

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

G. Becq

February 6, 2015

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 van 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
- ▶ 1999+ : Worldwide open source



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
- ▶ 1999+ : Worldwide open source



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 (59 ans)
 - ▶ Développeur néerlandais
 - ▶ 1982 : M. Sc
 - ▶ Développeur ABC.
- ▶ Créateur 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

sources : <http://www.wikipedia.org>, <http://www.python.org>

Spécificités

<http://www.python.org/about/>

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

Spécificités

Exemple 1er programme

Dans le fichier "hello.py" :

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

Exécution dans une interface système (terminal, shell) :

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

Spécificités

Exemple shell scripting

Exemple copies des fichiers '.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 et des définitions de fonctions. Ils sont exécutés dans le shell avec la commande "python".

Exemple : "python hello.py"

- ▶ La commande "python" ouvre une console utilisateur (*interpreter*) avec une invite de commande (prompt) caractéristique "> > > "

```
Enthought Canopy Python 2.7.3 | 64-bit | (default, Jun 14 2013, 18:17:36)
[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 noms, attributs, fonctions disponibles : `dir()`

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

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

Utilisations

Utilisation de l'interpréteur

► Besoin d'aide : help()

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

```
Help on built-in function find:
```

```
find(...)
```

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

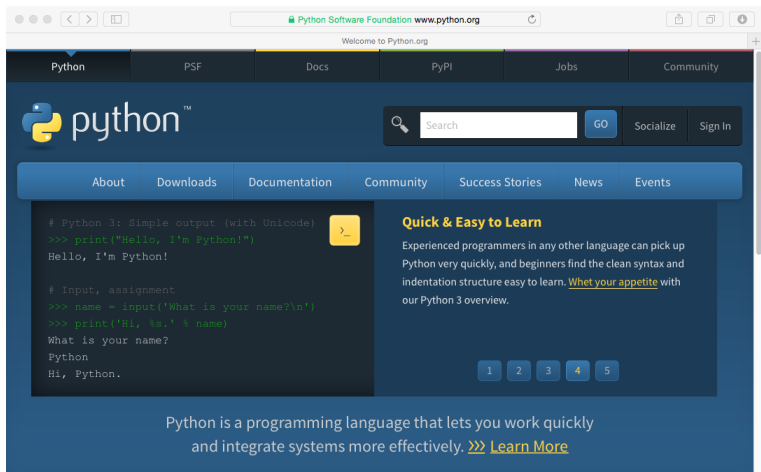
```
    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end]. Optional
    arguments start and end are interpreted as in slice notation.
```

```
    Return -1 on failure.
```

```
(END)
```

Site Web

www.python.org



The screenshot shows the Python Software Foundation website. The top navigation bar includes links for Python, PSF, Docs, PyPI, Jobs, and Community. Below this is a search bar and buttons for Socialize and Sign In. A secondary navigation bar contains links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features a Python 3 code snippet on the left and a 'Quick & Easy to Learn' section on the right. The code snippet demonstrates simple output and input/assignment. The 'Quick & Easy to Learn' section describes Python's syntax and indentation structure, with a link to 'Whet your appetite'. Below this is a row of five numbered buttons (1-5). At the bottom of the main section, a large blue banner states: 'Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)'.

```
# 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.
```

Quick & Easy to Learn

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. [Whet your appetite](#) with our Python 3 overview.

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

Download

Python source code and installers are available for download for all versions! Not sure which version to use? [Check here.](#)

Docs

Documentation for Python's standard library, along with tutorials and guides, are available online.

Jobs

Looking for work or have a Python related position that you're trying to hire for? Our community-run job board is the place to go.

Documentation

Trouver la documentation

Python Software Foundation docs.python.org/2/

Overview — Python 2.7.9 documentation

Python » 2.7.9 » Documentation » modules | index

Download

Download these documents

Docs for other versions

- Python 3.4 (stable)
- Python 3.5 (in development)
- Old versions

Other resources

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

Quick search

Enter search terms or a module, class or function name.

Python 2.7.9 documentation

Welcome! This is the documentation for Python 2.7.9, last updated Jan 13, 2015.

Parts of the documentation:

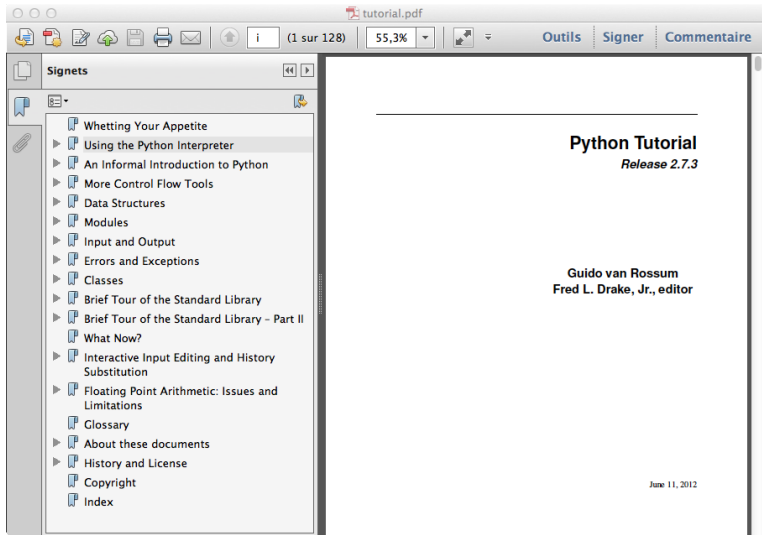
- What's new in Python 2.7?**
or all "What's new" documents since 2.0
- Tutorial**
start here
- Library Reference**
keep this under your pillow
- Language Reference**
describes syntax and language elements
- Python Setup and Usage**
how to use Python on different platforms
- Python HOWTOs**
in-depth documents on specific topics
- Extending and Embedding**
tutorial for C/C++ programmers
- Python/C API**
reference for C/C++ programmers
- Installing Python Modules**
information for installers & sys-admins
- Distributing Python Modules**
sharing modules with others
- FAQs**
frequently asked questions (with answers!)

Indices and tables:

- Global Module Index**
quick access to all modules
- General Index**
all functions, classes, terms
- Glossary**
the most important terms explained
- Search page**
search this documentation
- Complete Table of Contents**
lists all sections and subsections

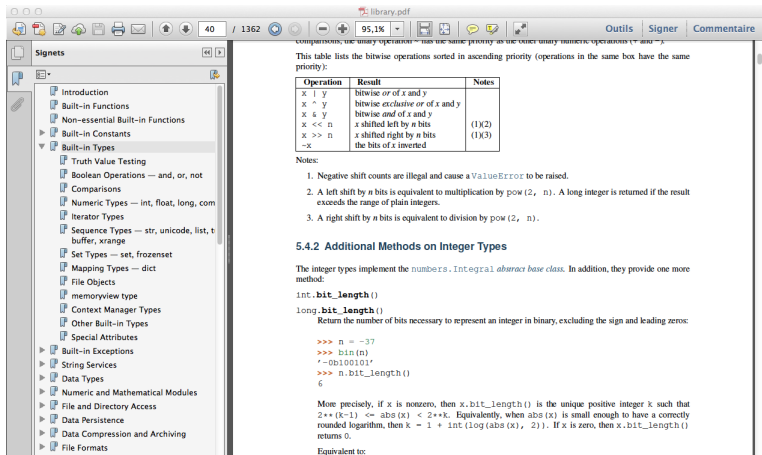
Documentation

Lire la documentation - Tutorial



Documentation

Lire la documentation - Library reference



comparisons, the unary operator `~` has the same priority as the other unary numeric operations (`+` and `-`).

This table lists the bitwise operations sorted in ascending priority (operations in the same box have the same priority):

Operation	Result	Notes
<code>x y</code>	bitwise <i>or</i> of <code>x</code> and <code>y</code>	
<code>x ^ y</code>	bitwise <i>exclusive or</i> of <code>x</code> and <code>y</code>	
<code>x & y</code>	bitwise <i>and</i> of <code>x</code> and <code>y</code>	
<code>x << n</code>	<code>x</code> shifted left by <code>n</code> bits	(1)(2)
<code>x >> n</code>	<code>x</code> shifted right by <code>n</code> bits	(1)(3)
<code>~x</code>	the bits of <code>x</code> inverted	

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 `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 `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 `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

library.pdf

383 / 1362 95,1%

Outils Signer Commentaire

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

reference.pdf

8 (12 sur 121) 95,1%

Outils Signer Commentaire

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

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 window with the URL www.python.org/dev/peps/. The page title is "PEP 0 -- Index of Python Enhancement Proposals (PEPs) | Python.org".

PEP 0 -- Index of Python Enhancement Proposals (PEPs)

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

Introduction

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.

Index by Category

num	title	owner
---	-----	-----
Meta-PEPs (PEPs about PEPs or Processes)		
P 1	PEP Purpose and Guidelines	Warsaw, Hylton, Goodger, Coghlan
P 4	Deprecation of Standard Modules	von Löwis
P 5	Guidelines for Language Evolution	Prescod
P 6	Bug Fix Releases	Aahz, Baxter
P 7	Style Guide for C Code	GvR
P 8	Style Guide for Python Code	GvR, Warsaw,

The PSF

The Python Software Foundation is the organization behind Python. Become a member of the PSF and help advance the software and our mission.

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 inconsistences du langage (ex `print 'a'` vs `print('a')`).
 - ▶ Résultats de la division des entiers ($1/2$: 2.x, = 0 ; 3.x, = 0.5).
 - ▶ ...
- ▶ Peut poser des problèmes :
 - ▶ Paquets portés 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), listes (list), tuples (tuple), dictionnaires (dict), ensembles (set)

Nombres

Entiers

- Attention à la division entière en 2.7.

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

Nombres

Réels

- Utilisation du point pour passer en réels.

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

Nombres

Entiers long vs court

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

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

Nombres

Réels notations scientifiques

- Notation scientifique de type signe, mantisse et exposant.

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

Nombres

Complexes

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

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

Nombres

Booléens

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

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

Nombres

Opérateurs

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

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

Nombres

Opérateurs de comparaisons

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

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

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)
<type '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 (0o) (oct)
- ▶ character (chr) (< 0xFF) ou unicode (unichr) (< 0xFFFF)

```
>>> a = 0x597d
>>> print(a)
22909
>>> a = 0b0101
>>> print(a)
5
>>> a = chr(42)
>>> print(a)
*
>>> b = unichr(0x597D)
>>> print(b)
chinese symbol ni
>>> a = bin(65447)
>>> print(a)
0b101100101111101
>>> type(a)
<type 'str'>
```

Séquences

Chaînes de caractères

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

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

Séquences

Chaînes de caractères

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

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

Séquences

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

```
>>> a = """1. Bisque de pigeonneaux\n... \tPrenez vos pigeonneaux apres qu'ils seront bien nettoyez\n...      & 'Troussez', faites les blanchir,\n...      & Les "empottez" avec un petit brun de fines herbes\n... """\n>>> print(a)\n1. Bisque de pigeonneaux\n\n\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>>> repr(a)\n'\\'1. Bisque de pigeonneaux\\n\\n\\n\\t\\tPrenez vos pigeonneaux apres qu\\'\\'ils seront bien nettoyez\\n\\n\\n\\t& \\\"\\\"Troussez\\\"\\\", faites les blanchir,\\n\\n\\t& Les \"empottez\" avec un petit brun de fines herbes\\n\\n\\'\n>>> str(a)\n\"1. Bisque de pigeonneaux\\n\\n\\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\"
```

Séquences

Chaînes de caractères

- ▶ forme brute (r" " ou r' ' ou r"" "" ou r''' '''')
- ▶ forme unicode (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\\t\\n\\tLe potage de sant\\xc3\\xa9 se fait de chapons...'"
>>> a = u"Potage de santé\n\tLe potage de santé se fait de chapons..."
>>> print(a)
Potage de santé
    Le potage de santé se fait de chapons...
>>> repr(a)
"u'Potage de sant\\xe9\\n\\t\\n\\tLe potage de sant\\xe9 se fait de chapons...',"
>>> print(u"\u4F60\u597D")
```

你好

Séquences

Listes

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

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


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 de matrices à la Matlab.

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

Séquences

Tuples

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

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

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


Syntaxe

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

Syntaxe

Fonctions prédéfinies

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

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

Syntaxe

Fonctions prédéfinies

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

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

déjà rencontrées

Syntaxe

Fonctions prédéfinies

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

<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 ce paramètre dans les éditeurs de texte : tabulation souple émulée avec des espaces ('soft tab' with 4 spaces vs 'hard tab').

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

Syntaxe

Structure de contrôle - if elif else

► blocs conditionnels : if elif else

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

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

Syntaxe

Structure de contrôle - if elif else

► blocs conditionnels : if elif else

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

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

Syntaxe

Structure de contrôle - while

► boucles conditionnelles : while

```
>>> question = "Voulez-vous continuer ? (o, oui, 0, OUI) "
>>> cond = True
>>> reponseOK = {"o", "0", "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, 0, OUI) 'o'
2 fois
Voulez-vous continuer ? (o, oui, 0, OUI) 'oui'
3 fois
Voulez-vous continuer ? (o, oui, 0, 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("problem entrées sorties : " + str(e))
... except TypeError as e :
...     print("problem de types : " + str(e))
... finally:
...     print("Avec ou sans erreurs, on continue")
...
problem de types : cannot concatenate 'str' and 'int' objects
Avec ou sans erreurs, on continue
```

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


Syntaxe

Fonctions - Définition de la fonction

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

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

Syntaxe

Fonctions - passage de paramètres obligatoires

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

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

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

Syntaxe

Fonctions - passage de paramètres optionnels

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

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

Syntaxe

Fonctions - renvoi de valeurs

► renvoi de valeurs : return

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

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

Syntaxe

Fonctions - autres spécificités

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

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

Syntaxe

Objets - Notion de programmation orientée objet

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

```
3 canards : riri, fifi, loulou

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

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

mange(aliments)
  vérification aliments : herbivore, carnivore occasionnel

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

ne fait rien()
```

Syntaxe

Objets : définition en Python

- ▶ Définition d'une nouvelle : "class" classname ":" suite
- ▶ propriétés : variables avec une valeur par défaut.
- ▶ méthodes : 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 espere(self):
...         self.status = "en attente"
...         return 0
... 
```

Syntaxe

Objets : définition en Python

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

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


Syntaxe

Objets : instantiation et usage

- ▶ instantiation : `objet = Classname()`
- ▶ propriétés obtenues par : `objet.property` (cf structure en C)
- ▶ méthodes obtenues par : `objet.method(argument)`

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

Syntaxe

Objets : exemple avec des nombres

- En python toutes les variables sont des objets.

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

Syntaxe

Objets : exemple avec les listes et les tuples

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

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

Syntaxe

Objets : exemple avec les chaînes de caractères

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

Syntaxe

Fichiers

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

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

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

Module

Exécution dans le shell

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

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

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

Module

Importation de code : import

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

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

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

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

Module

Importation de code : import

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

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

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

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

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


Module

Espace : `__name__`

► attribut système : `'__name__'`

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

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

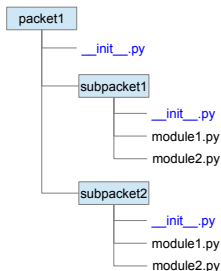
```
>>> import moduleMain
Affichage en import ou en exécution
__name__ : moduleMain
```

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

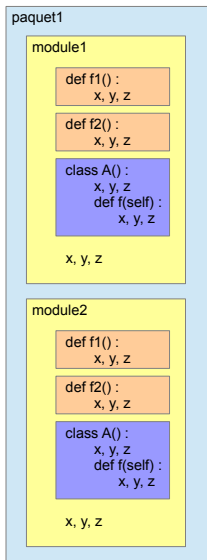
Paquet

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

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



Portée



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

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

Standard library

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

Exemple

Math

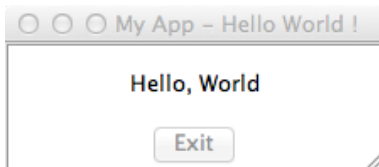
```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isinf',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

Exemple

Graphical User Interface / GUI

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

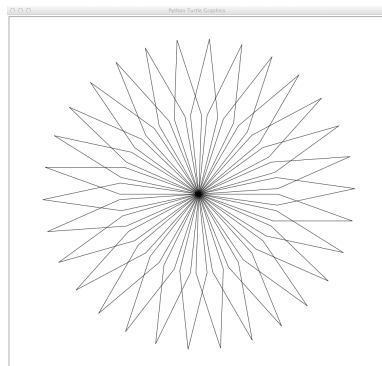
```
>>> import Tkinter
>>> from Tkconstants import *
>>> tk = Tkinter.Tk()
>>> tk.title("My App - Hello World !")
>>> frame = Tkinter.Frame(tk, relief=RIDGE, borderwidth=2)
>>> frame.pack(fill=BOTH,expand=1)
>>> label = Tkinter.Label(frame, text="Hello, World")
>>> label.pack(fill=X, expand=1)
>>> button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



Exemple

Graphical User Interface / GUI

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



Exemple

Base de données

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


Documentation

docstrings

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

```
def times(x, y):  
    """  
    Multiply x by y  
  
    Compute z = x * y  
  
    Input parameters  
    x and y, any numbers  
  
    Output parameters  
    z  
  
    Copyright 2013 G. Becq, Gipsa-lab, UMR 5216, CNRS.  
    """  
    return x * y
```

Documentation

docstrings – exemple

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

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

Contient deux fonctions et une classe.
Ne fait rien.

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

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

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

Documentation

docstrings – exemple

```
def fun2():  
    """  
    fun2 c'est mieux.  
  
    Sans parametres  
    Ne fait rien  
    """  
    return  
  
class A():  
    """  
    A est une classe.  
  
    C'est une classe vide qui ne contient qu'un docstring.  
    """
```

Documentation

help

- Génération automatique de documentation :
`help(nomDuModule)`

```
doc -- less -- 80x36

Help on module module:

NAME
  module - Module pour tester les docstrings.

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

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

  Copyright 2013 G. Beccq, Gipea-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.

```
Python: module module

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

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.iana.org",
      py_modules=["module1", "module2"],
      )
```

Packaging

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

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

Installation des paquets

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

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


Python Package Index

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

The screenshot shows the PyPI website interface. At the top, there's a navigation bar with the Python logo and the text 'python'. Below this, the main heading is 'PyPI - the Python Package Index'. A search bar is located on the right side of the header. The left sidebar contains a list of links under the heading 'PACKAGE INDEX', including 'Browse packages', 'Package submission', 'List trove classifiers', 'List packages', 'RSS (latest 40 updates)', 'RSS (lowest 40 packages)', 'Python 3 Packages', 'PyPI Tutorial', 'PyPI Security', 'PyPI Support', 'PyPI Bug Reports', 'PyPI Discussion', and 'PyPI Developer info'. The main content area is titled 'PyPI - the Python Package Index' and contains a paragraph: 'The Python Package Index is a repository of software for the Python programming language. There are currently 53855 packages here. To contact the PyPI admins, please use the [Support](#) or [Bug reports](#) links.' Below this, there are three columns of information. The first column is 'Get Packages', which explains how to use a package from the index. The second column is 'Package Authors', which provides instructions for submitting packages. The third column is 'Infrastructure', which describes the services used by the index. At the bottom, there is a table with columns 'Updated', 'Package', and 'Description'. The table lists various packages and their descriptions, such as 'collective.geo.location 0.1b7', 'django-theming 1.0', 'django-basic-email 0.0.1', and 'django-party-hb 9.49.3'.

Python Software Foundation pypi.python.org/pypi

PyPI - the Python Package Index : Python Package Index

python

PACKAGE INDEX

- Browse packages
- Package submission
- List trove classifiers
- List packages
- RSS (latest 40 updates)
- RSS (lowest 40 packages)
- Python 3 Packages
- PyPI Tutorial
- PyPI Security
- PyPI Support
- PyPI Bug Reports
- PyPI Discussion
- PyPI Developer info

ABOUT

NEWS

DOCUMENTATION

DOWNLOAD

COMMUNITY

FOUNDATION

CORE DEVELOPMENT

PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **53855** packages here.

To contact the PyPI admins, please use the [Support](#) or [Bug reports](#) links.

Not Logged In

[Login](#)

[Register](#)

[Lost Login?](#)

[Use OpenID](#)

Status

Nothing to report

Get Packages

To use a package from this index either "pip install package" (get pip) or download, unpack and "python setup.py install" it.

Package Authors

Submit packages with "python setup.py upload". The index hosts package docs. You may also use the web form. You must register. Testing? Use testpypi.

Infrastructure

To interoperate with the index use the JSON, OAuth, XML-RPC or HTTP interfaces. Use local mirroring or caching to make installation more robust.

Updated	Package	Description
2015-01-14	collective.geo.location 0.1b7	Add geo views for dexterity content with location js library
2015-01-14	reemote 0.1.3	Sync local databases from REST API
2015-01-14	hookit 0.8.1	Bind GitHub Webhooks to actions
2015-01-14	django-theming 1.0	Django application, implement theming concept, flexible and configurable. Allow theming for host url.
2015-01-14	bzio_app 0.1	A Hadoop's app to handle file download and upload
2015-01-14	book 2	BOKE - XMPP - IRC - CLI - RSS - LDAP - API
2015-01-14	swallow 1.0.0	Swallow for data transformation
2015-01-14	sal 0.2.4	Salabov CLI - Docker hosting
2015-01-14	opened-provider-pebble 0.8	Pebble fork of django_openid_provider
2015-01-14	casestest 0.2.0	Just a quick dockerfile test for my entertainment
2015-01-14	throbserver 0.3.0	F-Drive Server Tools
2015-01-14	logcollection 0-1-0	logging handler collection
2015-01-14	wordpresszinnia 1.0	Import your WordPress blog into Zinnia
2015-01-14	django-basic-email 0.0.1	Django Basic Email application
2015-01-14	counterparty-hb 9.49.3	Counterparty Reference Implementation
2015-01-14	gandi.cli 0.12	Gandi command line interface
2015-01-14	fms web 0.4.3	Frontend Web Application for Fedora Notifications
2015-01-14	pyremote algorithms 0.2.1	Pyremote algorithms
2015-01-14	fms consumer 0.4.3	Backend worker daemon for Fedora Notifications

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"`

