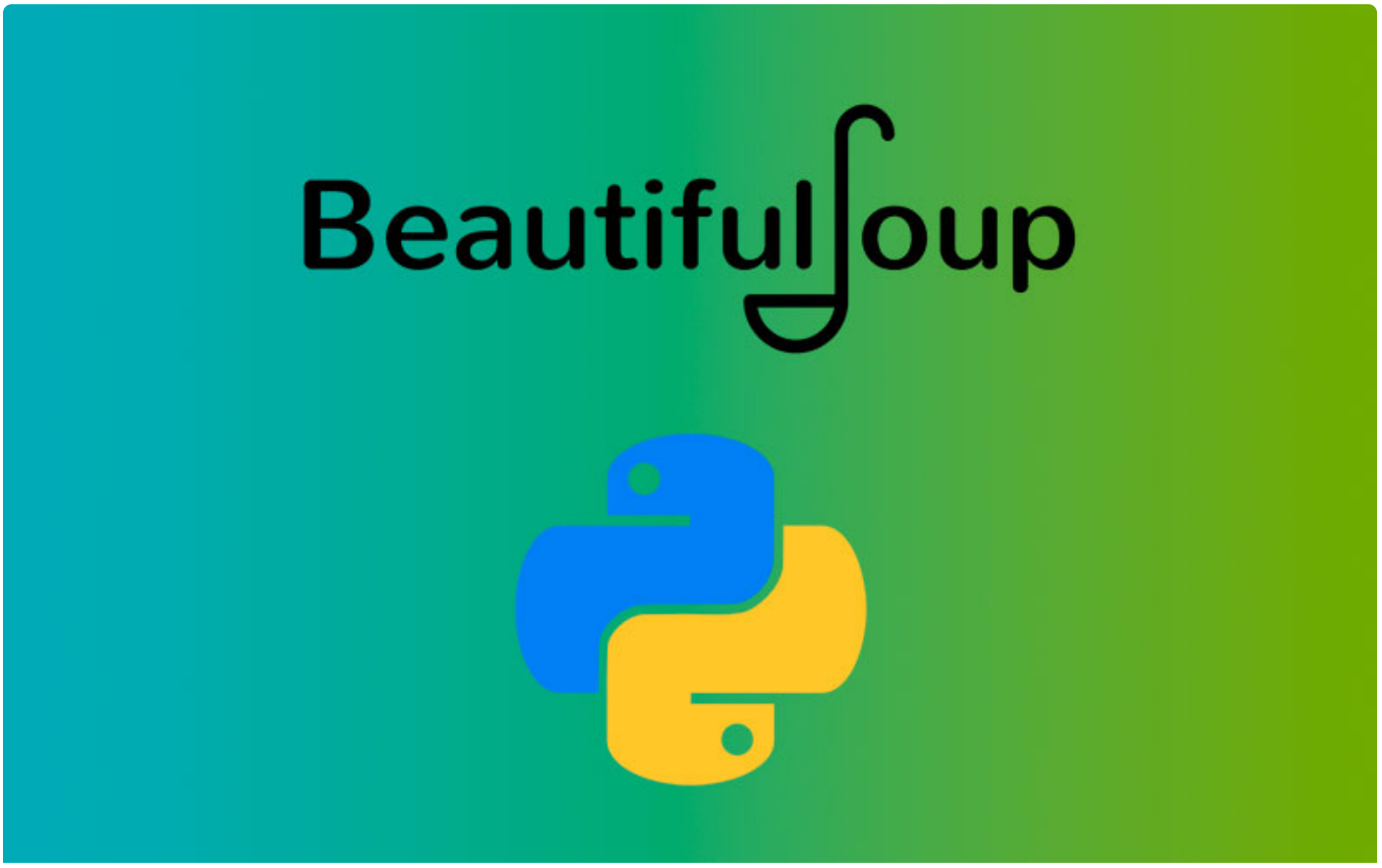
cosasdedevs_



Web scraping con requests y BeautifulSoup en Python

Alber
Abr 09, 2020

¡Ey! Espero que estéis aprovechando el confinamiento aprendiendo cosas, yo ahora estoy

trabajando en un proyecto y haciendo un curso de Big Data que espero que me ayude a poder crear más y mejor contenido en el blog.

Y bueno que me voy por las ramas, en el tutorial de hoy os quiero explicar como hacer **web scraping** de una web con las librerías **requests** y **bs4** de **Python,** pero definamos este término.

¿Qué es web scraping?

Para el que no lo sepa, **web scraping** es una técnica mediante software para extraer información de una web, por ejemplo google y otros buscadores lo usan para extraer la información de una web que luego es la que se muestra en los resultados de búsqueda.

Para empezar con nuestro proyecto, creamos una carpeta y dentro de ella generaremos el entorno virtual, si no sabéis como hacerlo **aquí lo explico**. Una vez creado instalaremos estas tres librerías que son las que utilizaremos.

```
pip install requests
pip install bs4
pip install lxml
```

Requests

La librería **requests** la utilizaremos para realizar las peticiones a la página de la que vamos a extraer los datos. Esta librería permite realizar peticiones usando cualquiera de los métodos (get, post, put, delete, patch), envío de parámetros y modificaciones de las cabeceras. Si queréis más información os dejo **la documentación de requests**.

Beautiful Soup

Beautiful Soup o bs4 es una librería que se utiliza para extraer datos de htmls y xml, como veréis a continuación, esta librería nos facilitará el trabajo a la hora de extraer la información.

La librería **lxml** será el parser que utilizaremos junto con **bs4** para realizar él parseo, es la que recomiendan en la documentación de **Beautiful Soup** por su velocidad aunque sea una librería externa así que vamos a hacerles caso 😄

Una vez activado el entorno virtual e instalado las librerías vamos a empezar a trabajar en nuestro proyecto, lo que vamos a hacer es extraer la información de los artículos principales de esta página **https://www.cmmedia.es/noticias/castilla-la-mancha/**, para ello crearemos el archivo **scraping.py** que es el archivo con el que trabajaremos.

Añadimos la primera función llamada **get_main_news()** y le decimos el punto de entrada a nuestro script con **if __name__ == '__main__':**

```
import requests
from bs4 import BeautifulSoup

if __name__ == '__main__':
    noticias = get_main_news()

# La función get_main_news retornará un diccionario con todas las urls y tí
def get_main_news():
    url = 'https://www.cmmedia.es/noticias/castilla-la-mancha/'

    respuesta = requests.get(
        url,
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleW
        }
    )
```

En esta función lo que hacemos es usar la librería **requests** para realizar la petición que será de tipo **get** y a parte de la url le pasaremos por cabecera un user agent válido para evitar problemas de compatibilidad por parte de la web.

Ahora que tenemos la respuesta, vamos a ver algunos datos que se nos devolverá en la variable respuesta.

```
# Si queremos ver la respuesta que ha dado el servidor al hacer la petición
print(respuesta.status_code)

# Para ver el contenido utilizaremos text, aquí se guarda todo el html de l
print(respuesta.text)

# Con header veremos las cabeceras que nos devuelve la página, esta puede c
print(respuesta.headers)

# Aquí podemos ver la cabecera de la petición, es decir, la que enviamos nc
print(respuesta.request.headers)

# Para ver el método que hemos usado lo haremos con method, en nuestro casc
print(respuesta.request.method)
```

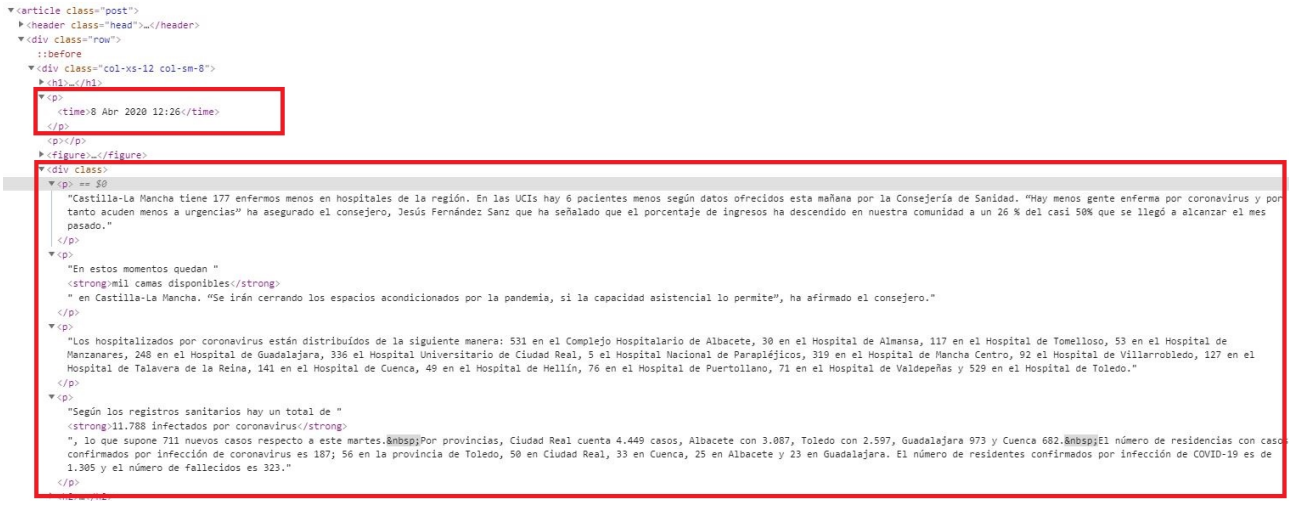
Una vez hecha la petición, vamos a extraer la información que nos interesa, para ello pasaren

todo el contenido de la respuesta a **BeautifulSoup** para que haga su magia.

```
contenido_web = BeautifulSoup(respuesta.text, 'lxml')
```

Ahora que tenemos parseado el html es hora de extraer los datos que nos interesen, nosotros queremos extraer los artículos que mostramos en la imagen mostrada a continuación y para localizar los datos que queremos extraer utilizaremos el inspector web de nuestro navegador. Para utilizar el inspector de un navegador debéis pulsar f12 o desde las opciones de vuestro navegador buscad herramientas de desarrollador y ahí os aparecerá, en mi caso estoy usando Chrome pero podéis usar cualquier navegador.

Una vez activado el inspector, seleccionamos el elemento que queremos buscar nos aparece algo como esto.



Como veis ahí tenemos la url y el título, ahora vamos a proceder a extraerlo con BeautifulSoup y de paso ver algunas de sus funciones.

```
noticias = contenido_web.find('ul', attrs={'class':'news-list'})
articulos = noticias.findChildren('div', attrs={'class':'media-body'})
```

Lo que estamos haciendo en estas líneas es buscar la etiqueta **ul,** que tenga la clase **news-list,** que es la lista que contiene todas las urls. Con el resultado de esa búsqueda, utilizaremos **findChildren** para encontrar todos los **divs** que usen la clase que **media-body** que es la que contiene la url y el título, esto nos devolverá una lista con todos los resultados encontrados.

Después recorreremos la lista para obtener la url y el título, para ello usaremos el siguiente código:

```
noticias = [];

for articulo in articulos:
    print('================================')
    print(articulo.find('h3').a.get('href'))
    print(articulo.find('h3').get_text())
    print('================================')
    noticias.append({
        'url': articulo.find('h3').a.get('href'),
        'titulo': articulo.find('h3').get_text()
    })

return noticias
```

Para obtener los datos que nos interesan buscaremos en la etiqueta **h3** y los extraeremos usando **a.get('href')** para extraer el link y **get_text()** para extraer el texto.

Una vez hecho esto retornaremos el listado para posteriormente acceder a esas páginas y extraer más información relevante.

Volvemos al main de nuestro código y realizamos las siguientes modificaciones:

```
if __name__ == '__main__':
    noticias = get_main_news()

    for noticia in noticias:
        noticia = get_all_info_by_new(noticia)
        print('================================')
        print(noticia)
        print('================================')
```

Ahora recorreremos el resultado de nuestro parseo de urls y esto lo haremos para extraer el resto de la información con la función get_all_info_by_new pasando por parámetro la noticia para poder acceder a su url, después imprimiremos el resultado con los nuevos datos añadidos.

Antes de ponernos con la función get_all_info_by_new y, ya que la petición a requests va a ser prácticamente igual, para no duplicar código, crearemos una función para lanzar los requests llamada launch_requests que recibirá por parámetro la url que queremos enviar.

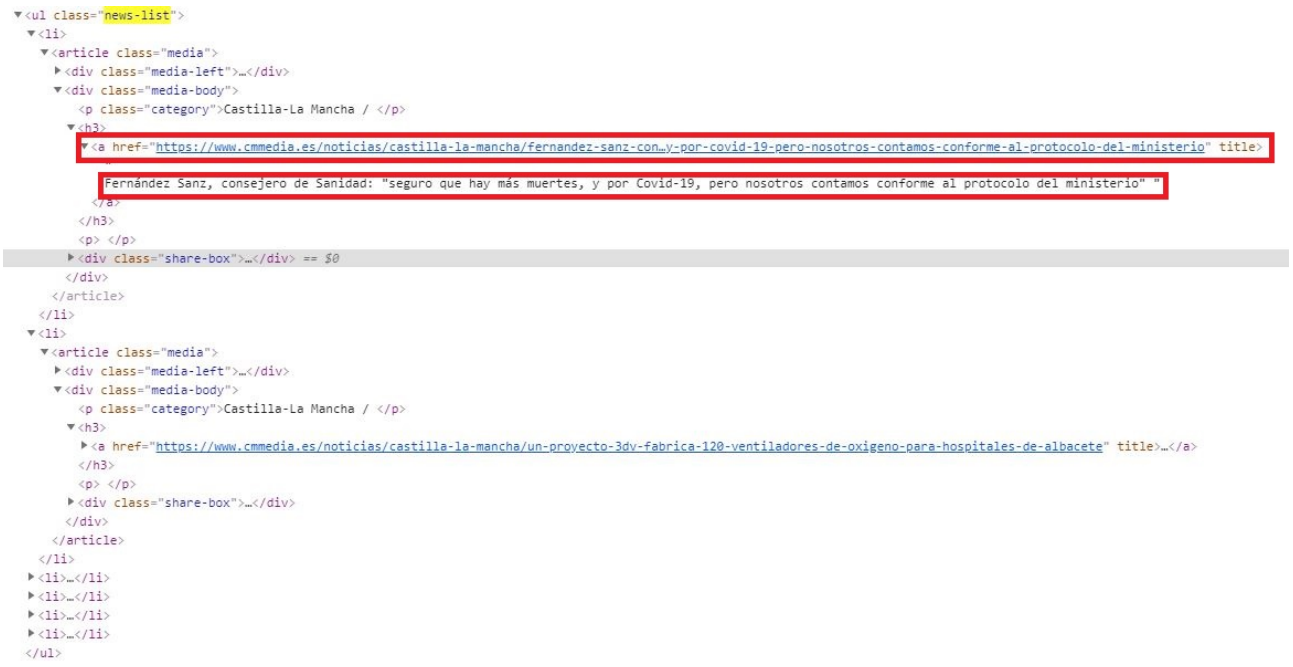
```
def launch_request(url):
    try:
        respuesta = requests.get(
            url,
            headers = {
                'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) Ap
            }
        )
        respuesta.raise_for_status()
    except requests.exceptions.HTTPError as err:
        raise SystemExit(err)


    return respuesta
```

Además en esta función añadiremos unos controles extra, encapsularemos todo en un bloque try y después de la petición lanzaremos la función respuesta.raise_for_status(), si no devuelve un estado válido lanzará una excepción que parará la ejecución.

Ahora que tenemos nuestro lanzador de peticiones, creamos la función get_all_info_by_new

recibirá por parámetro un diccionario con la url y el título y se encargará de extraer todo el artículo y la fecha.

Como en el caso anterior vamos al inspector de elementos y buscamos la información que nos interesa, lo podéis ver en la siguiente imagen.



Aquí se ve que todo está contenido en una etiqueta de tipo **article** que usa la clase post, la fecha está dentro de una etiqueta **time** y el texto del artículo está dentro de un **div** sin clase ni identificador que nos va a poner las cosas un poco difíciles, ya que es ambiguo pero bueno, seguro que lo conseguimos ;)

Dentro de la función añadimos el siguiente código:

```
def get_all_info_by_new(noticia):

    print(f'Scrapping {noticia["titulo"]}')

    respuesta = launch_request(noticia["url"])

    contenido_web = BeautifulSoup(respuesta.text, 'lxml')

    articulo = contenido_web.find('article', attrs={'class':'post'})

    noticia['fecha'] = articulo.find('time').get_text()
    noticia['articulo'] = articulo.find('div', attrs={'class':'', 'id':''})
```

Lanzamos la petición, la parseamos con **BeautifulSoup** y extraemos la información del artícu

la función **find**, después dentro del contenido del artículo buscamos la etiqueta **time** y extraemos su texto. Por último para obtener el texto del artículo, usamos **find** para buscar un **div** que no tenga ni clase ni identificador y obtenemos su texto.

Y listo, con esto ya tendríamos nuestro proyecto para extraer noticias, si queréis aprender más sobre estas librerías os recomiendo que practiquéis intentando extraer la información de páginas que os interesen y recodad que si tenéis dudas podéis dejarlas en la caja de comentarios.

Como siempre os dejo **el link al proyecto que hemos realizado en GitHub**.

Espero que os haya gustado y como siempre, os recomiendo seguirme en **Twitter** para estar al tanto de los nuevos tutoriales y ahora también podéis seguirme en **Instagram** donde estoy subiendo **tips, tutoriales en vídeo e información sobre herramientas para developers**.

Nos leemos 👋.

#PYTHON

Sobre mi

👋 Mi nombre es Alber, soy **Backend PHP/Python/Node.js Developer** y de vez en cuando enredo también en el **Frontend**.

Saber más

También te puede interesar

- Parte 6: Tests en FastAPI
- Parte 5: Cómo crear un CRUD con FastAPI
- Parte 4: Autenticación con JWT en FastAPI