3iL INGENIEURS
</les sciences informatiques>

Data Science Lecture

By Raymond Houé Ngouna

(raymond.houe-ngouna@enit.fr)

Python basics and funndamentals: modules and packages

Lecture objectives

To know the programming mechanisms allowing the **reuse of code**, in particular those implemented for scientific calculation, and to learn how to use them.

Main concepts covered:

- **Modules**.
- **Packages**.

In order to be able to use a module or package, it should be imported (into the project where it is to be used):

- `import nom_module`
- `import nom_package`

Useful link

Python 3 online Documentation

Module

- A **module** can be considered as a library of source code.
- This is a file containing a set of functions that can be included (used) in an application.
- Syntax: `import my_module`

Pratical usage:

- Let's assume that we have created a module containing, among other things, the function to calculate the Fibonacci sequence and to display it as a list → the underlying code will be created in a Python file (.py), let's call it `mon_fibonacci.py` .
- One can then (re)use this code i.e. this module in a new project (the current file for example).

Package

- A **package** is a set of different modules.
- An example, the module `pyplot` contained in the package `matplotlib`
  - This is a module to generate and manipulate graphics in the manner of MATLAB.
  - Syntax: `import matplotlib.pyplot`

Note:

- After importing a module, a folder is created: `__pycache__`
- This "hidden" folder manages all the connections established with the imported modules.

Documentation on the matplotlib package: https://matplotlib.org/py-modindex.html

Documentation on the module pyplot : https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot

Basic usages

Examples

```
# we assume that the file containing the computation of a Fibonaccci sequence is created
# the file my_fibonacci.py contains the two following functions
"""def fib(n):
    a = 0
    b = 1
    for i in range(n-1):
        print(a),
        a, b = b, a+b"""

"""def fib_list(n=10):
    a = 0
    b = 1
    res = []
    for i in range(n-1):
        res.append(a)
        a, b = b, a+b
    return res"""

# let's import it
import my_fibonacci
```

```
# call of the first function of the module
my_fibonacci.fib(8)
```

```
# call of the first function of the module => returns the result as a list
my_fibonacci.fib_list(8)
```

```
# import using ans alias
# to make it easier to call the module, it can be given an alias
import my_fibonacci as fib
```

```
fib.fib(8)
```

```
fib.fib_list(8)
```

Import only a sub-part of a module

For large modules, in order to mitigate memory issues due to the loading of the whole module, it is possible to import only a few functions.

Syntax: `from my_big_module import specific_function`

```
from my_fibonacci import fib_list
```

```
# only the function fib_liste is accessible
fib_list(8)
```

```
# the second function of our module, fib(), is not
fib(8)
```

Importing all functions once

Syntax: `from my_project import *`

```
# let us import all functions of our module => fib_liste() and fib()
from my_fibonacci import *
```

```
# fib_liste() is accessible
fib_list(8)
```

```
# fib() also
fib(8)
```

Basics Python modules

See the documentation here: http://docs.python.org/3/py-modindex.html

Illustration on some built-in functions

```
# computation of the square root -> let's use the function sqrt() directly
# it is not accessible since the associated module is not yet imported
sqrt(4)
```

```
# the required module to use the sqrt() function is the 'math' module
# let's import it to make the square root function accessible
import math

math.sqrt(4)
```

Examples of computations with few basic built-in functions

```
import math
import random
import statistics
import os
```

```
###################################
#--------------MATH---------------#
###################################

# Pi constant
print('Pi:',math.pi)

# compute cos(pi)
print('cos(pi):',math.cos(math.pi))
```

```
###################################
#----------STATISTIQUES-----------#
###################################

list_fib = fib_list(9)
print('liste:',list_fib)
print('mean:',statistics.mean(list_fib))
print('median:',statistics.median(list_fib))
print('mode:',statistics.mode(list_fib))
print('variance:',statistics.variance(list_fib))
```

```
###################################
#-------------RANDOM--------------#
###################################
```

Main methods of the random module:

- `choices()` returns a list with a random selection in the given sequence
- `choice()` returns a random element of the given sequence
- `seed()` initialises the random number generator
- `random()` returns a random floating-point number between 0 and 1
- `uniform()` returns a random floating number between the two given parameters
- `randrange()` returns a random number between the given range
- `randint()` returns a random number between the given range
- `shuffle()` takes a sequence and returns the sequence in a random order
- `sample()` returns a given sample of a sequence

Other useful methods

- `getstate()` returns the current internal state of the random number generator
- `setstate()` restores the internal state of the random number generator
- `getrandbits()` returns a number representing random bits
- `triangulaire()` returns a random floating-point number between two given parameters, you can also set a mode parameter to specify the midpoint between the other two values
- `betavariate()` returns a random floating number between 0 and 1 based on the Beta distribution (used in statistics)
- `expovariate()` returns a random floating-point number between 0 and 1, or between 0 and -1 if the parameter is negative, based on the exponential distribution (used in statistics)
- `gammavariate()` returns a random floating number between 0 and 1 based on the Gamma distribution (used in statistics)
- `gauss()` returns a random floating number between 0 and 1 based on the Gaussian distribution (used in probability theories)
- `lognormvariate()` returns a random floating-point number between 0 and 1 based on a log-normal distribution (used in probability theories)
- `normalvariate()` returns a random floating number between 0 and 1 based on the normal distribution (used in probability theories)
- `vonmisesvariate()` returns a random floating number between 0 and 1 based on the von Mises distribution (used in directional statistics)
- `paretovariate()` returns a random floating number between 0 and 1 based on the Pareto distribution (used in probability theories)
- `weibullvariate()` returns a random floating number between 0 and 1 based on the Weibull distribution (used in statistics)

random.choice()

Returns a random element of the given sequence

```
print(random.choice([1, 4, 13, 0, 11, 28, 9]))
```

```
print(random.choice([1, 4, 13, 0, 11, 28, 9]))
```

```
print(random.choice(['Pierre', 'Paul', 'Jacques', 'André', 'Jean']))
```

```
print(random.choice(['Pierre', 'Paul', 'Jacques', 'André', 'Jean']))
```

```
print(random.choice(['Pierre', 'Paul', 'Jacques', 'André', 'Jean']))
```

random.seed()

Initialises the random number generator

```
random.seed(100)
print(random.choice([1, 4, 28, 9]))
```

```
random.seed(100)
print(random.choice([1, 4, 28, 9]))
```

```
random.seed(100)
print(random.choice([1, 4, 28, 9]))
```

```
# without specifying the same random number "generator" number
# the resultt is different
print(random.choice([1, 4, 28, 9]))
```

```
random.seed(10)
print(random.choice(['Pierre', 'Paul', 'Jacques', 'André', 'Merveille']))
```

```
random.seed(10)
print(random.choice(['Pierre', 'Paul', 'Jacques', 'André', 'Merveille']))
```

```
random.seed(10)
print(random.choice(['Pierre', 'Paul', 'Jacques', 'André', 'Merveille']))
```

```
# sans préciser le même numéro de "générateur" de nombres aléatoires
# le résultat est différen
print(random.choice(['Pierre', 'Paul', 'Jacques', 'André', 'Merveille']))
```

random.random()

Returns a random floating-point number between 0 and 1

```
# to show the influence of the random generator
# let's now call the random() function of the random module many times
# the randomly generated values are different
for i in range(5):
    print(random.random())
```

random.uniform()

Returns a random floating-point number between the two given parameters

```
for i in range(5):
    print(random.uniform(2,100))
```

random.randrange():

Returns a random number between the given range

random.int():

Returns the same result

Note:

Both methods generate a random (integer) number within a given range

- `random.randrange()`
  - does not require to specify the start value (the search will then start from 0);
  - excludes the end of the range from the list of possible resulting values;
  - in addition, it allows you to specify the step for filtering possible values (for example, a choice between 2 and 10 with a step of 3 will be made among the values 2, 5, 8).
- `random.randint()`
  - requires to specify the start and end values of the range;
  - the end of the range is a possible value of the result.

```
print(random.randrange(0,5))
```

```
print(random.randint(0,5))
```

```
print(random.randrange(5))
```

```
# it is mandatory to specify the start of the range
print(random.randint(5))
```

```
print(random.randrange(2,5))
```

```
print(random.randint(2,5))
```

```
# the specified range is empty -> error
print(random.randrange(1,1))
```

```
# with this function(), the program can return a value
print(random.randint(1,1))
```

```
# since the ending value of the range is excluded
# the only value is '1'
for i in range(5):
    print(random.randrange(1,2))
```

```
# which is not the case here
for i in range(5):
    print(random.randint(1,2))
```

```
# specification of a step -> between start value and end value, every 'step'
# the program will choose among 2,5,8
print(random.randrange(2,10,3))
```

```
# the 'step' parameter is not defined for this function
print(random.randint(2,10,3))
```

random.shuffle()

Takes a sequence and returns the sequence in a random order

```
print('initial list', list_fib)

random.shuffle(list_fib)
print('shuffled list', list_fib)
```

random.sample()

Returns a given sample of a sequence

```
# starting population
population = [0,1,2,3,4,5,6,7,8,9,10]

# generation of 5 random samples in the population
print(random.sample(population, 5))
```

Congratulations!

In [1]:

```
from IPython.core.display import HTML
def css_styling():
    styles = open("../styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[1]: