passengers onboard in the console.

The instruction `car1 = Vehicle (4, ['Pierre', 'Adrian'])` corresponds to the *instantiation* of the class `Vehicle` .

#### Important Notes and Definitions

- `Vehicle` is a *class* of objects.
- `car1` is an *instance* of the `Vehicle` class.
- `seats` and `passengers` are called the **attributes** of the class `Vehicle` .
- The functions defined in the `Vehicle` class like `print_passengers` and `__init__` are called the **methods** of the `Vehicle` class.
- The `__init__` method takes as arguments the variables that will define the attributes of an instance when it is created.

> ℹ The `__init__` method is automatically called when instantiating any class.

> ⚠ All the methods defined within a class have the `self` argument as their first parameter. This parameter is used to specify the instance which called the method.

Based on syntax from the `Vehicle` class defined above:

- **(a)** Define a new `Complex` class with 2 attributes:
  - **`part_re`** which contains the real part of a complex number.
  - **`part_im`** which contains the imaginary part of a complex number.
- **(b)** Define in the `Complex` class a `display` method which prints a `Complex` in its algebraic form *a* ± *bi*. This method should adapt to the sign of the imaginary part (The method should be able to display `4 - 2i` , `6 + 2i` , `5` ,...).
- **(c)** Instantiate two `Complex` objects corresponding to the complex numbers $4 + 5i$ and $3 - 2i$, then print them on the console.

In [20]:

```
### Insert your code here

class Complex:

    def __init__(self, part_re, part_im):
        self.part_re = part_re
        self.part_im = part_im
        self.complex_number = complex(self.part_re, self.part_im)

    def display(self):
        print(self.complex_number)

c1 = Complex(4,5)
c2 = Complex(3,-2)

# Call the display method
Complex.display(c1)
c1.display()
c2.display()

print('\npart_re :', c1.part_re)
print('part_im :', c1.part_im)
```

```
(4+5j)
(4+5j)
(3-2j)

part_re : 4
part_im : 5
```

Show solution

> Once an object of a class is instantiated, it is possible to access its attributes and methods using the `.attribute` and `.method()` commands as shown below:

In [21]:

```
class Vehicle:
    def __init__(self, a, b=[]):
        self.seats = a
        self.passengers = b
    def print_passengers(self):
        for i in range(len(self.passengers)):
            print(self.passengers[i])


# Run the cell. You can modify the instantiation so that the changes are reflected.
car2 = Vehicle(4,['Dimitri', 'Charles', 'Yohan'])

print(car2.seats)          # Display of the 'seats' attribute
car2.print_passengers()    # Calling of the print_passengers method
```

```
4
Dimitri
Charles
Yohan
```

> The flexibility of classes in object-oriented programming allows the developer to broaden a class by adding new attributes and methods to it. All instances of this class will then be able to call these methods.
>
> For example, we can define in the `Vehicle` class a new `add` method which will add an individual to the passenger list:

```
class Vehicle:
    def __init__(self, a, b = []):
        self.seats = a
        self.passengers = b

    def print_passengers(self):
        for i in range (len(self.passengers)):
            print (self.passengers[i])

    def add(self, name):            #New method
        self.passengers.append(name)
```

> In Python, a list is an instance of the built-in `list` class. Thus, calling the `append` method is done in the same way as calling a method from the `Vehicle` or `Complex` classes.

In [22]:
```
class Vehicle:
    def __init__(self, a, b=[]):
        self.seats = a
        self.passengers = b

    def print_passengers(self):
        for i in range(len(self.passengers)):
            print(self.passengers[i])

    def add(self,name): #New methd
        self.passengers.append(name)


car1 = Vehicle(4, ['Charles', 'Paul'])  # Instantiation  of car1
car1.add('Raphaël')                     # 'Raphaël' is added to the list of passengers

car1.print_passengers()             # Display of the list of passengers
```

```
Charles
Paul
Raphaël
```

- **(d)** Define in the `Complex` class an `add` method which takes as argument a `Complex` object and adds it to the instance calling the method. The result of this sum will be stored in the attributes of the `Complex` calling the method.
- **(e)** Test the new `add` method on two instances of the `Complex` class and display their sum.

In [1]:
```
6+2j
```

Out[1]: (6+2j)

In [37]:
```
#  Solution 1
class Complex:

    def __init__(self, part_re, part_im):
        self.part_re = part_re
        self.part_im = part_im
        self.complex_number = complex(self.part_re, self.part_im)

    def display(self):
        print(self.complex_number)

    def add(self , c):
        self.c = complex(c)
        c_sum = self.c + self.complex_number
        print(c_sum)

c1 = Complex(4,5)
c2 = Complex(3,-2)

c1.add(10+20j)
c1.add(c2.complex_number)
```

```
(14+25j)
(7+3j)
```

In [59]:
```
#  Solution 2
class Complex:
    def __init__(self, a, b):
        self.part_re = a
        self.part_im = b

    def display(self):
        if(self.part_im < 0):
            print(self.part_re,'-', -self.part_im,'i')
        if(self.part_im == 0):
            print(self.part_re)
        if(self.part_im > 0):
            print(self.part_re, '+',self.part_im,'i')

    def add(self , c):
        if (type(c) == complex or type(c) == int or type(c) == float):
            self.c1 = complex(self.part_re, self.part_im)
            c_sum = self.c1 + c
            print(c_sum)
        else:
            self.part_re = self.part_re + c.part_re
            self.part_im = self.part_im + c.part_im
            print('Total = ', complex(self.part_re, self.part_im))

c1 = Complex(4,5)
c2 = Complex(3,-2)

c1.add(10+10j)
c1.add(c2)
```

```
(14+15j)
Total =  (7+3j)
```

In [25]:

```
class Complex:
    def __init__(self, a, b):
        self.part_re = a
        self.part_im = b

    def display(self):
        if(self.part_im < 0):
            print(self.part_re,'-', -self.part_im,'i')
        if(self.part_im == 0):
            print(self.part_re)
        if(self.part_im > 0):
            print(self.part_re, '+',self.part_im,'i')

    def add(self , c):
        raise NotImplementedError  #### Insert your code here
```

Show solution

Unvalidate