Sistemas Web

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

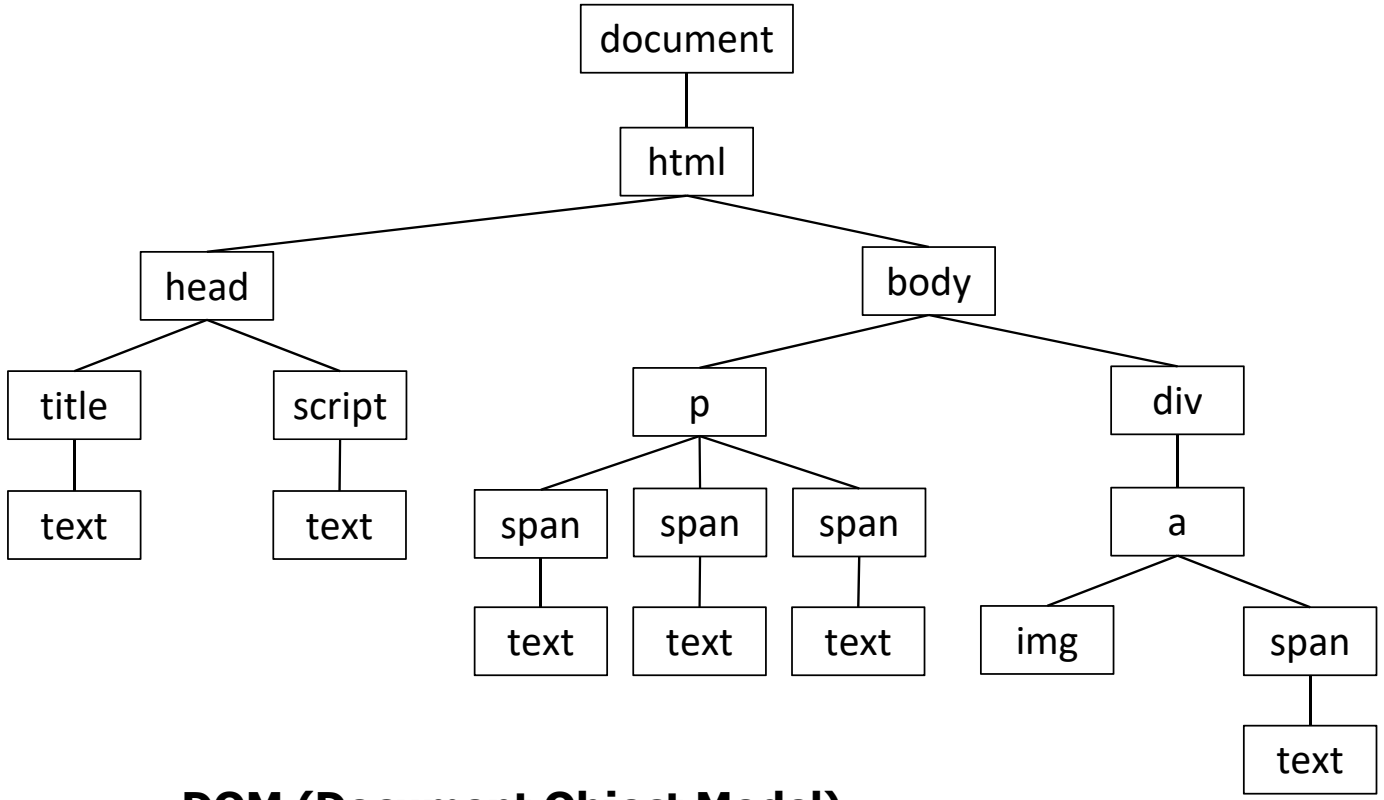
0 Estructura del tema

- Estructura de una página HTML
- Parsear una página HTML con BeautifulSoup
- Página descargada vs página renderizada
- *View Source Chart:* bookmarklet para ver la página renderizada
- *Selenium+Geckodriver*: automatizar navegador desde Python para renderizar HTML

- **Caso de Estudio:** 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+geckodriver*
  - ❑ Analizar estructura de la página HTML para saber dónde están los enlaces a las imágenes.
  - ❑ Parsear el HTML con BeautifulSoup
  - ❑ Almacenar imágenes:
    - Realizamos peticiones HTTP para descargar las imágenes enlazadas
    - Decodificamos las imágenes “inline” de base64 a binario

1 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>
```

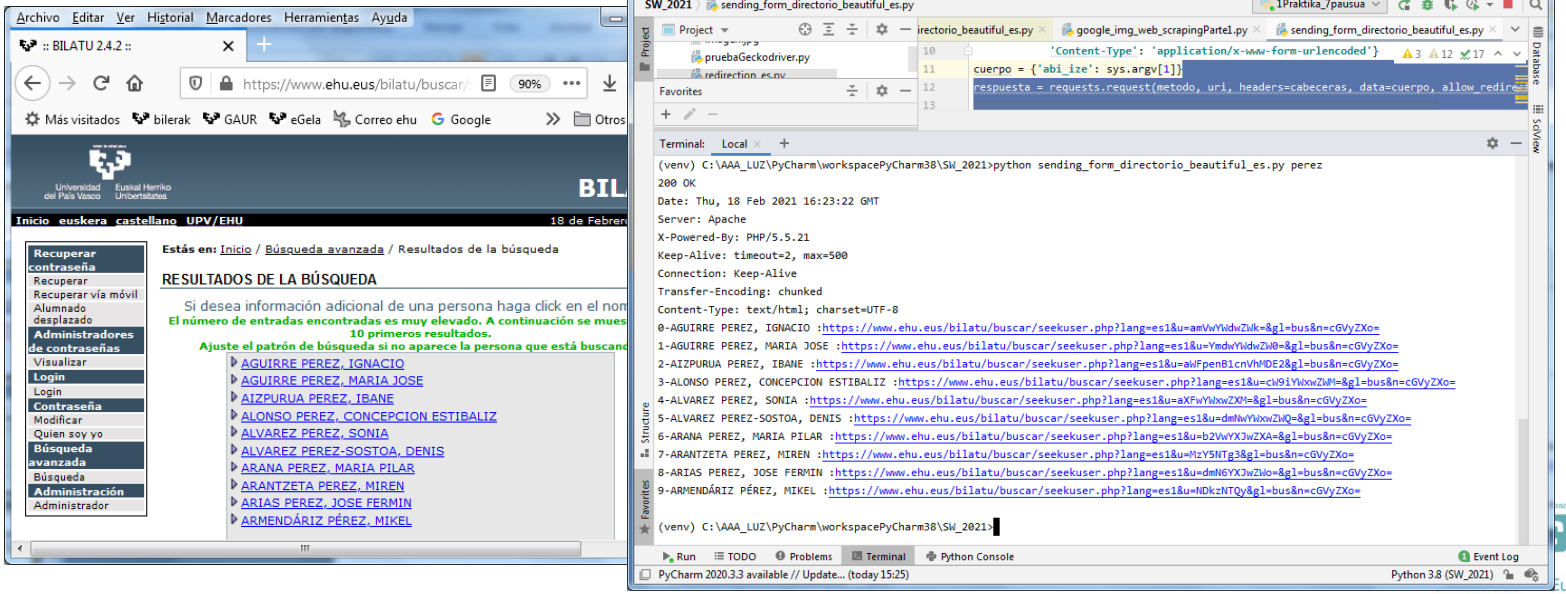


DOM (Document Object Model)

2 Parsear una página HTML

- Se utiliza la librería **Beautiful Soup** de Python para parsear HTML.
- Instalación de la libreria: pip install BeautifulSoup4
- Documentación: **https://www.crummy.com/software/BeautifulSoup/bs4/doc/**

- **Ejercicio:** Programar un cliente que permita realizar consultas en el directorio de la UPV/EHU para mostrar un resumen con el nombre, apellidos y enlace de los resultados.



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>
```

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

Crear una página HTML con este contenido

4 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>ERLAITZA</title>
</head>
<body>
    <ul id="zerrenda">
        <li>Server Date: Tue Feb 21 9:28:33 CET 2020</li>
    </ul>
    <br/> <!-- CRLF bezela -->
    <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("zerrenda").appendChild(li_element);
    </script>
</body>
</html><script>
```

"View Source Chart"

```
<html>
  <head>
    <title>
    ERLAITZA
    </title>
  </head>
  <body>
    <ul id="zerrenda">
      <li>
      Server Date: Tue Feb 21 9:28:33 CET 2020
      </li>
      <li>
      Client Date: Fri Feb 18 2022 15:50:09 GMT+0100 (Central European Standard Time)
      </li>
    </ul>
  <br>
    <!-- CRLF bezela -->
    <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("zerrenda").appendChild(li_element);
    </script>
    <script>
```

5 Selenium+Geckodriver:

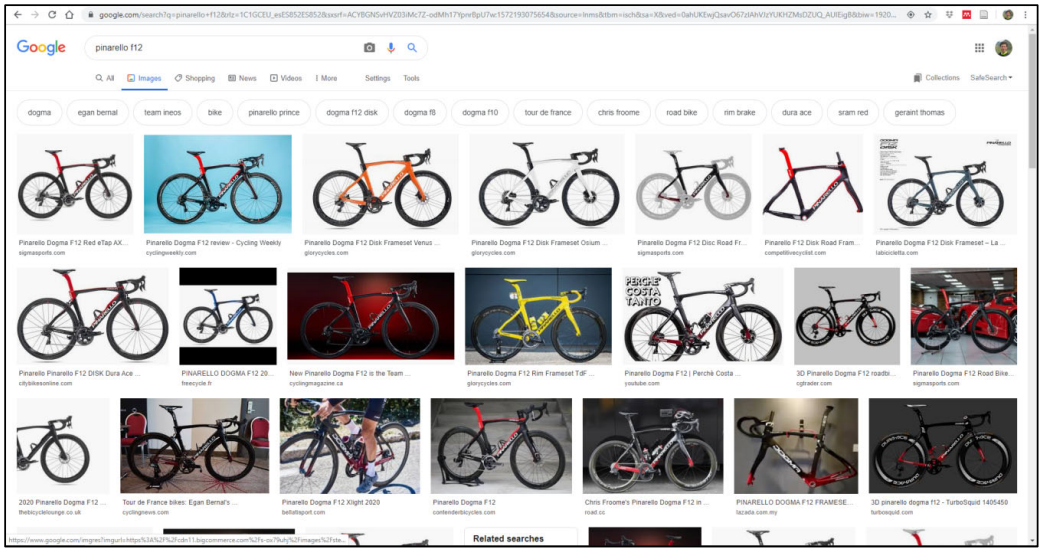
Automatizar navegador desde Python para renderizar HTML

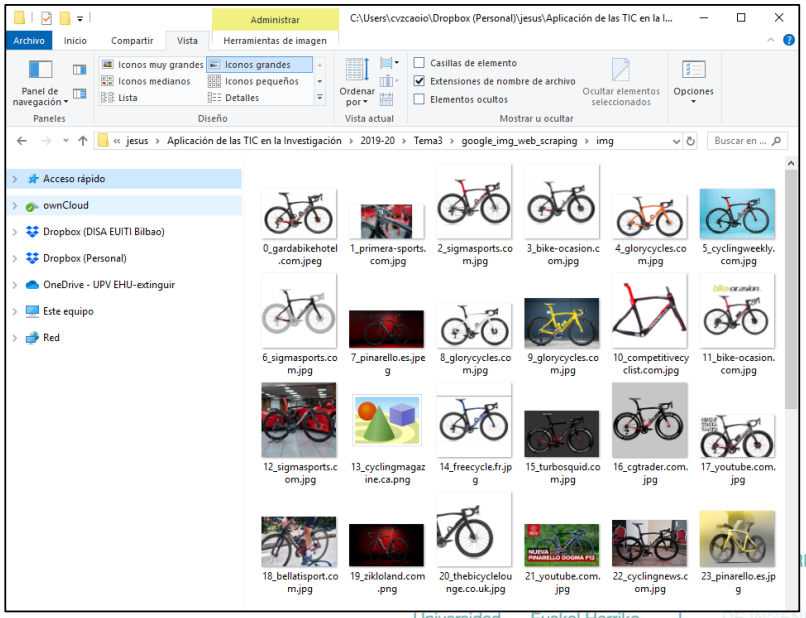
- ¿Cómo podemos hacer uso del navegador desde un programa escrito en Python?
  - Mediante Selenium, un herramienta para la automatización de tests de aplicaciones web.
    - Geckodriver es un drive que ofrece una interfaz para controlar el navegador


```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"

# abrir el navegador
browser = webdriver.Firefox()
# abrir la pagina
browser.get(uri)
# esperar hasta que se hayan renderizado los elementos que nos interesan (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 Caso de Estudio: Descargar imágenes de Google Images

Se pretende descargarse automáticamente las imágenes devueltas en una búsqueda en Google Images y desarrollar un programa que las almacene en una carpetas local.





6.1 ¿La página descargada coincide con la renderizada?

- En Firefox, lanzar una búsqueda en Google Images del término “pinarello F12”

  https://www.google.com/search?q=pinarello+f12&tbm=isch
- Revisa el código HTML de la página descargada utilizando la opción del navegador “View page source”:


```
</script><script nonce="9fsoPjkLj9C_3dT5WzoXkg">var AF_initDataKeys = ["ds:0","ds:1"]; var AF_dataServiceRequests = {'ds:0' : {id:'ewC93d',request:[null,2,null,null,[1676473650],null,nu
try{
_.Zd=function(a,b,c){if(!a.o)if(c instanceof Array)for(var d of c)_.Zd(a,b,d);else{d=(0,_.x)(a.F,a,b);const e=a.B+c;a.B++;b.setAttribute("data-eqid",e);a.D[e]=d;b&&b.addEventListener?b.
}catch(e){_._DumpException(e)}
try{
_.$d=function(){if(!_.l.addEventListener||!Object.defineProperty)return!1;var a=!1,b=Object.defineProperty({},"passive",{get:function(){a=!0}});try{_.l.addEventListener("test",()=>{},b)
}catch(e){_._DumpException(e)}
try{
var be=document.querySelector(".gb_M .gb_e"),ce=document.querySelector("#gb.gb_Qc");be&&!ce&&_.Zd(_.Od,be,"click");
}catch(e){_._DumpException(e)}
try{
_.Ph=function(a){if(a.A)return a.A;for(const b in a.j)if(a.j[b].tb()&&a.j[b].G())return a.j[b];return null};_.Qh=function(a,b){b&&_.Ph(a)&&b!=_.Ph(a)&&_.Ph(a).ta(!1);a.A=b};_.Rh=functio
}catch(e){_._DumpException(e)}
try{
var qj=document.querySelector(".gb_b .gb_e"),rj=document.querySelector("#gb.gb_Qc");qj&&!rj&&_.Zd(_.Od,qj,"click");
}catch(e){_._DumpException(e)}
})(this.gbar_);
// Google Inc.
</script></div></div></div></div></div></div><c-data id="i2"></c-data></c-wiz><c-data id="i1"></c-data></c-wiz><script aria-hidden="true" nonce="9fsoPjkLj9C_3dT5WzoXkg">window.wiz_progr
</script><script nonce="9fsoPjkLj9C_3dT5WzoXkg">_setImgSrcDynamic('0','data:image\/jpeg;base64,\/9j\/4AAQSkZJRgABAQAAAQABAAD\/2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCY
</script><script nonce="9fsoPjkLj9C_3dT5WzoXkg">_setImgSrc('24','data:image\/jpeg;base64,\/9j\/4AAQSkZJRgABAQAAAQABAAD\/2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYx3Jx8fL
c.initAft=function(){f=0;e=Array.prototype.slice.call(document.getElementsByTagName("img")).filter(function(a){return!!a.getAttribute("data-iml")});[].forEach.call(document.getElements
initAft()</script><script id="_ij" nonce="9fsoPjkLj9C_3dT5WzoXkg">window.IJ_values = [ false , false , false , false , true , false , 1.0 , 0.0 , false , true , true , false , false , f
try{
_.de=function(a,b){return 0<=_.kb(a,b)};_.ee=function(a){var b=typeof a;return"object"!=b?b:a?Array.isArray(a)?"array":b:"null"};_.fe=function(a){var b=_.ee(a);return"array"==b||"object
ke={cellpadding:"cellPadding",cellspacing:"cellSpacing",colspan:"colSpan",frameborder:"frameBorder",height:"height",maxlength:"maxLength",nonce:"nonce",role:"role",rowspan:"rowSpan",typ
ne=function(a,b,c){function d(h){h&&b.appendChild("string"===typeof h?a.createTextNode(h):h)}for(var e=2;e<c.length;e++){var f=c[e];if(!_.fe(f)||_.cb(f)&&0<f.nodeType)d(f);else{a:{if(fa
_.me=function(a,b){b=String(b);"application/xhtml+xml"===a.contentType&&(b=b.toLowerCase());return a.createElement(b)};_.qe=function(a){for(var b;b=a.firstChild;)a.removeChild(b)};_.re=
}catch(e){_._DumpException(e)}
try{
var Ie;_.Ge=function(a,b){if(void 0!==a.Ca||void 0!==a.j)throw Error("C");a.j=b;_.Fd(a)};_.He=class extends _.H{constructor(a){super(a)}};Ie=0;_.Je=function(a){return Object.prototype.h
}catch(e){_._DumpException(e)}
try{
_.vj=function(a,b,c){a.rel=c;-1!=c.toLowerCase().indexOf("stylesheet")?(a.href=_.Bc(b),(b=_.bd(a.ownerDocument&&a.ownerDocument.defaultView))&&a.setAttribute("nonce",b)):a.href=b instan
}catch(e){_._DumpException(e)}
try{
_.wj=function(a){const b=_.uc();a=b?b.createScriptURL(a):a;return new _.zc(a,_.yc)};/*

 SPDX-License-Identifier: Apache-2.0
*/
var xj;try{new URL("s://g"),xj=!0}catch(a){xj=!1}_.yj=xj;
}catch(e){_._DumpException(e)}
try{
_.zj=function(a){var b;let c;const d=null==(c=(b=(a.ownerDocument&&a.ownerDocument.defaultView||window).document).querySelector)?void 0:c.call(b,"script[nonce]");(b=d?d.nonce||d.getAttr
}catch(e){_._DumpException(e)}
try{
var Aj=function(a,b,c){_.Nd.log(46,{att:a,max:b,url:c})},Cj=function(a,b,c){_.Nd.log(47,{att:a,max:b,url:c});a<b?Bj(a+1,b):_.oc.log(Error("Z`"+a+"`"+b),{url:c})},Bj=function(a,b){if(Dj)
}catch(e){_._DumpException(e)}
})(this.gbar_);
// Google Inc.
```

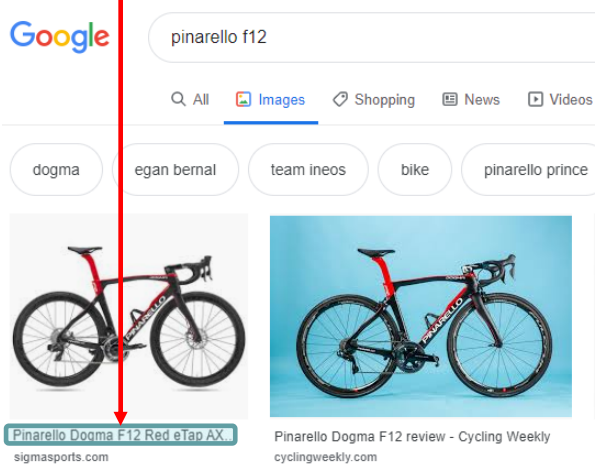

- El código contiene una pequeña estructura HTML básica y un gran cantidad de códigos JavaScript.
- Si se revisa el código con atención, se puede deducir que la función **_setImgSrc()** incluye las imágenes en el DOM del HTML.
- Por tanto, al no existir elementos < img> con imágenes directamente enlazadas en la página descargada, no se puede utilizar la biblioteca requests para alimentar BeautifulSoup.

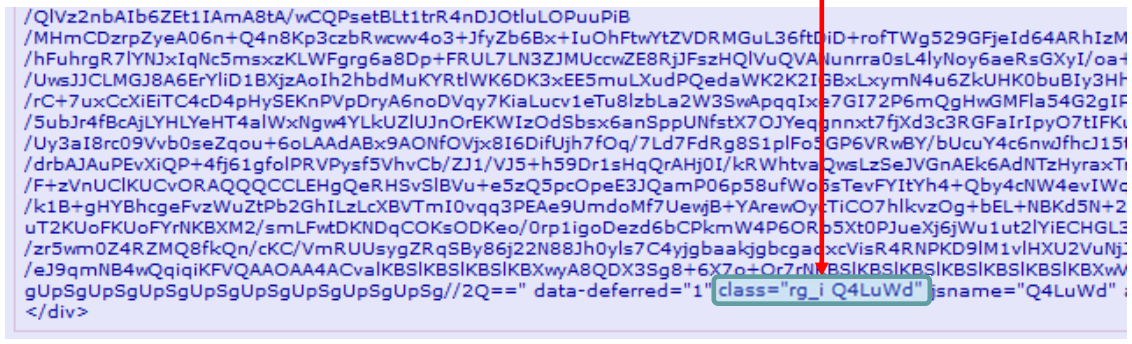
6.2 Analizar la estructura de la página HTML para saber dónde están las imágenes

- Por lo tanto, se deberá utilizar *selenium+geckodriver* para renderizar la hoja y poder parsear el html a DOM.
- Utilizando el bookmarklet "View Source Chart" buscaremos las imágenes dentro del código HTML

¿Dónde están las imágenes? Partimos de una cadena de texto que nos puede ayudar a ubicar una imagen, como por ejemplo, "F12 Red eTap".

Buscamos esa cadena en el código HTML y analizamos la estructura HTML que la incluye.
Las imágenes se encuentran en los elementos <img> con un atributo "class" que tiene como valor *"rg_i Q4LuWd"*

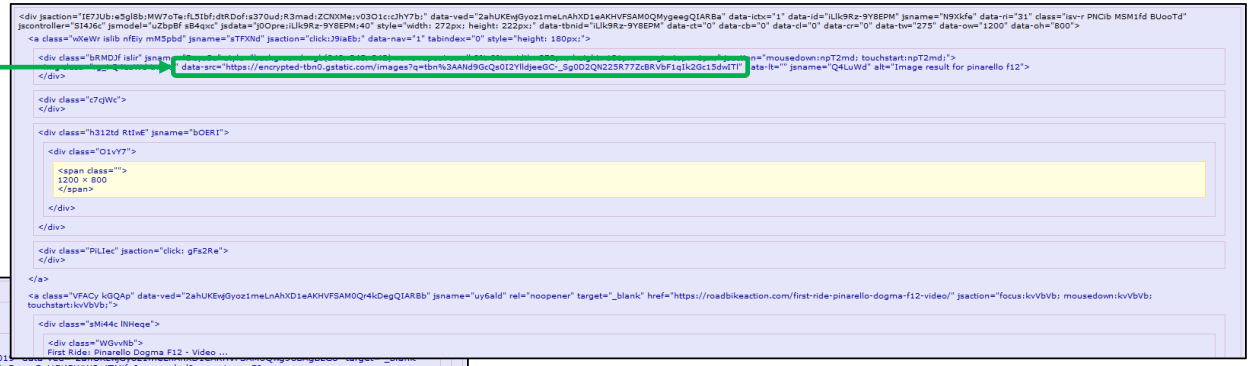




6.3 Analizar la estructura de la página HTML para saber dónde están las imágenes

Analizando en detalle la 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 la enlazan

Imagen enlazada en atributo “data-src”



a) imagen embebida en base64

Imagen *inline* en atributo “src”

b) imagen enlazada

6.4 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 elementoscuyo 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)
```

6.5 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("", ""))
    else:
        res = requests.get(src)
        img = res.content

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