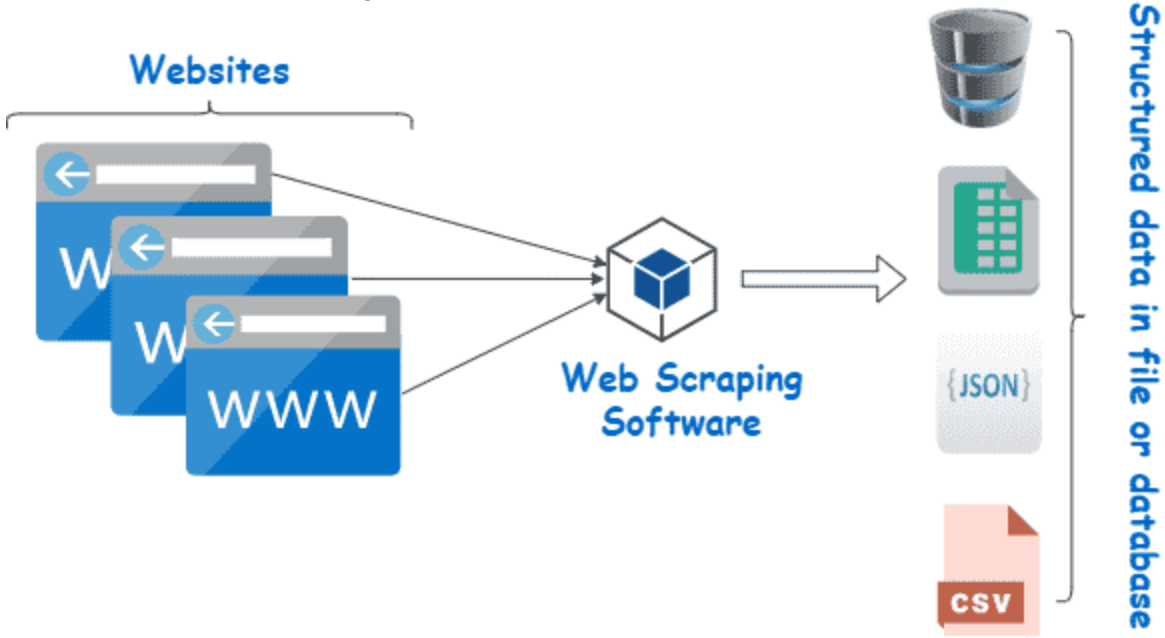
Data-Scraping con Python

Jonnatan Arias Garcia

¿Web Scraping?

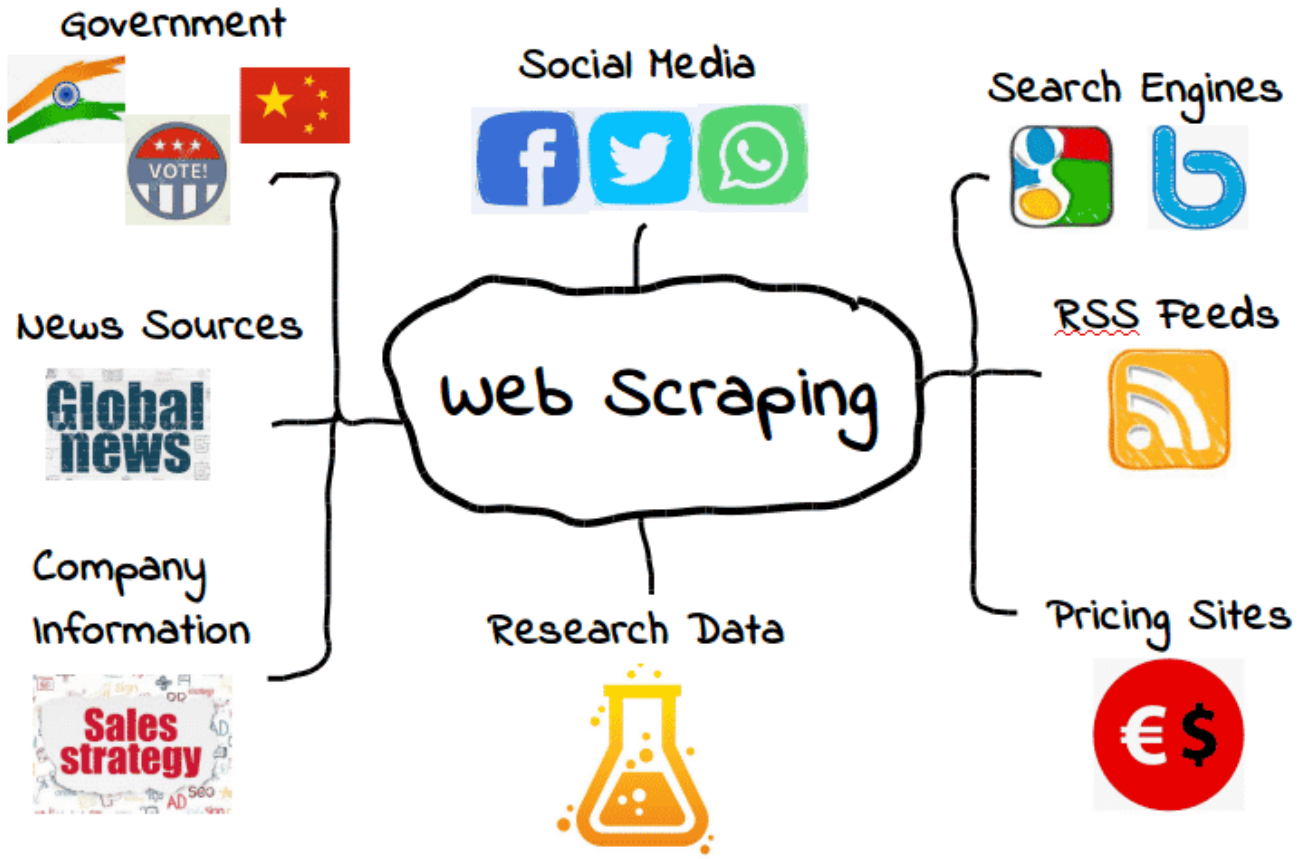
Extracción legal de contenidos en la web

Conjunto de prácticas utilizadas para extraer automáticamente ó <<scrapear>> datos de la web



¿Qué podemos <<Scrapear>>?

Desde motores de búsqueda y RSSS hasta info gubernamental.



¿LEGAL?

Si, a no ser que el sitio tenga bloqueadores de scraping significando que no desean compartir la información.

- Amazon lo permite para comparar productos.
- Wikipedia permite el acceso a información global

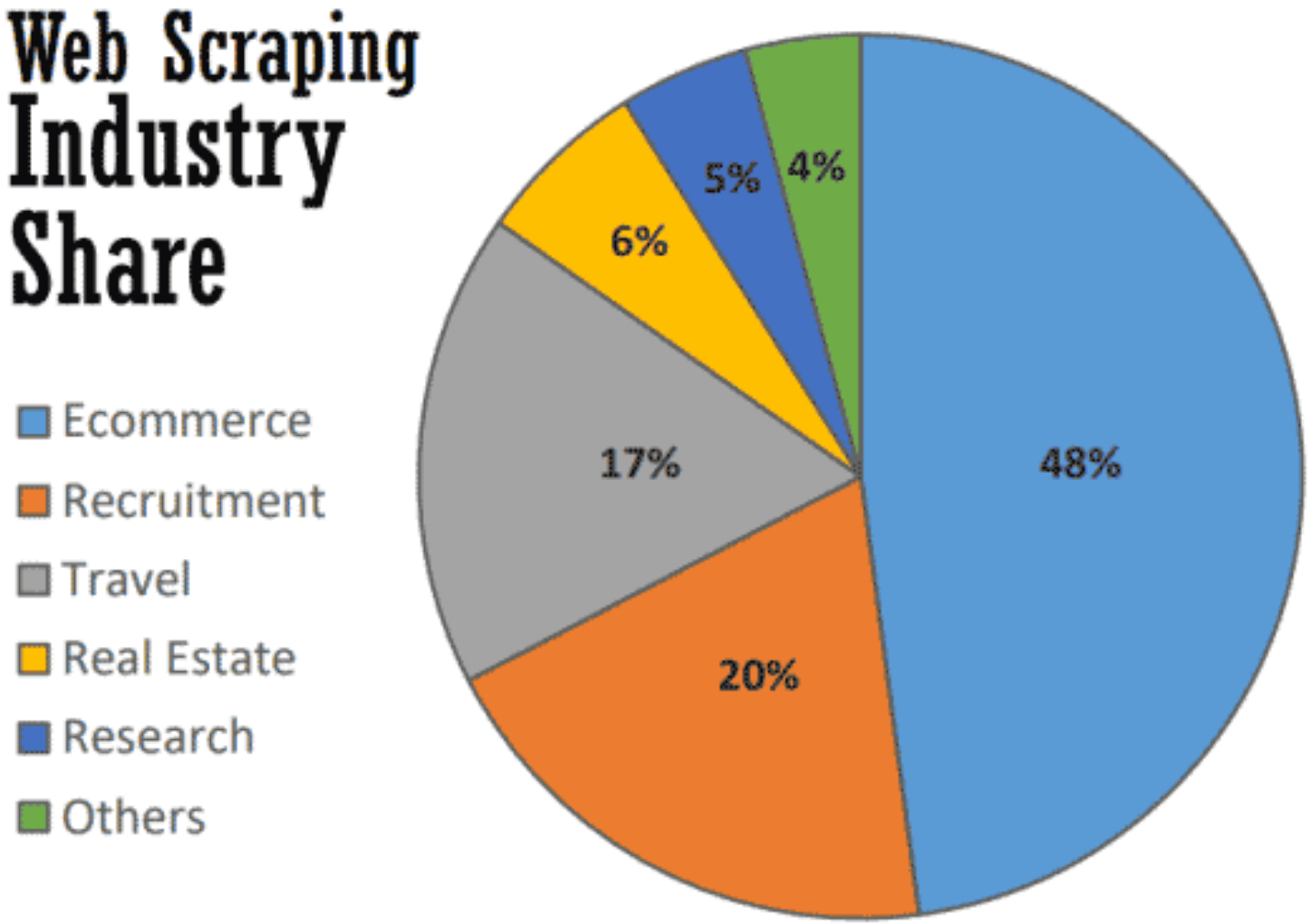
Web Scraping Malicioso

Datos personales

Propiedad intelectual

No abierto al publico

Web Scraping Hoy



¿Cómo Funciona el Scraping Web?

1. Navegador automatico web
2. Extractor de datos

Todo esto se hace mediante

Scrapers y Crawlers

Scrapers

hacen el trabajo sucio de extraer rápidamente la información relevante de los sitios web.

Dado que los sitios web están estructurados en HTML, los scrapers utilizan expresiones regulares (regex), XPath, selectores CSS y otros localizadores para encontrar y extraer rápidamente determinados contenidos.

Por ejemplo, puedes dar a tu web scraper una expresión regular que especifique el nombre de una marca o una palabra clave.

Crawlers

son programas básicos que navegan por la web buscando e indexando contenidos.

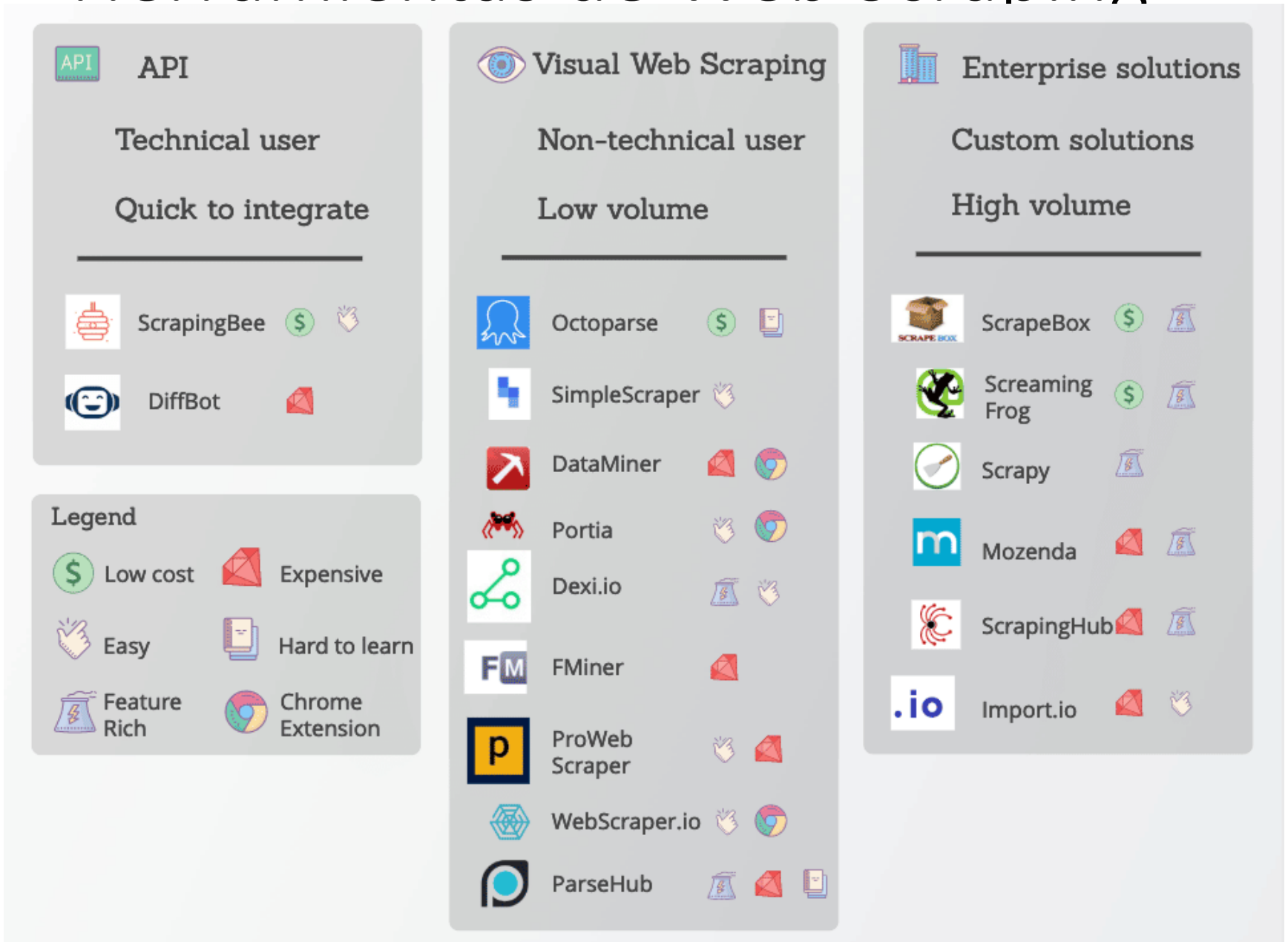
Aunque los crawlers(rastreadores) guían a los web scrapers, no se utilizan exclusivamente para este fin.

Por ejemplo, los motores de búsqueda como Google utilizan rastreadores para actualizar los índices y las clasificaciones de los sitios web. Los rastreadores suelen estar disponibles como herramientas preconstruidas que permiten especificar un determinado sitio web o término de búsqueda.

Proceso Base

1. Especifica las URLs de los sitios web y las páginas que quieres scrapear
2. Haz una petición HTML a las URL (es decir, «visita» las páginas)
3. Utiliza localizadores como expresiones regulares para extraer la información deseada del HTML
4. Guarda los datos en un formato estructurado (como CSV o JSON)

Herramientas de Web Scraping



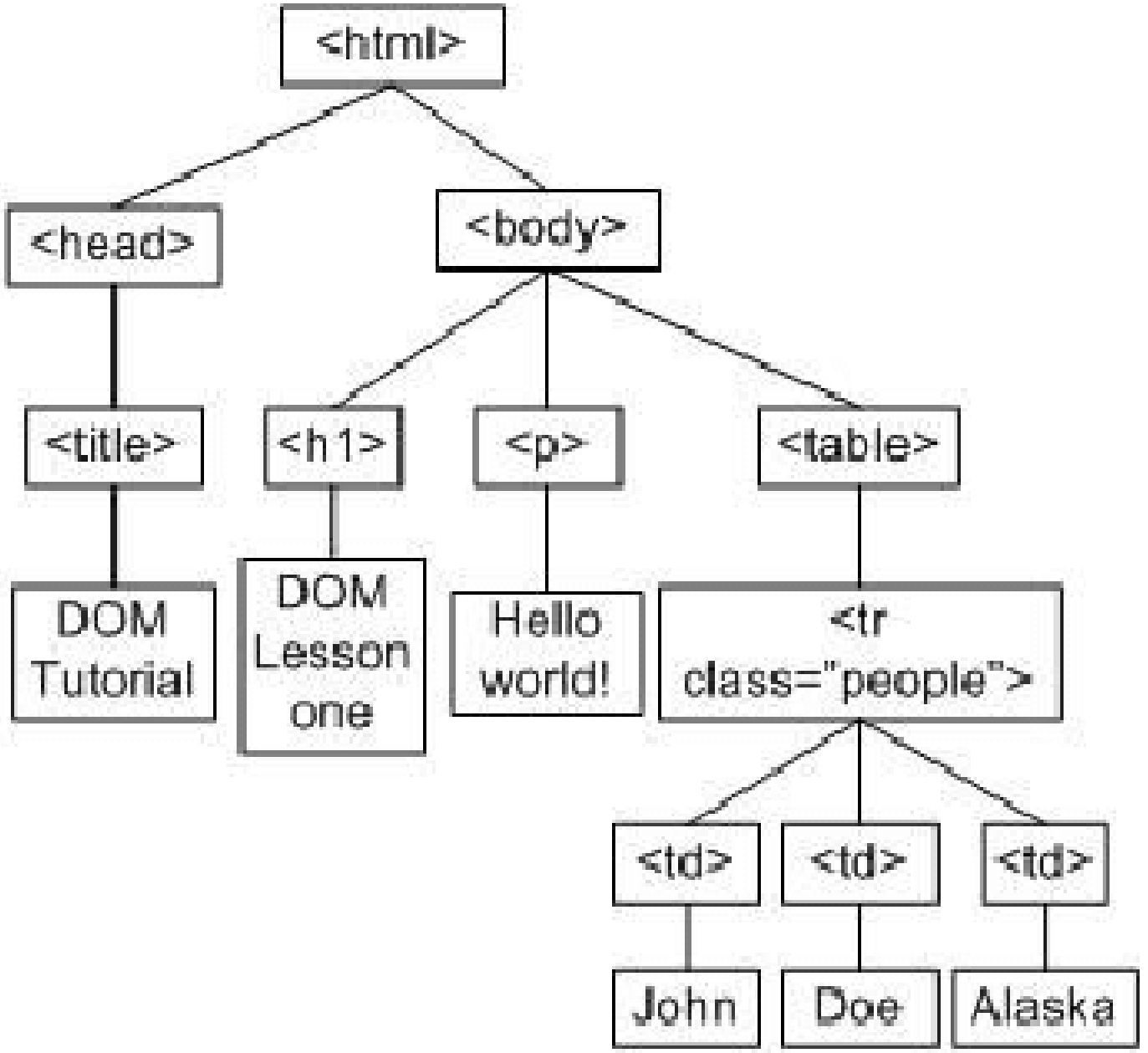
Protección contra Scraping

- Bloqueo IP
- Archivos robots.txt (indica acceso publico y privado)
- Filtro de solicitudes por peticiones HTML
- Captcha
- Honeypots (espacios trampa para distraer bots)

Web Scraping con Python

```
<html>
    <head>
        <title>DOM Tutorial</title>
    </head>
    <body>
        <h1>DOM Lesson one</h1>
        <p>Hello world!</p>
        <table>
            <tr class="people">
                <td>John</td>
                <td>Doe</td>
                <td>Alaska</td>
            </tr>
        </table>
    </body>
</html>
```

test.html

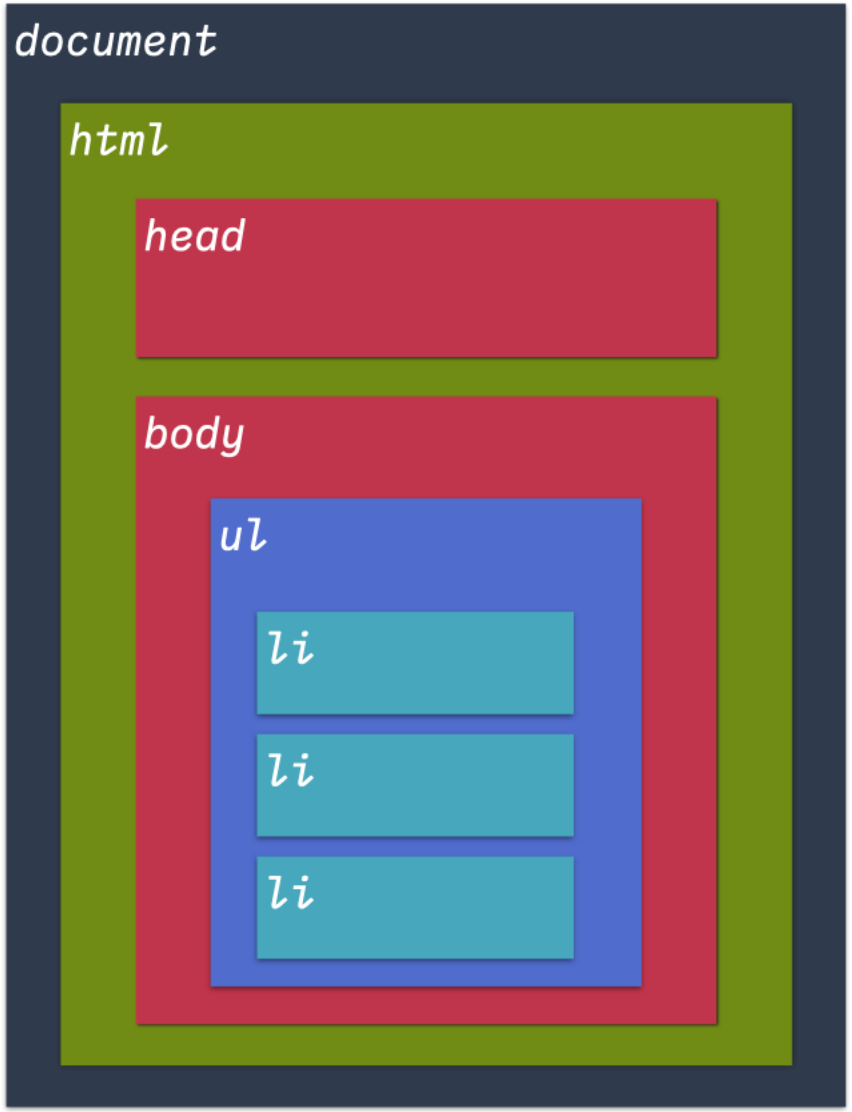


DOM Tree

```
<html lang="en">
  <head>…</head>
  <body>
    …
    <ul>
      <li>💃 dance</li>
      <li>🎉 cheer</li>
      <li>🌮 eat</li>
    </ul>
  </body>
</html>
```

Pick your party style!

- 💃 dance
- 🎉 cheer
- 🌮 eat



Selenium

1. Configuración inicial

```python
from selenium import webdriver
from selenium.webdriver.common.by import By

# Configurar el navegador (en este caso, Chrome)
driver = webdriver.Chrome(executable_path='ruta_al_chromedriver')

# Navegar a una página web
driver.get('https://www.ejemplo.com')
```

Selenium

2. Encontrar elementos en la pagina

```
# Encontrar un elemento por su ID
elemento = driver.find_element(By.ID, 'elemento_id')

# Encontrar un elemento por su nombre
elemento = driver.find_element(By.NAME, 'elemento_nombre')

# Encontrar un elemento por su XPath
elemento = driver.find_element(By.XPATH, '//*[@id="elemento_id"]')

# Obtener el texto de un elemento
texto = elemento.text
```

Selenium

3. interactuar: hacer click, enviar texto, obtener atributos..

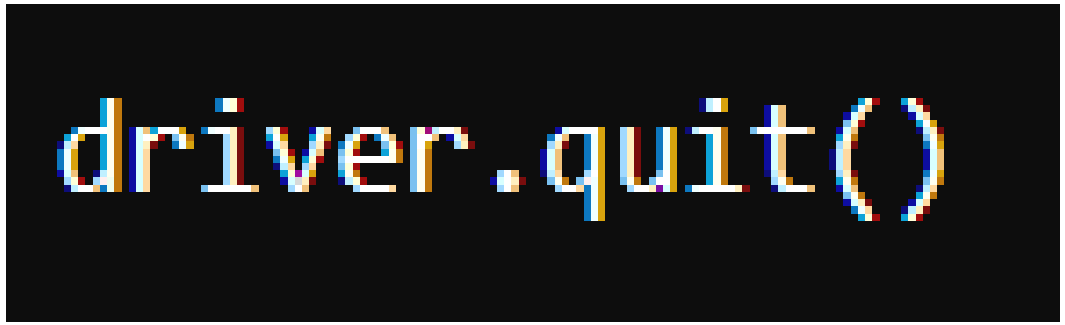
```
# Hacer clic en un elemento
elemento.click()

# Enviar texto a un campo de entrada
campo_texto = driver.find_element(By.NAME, 'campo_nombre')
campo_texto.send_keys('Texto de ejemplo')

# Obtener un atributo de un elemento
atributo = elemento.get_attribute('nombre_atributo')
```

Selenium

4. Una vez terminado el scraping, es importante cerrar para loberar recursos.



Selenium, Buenas Practicas

1. Manejo de esperas: Las paginas a menudo tienen carga dinámica, puedes hacer esperar explicitas para cargar elementos

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


# Esperar hasta que el elemento esté presente
elemento = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, 'elemento_id'))
)
```

Selenium, Buenas Practicas

2. Evitar Detección: algunos sitios detectan y bloquean bots, Técnicas como cambio de agente de user, usar proxies y manejar cookies pueden evitar la detección.

3. Manejo de Errores: Siempre es útil manejar posibles errores y excepciones para que tu scrip sea más robusta.

```
from selenium.common.exceptions import NoSuchElementException


try:
    elemento = driver.find_element(By.ID, 'elemento_id')
except NoSuchElementException:
    print("El elemento no fue encontrado")
```

Otros...

- BeutifulSoup -> easy to use (ejemplo el en notebook)
- Scrapy -> directamente para Python
- Asyncio -> gestión HL APIs