DigitalHouse > Coding School

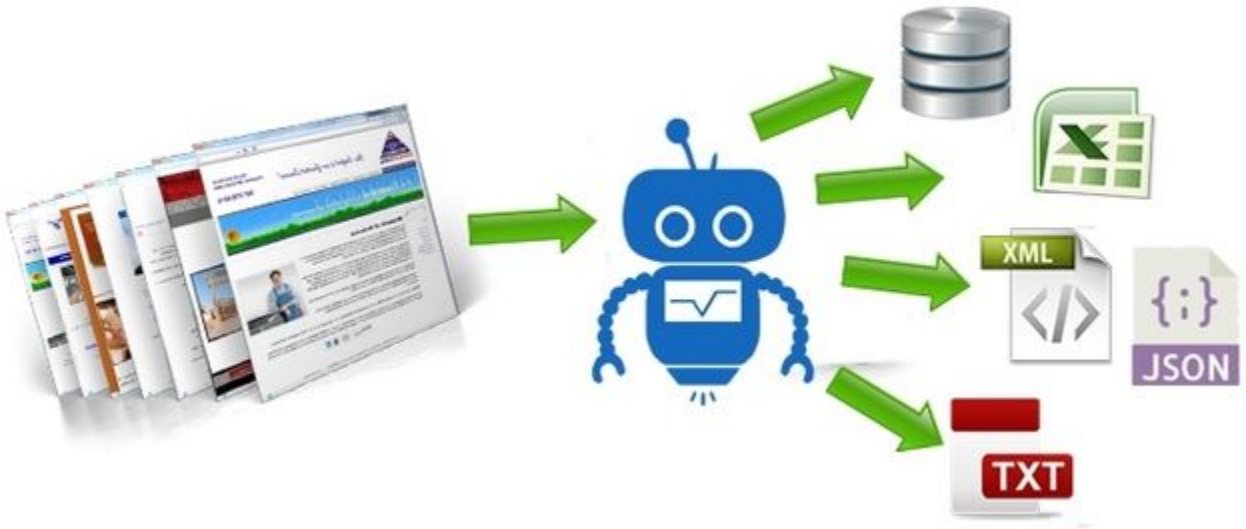
DATA SCIENCE

Web Scraping con Python

Agenda

1. **Describir los fundamentos del web scraping**
2. **Conocer cómo implementar web scraping usando Python**
   - **Requests**
   - **BeautifulSoup**

El **web scraping** es la extracción automática de información de la web mediante el uso de **scrapers** o **bots**.



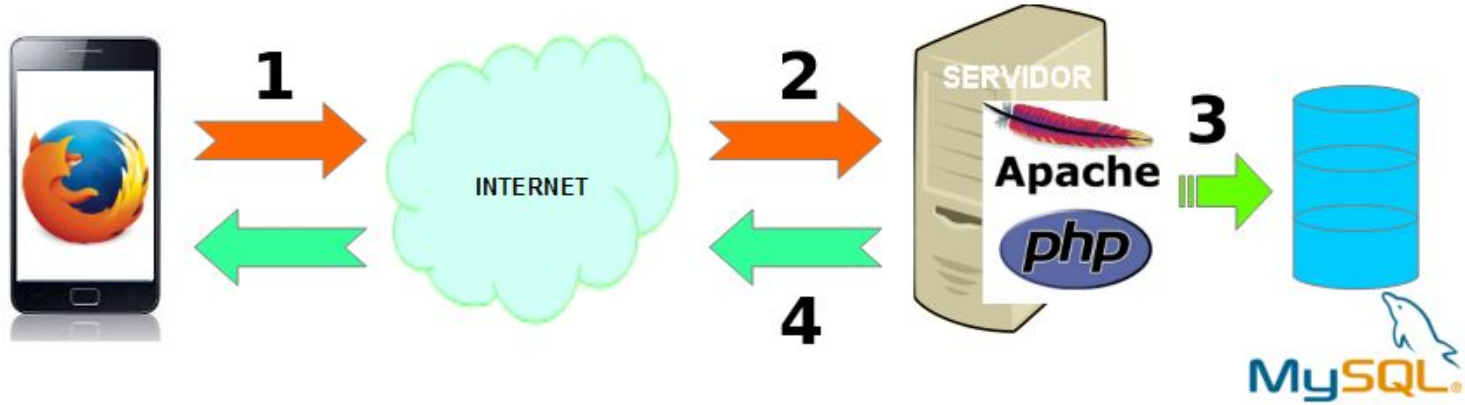
Receta de un web scraper:

1. Obtener datos **HTML** de un nombre de dominio
2. **Parsear** los datos en busca de cierta información objetivo
3. **Almacenar** esa información
4. Opcionalmente, pasar a otra página para **repetir** el proceso

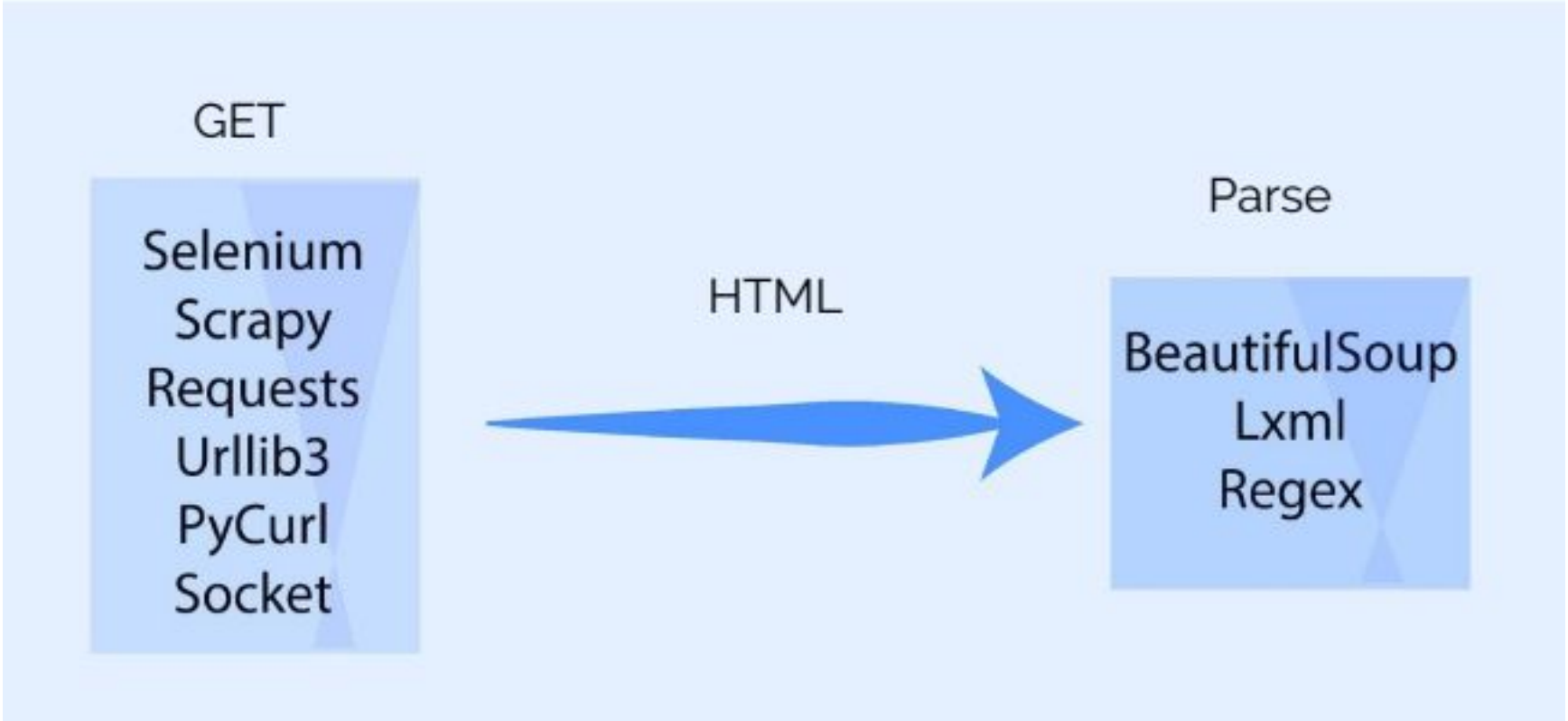
El **web crawling** es la práctica de navegación sistemática de la **World Wide Web**, generalmente con el propósito de indexación de páginas. Los bots que se encargan de la tarea de indexar y procesar contenido web se conocen como **spiders**.

HTTP y HTML

Las aplicaciones del lado del servidor son programas que corren en los web servers, procesan las solicitudes que el servidor recibe y generan respuestas que espera el cliente.



1. El cliente envía una solicitud, HTTP Request, al Servidor Web .
2. El Servidor Web interpreta la solicitud y ejecuta la Aplicación correspondiente.
3. La Aplicación del lado del servidor accede a la base de datos y genera la respuesta.
4. El Servidor Web envía la respuesta, HTTP Response, al cliente


GET
Selenium
Scrapy
Requests
Urllib3
PyCurl
Socket
HTML
Parse
BeautifulSoup
Lxml
Regex

Headers

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

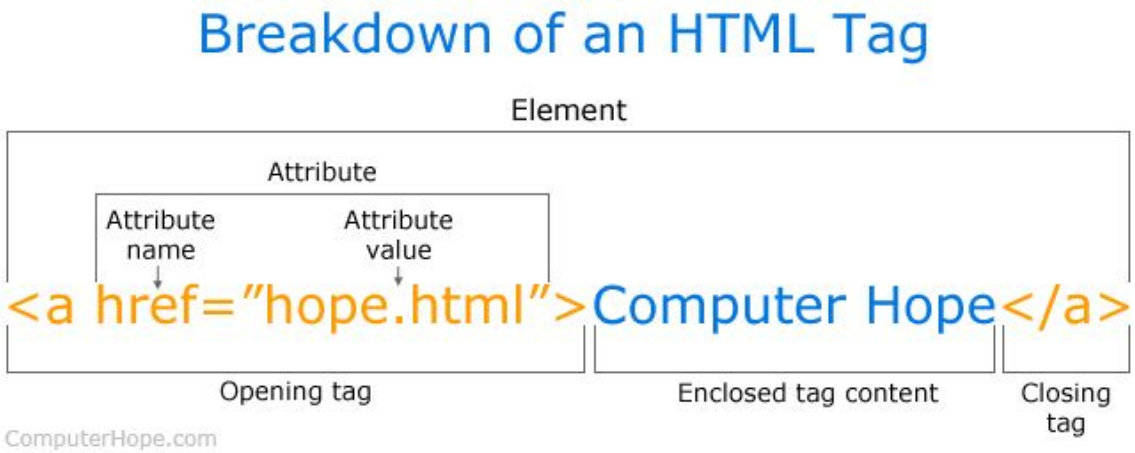
Content

```
<html>
   <body>

   <h1>Hello, World!</h1>

   </body>
</html>
```

– El **HyperText Markup Language (HTML)** es el lenguaje de marcado estándar en la web.

– Permite armar textos estructurados enriquecidos con archivos multimedia. Junto con **CSS** y **JavaScript** forman la tríada de tecnologías centrales de la web.

```
<!DOCTYPE html>
<html>
  <head>
     <!-- This is a head section -->
     <title> Title text goes here... </title>
  </head>
  <body>
     <!-- This is a body section -->
     <p> body text goes here...</p>
  </body>
</html>
```


Breakdown of an HTML Tag
Element
Attribute
Attribute name
Attribute value
Computer Hope
Opening tag
Enclosed tag content
Closing tag
ComputerHope.com

Requests y Beautiful Soup

Requests

- Requests es transparente: posee una interfaz simple de manejar, por tanto no tendremos que lidiar con las particularidades de **urllib3** (librería que funciona bajo Requests).

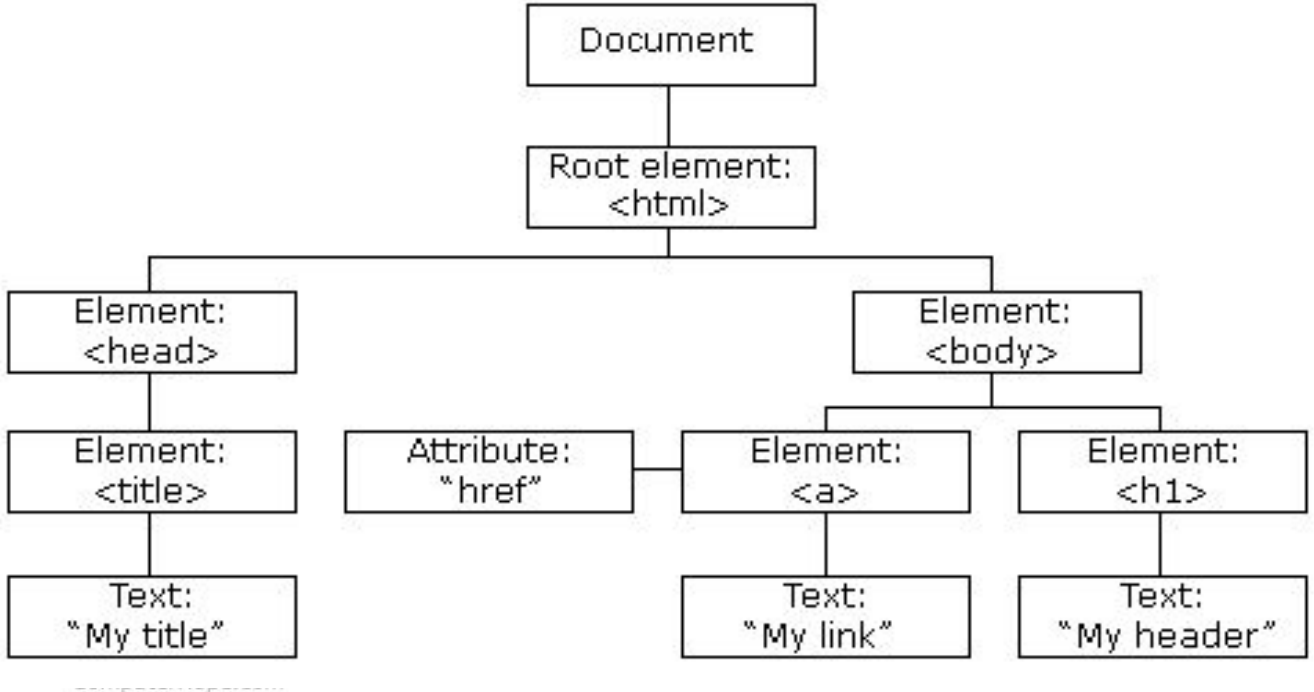
- Posee una **variedad de recursos** importantes para la interacción con servicios web, relacionados con el encoding, la compresión, persistencia de conexión, seguridad y otros protocolos de comunicación.

Beautiful Soup

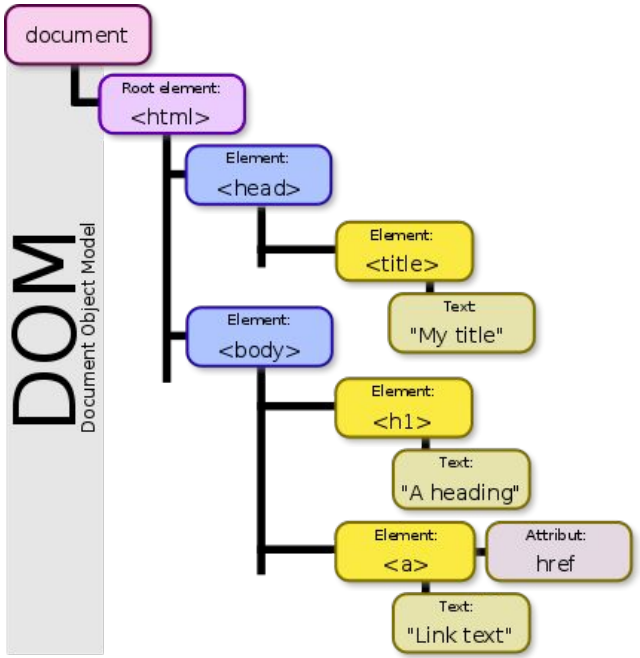
- Permite obtener la información contenida en formato HTML/XML y recorrer estas estructuras de forma ordenada.

- Transforma el contenido en objetos de Python a través de un conjunto de herramientas para manipular el contenido, incluso si éste no se encuentra bien formado.

- Los elementos que realizan la transformación desde el contenido hasta el objeto se conocen como **parsers**.


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
"href"
Element:
<a>
Element:
<h1>
Text:
"My title"
Text:
"My link"
Text:
"My header"

DOM es una convención para la manipulación de documentos en donde se define una **estructura jerárquica anidada tipo árbol.**



Beautiful Soup devuelve un objeto jerárquico de éstas características.

Una vez generada esta estructura de datos, es posible realizar búsquedas, extracciones y modificaciones sobre el árbol.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(response.content, "html.parser")
```

Algunas funciones útiles:

- `.prettify()`: Devuelve una representación conveniente del elemento
- `.attrs`: Devuelve los atributos y valores de atributo del elemento
- `.find_all(tag, atributos)`: Busca todos los elementos pertenecientes a un tag, con algún atributo (si se especifica, no es obligatorio definir atributos).
- `.contents`: Lista solo los subelementos del nivel jerárquico inferior inmediato.
- `.descendants`: Lista todos los subelementos de los niveles inferiores.

```
<tag atributo="valor"> Algun contenido </tag>
```

Anexo : Expresiones regulares

¿Qué son las expresiones regulares y para qué sirven?

Una expresión regular es una **secuencia de caracteres** que define un **patrón de búsqueda** de texto. Las regex constituyen lenguaje muy flexible que sirve para identificar y extraer información de un cuerpo de caracteres **no estructurado**.

Ejemplos:

- Identificar direcciones de correo electrónico (texto arroba texto punto com)
- Identificar DNI, CUIT, CUIL, etc. (dd.ddd.ddd, dd-dd.ddd.ddd-d)
- Identificar una fecha válida (dd/mm/yyyy)

Las **expresiones regulares** más fáciles son las **literales**, que matchean consigo mismas. Por ejemplo, la expresión regular **r“hola” sólo encuentra el texto “hola”**.

```
saludo = '¡Hola, Digital House!'
print(re.findall('hola',saludo,flags = re.IGNORECASE))
print(re.findall('digital house',saludo,flags = re.IGNORECASE))
```

```
['Hola']
['Digital House']
```

Los **metacaracteres especiales** son aquellos que no se encuentran a sí mismos, sino que indican que se debe *matchear* algo de forma no literal, o que afectan a otras partes de la regex repitiendo caracteres o cambiando su significado.

```
. ^ $ * + ? { } [ ] \ | ( )
```

Metacaracteres que definen conjuntos:

- **\d** o **[0-9]** Cualquier dígito del 0 al 9
- **\w** Cualquier caracter alfanumérico (A-Z, a-z, 0-9 y _)
- **\s** Cualquier espacio en blanco (espacio, tabulado, nueva línea, etc.)
- **.** (un punto) Cualquier caracter con excepción de nueva línea (\n)

Y sus complementos:

- **\D** Todo menos cualquier dígito
- **\W** Todo menos cualquier caracter alfanumérico
- **\S** Todo menos cualquier espacio en blanco

Otros:

- **[ ]** Definen un conjunto de caracteres, por ejemplo: [cdf340]
- **[A-Za-z0-9_]** Definen rangos

Metacaracteres cuantificadores:

- * Cero o más del elemento anterior
- **+** Uno o más del elemento anterior
- **{cant}** Define cantidad (\d{2} = dos dígitos)
- **{min,max}** Definen mínimos y máximos de repetición
- **?** Opcional: cero o uno del elemento anterior

Ejemplos:

- **\d+** Encuentra un número o más
- **.*** Encuentra cero o más de cualquier caracter
- **\w{2,6}** Encuentra un conjunto de caracteres alfanuméricos con una longitud que va de 2 a 6 caracteres

Otros:

- **( )** Definen un grupo de captura.
  - Convierte la “detección de patrones” en “extracción de patrones”
  - Con la captura puedo sacar el año de una fecha o el DNI de un CUIL
- **|** Operador "o"
- **** Escapa los caracteres especiales
  - Por ejemplo, si quiero matchear un punto literal usaría “\.”

Metacaracteres de posición:

- **^** Comienzo de un *string* (o negación dentro de un conjunto, [^])
- **$** Final de un *string*
- **\b** Límite de palabra

¿Qué patrón identifican para los siguientes *strings*?

ana_laura@hotmail.com
juan.perez@gmail.com
florgimenez@yahoo.com.ar

Documentación oficial:

https://docs.python.org/3/library/re.html

https://docs.python.org/3/howto/regex.html

Sitios web para practicar:

https://regexr.com/

https://regex101.com/

Conclusiones

- **Web scraping** nos permite obtener información no estructurada y normalmente poco accesible en un formato simple para un posterior análisis.

- El web scraping es de interés cuando se necesita acceder a información dinámica.

- Python nos facilita el acceso a la información de interés sin la necesidad de tener un conocimiento exhaustivo de HTML o XML, utilizando librerías de distinto nivel entre las cuales están Requests y Beautiful Soup.

Para profundizar...

Para persistencia de cookies:

requests.Session()

Cuando se necesita ejecutar JavaScript:

Selenium

Para mayor rendimiento temporal:

Lxml

Confirmación externa para peticiones:

Con docker:
docker run -p 80:80 kennethreitz/httpbin

En web:
http://httpbin.org/