

Captura de datos

Rafael Caballero Roldán

*Departamento de Sistemas
Informáticos y Computación*

UCM

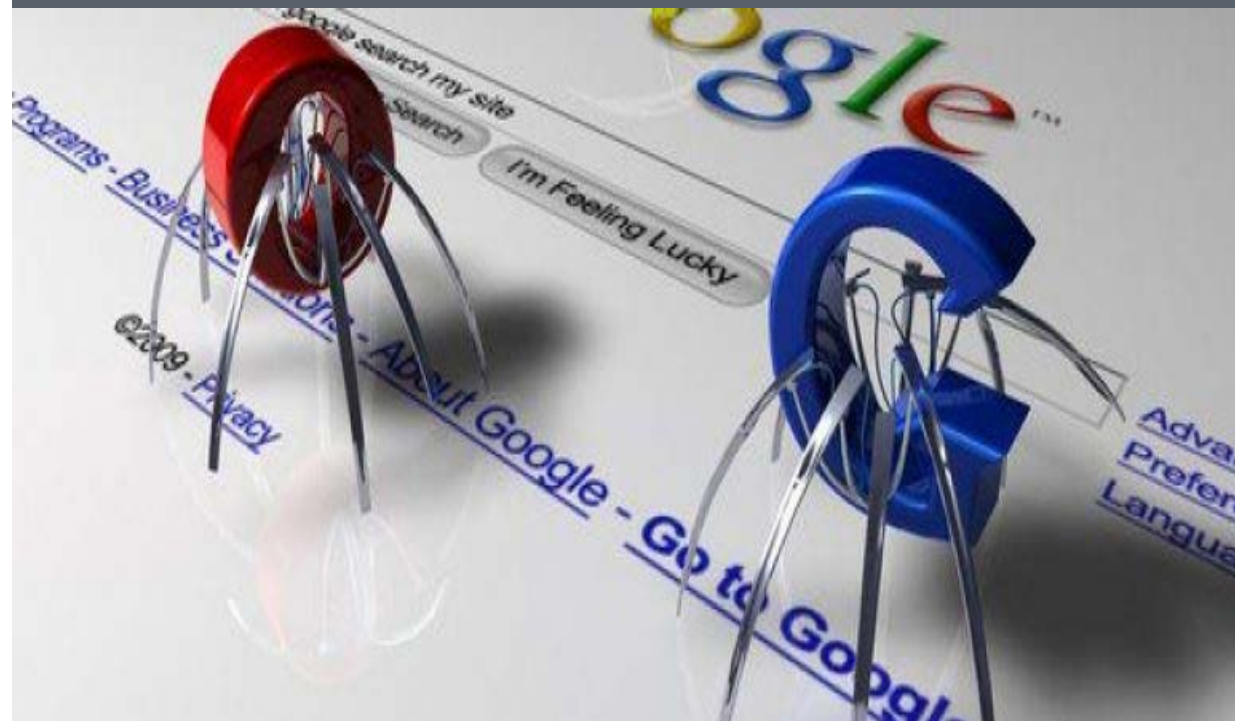
Web Scraping

Introducción

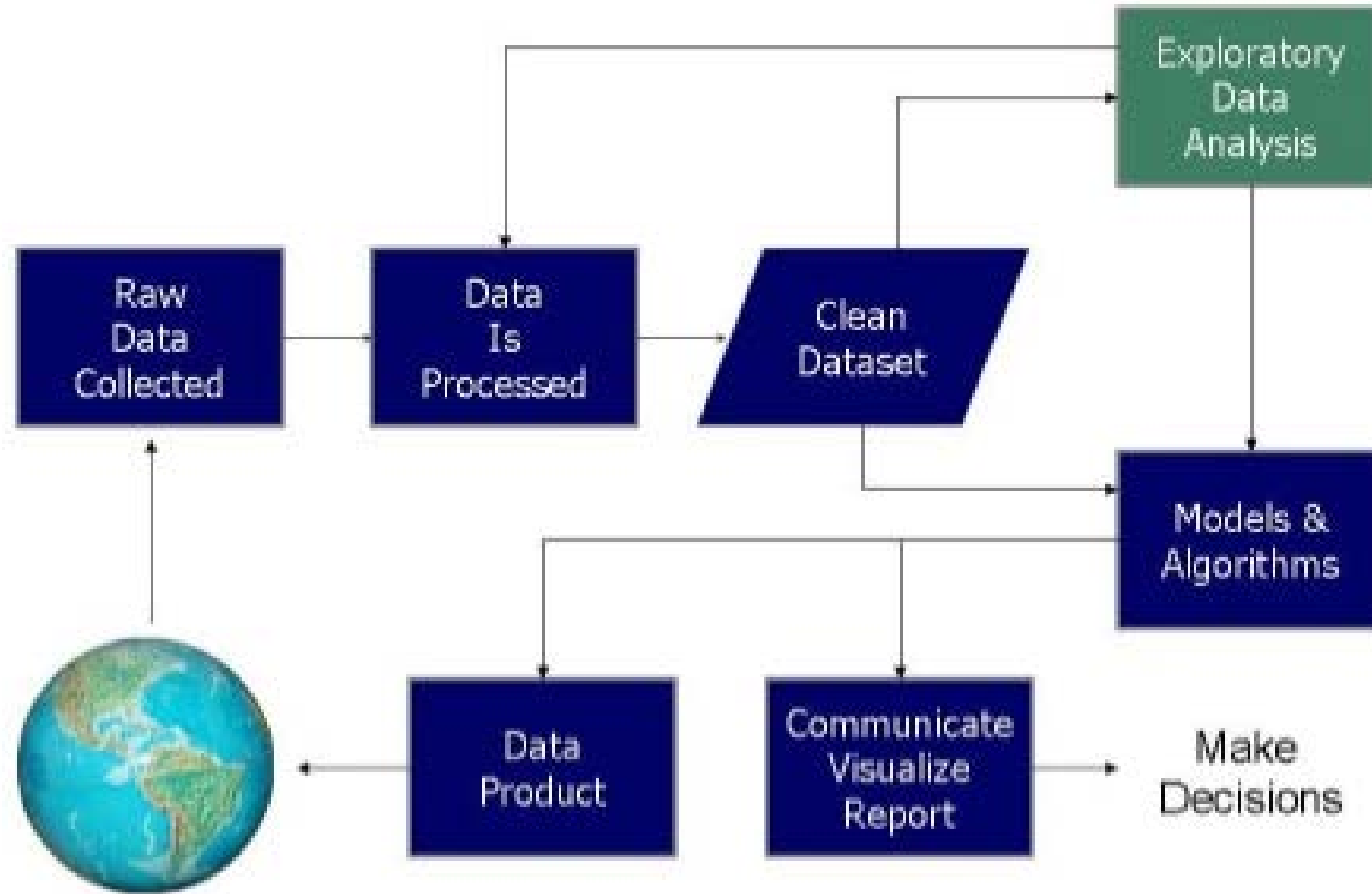
XPath

Selenium

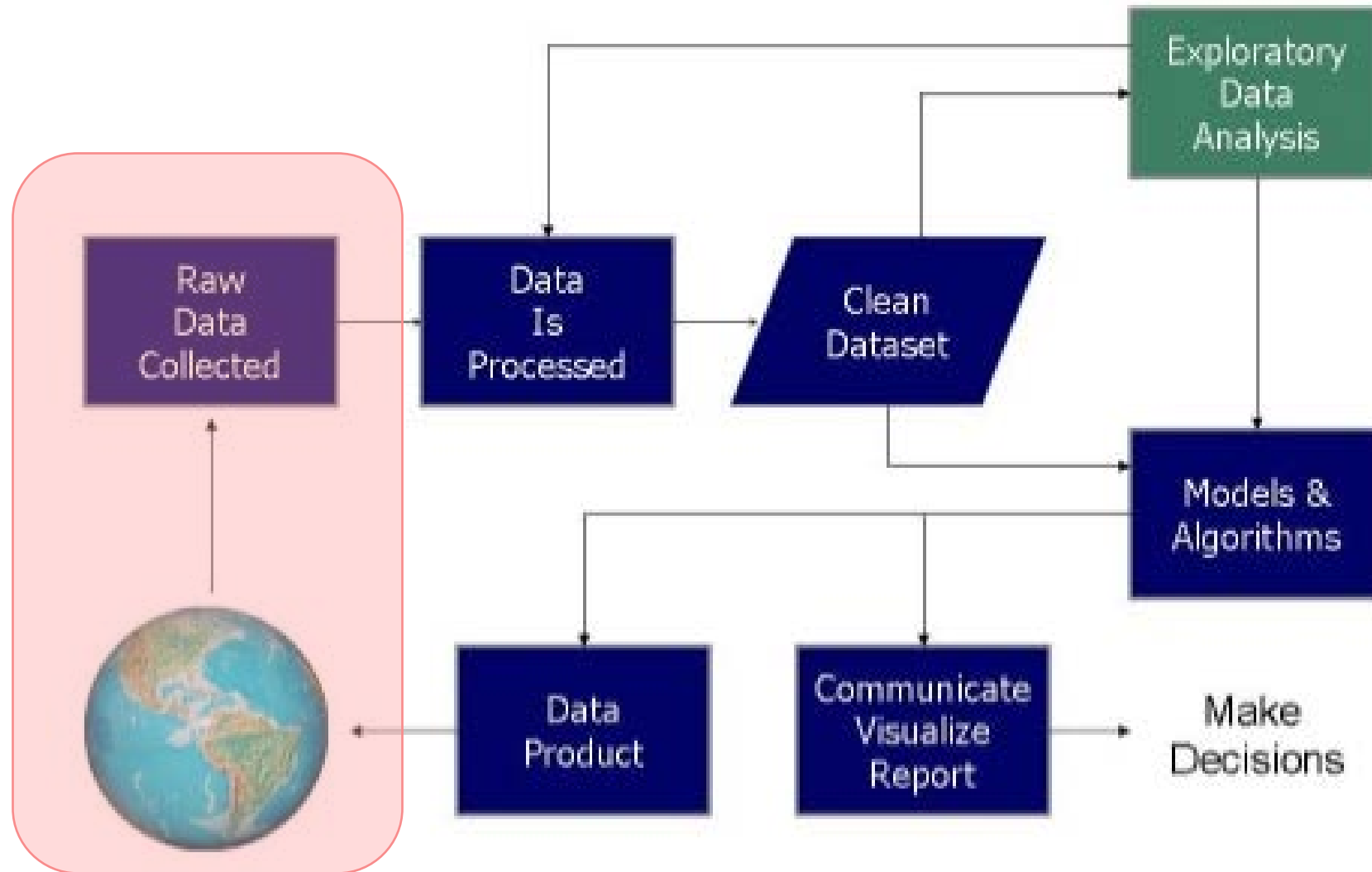
Ejemplo



Ciclo de análisis de datos



Ciclo de análisis de datos



Formas de acceso

- **Ficheros** accesibles a través de su propia URI → descarga directa (librería requests en Python)
- **APIs**: suelen exigir identificación y depende de cada caso
- **Web Scraping**: los datos están embebidos en una página, pensados para “humanos” y queremos extraerlos desde allí

Web scraping vs Web crawling

- **Web scraping** es básicamente el proceso de coger datos de una o varias webs previamente seleccionadas
- **Web crawling**: se supone que se consulta una gran cantidad de páginas, a menudo siguiendo links entre una y otra, buscando una determinada información. Google es el ejemplo de crawler por excelencia

Web scraping

- Es una fuente de datos de gran utilidad
- Muy habitual en IoT: páginas web que muestran datos obtenidos de sensores medioambientales, meteorológicos, etc.
- A menudo **no es fácil** “capturar” estos datos de forma automática, para hacerlo hay que examinar el código en el que está escrita la página web: **HTML**

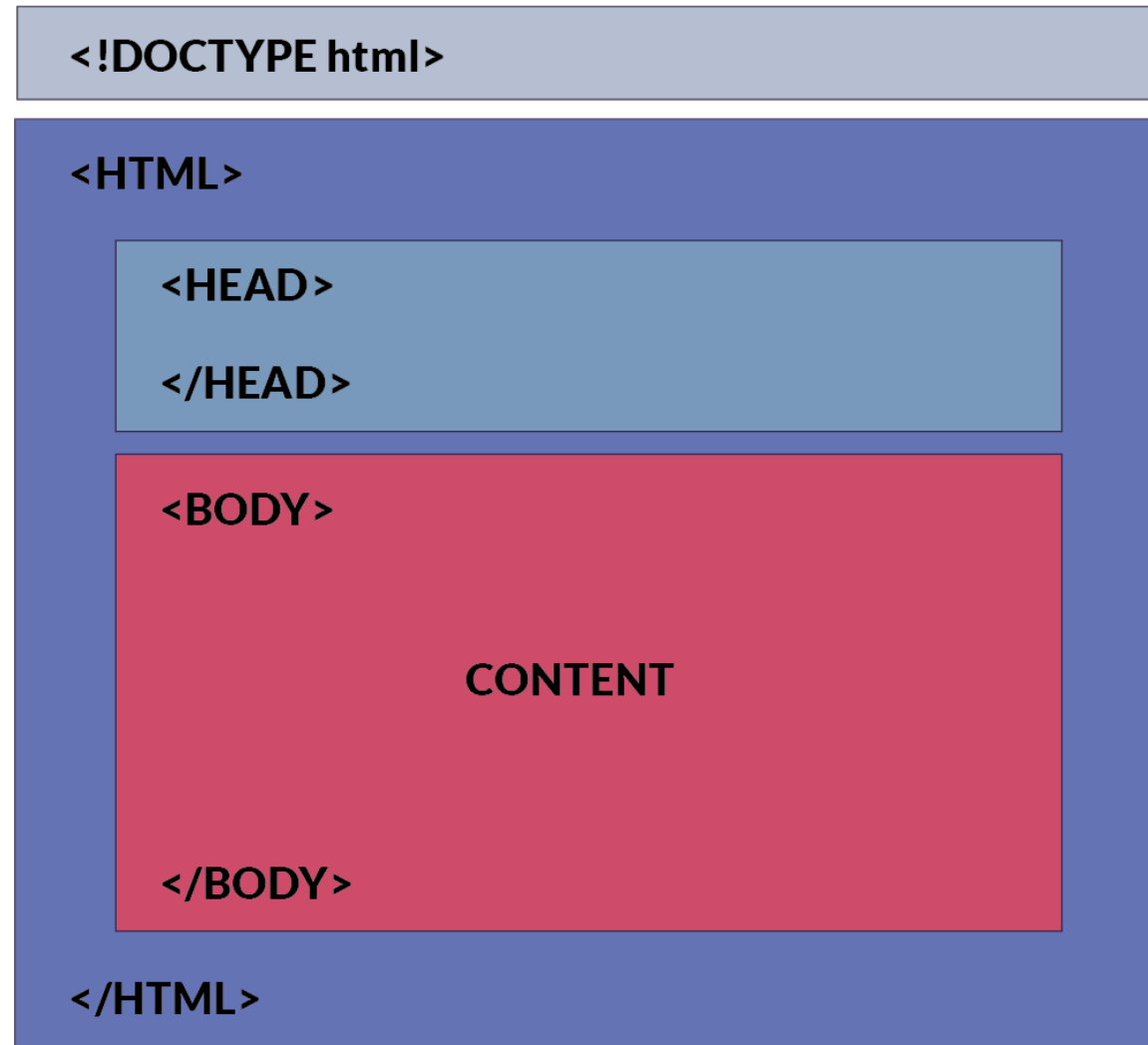
Tipos de páginas para Scraping

(en cada caso se indica la **biblioteca** Python recomendada)

1. **Estáticas**: al poner la URL en el navegador obtenemos la información como parte de la página obtenida. A su vez puede que los datos
 - a) Sean parte de la página → **BeautifulSoup**
 - b) Estén en un fichero que hay que analizar (XML, PDF, csv, JSON) -> **requests**
2. **Dinámicas**: la página incluye un formulario que se debe rellenar para llegar a la página con los datos. A su vez pueden distinguirse:
 - a) A través de URL: los datos del formulario generan una nueva URL que incluye los datos del formulario (tipo 1) → **requests**
 - b) El formulario simplemente genera una nueva página → **Selenium**

Aparte, hay que tener en cuenta características específicas de páginas concretas (marcos, imágenes dinámicas, peticiones, etc.)

Páginas HTML: estructura básica



Páginas HTML + CSS

HTML Markup

```
<html>
<body>
  <h1 class="blue">Hello World!</h1>
  <h1>I'm in the middle...</h1>
  <h1 class="blue">Goodbye World!</h1>
</body>
</html>
```

+

CSS

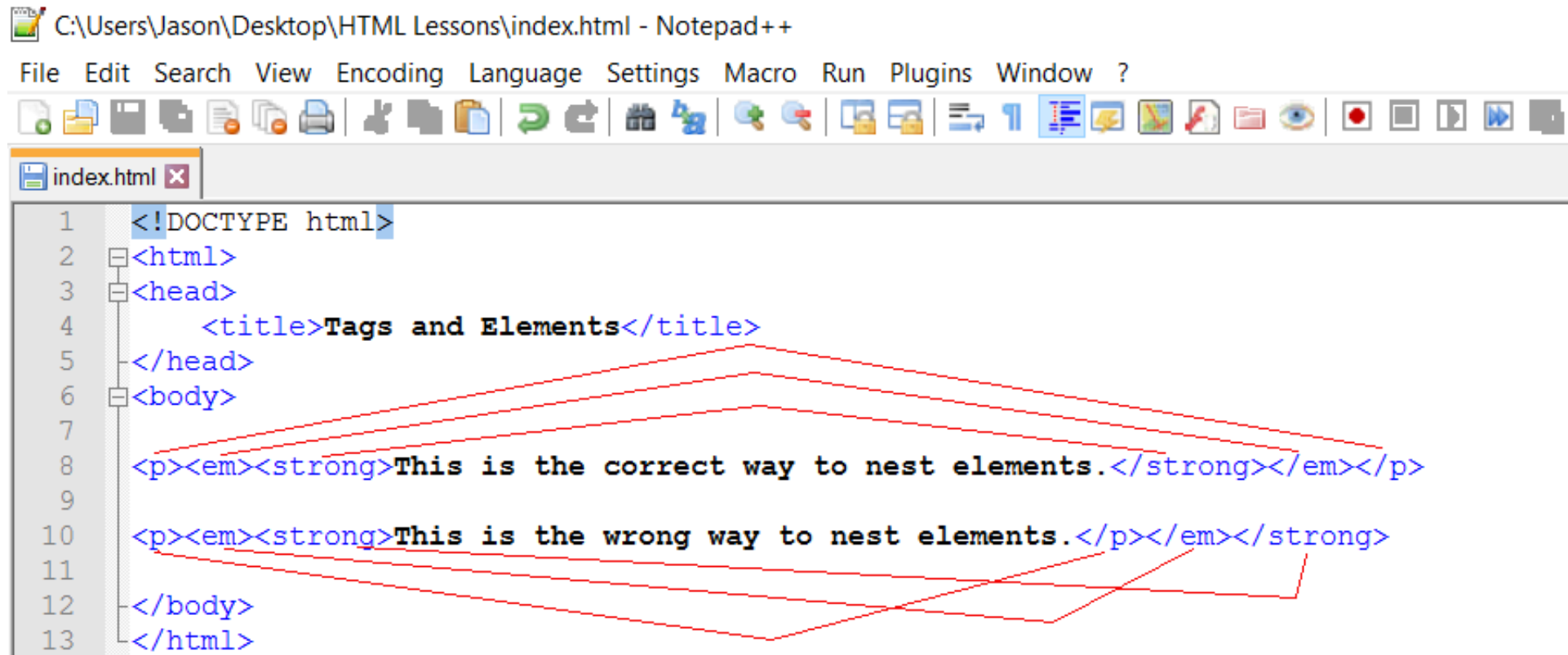
```
h1 {
  font-size: 16pt;
  font-style: italic;
  text-align: center;
  color: #FF0000;
}

.blue {
  color: #003366;
}
```

Resulting Page Render



HTML: etiquetas anidadas



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Tags and Elements</title>
5 </head>
6 <body>
7
8     <p><em><strong>This is the correct way to nest elements.</strong></em></p>
9
10    <p><em><strong>This is the wrong way to nest elements.</p></em></strong>
11
12 </body>
13 </html>
```

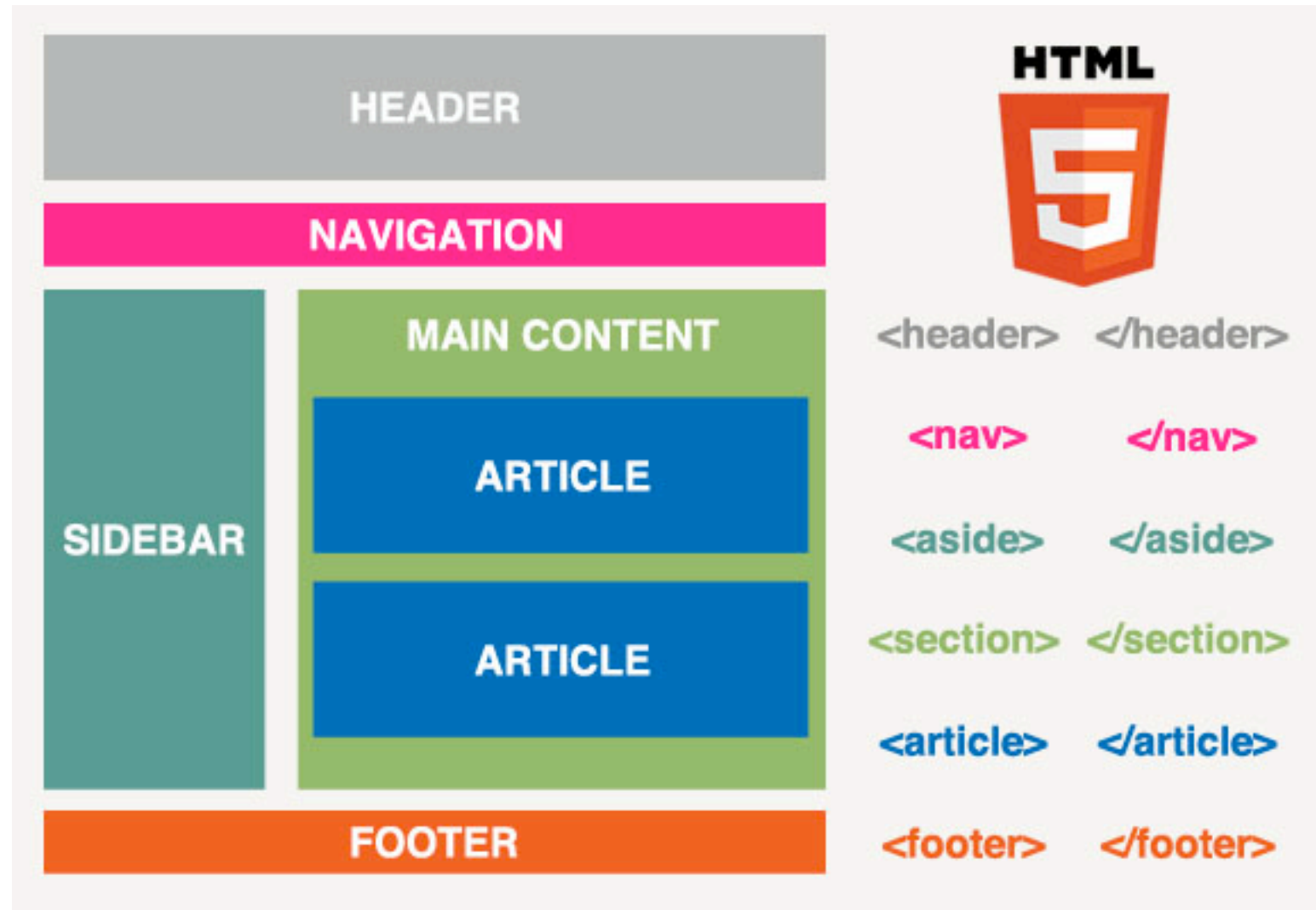
The screenshot shows a Notepad++ window with the file 'index.html' open. The code is as follows:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Tags and Elements</title>
5 </head>
6 <body>
7
8     <p><em><strong>This is the correct way to nest elements.</strong></em></p>
9
10    <p><em><strong>This is the wrong way to nest elements.</p></em></strong>
11
12 </body>
13 </html>
```

Red lines are drawn on the code to illustrate the nesting structure:

- Line 8: A red line connects the opening `<p>` tag to the closing `</p>` tag.
- Line 10: A red line connects the opening `<p>` tag to the closing `</p>` tag.
- Line 8: A red line connects the opening `` tag to the closing `` tag.
- Line 10: A red line connects the opening `` tag to the closing `` tag.
- Line 8: A red line connects the opening `` tag to the closing `` tag.
- Line 10: A red line connects the opening `` tag to the closing `` tag.

Páginas web: punto de vista de diseño



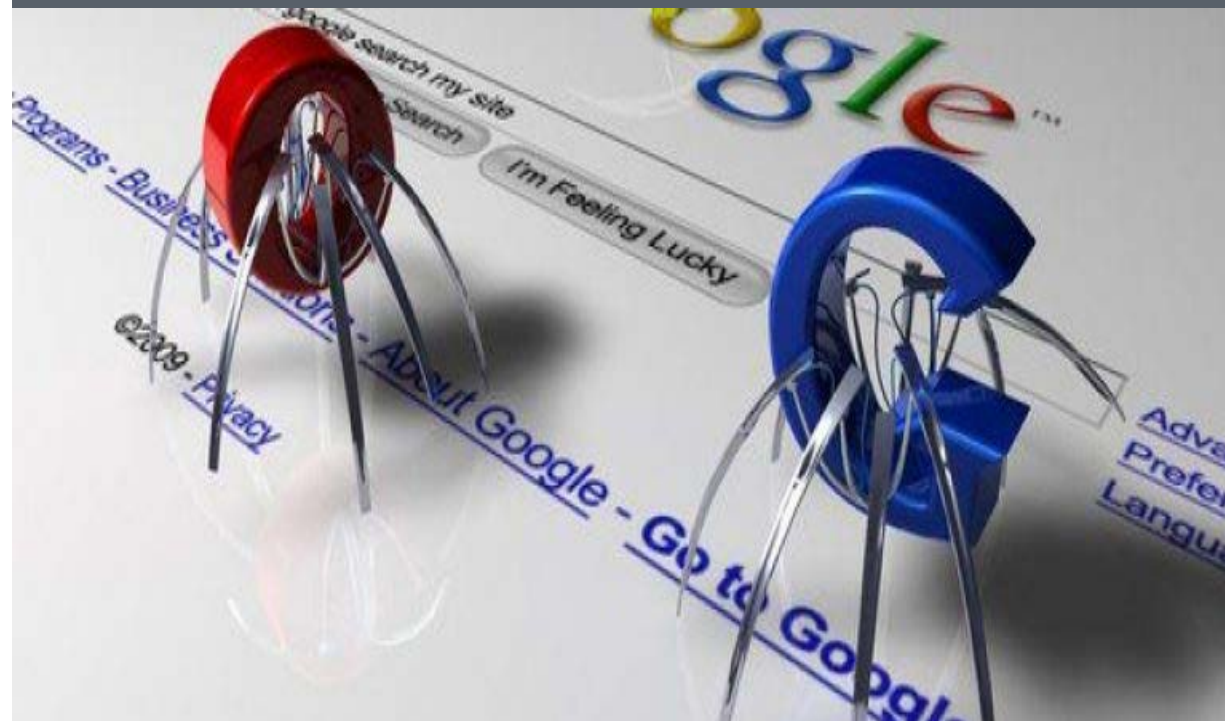
Web Scraping

Introducción

XPath

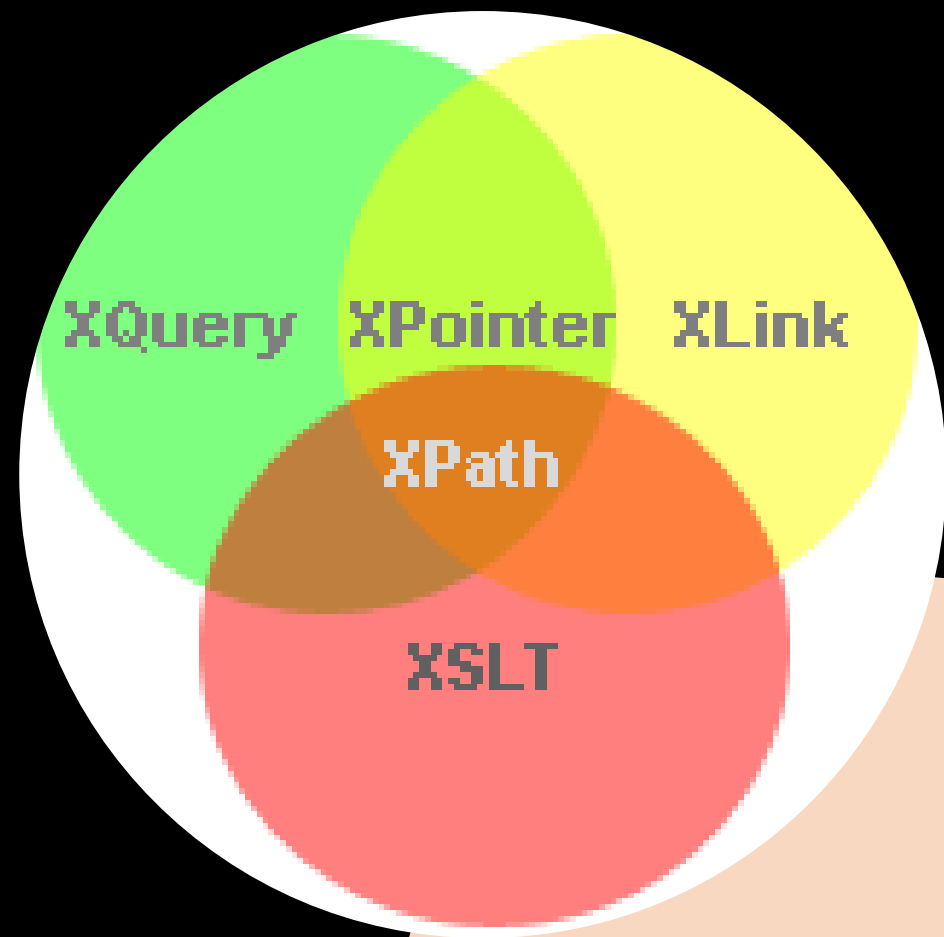
Selenium

Ejemplo



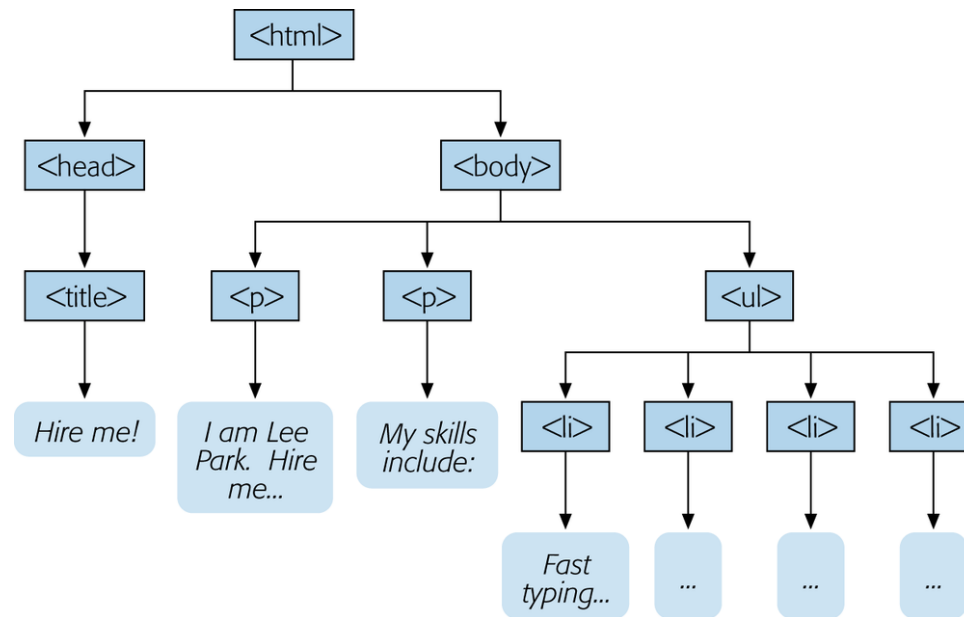
Qué es XPath

- Un lenguaje sencillo para recorrer y seleccionar elementos en documentos XML/HTML
- Integrado en Selenium, de todas formas solo lo usaremos si no hay otras posibilidades más sencillas disponibles
- Similar a los PATH que se usan en los comandos “cd” de LINUX/Windows
- Definido como estándar W3C
- Las expresiones XPath se utilizan en otros lenguajes más complejos



XPath: motivación

La idea detrás de XPath es ver la estructura de la página como un árbol



El lenguaje especifica cómo moverse por el árbol de arriba hacia abajo

Xpath: documento XML ejemplo

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

  <book>
    <title lang="en">Harry Potter</title>
    <price>29.99</price>
  </book>

  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>

</bookstore>
```


Expresiones XPath

Expression	Description
<i>nodename</i>	Selects all nodes with the name " <i>nodename</i> "
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Expresiones XPath: Ejemplo

bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore Note: If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//*[contains(@lang, 'eng')]	Selects nodes with attribute lang and value 'eng'

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```

Web Scraping

Introducción

XPath

Selenium

Ejemplo



Selenium

- Una **librería** para navegar e interactuar con páginas WEB
- Muy útil para **hacer tests de regresión** de servicios WEB
- Se pueden cargar URLs y navegarlas
- Además permite introducir **datos en formularios**:
 - “Teclear texto” automáticamente
 - Seleccionar elementos de listas
 - Pulsar botones

Selenium: abrir el navegador

- Necesitaremos tener un fichero externo que se encarga del enlace entre Python y el browser
- En el caso de Chrome “chromedriver.exe” (ver ejemplo)
- También se suelen configurar opciones para abrir el navegador y fijar alguna variable de entorno

```
driver = webdriver.Chrome(executable_path=chromedriver,  
                           options=chrome_options)
```

- La variable “driver” es el acceso al navegador

Selenium: carga de páginas

`driver.get(url)` # url la página a cargar

Ojo porque puede tardar; la solución puede ser esperar hasta que cierto elemento que conozcamos aparezca. Ejemplo:

```
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
...
timeout = 5
try:
    element_present = EC.presence_of_element_located((By.ID, 'ele'))
    WebDriverWait(driver, timeout).until(element_present)
except TimeoutException:
    print "Timed out waiting for page to load"
```

Selenium: localización de elementos

`find_element_by_XX`

Returns the first matching web element

Throws **NoSuchElementException** if the element is not found

find_element_by_id

find_element_by_name

find_element_by_xpath

find_element_by_link_text

find_element_by_partial_link_text

find_element_by_tag_name

find_element_by_class_name

find_element_by_css_selector

`find_elements_by`

Returns a **list** of multiple matching web elements

Returns an empty list if no matching element is found

find_elements_by_name

find_elements_by_xpath

find_elements_by_link_text

find_elements_by_partial_link_text

find_elements_by_tag_name

find_elements_by_class_name

find_elements_by_css_selector

Selenium: interacción

Una vez localizado el elemento, podemos interactuar con él

- Si es una caja de texto podemos meter texto

```
element.clear()  
element.send_keys("Bertoldo")  
element.send_keys(" and some", Keys.ARROW_DOWN)
```

- Si es un botón podemos pulsarlo

```
coord.click()  
coord.send_keys(Keys.SPACE) # alternativa
```

- [Muchas más posibilidades](#): selección de listas, cambiar de ventanas, cookies, etc.

Selenium y más



- Disponible para Java, Python y otros lenguajes
<https://github.com/corywalker/selenium-crawler>
- En el caso de Java basta con añadir al proyecto el .jar con la librería
- Un buen tutorial de web scraping en Python con Selenium:
<http://www.discoversdk.com/blog/web-scraping-with-selenium>
- Si solo se quiere analizar una página con la que no se quiere interactuar, se recomienda [BeautifulSoup](#): no hace falta el driver y es mucho más sencillo
- Si se quiere añadir interacciones sin tener que abrir un navegador se puede emplear [MechanicalSoup](#)

¡Gracias!

