Introduction à la Programmation Python

Généralités

Formation permanente du CNRS, Délégation Alsace
Février - Mars 2017

Auteurs :

- Vincent Legoll (vincent.legoll@iphc.cnrs.fr)
- Matthieu Boileau (matthieu.boileau@math.unistra.fr)

Programme de la formation

Généralités

- Prise en main des notebooks
- Généralités sur le langage Python

  Notebook : 00-InitPython-generalites.ipynb

Le langage [1/3]

- Variables
- Types de données

  Notebook : [01-InitPython-langage.ipynb]

Le langage [2/3]

- Opérateurs
- Structures de contrôle
- Fonctions
- Exceptions et gestionnaires de contexte
- Compréhensions de listes & expressions génératrices
- Modules
- Bonnes pratiques
- Python 3.x vs 2.x
- Le Zen de Python

  Notebook : [02-InitPython-langage.ipynb]

Micro-projet

  Notebook : [03-InitPython-microprojet.ipynb]

Le langage [3/3]

Prise en main des notebooks

Le document que vous lisez est un notebook iPython. Il est constitué de cellules comportant :

- soit du texte en **Markdown** comme ici.
- soit du code Python, comme dans la cellule suivante:

```
In [1]: print('Hello world!')
```

```
Hello world!
```

> **Note :** - `print()` est une fonction fournie par python pour afficher du texte à destination de l'utilisateur. - les crochets [ ] à gauche de la cellule indiquent le nombre d'exécutions de la cellule. Ici, la cellule n'a pas encore été exécutée.

Les deux modes du notebook :

- *Commande* : permet de se déplacer d'une cellule à l'autre et d'exécuter les cellules
- *Edition* : permet de modifier le contenu d'une cellule.

Changement de mode:

- *Commande* -> *Edition* : touche "Enter" ou double-click dans la cellule.
- *Edition* -> *Commande* :
  - Touche "Esc" pour basculer sans exécuter
  - Touche "Shift + Enter" pour "exécuter" la cellule et passer à la suivante. Le résultat de l'exécution d'une cellule en Markdown donne le rendu visuel de celle-ci.

> **Exercice** : Revenez en arrière, sélectionnez et exécutez (SHIFT+ENTER) la cellule de code qui contient **print('Hello world!')**

Mode Commande

- On se déplace à l'aide des flèches haut/bas ou en cliquant avec la souris.
- On peut ajouter, effacer, déplacer, créer ou modifier le contenu des cellules à l'aide des menus déroulants en haut de la page

Mode Edition

- signalé par la petite icône "crayon" en haut à droite, dans la barre de menu

Note :

La plupart des actions à la souris peuvent se faire à l'aide des raccourcis du menu *Help* > *Keyboard Shortcuts* (touche H)

> **Exercice** : Revenez en arrière, sélectionnez la cellule **"Hello world"**. Modifiez son contenu, par exemple traduisez le message en français, réexécutez-la et observez le nouveau résultat. Remarquez que le nombre d'exécutions augmente.

Autres commandes utiles

- Redémarrer le kernel Python : Bouton correspondant ou touche "0,0" en mode *Commande*
- Changer le type de cellule: *Code -> Markdown* : touche "M" en mode *Commande*
- Changer le type de cellule: *Markdown -> Code* : touche "Y" en mode *Commande*
- Ajouter une nouvelle cellule au dessus de l'actuelle: touche "A" (pour above) en mode *Commande*
- Ajouter une nouvelle cellule en dessous de l'actuelle: touche "B" (pour below) en mode *Commande*

  Exercice 1. Suivre le tour guidé de l'interface : *Help* > *User Interface Tour* 2. Avancer dans le notebook avec "Shift + Enter" 3. Passer du mode **Commande** au mode **Edition** de différentes façons 4. Ouvrez la rubrique **Keyboard Shortcuts** de l'aide et tenter de reproduire les actions uniquement avec les touches du clavier

Notes :

- Vous pouvez vous servir de votre copie d'un notebook pour faire office de bloc-notes : vous pouvez rajouter des cellules de texte pour vos commentaires et des cellules de code pour vos essais de résolution d'exercices.
- "CTRL + S" pour sauver vos modifications
- Pensez à emporter le fichier ".ipynb" avec vous à la fin de la session.

  Exercice : 1. Ajoutez une cellule de texte ci-dessous (touches 'b' puis 'm'), et mettez-y une note. 2. Ajoutez une cellule de code ci-dessous (touche 'b'), et mettez-y un exemple de code.

La programmation, qu'est-ce que c'est ?

- On "parle" à l'ordinateur pour lui "demander" de faire quelque chose
- Un programme : algorithme, formules mathématiques, logique, recette de cuisine
- Les ingrédients pour la recette => les données : fichiers, mouvements de la souris, des touches entrées au clavier, des données en provenance du réseau...
- On utilise un (des) langage(s) de programmation. Langage dans le sens linguistique: vocabulaire, orthographe, syntaxe et dialectes...
- Un programme est écrit sous forme de texte structuré : ensemble de phrases ordonnées qui exprimment la recette

Le langage Python

- Langage interprété, originellement écrit en C
- Open-source, portable et disponible sur Unix, Windows, Mac OSX, etc.
- Syntaxe claire et simple
- Orienté objet
- Types nombreux et puissants
- Interfaces avec de nombreux autres langages et librairies
- Large spectre d'applications

Plus d'informations sur wikipedia ou sur le site officiel de python.

Historique

La genèse du langage date de la fin des années 80. Guido van Rossum, alors à l'Institut de Recherche en Mathématiques et Informatique hollandais (CWI) à Amsterdam a publié la version

0.9.0 de l'interpréteur en Février 1991. Il travaille maintenant pour dropbox après 7 ans chez google.

Plus d'histoire sur wikipedia.

L'histoire racontée par le créateur lui-même sur son blog sous forme d'anecdotes.

Qu'est-ce qu'un langage interprété ?

- Ordinateur => CPU => jeu d'instructions (ISA) => langage binaire
- Un langage de programmation permet d'écrire des programmes dans des langages mieux adaptés aux humains, mais nécessite une étape de traduction.
- Comme pour une langue étrangère, il nous faut un traducteur ou un interprète...
  - Le traducteur va lire le texte et en produire une version dans la langue étrangère.
  - L'interprète va lire le texte, et pendant sa lecture, effectuer la traduction en direct.
- Pour un langage informatique, c'est quasiment pareil, nous avons des compilateurs et des interpréteurs.
  - Les compilateurs, traduisent tout le code source en langage binaire utilisable directement par le CPU.
  - L'interpréteur lit une partie du code source et exécute directement les instructions binaires qui correspondent et passe à la suite.

Un langage interprété sera souvent moins rapide qu'un langage compilé, car les optimisations sont plus faciles a réaliser lors d'une compilation.

Cette différence a tendance a s'estomper avec l'apparition des techniques suivantes:

- JIT: compilation à la volée (Just In Time compilation)
- RTTS: spécialisation de types au moment de l'exécution (Run Time Type Specialization)

Une autre possibilité pour contourner la lenteur d'exécution d'un langage est de faire appel a des librairies externes programmées dans un langage compilé et optimisées. Cela est très efficace pour les parties du code qui sont utilisées de manière répétitive.

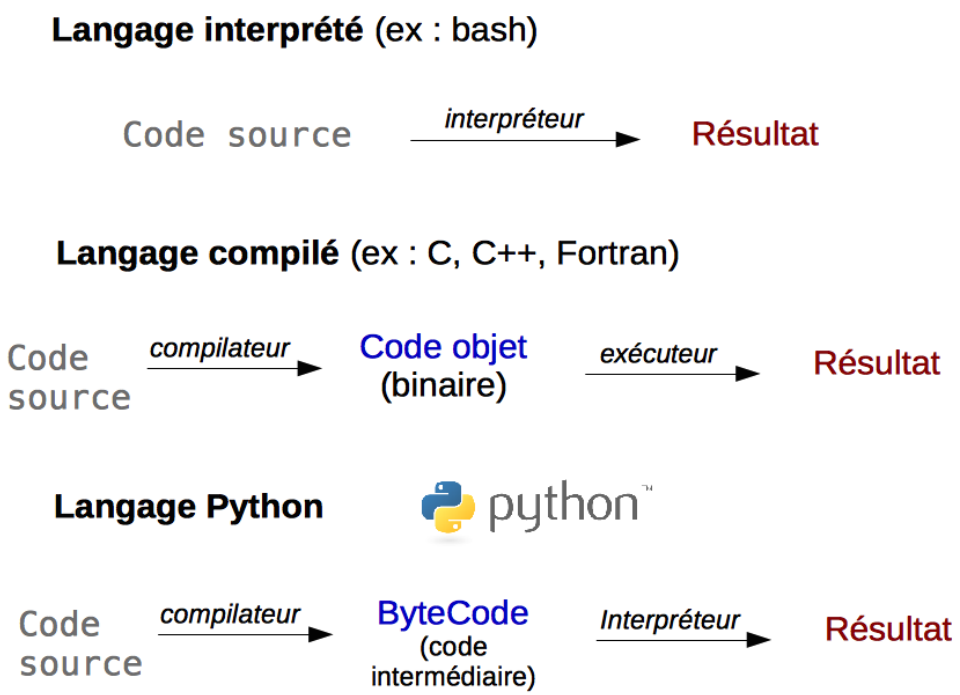

Figure inspirée du livre Apprendre à programmer en Python de G. Swinnen.

Quelques interpréteurs Python

- CPython => Implémentation de référence
- Jython => Java byte code, accès aux classes java
- IronPython => CLR byte code, accès aux classes .NET
- Pyjamas => JavaScript, Ajax, GWT
- Stackless Python => pas de pile, microthreads, coroutines
- Shed Skin => C++, typage statique
- Cython => C, créer des modules python
- Pyrex => langage proche de python, C
- Unladen Swallow => origine google, JIT, LLVM
- Pypy => JIT, RTTS, RPython -> C, Java byte code, CLR byte code
- Psyco => JIT, RTTS, x86, n'est plus maintenu
- Nuitka => C, fortement compatible
- Pyston => JIT, origine dropbox, n'est plus maintenu

Exécution d'un programme Python

Dans la console Python interactive

On peut exécuter le code directement à l'invite de l'interpréteur.

```
$ python
Python 3.5.2+ (default, Sep 22 2016, 12:18:14)
[GCC 6.2.0 20160927] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> print(a)
2
>>>
```

Note

Le caractère '$' est l'invite de commande typique d'un système d'exploitation linux, si vous utilisez une version de windows, l'invite de commande de cmd.exe sera plutot 'C:\>'.

Depuis la console système

On peut l'exécuter en paramètre de la ligne de commande

```
$ python -c 'a=3;print(a)'
```

Sous windows:

```
C:\> python.exe -c 'a=3;print(a)'
```

On peut exécuter un fichier (par exemple test.py) contenant notre code

```
$ python test.py
```

On peut exécuter directement un fichier python contenant notre code, grâce à l'utilisation du mode script avec, en rajoutant en première ligne du fichier:

```
#! python
```

Après avoir rendu le fichier exécutable

```
$ chmod a+x test.py
```

Ensuite il peut être exécuté sans donner l'interpréteur:

```
$ ./test.py
```

Dans la console iPython

Le terminal interactif iPython peut s'utiliser comme alternative à la console Python classique pour ses fonctionnalités :

- syntaxe additionnelle
- complétion
- commandes système
- historique enrichi

Aperçu du terminal ipython :

```
$ ipython
Python 3.5.2+ (default, Sep 22 2016, 12:18:14)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Un exemple d'utilisation

Sauvegarde de l'historique des commandes avec la *magic function* `%save`

```
$ ipython
Python 3.5.2+ (default, Sep 22 2016, 12:18:14)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: print('- Hello world!')
- Hello world!

In [2]: R = 'Hello you! '

In [3]: print(R*6)
Hello you! Hello you! Hello you! Hello you! Hello you! Hello you!

In [4]: %save hello.py 1-3
The following commands were written to file `hello.py`:
print('- Hello world!')
```

```
R = 'Hello you! '
print(R*6)

In [5]: quit
```

Au sein d'un notebook iPython

Les cellules de type *code* vous donnent accès à une console qui inclut la plupart des fonctionnalités de la console iPython.

Exercice :

1. Exécuter la cellule de code ci-dessous et observez le résultat
2. Exécuter cette cellule une deuxième fois, observez la différence

```
In [2]: print('- Hello world!')
        R = '- Hello you!\n'
        print(R*6)
        %history

- Hello world!
- Hello you!
- Hello you!
- Hello you!
- Hello you!
- Hello you!
- Hello you!

print('Hello world!')
print('- Hello world!')
R = '- Hello you!\n'
print(R*6)
%history
```

Utilisation d'un IDE (Environnement de développement intégré)

Un grand nombre d'IDE sont disponibles pour Python (cf. la revue wikipedia et la la revue wiki.python.org).

Citons simplement :

- **IDLE** : l'IDE par défaut de Python.
- **Spyder** : l'IDE qui sera utilisé pour certains exercices de ce cours.
- **Eclipse** : Un IDE initialement java, mais avec son système de greffons, il peut être utilisé pour d'autres langages, comme par exemple python avec PyDev.

Application du langage Python

Exemples d'applications courantes

Python est un langage complet, utilisable dans un grand nombre de domaines :

- bittorrent - téléchargement
- dropbox - partage de fichiers
- openstack - gestionnaire de cloud
- yum - installeur de paquets redhat, centos
- moinmoin - wiki
- trac - gestion de versions, bugs, wiki
- civilization IV - jeu de stratégie
- django - framework web
- zope - CMS, serveur d'applications web
- softimage - modelage et rendu 3D
- maya - modelage et rendu 3D
- paint shop pro - éditeur d'images
- gimp - éditeur d'images
- inkscape - éditeur de graphiques vectoriels
- reddit - réseau social, communauté
- yahoo groups - forums
- youtube - streaming vidéo

Plus d'exemples sur wikipedia

Intérêt pour les sciences

Pour un usage scientifique, Python est intéressant à plusieurs titres. En effet, il est capable de réaliser de manière automatique et efficace un certain nombre de tâches qui sont le quotidien des scientifiques : - Manipuler et traiter des données de simulations ou d'expériences - Visualiser des résultats - Communiquer ses résultats sous la forme de données numériques formatées, de figures ou d'animations

Dans le domaine du **calcul scientifique**, Python est particulièrement riche en fonctionnalités grâce à la contribution importante de la communauté des mathématiques et du calcul à travers le projet SciPy.

Le Python scientifique

1. Développement de code de simulation Bien que généralement moins performant que les langages compilés (C, C++ ou Fortran), Python est particulièrement intéressant et agréable à programmer dans les phases de développement pour tester rapidement de nouvelles méthodes. Une fois le prototypage terminé, il est posible de porter les parties critiques du code vers un langage compilé plus rapide, tout en gardant le reste en python.

2. Traitement de données

- langage de haut niveau produisant du code agréable à lire (par opposition à excel, par exemple...)

- nombreux modules spécialisés (algèbre, statistique, traitement d'images, etc.)
- le concept assez novateur des **notebooks iPython** qui combinent de l'exécution de code, du texte formaté, des formules mathématiques (via LaTeX), des tracés et du contenu média

3. Tracés graphiques (avec Matplotlib) :

- tracés 1D, 2D voire 3D
- animations

- On dispose ainsi d'un outil très complet et performant qui représente une alternative sérieuse aux outils commerciaux tels que Matlab, Maple, Mathematica, etc.
- SciPy fournit une excellente référence pour les applications scientifiques de Python dans ce document.

Les principaux paquets dédiés au Python scientifique :

- NumPy : calcul numérique, opérations mathématiques sur tableaux et matrices de grandes dimensions
- SciPy : ensemble d'outils scientifiques pour le traitement de signal, d'images, algèbre linéaire, etc...
- Sympy and SAGE : librairies et outils mathématiques pour le calcul symbolique
- Matplotlib : tracer et visualiser des données sous forme graphique, à la matlab ou mathematica
- Pandas : analyser vos données
- BioPython : problèmes de biologie : génomique, modélisation moléculaire, etc...
- AstroPy : librairie communautaire dédiée à l'astronomie

Migrer de MATLAB vers Python ?

- Une discussion intéressante : Python vs Matlab
- Un guide de migration : Numpy for Matlab users: guide
- Une table de conversion de la syntaxe : NumPy for MATLAB users: syntax

Documentation et sources

La documentation

- Officielle :
- L'index
- La FAQ
- La librairie standard
- Des tutoriels
- Stackoverflow : Forum de questions / réponses

Les sources de ce support

Quelques ressources qui ont inspiré le contenu de cette formation et qui pourront vous servir pour aller plus loin :

- **avec le langage**
  - Le MOOC de l'INRIA : Python : des fondamentaux à l'utilisation du langage hébergé sur la plateforme FUN (et qui sera probablement rejoué dans le futur)
  - La formation du groupe Calcul : ANF "Python avancé en calcul scientifique"
  - La formation de Pierre Navaro : Python pour le calcul
  - Le livre en ligne de Harold Erbin : Livre Python
  - Le livre de Gérard Swinnen : Apprendre à programmer en Python
  - Le cours de python scientifique de l'institut de science du téléscope spatial: STSCI's Scientific Python Course 2015 (en anglais)
- **avec les notebooks Jupyter**
  - Ce qu'on peut écrire en Markdown et en LaTeX dans les notebooks Jupyter et ce qu'on peut faire dans les cellules de code dans cette série de tutoriels
  - Pour mettre vos notebooks en ligne : nbviewer