# Une pas si courte introduction au langage de programmation Python comme alternative à Matlab pour réaliser des calculs scientifiques ou d'autres applications.

G. Becq, N. Le Bihan

02/07/2013

# Outline

# Historique

## Python

- 1989–1995 : Hollande
- 1995–1999 : USA
- 1999+ : Worldwide open source

**sources** : `http://www.wikipedia.org`, `http://www.python.org`

# Historique

## Python

- 1989–1995 : Hollande
  - Centrum voor Wiskunde en Informatica (CWI).
  - *Guido von Rossum*, fan des Monty Python, travaille sur :
    - ABC : langage de script, syntaxe et indentation.
    - Modula-3 : gestion des exceptions, orienté objet.
    - langage C, Unix.
    - OS distribué Amoeba: accès difficile en shell.
  - Créée le langage Python :
    - 1991/02 : versions 0.9.06 déposée sur un newsgroup de Usenet
    - 1995 : dépôt de la version 1.2
- 1995–1999 : USA
- 1999+ : Worldwide open source

**sources** : `http://www.wikipedia.org`, `http://www.python.org`

# Historique

## Python

- 1989–1995 : Hollande
- 1995–1999 : USA
  - Corporation for National Research Initiatives (CNRI), non profit organisation, Reston, USA.
  - Grail7 : navigateur internet utilisant Tk.
  - 1999 : projet *Computer Programming for Everybody* (CP4E) (CNRI, DARPA Defense Advanced Research Projects Agency) :
    - Python comme langage d'enseignement de la programmation.
    - Création de l'IDLE (Integrated Development Language En)
  - 1999 : Python 1.6
- 1999+ : Worldwide open source


**sources** : `http://www.wikipedia.org`, `http://www.python.org`

# Historique

## Python

- 1989–1995 : Hollande
- 1995–1999 : USA
- 1999+ : Worldwide open source
  - BeOpen.com :
    - compatibilité GPL (General Public Licence)
    - création de la branche pythonLabs
  - 2000 : Python Software Foundation
    - Python 2.1 : changement licence, dérivée de Apache Software Foundation (OO, svn, commons plutôt java) (`http://www.apache.org`).
  - 2008: Python 3.0


**sources** : `http://www.wikipedia.org`, `http://www.python.org`

# Historique

## Guido van Rossum

- Guido van Rossum :
  - 31 janvier 1956 (57 ans)
  - Développeur néerlandais
  - 1982 : M. Sc
  - Développeur ABC.
- Créateur Python : *Benevolent Dictator for Life*
  - 1991 : Python 0.9.06
  - 1999 : Grail
- 2002 : Prix pour le développement du logiciel libre 2001 décerné par la *Free Software Foundation*
- 2005–2012 : Google (python)
- 2013 : Dropbox

2006, source wikipedia

**sources** : `http://www.wikipedia.org`, `http://www.python.org`

# Spécificités

`http://www.python.org/about/`

- Fortement typé.
- Objet.
- Script, séquentiel, interprété : fichier génère du byte code.
- Comparé à Tcl, Perl, Ruby, Scheme, Java.

# Spécificités

## Exemple 1er programme

Dans le fichier "hello.py" :

```
print("Bonjour monde")
```

Exécution dans une interface système (terminal, shell) :

```
~/python/example> python hello.py
Bonjour monde
```

# Spécificités

## Exemple shell scripting

Exemple copies des fichiers '.jpeg' d'un répertoire vers un autre.

```
# -*- encode:  utf-8 -*-
import shutil, os
pathSrc = 'a'
pathDst = 'b'
fileList = os.listdir(pathSrc)
extList = ['.txt', '.tex']
for fileSrc in fileList:
    if (os.path.splitext(fileSrc)[1] in extList):
        fullfileSrc = os.path.join(pathSrc, os.path.basename(fileSrc))
        fullfileDst = os.path.join(pathDst, os.path.basename(fileSrc))
        shutil.move(fullfileSrc, fullfileDst)
```

# Utilisations

## A partir du shell

- Fichiers "*.py" contient des scripts et des définitions de fonctions. Ils sont exécutés dans le shell avec la commande "python".
  Exemple : "python hello.py"
- La commande "python" ouvre une console utilisateur (*interpreter*) avec une invite de commande (prompt) caractéristique "> > > "

```
Enthought Python Distribution -- www.enthought.com
Version: 7.3-2 (32-bit)

Python 2.7.3 |EPD 7.3-2 (32-bit)| (default, Apr 12 2012, 11:28:34)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "credits", "demo" or "enthought" for more information.
>>>
```

# Utilisations

## Utilisation de l'interpréteur

- Association d'une valeur à une variable : a = 'abc'
- Affichage de la valeur : print()
- Liste des noms, attributs, fonctions disponibles : dir()

```
>>> a = "abc"
>>> print(a)
abc
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'a']

>>> dir(a)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'_formatter_field_name_split', '_formatter_parser', 'capitalize',
'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

# Utilisations

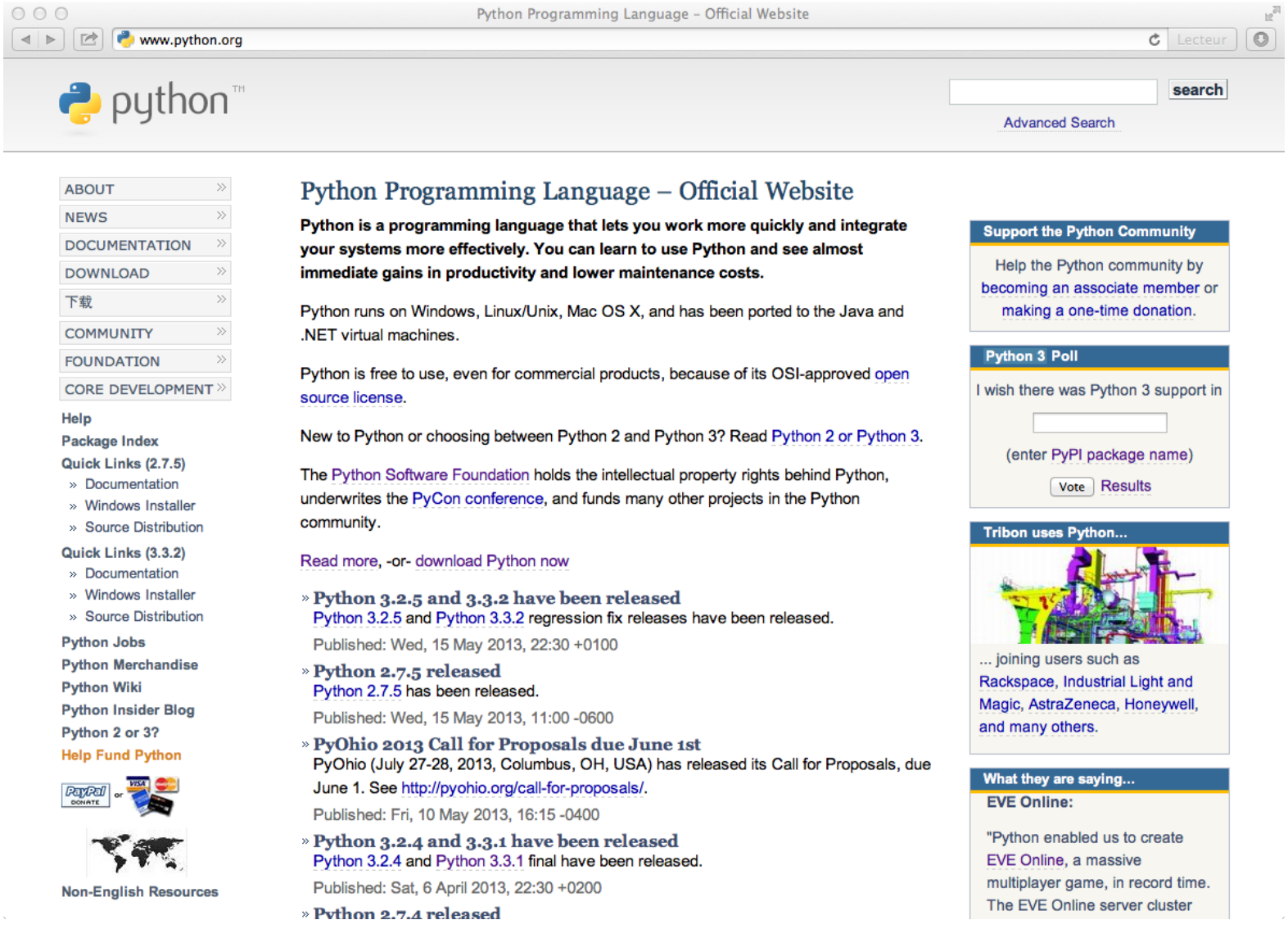
## Utilisation de l'interpréteur

- Besoin d'aide : help()

```
>>> help(a.find)

Help on built-in function find:

find(...)
    S.find(sub [,start [,end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
(END)
```

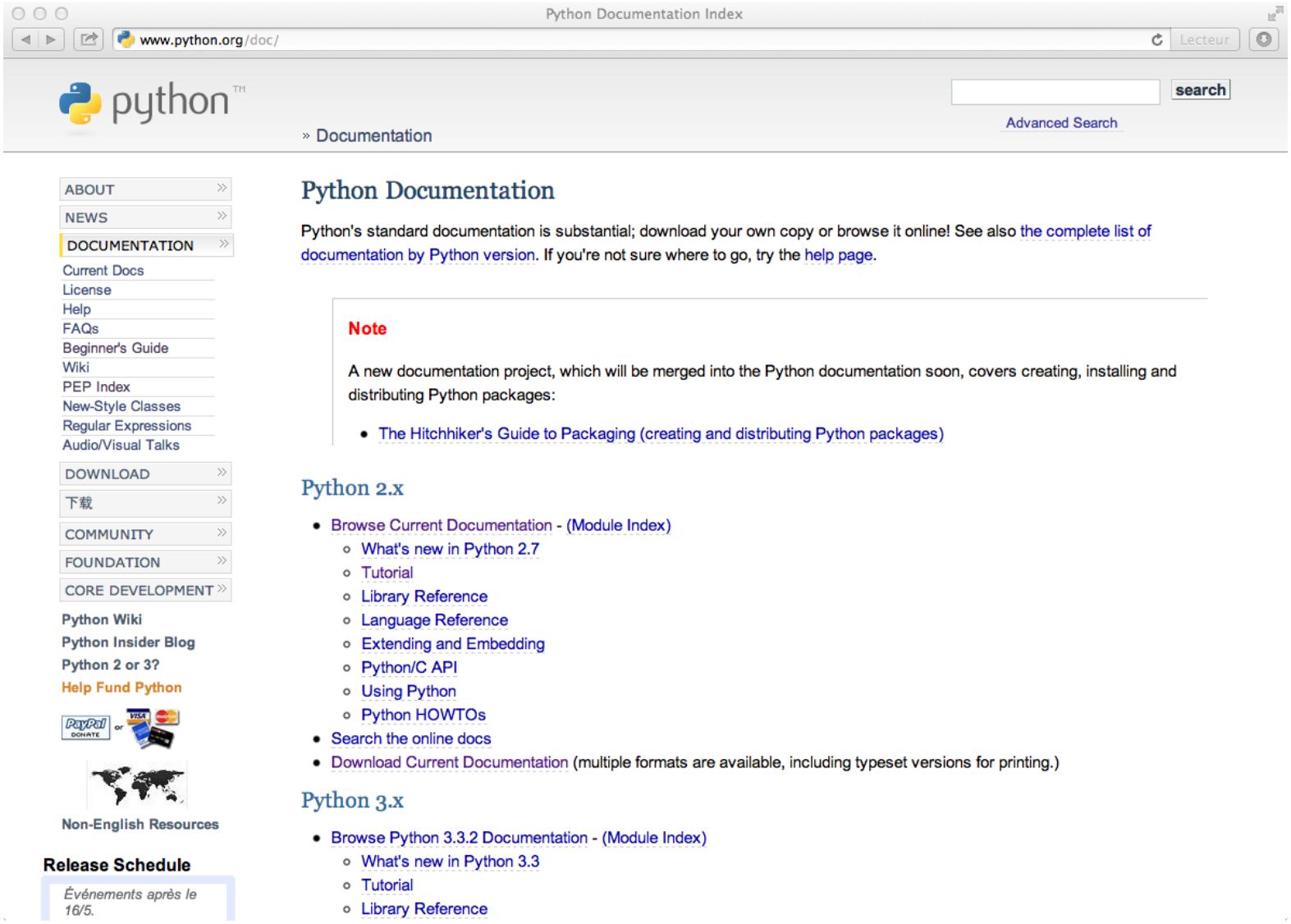
# Site Web

## www.python.org



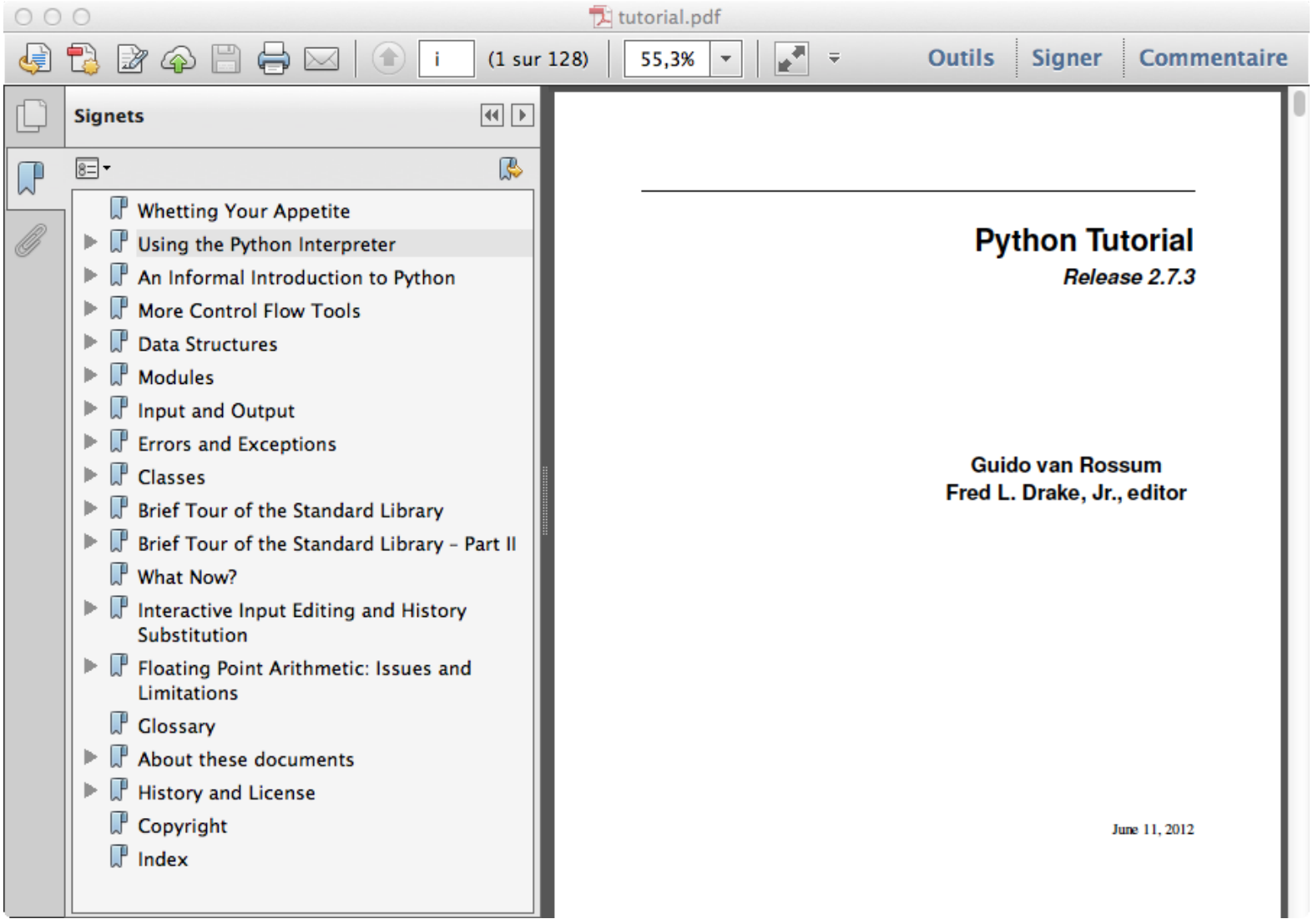
# Documentation

## Trouver la documentation



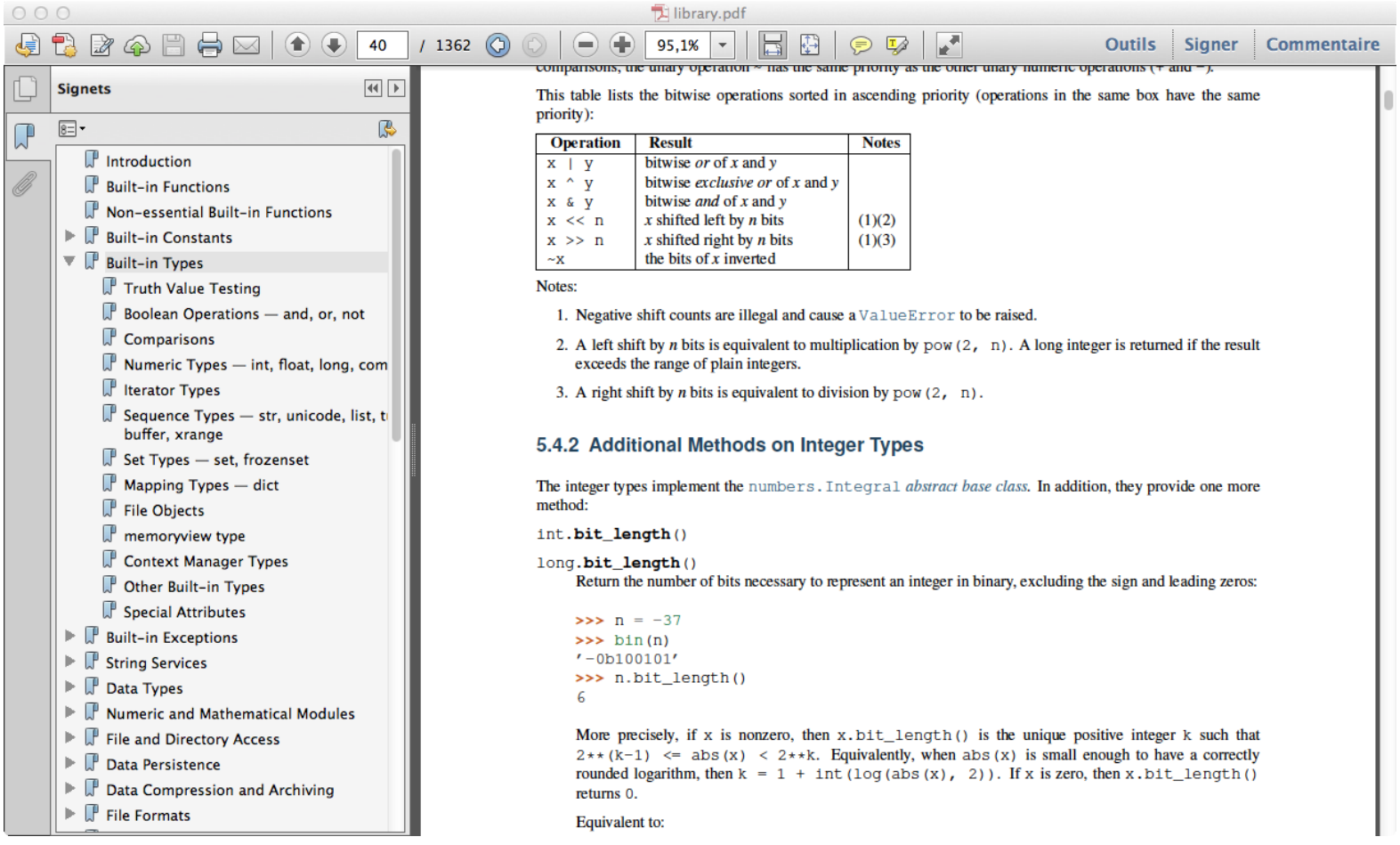
# Documentation

## Lire la documentation - Tutorial



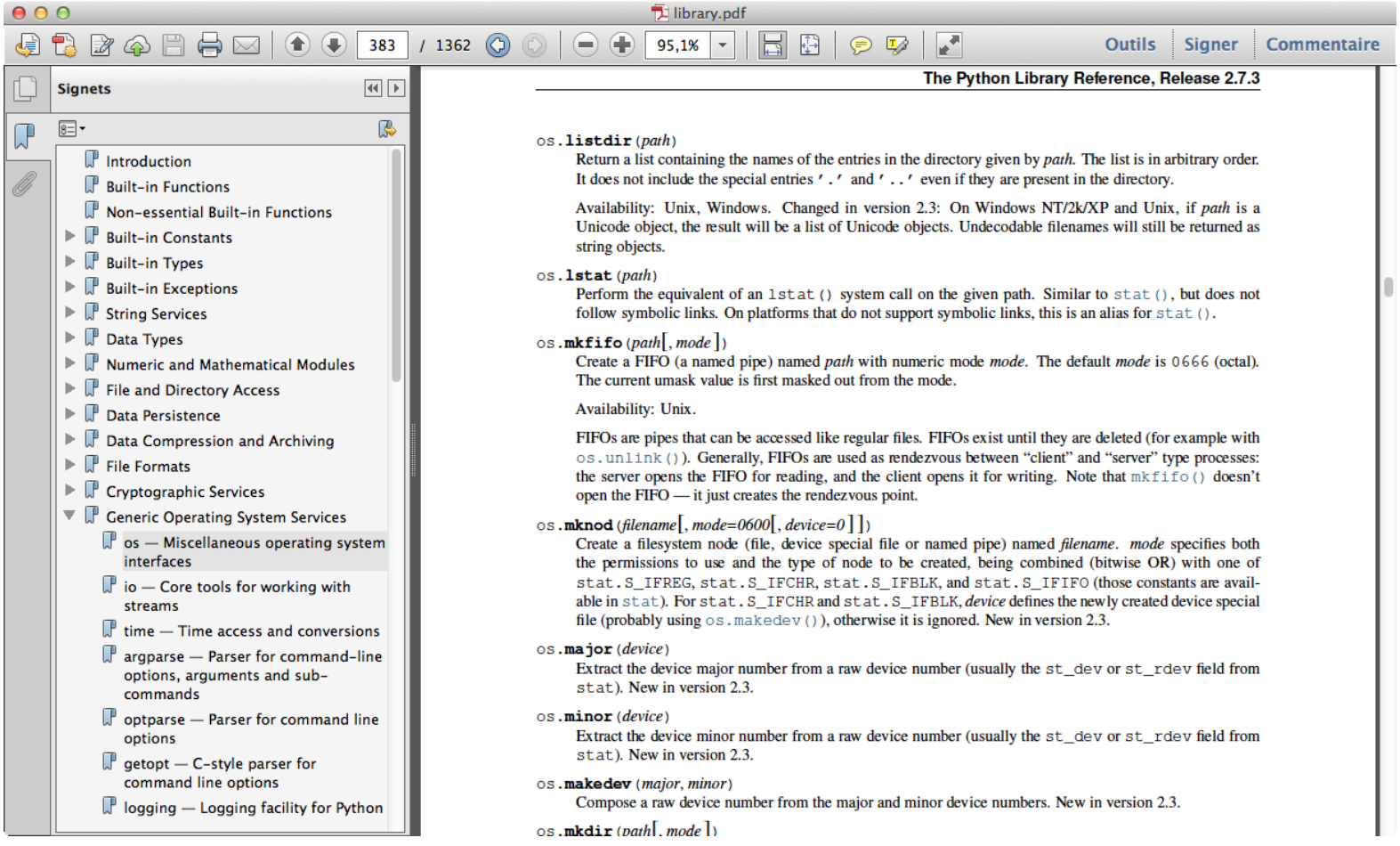
# Documentation

## Lire la documentation - Library reference



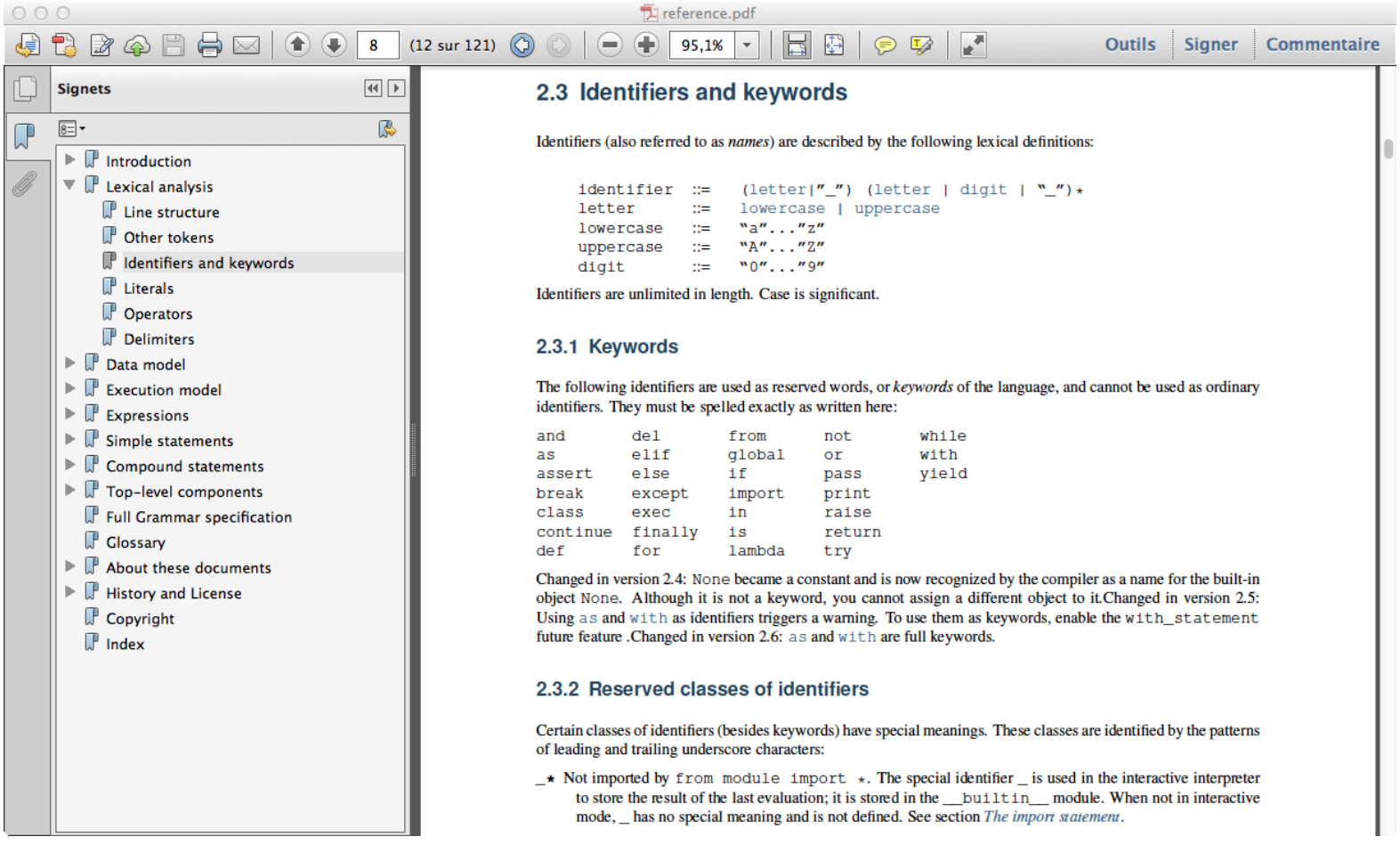
# Documentation

## Lire la documentation - Library reference



# Documentation

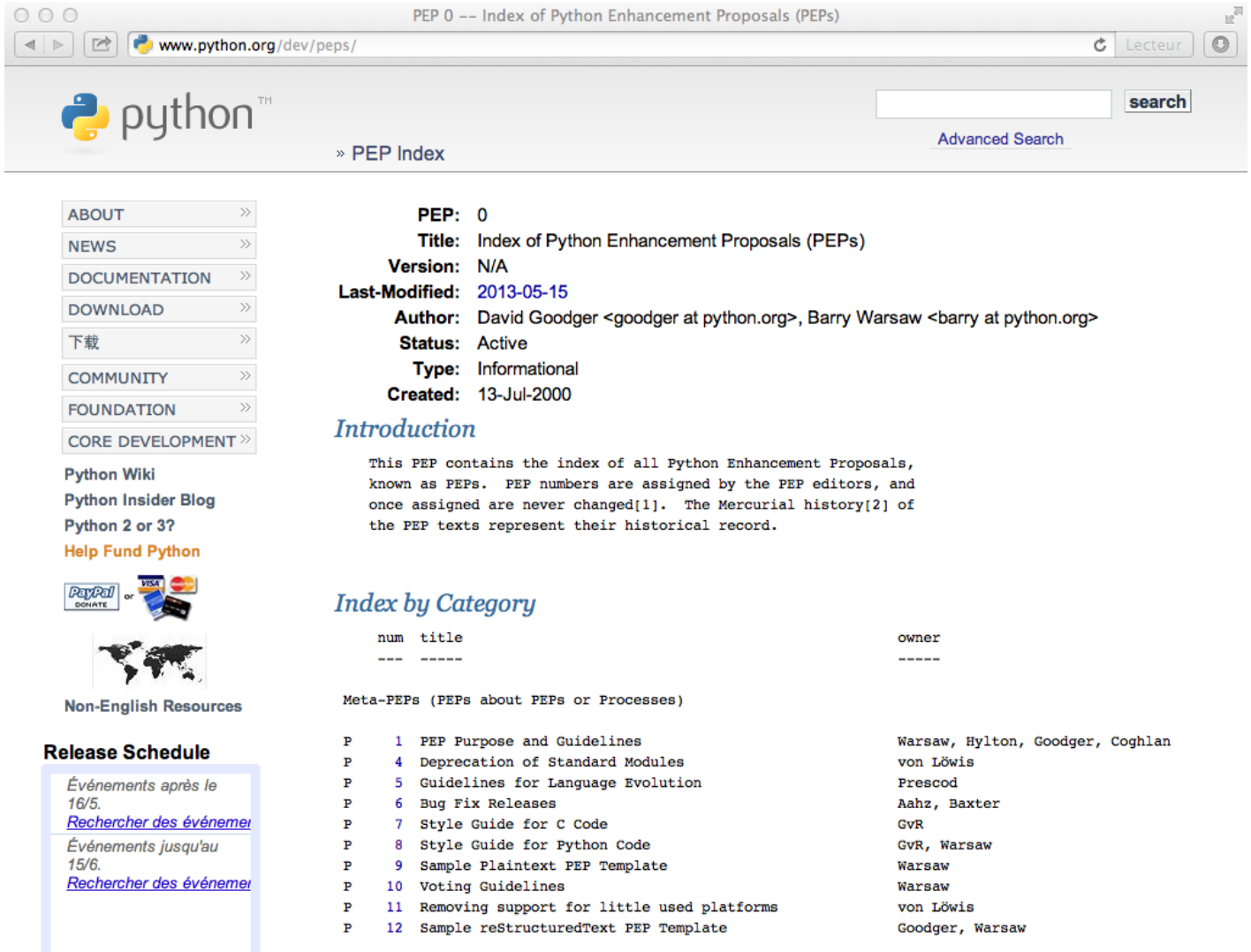
## Lire la documentation - Language reference



# Python Enhancements Proposals

## PEPs

Propositions et conseils pour l'utilisation et l'amélioration du langage.



# Installation

- Téléchargement sous `www.python.org` pour les OS courants : Windows, Linux, Mac.
- 32 bits vs 64 bits :
  - Performances vs compatibilités bibliothèques (paquets)
  - Problèmes de configuration des compilateurs (gcc).
  - Actuellement, peut générer des problèmes.
- Autres distributions : Enthought Canopy, pythonXY, WinPython, ...
- Implémentations alternatives :
  - Langage identique mais implémentation différentes permet par exemple : d'interfacer du java facilement (Jython), d'être plus performant pour les calculs (Pypy) ...

# Installation

## Python 2.7 vs 3.x

- 2.7 : figée.
- 3.x : présent et futur :
  - *better Unicode support*
  - Quelques inconsistences du langage (ex print 'a' vs print('a')).
  - Résultats de la division des entiers (1/2 : 2.x, = 0 ; 3.x, = 0.5).
  - ...
- Peut poser des problèmes :
  - Paquets portées sur 3.x ?
  - Compatibilité 64 bits ?

# Outline

# Type de base

- **Nombres** : entiers (int, long), réels (float), complex (complex), booléens (bool).
- **Séquences** : Chaînes de caractères (str), listes (list), tuples (tuple), dictionnaires (dict), ensembles (set)

# Nombres

## Entiers

- Attention à la division entière en 2.7.

```
>>> a = 1
>>> b = 2
>>> c = a / b
>>> print(c)
0
```

# Nombres

## Réels

- Utilisation du point pour passer en réels.

```
>>> a = 1.0
>>> b = 2.
>>> c = a / b
>>> print(c)
0.5
```

# Nombres

## Entiers long vs court

- Suffixe 'L' pour indiquer un passage en entier long.
- 'type()' indique le type d'une variable.

```
>>> a = 2 ** 30
>>> b = 2 ** 31
>>> a
1073741824
>>> b
2147483648L
>>> print(b)
2147483648
>>> type(a)
<type 'int'>
>>> type(b)
<type 'long'>
>>> c = 12345L
>>> type(c)
<type 'long'>
```

# Nombres

## Réels notations scientifiques

- Notation scientifique de type mantisse et exposant.

```
>>> a = 1.234e100
>>> a
1.234e+100
>>> type(a)
<type 'float'>
```

# Nombres

## Complexes

- z = x + yj (complex)
- Attention à la syntaxe.
- Génère une erreur (*exception*) de type "NameError".
- '_' variable courante (cf. *ans* matlab)

```
>>> a = 1 + 2j
>>> type(a)
<type 'complex'>
>>> a = 1 + j
Traceback (most recent call last):
  File "␣<stdin>", line 1, in <module>
NameError: name 'j' is not defined
>>> 123.456j
123.456j
>>> type(_)
<type 'complex'>
>>> a = 1 + 1j
>>> b = 2 + 3j
>>> c = a + b
>>> print(c)
(3+4j)
```

# Nombres

## Booléens

- True, False (bool)
- Attention aux résultats des opérations sur les booléens.
- + donne un int
- Opérateurs booléens spécifiques.

```
>>> a = True
>>> type(a)
<type 'bool'>
>>> b = False
>>> c = a + b
>>> print(c)
1
>>> type(c)
<type 'int'>
>>> c = a * b
>>> print(c)
0
>>> c = a & b
>>> print(c)
False
>>> c = a | b
>>> print(c)
True
```

# Nombres

## Opérateurs

- Redéfinition des opérateurs selon les types.
- Pour les règles de priorités, cf 'language reference'.
- Bitwise operations on integer, cf 'language reference'.

| | | | | |
|---|---|---|---|---|
| **entiers** | + | addition | - | substraction |
| | * | multiplication | / | division |
| | % | modulo | // | floor division |
| | ** | power | | |
| | « | shifting left | » | shifting right |
| **réels** | idem entiers hors shifting et bitwise op. | | | |
| **complexes** | idem réels | | | |
| **booléens** | &, and | and | \|, or | or |
| | ^ | xor | not | not |

# Nombres

## Opérateurs de comparaisons

**Comparaisons** $>$ , $<$ , $>=$, $<=$, $==$
$<>$, $!=$ (mieux, version C)
in, not in (sequence)
is, is not (objet)

```
>>> a = 1
>>> a in [0, 1, 2]
True
>>> b = 2
>>> a is b
False
>>> a is 1
True
>>> c = a
>>> a is c
True
>>> 0 < a <= 2
True
```

# Types de données

## Chaînes de caractères

- Texte délimité par ' ' ou " " (str)
- La chaîne de caractère a une longueur (len)

```
>>> a = 'abc'
>>> type(a)
<type 'str'>
>>> b = "def"
>>> c = a + b
>>> c
'abcdef'
>>> len(c)
6
```

# Types de données

## Chaînes de caractères

- Alternance des single/double quote.
- Caractères spéciaux à la C.

```
>>> a = "1. Bisque de pigeonneaux\n\t Prenez vos pigeonneaux apres qu'ils
seront bien nettoyez\n\t&'Troussez', faites les blanchir, \n\t& Les " +
'"empottez" avec un petit brin de fines herbes'
>>> print(a)
1. Bisque de pigeonneaux
    Prenez vos pigeonneaux apres qu'ils seront bien nettoyez
    & 'Troussez', faites les blanchir,
    & Les "empottez" avec un petit brin de fines herbes
```

# Types de données

## Chaînes de caractères

- forme mutliligne """ """ ou ''' ''' (! caractères spéciaux).
- Représentation (repr) différente de (str).

```
>>> a = """1. Bisque de pigeonneaux\n
... \tPrenez vos pigeonneaux apres qu'ils seront bien nettoyez\n
...     & 'Troussez', faites les blanchir,
...     & Les "empottez" avec un petit brun de fines herbes
... """
>>> print(a)
1. Bisque de pigeonneaux

    Prenez vos pigeonneaux apres qu'ils seront bien nettoyez

    & 'Troussez', faites les blanchir,
    & Les "empottez" avec un petit brun de fines herbes
>>> repr(a)
'\'1. Bisque de pigeonneaux\\n\\n\\t\\tPrenez vos pigeonneaux apres qu\\\'
ils seront bien nettoyez\\n\\n\\t& \\\'Troussez\\\', faites les blanchir,\
\n\\t& Les "empottez" avec un petit brun de fines herbes\\n\''
>>> str(a)
"1.␣Bisque␣de␣pigeonneaux\n\n\tPrenez␣vos␣pigeonneaux␣apres␣qu'ils␣seront␣bien␣nettoyez\
```

# Types de données

## Chaînes de caractères

- forme brute (r" " ou r' ' ou r""" """ ou r''' ''')
- forme unicode (u" " ou . . . ).

```
>>> a = r"Potage de santé\n\tLe potage de santé se fait de chapons..."
>>> print(a)
Potage de santé\n\tLe potage de santé se fait de chapons...
>>> repr(a)
"'Potage de sant\\xc3\\xa9\\\\n\\\\tLe potage de sant\\xc3\\xa9 se fait
de chapons...'"
>>> a = u"Potage de santé\n\tLe potage de santé se fait de chapons..."
>>> print(a)
Potage de santé
    Le potage de santé se fait de chapons...
>>> repr(a)
"u'Potage de sant\\xe9\\n\\tLe potage de sant\\xe9 se fait de chapons...
'"
```

# Types de données

## Listes

- Listes d'éléments ([ , ], list)
- Longueur $n$ (len)
- Accès aux élément : indice de 0 à $n-1$

```
>>> a = [1, 2, 3]
>>> type(a)
<type 'list'>
>>> len(a)
3
>>> b = ['abc', 'de', 'fghij']
>>> len(b)
3
>>> c = a + b
>>> print(c)
[1, 2, 3, 'abc', 'de', 'fghij']
>>> c[0]
1
>>> c[5]
'fghij'
```

# Types de données

## Listes

- découpage, slicing (:, i:, :j, i:j, i:j:k, slice(i,j,k))
- Indices négatifs de -1 à $-n$ peremttent de parcourir en partant par la fin.

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[:]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[3:]
[4, 5, 6, 7, 8, 9]
>>> a[:3]
[1, 2, 3]
>>> a[2:6]
[3, 4, 5, 6]
>>> a[2:6:2]
[3, 5]
>>> i = slice(2, 6, 2)
>>> a[i]
[3, 5]
>>> a[-1]
9
>>> a[-9]
1
```

# Types de données

## Listes

- Imbrications, nested.
- Attention aux erreurs d'indexation, les listes Python ne sont pas de matrices à la Matlab.

```
>>> a = [[1, 2, 3], [4, 5, 6]]
>>> a
[[1, 2, 3], [4, 5, 6]]
>>> len(a)
2
>>> a[0]
[1, 2, 3]
>>> a[0][0]
1
>>> a[0, 0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers, not tuple
```

# Types de données

## Tuples

- Couple, triplet, n-uplet, plus généralement tuple (( , ), tuple)
- Ce sont des séquences (de même que les types str, unicode, list, tuple, buffer, xrange) : même indexation que les listes.

```
>>> couple = ('papa', 'maman')
>>> type(couple)
<type 'tuple'>
>>> couple[0]
'papa'
>>> a = ('bob', 'alice')
>>> b = ('pierre', 'paul')
>>> c = a + b
>>> print(c)
('bob', 'alice', 'pierre', 'paul')
```

# Types de données

## Tuples

- Taille fixe immuable (immutable en anglais).
- Pratique pour le passage de paramètres de taille fixe.
- Affectation multiple implicite.
- On ne peut pas changer les valeurs des tuples.

```
>>> papa = 'bob'
>>> maman = 'alice'
>>> couple = (papa, maman)
>>> print(couple)
'bob', 'alice'
>>> couple = (papa, maman) = ('bob', 'alice')
>>> couple = papa, maman = 'bob', 'alice'
>>> couple[0] = 'robert'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

# Types de données

## Tuples

- Attention : couples immuables, listes muables

```
>>> address = '0001'
>>> instr = '011'
>>> code1 = (instr, address)
>>> code2 = ('100', '0010')
>>> code3 = ('110', '0011')
>>> code4 = ('010', '0001')
>>> prog = [code1, code2, code3]
>>> prog.append(code4)
>>> print(progr)
[('011', '0001'), ('100', '0010'), ('110', '0011'), ('010', '0001')]
```

# Types de données

## Tuples

- Exemple de liste de couples.

```
>>> name = 'abc'
>>> value = 123
>>> entry1 = (name, value)
>>> entry2 = ('def', 234)
>>> dictionary = [entry1, entry2]
>>> dictionary.append(('efg', 345))
>>> print(dictionary)
[('abc', 123), ('def', 234), ('efg', 345)]
```

# Types de données

## Dictionnaires

- Dictionnaires (dict)
- key (str): value

```
>>> d = {'abc': 123, 'def': 234}
>>> type(d)
<type 'dict'>
>>> print(d)
{'abc': 123, 'def': 234}
>>> d['abc']
123
>>> for key in d:
...   print((key, d[key]))
...
('abc', 123)
('def', 234)
```

# Types de données

## Ensemble

- Ensemble ({ , }, set)
- Collections d'éléments uniques non ordonnés.
- Opération ensemblistes (add, discard, union, intersect, . . . )

```
>>> E = {1, 2, 3}
>>> type(E)
<type 'set'>
>>> print(E)
set([1, 2, 3])
>>> E[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
>>> F = {1, 2, 3, 2, 1, 4, 1, -1}
>>> F
set([1, 2, 3, 4, -1])
>>> a = 1
>>> a in F
True
>>> G = E + F
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'set' and 'set'
>>> E.intersection(F)
set([1, 2, 3])
>>> E.add(-10)
>>> print(E)
set([1, 2, 3, -10])
```

# Syntaxe

## Résumé

- 'Builtin' Fonctions
- Structures de contrôles
- Fonctions
- Objets et Classes

# Syntaxe

## Fonctions prédéfinies

- Built-in Functions : fonctions de base du langage.

| | | | | |
|---|---|---|---|---|
| abs() | divmod() | input() | open() | staticmethod() |
| all() | enumerate() | int() | ord() | str() |
| any() | eval() | isinstance() | pow() | sum() |
| basestring() | execfile() | issubclass() | print() | super() |
| bin() | file() | iter() | property() | tuple() |
| bool() | filter() | len() | range() | type() |
| bytearray() | float() | list() | raw_input() | unichr() |
| callable() | format() | locals() | reduce() | unicode() |
| chr() | frozenset() | long() | reload() | vars() |
| classmethod() | getattr() | map() | repr() | xrange() |
| cmp() | globals() | max() | reversed() | zip() |
| compile() | hasattr() | memoryview() | round() | __import__() |
| complex() | hash() | min() | set() | apply() |
| delattr() | help() | next() | setattr() | buffer() |
| dict() | hex() | object() | slice() | coerce() |
| dir() | id() | oct() | sorted() | intern() |

# Syntaxe

## Fonctions prédéfinies

- Built-in Functions : fonctions de base du langage.

| | | | | |
|---|---|---|---|---|
| abs() | divmod() | input() | open() | staticmethod() |
| all() | enumerate() | int() | ord() | str() |
| any() | eval() | isinstance() | pow() | sum() |
| basestring() | execfile() | issubclass() | print() | super() |
| bin() | file() | iter() | property() | tuple() |
| bool() | filter() | len() | range() | type() |
| bytearray() | float() | list() | raw_input() | unichr() |
| callable() | format() | locals() | reduce() | unicode() |
| chr() | frozenset() | long() | reload() | vars() |
| classmethod() | getattr() | map() | repr() | xrange() |
| cmp() | globals() | max() | reversed() | zip() |
| compile() | hasattr() | memoryview() | round() | __import__() |
| complex() | hash() | min() | set() | apply() |
| delattr() | help() | next() | setattr() | buffer() |
| dict() | hex() | object() | slice() | coerce() |
| dir() | id() | oct() | sorted() | intern() |

déjà rencontrées

# Syntaxe

## Fonctions prédéfinies

- Built-in Functions : fonctions de base du langage.

| | | | | |
|---|---|---|---|---|
| abs() | divmod() | input() | open() | staticmethod() |
| all() | enumerate() | int() | ord() | str() |
| any() | eval() | isinstance() | pow() | sum() |
| basestring() | execfile() | issubclass() | print() | super() |
| bin() | file() | iter() | property() | tuple() |
| bool() | filter() | len() | range() | type() |
| bytearray() | float() | list() | raw_input() | unichr() |
| callable() | format() | locals() | reduce() | unicode() |
| chr() | frozenset() | long() | reload() | vars() |
| classmethod() | getattr() | map() | repr() | xrange() |
| cmp() | globals() | max() | reversed() | zip() |
| compile() | hasattr() | memoryview() | round() | __import__() |
| complex() | hash() | min() | set() | apply() |
| delattr() | help() | next() | setattr() | buffer() |
| dict() | hex() | object() | slice() | coerce() |
| dir() | id() | oct() | sorted() | intern() |

abordées tôt ou tard

# Syntaxe

## Structure de contrôle

- **blocs conditionnels** : if else
- **boucles conditionnelles** : while
- **boucles** : for in
- **gestion d'exceptions** : try except

# Syntaxe

## Structure de contrôle - if else

- blocs conditionnels : if else
- ':' et indentation suivis d'un bloc d'instruction.

```
>>> a = 1
>>> if (a == 1):
...     print("a␣vaut␣1")
... else:
...     print("a␣ne␣vaut␣pas␣1")
...
a vaut 1
```

# Syntaxe

## Indentation

- Pour l'indentation dans un fichier, il est conseillé d'utiliser 4 espaces plutôt qu'une tabulation.
- Possibilité de régler ce paramètres dans les éditeurs de texte : tabulation souple émulée avec des espaces ('soft tab' with 4 spaces vs 'hard tab').

```
if (a == 1):
    print("a vaut 1")
else:
    print("a ne vaut pas 1")
```

# Syntaxe

## Structure de contrôle - if elif else

- blocs conditionnels : if elif else

```
>>> a = 2
>>> if (a == 1):
...     print("a␣vaut␣1")
... elif a == 2:
...     print("a␣vaut␣2")
... elif a == 3:
...     print("a␣vaut␣3")
... else:
...     print("a␣ne␣vaut␣ni␣1␣ni␣2␣ni␣3")
...
a vaut 2
```

cf matlab (if elseif else end, switch case otherwise end)

# Syntaxe

## Structure de contrôle - if elif else

- blocs conditionnels : if elif else

```
>>> a = "green"
>>> if (a is "green"):
...     print("00FF00")
... elif (a == "red"):
...     print("FF0000")
... elif (a is "blue"):
...     print("0000FF")
... else:
...     print("Unknown␣color")
...
00FF00
```

cf matlab (if elseif else end, switch case otherwise end)

# Syntaxe

## Structure de contrôle - while

- boucles conditionnelles : while

```
>>> question = "Voulez-vous␣continuer␣?␣(o,␣oui,␣O,␣OUI)␣"
>>> cond = True
>>> reponseOK = {"o", "O", "oui", "OUI"}
>>> i = 0
>>> while cond:
...     i += 1
...     print(str(i) + "␣fois")
...     answer = input(question)
...     if (answer in reponseOK):
...         cond = True
...     else:
...         cond = False
...
1 fois
Voulez-vous continuer ? (o, oui, O, OUI) 'o'
2 fois
Voulez-vous continuer ? (o, oui, O, OUI) 'oui'
3 fois
Voulez-vous continuer ? (o, oui, O, OUI) 'non'
>>>
```

# Syntaxe

## Structure de contrôle - for

- boucles : for in :
- Dans 'language reference' :
  for_stmt ::= "for" target_list "in" expression_list ":" suite
  ["else" ":" suite]

```
>>> for i in [0, 1, 2, 3]:
...     print(i)
...
0
1
2
3
```

# Syntaxe

## Structure de contrôle - for

- [0, 1, 2, 3] : range(i,j,k)

```
>>> for i in range(4):
...     print(i)
...
0
1
2
3
```

```
>>> x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[range(1, 5, 2)]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers, not list
>>> x[slice(1, 5, 2)]
[1, 3]
>>> range(1, 5, 2)
[1, 3]
>>> slice(1, 5, 2)
slice(1, 5, 2)
```

# Syntaxe

## Structure de contrôle - for

- boucles : for

```
>>> for (i, j) in [(1, 2), (2, 3), (3, 4)]:
...     print((i,j))
...
(1, 2)
(2, 3)
(3, 4)
```

```
>>> for [i, j] in [[1, 2], [2, 3], [3, 4]]:
...     print([i, j])
...
[1, 2]
[2, 3]
[3, 4]
```

# Syntaxe

## Structure de contrôle - break, continue

- possibilité de break et continue.

```
>>> for i in range(10):
...     if (i % 2 == 0):
...         continue
...     if (i == 9):
...         break
...     print(i)
...
1
3
5
7
```

# Syntaxe

## Structure de contrôle - try except

- Gestion des erreurs ou exceptions : try except

```
>>> a = "abc"
>>> a += 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

```
>>> a = 'abc'
>>> try:
...     a += 1
... except :
...     print("problem" )
...
problem
```

# Syntaxe

## Structure de contrôle - try except else finally

- Gestion des erreurs ou exceptions : try except else finally

```
>>> a = 'abc'
>>> try:
...     a += 1
... except IOError as e :
...     print("problem entrées sorties : " + str(e))
... except TypeError as e :
...     print("problem de types : " + str(e))
... finally:
...     print("Avec ou sans erreurs , on continue")
...
problem de types : cannot concatenate 'str' and 'int' objects
Avec ou sans erreurs , on continue
```

```
>>> a = 1
>>> try:
...     a += 1
... except IOError as e :
...     print("problem entrées sorties : " + str(e))
... except TypeError as e :
...     print("problem de types : " + str(e))
... else:
...     print("a = " + str(a))
... finally:
...     print("Avec ou sans erreurs , on continue")
...
a = 2
Avec ou sans erreurs , on continue
```

# Syntaxe

## Fonctions - Définition de la fonction

- définition d'une fonction : "def" funcname "(" [parameter_list] ")" ":" suite
- indentation nécessaire pour le bloc d'instructions.

```
>>> def sayHello():
...     print("Hello")
...
>>> sayHello()
Hello
```

# Syntaxe

## Fonctions - passage de paramètres obligatoires

- Les arguments (dit args) sont définis entre parenthèse.
- args : obligatoires et ordonnés

```
>>> def sayHello(titre, prenom, nom):
...     print("Hello␣" + str(titre) + "␣" + str(prenom) + "␣" + str(nom))
...
>>> sayHello("détective privé de classe R", "John", "Difool")
Hello détective privé de classe R John Difool
```

```
>>> sayHello()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sayHello() takes exactly 3 arguments (0 given)
```

# Syntaxe

## Fonctions - passage de paramètres optionnels

- arguments mots clés (keywords arguments dits kwargs) non obligatoires.
- args avant les kwargs
- kwargs : optionnels et non ordonnés

```
>>> def sayHello(titre, prenom="", nom=""):
...     print("Hello␣" + str(titre) + "␣" + str(prenom) + "␣" + str(nom))
...
>>> sayHello("old")
Hello old
>>> sayHello("old", prenom="Nick")
Hello old Nick
>>> sayHello("old", "Tom", "Bombadil")
Hello old Tom Bombadil
>>> sayHello("old", nom="Bombadil", prenom="Tom")
Hello old Tom Bombadil
>>> sayHello(nom="Bombadil")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sayHello() takes at least 1 argument (1 given)
>>> sayHello("old", nom="Bombadil", "Tom")
  File "<stdin>", line 1
SyntaxError: non-keyword arg after keyword arg
```

# Syntaxe

## Fonctions - renvoi de valeurs

- renvoi de valeurs : return

```
>>> def fun(x):
...     y = 2 * x
...     return y
...
>>> fun(3)
6
```

```
>>> def fun(x):
...     y1 = x
...     y2 = 2 * x
...     y3 = 3 * x
...     name = "1, 2, 3 * " + str(x) + " = "
...     return name, y1, y2, y3 # the tuple (name, y1, y2, y3)
...
>>> fun(3)
('1, 2, 3 * 3 = ', 3, 6, 9)
>>> (name, y1, y2, y3) = fun(3)
>>> name, y1, y2, y3 = fun(3)
```

# Syntaxe

## Fonctions - autres spécificités

- bloc vide : pass
- None values
- variables de fonctions

```
>>> def fonctionQuiNeFaitRienEtQuiAUnNomHorribleMaisDescriptif(x):
...     pass
...
>>> def fonctionQuiNeFaitRienEtQuiAUnNomHorribleMaisDescriptif(x):
...     return
...
>>> def fonctionQuiNeFaitRienEtQuiAUnNomHorribleMaisDescriptif(x):
...     return None
...
>>> a = fonctionQuiNeFaitRienEtQuiAUnNomHorribleMaisDescriptif(3)
>>> a == None
True
>>> pfff = fonctionQuiNeFaitRienEtQuiAUnNomHorribleMaisDescriptif
>>> pfff(1)
>>> pfff("abc")
>>> pfff
<function fonctionQuiNeFaitRienEtQuiAUnNomHorribleMaisDescriptif at 0x2d49b0>
```

# Syntaxe

## Objets - Notion de programmation orientée objet

- Un objet possède :
  - des propriétés ou attributs, variables spécifiques à l'objet.
  - des méthodes, fonctions qui permettent de communiquer entre objets ou avec l'extérieur.
- Un objet appartient à une classe: *Un objet est une instance de classe.*
- Les propriétés et méthodes sont définies dans la classe.

```
3 canards : riri, fifi, loulou

classe Canard :
propriétés, attributs : 2 pattes, 2 ailes, état d'activité
méthodes : manger, chanter, ne rien faire.

Canard :
 à 2 pattes
 à 2 ailes
 status par défaut : ne fais rien

 mange(aliments)
  vérification aliments : herbivore, carnivore occasionnel

 chante()
  génère un son qui fait "couin␣couin"

 ne fait rien()
```

# Syntaxe

## Objets : définition en Python

- Définition d'une nouvelle : "class" classname ":" suite
- propriétés : variables avec une valeur par défaut.
- méthodes : fonction avec le premier argument l'objet lui même par convention 'self'

```
>>> class Canard:
...     pattes = 2
...     ailes = 2
...     status = "en attente"
...     def mange(self, aliment):
...         self.status = "mange"
...         if aliment in {"grain", "feuille", "fleur", "tige", "racine"}:
...             print("miam")
...         elif aliment in {"ver", "insecte", "friture"}:
...             print("j'avais faim")
...         else :
...             print("non merci")
...     def chante(self):
...         self.status = "chante"
...         print("couin couin")
...     def espere(self):
...         self.status = "en attente"
...         return 0
...
```

# Syntaxe

## Objets : définition en Python

- `__init__` : création lors de l'instanciation. Il est conseillé d'initialiser les attributs dans ce bloc.

```
>>> class Canard:
...     def __init__(self):
...         self.pattes = 2
...         self.ailes = 2
...         self.status = "en attente"
...
```

# Syntaxe

## Objets : instanciation et usage

- instanciation : objet = classname()
- propriétés obtenues par : objet.property (cf structure en C)
- méthode obtenues par : objet.method(argument)

```
>>> riri = Canard()
>>> riri
<__main__.Canard instance at 0x2c8760>
>>> fifi = Canard()
>>> loulou = Canard()
>>> riri.ailes
2
>>> riri.status
en attente
>>> riri.chante()
couin couin
>>> loulou.mange("grain")
miam
>>> riri.status, loulou.status, fifi.status
('chante', 'mange', 'en attente')
>>> fifi.espere()
0
>>> dir(riri)
['__doc__', '__module__', 'ailes', 'chante', 'espere', 'mange', 'pattes',
'status']
```

# Syntaxe

## Objets : exemple avec des nombres

- En python toutes les variables sont des objets.

```
>>> a = 123
>>> dir(a)
['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__',
'__delattr__', '__div__', '__divmod__', '__doc__', '__float__', '__floordiv__',
'__format__', '__getattribute__', '__getnewargs__', '__hash__', '__hex__',
'__index__', '__init__', '__int__', '__invert__', '__long__', '__lshift__',
'__mod__', '__mul__', '__neg__', '__new__', '__nonzero__', '__oct__', '__or__',
'__pos__', '__pow__', '__radd__', '__rand__', '__rdiv__', '__rdivmod__',
'__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__',
'__rmod__', '__rmul__', '__ror__', '__rpow__', '__rrshift__', '__rshift__',
'__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
'__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__',
'bit_length', 'conjugate', 'denominator', 'imag', 'numerator', 'real']
>>> type(a)
<type 'int'>
>>> a.real
123
>>> a.bit_length()
7
```

# Syntaxe

## Objets : exemple avec les listes et les tuples

```
>>> a = [1, 2, 3, 1, 2]
>>> dir(a)
[..., 'append', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
>>> type(a)
<type 'list'>
>>> a.append(3)
>>> a
[1, 2, 3, 1, 2, 3]
>>> a.pop()
>>> a
[1, 2, 3, 1, 2]
```

```
>>> a = (1, 2, 3, 1, 2)
>>> dir(a)
[..., 'count', 'index']
>>> type(a)
<type 'tuple'>
>>> a.count(1)
2
>>> a.index(1)
0
```

# Syntaxe

## Objets : exemple avec les chaînes de caractères

```
>>> a = "abc.def"
>>> dir(a)
[..., 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit',
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
>>> a.upper()
ABC.DEF
>>> a.index('.')
3
>>> (head, sep, tail) = a.partition('.')
('abc', '.', 'def')
>>> (head, sep, tail) = a.partition('.')
>>> head
'abc'
```

# Exemple

## Fichiers

- un fichier est considéré comme un objet de classe 'file'.

```
>>> f = file("cuisinierFrancois.txt", 'r')
>>> f
<open file 'cuisinierFrancois.txt', mode 'r' at 0x2a2390>
>>> dir(f)
dir(f)
[..., 'close', 'closed', 'encoding', 'errors', 'fileno', 'flush',
'isatty', 'mode', 'name', 'newlines', 'next', 'read', 'readinto',
'readline', 'readlines', 'seek', 'softspace', 'tell', 'truncate',
'write', 'writelines', 'xreadlines']
>>> f.readlines()
['LE␣CUISINIER␣FRANCOIS,␣\n', '\n', 'ENSEIGNANT␣LA␣MANIERE\n', 'de
bien␣apprester␣&␣assaisonner\n', 'toutes␣sortes␣de␣Viandes␣grasses
\n', '&␣maigres,␣Legumes,␣Patisseries,␣\n', '&␣autres␣mets␣qui␣se
servent␣tant\n', 'sur␣les␣Tables␣des␣Grands␣que␣des\n', 'particuli
ers.␣\n', '\n', '\n']
>>> f.seek(0)
>>> for line in f:
...     print(line)
...
LE CUISINIER FRANCOIS,

ENSEIGNANT LA MANIERE
de bien apprester & assaisonner
toutes sortes de Viandes grasses
& maigres, Legumes, Patisseries,
& autres mets qui se servent tant
sur les Tables des Grands que des
particuliers.
```

# Module

## Exécution dans le shell

- Enregistrement du code dans un fichier module1.py
- Contient des définitions des classes ou de fonctions, des scripts, une documentation (docstring).
- Exécution du code dans le shell : python module1.py

```
def f1(x):
    y = 10 * x
    return y
def f2(x):
    y = 100 * x
    return y
x = 3
y1 = f1(x)
y2 = f2(x)
print("(x, y1, y2) : " + str((x, y1, y2)))
```

```
~/python/example/module> python module1.py
(x, y1, y2) : (3, 30, 300)
```

# Module

## Importation de code : import

- Importation du code dans la console ou dans un autre module : import
- Variante en modifiant l'espace de nom (namespace) : import ... as fun

```
import module1
x = 4
y1 = module1.f1(x)
y2 = module1.f2(x)
print("module -> (x, y1, y2) : " + (x, y1, y2))
```

```
import moduleQuiAUnNomBeaucoupTropLong as myFun
x = 4
y1 = myFun.f1(x)
y2 = myFun.f2(x)
print("module -> (x, y1, y2) : " + (x, y1, y2))
```

```
(x, y1, y2) : (3, 30, 300)
module -> (x, y1, y2) : (4, 40, 400)
```

# Module

## Importation de code : import

- importation spécifique d'une ou plusieurs fonctions : from ... import (f1, f2) ; from ... import (f1 as fun1, f2 as fun2) ; from ... import f1 as fun1, f2 as fun2
- from ... import *

```
>>> from module1 import f1
>>> f1(5)
50
```

```
>>> from module1 import f1 as fois10
>>> fois10(5)
50
```

```
>>> from module1 import *
>>> dir()
[..., 'f1', 'f2', 'x', 'y1', 'y2']
```

```
>>> import module1
>>> dir()
[..., 'module1']
>>> type(module1)
<type 'module'>
>>> dir(module1)
[..., 'f1', 'f2', 'x', 'y1', 'y2']
```

# Module

## Espace : __name__

- attribut système : '__name__'

```
print("Affichage en import ou en exécution")
print("__name__ : " + str(__name__))

if ("__name__" == "__main__"):
    print("Que si executé de l'extérieur")
```

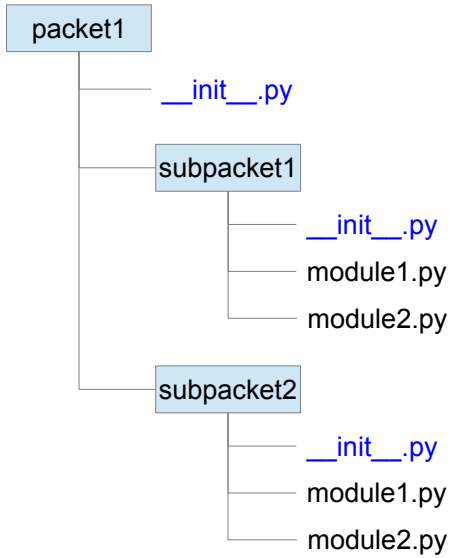
```
>>> import moduleMain
Affichage en import ou en exécution
__name__ : moduleMain
```

```
~/python/example/module> python moduleMain.py
Affichage en import ou en exécution
__name__ : __main__
```

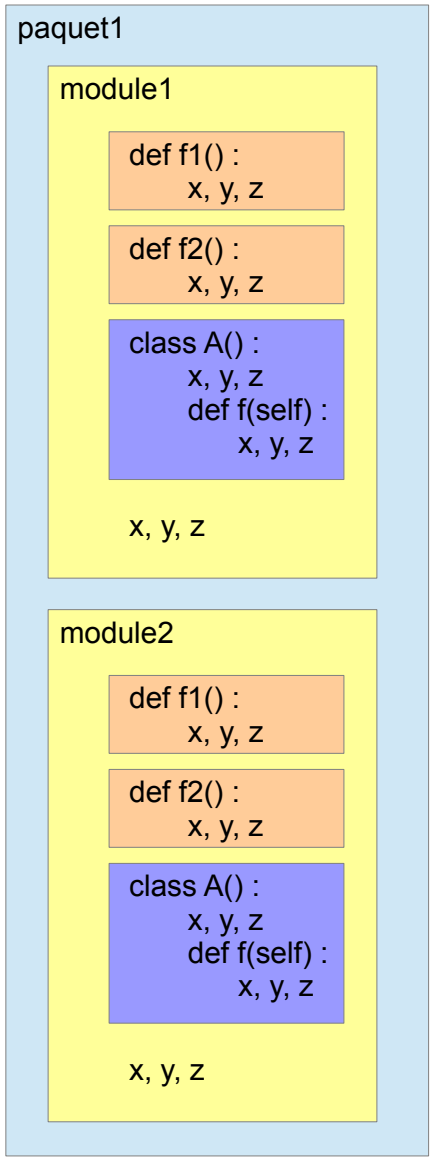
# Paquet

- Un dossier contenant :
  - des modules
  - un fichier '__init__.py', vide ou non.
- exemple : packet1/subpacket1/module1.py ; packet1/subpacket2/module2.py
- possibilité de faire des sous-paquet.

```
>>> from packet1.subpacket1 import module1 as mod1
```



# Portée




- paquet, module, def, class : chaque niveau à sa portée (scope).
- Attention à l'espace des noms (namespace) : par exemple, f1 est disponible dans module1 et module2.

```
>>> import paquet1
>>> x = 10
>>> a1 = paquet1.module1.A
>>> a2 = paquet1.module1.A
>>> a2.f()
>>> resF1 = paquet1.module1.f1(x)
>>> resF2 = paquet1.module2.f2(x)
>>> from paquet1.module1 import * # A EVITER
>>> resF1 = f1(x)
```

# Standard library

- Un ensemble de paquets et modules sont déjà livrés avec Python (Battery included) : math, os, sys, datetime ...
- Avant de recoder quelque chose, vérifier si cela existe dans la documentation : 'The Python Library Reference'.

# Exemple

## Math

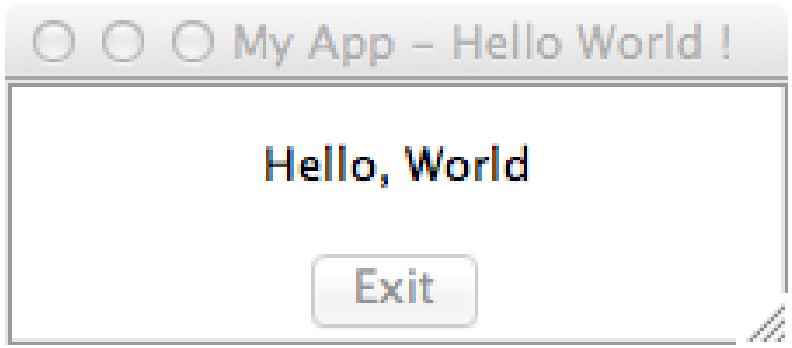
```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>> math.cos(math.pi/3)
0.5000000000000001
```

# Exemple

## Graphical User Interface / GUI

- interface graphique pour Tk (Tkinter), GTK (PyGTK), wxWidgets (wxPython), Qt (PyQt).

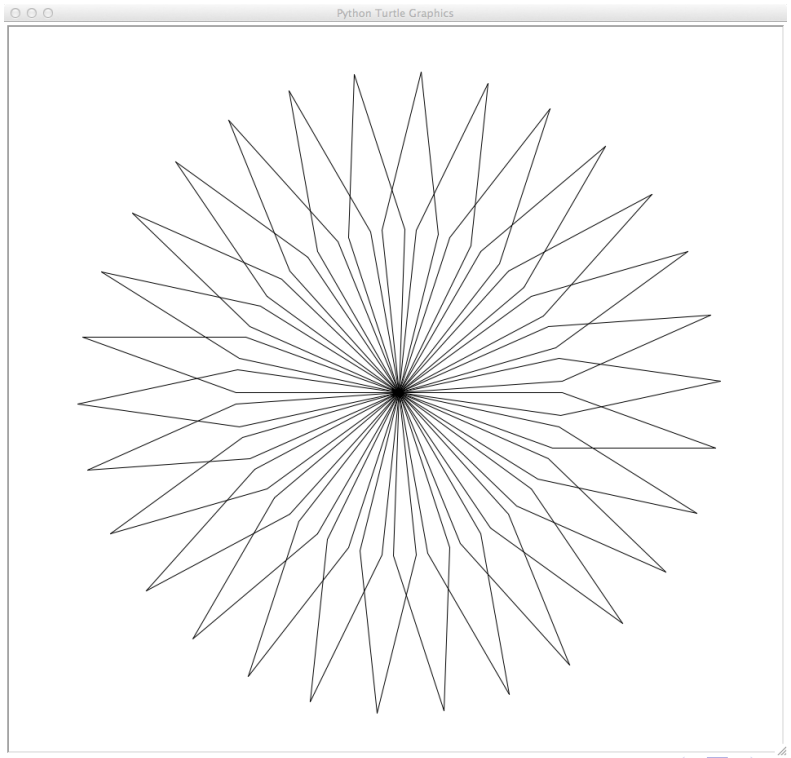
```
>>> import Tkinter
>>> from Tkconstants import *
>>> tk = Tkinter.Tk()
>>> tk.title("My App - Hello World !")
>>> frame = Tkinter.Frame(tk, relief=RIDGE, borderwidth=2)
>>> frame.pack(fill=BOTH,expand=1)
>>> label = Tkinter.Label(frame, text="Hello, World")
>>> label.pack(fill=X, expand=1)
>>> button = Tkinter.Button(frame,text="Exit",command=tk.destroy)
>>> button.pack(side=BOTTOM)
>>> tk.mainloop()
```



# Exemple

## Graphical User Interface / GUI

```
>>> import turtle
>>> for i in range(30):
...     turtle.forward(200)
...     turtle.rt(20)
...     turtle.forward(200)
...     turtle.rt(160)
...     turtle.forward(200)
...     turtle.rt(20)
...     turtle.forward(200)
...     turtle.rt(172)
...
```



# Exemple

## Base de données

```
>>> import sqlite3
>>> db = sqlite3.connect("myDB.sq3")
>>> cur = db.cursor()
>>> cur = cur.execute("CREATE␣TABLE␣membres
␣␣␣␣(nom␣TEXT,␣prenom␣TEXT,␣institut␣TEXT)")
>>> cur = cur.execute("INSERT␣INTO␣membres(nom,prenom,institut)
␣␣␣␣VALUES('Michel','Olivier','INPG')")
>>> cur = cur.execute("INSERT␣INTO␣membres(nom,prenom,institut)
␣␣␣␣VALUES('Brossier','Jean-Marc','UJF')")
>>> cur = cur.execute("INSERT␣INTO␣membres(nom,prenom,institut)
␣␣␣␣VALUES('Amblard','Pierre-Olivier','CNRS')")
>>> db.commit()
>>> cur = cur.execute("SELECT␣*␣FROM␣membres␣WHERE␣institut␣=␣'CNRS'")
>>> list(cur)
[(u'Amblard', u'Pierre-Olivier', u'CNRS')]
>>> db.close()
```

# Documentation

## docstrings

- Commentaires : ligne qui commence par #
- Texte de documentation (Docstring): *Une chaîne de caractère qui apparaît comme première expression dans un module, une fonction ou une classe*.
- """ """ recommandé (see PEP257 : `http://www.python.org/dev/peps/pep-0257/`)

```
def times(x, y):
    """
    Multiply x by y

    Compute z = x * y

    Input parameters
    x and y, any numbers

    Output parameters
    z

    Copyright 2013 G. Becq, Gipsa-lab, UMR 5216, CNRS.
    """
    return x * y
```

# Documentation

## docstrings – exemple

- Première ligne est un résumé suivi d'une ligne vide.
- Première ligne vide et indentation seront effacées.

```
# -*- encoding:utf-8 -*-
"""
Module pour tester les docstrings.

Contient deux fonctions et une classe.
Ne fait rien.

Copyright 2013 G. Becq, Gipsa-lab, UMR 5216, CNRS.
"""

def fun1():
    """
    fun1 c'est bien.

    Fonction sans parametres
    Ne fait rien.
    """
    print("blah blah")
    """
    Ce texte ne sera pas visible
    """
    print("blah blah")
    "Ce texte non plus, les declarations vides non plus"
    1
    print("blah blah")
    # Ca c'est un vrai commentaire delimite par un hash
    return
```

# Documentation

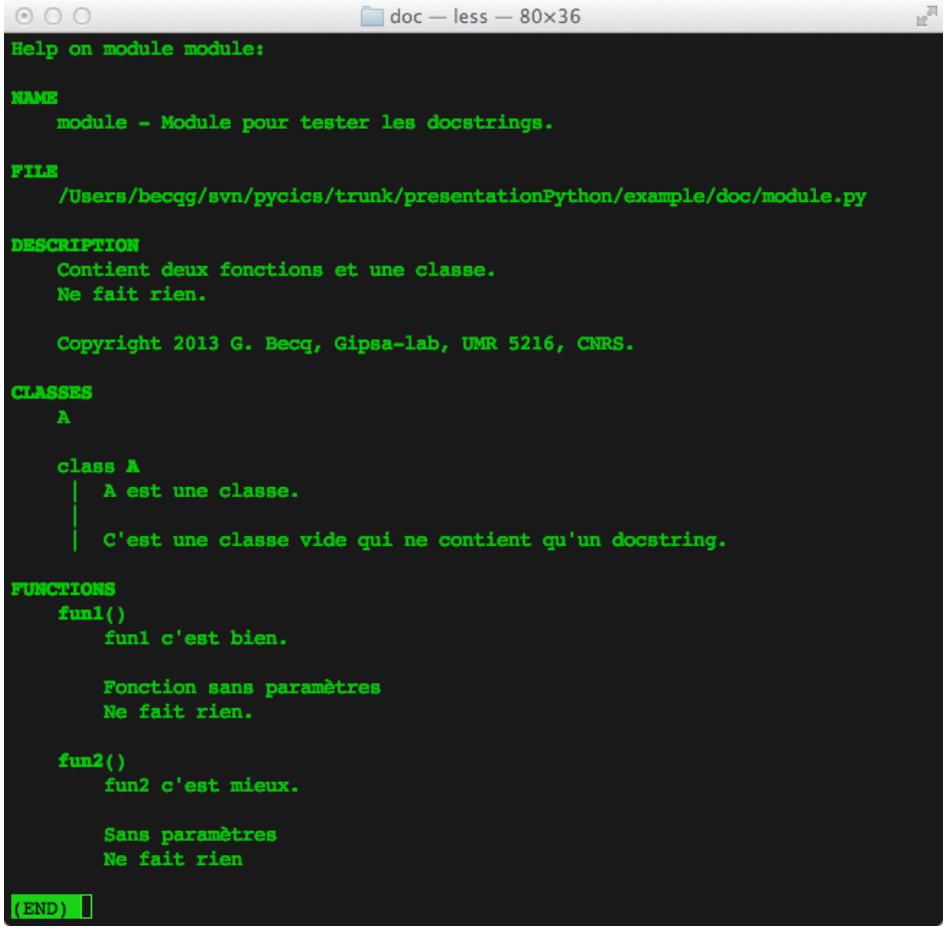
## docstrings – exemple

```
def fun2():
    """
    fun2 c'est mieux.

    Sans parametres
    Ne fait rien
    """
    return

class A():
    """
    A est une classe.

    C'est une classe vide qui ne contient qu'un docstring.
    """
```

# Documentation

## help

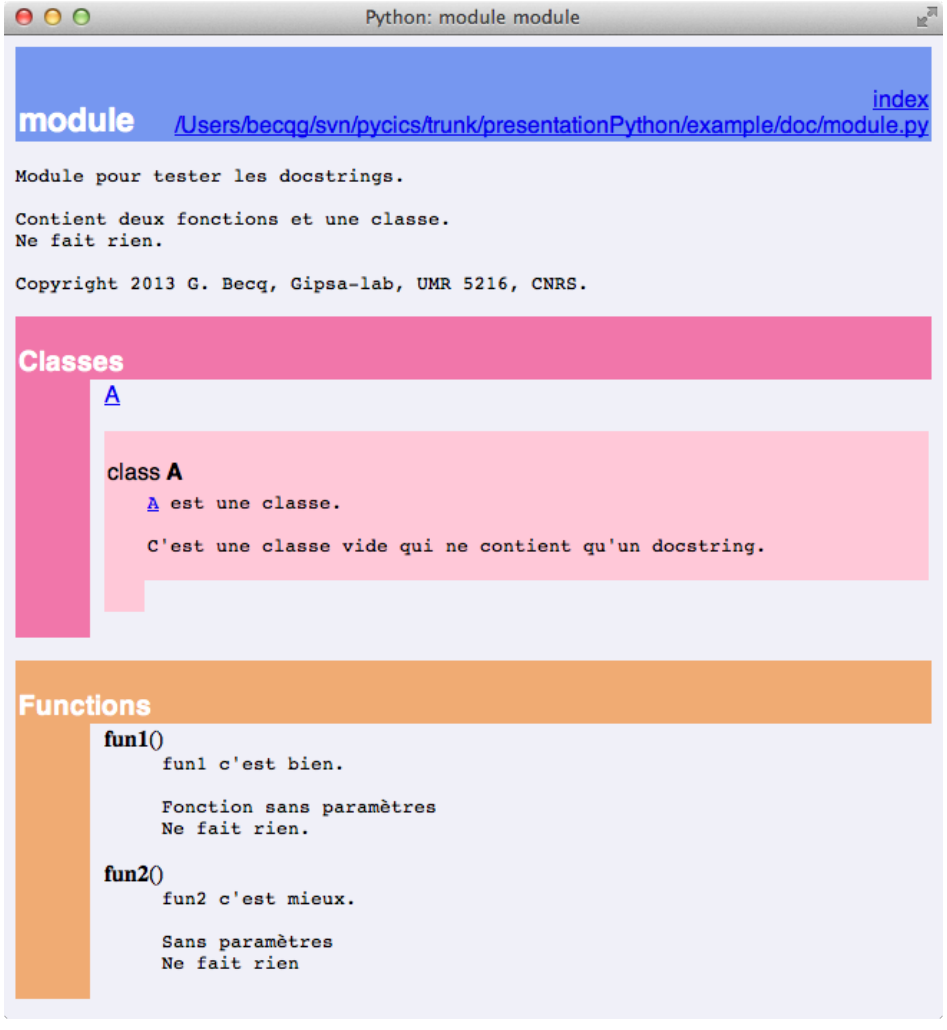
- Génération automatique de documentation : help(nomDuModule)



# Documentation

## pydoc

- Dans le shell 'pydoc -g', lance un serveur de documentation accessible via un navigateur.
- 'pydoc' génère aussi d'autres sorties.



# Packaging

- Repose sur *distutils* de la bibliothèque standard.
- Créer un fichier 'setup.py' qui contient des metadonnées.

```
from distutils.core import setup
setup(name="distrib1",
    description="une distribution de démos",
    version="1.0",
    author="Guillaume Becq",
    author_email="guillaume.becq@gipsa-lab.grenoble-inp.fr",
    url="http://example.iana.org",
    py_modules=["module1", "module2"],
    )
```

# Packaging

- Distribution : fichier compressé de type 'zip' ou '.tar.gz'
- Exécuter dans un terminal "python setup.py sdist"
- Génère :
  - un dossier 'dist' : contenant la distribution.
  - un fichier MANIFEST : liste des fichiers inclus dans le paquet.
- possibilité d'utiliser MANIFEST.in pour dire quels fichiers à inclure.
- Faire un README.txt sinon warning.

```
~/myPackage> python setup.py sdist
writing manifest file 'MANIFEST'
creating packet1-1.0
making hard links in packet1-1.0...
hard linking setup.py -> packet1-1.0
creating dist
Creating tar archive
removing 'packet1-1.0' (and everything under it)
~/myPackage> cd dist
~/myPackage/dist> ls
packet1-1.0.tar.gz
```

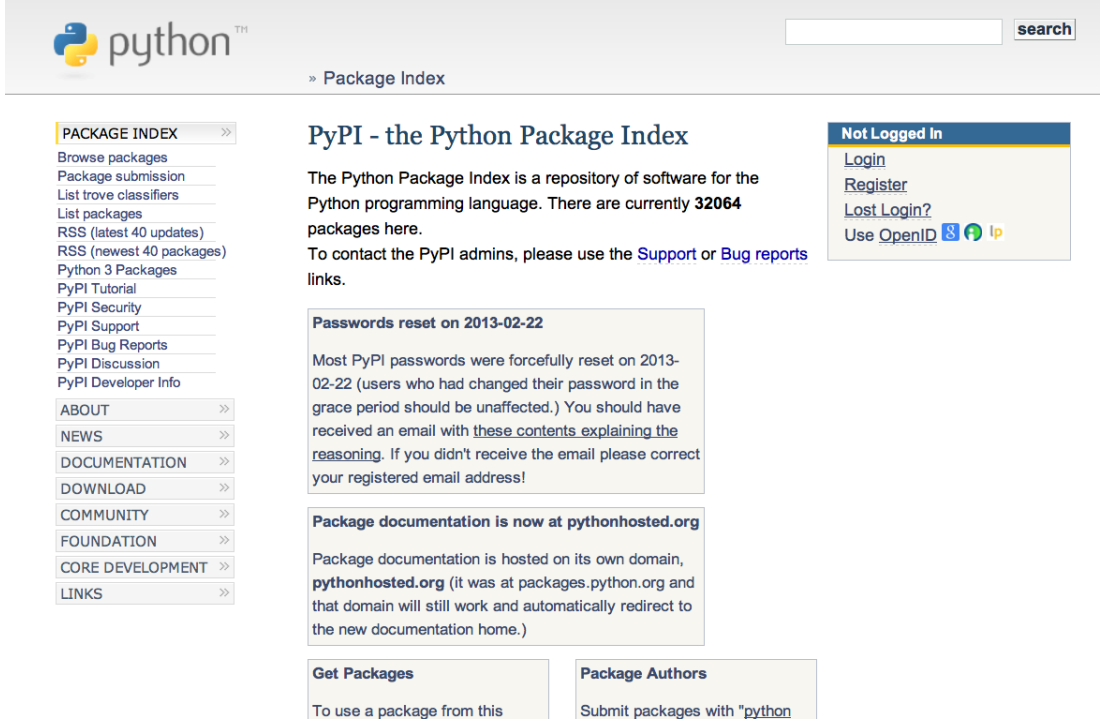
# Installation des paquets

- Décompresser le paquet : il existe une bibliothèque 'tarfile' dans la librairie standard de Python qui permet de décompresser les fichiers 'tar'.
- Exécuter dans un terminal : "python setup.py"
- L'installation se fait dans "PYTHONHOME/site-packages".

```
~/tempdir/packageToInstall> python setup.py
```

# Python Package Index

- Python Package Index : PyPI
  `https://pypi.python.org/pypi`
- Base de données de paquets disponibles.



# Installation des paquets

- A la base **distutils**, ne gère pas les dépendances : python setup.py
- **setuptools** : introduit la commande 'easy_install' qui opère sur de fichers 'egg': easy_install example.egg
- setuptools a généré **distribute** qui gère les dépendances.
- **pip** : remplace 'easy_install' : "pip install paquetExample"

# Outline

# Principales Distributions Scientifiques

- Numpy : essentielle en calcul numérique.
- Matplotlib, Pylab : tracé de figures et fonctions à la matlab.
- Scipy : outils scientifiques spécifiques.
- Maiavi : 3D avancée.

# Numpy

- 

: `http://www.numpy.org`
- *NumPy is the fundamental package for scientific computing with Python*
- Création d'objets ndarray.
- ndarray créés à partir de listes.
- Accès aux éléments : A[i, j]

```
>>> import numpy as np
>>> A = np.array([1, 5, 2, 4, 3])
>>> type(A)
<type 'numpy.ndarray'>
>>> A.dtype
dtype('int32')
>>> A = np.array([[1., 2, 3], [4, 5, 6], [7, 8, 9.]])
>>> A[0, 0]
1.0
>>> A[2, 2]
9.0
```

# Numpy

## Tableaux ND et fonctions associées

- Les exemples sont donnés dans IPython avec les fonctions Pylab chargées.

```
>>> A = array([[1,2,3],[4,5,6],[7,8,9]])
>>> whos

Variable   Type        Data/Info
--------------------------------
A          ndarray     3x3: 9 elems, type `int64`, 72 bytes

>>> A
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> A.size
9

>>> A.shape
(3,3)

>>> B = array([[1,0,0],[0,1,0],[0,0,1]])

>>> A*B

array([[ 1.,  0.,  0.],
       [ 0.,  5.,  0.],
       [ 0.,  0.,  9.]])
```

# Numpy

## Tableaux ND et fonctions associées

- Méthodes
  - nonzero, max, min, mean, std ...
  - sum, cumprod, cumsum ...
  - reshape, resize, flatten, transpose ...

- Fonctions
  - $*$ : produit élt./élt.
  - $dot(.,.)$ : produit matriciel
  - ...

# Numpy

## NDarray *vs.* matrix

```
>>> A = array([[1.,2,3],[4,5,6],[7,8,9]])
>>> B = array([[1,0,0],[0,1,0],[0,0,1]])
>>> C = matrix([[1.,2,3],[4,5,6],[7,8,9]])
>>> D = matrix([[1,0,0],[0,1,0],[0,0,1]])

>>> whos
Variable   Type        Data/Info
-------------------------------
A          ndarray     3x3: 9 elems, type `float64`, 72 bytes
B          ndarray     3x3: 9 elems, type `int64`, 72 bytes
C          matrix      [[ 1.  2.  3.]\n [ 4.  5.  6.]\n [ 7.  8.  9.]]
D          matrix      [[1 0 0]\n [0 1 0]\n [0 0 1]]

>>> dot(A,B)

array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])

>>>  C*D

matrix([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.],
        [ 7.,  8.,  9.]])
```

# Numpy

## matrix

- Méthodes
  - min, max, mean, std, ...
  - .T, .H, .I, ...
  - reshape, flatten, ...
- Fonctions
  - inv, svd, eig, ...

### Différences entre array et matrix

```
>>> A = array([[1.,2,3],[4,5,6],[7,8,9]])
>>> B = matrix([[1.,2,3],[4,5,6],[7,8,9]])

>>> A**2
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.],
       [ 49.,  64.,  81.]])

>>> B**2
matrix([[  30.,   36.,   42.],
        [  66.,   81.,   96.],
        [ 102.,  126.,  150.]])
```

# Numpy

## NDarray *vs.* matrix

```
>>> A = array([[1.,2,3],[4,5,6],[7,8,9]])
>>> B = array([[1,0,0],[0,1,0],[0,0,1]])
>>> C = matrix([[1.,2,3],[4,5,6],[7,8,9]])
>>> D = matrix([[1,0,0],[0,1,0],[0,0,1]])

>>> whos
Variable   Type        Data/Info
-------------------------------
A          ndarray     3x3: 9 elems, type `float64`, 72 bytes
B          ndarray     3x3: 9 elems, type `int64`, 72 bytes
C          matrix      [[ 1.  2.  3.]\n [ 4.  5.  6.]\n [ 7.  8.  9.]]
D          matrix      [[1 0 0]\n [0 1 0]\n [0 0 1]]

>>> dot(A,B)

array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])

>>>  C*D

matrix([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.],
        [ 7.,  8.,  9.]])
```

# Numpy

## matrix

- Méthodes
  - min, max, mean, std, ...
  - .T, .H, .I, ...
  - reshape, flatten, ...
- Fonctions
  - inv, svd, eig, ...

### Différences entre array et matrix

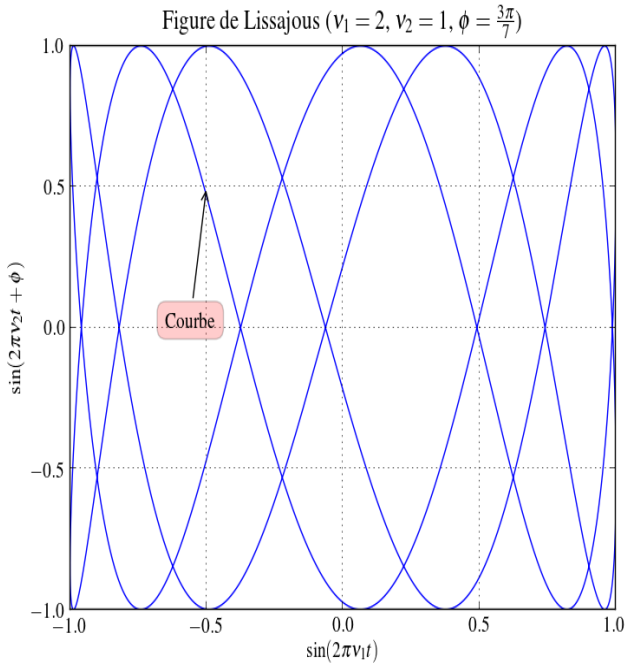
```
>>> A = array([[1.,2,3],[4,5,6],[7,8,9]])
>>> B = matrix([[1.,2,3],[4,5,6],[7,8,9]])

>>> A**2
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.],
       [ 49.,  64.,  81.]])

>>> B**2
matrix([[  30.,   36.,   42.],
        [  66.,   81.,   96.],
        [ 102.,  126.,  150.]])
```

# Matplotlib

## Librairie graphique

- Outils graphiques 2D
- Toolkits : Basemap, Axesgrid, mplot3d
- Beaucoup d'exemples : www.matplotlib.org

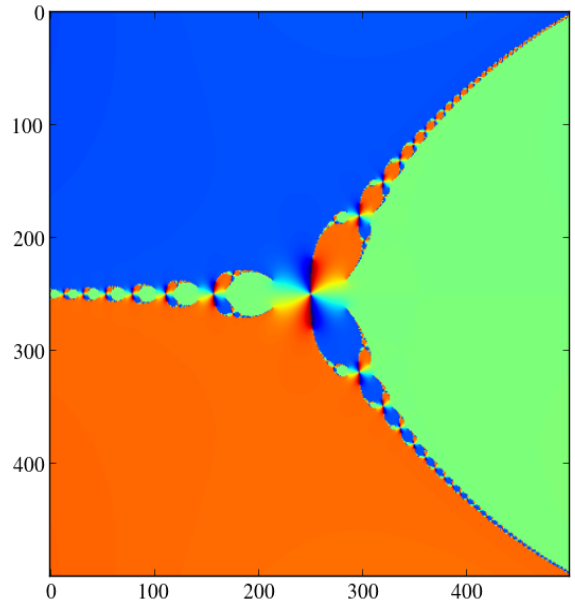
```
>>> t=arange(0,1,10e-5)
>>> S1=sin(2*pi*2*t);S2=sin(2*pi*7*t+(3*pi/7)),
>>> fig=figure()
>>> ax=fig.add_subplot(111,xlim=(-1,1),ylim=(-1,1))
>>> courbe=ax.plot(S1,S2);ax.grid('on')
>>> annotate('Courbe',xy=(-0.5,0.5),xytext=(-0.65,0),\
       bbox=dict(boxstyle='round,pad=0.5',fc='red',\
       alpha=0.2),arrowprops=dict(arrowstyle='->',\
         connectionstyle='arc3,rad=0'))
>>> title(r"Figure␣de␣Lissajous␣($\nu_1=2$,␣$\nu_2=1$,\
␣␣␣␣␣␣␣␣␣␣$\phi=\frac{3\pi}{7}$)")
>>> xlabel(r"$\sin(2\pi\nu_1␣t)$")
>>> ylabel(r"$\sin(2\pi\nu_2␣t␣+␣\phi)$")
>>> show()
```



# Matplotlib

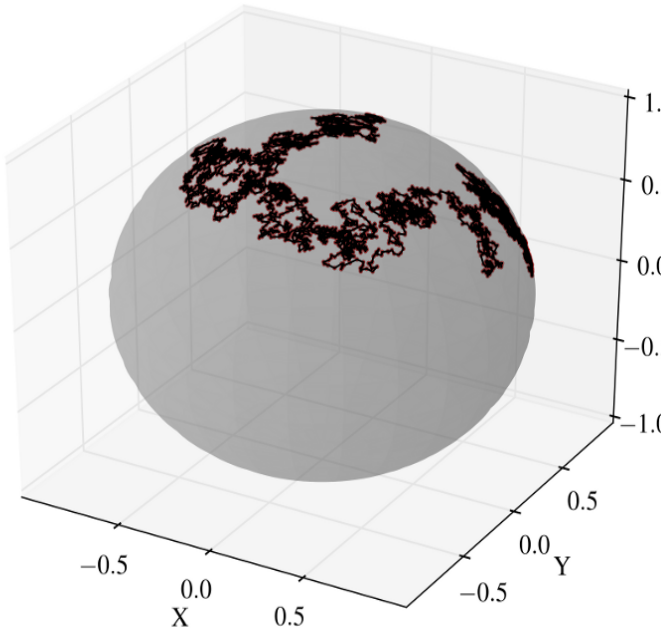
## Exemples

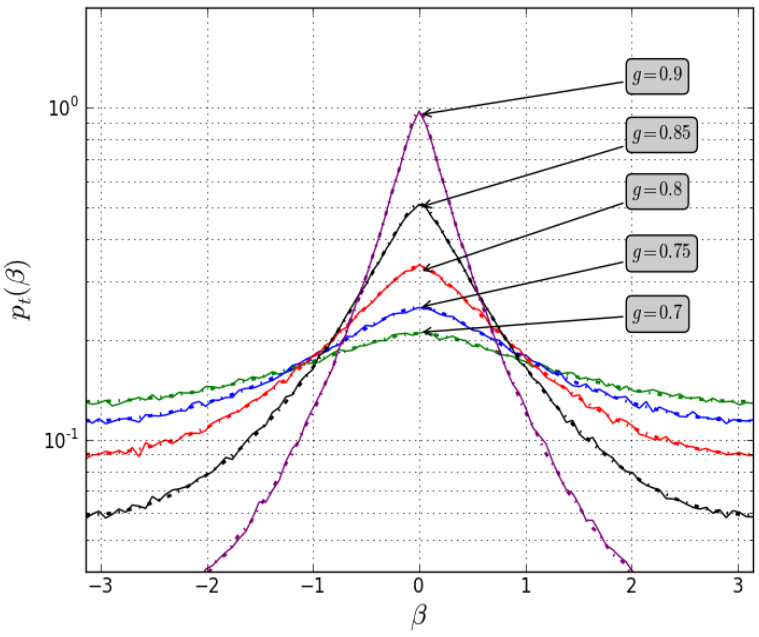
```
>>> [X,Y] = meshgrid(linspace(-2,2,500),\
linspace(-2,2,500))
>>> Z=X+Y*1j
>>> k=1
>>> while k<=75:
>>>  Z = Z - (Z/3 - 1) / (3*Z**2)
>>> k=k+1
>>> close('all')
>>> imshow(angle(Z),cmap=cm.gray)
>>> #savefig('/MonChemin/Lenom.pdf',format='pdf')
>>> show()
```



# Matplotlib

## Exemples



# Pylab

- Module de Matplotlib : matplotlib/pylab.py
- Redéfinit des fonctions à la Matlab.
- Raccourci : "import pylab" au lieu de "import matplotlib.pylab"
- En général, et exceptionnellement, s'utilise : "from pylab import *"

```
>>> from pylab import *
>>> plot([1, 5, 2, 4, 3])
[<matplotlib.lines.Line2D object at 0x4ec05b0>]
>>> show()
>>> a = randn(100, 10)
>>> type(a)
<type 'numpy.ndarray'>
>>> a.shape
(100, 10)
>>> help(matplotlib.pylab)
```

# Scipy

## Scientific library

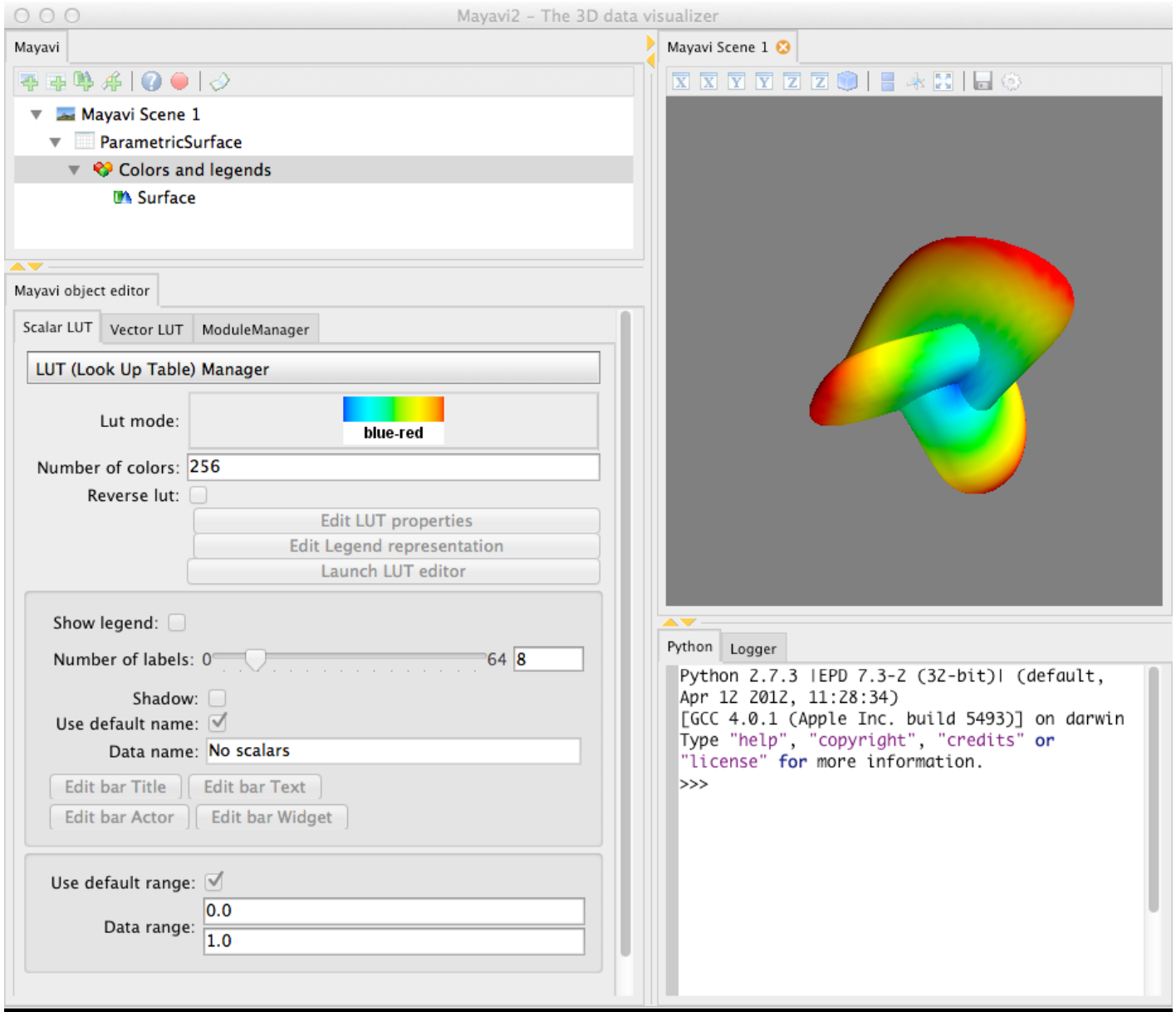
- 

: http://www.scipy.org

| | |
|---|---|
| Clustering package | scipy.cluster |
| Discrete Fourier transforms | scipy.fftpack |
| Interpolation | scipy.interpolate |
| Linear algebra | scipy.linalg |
| Multi-dimensional image processing | scipy.ndimage |
| Optimization and root finding | scipy.optimize |
| Sparse matrices | scipy.sparse |
| Compressed Sparse Graph Routines | scipy.sparse.csgraph |
| Special functions | scipy.special |
| Statistical functions for masked arrays | scipy.stats.mstats |
| Constants | scipy.constants |
| Integration and ODEs | scipy.integrate |
| Input and output | scipy.io |
| Miscellaneous routines | scipy.misc |
| Orthogonal distance regression | scipy.odr |
| **Signal processing** | scipy.signal |
| Sparse linear algebra | scipy.sparse.linalg |
| Spatial algorithms and data structures | scipy.spatial |
| Statistical functions | scipy.stats |
| C/C++ integration | scipy.weave |

# Mayavi

- Manipulation des objets 3D améliorée.



# Mayavi

## import mayavi.engine

- Dans une console Python : import mayavi. . . .

```
>>> from numpy import array
>>> from mayavi.api import Engine
>>> engine = Engine()
>>> engine.start()
>>> engine.new_scene()
>>> from mayavi.sources.parametric_surface import ParametricSurface
>>> parametric_surface1 = ParametricSurface()
>>> scene = engine.scenes[0]
>>> engine.add_source(parametric_surface1, scene)
>>> from mayavi.modules.surface import Surface
>>> surface1 = Surface()
>>> engine.add_filter(surface1,parametric_surface1)
```

# Importation de bibliothèques de fonction écrites en C

## Exemple using module ctypes

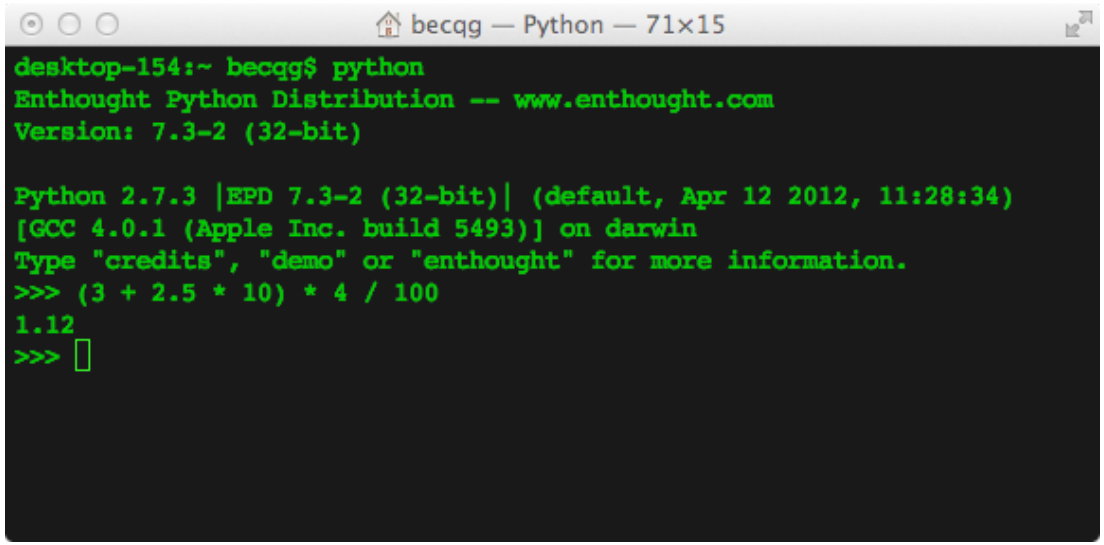
```
import ctypes

myLib = ctypes.LoadLibray('myLib.lib')
fun_c = myLib.fun
fun_c.argtypes = [ctypes.c_double, ctypes.c_int]
fun_c.restype = ctypes.c_int (default)

n = len(s)
pathFile = os.path.dirname(__file__)
libName = os.path.join(pathFile, 'clz.lib')
print('libName + ' + libName)
libCLZ = ctypes.CDLL(libName)
clz_c = libCLZ.clz
clz_c.restype = ctypes.c_uint
sequence = numpy.ctypeslib.ndpointer(dtype=numpy.int)
clz_c.argtypes = ([sequence, ctypes.c_uint])
# conversion of s into sequence with numpy.asarray
c = clz_c(numpy.asarray(s, dtype='int'), n)
```

# Outline

# Python.org

## Official

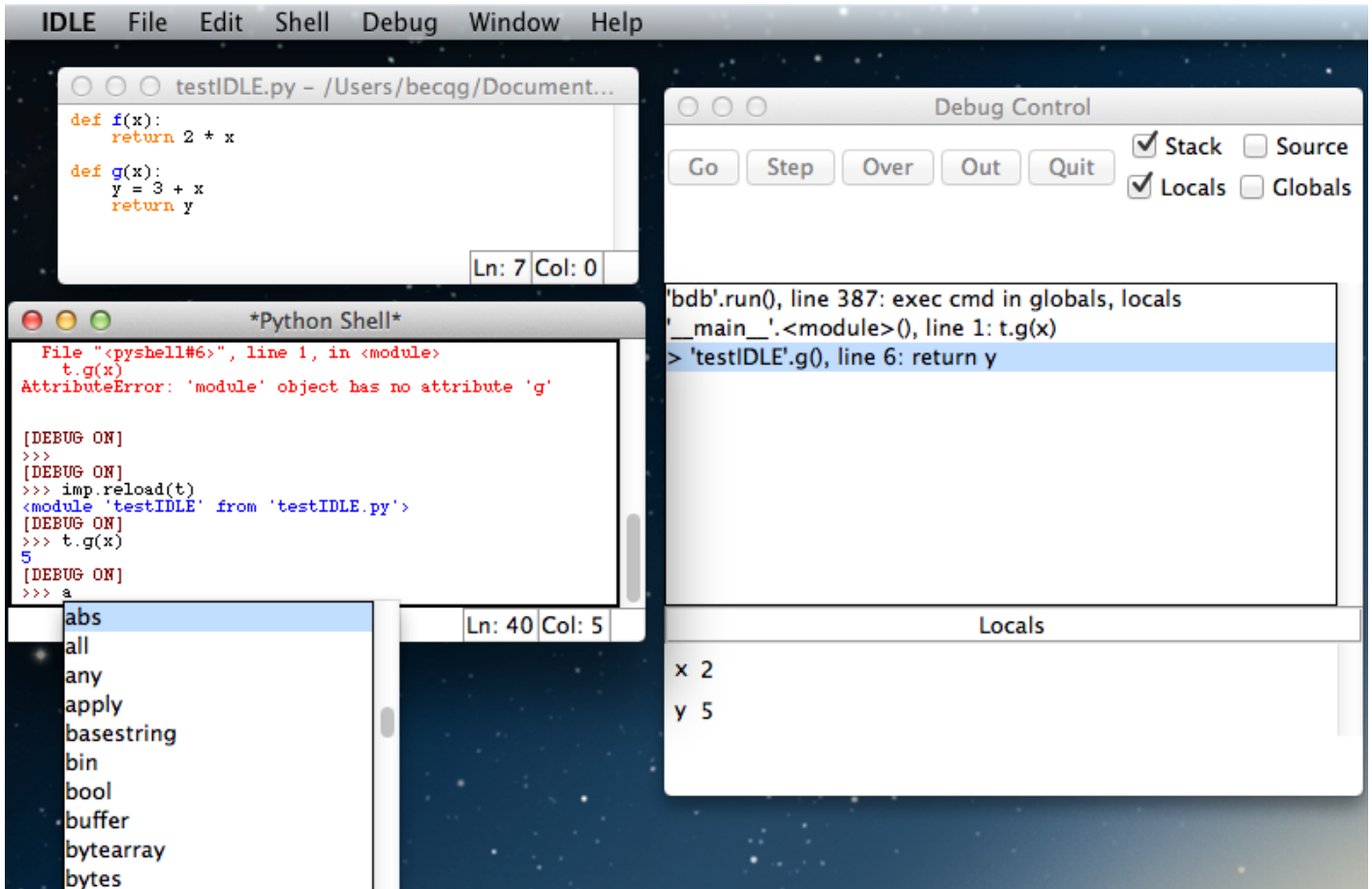
- Python 2.7 ou 3.x téléchargeable sur `www.python.org`.
- Livré uniquemenent avec la bibliothèque standard.
- Inclus l'interpréteur Python natif accessible à partir de l'environnement système.
- Parfait pour tester des petits bouts de codes.



# Python IDLE

## official

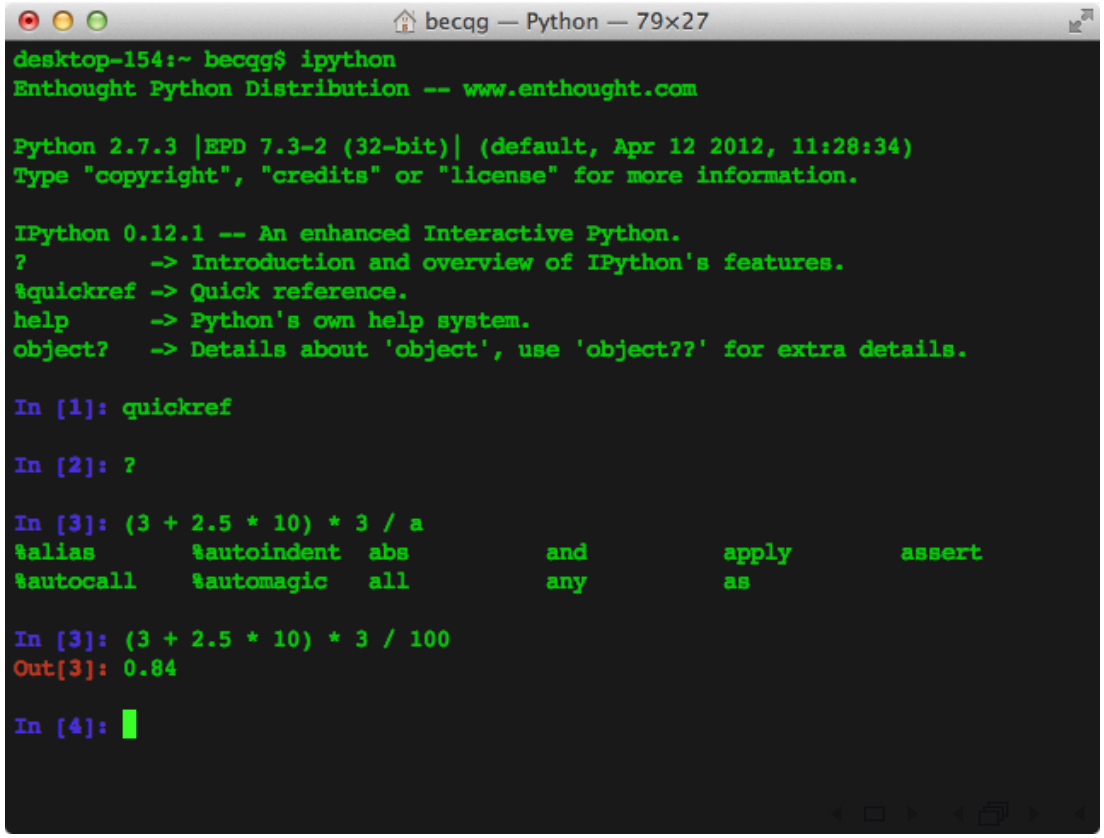
- Python livré avec un Integrated DeveLopment Environment (IDLE) :
  - Une console Python : coloration automatique, autocomplétion . . .
  - Un éditeur de texte : indentation automatique, coloration syntaxique, debuggueur, . . .



# IPython

## Intro

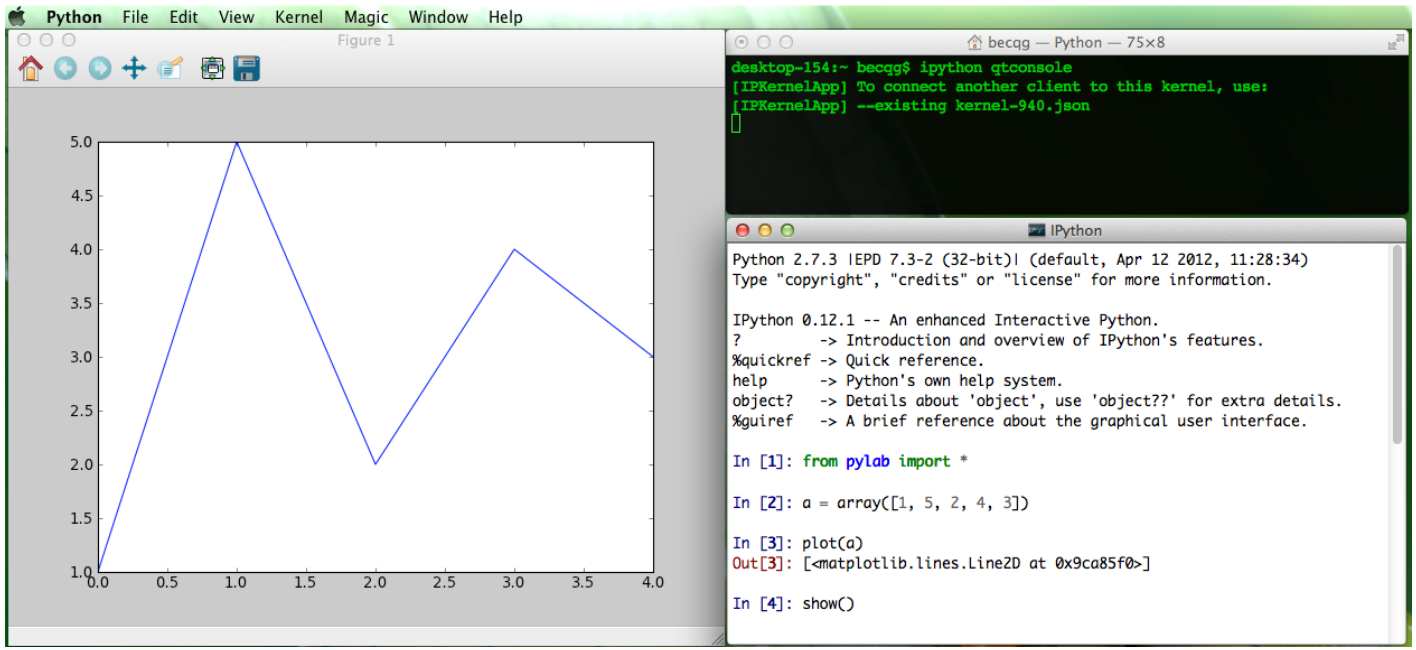
- IP[y]: IPython Interactive Computing IPy: `http://ipython.org`
- Console interactive accessible via le shell : coloration syntaxique, fonctions magiques, mémorisation des commandes, débuggueur, profileur, calculs parallèles . . .
- Plusieurs options cf. 'man ipython' ou 'ipython -help'.



# IPython

## Qtconsole

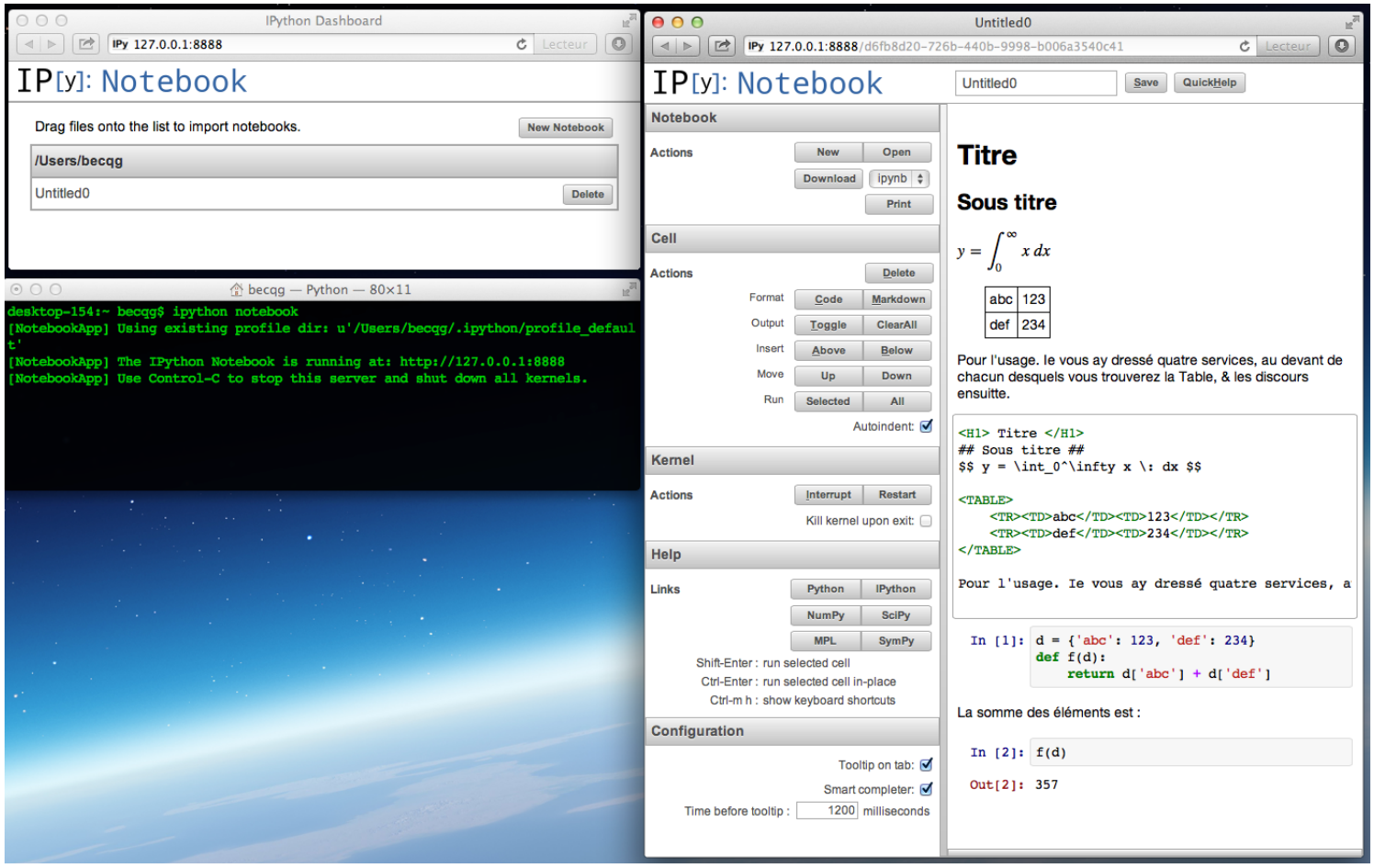
- Commande dans le shell : ipython qtconsole
- Environnement graphique Qt qui permet de tracer des figures via matplotlib ou pylab.
- Commande directe : ipython qtconsole –pylab



# IPython

## notebook

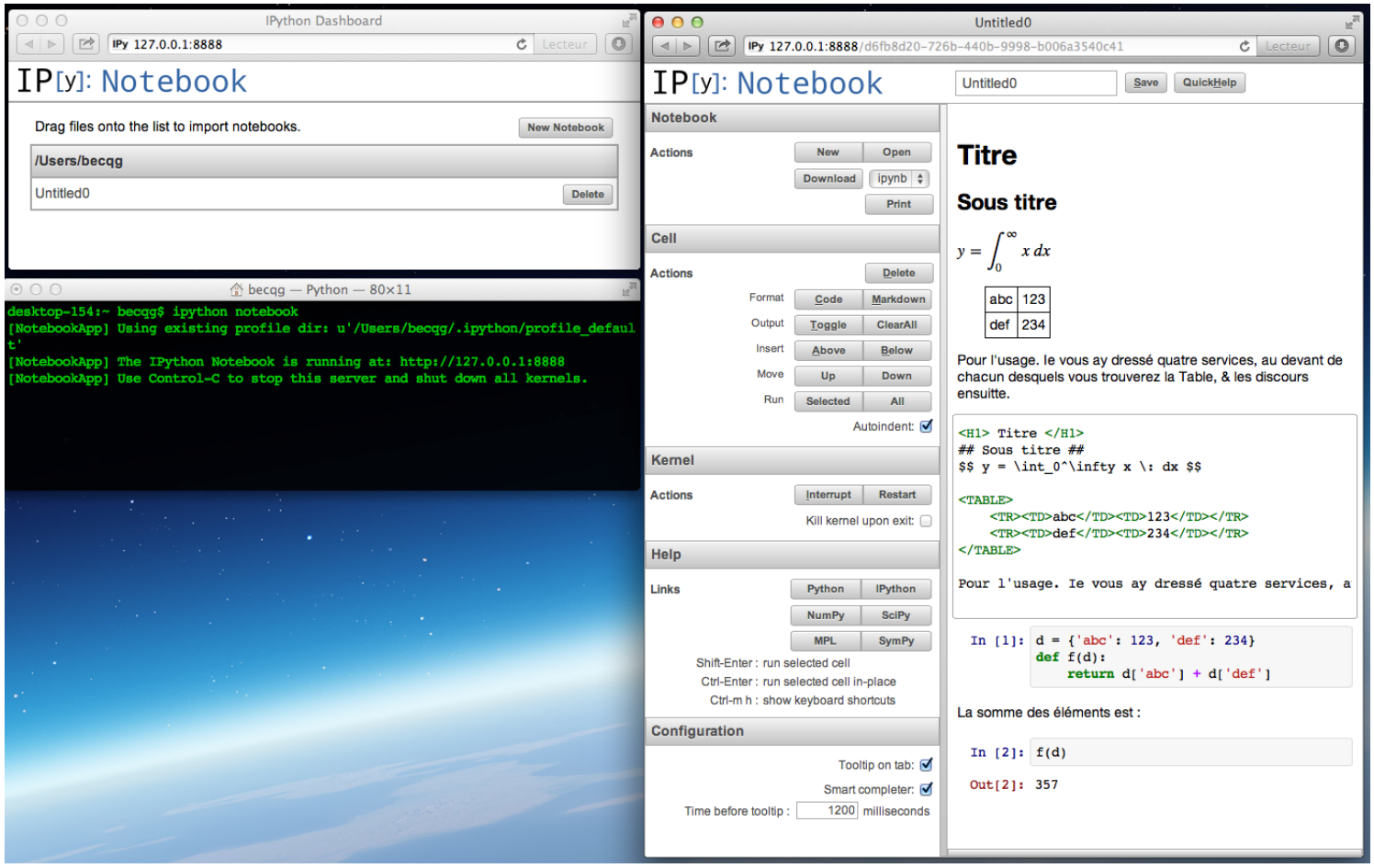
- Commande dans le shell : ipython notebook
- Editeur dans le navigateur HTML, *Web-based interactive computational environment*.



# IPython

## notebook

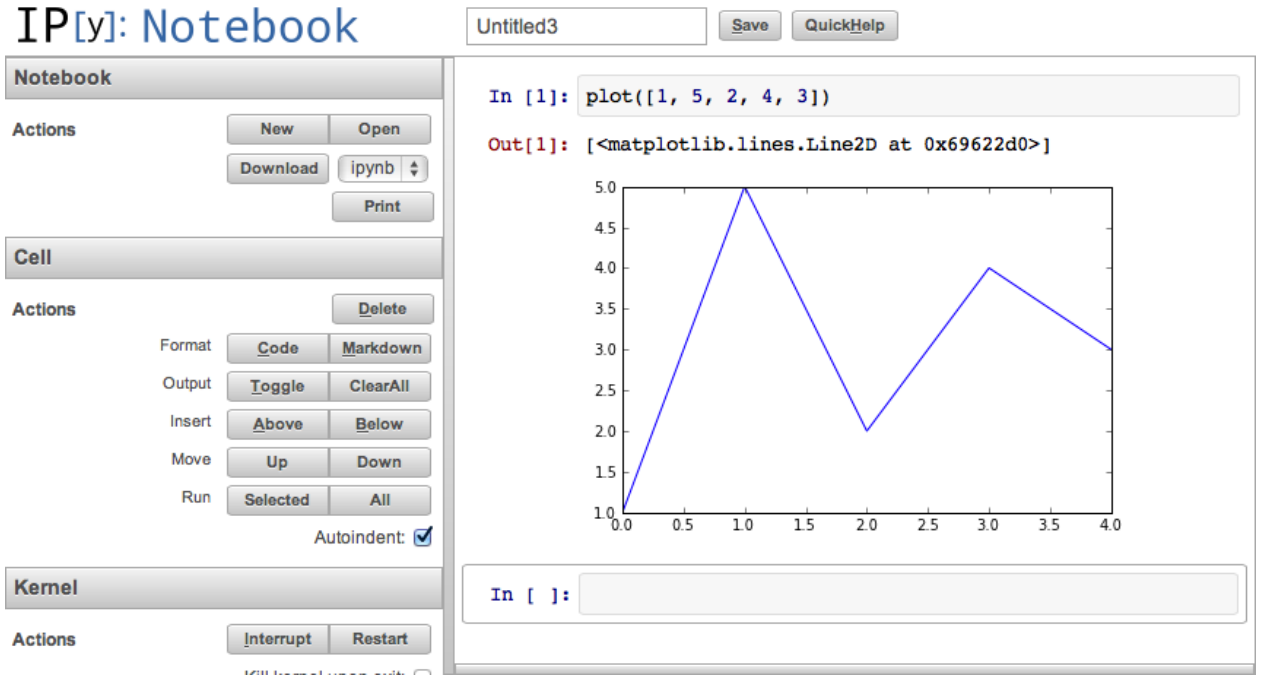
- Cellules de codes ou de documentation : *Cell Mode* à la Matlab, *Document-Based Workflow* à la Mathematica.
- Balisage du texte en LaTeX, HTML ou Markdown.



# IPython

## notebook

- Commande pour importer pylab et graphique dans l'interface html : 'ipython notebook –pylab inline'
- Génération de rapports dans le menu 'notebook > action > print' puis impression en pdf pour le navigateur HTML.

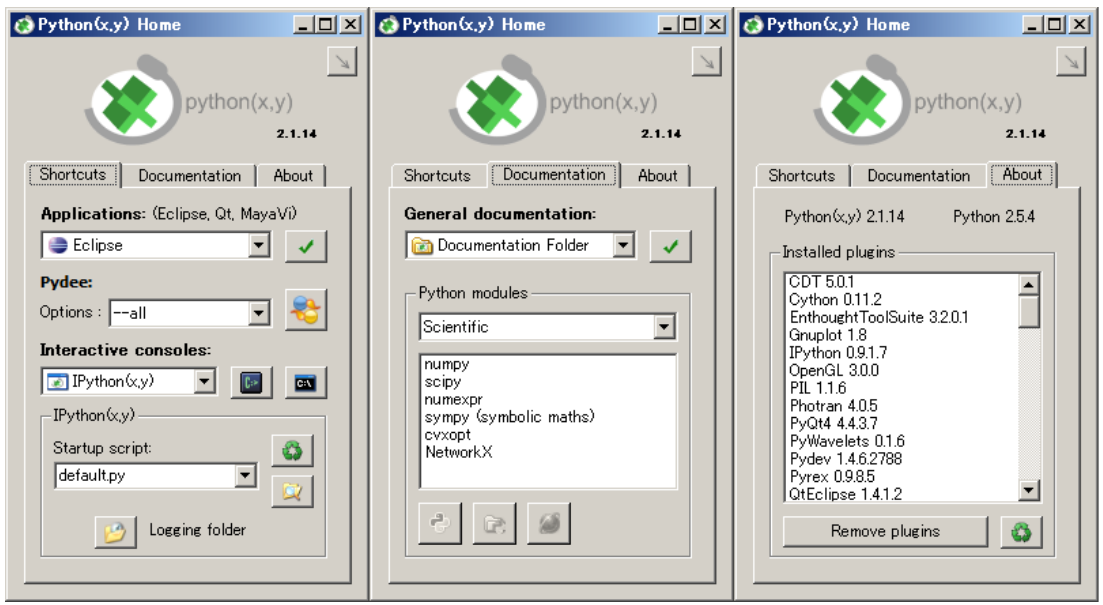


# Enthought Canopy

- ENTHOUGHT :
  `https://www.enthought.com/products/canopy/`
- Contient Python et +100 librairies orientées applications scientifiques.
- Multi-plateformes, *Easy installation and update*.
- Gratuit pour les étudiants et les universitaires.
- Anciennement Enthought Python Distribution EPD.
- QtConsole, iPython.

# Python(x,y)

- Python(x,y) : `https://code.google.com/p/pythonxy/`
- *Python(x,y) is a free scientific and engineering development software for numerical computations, data analysis and data visualization based on Python programming language, Qt graphical user interfaces and Spyder interactive scientific development environment.*
- Un grand nombre de librairies scientifiques, entre autres.
- Interface avec Eclipse (IDE principalement pour Java mais aussi Python).

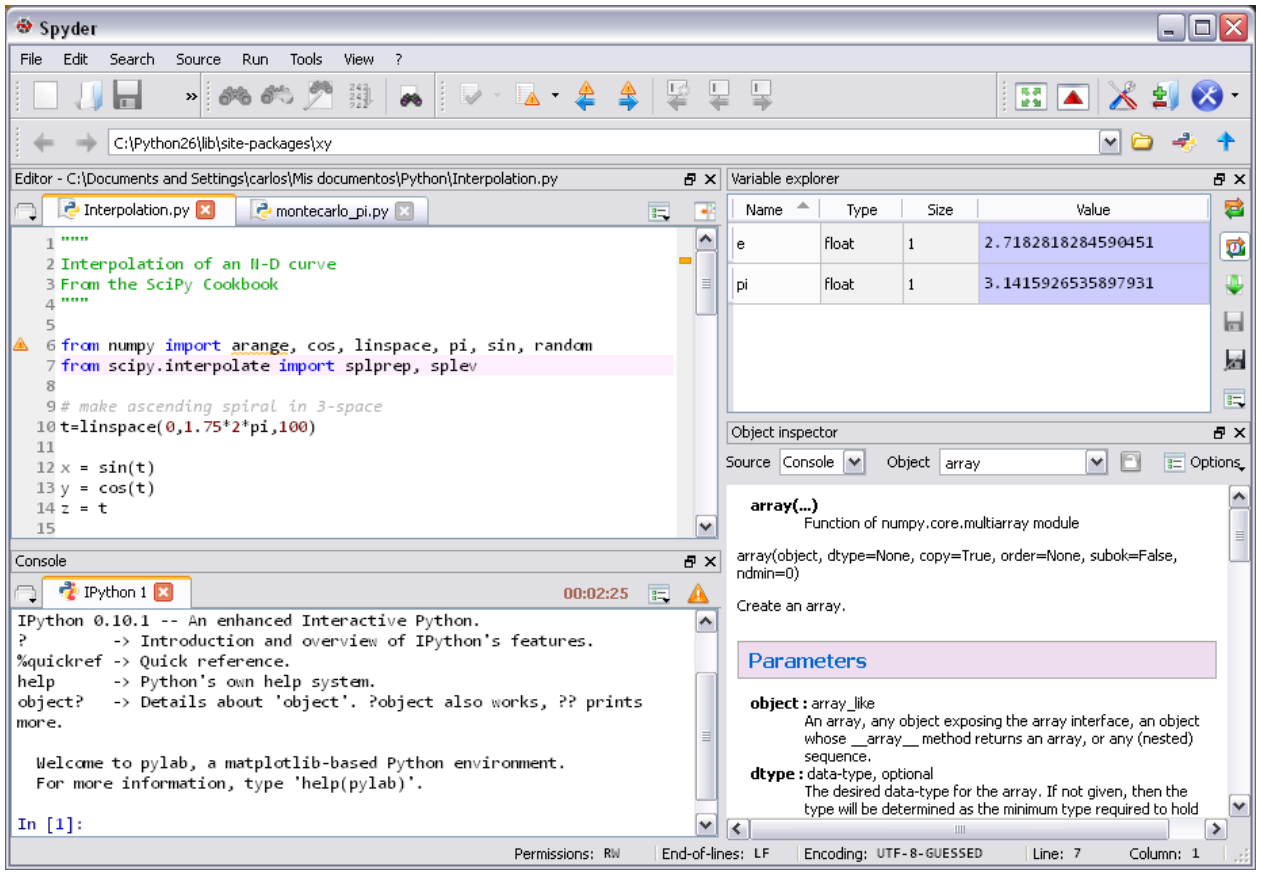



source web

# Python(x,y)

## Spyder

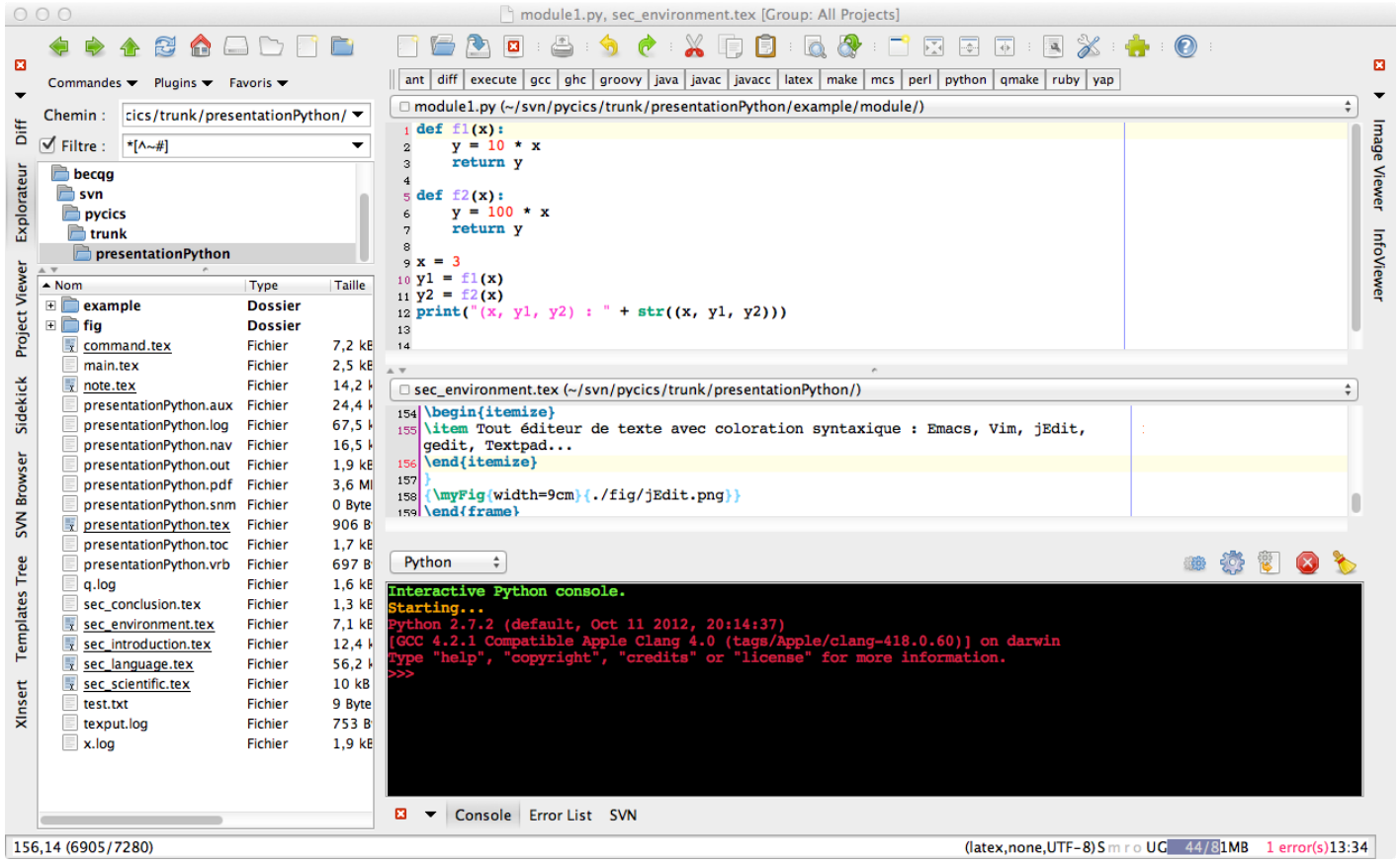
- Spyder (multiplateforme) : environnement de type Matlab pour Python.




source web

# Autres Integrated Development Environment

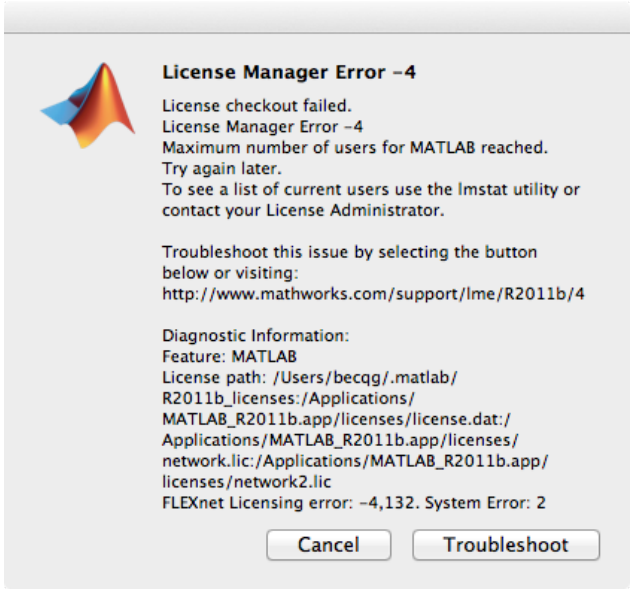
- Tout éditeur de texte avec coloration syntaxique : Emacs, Vim, jEdit, gedit, Textpad...



# Outline

# Python vs Matlab

- Outils équivalents : matrices vs ndarray, console, script, graphisme, GUI, cell mode vs ipython notebook ...
- Matlab, Matrix Laboratory, a des bibliothèques d'algèbre linéaire plus rapide que Numpy ou Scipy (actuellement).
- Python est un langage de programmation.
- Python est plus proche du code C pour prototyper.
- Chargement des modules à la volée en Python.
- Python est gratuit.



# Outline

# Classe exemples avec self

- Dans le corps de la classe, 'self' n'est pas défini.

```
class Canard():
...     self.a = 10
...
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
 File "<stdin>", line 2, in Canard
NameError: name 'self' is not defined
```

# Classe exemples avec self

- Dans une méthode seul self.nomAttribut est accessible.

```
class Canard():
...     a = 10
...     def __init__(self):
...             self.b = 100
...             print(self.a)
...             print(b)
...
riri = Canard()
10
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
 File "<stdin>", line 6, in __init__
NameError: global name 'b' is not defined
```

# Classe exemples avec self

- 'self' est conventionnel.
- Le premier paramètre d'une méthode est considéré comme l'objet lui même.

```
class Canard():
...     def __init__(obj):
...             obj.a = 10
...             print(obj.a)
...
riri = Canard()
10
```

# Classe exemples avec self

- Possibilité de rajouter des arguments optionnels lors de l'instanciation d'un objet.

```
class Canard():
...     def __init__(self, patte=2):
...             self.a = patte
...             print(self.a)
...
riri = Canard(patte=3)
3
```