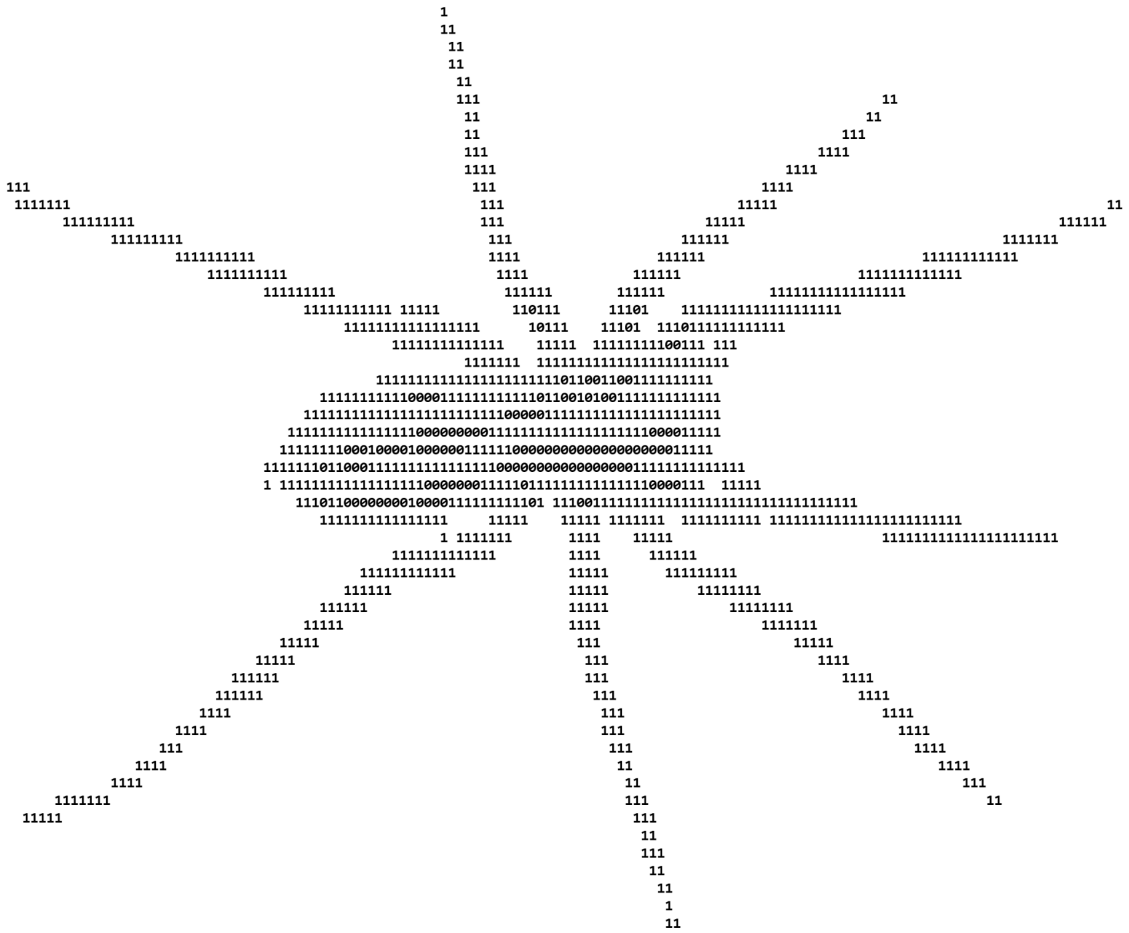
Web Scraping Projects



Objetivo del Documento

El presente documento tiene por finalidad abordar el estudio de los módulos de Python vinculados al Web Scraping, mediante la implementación de proyectos prácticos que permitan, por un lado, una comprensión profunda de la documentación técnica y el correcto uso de dichas herramientas; y por otro, el desarrollo de aplicaciones concretas que posean valor funcional.

Los proyectos estarán orientados principalmente a temáticas relacionadas con la economía y el trading algorítmico, con el objetivo de aportar soluciones prácticas dentro de estos campos de estudio.

Proyecto I: News Analysis.

Introducción

El objetivo de este proyecto es desarrollar un sistema de búsqueda y análisis de noticias, mediante técnicas de Web Scraping, que permita extraer contenidos relevantes a partir de palabras clave definidas por el usuario. La finalidad es facilitar el acceso a información actualizada proveniente de diversos portales de noticias, centrada en temáticas específicas de interés.

Para llevar a cabo esta tarea se emplearán los módulos requests y BeautifulSoup, fundamentales en la recolección y el procesamiento de contenido web en Python. Este enfoque no solo permite automatizar la obtención de datos informativos, sino que también contribuye al desarrollo de herramientas aplicables al análisis contextual en disciplinas como la economía, las finanzas y el trading algorítmico.

Este proyecto se enfoca específicamente en la extracción de contenido desde el portal The Daily Upside, con el objetivo de introducir de forma progresiva los conceptos fundamentales del Web Scraping. Al limitar el alcance a una única fuente, se busca facilitar la comprensión de los módulos involucrados —en este caso, requests y BeautifulSoup— sin introducir, en esta etapa inicial, las complejidades asociadas a estructuras web más dinámicas o problemáticas técnicas avanzadas.

Esta aproximación gradual permite construir una base sólida de conocimientos antes de abordar escenarios más complejos, manteniendo la coherencia con el propósito general del documento: desarrollar herramientas aplicables en contextos vinculados al trading algorítmico y materias afines.

Parte I: Introducción al Web Scraping con The Daily Upside

Este proyecto se centra en la implementación de técnicas de web scraping aplicadas al portal de noticias The Daily Upside. La elección de este sitio responde a su estructura HTML limpia y ordenada, lo que facilita la comprensión de las bases técnicas necesarias para utilizar eficientemente las bibliotecas requests y BeautifulSoup. A diferencia de muchos portales de noticias locales, este sitio permite ilustrar con mayor claridad el flujo de trabajo para la extracción de datos web.

Durante este primer paso, el objetivo será obtener los enlaces individuales de cada noticia publicada en el sitio y almacenarlos en un archivo CSV, lo cual permitirá su posterior análisis y procesamiento en la Parte II del proyecto.

1.1 Módulos a importar

El primer paso en cualquier proyecto de web scraping consiste en importar los módulos necesarios. En nuestro caso, trabajamos con tres bibliotecas fundamentales: requests, BeautifulSoup del módulo bs4 y csv. Estas herramientas permiten establecer conexiones con páginas web, extraer contenido de manera estructurada y, finalmente, almacenar los datos obtenidos en un formato legible y reutilizable.

La biblioteca requests se encarga de simplificar las solicitudes HTTP/1.1. Con ella, podemos acceder a una página web utilizando funciones como requests.get() o requests.post() de forma muy intuitiva. Además de esto, requests permite el manejo automático de cadenas de consulta (query strings) mediante el parámetro params, evitando que tengamos que concatenar manualmente los parámetros a las URLs. También maneja de forma eficiente las conexiones persistentes (keep-alive) y el pooling de conexiones, lo que se traduce en un mejor rendimiento al hacer múltiples solicitudes al mismo servidor sin tener que reabrir la conexión cada vez.

Solicitudes HTTP simplificadas: A través de métodos como requests.get() y requests.post(), se pueden realizar solicitudes a servidores que operan con el protocolo HTTP/1.1 sin preocuparse por los detalles internos del mismo.

Cadenas de consulta (query strings): Si es necesario pasar parámetros en la URL (por ejemplo: ?nombre=juan&edad=30), requests lo gestiona de forma automática mediante el parámetro params, evitando la concatenación manual.

Conexiones persistentes (Keep-Alive): Por defecto, requests mantiene viva la conexión al servidor siempre que sea posible, lo que permite realizar múltiples solicitudes sin reabrir la conexión, optimizando así el rendimiento.

Pooling de conexiones: La biblioteca reutiliza conexiones abiertas cuando se realizan múltiples solicitudes al mismo dominio, reduciendo la sobrecarga del sistema y acelerando la ejecución general del scraping.

Por otro lado, BeautifulSoup es una biblioteca muy popular para el análisis y procesamiento de documentos HTML y XML. Su función principal es construir un árbol de análisis (parse tree) del contenido web, sobre el cual se pueden aplicar búsquedas de etiquetas, clases, atributos y más. Gracias a su flexibilidad, esta herramienta permite extraer información de forma precisa, ahorrando tiempo en comparación con métodos más manuales o complejos de parsing.

El módulo csv se utiliza para guardar los datos extraídos en archivos delimitados por comas (.csv). Este tipo de formato es ideal cuando se desea trabajar con los datos en herramientas externas como Excel, Google Sheets o incluso con bibliotecas más avanzadas como Pandas. Su implementación es sencilla y adecuada para proyectos que no requieren estructuras complejas de datos, como cuando simplemente se quieren almacenar listas de enlaces o textos.

El módulo urllib se emplea para acceder y recuperar información desde páginas web, facilitando el proceso de extracción de datos mediante solicitudes HTTP. Esta herramienta es particularmente útil cuando se realiza web scraping básico, permitiendo obtener el contenido HTML de una URL de manera directa y eficiente.

Por su parte, el módulo sqlite3 permite almacenar los datos extraídos en una base de datos relacional liviana, ideal para proyectos que requieren persistencia de la información sin necesidad de configurar un servidor de base de datos externo. A diferencia del formato CSV, que es más adecuado para almacenamiento plano y operaciones simples, SQLite ofrece mayor flexibilidad al permitir consultas complejas, relaciones entre tablas, y una estructura más robusta para el manejo de datos.

1.2 Obtener la URL del sitio

Una vez definidos los módulos necesarios, podemos comenzar a escribir el código que nos permitirá obtener el contenido de la página web. Como se mencionó anteriormente, la biblioteca requests simplifica el proceso de enviar solicitudes HTTP, lo que nos facilita acceder a las páginas de forma eficiente.

En este caso, la función **requests.get()** se utiliza para acceder a la página principal de The Daily Upside, la cual se almacena en la variable thu_url. El código es el siguiente:

```
import requests
from bs4 import BeautifulSoup
import csv
import sqlite3
from urllib.parse import urlparse

thu_url = "https://www.thedailyupside.com/"
thu_response = requests.get(thu_url)
```

Aquí, el resultado de la solicitud se guarda en el objeto thu_response. Este objeto contiene tanto el contenido de la página web como metainformación útil, como el código de estado de la respuesta. Es importante tener en cuenta que el código de estado HTTP nos proporciona información sobre el resultado de la solicitud.

El atributo status_code del objeto Response es clave para determinar si la solicitud fue exitosa. Los códigos más relevantes que podemos encontrar son:

- 200: Éxito. La página fue encontrada y devuelta correctamente.
- 404: No encontrado. El recurso solicitado no existe.
- 500: Error interno del servidor. Hubo un problema en el servidor.

En este caso, se configura el objeto thu_response para que almacene la respuesta de la solicitud a la URL de la página principal de The Daily Upside. Sin embargo, no basta con obtener la respuesta; es esencial verificar que la conexión fue exitosa antes de proceder.

1.2.1 Verificación del estado de la conexión

Una buena práctica al trabajar con solicitudes HTTP es verificar siempre el código de estado antes de procesar el contenido de la página. Esto nos garantiza que estamos trabajando con una respuesta válida. Esta verificación se puede realizar dentro de una función que será responsable de recolectar los enlaces de noticias.

```
thu_themes = ["economics", "finance", "technology",
                "business", "industries", "investments",
                "newsletters"]
links = set()

obtener_links_de_noticias(thu_url, thu_response, thu_themes):
        if response.status_code == 200:
                …
        else:
                print(f"Error al acceder a la página: {response.status_code}")
```

La función no solo recibirá la URL y la respuesta HTTP, sino también una lista de temas clave (por ejemplo, "economics", "finance", etc.), así como una estructura de datos tipo set que almacenará los enlaces sin permitir duplicados. El uso de un set es fundamental, ya que garantiza que cada enlace sea único, evitando la acumulación de URLs repetidas.

Este enfoque inicial es útil para controlar el flujo del programa, asegurando que solo se procese contenido si la respuesta del servidor fue exitosa (es decir, si el código de estado es 200).

1.3 Implementación de BeautifulSoup

Una vez que nos aseguramos de haber recibido correctamente el contenido del sitio web mediante requests, el siguiente paso es procesar ese contenido HTML. Para ello, utilizamos BeautifulSoup, una poderosa librería que nos permite analizar documentos HTML o XML y extraer información específica de forma sencilla y estructurada.

1.3.1 Creando el objeto soup

La función BeautifulSoup() se encarga de convertir el texto HTML crudo (contenido que devuelve response.text) en un árbol de objetos de Python que puede ser recorrido, buscado y modificado. Para comenzar, creamos un objeto llamado soup, que representa la estructura del sitio web. Este objeto será el punto de entrada para todas las búsquedas y manipulaciones que realizaremos sobre el HTML. A través de este objeto, podemos navegar por el contenido del sitio web, buscar etiquetas específicas como <a>, , <h1>, etc., y extraer los datos que necesitamos.

El código para crear el objeto soup es el siguiente:

```
def obtener_links_de_noticias(url, response, themes, links):
    if response.status_code == 200:
        soup = BeautifulSoup(response.text, 'html.parser')

        # Buscar todos los enlaces que podrían ser noticias
        enlaces = soup.find_all('a', href=True)
    else:
        print(f"Error al acceder a la página: {response.status_code}")
```

Aquí, se crea una instancia de la clase BeautifulSoup, proveniente de la librería bs4. Su propósito es convertir el texto crudo de una página web (HTML o XML) en un árbol de objetos que se puede recorrer y manipular fácilmente. En otras palabras, transforma el texto HTML en una estructura jerárquica entendible por Python.

El contenido de la respuesta HTTP, obtenido mediante **requests.get(...)**, es lo que contiene todo el código fuente HTML como una cadena de texto (tipo str). Este contenido es similar al código fuente que se visualiza al hacer clic derecho en una página web y elegir "Ver código fuente".

El parámetro 'html.parser' especifica el "analizador sintáctico" (**parser**) que BeautifulSoup debe usar para interpretar el HTML. Hay varios analizadores disponibles, y la elección depende de los requerimientos del proyecto:

- 'html.parser': El parser que viene por defecto con Python. Es rápido y no requiere instalar nada extra.
- 'lxml': Más rápido y robusto, pero requiere que instales la librería lxml.
- 'html5lib': Trata de imitar cómo un navegador interpreta el HTML, pero es más lento.

Usar 'html.parser' es común cuando se busca algo liviano y sin dependencias externas.

1.3.2 Buscar enlaces

Una vez que tenemos el objeto soup, podemos realizar búsquedas específicas dentro del HTML. En este caso, queremos buscar todos los enlaces de noticias. Para ello, creamos una variable llamada enlaces, que almacenará el resultado de una búsqueda dentro del HTML.

El método **soup.find_all(...)** del objeto soup, que proviene de BeautifulSoup, busca todas las ocurrencias de una etiqueta HTML específica dentro del documento. En este caso, la etiqueta buscada es <a>, que se utiliza para definir los enlaces en HTML. El filtro **href=True** asegura que solo se devuelvan las etiquetas <a> que tengan el atributo href, que es donde se almacena el enlace real.

Este método retornará una lista de todas las etiquetas <a> que contienen un atributo href. Cada uno de esos atributos será el enlace que nos interesa extraer para nuestra posterior recopilación.

1.4 Bucle for para filtrar enlaces relevantes

Una vez identificados los posibles enlaces presentes en una página web, el paso siguiente consiste en analizarlos individualmente para determinar su relevancia. Este procedimiento se lleva a cabo mediante un bucle for, el cual permite recorrer cada enlace extraído, evaluar si se ajusta a las categorías temáticas deseadas y descartar aquellos que no resulten pertinentes o que estén duplicados.

```
def es_url_articulo(url):
    """
    Determina si una URL probablemente corresponde a un artículo.
    Se considera que una URL es de un artículo si tiene al menos tres segmentos en su ruta.
    """
    path = urlparse(url).path
    segmentos = path.strip('/').split('/')
    return len(segmentos) >= 3

def obtener_links_de_noticias(url, response, themes, links):
    if response.status_code == 200:
        soup = BeautifulSoup(response.text, 'html.parser')

        # Buscar todos los enlaces que podrían ser noticias
        enlaces = soup.find_all('a', href=True)

        for enlace in enlaces:
            href = enlace['href']
            link_completo = href if href.startswith(url) else url + href.lstrip('/')

            if any(t in href for t in themes) and es_url_articulo(link_completo):
                links.add(link_completo)
    else:
        print(f"Error al acceder a la página: {response.status_code}")
```

Dentro de la función obtener_links_de_noticias, luego de haber convertido el HTML de la página en un objeto estructurado con BeautifulSoup y de haber extraído todos los enlaces con **soup.find_all('a', href=True)**, se inicia un bucle que analiza cada uno de esos enlaces. Con **href = enlace['href']**, accedemos al valor del atributo href de la etiqueta <a>, que es el link propiamente dicho.

A continuación, se normaliza el enlace mediante una condición que verifica si la URL ya es completa (es decir, si comienza con https://www.thedailyupside.com/). Si lo es, se utiliza tal cual; en cambio, si es relativa (por ejemplo, "/noticia"), se le antepone la URL base del sitio (url) y se le remueven las barras iniciales innecesarias con **href.lstrip('/').** Esta normalización es fundamental para asegurar que todos los enlaces sean utilizables desde fuera del sitio web original.

Luego se aplica un filtro muy importante: se utiliza **any(t in href for t in themes)** and es_url_articulo(link_completo) para chequear si el enlace contiene alguno de los temas principales que nos interesan, utilizando la librerira urllib se verifica, con la función es_url_articulo, que la estructura de la URL corresponda a la de un artículo (por ejemplo, para evitar enlaces a secciones generales como (que a menudo llevan a nuevas listas de artículos), como "economics", "technology" o "finance". Si el enlace incluye al menos una de esas palabras clave, consideramos que es una noticia válida y relevante. Esta verificación permite enfocarnos únicamente en los contenidos de interés, y evita que obtengamos enlaces inútiles como formularios, publicidad o contenido institucional.

Finalmente, si el enlace cumple con las condiciones anteriores, se agrega al conjunto links mediante links.add(link_completo). Utilizar un conjunto (set) es una decisión estratégica, ya que automáticamente elimina duplicados. Esto es especialmente útil si la misma noticia aparece en diferentes secciones del sitio, o si hay múltiples enlaces al mismo contenido.

1.5 Ejecución

Para poner en marcha nuestro código y comenzar efectivamente con el proceso de recolección de datos, debemos considerar cómo está estructurada la página de The Daily Upside. Si solo ejecutamos la función sobre la página principal, obtendremos únicamente un subconjunto limitado de noticias. Sin embargo, esta web organiza su contenido en secciones temáticas específicas —como economics, technology o finance—, por lo que es mucho más eficiente y productivo ingresar directamente a la grilla de noticias de cada una de estas secciones.

```
# Iteramos por tema
for tema in thu_themes:
    full_url = thu_url + tema + '/'
    response = requests.get(full_url)
    obtener links de noticias(thu_url, response, thu_themes, links)
```

Para lograr esto, ejecutamos un bucle for que recorre cada uno de los temas definidos en la lista thu_themes. En cada iteración, concatenamos la URL base del sitio (thu_url) con el nombre del tema y un / al final, ya que así es como están estructuradas las URLs internas del sitio. Esto se realiza con la línea:

full_url = thu_url + tema + '/'. De esta manera, por cada uno de los temas, generamos una URL completa que apunta directamente a la sección correspondiente del sitio. A continuación, realizamos una solicitud HTTP usando **requests.get(full_url)** para obtener el contenido HTML de esa página específica. Con esa respuesta, invocamos nuestra función obtener_links_de_noticias(...) pasando como argumentos la URL base, la respuesta, la lista de temas y el conjunto de links ya creado.

Esto nos permite automatizar el ingreso a cada una de las secciones del sitio, garantizando así una recolección más exhaustiva de los enlaces a noticias. Es decir, estamos ampliando nuestro alcance de búsqueda más allá de lo que aparece en la página de inicio, lo que resulta especialmente útil cuando se desea construir una base de datos más rica y completa para análisis posteriores.

Además, el hecho de agregar la barra / directamente dentro del bucle (en lugar de en la lista de temas) brinda una mayor flexibilidad y generalidad al código. De esta forma, podemos reutilizar la lista thu_themes en otros contextos sin preocuparnos por la estructura del enlace. Todo esto potencia la reutilización y escalabilidad del script, manteniéndolo limpio y funcional.

1.6 Almacenar los links

Una vez que obtenemos todos los enlaces relevantes a noticias, necesitamos almacenarlos de manera eficiente para su posterior análisis o uso. Existen múltiples alternativas para hacerlo, entre ellas, utilizar herramientas como Pandas o bases de datos como SQL, que ofrecen una gran versatilidad a la hora de trabajar con información más compleja. Sin embargo, dado que los datos que estamos recolectando son simplemente enlaces (URLs) y no requieren una estructura demasiado sofisticada, optamos por una solución más simple: el módulo csv de Python.

Este módulo nos permite guardar los datos en un archivo .csv, un formato ampliamente compatible con hojas de cálculo como Excel, Google Sheets o incluso otros scripts en Python. La ventaja de este enfoque es su sencillez: no necesita instalación adicional y es suficiente para manejar una lista de enlaces como la que estamos generando.

El código que utilizamos para guardar los links se divide en dos bloques:

```
with open('links_noticias.csv', mode='w', newline='', encoding='utf-8') as f:
    csv.writer(f)
```

Este primer bloque se encarga de limpiar el archivo antes de usarlo, asegurándonos de que, cada vez que ejecutemos el script, el archivo comience vacío. Esto es útil cuando no nos interesa mantener un historial de ejecuciones anteriores, sino que queremos trabajar siempre

con los links más recientes. Se utiliza **mode='w'** para sobrescribir el archivo (si ya existe), y se establece **encoding='utf-8'** para asegurar la correcta escritura de caracteres especiales. Además, **newline=''** evita que se agreguen líneas en blanco entre filas, un problema común en sistemas Windows cuando se trabaja con archivos CSV.

Una vez que tenemos nuestros enlaces recolectados, los escribimos en el archivo con el siguiente bloque:

```
with open('links_noticias.csv', mode='w', newline='', encoding='utf-8') as archivo_csv:
    escritor = csv.writer(archivo_csv)
    for link in links:
        escritor.writerow([link])
```

Este segundo bloque es similar al primero, pero ahora sí escribe el contenido real: recorre el conjunto links (que contiene todos los enlaces recopilados) y los guarda línea por línea. Al igual que antes, el archivo se abre en modo controlado **(with open(...))** para asegurar que se cierre correctamente, incluso si ocurre un error durante la escritura.

Ambos bloques son importantes: el primero para evitar duplicaciones no deseadas al reiniciar el script, y el segundo para asegurar que todos los enlaces extraídos se almacenen correctamente. Este enfoque simple, pero efectivo, nos permite contar con un archivo ordenado y limpio, que puede ser utilizado directamente en análisis posteriores o incluso compartido fácilmente.

Parte II: Lectura de artículos

La lectura de los artículos que se obtienen desde fuentes como The Daily Upside constituye el objetivo principal de todo este proceso, ya que nos permite acceder a la información contenida en cada nota en un formato estructurado y legible, tanto para nosotros como para futuras implementaciones de código orientadas al análisis y comprensión automatizada del contenido.

Dado que el propósito de este documento no es analizar la información, sino extraerla, nos enfocaremos en detallar los distintos métodos mediante los cuales se pueden recuperar datos relevantes. Esta información podrá ser utilizada tanto para tareas rutinarias como para aplicaciones más específicas, tales como la generación de resúmenes automáticos, la clasificación temática de artículos, o el entrenamiento de modelos de lenguaje.

2.1 Obtener la url

Para comenzar con la lectura de los artículos, el primer paso consiste en recuperar las URLs previamente almacenadas en el archivo CSV llamado “links_noticias.csv”. Con este propósito, desarrollamos una función que automatiza el proceso de lectura del archivo:

```
def leer_y_guardar_articulo():
    with open('links_noticias.csv', mode='r', encoding='utf-8') as archivo_csv:
        lector = csv.reader(archivo_csv)
```

En este bloque, utilizamos el módulo csv. El argumento mode='r' indica que abrimos el archivo en modo lectura, mientras que la línea lector = csv.reader(archivo_csv) transforma el contenido del archivo en una secuencia iterable de listas, donde cada lista representa una fila del archivo CSV.

2.1.2 Leer URLs de forma individual

Al leer el archivo CSV, se accede a todas las URLs de forma conjunta. Sin embargo, para procesar cada artículo por separado, necesitamos iterar individualmente sobre cada enlace. A continuación, mostramos un bloque de código que permite lograr este comportamiento:

```
def leer_y_guardar_articulo():
    with open('links_noticias.csv', mode='r', encoding='utf-8') as archivo_csv:
        lector = csv.reader(archivo_csv)
        for i, fila in enumerate(lector, 1):
            url = fila[0]
            try:
                response = requests.get(url, timeout=10)
```

En este caso, la línea **for i, fila in enumerate(lector, 1)**: permite recorrer cada fila del archivo, numerándolas desde el 1. Dado que cada fila es una lista, accedemos al enlace mediante **url = fila[0]**. Luego, realizamos una solicitud HTTP utilizando **requests.get(url, timeout=10)**. El argumento timeout=10 es fundamental para evitar bloqueos prolongados: si la página tarda más de 10 segundos en responder, la función cancelará la solicitud y podrá continuar con la siguiente URL.

2.2 Leer los párrafos

Para extraer el contenido textual de los artículos, utilizamos nuevamente el módulo BeautifulSoup, que nos permite navegar y filtrar elementos dentro del documento HTML de cada página web.

A continuación, se muestra una función completa que recupera los párrafos de cada artículo a partir de sus URLs:

```
def leer_y_guardar_articulos():
    with open('links_noticias.csv', mode='r', encoding='utf-8') as archivo_csv:
        lector = csv.reader(archivo_csv)
        for i, fila in enumerate(lector, 1):
            url = fila[0]
            try:
                response = requests.get(url, timeout=10)

                if response.status_code == 200:
                    soup = BeautifulSoup(response.text, 'html.parser')
                    parrafos = soup.find_all(['<p>'])

                    texto = "\n".join([p.get_text(strip=True) for p in parrafos if p.get_text(strip=True)])
                    if texto:
                        guardar_articulo_en_db(url, texto)
                        print(f"[{i}] Guardado correctamente: {url}")
                    else:
                        print(f"[{i}] Artículo sin contenido: {url}")
                else:
                    print(f"[{i}] Error {response.status_code} en: {url}")
            except Exception as e:
                print(f"[{i}] Excepción con {url}: {e}")
```

La principal diferencia con respecto a la sección anterior es la incorporación del método **soup.find_all()**, al cual se le pasa una lista de etiquetas HTML que queremos extraer solo ['p'], lo que representa los párrafos, en caso de querer extraer más información, podemos agregar más etiquetas:

- <p>: Representa los párrafos principales del artículo.
- <h2>: Suelen ser subtítulos o divisiones temáticas del texto.
- : Son listas no ordenadas que también contienen información relevante en muchos casos.

Luego, recorremos cada uno de estos elementos con un bucle for, y utilizamos **p.get_text(strip=True)** para obtener el contenido textual, eliminando espacios innecesarios al principio y al final. Finalmente, imprimimos el texto si no está vacío.

Este procedimiento nos permite transformar el contenido HTML en texto plano, facilitando su posterior análisis o almacenamiento en una base de datos estructurada.

Parte III: Almacenaje y utilización de artículos

Para finalizar con el proyecto, debemos encontrar la manera de almacenar los artículos de manera que podamos utilizarlos en un futuro. Para ello deberemos destacar que información se está almacenando según el artículo que estamos extrayendo, es decir, categorizar la información y definir de qué es lo que se está hablando.

Para almacenar la información que estamos extrayendo comenzaremos utilizaremos una librería llamada sqlite3 que nos permitirá utilizar SQL para poder almacenar toda la información de manera correcta y posteriormente, acceder a ella

3.1 Librería SQL

En la sección anterior, no se hizo referencia explícita a una función denominada guardar_articulo_en_db(url, texto), ya que ésta forma parte de una etapa posterior del flujo de trabajo: el almacenamiento persistente de los datos obtenidos. Dicha función hace uso de la biblioteca sqlite3, que permite la interacción con bases de datos SQL desde Python de manera sencilla y eficiente.

Dado que el enfoque principal de este documento no recae en la gestión avanzada de bases de datos, no se profundizará en los aspectos técnicos de la biblioteca sqlite3 más allá de lo estrictamente necesario para comprender y aplicar las funciones utilizadas en los fragmentos de código aquí presentados.

La función guardar_articulo_en_db está diseñada para insertar en una base de datos local los artículos recopilados a través del proceso de web scraping, almacenando tanto la URL fuente como el contenido textual asociado. Esta estrategia permite conservar la información estructurada y consultarla posteriormente mediante instrucciones SQL, facilitando su análisis, clasificación o reutilización en etapas posteriores del proyecto.

El uso de sqlite3 resulta especialmente adecuado en entornos donde se requiere una base de datos liviana, integrada al proyecto, y sin necesidad de configuración externa de servidores, lo que lo convierte en una solución práctica para prototipos o aplicaciones personales.

```
# Almacenar datos
def guardar_articulo_en_db(url, contenido, nombre_db='noticias.db'):
    conn = sqlite3.connect(nombre_db)
    cursor = conn.cursor()

    cursor.execute('''
        INSERT INTO articulos (url, contenido) VALUES (?, ?)
    ''', (url, contenido))

    conn.commit()
    conn.close()
```

La función guardar_articulo_en_db tiene como propósito almacenar de forma persistente los datos obtenidos durante el proceso de extracción web, utilizando una base de datos SQLite. Esta implementación permite conservar la información en un formato estructurado y fácilmente accesible para futuras consultas o análisis.

El primer paso dentro de la función consiste en establecer una conexión con la base de datos mediante **sqlite3.connect(nombre_db).** En este caso, la base de datos por defecto se denomina noticias.db, aunque este valor puede modificarse al invocar la función. Una vez abierta la conexión, se crea un cursor utilizando **conn.cursor(),** el cual actúa como intermediario para ejecutar comandos SQL dentro del entorno de la base de datos.

A continuación, se ejecuta una instrucción **INSERT INTO** que inserta un nuevo registro en la tabla artículos, almacenando dos campos: la URL de la fuente y el contenido del artículo. Estos valores se pasan de forma segura utilizando marcadores de posición (?) y una tupla de parámetros, lo cual previene posibles inyecciones SQL y asegura una mayor robustez en el manejo de datos externos.

Una vez insertados los datos, se utiliza **conn.commit()** para guardar permanentemente los cambios realizados en la base de datos. Finalmente, se cierra la conexión con c**onn.close()**, lo cual libera los recursos asociados y garantiza una correcta finalización del proceso.

3.2. Creación de bases SQL

Si corremos el código anterior resultará en error ya que no encontrará ninguna base para guardar los archivos, por eso debemos aplicar una función previa a esta que se encargue de crear la base

```
# Crear base de datos SQL
def crear_base_de_datos(nombre_db='noticias.db'):
    conn = sqlite3.connect(nombre_db)
    cursor = conn.cursor()

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS articulos (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            url TEXT NOT NULL,
            contenido TEXT NOT NULL
        )
    ''')

    conn.commit()
    conn.close()
```

La función crear_base_de_datos tiene como finalidad inicializar una base de datos SQLite local en la cual se almacenarán los artículos extraídos durante el proceso de recolección de datos. Esta función establece la estructura fundamental del almacenamiento, asegurando que exista una tabla adecuada para contener la información relevante.

En primer lugar, se establece una conexión con la base de datos a través de **sqlite3.connect(nombre_db)**, siendo noticias.db el nombre predeterminado. Este archivo actuará como contenedor de todos los registros y será creado automáticamente si no existe previamente. Luego, se genera un cursor con **conn.cursor()**, necesario para ejecutar comandos SQL dentro del entorno de la conexión activa.

El comando principal de esta función es una instrucción **CREATE TABLE IF NOT EXISTS**, la cual se encarga de crear una tabla llamada articulos, en caso de que esta no haya sido definida anteriormente. Esta tabla contiene tres columnas: id, que es una clave primaria con incremento automático; url, donde se almacena la dirección de origen del contenido; y contenido, donde se guarda el cuerpo del artículo. Ambas columnas de datos (url y contenido) están definidas como no nulas, lo cual impone una restricción de integridad sobre los datos ingresados.

Luego de ejecutar esta instrucción, se llama a **conn.commit()** para aplicar los cambios de forma definitiva en la base de datos, y se cierra la conexión con **conn.close()** para liberar los recursos utilizados. Esta función garantiza que la base de datos y su estructura estén correctamente configuradas antes de iniciar el proceso de inserción de artículos.

3.3 Ejecución del proceso

La ejecución del flujo completo de trabajo requiere únicamente la llamada secuencial de las funciones previamente definidas. En primer lugar, se invoca la función crear_base_de_datos(), encargada de inicializar la base de datos local y asegurarse de que la estructura necesaria esté disponible para almacenar los artículos. Posteriormente, se ejecuta leer_y_guardar_articulos(), función responsable de realizar la lectura de los enlaces, extraer el contenido de cada artículo y almacenarlo adecuadamente.

```
crear_base_de_datos()
leer_y_guardar_articulos()
```

Este procedimiento es suficiente para poner en marcha todo el sistema, ya que ambas funciones operan de forma coordinada. El diseño modular permite que cada componente se ejecute de forma independiente, lo que favorece la reutilización del código, su mantenimiento y posibles ampliaciones futuras.

Anexo proyecto I: Análisis de los instrumentos utilizados