Web Scrapping

Pasquel Johann

Tabla de Contenidos

GITHUB

https://github.com/Vladimirjon/MetodosNumericos_PasquelJohann/tree/main/Extra_MN/Web_scrapping

1. OBJETIVOS

- Investigar y comprender el funcionamiento de las librerías **requests** y **BeautifulSoup** para realizar solicitudes HTTP y analizar documentos HTML en el contexto de web scraping.
- Desarrollar un script en Python que implemente ambas librerías para extraer información estructurada, como títulos, encabezados y enlaces, desde una página web seleccionada.

2. INTRODUCCIÓN

El ***web scraping*** es una técnica que permite extraer datos de sitios web de manera automatizada. En un mundo impulsado por la información, esta práctica ha ganado relevancia en diversos campos, como el análisis de datos, el comercio electrónico, las investigaciones académicas y la minería de información.

Herramientas como Python, con su ecosistema de bibliotecas especializadas, han hecho que el web scraping sea más accesible y eficiente para desarrolladores y analistas.

PROCESO DE WEB SCRAPING

1. Estructura del sitio Web

Analizar la estructura del sitio web objetivo. Esto implica inspeccionar el código HTML del sitio, identificando las etiquetas y atributos relevantes para los datos que se desean extraer.

2. Configuración del Entorno

Configurar un entorno de desarrollo que incluya las herramientas adecuadas. Librerías adecuadas y dependencias.

3. Implementación del Scraping

- Realizar una solicitud HTTP al servidor para obtener el HTML de la página.
- Analizar el HTML recibido
- Extraer los datos deseados

El web scraping es una herramienta poderosa que transforma el acceso y análisis de datos en procesos automatizados y eficientes. A través de Python, los desarrolladores pueden construir soluciones personalizadas para extraer información valiosa de sitios web, respetando siempre las normativas legales y éticas. Al combinar las mejores prácticas y las herramientas adecuadas, el web scraping se convierte en un recurso indispensable para el manejo de datos en el mundo actual.

3. Código en Python

3.1 Configuración del Entorno

```
pip install requests
```

```
Requirement already satisfied: requests in /home/johann/miniforge3/envs/vladimirjon/lib/pytho
Requirement already satisfied: charset-normalizer<4,>=2 in /home/johann/miniforge3/envs/vladi
Requirement already satisfied: idna<4,>=2.5 in /home/johann/miniforge3/envs/vladimirjon/lib/p
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/johann/miniforge3/envs/vladimirjon
Requirement already satisfied: certifi>=2017.4.17 in /home/johann/miniforge3/envs/vladimirjon
Note: you may need to restart the kernel to use updated packages.
```

```
pip install beautifulsoup4
```

```
Requirement already satisfied: beautifulsoup4 in /home/johann/miniforge3/envs/vladimirjon/lib
Requirement already satisfied: soupsieve>1.2 in /home/johann/miniforge3/envs/vladimirjon/lib/
Note: you may need to restart the kernel to use updated packages.
```

3.2 Implementación en Python

Importación de Librerías

En esta sección se importan las bibliotecas necesarias para realizar web scraping, analizar los datos obtenidos y almacenarlos de forma estructurada en un archivo CSV: - **`requests`**: Permite realizar solicitudes HTTP. - **`BeautifulSoup`**: Facilita el análisis del contenido HTML. - **`csv`**: Proporciona herramientas para manipular y guardar datos en formato CSV.

```
import requests as req
from bs4 import BeautifulSoup
import csv
```

Obtención del Contenido HTML

En esta sección, se realiza una solicitud HTTP al servidor utilizando la biblioteca `requests`.

Esto permite obtener el código fuente HTML de la página web objetivo, que será analizado en pasos posteriores. El contenido es almacenado en la variable `response` para su procesamiento.

```
url = "https://www.scrapingbee.com/blog/web-scraping-101-with-python/"
response = req.get(url)
```

Extracción y Almacenamiento de Datos

Análisis del contenido HTML previamente obtenido para extraer información estructurada, como el título principal de la página, encabezados y enlaces. Finalmente, los datos son almacenados en un archivo CSV para su posterior análisis.

Resultado esperado:

- Un archivo llamado `scraped_data.csv` con el siguiente contenido estructurado:
  - Título principal de la página.
  - Lista de encabezados encontrados.
  - Lista de enlaces (hasta 10) con sus respectivas URLs.

```
if response.status_code == 200:
    soup = BeautifulSoup(response.text, 'html.parser')
    main_title = soup.find('h1').text
    print(f"Título principal: {main_title}\n")

    headers = soup.find_all(['h2', 'h3'])
    links = soup.find_all('a', href=True)

    # Guardar en un archivo CSV
    with open('scraped_data.csv', mode='w', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        writer.writerow(['Tipo', 'Contenido'])

        writer.writerow(['Título principal', main_title])
        writer.writerow(['Encabezados'])
        for header in headers:
            writer.writerow(['Encabezado', header.text])


        writer.writerow(['Enlaces'])
        for link in links[:10]:
            writer.writerow(['Enlace', link['href']])

    print("Datos guardados en scraped_data.csv")
else:
    print(f"Error al acceder a la página: {response.status_code}")
```

```
Título principal: Python Web Scraping: Full Tutorial With Examples (2024)
```

```
Datos guardados en scraped_data.csv
```

4. CONCLUSIONES

- El uso combinado de **requests** y **BeautifulSoup** facilita el proceso de web scraping al permitir realizar solicitudes HTTP eficientes y analizar documentos HTML de manera estructurada.
- El script en Python permitió validar la capacidad de estas librerías para extraer información significativa, como títulos, encabezados y enlaces, y almacenar los resultados en un formato reutilizable como CSV.

5. REFERENCIAS BIBLIOGRÁFICAS

https://www.scrapingbee.com/blog/web-scraping-with-scrapy/

https://www.scrapingbee.com/blog/web-scraping-101-with-python/

https://www.scrapingbee.com/blog/web-scraping-without-getting-blocked/