apius

Unknown Author

October 24, 2013

Part I

APIUS.py - L'état du Python

Mathieu Leduc-Hamel

Programmeur Python mais aussi:

Du haut de ma petite expérience Pythonesque et de ce notebook, j'aimerais vous faire aimer ce langage

1 L'histoire de Python

1.1 La définition officielle

Python est un langage de programmation objet, multi-paradigme et multi-plateformes. Il favorise la programmation impérative structurée et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

http://fr.wikipedia.org/wiki/Python_%28langage%29

1.2 Qu'est-ce que ça veut dire ?

Que Python est un language libre, moderne de programmation et qu'il est écrit **actuellement** en C (CPython).

In [15]:
```
from pyquery import PyQuery as pq
element = pq(url="http://www.python.org/2.7/license.html").find("img")[-1]
Image("http://www.python.org" + element.attrib["src"])
```

Out [15]:

1.3 Comment Python évolue

En soumettant un PEP, comme un blueprints dans le monde d'Ubuntu

1.4 Qu'est-ce qu'un PEP

- Un PEP c'est un "Python Enhancement Proposals"
- C'est un document qui sert à proposer des améliorations à Python et à sa librairie standard
- Il en existe plus plusieurs centaines, soit:
  - Meta (des PEPs à propos du processus des PEPs)
  - Autre
  - Accepté
  - Ouvert
  - Terminé
  - Historique
  - Reporté
  - Abandonné

1.5 Est-ce que les PEP sont courant ?

Le plus récent PEP a être accepté l'a été cette semaine, le 22 octobre 2013.

http://www.python.org/dev/peps/pep-0453/

À propos de l'intégration de pip dans la librairie standard, un installateur de paquets Python.

Bravo à Donald Stufft !

1.6 pep-0020: Le Zen de Python

C'est ce qui guide le développement de notre language mais aussi notre travail et notre façon de voir les choses

In [4]:
```
from pyquery import PyQuery as pq
print(pq(url="http://www.python.org/dev/peps/pep-0020/")("#body-main").find("pre")[1].
```

```
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.
```

```
    Sparse is better than dense.
    Readability counts.
    Special cases aren't special enough to break the rules.
    Although practicality beats purity.
    Errors should never pass silently.
    Unless explicitly silenced.
    In the face of ambiguity, refuse the temptation to guess.
    There should be one-- and preferably only one --obvious way to do
it.
    Although that way may not be obvious at first unless you're Dutch.
    Now is better than never.
    Although never is often better than *right* now.
    If the implementation is hard to explain, it's a bad idea.
    If the implementation is easy to explain, it may be a good idea.
    Namespaces are one honking great idea -- let's do more of those!
```

1.7 En résumé

C'est notre code, ça ne fait pas force de lois et on a chacun notre interpréation du Zen de Python, mais ce qui est clair c'est:

- Pragmatisme
- Simplicité
- Éléguance
- Drôle !

  "Python – This is a Real English Word (honest, look it up!) that happens to refer to a type of snake, which you'll notice is an object. With the two vowels, python is quite readable." – Andrew Dalke

In [13]:
```
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.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
```

```
Namespaces are one honking great idea -- let's do more of those!
```

1.8 Guido van Rossum

In [17]:
```
Image("http://pytalent.zandstrasystems.com/Guido-van-Rossum_DO6GvRhi.jpg")
```

Out [17]:

1.9 Qui Guido ?

- Née aux Pays-Bas, en 1957
- Complété un master en mathématiques en sciences informatiques à l'université d'Amsterdam

- Soumet au ministère de la défense américaine une proposition d'un nouveau language de programmation en 1999 (Python...)
- Possède le titre de dictateur bien aimé et à vie (BDFL)
- Travail chez Dropbox depuis Janvier 2013.
- A travaillé dans le passé chez:
  - Google
  - Elemental Security
  - Zope Corporation
  - BeOpen.com
  - CNRI (où il a créé Python)
  - CWI
  - SARA

1.10 L'âge préhistorique

```
La première version publique de Python date de 1991.

À l'époque Guido était fortement influencé par le LISP et la programmation fonctionnelle.

0.9.0 - Février 1991

Première publication officielle sur alt.sources.

Python 1.0 - Janvier 1994

Guido travail au CNRI, Corporation for National Research Initiatives

  * Ajout du lambda, map, filter et du reduce.
```

1.11 Les temps modernes

```
Python 2.0 - Octobre 16, 2000

* Ajout des listes par compréhension et du "garbage collector"

    Python 2.1 - Avril 17, 2001

    * Naissance de la PSF

    Python 2.2 - Décembre 21, 2001

    * Le langage évolue et rafine avec les "nested scope" (la portée des variables)

    Python 2.3 - Juillet 29, 2003

    * On gagne de la vitesse et on peut le savoir grâce à timeit !

    Python 2.4 - Novembre 30, 2004
```

```
    * Set fait son entrée
    * Générateurs (pep 289)
    * Décorateurs

    Python 2.6 - Octobre 1, 2008

    * On structure l'évolution des releases (pep 361) jusqu'à 3.0
    * with et les context managers
    * multiprocessing

    Python 2.7 - Juillet 3, 2010

    * On backport des modules de Python 3.1
    * Une release de transition
```

1.12 La période contemporaine

```
Python 3.0 - Décembre 3, 2008

    Python 3.1 - Juin 27, 2009

    Python 3.2 - Février 20, 2011

    Python 3.3 - Septembre 29, 2012
```

1.13 Pourquoi migrer à Python 3

Selon "Nick Coghlan's Python Notes" http://python-notes.boredomandlaziness.org/en/latest/python3/

- Supprimer les fonctionalités déjà "dépréciées"
- Réduire le nombre de mot réservés dans le langage
- Utiliser des conteneur plus efficace en mémoire
- Rendre la librairie standard plus **PEP8**

et...

2 Unicode !!!

```
In [16]: "évoluton du language"

Out [16]:
         '\xc3\xa9voluton du language'
```

2.1 Comment va le passage à Python 3

- Souffrant dans le passé, ça avance bien.
- Porter son code c'est facile (2to3).
- Plupart des packages important ont fait le saut.

In [29]:
```
from IPython.core.display import HTML

HTML("<iframe src='https://python3wos.appspot.com/' height='600'></iframe>")
```

Out [29]:
```
<IPython.core.display.HTML at 0x10e20c150>
```

2.2 Évolution de la taille de librairie standard

In [6]:
```
from pyquery import PyQuery as pq

base_url = "http://docs.python.org/release/%s/modindex.html"
new_url = "http://docs.python.org/release/%s/py-modindex.html"
newer_url = "http://docs.python.org/%s/py-modindex.html"

data = {}

def count_modules(version, dt=False, old=True, newer=False):
    if dt:
        data[version] = len(pq(url=base_url % version).find("table dt"))
    elif old:
        data[version] = len(pq(url=base_url % version).find("table tt"))
    elif newer:
        data[version] = len(pq(url=newer_url % version).find("table tt"))
    else:
        data[version] = len(pq(url=new_url % version).find("table tt"))
    return data

for v in range(1, 6):
    count_modules("2.%d" % v, dt=True)

count_modules("2.6")
count_modules("2.7")
count_modules("3.0")
count_modules("3.1")
count_modules("3.2", old=False)
print(count_modules("3.3", old=False, newer=True))
```

```
{'3.2': 306, '3.3': 313, '3.0': 292, '3.1': 298, '2.3': 314, '2.2':
285, '2.1': 255, '2.7': 430, '2.6': 392, '2.5': 379, '2.4': 362}
```

2.3 Concrètement ça donne ? (merci matplotlib)

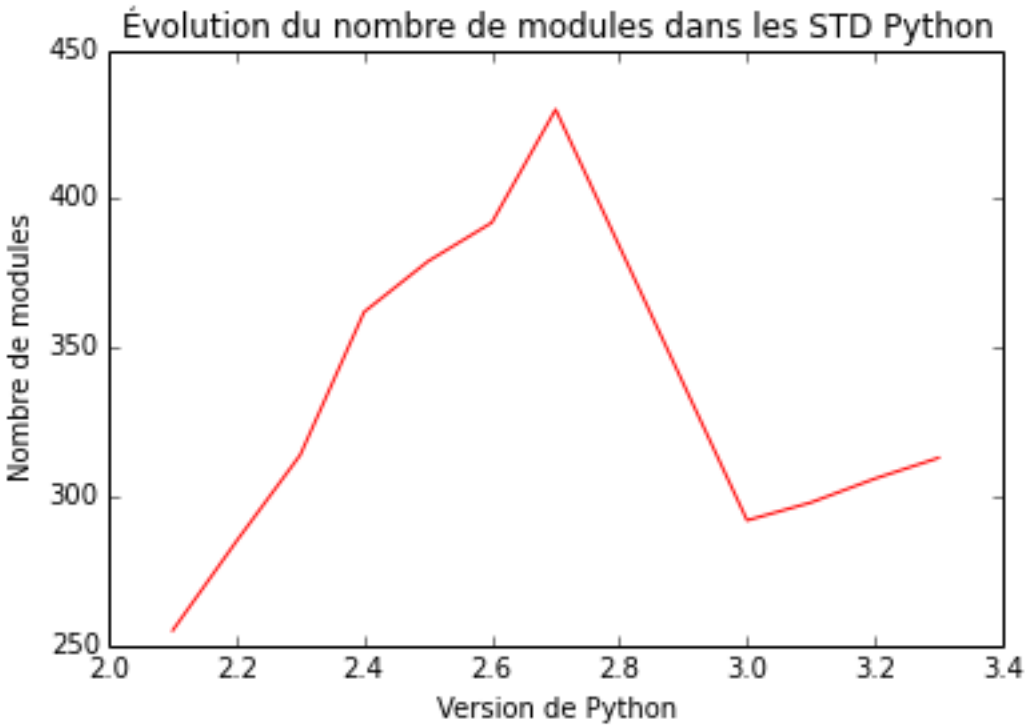
In [14]:
```
%pylab inline --no-import-all

from pylab import figure, plot, title, xlabel, ylabel, show

x = data.keys()
x.sort()
y = [data[k] for k in x]

figure()
plot(x, y, 'r')
xlabel('Version de Python')
ylabel('Nombre de modules')
title(u'Évolution du nombre de modules dans les STD Python')
show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



2.4 La post-histoire

PyPY 1.0 - Juin 2007

Une complète ré-écriture de Python en Python. Le but de l'opération :

* implanter un compilateur Just-In-time (JIT)
* conserver la compabilité avec le CPython tradionnel
* créer une plateforme d'expérimentation pour les languages de programmation dynamiques

PyPY 1.2 - Mars 2010

PyPy 1.5 - Avril 2011

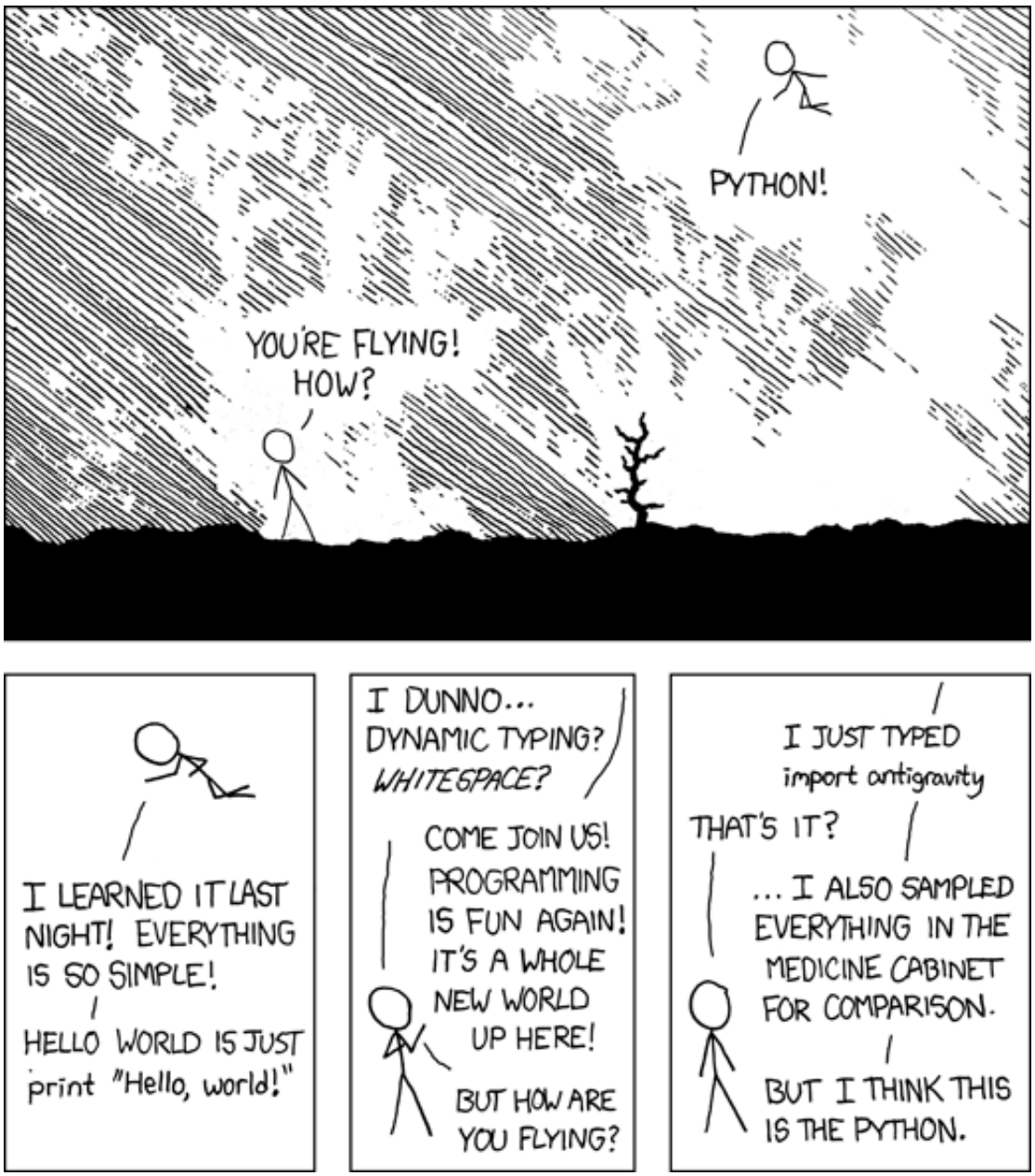
PyPy 1.9 - Mars 2012

PyPy 2.1 - Auoût 2013

3 La Communauté

In [35]:
```
from IPython.core.display import Image
import antigravity
Image("http://imgs.xkcd.com/comics/python.png")
```

Out [35]:



3.1 La Python Software Foundation

- Une organization sans but lucratif enregistrée
- Défend et gère le nom de marque Python (Bataille juridique plus tôt cette année)
- Aide la formation de nouveaux groupes d'usagers
- Finance le développement de projets Python
- Promouvoit le Python par PyCon

3.2 Les groupes d'usagers

- Base de la communauté Python
- Des groupe d'usagers sur 5 continents (https://wiki.python.org/moin/LocalUserGroups)
- Dans la plupart des langues du monde

In [5]:
```
from pyquery import PyQuery as pq
len(pq(url="https://wiki.python.org/moin/LocalUserGroups").find(".table-of-contents li
```

Out [5]:
```
94
```

3.3 PyCon au travers le monde

- PyCon 2014 aura lieu à Montréal
- PyCon dans plus de 30 pays à chaque année
- Les conférences sont menées et gérées par des volontaires
- l'horaire mondial est très chargé (http://www.pycon.org/)

4 Python dans le futur

In [31]:
```
from IPython.core.display import Image

Image("http://www.python.org/~guido/images/license.jpg")
```

Out [31]:

In [1]:
```
from IPython.core.display import Image
Image("http://apius.ca/images/27c4a580.mathieu.jpg")
```

Out [1]:

twitter: @mlhamel

courriel: mathieu@mtlpy.org