# Introduction à Python

(orientée web)

Jan Krause / Bibliothèque de la Faculté de Médecine / Université de Genève

# Qu'es-ce que Python?

## Efficient + Rapide + Universel = Python

### Quelles plate-formes?

UNIX (en général), Linux, Mac, Windows, Playstation (2 et PSP), QNX, Cray supercomputers, IBM mainframes, PDAs (PalmOS, iPods, iPhone, Windows Mobile), BeOS (Haiku), OS/2, etc.

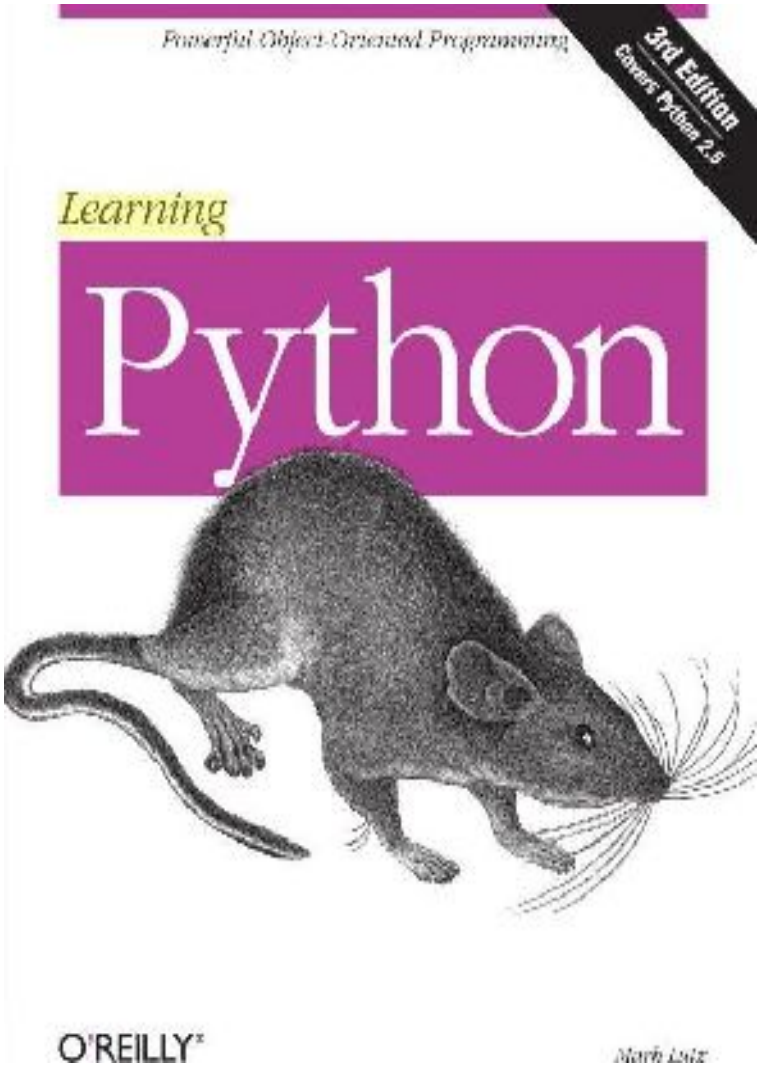
### Qui utilise python?

Google, YouTube, BitTorrent, Intel, HP, Cisco, IBM, UBS, NASA, Los Alamos, FemiLab, CERN, NSA (cryptanalyse), Industrial Light and Magic (StarWars, Forest Gump...), Canonical (Ubuntu), etc.

### Licence ?

Python Software Licence (PSL), une licence open source, calquée sur la licence Apache et compatible GNU GPL

# Apprendre: un livre




**Learning Python**
Par Mark Lutz
Edition: 3, revised
Publié par O'Reilly, 2007
ISBN 9780596513986
700 pages

UNIGE, Bibliothèque d'Anthropologie
[lien sur le catalogue]

# Apprendre: ressources web

Cette présentation en ligne et les scripts donnés en exemple:

http://jankrause.net/python/tutorial/

- Incontournable: http://www.python.org/doc/
  - Le module index (bibliothèque standard)
  - Le tutorial
  - Les howtos

- Dévleoppez.com: http://python.developpez.com/

# Types de données et opérateurs

les opérateurs: + - * / ** % and or not += -= *= /= **= ( )

nombres (entiers, flottants complexes)

ex: n = 24 r = 3.1415 c = 1.5 + 2j

- chaînes de caractères ( 'abc' ou ''abc'' ou """multi-lignes""" )

  s = 'une chaîne\t' + ''une autre chaîne'' + '\nune autre linge'

- listes:

  L = [1, 'abc', 3.14] emboitées: L = [ [1, 2], 3 ]

- dictionnaires (dont les entrées sont repérées par des clés)

  D = {'jan': 74, 'aline':23, 'christian':None, 'z':[1, 2, 3]}

  D['jan'] → 74130

- booléens A = True B= False

- fonctrions arithmétiques -> voir le module math: import math

# Types de données: plus sur les strings

Rechercher, remplacer splitter et joindre:

le module string de la librairie standard

```
import string

s = "La connaissance s'acquiert par
   l'expérience, tout le reste n'est que de
   l'information. Albert Einstein."

s.find('reste')

s2 = s.replace('connaissance', 'savoir')

liste_mots =s.split(' ')

' '.join(liste_mots)
```

# Types de données: plus sur les strings expressions rationnelles (RegEx)

Le module re – regular expressions – de la bibliothèque standard permet d'utiliser les expressions rationnelles

```
>>> import re
>>> u = "saf, sdf, sgf, fsdlkjlkdf, str"
>>> r = re.compile(r's.f')
>>> re.findall(r, u)
['saf', 'sdf', 'sgf']
```

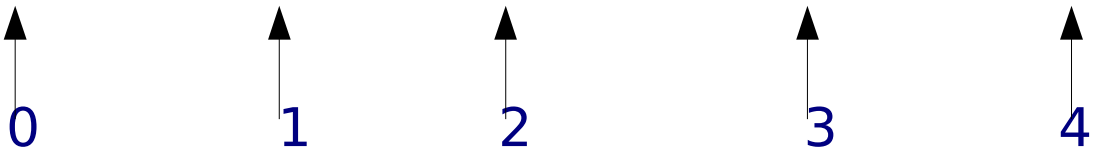
Doc: http://docs.python.org/library/re.html

Exemple: ./divers/regexp.py

# Types de données

Indexes des types de données composés (listes, chaines de caractères, ...) : maîtriser le slicing.

Liste = [ 'zero' , 'un' , 'deux' , 'trois' , 'quatre' ]




Liste[0] → 'zero'

Liste[0:2] → ['zero', 'un']

Liste[1:2] → 'un'

Liste[-1] → 'quatre'

Liste[:2] → ['zero', 'un']

Liste[-2:] → ['trois', 'quatre']

Liste[2:2] = []

Liste[2:2] = 'un bis' → insère en position 2

# Types de données: listes

Méthodes et fonctions utiles:

```
liste = [1, 2, 3, 4, 5, 6, 7, 'livre', 'livre']
```

Longueur de la liste (string, dictionnaire, ...): l = len(liste)

Ajouter un élément à la fin: list.append('nouvel élément')

Extraire un élément: element = list.pop(indice)

Compter le nombre d'occurrences: n = list.count('livre')

Trier les éléments: list.sort()

Renverser la liste: list.reverse()

Etc. -> http://docs.python.org/tutorial/datastructures.html

# Types de données: dates

```
>>> import time
>>> import datetime

>>> now = datetime.datetime.now()
>>> now.strftime("%Y-%m-%d %H:%M:%S") # datetime -> string
'2009-02-13 13:03:38'

>>> now + datetime.timedelta(weeks=1) # ajouter une semaine
datetime.datetime(2009, 2, 20, 13, 3, 38, 412821)

>>> naissance_jan = datetime.datetime(1978, 10, 19, 19, 15, 00)
>>> age_jan = now - naissance_jan
>>> print(age_jan)
11074 days, 17:48:38.412821
```

# Types de données: ensembles

```
>>> list = ['a', 'a', 'a', 'b', 'c', 'd']
>>> s = set(list) # enlève les doublons -> chaque élément est unique
>>> s
set(['a', 'c', 'b', 'd'])

>>> t = set(['c', 'd', 'e'])
>>> t.add(123)
>>> t.remove(123)

>>> s | t    # UNION / OR
set(['a', 'c', 'b', 'e', 'd'])

>>> s ^ t    #XOR / dans l'un ou l'autre mais pas dans les 2 à la fois
set(['a', 'b', 'e'])

>>> s & t   # INTERSECT / AND
set(['c', 'd'])
```

# Lecture / écriture de fichiers

## Lire un fichier:

```
s = open( "fichier.txt", "r" ).read()  # renvoie un string
```

## Ou

```
lignes = open( "fichier.txt", "r" ).readlines()  # liste de lignes
```

## Ecrire dans un fichier

```
f= open( "fichier.txt", "w" ).write( s ) # le fichier est écrasé
                                         # si il existait déjà
```

# Lecture / écriture: UTF-8 (et autres encodages)

## Lire de l'UTF-8

```
import codecs # module de la bibliothèque standard
s = codecs.open( "fichier.txt", "r", "utf-8" ).read()
```

## Ecrire de l'UTF-8

```
import codecs # module de la bibliothèque standard
open( "fichier.txt", "w" ).write( s.encode( "utf-8" ) )
```

# Syntaxe: boucles et conditions

Les blocs de code sont définis par leur indentation (espaces depuis la marges). Les «  :  » introduisent un nouveau bloc.

## Conditions

```
x = 24
if  x > 0:
        y = 1 / x
        print( y )
x = 3232
...
```

## Conditions

```
if a==0:
        print('non')
elif a>0:
        print('oui')
else:
        print('autre')
```

## Boucles For (sur tout type de séquences)

```
list = [0,1,2,3]
for i in list:
        print(i)

 for i in range(4):
        print(i)
```

## Boucles While

```
j = 1
while j < 100:
        j += 1
        print(j)
```

# Conditions

## Les opérateurs booléens:

< > <= >= == not and or

## L'opérateur « in »

- 'a' in 'abc' -> True
- 12 in [1, 12, 23] -> True
- 'jan' in {'jan':30, 'aline':30} -> True

## Remarque:

- Les chaines vides sont False, sinon True
- Nombres: 0 et 0.0 sont false, sinon True
- Listes et dictionnaires: si vides sont False, sinon True

# Gestion de la mémoire: del fait le ménage

« del » permet de libérer des variables modules et objets de la mémoire, exemple:

- Variables

```
n = 128
del(n)
```

- Entrées de dictionnaires

```
d = {'a':1, 'b':2}
del( d['a'] )
del( d )
```

- Modules (voir ci-après)

```
import string
del(string)
```

- Eléments de listes

```
l = ['a', 'b', 'c']
del( l[0] )
del ( l )
```

- Objets (voir ci-après)

del objet
remarque: appelle la méthode destructor (__del__) si il y en a une.

# Ecrire un script: coding style (cf .PEP8)

- Indentation par groupes de 4 espaces
  - Ne pas utiliser de 'tabs' !
- Eviter les lignes qui «  dépassent de l'écran  »
  - On peut continuer un ligne logique sur la ligne suivante du script au moyen d'un «  \  »
- Utiliser utf-8 pour l'encodage (pas de BOM)
- Début d'un script (*.py):

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

Chemin de l'interpréteur python

Déclaration d'encodage, pour python et l'éditeur

La ligne est commentée

# Ecrire un script: éditeur

Choisir un environnement de développement approprié:

- Eclipse / PyDev
- Emacs
  - Multi-platteformes
  - Hautement configurable
  - python_mode (ex: touche tab -> 4 espaces...)
  - PyMacs (macros d'emacs en python)
- Komodo (notamment windows)... etc.
- Votre éditeur préféré .....

# Fonctions, modules, packages

Ces structures permettent de structurer le code:

- éviter la duplication de code (redondance)
- simplifier la maintenance, le développement
- augmenter la lisibilité
- décomposer une tâche complexe
- répartir le travail entre programmeurs
- cacher les détails de l'implémentation aux utilisateurs

# Fonctions

```
def nom_fonction ( pa1, pa2, ... ) :
    """ Documentation ... """
    corps de la fonction
    return variable
```

Les blocs de codes sont définis par l'indentation du code.

```
def fib(n):
    """
    Calcule la suite de Fibonacci jusqu'à n.
    1, 1, 2, 3, 5 ,8 ,13, 21, 34 ...
    """
    a, b = 0, 1
    output = []
    while b <= n:
        output.append(b)
        a, b = b, a+b
    return output

fib(100)
help(fib)
```

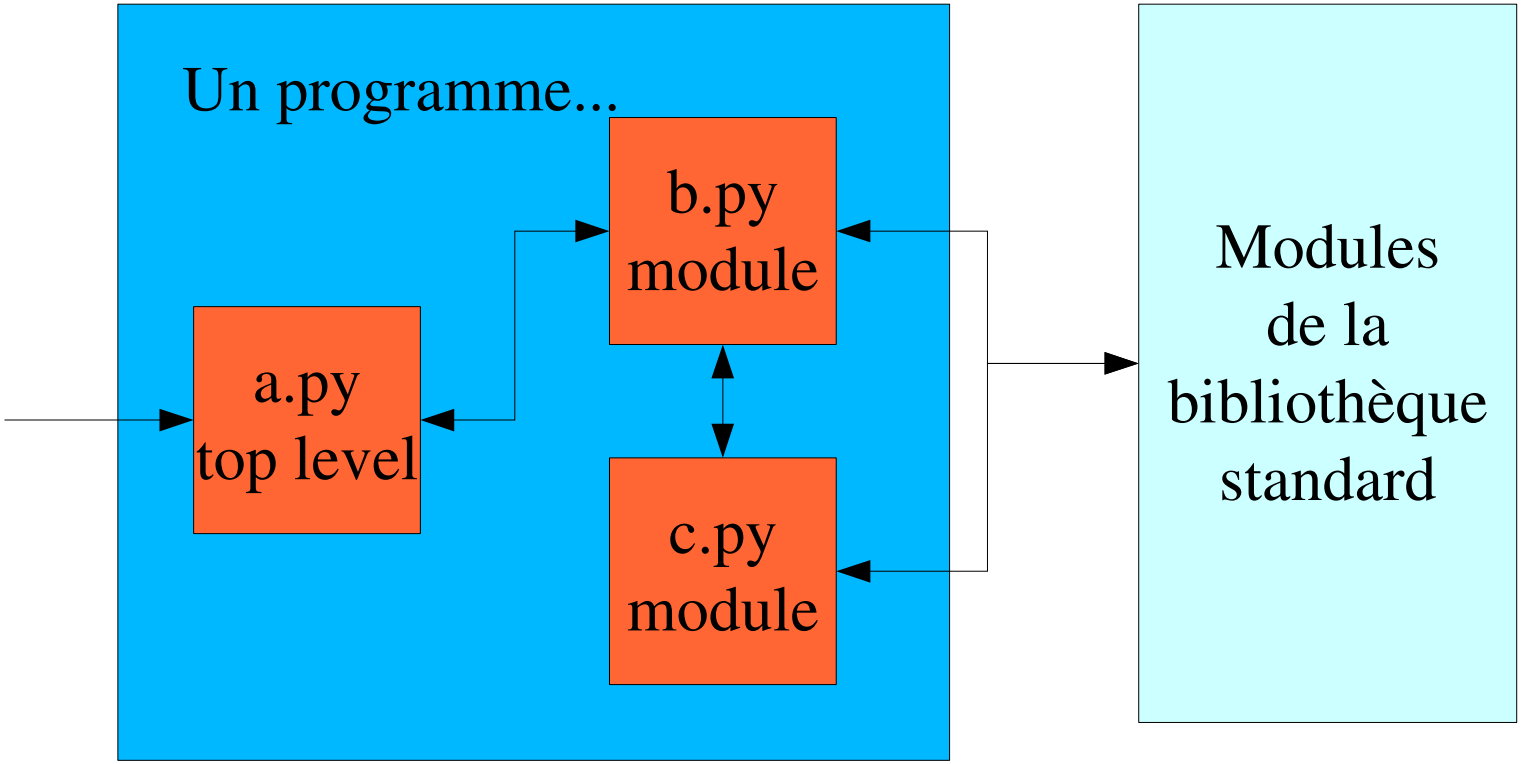
# Modules

Permettent de regrouper les fonctions par thème dans des fichiers, les rendant ainsi réutilisables dans plusieurs programmes.

Un module, c'est simplement un fichier .py qui contient plusieurs fonctions, générateurs et/ou classes!

# Modules

Exemple d'architecture d'un programme en python:



# Packages

Le packages sont des ensembles de modules.

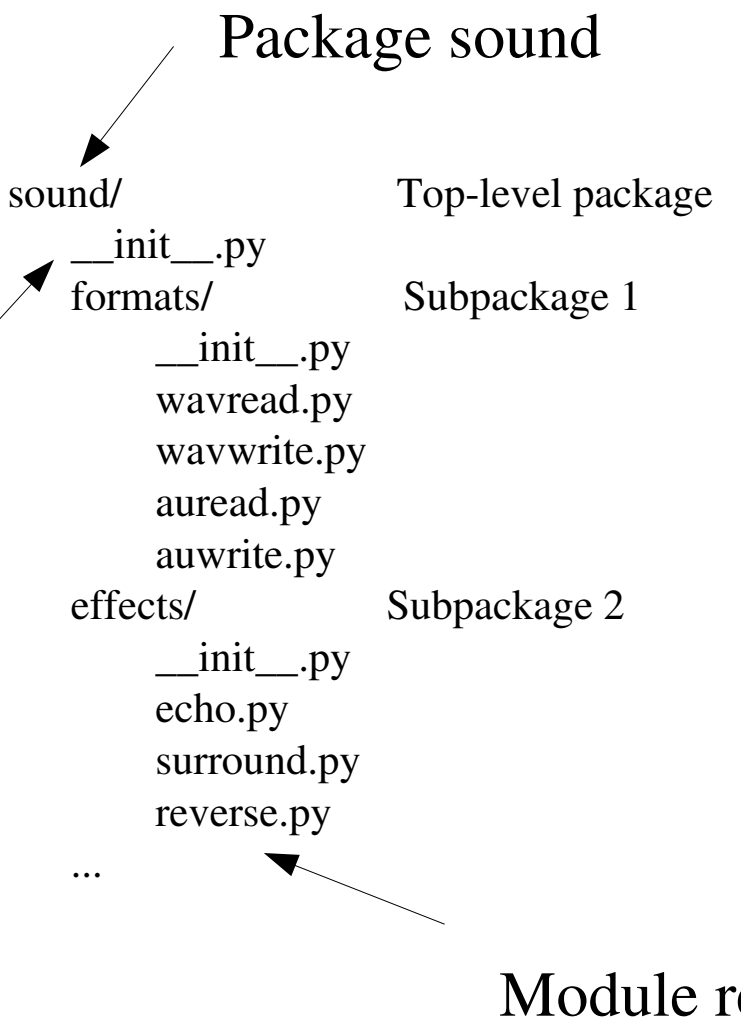
Concrètement, il s'agit simplement de répertoires contenant plusieus modules.

Pour consulter la structure d'un package, d'un module, ou d'un objet:

```
import os

dir( os )

help(os.chdir)
```



# Fonctions, modules, packages

Comment importer un module?

```
import os # NB: on laisse tomber le .py de os.py
os.chdir('/home/jan')
```

Une fonction dans un module?

```
from os import chdir
chdir('/home/jan/')
```

Un packgage?

```
import sound
import sound.effects #importer le sous paquet
```

# Programmation orientée objet

On peut tout à fait s'en passer en python!

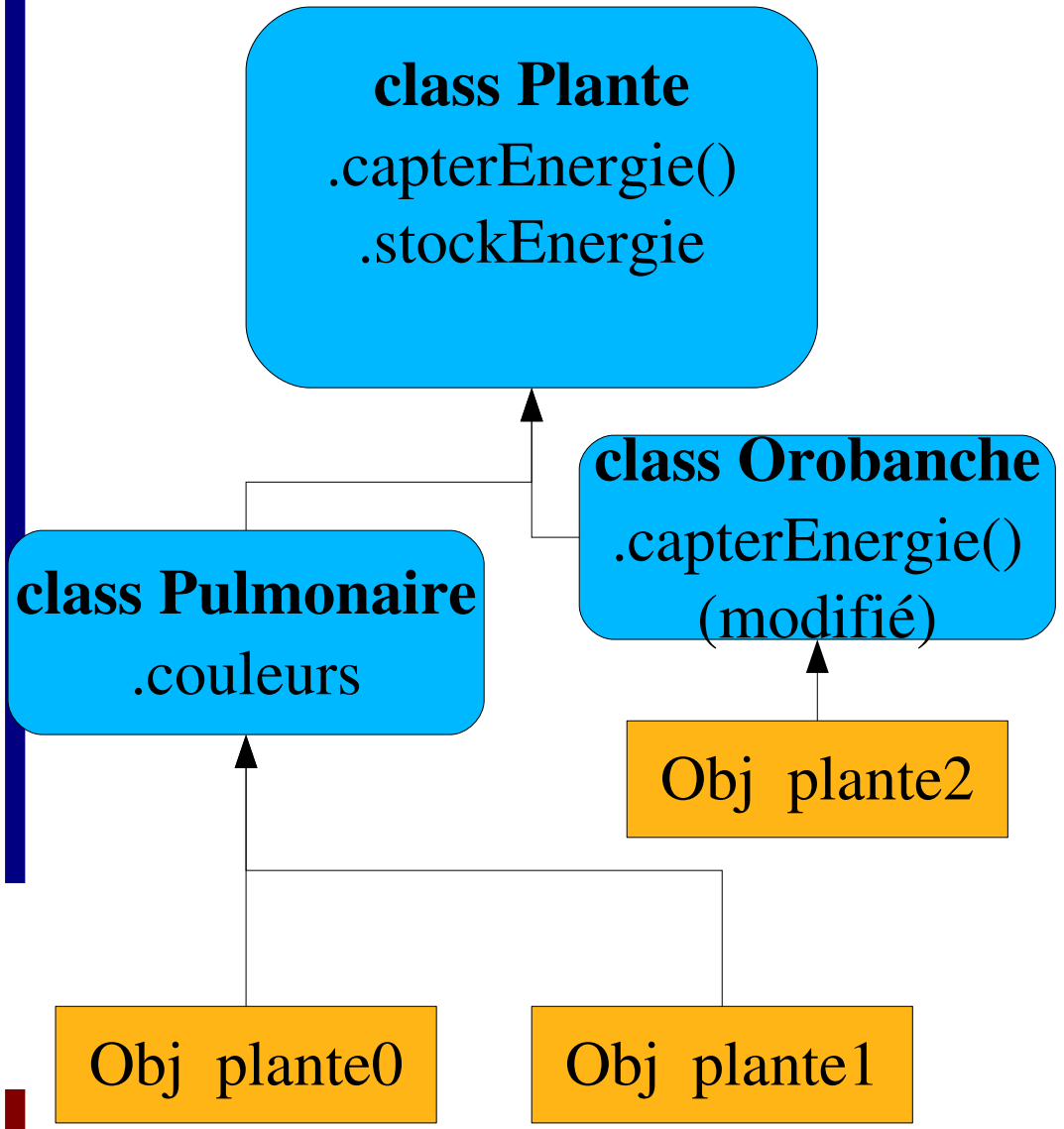
Mais on a beaucoup à y gagner!

- Permet de minimiser la redondance;
- Permet d'écrire de nouveaux programmes en jouant sur l'héritage du code existant et sa personnalisation plutôt qu'en le réécrivant;
- Profiter des web-frameworks OO du type (voir ci-après: tutorial Django). Ruby on Rails fonctionne aussi sur ce principe.
- Faire abstraction du SQL (voir ci-après ORM: object relaitonal mapping).

# Programmation orientée objet: terminologie

- un objet représente un concept ou toute entité du monde physique
  - Exemple: une voiture, une pensée, une personne, une fleur, une encyclopédie, un codex, un article ...
- un objet est définit par un identifiant unique, des méthodes (fonctions s'appliquant à lui-même), et des attributs (variables propres à lui-même);
- un objet est dérivé d'une classe: on dit qu'un objet est une instance d'une classe;
- une classe peut hériter des propriétés (méthodes, attributs) d'une ou de plusieurs autres classes.

# Programmation orientée objet




```
class Plante:
    def capterEnergie(self, lux):
        self.stockEnergie += lux
    stockEnergie = 1

class Pulmonaire(Plante):
    couleurs = ['violet', 'mauve']

class Orobanche(Plante):
    def capterEnergie(self, autre, lux):
        autre.stockEnergie -= lux
        self.stockEnergie += lux

plante0 = Pulmonaire()
plante0.capterEnergie(10)
plante2 = Orobanche()
plante2.capterEnergie(plante0, 5)
```

Voir autre exemple: ./oo/oo.py

# Gestion des exceptions

Lors qu'une erreur survient, python permet d'exécuter des instructions spécifiques. Les exceptions sont des classes.

```
try:
    a = 1 / 0
except:
    print('Une erreur!')
```

```
try:
    a = 1 / 0
except ZeroDivisionError:
    print('Division par zéro!')
except:
    print('Problème indéterminé!')
```

De l'aide? Liste des exceptions.

```
import exceptions
help(exceptions)
```

# Ou trouver des modules?

Python vient avec de nombreux modules, c'est-à-dire la bibliothèque standard.

http://docs.python.org/modindex.html

De plus, de très nombreux modules sont disponibles via plusieurs sources, voir par exemple le Python Package Index (PyPI)

http://pypi.python.org/pypi

Et encore de nombreux modules disséminés sur le web (sourceforge, google code, ...)

# Acquérir des modules

## Acquérir des modules python en toute simplicité? Utilisez Ubuntu (ou Debian)!

- Riche sélection: 1325 paquets liés à python
- Les paquets en C sont précompilés
- Installation
  - en un clic avec synaptic
  - en une commande: apt-get install <paquet>
- Mise-à-jour automatique

## Alternativement, utiliser easy_install / setuptools (gestionnaire de paquets python).

- Multi-platte-forme: Windows, MacOSX, Linux (aussi sous Ubuntu)
- Gestion des dépendances
- http://pypi.python.org/pypi/setuptools

Distibs Python: PythonXY (win, libre) ; Enthought (win/mac, commercial)

# Opérations sur le système de fichiers

- Opérations de haut-niveau
- Module shutils (bibliothèque standard)
- Exemples:

```
shutil.copy(src, dst) # copie de src vers dst

shutil.copytree(src, dst) # copie récursive

shutil.copytree(src, dst, shutil.ignore_patterns('*.pyc', \
                                                 'tmp*') )

shutil.rmtree(path) # effacement récursif

shutil.move(src, dst) # déplacement récursif
```

# Accéder à internet : urllib2

Le plus simple: urllib2 (Bibliothèque Standard)

- Multiprotocole (ftp, http, file, ...)
- Envoi de formulaires par POST et GET
- Supporte authentification, user-agent, etc.
- Mais relativement de bas niveau => beaucoup de code
- HOWTO: http://docs.python.org/howto/urllib2.html
- Exemple simple:

```
import urllib2
html = urllib2.urlopen('http://python.org/').read()
```

# Accéder à internet : mechanize

## Un navigateur programmable:

- Module: mechanize
- Package Ubuntu: python-mechanize
- URL: http://wwwsearch.sourceforge.net/mechanize/

## Fonctionnalités:

- Remplissage de formulaire simplifiée.
- Système de parsing et de suivi des liens
- Historique (méthodes .back() et .reload() ).
- Prise en compte automatique de robots.txt
- Gestion automatique de HTTP-Equiv et Refresh

## Exemple: ./network/cff.py

# Parser du HTML

## HTMLparser (Bibliothèque Standard)

Permet de parser du HTML... mais n'est pas tolère mal les horreurs que l'on trouve sur le web (les navigateurs sont tolérants): balises non fermés, overlappées, etc.

## Beautifulsoup règle ces soucis

- module: beautifulsoup
- Package ubuntu: python-beautifulsoup
- URL: http://www.crummy.com/software/BeautifulSoup/
- Fonctionne aussi pour le XML mal formatté!
- Exemple: ./network/cff.py

# Recherche via des API: ex Yahoo

- Module: pysearch
- URL: http://pysearch.sourceforge.net/
- Exemples:
  - ./network/pYsearch/exemple.py
  - ./network/pYsearch/websearch.py

# RSS : feedparser & pyrss2gen

Module: feedparser

Package Ubuntu: python-feedparser

URL: http://www.feedparser.org/

```
d = feedparser.parse('http://www.foo.ch/rss.xml')
for entry in d['entries']:
        print( entry['title'] + ' -> ' + entry['link'] )
```

Exemple: ./xml/rss/rss.py

NB: Pour générer des fils RSS: python-pyrss2gen

# Envoyer des Emails

Module: smtplib (biblio std.)

Exemple ./network/email-send.py

Basiquement:

```
s = smtplib.SMTP()
s.connect()
s.sendmail( 'from@foo.ch', ['to@foo.ch'], '...le message...' )
s.close()
```

NB:

- Sendmail doit être configuré
- Penser aux headers (cf. exemple)
- Relever une boite POP ou IMAP : imaplib (bib. Std.)

# Traiter des méta-données: XML

Python vient avec des parseurs DOM et SAX.

Exemple d'utilisation de DOM (lecture/écriture)

- ./xml/simple/2marcxml.py

Mais également:

- XSLT : python-libxslt1 (package Ubuntu)
  - Utile pour réutiliser stylesheets standards:
  - http://www.loc.gov/standards/marcxml/

# Bases de données

## Python supporte les principaux SGBDR:

- SQLite (stockage dans des fichiers)
  - Module: sqlite3 – biblio. std.
- PostgresSQL,
  - Module: pygresql
  - URL: http://www.pygresql.org/
  - Package Ubuntu: python-pygresql
- MySQL,
  - Module: MySQLdb
  - Package Ubuntu: python-mysqldb
  - URL: http://mysql-python.sourceforge.net/

- Et encore ...
- Oracle,
- Sybase,
- DB2

Exemple avec MySQLdb: ./databases/mysql.py

# Générer des documents: reportlab

Module: reportlab

Package Ubuntu: python-reportlab

URL: http://www.reportlab.org

- Solution très complète de génération de PDF
- «  Bas niveau  » et templating de texte et tables avec «  platypus  »
- Il s'agit d'un module compilé en C++
- Exemples ./documents/reportlab*

# Générer des documents: pod (appy)

Module: pod (Python Open Document)

URL: http://appyframework.org/pod.html

- Pod permet de générer des documents ODF (odt, ods, ...)
- Il se base sur des templates dans les même formats, édités à l'aide d'OpenOffice. C'est très confortable!
- Les variables python sont entrées en mode « change » activable depuis le menu « Edit / Changes / Records>
- Pod permet aussi de créer des tables et de faire des boucles en mettant du code python dans des notes
- Exemple: ./documents/pod*

# Générer des documents: uno-bridge

Module: uno

Package Ubuntu: python-uno

URL: http://udk.openoffice.org/python/python-bridge.html

- Uno permet de
  - Piloter Open Office (tâche de fond par exemple)
  - D'utiliser des macros python dans OOo
- Il permet d'exploiter les nombreuses fonctionnalités d'OOo
  - Génération de documents bureautiques textes, tableur, ...
  - Conversion de formats: word, excel, odt, ods, ...
  - Production de PDF
  - Diffs, extraction de texte (pour indexer p.ex.)
- Exemples: ./documents/uno*  || utilisé pour OOo Zotero Plugin
- NB: python-ooolib permet créer des docs ODF sans open office

# Python Image Library (PIL)

Package: PIL

Package Ubuntu: python-imaging

URL:http://effbot.org/zone/pil-index.htm

Manipulaiton d'images.

Exemples: ./imaging/*

NB:

- Les « nouveaux  » plugins de The Gimp (le PhotoShop libre) sont en python!
- Alternative à PIL: contrôler ImageMagick: python-pythonmagick

# Matlab-like: matplotlib

Module: matplotlib

Package Ubuntu: python-matplotlib

URL: http://matplotlib.sourceforge.net/

Calcul scientifique, production de graphiques, et manipulaiton d'images.

Matplotlib est très puissant et rapide (compilé en C). Il reprend de nombreux concepts, fonctionnalité, mots réservés et syntaxe de matlab!

Exemples: ./matplotlib/*

# Autres modules intéressants

- Gaming !
- SSH: Paramiko
- Son: sunau, pyao, etc.
- Hashs: Sha, md5
- Bluetooth: lightblue, PyBluez
- Interfaces: Curses, WX, Gtk
- Compression: Gzip, zlib, zipfile, tarfile
- Bioinformatique: Biopython
- Statistique: Rpy (pilotage de GNU R-project)
- Calculs symboliques: Sympy (primitive, résolutions d'équations, simplifications, ...)
- Graphes (réseaux): neworkX

# Applications utilisant python

**Inkscape :** logiciel de dessin vectoriel, retouche de PDF,  SVG, etc.  Extensions en python!

**The Gimp :** manipulation d'images, plugins en python

**Blender :** animations en 3D : plugins en python

**NodeBox (Mac/Carbon) :** animation en 3D basé sur python  http://nodebox.net

**OpenOffice :** permet d'écrire des macros en python  http://wiki.services.openoffice.org/wiki/Python

**Bibus :** logiciel de gestion de références bibliographiques http://bibus-biblio.sourceforge.net/

**Zotero:** ref. biblio., Open Office Plugin http://www.zotero.org/support/word_processor_integration

**PyBliographer:** ref. biblio., nombreuses formats, conversio http://pybliographer.org/

**PyBibTex:** Amélioratoin de BibTex en python (plus de formats, etc.) http://pybtex.sourceforge.net/

**Calibre:** managment d'ebooks http://calibre.kovidgoyal.net/

**Bazaar :** versionning décentralisé, simple à utiliser! Par Canonical (Ubuntu 9.04) http://bazaar-vcs.org/

**MindRetrieve :** search your personnal web http://www.mindretrieve.net/

**PyMol :** visualisation de molécules en 3D. http://pymol.sourceforge.net/

# Python et le web / bas niveau

La plupart des serveurs HTTP, comme Apache, sont écrits en C++, et nécessitent donc un pont pour exécuter du python. Il existe plusieurs stratégies:

- CGI (common gateway intreface) – OBSOLETE !
  - Lent: un interpréteur exécuté par requête
  - Pas pratique: peu (ou pas!) d'aide au développement
- Apache: mod_python, FastCGI, etc.
  - Rapide: différentes stratégies pour contourner le problème de CGI (NB: aussi plus rapide que PHP!)
  - Mais peu d'outils d'aide au développement
- WSGI (Web Server Gateway Interface) – LA BONNE SOLUTION !
  - Standardisation des ponts de bas niveau (authentification/session, upload fichiers, connexion BD ...)
  - Nombreux serveurs compatible en python et Apache
  - Les frameworks web sont tous compatibles WSGI

# Python et le web / haut-niveau

- Il existe de nombreux frameworks web python déployables par WSGI.
  Il s'agit d'environnements de développements complets.
  - Django : simple à apprendre
    - Constitué d'éléments fortement couplés (écrits exprès)
    - ORM (object relational mapping) puissant et simple à utiliser
    - Interface d'administration online (y.c. édition de données)
    - Le templating nécessite peu de conaissances en python.
    - GoogleAppEngine: http://code.google.com/appengine/
  - Pylons : le plus flexible
    - On peut choisir chaque composant (nombreuses possibilités)
    - Et donc, très puissant mais complexe à maîtriser!
  - CherryPy -> cf. Skeletonz
  - Zope (Commence à se faire vieux et pas d'ORM) -> cf. Plone

# Python et le web / Django

- Module: django
- Paquet Ubuntu: python-django
- URL: http://www.djangoproject.com/
- Tutorial: http://www.django-fr.org/documentation/tutorial01/
- Ci-après: ce tutorial en condensé, adapté sur Ubuntu
- Lié à l'exemple ./django/*

# Python et le web / Django

De l'intérêt d'utiliser Django:

- Applications portables (grâce au WSGI)
- Développement plus rapide, car aide intégrée:
  - Authentification, internationalisation, file upload, gestion/validation de formulaires, templating des pages, ORM, RSS, envoi emails ...
- Nombreux outils additionnels disponibles pour enrichir vos applications:
  - http://pypi.python.org/pypi?%3Aaction=search&term=django&submit=search
  - Ex: wikis, blogging, micorblogging, registration, notification, avatars, smileys, visitor tracking, watermarking, version control,  email-to-a-friend, ...

# Python et le web / Django

## Dans un répertoire:

django-admin startproject monsite

## Que ce passe-il? Django crée le projet:

monsite/

- __init__.py — Signale un package python (est vide!)
- manage.py — Outil linge de commande pour interagir avec Django
- settings.py — Fichier de configuration de Django
- urls.py — « Tables des matières » du site (basé sur des RegExp)

# Python et le web / Django

## On lance le serveur:

```
python manage.py runserver 8080
```

## Configuration : éditer settings.py

```
DATABASE_ENGINE = 'sqlite3' # MySQL, Oracle, Postgres...
DATABASE_NAME = './sqlite3/db.sqllite'
(optionnel cf. aussi: TEMPLATE_DIRS et INSTALLED_APPS)
```

## Initialiser les BD et password admin

```
python manage.py syncdb
```

## Pas besoin de relancer le serveur :-)

# Python et le web / Django

## Créer une application ( plusieurs applis par projet possibles ):

python manage.py startapp polls

## Que ce passe-il? Django crée une application:

polls/

__init__.py ← Signale un package python (est vide!)

models.py ← Strucure des données (modèle orienté objet)

views.py ← ~ Contenu des « pages ». Une fonction par « page »

## Django respecte le principe MVC (séparation de MVC...)

- Modèle: traitement des données, etc.
- Vue: interface utilisateur = présenter les données, etc.
- Contrôleur:  gestion des évènements + màj vue et modèle

# Python et le web / Django - Modèles

## Créer la structure de données: polls/models.py

- Voir fichier modèle simplifié models-simple.py
- Création de classes de données
- Object relationnal mapping:
  - Les classes correspondent aux tables SQL
  - Leurs attributs correspondent aux colonnes

## Activation des modèles

python manage.py sql polls

-> détermine et affiche le SQL adéquat

python manage.py syncdb

-> remet les tables à jour

# Python et le web / Django: API

```
python manage.py shell  # un shell pyhton normal, mais avec les bon paths
>>> from monsite.polls.models import Poll, Choice
>>> from datetime import datetime
>>> p = Poll(question="Quoi de neuf ?", pub_date=datetime.now())
        # revient à faire un INSERT en SQL... ici on a instancié un objet
>>> p.save() # contrairement à SQL, avec ORM il faut sauver
>>> p.id # La clé primaire? A été attribuée (genre AUTOINCREMENT)
1
>>> p.question
'Quoi de neuf ?'
>>> p.question = 'Quoi de neuf ????????' # revient à faire un UPDATE
>>> p.save()                                    # mais c'est plus court :-)
>>> Poll.objects.all()
[<Poll: Quoi de neuf ????????>]
```

# Python et le web / Django: API

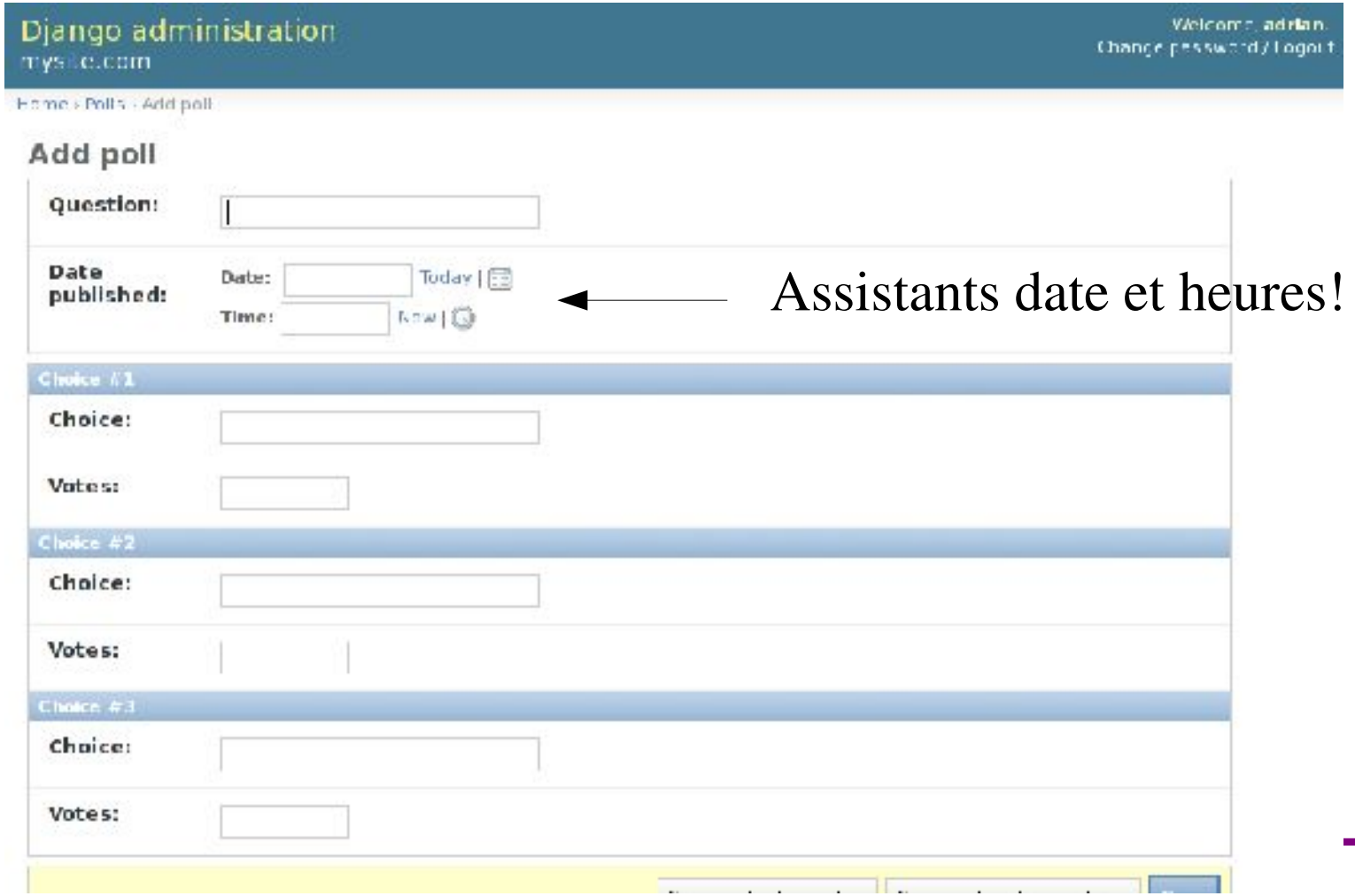
```
>>> p = Poll.objects.get(pk=1) # on fait un 'SELECT' sur la pk (primary key)
>>> p.was_published_today() # on appelle notre méthode de models.py
False
>>> p.choice_set.create(choice='Pas grand chose', votes=0) # on fait INSERT
<Choice: Pas grand chose>
>>> c = p.choice_set.create(choice='Il se fait tard', votes=0) # on fait INSERT
>>> c.poll # API d'accès au poll associé depuis un choix (fait une jointure), ex:
                # SELECT Poll.question FROM Choice, Poll WHERE Poll.id = Choice.poll
<Poll: Quoi de neuf ????????>
>>> p.choice_set.all() # et vice-versa_ access aux choices assoiciés a un poll
[<Choice: Pas grand chose>, <Choice: Le ciel>, <Choice: Il se fait tard>]
>>> c.delete()  # finalement, on supprime le choix qu'on vient de créer!
>>> Choice.objects.filter(poll__pub_date__year=2009) # SELECT ... WHERE ...
[<Choice: Pas grand chose>, <Choice: Le ciel>, <Choice: Il se fait tard>]
```

# Python web / Django: WebAdmin

Relativement simple à configurer (cf. tutorial Django)

C'est semblable à phpMyAdmin, mais en plus puissant:

cette interface tient compte des relations entre tables!




Assistants date et heures!

# Python web / Django: URLs mapping

Cf. monsite/urls.py

Le principe: renvoyer vers la bonne «  page  » (fonction) selon l'URL. Pour ce faire des expressions rationnelles sont utilisées.

On précise une liste d'expression rationnelles correspondant chacune à une fonction. Django les parcourt dans l'ordre et, c'est le premier match qui détermine la fonction choisie.

Syntaxe :

```
patterns ( '', (r'RegExp', 'fonction.a.appeler') ,...  )
```

# Python web / Django: les vues

- Cf. monsite/polls/views.py
- Chaque fonction correspond à une «  page  ».
- Exemple le plus simple possible de views.py:

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello World!")
```

# Python / Django: vues et templating

- Les vues peuvent appeler des templates:

```
def login_view(request):
    return render_to_response('polls/login.html', {... vars ... })
```

Un template

Un dico de variables

- Les templates servent à présenter, pas à programmer: ce n'est pas la même chose que l'on rencontre souvent en PHP (mélange de logique et présentation, sauf phpmvc.net)!

- Ils sont basés sur le HTML, et permettent:
  - De définir une hiérarchie de templates (par héritage)
  - D'y intégrer des variables, et de les reformater avec filtres
  - De faire des boucles (for), des tests (if/else) ...

# Python / Django: vues et templating

- Exemple de template:

Une variable

```
<html>
<head><title>{{story.headline}}</title></head>
<h1> {{story.headline|upper}} </h1>
{% for paragraph in story.pagagraphs %}
   <p>{{paragraph}}</p>
{% endfor %}
</html>
```

Un filtre, autre ex:
| striptags
| lower

Une boucle for
NB: if à le même genre de sytaxe

# Django: template inheritance

## Master template (master.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
   <link rel="stylesheet" href="style.css" />
   <title>{% block title %}My amazing site{% endblock %}</title>
</head>
<body>
   <div id="sidebar">
      {% block sidebar %}
      <ul>
         <li><a href="/">Home</a></li>
         <li><a href="/blog/">Blog</a></li>
      </ul>
      {% endblock %}
   </div>
   <div id="content">
      {% block content %}
      {% endblock %}
   </div>
</body>
</html>
```

Valeur par défaut

Valeur de remplacement

## Child template

```
{% extends "master.html" %}
{% block title %}My amazing blog{% endblock %}
{% block content %}
{% for entry in blog_entries %}
   <h2>{{ entry.title }}</h2>
   <p>{{ entry.body }}</p>
{% endfor %}
{% endblock %}
```

# Python web / Django: autentification

Créer un user:

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user('jan', \
                    'jan.krause@unige.ch', 'cccccc') # cccccc = pwd
>>> user.save()
```

Mettre à jour un user:

```
>>> u = User.objects.get(username__exact='jan')
>>> u.set_password('new password')
>>> u.save()
```

Pour supprimer un user: u.delete()

# Python web / Django: autentification

Voir les vues correspondantes dans l'exemple (views.py).

Login = 'jan'

Pwd = 'cccccc'

Pour tester cela:

- http://127.0.0.1:8080/polls/restricted
- http://127.0.0.1:8080/polls/logout

# Python web / Django: FORMS

L'intérêt d'un framework est de se simplifier le travail. Exemple avec les formulaires.

Définir un formulaire est plus simple et plus clair qu'en HTML... ou qu'en PHP...

A mettre dans models.py:

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField()
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

NB:
Permet une vérification auto de la validité!

# Python web / Django: FORMS

## Intégration du formulaire dans une vue (views.py)

```
def contact(request):
    if request.method == 'POST': # si qqn à cliqué sur envoyer...
                                 # on traite les données du FORM
        form = ContactForm(request.POST)
        if form.is_valid(): # toutes les règles de validité sont testées!
            # utiliser les données nettoyées:  form.cleaned_data
            subject = form.cleaned_data['subject']
            # ...
            return HttpResponseRedirect('/thanks/') # redirection
    else: # sinon on affiche un formulaire vide
        form = ContactForm()
        return render_to_response('contact.html', { 'form': form,})
```

# Python web / Django: FORMS

Templating d'un formulaire)

Django s'occupe de créer les champs... il nous reste à ajouter les balises <form>:

```
<form action="/contact/" method="POST">
{{ form.as_p }}
<input type="submit" value="Submit" />
</form>
```

# Python web / upload de fichiers

Créer une vue et un template pour produire un formulaire d'upload qui ressemble à cela:

```
<form action="URL_de_la_vue" method="post" enctype="multipart/form-data">
    <input type="submit" value="Upload" />
</form>
```

Les fichiers uploadés sont stockés dans 'request', donc créer un vue du style (attention à la sécurité!!):

```
def directupload(request):
    if request['method'] == 'POST':
        if 'file' in request.FILES:
            file = request.FILES['file']
            filename = file['filename']
            open('/chemin/stockage/' + filename, 'w').write(file['content'])
    return http.HttpResponseRedirect('une_URL_quelconque')
```

# Python web / Django: création de PDF

Via une vue et n'importe quel outil python (p.ex. reportlab):

```
from reportlab.pdfgen import canvas
from django.http import HttpResponse
def une_vue(request):
    response = HttpResponse(mimetype='application/pdf')
    response['Content-Disposition'] = 'attachment; ' +\
                                      'filename = nom_fichier.pdf'
    p = canvas.Canvas(response) #objet HttpReponse comme fichier
    p.drawString(100, 100, "Hello world!")
    p.showPage()
    p.save()
    return response
```

# Python web / Django: AJAX

Exemple: http://127.0.0.1:8080/polls/ajax

Cela fonctionne avec AJAX (Asynchronus Javascript and XML), et en pur Django si JavaScript est désactivé.

Les composants impliqués:

- Côté python, on utilise simplejson. JSON est un format d'échange, adapté au JavaScript
- La vue ajax_exemple (cf. views.py)
- Le toolkit AJAX Yahoo: http://developer.yahoo.com/yui/
- La partie JavaScript: ./django/static/ajax_example_1.js
  - Attention: le service statique ne fonctionne qu'en mode dvpt.

Une façon plus simple de faire de l'AJAX avec Django?

Utiliser la bibliothèque  jQuery p.ex.:  http://jquery.com/

# Exemples d'applications web

## Wiki: moin-moin

http://www.ooowiki.de/ , https://help.ubuntu.com/community/ ,

http://wiki.apache.org/general/

## CMS: Skeletonz, Plone (sous Zope)

http://orangoo.com/skeletonz/ , http://plone.net/sites

## Bibliothèques: CDS Invenio , Organisation: Indico

http://cdsweb.cern.ch , http://indico.cern.ch

## Divers: ajaxterm, GNU mailman (Swisslib, Unilist)

sudo apt-get install ajaxterm ||  sudo /etc/init.d/ajaxterm start  http://localhost:8022

## Pyjama (pur AJAX, semblable au Google Web Toolkit)

http://pyjs.org/ || http://pyjs.org/showcase/Showcase.html || http://pyjs.org/examples/

# Conclusion

# Vos questions?

Open Office UNO

Google AppEngine






Open Office Plugin

Blender Plugins

Gimp Python-Fu