EBook Gratis

APRENDIZAJE beautifulsoup

Free unaffiliated eBook created from
Stack Overflow contributors.

#beautifulso

up

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: beautifulsoup

It is an unofficial and free beautifulsoup ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official beautifulsoup.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con beautifulsoup

Observaciones

En esta sección, discutimos qué es Beautiful Soup, para qué se usa y un breve resumen de cómo usarlo.

Beautiful Soup es una biblioteca de Python que utiliza su analizador html / xml preinstalado y convierte la página web / html / xml en un árbol que consta de etiquetas, elementos, atributos y valores. Para ser más exactos, el árbol consta de cuatro tipos de objetos, Tag, NavigableString, BeautifulSoup y Comment. Este árbol se puede "consultar" utilizando los métodos / propiedades del objeto BeautifulSoup que se crea a partir de la biblioteca del analizador.

Su necesidad: a menudo, puede tener una de las siguientes necesidades:

1. Es posible que desee analizar una página web para determinar, cuántas de las etiquetas se encuentran, cuántos elementos de cada etiqueta se encuentran y sus valores. Es posible que desee cambiarlos.

2. Es posible que desee determinar los nombres y valores de los elementos, de modo que pueda utilizarlos junto con otras bibliotecas para la automatización de páginas web, como Selenium .

3. Es posible que desee transferir / extraer los datos que se muestran en una página web a otros formatos, como un archivo CSV o una base de datos relacional como SQLite o mysql. En este caso, la biblioteca lo ayuda con el primer paso, para comprender la estructura de la página web, aunque utilizará otras bibliotecas para realizar la transferencia.

4. Es posible que desee saber cuántos elementos tienen un estilo de estilo CSS determinado y cuáles.

Secuencia para uso básico típico en su código Python:

1. Importar la biblioteca de Beautiful Soup

2. Abra una página web o texto html con la biblioteca BeautifulSoup, mencionando qué analizador se usará. El resultado de este paso es un objeto BeautifulSoup. (Nota: Este nombre de analizador mencionado, ya debe estar instalado como parte de sus paquetes de Python. Por ejemplo, `html.parser` , es un paquete incorporado con baterías que se incluye con Python. Puede instalar otros analizadores como `lxml` o `html5lib` .)

3. Haga "consulta" o busque el objeto BeautifulSoup utilizando la sintaxis `'object.method'` y obtenga el resultado en una colección, como un diccionario Python. Para algunos métodos, la salida será un valor simple.

4. Usa el resultado del paso anterior para hacer lo que quieras hacer con él, en el resto de tu código de Python. También puede modificar los valores de elemento o los valores de

atributo en el objeto de árbol. Las modificaciones no afectan la fuente del código html, pero puede llamar a métodos de formato de salida (como `prettify`) para crear una nueva salida del objeto BeautifulSoup.

Métodos de uso común: por lo general, los métodos `.find` y `.find_all` se utilizan para buscar en el árbol, dando los argumentos de entrada.

Los argumentos de entrada son: el nombre de la etiqueta que se está buscando, los nombres de los atributos y otros argumentos relacionados. Estos argumentos podrían presentarse como: una cadena, una expresión regular, una lista o incluso una función.

Los usos comunes del objeto BeautifulSoup incluyen:

1. Buscar por clase CSS
2. Búsqueda por dirección de hipervínculo
3. Búsqueda por elemento Id, tag
4. Búsqueda por nombre de atributo. Valor de atributo.

Si necesita filtrar el árbol con una combinación de los criterios anteriores, también puede escribir una función que se evalúe como verdadera o falsa, y buscar por esa función.

Versiones

Versión	Observaciones	Nombre del paquete	Fecha de lanzamiento
3.x	Versión 3.2.1; Solo Python 2	hermoso	2012-02-16
4.x	Versión 4.5.0; Python 2 y 3	beautifulsoup4	2016-07-20

Examples

Instalación o configuración

pip se puede utilizar para instalar BeautifulSoup. Para instalar la versión 4 de BeautifulSoup, ejecute el comando:

```
pip install beautifulsoup4
```

Tenga en cuenta que el nombre del paquete es `beautifulsoup4` lugar de `beautifulsoup` , este último nombre significa versión antigua, vea old beautifulsoup

Un ejemplo de raspado "Hello World" de BeautifulSoup

```
from bs4 import BeautifulSoup
import requests

main_url = "https://fr.wikipedia.org/wiki/Hello_world"
```

```
req = requests.get(main_url)
soup = BeautifulSoup(req.text, "html.parser")

# Finding the main title tag.
title = soup.find("h1", class_ = "firstHeading")
print title.get_text()

# Finding the mid-titles tags and storing them in a list.
mid_titles = [tag.get_text() for tag in soup.find_all("span", class_ = "mw-headline")]

# Now using css selectors to retrieve the article shortcut links
links_tags = soup.select("li.toclevel-1")
for tag in links_tags:
    print tag.a.get("href")

# Retrieving the side page links by "blocks" and storing them in a dictionary
side_page_blocks = soup.find("div",
                             id = "mw-panel").find_all("div",
                                                       class_ = "portal")
blocks_links = {}
for num, block in enumerate(side_page_blocks):
    blocks_links[num] = [link.get("href") for link in block.find_all("a", href = True)]

print blocks_links[0]
```

Salida:

```
"Hello, World!" program
#Purpose
#History
#Variations
#See_also
#References
#External_links
[u'/wiki/Main_Page', u'/wiki/Portal:Contents', u'/wiki/Portal:Featured_content',
u'/wiki/Portal:Current_events', u'/wiki/Special:Random',
u'https://donate.wikimedia.org/wiki/Special:FundraiserRedirector?utm_source=donate&utm_medium=sidebar&u
u'//shop.wikimedia.org']
```

Al ingresar su analizador preferido al crear una instancia de Beautiful Soup, se evita el habitual `Warning` que declara que `no parser was explicitely specified`.

Se pueden utilizar diferentes métodos para encontrar un elemento dentro del árbol de la página web.

Aunque existen otros métodos, las `CSS classes` y los `CSS selectors` son dos formas útiles de encontrar elementos en el árbol.

Se debe tener en cuenta que podemos buscar etiquetas estableciendo su valor de atributo en Verdadero al buscarlas.

`get_text()` nos permite recuperar texto contenido dentro de una etiqueta. Lo devuelve como una sola cadena de Unicode. `tag.get("attribute")` permite obtener el valor de atributo de una etiqueta.

Lea Empezando con beautifulsoup en línea:
https://riptutorial.com/es/beautifulsoup/topic/1817/empezando-con-beautifulsoup

Capítulo 2: Elementos de localización

Examples

Localiza un texto después de un elemento en BeautifulSoup.

Imagina que tienes el siguiente HTML:

```
<div>
    <label>Name:</label>
    John Smith
</div>
```

Y necesitas ubicar el texto "John Smith" después del elemento de `label` .

En este caso, puede ubicar el elemento de `label` por texto y luego usar la propiedad `.next_sibling` :

```
from bs4 import BeautifulSoup

data = """
<div>
    <label>Name:</label>
    John Smith
</div>
"""

soup = BeautifulSoup(data, "html.parser")

label = soup.find("label", text="Name:")
print(label.next_sibling.strip())
```

Imprime `John Smith` .

Usando selectores de CSS para ubicar elementos en BeautifulSoup

BeautifulSoup tiene un soporte limitado para los selectores de CSS , pero cubre los más utilizados. Use el método `select()` para encontrar múltiples elementos y `select_one()` para encontrar un solo elemento.

Ejemplo básico:

```
from bs4 import BeautifulSoup

data = """
<ul>
    <li class="item">item1</li>
    <li class="item">item2</li>
    <li class="item">item3</li>
</ul>
"""
```

```
soup = BeautifulSoup(data, "html.parser")

for item in soup.select("li.item"):
    print(item.get_text())
```

Huellas dactilares:

```
item1
item2
item3
```

Ubicando comentarios

Para ubicar los comentarios en `BeautifulSoup` , use el argumento de `text` (o `string` en las versiones recientes) que verifique el tipo de `Comment` :

```
from bs4 import BeautifulSoup
from bs4 import Comment

data = """
<html>
    <body>
        <div>
        <!-- desired text -->
        </div>
    </body>
</html>
"""

soup = BeautifulSoup(data, "html.parser")
comment = soup.find(text=lambda text: isinstance(text, Comment))
print(comment)
```

Imprime el `desired text` .

Funciones de filtro

BeautifulSoup le permite filtrar resultados al proporcionar una función para `find_all` y todas funciones similares. Esto puede ser útil para filtros complejos, así como una herramienta para la reutilización de código.

Uso básico

Defina una función que tome un elemento como único argumento. La función debería devolver `True` si el argumento coincide.

```
def has_href(tag):
    '''Returns True for tags with a href attribute'''
    return  bool(tag.get("href"))
```

```
soup.find_all(has_href) #find all elements with a href attribute
#equivilent using lambda:
soup.find_all(lambda tag: bool(tag.get("href")))
```

Otro ejemplo que encuentra etiquetas con un valor `href` que no comienza con

Proporcionar argumentos adicionales para filtrar funciones

Dado que la función pasada a `find_all` solo puede tomar un argumento, a veces es útil hacer que las "fábricas de funciones" que producen funciones se ajusten para su uso en `find_all` . Esto es útil para hacer que las funciones de búsqueda de etiquetas sean más flexibles.

```
def present_in_href(check_string):
    return lambda tag: tag.get("href") and check_string in tag.get("href")

soup.find_all(present_in_href("/partial/path"))
```

Acceso a etiquetas internas y sus atributos de etiqueta inicialmente seleccionada

Supongamos que tienes un `html` después de seleccionar con `soup.find('div', class_='base class')` :

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(SomePage, 'lxml')
html = soup.find('div', class_='base class')
print(html)

<div class="base class">
  <div>Sample text 1</div>
  <div>Sample text 2</div>
  <div>
    <a class="ordinary link" href="https://example.com">URL text</a>
  </div>
</div>

<div class="Confusing class"></div>
'''
```

Y si desea acceder a `href` la etiqueta `<a>` , puede hacerlo de esta manera:

```
a_tag = html.a
link = a_tag['href']
print(link)

https://example.com
```

Esto es útil cuando no puede seleccionar directamente la etiqueta `<a>` porque los `attrs` no le dan

una identificación única, hay otras etiquetas "gemelas" `<a>` en la página analizada. Pero puede seleccionar de forma única una etiqueta principal que contenga la `<a>` necesaria.

Recopilación de elementos opcionales y / o sus atributos de series de páginas.

Consideremos una situación cuando analiza el número de páginas y desea recopilar el valor del elemento que es *opcional* (puede presentarse en una página y puede estar ausente en otra) para una página específica.

Además, el elemento en sí mismo, por ejemplo, es el *elemento más común* en la página, en otras palabras, ningún atributo específico puede ubicarlo de manera única. Pero ve que puede seleccionar correctamente su elemento principal y sabe el *número de orden* del elemento deseado en el nivel de anidamiento respectivo.

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(SomePage, 'lxml')
html = soup.find('div', class_='base class') # Below it refers to html_1 and html_2
```

El elemento deseado es opcional, por lo que podría haber 2 situaciones para que `html` sea:

```
html_1 = '''
<div class="base class">    # №0
  <div>Sample text 1</div>  # №1
  <div>Sample text 2</div>  # №2
  <div>!Needed text!</div>  # №3
</div>

<div>Confusing div text</div>  # №4
'''

html_2 = '''
<div class="base class">    # №0
  <div>Sample text 1</div>  # №1
  <div>Sample text 2</div>  # №2
</div>

<div>Confusing div text</div>  # №4
'''
```

¡Si tienes `html_1` puedes recopilar `!Needed text!` de la etiqueta №3 de esta manera:

```
wanted tag = html_1.div.find_next_sibling().find_next_sibling() # this gives you whole tag №3
```

Inicialmente obtiene div1 `div` , luego 2 veces cambia al siguiente `div` en el mismo nivel de anidamiento para llegar a №3.

```
wanted_text = wanted_tag.text # extracting !Needed text!
```

La utilidad de este enfoque viene cuando se obtiene `html_2` : el enfoque no le dará un error, le

dará `None` :

```
print(html_2.div.find_next_sibling().find_next_sibling())
None
```

El uso de `find_next_sibling()` aquí es crucial porque limita la búsqueda de elementos por nivel de anidamiento respectivo. Si utilizarías `find_next()` , la etiqueta №4 se recopilará y no la querrás:

```
print(html_2.div.find_next().find_next())
<div>Confusing div text</div>
```

También puede explorar `find_previous_sibling()` y `find_previous()` que funcionan de manera opuesta.

Todas las funciones descritas tienen sus *múltiples* variantes para capturar todas las etiquetas, no solo la primera:

```
find_next_siblings()
find_previous_siblings()
find_all_next()
find_all_previous()
```

Lea Elementos de localización en línea:
https://riptutorial.com/es/beautifulsoup/topic/1940/elementos-de-localizacion

Creditos

S. No	Capítulos	Contributors
1	Empezando con beautifulsoup	Antti Haapala, Community, Dair, DMPierre, Larry Cai, vikingben, Whirl Mind
2	Elementos de localización	alecxe, Dmitriy Fialkovskiy, sytech