

# Python avancé

Pierre Poulain

[pierre.poulain@univ-paris-diderot.fr](mailto:pierre.poulain@univ-paris-diderot.fr)

M2 BI – 09/2011



À l'exception des illustrations et images dont les crédits sont indiqués à la fin du document et dont les droits appartiennent à leurs auteurs respectifs, le reste de ce cours est sous licence Creative Commons Paternité (CC-BY).

<http://creativecommons.org/licenses/by/2.0/fr/>

# Menu

- |   |                        |   |                     |
|---|------------------------|---|---------------------|
| 1 | Introduction / rappels | 6 | Biopython           |
| 2 | Bonnes pratiques       | 7 | Programmation objet |
| 3 | Gestion des erreurs    | 8 | Tkinter             |
| 4 | Numpy                  | 9 | Subprocess          |
| 5 | Rpy                    |   |                     |

# Menu

1 **Introduction / rappels**

2 Bonnes pratiques

3 Gestion des erreurs

4 Numpy

5 Rpy

6 Biopython

7 Programmation objet

8 Tkinter

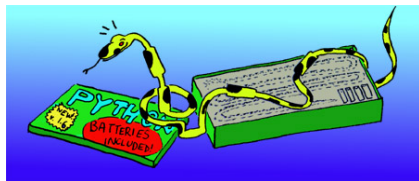
9 Subprocess

# Python, langage :

- interprété à *bytecode*
- multi-plateforme
- orienté objet (classes) « **tout est objet** »

# Python, langage : (2)

- « *batteries included* » nombreux modules fournis en standard (math, random, sys, os, re, urllib2, Tkinter)



- modules extérieurs (numpy, Biopython, rpy)
- une bibliothèque doit exister pour votre problème (sinon écrivez la)

## Objectif

Tour d'horizon de quelques fonctionnalités avancées (modules, classes, astuces)

# Menu

- 1 Introduction / rappels
- 2 Bonnes pratiques**
- 3 Gestion des erreurs
- 4 Numpy
- 5 Rpy
- 6 Biopython
- 7 Programmation objet
- 8 Tkinter
- 9 Subprocess

# Localisation Python et UTF-8

- 1 Dire où se trouve Python
- 2 Gérer les caractères accentués

```
#!/usr/bin/env python  
# -*- coding: utf-8 -*-
```



# Optimisation et itérations des boucles

```
toto = range( 1000000 )
```

```
# methode 1
```

```
for i in range( len(toto) ):  
    print toto[i]
```

```
# methode 2
```

```
for ele in toto:  
    print ele
```

```
# methode 3
```

```
for i in xrange( len(toto) ):  
    print toto[i]
```

Quelle méthode est la plus rapide ?

1. `for x in y`
2. `for z in xrange(len(y))`
3. `for z in range(len(y))` (très lente)



# Menu

- 1 Introduction / rappels
- 2 Bonnes pratiques
- 3 Gestion des erreurs**
- 4 Numpy
- 5 Rpy
- 6 Biopython
- 7 Programmation objet
- 8 Tkinter
- 9 Subprocess

# Gestion des erreurs

## But

Éviter le plantage d'un programme Python

- anticiper les erreurs
- et les intercepter

# Exemple

```
>>> nb = int(raw_input("Entrez un nombre: "))
Entrez un nombre: 23
>>> print nb
23
```

Mais si...

```
>>> nb = int(raw_input("Entrez un nombre: "))
Entrez un nombre: ATCG
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'ATCG'
```

`int()` ne peut pas convertir "ATCG" en entier.

# Try / except

```
>>> try:
...     nb = int(raw_input("Entrez un nombre: "))
... except:
...     print "Vous n'avez pas entré un nombre !"
...
```

```
Entrez un nombre:  ATCG
Vous n'avez pas entré un nombre !
```

`try` permet de « tenter » une instruction.  
En cas de problème, `except` prend la main.

# Try / except et les fichiers

```
>>> nom = "toto.pdb"
>>> try:
...     f = open(nom, "r")
... except:
...     print "Impossible d'ouvrir le fichier", nom
```

Fichier absent ou problème de droits en lecture → exception.

# Typage des exceptions

```
>>> try:
...     nb = int(raw_input("Entrez un nombre: "))
... except ValueError:
...     print "Vous n'avez pas entre un nombre !"
...
Entrez un nombre: ATCG
Vous n'avez pas entre un nombre !
```

`ValueError` → problème de conversion (avec `int()`)

`RuntimeError`, `TypeError`, `IOError`



# Menu


- 1 Introduction / rappels
- 2 Bonnes pratiques
- 3 Gestion des erreurs
- 4 Numpy**
- 5 Rpy
- 6 Biopython
- 7 Programmation objet
- 8 Tkinter
- 9 Subprocess

# Numpy (*Numerical Python*)

`http://numpy.scipy.org`

Bibliothèque de base pour le calcul numérique

- objet de type `array` à  $n$ -dimensions (matrices)
- algèbre linéaire (de base)
- transformée de Fourier (de base)
- générateur de nombre aléatoire (avancé)

 graphiques → autre modules (rpy, matplotlib)

# Objets de type array


Chargement du module numpy

```
>>> import numpy
```

Définition d'un vecteur

```
>>> vector1 = numpy.array([1,2,3,4])  
>>> vector1  
array([1, 2, 3, 4])
```

Construction automatique d'un vecteur

```
>>> data = numpy.arange(5)  
>>> data  
array([0, 1, 2, 3, 4])  
 numpy.arange() ~ range()
```

# Objets de type array (2)

## Opération vectorielle

```
>>> data = numpy.arange(5)
>>> data + 0.1
array([ 0.1,  1.1,  2.1,  3.1,  4.1])
```

# Redimensionnement

```
>>> v = numpy.arange(4)
>>> v
array([0, 1, 2, 3])
>>> numpy.reshape(v, (2,2))
array([[0, 1],
       [2, 3]])
```

array v doit avoir un nombre d'éléments compatibles avec la nouvelle matrice 2 x 2

```
>>> w = numpy.arange(4)
>>> numpy.resize(w, (3,2))
array([[0, 1],
       [2, 3],
       [0, 1]])
```

array w peut contenir un nombre quelconque d'éléments

# 0 & 1

Création d'une matrice de 0

```
>>> numpy.zeros((3,3))  
array([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])
```

Création d'une matrice de 1

```
>>> numpy.ones((2,2))  
array([[ 1.,  1.],  
       [ 1.,  1.]])  
>>> numpy.ones((2,2), int)  
array([[1, 1],  
       [1, 1]])
```

# Dimensions

```
>>> mat = numpy.reshape(numpy.arange(81), (9,9))
>>> numpy.shape(mat)
(9, 9)
>>> print mat.shape
(9, 9)
```

# Extraction

```
>>> a = numpy.resize(numpy.arange(9), (3,3))
```

```
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
>>> a [1,:]          # 2e ligne
array([3, 4, 5])
```

```
>>> a[:,2]          # 3e colonne
array([2, 5, 8])
```

```
>>> a [2,2]          # élément de la 3e ligne et 3e colonne
8
```



# Algèbre linéaire

```
>>> a = numpy.array([[1,2], [3,4]])  
>>> a  
array([[1, 2],  
       [3, 4]])
```

## Multiplication de matrices

```
>>> numpy.dot(a, a)  
array([[ 7, 10],  
       [15, 22]])
```


## Inversion de matrice

```
>>> from numpy import linalg  
# importation du module d'algebre lineaire  
>>> inv_a = linalg.inv(a)  
>>> inv_a  
array([[ -2. ,  1. ],  
       [ 1.5, -0.5]])
```

# Valeurs et vecteurs propres

```
>>> a = numpy.resize(numpy.arange(9), (3,3))
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

>>> import numpy.linalg as linalg
>>> eig_val, eig_vec = linalg.eig(a)
>>> eig_val
array([ 1.33484692e+01, -1.34846923e+00, -2.48477279e-16])
>>> eig_vec
array([[ 0.16476382,  0.79969966,  0.40824829],
       [ 0.50577448,  0.10420579, -0.81649658],
       [ 0.84678513, -0.59128809,  0.40824829]])
```

 utile pour connaître les axes principaux d'inertie d'une molécule

# Menu

- 1 Introduction / rappels
- 2 Bonnes pratiques
- 3 Gestion des erreurs
- 4 Numpy
- 5 Rpy**
- 6 Biopython
- 7 Programmation objet
- 8 Tkinter
- 9 Subprocess

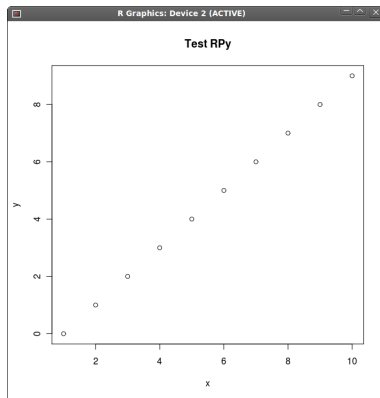
# Rpy (*R* for *Python*)

utilisation des fonctions de R dans Python

<http://rpy.sourceforge.net/>

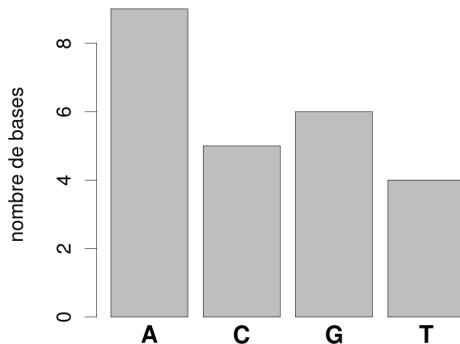
```
>>> from rpy import r as R
```

```
>>> R.plot(range(10), main="Test RPy", xlab="x", ylab="y")
```



# Exemple

Nombre de chaque base pour la séquence  
ACGATCATAGCGAGCTACGTAGAA



# Exemple

```
# -*- coding: utf-8 -*-

from rpy import r as R

seq = "ACGATCATAGCGAGCTACGTAGAA"
seq2 = list( seq )

R.png("rpy_test.png", width=800, height=800, pointsize=30)
# tri des bases car unique() n'ordonne pas les données
# alors que table() le fait
seq3 = R.sort( seq2 )
# listes des bases présentes
bases = R.unique( seq3 )
# effectif de chaque base
effectifs = R.table( seq3 )
# dessin du barplot et sauvegarde de la position des abscisses
coords = R.barplot( effectifs, ylab="nombre de bases")
# ajout du texte pour l'axe des abscisses
R.text(coords, -0.5, bases, xpd = True, cex = 1.2, font = 2 )
# fermeture du graphique
R.dev_off()
```



dev\_off()

# Menu

- 1 Introduction / rappels
- 2 Bonnes pratiques
- 3 Gestion des erreurs
- 4 Numpy
- 5 Rpy
- 6 Biopython**
- 7 Programmation objet
- 8 Tkinter
- 9 Subprocess

# Biopython

<http://biopython.org>

- manipulations de séquences (ADN, ARN, protéine)
- interrogations de banques de données biologiques (ExPASy, Entrez [NCBI], SCOP)
- recherches BLAST
- alignements multiples (clustalw)
- lectures de fichiers PDB
- ...

Très bon tutoriel à <http://www.biopython.org/DIST/docs/tutorial/Tutorial.html>



# Manipulation de séquences

## Définition d'un alphabet (ADN, ARN, protéine)

```
>>> from Bio.Alphabet import IUPAC
# module Biopython s'appelle Bio
# IUPAC = International Union of Pure and Applied Chemistry
```

## Définition d'un objet séquence

```
>>> my_dna_alphabet = IUPAC.unambiguous_dna
>>> from Bio.Seq import Seq
>>> my_seq = Seq('CATCCCTTCGATCGGGGCTATAGCTAGC', my_dna_alphabet)
>>> print my_seq
CATCCCTTCGATCGGGGCTATAGCTAGC
>>> my_seq
Seq('CATCCCTTCGATCGGGGCTATAGCTAGC', IUPACUnambiguousDNA())
```

# Manipulation de séquences (2)

## Propriétés des chaînes de caractères

```
>>> print my_seq[4:12]
CCTTCGAT
>>> print len(my_seq)
28
>>> new_seq = my_seq[0:5]
>>> new_seq
Seq('CATCC', IUPACUnambiguousDNA())
>>> my_seq + new_seq
Seq('CATCCCTTCGATCGGGGCTATAGCTAGCCATCC', IUPACUnambiguousDNA())
```

# Interrogation d'Entrez I

```
>>> from Bio import Entrez # chargement du module  
  
# definition de l'e-mail, obligatoire pour eviter les abus  
>>> Entrez.email = "votremail@provider.fr"
```

# Interrogation d'Entrez II

```
# requete dans la base de donnees pubmed
# des termes "small heat shock proteins"
>>> ma_req = Entrez.esearch(db="pubmed", \
... term="small heat shock proteins", retmax=50)

# recuperation des resultats
# sous la forme d'un dictionnaire
>>> mon_res = Entrez.read(ma_req)

# clefs disponibles
>>> print(mon_res.keys())
[u'Count', u'RetMax', u'IdList', u'TranslationStack',
u'TranslationSet', u'RetStart', u'QueryTranslation']

# liste des identifiants pubmed
>>> print(mon_res["IdList"])
['20679393', '20668846', '20668218', ...]
```

# Interrogation d'Entrez III

```
# requete sur un article en particulier
```

```
>>> requete = Entrez.esummary(db="pubmed", id='20668846')
```

```
# lecture du resultat
```

```
# sous forme d'une liste de dictionnaire
```

```
>>> res_parse = Entrez.read(requete)
```

```
# clefs disponibles
```

```
>>> print(res_parse[0].keys())
```

```
['DOI', 'Title', 'Source', 'PmcRefCount', 'Issue', ...]
```

```
# affiche du titre
```

```
>>> res_parse[0]["Title"]
```

```
'Characterization of Xanthomonas campestris pv. campestris  
heat shock protein A (HspA), which possesses an intrinsic  
ability to reactivate inactivated proteins.'
```

# En résumé

## Biopython

- puissant
- polyvalent
- la syntaxe change régulièrement
- le format des bases des données aussi...

Exemple plus poussé pendant le TP.

# Menu

- 1 Introduction / rappels
- 2 Bonnes pratiques
- 3 Gestion des erreurs
- 4 Numpy
- 5 Rpy
- 6 Biopython
- 7 Programmation objet**
- 8 Tkinter
- 9 Subprocess

# Programmation objet et classe

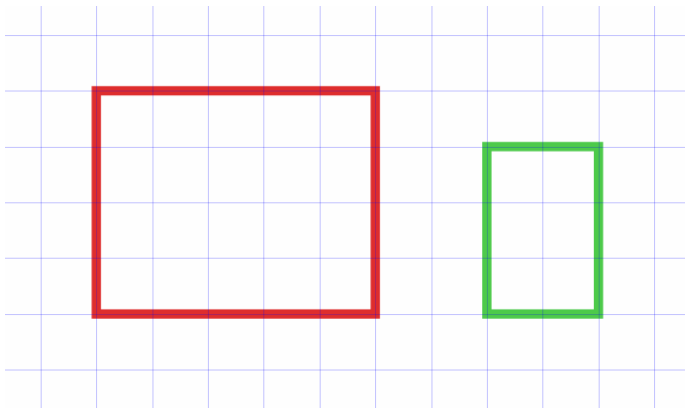
## Une minuscule introduction

Une classe définit des objets qui sont des **instances** (~ représentants) de cette classe.

Les objets possèdent des **attributs** (~ variables) et des **méthodes** (~ fonctions) associés à cette classe.



# Programmation objet et classe (2)



# Quelques propriétés

**Encapsulation.** Interface (attributs ou méthodes) « publique ». Possibilité d'interface « privée ».

**Polymorphisme.** Transformation des opérateurs standards (\*, +, /, -) suivant le contexte.

**Héritage multiple.** Création de sous-classes héritant des propriétés de la classe mère.

Python → programmation objet implicite.


# Exemple de classe Rectangle()

```
class Rectangle:
    """ceci est la classe Rectangle"""

    def __init__(self, long = 0.0, larg = 0.0, coul = "blanc"):
        """initialisation d'un objet (constructeur)"""
        self.longueur = long
        self.largeur = larg
        self.couleur = coul

    def calcule_surface(self):
        """calcule la surface"""
        return self.longueur * self.largeur

    def change_carre(self, cote):
        """transforme un rectangle en carre"""
        self.longueur = cote
        self.largeur = cote
```

 **self** désigne l'objet lui-même et est obligatoire.

# Utilisation de la classe Rectangle()

```
# creation d'un objet Rectangle avec les parametres par défaut
```

```
>>> rect1 = Rectangle()
```

```
# affichage des attributs
```

```
>>> print rect1.longueur, rect1.largeur, rect1.couleur  
0.0 0.0 blanc
```

```
# calcul de la surface
```

```
>>> print rect1.calcule_surface()  
0.0
```

```
# on change le rectangle en carre
```

```
>>> rect1.change_carre(30)  
>>> print rect1.calcule_surface()  
900
```

```
# creation d'un objet Rectangle avec des parametres imposes
```

```
>>> rect2 = Rectangle(2, 3, "rouge")  
>>> print rect2.calcule_surface()  
6
```

# Autres attributs redéfinissables

- `__add__(self, other)` opérateur +
- `__mul__(self, number)` opérateur \*
- `__del__(self)` destructeur
- `__len__(self)` opérateur `len` (taille)
- `__getslice__(self, low, high)` tranchage
- etc.

# Menu

- 1 Introduction / rappels
- 2 Bonnes pratiques
- 3 Gestion des erreurs
- 4 Numpy
- 5 Rpy
- 6 Biopython
- 7 Programmation objet
- 8 Tkinter**
- 9 Subprocess

# Tkinter

Tk (*Tool kit*) ensemble de fonctionnalités graphiques

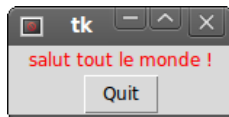
Tkinter → interface Python pour Tk

Développement de programmes Python avec des interfaces graphiques (*Graphical User Interface*, GUI).

Fourni en standard dans Python (Windows, Linux, MacOS)

# Exemple Tkinter

```
from Tkinter import *  
  
racine = Tk()  
  
texte = Label(racine, text="Salut tout le monde !", fg="red")  
texte.pack()  
  
bouton = Button(racine, text="Quit", command=racine.destroy)  
bouton.pack()  
  
racine.mainloop()
```





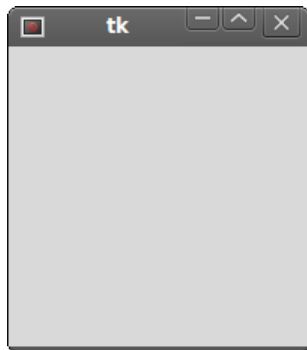
# Exemple Tkinter commenté I

- ✓ Importation du module Tkinter

```
from Tkinter import *
```

- ✓ Création d'une instance d'un objet graphique (*widget*) Tk, ici une fenêtre

```
racine = Tk()
```



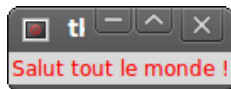
# Exemple Tkinter commenté II

- ✓ Création d'un objet de la classe `Label()` contenu dans `racine`.

```
message = Label(racine, text="Salut tout le monde !", fg="red")
```

Attributs de `message` : un texte et une couleur de texte.

- ✓ Accrochage du texte dans la fenêtre et ajustement de sa taille  
`message.pack()`



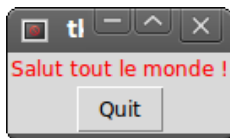
## Exemple Tkinter commenté III

- ✓ Création d'un objet graphique de la classe `Button()` dans `racine`.

```
bouton = Button(racine, text="Quit", command=racine.destroy)
```

Attributs de `bouton` : un texte et une commande.

- ✓ Accrochage du texte dans la fenêtre et ajustement de sa taille  
`bouton.pack()`



- ✓ Démarrage du gestionnaire d'évènements (clavier, souris).  
Obligatoire dans un script mais pas dans l'interpréteur.  
`racine.mainloop()`

# Menu

- 1 Introduction / rappels
- 2 Bonnes pratiques
- 3 Gestion des erreurs
- 4 Numpy
- 5 Rpy
- 6 Biopython
- 7 Programmation objet
- 8 Tkinter
- 9 Subprocess**

# subprocess

gestion des entrées / sorties d'une commande Unix

<http://docs.python.org/library/subprocess.html>

# Sortie standard

## Code

```
import subprocess
command = "ls"
proc = subprocess.Popen(command, shell=True,
stdout=subprocess.PIPE)

# contenu de la sortie standard
out = proc.communicate()[0]
print "===Sortie standard :"
print out
# affiche le code de sortie (0 = OK)
print "===Code de sortie :"
print proc.wait()
```

# Sortie standard

## Résultat

```
===Sortie standard :  
cours_CGI_Python.aux  
cours_CGI_Python.log  
cours_CGI_Python.nav  
cours_CGI_Python.out  
cours_CGI_Python.pdf  
cours_CGI_Python.snm  
cours_CGI_Python.tex
```

```
===Code de sortie :  
0
```

# Sortie et erreur standards

## Code

```
import subprocess
command = "ls *.blabla"
proc = subprocess.Popen(command, shell = True,
stdout = subprocess.PIPE, stderr = subprocess.PIPE)
# contenu de la sortie standard
# et de la sortie d'erreur standard
(out, err) = proc.communicate()
print "===Sortie standard :"
print out
print "===Sortie d'erreur standard :"
print err
# affiche le code de sortie (0 = OK)
print "===Code de sortie :"
print proc.wait()
```



# Sortie et erreur standards

## Résultat

```
===Sortie standard :
```

```
===Sortie d'erreur standard :
```

```
ls: impossible d'accéder à *.blabla: Aucun fichier ou dossier
```

```
===Code de sortie :
```

```
2
```

# Entrée, sortie et erreur standards

## Code

```
import subprocess
command = "wc -l"
data = "">sp|Q41560|HS16B_WHEAT 16.9 kDa class I heat shock protein
MSIVRRTNVFDPFADLWADPFDTFRSIVPAISGGGSETAAAFANARMDWKETPEAHVFKAD
LPGVKKEEVKVEVEDGNVLVVSGETRKEKEDKNDKWHRVERSSGKFVRRFRLLLEDAKVEE
VKAGLENGVLTVTPKAEVKKPEVKAIQISG
""
proc = subprocess.Popen(command, shell = True,
stdout = subprocess.PIPE, stderr = subprocess.PIPE,
stdin = subprocess.PIPE)
# contenu de la sortie standard
# et de la sortie d'erreur standard
(out, err) = proc.communicate(data)
print "===Entree standard :"
print data
print "===Sortie standard :"
print out
print "===Sortie d'erreur standard :"
print err
# affiche le code de sortie (0 = OK)
print "===Code de sortie :"
print proc.wait()
```

# Entrée, sortie et erreur standards

## Résultat

===Entree standard :

```
>sp|Q41560|HS16B_WHEAT 16.9 kDa class I heat shock protein  
MSIVRRRTNVFDPFADLWADPFDTFRSIVPAISGGGSETAAAFANARMDWKETPEAHVFKAD  
LPGVKKEEVKVEVEDGNVLVVSGERTKEKEDKNDKWHRVERSSGKFVRRFRLLLEDAKVEE  
VKAGLENGVLTVTVPKAEVKKPEVKAIQISG
```

===Sortie standard :

4

===Sortie d'erreur standard :

===Code de sortie :

0

# Références

## Cours de Python – Patrick Fuchs & PP

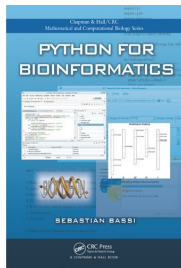
<http://www.dsimb.inserm.fr/~fuchs/python/index.html>



## Bonnes pratiques et astuces Python – David Larlet – BioloGeek

<http://www.biologeeek.com/bonnes-pratiques,conferences,django,python,traduction/bonnes-pratiques-et-astuces-python/>

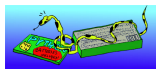
## Python for Bioinformatics – Sebastian Bassi



# Contributeurs

Ce cours est basé sur un cours original de Patrick Fuchs.

# Crédits graphiques



Frank Stajano (Wikimedia Commons)