Python
Introduction

Romain Jouin
Romain.jouin@memorandum.pro **06 52 86 87 30**

Présentation des intervenants

Romain Jouin

06 52 86 87 30 – romain.jouin@memorandum.pro

- **Romain a une double culture :**
  - Informatique : **développement logiciels**
  - Commerciale : **chasse de clients et développement de portefeuille**

Compétences principales

- Développement informatique : web, infrastructures, Machine Learning (Python)
- Processus de vente : démarches commerciales, processus d'élevage / chasse

Compétences sectorielles

- Télécommunications / Cloud
- Web

Parcours

- 2 ans en tant que Freelance en informatique (sites web, macros…)
- 3 ans en tant que Business Developer de la branche Network Integration Services chez Alactel-Lucent Russie
- 2 ans de Chargé d'affaires chez Toshiba Cloud Services France

Formation

Mastère Big Data Telecom Paristech 2014

INT école de Management 2006

Quelques références

Valorisation des données

- Acteur de l'assurance: évaluation du risque de churn – 200 millions de lignes à analyser
- Acteurs de la santé : Prédiction de concentrations moléculaires
- Evolution de la satisfaction client : déceler les différences de tendances dans un réseau de distribution
- Acteur de l'énergie : scrapping web et analyse de l'e-réputation sur les forums Français
- Cabinet de conseil : Formation aux Big Data
- École d'ingénieur : Initiation aux technologies du Big Data et à la Datavisualisation

Participation à l'écosystème Data sur Paris

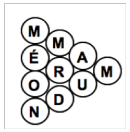
- Animateur de groupes d'intérêts autour de la Data (plus de 2000 personnes à Paris). 15 événements organisés depuis Octobre 2014.

Expertise métier

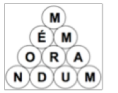
Informatique
- Développement web full-stack Python & Jquery
- Gestion d'infrastructure Big Data : Hadoop / Spark / Mesos / Docker

Commerce
- Choix des marchés stratégiques - Plan d'attaques du marché
- Reporting (>15 Meur) - Prévision annuel du CA
- Chasse - Présentation clients - Ajustement des prix
- Gestion d'équipes de consultants

Python - Introduction



1. Historique et principes
2. Environnement de développement
3. Introduction à Python : syntaxe
    - Python avancé
4. Introduction à Scipy (Pandas)
5. Introduction à Matplotlib
    - Autres librairies graphiques
6. Introduction à Scikit-Learn
7. Préparation de données
8. DASK
9. Création d'une API Rest Django
10. Test Driven Development en Django
11. Gestion des environnements
12. Annexes



Historique

1990	Création	Guido Van Rossum Langage orienté scripting / testing
1991	Rendu public	
1994	Groupe Usenet	comp.lang.python
1996	Livre	Programming Python (O'reilly)
2008	Envol	Reconnaissance par la communauté internationale Notamment les data-scientists Outils de calculs qui commencent à être reconnus (scipy)
	Python 3	Incompatible avec Python 2 fonction print gestion des chaines de caractères Itérateurs Vs lists

Principes

$ python -c "import this"
The Zen of Python, by Tim Peters

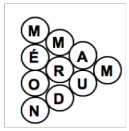
Beautiful **is better than ugly**.
Explicit **is better than implicit**.
Simple **is better than complex**.
Complex **is better than complicated**.
Flat **is better than nested**.
Sparse **is better than dense**.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never **pass silently**.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- **and preferably only one --obvious way to do it.**
Although that way may **not be obvious at first unless you're Dutch.**
Now **is better than never**.
Although never **is often better than *right* now.**
If the implementation **is hard to explain, it's a bad idea.**
If the implementation **is easy to explain, it may be a good idea**.
Namespaces are one honking great idea -- let's do more of those!

Plateformes supportées

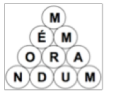
OS

- Windows
- Mac
- Linux
- OS ANSII Compliant – avec compilateur C (Open Source)
- JVM
- .NET VM

Python - Introduction



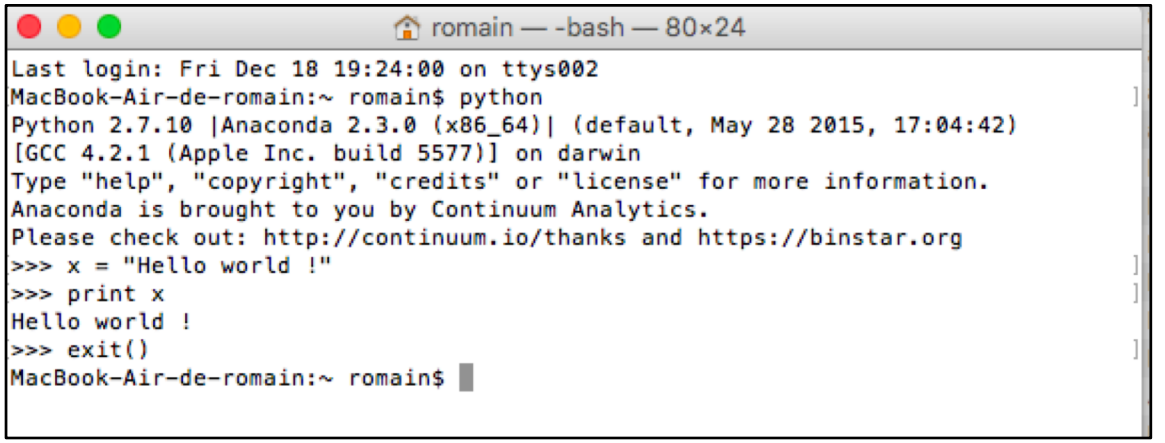
1. Historique et principes
2. Environnement de développement
3. Introduction à Python : syntaxe
    - Python avancé
4. Introduction à Scipy (Pandas)
5. Introduction à Matplotlib
    - Autres librairies graphiques
6. Introduction à Scikit-Learn
7. Préparation de données
8. DASK
9. Création d’une API Rest Django
10. Test Driven Development en Django
11. Gestion des environnements
12. Annexes



Console

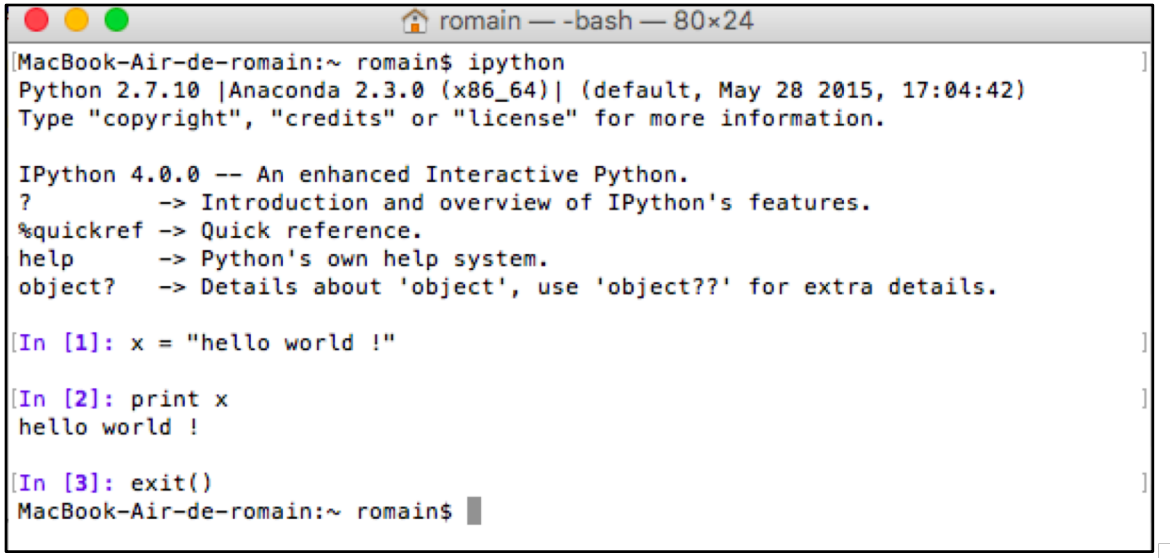
Python propose nativement une console d'interprétation de commandes.

L'interface est minimaliste.



L'outil **iPython** permet d'obtenir la numérotation des lignes d'entrées / sorties, de la couleur, et de l'autocomplétion (via la touche tab).

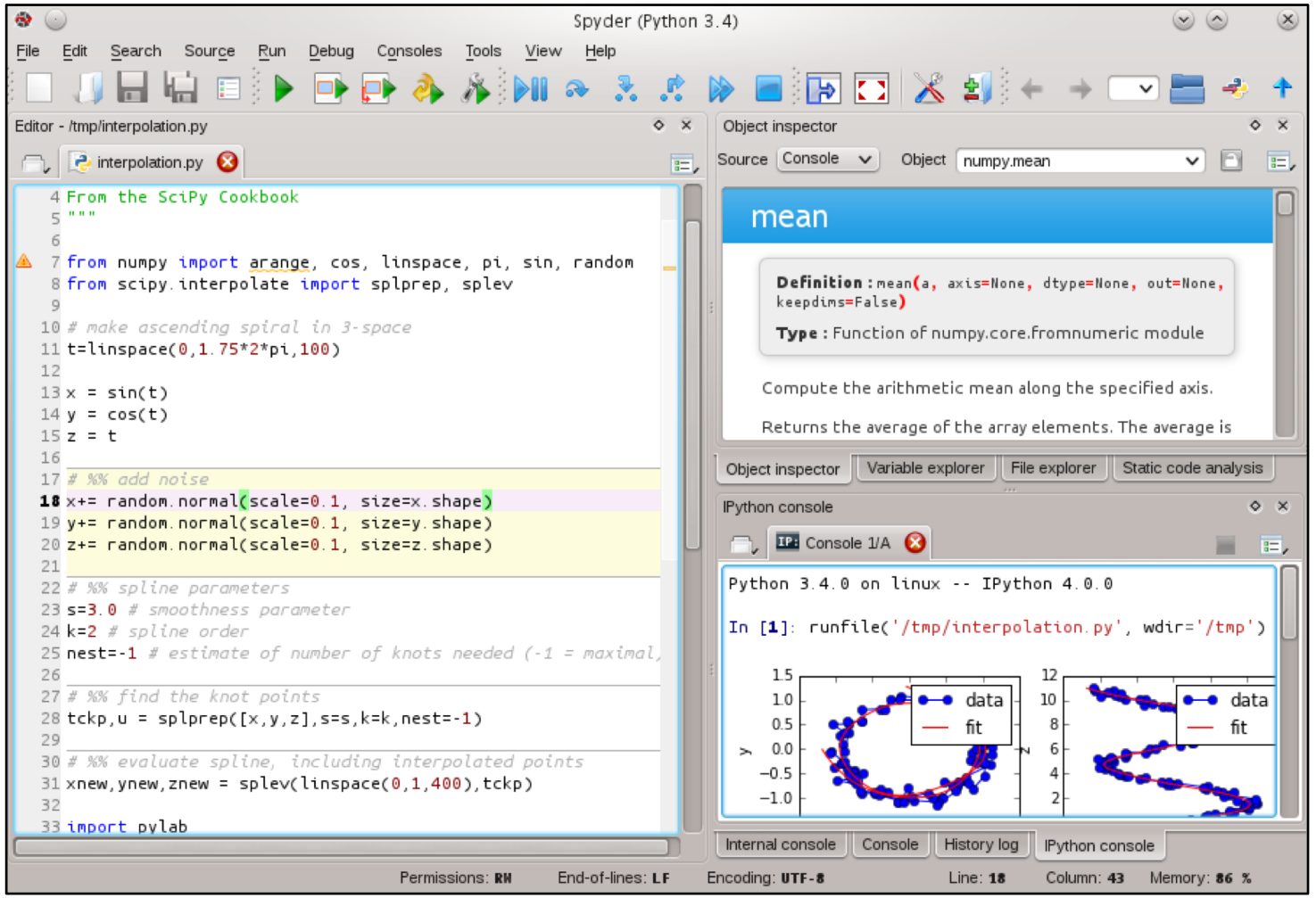
sudo apt-get install python-pip



Spider

Spider est une interface qui ressemble à R-studio pour Python:

- **Editor**
- **Interactive console**
- **Documentation viewer**
- **Variable explorer**
- **Find in files**
- **File explorer**
- **History log**

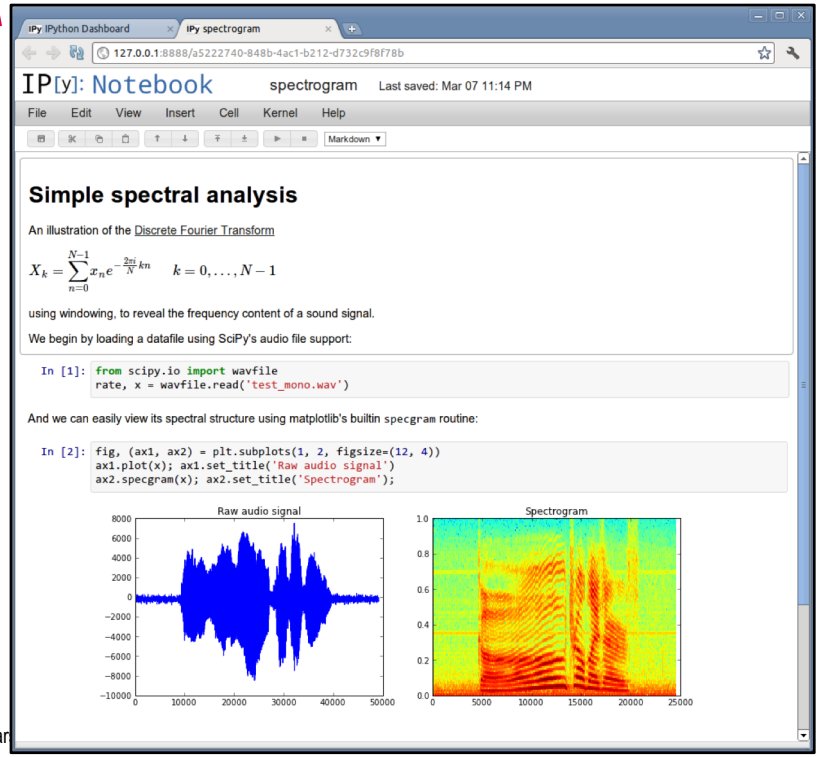


Ipython notebook

Ipython propose un mode “notebook”, qui déclenche un serveur local qui génére des pages html pour exécuter du python.
Cela permet de profiter de la souplesse de HTML5 et CSS pour une programmation plus intéractive.

Ce projet a évolé en 2015 vers “**Jupyter**”.

```
romain — python ◂ python.app ~/anaconda/bin/ipython notebook — 80×24
MacBook-Air-de-romain:~ romain$ ipython notebook
[I 12:27:18.800 NotebookApp] The port 8888 is already in use, trying another ran
dom port.
[I 12:27:18.801 NotebookApp] The port 8889 is already in use, trying another ran
dom port.
[I 12:27:18.806 NotebookApp] Serving notebooks from local directory: /Users/roma
in
[I 12:27:18.806 NotebookApp] 0 active kernels
[I 12:27:18.806 NotebookApp] The IPython Notebook is running at: http://localhos
t:8890/
[I 12:27:18.806 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
```



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.

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,
> consectetuer 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 consectetuer 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 [an example](http://example.com/ "Title") inline link.

> ## 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

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.

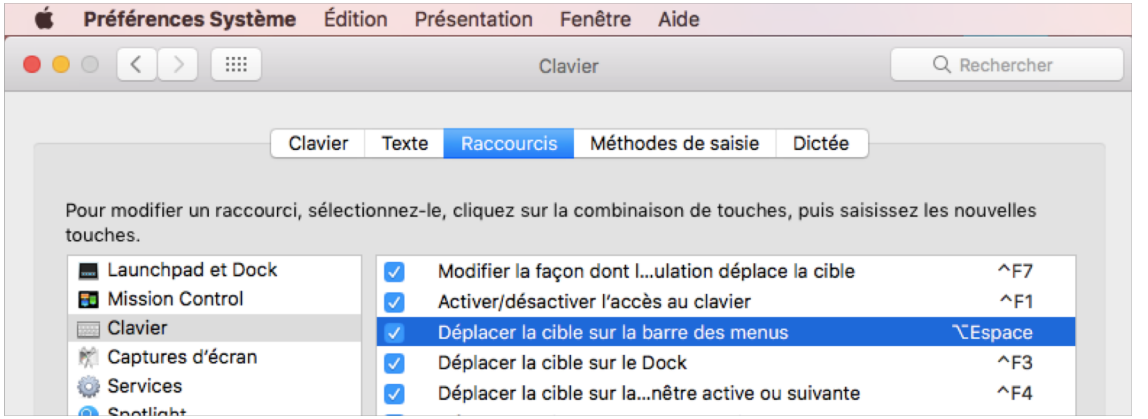
Code meets data

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

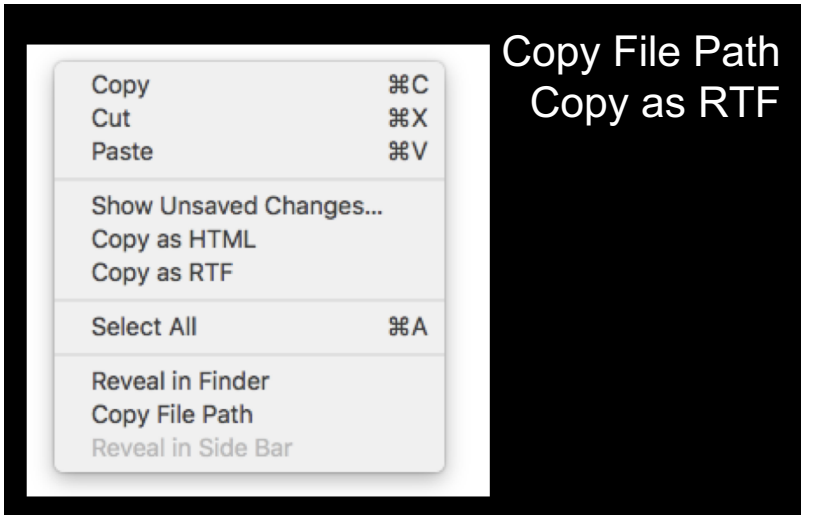
Sublime Text 1

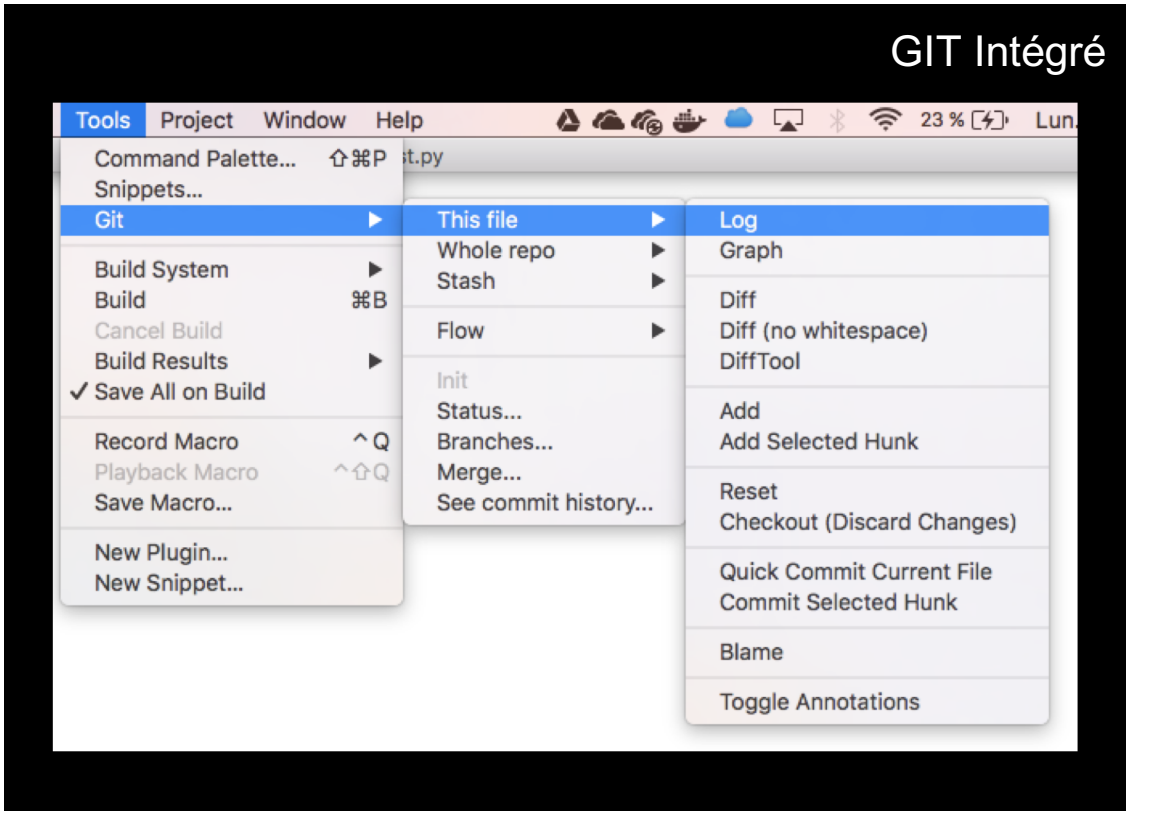
⌘ + p + :x	go-to ligne x
⌘ + ⇧ + P	afficher une aide
⌘ + ⇧ + L	sélection multi ligne
⌘ + d	sélectionne la prochaine occurence du texte actuellement sélectionné
alt + ⌘ + F	rechercher et remplacer par expression régulière

configurer le clavier pour avoir ALT + ESP ouvrant la barre de menu



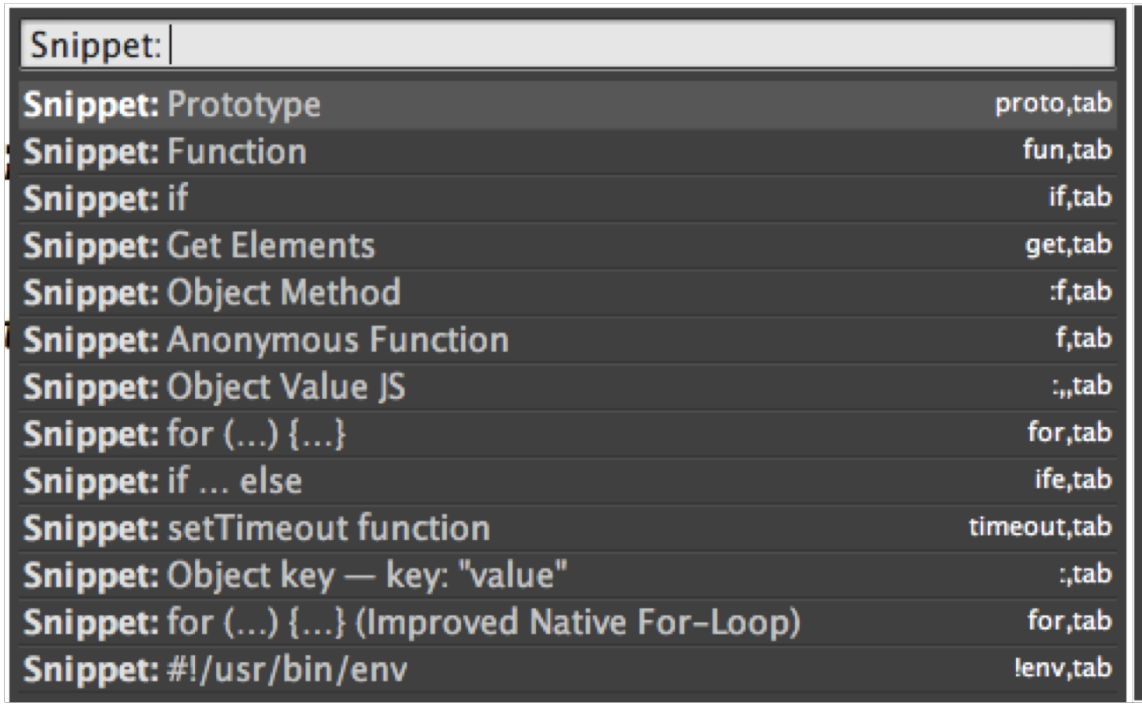
Sublime Text 2



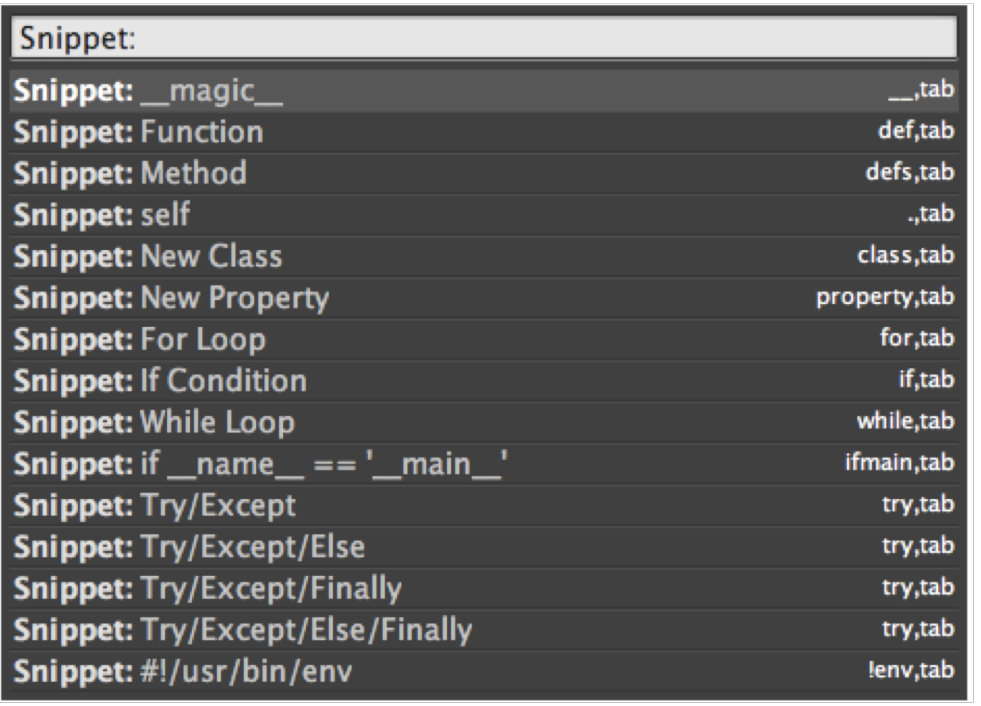


Sublime Text 3

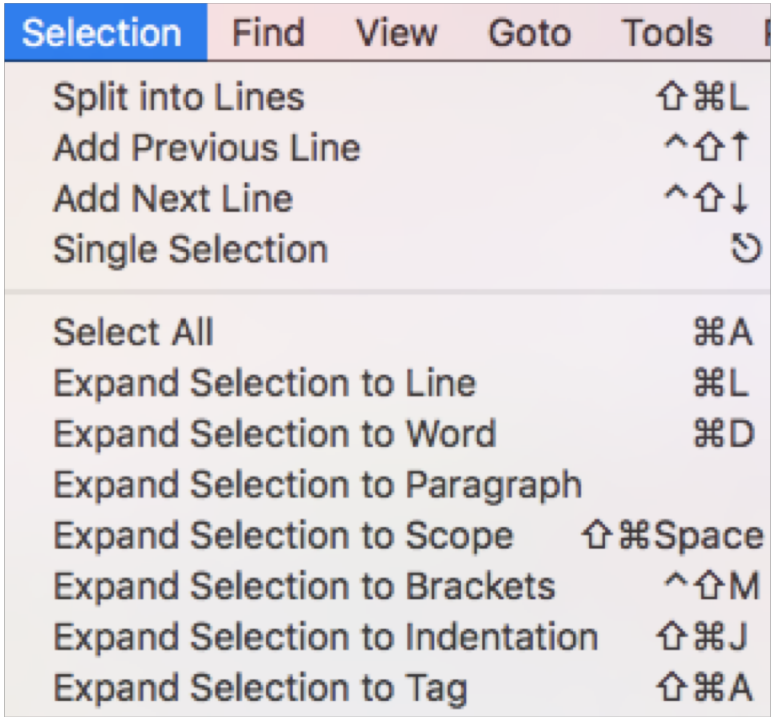
Javascript



Python



Sublime Text 4



Python - Environnement

Variables d'environnement (python 3)

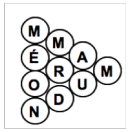
PYTHONPATH	Liste de répertoires où python peut chercher des modules
PYTHONSTARTUP	Chemin vers un fichier à exécuter au démarrage de python

Version de python

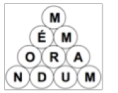
python2 **python3**	dans certains os on trouve deux commandes, pour pouvoir faire marcher les différents scripts
#! /usr/bin/env **python3**	**shebang** (1ère ligne du script) : permet de préciser le type de python voulu

$**chmod 755 script.py** rend le script exécutable (linux)
$**script.py** exécute le script avec la version de python du **shebang**

Python - Introduction



1. Historique et principes
2. Environnement de développement
3. Introduction à Python : syntaxe
    - Python avancé
4. Introduction à Scipy (Pandas)
5. Introduction à Matplotlib
    - Autres librairies graphiques
6. Introduction à Scikit-Learn
7. Préparation de données
8. DASK
9. Création d’une API Rest Django
10. Test Driven Development en Django
11. Gestion des environnements
12. Annexes



Bases
Ecrire un programme

Interpréteur

Python est interprété
Les variables sont des pointeurs vers des objets
Les instructions sont exécutées les unes à la suite des autres
Python est un ***eager language*** supportant mal la ***lazy computation*** (à l'inverse de spark)

Sémantique

L'indentation est signifiante

Une **tabulation** indique une dépendance des lignes par rapport à la précédente ligne non indentée
Cela permet
Définition de **boucles**
Définition de **fonctions**
Absence des points virgules et des accolades (mais on peut mettre plusieurs expression à la suite séparées par des **;**)

Comme en cobol, un numéro de colonne est obligatoire pour démarrer une ligne : la première colonne

la tabulation indique les instructions liées à la boucle

```
for mot in ['hello', 'world']:
    print mot
print "!"
$ hello
$ world
$ !
```

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

for cote in [ 1, 2, 3 ]:
    print "surface = ",au_carre(cote)
$ surface = 1
$ surface = 4
$ surface = 9
```

Bases
Scope

Local

Dans la même « suite » d'instructions
Par défaut les variables de la suite sont locales à la suite
Python cherche d'abord les variables locales

Global

Disponibles dans tout le **module**
Par défaut les variables ne sont pas globales
S'il ne trouve pas une variable locale avec le nom de la variable, Python cherche dans les variables globales
On déclare une variable globale via le mot clef **global**

Built-in

Définies par python
Disponible partout
Si le nom de variable n'est ni trouvé dans le scope **local** ni dans le scope **global** alors elle est cherchée dans le scope **built in**

Bases
Tout est objet

Sémantique

Tout est objet
- Les méthodes sont appelées via des points [.]
- Le type de l'objet est enregistré dans l'objet lui même

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

Les variables
- Ce sont des pointeurs vers des objets,
- Elle peuvent référer à des objets différents d'un moment à l'autre
- **Les objets sont typés, pas les variables.**

La fonction list renvoie une copie, et non un pointeur !

Les strings et les set sont immutables

```
x      = ["hello",  "world",  "!"]
y      = x
y[0] = "good night"
x
$ ["good night", "world", "!"]
```

```
x      = ["hello",  "world",  "!"]
y      = list(x)
y[0] = "good night"
x
$ ["hello", "world", "!"]
```

```
x      = "hello world !"
x[0] = "H"
$ error
type(x)
$ str
```

Bases
Types de données

Types de variables

Scalaires

None	unique dans le système
str	en python 2.7
unicode	préfixer par un u : **u**”hello world !”
bool	**True** / **False**
int	attention aux divisions
float	
long	

Objets

déclaration	**class**
auto-référence	**self**
attributs	variables accédés par un [.]
méthodes	fonctions(**self**) accédés par un [.]
attributs par défauts	**__dict__**

```
class livre:
        titre     = “”
        auteur = “”
        def afficher(self):
                print self.auteur + “:” + self.titre
x = livre()
x.titre = “Guerre et paix”
x.auteur = “Tolstoï”
x.afficher()
>Tolstoï : Guerre et paix
x.__dict__
>{“titre”:”Guerre et paix”, “auteur”: “Tolstoï “}
```

Bases
Subtilités des types de données

Nombres

En python 2.7 la division de deux entiers renvoie un entier (différent en python 3)

3/2 = 1

Pour diviser deux entiers: caster l'un des deux en float

3/**float**(2) = 1.5

Arrondi : //

3.0 // 2.0 = 1

Booléens

True / False

Les classes ont leur propre interprétations de True / False :

List [] == False
Tuple () == False
Set set() == False
['a', 'b', 'c'] == True
('a', 'b', 'c') == True
set('a', 'b', 'c') == True

String

Chaîne de caractère multilignes : """

"""
line 1
line 2
line 3 """""

Bases
Conditions, boucles

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

```
if  age < 10 :
        print “enfant”
elif age <18:
        print “adolescent”
else:
        print “adulte
```

```
def test(age):
        if  age < 10 :
                print “enfant”
        elif age <18:
                print “adolescent”
        else:
                print “adulte
```

```
for age in [9, 12, 20]:
        test(age)

enfant
adolescent
adulte
```

```
def divise(x,y):
        try :
                print x/y
        except:
                print “pbm”

for x,y in [( 10,5 ), (10, “hello world !”) ]:
                divise(x,y)
2
pbm
```

```
iteration = 0
while iteration < 10:
        iteration += 1
        print iteration
else :
       print « fin »
```

```
import random
max       = 100
cherche = random.randint(0, max)
iteration = 0
while iteration < max:
        if iteration == cherche :
                break
        iteration = iteration + 1
print u« la valeur random était :», iteration
```

Bases
Chaines de caractères - Formater des valeurs

Par position

```
" Bonjour %s                  ! " %(nom)
" La valeur est de %.2f       ! " %(valeur_float)     => 2 chiffres après la virgule
" La valeur est de %010.2f ! " %(valeur_float)     => 10 colonnes emplies de 0
" La valeur est de % 10.2f  ! " %(valeur_float)     => 10 colonnes emplies de blanc
```

Par noms de variables

```
nom = « romain »
message = "Bonjour {nom} ! »
message.format ( **locals() )
'Bonjour romain !'
for nom in ["a", "b"]:
   print message.format ( **locals() )
Bonjour a !
Bonjour b !
```

Listes - 1
Créations / insertions …

Création

[]	une_liste = [2,3,7, None]	

Insertions

append	une_liste.**append**(8)	[2,3,7, None, 8]
insert	**position** = 2 **valeur** = 4 une_liste.**insert**(position, valeur)	[2,3,**4**,7, None, 8]
+	une_liste **+** une_liste	[2,3, 4, 7, None, 8, **2,3,7, None, 8**]
extend	une_liste.**extend**(une_liste)	[2,3,4, 7, None, 8, **2,3,7, None, 8**]

Suppressions

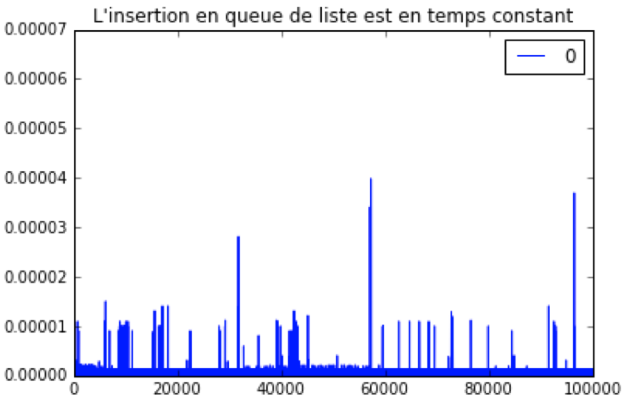
pop	une_liste.**pop**(**position**)	4
	une_liste	[2,3,7, None, 8 , 2, 3, 4, 7, None, 8]
remove	une_liste.**insert**(**position**, **valeur**)	[2, 3, **4,** 7, None, 8, 2, 3, 4, 7, None, 8]
	une_liste.**remove**(**valeur**)	[2, 3, 7, None, 8, 2, 3, 4, 7, None, 8]

Tris

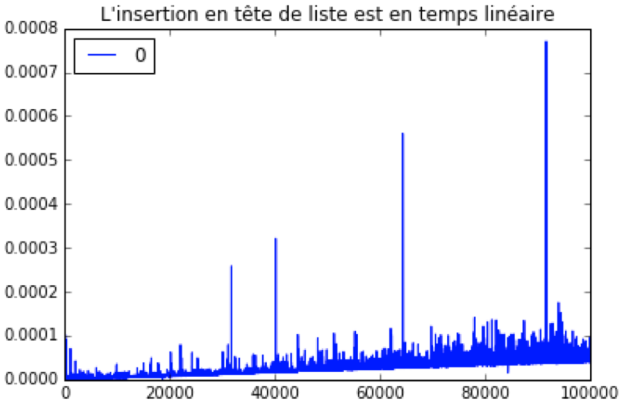
sort	une_liste.**sort**()	[None, None, 2, 2, 3, 3, 4, 7, 7, 8, 8]
bisect	import **bisect**	**// suppose la liste triée**
	bisect.**bisect**(une_liste, **5**)	**7** (indice du premier chiffre > 5)

Listes – 1 – L'insertion en fin de liste est en temps constant, en début de liste est linéaire

```python
conclusion = u"L'insertion en queue de liste est en temps constant"
liste      = list()
durees     = {}
for i in range(100000):
    debut = time.time()
    liste.append(i)
    fin   = time.time()
    durees[i] = fin - debut

df = pd.DataFrame.from_dict(durees, orient='index')
df.plot(title=conclusion)
```



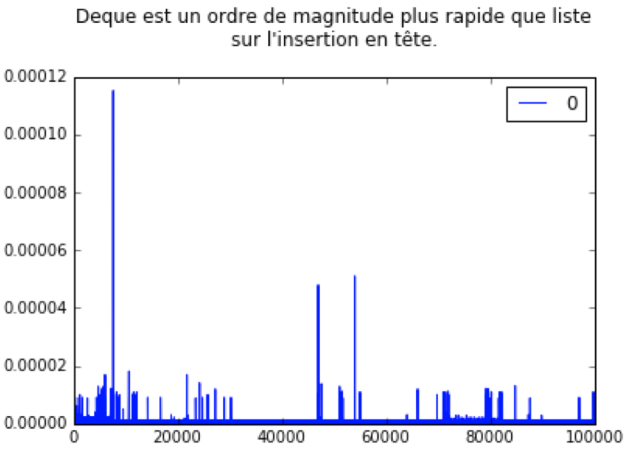

```python
conclusion         = u"L'insertion en tête de liste est en temps linéaire"
liste              = list()
position_insertion = 0
durees             = {}
for i in range(100000):
    debut = time.time()
    liste.insert(position_insertion, i)
    fin = time.time()
    durees[i] = fin - debut

df = pd.DataFrame.from_dict(durees, orient='index')
df.plot(title=conclusion)
```



https://github.com/romainjouin/formation_python/blob/master/python_avance_listes_et_deque.ipynb

Listes – 1 – la classe « double queue » permet d'obtenir un temps constant sur les deux types d'insertion

```
1  from collections import deque
2  conclusion          = u"""
3  Deque insert en tête de liste en temps constant
4  et en étant  un ordre de magnitude plus rapide.
5  """
6  double_queue        = deque()
7  position_insertion = 0
8  durees              = {}
9  for i in range(100000):
10     debut = time.time()
11     double_queue.appendleft(i)
12     fin = time.time()
13     durees[i] = fin - debut
14
15 df = pd.DataFrame.from_dict(durees, orient='index')
16 df.plot(title=conclusion)
```



Par contre deque n'a pas de fonction pour insérer à un index donné

Listes – 2
Extraction de sous parties, énumération

Slicing

```
une_liste                          = [ 1,  2,3,4 ]
position                           = 0

une_liste [  position ]            1
une_liste [ position : position +1 ]   [1]
une_liste [ position : position +2 ]   [1,2]
une_liste [  -1     ]              4
une_liste [  -1 :   ]               [4]
une_liste [  -2 :   ]              [3,4]
une_liste [  1  : 3]                [2,3]
```

Enumerate

```
for indice, valeur in enumerate(une_liste):
    #do something
```

Zip

```
zip( une_liste, une_liste )        [ (1,1) , (2,2), (3,3) , (4,4) ]
```

Tris

```
sorted    ( une_liste )            [ 1, 2, 3, 4 ]
reversed( une_liste )              [ 4, 3, 2, 1 ]
```

Fonctions - 1
Création, invocation

Définition

def nom (param1, parm2) Paramètres nommés

Référence

nom pointeur vers la fonction

Opérateur d'invocation

() évaluation de la fonction

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

```
square(3)
```

```
9
```

```
lien_vers_square = square
```

```
def apply(fonction, valeur):
    return fonction(valeur)
```

```
apply(lien_vers_square, 3)
```

```
9
```

Fonctions - 2
Map / Filter

Appliquer une fonction sur chaque élement d'une liste

map (**fonction**, **liste**) // retourne la liste des valeurs retournées par la **fonction**

filter (**fonction**, **liste**) // **fonction** doit retourner un booléen
// retourne la liste des valeurs de **liste** qui a retourné un booléen positif

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

```
liste          = [ -10, -5, 0, 5, 10]
liste_au_carre = map(square, liste)
liste_au_carre
```

```
[100, 25, 0, 25, 100]
```

```
def est_positif(nombre):
    return nombre > 0
```

```
liste_de_positifs = filter(est_positif, liste)
liste_de_positifs
```

```
[5, 10]
```

Fonctions - 3
lambda

lambda

carre = lambda x: x ** 2

Une ligne
Paramètres avant le double point
Algo après le double point
Pas de return

```
carre = lambda x: x*x
```

```
carre(3)
```

```
9
```

```
liste_au_carre = map(carre, liste)
liste_au_carre
```

```
[100, 25, 0, 25, 100]
```

Exercice :
Filtrez la liste sur les valeurs positives via un map

```
liste_au_carre = map(lambda x: x*x, liste)
liste_au_carre
```

```
[100, 25, 0, 25, 100]
```

```
En Python 3 : activer l'itérateur
avec la fonction 'list'
```

```
list(map(lambda nombre: nombre>0, liste))
```

```
[False, True, True, True, True, True, True,
```

```
# Python 3
list(filter(lambda nombre: nombre>0, liste))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Fonctions - 4
Avancées : *args et **kwargs

Définition

Voir : http://deusyss.developpez.com/tutoriels/Python/args_kwargs/

`def nom ( *args )` Liste de paramètres (longueur variable)

`def nom ( **kwargs )` Un dictionnaire en tant que paramètre

Passer une fonction en paramètre

```
def apply( fonction, liste) :
    for index, item in enumerate(liste):        # enumerate renvoie l'indice et la valeur
        liste[index] = fonction(item)           # évaluation de la fonction
```

```
def apply( fonction, liste) :
    for index, item in enumerate(liste):# enumerate renvoie l'indice et la valeur
        liste[index] = fonction(item) # évaluation de la fonction
```

```
liste=  [2,3]
apply(lien_vers_square, liste)
liste
```

```
[4, 9]
```

Set et tuples

Tuples

un_tuple = (1, 2,3,4)
une_liste = list(un_tuple)
un_tuple = tuple(une_liste)

Sets (ensembles)

un_ensemble = { 1, 2,3,4 }

```
valeurs_uniques = { 3, 3, 2}
valeurs_uniques
```

```
{2, 3}
```

valeur in ensemble

```
valeurs_uniques.add(5)
valeurs_uniques
```

```
{2, 3, 5}
```

```
valeurs_uniques.add(2)
valeurs_uniques
```

```
{2, 3, 5}
```

un_ensemble.**add (valeur)**
un_ensemble.**remove (valeur)** émet une erreur si la valeur est absente
un_ensemble.**discard (valeur)** silencieux
un_ensemble.**clear()**

```
ages   = set(range(0,120))
mineur = set(range(0,18))
mineur.issubset(ages)
```

```
True
```

un_ensemble.**issubset** (autre_ensemble)
un_ensemble.**issuperset** (autre_ensemble)

```
# intersection
mineur  = set(range(0,18))
adultes = set(range(19,120))
mineur & adultes
```

```
set()
```

un_ensemble | autre_ensemble union
un_ensemble & autre_ensemble intersection
un_ensemble - autre_ensemble difference (éléments dans le premier set, pas dans le second)

un_ensemble ^ autre_ensemble difference symétrique

Set et tuples

```
def compare_two_sets(big_liste, small_liste, to_print =""):
    """
    Print on stdout some info comparing to lists of elements (interesction / difference / subset...).

    Parameters :
        big_liste : left join columns
            pandas Series
        small_list : right join columns
            pandas Series
        to_print : message to print for the user
            string
    Return :
        nothing (print on stdout)

    Usage:
        It is useful to study to columns that are used as joining key to detect some
        unexpected unfitting. This function display on screen some set difference/intersection
        calculus as first approach to join operations.

    """

    big, small                  = set(big_liste), set(small_liste)
    len_big , len_small         = len(big), len(small)
    intersection                = big.intersection(small)
    difference                  = big.difference(small)
    big_is_subset_of_small      = big.issubset(small)
    small_is_subset_of_big      = small.issubset(big)
    elem_in_big_not_in_small    = big.difference(small)
    elem_in_small_not_in_big    = small.difference(big)
    nb_elem_in_big_not_in_small = len(elem_in_big_not_in_small)
    nb_elem_in_small_not_in_big = len(elem_in_small_not_in_big)
    print to_print
    print "len_big                      : ", len_big
    print "len_small                    : ", len_small
    print "nb of intersection           : ", len(intersection)
    print "nb_elem_in_big_not_in_small  : ", nb_elem_in_big_not_in_small
    print "nb_elem_in_small_not_in_big  : ", nb_elem_in_small_not_in_big
    print "big_is_subset_of_small       : ", big_is_subset_of_small
    print "small_is_subset_of_big       :  ", small_is_subset_of_big
    print "ex 5 elem_in_big_not_in_small : ", list(elem_in_big_not_in_small)[:5]
    print "ex 5 elem_in_small_not_in_big : ", list(elem_in_small_not_in_big)[:5]
```

https://github.com/romainjouin/data-science/blob/master/useful_functions.py

Dictionnaires - 1
Création, clefs, valeurs

Création

```
{ }          un_dico= { "nom"      : "jouin"    ,
                        "prénom"   : "romain"  }
```

Clefs et valeurs

```
un_dico.keys()                  ["nom", "prénom"]
un_dico.values()                ["jouin", "romain"]
```

Test l'existence d'une clef

```
"nom"      in un_dico           True
"romain"   in un_dico           False
```

Récupérer une valeur

```
un_dico["nom"]                  "jouin"
un_dico["prénom"]               "romain"
un_dico.get("ville", "inconnue")  "inconnue"
```

si ya pas cley cille va retourner incconnue comme valeur

Dictionnaires - 2
Manipulations : insertions / concaténation

Insertions

clef = “ville”
valeur = “paris”
un_dico[clef] = valeur

{“nom” : “jouin”, “prenom” : “romain”, “ville” : “paris” }

Concaténation

dico2 = {“mail” : “romain.jouin@memorandum.pro”}
un_dico.update(dico2)

{“nom” : “jouin”, “prenom” : “romain”, “ville” : “paris” , “mail” : “romain.jouin@memorandum.pro”}

Suppression

del	del un_dico[“mail”]	{“nom” : “jouin”, “prenom” : “romain”, “ville” : “paris” }
remplacement	un_dico[clef] = “rouen”	{“nom” : “jouin”, “prenom” : “romain”, “ville” : “rouen” }
pop	un_dico.pop(“prenom”)	{“nom” : “jouin”, “ville” : “rouen” }

Tri

sorted	sorted(dico)	renvoie une liste triée des clefs du dictionnaire !

Dictionnaires - 3
Tris

Tri de dictionnaire par valeur associée aux clefs

On ne peut pas trier un dictionnaire.
Ce qu'on peut faire ses faire une copie de ses clefs, trier ces clefs selon les valeurs associées à ces clefs, et les utiliser pour afficher les valeurs dans l'ordre de tri.

```
states= {'New Hampshire':'NH', 'Maryland':'MD',          # dictionnaire qu'on aimerait trier par clef
                        'Nevada':'NV', 'Maine':'ME'}

def fonction_de_tri(akey):                               # on s'appuie sur une fonction de tri
    return states[akey]                                  # ici on s'appuie sur le scope de variable global :-/


long_names = list(states.keys())                         # on prend les clefs du dictionnaire
long_names.sort(key=fonction_de_tri)                     # on trie la liste des clefs via la fonction ci dessus

for name in long_names:                                  # on utilise la liste triée pour afficher le dico dans
  print(name, states[name])                              # l'ordre naturel de ses valeurs associées
```

Dictionnaires - 4
DefaultDict

Dictionnaires

On veut parfois lister des éléments dans un dictionnaire.
Par exemple, lister les noms de clients par première lettre

```
clients = [‘airbus”, ‘airfrance’, ‘psa’]
 carnet = { }
for client in clients :
        first_letter = client [0]
        if first_letter not in carnet :
                carnet [ first_letter] = []
         carnet [ first_letter ].append(client)
```

DefaultDict

On peut supprimer le test d’existence de la clef en précisant que nous voulons une liste associée à la clef si la clef n’est pas présente. Cela grâce à la classe **defaultdict** :

```
from collections import defaultdict
carnet = defaultdict(list)
for client in clients :
        first_letter = client [0]
         carnet [first_letter ].append(client )
```

Dates

Dates

Les dates sont grées via le module ***datetime :***

Il propose trois types d'objets :

1 - **date**
2- **time**
3- **datetime** (contient une date et un time)

Import

```
from datetime import datetime, date, time
```

Utilisation

```
dt = datetime(2011,10,29,20,30,21)
dt.day
        29
dt.minute
        30
dt.date()
        datetime.date(2011,1,0,29)
dt.time()
        datetime.time(20,30,21)
```

Conversion

```
dt.strftime("%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)                      datetime.datetime(2011,10,29,20,0)
```

Timedelta

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

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 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 wee 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 wee as a decimal number. All days in a new year preceding the fir 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.	%

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'
```

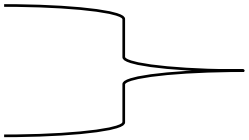
Vectorization de code
Comprehension list

Vectorization

En général, R et Python sont optimisés pour les opérations vectorielles.
En python il faut apprendre à remplacer les boucles par des ***comprehension list***.

Ex:

```
mois = [“janvier”, “fevrier”,”mars”]

liste = []
for month in mois :
    liste.append(len(month))
```



```
liste = [ len(month) for month in mois ]
```

Lambda functions

```
mois.sort(key = lambda x : len(x) )          # Lambda function !
```

Fichiers et Dossiers

FICHIER

path = “C:/.../fichier.csv”

Lecture

```
with open(path, ‘r’) as fichier:
    for line in fichier:
        print line
```

Ajout

```
with open(path, ‘a’) as fichier:
    fichier.write(“hello world!”)
```

Tout lire

```
contenu = open(path).readlines()
```

Ecrire plusieurs lignes

```
open(path).writelines(contenu)
```

DOSSIER

path_dir = “C:/dossier/”

Importer os

```
Import os
```

Tester l’existence

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

Créer un dossier

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

Fonction utile

```
import os
if not os.path.exists (path_dir):
        os.makedirs (path_dir)
        print "{0} created".format(path_dir)
```

Joindre un path de fichier et un path de dossier

```
os.join(dir_path, file_path)
```

Parcourir l'arborescence de fichiers

```
def get_all_specific_paths_under_dir(directory_path, wanted_end = ".csv"):
    """
    Recursevely walk a directory and create an array of path leading to a csv file.
        Parameter:
            directory:string
                Path to the root directory to walk through.
            wanted_ends:string
                Ending we are looking for into the filename. (ie: '.csv' for csv files)
                Default = ".csv" (ie returning list of csv files)int
        Return: array
            Array of paths (csv as defaults).
    """
    import os
    files_paths = []
    for dirname, dirnames, filenames in os.walk(directory_path) :
        for filename in filenames:
            if filename.endswith(wanted_end):
                files_paths.append(os.path.join(dirname, filename))
    return files_paths
```

```
get_all_specific_paths_under_dir(".")
```

```
[]
```


```
def get_all_specific_paths_under_dir(directory_path, wanted_end = ".csv"):
    """
    Recursevely walk a directory and create an array of path leading to a csv file.
        Parameter:
            directory:string
                Path to the root directory to walk through.
            wanted_ends:string
                Ending we are looking for into the filename. (ie: '.csv' for csv files)
                Default = ".csv" (ie returning list of csv files)int
        Return: array
            Array of paths (csv as defaults).
    """
    import os
    files_paths = []
    for dirname, dirnames, filenames in os.walk(directory_path) :
        for filename in filenames:
            if filename.endswith(wanted_end):
                files_paths.append(os.path.join(dirname, filename))
    return files_paths
```

Structuration des fichiers de code
Modules

Modules

Tout fichier python **.py** est un module — **module**.py

On importe un fichier avec le mot clef **import** — ***import*** **module**
Ensuite on peut utiliser les fonctions ou variables du module — **module.*function()***

On peut aussi renommer le module — ***import*** **module** ***as new_name***

On peut aussi importer les fonctions — **from module *import* function**
Et les renommer — **from module *import* function as new_name**

```
> import numpy
> liste = [ 1, 2, 3 ]
> numpy.max(liste)
3
```

```
> import numpy as np
> liste = [ 1, 2, 3 ]
> np.max(liste)
3
```

```
> from numpy import max
> liste = [ 1, 2, 3 ]
> max(liste)
3
```

```
> from numpy import max as maximum
> liste = [ 1, 2, 3 ]
> maximum(liste)
3
```

Python Path
Chemin de recherche de modules

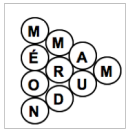
```
>>> import sys
>>> for path in sys.path:
...     print path
...
```

```
/Users/romain/Informatique
/Applications/anaconda/lib/python27.zip
/Applications/anaconda/lib/python2.7
/Applications/anaconda/lib/python2.7/plat-darwin
/Applications/anaconda/lib/python2.7/plat-mac
/Applications/anaconda/lib/python2.7/plat-mac/lib-scriptpackages
/Applications/anaconda/lib/python2.7/lib-tk
/Applications/anaconda/lib/python2.7/lib-old
/Applications/anaconda/lib/python2.7/lib-dynload
/Users/romain/.local/lib/python2.7/site-packages
/Applications/anaconda/lib/python2.7/site-packages
/Applications/anaconda/lib/python2.7/site-packages/Sphinx-1.3.5-py2.7.egg
/Applications/anaconda/lib/python2.7/site-packages/aeosa
/Applications/anaconda/lib/python2.7/site-packages/setuptools-20.3-py2.7.egg
```

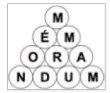
C’est la liste des chemins dans lesquels python cherche quand il y a un **import**

(Mis à jour par un **manage.py** dans Django)

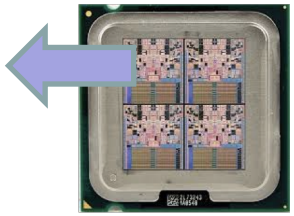
Python - Introduction



1. Historique et principes
2. Environnement de développement
3. Introduction à Python : syntaxe
    - Python avancé
4. Introduction à Scipy (Pandas)
5. Introduction à Matplotlib
    - Autres librairies graphiques
6. Introduction à Scikit-Learn
7. Préparation de données
8. DASK
9. Création d'une API Rest Django
10. Test Driven Development en Django
11. Gestion des environnements
12. Annexes

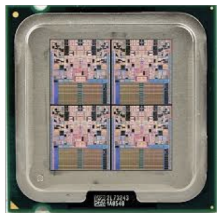


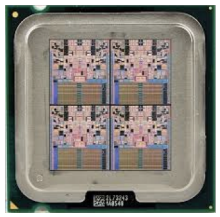
Parallélisation de code : architecture d'ordinateur

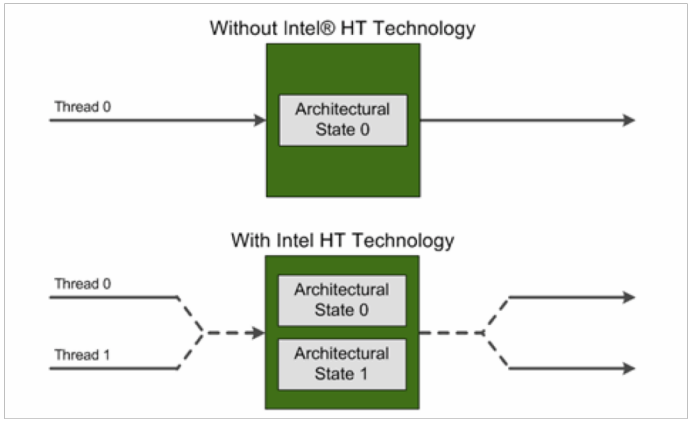


Intel® Hyper-Threading Technology

8 Cores logiques







Parallélisation de code
Multi processing

```
for indice_secteur in range(1,24):
    for annee in range(1997, 2017):
        for quarter in [1,2]:
            try:
                driver, ok = deal_with (driver, annee, quarter, indice_secteur, ok)
            except Exception as e:
                print "="*15
                print "boucle generale"
                print "secteur ", indice_secteur, annee, quarter
                print e
                a_refaire.append((indice_secteur, annee, quarter))
```

```
import multiprocessing
```

```
process_pool = multiprocessing.Pool(4)
inputs = []
for indice_secteur in range(1,25):
    for annee in range(1997, 2016):
        for quarter in [1,2]:
                inputs.append( (annee, quarter, indice_secteur))
```

```
process_pool.map(deal_with, inputs)
```

Sérialisation et désérialisation

Sérialiser	**Convertir un objet en chaine d'octets**	Permet le stockage ou le transfert via le réseau
Désérialiser	**Convertir une chaine d'octets en objet**	Permet de reconstituer l'objet reçu / stocké
Persistence	**Enregistrer une chaine d'octets sur un disque**	Permet de redémarrer un programme dans un état sauvé
Marshalling	**Envoyer une chaine d'octets sur un réseau**	Permet de distribuer des objets entre des PC
Caching	**Positionner un objet en mémoire**	Permet d'accéder rapidement à la donnée
Partionning	**Diviser une matrice entre *n* machine**	Permet d'optimiser l'usage des ressource

Named Tuple et persistance (1)

Théorie

from collections import namedtuple

namedtuple (‘nom’, [‘champ1’, ‘champ2’, ‘champ3’]) — Retourne une classe de nom ’nom ’

Pratique

positions = **namedtuple** ('Point', ['x' , 'y']) — Crée une classe 'Point’ Avec deux attributs x et y

coordonnee = **positions**(11, 22) — Instancie la classe 'Point'

print **type**(coordonnee) => <**class** '**collections**.**Point**'> — L’objet créé est bien de la classe 'Point'
print coordonnee => **Point**(**x**=11, **y**=22) — L’objet créé est un 'Point'

Pratique 1

```
diagonale = [("1", "1"), ("2", "2")]
for x_y in map(positions._make, diagonale):
    print x_y
```

La classe a une fonction factory **_make**

```
In [184]: positions   = namedtuple('Point', ['x', 'y'])
          coordonnee  = positions(11, 22)
          print type(coordonnee)
          print      coordonnee

          <class 'collections.Point'>
          Point(x=11, y=22)

In [191]: diagonale = [("1", "1"), ("2", "2")]

          for x_y in map(positions._make, diagonale):
              print x_y

          Point(x='1', y='1')
          Point(x='2', y='2')
```

Named Tuple et persistance (2)

Pratique 2

```
attributs      =    « nom, age, salaire »

employés       = namedtuple('employés', attributs)

for emp in map (          employés._make ,
                          csv.reader(open("employees.csv", "rb"))):
                print emp.nom, emp.salaire
```

les attributs peuvent être donnés sous forme de string

Cette string peut être la première ligne d'un fichier csv

La fonction **_make** nous permet de créer des objets très facilement à partir d'un fichier

Named Tuple (2)

Création d'objets

Ecriture dans un fichier csv

Vérification de l'écriture

Lecture à partir d'un fichier et création automatique d'objets

```
In [219]: champs = ["x","y"]
          path   = "./test.csv"

          diagonale = [("1", "1"), ("2", "2")]

          for x_y in map(positions._make, diagonale):
              print x_y

          Point(x='1', y='1')
          Point(x='2', y='2')
```

```
In [220]: with open(path, 'wb') as f:
              csv.writer(f).writerow(champs)

          with open(path, 'ab') as f:
              points = [point for point in map(positions._make, diagonale)]
              csv.writer(f).writerows(points)
              print points

          [Point(x='1', y='1'), Point(x='2', y='2')]
```

```
In [221]: !head test.csv

          x,y
          1,1
          2,2
```

```
In [230]: with open(path,'rU') as f:
              reader = csv.reader(f)
              header_lu = False
              for row in map(positions._make, reader):
                  if header_lu:
                      print row
                  header_lu = True

          Point(x='1', y='1')
          Point(x='2', y='2')
```

Named Tuple (2)

In [219]:
```
champs = ["x","y"]
path   = "./test.csv"

diagonale = [("1", "1"), ("2", "2")]

for x_y in map(positions._make, diagonale):
    print x_y
```

```
Point(x='1', y='1')
Point(x='2', y='2')
```

In [220]:
```
with open(path, 'wb') as f:
    csv.writer(f).writerow(champs)

with open(path, 'ab') as f:
    points = [point for point in map(positions._make, diagonale)]
    csv.writer(f).writerows(points)
    print points
```

```
[Point(x='1', y='1'), Point(x='2', y='2')]
```

In [221]:
```
!head test.csv
```

```
x,y
1,1
2,2
```

Named Tuple (3)
Stocker de la donnée (persistence)

CSV

```
import csv                                                    Module de gestion des csvs par python

with open ( path, 'wb') as f :                                Pour compléter un fichier existant : ab
        writer =  csv.writer ( f )
        writer.writerows ( fields )
        writer.writerows ( [ row for row in map ( positions._make , data ) ] )
```