



ÉCOLE  
**D'INGÉNIEURS**  
PARIS-LA DÉFENSE

# Python For Data Analysis

**Imen Ouled Dlala**

imen.ouled\_dlala@devinci.fr  
**Bureau: L405**





## General Plan

- Basics of Python for Data science
- Data Science Modules
  - Numpy, Pandas, Scipy
- Data Analysis and Visualization
  - Seaborn, Matplotlib, Bokeh
- Webscrapping
- Machine Learning Libraries
  - Sklearn, tensorflow
- API Django / Flask



## Basics of Python for Data Science : Plan

1. Development Environment
2. Introduction to Python

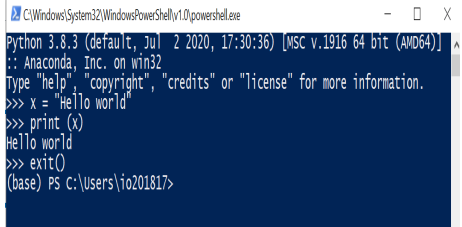
# Development Environment

## Development Environment : Console

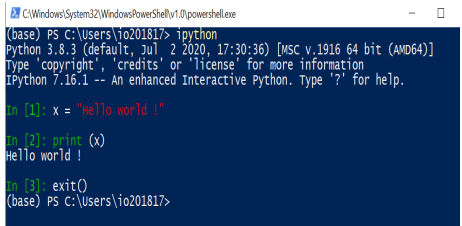
**Python** offers an interactive console (also called the Python interpreter or Python shell).

It provides programmers with a quick way to execute commands and try out or test code without creating a file.

**ipython** allows to get the numbering of the input/output lines, the color and the auto-compilation (via the tab key).

A screenshot of a Windows PowerShell window titled 'C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe'. The prompt is '(base) PS C:\Users\io201817>'. The user has entered 'python' and the output shows 'Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32'. The user then enters 'Type "help", "copyright", "credits" or "license" for more information.', 'x = "Hello world"', 'print(x)', and 'exit()'. The output for 'print(x)' is 'Hello world'.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
:: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = "Hello world"
>>> print(x)
Hello world
>>> exit()
(base) PS C:\Users\io201817>
```

A screenshot of a Windows PowerShell window titled 'C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe'. The prompt is '(base) PS C:\Users\io201817>'. The user has entered 'ipython' and the output shows 'Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] Type "copyright", "credits" or "license" for more information IPython 7.16.1 -- An enhanced Interactive Python. Type "?" for help.'. The user then enters 'In [1]: x = "Hello world !"', 'In [2]: print(x)', and 'In [3]: exit()'. The output for 'print(x)' is 'Hello world !'.

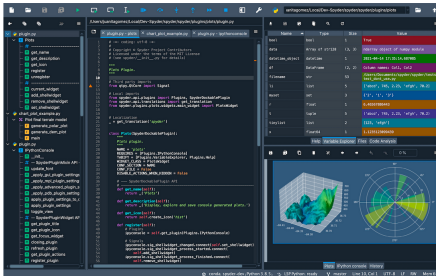
```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
(base) PS C:\Users\io201817> ipython
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information
IPython 7.16.1 -- An enhanced Interactive Python. Type "?" for help.

In [1]: x = "Hello world !"
In [2]: print(x)
Hello world !
In [3]: exit()
(base) PS C:\Users\io201817>
```



## Development Environment : Spyder

- Type : IDE.
- Price : Open Source
- Platform Support : QT, WINDOWS, LINUX, MAC OS etc.
- Editor, Interactive console, Documentation viewer, Variable explorer ,Find in files, File explorer, History log...



## Development Environment :Jupyter



### Language of choice

The Notebook has support for over 40 programming languages, including those popular in Data Science such as Python, R, Julia and Scala.



### Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the [Jupyter Notebook Viewer](#).



### Interactive widgets

Code can produce rich output such as images, videos, LaTeX, and JavaScript. Interactive widgets can be used to manipulate and visualize data in realtime.



### Big data integration

Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, dplyr, etc.



## Jupyter Architecture

The Jupyter Notebook is based on a set of open standards for interactive computing. Think HTML and CSS for interactive computing on the web. These open standards can be leveraged by third party developers to build customized applications with embedded interactive computing.

## Development Environment :Jupyter

<http://daringfireball.net/projects/markdown/>

Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).

Thus, "Markdown" is two things:

- (1) a plain text formatting syntax; and
- (2) a software tool, written in Perl, that converts the plain text formatting to HTML.

```
# This is an H1
## This is an H2
##### This is an H6
```

```
> This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet,
> consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.
> Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
>
> Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse
> id sem consectetur libero luctus adipiscing.
```

You can produce a horizontal rule tag (`<hr />`) by placing three or more hyphens, asterisks, or underscores on a line by themselves.

```
> ## This is a header.
>
> 1. This is the first list item.
> 2. This is the second list item.
```

LISTS :

- \* Red
- \* Green
- \* Blue

1. Bird
2. McHale
3. Parish



## Development Environment :Jupyter

### Jupyter hub



### Jupyter for Organizations

JupyterHub is a multiuser version of the notebook designed for centralized deployments in companies, university classrooms and research labs.



#### Pluggable authentication

Manage users and authentication with PAM, OAuth or integrate with your own directory service system. Collaborate with others through the Linux permission model.



#### Centralized deployment

Deploy the Jupyter Notebook to all users in your organization on centralized servers on- or off-site.



#### Container friendly

Use Docker containers to scale your deployment and isolate user processes using a growing ecosystem of prebuilt Docker containers.



#### Code meets data

Deploy the Notebook next to your data to provide unified software management and data access within your organization.

# Introduction to Python

# Introduction to Python

- Python is both an interpreted and a compiled language.
- Python variables are references to objects.
- Instructions are executed one after the other.

## Indenting Code

- Code blocks are defined by their indentation (this allows to define the loops and functions) .
- No semicolons and braces (but we can put several expressions in a row separated by ";" ) .
- Loops and Conditional statements end with a colon ":" .

```
for a in range(1,n) :  
    for b in range(a,n) :  
        c_square = a**2 + b**2  
        c = int(sqrt(c_square))  
        if ((c_square - c**2) == 0) :  
            print(a, b, c)
```



# Introduction to Python

The **scope** of a variable in python is that part of the code where it is visible.

## ■ Local scope

- A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.
- The local variable can be accessed from a function within the function.

## ■ Global scope

- A variable declared outside of the function or in global scope.
- We can read it anywhere in the program.
- Global variables are available from within any scope, global and local.

## ■ Built-in

- The built-in scope has all the names that are loaded into python variable scope when we start the interpreter.
- We never need to import any module to access.

# Introduction to Python

## Semantics

### Everything is an object

The methods are called via dots [.]

The type of the object is stored in the object itself.

```
x = "hello world !"  
x.capitalize()
```

Hello world !

### Variables

They are pointers to objects.

They can refer to different objects from one moment to another.

Objects are typed, but not the variables !

# Introduction to Python : Data Type

## Scalars

None

bool True/False

int

float

long

## Objects

Declaration **class**

Self-reference **self**

Attributes variables

access by a `[.]`

Methods functions(**self**)

access by a `[.]`

default attributes **`__dict__`**

**`x.__dict__`**

```
{'title' : 'Guerre et paix', 'author' :  
'Tolstoï'}
```

```
class book :  
    title = ""  
    author = ""  
    def display(self) :  
        print (self.author + " :" + self.title)  
x = book()  
x.title = "Guerre et paix"  
x.author = "Tolstoï"  
x.display()
```

Tolstoï :Guerre et paix

## Introduction to Python : Conditional statements and loops

```
if age > 18 :  
    adult =True  
else :  
    adult =False
```

```
if age < 10 :  
    print("child")  
elif age < 18  
    print("teenager")  
else :  
    print("adult")
```

```
for age in range(9,19) :  
    if age < 10 :  
        print("child")  
    elif age < :18  
        print("teenager")  
    else :  
        print("adult")
```

```
iteration = 0  
while iteration < 10 :  
    iteration+= 1  
    print (iteration)  
else :  
    print ("fin")
```

```
import random  
max = 100  
value = random.randint(0, max)  
iteration= 0  
while iteration < max :  
    if iteration == value :  
        break  
    iteration= iteration+ 1  
print u"the value of random was :", iteration
```

## Introduction to Python : String

```
name = "Antoine"  
message = "Hello {name}!"  
message.format(**locals())  
'Hello Antoine!'
```

```
for name in ["a", "b"] :  
    print (message.format(**locals()))  
Hello a!  
Hello b!
```

```
"Hello %s!"%(name)
```

"The value is %.2f!" %(value\_float)  $\Rightarrow$  2 digits after the comma

"The value is %.10.2f!" %(value\_float)  $\Rightarrow$  10 (blank) columns

"The value is %010.2f!" %(value\_float)  $\Rightarrow$  10 columns filled with (zeros)



## Introduction to Python : List (1/2)

```
lst_1 = [1, 4, 77, 43, 100]
lst_2 = ['Paris', 'Berlin', 'Amsterdam']
lst_3 = [lst_1, lst_2]
lst_3 = [ ]
```

1	4	77	43	100
---	---	----	----	-----

#index #0 #1 #2 #3 #4

position = 0  
step = 1

### #Indexing

- `lst_1 [#index]` ← element
- `lst_1 [0]` ← first element
- `lst_1 [-1]` ← last element

### #Slicing

- `lst_1 [position : position+1 : step]` [1]
- `lst_1 [position : position+2 : step]` [1, 4]
- `lst_1 [position : position+4 : step+1]` [1, 77]
- `lst_1 [-1 : ]` [100]
- `lst_1 [-2 : ]` [43, 100]
- `lst_1 [1 :3]` [4, 77]

## Introduction to Python : List (2/3)

Output :

```
if 'Barcelona' in lst_2 :  
    print("yes")  
else :  
    print("no")
```

no

```
for i in lst_2 :  
    print(i)
```

Paris  
Berlin  
Amsterdam

```
for index, valeur in enumerate(lst_2) :  
    print(index,valeur)
```

0 Paris  
1 Berlin  
2 Amsterdam

```
for x,y in zip(lst_2,lst_1) :  
    print(x,y)  
# enumeration stops when the shortest list  
ends
```

Paris 1  
Berlin 4  
Amsterdam 77

## Introduction to Python : List (3/3)

### Changing lists

append	lst_1.append(8)	[ 1, 4, 77, 43, 100, 8]
insert	position= 2 value= 5 lst_1.insert(2, 5)	[ 1, 4, 5, 77, 43, 100]
+	lst_1+lst_1	[ 1, 4, 77, 43, 100, 1, 4, 77, 43, 100]
extend	lst_1.extend(lst_1)	[ 1, 4, 77, 43, 100, 1, 4, 77, 43, 100]
pop	position = 4 list_1.pop(position)	[1, 4, 77, 43]
remove	value = 77 lst_1.remove(value)	[1, 4, 43, 100]
sort	lst_1.sort()	[ 1, 4, 43, 77, 100]
bisect	import bisect bisect.bisect(lst_1, 5)	2 ( index of the first element > 5 )

## Introduction to Python : List Comprehensions

```
list=[ ]  
for i in range (10) :  
    list.append(i**2)  
print(list)
```

```
import time  
start=time.time()  
list=[ ]  
for i in range(10) :  
    list.append(i**2)  
end=time.time()  
print(end-start)
```

```
list_comp=[i**2 for i in range (10)]  
print(list_comp)
```

```
import time  
start=time.time()  
list=[i**2 for i in range(10)]  
end=time.time()  
print(end-start)
```

## Introduction to Python : Dictionaries (1/3)

```
dictionary_1 = {  
    "un" : "one",  
    "deux" : "two",  
}
```

```
dictionary_2={  
    "girls" :40,  
    "boys" :30  
}
```

### ■ key, value

- dictionary\_1.values()
- dictionary\_1.keys()

### ■ Retrieve a value

- dictionary\_1.get("cinq", "dont exist")
- dictionary\_1["deux"]

### ■ Test the existence of a key

- "un" in dictionary\_1
- "two" in dictionary\_1

## Introduction to Python : Dictionaries (2/3)

Display dictionary keys

```
for i in dictionary_1 :  
    print(i)
```

Display dictionary values

```
for i in dictionary_1.values() :  
    print(i)
```

Display dictionary keys and values

```
for k,v in dictionary_1.items() :  
    print(k,v)
```

## Introduction to Python : Dictionaries (3/3)

### Changing dictionaries

#### Insertion

```
key= "trois"  
value= "three"  
dictionary_1[key] = value  
{"un" : "one", "deux" : "two", "trois" : "three" }
```

#### Concatenation

```
dictionary_3={ "quatre" : "four" }  
dictionary_1.update(dictionary_3)  
{"un" : "one", "deux" : "two", "trois" : "three", "quatre" : "four" }
```

#### Deletion

```
del ==> del dictionary_1["quatre"] {"un" : "one", "deux" : "two", "trois" : "three" }  
pop ==> dictionary_1.pop("trois") {"un" : "one", "deux" : "two" }
```

#### Sort

```
soted ==> sorted(dictionary_1) returns a sorted list of dictionary keys
```

## Introduction to Python : Dictionary Comprehension

```
age = [22, 1, 40, 33, 55, 10]
```

```
first_name = ["Antoine", "Sarrah", "Luc", "Julien"]
```

```
dict_comp_1 = { k :v for k,v in enumerate (first_name)}
```

```
dict_comp_1 = { 0 : "Antoine" , 1 : "Sarrah", 2 : "Luc", 3 : "Julien" }
```

```
dict_comp_2 = { name :age for name,age in zip ( first_name,age)}
```

```
dict_comp_2 = { "Antoine" : 22, "Sarrah" : 1, "Luc" : 40, "Julien" : 33}
```

```
dict_comp_3 = { name :age for name,age in zip (first_name,age) if age>22 }
```

```
dict_comp_3 = { "Luc" : 40, "Julien" : 33 }
```



## Introduction to Python : Tuples and Sets

### Tuples

```
tuple_int = (1, 2, 3, 4 )  
lst = list(tuple_int)  
tuple_int = tuple(lst)
```

### Sets

```
set_int = {1, 2, 3, 4}  
  
set_int.add(value)  
set_int.remove(value)  
set_int.discard(value)  
set_int.clear()  
  
set_int.issubset(another_set)  
set_int.issuperset(another_set)  
  
set_int | another_set union  
set_int & another_set intersection  
set_int - another_set difference(elements in the first set, not in the second)  
set_int ^ another_set symmetrical difference
```

# Introduction to Python : Functions

## Definition

```
def noun ( param1, parm2)
```

```
def square (x) :  
    return x*x
```

```
square(3)
```

Output : 9

```
path_to_square = square
```

```
def apply (function, value) :  
    return function(value)
```

```
apply(path_to_square, 3)
```

Output : 9

## Introduction to Python : Functions

### map/filter

**map**(function\_to\_apply, list\_of\_inputs) : applies a function to all the items in an input\_list.

**filter**(function\_to\_apply, list\_of\_inputs) : creates a list of elements for which a function returns true.

```
def square (x) :  
    return x*x
```

```
def is_positive (x) :  
    return x > 0
```

```
list_int = [0, 4, -5, 7, -100]  
list_square = map(square, list_int)  
list_square
```

Output : [0, 16, 25, 49, 1000]

```
list_positive = filter(is_positive, list_int)  
list_positive
```

Output : [4, 7]

## Introduction to Python : Functions

```
square = lambda x : x**2  
print(square(3))
```

```
function = lambda x,y : x**2+y  
print(function(3,2))
```

```
list_square = map(lambda x : x**2,list_int)  
list_square
```

Output : [0, 16, 25, 49, 1000]

```
list(map(lambda x : x > 0,list_int))
```

Output : [False, True, False, True, False]

```
list(filter(lambda x : x > 0,list_int))
```

Output : [4, 7]

## Introduction to Python : Functions

`*args` and `**kwargs`

cf : [http://deussyss.developpez.com/tutoriels/Python/args\\_kwargs/](http://deussyss.developpez.com/tutoriels/Python/args_kwargs/)

### Definition

`def` noun ( `*args` ) List of parameters (variable length)

`def` noun ( `**kwargs` ) A dictionary as a parameter

### A function as a parameter

`def` apply( function, liste) :

`for` index, item `in` enumerate(liste) : `#enumerate` returns the index and the value  
        liste[index] = function(item) `#evaluation of the function.`

`def` apply( function, liste) :

`for` index, item `in` enumerate(liste) : `#enumerate` returns the index and the value  
        liste[index] = function(item) `#evaluation of the function`

list\_int =[2,3]

apply(path\_to\_square, liste\_int)

## Introduction to Python : Dates

The `datetime` module supplies classes for manipulating dates and times. It offers three types of objects :

1. `date`
2. `time`
3. `datetime` (contains a date and a time)

Import	<code>from datetime import datetime, date, time</code>
Utilization	<code>dt = datetime(2011,10,29,20,30,21)</code>
	<code>dt.day</code>
	<code>29</code>
	<code>dt.minute</code>
	<code>30</code>
	<code>dt.date()</code>
	<code>datetime.date(2011,10,29)</code>
	<code>dt.time()</code>
	<code>datetime.time(20,30,21)</code>

## Introduction to Python : Dates

Conversion

```
dt.strptime('%m/%d/%Y %H :%m/%d')  
'10/29/2011 20 :10/29'  
datetime.strptime('20091031', '%Y/%m/%d')  
datetime.datetime(2009,10,31,0,0)
```

Replace

```
dt.replace(minute=0, second=0)
```

Timedelta

```
datetime.datetime(2011,10,29,20,0)  
dt2 = datetime.datetime(2011,10,29,20,30,21)  
delta = dt2 - dt  
dt + delta
```

# Introduction to Python : Dates

## Dates – format iso

source : <https://docs.python.org/2/library/datetime.html>

Directive	Meaning	Example
%a	Weekday as locale's abbreviated name.	Sun, Mon, ..., Sat (en_US); So, Mo, ..., Sa (de_DE)
%A	Weekday as locale's full name.	Sunday, Monday, ..., Saturday (en_US); Sonntag, Montag, ..., Samstag (de_DE)
%w	Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.	0, 1, ..., 6
%d	Day of the month as a zero-padded decimal number.	01, 02, ..., 31
%b	Month as locale's abbreviated name.	Jan, Feb, ..., Dec (en_US); Jan, Feb, ..., Dez (de_DE)
%B	Month as locale's full name.	January, February, ..., December (en_US); Januar, Februar, ..., Dezember (de_DE)
%m	Month as a zero-padded decimal number.	01, 02, ..., 12
%y	Year without century as a zero-padded decimal number.	00, 01, ..., 99
%Y	Year with century as a decimal number.	1970, 1988, 2001, 2013
%H	Hour (24-hour clock) as a zero-padded decimal number.	00, 01, ..., 23
%I	Hour (12-hour clock) as a zero-padded decimal number.	01, 02, ..., 12
%p	Locale's equivalent of either AM or PM.	AM, PM (en_US); am, pm (de_DE)
%M	Minute as a zero-padded decimal number.	00, 01, ..., 59
%S	Second as a zero-padded decimal number.	00, 01, ..., 59
%f	Microsecond as a decimal number, zero-padded on the left.	000000, 000001, ..., 999999
%z	UTC offset in the form +HHMM or -HHMM (empty string if the object is naive).	(empty), +0000, -0400, +1030
%Z	Time zone name (empty string if the object is naive).	(empty), UTC, EST, CST
%j	Day of the year as a zero-padded decimal number.	001, 002, ..., 366
%U	Week number of the year (Sunday as the first day of the week) as a zero padded decimal number. All days in a new year preceding the first Sunday are considered to be in week 0.	00, 01, ..., 53
%W	Week number of the year (Monday as the first day of the week) as a decimal number. All days in a new year preceding the first Monday are considered to be in week 0.	00, 01, ..., 53
%c	Locale's appropriate date and time representation.	Tue Aug 16 21:30:00 1988 (en_US); Di 16 Aug 21:30:00 1988 (de_DE)
%x	Locale's appropriate date representation.	08/16/88 (None); 08/16/1988 (en_US); 16.08.1988 (de_DE)
%X	Locale's appropriate time representation.	21:30:00 (en_US); 21:30:00 (de_DE)
%%	A literal '%' character.	%



# Introduction to Python : Dates

```
def find_type(string):
    """
        Test the data type contained inside a string (int, float, chararray).
        Returns (str): string's type
    """
    from datetime import datetime
    tests = [
        ("int", int),
        ("float", float),
        ("date-dmy", lambda value: datetime.strptime(value, "%d/%m/%y")),
        ("date-Ymd", lambda value: datetime.strptime(value, "%Y/%m/%d")),
        ("date-dmY", lambda value: datetime.strptime(value, "%d/%m/%Y")),
        ("date-Y-m-d", lambda value: datetime.strptime(value, "%Y-%m-%d")),
        ("date-d-m-d", lambda value: datetime.strptime(value, "%d-%m-%d")),
        ("date-ymd", lambda value: datetime.strptime(value, "%y/%m/%d")),]

    for type_de_string, function_de_test in tests:
        try:
            function_de_test(string)
            return type_de_string
        except:
            pass

    try:
        s = string.replace(",", ".")
        float(s)
        return "float"
    except:
        pass

    return "chararray"
```

```
find_type("18/10/21")
```

```
'date-dmy'
```

## Introduction to Python : Files and Folders

### File

```
path_file = "C :/.../file.csv"
```

#### Read

```
with open(path_file, 'r') as file :  
    for line in file :  
        print line
```

#### Add

```
with open(path_file, 'a') as file :  
    file.write("hello world!")
```

#### Read

```
content =  
open(path_file).readlines()
```

#### Write many lines

```
open(path_file).writelines(content)
```

### Folder

```
path_dir = "C :/folder/"
```

```
Import os
```

#### Verify the existence

```
os.path.exists(path_dir)
```

#### Create a folder

```
os.makedirs(path_dir)
```

#### Useful function

```
import os
```

```
if not os.path.exists(path_dir) :
```

```
    os.makedirs(path_dir)
```

```
    print " {0} created".format(path_dir)
```

#### Join a file path and a folder path

```
os.join(path_dir, path_file)
```



End

Good Lecture !

This course is inspired from Romain Jouin that I thank for sharing