Grenoble INP ENSIMAG

stage liesse

programmer en python

jeudi 2 mai

grenoble inp - ensimag

ÉCOLE NATIONALE SUPÉRIEURE
D'INFORMATIQUE ET DE MATHÉMATIQUES APPLIQUÉES

Programmer en Python

- **Fonctionnalités avancées**
- **Bibliothèques**
  - Base de données, interfaces graphiques
- **iPython**
  - Python de manière interactif / notebook
- **Programmation orientée objet**
- **Ecosystème**
  - Création de modules et packages
  - Tests et documentation

Fonctionnalités avancées

- **Générateurs**
  - Récupère un itérateur pour les valeurs d'une fonction
- **Décorateurs**
  - Exécute du code avant et après une fonction sans modifier cette fonction
- **Fonctions anonymes**
  - Une autre manière de définir de petites fonctions

Les générateurs

Fonctionnalités avancées

- Fonction contenant l'instruction « yield »
- Retourne un objet de type « generator » qui va générer une valeur à chaque « yield »

```
def syracuse(n):
    yield n
    while n != 1:
        if n % 2 == 0:
            n = n / 2
        else:
            n = 3 * n + 1
        yield n

for n in syracuse(3):
    print n
```

```
3
10
5
16
8
4
2
1
```

Les décorateurs

Fonctionnalités avancées

- Il permettent de « décorer » une fonction en exécutant du code avant et après.
  (On peut aussi agir sur les paramètres)
- Peuvent être utilisés comme cache, logs, protection …


```
from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
    ...
```

Les fonctions anonymes

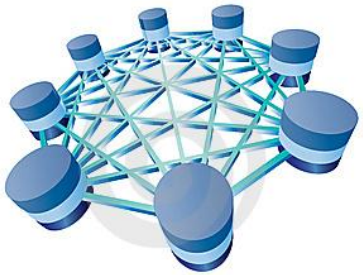
Fonctionnalités avancées

- Elles sont utilisées quand une fonction n'est appelée qu'une seule fois: code plus clair et simple
- Souvent utilisées avec les interfaces graphiques

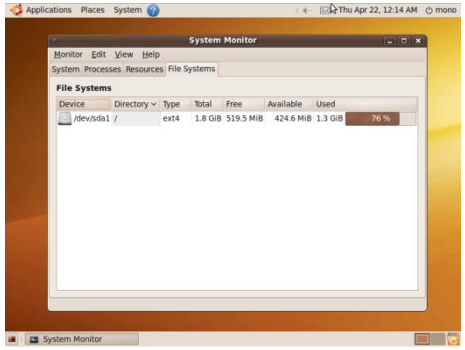
```
frame = tk.Frame(parent)
frame.pack()

btn22 = tk.Button(frame,
        text="22", command=lambda: self.printNum(22))
btn22.pack(side=tk.LEFT)

btn44 = tk.Button(frame,
        text="44", command=lambda: self.printNum(44))
btn44.pack(side=tk.LEFT)
```

Bibliothèques

- **Base de données**



- **Interfaces graphiques avec Tkinter**



Tkinter

Bibliothèques

- Bibliothèques fournie avec Python
- Alternatives : wxPython, Jython …

- Créer une fenêtre

```
#!/usr/bin/python

import Tkinter
top = Tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

- Quitter
  - top.quit()

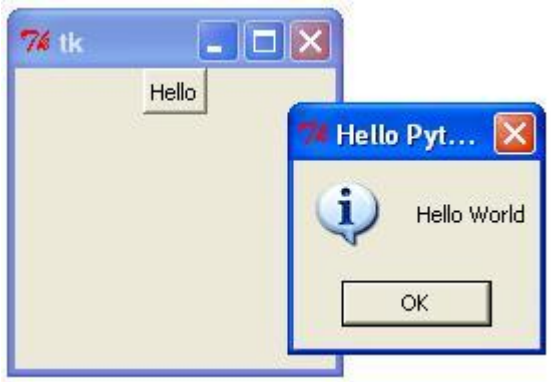
Tkinter & ses widgets

Bibliothèques

- Environ 15 types de widgets
  - Button, Canvas, Frame, ListBox, Label, Scrollbar, Entry, Scale, MessageBox …

- Bouton

```
import Tkinter
import tkMessageBox

top = Tkinter.Tk()

def helloCallBack():
   tkMessageBox.showinfo( "Hello Python", "Hello World")

B = Tkinter.Button(top, text ="Hello", command = helloCallBack)

B.pack()
top.mainloop()
```

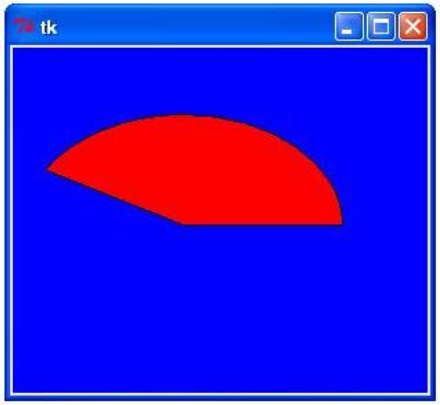


Tkinter & ses widgets

Bibliothèques

- Canevas (surface dessinable)

```
import Tkinter
import tkMessageBox

top = Tkinter.Tk()

C = Tkinter.Canvas(top, bg="blue", height=250, width=300)

coord = 10, 50, 240, 210
arc = C.create_arc(coord, start=0, extent=150, fill="red")

C.pack()
top.mainloop()
```



- On peut y dessiner des images (create_image), arcs, ovales, lignes et des polygones

Tkinter & ses widgets

Bibliothèques

- Placement des widgets
  - Méthode « pack » ([BOTTOM, LEFT, RIGHT, TOP])
  - On peut aussi les regrouper dans une « frame »

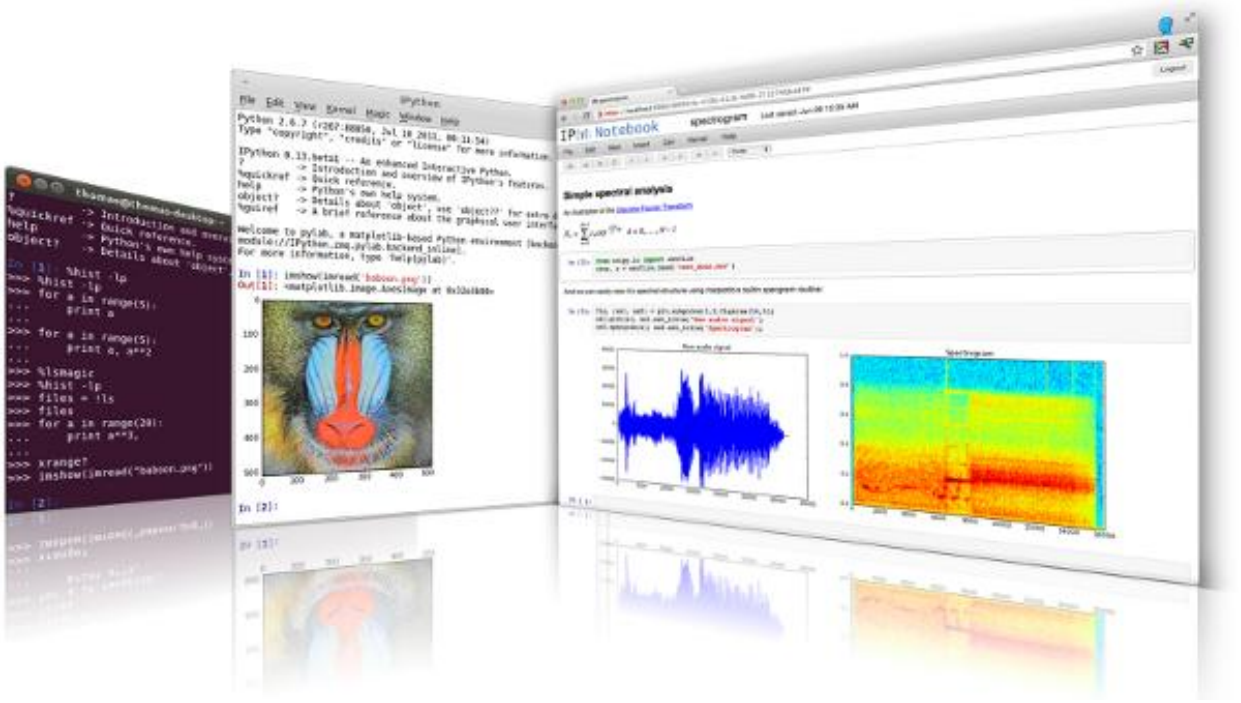
```
from Tkinter import *

root = Tk()
frame = Frame(root)
frame.pack()

bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )

redbutton = Button(frame, text="Red", fg="red")
redbutton.pack( side = LEFT)

greenbutton = Button(frame, text="Brown", fg="brown")
greenbutton.pack( side = LEFT )

bluebutton = Button(frame, text="Blue", fg="blue")
bluebutton.pack( side = LEFT )

blackbutton = Button(bottomframe, text="Black", fg="black")
blackbutton.pack( side = BOTTOM)

root.mainloop()
```

IPython

- Shell interactif commencé en 2001 par le créateur de matplotlib
- **Introspection**, **auto-complétion**, commandes Shell, aides sur les objets, « fonctions magiques », historique, **notebook**

Interfaces

IPython



Ex : https://www.pythonanywhere.com/try-ipython/

Interfaces

IPython

- Notebook
  - IPython disponible depuis le Web
    - Texte, math, graphique, exécution de code
  - Nécessite qu'un accès Internet
  - Rendu clair, propre et peut être sauvegardé dans un fichier (HTML/PDF)
    (Ex: http://nbviewer.ipython.org)

Fonctionnalités

IPython

- Historique : _ __ ___ _4 _5 … et %history
- Sortie asynchrone

```
In [30]: import time, sys
         for i in range(8):
             print(i)
             time.sleep(0.5)

0 1 2 3 4 5 6 7
```

- %timeit, %save, %run

```
In [1]: %timeit range(1000)
100000 loops, best of 3: 7.76 us per loop

In [2]: %%timeit x = range(10000)
   ...: max(x)
   ...:
1000 loops, best of 3: 223 us per loop
```

- Object?, Object??

```
In [8]: f??

Type:       function
String Form:<function f at 0x03DF61F0>
File:       c:\python27\scripts\<ipython-input-6-6509335aee61>
Definition: f()
Source:
def f():
    return 0
```

- Formules mathématiques (MathJax)

Math with font-size set to 500%

$$g\frac{d^2u}{dx^2} + L\sin u = 0$$

Math with font-size set to 200%

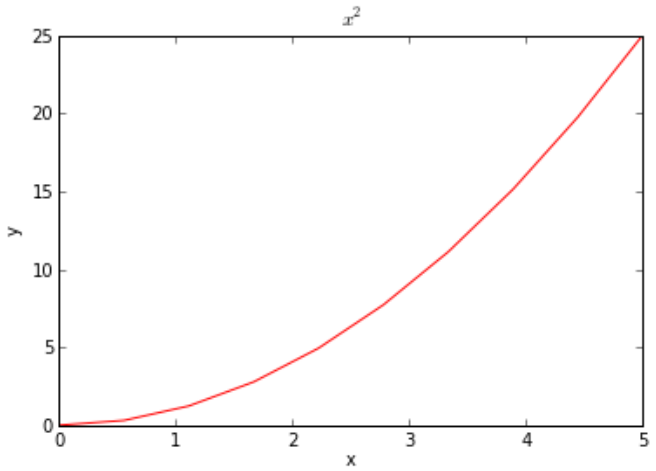
$$g\frac{d^2u}{dx^2} + L\sin u = 0$$

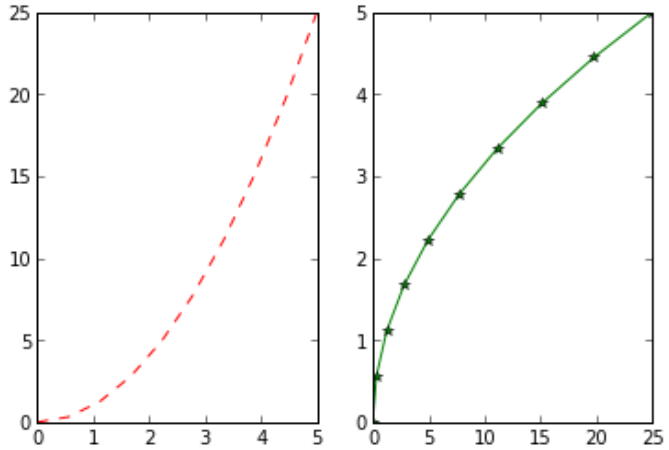
Matplotlib

IPython

- Plot simple, subplot et formules en légende

```
x = linspace(0, 5, 10)
y = x ** 2

plot(x, y, 'r')
xlabel('x')
ylabel('y')
title('$x^2$')
show()
```





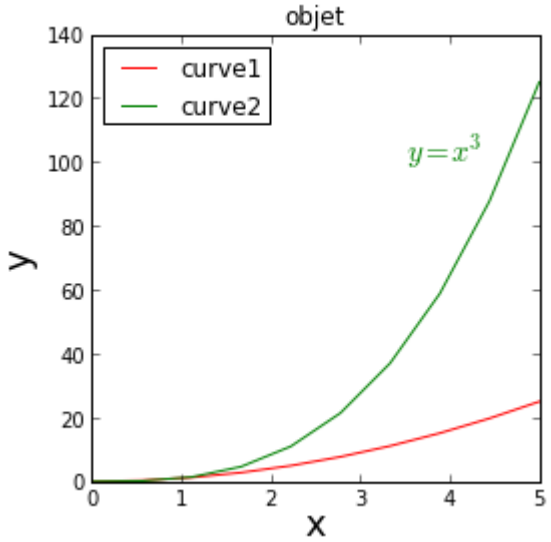

```
subplot(1,2,1)
plot(x, y, 'r--')
subplot(1,2,2)
plot(y, x, 'g*-')
show()
```

Matplotlib

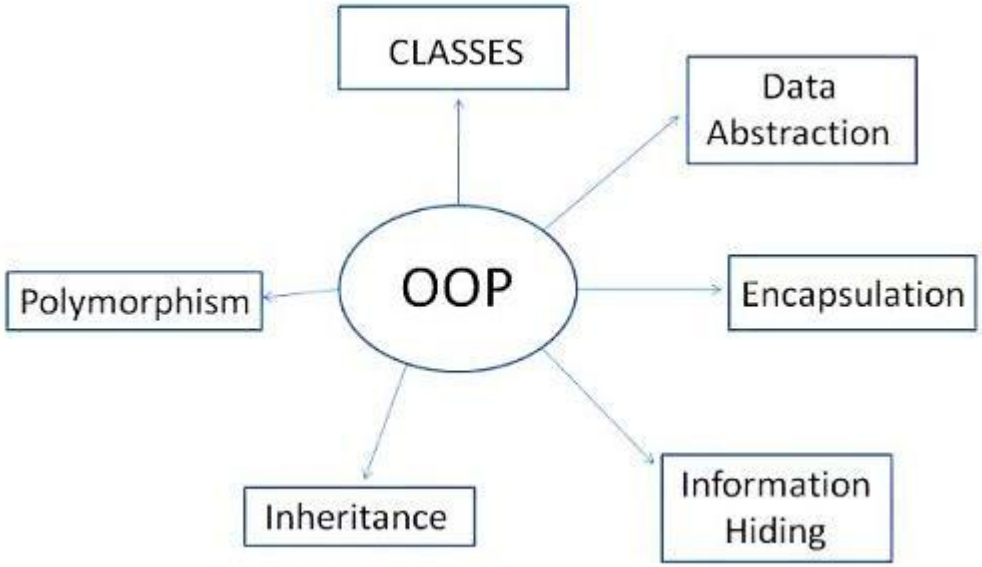
IPython

- Mode « objet » et légende

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(4,4), dpi=100) #  400x400 pixels
ax.plot(x, x**2, color="red", label="curve1")
ax.plot(x, x**3, color="green", label="curve2")
ax.set_xlabel('x', fontsize=18)
ax.set_ylabel('y', fontsize=18)
ax.text(3.5, 100, r"$y=x^3$", fontsize=15, color="green")
ax.legend(loc=2);
fig.savefig("mygraph.png", dpi=200) # PNG, JPG, EPS, SVG, PDF
```



Programmation objet



Bases

Programmation objet

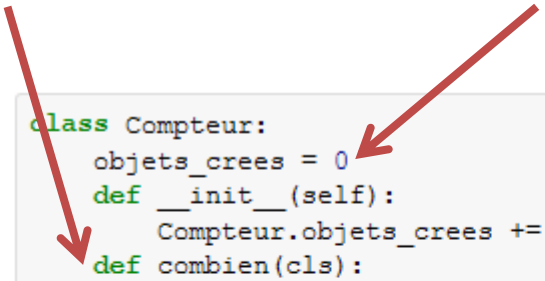
- Classe, constructeur et attributs

```
class Personne:
    def __init__(self):
        self.nom = "Dupont"
        self.prenom = "Jean"
    def __str__(self):
        return self.prenom + " " + self.nom

p = Personne()
print p
```

```
Jean Dupont
```

- Instanciation, représentation texte

- Méthodes et attributs de classe



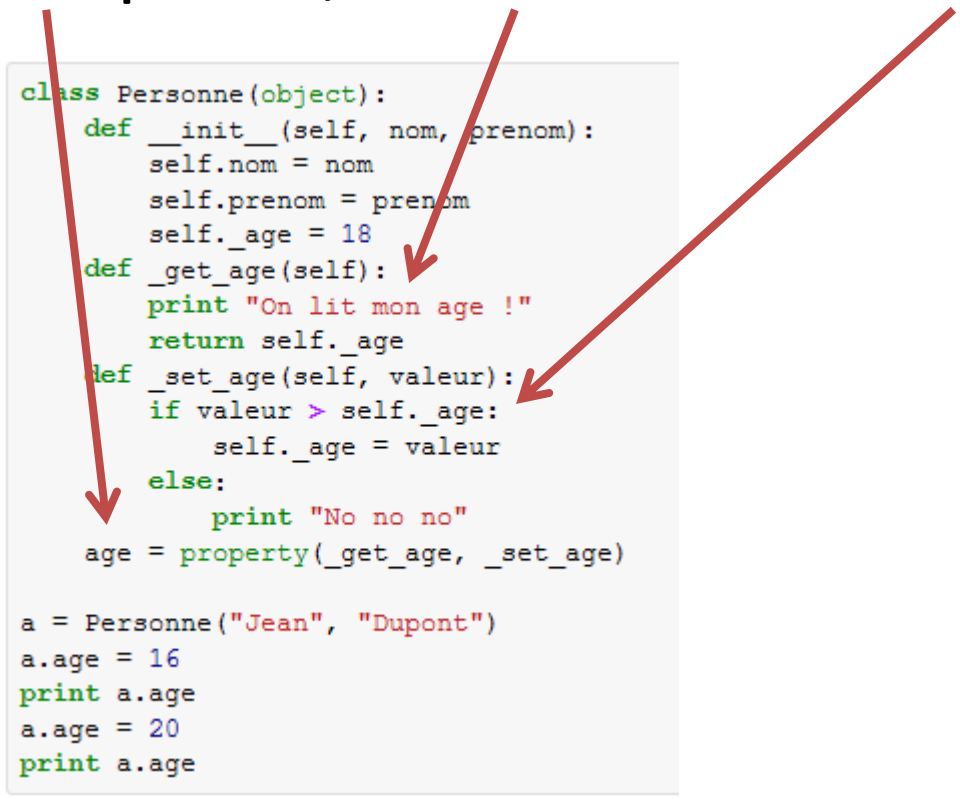
```
class Compteur:
    objets_crees = 0
    def __init__(self):
        Compteur.objets_crees += 1
    def combien(cls):
        print("Jusqu'à présent, {} objets ont été créés.".format(cls.objets_crees))
    combien = classmethod(combien)

a = Compteur()
Compteur.combien()
b = Compteur()
Compteur.combien()
```

```
Jusqu'à présent, 1 objets ont été créés.
Jusqu'à présent, 2 objets ont été créés.
```

Propriétés

Programmation objet

- Propriété, accesseur et mutateur



```
class Personne(object):
    def __init__(self, nom, prenom):
        self.nom = nom
        self.prenom = prenom
        self._age = 18
    def _get_age(self):
        print "On lit mon age !"
        return self._age
    def _set_age(self, valeur):
        if valeur > self._age:
            self._age = valeur
        else:
            print "No no no"
    age = property(_get_age, _set_age)

a = Personne("Jean", "Dupont")
a.age = 16
print a.age
a.age = 20
print a.age
```

```
No no no
On lit mon age !
18
On lit mon age !
20
```

property(**methode_accesseur**, **methode_mutateur**, **methode_suppression**, **methode_aide**)

Méthodes spéciales

Programmation objet

Nom	Fonction
__del__	destructeur
__repr__	Représentation informelle texte
__getitem__ __setitem__ __delitem__	Object[x] = y
__add__ __sub__ __mul__ __mod__ __truediv__ …	Mathématique
__eq__ __ne__ __gt__ __ge__ __lt__ __le__	Comparaison

Ecosystème

- **Modules**
- **Packages**
- **Tests**
- **Documentation**

Modules

Ecosystème

- Contient des fonctions et instructions
- Le nom du fichier est le nom du module
  Ex. fibo.py

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Importation

Modules

- import fibo
- from fibo import func1, func2
- from fibo import *

- Exécution / Importation

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

```
$ python fibo.py 50
1 1 2 3 5 8 13 21 34
```

```
>>> import fibo
>>>
```

Packages

Ecosystème

- Regroupe un ensemble de module de façon structurée

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Packages

Ecosystème

- « Init.py » (vide) dans chaque dossier

- from mypackage import * ?
  - Chercher dans les fichiers n'est pas efficace
  - Utilise __all__ dans le « init.py »

```
__all__ = ["echo", "surround", "reverse"]
```

  - from sound.effects import *

Tests

Ecosystème

- Unittest (PyUnit)
  - Module de tests unitaires
  - Fourni avec Python
  - Mode discovery

```python
import unittest
from mymodule import multiply

class TestM(unittest.TestCase):

    def setUp(self):
        pass

    def test_numbers_3_4(self):
        self.assertEqual( multiply(3,4), 12)

    def test_strings_a_3(self):
        self.assertEqual( multiply('a',3), 'aaa')

if __name__ == '__main__':
    unittest.main()
```

```
> python test_um_unittest.py
..
----------------------------------------------
Ran 2 tests in 0.000s

OK
> python test_um_unittest.py -v
test_numbers_3_4 (__main__.TestM) ... ok
test_strings_a_3 (__main__.TestM) ... ok

----------------------------------------------
Ran 2 tests in 0.000s

OK
```

Méthodes de tests

- setUp / tearDown
  - Initialise/libère les données de tests
- Asserts (une partie)

Method	Checks that	New in
`assertEqual(a, b)`	`a == b`	
`assertNotEqual(a, b)`	`a != b`	
`assertTrue(x)`	`bool(x) is True`	
`assertFalse(x)`	`bool(x) is False`	
`assertIs(a, b)`	`a is b`	2.7
`assertIsNot(a, b)`	`a is not b`	2.7
`assertIsNone(x)`	`x is None`	2.7
`assertIsNotNone(x)`	`x is not None`	2.7
`assertIn(a, b)`	`a in b`	2.7
`assertNotIn(a, b)`	`a not in b`	2.7
`assertIsInstance(a, b)`	`isinstance(a, b)`	2.7
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`	2.7

Documentation

Ecosystème

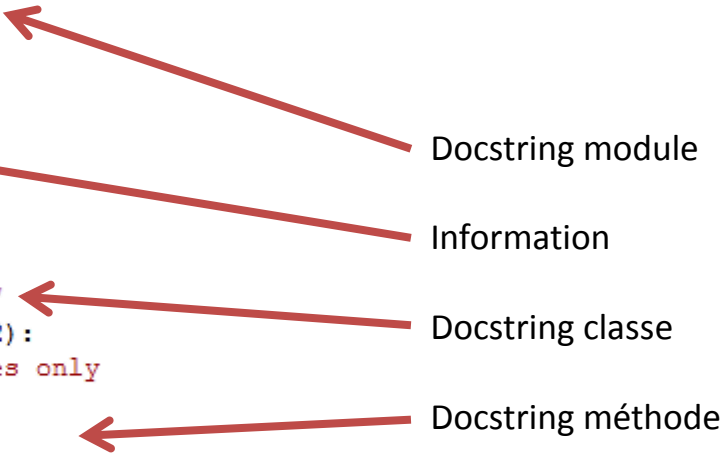
- Pydoc
  - Module de génération de documentation
  - Disponible dans %Python%/Lib/
  - Utilise la docstring des modules, classes et méthodes

- Commandes
  - pydoc -w hello > hello.html
  - pydoc -p 9999

Module documenté

Documentation

```
"""Show off features of [pydoc] module

This is a silly module to
demonstrate docstrings
"""
__author__ =  'David Mertz'
__version__=  '1.0'
__nonsense__ = 'jabberwocky'

class MyClass:
    """Demonstrate class docstrings"""
    def __init__ (self, spam=1, eggs=2):
        """Set default attribute values only

        Keyword arguments:
        spam -- a processed meat product
        eggs -- a fine breakfast for lumberjacks
        """
        self.spam = spam
        self.eggs = eggs
```



Références

Python	http://docs.python.org/2.7/
IPython	http://ipython.org
Tutorialspoint	http://www.tutorialspoint.com/python/index.htm
Siteduzero	http://www.siteduzero.com/informatique/python/tutoriels
Programmation objet	http://docs.python.org/2/tutorial/classes.html
Tests unitaires	http://docs.python.org/2/library/unittest.html
PyDoc	http://docs.python.org/2/library/pydoc.html
Générateurs	http://wiki.python.org/moin/Generators
Décorateurs	http://wiki.python.org/moin/PythonDecorators
Fonctions anonymes	http://wiki.python.org/moin/AlternateLambdaSyntax
Modules & Packages	http://docs.python.org/2/tutorial/modules.html
SQLite	http://docs.python.org/2/library/sqlite3.html