‘Data Analysis’
Computer Lab Session
General Introduction to Data Analysis with Python

Master 1 MLDM / CPS^2 / COSI / 3DMT
Saint-Étienne, France

Ievgen Redko

1 Outline

1. Introduction to Python (1/2)
2. **Introduction to Python (2/2)**
3. Probability, Random Variables and Probability Distributions
4. Linear Algebra (1/2)
5. Linear Algebra (2/2)
6. Principal Component Analysis
7. Linear Regression (1/2)
8. Linear Regression (2/2)
9. Clustering (1/2)
10. Clustering (2/2)

2 Functions

The key word **def** allows to define a function. The key-word **return** halts its execution and defines the output value.

```
def fibonacci(n_terms):
    fib = [1, 1]
    if n_terms < 2:
        return fib[0:n_terms]

    for i in range(n_terms-2):
        element = fib[-2] + fib[-1]
        fib.append(element)
```

```
    return fib
```

It is also possible to have a function **without the return** value when the result is shown directly during the execution.

```
def show_fibonacci(n_terms):
    for terme in fibonacci(n_terms):
        print(terme)

show_fibonacci(12)
```

We can also define the default values of a function as follows.

```
def fibonacci_general(n_terms, first_term=1, second_term=1):
    fib = [first_term, second_term]
    if n_terms < 2:
        return fib[0:n_terms]

    for i in range(n_terms-2):
        element = fib[-2] + fib[-1]
        fib.append(element)

    return fib
```

You can now call this function using the following commands:

```
fibonacci_general(12)
fibonacci_general(12, .5)
fibonacci_general(12, 2, 2)
fibonacci_general(12,  second_terme=3)
```

3 List comprehension

There is a particular way of defining a function that applies to all values of a given list. This is called **list comprehension**.

```
[len(word) for word in list_fib_ab]
[count_a_et_b(mot) for mot in liste_fib_ab]
[x ** 2 for x in range(10)]
# We can also add a condition to filter out some values depending on it
[x ** 2 for x in range(10) if x % 2 == 0]
```

4 Classes

Python allows to code in object-orientated fashion as well. For this, we have to first define a class given by a structure representing a certain concept. In Python, defining a class is equivalent to defining a new type of objects. It enables us to further instantiate different values of this newly created type.

Below, we define the class for the vector object.

```
class vector:
    def __init__(self, size, value_initial=0):
```

```
        self.size = size
        self.elements = [value_initial] * size

    def show(self):
        print(self.elements)

    def copy(self):
        new = vector(self.size)
        new.elements = self.elements.copy()
        return new

    def access(self, index):
        return self.elements[index]

    def modif(self, index, value):
        self.elements[index] = value

    def addition(self, value):
        for i in range(self.size):
            self.elements[i] += value

    def norm(self):
        cumul = 0.0
        for x in self.elements:
            cumul += x ** 2
        return cumul ** 0.5
```

Some comments:

- The class is defined using the key-word **class**
- A class has a certain numbers of its methods defined with the key-word **def**
- The first parameter of each methods is the **self** variable. When an object is created, the key-word self points explicitly to the current object.
- __init(self)__ method is called by Python when the object is created. We use this method to initialize the different attributes of the object.
- The attributes are created (and modified) by simply writing self.name_of_attribute = value

We can now use this class to create objects of type vector.

```
vec_v = vecteur(10)
type(vec_v)
vec_v.size
vec_v.elements

# calling different method
vec_v.show()
vec_v.norm()
vec_v.modif(2, 3)
vec_v.show()
vec_v.norm()
```

5 Libraries and plots

A Python library is a collection of modules. For instance, the list of all the modules of the matplotlib library can be found here:

`https://matplotlib.org/py-modindex.html`

Below, we illustrate the use of this library via the use of its module, pyplot that allows to create figures.

```
from matplotlib import pyplot as plt

# We can also write the same line as follows
# import matplotlib.pyplot as plt

# Create a figure of a given size
plt.figure(figsize=(16, 4))
values_x = range(1, 100) # define the list of axis-x values
values_y = [sqrt(x) for x in values_x] # define the list of axis-y values

plt.scatter(values_x, values_y, label='Square root') # scatter plot
plt.plot(values_x, [log(x) for x in values_x], label='Logarithm') # ordinary plot
plt.legend() # figure legend
plt.show() # show the resulting figure
```

For other examples on how to use matplotlib.pyplot please refer to

`https://matplotlib.org/2.2.3/gallery/index.html#pyplots-examples`