

Introduction au langage de programmation Python

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

Guillaume Becq

November 22, 2017



Introduction

Description du Langage

Description des Paquets Scientifiques

Distributions et Environnements de travail

Conclusion

Outline

Introduction

Description du Langage

Description des Paquets Scientifiques

Distributions et Environnements de travail

Conclusion

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.
 - ▶ Crée 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



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 Environment)
 - ▶ 1999 : Python 1.6



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 (61 ans)
 - ▶ Développeur néerlandais
 - ▶ 1982 : M. Sc
 - ▶ Développeur ABC.
- ▶ Créeateur Python : *Benevolent Dictator For Life (BDFL)*
 - ▶ 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

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 '.txt' et '.tex' du répertoire 'a' vers 'b'.

```
# -*- encode: utf-8 -*-
import shutil, os
extList = ['.txt', '.tex']
pathSrc = 'a'
pathDst = 'b'
fileList = os.listdir(pathSrc)
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, des définitions de fonctions, de classes... 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 "> > >"

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Utilisations

Utilisation de l'interpréteur

- ▶ Association d'une valeur à une variable : `a = 'abc'`
- ▶ Affichage de la valeur : `print()`
- ▶ Liste des variables, 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

The screenshot shows the Python Software Foundation website at www.python.org. The page features a dark blue header with the Python logo and navigation links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header is a search bar and social media links. The main content area has a dark background with a light blue navigation bar at the top. On the left, there's a code snippet demonstrating Python syntax:

```
# Python 3: Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!

# Input, assignment
>>> name = input('What is your name?\n')
>>> print('Hi, %s.' % name)
What is your name?
Python
Hi, Python.
```

To the right of the code, a yellow button with a right-pointing arrow is overlaid. The text "Quick & Easy to Learn" is displayed above a paragraph about Python's learnability. At the bottom of the main content area, there are five numbered buttons (1, 2, 3, 4, 5) and a call-to-action message: "Python is a programming language that lets you work quickly and integrate systems more effectively. [»» Learn More](#)".

[Get Started](#)

[Download](#)

[Docs](#)

[Jobs](#)



Documentation

Trouver la documentation

Download

Download these documents

Docs for other versions

Python 3.7 (in development)
Python 3.5 (stable)
Python 2.7 (stable)
Old versions

Other resources

PEP Index
Beginner's Guide
Book List
Audio/Visual Talks

Python 3.6.3 documentation

Welcome! This is the documentation for Python 3.6.3.

Parts of the documentation:

What's new in Python 3.6?

or all "What's new" documents since 2.0

Tutorial

start here

Library Reference

keep this under your pillow

Language Reference

describes syntax and language elements

Python Setup and Usage

how to use Python on different platforms

Python HOWTOs

in-depth documents on specific topics

Installing Python Modules

installing from the Python Package Index & other sources

Distributing Python Modules

publishing modules for installation by others

Extending and Embedding

tutorial for C/C++ programmers

Python/C API

reference for C/C++ programmers

FAQs

frequently asked questions (with answers!)



Documentation

Lire la documentation - Tutorial

The screenshot shows a PDF viewer window titled "tutorial.pdf". The top menu bar includes "Outils", "Signer", and "Commentaire". The left sidebar is titled "Signets" and lists various chapters of the tutorial, each with a small icon. The main content area displays the title "Python Tutorial" and "Release 2.7.3", followed by author information: "Guido van Rossum" and "Fred L. Drake, Jr., editor". At the bottom right of the content area, the date "June 11, 2012" is visible. The status bar at the bottom of the viewer window shows "1 sur 128" and "55,3%".

Signets

- Whetting Your Appetite
- Using the Python Interpreter
- An Informal Introduction to Python
- More Control Flow Tools
- Data Structures
- Modules
- Input and Output
- Errors and Exceptions
- Classes
- Brief Tour of the Standard Library
- Brief Tour of the Standard Library - Part II
- What Now?
- Interactive Input Editing and History Substitution
- Floating Point Arithmetic: Issues and Limitations
- Glossary
- About these documents
- History and License
- Copyright
- Index

Python Tutorial
Release 2.7.3

Guido van Rossum
Fred L. Drake, Jr., editor

June 11, 2012



Documentation

Lire la documentation - Library reference

The screenshot shows a PDF viewer window titled "library.pdf". The left sidebar contains a table of contents for built-in types, including sections like "Introduction", "Built-in Functions", "Non-essential Built-in Functions", "Built-in Constants", and "Built-in Types" (which is currently selected). The main content area displays a table of bitwise operations:

Operation	Result	Notes
$x \mid y$	bitwise or of x and y	
$x \wedge y$	bitwise exclusive or of x and y	
$x \& y$	bitwise and of x and y	
$x \ll n$	x shifted left by n bits	(1)(2)
$x \gg n$	x shifted right by n bits	(1)(3)
$\sim x$	the bits of x inverted	

Below the table, there are three notes:

1. Negative shift counts are illegal and cause a `ValueError` to be raised.
2. A left shift by n bits is equivalent to multiplication by $\text{pow}(2, n)$. A long integer is returned if the result exceeds the range of plain integers.
3. A right shift by n bits is equivalent to division by $\text{pow}(2, n)$.

5.4.2 Additional Methods on Integer Types

The integer types implement the `numbers.Integral abstract base class`. In addition, they provide one more method:

```
int.bit_length()  
long.bit_length()
```

Return the number of bits necessary to represent an integer in binary, excluding the sign and leading zeros:

```
>>> n = -37  
>>> bin(n)  
'-0b100101'  
>>> n.bit_length()  
6
```

More precisely, if x is nonzero, then `x.bit_length()` is the unique positive integer k such that $2^{k-1} \leq \text{abs}(x) < 2^k$. Equivalently, when $\text{abs}(x)$ is small enough to have a correctly rounded logarithm, then $k = 1 + \text{int}(\log(\text{abs}(x), 2))$. If x is zero, then `x.bit_length()` returns 0.

Equivalent to:



Documentation

Lire la documentation - Library reference

The Python Library Reference, Release 2.7.3

os.listdir(path)
Return a list containing the names of the entries in the directory given by *path*. The list is in arbitrary order. It does not include the special entries '.', '.,' and '..', even if they are present in the directory.

Availability: Unix, Windows. Changed in version 2.3: On Windows NT/2k/XP and Unix, if *path* is a Unicode object, the result will be a list of Unicode objects. Undecodable filenames will still be returned as string objects.

os.lstat(path)
Perform the equivalent of an `lstat()` system call on the given path. Similar to `stat()`, but does not follow symbolic links. On platforms that do not support symbolic links, this is an alias for `stat()`.

os.mkfifo(path[, mode])
Create a FIFO (a named pipe) named *path* with numeric mode *mode*. The default *mode* is 0666 (octal). The current umask value is first masked out from the mode.

Availability: Unix.

FIFOs are pipes that can be accessed like regular files. FIFOs exist until they are deleted (for example with `os.unlink()`). Generally, FIFOs are used as rendezvous between "client" and "server" type processes: the server opens the FIFO for reading, and the client opens it for writing. Note that `mkfifo()` doesn't open the FIFO — it just creates the rendezvous point.

os.mknod(filename[, mode=0600][, device=0])
Create a filesystem node (file, device special file or named pipe) named *filename*. *mode* specifies both the permissions to use and the type of node to be created, being combined (bitwise OR) with one of `stat.S_IFREG`, `stat.S_IFCHR`, `stat.S_IFBLK`, and `stat.S_IFIFO` (those constants are available in `stat`). For `stat.S_IFCHR` and `stat.S_IFBLK`, *device* defines the newly created device special file (probably using `os.makedev()`), otherwise it is ignored. New in version 2.3.

os.major(device)
Extract the device major number from a raw device number (usually the `st_dev` or `st_rdev` field from `stat`). New in version 2.3.

os.minor(device)
Extract the device minor number from a raw device number (usually the `st_dev` or `st_rdev` field from `stat`). New in version 2.3.

os.makedev(major, minor)
Compose a raw device number from the major and minor device numbers. New in version 2.3.

os.mkdir(path[, mode])



Documentation

Lire la documentation - Language reference

The screenshot shows a PDF viewer window with the title "reference.pdf" at the top. The toolbar includes icons for file operations like Open, Save, Print, and a magnifying glass. The status bar shows "12 sur 121" and "95,1K". On the right, there are buttons for "Outils", "Signer", and "Commentaire".

The left sidebar is a tree view of the document structure:

- Signets
- Introduction
- Lexical analysis
 - Line structure
 - Other tokens
 - Identifiers and keywords
 - Literals
 - Operators
 - Delimiters
- Data model
- Execution model
- Expressions
- Simple statements
- Compound statements
- Top-level components
 - Full Grammar specification
 - Glossary
- About these documents
- History and License
- Copyright
- Index

The main content area displays section 2.3 Identifiers and keywords:

2.3 Identifiers and keywords

Identifiers (also referred to as *names*) are described by the following lexical definitions:

```
identifier ::= (letter|"_") (letter | digit | "_")*
letter ::= lowercase | uppercase
lowercase ::= "a"..."z"
uppercase ::= "A"..."Z"
digit ::= "0"..."9"
```

Identifiers are unlimited in length. Case is significant.

2.3.1 Keywords

The following identifiers are used as reserved words, or *keywords* of the language, and cannot be used as ordinary identifiers. They must be spelled exactly as written here:

```
and      del      from      not      while
as       elif     global    or       with
assert   else     if        pass     yield
break   except   import   print
class    exec    in       raise
continue finally  is      return
def     for     lambda  try
```

Changed in version 2.4: `None` became a constant and is now recognized by the compiler as a name for the built-in object `None`. Although it is not a keyword, you cannot assign a different object to it. Changed in version 2.5: Using `as` and `with` as identifiers triggers a warning. To use them as keywords, enable the `with_statement` future feature. Changed in version 2.6: `as` and `with` are full keywords.

2.3.2 Reserved classes of identifiers

Certain classes of identifiers (besides keywords) have special meanings. These classes are identified by the patterns of leading and trailing underscore characters:

- Not imported by `from module import *`. The special identifier `_` is used in the interactive interpreter to store the result of the last evaluation; it is stored in the `__builtin__` module. When not in interactive mode, `_` has no special meaning and is not defined. See section *The import statement*.

p. ex. grammaire sous forme de Backus-Naur (BNF)



Python Enhancements Proposals

PEPs

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

The screenshot shows a web browser displaying the Python Software Foundation's PEP index page at www.python.org/dev/peps/. The page title is "PEP 0 -- Index of Python Enhancement Proposals (PEPs)". On the left, there is a sidebar with several tweets from the Python Software Foundation (@ThePSF) account. The main content area displays the PEP 0 details:

PEP:	0
Title:	Index of Python Enhancement Proposals (PEPs)
Last-Modified:	2015-01-12
Author:	David Goodger <goodger at python.org>, Barry Warsaw <barry at python.org>
Status:	Active
Type:	Informational
Created:	13-Jul-2000

Below the table, there is an "Introduction" section with the following text:

This PEP contains the index of all Python Enhancement Proposals, known as PEPs. PEP numbers are assigned by the PEP editors, and once assigned are never changed[1]. The Mercurial history[2] of the PEP texts represent their historical record.

At the bottom of the page, there is a "Index by Category" section with a table:

num	title	owner
---	-----	-----

Footnote: Meta-PEPs (PEPs about PEPs or Processes)



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).
- ▶ Autres distributions : Enthought Canopy, pythonXY, Anaconda, WinPython, ...
- ▶ Implémentations alternatives :
 - ▶ Langage identique mais implémentation différentes permet par exemple : d'interfacer du java facilement (Jython), d'être plus performant pour les calculs avec un compilateur JIT (Pypy)...

Installation

Python 2.7 vs 3.x

- ▶ 2.7 : figée.
- ▶ 3.x : présent et futur :
 - ▶ *better Unicode support*
 - ▶ Quelques inconsistances 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

Introduction

Description du Langage

Description des Paquets Scientifiques

Distributions et Environnements de travail

Conclusion

Type de base

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

Nombres

Entiers

- ▶ Division non entière en 3.x, conversion de type non implicite.
- ▶ 'type()' indique le type d'une variable.

```
>>> a = 1
>>> b = 2
>>> c = a / b
>>> print(c)
0.5
>>> print(type(a))
<class 'int'>
>>> print(type(c))
<class 'float'>
```

Nombres

Réels

- ▶ Utilisation du point pour passer en réels.

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

Nombres

Réels notations scientifiques

- ▶ Notation scientifique de type signe, mantisse et exposant.

```
>>> a = 1.234e100
>>> a
1.234e+100
>>> type(a)
<class '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)
<class '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(_)
<class '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.
- ▶ + et * donne des entiers 'int'.
- ▶ Opérateurs booléens spécifiques.

```
>>> a = True
>>> type(a)
<class 'bool'>
>>> b = False
>>> c = a + b
>>> print(c)
1
>>> type(c)
<class '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	-	subtraction
	*	multiplication	/	division
	%	modulo	//	floor division
	**	power		
	<<	shifting left	>>	shifting right
		idem entiers hors shifting et bitwise op.		
réels				
complexes		idem réels		
booléens	&, and	and	, or	or
	^	xor	not	not

Nombres

Opérateurs de comparaisons

Comparaisons $>$, $<$, \geq , \leq , \equiv
 \neq , \neq (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
```

Nombres

Typage explicite, conversion de type

- ▶ possibilité de typer explicitement avec la fonction associée au type voulue : `int()`, `float()` ...
- ▶ conversion, implicite pour certaines opérations.

```
>>> a = float(9)
>>> type(a)
<class 'float'>
>>> c = 1 / a
>>> print(c)
0.1111111111111111
>>> b = 9
>>> c = 1 / float(b)
>>> print(c)
0.1111111111111111
>>> d = complex(1, 2)
>>> print(d)
(1+2j)
>>> print(str(d))
(1+2j)
```

Nombres et Caractères

Binary, Hexadecimal, Octal, Character, Unicode Character

- ▶ binary (0b) (bin), hexadecimal (0x) (hex), octal (0c) (oct)
- ▶ character ascii + unicode (chr) (< 0x10FFFF)

```
>>> a = 0x597d
>>> print(a)
22909
>>> a = 0b0101
>>> print(a)
5
>>> a = chr(42)
>>> print(a)
*
>>> b = chr(0x597D)
>>> print(b)
chinese symbol ni
>>> a = bin(22909)
>>> print(a)
0b101100101111101
>>> type(a)
<class 'str'>
```

Séquences

Chaînes de caractères

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

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

Séquences

Chaînes de caractères

- ▶ Alternance des single/double quote.
- ▶ Control code (\n, \t, etc.) façon 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 " +
'"empotterez" 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 "empotterez" avec un petit brin de fines herbes
```

Séquences

Chaînes de caractères

- ▶ forme multiligne """ """ 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)
'\n'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\\n\\n    & 'Troussez', faites les blanchir, \\n    & Les "empottez" avec
un petit brun de fines herbes"
```

Séquences

Chaînes de caractères

- ▶ forme brute (`r" "` ou `r' '` ou `r"""" """"` ou `r'''' ''''`)
- ▶ forme unicode (`" "` ou `u" "` ou `u' '` ou `u"""" """"` ou `u'''' ''''`)

```
>>> 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)  
"Potage de sant\\xe9\\n\\tLe potage de sant\\xe9 se fait de chapons...  
, "  
>>> print("\u4F60\u597D")
```

你好

Séquences

Chaînes de bytes

- ▶ Distinction entre chaîne de caractère et chaîne de bytes.

```
>>> a = 'abcd'  
>>> a.encode('utf8')  
b'abcd'  
>>> a.encode('ascii')  
b'abcd'  
>>> b = a.encode('utf32')  
>>> print(b)  
b'\xff\xfe\x00\x00a\x00\x00\x00b\x00\x00\x00c\x00\x00\x00d\x00\x00'  
>>> c = b.decode('utf32')  
>>> print(c)  
abcdef
```

Séquences

Listes

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

```
>>> a = [1, 2, 3]
>>> type(a)
<class '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'
```

Séquences

Listes

- ▶ découpage, slicing (`(:, i:, :j, i:j, i:j:k, slice(i,j,k))`)
- ▶ Indices négatifs de -1 à $-n$ permettent 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[2:2+4] # a[2:6]
[3, 4, 5, 6]
>>> a[2:6:2]
[3, 5]
>>> s = slice(2, 6, 2)
>>> a[s]
[3, 5]
>>> a[-1]
9
>>> a[-9]
1
```

Séquences

Listes

- ▶ Imbrications, nested.
- ▶ Attention aux erreurs d'indexation, les listes Python ne sont pas des 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
```

Séquences

Tuples

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

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

Séquences

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
```

Séquences

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(prog)  
[('011', '0001'), ('100', '0010'), ('110', '0011'), ('010', '0001')]
```

Séquences

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)]
```

Séquences

Dictionnaires

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

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

Séquences

Ensemble

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

```
>>> E = {1, 2, 3}
>>> type(E)
<class 'set'>
>>> print(E)
set([1, 2, 3])
>>> E[0]
... 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
... 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])
```

Séquences

Résumé

- ▶ Chaîne de caractères " " ou ''
- ▶ Liste [,]
- ▶ Tuple (,)
- ▶ Dictionnaire {"": , "": }
- ▶ Ensemble { , }

Syntaxe

- ▶ Fonctions prédéfinies, 'Builtin' Functions
- ▶ Structures de contrôles
- ▶ Fonctions
- ▶ Objets et Classes

Syntaxe

Fonctions prédefinies

► 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 prédefinies

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

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

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 suivie 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 ces 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 == "green"):
...     print("00FF00")
... elif (a == "red"):
...     print("FF0000")
... elif (a == "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

- ▶ une étendue (range) de $[0, n[$: range(n)
- ▶ $[i, i + k, \dots, j[$: 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("problème entrées sorties : " + str(e))  
... except TypeError as e :  
...     print("problème 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("problème entrées sorties : " + str(e))  
... except TypeError as e :  
...     print("problème 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 d'une 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*.

```
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
    statut par défaut : ne fait 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
- ▶ par convention, majuscule pour la première lettre de classname : Canard
- ▶ propriétés : variables avec une valeur par défaut.
- ▶ méthodes : fonctions 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 expekte(self):  
...         self.status = "en attente"
```

Syntaxe

Objets : définition des attributs et méthodes spéciales

- ▶ `__init__` : création lors de l'instanciation. Il est conseillé d'initialiser les attributs dans ce bloc.
- ▶ `__...__` : attributs spéciaux ou méthodes spéciales (special attributes, special methods), reconnues par Python.

```
...
def __init__(self):
    self.pattes = 2
    self.ailes = 2
    self.status = "en attente"
def __del__(self):
    print('arrrggghhh')
def __add__(self, canard2):
    canard_fils = Canard()
    canard_fils.pattes = int(self.pattes + canard2.pattes / 2)
    return canard_fils
...
```

Syntaxe

Objets : instantiation et usage

- ▶ instantiation : objet = Classname()
- ▶ propriétés obtenues par : object.property (cf structure en C)
- ▶ méthodes obtenues par : object.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.expecte()
0
>>> dir(riri)
[..., 'ailes', 'chante', 'espere', 'mange', 'pattes',
'status']
```

Syntaxe

Objets : méthodes spéciales

- ▶ destruction : `del(instance)`

```
>>> rilou = riri + loulou
>>> rilou.pattes
3
>>> del(rilou)
arrrgggghhh
```

Syntaxe

Objets : exemple avec des nombres

- En python toutes les variables sont des objets.

```
>>> a = 123
>>> dir(a)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__int__',
'__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__',
'__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__',
'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__',
'__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__',
'__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__',
'__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__',
'__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator',
'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
>>> type(a)
<class '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', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
>>> type(a)
<class '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)
<class 'tuple'>
>>> a.count(1)
2
>>> a.index(1)
0
```

Syntaxe

Objets : exemple avec les chaînes de caractères

```
>>> a = "abc.def"
>>> dir(a)
[..., 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha',
'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', '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'
```

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("from import -> (x, y1, y2) : " + (x, y1, y2))
```

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

```
import moduleQuiAUnNomBeaucoupTropLong as myMod
x = 4
y1 = myMod.f1(x)
y2 = myMod.f2(x)
```

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

Module

Importation ou exécution de code

- attribut spécial : '`__name__`'

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

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

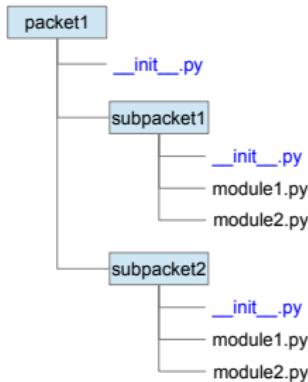
```
>>> import moduleMain
Affichage en import ou en exécution
__name__ : moduleMain
```

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

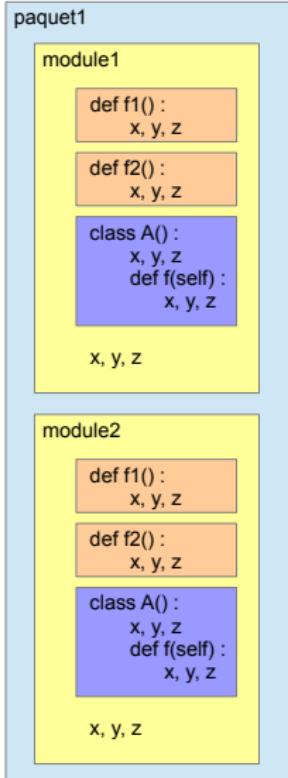
Paquet

- ▶ Un dossier contenant :
 - ▶ des modules
 - ▶ un fichier '`__init__.py`', vide ou non.
- ▶ possibilité de faire des sous-paquets.

```
>>> 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.module2.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

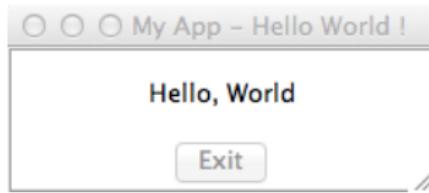
```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10',
'log1p', 'log2', 'modf', 'nan', '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).

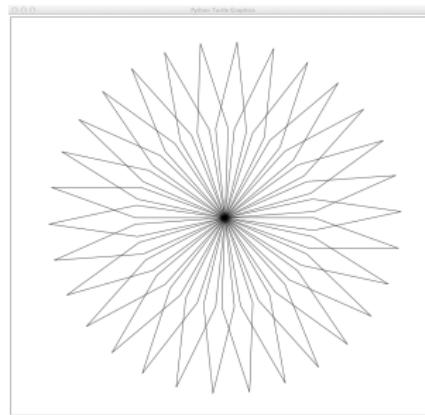
```
>>> import tkinter
>>> from tkinter.constants import *
>>> tk = tkinter.Tk()
>>> tk.title("My App - Hello World !")
>>> frame = tkinter.Frame(tk, relief=RIDGE, borderwidth=2)
>>> frame.pack(fill=BOTH, expand=1)
>>> label = tkinter.Label(frame, text="Hello, World")
>>> label.pack(fill=X, expand=1)
>>> 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 caractères 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
```



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)`

```
Help on module modules:

NAME
    module - Module pour tester les docstrings.

FILE
    /Users/becqg/svn/pycics/trunk/presentationPython/example/doc/module.py

DESCRIPTION
    Contient deux fonctions et une classe.
    Ne fait rien.

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

CLASSES
    A

        class A
            | A est une classe.
            |
            | C'est une classe vide qui ne contient qu'un docstring.

FUNCTIONS
    fun1()
        fun1 c'est bien.

        Fonction sans paramètres
        Ne fait rien.

    fun2()
        fun2 c'est mieux.

        Sans paramètres
        Ne fait rien

(END) [ ]
```



Documentation

pydoc

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

The screenshot shows a web browser window titled "Python: module module". The URL in the address bar is "/Users/becq/svn/pycics/trunk/presentationPython/example/doc/module.py". On the right side of the page, there is a blue header bar with the word "module" and the path. Below it, a link to "index". The main content area has a blue header "Module pour tester les docstrings." followed by a pink section titled "Classes". It contains a class definition for "class A" which is described as a empty class containing only a docstring. Below this is an orange section titled "Functions" which lists two functions: "fun1()" and "fun2()". Each function has its own docstring describing what it does.

```
Module pour tester les docstrings.  
Contient deux fonctions et une classe.  
Ne fait rien.  
Copyright 2013 G. Becq, Gipsa-lab, UMR 5216, CNRS.  
  
Classes  
A  
  
class A  
    A est une classe.  
    C'est une classe vide qui ne contient qu'un docstring.  
  
Functions  
fun1()  
    fun1 c'est bien.  
    Fonction sans paramètres  
    Ne fait rien.  
fun2()  
    fun2 c'est mieux.  
    Sans paramètres  
    Ne fait rien
```



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.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 install
```

Python Package Index

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

The screenshot shows the PyPI homepage. At the top, there's a navigation bar with links for "Logout", "Lost Login?", "Use OpenID", "Status", and "Nothing to report". Below the navigation is a search bar labeled "search". The main content area has several sections: "Get Packages", "Package Authors", and "Infrastructure". The "Get Packages" section contains a table of recently updated packages, each with a link to its details page. The table includes columns for "Updated" and "Package". The first few rows of the table are as follows:

Updated	Package	Description
2015-01-14	collective.geo.wafNet 0.1b7	Add geo Views for dynamically content with wafNet js library
2015-01-14	restrmote 0.1.3	Sync local databases from REST API.
2015-01-14	restrmote 0.1.2	Bind Google Sheets
2015-01-14	django-theming 1.0	Django application. Implement theming concept, flexible and configurable. Allow theming for host URL.
2015-01-14	book 2	A Tekton's app to handle file download and upload
2015-01-14	swallow 1.0.0	BCEK - XMPP - IMC - CLI - RSS - UDP - API
2015-01-14	swallow 1.0.2	Swallow for data distribution
2015-01-14	openid-provider-pebble 0.8	Swallow - Data sharing
2015-01-14	casstion 0.2.0	Pebble fork of django_openid_provider
2015-01-14	fedoridriver 0.3.0	Just a quick dockerfile test for my entertainment
2015-01-14	logomatic 0.1.0	F-Droid Server Tools
2015-01-14	django-basic-auth 1.5	logging handler collection
2015-01-14	django-basic-email 0.0.1	Integrate a WordPress blog into Zimbra
2015-01-14	countryparty-lb 9.49.3	Django Basic Email application
2015-01-14	gandi-cli 0.12	CountryParty Reference Implementation
2015-01-14	firm-web 0.4.3	Gandi command line interface
2015-01-14	pyannote.algorithms 0.2.1	Frontend Web Application for Fedora Notifications
2015-01-14	firm.consumer 0.4.3	PyAnnote algorithms



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 fichiers 'egg': `easy_install example.egg`
- ▶ setuptools a généré **distribute** qui gère les dépendances.
- ▶ **pip** : remplace 'easy_install' : "`pip install paquetExample`"

Outline

Introduction

Description du Langage

Description des Paquets Scientifiques

Distributions et Environnements de travail

Conclusion

Quelques Paquets et Outils Scientifiques

- ▶ SciPy : Scientific Python
 - ▶ Numpy
 - ▶ SciPy library
 - ▶ Matplotlib
 - ▶ Pandas
 - ▶ IPython -> Jupyter notebook
- ▶ Scikit-learn : machine learning.
- ▶ Mayavi : objets 3D avancés
- ▶ ...

SciPy

► <http://www.scipy.org/>

The screenshot shows the official website for SciPy.org. At the top, there's a navigation bar with links like "About SciPy", "Install", "Getting Started", "Documentation", "Bug Reports", "Topical Software", "Citing", "Cookbook", "SciPy Conferences", "Blogs", and "NumFOCUS". Below the header, there are five main icons: "Install" (blue circle with a green arrow), "Getting Started" (yellow circle with a green S), "Documentation" (blue circle with a white book and S), "Report Bugs" (blue circle with a red strawberry and S), and "Blogs" (orange square with a white RSS icon). A text block below these says: "SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:". There are two rows of package cards. The first row includes NumPy (blue cube icon), SciPy library (red S icon), and Matplotlib (radar chart icon). The second row includes IPython (IPython logo icon), Sympy (green S icon), and pandas (red panda icon). A "More information..." button is located between the two rows. The "News" section at the bottom left mentions "SciPy 1.0.0 released" (2017-10-25) and "See Obtaining NumPy & SciPy libraries.". On the right side, there are two boxes: one titled "CORE PACKAGES:" listing Numpy, SciPy library, Matplotlib, IPython, Sympy, and Pandas; and another box containing three circular icons representing eye, ear, and brain.

SciPy.org Sponsored By ENTHOUGHT

Install Getting Started Documentation Report Bugs Blogs

SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

IP[y]:
IPython Enhanced Interactive Console

Sympy
Symbolic mathematics

pandas
Data structures & analysis

More information...

News

SciPy 1.0.0 released See Obtaining NumPy & SciPy libraries.
(2017-10-25)

About SciPy
Install
Getting Started
Documentation
Bug Reports
Topical Software
Citing
Cookbook
SciPy Conferences
Blogs
NumFOCUS

CORE PACKAGES:
Numpy
SciPy library
Matplotlib
IPython
Sympy
Pandas



Numpy

- ▶  NumPy <http://www.numpy.org>
- ▶ *NumPy is the fundamental package for scientific computing with Python*

```
>>> import numpy
>>> help(numpy)

Help on package numpy:

NAME
    numpy

DESCRIPTION
    NumPy
    =====

    Provides
        1. An array object of arbitrary homogeneous items
        2. Fast mathematical operations over arrays
        3. Linear Algebra, Fourier Transforms, Random Number Generation
    ...

...
```

N-dimensional array Object

- ▶ N-dimensional array : ndarray
- ▶ Création d'un tableau vide et réservation de l'espace (empty)
- ▶ Accès aux éléments : $A[i, j, \dots]$
- ▶ indices de 0 à $n - 1$
- ▶ indices négatifs de $-n$ à -1

```
>>> A = numpy.empty((2, 2))
>>> print(A)
[[ -1.28822975e-231    2.68678092e+154]
 [ 2.24497156e-314    2.24499315e-314]]
>>> A[0, 0] = 1
>>> A[1, 0] = 2
>>> A[0, 1] = 11
>>> A[1, 1] = 12
>>> print(A)
[[ 1.   2.]
 [ 11.  12.]]
>>> type(A)
<class 'numpy.ndarray'>
>>> A[-1, -2] = 11
```

N-dimensional array Object

- ▶ Rappel : accès aux propriétés et méthodes (dir)

```
>>> dir(A)
['T', ..., 'all', 'any', 'argmax', 'argmin', 'argpartition', 'argsort',
'astype', 'base', 'byteswap', 'choose', 'clip', 'compress', 'conj', 'conjugate',
'copy', 'ctypes', 'cumprod', 'cumsum', 'data', 'diagonal', 'dot', 'dtype',
'dump', 'dumps', 'fill', 'flags', 'flat', 'flatten', 'getfield', 'imag', 'item',
'itemset', 'itemsize', 'max', 'mean', 'min', ' nbytes', 'ndim', 'newbyteorder',
'nonzero', 'partition', 'prod', 'ptp', 'put', 'ravel', 'real', 'repeat',
'reshape', 'resize', 'round', 'searchsorted', 'setfield', 'setflags', 'shape',
'size', 'sort', 'squeeze', 'std', 'strides', 'sum', 'swapaxes', 'take',
'tobytes', 'tofile', 'tolist', 'tostring', 'trace', 'transpose', 'var', 'view']
```

N-dimensional array Object

Attributs sur la forme du tableau

- ▶ Forme du tableau (`shape`), c'est un tuple.
- ▶ Nombre de dimension (`ndim`)
- ▶ Type des éléments (`dtype`)
- ▶ Taille du tableau (`size`), c'est le nombre de cellules totales.

```
>>> A.shape
(2, 2)
>>> (nRow, nCol) = A.shape
>>> nRow = A.shape[0]
>>> nCol = A.shape[1]
>>> A.ndim
2
>>> A.dtype
dtype('float64')
>>> A.size
4
```

N-dimensional array Object

Changement de forme

- ▶ Pour changer la forme (reshape)
- ▶ Transposition (T)

```
>>> B = A.reshape((4, 1))
array([[ 1.],
       [ 2.],
       [11.],
       [12.]])
>>> B.ndim
2
>>> B.size
4
>>> B.T
array([[ 1.,   2.,  11.,  12.]])
```

N-dimensional array Object

Vues et copies de tableaux

- ▶ Les éléments de B sont les mêmes que ceux de A, seule la forme change. On a une vue de l'objet (view).
- ▶ Si on veut une copie : méthode copy(), numpy.copy().

```
>>> B[0, 0] = 21
>>> print(A[0, 0], B[0, 0])
21.0 21.0
>>> B[1, 0]
>>> C = A.copy()
>>> C[0, 0] = 31
>>> print(A[0,0], C[0,0])
(21.0, 31.0)
```

N-dimensional array Object

Création de tableaux

- ▶ tableau vide et réservation de l'espace (empty).
- ▶ initialisation à zeros (zeros) ou avec des uns (ones).
- ▶ tableau identité (eye) avec la dimension.
- ▶ à partir de listes (array) ou suivant une étendue (arange).

```
>>> A = numpy.zeros((2, 4))
>>> print(A)
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
>>> A = numpy.ones((3, 2))
>>> print(A)
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]
>>> A = numpy.eye(2)
>>> print(A)
[[ 1.  0.]
 [ 0.  1.]]
>>> A = numpy.array([[1, 2], [11, 12]])
>>> print(A)
[[ 1  2]
 [11 12]]
>>> print(numpy.arange(0.5, 1.7, 0.1))
[ 0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3  1.4  1.5  1.6]
```



N-dimensional array Object

Types

- ▶ Définition du type à la création
- ▶ Changement de type (astype)
- ▶ Multiplication ou addition avec un scalaire typé.

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> print(A.dtype)
int64
>>> A = numpy.array([[1., 2], [11, 12]])
>>> print(A.dtype)
float64
>>> A = numpy.array([[1, 2], [11, 12]], dtype="float")
>>> print(A.dtype)
float64
>>> A = A.astype("complex")
>>> print(A)
[[ 1.+0.j  2.+0.j]
 [ 11.+0.j 12.+0.j]]
>>> A = numpy.array([[1, 2], [11, 12]]) * 1.
>>> print(A.dtype)
float64
```

N-dimensional array Object

Additions, soustractions, multiplications sur les tableaux

- ▶ Addition, soustraction de tableaux ou d'un scalaire (+, -)
- ▶ Multiplication par un scalaire (*)
- ▶ Produit élément par élément (*)

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> print(A + 10)
[[ 11.   12.]
 [ 21.   22.]]
>>> print(A + B)
[[ 4.   6.]
 [ 24.  26.]]
>>> print(A * 10)
[[ 10.   20.]
 [ 110.  120.]]
>>> print(A * B)
[[ 3.   8.]
 [ 143. 168.]]
>>> C = numpy.ones((10, ))
>>> print(A * C)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (2,2) (10)
```

N-dimensional array Object

Produit scalaire

- ▶ Produit scalaire (dot)
- ▶ See also numpy.dot : en général, pour chaque méthode associée à un ndarray, il existe une fonction équivalente dans numpy.

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> print(A.dot(B))
[[ 29.   32.]
 [189.  212.]]
>>> print(numpy.dot(A, B))
[[ 29.   32.]
 [189.  212.]]
>>> C = numpy.ones((10, ))
>>> print(A.dot(C))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: matrices are not aligned
>>>
```

N-dimensional array Object

Division

- ▶ Division par un scalaire (/)
- ▶ Division éléments par éléments (/)

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> print(A / 2)
[[ 0.5   1. ]
 [ 5.5   6. ]]
>>> print(A / B)
[[ 0.33333333  0.5        ]
 [ 0.84615385  0.85714286]]
```

N-dimensional array Object

Autres méthodes

- max, min, sum, mean, std, cumsum, cumprod ... sur tous les éléments ou sur une dimension particulière (kwarg axis).

```
>>> A = numpy.ones((2, 3, 4))
>>> print(A)
[[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]

[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]]
>>> print(A.cumsum())
[ 1.  2.  3.  4.  5.  6.
 7.  8.  9.  10. 11. 12. 13.
14. 15. 16. 17. 18. 19. 20.
21. 22. 23. 24.]
>>> print(A.cumsum(axis=0))
[[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]

[[ 2.  2.  2.  2.]
 [ 2.  2.  2.  2.]
 [ 2.  2.  2.  2.]]]
```

```
>>> print(A.cumsum(axis=1))
[[[ 1.  1.  1.  1.]
 [ 2.  2.  2.  2.]
 [ 3.  3.  3.  3.]]

[[ 1.  1.  1.  1.]
 [ 2.  2.  2.  2.]
 [ 3.  3.  3.  3.]]]
>>> print(A.cumsum(2))
[[[ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]]]

[[ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]]]
```

N-dimensional array Object

Fonctions universelles et constantes

- ▶ Fonction universelle (universal function, ufunc) : fonction qui opère sur tous les éléments du tableau, un par un.
- ▶ Quelques constantes : pi, e, euler_gamma, inf, nan.

```
>>> pi = numpy.pi
>>> A = numpy.deg2rad(numpy.array([[30 / num, 60], [45, 90]]))
>>> B = numpy.cos(A)
>>> print(B)
[[ 8.66025404e-01    5.0000000e-01]
 [ 7.07106781e-01   6.12323400e-17]]
>>> B[1, 1] = numpy.nan
>>> print(numpy.isfinite(B))
[[ True  True]
 [ True False]]
```

N-dimensional array Object

Sélection de sous-tableaux

- ▶ découpage, slicing, comme pour les séquences.
- ▶ ellipse (ellipsis, ...)

```
>>> A = numpy.array([[1, 2, 3, 4], [11, 12, 13, 14]])
>>> print(A)
[[ 1  2  3  4]
 [11 12 13 14]]
>>> print(A[1, :])
[11 12 13 14]
>>> print(A[:, 1:3])
[[ 2  3]
 [12 13]]
>>> A = numpy.arange(24).reshape((2, 3, 4))
>>> print(A[0, ...])
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

N-dimensional array Object

Sélection de sous-tableaux

- ▶ Comparaison et opérateurs logiques
- ▶ Opérations logiques pour sélectionner des éléments (voir aussi advanced indexing + masking)
- ▶ Récupérer les indices (where)

```
>>> A = numpy.array([[1, 2, 3, 4], [11, 12, 13, 14]])
>>> print(A)
[[ 1  2  3  4]
 [11 12 13 14]]
>>> B = A > 2
>>> print(B)
[[False False  True  True]
 [ True  True  True  True]]
>>> print(A[B])
[ 3  4 11 12 13 14]
>>> indices = numpy.where(B)
>>> print(indices[0])
array([0, 0, 1, 1, 1, 1])
>>> print(indices[1])
array([2, 3, 0, 1, 2, 3])
>>> (i, j) = numpy.where(A > 2)
```

N-dimensional array Object

Concaténations

- ▶ Concaténation verticale (vstack, r_[])
- ▶ Concaténation horizontale (hstack, c_[])
- ▶ Concatenate (concatenate, kwarg axis)

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> C = numpy.vstack((A, B)) # r_[A, B]
>>> print(C)
[[ 1  2]
 [11 12]
 [ 3  4]
 [13 14]]
>>> D = numpy.hstack((A, B, A, A)) # c_[A, B, A, A]
>>> print(D)
[[ 1  2  3  4  1  2  1  2]
 [11 12 13 14 11 12 11 12]]
>>> E = numpy.concatenate((A, B, A, A), axis=1)
>>> print(E)
[[ 1  2  3  4  1  2  1  2]
 [11 12 13 14 11 12 11 12]]
```

N-dimensional array Object

Structured Arrays

- ▶ Possibilité de mettre des éléments de types différents.
- ▶ Possibilité de tableaux structurés ...

```
>>> A = numpy.array([[ "a", 1], [ "b", 2]], dtype="object")
>>> print(A)
[[ 'a', 1]
 [ 'b', 2]]
>>> print(A.dtype)
object

>>> A = numpy.array([(1, "abc"), (2, "def")], dtype=[("index", "int"),
        ("name", "S8")])
>>> print(A)
[(1, b'abc') (2, b'def')]
>>> A["index"]
array([1, 2])
>>> A["name"]
array([b'abc', b'def'],
      dtype='|S8')
```

Sauvegarde et lecture de données

- ▶ Enregistrement d'un tableau (save) dans un fichier ".npy"
- ▶ Enregistrement compressé de plusieurs tableaux (savez) au format ".npz"
- ▶ Lecture (load) des fichiers ".npy", ".npz"

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> numpy.save("save_A", A)
>>> del(A)
>>> A = numpy.load("save_A.npy")
>>> print(A)
[[ 1  2]
 [11 12]]
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[21, 22], [31, 32]])
>>> numpy.savez("save_AB", tab1=A, B=B)
>>> del(A, B)
>>> data = numpy.load("save_AB.npz")
>>> print(data["tab1"])
[[ 1  2]
 [11 12]]
>>> print(data["B"])
[[21 22]
 [31 32]]
```

Sauvegarde et lecture de données txt

- ▶ Lecture de fichier texte ".txt" (load ou loadtxt)
- ▶ Enregistrement (savetxt)

```
>>> A = numpy.loadtxt("data.txt")
>>> A
array([[ 1.,   2.,   3.,   4.,   5.],
       [11.,  12.,  13.,  14.,  15.],
       [21.,  22.,  23.,  24.,  25.]])
>>> numpy.savetxt("data.txt", A)
>>> numpy.savetxt("data.csv", A, delimiter=",", header="A, B, C, D, E")
```

Matrix

Définition

- ▶ Classe héritée de ndarray avec ndim = 2.

```
>>> help(numpy.matrix)
class matrix(numpy.ndarray)
|   matrix(data, dtype=None, copy=True)
|
|   Returns a matrix from an array-like object, or from a string of data.
|   A matrix is a specialized 2-D array that retains its 2-D nature
|   through operations. It has certain special operators, such as "*" 
|   (matrix multiplication) and "***" (matrix power).
...
```

Matrix

Saisie

- ▶ Saisie directe de type ndarray avec des listes imbriquées.
- ▶ Possibilité de saisie type Matlab.

```
>>> A = numpy.matrix([[1, 2], [11, 12]])
>>> print(A)
[[ 1  2]
 [11 12]]
>>> type(A)
<class 'numpy.matrixlib.defmatrix.matrix'>
>>> A = numpy.matrix("[1, 2, 3, 4; 11, 12, 13, 14]")
>>> print(A)
[[ 1  2  3  4]
 [11 12 13 14]]
```

Matrix

Multiplication et exposant

- ▶ Produit de matrices (*)
- ▶ Exposant de matrice (**)

```
>>> A = numpy.matrix([[1, 2], [11, 12]])
>>> B = numpy.matrix([[3, 4], [13, 14]])
>>> print(A * B)
[[ 29  32]
 [189 212]]
>>> print(A ** 2)
[[ 23  26]
 [143 166]]
```

Matrix

Opérateurs matriciels courants

- ▶ Transposition (T)
- ▶ Inversion (I)
- ▶ Opérateur Hermitien (H)

```
>>> A = numpy.matrix([[1, 2], [11, 12]])
>>> print(A)
[[ 1  2]
 [11 12]]
>>> print(A.T)
[[ 1 11]
 [ 2 12]]
>>> print(A.I)
print(A.I)
[[[-1.2  0.2]
 [ 1.1 -0.1]]
>>> B = numpy.matrix([[1, 2+1j], [11+1j, 12]])
>>> print(B)
[[ 1.+0.j  2.+1.j]
 [11.+1.j 12.+0.j]]
>>> print(B.H)
[[ 1.-0.j 11.-1.j]
 [ 2.-1.j 12.-0.j]]
```

Autres opérations d'algèbre linéaire

Sous paquet linalg

- ▶ Interface vers des bibliothèques d'algèbre linéaire (numpy.linalg).

```
>>> help(numpy.linalg)
...
Linear algebra basics:
- norm          Vector or matrix norm
- inv           Inverse of a square matrix
- solve         Solve a linear system of equations
- det           Determinant of a square matrix
- lstsq         Solve linear least-squares problem
- pinv          Pseudo-inverse (Moore-Penrose)...
- matrix_power Integer power of a square matrix
Eigenvalues and decompositions:
- eig           Eigenvalues and vectors of a square matrix
- eigh          Eigenvalues and eigenvectors of a Hermitian matrix
- eigvals       Eigenvalues of a square matrix
- eigvalsh      Eigenvalues of a Hermitian matrix
- qr            QR decomposition of a matrix
- svd           Singular value decomposition of a matrix
- cholesky      Cholesky decomposition of a matrix
Tensor operations:
- tensorsolve    Solve a linear tensor equation
- tensorinv     Calculate an inverse of a tensor
...
```

Autres paquets de numpy

```
>>> help(numpy)
...
doc
    Topical documentation on broadcasting, indexing, etc.
lib
    Basic functions used by several sub-packages.
random
    Core Random Tools
linalg
    Core Linear Algebra Tools
fft
    Core FFT routines
polynomial
    Polynomial tools
testing
    Numpy testing tools
f2py
    Fortran to Python Interface Generator.
distutils
    Enhancements to distutils with support for
        Fortran compilers support and more.
...
...
```

Autres paquets de numpy

Random

- ▶ Sous paquet random : générateurs de nombres aléatoires.

```
>>> numpy.random.seed(0)
>>> A = numpy.random.randn(2, 3, 4)
>>> print(A)
[[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]

 [[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.31306777 -0.85409574]
 [-2.55298982  0.6536186   0.8644362   -0.74216502]]]
```

Scipy library

Librairie scientifique

- ▶  <http://www.scipy.org/scipylib/index.html>
- ▶ *It provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization.*

```
>>> import scipy
>>> help(scipy)
...
cluster                               --- Vector Quantization / Kmeans
fftpack                                --- Discrete Fourier Transform algorithms
integrate                             --- Integration routines
interpolate                            --- Interpolation Tools
io                                     --- Data input and output
linalg                                 --- Linear algebra routines
linalg.blas                            --- Wrappers to BLAS library
linalg.lapack                           --- Wrappers to LAPACK library
misc                                    --- Various utilities that don't have
                                         another home.
ndimage                                --- n-dimensional image package
odr                                     --- Orthogonal Distance Regression
optimize                               --- Optimization Tools
signal                                  --- Signal Processing Tools
...
...
```

- ▶ Optimisation, Intégration, Interpolation, Algèbre linéaire, Algèbre linéaire creuse, Signal, Image, Statistiques, Fonctions spéciales (Γ , ψ)...

```
...  
sparse                      --- Sparse Matrices  
sparse.linalg                 --- Sparse Linear Algebra  
sparse.linalg.dsolve          --- Linear Solvers  
sparse.linalg.dsolve.umfpack  --- :Interface to the UMFPACK library:  
                                Conjugate Gradient Method (LOBPCG)  
sparse.linalg.eigen            --- Sparse Eigenvalue Solvers  
sparse.linalg.eigen.lobpcg    --- Locally Optimal Block Preconditioned  
                                Conjugate Gradient Method (LOBPCG)  
spatial                      --- Spatial data structures and algorithms  
special                      --- Special functions  
stats                        --- Statistical Functions  
...
```

Scipy

lecture de fichiers Matlab

- ▶ Exemple, lectures de fichiers Matlab dans le subpackage io (scipy.io)

```
>>> import scipy.io
>>> data = scipy.io.loadmat('file.mat')
>>> data
{ '__header__': b'MATLAB 5.0 MAT-file, Platform: MACI64,
    Created on: Tue Nov 14 14:40:01 2017', '__version__': '1.0',
  'B': array([[ 1,  2],
              [11, 21]], dtype=uint8), 'A': array([[ 1,  2,  3],
              [11, 12, 13]], dtype=uint8), '__globals__': []}
>>> data['A']
array([[ 1,  2,  3],
       [11, 12, 13]], dtype=uint8)
```

Matplotlib

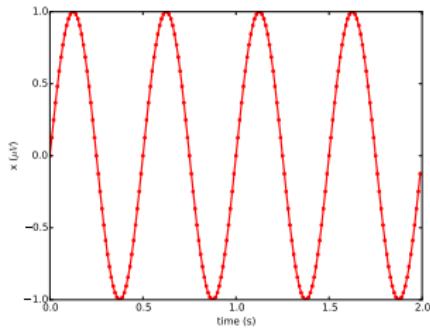
- ▶  <http://www.matplotlib.org>
- ▶ *matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.*
- ▶ Contient des classes : programmation orientée objets avec différents backends pour différentes interfaces graphiques, graphical user interfaces (GUI) : agg, gtk, qt, svg, ps, pdf...
- ▶ Contient des procédures pour faciliter l'accès à ces classes .

Matplotlib

pyplot

- ▶ Fonctions procédurales dans le sous-paquet pyplot
(`matplotlib.pyplot`)

```
>>> import matplotlib.pyplot
>>> t = numpy.arange(0, 10, 0.01)
>>> x = numpy.sin(2 * numpy.pi * 3 * t)
>>> matplotlib.pyplot.plot(t, x)
>>> matplotlib.pyplot.xlabel("time (s)")
>>> matplotlib.pyplot.ylabel("x ($\mu V$)")
>>> matplotlib.pyplot.show()
```



Matplotlib

Saving figures

- ▶ Sauvegarde manuelle à partir de la fenêtre ouverte sur l'icône save.
- ▶ sauvegarde en ligne de commande (`savefig`) sans passage par un affichage à l'écran.

```
...>>> matplotlib.pyplot.ylabel("x ($\mu V$)")  
>>> # matplotlib.pyplot.show()  
>>> matplotlib.pyplot.savefig("./sinus.ps")  
>>> matplotlib.pyplot.savefig("./sinus.pdf")  
>>> matplotlib.pyplot.savefig("./sinus.svg")  
>>> matplotlib.pyplot.savefig("./sinus.tiff")  
>>> matplotlib.pyplot.savefig("./sinus.png")  
>>> matplotlib.pyplot.savefig("./sinus.jpg")
```

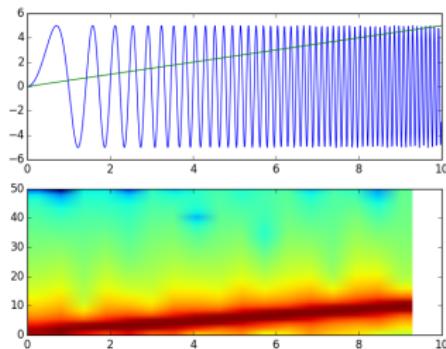
Matplotlib

Pylab

- ▶ Fonctions à la Matlab dans le sous-paquet pylab (matplotlib.pyplot)
- ▶ Beaucoup de fonctions (≈ 973) sous formes abrégées . . .

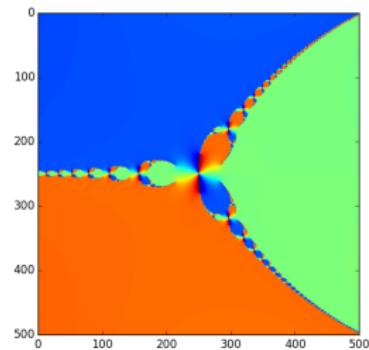
```
>>> import matplotlib.pyplot as pylab
>>> len(dir(pylab))
973

>>> from pylab import *
>>> t = arange(0, 10, 0.01)
>>> f = arange(0, 5, 0.005)
>>> x = sin(2 * pi * f * t)
>>> subplot(2, 1, 1)
>>> plot(t, x * 5)
>>> plot(t, f)
>>> subplot(2, 1, 2)
>>> res = specgram(x, NFFT=64,
... Fs=100, noverlap=8)
>>> show()
```



Matplotlib Pylab

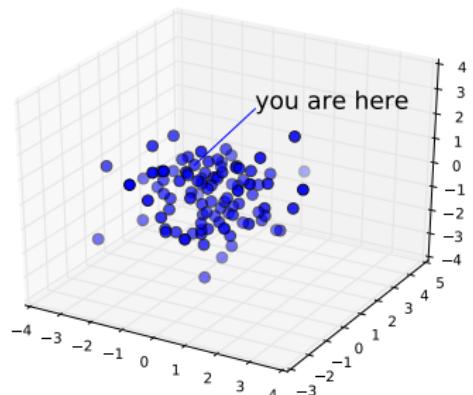
```
>>> (X, Y) = meshgrid(linspace(-2, 2,  
... 500), linspace(-2, 2, 500))  
>>> Z = X + Y * 1j  
>>> for k in range(76):  
... Z -= (Z / 3 - 1) / (3 * Z ** 2)  
>>> close("all")  
>>> imshow(angle(Z))  
>>> # savefig('/MonChemin/Lenom.pdf')  
>>> show()
```



Matplotlib

Exemples en 3D

```
>>> import matplotlib.pyplot as plt
>>> from mpl_toolkits.mplot3d import \
...     Axes3D
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, \
...     projection='3d')
>>> X = numpy.random.randn(100, 3)
>>> ax.scatter(X[:,0], X[:,1], X[:,2])
>>> ax.plot([X[0,0], X[0,0]+1], \
...           [X[0,1], X[0,1]+1], [X[0,2], \
...           X[0,2]+1])
>>> ax.text(X[0,0]+1.2, X[0,1]+1.2, \
...           X[0,2]+1.2, 'you are here', \
...           fontsize=20)
>>> plt.savefig('plot3D.pdf')
```



Pandas

- ▶  <http://www.scipy.org/scipylib/index.html>
- ▶ *data structure and analysis*
- ▶ ressemble au traitement de données sous R (data.frame)

```
>>> import numpy as np
>>> import pandas as pd
>>> A = np.array([[141, 70, 'M'], [176, 75, 'F'], [164, 66, 'M']])
>>> df = pd.DataFrame(A)
>>> print(df)
   0   1   2
0  141  70  M
1  176  75  F
2  164  66  M
>>> l_ind = ['riri', 'fifi', 'loulou']
>>> l_col = ['taille', 'masse', 'genre']
>>> df = pd.DataFrame(A, index=l_ind, columns=l_col)
>>> print(df)
     taille  masse  genre
riri      141      70    M
fifi      176      75    F
loulou    164      66    M
```

Pandas

- ▶ Facilités pour traiter les données
- ▶ sélection d'une colonne : df['...']
- ▶ sélection d'une ligne : df.loc['...']
- ▶ ou par indices, slicing etc : df.iloc[i, j], df.iloc[i:j,k:l]

```
>>> df[['masse']]
riri      70
fifi      75
loulou    66
Name: masse, dtype: object
>>> df.loc['riri']
df.loc['riri']
taille    141
masse     70
genre      M
Name: riri, dtype: object
>>> df.iloc[0,0]
'141'
>>> df.iloc[0:2, 1:2]
   masse
riri    70
fifi    75
```

► Extraction de données

```
>>> df.values
array([[141, 70, 'M'],
       [176, 75, 'F'],
       [164, 66, 'M']], dtype=object)
>>> df[df['masse'].astype('int')>67][['masse', 'taille']].values
array([[70, 141],
       [75, 176]], dtype=object)
```

- ▶ Beaucoup de méthodes pour traiter les tableaux de données.

```
>>> taille = 170 + 10 * np.random.randn(100)
>>> masse = 70 + 10 * np.random.randn(100)
>>> dico = {'taille': taille, 'masse': masse}
>>> df = pd.DataFrame(dico)
>>> df.describe()
      masse        taille
count  100.000000  100.000000
mean   70.507943  169.915690
std    9.999280   10.264489
min    45.467455  144.149219
25%    63.300037  162.736237
50%    70.043825  171.168028
75%    78.153612  176.164359
max    94.118966  201.781565
```

- ▶ Beaucoup de méthodes pour traiter les tableaux de données.

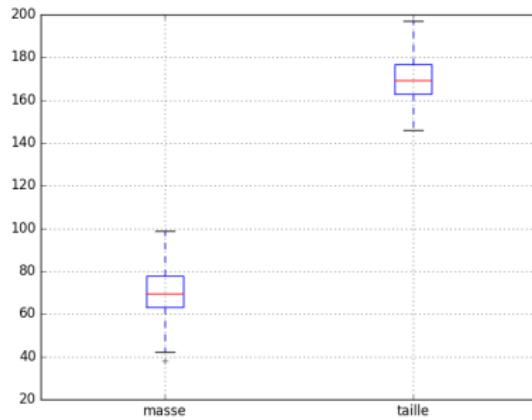
```
>>> df.sort('masse').head(5)
      masse      taille
7    38.125336  173.929789
10   42.258867  168.138387
73   52.573235  169.219885
93   53.767518  179.082610
70   54.847734  155.914292
>>> df.sort('masse').tail(5)
      masse      taille
48   88.451660  152.918614
89   89.600090  169.169344
20   91.230047  195.892489
83   92.329533  156.447108
25   98.779502  173.894076
```

Pandas

Tracés

- ▶ Facilités pour afficher les données.

```
>>> df.boxplot()
```



Pandas

Series

- ▶ Séries (Series)
- ▶ Possibilité d'indexation spéciale.
- ▶ DataFrame : colonnes de series

```
>>> help(pd.Series)
class Series(Series)
| One-dimensional ndarray with axis labels (including time series).
>>> t = np.arange(0, 3, 0.1)
>>> x = np.cos(2 * np.pi * 2 * t)
>>> s = pd.Series(x, t)
>>> print(s[0.2:0.5])
0.2    -0.809017
0.3    -0.809017
0.4     0.309017
0.5     1.000000
```

Pandas

Series

- ▶ Possibilité d'indexation spéciale.

```
>>> t1 = np.arange(0, 3, 0.3)
>>> t2 = np.arange(1, 2, 0.2)
>>> x1 = np.cos(2 * np.pi * 0.01 * t1)
>>> x2 = np.cos(2 * np.pi * 0.03 * t2)
>>> s1 = pd.Series(x1, t1)
>>> s2 = pd.Series(x2, t2)
>>> df = pd.DataFrame({'x1':s1, 'x2':s2})
>>> print(df)
      x1      x2
0.0  1.000000    NaN
0.3  0.999822    NaN
0.6  0.999289    NaN
0.9  0.998402    NaN
1.0    NaN  0.982287
1.2  0.997159  0.974527
1.4    NaN  0.965382
1.5  0.995562    NaN
1.6    NaN  0.954865
1.8  0.993611  0.942991
2.1  0.991308    NaN
2.4  0.988652    NaN
2.7  0.985645    NaN
>>> print(df.fillna(method='ffill'))
      x1      x2
0.0  1.000000  0.982287
0.3  0.999822  0.974527
0.6  0.999289  0.965382
0.9  0.998402  0.954865
1.0  0.998402  0.982287
1.2  0.997159  0.974527
1.4  0.997159  0.965382
1.5  0.995562  0.965382
1.6  0.995562  0.954865
1.8  0.993611  0.942991
2.1  0.991308  0.942991
2.4  0.988652  0.942991
2.7  0.985645  0.942991
```

- ▶ **IP[y]:** IPython
Interactive Computing <http://ipython.org>
- ▶ *Enhanced python console*
- ▶ Attention, ce n'est pas un paquet mais une console améliorée !
- ▶ Accessible à partir d'un terminal (ipython)
- ▶ Aide simplifiée

```
$ ipython
```

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.1 -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.
```

- ▶ Complétion automatique avec la touche tab.
- ▶ Accès rapide à l'historique avec ↑ et ↓.
- ▶ Fonctions magiques (magic functions) :
 - ▶ % line oriented
 - ▶ %% cell oriented

```
In [1]: g = lambda x: cos(x ** 2)
In [2]: In [21]: %time g(10)
CPU times: user 41 µs, sys: 1 µs, total: 42 µs
Wall time: 46 µs
Out[2]: 0.862318872287684
In [3]: %%time
...: a = 10
...: g(a)
...
CPU times: user 42 µs, sys: 1 µs, total: 43 µs
Wall time: 47 µs
```

```
In [4]: magic # or \%magic
IPython's 'magic' functions
=====
```

The magic function system provides a series of functions which allow you to control the behavior of IPython itself, plus a lot of system-type features. There are two kinds of magics, line-oriented and cell-oriented.

- ▶ exécution de modules Python (%run) + debugger + profiler (see %run? for details).
- ▶ accès aux fonctions du système.
- ▶ rechargement dynamique des modules changés à l'extérieur.

```
In [1]: %run mod1.py

In [2]: %cd ./Documents/
/Volumes/HD1/Users/becqg/Documents
In [3]: %%system
...: pwd
...:
Out[3]: ['/Volumes/HD1/Users/becqg/Documents']

In [4]: %load_ext autoreload
In [5]: %import mod1
In [6]: %autoreload 1
In [4]: import importlib as imp
In [5]: imp.reload(mod1)
```

- ▶ import des fonctions pylab amélioré.

```
In [1]: %pylab
Using matplotlib backend: MacOSX
Populating the interactive namespace from numpy and matplotlib

In [2]: A = empty((3,3))

In [3]: %pylab?
...
%pylab makes the following imports::

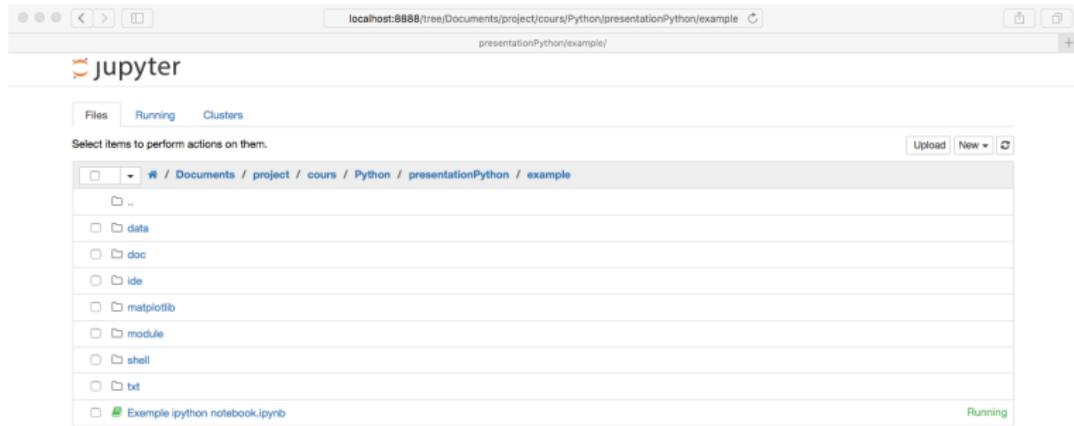
    import numpy
    import matplotlib
    from matplotlib import pylab, mlab, pyplot
    np = numpy
    plt = pyplot

    from IPython.display import display
    from IPython.core.pylabtools import figsize, getfigs

    from pylab import *
    from numpy import *
```

IPython notebook

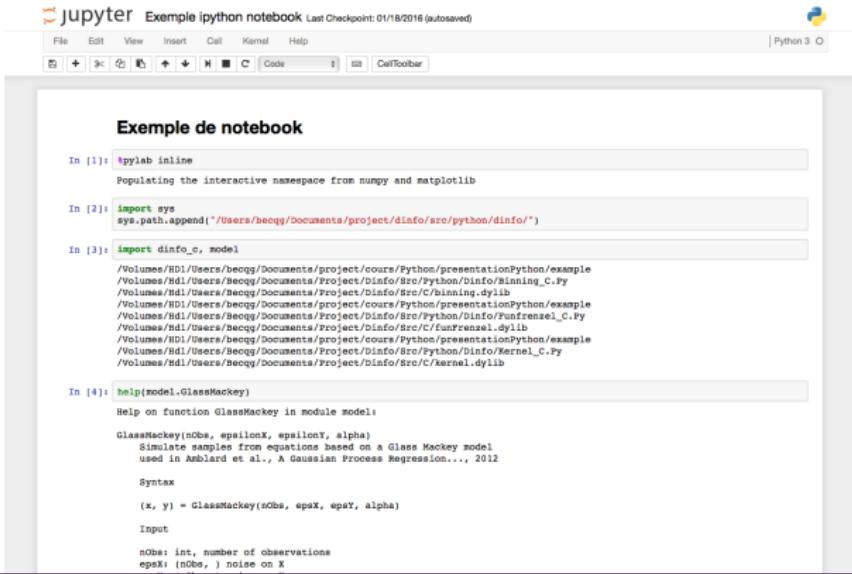
- ▶ Accessible à partir d'un terminal (*ipython notebook*, *jupyter notebook*)
- ▶ Ouverture et création de notebooks (**.ipynb*)



Jupyter

jupyter notebook

- ▶ Les cellules exécutent du code ou formatent du texte
- ▶ Chargement des fonctions pylab avec affichage dans le notebook : %pylab inline



The screenshot shows a Jupyter Notebook window titled "Exemple ipython notebook". The interface includes a toolbar with file operations like File, Edit, View, Insert, Cell, Kernel, Help, and a CellToolbar. A status bar at the bottom indicates "Python 3" and "Last Checkpoint: 01/18/2016 (autosaved)".

The notebook contains four code cells:

- In [1]: `%pylab inline`
Populating the interactive namespace from numpy and matplotlib
- In [2]: `import sys
sys.path.append("/Users/becqg/Documents/project/dinfo/src/python/dinfo")`
- In [3]: `import dinfo.c, model`
A long list of module paths from /Volumes/Hd1/Users/becqg/Documents/project/cours/Python/presentationPython/example to /Volumes/Hd1/Users/becqg/Documents/project/dinfo/src/c/kernel_dylib
- In [4]: `help(model.GlassMackey)`
Help on function GlassMackey in module model:

GlassMackey(nObs, epsilonX, epsilonY, alpha)
Simulate samples from equations based on a Glass Mackey model
used in Ambard et al., A Gaussian Process Regression..., 2012

Syntax

`(x, y) = GlassMackey(nObs, epsX, epsY, alpha)`

Input

`nObs: int, number of observations`
`epsX: (nObs,) noise on X`



Jupyter

jupyter notebook

- ▶ Saisie et affichage en HTML et LaTeX, entre autres.

The screenshot shows a Jupyter Notebook interface with the title "Jupyter Exemple ipython notebook Last Checkpoint: 01/18/2016 (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Help, and CellToolbar. A Python 3 logo is in the top right. The code cell contains:

```
alpha: 0.05, dxti:-102.23, dtxi:-43.26  
alpha: 0.01, dxti:-79.77, dtxi:-44.55  
alpha: 0.10, dxti:-83.92, dtxi:-41.02  
alpha: 0.20, dxti:-24.98, dtxi:-16.54
```

The system is given by this equation:

$$\begin{cases} x_t = x_{t-1} - 0.4 \left(x_{t-1} - \frac{x_{t-5}}{1 + x_{t-5}^2} \right) y_{t-5} + 0.3 y_{t-3} + e_{1,t} \\ y_t = y_{t-1} - 0.4 \left(y_{t-1} - \frac{y_{t-5}}{1 + y_{t-5}^2} \right) + \alpha x_{t-2} + e_{2,t} \end{cases}$$

In [5]:

```
nObs = 10000  
(x, y) = model.GlassMackey(nObs, 0.03 * randn(nObs), 0.02 * randn(nObs), 0.1)
```

In [6]:

```
plot(x, y, '^')
```

Out[6]:

```
<matplotlib.lines.Line2D at 0x108c47550>
```

A scatter plot showing two clusters of data points (blue diamonds) on a coordinate system from 0.5 to 2.2 on both axes.

In [7]:

```
mi_xy = dinfo_c.mi(x, y, "Frenzel", (20, "Euclidean"))
```

In [8]:

```
print(mi_xy)
```

```
0.8649664124647107
```

Jupyter

- ▶ <https://jupyter.org>
- ▶ Interface avec d'autres langages de programmation.



Project Jupyter exists to develop open-source software, open-standards, and services
for interactive computing across dozens of programming languages.

Scikit-learn

- ▶ <http://scikit-learn.org/>
- ▶ <https://github.com/scikit-learn/scikit-learn>

The screenshot shows the official scikit-learn website. At the top, there's a navigation bar with links for Home, Installation, Documentation, Examples, a search bar, and a custom search button. Below the header is a banner featuring a grid of 25 small plots related to machine learning, followed by a large title "scikit-learn" and the subtitle "Machine Learning in Python". To the right of the title is a bulleted list of features: Simple and efficient tools for data mining and data analysis; Accessible to everybody, and reusable in various contexts; Built on NumPy, SciPy, and matplotlib; and Open source, commercially usable - BSD license. The main content area is divided into six sections: Classification, Regression, Clustering, Dimensionality reduction, Model selection, and Preprocessing. Each section has a brief description, a list of applications, a list of algorithms, and a "Examples" link.

Classification	Regression	Clustering
Identifying to which category an object belongs to. Applications: Spam detection, image recognition. Algorithms: SVM, nearest neighbors, random forest, ... — Examples	Predicting a continuous-valued attribute associated with an object. Applications: Drug response, Stock prices. Algorithms: SVR, ridge regression, Lasso, ... — Examples	Automatic grouping of similar objects into sets. Applications: Customer segmentation, Grouping experiment outcomes Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples
Dimensionality reduction	Model selection	Preprocessing
Reducing the number of random variables to consider. Applications: Visualization, Increased efficiency Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples	Comparing, validating and choosing parameters and models. Goal: Improved accuracy via parameter tuning Modules: grid search, cross validation, metrics. — Examples	Feature extraction and normalization. Application: Transforming input data such as text for use with machine learning algorithms. Modules: preprocessing, feature extraction. — Examples



Scikit-learn

Help content

```
>>> import sklearn
>>> help(sklearn)

DESCRIPTION
Machine learning module for Python
=====
sklearn is a Python module integrating classical machine
learning algorithms in the tightly-knit world of scientific Python
packages (numpy, scipy, matplotlib).

It aims to provide simple and efficient solutions to learning problems
that are accessible to everybody and reusable in various contexts:
machine-learning as a versatile tool for science and engineering.

See http://scikit-learn.org for complete documentation.

PACKAGE CONTENTS
__check_build (package)
_build_utils
_isotonic
base
calibration
cluster (package)
covariance (package)
cross_decomposition (package)
cross_validation
```

Scikit-learn

Help content

```
datasets (package)
decomposition (package)
discriminant_analysis
dummy
ensemble (package)
externals (package)
feature_extraction (package)
feature_selection (package)
gaussian_process (package)
grid_search
isotonic
kernel_approximation
kernel_ridge
lda
learning_curve
linear_model (package)
manifold (package)
metrics (package)
mixture (package)
multiclass
naive_bayes
neighbors (package)
neural_network (package)
pipeline
preprocessing (package)
qda
random_projection
semi_supervised (package)
setup
svm (package)
tests (package)
tree (package)
utils (package)
```

Scikit-learn

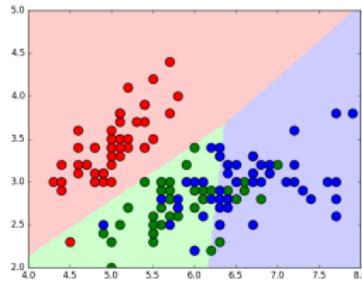
Exemple de Classifieur : Linear Discriminant Analysis

```
>>> from pylab import *
>>> from sklearn.datasets import load_iris
>>> from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
>>> dic = load_iris()
>>> x = dic["data"]
>>> y = dic["target"]
>>> nClasses = unique(y).size
>>> clf = LDA()
>>> clf.fit(x[:, :2], y)
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                           solver='svd', store_covariance=False, tol=0.0001)
```

Scikit-learn

Exemple de Classifieur : Linear Discriminant Analysis

```
>>> from matplotlib import colors
>>> cmap = colors.LinearSegmentedColormap.from_list('3_classes',
[(1, 0.8, 0.8), (0.8, 1, 0.8), (0.8, 0.8, 1)], N=3)
>>> cm.register_cmap(cmap=cmap)
>>> (mx1, mx2) = meshgrid(linspace(4, 8, 100), linspace(2, 5, 100))
>>> Z = clf.predict(vstack((mx1.flatten().T, mx2.flatten().T)).T)
>>> pcolor(mx1, mx2, Z.reshape(mx1.shape), cmap='3_classes')
>>> colors = ["r", "g", "b"]
>>> for i in range(nClasses):
...     scatter(x[y==i, 0], x[y==i, 1], marker='o', edgecolor='k', linewidths=1,
c=colors[i], s=150)
>>> axis(xmin=4, xmax=8, ymin=2, ymax=5); show()
```



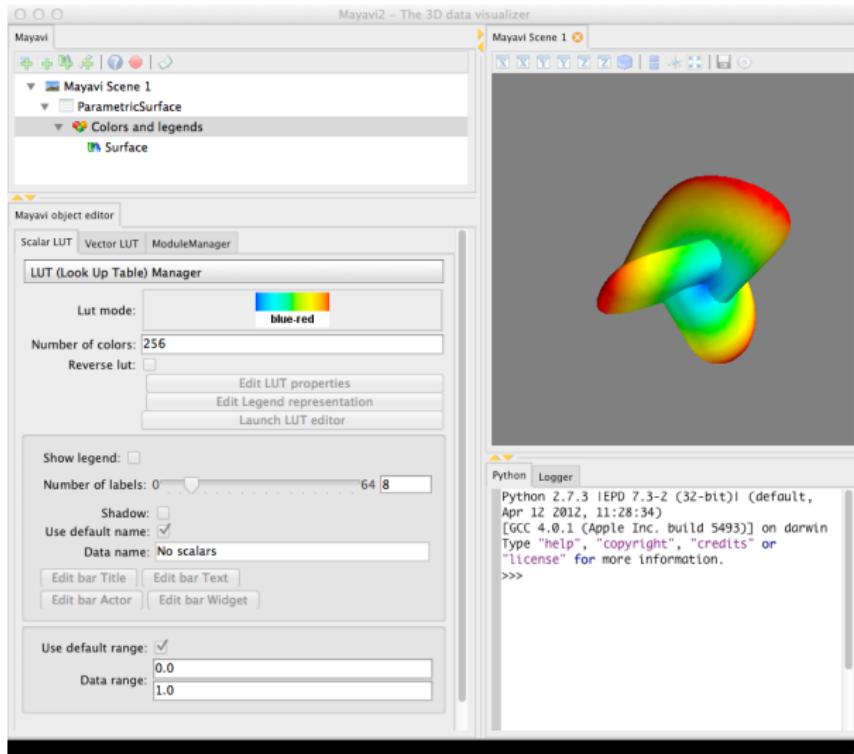
Mayavi

- ▶ <http://docs.enthought.com/mayavi/mayavi/>
- ▶ <https://github.com/enthought/mayavi>
- ▶ Manipulation des objets 3D améliorée (objet VTK).

The screenshot shows the Mayavi 4.5.0 documentation page. The header reads "Mayavi: 3D scientific data visualization and plotting in Python". The main content area features a large image of a 3D visualization of a blue and yellow object, with smaller images showing the application interface and code snippets. The sidebar on the left contains links for "Next topic", "This Page", "Show Source", "Quick search" (with a search bar and "Go" button), "Google Search" (with a search bar and "Go" button), and "Citing Mayavi" (with a note about citation requirements). The footer at the bottom of the page says "User guide: full table of contents".



Mayavi



Mayavi

```
import mayavi.engine
```

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

```
>>> 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 fonctions écrites en C

Exemple using module ctypes

```
import ctypes
libName = './clz.lib'
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

Introduction

Description du Langage

Description des Paquets Scientifiques

Distributions et Environnements de travail

Conclusion

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

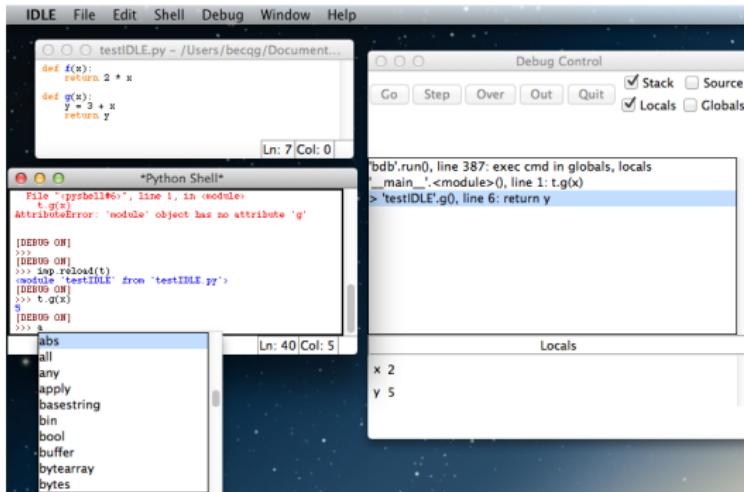
```
becqg — Python — 71x15
desktop-154:~ becqg$ python
Enthought Python Distribution -- www.enthought.com
Version: 7.3-2 (32-bit)

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.
>>> (3 + 2.5 * 10) * 4 / 100
1.12
>>> [
```

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, debuggeur, ...



Enthought Canopy

- ▶  ENTHOUGHT :
 - <https://www.enthought.com/products/canopy/>
 - ▶ Contient Python et +100 librairies orientées applications scientifiques.
 - ▶ Canopy express *free license for all users*.

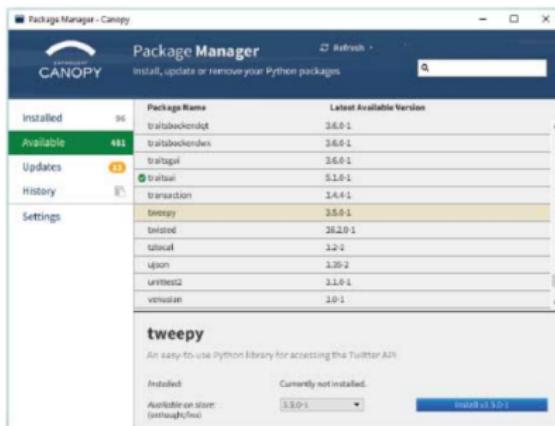
Enthought Canopy



<https://www.enthought.com/products/canopy/>

- ▶ *Easy installation and update.*

Graphical Package Manager with Integrated Dependency Solver and Version Rollback



Enthought Canopy



<https://www.enthought.com/products/canopy/>

- + Jupyter notebook + Canopy editor.

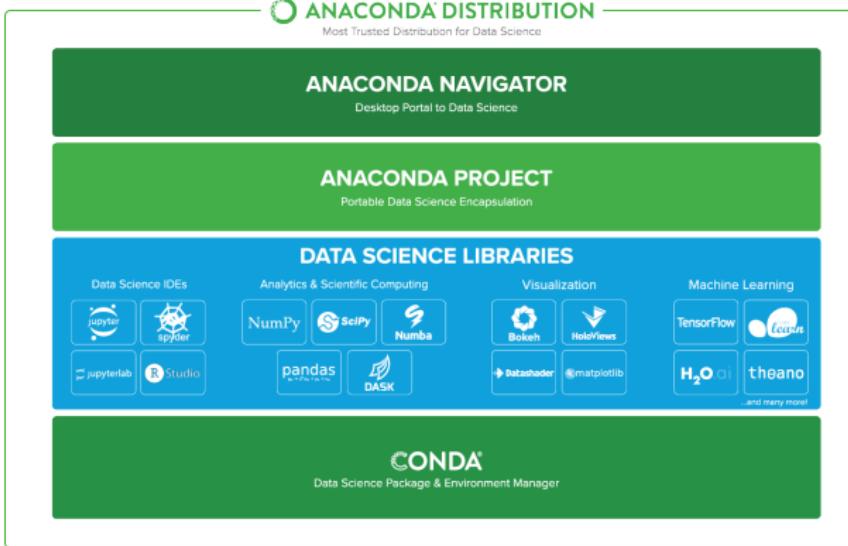
The screenshot shows the Spyder IDE interface with the following details:

- File Browser:** Shows the current project structure with files like `robnd`, `histogram.py`, `streamplot.py`, and `hellocpp`.
- Code Editor:** Displays the `histogram.py` script. The script imports `matplotlib.patches` as `patches`, defines a `main` function, and uses `np.random` to generate data for a histogram with 100 bins. It then creates subplots for the histogram and a scatter plot. The main part of the script uses a `for` loop to iterate over the data and draw rectangles with a blue fill and black edges. Finally, it shows the plot.
- Terminal:** Shows the command `$ ipdb> continue` and the output of the script, which includes the histogram plot and some numerical values.
- Debug Variable Browser:** Shows variables `ax`, `data`, and `fig`. `ax` is an `AxesSubplot` object, `data` is a `float64[100]` array, and `fig` is a `Figure` object.
- Stack Browser:** Shows the call stack with `main` at the top level.
- Breakpoint Viewer:** Shows breakpoints set on line 23 and line 50 of `histogram.py`.



Anaconda

►  <https://www.anaconda.com>



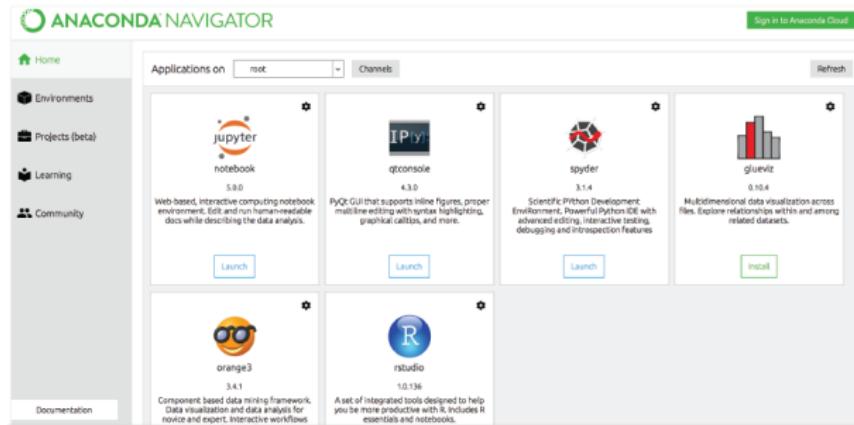
The screenshot displays the Anaconda Distribution website's main features:

- ANACONDA NAVIGATOR**: Desktop Portal to Data Science.
- ANACONDA PROJECT**: Portable Data Science Encapsulation.
- DATA SCIENCE LIBRARIES**: A collection of tools categorized into four groups:
 - Data Science IDEs: Jupyter, Spyder, jupyterlab, R Studio
 - Analytics & Scientific Computing: NumPy, SciPy, Numba, pandas, DASK
 - Visualization: Bokeh, Holoviews, DataShader, matplotlib
 - Machine Learning: TensorFlow, Theano, H2O, and many more!
- CONDA**: Data Science Package & Environment Manager.



Anaconda

▶  ANACONDA <https://www.anaconda.com>



The screenshot shows the Anaconda Navigator interface. On the left, there's a sidebar with links to Home, Environments, Projects (beta), Learning, Community, and Documentation. The main area displays a grid of application icons:

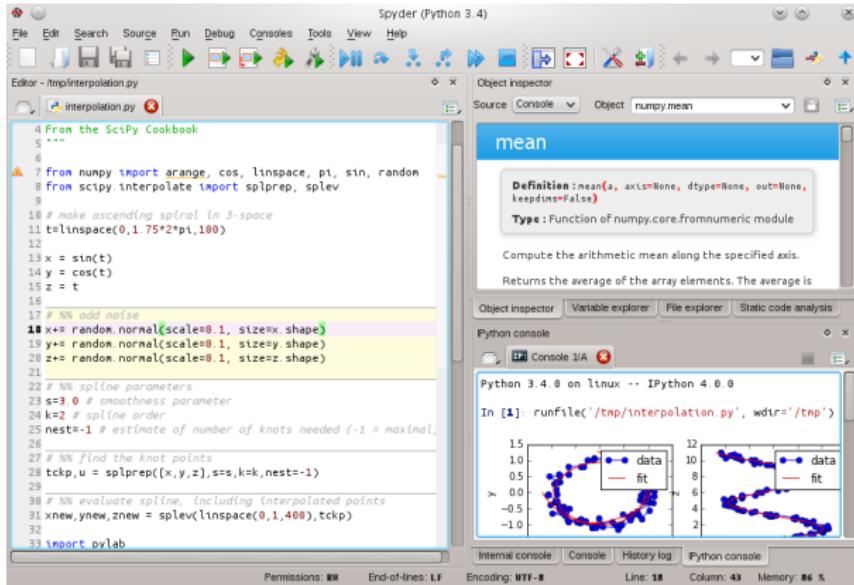
- jupyter notebook**: Version 5.8.0. Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
- qtconsole**: Version 4.3.0. PyQt GUI that supports inline figures, proper multi-line editing with syntax highlighting, graphical cellbars, and more.
- spyder**: Version 3.1.4. Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features.
- glueviz**: Version 0.16.4. Multi-dimensional data visualization across files. Explore relationships within and among related datasets.
- orange3**: Version 3.4.1. Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows.
- rstudio**: Version 1.0.136. A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Each application card has a "Launch" button (except for glueviz) and an "Install" button (for glueviz). There are also "Channels" and "Refresh" buttons at the top right of the main area.



Spyder

- ▶ Spyder (multiplateforme) : environnement de type Matlab pour Python.
- ▶ <https://github.com/spyder-ide/spyder>

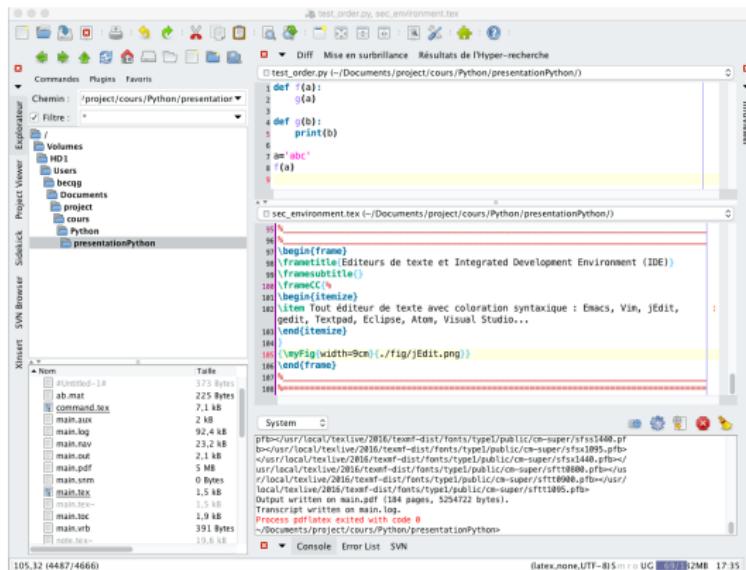


Autres distributions

- ▶ SageMath  <http://www.sagemath.org>
- ▶ PythonXY  <https://code.google.com/p/pythonxy/>
- ▶ ?

Editeurs de texte et Integrated Development Environment (IDE)

- ▶ Tout éditeur de texte avec coloration syntaxique : Emacs, Vim, jEdit, gedit, Texpad, Eclipse, Atom, Visual Studio...



Outline

Introduction

Description du Langage

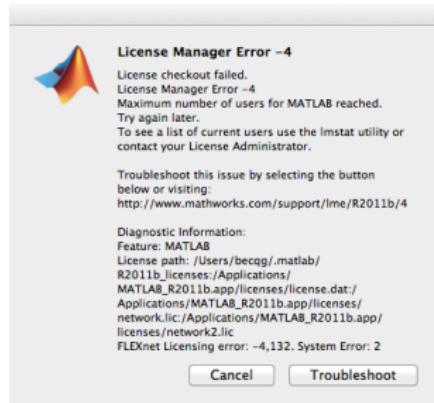
Description des Paquets Scientifiques

Distributions et Environnements de travail

Conclusion

Python vs Matlab

- ▶ Outils équivalents : matrices vs ndarray, console, script, graphisme, GUI, cell mode vs jupyter notebook
- ...
- ▶ Matlab, Matrix Laboratory, a des bibliothèques d'algèbre linéaire plus rapide que Numpy ou Scipy (sauf avec certaines distributions payantes).
- ▶ 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.
- ▶ Votre code en Python peut être utilisé gratuitement.





Outline

classe et variable 'self'

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
```

Transposition NDarray exemple 3D

- ▶ A.shape (1, 2, 3, 4)
- ▶ A.T.shape (4, 3, 2, 1) by default.
- ▶ can set the axes order. see help(numpy.matrix.transpose)

```
>>> A = numpy.array([[[[1, 2, 3, 4], [11, 12, 13, 14], [21, 22, 23, 24]],  
    [[101, 102, 103, 104], [111, 112, 113, 114], [121, 122, 123, 124]]]])  
  
>>> A  
array([[[[ 1,  2,  3,  4],  
        [ 11, 12, 13, 14],  
        [ 21, 22, 23, 24]],  
  
       [[101, 102, 103, 104],  
        [111, 112, 113, 114],  
        [121, 122, 123, 124]]]])  
  
>>> A.T  
array([[[[ 1],  
        [101]],  
        ...  
       [[ 24],  
        [124]]]])
```

Identity of an object

- ▶ Return the identity of an object. This is guaranteed to be unique among simultaneously existing objects. (CPython uses the object's memory address.)

```
>>> a = 123
>>> id(a)
4297541856
>>> id(123)
4297541856
```

Order of definition in a module

- ▶ L'ordre de définition des fonctions n'est pas important à partir du moment où lors de leurs utilisations, celles-ci ont été définies. C'est particulièrement fréquents dans les modules.

```
>>> def f(a):
...     g(a)
>>> def g(b):
...     print(b)
>>> a='abc'
>>> f(a)
abc
```

Order of definition in a module

- ▶ l'ordre de définition des fonctions n'est pas important à partir du moment où lors de l'utilisation es focntions ont été définies.
Fonctionne dans la console ou dans un module.

```
>>> def f(a):
>>>     g(a)
>>> def g(b):
>>>     print(b)
>>> a='abc'
>>> f(a)
abc
```

slice and ellipsis

► l'ellipse

```
>>> A = arange(24).reshape(2, 3, 4)
>>> A
array([[[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]],

      [[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]]])
>>> A[:,0]
array([[ 0,  1,  2,  3],
       [12, 13, 14, 15]])
>>> A[...,:,0]
array([[ 0,  4,  8],
       [12, 16, 20]])
```

slice and ellipsis

- ▶ $A[i, \dots]$ est identique à $A[i, :, :, \dots]$.
- ▶ $A[i, :]$ est identique à $A[i, :, :, \dots]$, donc identique à ...
- ▶ $A[:, i, :] = A[\dots, i, :]$ dans l'exemple mais
- ▶ $A[\dots, 0]$ est différent de $A[:, 0]$ car ici c'est $A[:, 0, :]$ et non $A[:, :, 0]$!

```
>>> A = arange(12).reshape(2, 3, 2)
>>> A
array([[[ 0,  1],
       [ 2,  3],
       [ 4,  5]],

       [[ 6,  7],
       [ 8,  9],
       [10, 11]]])
>>> A[:,0]
array([[0, 1],
       [6, 7]])
>>> A[...,:,0]
array([[0, 2, 4],
       [6, 8, 10]])
```

Zen of Python - Pythonic

Thanks to Branislav Gerazov (Branko) in regard to his friendly mail about this presentation: *... one thing that I find awesome in Python it's its emphasis on aesthetics and readability. On this note I would add the Zen of Python ('import this') and the concept of being 'pythonic', and contrast it to Perl which is the most awful computer language in the world (I am recoding something now from Perl and that — is really frustrating!). In this sense I've had students often saying they love Python ...*

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.

...
>>> # a pythonic list comprehension for example !
>>> list_of_number = [(i, 2*i, 3*i) for i in range(10)]
>>> list_of_number
[(0, 0, 0), (1, 2, 3), (2, 4, 6), (3, 6, 9), (4, 8, 12), (5, 10, 15),
 (6, 12, 18), (7, 14, 21), (8, 16, 24), (9, 18, 27)]
```

- ▶ Cours d'une collègue Patricia Ladret pour Python via Anaconda: <http://chamilo1.grenet.fr/ujf/courses/FAMILIARISATIONAVECPYTHONSUITEANACON/index.php>