

Sistemas Web

2022-02-24 PA

Grado en Informática de Gestión
y Sistemas de Información

Dpto. de Ingeniería de Sistemas y Automática

Web Scrapping

Ejemplo

Oskar Casquero (oskar.casquero@ehu.eus)
María Luz Álvarez (marialuz.alvarez@ehu.eus)



Esta obra está bajo una [licencia de Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/).



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

BILBOKO
INGENIARITZA
ESKOLA
ESCUELA
DE INGENIERÍA
DE BILBAO



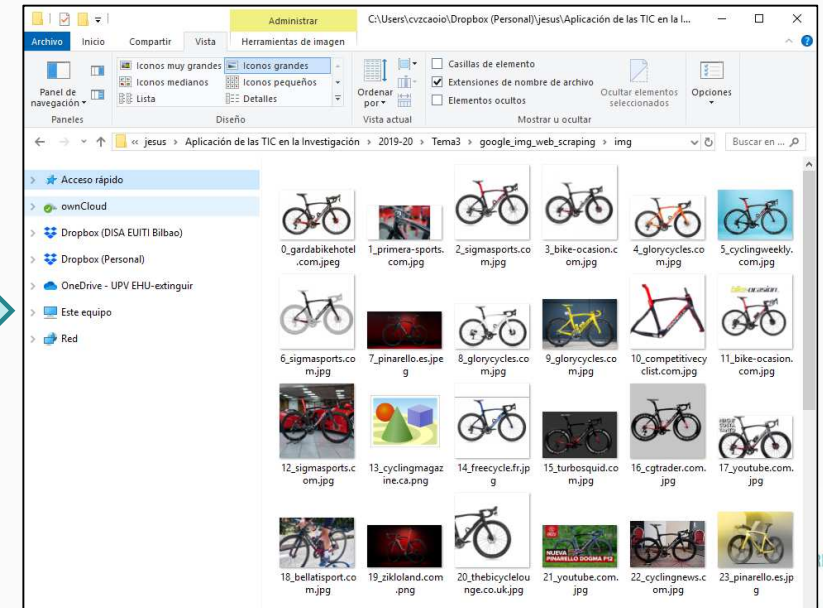
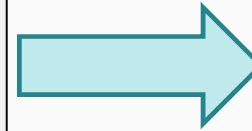
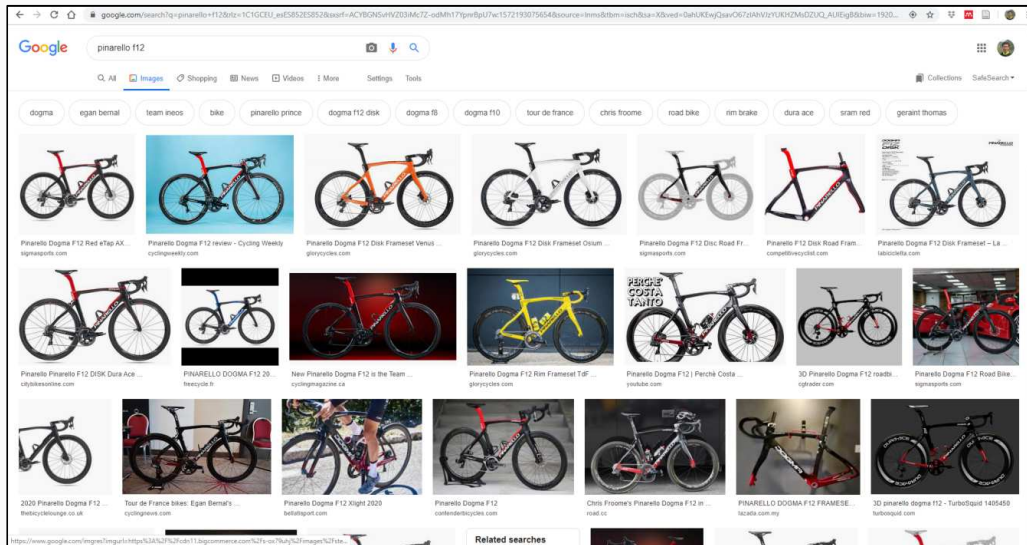
Estructura del tema

- Contenido y estructura de una pagina HTML
- Parsear una pagina con BeautifulSoup
- Pagina descargada vs pagina renderizada
- View Source Chart: bookmarklet para ver la pagina renderizada
- Selenium+Geckodirve: automatizar el navegador desde Python para renderizar HTML
- **Ejemplo:** descargar imágenes de Google Images
 - ¿La página HTML descargada coincide con la renderizada?
 - Sí → Obtener código HTML con requests
 - No → Obtener código HTML con Selenium+Geckodirve
 - Analizar estructura de la página HTML para saber dónde están los enlaces a las imágenes.
 - Parsear el HTML con BeautifulSoup
 - Almacenar imágenes

1

Ejemplo: Descargar imágenes de Google Images

Se quiere programar una aplicación que descargue automáticamente las imágenes obtenidas como resultado de una búsqueda en *Google Images* y las almacene en una carpeta local.

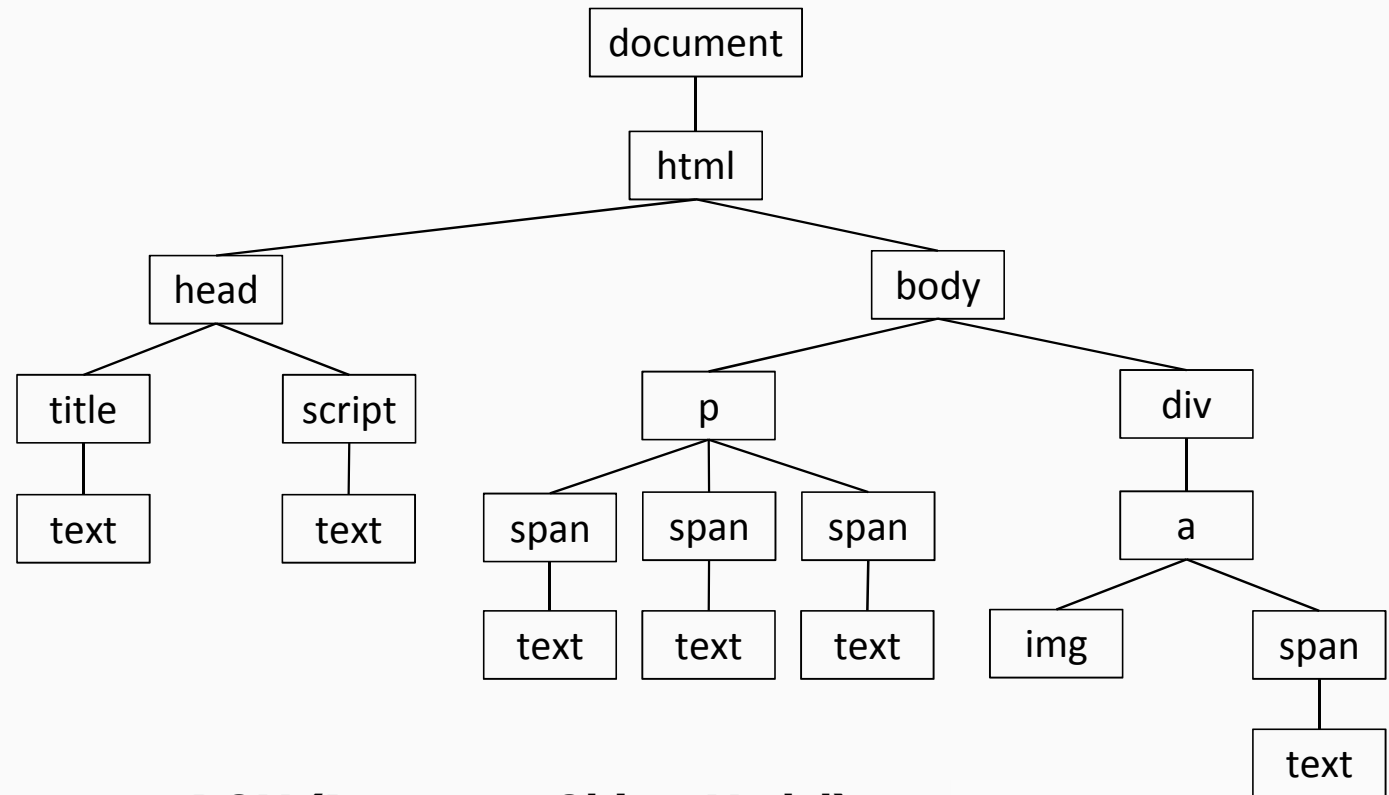


Estructura de una página HTML

```

<html>
  <head>
    <title>título</title>
    <script>
      var data = new Date();
      document.write("Client Date: ");
      document.write(data);
    </script>
  </head>
  <body>
    <p>
      <span class="a">bla</span>
      <span class="a">bla</span>
      <span class="a">bla</span>
    </p>
    <div id="identifier">
      <a href="http://...">
        
        <span>bla</span>
      </a>
    </div>
  </body>
</html>

```



DOM (Document Object Model)

3

Página descargada vs página renderizada

El navegador ejecuta el código javascript antes de renderizar la página.
El código javascript tiene capacidad para modificar la estructura y el contenido del HTML.

Página descargada

```
...  
<body>  
  Server Date: Fri Mar 04 08:43:33 CET 2016  
  <br/>  
  <script language="javascript">  
    var data = new Date();  
    document.write("Client Date: ");  
    document.write(data);  
  </script>  
</body>
```

**Crear una página HTML
con este contenido**

**Abrir página
en Firefox**

Página renderizada

```
Server Date: Fri Mar 04 08:43:33 CET 2016  
Client Date: Fri Mar 04 08:43:34 GMT+0100
```

View Source Chart: bookmarklet para ver la página renderizada

- La opción “Ver código fuente” del navegador permite ver el código de la página descargada.
- El bookmarklet “View Source Chart” permite ver el código de la página renderizada:
<http://viewsourcechart.com/getthebookmarklet.html> → **instalarlo en Firefox**

“View Page Source”

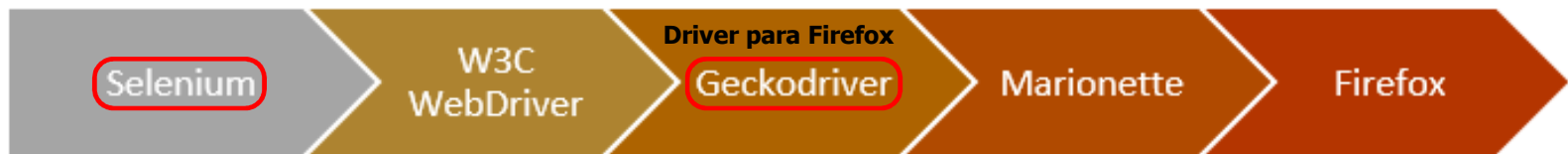
```
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>
    <ul id="lista"><li>Server Date: Tue Feb 21 9:28:33 CET 2020</li></ul>
    <br/>
    <script language="javascript">
      var data = new Date();
      var li_element = document.createElement("li");
      var text_element = document.createTextNode("Client Date: " +
        data);
      li_element.appendChild(text_element);
      document.getElementById("lista").appendChild(li_element);
    </script>
  </body>
</html>
```

“View Source Chart”

```
<html>
  <body>
    <ul id="lista">
      <li>
        Server Date: Tue Feb 21 9:28:33 CET 2020
      </li>
      <li>
        Client Date: Mon Feb 21 2022 11:59:32 GMT+0100 (hora estándar de Europa central)
      </li>
    </ul>
    <br>
    <script language="javascript">
      var data = new Date();
      var li_element = document.createElement("li");
      var text_element = document.createTextNode("Client Date: " + data);
      li_element.appendChild(text_element);
      document.getElementById("lista").appendChild(li_element);
    </script>
  </body>
</html>
```

Selenium+Geckodriver: automatizar navegador desde Python para renderizar HTML

- ¿Cómo podemos hacer uso del navegador desde un programa escrito en Python?
 - Mediante *Selenium*, un herramienta para la automatización de test para aplicaciones web.



```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

uri = "https://www.google.com/search?q=pinarello+f12&rlz"

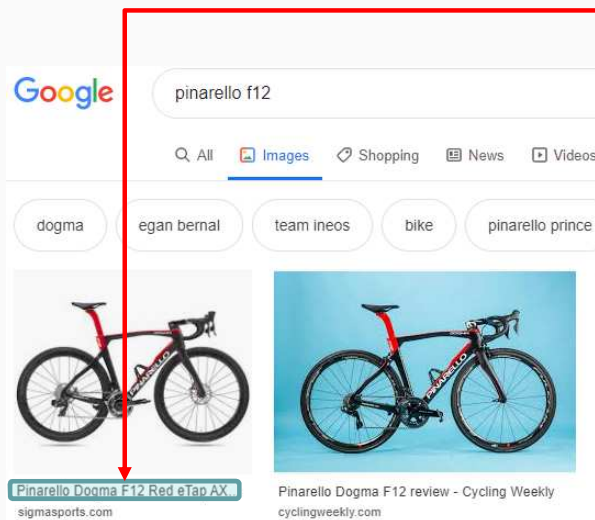
# abrir el navegador
browser = webdriver.Firefox()
# abrir la pagina
browser.get(uri)
# esperar hasta que se haya renderizado el elemento que nos interesa (timeout=30s)
WebDriverWait(browser, 30).until(EC.presence_of_all_elements_located((By.CLASS_NAME, "rg_i.Q4Luwd"))))
# obtener el código HTML
html = browser.page_source
# cerrar el navegador
browser.close()
```

6

¿La página HTML descargada coincide con la renderizada?

En Firefox, lanzar una búsqueda con el término "pinarello F12" en Google Images.

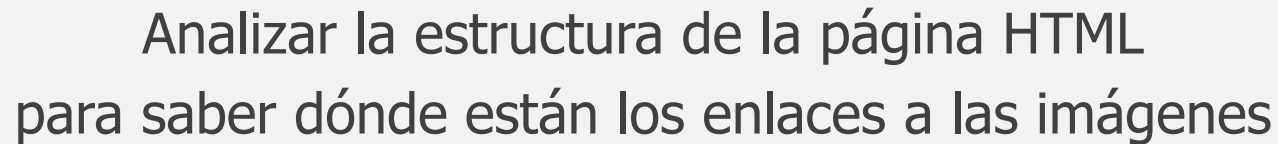
- Ejecutar la opción "Ver código fuente" sobre la página de resultados → **iFaltan las imágenes enlazadas!**
- Hacer lo mismo con el bookmarklet "View Source Chart" → **Aparecen todas las imágenes**



¿Dónde están las imágenes? Nos guiamos por alguna cadena de texto que nos permita ubicar una imagen concreta, por ejemplo "F12 Red eTap".

Buscamos dicha cadena en el código HTML y analizamos la estructura HTML donde se encuentra. Vemos que cada imagen está embebida dentro en un elemento "img" cuyo atributo "class" tiene el valor "rg_i Q4LuWd".

```
<div jsaction="IE7JUb;e5gl8b;MW7oTe:fl5lbf;dtRDf:s370ud;R3mad:ZCNXMe;v03O1c:cJhY7b;" data-ved="0CAsQMyhpahcKEwiYyL_Crt3nAhUAAAAHQAAAAQAg" data-ictx="1" data-id="r87qZtA34i52sM" jsname="N9Xkfe" data-ri="10" class="isv-r PNCib MSM1fd BUooTd" jscontroller="S4J6c" jsmodel="uZbpBf sB4qxc" jsdata="j0OpRe;r87qZtA34i52sM;$5" style="width: 186px; height: 231px;" data-tbnid="r87qZtA34i52sM" data-ct="0" data-cb="0" data-cl="0" data-cr="0" data-tw="221" data-ow="1021" data-oh="150">
  <a class="wXeWr islib nfEiy mM5pbd" jsname="sFXNd" jsaction="click:J9iaEb;" data-nav="1" tabindex="0" style="height: 189px;">
    <div class="bRMDJf islr" jsname="DeysSe" style="width: 186px; height: 191px; margin-top: -1px;" jsaction="mousedown:npT2md; touchstart:npT2md;">
      
    </div>
  </div>
  <div class="c7cjWc">
  </div>
  <div class="h312td RtIwE" jsname="boERI">
    <div class="O1vY7">
      <span class="">
        1021 x 1050
      </span>
    </div>
  </div>
  <div class="PiLIec" jsaction="click: gFs2Re">
  </div>
</a>
```

**Imagen enlazada
en atributo "data-src"**

b) imagen enlazada

Imagen *inline* en atributo "src"

- Repetimos la petición HTTP desde un cliente Python.
 - Para simular que este cliente es el navegador Firefox, incluimos la cabecera "***User-Agent***" en la petición, cuyo valor podemos obtener del análisis de cualquier petición con F12 en Firefox.

```
import requests

uri = "https://www.google.com/search?q=pinarello+f12&rlz=1C1GCEU_esES852ES852&sxsrf=ACYBGNSC3GcurH4DwTMGQcWPle0C5SNibw:1572336121832&source=..."
headers = {'Host': 'www.google.com',
           'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.130 Safari/537.36"}
res = requests.get(uri, headers=headers, allow_redirects=False)
print(res.content)
```

- **iFaltan las imágenes enlazadas!** Habrá que buscarlas en el código HTML de la página renderizada.
 - Hacemos uso del navegador para renderizar la página HTML.
 - Los motores de los navegadores disponen de interfaces para poder controlarlos.

Parsear el código HTML con BeautifulSoup

Para navegar por el árbol DOM del documento HTML utilizamos la librería BeautifulSoup:

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

```
from bs4 import BeautifulSoup

# instanciar un parser para html y cargar en memoria el DOM del html
# "soup" es una ref. al elemento raíz del DOM
document = BeautifulSoup(html, 'html.parser')
# buscar en el DOM todos aquellos elementos cuyo atributo "class" valga "rg_i Q4LuWd tx8vtf"
img_results = document.find_all('img', {'class': 'rg_i Q4LuWd'})
for idx, each in enumerate(img_results):
    src = ""
    if each.has_attr('src'):
        src = each['src']
    else:
        src = each['data-src']
print(str(idx) + " " + src)
```

Almacenar imágenes

Realizamos peticiones HTTP para descargar las imágenes enlazadas y decodificamos las imágenes *inline* de base64 a binario.

```
import base64

# instanciar un parser para html y cargar en memoria el DOM del html
# "soup" es una ref. al elemento raíz del DOM
document = BeautifulSoup(html, 'html.parser')
# buscar en el DOM todos aquellos elementos cuyo atributo "class" valga "rg_i Q4LuWd"
img_results = document.find_all('img', {'class': 'rg_i Q4LuWd'})
for idx, each in enumerate(img_results):
    src = ""
    if each.has_attr('src'):
        src = each['src']
    else:
        src = each['data-src']
    print(str(idx) + " " + src)

img = None
if src.find("data:image") != -1:
    # data:[<mime type>][;charset=<charset>][;base64],<encoded data>
    img = base64.b64decode(src.replace("data:image/jpeg;base64,", ""))
else:
    res = requests.get(src)
    img = res.content

file = open("./img/" + str(idx) + ".jpeg", "wb")
file.write(img)
file.close()
```