Python – Cours 3

Jean-Yves Thibon

Université Paris-Est Marne-la-Vallée

ESIPE – IR3

Python et le web I

Pour le protocole `http` :

- `urllib` permet d'effectuer facilement des opérations simples (ouvrir une URL comme un fichier, GET et POST)
- `urllib2` permet des transactions plus avancées (authentification, cookies, redirections ...)
- `urlparse` analyse ou construit des URL complexes
- `SimpleHTTPServer` permet de monter un seveur en quelques lignes
- `cgi` permet d'écrire des serveurs de scripts
- `httplib`, `BaseHTTPServer`, `CGIHTTPServer` : fonctionnalités de plus bas niveau, à éviter si possible

Python et le web II

Pour le traitement du HTML :

- `HTMLParser` analyse HTML et XHTML
- `htmlentitydefs`
- Nombreuses contributions externes, comme `Beautiful Soup` et `request`
- Bon support du XML : `xml.*`

Le module `urllib` I

Fonctions de base pour lire des données à partir d'une URL.
Protocoles : *http, https, ftp, gopher, file*.

```
>>> import urllib
>>> s = urllib.urlopen(
                'http://igm.univ-mlv.fr/~jyt/python').read()
>>> print s
<html><body>
    <ul>
       <li> <a href="cours1.pdf">Cours 1</a>
       <li> <a href="td1.html">TD 1</a>
       <li> <a href="cours2.pdf">Cours 2</a>
   </ul>
</body></html>
```

Le module `urllib` II

`urllib.urlopen` renvoie un objet "file-like". Méthodes `read()`, `readline()`, `readlines()`, `fileno()`, `close()`, et en plus `info()`, `geturl()`.

```
>>> f = urllib.urlopen('http://igm.univ-mlv.fr/~jyt/python')
>>> f.fileno()
7
>>> dir(f)
['__doc__', '__init__', '__iter__', '__module__', '__repr__',
'close', 'fileno', 'fp', 'geturl', 'headers', 'info',
'next', 'read', 'readline', 'readlines', 'url']
>>> f.geturl()
'http://igm.univ-mlv.fr/~jyt/python/'

>>> list(f.headers)
['content-length', 'accept-ranges', 'server', 'last-modified',
'connection', 'etag', 'date', 'content-type']
```

Le module `urllib` III

```
>>> f.info()
<httplib.HTTPMessage instance at 0x40a84c4c>
>>> m = f.info()
>>> m.items()
[('content-length', '169'), ('content-language', 'fr'),
('accept-ranges', 'bytes'),
('server', 'Apache/2.0.50 (Unix) mod_ssl/2.0.50 OpenSSL/0.9.7c
('last-modified', 'Sun, 17 Feb 2008 12:33:33 GMT'),
('connection', 'close'),
('etag', '"217c009-a9-d8f7e140"'),
('date', 'Sun, 17 Feb 2008 12:59:09 GMT'),
('content-type', 'text/html')]
```

Le module `urllib` IV

Pour savoir si un nouveau document a été mis en ligne (le fichier `etag_python` doit avoir été initialisé) :

```
import urllib

url = 'http://igm.univ-mlv.fr/~jyt/python'
t = open('etag_python').read()
d = urllib.urlopen(url).info()
s = d['etag']
print d['last-modified']
if s <> t :
    print "La page du cours de Python a été modifiée"
    open('etag_python','w').write(s)
else: print "Aucune modification"
```

Le module `urllib` V

`urlopen` prend un paramètre optionnel, `data`. Si `data` est `None`, elle envoie une requête GET, sinon une requête POST.

```
>>> url='http://oeis.org'
>>> query={'q':'1,1,3,11,49,257',
           'language':'english', 'go':'Search'}
>>> data = urllib.urlencode(query)
>>> data
'q=1%2C1%2C3%2C11%2C49%2C257&go=Search&language=english'
>>> s = urllib.urlopen(url,data).read()

>>> urllib.unquote(data)
'q=1,1,3,11,49,257&go=Search&language=english'
```

Le module `urlparse` I

La manipulation des URLs est facilitée par le module `urlparse`

```
>>> x='http://www.google.fr/search?as_q=python&hl=fr&num=10
&btnG=Recherche+Google&as_epq=&as_oq=&as_eq=&lr=&cr=&as_ft=i
&as_filetype=pdf&as_qdr=all&as_occt=any&as_dt=i
&as_sitesearch=univ-mlv.fr&as_rights=&safe=images'
>>> from urlparse import *
>>> urlsplit(x)
('http', 'www.google.fr', '/search', 'as_q=python&hl=fr
&num=10&btnG=Recherche+Google&as_epq=&as_oq=&as_eq=
&lr=&cr=&as_ft=i&as_filetype=pdf&as_qdr=all&as_occt=any
&as_dt=i&as_sitesearch=univ-mlv.fr
&as_rights=&safe=images', '')
>>> urlunsplit(_)
'http://www.google.fr/search?as_q=python&hl=fr&num=10
&btnG=Recherche+Google&as_epq=&as_oq=&as_eq=&lr=
&cr=&as_ft=i&as_filetype=pdf&as_qdr=all&as_occt=any
&as_dt=i&as_sitesearch=univ-mlv.fr&as_rights=&safe=images'
```

Le module `urlparse` II

```
>>> urlparse.parse_qs(urlparse.urlsplit(x).query)
{'safe': ['images'], 'as_qdr': ['all'], 'as_dt': ['i\n'],
'btnG': ['Recherche Google'], 'as_ft': ['i\n'],
'num': ['10\n'], 'hl': ['fr'], 'as_occt': ['any'],
'as_q': ['python'], 'as_filetype': ['pdf'],
'as_sitesearch': ['univ-mlv.fr']}
```

URL schemes : file, ftp, gopher, hdl, http, https, imap, mailto, mms, news, nntp, prospero, rsync, rtsp, rtspu, sftp, shttp, sip, sips, snews, svn, svn+ssh, telnet, wais

Le module `urllib2` I

La fonction `urllib.urlopen` suffit pour les applications les plus courantes. Elle supporte les proxys pourvu qu'ils ne demandent pas d'authentification. Il suffit de positionner les variables d'environnement `http_proxy, ftp_proxy`, etc.

```
$ http_proxy="http://www.monproxy.com:1234"
$ export http_proxy
$ python
```

Pour un contrôle plus fin (authentification, user-agent, cookies) on peut utiliser `urllib2`.

Le module `urllib2` II

Le fonction `urllib2.urlopen` prend comme paramètre un objet de type `Request`

```
>>> import urllib2
>>> url='http://oeis.org'
>>> req = urllib2.Request(url)
>>> dir(req)
['_Request__fragment', '_Request__original', '__doc__', '__get
'__init__', '__module__', '_tunnel_host',
'add_data', 'add_header', 'add_unredirected_header', 'data',
'get_data', 'get_full_url', 'get_header', 'get_host',
'get_method', 'get_origin_req_host', 'get_selector', 'get_type
'has_data', 'has_header', 'has_proxy', 'header_items', 'header
'host', 'is_unverifiable', 'origin_req_host', 'port',
'set_proxy', 'type', 'unredirected_hdrs', 'unverifiable']
```

Le module `urllib2` III

En spécifiant les en-têtes, on peut par exemple se faire passer pour IE et envoyer un cookie :

```
import urllib2
url='http://www.hargneux.com/patteblanche.php'
req=urllib2.Request(url)
req.add_header('Accept','text/html')
req.add_header('User-agent','Mozilla/4.0
        (compatible; MSIE 5.5; Windows NT)')
req.add_header('Cookie',
       'info=En-veux-tu%3F%20En%20voil%E0%21')
handle=urllib2.urlopen(req)
```

Le module `urllib2` IV

Quand on ouvre une URL, on utilise un `opener`. On peut remplacer l'`opener` par défaut pour gérér l'authentification, les proxys, etc. Les `opener`s utilisent des `handlers`.
`build_opener` est utilisé pour créer des objets `opener`, qui permettent d'ouvrir des URLs avec des handlers spécifiques.
Les `handler`s peuvent gérer des cookies, l'authentification, et autres cas communs mais un peu spécifiques.
Les objets `Opener` ont une méthode `open`, qui peut être appelée directement pour ouvrir des urls de la même manière que la fonction urlopen.
`install_opener` peut être utilisé pour rendre l'objet `opener` l'opener par défaut. Cela signifie que les appels à `urlopen` l'utiliseront.

Exemple : authentification basique.
Pour demander une authentification, le serveur envoie le code d'erreur 401 et un en-tête du type
`www-authenticate: SCHEME realm="REALM"`
Le client doit alors re-essayer la requête avec un couple `(username, password)` correct pour le domaine (realm).
On peut gérer cela avec une instance de `HTTPBasicAuthHandler` et un opener pour utiliser ce handler.
`HTTPBasicAuthHandler` utilise un "password manager" pour gérer la correspondance entre les URIs et realms (domaines) et les couple (password, username).
En général un seul domaine (realm) par URI : `HTTPPasswordMgrWithDefaultRealm`.

Le module `urllib2` VI

```
password_mgr = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_mgr.add_password(None,
    top_level_url, username, password)
handler = urllib2.HTTPBasicAuthHandler(password_mgr)
opener = urllib2.build_opener(handler)

opener.open(a_url)

urllib2.install_opener(opener)
```

Pour en savoir plus : urllib2 - The Missing Manual

`http://www.voidspace.org.uk/python/articles/urllib2.shtml`

Il existe un module tiers (`request`) beaucoup plus pratique

`http://docs.python-requests.org/`

Côté serveur I

Les modules `BaseHTTPServer`, `SimpleHTTPServer` et `CGIHTTPServer` permettent de mettre en place un serveur web opérationnel en quelques lignes.
En fait, en une ligne :

```
[jyt@scriabine ~]$ python -m SimpleHTTPServer 8000
Serving HTTP on 0.0.0.0 port 8000 ...
scriabine - - [13/Oct/2012 19:19:10] "GET / HTTP/1.
...
```

Côté serveur II

Et sous forme de programme :

```
import SimpleHTTPServer
import SocketServer

PORT = 8000
Handler = SimpleHTTPServer.SimpleHTTPRequestHandler

httpd = SocketServer.TCPServer(("", PORT), Handler)

print "serving at port", PORT
httpd.serve_forever()
```

Côté serveur III

Un serveur de scripts minimal serait (cf. TD4)

```
#!/usr/bin/python

import os

from BaseHTTPServer import HTTPServer
from CGIHTTPServer import CGIHTTPRequestHandler
srvaddr = ('127.0.0.1',80)
server = HTTPServer(srvaddr, CGIHTTPRequestHandler)
server.serve_forever()
```

Le script cgi devra être placé dans un sous-répertoire cgi-bin, et le serveur devra avoir le droit d'exécution. Seul root peut lancer le serveur sur le port 80. En tant qu'utilisateur normal, on pourra le lancer sur un port libre, par exemple 8888. Le formulaire sera alors déclaré avec l'action

```
ACTION="http://127.0.01:8888/cgi-bin/monscript.cgi"
```

Traitement du HTML I

Pour extraire des informations d'une page web, on peut parfois se débrouiller avec des expressions régulières.
Mais on a aussi souvent besoin d'une analyse complète.
Il existe pour cela un module `HTMLParser`, qui exporte une classe du même nom.
On l'illustrera sur un exemple tiré de "Dive into Python" : traduire à la volée des pages web dans des dialectes farfelus (*chef, fudd, olde*). Les textes sont supposés en anglais, donc en ASCII.
On pourra aussi utiliser le touilleur de texte vu en TD pour brouiller une page web sans modifier sa mise en page.
Pour des exemples instructifs, voir
`http ://www.rinkworks.com/dialect/`

Traitement du HTML II

Exemple : Elmer Fudd (*cf.* Bugs Bunny)
Les dialectes sont définis par des substitutions, attribut `subs` d'une sous-classe `Dialectizer` de `BaseHTMLProcessor`, elle même dérivée de `HTMLParser`.

```
class FuddDialectizer(Dialectizer):
    """convert HTML to Elmer Fudd-speak"""
    subs = ((r'[rl]', r'w'),
            (r'qu', r'qw'),
            (r'th\b', r'f'),
            (r'th', r'd'),
            (r'n[.]', r'n, uh-hah-hah-hah.'))
```

Le module `HTMLParser` I

Il contient la classe `HTMLParser`, qui réalise l'analyse syntaxique du HTML.

Dès qu'un élément utile est identifié (un start tag `<a ...>` par exemple), une méthode (`handle_starttag, do_tag, ...`) est appelée.

`HTMLParser` analyse le HTML en 8 types de données, et appelle une méthose différente pour chaque cas :

Le module `HTMLParser` II

Start tag : `start_tagname` or `do_tagname` (ex. `start_pre` or `do_pre`). S'il la trouve, `HTMLParser` appelle cette méthode avec comme arguments la liste des attributs. Sinon, il appelle `unknown_starttag` avec le nom de la balise (tag) et la liste des attributs.
End tag : idem
Character reference : par exemple ` `. Méthode `handle_charref`.
Entity reference : par exemple `&copy;`.
Comment : `handle_comment`
Processing instruction : `<? ... >`. `handle_pi`.
Declaration : `<! ... >`. `handle_decl`
Text data : `handle_data`.

Le module `HTMLParser` III

Pour comprendre le fonctionnement, on peut utiliser la fonction de test de `sgmllib.py` (dont `HTMLParser` est une variante).

```
$ python /usr/lib64/python2.7/sgmllib.py index.html
start tag: <html>
data: '\n\t'
start tag: <body>
data: '\n\t\t'
start tag: <ul>
data: '\n\t\t\t'
start tag: <li>
data: ' '
start tag: <a href="cours1.pdf" >
data: 'Cours 1'
end tag: </a>
data: '\n\t\t\t'
start tag: <li>
data: ' '
start tag: <a href="td1.html" >
data: 'TD 1'
end tag: </a>
data: '\n\t\t\t'
[snip ...]
```

Le module `HTMLParser` IV

Exemple : `URLLister`
But : extraire les liens d'une page web.

```
from HTMLParser import HTMLParser

class URLLister(HTMLParser):
    def reset(self):
        HTMLParser.reset(self)
        self.urls = []

    def start_a(self, attrs):
        href = [v for k, v in attrs if k=='href']
        if href:
            self.urls.extend(href)
```

Le module HTMLParser V

Utilisation :

```
>>> from urllib import urlopen
>>> from urllister import URLLister
>>> s = urlopen('http://igm.univ-mlv.fr/~jyt').read()
>>> p = URLLister()
>>> p.feed(s)
>>> p.close()
>>> for u in p.urls: print u

http://igm.univ-mlv.fr/
http://www.cnrs.fr/
index_en.html
javascript:popup('http://www.univ-mlv.fr/fr/
index.php?rub=presentation&srub=planumlv&ssrub=batcopernic')
```

En appliquant ceci à la page du cours, on pourrait tester (avec une regexp) si un nouveau pdf a été mis en ligne et le récuperer automatiquement ...

Le problème I

Il s'agit de reproduire à l'identique le document HTML, en transformant seulement le texte, sauf s'il est encadré par un tag `pre`. Pour varier les plaisirs, on pourra traduire de l'anglais en texan :

`http ://www.discordia.ch/Programs/drawl.x`

```
(^|" ")"American" changeCase(" Amerkin");
(^|" ")"California" changeCase(" Caleyfornyuh");
(^|" ")"Dallas" changeCase(" Big D.");
(^|" ")"Fort Worth" changeCase(" Fowert Wurth");
(^|" ")"Houston" changeCase(" Useton");
(^|" ")"I don't know" changeCase(" I-O-no");
(^|" ")"I will"|" I'll" changeCase(" Ahl");
```

La classe `BaseHTMLProcessor` I

On commence par construire une classe dérivée qui ne fait rien : elle recompose la page analysée sans la modifier. On surchargera ensuite la méthode `handle_data` pour modifier le texte à notre convenance.

```
class BaseHTMLProcessor(HTMLParser):
    def reset(self):
        self.pieces = []
        HTMLParser.reset(self)

    def unknown_starttag(self, tag, attrs):
        strattrs = "".join([' %s="%s"' % (key, value)
                            for key, value in attrs])
        self.pieces.append("<%(tag)s%(strattrs)s>" % locals())

    def unknown_endtag(self, tag):
        self.pieces.append("</%(tag)s>" % locals())
```

La classe `BaseHTMLProcessor` II

```
def handle_charref(self, ref):
    self.pieces.append("&#%(ref)s;" % locals())

def handle_entityref(self, ref):
    self.pieces.append("&%(ref)s" % locals())
    if htmlentitydefs.entitydefs.has_key(ref):
        self.pieces.append(";")

def handle_data(self, text):  # A surcharger
    self.pieces.append(text)

def handle_comment(self, text):
    self.pieces.append("<!--%(text)s-->" % locals())
```

La classe `BaseHTMLProcessor` III

```
def handle_pi(self, text):
    self.pieces.append("<?%(text)s>" % locals())

def handle_decl(self, text):
    self.pieces.append("<!%(text)s>" % locals())

def output(self):
        """Return processed HTML as a single string"""
        return "".join(self.pieces)
```

La classe `BaseHTMLProcessor` IV

Exemple d'utilisation :

```
>>> from BaseHTMLProcessor import BaseHTMLProcessor
>>> from urllib import urlopen
>>> s = urlopen('http://igm.univ-mlv.fr/~jyt/python').read()
>>> p = BaseHTMLProcessor()
>>> p.feed(s)
>>> p.close()
>>> p.output()
'<html>\n\t<body>\n\t\t<ul>\n\t\t\t<li> <a href="cours1.pdf">
Cours 1</a>\n\t\t\t<li> <a href="td1.html">TD 1</a>\n\t\t\t<li
<a href="cours2.pdf">Cours 2</a>\n\t\t</ul>\n\
t</body>\n</html>\n\n\t\t\n'
>>>
```

Commentaires sur la syntaxe I

Ce code utilise quelques astuces typiquement pythonesques. `locals()` et `globals()` renvoient des dictionnaires de variables locales et globales ...

```
>>> def f(x):
        y = 'toto'
        print locals()

>>> f(8)
{'y': 'toto', 'x': 8}
>>> print globals()
{'f': <function f at 0x402d35a4>, '__builtins__':
<module '__builtin__' (built-in)>, '__name__': '__main__',
'__doc__': None}
>>> dir()
['__builtins__', '__doc__', '__name__', 'f']
```

Commentaires sur la syntaxe II

Les lignes

```
if __name__ == "__main__":
        for k, v in globals().items():
                print k, "=", v
```

à la fin de `BaseHTMLProcessor.py` produisent l'effet suivant, quand le programme est lancé en ligne de commande :

```
$ python BaseHTMLProcessor.py
__copyright__ = Copyright (c) 2001 Mark Pilgrim
HTMLParser = HTMLParser.HTMLParser
__license__ = Python
__builtins__ = <module '__builtin__' (built-in)>
__file__ = BaseHTMLProcessor.py  [ ... snip ...]
```

Commentaires sur la syntaxe III

Rappel : formatage par dictionnaire

```
>>> d = {'animal':'cheval', 'parent':'cousin',
         'aliment':'foin', 'jour':'dimanche'}

>>> s = ''''Le %(animal)s de mon %(parent)s
ne mange du %(aliment)s que le %(jour)s'''
>>> print s % d
Le cheval de mon cousin
ne mange du foin que le dimanche
>>>
```

Commentaires sur la syntaxe IV

On peut donc utiliser `locals()` pour remettre en place les attributs des balises sans avoir à les connaître :

```
def unknown_starttag(self, tag, attrs):
    strattrs = "".join([' %s="%s"' % (key, value)
                        for key, value in attrs])
    self.pieces.append("<%(tag)s%(strattrs)s>" % locals())
```

C'est ce procédé qui permet de reconstituer (essentiellement) le HTML qu'on ne souhaite pas modifier.

Commentaires sur la syntaxe V

Pourquoi “essentiellement” ? A cause des "guillemets" :

```
>>> htmlSource = """
...     <html>
...     <head>
...     <title>Test page</title>
...     </head>
...     <body>
...     <ul>
...     <li><a href=index.html>Home</a></li>
...     <li><a href=toc.html>Table of contents</a></li>
...     <li><a href=history.html>Revision history</a></li>
...     </body>
...     </html>"""
```

Commentaires sur la syntaxe VI

```
>>> from BaseHTMLProcessor import BaseHTMLProcessor
>>> parser = BaseHTMLProcessor()
>>> parser.feed(htmlSource)
>>> print parser.output()
<html>
<head>
<title>Test page</title>
</head>
<body>
<ul>
<li><a href="index.html">Home</a></li>
<li><a href="toc.html">Table of contents</a></li>
<li><a href="history.html">Revision history</a></li>
</body>
</html>
```

Le dialectiseur I

La classe `Dialectizer` dérive de `BaseHTMLProcessor`. Son rôle est de "traduire" le texte de la page, sauf lorsqu'il doit être rendu verbatim (<pre>...</pre>).
Il fait donc un attribut `verbatim` qui permet de savoir si l'on doit traduire ou pas :

```
def start_pre(self, attrs):
    self.verbatim += 1
    self.unknown_starttag("pre", attrs)

def end_pre(self):
    self.unknown_endtag("pre")
    self.verbatim -= 1
```

Ceci étant acquis, on peut maintenant surcharger `handle_data` :

Le dialectiseur II

```
def handle_data(self, text):
    self.pieces.append(self.verbatim
                       and text
                       or self.process(text))
```

La méthode `process` dépendra de la traduction désirée.
On remarquera l'usage astucieux des booléens

```
>>> (1==1) and 'toto'
'toto'
>>> (1==0) and 'toto'
False
>>> (1==0) or 'toto'
'toto'
>>> (1==0) and 'toto' or 'titi'
'titi'
```

Le dialectiseur III

Explication (attention à l'ordre !) :

```
>>> 'toto' and (1==1)
True
>>> 'toto' and (1==0)
False
>>>
```

La sémantique est

```
x or y   -->  if x is false, then y, else x
x and y  -->  if x is false, then x, else y
```

Le dialectiseur IV

Le code du dialectiseur :

```
class Dialectizer(BaseHTMLProcessor):
    subs = ()

    def reset(self):
        self.verbatim = 0
        BaseHTMLProcessor.reset(self)

    def start_pre(self, attrs):
        self.verbatim += 1
        self.unknown_starttag("pre", attrs)

    def end_pre(self):
        self.unknown_endtag("pre")
        self.verbatim -= 1
```

Le dialectiseur V

```
def handle_data(self, text):
    self.pieces.append(self.verbatim and text
                                    or self.process(text))

def process(self, text):
    for fromPattern, toPattern in self.subs:
        text = re.sub(fromPattern, toPattern, text)
    return text
```

Le programme principal I

On définit une fonction`translate` qui prend comme arguments une URL et un dialecte

```
def translate(url, dialectName="chef"):
    import urllib
    sock = urllib.urlopen(url)
    htmlSource = sock.read()
    sock.close()
    parserName = "%sDialectizer" % dialectName.capitaliz
    parserClass = globals()[parserName]
    parser = parserClass()
    parser.feed(htmlSource)
    parser.close()
    return parser.output()
```

On notera l'astuce pour construire le nom du parseur : la fonction `translate` n'a pas besoin de savoir quels sont les dialectiseurs déjà définis.

Le programme principal II

```
def test(url):
    """test all dialects against URL"""
    for dialect in ("chef", "fudd", "olde"):
        outfile = "%s.html" % dialect
        fsock = open(outfile, "wb")
        fsock.write(translate(url, dialect))
        fsock.close()
        import webbrowser
        K=webbrowser.Konqueror() # necessaire chez moi
        webbrowser.register('konqueror',None,K)
        K.open_new(outfile)
#       webbrowser.open_new(outfile)
```

Le programme principal III

Le module `webbrowser` permet à Python de lancer un navigateur pour afficher le résultat

```
def test(url):
    """test all dialects against URL"""
    for dialect in ("chef", "fudd", "olde"):
        outfile = "%s.html" % dialect
        fsock = open(outfile, "wb")
        fsock.write(translate(url, dialect))
        fsock.close()
        import webbrowser
        K=webbrowser.Konqueror() # necessaire chez moi
        webbrowser.register('konqueror',None,K)
        K.open_new(outfile)
```

Brouilleur de page web : le code complet I

```
# module touille.py
import random, re

p = re.compile('(\w)(\w+)(\w)', re.M|re.L|re.U)

def touille(m):
    milieu = list(m.group(2))
    random.shuffle(milieu)
    return m.group(1) + ''.join(milieu) + m.group(3

def blurr(s):
    return p.sub(touille,s)
```

```
#!/usr/bin/env python
"""Web page blurrer for Python

This program is adapted from "Dive Into Python", a free Python book for
experienced programmers.  Visit http://diveintopython.org/ for the
latest version.

New version using HTMLParser and including images.
"""

__author__ = "Mark Pilgrim (mark@diveintopython.org)"
__updated_by__ = "Jean-Yves Thibon"
__version__ = "$Revision: 1.3 $"
__date__ = "$Date: 2009/02/12 $"
__copyright__ = "Copyright (c) 2001 Mark Pilgrim"
__license__ = "Python"

import re
from newHTMLProcessor import BaseHTMLProcessor
from touille import blurr
import codecs
from urlparse import urljoin

class Dialectizer(BaseHTMLProcessor):
        subs = ()
        def __init__(self,root_url=None):# ajout
                BaseHTMLProcessor.__init__(self)
                self.url = root_url

        def __compl(self, x):# pair (key, value)
                if x[0] == 'href' or x[0] == 'src':
                        return (x[0], urljoin(self.url,x[1]))
                else: return x

        def reset(self):
                # extend (called from __init__ in ancestor)
                # Reset all data attributes
                self.verbatim = 0
                BaseHTMLProcessor.reset(self)

        def handle_starttag(self,tag,args):
                if self.url:
                        args = [self.__compl(x) for x in args]
                if tag in ["pre","script","style"]: self.verbatim += 1
                strattrs = "".join([' %s="%s"' % (key, value) for key, value in args])
                self.pieces.append("<%(tag)s%(strattrs)s>" % locals())
```

```
        def handle_endtag(self,tag):
                # called for every </pre> tag in HTML source
                # Decrement verbatim mode count
                if tag in ["pre","script","style"]: self.verbatim -= 1
                self.pieces.append("</%(tag)s>" % locals())

        def handle_data(self, text):
                # override
                # called for every block of text in HTML source
                # If in verbatim mode, save text unaltered;
                # otherwise process the text with a series of substitutions
                self.pieces.append(self.verbatim and text or self.process(text))

        def process(self, text):
                # called from handle_data
                text = blurr(text)
                return text


def translate(url):
        """fetch URL and blurr"""
        import urllib
        sock = urllib.urlopen(url)
        htmlSource = sock.read()
        sock.close()
        s = unicode(htmlSource,'utf-8')
        parser = Dialectizer(url)#test
        parser.feed(s)
        parser.close()
        return parser.output()

def test(url):
        """test  against URL"""
        outfile = "touillage.html"
        fsock = codecs.open(outfile, "wb",encoding='UTF-8')
        fsock.write(translate(url))
        fsock.close()
        import webbrowser
        K = webbrowser.Konqueror()
        webbrowser.register('konqueror',None,K)

        K.open_new(outfile)

if __name__ == "__main__":
        s = test("http://igm.univ-mlv.fr/~jyt/")
```