Basic Programming

Lesson 12. Object-Oriented Programming

Alexandra Ciobotaru

14 ian 2022

Syllabus

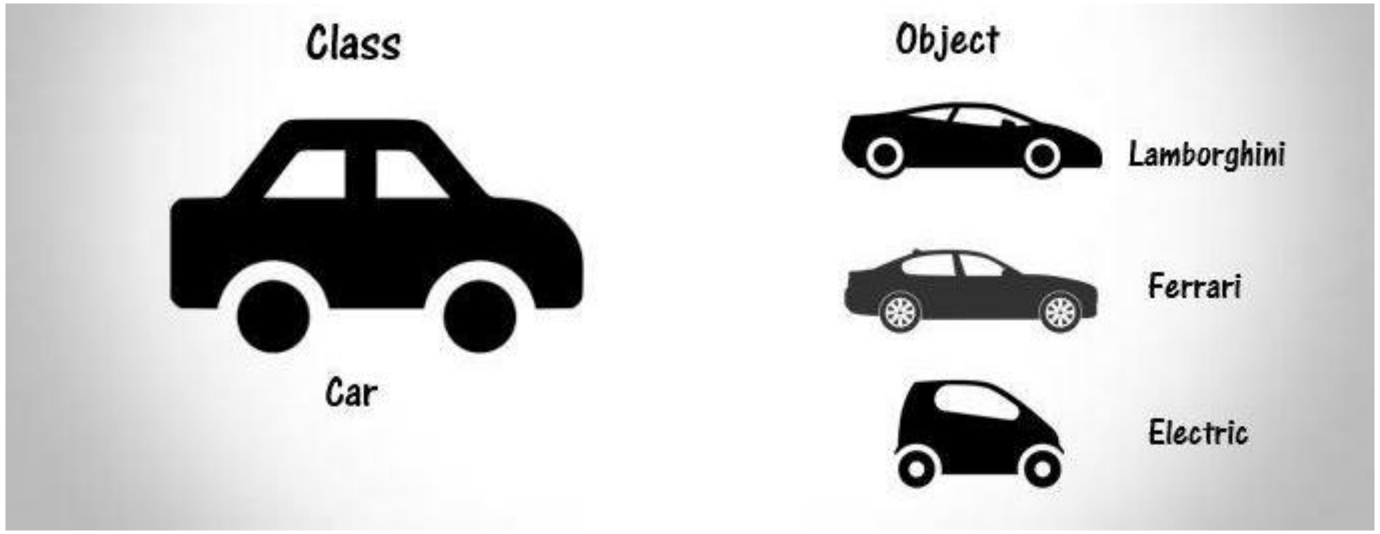
- **Lesson 1. Computers, Programming and Cognitive Science. From pseudocode to programming languages.**
- **Lesson 2. Variables in Python. Basic calculus. Using Math library, type() and help() functions.**
- **Lesson 3. Collections: Lists, Tuples.**
- **Lesson 4. Strings. Working with strings.**
- **Lesson 5. Branching and decisions: Logical operators, If-Statements, Nested conditions. Loops: For and While**
- **Lesson 6**. **Lesson 5 continued** Quiz 1 (25%).
- **Lesson 7. Creating functions. Recursive functions. Matrices**.
- **Lesson 8. Collections: Dictionaries. JSON construction.**
- **Lesson 9. Working with files. Reading and Writing.**
- **Lesson 10. Analyzing dataframes. Pandas and Matplotlib Python libraries.**
- **Lesson 12. Lesson 10 continued. Error handling.** Quiz 2 (25%).
- Lesson 13. Object-Oriented Programming: Encapsulation, Inheritance and Polymorphism.
- Lab (20%) + final exam (30%).

What you’ll learn today:

- 1. What is a Python class
- 2. Classes define Objects
- 3. Encapsulation
- 4. Inheritance
- 5.Polimorphism

1. What is a Python class

- Python is OOP = Object Oriented Programming.
- Almost everything in Python is an object, with its properties and methods.
- A **Class** is a "blueprint" for creating objects.



We use classes to create Objects.

Image source: http://www.trytoprogram.com/python-programming/python-object-and-class/

2. Classes define Objects

- An object has:
  - ***Attributes*** – data about the object (name, color, speed etc.)
  - ***Methods*** – functionalities or tasks that the object can do (eg. how it adjusts the temperature inside)
- Let's define a class:

```
class Fruit:
    def __init__(self):
        self.name = "apple"
        self.colour = "red"
my_fruit = Fruit()
print(my_fruit.name)
print(my_fruit.colour)
my_fruit.colour = “green”
```

We can create **attributes** by using the self parameter, followed by the name of the attribute, and then we can assign to it a value of our choice.

We create a **Fruit object** and assign it to a variable, my_fruit

We can assess the attribute by typing the object followed by the name of the attribute

We create also update attributes

- But it seems that my class creates only apples!

- We can also define a class with parameters in the initialization, just like we did when defining functions:

```
class Fruit:
    def __init__(self, name, clr):
        self.name = name
        self.colour = clr
apple = Fruit("apple", "red")
banana = Fruit("banana", "yellow")
kiwi = Fruit("kiwi", "green")
print(apple.colour)
```

Now we assigned parameters to our attributes

Whenever we create a Fruit object we need to pass in two arguments, the name and colour

- We can also delete a property from an object:

```
class Fruit:
    def __init__(self, name, clr, prc):
        self.name = name
        self.colour = clr
        self.price = prc
apple = Fruit("apple", "red", 2)
del apple.price
print(apple.price)
```

-> AttributeError: 'Fruit' object has no attribute 'price'

Object methods

- Methods are functionalities of objects – functions, in short, related to objects

```
class Fruit:
    def __init__(self, name, clr):
        self.name = name
        self.colour = clr
    def details(self):
        print("my " + self.name + " is " + self.colour)
apple = Fruit("apple", "red")
print(apple.details())
```

- Self parameter ensures that all attributes are accessible by all methods in the same object.

__init__

- Initialization is important because:
  - Here we initialize all attributes that must be accessible accross the class;
  - It automatically executes whenever we instantiate an object of that class;
  - It also has a reserved name.

```
def __init__(self, name, clr):
        self.name = name
        self.colour = clr
```

3. Encapsulation

- We can restrict the acces to our variables and methods
- Why? To prevent the data from being modified accidentaly
- There are two types of access specifiers: private and public
- In our example, the method is public, it can be accessed outside the class aslo:

```
class Fruit:
    def __init__(self, name, clr):
        self.name = name
        self.colour = clr
    def details(self):
        print("my " + self.name + " is " + self.colour)
apple = Fruit("apple", "red")
print(apple.details())
```

- Private methods are not accessible outside the class

- **Private methods**

```
class car:
    def __init__(self):
        # in the initialization method we call the private 
method updatesoftware
        self.__updatesoftware()
    def drive(self):
        print("driving")
    def __updatesoftware(self):
        print("updtating sw")
# we create a car object
blackcar = car()
blackcar.drive()
```

- If I try to call the private method __updatesoftware directly, it will give an error:

```
blackcar.__updatesoftware()
```

- Private variables – can be modified only inside the class, we can’t modify them outside the class

```
class car:
    # private variables
    __maxspeed = 0
    __name = " "
    def __init__(self):
        # private variables in the initialization
        self.__maxspeed = 200
        self.__name = "supercar"
    def drive(self):
        print("driving")
        print(self.__maxspeed)
    def setspeed(self, speed):
        self.__maxspeed = speed
        print(self.__maxspeed)

# we create a car object
redcar = car()
redcar.drive()
redcar.setspeed(100)
redcar.__maxspeed = 100
redcar.drive()
```

(this action doesn’t modify my variable)

4. Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.
- Any class can be a parent class:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
#Use the Person class to create an object, and then execute the printname
method:
x = Person("John", "Doe")
x.printname()
-> John Doe
```

Create a child class

- To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class.
- Example: the class named Student will inherit the properties and methods from the Person class:

```
class Student(Person):
  pass
y = Student("John", "Doe")
y.printname()
-> John Doe
```

- We used the pass keyword because we didn’t add any other properties or methods to the class.

5. Polimorphism

Poly + morphism = many + having different forms
-> changing from one form to many different forms

```
class French:
    def say_hello(self):
        print('Bonjour')
class Chinese:
    def say_hello(self):
        print('Ni hao')
def intro(lang):
    '''
    Function that takes as an input an object and applies to it the method say_hello(), if
the object has that method.
    :param lang: object
    :return: whatever say_hello() method returns
    '''
    lang.say_hello()
alex = French()
ander = Chinese()
intro(alex)
```

But if I add:

```
class Something:
    def count_me(self, x):
        return len(x)

some = Something()
intro(some)
```

Thank you!