Une introduction au langage Python pour l'analyse de données (génétiques)

Bertrand Servin

Mars 2014

INRA
SCIENCE & IMPACT

Plan

1 Introduction

2 Le langage
- Exécution d'un programme python
- Types de données
- Fonctions de base
- Définir ses propres fonctions
- Conclusions

3 Liste d'outils pour le calcul et l'analyse de données
- Modules du projet Scipy
- Intégration de code C et fortran

Plan

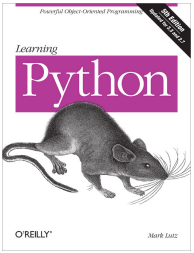
Caractéristiques du langage python

- Langage de **haut niveau** ("loin du hardware")
- Généraliste : nombreux domaines d'application (**calcul numérique** , administration système, développement web, jeux ...)
- multi paradigme : procédural, orienté objet, fonctionnel
- Gestion mémoire automatique (garbage collection)
- Très grande **bibliothèque standard**
- Syntaxe orientée sur la **lisibilité** du code
- **Plusieurs contextes d'utilisation**: Interface interactive (shell) > scripts > programmes > bibliothèques (modules)

Historique

- Premières versions du langage datent du début des années 90. Version 1.0 : 1994.
- Python 2.0 : 2000
- Aujourd'hui deux versions co-existent:
  - python 2.7 : Dernière version de la série des 2.x, la plus utilisée (ici aussi)
  - python 3.0 : version destinée à corriger certaines erreurs de design qui nécessitent de casser la compatibilité.
  - Certaines bibliothèques (modules) dont nous parlerons sont compatibles 2.7 pas 3.x
  - Il existe un outil de conversion automatique `2to3` qui semble assez efficace (non testé).

Mon historique personelle avec python



- Avant: développement C et R exclusivement
- Volonté d'apprendre un langage de haut niveau pour:
  - Manipuler / formater des fichiers rapidement
  - Développer rapidement. Tester des idées.
- Apprentissage très rapide, coder est agréable
- Exécution suffisamment rapide
- Réutilisation du code simple
- Aujourd'hui: remplace 99.9% du C et 80% du R (subsiste pour plot + packages spécifiques)

Philosophie

Les principes

There should be one – and preferably only one – obvious way to do it

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Plan

1 Introduction

2 Le langage
- Exécution d'un programme python
- Types de données
- Fonctions de base
- Définir ses propres fonctions
- Conclusions

3 Liste d'outils pour le calcul et l'analyse de données
- Modules du projet Scipy
- Intégration de code C et fortran

Plan

Lancer un programme python

Principe

- Ecrire une suite d'instruction dans un fichier texte `programme.py`.

```
print 'Hello World'
```

- Exécuter `python programme.py`

Déroulement

- L'**interpréteur** python compile programme.py en un bytecode (indépendant du système)
- Le bytecode est exécuté par une **Machine Virtuelle**.

Quelques remarques sur ce fonctionnement

- Pas d'étape de compilation explicite : l'environnement d'exécution et de développement sont les mêmes.
- Le bytecode n'est pas du langage machine: peut être plus lent que du code compilé
- Mais le code n'est pas toujours réinterprété non plus. Pour des modules (cf plus loin), python utilisera le byte-code directement sans passer par l'étape de compilation.
- Typiquement plus rapide qu'un langage interprété

Plan

1 Introduction

2 Le langage

- Exécution d'un programme python
- Types de données
- Fonctions de base
- Définir ses propres fonctions
- Conclusions

3 Liste d'outils pour le calcul et l'analyse de données

- Modules du projet Scipy
- Intégration de code C et fortran

Les nombres

Types

- Entiers (dimension arbitraire): `3486, -12, 999999999999L`
- Réels (précision dépend de l'architecture) : `3.2, 2.5e-10, 4e6`
- Complexes : `3+4j, 5J`
- Octets, Hexadécimaux : `0177, 0xFF`

- Opérateurs : `+, *, >>, **, /, %` ...
- Fonctions : `pow, abs, round, ...`
- Modules : `math (log, cos, gamma, pi ...)` , `random (randint, gauss, sample, shuffle ...)`

Chaînes de caractères (Strings)

Écriture

- Simple guillemets: `'Carmen'`
- Double guillemets: `"L'amour est l'enfant de bohème"`
- Échappement: `'L\' amour est l\'enfant de bohème\n'`
- Triple guillemens:
```
'''...opéra tragique en quatre actes de Georges Bizet...''',"""...un des rôles les plus exigeants du répertoire..."""
```

- Les formes simple et double guillemets sont les plus utilisées.
- La forme triple permet d'écrire du texte sur plusieurs lignes (inclusion de scripts, documentation ...).

Chaînes de caractères (Strings)

Actions

- Taille : `len('Carmen')`
- Concaténation : `"L'amour " + "est l'enfant de bohème"`
- Répétition : `"Il n'a " + 'jamais '*2 + 'connu de loi'`
- Exemples de **surcharge d'opérateurs**

Chaînes de caractères (Strings)

Actions

- Les strings sont des **séquences**

```
>>> titre='Carmen'
>>> for c in titre: print c,
...
C a r m e n
```

- Autre utilisation de l'opérateur `in`, recherche:

```
>>> 'b' in titre
False
>>> 'a' in titre
True
```

Chaînes de caractères (Strings)

Indexation, sous-chaînes

- En tant que séquence, on peut accéder aux éléments par index
- Les index sont utilisés 'à la C' de 0 à (n-1)
- Il est possible d'indéxer en partant de la fin avec des indices négatifs:

```
>>> titre[0]
'C'
>>> titre[-1]
'n'
```

- On peut accéder à une sous-chaine par tranchage (`slicing`):

```
>>> titre[1:5] ## ou titre[1:-1]
'arme'
```

Chaînes de caractères (Strings)

- Les chaînes ne sont pas modifiables (immutable):

```
>>> titre[0]='B'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

- Pour modifier, il faut créer :

```
>>> titre='B'+titre[1:]
>>> titre
'Barmen'
```

Chaînes de caractères (Strings)

Formatage

Utilisation du caractère % pour formater des chaînes de caractères. Similaire à la fonction C `sprintf`

```
>>> auteur='Georges Bizet'
>>> annee=1875
>>> '%s, %s (%d)' % (titre,auteur, annee)
'Carmen, Georges Bizet (1875)'
```

Ou utiliser la méthode `format` (compat. python3):

```
>>> '{0}, {1} ({2})'.format(titre,auteur,annee)
'Carmen, Georges Bizet (1875)'
```

qui permet de nombreuses subtilités.

Chaînes de caractères (Strings)

Méthodes

- Comptage : `'0001100110'.count('0')` → 6
- Recherche : opérateur `in` méthodes `find` , `index`

```
>>> s='genes.csv'
>>> '.' in s , s.find('.'), s.index('.')
(True, 5, 5)
>>> ',' in s , s.find(',')
(False, -1)
>>> s.index(',')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

Chaînes de caractères (Strings)

Méthodes: parsing de texte

Deux méthodes principales : `split` et `join`

```
>>> gene='14,14231363,14232541,MC1R'
>>> gene.split(',') ## renvoie une "list"
['14', '14231363', '14232541', 'MC1R']
>>> items=gene.split(',')
>>> '\t'.join(items) ## conc. les éléments avec la chaîne \t
'14\t14231363\t14232541\tMC1R'
>>> print '\t'.join(items)
14 14231363 14232541 MC1R
>>> gene.replace(',','\t') ## plus simple :)
'14\t14231363\t14232541\tMC1R'
```

Listes

- Le type **list** est un des types les plus utilisés en python
- Collection ordonnée (de gauche à droite) d'éléments
- De taille quelconque, peut grandir, rétrécir, être modifiée, être encapsulée (liste de listes de listes)
- Actions similaires à celles vues sur les chaînes

Listes par l'exemple


```
>>> gene='14,14231363,14232541,MC1R'
>>> L=gene.split(',')
>>> L[3]='ENSOARG00000002239' ### une liste est modifiable
>>> L
['14', '14231363', '14232541', 'ENSOARG00000002239']
>>> L.append('MC1R')
>>> L
['14', '14231363', '14232541', 'ENSOARG00000002239', 'MC1R']
>>> L[1]=int(L[1]) ## conversion de string vers integer
>>> L[2]=int(L[2])
>>> L ## les éléments sont de types différents
['14', 14231363, 14232541, 'ENSOARG00000002239', 'MC1R']
```

Listes par l'exemple


```
>>> L
['14', 14231363, 14232541, 'ENSOARG00000002239', 'MC1R']
>>> del L[3] ## suppression
>>> L
['14', 14231363, 14232541, 'MC1R']
>>> L2
['13', 63237431, 63242627, 'ASIP']
>>> genes=[L,L2] ### liste de listes
>>> genes
[['14', 14231363, 14232541, 'MC1R'],
 ['13', 63237431, 63242627, 'ASIP']]
```

Listes par l'exemple


```
>>> new_gene
['6', 70189729, 70234612, 'KIT']
>>> genes.append(new_gene)
>>> genes
[['14', 14231363, 14232541, 'MC1R'],
['13', 63237431, 63242627, 'ASIP'],
['6', 70189729, 70234612, 'KIT']]
>>> mygene=genes.pop() ## renvoie et supprime le dernier éléme
>>> mygene
['6', 70189729, 70234612, 'KIT']
>>> genes
[['14', 14231363, 14232541, 'MC1R'],
 ['13', 63237431, 63242627, 'ASIP']]
```

Et aussi

List Comprehension Un outil puissant permettant de générer de nouvelles listes avec une syntaxe simple.

```
### Un code horriblement long
>>> glen=[]
>>> for g in genes:
...     glen.append(g[2]-g[1])
### Un code beaucoup plus court et plus explicite :)
>>> glen=[ g[2]-g[1] for g in genes ]
```

Tuples : "Listes non modifiables", pour stocker une collection d'objet en garantissant son intégrité. Ils n'ont que deux méthodes : `count` et `index`. Ils s'écrivent en général avec des parenthèses :

```
('Carmen', 'Georges Bizet', 1875)
```

Dictionnaires

- Liste: collection d'objets ordonnée, accès par indice : `L[0]`
- Dictionnaire: collection d'objets non ordonnée, accès par `clef` : `R['D2']`
- Les clefs doivent être de type immutable : nombres, strings, ...

```
>>> genes ## rappel
[['14', 14231363, 14232541, 'MC1R'],
 ['13', 63237431, 63242627, 'ASIP'],
 ['6', 70189729, 70234612, 'KIT']]
>>> glen={} ## dictionnaire vide
>>> for g in genes:
...     glen[g[-1]]=g[2]-g[1] ## clef = nom du gène
>>> glen
{'MC1R': 1178, 'KIT': 44883, 'ASIP': 5196}
```

Dictionnaires par l'exemple


```
>>> glen.keys()
['MC1R', 'KIT', 'ASIP']
>>> glen.values()
[1178, 44883, 5196]
>>> glen.items() ## liste de tuples
[('MC1R', 1178), ('KIT', 44883), ('ASIP', 5196)]
>>> len(glen)
3
>>> for g in glen.keys():
...     print g,'\t',glen[g]
MC1R  1178
KIT  44883
ASIP  5196
```

Dictionnaires par l'exemple


```
>>> glen.items() ## rappel
[('MC1R', 1178), ('KIT', 44883), ('ASIP', 5196)]
## création a partir d'une liste de tuples
>>> dict(glen.items())
{'MC1R': 1178, 'ASIP': 5196, 'KIT': 44883}
## Warning: Ça commence à devenir moche :(
>>> dict([ (g[-1],g[2]-g[1]) for g in genes ])
{'MC1R': 1178, 'KIT': 44883, 'ASIP': 5196}
```

Fichiers

- Le type d'objet `file` sert à manipuler les fichiers dans des programmes python.
- Ces objets sont créés avec la fonction `open`.
- Ils fournissent des méthodes pour travailler sur les fichiers

```
>>> input=open('small_gene_list.csv','r') ## r: read, par défa
>>> input=open('small_gene_list.csv')
>>> input.close()
>>> output=open('/tmp/small_gene_list.tsv','w') ## write
>>> output=open('/tmp/small_gene_list.tsv','a') ## append
```

Lecture de fichiers


```
>>> input=open('small_gene_list.csv','r')
```

- Lecture de l'ensemble du fichier:
  - `input.read()` Une seule chaîne
  - `input.readlines()` Une liste de lignes (chaînes)

```
>>> input=open('small_gene_list.csv')
>>> input.read()
'14,14231363,14232541,MC1R\n13,63237431,63242627,ASIP\n6,7018
>>> input=open('small_gene_list.csv')
>>> input.readlines()
['14,14231363,14232541,MC1R\n',
 '13,63237431,63242627,ASIP\n',
 '6,70189729,70234612,KIT\n']
```

Lecture de fichiers

- Lecture d'une partie du fichier:
  - `input.read(N)` Lit N caractères
  - `input.readline()` lit une ligne

```
>>> input.read(10)
'14,1423136'
>>> input.readline()
'14,14231363,14232541,MC1R\n'
```

- Parcourir ligne par ligne:

```
>>> for ligne in input:
...     pass ## ici, on ne fait rien.
```

Écriture de fichiers


```
output=open('/tmp/toto','w')
```

- Méthodes d'un objet `file`:
  - `output.write(S)` : Écrit la chaîne S dans output
  - `output.writelines(L)` : Écrit la liste de chaines L dans output
- Utilisation du mot clef `print`:

```
print >> output, S  ## ajoute un \n final
print >> output, S, ## sans \n ajouté
```

Note: Incompatible avec python3 qui introduit une fonction de base `print()` en remplacement.

Fichiers compressés , archives

- gzip:

```
import gzip
f = gzip.open('small_gene_list.csv.gz')
file_content = f.read()
```

- bz2

```
import bz2
f = bz2.BZ2File('small_gene_list.csv.bz2')
file_content = f.read()
```

- Dans une archive tar:

```
import tarfile
myarch = tarfile.open('monarchive.tar.bz2') ## bzip'ed archiv
f = myarch.extractfile('small_gene_list.csv')
file_content = f.read()
```

Plan

help

Python dispose d'une aide en ligne consultable à partir de l'interpréteur en ligne de commande (à la R):

- `help()` ouvre une session interactive pour consulter l'aide. Rubriques: modules, keywords, topics
- Pour les modules, fonctions, méthodes `help(item)`:

```
>>> import random
>>> help(random) ## importer le module pour consulter
>>> help(abs)
>>> help(''.format)
```

Fonctions de conversion

float(x) Convertit la chaine ou le nombre x en réel.

```
>>> float("1.32e6")  ## ou float(1320000)
1320000.0
```

int(x) Convertit la chaine ou le nombre x en entier. Arrondit à l'entier inférieur si x est un réel

```
>>> int("42")
42
>>> int(2.8)
2
```

str(o) Renvoie une chaine de caractère repésentant l'objet o.

```
>>> str(10)
'10'
```

range, xrange

- `range` permet de créer des listes d'entiers successifs.
- La fonction `xrange` crée un **générateur**: elle ne construit pas la liste mais renvoie les éléments successivement.

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(1,5)
[1, 2, 3, 4]
>>> range(0,6,2)
[0, 2, 4]
>>> xrange(0,6,2)
xrange(0, 6, 2)
```

range, xrange


```
>>> L=range(100000000000) ## trop gros passera pas
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
MemoryError
>>> L=xrange(100000000000)
>>> for x in L:
        if x < 10:
            print x,
        else:
            break
0 1 2 3 4 5 6 7 8 9
```

sorted

Fonction générique de tri de liste (d'`iterables`)

```
>>> sorted(glen.keys()) ## ['ASIP', 'KIT', 'MC1R']
>>> sorted(glen.values()) ## [1178, 5196, 44883]
>>> sorted(glen.values(),reverse=True) ## [44883, 5196, 1178]
```

Arguments optionnels:

cmp Spécifie une fonction de comparaison entre deux éléments.

key Spécifie une fonction calculant une clef associée à un élément sur laquel le tri s'effectue.

Plan

1 Introduction

2 Le langage

- Exécution d'un programme python
- Types de données
- Fonctions de base
- Définir ses propres fonctions
- Conclusions

3 Liste d'outils pour le calcul et l'analyse de données

- Modules du projet Scipy
- Intégration de code C et fortran

Syntaxe de base

Une fonction est définie avec le mot clef `def`. Il est suivi du nom de la fonction et d'une liste entre () de ses arguments.

```
>>> def droite(x): # Ecrit le résultat de 2x+1
...     """Ecrit le résultat de 2x+1"""
...     print 2*x+1
>>> x=droite(2)
5
>>> x
>>> print x
None
>>> help(droite) ## grace à la docstring
```

Syntaxe

return

Le mot clef `return` permet de renvoyer la valeur de la fonction. S'il est omis, ou n'est pas suivi d'une expression, la fonction renvoie la valeur `None`.

```
>>> def droite(x): # Renvoie le résultat de 2x+1
...     """Renvoie le résultat de 2x+1"""
...     return 2*x+1
>>> x=droite(2)
>>> x
5
```

Arguments nommés et valeurs par défaut

il est possible de définir des valeurs par défaut pour certains arguments d'une fonction. Ces arguments nommés doivent être donnés après les autres (dits positionnels).

```
>>> def droite(x,a=2,b=1):
...     """Renvoie le résultat de a*x+b"""
...     return a*x+b
>>> droite(2) ## 5
>>> droite(2,a=1) ## 3
>>> droite(2,b=2,a=3) ## 8 ordre b,a n'import pas
>>> droite(b=2,a=3,x=2) ## ici non plus car nommés
>>> droite(b=2,a=3,2) ## aie
  File "<stdin>", line 1
SyntaxError: non-keyword arg after keyword arg
```

Expressions Lambda

De petites fonctions anonymes peuvent être créées en utilisant le mot clef `lambda`. Elles ne doivent être constituées que d'une seule expression.

Exemple: Tri des items d'un dictionnaire

```
>>> glen.items()
[('ASIP', 5196), ('MC1R', 1178), ('KIT', 44883)]
>>> sorted(glen.items(),key=lambda i: i[0])
[('ASIP', 5196), ('KIT', 44883), ('MC1R', 1178)]
>>> sorted(glen.items(),key=lambda i: i[1])
[('MC1R', 1178), ('ASIP', 5196), ('KIT', 44883)]
```

Plan

Interface interactive


```
$ python
Python 2.7.6 (default, Mar  8 2014, 08:59:18)
 ....
>>> print "Hello World"
Hello World
>>>
```

Script

script.py

```
#!/usr/bin/env python
print "Hello World"
```

Execution

```
python script.py
chmod +x script.py
./script.py
```

Programme

bonjour.py

```
def dis_bonjour():
    ''' Dis Bonjour '''
    print 'Hello World'

def main():
    ''' Fonction principale '''
    dis_bonjour()

if __name__=='__main__':
    # vaut True si éxécuté avec: python bonjour.py
    main()
```

Module

Importer tout le module

```
import bonjour
bonjour.dis_bonjour()
```

Importer un sous ensemble de fonctions

```
from bonjour import dis_bonjour
dis_bonjour()
```

Pour continuer

- Syntaxe
- Autres types
- Programmation Objet
- Exceptions
- Modules de la bibliothèque standard

Pour continuer à découvrir, apprendre les bases du langage:
`http://docs.python.org/2/`
et en particulier le tutorial:
`http://docs.python.org/2/tutorial/index.html`

Plan

1 Introduction

2 Le langage
- Exécution d'un programme python
- Types de données
- Fonctions de base
- Définir ses propres fonctions
- Conclusions

3 Liste d'outils pour le calcul et l'analyse de données
- Modules du projet Scipy
- Intégration de code C et fortran

Plan

1 Introduction

2 Le langage

- Exécution d'un programme python
- Types de données
- Fonctions de base
- Définir ses propres fonctions
- Conclusions

3 Liste d'outils pour le calcul et l'analyse de données

- Modules du projet Scipy
- Intégration de code C et fortran

Numpy

- Numpy est un ensemble de modules (package) dédié au calcul numérique en python
- Basé sur un nouveau type d'objet : le tableau (**array**) multidimensionnel homogène. Il est associé à un type (entier, flottant, complexe ...).
- Des modules numpy fournissent des fonctions pour le calcul numérique, en particulier:
  - Algèbre Linéaire (`numpy.linalg`): calcul matriciel, décompositions, résolution d'équations
  - Echantillonage aléatoire (`numpy.random`): Générateurs de nombres aléatoires, principales distributions, permutations, mélanges

Autres éléments du projet scipy

Bibliotheque Scipy
- Basé sur `numpy`
- Etend le panel de fonction de `numpy`
- `scipy.linalg` = extension de `numpy.linalg`
- `scipy.stats` : accès à l'ensemble des distributions statistiques /tests ...

Shell ipython Une interface en ligne de commande améliorée, qui inclut d'emblée les modules numpy et permet d'exécuter des commandes shell (`ls, cd ...`)

matplotlib Une bibliothèque de représentation graphique 2D. Peut être utilisée de manière interactive (dans ipython) ou au sein de programmes python /numpy.

Scikits

Packages pour certaines analyses plus spécialisées. En particulier, pour l'analyse de données génétiques on peut mentionner les packages :

- `scikit-learn`: Machine Learning. Stable.
  - classification (SVM, random forest ...)
  - clustering : k-means, hclust, melange gaussiens ...
  - regression : ridge, lasso, elastic net, Automatic Relevance Determination ...
  - Reduction de dimension : PCA, Analyse factorielle
- `scikit-statsmodel` : en developpement, : Modèles linéaires et linéaires généralisés, ANOVA, tests statistiques ...

Plan

1 Introduction

2 Le langage
- Exécution d'un programme python
- Types de données
- Fonctions de base
- Définir ses propres fonctions
- Conclusions

3 Liste d'outils pour le calcul et l'analyse de données
- Modules du projet Scipy
- Intégration de code C et fortran

Ecrire des extensions en C

- Python fournit une interface permettant :
  - d'écrire des extensions en C
  - d'inclure l'interpréteur dans un programme C.
  - Les mains dans le camboui ...
- `Cython` est un logiciel permettant de faciliter l'écriture d'extensions C.
  - Convertit automatiquement du code python en code C
  - Fournit une extension du langage python pour générer du code C plus efficace.
  - Compatible openMP pour la parallélisation fine du code.
- `pythran` : Meme objectif que cython. Génère du C++

Ecrire des extensions en Fortran

- `F2py` est un ensemble d'outils permettant de construire des interfaces entre fortran et python. Fait partie du projet `scipy`, dépend de numpy.
- Construire des modules codés en fortran utilisables en python
- Appeller des fonctions python depuis un code fortran

Conclusions

- Python est un langage complet, facile à apprendre
- Apprentissage progressif:
  - Interface interactive pour tester /apprendre
  - Langage de script pour la manipulation de fichiers
  - Langage objet pour l'écriture de programmes complets / de bibliothèques
  - Extension / Optimisation pour le calcul intensif
  - Packages d'analyse numérique / statistique
- Potentiel certain à remplacer R ...
- Bon candidat pour l'apprentissage d'un nouveau langage :)