Sistemas Web

2021-02-19 M

Grado en Informática de Gestión
y Sistemas de Información

Dpto. de Ingeniería de Sistemas y Automática

Web Scraping

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

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

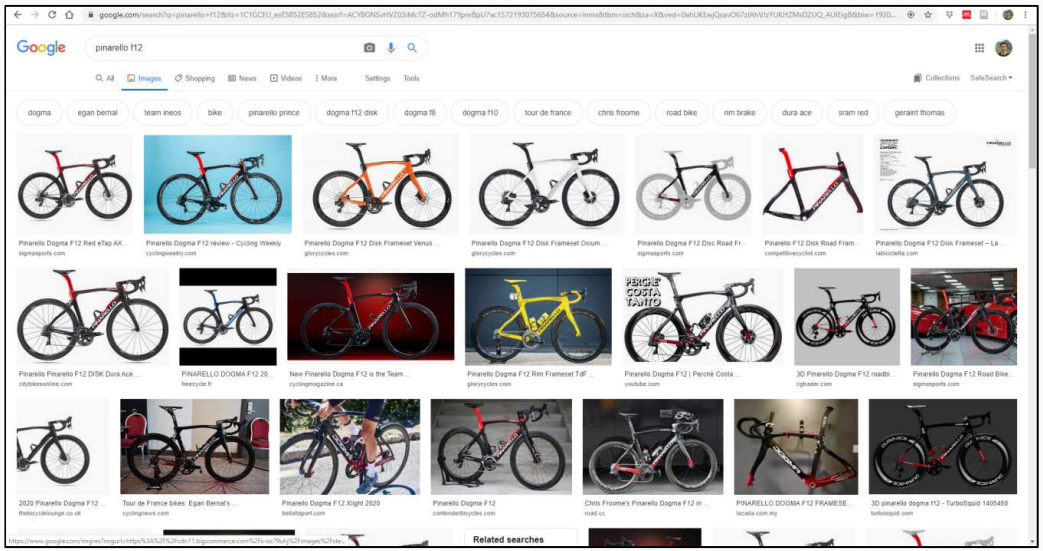
Estructura del tema

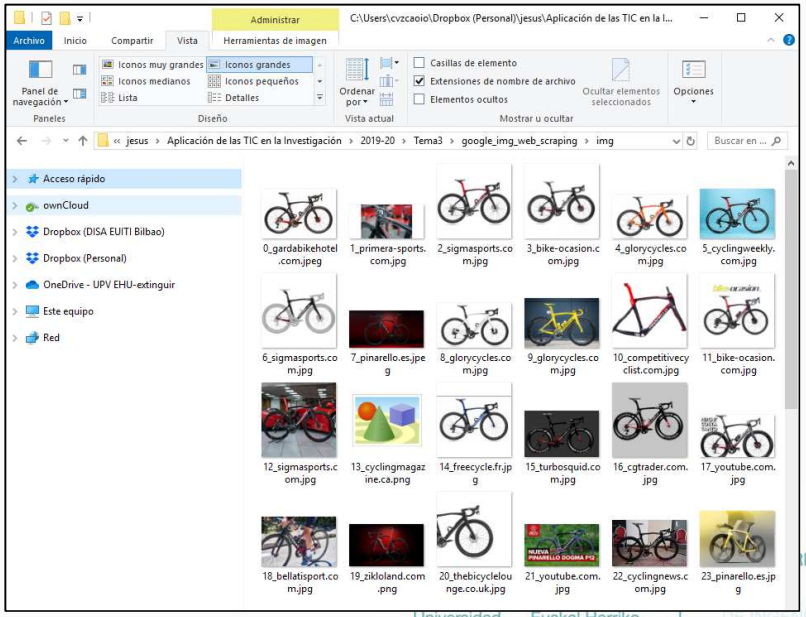
- Estudio de un caso: descargar imágenes de Google Images
- Contenido y estructura de una página HMTL
- Código JavaScript: página descargada vs página renderizada
- Proceso de web scraping:
  - Conocer la estructura de la página HTML para saber dónde están los enlaces a las imágenes.
  - ¿La página HTML descargada coincide con la renderizada?
    - Sí → Parsear HTML
    - No → Renderizar HTML y luego parsearlo
  - Almacenar imágenes

1

Estudio de un caso: descargar imágenes de Google Images

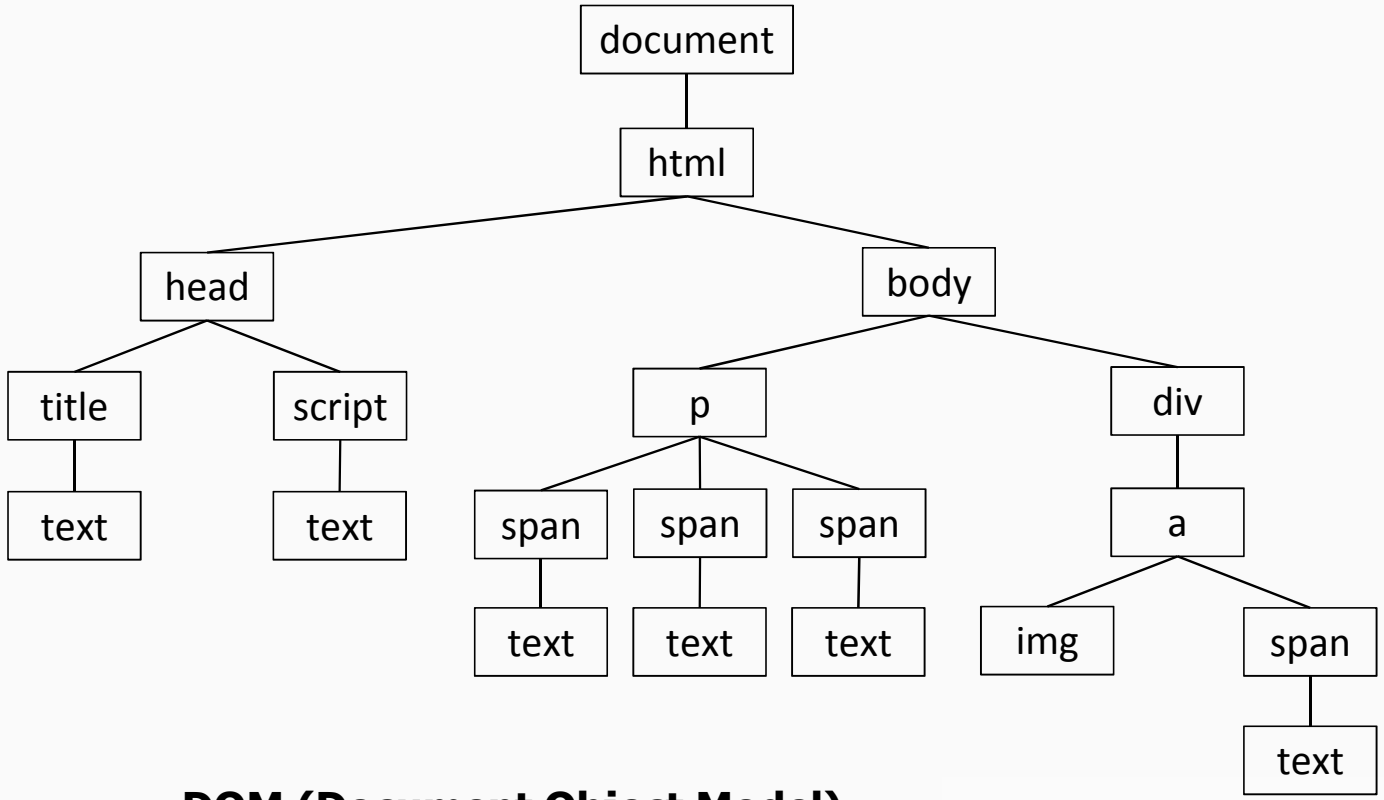
Se quiere programar una aplicación que descargue automáticamente las imágenes obtenidas como resultado de una búsqueda en *Google Images* y las almacene en una carpeta local.





2 Contenido y 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://...">
            <img src="http://..." />
            <span>bla</span>
         </a>
      </div>
   </body>
</html>
```



DOM (Document Object Model)

3

Código Javascript: 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

4 Conocer la estructura de la página HTML para saber dónde están los enlaces a las imágenes

- 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"

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

<html>
    <head>
        <title>jspVSjs</title>
    </head>
    <body>
        Server Date: Tue Mar 07 11:48:34 CET 2017
        <br/>
        <script src="/jspVSjs/js/data.js"></script>
    </body>
</html>
```

"View Source Chart"

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML>
  <head>
    <title>
    jspVSjs
    </title>
  </head>
  <body>
  Server Date: Tue Mar 07 11:48:34 CET 2017
  <br>
    <script src="/jspVSjs/js/data.js">
    </script>
  Client Date: Tue Mar 07 2017 11:48:34 GMT+0100 (Romance Standard Time)
  </body>
</html>
```

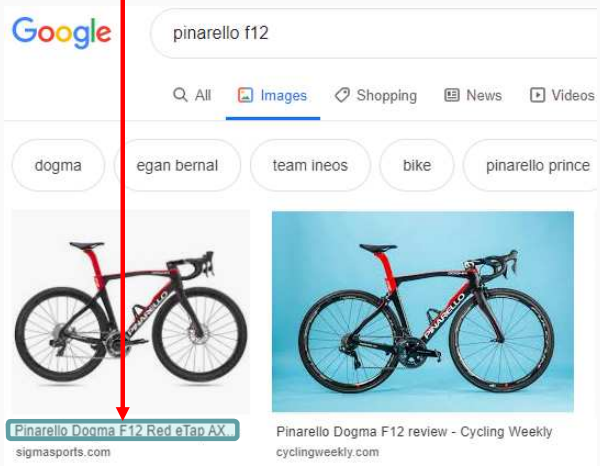
4 Conocer la estructura de la página HTML para saber dónde están los enlaces a las imágenes

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

- Ejecutar la opción "Ver código fuente" sobre la página de resultados → **¡Faltan 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 tx8vtf*".

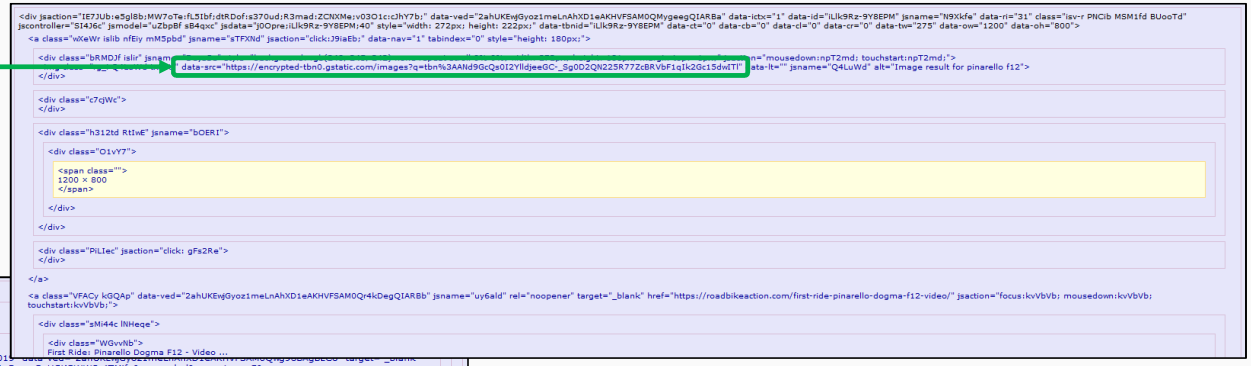



```
<div jsaction="IE7JUb:e5gl8b;MW7oTe:fL5Ibf;dtRD
class="isv-r PNCib MSM1fd BUooTd" jscontroller="S
cr="0" data-tw="221" data-ow="1021" data-oh="1
  <a class="wXeWr islib nfEiy mM5pbd" jsname="s
    <div class="bRMDJf islir" jsname="DeysSe" s
    <img class="rg_i Q4LuWd tx8vtf"
    iml="124175.57500000112">
    </div>
    <div class="c7cjWc">
    </div>
    <div class="h312td RtIwE" jsname="bOERI">
      <div class="O1vY7">
        <span class="">
        1021 × 1050
        </span>
      </div>
    </div>
    <div class="PiLIec" jsaction="click: gFs2Re">
    </div>
  </a>
```

4 Conocer la estructura de la página HTML para saber dónde están los enlaces a las imágenes

Analizando el estructura de los elementos “img” que contienen los registros de resultados, se observa que hay dos tipos: a) los que tienen la imagen embebida en base64 y b) los que sólo la enlazan

Imagen enlazada en atributo “data-src”



a) imagen embebida en base64

b) imagen enlazada

Imagen *inline* en atributo “src”

5 Renderizar HTML

- 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.

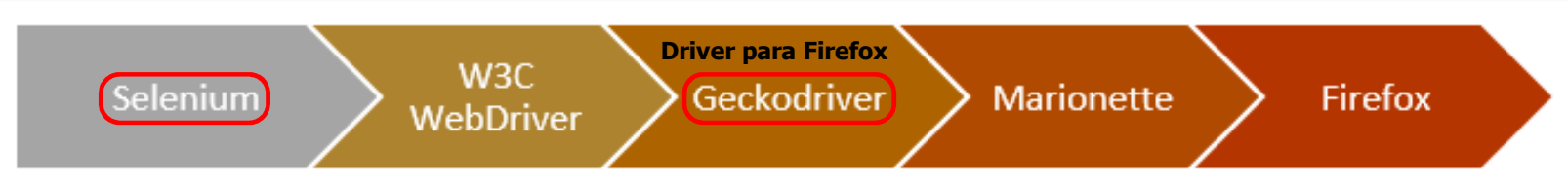
```
import requests

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)
```

- **¡Faltan 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.

5

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.




```python
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.tx8vtf")))
# obtener el código HTML
html = browser.page_source
# cerrar el navegador
browser.close()
```

6

Parsear el HTML

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 tx8vtf'})
for idx, each in enumerate(img_results):
    print idx

    src = ""
    if each.has_attr('src'):
        src = each['src']
    else:
        src = each['data-src']
```

7

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
# "document" 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 tx8vtf'})
for idx, each in enumerate(img_results):
    print idx

    src = ""
    if each.has_attr('src'):
        src = each['src']
    else:
        src = each['data-src']

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

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