Web Scraping

Antonio Sarasa Cabezuelo

Recuperación de recursos web

- Se van a mostrar varias formas de recuperar recursos que se encuentran en la web:
  - Recuperación via http
  - Recuperación con el módulo urllib.
  - Recuperación con el módulo Request

Recuperación via http

- Un socket proporciona una conexión de doble sentido entre dos programas, de manera que es posible tanto leer como en el mismo socket. Si se escribe algo en un socket, es enviado hacia la aplicación que está al otro lado del socket. Si se lee desde un socket, se obtienen los datos que la otra aplicación ha enviado.
- En Python existe un soporte integrado para los sockets.

Recuperación via http

- El protocolo para el intercambio de recursos web es HTTP.
- En este protocolo para pedir un documento a un servidor web se escribe una línea en la cual el segundo parámetro es la página web que estamos solicitando, y a continuación enviamos una línea en blanco:

GET servidor web HTTP/1.0

Recuperación via http

- El servidor web responderá con una cabecera que contiene cierta información acerca del documento y una línea en blanco, seguido por el contenido del documento.

Recuperación via http

- En el siguiente ejemplo se va a recuperar una imagen. Para ello se realiza una conexión con el puerto 80 del servidor y se envía el comando GET seguido por una línea en blanco. Una vez enviada la línea en blanco, se utiliza un bucle que recibe los datos desde el socket en bloques de 512 caracteres y se imprime en pantalla hasta que no quedan más datos por leer (recv() devuelve una cadena vacía).

Recuperación via http

```
import socket
misock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
misock.connect(('www.purotrend.com', 80))
misock.send('GET http://purotrend.com/wp-content/uploads/2014/09/tren-2.jpg  HTTP/1.0\n\n')
while True:
    datos = misock.recv(512)
    if ( len(datos) < 1 ) :
        break
    print datos

misock.close()
```

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
Last-Modified: Tue, 16 Sep 2014 18:35:35 GMT
ETag: "541882f7-b27a"
Server: Varnish
X-Cacheable: YES
Content-Length: 45690
Accept-Ranges: bytes
Date: Wed, 30 Sep 2015 19:27:08 GMT
X-Varnish: 1704897831
Via: 1.1 varnish
Connection: close
age: 0
X-Cache: MISS

ÿØÿà ┼JFIF            ÿþ ;CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality
= 70
ÿÛ C
••▪•-
▪▪▪♪
```

Recuperación via http

- La salida comienza con las cabecera que el servidor web envía para describir el documento. Por ejemplo Content-Type indica que es imagen en formato jpeg (image/jpeg). A continuación de la cabecera, añade una línea en blanco para indicar el final de la misma, y envía los datos reales del fichero.

Recuperación via http

- Se puede mejorar el programa guardando los datos en una cadena, recortando las cabeceras y luego guardando los datos de la imagen en un archivo.

Recuperación via http

```
import socket
import time
misock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
misock.connect(('www.purotrend.com', 80))
misock.send('GET http://purotrend.com/wp-content/uploads/2014/09/tren-2.jpg HTTP/1.0\n\n')
contador = 0
imagen = "";
while True:
    datos = misock.recv(5120)
    if ( len(datos) < 1 ) : break
    imagen = imagen + datos
misock.close()

# Búsqueda del final de la cabecera
pos = imagen.find("\r\n\r\n");
print imagen[:pos]

# Saltar la cabecera y guardar los datos de la imagen
imagen = imagen[pos+4:]
manf = open("tren.jpg","wb")
manf.write(imagen);
manf.close()
```

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
Last-Modified: Tue, 16 Sep 2014 18:35:35 GMT
ETag: "541882f7-b27a"
Server: Varnish
X-Cacheable: YES
Content-Length: 45690
Accept-Ranges: bytes
Date: Wed, 30 Sep 2015 19:41:43 GMT
X-Varnish: 1704925541 1704897831
Via: 1.1 varnish
Connection: close
age: 0
X-Cache: HIT
X-Cache-Hits: 1
```

Recuperación via http

- Una vez que el programa termina, se pueden ver los datos de la imagen abriendo el archivo tren.jpg con un editor de imágenes.

Recuperación con el módulo urllib

- urllib es una librería que permite tratar una página web de forma parecida a la apertura de un fichero, de forma que la librería gestiona de manera transparente todo lo referente al protocolo HTTP y los detalles de la cabecera.

Recuperación con el módulo urllib

- Una vez que la página web ha sido abierta con urllib.urlopen, se puede tratar como un archivo y leer a través de ella usando un bucle for.
- Cuando se recupera la página web, sólo se accede al contenido puesto las cabeceras aunque se envián, el código de urllib se queda con ellas y sólo devuelve los datos.

Recuperación con el módulo urllib

- En el siguiente ejemplo se recupera la información de una página web

```
import urllib
manf = urllib.urlopen('http://www.fdi.ucm.es')
for linea in manf:
    print linea.strip()
```

Recuperación con el módulo urllib

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Facultad de InformÃ¡tica. Universidad Complutense de Madrid</title>
<meta charset="UTF-8" />
<meta name="description" content="La Universidad Complutense de Madrid es una in
stituciÃ³n de larga trayectoria y amplio reconocimiento social que aspira a situ
arse entre las primeras universidades de Europa y a consolidarse como centro de
referencia para el continente latinoamericano" />
<meta name="keywords" content="Universidad Complutense de Madrid, Complutense, U
CM, EducaciÃ³n, InnovaciÃ³n, Universidad de excelencia, FormaciÃ³n, Grado, MÃ¡st
er, Doctorado, Postgrado" />
<link rel="apple-touch-icon" href="/themes/ucm3/media/img/logoucm.ico" />
<link rel="shortcut icon" href="/themes/ucm3/media/img/logoucm.ico" />
<link rel="icon" href="/themes/ucm3/media/img/logoucm.ico" />
<link type="text/css" media="screen" rel="stylesheet" href="/themes/ucm3/css/nor
mal.css?a=1" />
<link type="text/css" media="screen" rel="stylesheet" href="/themes/ucm3/css/por
tada.css?a=1" />
<link type="text/css" media="screen" rel="stylesheet" href="/themes/ucm3/css/bar
ra.css" />
```

Recuperación con el módulo urllib

- En el siguiente programa se procesa la información de un página web y se calcula la frecuencia de cada palabra:

```
import urllib
contadores = dict()
manf = urllib.urlopen('http://www.fdi.ucm.es')
for linea in manf:
    palabras = linea.split()
    for palabra in palabras:
        contadores[palabra] = contadores.get(palabra,0) + 1
print contadores
```

Recuperación con el módulo urllib

```
{'!function(d,s,id){var': 1, "setTimeout('_gaq.push([\\'_trackEvent\\',": 2, 'co
nferencias': 2, 'ga': 2, '[];': 2, "seconds\\'])',5000);": 2, "'ucm.es');": 1, '
s.parentNode.insertBefore(ga,': 2, 'class="menu2_linea2">FDI</span></a></li>': 1
, 'href="/presentacion">Presentaci\xc3\xb3n</a></li>': 1, 'href="http://www.ucm.
es/"': 1, "'text/javascript';": 2, 'center;': 4, '}#barra': 1, 'jq(document).rea
dy(function()': 1, '/>(<span': 1, "url('https://www.ucm.es/themes/ucm3/media/img
/cabecera-1.png')": 1, 'id="search"': 1, 'propios</a></li>': 1, 'lang="es">': 1,
 'style="text-decoration:': 1, 'larga': 1, 'turno': 1, 'icon"': 1, '</ul>': 11,
'<script': 4, 'height="75"': 2, 'window.jQuery)': 1, 'src="https://web.fdi.ucm.e
s/Avisos_Der.aspx"': 1, 'style="width:': 1, "('https:'": 2, 'width="300"': 1, '/
></div>': 3, 'null;': 1, "_gaq.push(['_trackPageview']);": 2, 'curso': 1, '+': 2
, "src='/data/cont/media/portada/fijas/26-portada-5.jpg'": 1, '<header>': 1, 'id
="area_dcha">': 1, 'href="/programacion-docente">Programacion': 1, '<form': 1, "
'twitter-wjs');": 1, 'href="/titulos-propios">T\xc3\xadtulos': 1, 'rgba(18,37,98
,': 1, 'title="Universidad': 1, 'Innovaci\xc3\xb3n,': 1, 'content="Universidad':
 1, 'social': 1, 'ma&ntilde;ana?,': 1, '{': 16, '2015-2016</a></div>': 1, 'ya!<b
```

Recuperación con el módulo urllib

- La librería urllib también se puede utilizar para recuperar ficheros que no son de texto como un archivo de imagen o de video, y para los que se quiere hacer una copia de la URL en un archivo local.

Recuperación con el módulo urllib

- En estos casos se usa el método read para descargar el contenido completo del documento en una variable de tipo cadena y luego escribir la información a un archivo local.

```
img = urllib.urlopen('http://purotrend.com/wp-content/uploads/2014/09/tren-2.jpg').read()
manf = open('portada.jpg', 'wb')
manf.write(img)
manf.close()
```

Recuperación con el módulo urllib

- El programa lee todos los datos de una sola vez a través de la red y los almacena en la variable img en la memoria principal y luego abre el fichero portada.jpg y escribe los datos en el disco. Esto funciona sólo si el tamaño del fichero es menor que el tamaño de la memoria del PC.

Recuperación con el módulo urllib

- Pero si se trata de un fichero demasiado grande el programa puede fallar cuando el equipo se quede sin memoria. Para evitar agotar la memoria, se puede recuperar los datos en bloques, y luego escribir cada bloque en el disco antes de recuperar el siguiente. De este modo el programa puede leer archivos de cualquier tamaño sin usar toda la memoria del equipo.

Recuperación con el módulo urllib

- En este ejemplo, se leen 100.000 caracteres cada vez y luego se escriben esos caracteres en el archivo portada.jpg, antes de recuperar los 100.000 caracteres siguientes.

```
import urllib
img = urllib.urlopen('http://purotrend.com/wp-content/uploads/2014/09/tren-2.jpg')
manf = open('portada.jpg', 'wb')
tamano = 0
while True:
    info = img.read(100000)
    if len(info) < 1 : break
    tamano = tamano + len(info)
    manf.write(info)
print tamano,'caracteres copiados.'
manf.close()
```

Recuperación con el módulo urllib

- Una versión más sofisticada sería:

```
import os
import urllib

print 'Entrar una url'
urlstr = raw_input().strip()
img = urllib.urlopen(urlstr)
words = urlstr.split('/')
fname = words[-1]

if os.path.exists(fname) :
   if raw_input('Reemplazar archivo '+fname+' (S/n)?') != 'S' :
      print 'Datos no copiados'
      exit()
   print 'Reemplazando',fname

fhand = open(fname, 'wb')
size = 0
while True:
    info = img.read(100000)
    if len(info) < 1 : break
    size = size + len(info)
    fhand.write(info)

print size,'caracteres copiados',fname
fhand.close()
```

Recuperación con el módulo requests

- El módulo requests permite bajarse archivos de la red de una forma transparente.
- No se encuentra instalado por defecto con Python.

Recuperación con el módulo requests

- La función get() del módulo requests toma una cadena que representa la url que se quiere descargar. El resultado de la llamada es un objeto de tipo Response que contiene la respuesta que el servidor web devuelve como respuesta a la petición.

```
import request
res=requests.get('http://www.fdi.ucm.es')
if res.status.code==requests.codes.ok:
    print len(res.text)
    print res.text[:250]
```

Recuperación con el módulo requests

- En el ejemplo además de recuperar la pagina descrita, se ha comprobado que la descarga se ha realizado con éxito chequeando el valor del atributo status_code del objeto Response. Si el valor que toma es requests.codes.ok significa que se ha realizado correctamente, y la descargada se ha almacenado como una cadena en la variable text del objeto Response.

Recuperación con el módulo requests

- Otra forma de comprobar si una descarga se ha realizado con éxito consiste en utilizar el método raise_for_status() del objeto Response. Este método lanzará una excepción si se ha producido algún error en la descarga y no hará nada en caso de que la descarga se haya realizado con éxito.

Recuperación con el módulo requests

- Es por ello que una buena práctica consiste en encerrar la llamada al método raise_for_status() entre un try/except con el objetivo de tratar los casos en que se produzca una descarga errónea.

```
import request
res=requests.get('http://www.fdi.ucm.es')
try:
    res.raise_for_status()
except:
    print "Hubo un problema"
```

Recuperación con el módulo requests

- Observar que es la llamada al método raise_for_status() hay que realizarlo siempre después de llamar al método requests.get() dado que el objetivo es asegurarse que la descarga se realizó con éxito antes de que el programa continúe su ejecución.

Recuperación con el módulo requests

- Una vez que ha realizado la descarga, lo que interesa es guardar el contenido en un archivo. Para ello se puede utilizar la función open() y el método write(). En este sentido lo primero que hay que hacer es abrir un fichero en modo “wb”(escribir un modo binario), y a continuación escribir al fichero usando un bucle for que utilice el método iter_content del objeto Response.

Recuperación con el módulo requests

```
import request
res=requests.get('http://www.fdi.ucm.es')
try:
    res.raise_for_status()
    archivo=open("Archivo.txt", "wb")
    for bloque in res.iter_content(10000):
        archivo.write(bloque)
    archivo.close()
except:
    print "Hubo un problema"
```

Recuperación con el módulo requests

- Observar que el método iter_content recupera bloques de contenido en cada iteración a través del bucle. Cada bloque es un conjunto de bytes de datos en un tamaño igual al especificado. Así mismo observar que el método write() retorna el número de bytes escritos al fichero.

Web Scraping

- El “web scraping” consiste en escribir un programa que finge ser un navegador web y recupera páginas, examinando luego los datos de esas páginas para encontrar ciertos patrones.

Web Scraping

- Por ejemplo los buscadores:
  - Buscan en el código de una página web, extraen los enlaces a otras páginas y recuperan esas páginas, extrayendo los enlaces que haya en ellas y así sucesivamente.
  - Usan la frecuencia con que las páginas que encuentra enlazan hacia una página concreta para calcular la "importancia" de esa página, y la posición en la que debe aparecer dentro de sus resultados de bússqueda.

Web Scraping

- Se van a estudiar dos formas de analizar páginas web:
  - Mediante expresiones regulares.
  - Mediante la librería BeautifulSoup.

Análisis mediante expresiones regulares

- Un forma fácil de analizar HTML consiste en utilizar expresiones regulares para hacer búsquedas repetidas que extraigan subcadenas coincidentes con un modelo concreto.

Análisis mediante expresiones regulares

- Considerar por ejemplo una página web que contiene enlaces y se quieren detectar dicho enlaces. Para ello se podría construir una expresión regular que busque y extraiga los valores de los enlaces:

  href="http://.+?"

Análisis mediante expresiones regulares

- La expresión regular busca cadenas que comiencen por “href=”http://”, seguido de uno o más caracteres (“.+?”), seguidos por otra comilla doble. El signo de interrogación añadido a “.+?” indica que se busca la cadena coincidente más pequeña posible. Por último se añaden paréntesis a la expresión regular para indicar qué parte de la cadena localizada se quiere extraer.

Análisis mediante expresiones regulares

- Se puede construir el siguiente programa:

```
import urllib
import re
url = raw_input('Introduzca - ')
html = urllib.urlopen(url).read()
enlaces = re.findall('href="(http://.*?)"', html)
for enlace in enlaces:
    print enlace
```

Análisis mediante expresiones regulares

```
>>>
Introduzca - http://www.fdi.ucm.es
http://www.ucm.es/
http://www.campusmoncloa.es/en/
http://informatica.ucm.es/facultad
http://informatica.ucm.es/estudiar
http://informatica.ucm.es/estudiantes
http://biblioteca.ucm.es/fdi
http://www.ucm.es/e-sede
http://informatica.ucm.es/asistencia
http://informatica.ucm.es/investigacion
http://informatica.ucm.es/eventos-y-conferencias
http://www.fdi.ucm.es/migs/
http://www.ucm.es/BUCM/revcul/sci-fdi/
http://mentorias.fdi.ucm.es
http://www.ucm.es/csim
http://www.fdi.ucm.es/cfi/
http://www.ucm.es/e-sede
http://www.ucm.es/fundacion
http://www.ucm.es/ucm-en-linea
http://informatica.ucm.es/laboratorios-fdi
http://Informatica.ucm.es/buzon-de-sugerencias-y-quejas
```

Análisis mediante expresiones regulares

- El método findall de las expresiones regulares proporciona una lista de todas las cadenas que coinciden con la expresión regular, devolviendo sólo el texto del enlace situado dentro de las comillas dobles.

Análisis mediante expresiones regulares

- Las expresiones regulares funcionan bien cuando el HTML está bien formado y es predecible. Sin embargo si el HTML no está bien construido, se pueden perder parte de los enlaces correctos, o terminar obteniendo datos erróneos.

Análisis mediante BeautifulSoup

- BeautifulSoup es un módulo para extraer información de un página web.
- Se puede descargar e “instalar” desde http://www.crummy.com/software/o simplemente colocar el archivo BeautifulSoup.py en la misma carpeta que nuestra aplicación.

Análisis mediante BeautifulSoup

- En este ejemplo se va analizar una página web y se van a extraer todos sus enlaces. Para ello el programa solicita una dirección web, luego abre la página web usando urllib, se leen los datos y se los pasa al analizador BeautifulSoup, que recupera todas las etiquetas de anclas(a) e imprime en pantalla el atributo href de cada una de ellas.

Análisis mediante BeautifulSoup

```
import urllib
from BeautifulSoup import *
url = raw_input('Introduzca - ')
html = urllib.urlopen(url).read()
sopa = BeautifulSoup(html)
etiquetas = sopa('a')
for etiqueta in etiquetas:
    print etiqueta.get('href', None)
```

```
Introduzca - http://www.fdi.ucm.es
https://www.ucm.es/login_sso/
http://www.ucm.es/
/
http://www.campusmoncloa.es/en/
/facultad
/presentacion
/gobierno
/departamentos_1
http://informatica.ucm.es/facultad
/estudiar
/grado
/master
/doctorado
/titulos-propios
http://informatica.ucm.es/estudiar
/estudiantes
/actividades-formativas
/ofertas-de-empleo
/secretaria
/programacion-docente
/programas-de-movilidad
```

Análisis mediante BeautifulSoup

- Observar que el método BeautifulSoup() también admite como entrada un archivo html que esté descargado en el disco duro local.

```
from BeautifulSoup import *
html = open("fdi.htm")
sopa = BeautifulSoup(html)
etiquetas = sopa('a')
for etiqueta in etiquetas:
    print etiqueta.get('href', None)
```

Análisis mediante BeautifulSoup

- El método BeautifulSoup() genera un objeto de tipo BeautifulSoup que tiene un conjunto de métodos que se pueden utilizar para localizar partes específicas de un documento html.

Análisis mediante BeautifulSoup

- En el siguiente ejemplo se van a extraer varías partes de una etiqueta.

```
import urllib
from BeautifulSoup import *
url = raw_input('Introduzca - ')
html = urllib.urlopen(url).read()
sopa = BeautifulSoup(html)
# Recupera todas las etiquetas de anclaje
etiquetas = sopa('a')
# Busca las partes de una etiqueta
for etiqueta in etiquetas:
        print 'ETIQUETA:',etiqueta
        print 'URL:',etiqueta.get('href', None)
        print 'Contenido:', str(etiqueta.contents[0])
        print 'Atributos:',etiqueta.attrs
```

```
Introduzca - http://www.fdi.ucm.es
ETIQUETA: <a href="https://www.ucm.es/login_sso/" title="Navegar identificado">Navegar identificado</a>
URL: https://www.ucm.es/login_sso/
Contenido: Navegar identificado
Atributos: [(u'href', u'https://www.ucm.es/login_sso/'), (u'title', u'Navegar identificado')]
ETIQUETA: <a href="http://www.ucm.es/" title="Universidad Complutense de Madrid - Portada"><img src="/node
Madrid" /></a>
URL: http://www.ucm.es/
Contenido: <img src="/nodes/web/00media/img/ucm-ws.png" alt="Universidad Complutense de Madrid" />
Atributos: [(u'href', u'http://www.ucm.es/'), (u'title', u'Universidad Complutense de Madrid - Portada')]
```

Análisis mediante BeautifulSoup

- En general BeautifulSoup se puede usar para diferentes acciones:
  - Imprimir un documento
  - Parsear un documento html
  - Analizar un documento html

Análisis mediante BeautifulSoup

- Imprimir un documento html:
  - Existen varios métodos para imprimir un documento:
    - La función str() muestra el documento como una cadena, pero no elimina nodos que solo tengan un espacio en blanco ni añade espacios en blanco entre los nodos.
    - La función unicode() muestra el documento como una cadena unicode, pero no elimina nodos que solo tengan un espacio en blanco ni añade espacios en blanco entre los nodos.
    - La función prettify() añade nuevas líneas y espacios para mostrar la estructura del documento html, y elimina nodos que solo contengan espacios en blanco.
    - La función renderContents muestra el documento como una cadena en la codificación dada, y si no se indica se muestra como una cadena unicode.

Análisis mediante BeautifulSoup

- Imprimir un documento html:

```
from BeautifulSoup import BeautifulSoup
doc = "<html><h1>Cabecera</h1><p>Text"
soup = BeautifulSoup(doc)
print str(soup)
print soup.renderContents()
print soup.__str__()
print unicode(soup)
print soup.prettify()
print soup.prettify()
```

```
>>>
<html><h1>Cabecera</h1><p>Text</p></html>
<html><h1>Cabecera</h1><p>Text</p></html>
<html><h1>Cabecera</h1><p>Text</p></html>
<html><h1>Cabecera</h1><p>Text</p></html>
<html>
 <h1>
  Cabecera
 </h1>
 <p>
  Text
 </p>
</html>
<html>
 <h1>
  Cabecera
 </h1>
 <p>
  Text
 </p>
</html>
>>>
```

Análisis mediante BeautifulSoup

- Imprimir un documento html:
  - Las funciones str y renderContents no funcionan igual cuando se usa sobre una etiqueta dentro del documento. En el caso de str se imprime la etiqueta y sus contenidos y en el caso de renderContents solo se imprimen los contenidos.

```
from BeautifulSoup import BeautifulSoup
doc = "<html><h1>Cabecera</h1><p>Text"
soup = BeautifulSoup(doc)
cabecera = soup.h1
print str(cabecera)
print cabecera.renderContents()
```

```
>>>
<h1>Cabecera</h1>
Cabecera
>>>
```

Análisis mediante BeautifulSoup

- Imprimir un documento html:
  - Cuando se llama a __str__, prettify o renderContents se puede especifica la codificación de salida de la cadena. Por defecto es UTF-8

```
from BeautifulSoup import BeautifulSoup
doc = "Sacr\xe9 bleu!"
soup = BeautifulSoup(doc)
print str(soup)
print soup.__str__("ISO-8859-1")
print soup.__str__("UTF-16")
print soup.__str__("EUC-JP")
```

```
>>>
SacrÃ© bleu!
Sacré bleu!
ÿþS a c r é   b l e u !
Sacr7㎜ bleu!
>>>
```

Análisis mediante BeautifulSoup

- Imprimir un documento html:
  - Si el documento original contenía una especificación de codificación entonces se reescribe cuando se convierte a cadena a UTF-8

```
from BeautifulSoup import BeautifulSoup
doc = """<html>
<meta http-equiv="Content-type" content="text/html; charset=ISO-Latin-1" >
Sacr\xe9 bleu!
</html>"""
print BeautifulSoup(doc).prettify()
```

```
>>>
<html>
 <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
 SacrÃ© bleu!
</html>
>>>
```

Análisis mediante BeautifulSoup

- Parsear un documento html:
  - BeautifulSoup toma un documento html y lo parsea a una estructura de datos en forma de árbol. Si el documento está bien formado el árbol se parece al documento original pero si no lo está entonces usa heurísticas para conseguir una estructura razonable.

Análisis mediante BeautifulSoup

- Parsear un documento html:
  - En este sentido se usa el siguiente conocimiento:
    - Hay etiquetas que pueden estar anidadas(<BLOCKQUOTE>) y otras no(<P>).
    - Las tablas y listas de etiquetas tienen un orden de anidamiento natural. Por ejemplo <TD> está en el interior de <TR>.
    - Los contenidos de una etiqueta <SCRIPT> no deben ser parseados.
    - Una etiqueta <META> puede especificar una codificación del documento.

Análisis mediante BeautifulSoup

- Parsear un documento html:

```
from BeautifulSoup import BeautifulSoup
html = "<html><p>Parra 1<p>Parra 2<blockquote>Quote 1<blockquote>Quote 2"
soup = BeautifulSoup(html)
print soup.prettify()
```

```
>>>
<html>
 <p>
  Parra 1
 </p>
 <p>
  Parra 2
  <blockquote>
   Quote 1
   <blockquote>
    Quote 2
   </blockquote>
  </blockquote>
 </p>
</html>
>>>
```

Análisis mediante BeautifulSoup

- Parsear un documento html:

```
from BeautifulSoup import BeautifulSoup
html = """
<html>
<form>
 <table>
 <td><input name="input1">Row 1 cell 1
 <tr><td>Row 2 cell 1
 </form>
 <td>Row 2 cell 2<br>This</br> sure is a long cell
</body>
</html>"""
print BeautifulSoup(html).prettify()
```

```
>>>
<html>
 <form>
  <table>
   <td>
    <input name="input1" />
    Row 1 cell 1
   </td>
   <tr>
    <td>
     Row 2 cell 1
    </td>
   </tr>
  </table>
 </form>
 <td>
  Row 2 cell 2
  <br />
  This
  sure is a long cell
 </td>
</html>
>>>
```

Análisis mediante BeautifulSoup

- El objeto Beautifulsoup representa un árbol de procesamiento que contiene dos tipos de objetos:
  - Objetos de tipo tag que se corresponden con etiquetas o elementos del documento HTML
  - Objetos de tipo NavigableString que se corresponden con cadenas. Existen subclases de NavigableString que corresponden a construcciones especiales XML tales como CData, Comment, Declaration, and ProcessingInstruction.

Análisis mediante BeautifulSoup

- Los elementos representados por los objetos de tipo tag pueden tener asociados atributos y se puede acceder a ellos como si fuera un diccionario. Sin embargo los elementos representados por los objetos NavigableString no tienen asociados atributos.

Análisis mediante BeautifulSoup

- Considerar el siguiente ejemplo para las explicaciones siguientes:

```
<html>
<head>
 <title>
  Título de la página
   </title>
  </head>
  <body>
   <p id="primerparrafo" align="center">
    Esto es un parrafo
    <b>
     one
    </b>
    .
   </p>
   <p id="segundoparrafo" align="blah">
    Esto es un parrafo
    <b>
     two
    </b>
    .
   </p>
  </body>
</html>
```

Análisis mediante BeautifulSoup

- En el siguiente ejemplo se recuperan las etiquetas “p” y se accede a sus atributos como si fuera un diccionario.

```
from BeautifulSoup import BeautifulSoup
html=open("prueba.html")
soup=BeautifulSoup(html.read())
Primero,Segundo=soup.findAll('p')
print Primero['id']
print Segundo['id']
```

```
>>>
primerparrafo
segundoparrafo
>>>
```

Análisis mediante BeautifulSoup

- Los valores Tag pueden ser pasados a la función str() para mostrar las etiquetas que representan. Además los valores Tag tienen un atributo llamado attrs que muestra todos los atributos HTML que tiene el elemento en forma de un diccionario.

Análisis mediante BeautifulSoup

- Los objetos Tag y NavigableString disponen de un conjunto de atributos y métodos:
  - parent: Permite acceder al objeto que representa la etiqueta padre del objeto que representa a una etiqueta. Permite navegar por el árbol de procesamiento.

```
>>> soup.head.parent.name
u'html'
```

Análisis mediante BeautifulSoup

- contents: Representa una lista ordenada de los objetos Tags y NavigableString contenidos dentro de un elemento. Solo el objeto que representa al árbol y los objetos tags poseen este atributo. Los objetos NavigableString no tienen el atributo contents, pues sólo tienen cadenas.

```
>>> etiquetasp=soup.p
>>> etiquetasp.contents
[u'\n    Esto es un parrafo\n    ', <b>
     one
    </b>, u'\n    .\n    ']
>>> etiquetasp.contents[1].contents
[u'\n     one\n    ']
>>> etiquetasp.contents[0].contents

Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    etiquetasp.contents[0].contents
  File "C:/Users/asarasa/Desktop\BeautifulSoup.py", line 448, in __getattr__
    raise AttributeError, "'%s' object has no attribute '%s'" % (self.__class__.
__name__, attr)
AttributeError: 'NavigableString' object has no attribute 'contents'
```

Análisis mediante BeautifulSoup

- string: Si un tag solo tiene un nodo hijo y se trata de una cadena, entonces se puede acceder al mismo mediante tag.string o mediante tag.contents[0]. Cuando existen hijos, y se trata de acceder al atributo string, devuelve como resultado el valor None. Los objetos NavigableString no tienen este atributo.

```
>>> soup.b.string
u'\n     one\n    '
>>> soup.b.contents[0]
u'\n     one\n    '
>>> soup.p.string==None
True
>>> soup.head.string==None
True
```

Análisis mediante BeautifulSoup

- nextSibling y previousSibling: Permite recuperar el objeto anterior o posterior que se encuentra al mismo nivel del objeto considerado. En el ejemplo anterior la etiqueta <Body> está al mismo nivel que la etiqueta <Head> pero esta última aparece antes.

```
>>> soup.head.nextSibling.name
u'body'
>>> soup.head.previousSibling==None
True
```

Análisis mediante BeautifulSoup

- next y previous: Permiten navegar en el árbol de procesamiento sobre los objetos en el orden en que fueron procesados en vez del orden dado por el árbol. En el ejemplo el objeto next al objeto que representa a <HEAD> es el objeto que representa a <TITLE> y no el objeto que representa a <BODY>.

```
>>> soup.head.next
<title>TÃ-tulo de la pagina</title>
>>> soup.head.nextSibling.name
u'body'
>>> soup.head.previous.name
u'html'
```

Análisis mediante BeautifulSoup

- Observaciones:
  - Se puede iterar sobre el atributo contents de un objeto tag y tratarlo como una lista, y de forma similar se puede obtener el número de hijos mediante len(tag) o len(tag.contents).

```
>>> for i in soup.body:
        print i


<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>
<p id="segundoparrafo" align="blah">Esto es un parrafo<b>dos</b>.</p>
>>> len(soup.body)
2
>>> len(soup.body.contents)
2
```

Análisis mediante BeautifulSoup

- Observaciones:
  - Se puede usar los nombres de un objeto tag como si fueran atributos del árbol de procesamiento o de un objeto tag. Siempre que se usa de esta forma, devuelve el primer nodo hijo cuyo nombre sea el considerado o bien retorna None si no existen hijos con ese nombre. En el ejemplo anterior para acceder a la etiqueta <Title> se puede ir a partir de la etiqueta <Head> o bien directamente.

```
>>> soup.head.title
<title>TÃ-tulo de la pagina</title>
>>> soup.title
<title>TÃ-tulo de la pagina</title>
```

Análisis mediante BeautifulSoup

- Ahora se van a considerar los métodos que permiten buscar en el árbol de procesamiento: findAll y find.
- Solo se encuentran disponibles para el objeto que representa el árbol de procesamiento y en los objetos de tipo tag pero no en los objetos de tipo NavigableString

Análisis mediante BeautifulSoup

- El método findAll busca todos los objetos tag y NavigableString que coinciden con un criterio dado desde un punto dado. Sus principales argumentos son:

  findAll(name, attrs, recursive, text, limit)

  - Un nombre que restringe el conjunto de búsqueda por el nombre de las etiquetas.
  - Pares atributo-valor que restringen el conjunto de búsqueda por los valores que toman los atributos de las etiquetas.

Análisis mediante BeautifulSoup

- El argumento “text” permite buscar objetos NavigableString , y puede tomar como valores una cadena, una expresión regular, una lista o diccionario, True o None o bien una expresión booleana. Cuando se usa este argumento, las restricciones sobre nombre o atributo no se tienen en cuenta.
- El argumento “recursive” que puede tomar los valores True o False que indica si busca por debajo del árbol o bien por los hijos inmediatos del árbol. Por defecto es True.
- El argumento limit permite parar la búsqueda cuando se han conseguido un número de coincidencias dado. Por defecto no tienen límite.

Análisis mediante BeautifulSoup

- Algunos ejemplos de restricción por nombre:
  - Argumento con el nombre de una etiqueta.

```
>>> soup.findAll('b')
[<b>uno</b>, <b>dos</b>]
```

  - Argumento con una expresión regular

```
>>> import re
>>> etiquetasconb=soup.findAll(re.compile('^b'))
>>> for tag in etiquetasconb:
        print tag.name


body
b
b
```

Análisis mediante BeautifulSoup

- Algunos ejemplos de restricción por nombre:
  - Argumento con una lista o diccionario.

```
>>> soup.findAll(['title','p'])
[<title>TÃ-tulo de la pagina</title>, <p id="primerparrafo" align="center">Esto
es un parrafo<b>uno</b>.</p>, <p id="segundoparrafo" align="blah">Esto es un par
rafo<b>dos</b>.</p>]
>>> soup.findAll({'title':True,'p':True})
[<title>TÃ-tulo de la pagina</title>, <p id="primerparrafo" align="center">Esto
es un parrafo<b>uno</b>.</p>, <p id="segundoparrafo" align="blah">Esto es un par
rafo<b>dos</b>.</p>]
```

Análisis mediante BeautifulSoup

- Algunos ejemplos de restricción por nombre:
  - Argumento con el valor True que produce una coincidencia con cada objeto tag.

```
>>> etiquetas=soup.findAll(True)
>>> for tag in etiquetas:
        print tag.name


html
head
title
body
p
b
p
b
```

Análisis mediante BeautifulSoup

- Algunos ejemplos de restricción por nombre:
  - Argumento con una expresión que evalúa a cierto o falso.

```
>>> soup.findAll(lambda tag: len(tag.attrs)==2)
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>, <p id="
segundoparrafo" align="blah">Esto es un parrafo<b>dos</b>.</p>]
>>> soup.findAll(lambda tag: len(tag.name)==1 and not tag.attrs)
[<b>uno</b>, <b>dos</b>]
```

Análisis mediante BeautifulSoup

- Algunos ejemplos de restricción por atributo:
  - Argumento que impone una condición al valor que toma un atributo.

```
>>> soup.findAll(align="center")
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>]
```

  - Argumento que impone una condición en forma de expresión regular al valor que toma un atributo.

```
>>> soup.findAll(id=re.compile("parra$"))
```

Análisis mediante BeautifulSoup

- Algunos ejemplos de restricción por atributo:
  - Argumento que impone una condición en forma de lista al valor que toma un atributo.

```
>>> soup.findAll(align=["center","blah"])
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>, <p id="
segundoparrafo" align="blah">Esto es un parrafo<b>dos</b>.</p>]
```

  - Argumento que impone una condición en forma de una expresión booleano al valor que toma un atributo.

```
>>> soup.findAll(align=lambda(value): value and len(value) < 5)
[<p id="segundoparrafo" align="blah">Esto es un parrafo<b>dos</b>.</p>]
```

Análisis mediante BeautifulSoup

- Algunos ejemplos de restricción por atributo:
  - Argumento que iguala a True o None el valor de un atributo. True encaja con etiquetas que toman cualquier valor para el atributo y None encaja con etiquetas que no tienen valor para ese atributo.

```
>>> soup.findAll(align=True)
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>, <p id="
segundoparrafo" align="blah">Esto es un parrafo<b>dos</b>.</p>]
>>> etiquetas=soup.findAll(align=None)
>>> for eti in etiquetas:
        print eti.name


html
head
title
body
b
b
```

Análisis mediante BeautifulSoup

- Algunos ejemplos del argumento text:

```
>>> soup.findAll(text="uno")
[u'uno']
>>> soup.findAll(text=['uno','dos'])
[u'uno', u'dos']
>>> soup.findAll(text=True)
[u'T\xedtulo de la pagina', u'Esto es un parrafo', u'uno', u'.', u'Esto es un pa
rrafo', u'dos', u'.']
>>> soup.findAll(text=lambda(x): len(x) < 12)
[u'uno', u'.', u'dos', u'.']
>>> soup.findAll(text=re.compile("parrafo"))
[u'Esto es un parrafo', u'Esto es un parrafo']
```

Análisis mediante BeautifulSoup

- Algunos ejemplos del argumento recursive:

```
>>> [tag.name for tag in soup.html.findAll()]
[u'head', u'title', u'body', u'p', u'b', u'p', u'b']
>>> [tag.name for tag in soup.html.findAll(recursive=False)]
[u'head', u'body']
```

Análisis mediante BeautifulSoup

- Algunos ejemplos del argumento limit:

```
>>> soup.findAll('p',limit=1)
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>]
>>> soup.findAll('p',limit=100)
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>, <p id="
segundoparrafo" align="blah">Esto es un parrafo<b>dos</b>.</p>]
```

Análisis mediante BeautifulSoup

- El método find tiene una estructura similar a findAll:

  find(name, attrs, recursive, text)

- Es similar a findAll con la única diferencia de que en vez de recuperar todas las coincidencias, recupera sólo la primera. Tiene el mismo comportamiento que si el parámetro limit de findAll tomara el valor 1.

```
>>> soup.findAll('p',limit=1)
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>]
>>> soup.find('p')
<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>
```

Análisis mediante BeautifulSoup

- Observaciones:
  - Puede ocurrir que algún atributo de una etiqueta coincida con las palabras reservadas en BeautifulSoup. En estos casos no se puede hacer una referencia directa al atributo y es necesario usar un atributo que tienen las etiquetas denominado attrs que actúa como un diccionario que hace referencia a los atributos de una etiqueta.

```
>>> soup.findAll(attrs={"id": "primerparrafo"})
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>]
```

Análisis mediante BeautifulSoup

- Observaciones:
  - Se pueden combinar búsqueda por nombre y por atributo:

```
>>> soup.findAll('p',{'align':'center'})
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>]
```

  - Cuando el objeto que representa al árbol de procesamiento o un objeto tag se usa como una función y se le pasan los mismos argumentos que a findAll, actúa como findAll.

```
>>> soup(text=lambda(x): len(x) <12)
[u'uno', u'.', u'dos', u'.']
>>> soup.body('p',limit=1)
[<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>]
```

Análisis mediante BeautifulSoup

- Observaciones:
  - Los métodos findAll y find empiezan en un cierto punto del árbol y lo recorren hacia abajo, iterando recursivamente sobre los elementos del atributo contents de los objetos tags. Es por ello que no se pueden usar sobre objetos de tipo NavigableString pues no tienen atributo contents.

Análisis mediante BeautifulSoup

- Existen otros métodos de navegación por el árbol de procesamiento:
  - findNextSiblings(name, attrs, text, limit) y findNextSibling(name, attrs, text): Devuelve el nodo/s hermano/s más cercanos a la etiqueta dada que coincida con los criterios de búsqueda y que aparece después de la etiqueta considerada.

```
>>> parrafo=soup.find(text='Esto es un parrafo')
>>> parrafo.findNextSiblings('b')
[<b>uno</b>]
>>> parrafo.findNextSibling(text=lambda(text): len(text)==1)
u'.'
```

Análisis mediante BeautifulSoup

- Existen otros métodos de navegación por el árbol de procesamiento:
  - findPreviousSiblings(name, attrs, text, limit) y findPreviousSibling(name, attrs, text): Devuelve el nodo/s hermano/s más cercanos a la etiqueta dada que coincida con los criterios de búsqueda y que aparece antes de la etiqueta considerada.

```
>>> parrafo=soup.find(text='.')
>>> parrafo.findPreviousSiblings('b')
[<b>uno</b>]
>>> parrafo.findPreviousSibling(text=True)
u'Esto es un parrafo'
```

Análisis mediante BeautifulSoup

- Existen otros métodos de navegación por el árbol de procesamiento:
  - findAllNext(name, attrs, text, limit) y findNext(name, attrs, text):
    Devuelve todos los elementos(el elemento) que coincida con los criterios de búsqueda y que aparece después de la etiqueta considerada.

```
>>> etiqueta=soup.find('p')
>>> etiqueta.findAllNext(text=True)
[u'Esto es un parrafo', u'uno', u'.', u'Esto es un parrafo', u'dos', u'.']
>>> etiqueta.findNext('p')
<p id="segundoparrafo" align="blah">Esto es un parrafo<b>dos</b>.</p>
>>> etiqueta.findNext('b')
<b>uno</b>
```

Análisis mediante BeautifulSoup

- Existen otros métodos de navegación por el árbol de procesamiento:
  - findAllPrevious(name, attrs, text, limit) y findPrevious(name, attrs, text):
    Devuelve todos los elementos(el elemento) que coincida con los criterios de búsqueda y que aparece antes de la etiqueta considerada.

```
>>> etiqueta=soup('p')[-1]
>>> etiqueta.findAllPrevious(text=True)
[u'.', u'uno', u'Esto es un parrafo', u'T\xedtulo de la pagina']
>>> etiqueta.findPrevious('p')
<p id="primerparrafo" align="center">Esto es un parrafo<b>uno</b>.</p>
>>> etiqueta.findPrevious('b')
<b>uno</b>
```

Análisis mediante BeautifulSoup

- Existen otros métodos de navegación por el árbol de procesamiento:
  - findParents(name, attrs, limit) y findParent(name, attrs):
    Devuelve los padres de la etiqueta considerada que coinciden con los criterios de búsqueda.

```
>>> etiqueta=soup.find('b')
>>> [eti.name for eti in etiqueta.findParents()]
[u'p', u'body', u'html', u'[document]']
>>> etiqueta.findParent('body').name
u'body'
```

Análisis mediante BeautifulSoup

- Otra utilidad que ofrece BeautifulSoup es la modificación del árbol de procesamiento:
  - Cambio de valores de atributos.
  - Eliminación de elementos.
  - Reemplazamiento de un elemento por otro.
  - Añadir una rama con un nuevo elemento.

Análisis mediante BeautifulSoup

- Cambio de valores de atributos:
  - Se puede usar la asignación de diccionarios para modificar los valores de los atributos de un objeto tag.

```
from BeautifulSoup import BeautifulSoup
soup = BeautifulSoup("<b id='2'>Argh!</b>"
print soup
b = soup.b
b['id'] = 10
print soup
b['id'] = 'diez'
print soup
b['id'] = 'un millon'
print soup
```

```
>>>
<b id="2">Argh!</b>
<b id="10">Argh!</b>
<b id="diez">Argh!</b>
<b id="un millon">Argh!</b>
>>>
```

Análisis mediante BeautifulSoup

- Cambio de valores de atributos:
  - Se pueden eliminar atributos y añadir otros nuevos.

```
from BeautifulSoup import BeautifulSoup
soup = BeautifulSoup("<b id='2'>Argh!</b>")
b=soup.b
del(b['id'])
print soup
b['class'] = "extra negro"
print soup
```

```
>>>
<b>Argh!</b>
<b class="extra negro">Argh!</b>
>>>
```

Análisis mediante BeautifulSoup

- Eliminación de elementos
  - Cuando se sitúa sobre un elemento concreto, se puede eliminar del árbol de procesamiento con el método extract.

```
#Se eliminan todos los comentarios del documento html
from BeautifulSoup import BeautifulSoup, Comment
soup = BeautifulSoup("""1<!--Comentario 1-->
                        <a>2<!--Comentario 2--><b>3""")
comentarios = soup.findAll(text=lambda text:isinstance(text, Comment))
[comentario.extract() for comentario in comentarios]
print "Eliminación de todos los comentarios del documento html"
print soup
```

```
>>>
1
<a>2<b>3</b></a>
>>>
```

Análisis mediante BeautifulSoup

- Eliminación de elementos

```
#Eliminación de un subárbol entero que da lugar a dos árboles disjuntos
from BeautifulSoup import BeautifulSoup
soup = BeautifulSoup("<a1></a1><a><b>Contenido de prueba<c><d></a><a2></a2>")
print soup.a1.nextSibling
print soup.a2.previousSibling
subarbol = soup.a
subarbol.extract()
print soup
print soup.a1.nextSibling
print soup.a2.previousSibling
print subarbol.previousSibling == None
print subarbol.parent == None
```

```
>>>
<a><b>Contenido de prueba<c><d></d></c></b></a>
<a><b>Contenido de prueba<c><d></d></c></b></a>
<a1></a1><a2></a2>
<a2></a2>
<a1></a1>
True
True
>>>
```

Análisis mediante BeautifulSoup

- Reemplazar un elemento por otro
  - El método replaceWith extrae un elemento y lo reemplaza por otro que puede ser tanto un objeto de tipo tag o de tipo NavigableString. Si se pasa como argumento una cadena entonces lo considera un objeto de tipo NavigableString

```
#Ejemplo 1 de reemplazamiento del contenido de una etiqueta
from BeautifulSoup import BeautifulSoup
soup = BeautifulSoup("<b>Argh!</b>")
soup.find(text="Argh!").replaceWith("Buff!")
print soup
nuevoText = soup.find(text="Buff!")
print nuevoText.previous
s=nuevoText.previous.next
print str(s)
print nuevoText.parent
print soup.b.contents
```

```
>>>
<b>Buff!</b>
<b>Buff!</b>
Buff!
<b>Buff!</b>
[u'Buff!']
>>>
```

Análisis mediante BeautifulSoup

- Reemplazar un elemento por otro

```
#Ejemplo 2 de reemplazamiento de un elemento por otro nuevo
from BeautifulSoup import BeautifulSoup, Tag
soup = BeautifulSoup("<b>Argh!<a>Hola</a></b><i>Adios</i>")
tag = Tag(soup, 'newTag', [('id', '1')])
tag.insert(0, "Noche")
soup.a.replaceWith(tag)
print soup
```

```
>>>
<b>Argh!<newTag id="1">Noche</newTag></b><i>Adios</i>
>>>
```

Análisis mediante BeautifulSoup

- Reemplazar un elemento por otro

```
#Ejemplo 3 de reemplazamiento dónde se elimina un elemento y se sustituye por
#otro elemento del árbol de procesamiento
from BeautifulSoup import BeautifulSoup
text = "<html>Esto es <b>un</b> ejemplo de <b>documento </b> html</html>"
soup = BeautifulSoup(text)
uno, dos = soup.findAll('b')
dos.replaceWith(uno)
print soup
```

```
>>>
<html>Esto es  ejemplo de <b>un</b> html</html>
>>>
```

Análisis mediante BeautifulSoup

- Añadir una rama con un nuevo elemento.
  - El método insert toma un índice sobre el atributo contents del objeto tag e inserta un nuevo elemento en dicha posición.

```
#Ejemplo de creación de un árbol de procesamiento desde cero
from BeautifulSoup import BeautifulSoup, Tag, NavigableString
soup = BeautifulSoup()
tag1 = Tag(soup, "etiqueta1")
tag2 = Tag(soup, "etiqueta2")
tag3 = Tag(soup, "etiqueta3")
soup.insert(0, tag1)
tag1.insert(0, tag2)
tag1.insert(1, tag3)
print soup
text = NavigableString("Hola")
tag3.insert(0, text)
print soup
```

```
>>>
<etiqueta1><etiqueta2></etiqueta2><etiqueta3></etiqueta3></etiqueta1>
<etiqueta1><etiqueta2></etiqueta2><etiqueta3>Hola</etiqueta3></etiqueta1>
>>>
```

Análisis mediante BeautifulSoup

- Añadir una rama con un nuevo elemento.
  - Un elemento sólo puede estar en un lugar del árbol de procesamiento. Si con el método insert se intenta insertar nuevamente un elemento, entonces es eliminado del lugar en el que estaba anteriormente.

```
#Ejemplo de intentar insertar un elemento dos veces
from BeautifulSoup import BeautifulSoup, Tag, NavigableString
soup = BeautifulSoup()
tag1 = Tag(soup, "etiqueta1")
tag2 = Tag(soup, "etiqueta2")
tag3 = Tag(soup, "etiqueta3")
soup.insert(0, tag1)
tag1.insert(0, tag2)
tag1.insert(1, tag3)
print soup
text = NavigableString("Hola")
tag3.insert(0, text)
print soup
tag2.insert(0,text)
print soup
```

```
>>>
<etiqueta1><etiqueta2></etiqueta2><etiqueta3></etiqueta3></etiqueta1>
<etiqueta1><etiqueta2></etiqueta2><etiqueta3>Hola</etiqueta3></etiqueta1>
<etiqueta1><etiqueta2>Hola</etiqueta2><etiqueta3></etiqueta3></etiqueta1>
>>>
```

Análisis mediante BeautifulSoup

- Se va a considerar el siguiente problema. Se quiere automatizar la bajada de imágenes de tren procedentes de un blog especializado.

Análisis mediante BeautifulSoup

- En primer lugar recuperamos la página principal del blog mediante urllib

```
import urllib
from BeautifulSoup import *
html = urllib.urlopen('http://trenesytiempos.blogspot.com.es/').read()
```

Análisis mediante BeautifulSoup

- Vemos la estructura que tiene la página para ver dónde se encuentran las imágenes, y se observa que aparecen en etiquetas de anclaje que tienen entre sus atributos imageanchor="1".

```
<a href="http://3.bp.blogspot.com/-0xCsiUgrBpw/VeGRg4rYNLI/AAAAAAAAAGvQ/U15pYh4UQGY/s1600/1300%2Bnorte%2Birun%2B1864.jpg" imageanchor="1"
```

Análisis mediante BeautifulSoup

- Se crea un árbol de procesamiento con BeautifulSoup y se recuperan todas las etiquetas con las características anteriores:

```
import urllib
from BeautifulSoup import *
html = urllib.urlopen('http://trenesytiempos.blogspot.com.es/').read()
sopa = BeautifulSoup(html)
etiquetas=sopa('a',{"imageanchor":"1"})
```

Análisis mediante BeautifulSoup

- Recuperadas las direcciones se quieren almacenar las imágenes en el disco local, entonces para ello se usa un bucle for que recorre cada dirección, la abre y la almacena en un archivo distinto.

```
j=0
for i in etiquetas:
        archivo=open("foto"+str(j)+".jpg","wb")
        imagen=urllib.urlopen(i.get('href',None))
        while True:
                info = imagen.read(100000)
                if len(info) < 1 : break
                archivo.write(info)
        archivo.close()
        j=j+1
```

Análisis mediante BeautifulSoup

- El código completo sería:

```
import urllib
from BeautifulSoup import *
html = urllib.urlopen('http://trenesytiempos.blogspot.com.es/').read()
sopa = BeautifulSoup(html)
etiquetas=sopa('a',{"imageanchor":"1"})
j=0
for i in etiquetas:
        archivo=open("foto"+str(j)+".jpg","wb")
        imagen=urllib.urlopen(i.get('href',None))
        while True:
                info = imagen.read(100000)
                if len(info) < 1 : break
                archivo.write(info)
        archivo.close()
        j=j+1
```

Otras herramientas: Módulo webbrowser

- La función open del módulo webbrowser de Python permite lanzar un navegador sobre una URL específica. Por ejemplo el siguiente trozo de código abriría un navegador con la página de la Facultad de Informática

```
import webbrowser
webbrowser.open("http://www.fdi.ucm.es")
```

Otras herramientas: Módulo webbrowser

- Se va a considerar un programa que recupera del teclado una dirección y que permite buscar dicha dirección en Google Maps.Para ello en primer lugar observar que la url que se usa cuando se busca una dirección en Google Maps tiene la siguiente estructura:

  https://www.google.com/maps/place/dirección-buscada

Otras herramientas: Módulo webbrowser

- El programa debe recuperar de la línea de comandos la dirección . Para ello se usará el módulo sys que permite almacena en la variable sys.argv una lista con el nombre del programa y los argumentos pasados al programa en caso de tenerlos. Para saber si la lista incluye más elementos además del nombre del programa, se puede utilizar len(sys.argv) y comprobar que se evalúa a un número mayor que 1.

Otras herramientas: Módulo webbrowser

- Se tiene el siguiente código:

```
import sys
if len(sys.argv) > 1:
    direccion =' '.join(sys.argv[1:])
```

Otras herramientas: Módulo webbrowser

- Observar que los argumentos en la línea de comandos en general se separan mediante espacios, sin embargo en este caso sería necesario interpretar todos los argumentos como una única cadena. Dado que sys.argv es una lista de cadenas, se puede utilizar el método join() que retorna una única cadena. Además como no interesa el nombre del programa, se puede eliminar el mismo si se selecciona la última parte de la lista con sys.argv[1:]

Otras herramientas: Módulo webbrowser

- Si en la línea de comandos no hay argumentos, se asumirá que la dirección se ha copiado. Para obtener el contenido de una copia se utiliza el método paste del módulo pyperclip

```
import sys, pyperclip
if len(sys.argv) > 1:
    direccion =' '.join(sys.argv[1:])
else:
    direccion= pyperclip.paste()
```

Otras herramientas: Módulo webbrowser

- Por último para lanzar el navegador se utiliza el método open del módulo browser.

```
import sys, pyperclip,webbrowser
if len(sys.argv) > 1:
    direccion =' '.join(sys.argv[1:])
else:
    direccion= pyperclip.paste()

webbrowser.open('http://www.google.com/maps/place/'+ direccion)
```