[DS-02Py] Introduction to Python

Miguel-Angel Canela

Associate Professor, IESE Business School

General introduction

Python is a programming language, introduced in 1991. We find it everywhere, and it is actually ranked second in the list of the most used programming languages, afer JavaScript. It is said to be the preferred choice of the developers of malicious software (and those people are knowledgeable).

There are currently two versions of the Python language: Python 2 and Python 3. This course uses version 3. Beginners are actually adopting Python 3, but there is still a lot of Python code written in version 2, and many books base their explanations about how to do something in Python on that version. Although most of the Python 2 code runs in Python 3, one finds trouble from time to time. So, it is recommended to check Python's version before starting to read a book or before copypasting somebody else's code.

Python can be extended by more than 200 **packages**, although we will meet just a few ones in this course. We do not look at Python as a programming language, that is, for developing software applications, but from a Data Science perspective. About 2008, three packages were added to the Python portfolio: Pandas, for managing data sets, Matplotlib, for plotting, and scikit-learn, for machine learning. This trio, which we call **Python for Data Science**, put Python in the data analytics arena. Since then, Python's popularity has been growing steadily among data analysts and, nowadays, it is the preferred tool for developing **machine learning models**. Pandas is the cornerstone of Python for Data Science. It was developed by Wes McKinney as a replacement of R, the leading statistical language. So, many things in Pandas feel like R, and translation of R code to Python is easy.

There are many distributions of Python. In the Data Science world, **Anaconda** (`https://www.anaconda.com`) is the favorite one. Anaconda distribution comes with the three packages mentioned above. Downloading and installing Anaconda (choose Python 3 when the choice is presented to you) will leave you with the **Anaconda Navigator**, which opens in the browser and allows us to choose among different interfaces to Python. Once Anaconda is installed, you can bypass the navigator using a **command-line interface** (CLI), like Terminal in Mac computers or the Anaconda prompt in Windows. In Windows, you can also create a direct access link to the notebooks or the console.

Among the many choices offered by Anaconda, I use for these notes the **Jupyter QtConsole**, which is an input/output text interface. Jupyter is a new name for an older project called **IPython**, so you may find in many places reference to the "IPython console", which is the same as the Jupyter QtConsole.

An alternative approach is based on the **notebook** concept. In a notebook, you can combine input, output and ordinary text. In the notebook field, the **Jupyter Notebook** is the leading choice, followed by **Apache Zeppelin**. These two are multilingual, that is, they can be used with other languages, like R, besides Python. Jupyter has powerful supporters and very smart people in the development team, so we will probably see plenty of Jupyter notebooks in the immediate future.

In the console, you can type or paste your code. When you open it, you find an input prompt (such as `In[1]:`), where you can type a command and press `Return`. Then Python returns an output (such as `Out[1]:`), an error message or no answer at all. A supersimple example:

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

```
Out[1]: 4
```

So, if we input `2 + 2`, the output is the result of this calculation. But, when we want to store this result, we input it with a name, as follows.

```
In [2]: a = 2 + 2
```

Note that the value of `2 + 2` is not outputted now. If we want it to be outputted, we have to ask for that explicitly.

```
In [3]: a
Out[3]: 4
```

¶ In some programming environments, you should type `print(a)` or similar to print `a` on the screen.

If you copypaste code from a text editor (which is what you would do if you were working in the console, so that you could save your code), you can input several lines of code at once. In that case, you will get the output only for the last line. To get the output, you have to press now `Shift + Return`. A simple example:

```
In [4]: b = 2 * 3
   ...: b - 1
   ...: b**2
Out[4]: 36
```

¶ You would probably have written `b^2` for the square of 2, but the "hat" symbol does not work in Python.

As said above, Python is a programming language to which many additional resources have been added in the form of packages. The basic Python (without any package) is quite limited, so you need packages for almost everything. For instance, suppose that your math work goes beyond the above calculations, and you want to calculate the square root of 2. You will import first the package `math`, whose resources include the square root and many other mathematical functions, and then apply the function `math.sqrt`. This notation indicates that `sqrt` is a **method** of the object `math`. As you will discover very soon, objects in Python come with methods attached. In the console, the square root calculation shows up as:

```
In [5]: import math
   ...: math.sqrt(2)
Out[5]: 1.4142135623730951
```

Learning about Python

There are many books for learning about Python, but most of them would not be appropriate for learning Python for Data Science. It can even happen that you do not find anything about data in many of them. Mind that Python has so many applications that the intersection of the know-how of all Python users is very narrow. For an introduction to Python as a programming language, in a computer science context, I would recommend Zelle (2010). In the context of this course, McKinney (2017) and VanderPlas (2017) are both worth the price. To those who are not afraid of manuals, I would recommend the Pandas manual (which is free).

There is also plenty of learning materials in Internet, including MOOC's. For instance, **Coursera** has a pack of courses on Python (`https://www.coursera.org/courses?query=python`). But, probably, the most attractive marketplace for Data Science courses is **DataCamp**. They offer, under subscription or academic license, an impressive collection of courses, using either R or Python.

In addition to follow DataCamp courses, you can benefit from the **Datacamp Community Tutorials**, which are free and cover a wide range of topics.

Numbers and strings

As said in our first example, the equal sign (=) is used to assign a value to a variable. For the variable `a` defined in the first place:

```
In [6]: type(a)
Out[6]: int
```

So, `a` has **integer type**. Another numeric type is **float**:

```
In [7]: b = math.sqrt(2)
   ...: type(b)
Out[7]: float
```

There are subdivisions of integers and floats, but I skip them in this brief introduction. Note that, in Python, integers are not just numbers with zero after the comma, but a different type:

```
In [8]: type(2)
Out[8]: int
In [9]: type(2.0)
Out[9]: float
```

Besides numbers, we can also manage **strings**:

```
In [10]: c = 'Messi'
   ...: type(c)
Out[10]: str
```

The quote marks indicate string type. You can use single or double quotes, but take care of using the same on both sides of the string. Strings come in Python with many methods attached, but I postpone the discussion, so string methods will be discussed in the context of `pandas` data frame methods.

Finally, we have **Boolean** variables, which are either `True` or `False`:

```
In [11]: d = 5 < a
   ...: d
Out[11]: False
In [12]: type(d)
Out[12]: bool
```

Note that, if we define a variable with an expression like the above one, it has Boolean type. Also, note that to put equality in the expression, we need two equal signs (this may surprise you):

```
In [13]: a == 4
Out[13]: True
```

Lists

Python has several **compound data types**, which are used to group together other values. The most versatile is the **list**, which can be written as a sequence of comma-separated values between square brackets. Lists can contain items of different type, although this not usual.

A simple example of a list is:

```
In [14]: x = ['Messi', 'Cristiano', 'Neymar', 'Coutinho']
```

Lists can be joined in a very simple way in Python:

```
In [15]: y = x + [2, 3]
   ...: y
Out[15]: ['Messi', 'Cristiano', 'Neymar', 'Coutinho', 2, 3]
```

Now, the length of the list `y` is 6:

```
In [16]: len(y)
Out[16]: 6
```

The first item of `y` can be extracted as `y[0]`, the second item as `y[1]`, etc. The last item can be extracted as `y[5]` or as `y[-1]`. Sublists can be extracted putting a colon within the brackets, as in:

```
In [17]: y[0:2]
Out[17]: ['Messi', 'Cristiano']
```

Note that `0:2` includes 0 but not 2. This is a general rule for indexing in Python. Other examples:

```
In [18]: y[3:]
Out[18]: ['Coutinho', 2, 3]

In [19]: y[:23]
Out[19]: ['Messi', 'Cristiano']
```

In Pandas data frames, there are other ways of extracting parts of the data, based on expressions such as `y > 0`, as we will see later in this course. The items of a list are ordered, and can be repeated. This is not so in other compound data types, like **sets**:

```
In [20]: set(y)
Out[20]: {2, 3, 'Coutinho', 'Cristiano', 'Messi', 'Neymar'}
```

Note that the items in the set are printed in alphabetic order, which means that there is no order. Also, repeated items are dropped, which some coders use to extract a list of unique values of a list with repeated items:

```
In [21]: list(set([1, 0, 1, 0, 7]))
Out[21]: [0, 1, 7]
```

For loops

The **for loop** exists in practically all current programming languages, and all coders identify it as a way to avoid repetition. I give first a supersimple example:

```
In [22]: squares = [0]
    ...: for i in range(1, 4):
    ...:       squares = squares + [i**2]

In [23]: squares
Out[23]: [0, 1, 4, 9]
```

When typing this in the console, you may have noted that, in the definition of `squares`, the third line comes indented. This is triggered by the colon. Also, note that `range(1, 4)` contains `1` but not `4`, as it is the rule in Python.

The loop for generating `squares` has been presented in a standard form. But in Python, you can do better since it is possible to transform a list into another list with a loop, in one line of code:

```
In [24]: squares = [i**2 for i in range(0,4)]
    ...: squares
Out[24]: [0, 1, 4, 9]
```

Ranges generated by the function `range` are not the same as lists, but similar. I skip the details. Instead of a range, we can also use a list, as in the following example:

```
In [25]: [len(name) for name in x]
Out[25]: [5, 9, 6, 8]
```

Now, a bit more difficult. The following loop generates a sequence of Fibonacci numbers (quite popular since they appeared in the *The Da Vinci Code*, where they are used to unlock a safe).

```
In [26]: fib = [1, 1]
    ...: for i in range(2, 10):
    ...:       fib = fib + [fib[i-1] + fib[i-2]]
In [27]: fib
Out[27]: [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Functions

Python is a fully functional language. Part of its real power comes from defining the operations that we wish to perform as **functions**, so they can be applied many times. A simple example of the definition of a function follows. Again, note the indent after the colon.

```
In [28]: def f(x):
    ...:       y = 1/(1 - x**2)
    ...:       return(y)
```

When we define a function, Python just takes note of the definition, accepting it when it is syntactically correct (parentheses, commas, etc). The function can be applied later to different arguments.

```
In [29]: f(2)
Out[29]: -0.3333333333333333
```

If we apply the function to an argument for which it does not make sense, Python will return an error message which depends on the values supplied for the argument.

Functions can have more than one argument, as in:

```
In [30]: def g(x, y): return x*y/(x**2 + y**2)
In [31]: g(1, 1)
Out[31]: 0.5
```

Note that, in the definition of `g`, I have used a shorter form. Most programmers would make it longer, as I did previously for `f`.

References

1. W McKinney (2017), *Python for Data Analysis — Data Wrangling with Pandas, NumPy, and IPython*, O'Reilly.
2. W McKinney & PyData Development Team (2018), *pandas — powerful data analysis toolkit*.
3. J VanderPlas (2017), *Python Data Science Handbook*, O'Reilly.

4. J Zelle (2010), *Python Programming — An Introduction to Computer Science*, Franklin, Beedle & Associates.