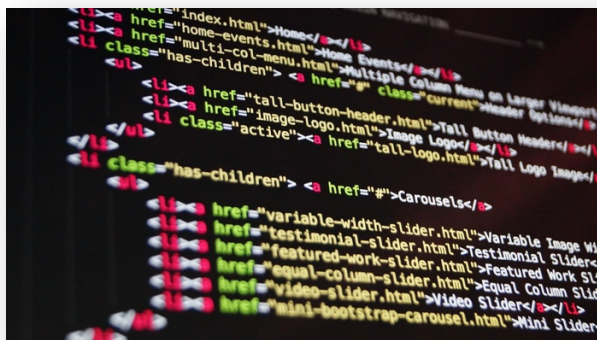


Adquisición datos con BeautifulSoup

En la red podemos encontrar muchas páginas que publican información constantemente que puede resultar de nuestro interés. En bastantes ocasiones esta información no está adaptada a poder usarla de manera automatizada y requiere de técnicas y herramientas para obtener la información del conjunto de la página.



Parte de los conocimientos necesarios para obtener datos de una página web consiste en entender el lenguaje de marcas HTML en el que están escritas finalmente las páginas web. En esencia, la página web que vemos en el navegador es una interpretación del lenguaje HTML. Este lenguaje consiste en un conjunto de "etiquetas" identificadas entre puntas de flecha (<>) que pueden encerrar al texto o elementos a los que se aplica.

No es el objetivo de esta guía profundizar en HTML, daremos por supuesto un conocimiento de HTML suficiente como para interpretar el código de una web ya hecha.

Una alternativa de fuerza bruta para obtener resultados sería descargar la web con una petición a "requests" y recorrer el archivo usando métodos Python en busca de la información que buscamos.

La técnica que vamos a usar se llama **"Web Scrapping"** Librerías como BeautifulSoup o Scrapy facilitan esta búsqueda proporcionando métodos como buscar por id o por nombre que nos hacen transparente toda la lógica necesaria en Python para realizar la misma tarea. Por su sencillez, en esta guía nos centraremos en "BeautifulSoup"

BeautifulSoup

Para hacer "web scrapping" con la librería "BeautifulSoup" necesitaremos instalarla con pip o conda en el entorno que estemos usando

```
pip install beautifulsoup4  
conda install beautifulsoup4
```

Como también solicitaremos la descarga de la web en cuestión necesitaremos la librería "requests" que se instala de la misma manera. Con esto tendremos solucionadas las dependencias de esta parte.

Desde la web oficial de Beautiful Soup (<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>) tenemos disponible la documentación necesaria para poder consultar.

En general, todos los proyectos en los que usemos esta librería tendrán un aspecto muy similar a este:

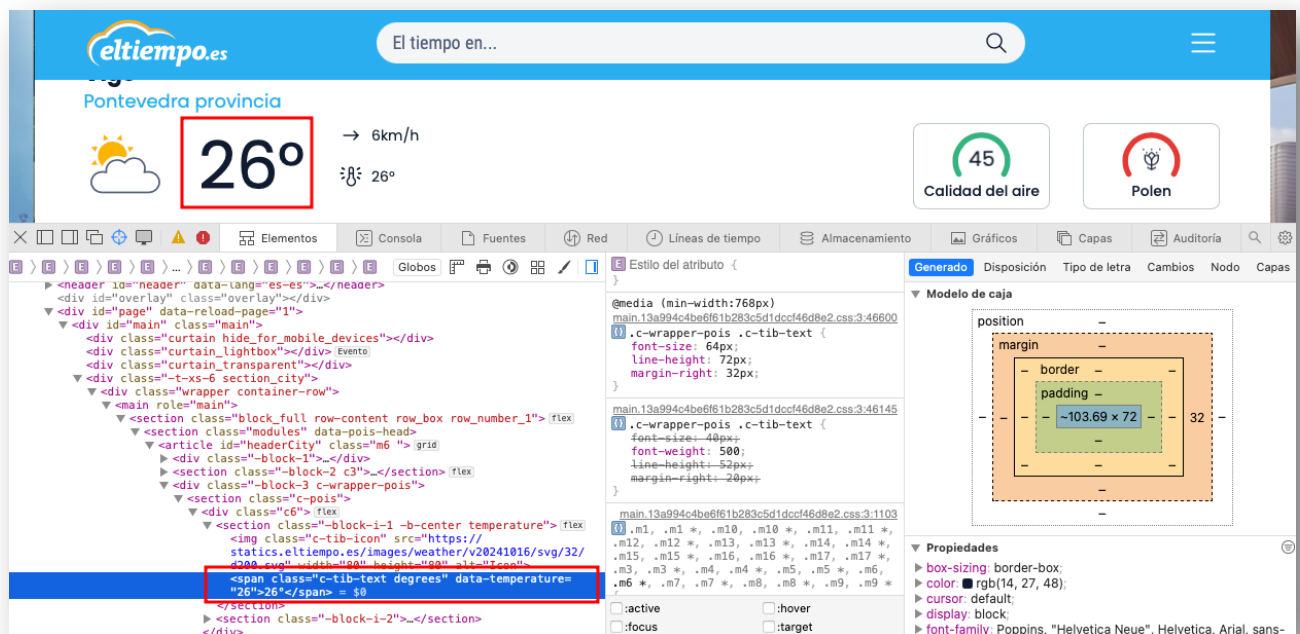
```
from bs4 import BeautifulSoup  
import requests  
  
url = "https://www.eltiempo.es/vigo.html"  
response = requests.get(url)  
html = response.text  
soup = BeautifulSoup(html, "html.parser")
```

Con este fragmento de código nos haremos una petición GET a la URL que indiquemos. Lo esperado es que nos devuelva un objeto de tipo "Response" con la respuesta HTTP. En el "status_code" debería venir un "200" indicando que todo ha funcionado bien. En ese caso podemos usar la propiedad "text" para descargarnos el HTML, XML, JSON o cualquier archivo en texto plano. Por último creamos el objeto de tipo "BeautifulSoup" para poder analizar el código.

Web Scrapping sencillo

Partiendo del código anterior vamos a hacer "web scrapping" de esa web para obtener un dato concreto y sencillo dentro de la página, la temperatura actual en Vigo desde la web de eltiempo.es.

Siempre tendremos que inspeccionar el HTML para localizar la información que nos interesa.



Con ayuda del inspector que incluyen todos los navegadores podemos apuntar a la parte de la web que nos interesa e inmediatamente obtenemos su etiqueta.

En BeautifulSoup tenemos dos métodos muy utilizados: **find()** y **find_all()**. Su nombre es bastante descriptivo y, como es de esperar, el primero localiza la primera aparición del filtro que pasemos mientras que el segundo devuelve un array de todas las apariciones de ese filtro. Es altamente recomendable echar un ojo a la documentación para conocer las posibilidades que nos ofrecen los filtros. Por ejemplo, un filtro "True" devuelve todo. En caso de usar **find(True)** solo nos devolverá la etiqueta `<html Lang="es-es">`. Si usamos **find_all(True)** nos devolverá un array con todas las etiquetas del HTML.

En este caso particular solo hay una ocurrencia de la etiqueta "span" con la clase "c-tib-text degrees" por lo que podemos usar **find()**.

Una vez tengamos aislada la etiqueta que nos interesa podemos acceder directamente a su contenido (lo que queda entre la etiqueta de apertura y cierre) usando el método **"text"**.

En esta ocasión vemos que en la propia etiqueta tenemos un atributo que almacena el valor en forma de entero así que también lo voy a usar a modo de ejemplo. Para acceder a los atributos de una etiqueta podemos utilizar el método **"get"** pasándole la clave de la propiedad. Abusando del lenguaje también es posible llamar a la clave entre corchetes directamente desde la variable de BeautifulSoup como si fuera un diccionario apuntando directamente a los atributos.

```
ejemplo.py x
beautifulsoup4 > ejemplo.py > ...
1 from bs4 import BeautifulSoup
2 import requests
3
4 url = "https://www.eltiempo.es/vigo.html"
5 response = requests.get(url) response = <Response [200]>, url = 'https://www.eltiempo.es/vigo.html'
6 html = response.text
7 soup = BeautifulSoup(html, "html.parser") soup = <!DOCTYPE html><html lang="es-es"><head><meta cha
8
9 etiqueta = soup.find("span", class_="c-tib-text degrees")
10
11 print(etiqueta["data-temperature"]) etiqueta = <span class="c-tib-text degrees" data-temperature="25">
12
```

VARIABLES
Locals
etiqueta = <bs4.element.Tag object at 0x114350a58>
attrs = {'class': ['c-tib-text', 'degrees'], 'data-temperature': '25'}
can_be_empty_element = False
cdata_list_attributes = {'*': ['class', 'accesskey', 'dropzone', 'a': ['rel', 'rev'], 'link': ['rel']
children = <list_iterator object at 0x11021b790>
contents = ['25']
css = <bs4.css.CSS object at 0x114350a58>
decomposed = False
default = <object object at 0x10fdeac0b>
descendants = <generator object Tag.descendants at 0x111d049e0>
hidden = False
interesting_string_types = (<class 'bs4.element.NavigableString'>, <class 'bs4.element.CData'>)
is_self_closing = False
is_empty_element = False
known_xml = False
name = 'span'
namespace = None
next = '25'
next_sibling = '\n'
next_element = '25'
next_elements = <generator object PageElement.next_elements at 0x114329300>
next_sibling = '\n'
next_siblings = <generator object PageElement.next_siblings at 0x114329480>
parent = <section class="block-i-1 -b-center temperature">
parents = <generator object PageElement.parents at 0x1143293c0>
prefix = None
preserve_whitespace_tags = {'pre', 'textarea'}
previous = '\n'
previous_sibling = '\n'

Siempre es bueno echar un ojo a la documentación o directamente depurar desde nuestro IDE y comprobar qué propiedades y métodos tiene la variable de BeautifulSoup.

```
ejemplo.py x
beautifulsoup4 > ejemplo.py > ...
1 from bs4 import BeautifulSoup
2 import requests
3
4 url = "https://www.eltiempo.es/vigo.html"
5 response = requests.get(url)
6 html = response.text
7 soup = BeautifulSoup(html, "html.parser")
8
9 etiqueta = soup.find("span", class_="c-tib-text degrees")
10
11 # Mostrando el CONTENIDO entre la etiqueta de apertura y cierre
12 valor = etiqueta.text
13 print("El contenido es ",valor)
14
15 # Mostrando el valor del atributo data-temperature directamente
16 valor = etiqueta["data-temperature"]
17 print("El valor del atributo tomado directamente es ",valor)
18
19 # Mostrando el valor del atributo data-temperature usando get
20 valor = etiqueta.get("data-temperature")
21 print("El valor del atributo usando get es ",valor)
22
23
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS COMENTARIOS

```
(base) javi@Javiers-MacBook-Pro Proyectos % /Users/javi/anaconda3/bin/python
El contenido es 25
El valor del atributo tomado directamente es 25
El valor del atributo usando get es 25
(base) javi@Javiers-MacBook-Pro Proyectos %
```

Al capturar los datos debemos tener en cuenta que tal vez requieran algún tipo de procesado. En este caso, capturar la temperatura con una cadena acabada en "o" o capturar solamente el número puede facilitar mucho el código a la hora de interactuar con ese número.

En general, si vamos a operar matemática o lógicamente con ese dato es conveniente asegurarnos que la conversión se hace correctamente o arrastraremos un error que será difícil de detectar después. Un código postal, un número de teléfono o un DNI nunca lo vamos a sumar con otro ni vamos a comparar dos de ellos para ver cuál es el mayor, en ese caso podemos capturarlo como una cadena sin demasiado problema.

En caso de querer comparar o aplicar alguna operación aritmética entonces hay que preocuparse en el código de que la transformación se hace correctamente. Por ejemplo, la letra del DNI se calcula mediante una operación matemática sencilla; en este caso habría que tratar el DNI como un número.

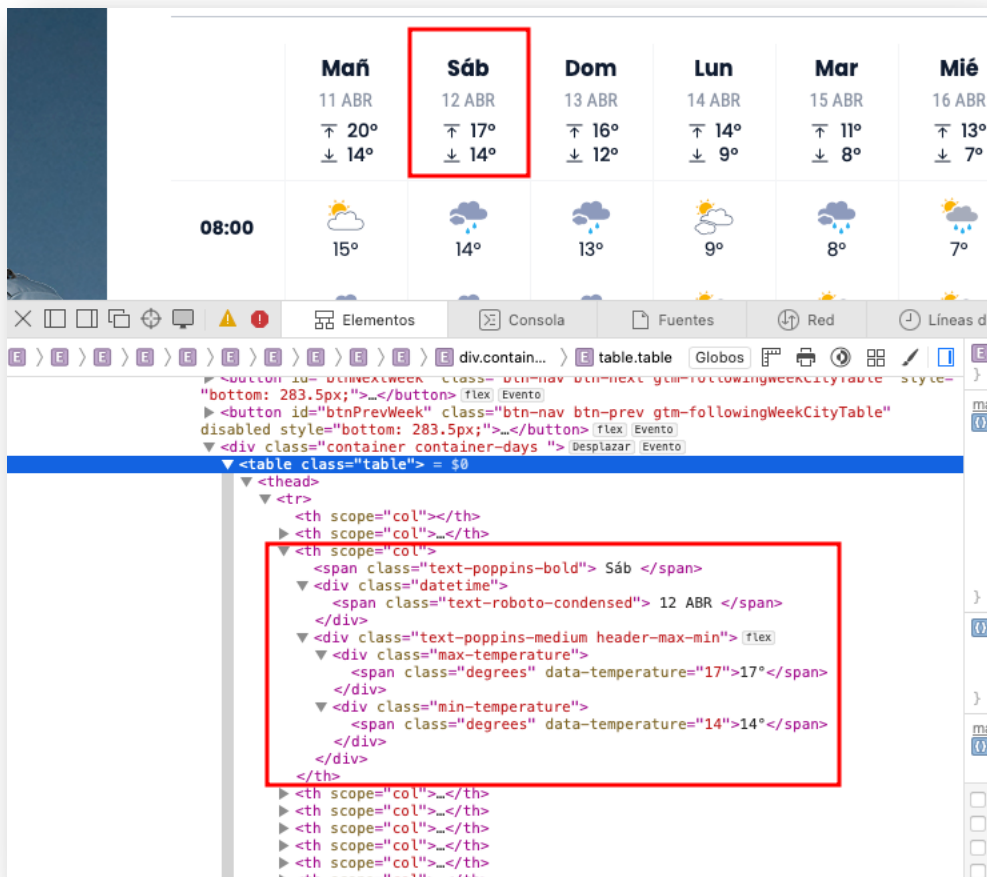
En el caso de la temperatura es altamente probable que quiera operar matemáticamente (para calcular la máxima del día o la media) y que lo quiera comparar con otras temperaturas (para saber si está subiendo o bajando) así que preferente debería usar el número.

Web Scrapping medio

En muchas ocasiones los datos los encontramos en forma de tabla por lo que debemos conocer la estructura básica de una tabla en HTML. La norma es definir y, por tanto, recorrer la tabla de fila en fila. Piensa que se recorre de la misma manera que escribimos a lápiz y papel, de izquierda a derecha y de arriba abajo. Podemos esperar una etiqueta "table" que marca el principio y fin de la tabla. Dentro de esta etiqueta tendremos tantas etiquetas "tr" (table row) como filas tengamos en la tabla. Dentro de cada fila tendremos tantos "td" (table data) como columnas tenga esa fila. A partir de aquí se puede complicar una tabla hasta el infinito ya sea definiendo celdas combinadas o incluyendo cuerpo y cabeceras.

Como ejemplo intermedio vamos a recorrer una tabla para capturar sus datos en Python, en concreto los de la primera fila (la cabecera) donde pone el día de la semana, fecha y temperaturas máxima y mínima. Usaremos la tabla que aparece en la web donde se muestra la predicción en Vigo. Al inspeccionarla encontramos la etiqueta "table" con un único atributo indicando que su clase es "table". Volvemos a tener suerte, y esta combinación de etiqueta y atributo es única en la página por lo que podemos usar find() para obtener directamente la etiqueta.

Hay que tener en cuenta que la etiqueta incluye desde "<table>" hasta "</table>" por lo que dentro de esa variable tendremos todo el contenido.



The screenshot shows a web application interface with a weather forecast table. The table has columns for days of the week (Mañ, Sáb, Dom, Lun, Mar, Mié) and rows for dates (11 ABR, 12 ABR, 13 ABR, 14 ABR, 15 ABR, 16 ABR). The 'Sáb' column for '12 ABR' is highlighted with a red box. Below the table, the browser's developer tools are open, showing the HTML structure of the table. The HTML structure is as follows:

```

<table class="table"> = $0
  <thead>
    <tr>
      <th scope="col"></th>
      <th scope="col">_</th>
      <th scope="col">
        <span class="text-poppins-bold"> Sáb </span>
        <div class="datetime">
          <span class="text-roboto-condensed"> 12 ABR </span>
        </div>
        <div class="text-poppins-medium header-max-min"> (flex)
          <div class="max-temperature">
            <span class="degrees" data-temperature="17">17°</span>
          </div>
          <div class="min-temperature">
            <span class="degrees" data-temperature="14">14°</span>
          </div>
        </div>
      </th>
      <th scope="col"></th>
      <th scope="col"></th>
      <th scope="col"></th>
      <th scope="col"></th>
      <th scope="col"></th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>08:00</td>
      <td>15°</td>
      <td>14°</td>
      <td>13°</td>
      <td>9°</td>
      <td>8°</td>
      <td>7°</td>
    </tr>
  </tbody>
</table>

```

En la captura anterior podemos ver que esta tabla tiene una cabecera y un cuerpo. La parte que nos interesa está en la cabecera ("thead"). Podemos ver que en la primera fila ("tr") no hay nada. En la captura he desplegado la tercera celda de la fila de cabecera y allí vemos que el contenido está desperdigado en varios bloques y que va a requerir procesamiento.

Analizando el código HTML podemos ver que las celdas que buscamos (y solo ellas) comparten el atributo con clave "scope" y valor "col". Es una situación ideal para utilizar find_all para obtener todas las celdas en un array.

Ahora que tenemos todo en un array podemos repetir el proceso dentro de un bucle "for" buscando para cada celda los 4 datos que tenemos. Al usar un iterador en el bucle "for" tenemos una variable de tipo BeautifulSoup apuntando a una celda en cada iteración. Dentro de esa celda el dato no está accesible directamente y tendremos que buscar cada una de las etiquetas que lo contienen. Como en cada celda la aparición de cada etiqueta es única podemos usar find().

Puede consultar la documentación de BeautifulSoup para obtener ayuda y ejemplos de estos métodos.

Para nosotros será suficiente dominar el uso de "find" y "find_all" con los filtros adecuados. La verdadera complicación en "web scrapping" consiste en investigar la estructura de la web y estar atentos a los cambios que introduce el propietario.

Aunque la información de los sitios web es pública muchos propietarios luchan contra la recolección automática de datos porque pierden ingresos por publicidad y análisis del comportamiento de los usuarios.

La manera clásica era forzar al usuario a un desafío como un CAPTCHA o algún tipo de autenticación antes de mostrar los datos.

En muchas ocasiones los sitios webs limitan la cantidad de solicitudes por segundo dando por hecho que un número elevado de solicitudes hechas en un par de segundos se debe a un web scrapping automatizado. En estos casos las IPs pueden quedar en una lista negra de IPs bloqueadas.

Otra forma habitual para evitar el web scrapping es forzar la carga lenta de la página para que los filtros no encuentren las etiquetas más preciadas.

Es habitual que los propietarios de las webs hagan cambios importantes en la estructura de manera que los filtros que utilizemos (o XPATH,...) ya no funcionen y tengamos que inspeccionar de nuevo la web.

También nos encontraremos con webs que usan mucho JavaScript que entran en funcionamiento cuando la web ya se ha cargado haciendo que los filtros no obtengan ningún dato usable.

Esto es el juego del gato y el ratón en el que para cada impedimento hay una nueva técnica.