DataScientest • com

Python for Data Science

Classes and modules

Introduction

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 manipulation.

All the tools that we use for Data Science and that you will see later (numpy arrays, pandas DataFrames, scikit-learn models, matplotlib figures, ...) are built this way. Understanding the mechanics behind Python objects and knowing how to use them is essential to make the most of these tools.

1. Classes

An object class contains 3 **fundamental** elements:

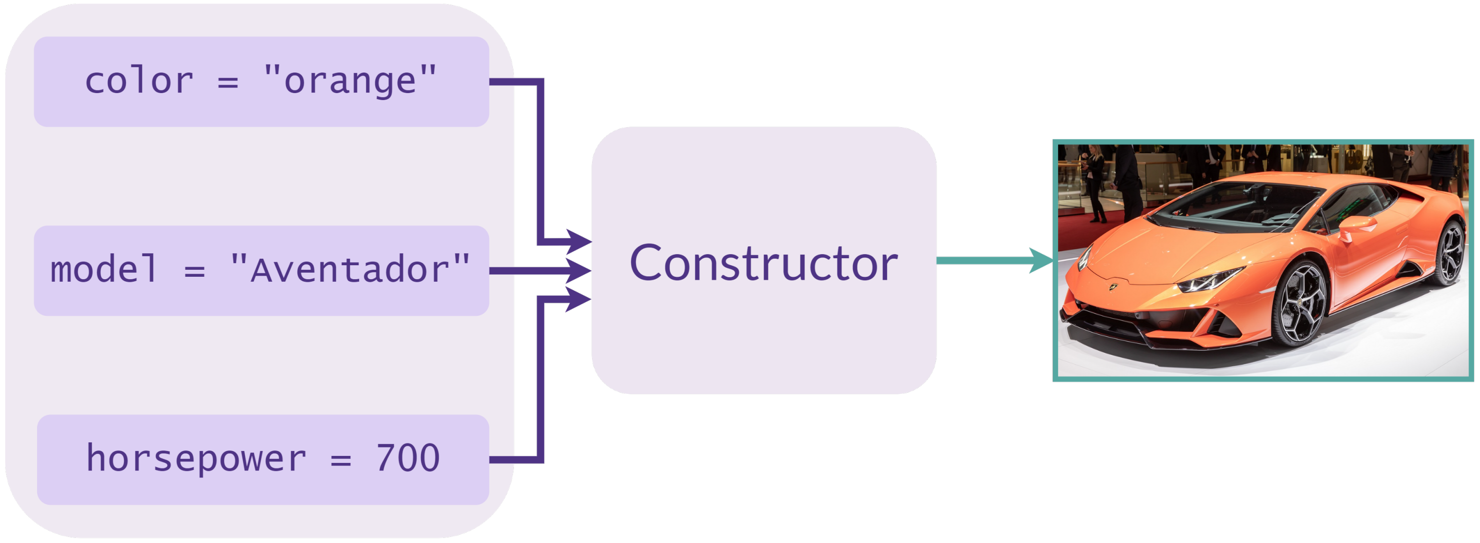
- A **constructor**: a function that **initializes** an object of the class.
- **Attributes**: Variables **specific** to the created object used to define its **properties** and store information.
- **Methods**: Class-specific functions that allow you to interact with an object.

To understand object classes, let us make an analogy with a car.

Cars are a class of objects designating motorized and wheeled vehicles intended for land transport.

Assume that the **constructor** of the class of cars is a **factory** which produces them. From the **attributes** desired for the car such as its **color**, its **model** or its **horsepower**, the factory must be able to produce a car that matches the given specifications.

Thus, the factory is analogous to a **function** which takes as **argument** the **attributes** of the car and **returns** the car **built** and ready to use.



The **methods** of the car are the commands to accelerate or brake the vehicle. These commands are analogous to **functions** which take as argument the **pressure** on the pedal and **apply** an acceleration to the car.

In Python, a class is defined with the `class` keyword. This keyword allows us to start a block of code where we can define the **constructor** of the class and its **methods**.

The definition of the constructor is done with the method named `__init__` which allows us to initialize the **attributes** of the object that is being built (note that this method takes **two** underscores before **and** after `init`):

```python
# Definition of the Car class
class Car:
# Definition of the constructor of the Car class
    def __init__(self, color, model, horsepower):
# Initialization of the attributes of the class with the arguments of the constructor
        self.color = color
        self.model = model
        self.horsepower = horsepower
```

The `self` argument corresponds to **the object calling the method**. This argument allows us to access the attributes of the object within the method.

All the methods of a class must have as **first argument** in their definition the argument `self` because the methods of a class **systematically** receive the object which calls them as the first argument.

We can define other methods within this class:

```python
# Definition of the Car class
class Car:
     # Definition of the constructor of the Car class
    def __init__(self, color, model, horsepower):
        # Initialization of the attributes of the class with the arguments of the constructor
        self.color = color
        self.model = model
        self.horsepower = horsepower

    # Definition of a method to change the color of a car
    def change_color(self, new_color):
       self.color = new_color
```

To create an object of this class, we must use the following syntax:

```python
# Creation of an object of the Car class
aventador = Car(color = "orange",
                model = "Aventador",
                horsepower = 700)
```

In [4]:

```python
# Insert your code here

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

    def display(self):
        print(complex(self.part_re, self.part_im))

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

Complex.display(c1)
Complex.display(c2)

c1.display()
c2.display()
```

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

Hide solution

In [ ]:

```python
# Definition of the Complex class
class Complex:

    # Definition of the constructor
    def __init__(self, a, b):
        """
        Builds a complex number.

        Parameters:
        -----------
        a : real part
        b : imaginary part
        """
        # Initialization of the attributes with the constructor's arguments
        self.part_re = a
        self.part_im = b

    def display(self):
        """
        Displays a complex number in its algebraic form a + bi
        """
        # We must pay attention to the sign of the imaginary part to not
        # print something like "4 + -5i" if the imaginary part is negative
        # or "4 + 0i" if the imaginary part is 0
        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')

# Instantiation of complex numbers 4 + 5i and 3-2i
C1 = Complex(4, 5)
C2 = Complex(3, -2)

# Display
C1.display()
C2.display()
```

2. Classes and Documentation

To be usable by others, a class must always be **properly documented**.

As with functions, writing the documentation for a class begins and ends with three quotes `"""` :

```python
class Car:
    """
    The Car class allows you to build a car.

    Parameters:
    ----------
    color: Character string: Color of the car.
    model: Character string: Model of the car.
    horsepower: Integer: Displacement of the car.

    Example
    -------
    aventador = Car (color = "orange", model = "Aventador", horsepower = 700)
    """
    def __init__(self, color, model, horsepower):
        self.color = color
        self.model = model
        self.horsepower = horsepower

    def change_color(self, new_color):
        """
        Changes the color of a car.

        Parameters:
        ----------
        new_color: Character string: New color of the car.
        """
        self.color = new_color
```

Now, when another user needs help using this class, they can use the **`help`** function to display its documentation:

```python
help(Car)
```

```
class Car(builtins.object)
 |  Car(color, model, horsepower)
 |
 |  The Car class allows you to build a car.
 |
 |  Parameters:
 |  ----------
 |  color: Character string: Color of the car.
 |  model: Character string: Model of the car.
 |  horsepower: Integer: Displacement of the car.
 |
 |  Example
 |  -------
 |  aventador = Car(color = "orange", model = "Aventador", horsepower = 700)
 |
 |  Methods defined here:
 |
 |  __init__(self, color, model, horsepower)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  change_color(self, new_color)
 |      Changes the color of a car.
 |
 |      Parameters:
```

In [2]:

```python
u = [1, 9, -3, 3, -5, 4, -4, 7, 3, 4, 5, 0, 8, 7, -1, -3, 7, 6, 0, 2]

# We can display the documentation of a class from
# an instance or its constructor
#help(u)

# Insert your code here
print(u.sort(), u)
print(u.clear(), u)
```

```
None [-5, -4, -3, -3, -1, 0, 0, 1, 2, 3, 3, 4, 4, 5, 6, 7, 7, 7, 8, 9]
None []
```

Hide solution

In [37]:

```
u = [1, 9, -3, 3, -5, 4, -4, 7, 3, 4, 5, 0, 8, 7, -1, -3, 7, 6, 0, 2]
print(" u :\n", u)

# We sort u in ascending order
u.sort()
print("\n u sorted:\n", u)

# We remove all elements from u
u.clear()
print("\n u empty: \n", u)
```

```
 u :
 [1, 9, -3, 3, -5, 4, -4, 7, 3, 4, 5, 0, 8, 7, -1, -3, 7, 6, 0, 2]

 u sorted:
 [-5, -4, -3, -3, -1, 0, 0, 1, 2, 3, 3, 4, 4, 5, 6, 7, 7, 7, 8, 9]

 u empty: 
 []
```

3. Modules

A module (also known as *package* or *library*) is a Python file containing definitions of classes and functions.

Modules allow you to reuse functions already written without having to copy them.

Modules are easily shareable and in general modules specialize in very specific tasks such as:

- Database management (`pandas`)
- Optimized computation (`numpy`)
- Graphs plotting (`matplotlib`)
- Machine learning (`scikit-learn`)

All the Data Science tasks that you will learn in your training rely on modules written by other developers. These modules are among the greatest assets of the Python language and the absence of these modules would make Python a sub-optimal language for Data Science.

There are many programming languages which are fundamentally superior to Python. However, as long as the modules we use are not available in those languages, it will be very difficult to migrate.

In order to import a module, you must use the **`import`** keyword:

```
# We import the entire Numpy library
import numpy
```

To use a function of this module, it must be accessed through the module:

```
x = 0

# The 'cos' function of numpy allows you to compute the cosine of a number
print(numpy.cos(x))
>>> 1.0
```

It is not very practical to have to write `numpy` every time we want to use a function from this module. For that, we can **abbreviate** its name using the keyword **`as`** :

In [45]:

```
# Insert your code here
from sklearn.datasets import load_boston
from sklearn.datasets import fetch_openml

boston_dataset = load_boston()
print(boston_dataset.data[:2], '\n target: \n', boston_dataset.target[:2],
      '\n feature names: \n', boston_dataset.feature_names)

mice = fetch_openml(name='miceprotein')
mice.data.shape
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 0.0000e+00 5.3800e-01 6.5750e+00
  6.5200e+01 4.0900e+00 1.0000e+00 2.9600e+02 1.5300e+01 3.9690e+02
  4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 6.4210e+00
  7.8900e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9690e+02
  9.1400e+00]]
 target:
 [24.  21.6]
 feature names:
 ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

Out[45]:

```
(1080, 77)
```

Hide solution

In [ ]:

```
# We import the load_boston function from the sklearn.datasets module
from sklearn.datasets import load_boston

# We run the load_boston function and store its
# output in the boston_dataset variable
boston_dataset = load_boston()

#print(boston_dataset)
```

The `boston_dataset` variable is a **dictionary**. Remember that a dictionary is a data structure where the data is indexed by **keys**. To access a key from a dictionary, all you have to do is enter the key in square brackets:

```
a_dictionary['key']
```

The `boston_dataset` dictionary contains 4 keys:

- `'data'` : The Boston real estate dataset. It contains characteristics about real estate properties in Boston.
- `'target'` : The price of these properties. The objective of the dataset is to determine the selling price of a property based on its

In [27]:

```
# Insert your code here

X = boston_dataset.data
feature_names = boston_dataset.feature_names
target = boston_dataset.target
descr = boston_dataset.DESCR
```

Hide solution

In [28]:

```
X = boston_dataset['data']

feature_names = boston_dataset['feature_names']
```

We will now instantiate an object of the **`DataFrame`** class which is very useful for visualizing and processing datasets.

- **(e)** Import the `pandas` module under the alias **`pd`**.
- **(f)** Instantiate an object of the `DataFrame` class using the `pd.DataFrame()` constructor. The object built will be named **`df`** and the constructor's arguments will be **`data = X, columns = feature_names`**.
- **(g)** Display the first 10 lines of the `DataFrame` `df` by calling its **`head`** method with the argument **`n = 10`**.

In [30]:

```python
# Insert your code here
import pandas as pd

df = pd.DataFrame(data=X, columns=feature_names)
df.head(10)
```

Out[30]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10

Hide solution

In [46]:

```python
# We import the pandas module under the alias pd
import pandas as pd

# Instantiate a DataFrame object with its constructor
df = pd.DataFrame(data = X, columns = feature_names)

# Display the first 10 lines of the DataFrame using the head method
df.head(n = 10)
```

Out[46]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10

> You have just imported and displayed your first dataset using the `sklearn.datasets` and `pandas` modules.
>
> As you will see in the rest of your training, the `DataFrame` class of `pandas` is a much more practical class than the list or dictionary class for handling tabular data. It is one of the fundamental tools for data analysis with Python.

Conclusion

> You have now acquired the basics of Python. You can now discover the excellent modules available for Data Science and Analysis.
>
> The rest of your training will focus on the introduction of specialized modules for Data Science. This will give you the best existing tools for data processing and analysis and will allow you to carry out your data projects.