Formation à Python pour les Sciences

Release 0.2

December 15, 2012

CONTENTS

Contenu

Matériel pédagogique sur l'écosystème Python. Une rapide introduction sur les outils et techniques clés. Chaque chapitre correspond à un cours de 1 à 2 heures avec un degré croissant de complexité.

Emmanuelle Gouillard

Pierre Haessig

Bernard Uguen

CHAPTER

ONE

FORMATIONPYTHON

Préparation d'une formation à Python et Python Scientifique

make html pour construire de document html - index.html dans _build/html/index.html make latexpdf pour construire le pdf

- le répertoire *discussions* est destiné à échanger sur le contenu, la forme, le fond, le plan
- le répertoire *contenu* est destiné à recevoir le contenu des séances de formation
- le répertoire *document* contient les ressources pdf, en particulier le document en cour d'élaboration *PythonScientifique.pdf*
- le répertoire *notebook* contient les notebooks associés à la formation

CHAPTER

TWO

LICENSE

CHAPTER

THREE

PREMIÈRE JOURNÉE : LE LANGAGE PYTHON

3.1 Séance 1 : Un apercu des possibles utilisations de Python

- Objectif de la formation
- **Présentation générale du monde Python**
  - Historique
  - Le tao de python
  - Les différentes communautés Python
  - Modules et espace de nom
  - **Les bibliothèques - modules**
    - numpy scipy matplotlib
    - scikit-image scikits-learn
    - networkx, shapely
    - simpy, SymPy
- **les outils de travail avec Python**
  - éditeur de texte + IPython (commandes intéractives + *%run*)
  - environnement de développement intégré (ex. Spyder)
  - Notebook IPython (proche de Maple/Mathematica)

3.2 Séance 2 : Éléments de base du langage

- **structures de données**
  - (),[],{}
  - listes en compréhension
- **éléments du langage** boucles, conditions, fonctions, itérateur, map , enumerate
- Exemple en algorithmique de base

```
In [1]: def tri1():
```

3.3 Séance 3 : Chaînes et fichiers

- **traitement des chaines de caractères**
  - s.replace()
  - s1 + s2
- un exemple de regexp simple
- **type de fichiers**
  - mode d'accès
- glob.glob
- Sans doute ces points peuvent être intégrés dans la séance 2.

3.4 Séance 4 : Python Objet & Bases de données

- Classe
- Méthodes
- Surcharge d'opérateurs
- Construire un exemple de classe progressivement

  Idéalement un exemple avec l'utilisation d'une base de données MySQL utiliser *pymysql*

CHAPTER

FOUR

DEUXIÈME JOURNÉE : PYTHON SCIENTIFIQUE

4.1 Séance 5 : Calcul numérique avec Numpy

- Lecture fichiers (type structuré)
- Algèbre de base
- broadcasting
- stacking(hstack,vstack,dstack)
- boucle ou pas boucle einsum vs numba comme exemple

4.2 Séance 6 : Graphiques avec Matplotlib

- visite de la grande galerie
- construction d'un graphe simple en 2d en ajoutant des éléments graduellement pour enrichir le graphe (légendes, titre, ...)
- imshow , contouplot
- 3D ?

4.3 Séance 7 : Calcul scientifique avec Scipy

- optimisation
- intégration, ode
- stats

4.4 Séance 8 : Cas pratique de mise en oeuvre

1. Récupérer des données physiques ouvertes sur le réseau (T°, ...)
2. Appliquer un traitement

3. Mettre en forme une représentation graphique des données

CHAPTER

FIVE

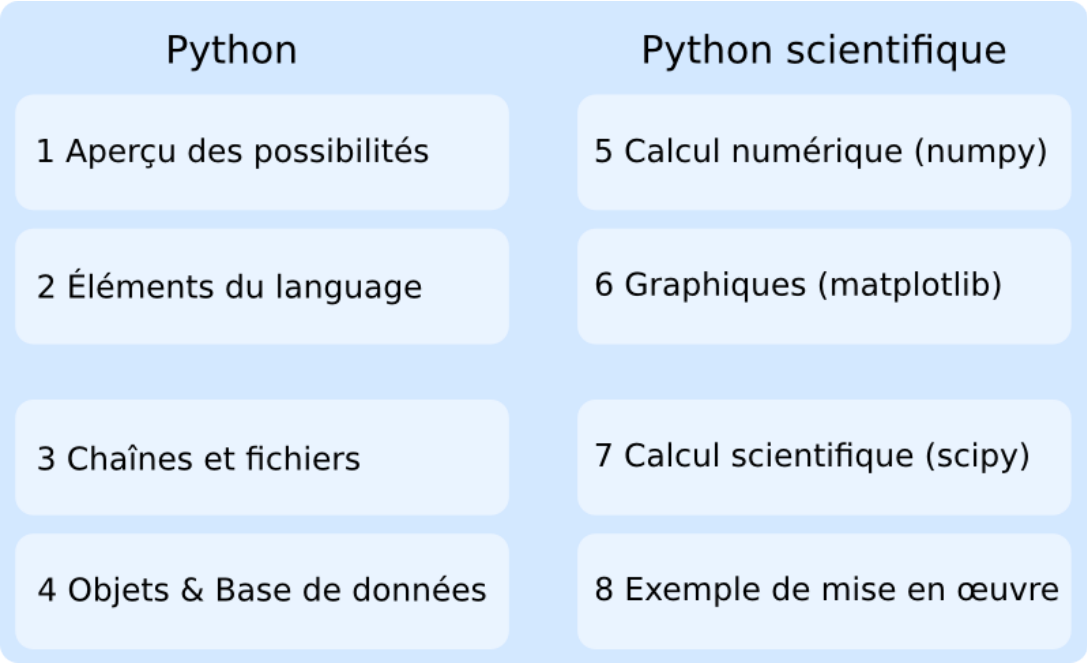
INTRODUCTION

Ce document est destiné à élaborer collaborativement une formation à Python et Python scientifique (Numpy, Scipy, matplotlib, ...) à destination des enseignants des classes préparatoires aux grandes écoles.

La formation, prévue sur *deux jours* comporte 2x4 séances de 1h30 (12h au total), avec

- un premier jour centré sur Python "en général"
- et le 2ème sur Python scientifique

Organisation éventuellement à revoir avec une possibilité de démarrer la présentation des aspects scientiques plus tôt



Pour rendre un peu plus concrets les concepts présentés, chaque séance s'appuira sur un ou des exemples.

CHAPTER

SIX

SÉANCES

6.1 Généralités sur Python et l'intérêt de son utilisation

Python est un language interprété. L'utilisateur voit immédiatement l'effet de sa commande. Ce mécanisme s'avère être très utile en sciences où beaucoup de traitements complexes sont menées pas à pas, en vérifiant et en testant à chaque pas la validité des résultats.

Le langage Python rencontre aujourd'hui un engouement très grand dans de nombreuses communautés scientifiques qui ont la nécessité de "manipuler" des données et de les faire parler. Ce mouvement est profond et potentiellement extrèmement fécond car il génère des liens inter-disciplinaires autour de besoins communs : la maîtrise du calcul, la maîtrise des données, la communication des résultats.

Pour accompagner ce mouvement et pour l'accentuer, il importe que de plus en plus de scientifiques s'emparent de ces outils et apprennent à en maîtriser la puissance et à en apprécier l'élégance.

6.1.1 Premier exemple

Voici un exemple d'une session très simple, exécutée dans l'environnement interactif Ipython (développé à l'Université de Berkeley par Fernando Perez) qui vise à devenir une plateforme générique de l'utilisation du langage Python dans le domaine des sciences.

```
In [1]: from math import *

In [2]: cos(pi/3)**2+sin(pi/3)**2
Out[2]: 1.0

In [3]: 1j*1j
Out[3]: (-1+0j)
```

Note: La première étape d'un programme Python consiste à charger des modules spécialisés. Ceux-ci sont très nombreux et couvrent un très large éventail de besoins. **L'objectif de la formation sera de comprendre les modules importants pour le calcul scientifique**

Par exemple dans l'exemple précédent le module *math* donne accès aux fonctions de la librairie standard *math* du langage C. Sans cette importation, il serait impossible d'accèder aux fonctions $sin(x)$ et $cos(x)$.

Note: Une fonction se reconnait à son prototype qui utilise obligatoirement les parenthèses *unnomdefunction()*

```
In [1]: exp(1j*pi)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-860da8d890b6> in <module>()
----> 1 exp(1j*pi)

TypeError: can't convert complex to float
```

L'interpréteur du langage renvoie un message d'erreur indiquant qu'il ne sait pas calculer une exponentielle avec un argument complexe.

C'est gênant.

Fort heureusement, il existe d'autres modules, plus complets, qui peuvent prendre le relais. Parmi ceux-ci le module *numpy*.

```
In [1]: from numpy import *

In [2]: exp(1j*pi)
Out[2]: (-1+1.2246063538223773e-16j)
```

C'est mieux ! Mais viens alors la question : Comment fait Python pour savoir quelle exponentielle choisir, puisqu'il a le choix entre l'exponentielle du module *math* et l'exponentielle du module *numpy* ?

C'est effectivement un problème. Il vient de se créer un conflit entre espaces de noms : celui du module *math* et celui de *numpy*. Un même nom est partagé par plusieurs modules.

On peut régler ce probléme en important le module différemment. Il y a trois possibilités :

- importer le module (sans son espace de noms)
- importer le module * (avec son espace de noms)
- importer le module avec un alias (souvent court par commodité)

```
In [1]: import numpy

In [2]: from numpy import *

In [3]: import numpy as np
```

La troisième solution est la bonne pratique.

L'inconvénient, c'est qu'il faut alonger le nom des fonctions appartenant au module en les faisant précéder de **np.**, (**numpy.** dans le premier cas), ce qui nuit (un peu) à la lisibilité du code.

Le grand avantage est que l'on règle ainsi le problème du conflit entre espaces de noms. Celui ci peut être très gênant et souvent source d'erreurs pénibles à découvrir (*la fonction appelée n'étant pas la fonction que l'on croit être appelée*). Ceci peut facilement se produire, car quel développeur peut prétendre avoir mémorisé l'intégralité des fonctions de tous les modules dont il a l'usage ? C'est trop gros, trop vaste, il faut cloisonner, et Python permet cela.

Bien sûr si l'usage que l'on a de python implique très peu de module l'import * est raisonable.

Les librairies importantes ont des alias génériques adoptés par tous qu'il convient d'employer.

```
In [1]: import numpy as np

In [2]: import scipy as sp

In [3]: import scipy.io as ios

In [4]: import scipy.linalg as la
```

```
In [5]: import matplotlib.pyplot as plt
```

6.2 Les bases du langage Python

6.3 Manipulation des chaines de caractères et des fichiers

6.4 Algorithmique et notions d'objects

6.5 Numpy : Créer et manipuler des données numériques où Python et les Nombres

numpy chez scipy lectures

6.6 Séance 5 : Tracé de graphes avec Matplotlib

scipy-lectures

6.7 Scipy et le calcul scientifique de haut niveau

scipy

6.8 Une application pratique de synthèse

CHAPTER

SEVEN

AUTRES

7.1 Liens Utiles

scipy-lectures pep8