Python pour le calcul scientifique

Petit tour d'horizon

Loïc Gouarin

Laboratoire de Mathématiques d'Orsay

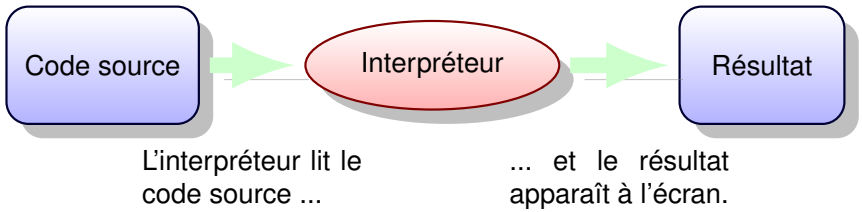
Le langage Python

1. développé en 1989 par Guido van Rossum
2. open-source
3. portable
4. orienté objet
5. dynamique
6. extensible
7. support pour l'intégration d'autres langages

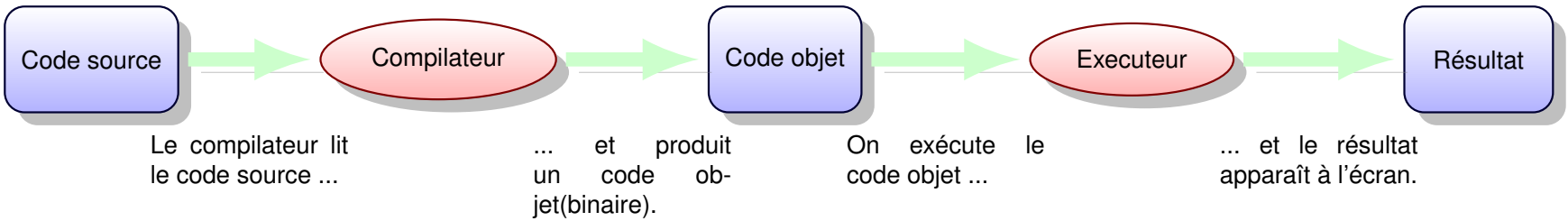
Comment faire fonctionner mon code source ?

Il existe 2 techniques principales pour effectuer la traduction en langage machine de mon code source :

- Interprétation

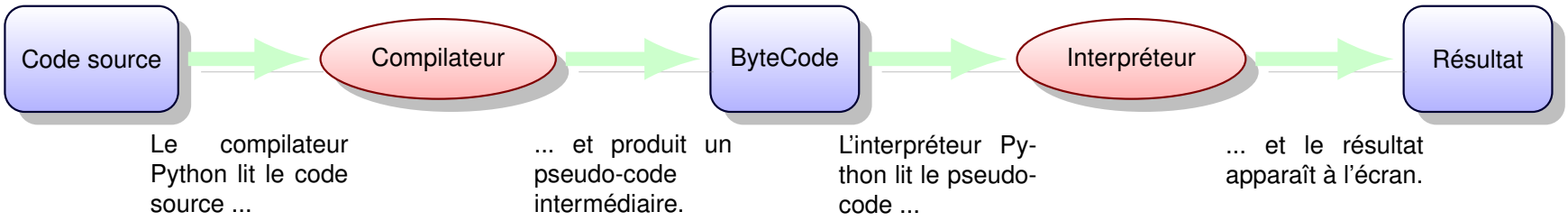


- Compilation



dessins tirés du livre Apprendre à programmer avec Python.

Et Python ?



Avantages :

- interpréteur permettant de tester n'importe quel petit bout de code,
- compilation transparente.

Inconvénients :

- peut être lent.

dessins tirés du livre Apprendre à programmer avec Python.

Les différentes implémentations

- **CPython**
  Implémentation de base basé sur le langage C ANSI
- **Jython**
  Implémentation permettant de mixer Python et java dans la même JVM
- **IronPython**
  Implémentation permettant d'utiliser Python pour Microsoft .NET
- **PyPy**
  Implémentation de Python en Python
- **...**

Les différentes versions

- Il existe 2 versions de Python : 2.7 et 3.3.
- Python 3.x n'est pas une simple amélioration ou extension de Python 2.x.
- Tant que les auteurs de librairies n'auront pas effectué la migration, les deux versions devront coexister.
- Nous nous intéresserons uniquement à Python 2.x.

Que peut-on faire avec Python ?

- **web**
  Django, TurboGears, Zope, Plone, ...
- **bases de données**
  MySQL, PostgrSQL, Oracle, ...
- **réseaux**
  TwistedMatrix, PyRO, ...
- **Gui**
  Gtk, Qt, Tcl/Tk, WxWidgets
- **représentation graphique**
  gnuplot, matplotlib, VTK, ...
- **calcul scientifique**
  numpy, scipy, sage, ...
- **...**

Pourquoi utiliser Python pour le calcul scientifique ?

- Peut être appris en quelques jours
- Permet de faire des tests rapides
- Alternative à Matlab, Octave, Scilab, ...
- Nombreux modules pour le calcul scientifique
- Tourne sur la plupart des plateformes
- Installation et tests unitaires très simples
- Parallélisation (MPI, cuda, openCL, thread, ...)
- Intégration avec C, C++, Fortran
- Recherche reproductible (Sumatra, IPython Notebook, ...)

Sous Linux

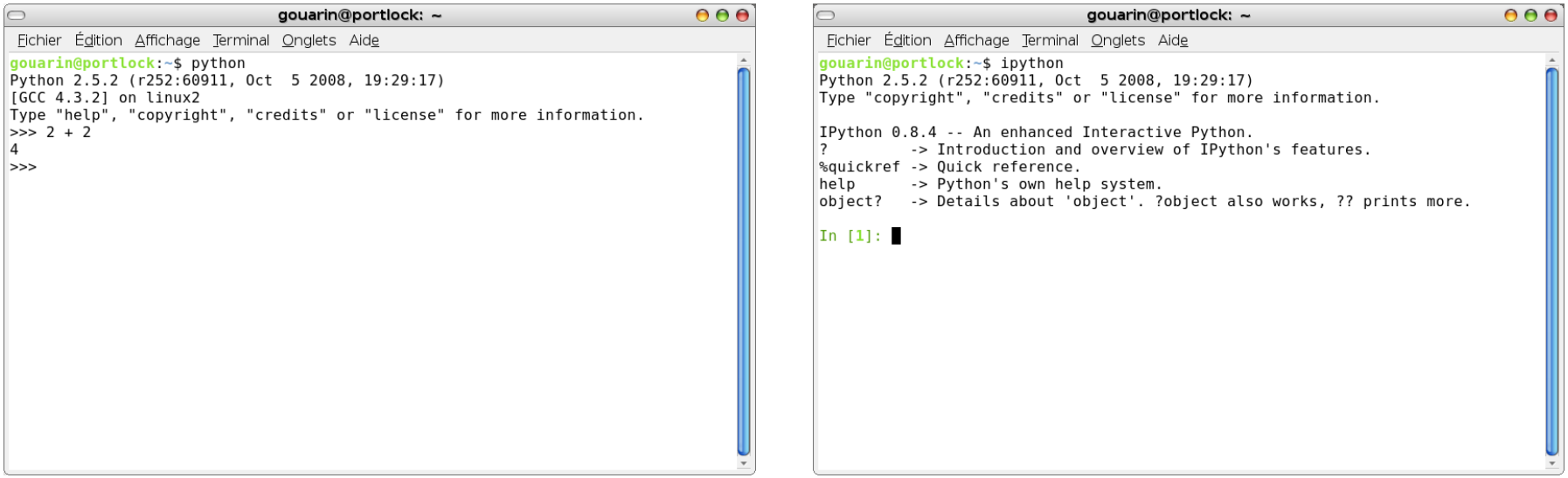


FIGURE: Interpréteur classique (gauche) et ipython (droite)

A vos claviers ! ! !

Les scripts

L'écriture de scripts Python se fait dans des fichiers ayant l'extension *.py*.

fibo.py

```
def print_fib(n):
    """
    ecrit la serie de Fibonacci jusqu'a n
    """
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a + b
    print

print_fib(10)
```

Exécution du script

```
$ python fibo.py
```

Options utiles de l'interpréteur

- `-c` : exécute la commande Python entrée après,
- `-i` : passe en mode intéractif après avoir exécuter un script ou une commande,
- `-d` : passe en mode debug.

Qu'est-ce qu'un module ?

Un module est un fichier comprenant un ensemble de définitions et d'instructions compréhensibles par Python.

Il permet d'étendre les fonctionnalités du langage.

Un module `M` peut être

- un fichier `M.pyd` ou `M.dll` (windows) ou `M.so` (linux)
- un fichier `M.py`
- un fichier `M.pyc`
- un répertoire `M` contenant un fichier `__init__.py`

Exemple : fibo.py

```
# Module nombres de Fibonacci
def print_fib(n):
    """
    ecrit la serie de Fibonacci jusqu'a n
    """
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a + b
    print
```

Exemple : fibo.py (suite)

```
def list_fib(n):
    """
    retourne la serie de Fibonacci jusqu'a n
    """
    result, a, b  = [], 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result
```

Utilisation du module fibo

```
>>> import fibo
>>> fibo.print_fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.list_fib(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

L'importation

Les différentes manières d'importer un module

- **import** fibo
- **import** fibo **as** f
- **from** fibo **import** print_fib, list_fib
- **from** fibo **import** * (importe tous les noms sauf variables et fonctions privées)

Remarque : En Python, les variables ou les fonctions privées commencent par _.

L'importation

Compléments sur *import*

import définit explicitement certains attributs du module :

- **__dict__** : dictionnaire utilisé par le module pour l'espace de noms des attributs
- **__name__** : nom du module
- **__file__** : fichier du module
- **__doc__** : documentation du module

Exécution d'un module

Ajout à la fin de fibo.py

```
if __name__ == '__main__':
    print_fib(1000)
    print list_fib(100)
```

Résultat

```
$ python fibo.py
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Installer ses propres modules

Le module `distutils`

- fournit un ensemble de commandes pour l'installation de packages ou de modules,
- gère les fichiers Python, C et C++,
- peut créer une distribution des sources,
- peut créer une distribution des binaires,
- ...

Exemple d'utilisation

setup.py

```
from distutils.core import setup

setup(name = 'monmodule',
      version = '1.0',
      author = 'loic Gouarin',
      description = 'mon premier module!!',
      url = 'http://monmodule.fr',
      py_modules = ['monfichier'],
      #packages = ['pack1'],
      #ou pack1 est un repertoire
      )
```

Construction du module

```
$ python setup.py build
```

Création du répertoire build

- contient les fichiers à installer
- lib.*plateforme* : modules pure Python et extensions
- temp.*plateforme* : fichiers temporaires générés lors de l'utilisation d'extension.

Installation du module

```
$ python setup.py install
```

- copie tout ce qu'il y a dans build/lib.*plateforme* dans le répertoire d'installation
- le répertoire d'installation par défaut est
  - windows :

    ```
    C:\Python
    ```
  - Unix :

    ```
    /usr/local/lib/pythonX.Y/site-packages
    /usr/local/lib/pythonX.Y/dist-packages
    ```

Installation du module

```
$ python setup.py install --user
```

Installation dans

```
$HOME/.local/lib/pythonX.Y/site-packages
```

```
$ python setup.py install --home=<dir>
```

Installation dans

```
<dir>/lib/python
```

```
$ python setup.py install --prefix=<dir>
```

Installation dans

```
<dir>/lib/pythonX.Y/site-packages
```

Ressources

Ressources générales

1. site officiel www.python.org
2. Apprendre à programmer avec Python

Ressources pour le calcul scientifique

1. site de Numpy.
2. Hans P. Langtangen, *Python Scripting for Computational Science*, Edition Springer, 2004.
3. Hans P. Langtangen, *A Primer on Scientific Programming with Python*, Edition Springer, 2009.