

Búsqueda y Recopilación de Datos (Parte 1)



Bienvenidos a la primera parte de nuestro viaje al mundo del **web scraping**. El **web scraping**, también conocido como **recolección web** o **extracción de datos web**, es una técnica utilizada para extraer datos de sitios web. Este proceso implica obtener la página web y luego extraer datos de ella de manera estructurada, lo que facilita su análisis y uso posterior en diferentes contextos, desde el análisis de datos hasta la toma de decisiones.

¿Por Qué Aprender Web Scraping?

Entender cómo hacer scraping de datos de la web es una habilidad valiosa para cualquier profesional de datos, ya que abre la puerta a una gran cantidad de fuentes de información que pueden enriquecer análisis y mejorar la toma de decisiones. A continuación se detallan las razones principales por las que aprender web scraping es esencial:

- **Disponibilidad de Datos:** Internet es una vastísima fuente de datos. Desde bases de datos públicas hasta información accesible en blogs, foros y redes sociales, todo esto está a tu disposición para ser recolectado y analizado.

Ejemplo: Si eres un analista de mercado, podrías hacer scraping de los precios de productos de diferentes sitios web de comercio electrónico para analizar cómo se posicionan los precios de tus competidores.

- **Automatización del Proceso:** Muchas tareas de recopilación de datos son repetitivas y consumen mucho tiempo. Con el web scraping, puedes automatizar la recolección de datos en intervalos regulares sin necesidad de intervención humana.

Ejemplo: Imagina que estás interesado en obtener los resultados de los partidos de fútbol cada semana desde varios sitios web deportivos. En lugar de ingresar manualmente a cada página, puedes configurar un script de web scraping que descargue la información automáticamente y te entregue un archivo con los resultados cada lunes.

- **Ventaja Competitiva:** Obtener datos en tiempo real sobre el comportamiento del mercado, las tendencias de consumo o la actividad de la competencia puede brindarte una ventaja estratégica. Los datos pueden usarse para mejorar los procesos de toma de decisiones y desarrollar nuevos productos o servicios.

Ejemplo: Las empresas de tecnología pueden usar web scraping para obtener comentarios sobre sus productos en foros y redes sociales, analizando las opiniones de los usuarios para mejorar el diseño o las características de sus próximos lanzamientos.

Aplicaciones en el Mundo Real

El web scraping tiene muchas aplicaciones prácticas en diferentes campos. A continuación, se detallan algunas de las áreas más comunes y cómo el web scraping puede ser útil en cada caso:

- **Investigación de Mercado:** El scraping permite obtener información sobre los precios, características de productos y opiniones de los consumidores de múltiples fuentes, lo que resulta valioso para empresas que desean entender mejor el mercado y a su competencia.

Ejemplo: Si eres una empresa que vende productos electrónicos, podrías hacer scraping de los precios de tus competidores en sitios como Amazon, eBay y Walmart para ajustar tus precios de manera competitiva y entender las tendencias del mercado.

- **Comparación de Precios:** Al hacer scraping de los precios de productos en varios sitios de comercio electrónico, puedes crear comparadores de precios para que los consumidores puedan ver de forma clara dónde encontrar las mejores ofertas en línea.

Ejemplo: Una aplicación que recolecte información sobre el precio de los vuelos de diferentes aerolíneas para mostrar al usuario las mejores ofertas basadas en las fechas que ha seleccionado.

- **Análisis de Redes Sociales:** Muchas empresas y agencias de marketing usan web scraping para obtener datos de redes sociales como Twitter, Facebook e Instagram. Estos datos se usan para analizar tendencias, emociones de los usuarios o el rendimiento de las campañas publicitarias.

Ejemplo: Un analista de datos puede hacer scraping de Twitter para obtener tuits sobre un evento reciente y realizar un análisis de sentimientos para ver cómo reaccionaron los usuarios (por ejemplo, si el evento fue bien recibido o si hubo críticas).

Introducción a las Variables Exógenas en Ciencia de Datos

En el ámbito de **ciencias de datos**, las **variables exógenas** se refieren a aquellas variables que no son influenciadas por el sistema o modelo en cuestión, pero que pueden tener un impacto significativo en el comportamiento de las variables dependientes. Estas variables son externas al modelo que se está analizando, pero pueden influir en los resultados de los datos de manera indirecta o directa.

Por ejemplo, en un análisis de predicción del precio de una acción, las variables exógenas pueden incluir la tasa de interés, políticas gubernamentales, o eventos económicos externos que no forman parte del modelo de precios de las acciones pero que influyen en su comportamiento.

¿Cómo Utilizar el Web Scraping para Obtener Variables Exógenas?

El web scraping es una herramienta clave para la extracción de variables exógenas, ya que permite acceder a fuentes de datos externas y obtener información de sitios web relevantes que de otra forma sería difícil obtener. Algunos ejemplos de cómo las variables exógenas pueden ser recolectadas mediante scraping incluyen:

- **Datos de Clima:** Si estás analizando el impacto de las condiciones climáticas en la agricultura, podrías hacer scraping de sitios web meteorológicos para obtener datos sobre la temperatura, la humedad, la precipitación, etc.
- **Indicadores Económicos:** Puedes hacer scraping de portales económicos para obtener datos sobre tasas de inflación, crecimiento económico o el valor de divisas, que pueden influir en los modelos de predicción económica.
- **Redes Sociales y Sentimientos Públicos:** Al realizar scraping de redes sociales, puedes recolectar datos sobre opiniones de usuarios acerca de un producto o evento, lo que puede servir como una variable exógena en el análisis de la percepción pública.

Ejemplo: Imagina que deseas predecir las ventas de un producto. Puedes hacer scraping de reseñas de productos en sitios como Amazon, eBay o foros de usuarios para obtener sentimientos y opiniones, los cuales podrían influir en las decisiones de compra. Estas opiniones se considerarían variables exógenas, ya que no están directamente relacionadas con las ventas, pero impactan en ellas.

Consideraciones Éticas en el Web Scraping

Aunque el web scraping es una técnica poderosa y útil, también conlleva responsabilidades legales y éticas que deben ser consideradas cuidadosamente. Es importante que cualquier persona que haga scraping lo haga respetando las reglas de los sitios web y las leyes en vigor. A continuación se detallan algunas de las principales consideraciones éticas y legales:

Respetando las Políticas de los Sitios Web y las Leyes

- **Adherencia a los Términos de Servicio (ToS):** Cada sitio web tiene su propio conjunto de reglas que rigen el uso de sus servicios. Estos términos se encuentran generalmente en los **Términos de Servicio (ToS)** o en las **Políticas de Privacidad** del sitio. Es importante revisar estos términos antes de realizar scraping, ya que algunos sitios prohíben explícitamente esta práctica.

Ejemplo: Si deseas hacer scraping de un sitio como Amazon para obtener datos de productos, debes asegurarte de que sus Términos de Servicio permitan este tipo de acceso. Amazon tiene medidas estrictas contra el scraping y proporciona su propia API para la recopilación de datos.

- **Cumplimiento de las Leyes de Derechos de Autor:** Los datos que haces scraping pueden estar protegidos por derechos de autor. El uso indebido de estos datos puede resultar en demandas legales. Es esencial asegurarse de que el uso de los datos raspados sea compatible con las leyes de propiedad intelectual.

Ejemplo: Hacer scraping de contenido de un sitio web que publica artículos o investigaciones científicas sin el permiso adecuado puede violar las leyes de derechos de autor. En lugar de hacer scraping directamente, una buena práctica es utilizar APIs proporcionadas por los sitios web que ofrezcan acceso legal a los datos.

- **Preocupaciones de Privacidad:** Hacer scraping de datos personales sin el consentimiento explícito de las personas afectadas puede violar las leyes de privacidad como el GDPR en Europa o la CCPA en California. Los datos personales, como nombres, direcciones o correos electrónicos, deben manejarse con especial cuidado.

Ejemplo: Si un scraper obtiene datos de perfiles de usuarios en redes sociales (como información personal o mensajes privados), puede estar violando regulaciones de privacidad. Es crucial evitar este tipo de prácticas y centrarse solo en datos que no comprometan la privacidad.

Ejemplo: Entendiendo el robots.txt de Google

El archivo **robots.txt** es un archivo de texto que los administradores de sitios web usan para dar instrucciones a los motores de búsqueda y otros rastreadores web sobre qué partes de su sitio pueden ser accedidas. Es importante respetar las directrices contenidas en estos archivos.

Implicaciones del robots.txt de Google

- **Acceso Selectivo:** Google, por ejemplo, permite que sus motores de búsqueda rastreen ciertos contenidos, pero limita el acceso a otras secciones del sitio. Esto puede incluir áreas con datos sensibles, dinámicos o que generen una carga significativa en el servidor.

Ejemplo: El archivo **robots.txt** de Google prohíbe que los rastreadores accedan a ciertas páginas, como las de resultados de búsqueda, para evitar que los motores de búsqueda sobrecarguen sus servidores con peticiones innecesarias.

- **Naturaleza Dinámica:** El archivo `robots.txt` puede cambiar con el tiempo. Esto refleja las decisiones del propietario del sitio sobre qué permitir o restringir en función del comportamiento de los usuarios y las cargas del servidor.

Ejemplo: Si un sitio web observa un aumento en el tráfico de scraping, puede actualizar su archivo `robots.txt` para bloquear el acceso a determinadas páginas, limitando así el scraping de contenido sensible.

- **Respetando los Límites:** Aunque un archivo `robots.txt` permita el acceso a ciertas áreas de un sitio, esto no significa que todo tipo de scraping sea aceptable. Algunas actividades pueden ser consideradas ilegales o contrarias a la ética, incluso si están permitidas por el archivo `robots.txt`.

Ejemplo: Un sitio puede permitir el acceso a su blog, pero si el scraper está haciendo peticiones demasiado frecuentes o está extrayendo datos en gran cantidad, puede afectar negativamente el rendimiento del sitio, lo que podría ser un comportamiento éticamente cuestionable.

Conclusión

El **web scraping** es una herramienta poderosa para obtener datos de la web, pero debe ser utilizado con responsabilidad. Es crucial conocer las políticas de los sitios web, respetar las leyes de propiedad intelectual y privacidad, y actuar con ética en todo momento. Al seguir estas pautas, los que hagan scraping pueden extraer datos de manera efectiva y respetuosa, contribuyendo al desarrollo de soluciones basadas en datos que beneficien tanto a las empresas como a los usuarios.

Política de `robots.txt` y Web Scraping

El archivo `robots.txt` es fundamental para la administración de sitios web y la ética del **web scraping**. Se utiliza para indicar a los motores de búsqueda y robots de scraping qué partes del sitio pueden ser accedidas y rastreadas, y cuáles deben ser evitadas. Aunque no es una herramienta de seguridad, cumple un rol esencial en la organización y el control de cómo los bots interactúan con un sitio web. Es importante comprender cómo funciona, ya que respeta los límites éticos y legales del scraping.

1. ¿Qué es `robots.txt`?

El archivo `robots.txt` es un archivo de texto ubicado en el directorio raíz de un sitio web. Este archivo sigue el estándar del **Protocolo de Exclusión de Robots (REP)**, un acuerdo voluntario entre los administradores de sitios web y los desarrolladores de bots que permite regular el acceso automatizado a los sitios. El principal objetivo de este archivo es evitar la sobrecarga de los servidores web, impedir el acceso no deseado a contenido privado o restringido, y garantizar la correcta indexación del sitio por parte de los motores de búsqueda.

- **Propósito principal:** Limitar o permitir el acceso a determinadas secciones del sitio según el tipo de bot, ya sea un motor de búsqueda como Googlebot o un scraper.
- **Ubicación estándar:** Se debe colocar en la raíz del servidor web. Por ejemplo, `https://www.ejemplo.com/robots.txt`.

- **No es vinculante:** Aunque los bots respetan las directrices del archivo `robots.txt`, no es una herramienta de seguridad. Los bots malintencionados pueden ignorarlo, por lo que no debe usarse como la única barrera de protección.

2. Sintaxis y Directivas Comunes

El archivo `robots.txt` utiliza una sintaxis sencilla, pero debe ser configurado correctamente para garantizar que las instrucciones sean seguidas por los robots. A continuación, se describen las directivas más comunes y su uso:

- **User-agent:** Esta directiva especifica el bot o robot al que se aplican las reglas. Se puede usar para dirigir reglas específicas a diferentes tipos de robots. Cada motor de búsqueda o bot tiene un nombre de agente de usuario único que identifica su actividad.

Ejemplo:

```
User-agent: Googlebot
```

En este caso, las reglas siguientes solo se aplicarán al bot de Google.

- **Disallow:** Esta directiva se usa para bloquear el acceso a una o más URLs o directorios. Los bots que sigan el protocolo REP evitarán rastrear estas rutas.

Ejemplo:

```
Disallow: /privado/
```

Aquí, cualquier bot que siga las reglas de `robots.txt` evitará acceder a cualquier página dentro del directorio `/privado/`.

- **Allow:** Es lo contrario de `Disallow`. Permite a los bots acceder a una URL específica incluso si se encuentra dentro de una ruta bloqueada por `Disallow`. Esto es útil cuando deseas permitir el acceso a subpáginas dentro de un directorio restringido.

Ejemplo:

```
Disallow: /productos/  
Allow: /productos/nuevo-producto
```

En este ejemplo, el directorio `/productos/` está bloqueado, pero la página `/productos/nuevo-producto` está permitida para los robots.

- **Sitemap:** Esta directiva es utilizada para indicar la ubicación del **sitemap** del sitio, lo cual ayuda a los motores de búsqueda a encontrar e indexar más eficientemente todas las páginas relevantes del sitio.

Ejemplo:

Sitemap: <https://www.ejemplo.com/sitemap.xml>

Aquí, se le indica a los robots que el mapa del sitio se encuentra en <https://www.ejemplo.com/sitemap.xml>. Esto mejora la visibilidad del sitio y asegura que los bots rastreen y indexen correctamente todo el contenido relevante.

3. Consideraciones Éticas en el Uso de robots.txt

Aunque el archivo robots.txt proporciona una forma de gestionar el comportamiento de los bots, no es una solución perfecta para todas las situaciones. Existen varias consideraciones éticas que los desarrolladores y usuarios de web scraping deben tener en cuenta al hacer uso de este archivo:

- **Cumplimiento Voluntario:** Los bots que siguen el Protocolo de Exclusión de Robots (REP) respetan las reglas del archivo robots.txt. Sin embargo, algunos bots malintencionados pueden ignorarlo, lo que significa que este archivo no garantiza que los datos no sean extraídos sin permiso.

Ejemplo: Si un scraper no respetara el archivo robots.txt de un sitio y accediera a contenido restringido, esto podría considerarse una violación de las reglas del sitio, aunque técnicamente no sea ilegal. Es importante evaluar las implicaciones legales y éticas de hacer scraping sin tener en cuenta el robots.txt.

- **Protección de la Propiedad Intelectual:** Algunos sitios utilizan robots.txt para proteger contenido sensible o con derechos de autor. Hacer scraping de este contenido, incluso si el archivo lo permite, puede violar las leyes de propiedad intelectual. Es esencial que los scrapers respeten las leyes sobre derechos de autor y se abstengan de extraer datos que están protegidos legalmente.

Ejemplo: Si un sitio web indica en su archivo robots.txt que el contenido de un artículo está bloqueado para bots, pero un scraper aún accede al contenido para obtener los datos, podría estar infringiendo las leyes de derechos de autor.

- **Interferencia con el Rendimiento del Sitio:** Aunque el archivo robots.txt puede permitir el acceso a ciertas áreas del sitio, los scrapers deben ser cuidadosos al realizar solicitudes a los servidores para no sobrecargar el sitio web. La ejecución excesiva de scraping puede llevar a una disminución del rendimiento del servidor, lo que afecta la experiencia de los usuarios humanos.

Ejemplo: Un scraper que realice demasiadas solicitudes en poco tiempo puede causar que un servidor se vuelva inestable o incluso caiga. Este comportamiento no solo afecta al sitio web en cuestión, sino que también puede violar las mejores prácticas y normas éticas.

4. Cómo Leer y Usar el Archivo robots.txt

Para realizar un scraping respetuoso con las normas del archivo robots.txt, es necesario entender cómo leer y aplicar las reglas que este archivo contiene. A continuación, se explican algunos pasos básicos para hacerlo:

- **Acceder al archivo:** Puedes acceder al archivo `robots.txt` de cualquier sitio web añadiendo `/robots.txt` a la URL base del dominio.

Ejemplo:

- `https://www.ejemplo.com/robots.txt`
- `https://www.amazon.com/robots.txt`
- `https://www.google.com/robots.txt`

Una vez que hayas accedido al archivo, revisa las directivas de `User-agent`, `Disallow`, `Allow` y `Sitemap` para entender qué áreas del sitio están permitidas o bloqueadas.

- **Configurar tu bot para seguir las reglas:** Si estás desarrollando un scraper, asegúrate de que tu bot respete las reglas establecidas en el archivo `robots.txt`. Esto es fundamental para no causar problemas legales o éticos, y para evitar que tu scraper sea bloqueado por el sitio.

5. Casos Comunes de `robots.txt` y Web Scraping

- **Google y otros motores de búsqueda:** Los motores de búsqueda generalmente permiten el acceso a las páginas de su sitio web para ser indexadas, pero bloquean áreas como resultados de búsqueda o contenido duplicado. Google, por ejemplo, tiene un archivo `robots.txt` muy estricto que prohíbe el acceso a sus resultados de búsqueda para evitar la indexación de contenido no útil.
- **E-commerce:** Muchos sitios de comercio electrónico usan `robots.txt` para bloquear el scraping de sus páginas de productos o resultados de búsqueda. Sin embargo, algunos permiten el scraping de páginas de productos específicos si se configuran correctamente las reglas.
- **Redes sociales:** Las redes sociales a menudo limitan el acceso de los bots a sus plataformas mediante `robots.txt`. Sin embargo, también ofrecen APIs públicas que permiten obtener datos de forma legal y ética sin tener que recurrir al scraping directo de sus sitios web.

6. Herramientas y Buenas Prácticas para Scraping Ético

Para realizar un scraping ético, se recomienda utilizar herramientas y buenas prácticas que respeten tanto las leyes como las políticas del sitio web. Aquí algunos consejos:

- **Revisar el archivo `robots.txt`** antes de comenzar a hacer scraping de cualquier sitio.
- **Uso de APIs oficiales:** Si un sitio ofrece una API oficial, utilízala en lugar de hacer scraping directo. Las APIs están diseñadas para permitir el acceso a los datos de forma controlada y segura.
- **Control de la frecuencia de solicitudes:** Para evitar sobrecargar los servidores, configura tu scraper para hacer solicitudes a un ritmo razonable (por ejemplo, no más de una solicitud por segundo).
- **Ser transparente:** Si es posible, identifica tu bot con un agente de usuario claro y responsable.

1. Introducción a la Búsqueda de Datos en la Era Digital

La Evolución de la Obtención de Datos

En este curso, nos centramos en los datos como nuestro elemento fundamental. Tradicionalmente, los datos se han obtenido de formatos estructurados como hojas de cálculo de experimentos científicos o registros en bases de datos relacionales dentro de las organizaciones. Estos datos estructurados permiten un análisis eficiente utilizando herramientas tradicionales como SQL, Excel y otros software de análisis de datos. Sin embargo, con la revolución digital, particularmente el advenimiento de internet, nuestro enfoque para la recopilación de datos debe evolucionar. Internet se ha convertido en un vasto reservorio de datos no estructurados, lo que plantea tanto desafíos como oportunidades para la recuperación, el análisis y la interpretación de estos datos.

- **Datos Estructurados vs No Estructurados:** Los datos estructurados son fáciles de manejar ya que se presentan en un formato predefinido, como filas y columnas en una base de datos. Los datos no estructurados, en cambio, pueden ser textos, imágenes, videos o cualquier tipo de información que no siga un formato ordenado, y requieren técnicas más complejas para su recopilación y análisis.
- **Desafíos del Big Data:** El análisis de grandes volúmenes de datos no estructurados requiere herramientas avanzadas como el web scraping, que facilita la recolección de datos directamente desde páginas web.

Entendiendo el Paisaje de los Datos Web

Al buscar datos en internet, es esencial considerar primero cómo el sitio web en cuestión proporciona acceso a sus datos. La mayoría de los sitios web hoy en día gestionan vastos volúmenes de información. Afortunadamente, muchos de ellos ofrecen una **Interfaz de Programación de Aplicaciones (API)** para facilitar el acceso a sus datos.

- **¿Qué es una API?:** Las API permiten a los desarrolladores interactuar de manera programática con un servicio web, extrayendo datos sin necesidad de realizar scraping. Usualmente, las API proporcionan datos en formatos estructurados como JSON o XML, lo que facilita el procesamiento posterior.
- **Ejemplos de Uso de API:** Google, Facebook y Twitter ofrecen API que permiten acceder a sus datos de manera ordenada. Por ejemplo, la API de Twitter permite obtener tweets recientes, tendencias, y detalles de usuarios sin necesidad de navegar por la web.

El Papel de las API

Las API son fundamentales en la recuperación eficiente de datos desde sitios web grandes y populares. Sirven como la herramienta primaria para obtener datos de manera estructurada y controlada.

- **APIs como Herramienta Primaria:** Una API actúa como un puente entre el buscador de datos y la base de datos del sitio web, permitiendo una recuperación de datos simplificada. Por ejemplo, utilizando la API de OpenWeatherMap, se pueden obtener datos meteorológicos actualizados de cualquier lugar del mundo sin necesidad de hacer scraping en su página web.
- **Ventajas:** Las API están diseñadas para proporcionar acceso a grandes volúmenes de datos de manera rápida, eficiente y sin necesidad de preocuparse por los detalles de la estructura del sitio web o sus cambios. Además, las API suelen ofrecer datos más completos y actualizados en comparación con lo que está disponible en la interfaz pública del sitio web.
- **Limitaciones:** Sin embargo, no todos los sitios web proporcionan una API. Algunos sitios grandes, como Amazon o Craigslist, no ofrecen APIs públicas o limitan fuertemente el acceso. Incluso cuando una API está disponible, puede no conceder acceso a todos los datos que un usuario podría necesitar. Por ejemplo, las APIs de Google Maps tienen límites en la cantidad de peticiones que se pueden hacer, y no todos los datos de ubicación están disponibles.

La Necesidad del Web Scraping

En casos donde una API está ausente o no concede el acceso completo a los datos necesarios, recurrimos al **web scraping**. El web scraping implica extraer datos directamente del frontend de un sitio web, es decir, la misma información que se muestra a los usuarios en sus navegadores. Esto permite a los scrapers acceder a datos que no están disponibles de otra forma, como los precios de productos en e-commerce, comentarios de usuarios o información detallada de artículos.

- **Desafíos del Web Scraping:** El scraping a menudo implica trabajar con datos no estructurados que están incrustados en el código HTML de una página web. Esto significa que se debe analizar cuidadosamente el HTML para extraer los datos relevantes, lo que requiere conocimientos técnicos de programación y un enfoque meticuloso.
- **Estrategias para Scraping Eficiente:** Es importante desarrollar scrapers que puedan adaptarse a los cambios en el diseño y estructura de los sitios web, ya que los sitios a menudo actualizan su código. Las herramientas como BeautifulSoup en Python o Scrapy permiten navegar a través de las páginas HTML, buscar y extraer información específica.

Sumergiéndose en el Scraping

El scraping es una habilidad avanzada que exige tener en cuenta varios aspectos técnicos y éticos. Para extraer datos de manera efectiva, es fundamental conocer las herramientas, las técnicas de programación y las buenas prácticas para hacerlo de manera ética.

- **Tratando con Datos No Estructurados:** El scraping nos obliga a interactuar con datos no estructurados, lo que implica manejar HTML crudo, CSS y JavaScript. En lugar de trabajar con bases de datos organizadas, se debe diseñar un código que extraiga la información que se necesita del código fuente de la página.
- **Técnicas de Scraping:** Existen diversas técnicas de scraping, como la extracción de datos de tablas HTML, la lectura de metadatos o incluso la simulación de clics para

navegar a través de formularios. Sin embargo, se deben manejar adecuadamente las solicitudes de red para evitar problemas de sobrecarga en los servidores.

- **Consideraciones Legales y Éticas:** Es crucial abordar el web scraping con conciencia de las implicaciones legales y éticas, respetando las políticas del sitio web y la privacidad del usuario. Por ejemplo, aunque un sitio web no bloquee el acceso a sus datos, el scraping excesivo puede violar los términos de servicio del sitio o incluso las leyes de propiedad intelectual.
 - **Ejemplo Ético:** Si se está realizando scraping de datos de productos para un análisis de precios, es importante asegurarse de que no se están violando las restricciones impuestas por el sitio web en su archivo `robots.txt`, y que no se está causando una sobrecarga innecesaria en el servidor.
-

Comenzando Nuestro Viaje

Nuestro primer paso práctico en este viaje será explorar cómo conectarnos a internet y recuperar una página web básica. Comenzaremos utilizando el módulo `urllib.request` de Python, una herramienta poderosa para interactuar con URLs y manejar solicitudes web.

La habilidad de recuperar páginas web es uno de los primeros pasos fundamentales en la práctica del **web scraping**. Esta es una actividad clave, ya que una vez que somos capaces de conectar con un sitio web y recuperar su contenido, podemos empezar a analizarlo y extraer datos útiles de él.

Únete a nosotros mientras nos embarcamos en este emocionante viaje para dominar el arte de la búsqueda de datos en la era digital, donde navegaremos por las complejidades de las API, el web scraping y las consideraciones éticas que vienen con ellas.

Vamos a comenzar con un ejemplo básico utilizando el módulo `urllib.request` de Python para abrir una página web y obtener su contenido.

```
# Importar la función 'urlopen' del módulo 'urllib.request'.
# La función urlopen permite abrir una URL en Python.
# La utilización de esta función es crucial para iniciar el proceso de
# scraping, ya que nos permite acceder al contenido de la web.
from urllib.request import urlopen

# Usar la función 'urlopen' para abrir una URL.
# La URL proporcionada puede ser cualquier página web accesible. En
# este ejemplo, estamos utilizando la página de Mitsubishi Electric.
# El resultado es un objeto que contiene información sobre la página
# web, como el código HTML, los encabezados HTTP y otros metadatos.
source = urlopen("https://www.mitsubishielectric.es/aire-
acondicionado/")
```

```
# Imprimir el objeto de respuesta HTTP.  
# Es importante notar que esto no muestra el contenido HTML  
# directamente, sino una representación del objeto de respuesta HTTP.  
# Este objeto de respuesta contiene metadatos como el estado de la  
# solicitud, las cabeceras HTTP y más.  
print(source)  
  
<http.client.HTTPResponse object at 0x7f6d25791cc0>
```

Este fragmento de código demuestra cómo usar la función `urlopen` de `urllib.request` para abrir una URL y obtener un objeto de respuesta. Sin embargo, **es importante entender que `print(source)` no mostrará el contenido HTML de la página**. Solo mostrará una representación del objeto de respuesta HTTP, lo que incluye detalles como los códigos de estado HTTP (por ejemplo, 200 para éxito, 404 para página no encontrada), las cabeceras de respuesta, entre otros.

Si deseas obtener el contenido real de la página web (el código HTML que define la estructura visual del sitio), debemos leer el contenido de la respuesta utilizando el método `.read()`.

Explorando el Contenido Recuperado por `urlopen`

Tras abrir una URL utilizando la función `urlopen`, el siguiente paso lógico es obtener el **contenido real** de la página web. Como mencionamos antes, `urlopen` devuelve un objeto de respuesta que contiene no solo los datos de la página sino también información adicional como el estado HTTP, los encabezados, entre otros.

¿Qué es `source.read()`?

Cuando se utiliza `urlopen`, la función devuelve un objeto `HTTPResponse`. Este objeto contiene la información que el servidor nos ha enviado al realizar la solicitud HTTP. Para acceder al contenido HTML de la página web que nos interesa, necesitamos usar el método `.read()` del objeto `HTTPResponse`. Este método devuelve todo el contenido de la página web en formato binario.

¿Qué Hace `source.read()`?

- **Recupera el Contenido de la Página Web:** El método `.read()` obtiene todos los datos que el servidor nos ha enviado. En este caso, los datos corresponden al código HTML de la página solicitada, que es el lenguaje utilizado para estructurar la información visible en la web.
- **Formato Binario:** El contenido que obtenemos con `.read()` es en formato binario. Esto significa que los datos no son directamente legibles como texto. Para poder trabajar con ellos como una cadena de texto en Python, necesitamos **decodificarlos** utilizando un método como `.decode('utf-8')`, que convierte los bytes binarios en una cadena de texto legible.

- **Operación de Única Vez:** Es importante destacar que, una vez que llamas a `source.read()`, el contenido solo se puede leer una vez. Después de que el contenido haya sido leído, el objeto `source` no retendrá la información de forma accesible. Si quieres acceder al contenido de nuevo, tendrás que volver a abrir la URL.

Ejemplo de Uso de `source.read()`

Para ilustrar cómo trabajar con el contenido recuperado de una página web, primero vamos a leer el contenido utilizando `.read()` y luego decodificaremos ese contenido para poder trabajar con él.

```
# Leer el contenido de la respuesta HTTP.  
# Al llamar a source.read(), obtenemos el contenido crudo en formato binario.  
something = source.read()  
  
# Imprimir el contenido crudo de la página.  
# Este contenido está en bytes, lo que significa que no será legible como texto en la consola aún.  
# Imprimir esto directamente mostrará una secuencia de bytes que representa el código HTML de la página web.  
# print(something)
```

Decodificando el Contenido

El contenido que hemos obtenido hasta ahora está en formato binario (bytes). Si queremos trabajar con el contenido como texto legible, debemos decodificarlo utilizando el método `.decode('utf-8')`, que convierte los bytes en una cadena de texto en el formato UTF-8, el cual es el formato de codificación estándar para las páginas web modernas.

```
# Decodificar el contenido binario a una cadena de texto.  
# Al usar .decode('utf-8'), transformamos los datos binarios en texto legible en formato UTF-8.  
decoded_content = something.decode('utf-8')  
  
# Imprimir el contenido decodificado.  
# Ahora el contenido es una cadena de texto, y se puede visualizar como el código HTML de la página web.  
# print(decoded_content)
```

Recapitulación del Uso de `source.read()`

- **Lectura de una sola vez:** Una vez que se llama a `source.read()`, el contenido de la respuesta HTTP se consume y no estará disponible para ser leído nuevamente a menos que reabran la URL. Si necesitas acceder al contenido de nuevo, debes realizar otra solicitud con `urlopen`.

- **Decodificación de Datos Binarios:** Como los datos que recibimos son binarios, debemos decodificarlos para trabajar con ellos como texto. El método `.decode('utf-8')` es ideal para convertir estos datos en una cadena que Python pueda manejar fácilmente.
- **Limitaciones de `urlopen`:** La función `urlopen` es útil para obtener el contenido crudo de una página web, pero no proporciona herramientas para analizar el HTML. Si quieres extraer datos específicos de la página, como títulos, enlaces o tablas, necesitarás herramientas adicionales como BeautifulSoup, lxml o Regex.

¿Por Qué Utilizamos `urllib.request`?

- **Facilidad de Uso:** `urllib.request` es parte de la biblioteca estándar de Python, lo que significa que no necesitas instalar dependencias adicionales. Esto lo convierte en una opción conveniente para tareas simples de recuperación de contenido web.
- **Operación de Bajo Nivel:** Aunque `urllib.request` es una herramienta simple y accesible, también permite un control más detallado de las solicitudes HTTP, lo que te da flexibilidad si necesitas ajustar las cabeceras de las solicitudes, manejar redirecciones o configurar proxies.

Sin embargo, cuando se trata de tareas de scraping más complejas, como extraer datos estructurados (por ejemplo, titulares, tablas, enlaces), **librerías adicionales** como BeautifulSoup o lxml se convierten en herramientas esenciales, ya que permiten hacer un análisis más estructurado del contenido HTML.

En resumen, `urlopen` es útil para obtener contenido web básico. Sin embargo, para realizar análisis más avanzados o extraer datos específicos de una página web, necesitarás herramientas adicionales de parsing como BeautifulSoup o lxml.

Ejercicios de Calentamiento

¡Manos a la obra con algunos ejercicios iniciales para calentar con el web scraping!

Ejercicios

1. **Verificación de Contenido en Python.org:** ¿Contiene <https://www.python.org> la palabra Python?
Pista: Puedes usar la palabra clave `in` para verificar.
2. **Búsqueda de Imagen en Google.com:** ¿Contiene <http://google.com> una imagen?
Pista: Busca la etiqueta ``.
3. **Primeros Caracteres de Python.org:** ¿Cuáles son los primeros diez caracteres de <https://www.python.org>?

4. Verificación de Palabra Clave en Pyladies.com: ¿Está la palabra 'python' en <https://pyladies.com?>

```
# EJ1: Verificar si la palabra 'Python' está en el contenido de
http://www.python.org/

# Importar la función urlopen del módulo urllib.request
# Esta función se usa para abrir una URL y recuperar su contenido
from urllib.request import urlopen

# Usar la función urlopen para acceder a la página web en
http://www.python.org/
# La función devuelve un objeto HTTPResponse que almacenamos en la
variable 'source'
source = urlopen("http://www.python.org/")

# Leer el contenido del objeto de respuesta usando el método read()
# El método read() recupera el contenido de la página web en formato
binario
# El contenido binario se decodifica a una cadena usando la
codificación 'latin-1'
# La cadena decodificada se almacena en la variable 'something'
something = source.read().decode('latin-1')

# Verificar si la palabra "Python" está en la cadena decodificada
# Esto se hace usando la palabra clave 'in', que verifica la presencia
de un substring en una cadena
# El resultado será un valor booleano: True si "Python" se encuentra,
False si no.
"Python" in something

# Nota: La elección de 'latin-1' para la decodificación no siempre es
la más adecuada
# Normalmente es mejor usar 'utf-8', que es una codificación más común
para páginas web
# Por ejemplo: something = source.read().decode('utf-8')

True

# EJ2: Verificar si 'https://www.google.com/' contiene una etiqueta de
imagen ("<img>")

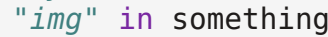
# Importar la función urlopen del módulo urllib.request.
# Esta función se usa para abrir una URL y recuperar su contenido.
from urllib.request import urlopen

# Usar la función urlopen para abrir la página web en
'https://www.google.com/'.
# La función devuelve un objeto HTTPResponse, que almacenamos en la
variable 'source'.
```

```

source = urlopen("https://www.google.com/")

# Leer el contenido del objeto de respuesta usando el método read().
# El método read() recupera el contenido de la página web en formato binario.
# Después de leer, el contenido está en bytes, lo cual no es legible para los humanos.
# Luego decodificamos este contenido binario en una cadena usando la codificación 'latin-1'.
# La cadena resultante, que contiene el HTML de la página, se almacena en 'something'.
something = source.read().decode('latin-1')

# Verificar si la cadena "img" está en el HTML decodificado.
# Esta es una manera sencilla de verificar si hay una etiqueta <img> en el HTML,
# ya que "img" es parte de la etiqueta estándar de HTML para imágenes.
# El resultado será True si se encuentra "img" (lo que indica la presencia de una imagen),
# y False si no se encuentra.

"img" in something

# Nota: Decodificar con 'latin-1' podría no ser adecuado para todos los sitios web,
# especialmente si el sitio web usa un conjunto de caracteres diferente.
# 'utf-8' es una codificación más común y suele ser una mejor opción.
# Por ejemplo: something = source.read().decode('utf-8')

True

# EJ3: Verificar los primeros 10 caracteres del contenido de
http://www.python.org/

# Importar la función urlopen del módulo urllib.request
# Esta función se usa para abrir una URL y recuperar su contenido
from urllib.request import urlopen

# Usar la función urlopen para acceder a la página web en
https://www.python.org/
# La función devuelve un objeto HTTPResponse que almacenamos en la variable 'source'
source = urlopen("https://www.python.org/")

# Leer el contenido del objeto de respuesta usando el método read()
# El método read() recupera el contenido de la página web en formato binario
# El contenido binario se decodifica a una cadena usando la codificación 'utf-8'
# La cadena decodificada se almacena en la variable 'content'

```



```

content = source.read().decode()

# Extraer los primeros 10 caracteres del contenido decodificado
first_ten_characters = content[:10] + "$"

# Imprimir los primeros 10 caracteres
print(first_ten_characters)

<!doctype $

# EJ4: Verificar si 'Python' está en el contenido de
https://pyladies.com?

# Importar la función urlopen del módulo urllib.request
# Esta función se usa para abrir una URL y recuperar su contenido
from urllib.request import urlopen

# Usar la función urlopen para acceder a la página web en
https://pyladies.com/
# La función devuelve un objeto HTTPResponse que almacenamos en la
variable 'source'
source = urlopen("https://pyladies.com/")

# Leer el contenido del objeto de respuesta usando el método read()
# El método read() recupera el contenido de la página web en formato
binario
# El contenido binario se decodifica a una cadena usando la
codificación 'utf-8'
# La cadena decodificada se almacena en la variable 'content'
content = source.read().decode('utf-8')

# Verificar si la palabra "python" está en la cadena decodificada
# Esto se hace usando la palabra clave 'in', que verifica la presencia
de un substring en una cadena
# El resultado será un valor booleano: True si "python" se encuentra,
False si no.
"python" in content.lower()

True

```

Usando `urlopen` vs. `Request` en Web Scraping

Al realizar tareas de web scraping en Python, tienes la opción de usar la función `urlopen` del módulo `urllib.request` o el objeto `Request` en combinación con `urlopen`. Aquí explicaremos por qué podrías elegir un enfoque sobre el otro.

Usando `urlopen` Directamente

Ventajas:

- **Simplicidad:** Es una forma directa de acceder a una página web y recuperar su contenido sin la necesidad de objetos adicionales o personalización. Esta opción es adecuada cuando lo que necesitas es una solución rápida y sencilla para obtener los datos de una página web.
- **Comportamiento Predeterminado:** `urlopen` utiliza la configuración predeterminada para la solicitud HTTP, lo cual es adecuado para muchos casos de uso comunes. Si solo necesitas enviar una solicitud GET y obtener el contenido, `urlopen` hace bien este trabajo sin complicaciones.
- **Conveniencia:** Para tareas simples de web scraping, proporciona una solución concisa y legible, ya que no necesitas preocuparte por los detalles técnicos que podrían complicar el código.

Limitaciones:

- **Falta de personalización:** `urlopen` no te permite configurar encabezados personalizados o trabajar con otros métodos HTTP (como POST, PUT). Esto puede ser un inconveniente cuando los sitios web requieren configuraciones específicas para acceder a sus datos.

Usando `Request` con `urlopen`

Ventajas:

- **Personalización:** Puedes establecer encabezados personalizados, utilizar diferentes métodos HTTP (por ejemplo, POST, PUT) y configurar opciones avanzadas como manejar redirecciones, cookies y tiempos de espera. Esto es útil cuando un sitio web requiere detalles específicos en la solicitud, como un agente de usuario (User-Agent) o credenciales de autenticación.
- **Control Detallado:** Ofrece mayor flexibilidad para manejar escenarios complejos, como enviar formularios, gestionar sesiones y realizar solicitudes con diferentes tipos de autenticación. Si el sitio web utiliza medidas de seguridad (como un sistema de cookies o autenticación), `Request` es más adecuado.
- **Compatibilidad con servicios complejos:** Si necesitas interactuar con una API o realizar solicitudes que no sean simples GET, como solicitudes POST con datos de formularios, `Request` es el mejor enfoque.

Limitaciones:

- **Mayor complejidad:** Configurar `Request` puede ser más complejo y, por lo tanto, menos adecuado para tareas muy simples. Requiere una mayor comprensión de las solicitudes HTTP y cómo funcionan.

Resumen de Cuándo Elegir Cada Método

En resumen, la elección entre usar `urlopen` directamente y crear un objeto `Request` depende de la complejidad de tu tarea de web scraping:

- **Usa `urlopen`:** Si solo necesitas obtener el contenido de una página web sin necesidad de personalización, como acceder a una página estática simple.
- **Usa `Request` con `urlopen`:** Si necesitas controlar aspectos más detallados de la solicitud HTTP, como personalizar encabezados (por ejemplo, establecer un `User-Agent` específico), realizar métodos POST, gestionar cookies, manejar redirecciones, o autenticarte.

La clave está en elegir el enfoque adecuado según las necesidades de tu proyecto. Para tareas simples y directas, `urlopen` es más rápido y fácil. Sin embargo, si te enfrentas a sitios web más complejos o APIs que requieren configuraciones avanzadas, `Request` es la herramienta más poderosa.

```
# Solución al ejercicio 4
# Importa el módulo urllib y la submódulo request
import urllib.request

# Define la URL del sitio web que se va a analizar
url = 'https://www.pyldadies.com'

# Crea una solicitud HTTP con una cabecera de 'User-Agent'
# La cabecera 'User-Agent' se utiliza para identificar el agente de
# usuario que realiza la solicitud
# 'Magic Browser' es una cadena personalizada que estamos usando como
# ejemplo de User-Agent
# Esta cadena puede ser cualquier cosa, pero típicamente se usa para
# hacerse pasar por navegadores conocidos
# Algunos servidores web usan el User-Agent para permitir o denegar el
# acceso, o para servir contenido específico a diferentes tipos de
# dispositivos
req = urllib.request.Request(url, headers={'User-Agent': 'Magic
Browser'})

# Abre la URL utilizando la solicitud creada y almacena la respuesta
# en la variable 'con'
con = urllib.request.urlopen(req)

# Lee el contenido de la respuesta y decodifica el contenido de bytes
# a una cadena
# La decodificación se realiza utilizando la codificación
# predeterminada (UTF-8)
html = con.read().decode()

# Verifica si la palabra "Python" está en el contenido de la página
# web
# La verificación es sensible a mayúsculas y minúsculas
```

```
print('Python' in html)
print('python' in html)
```

True
False

Explicación Detallada del Código:

1. **Importación del Módulo:** Se importa el módulo `urllib.request`, que contiene las funciones necesarias para realizar solicitudes HTTP en Python.
2. **Definición de la URL:** Se define la URL que se va a analizar, en este caso, el sitio web de Pyladies.
3. **Creación de la Solicitud HTTP:** Usamos `urllib.request.Request` para crear una solicitud HTTP personalizada. El parámetro `headers` nos permite incluir un encabezado `User-Agent` personalizado, que es útil cuando se quiere simular el comportamiento de un navegador web.
4. **Abrir la URL:** `urllib.request.urlopen` se usa para abrir la URL proporcionada, y la respuesta de la solicitud se almacena en la variable `con`.
5. **Leer y Decodificar el Contenido:** Usamos `con.read()` para obtener el contenido de la respuesta en formato binario. Luego, decodificamos ese contenido a texto usando `.decode()` y lo almacenamos en la variable `html`.
6. **Verificación de la Palabra:** Finalmente, verificamos si la palabra "Python" aparece en el contenido HTML decodificado. La comprobación es sensible a mayúsculas y minúsculas, por lo que se realiza de forma directa.

Consideraciones Adicionales:

- El uso del `User-Agent` es importante porque algunos sitios web bloquean el acceso si detectan que la solicitud no proviene de un navegador real.
 - Los sitios web pueden ofrecer diferentes contenidos según el `User-Agent` que se les envíe, lo que puede ser útil para obtener datos específicos de versiones móviles o de escritorio.
-

Crawling y Scraping: Revelando los Secretos de la Web

El rastreo y el scraping son dos técnicas fundamentales en el mundo de la adquisición de datos web. Forman la columna vertebral de muchas aplicaciones basadas en datos y son habilidades cruciales para analistas de datos y desarrolladores web.

Crawling/Rastreo: Navegando por la Web

El rastreo, a menudo referido como crawling o scraping web, es el proceso de navegar sistemáticamente por la World Wide Web para recuperar páginas web. Piénsalo como un robot web o araña, incansablemente atravesando internet para descubrir e indexar contenido web. Esta técnica está en el corazón de motores de búsqueda como Google y Bing.

¿Por Qué Rastreamos?

El rastreo sirve varios propósitos importantes:

- **Indexación:** Permite a los motores de búsqueda indexar y catalogar páginas web, haciéndolas buscables por los usuarios.
- **Descubrimiento de Enlaces:** Los rastreadores extraen enlaces de páginas web, ayudando a construir una vasta red de recursos web interconectados. Esta estructura de enlaces es crucial para comprender la arquitectura de la web.
- **Recuperación de Datos:** Los rastreadores pueden raspar o extraer datos de páginas web, pero su objetivo principal es descubrir y navegar a otras páginas web.

Scraping: Recolectando Datos

El scraping es el proceso de extraer datos o información específica de una sola página web. A diferencia del rastreo, que se centra en navegar por la web, el scraping se enfoca en una sola página web para recolectar datos valiosos.

Casos de Uso del Scraping

El scraping se usa para una variedad de propósitos, tales como:

- **Extracción de Datos:** Nos permite extraer datos estructurados como precios de productos, titulares de noticias o información del mercado de valores de sitios web.
- **Monitoreo de Contenido:** El scraping se puede emplear para rastrear cambios en el contenido de páginas web específicas, como monitorear cambios de precios en sitios de comercio electrónico o seguir actualizaciones de noticias.
- **Análisis Competitivo:** Las empresas a menudo usan el scraping para recopilar datos sobre competidores, como estrategias de precios o listados de productos.
- **Investigación y Análisis:** Analistas de datos e investigadores usan el scraping para recopilar datos para estudios, informes y perspectivas basadas en datos.

Beneficios del Scraping

- **Automatización:** Permite automatizar la recopilación de datos desde múltiples fuentes, lo que ahorra tiempo y esfuerzo manual.

- **Acceso a Datos No Estructurados:** Mucho contenido valioso en la web no está disponible en formatos estructurados como bases de datos. El scraping permite acceder a este contenido y convertirlo en datos estructurados para su análisis.
- **Análisis a Gran Escala:** Los datos extraídos mediante scraping pueden analizarse a gran escala, lo que permite obtener información valiosa para la toma de decisiones en tiempo real.

Sinergia entre Rastreo y Scraping

En la práctica, el rastreo y el scraping a menudo trabajan juntos. Los rastreadores recorren la web para encontrar nuevas páginas, y una vez que alcanzan una página de interés, se aplican técnicas de scraping para extraer datos valiosos. Esta sinergia es lo que potencia motores de búsqueda, agregadores de noticias y aplicaciones basadas en datos en internet.

Cómo Funcionan Juntos

- **Rastreo para Descubrimiento:** Los rastreadores son responsables de descubrir nuevas páginas web. Usan algoritmos que siguen enlaces de una página a otra, asegurándose de que el contenido más reciente y relevante esté disponible para su análisis.
- **Scraping para Extracción Específica:** Una vez que el rastreador ha descubierto una página web relevante, las herramientas de scraping extraen los datos específicos de esa página. Esto es útil cuando se necesitan detalles como el precio de un producto, la última publicación en un blog o información de contacto de una empresa.

Desafíos del Scraping y Rastreo

Aunque estas técnicas son poderosas, también presentan desafíos:

- **Políticas de Uso:** Muchos sitios web limitan el acceso a sus datos a través de archivos `robots.txt` o restricciones legales, lo que requiere que los rastreadores y scrapers respeten estas políticas.
- **Bloqueo de IPs:** Los sitios web pueden bloquear direcciones IP que realizan demasiadas solicitudes en un corto período de tiempo. Esto puede interrumpir las actividades de scraping y rastreo, requiriendo el uso de técnicas de evasión como proxies.
- **Datos No Estructurados:** Si bien el scraping permite extraer datos de páginas web, a menudo los datos son desordenados y no siguen un formato estructurado, lo que puede hacer que el proceso de limpieza de datos sea más desafiante.

Conclusión

Entender los conceptos de rastreo y scraping es esencial para cualquiera que busque trabajar con datos web. Ya sea que desees construir un motor de búsqueda, reunir investigación de

mercado o simplemente automatizar la recolección de datos, estas técnicas son tu puerta de entrada para desbloquear la riqueza de información disponible en la web.

PROYECTO DE CALENTAMIENTO: Construyendo una Sencilla Araña Web

Visión General del Proyecto

En este proyecto de calentamiento, nos adentraremos en el mundo de las arañas web o "web crawlers". Estos son programas especializados diseñados para explorar la World Wide Web de manera automatizada, recuperando información de las páginas web que encuentran a medida que siguen los enlaces. Las arañas web son esenciales para diversas aplicaciones, como motores de búsqueda, extracción de datos y análisis de contenido web.

¿Qué es una Araña Web?

Una araña web, también conocida como rastreador o crawler, es un tipo de software que navega por Internet de forma sistemática. Su función principal es acceder a páginas web, descargar su contenido, extraer información valiosa, y seguir los enlaces para continuar rastreando más páginas. Estos procesos forman la base de tecnologías como los motores de búsqueda (por ejemplo, Google, Bing) y otros servicios de datos en línea.

¿Por Qué Construir una Araña Web?

Comprender cómo funciona una araña web es útil para tareas como:

- **Rastreo e Indexación:** Los motores de búsqueda utilizan arañas web para indexar millones de páginas.
- **Recopilación de Datos:** Para obtener información de diversas fuentes, como precios, reseñas, o estadísticas.
- **Monitoreo de Páginas:** Las arañas pueden ser configuradas para rastrear cambios en páginas web.
- **Análisis Web:** Obtener datos para realizar análisis de tendencias o rendimiento de sitios.

Tareas Clave del Proyecto

En este proyecto, nuestro objetivo es construir una araña web básica que pueda:

1. **Explorar Páginas Web:** Iniciar desde una URL proporcionada y recorrer los enlaces en ella.
2. **Extraer Enlaces:** Identificar los enlaces presentes en cada página visitada.
3. **Almacenar Datos:** Guardar el contenido HTML de las páginas visitadas junto con los enlaces extraídos.
4. **Controlar el Número de Rastreos:** Establecer un límite de cuántas páginas rastrear.

Descripción del Proyecto

Vamos a construir una clase `Spider` en Python para implementar nuestra araña web. Esta clase tendrá tres parámetros esenciales:

- `starting_url`: La URL de inicio desde la que la araña comienza a rastrear.
- `crawl_domain`: El dominio de la página que la araña debe rastrear.
- `max_iter`: El número máximo de páginas a rastrear.

La función principal será `Spider.run()`, que gestionará el proceso de rastreo, desde la recuperación del contenido hasta el almacenamiento de los datos extraídos.

Pasos para la Implementación

1. **Definir la Clase Spider:** Crearemos una clase `Spider` con métodos para manejar la lógica de rastreo.
2. **Recuperar Contenido Web:** Utilizaremos la biblioteca `urllib` para abrir y leer el contenido de las páginas web.
3. **Extraer Enlaces:** Implementaremos una función que extraiga los enlaces presentes en el HTML de la página.
4. **Almacenar y Organizar los Datos:** Guardaremos los datos de cada página, como la URL y el contenido HTML, para su posterior análisis.

Flujo de Trabajo de la Araña Web

El proceso de rastreo web generalmente sigue estos pasos:

1. **Acceder a la Web (Acceder web):**
 - En este paso, la araña abre una página web a partir de una URL proporcionada.
 - Para acceder a una página, se realiza una solicitud HTTP al servidor, que devuelve el contenido de la página (HTML, imágenes, etc.).
2. **Descargar Contenido Web (Descargar web):**
 - La araña descarga el contenido de la página web, generalmente en formato HTML. Este contenido incluye texto, imágenes, enlaces y otros recursos.
 - Puede incluir JavaScript y CSS, aunque para simplificar el proyecto, nos centraremos principalmente en el HTML.
3. **Extraer Enlaces (Extraer enlaces):**
 - El siguiente paso es identificar los enlaces dentro del contenido HTML de la página.
 - Estos enlaces, que generalmente están dentro de las etiquetas ``, permitirán a la araña rastrear más páginas.
4. **Almacenar Enlaces Extraídos (Almacenar enlaces):**
 - Los enlaces extraídos se guardan en una lista o cola para seguirlos en el futuro.
 - Esto forma la base del proceso de rastreo, donde cada enlace se puede seguir para recuperar más contenido.
5. **Verificar y Validar Enlaces (Verificar enlaces):**
 - Los enlaces no siempre son útiles o válidos. Este paso asegura que los enlaces son válidos (es decir, no están rotos o erróneos).
 - Se verifica que los enlaces lleven a páginas dentro del dominio especificado y que no hayan sido visitados previamente.
6. **Control de Iteraciones (Limitar iteraciones):**

- Para evitar que el rastreo se vuelva interminable, es importante establecer un límite en el número de páginas que la araña puede visitar.

Código de la Araña Web

Aquí te mostramos cómo implementar la araña web en Python, utilizando las bibliotecas estándar `urllib` para acceder a las páginas y extraer enlaces.

```
# Importando las bibliotecas necesarias
from urllib.request import urlopen # Para abrir URLs
from urllib.error import HTTPError # Para manejar errores HTTP
import time # Para implementar retrasos si es necesario

# Función para extraer enlaces del contenido HTML
def getLinks(html, max_links=10):
    url = [] # Lista para almacenar las URLs extraídas
    cursor = 0 # Cursor para rastrear la posición en el contenido HTML
    nlinks = 0 # Contador para el número de enlaces extraídos

    # Bucle para extraer enlaces hasta que se alcance el máximo o no se encuentren más enlaces
    while (cursor >= 0 and nlinks < max_links):
        start_link = html.find("a href", cursor) # Encontrar el inicio de un enlace
        if start_link == -1: # Si no se encuentran más enlaces, devolver la lista de URLs
            return url
        start_quote = html.find('"', start_link) # Encontrar la comilla de apertura de la URL
        end_quote = html.find('"', start_quote + 1) # Encontrar la comilla de cierre de la URL
        url.append(html[start_quote + 1: end_quote]) # Extraer y agregar la URL a la lista
        cursor = end_quote + 1 # Mover el cursor más allá de esta URL
        nlinks += 1 # Incrementar el contador de enlaces

    return url # Devolver la lista de URLs

# Ejemplo de uso:
# Supongamos que tienes contenido HTML almacenado en una variable `html_content`
html_content = decoded_content
# Llamarías a la función de esta forma:
links = getLinks(html_content)
# Esto devolvería una lista de URLs extraídas del contenido HTML
links

['https://www.mitsubishielectric.es/aire-acondicionado/',
 'https://www.mitsubishielectric.es/aire-acondicionado/']
```

```

'https://www.mitsubishielectric.es/aire-acondicionado/profesionales/',
'https://www.mitsubishielectric.es/aire-acondicionado/descubre-
mitsubishi-electric/',
'#',
'https://www.mitsubishielectric.com/en/index.html',
'#',
'https://www.mitsubishielectric.es/aire-acondicionado/es-tu-
calefaccion/',
'#',
'https://www.mitsubishielectric.es/aire-acondicionado/producto/aire-
acondicionado/']

# Ejemplo de uso:
# Define la URL del sitio web que se va a analizar
url = 'https://www.python.org/'

# Crea una solicitud HTTP con una cabecera de 'User-Agent'
req = urllib.request.Request(url, headers={'User-Agent': 'Magic
Browser'})

# Abre la URL utilizando la solicitud creada y almacena la respuesta
en la variable 'con'
con = urllib.request.urlopen(req)

# Lee el contenido de la respuesta y decodifica el contenido de bytes
a una cadena
# La decodificación se realiza utilizando la codificación
predeterminada (UTF-8)
html_content = con.read().decode()

# Usa la función getLinks para extraer los enlaces del contenido HTML
links = getLinks(html_content)

# Imprime los enlaces extraídos
print(links)

['http://browsehappy.com/', '#content', '/',
'https://www.python.org/psf/', 'https://docs.python.org',
'https://pypi.org/', '/jobs/', '/community/', '/',
'https://psfmember.org/civicrm/contribute/transact?reset=1&id=2']

```

Clase Spider para el Rastreo Web

Vamos a crear una clase `Spider` que utilizará estos métodos para rastrear las páginas, extraer enlaces y almacenar datos. La clase tendrá un diseño modular con varios métodos para cada tarea.

```

# Define la clase Spider para el rastreo web
class Spider:
    # Inicializador o constructor para la clase Spider

```

```

def __init__(self, starting_url, crawl_domain, max_iter):
    self.crawl_domain = crawl_domain # El dominio dentro del cual
la araña rastreará
    self.max_iter = max_iter # El número máximo de páginas a
rastrear
    self.links_to_crawl = [starting_url] # Cola de enlaces a
rastrear
    self.links_visited = [] # Lista para hacer un seguimiento de
los enlaces visitados
    self.collection = [] # Lista para almacenar los datos
recolectados

# Método para recuperar el contenido HTML desde una URL
def retrieveHtml(self):
    try:
        # Abrir la URL y leer la respuesta
        socket = urlopen(self.url)
        # Decodificar la respuesta usando la codificación 'latin-
1'

        self.html = socket.read().decode('latin-1')
        return 0 # Retornar 0 si es exitoso
    except HTTPError as e:
        # Si ocurre un error HTTP, imprimir el error y retornar -1
        print(f"Se ha encontrado un error HTTP: {e}")
        return -1

# Método principal para controlar el proceso de rastreo
def run(self):
    # Continuar el rastreo mientras haya enlaces por rastrear y no
se haya alcanzado max_iter
    while self.links_to_crawl and len(self.collection) <
self.max_iter:
        # Obtener el siguiente enlace a rastrear
        self.url = self.links_to_crawl.pop(0)
        print(f"Actualmente rastreando: {self.url}")
        # Agregar el enlace a la lista de enlaces visitados
        self.links_visited.append(self.url)
        # Si la recuperación del HTML es exitosa, almacenar el
HTML y encontrar nuevos enlaces
        if self.retrieveHtml() >= 0:
            self.storeHtml()
            self.retrieveAndValidateLinks()

# Método para recuperar y validar los enlaces en el contenido HTML
def retrieveAndValidateLinks(self):
    # Obtener una lista de enlaces desde el contenido HTML actual
    items = getLinks(self.html)
    # Lista temporal para almacenar los enlaces válidos
    tmpList = []

```

```

# Iterar sobre los enlaces encontrados
for item in items:
    item = item.strip('\"') # Eliminar comillas extra

    # Verificar si el enlace es una URL absoluta que contiene
    el dominio a rastrear
    if self.crawl_domain in item and item.startswith('http'):
        tmpList.append(item)
    # Manejar enlaces relativos
    elif item.startswith('/'):
        # Construir la URL completa usando el dominio a
        rastrear y el enlace relativo
        tmpList.append('https://' + self.crawl_domain + item)
    # Manejar posibles enlaces relativos sin barra inicial
    (asumiendo que no son URLs absolutas)
    elif not item.startswith('http'):
        # Construir la URL completa asumiendo que es un enlace
        relativo
        tmpList.append('https://' + self.crawl_domain + '/' +
item)

    # Agregar los enlaces válidos y no visitados a la cola de
    rastreo
    for item in tmpList:
        if item not in self.links_visited and item not in
self.links_to_crawl:
            self.links_to_crawl.append(item)
            print(f'Agregado a la cola de rastreo: {item}')

# Método para almacenar el contenido HTML y los metadatos
asociados
def storeHtml(self):
    # Crear un diccionario para representar el documento
    doc = {
        'url': self.url, # URL de la página
        'date': time.strftime("%d/%m/%Y"), # Fecha actual
        'html': self.html # Contenido HTML de la página
    }
    # Agregar el documento a la colección
    self.collection.append(doc)
    print(f"HTML almacenado desde: {self.url}")

# Inicializa el spider con la URL inicial, el dominio a rastrear y el
número máximo de iteraciones.
my_spider = Spider("https://www.python.org/", "python.org", 20)

# Inicia el proceso de rastreo.
my_spider.run()

```

Actualmente rastreando: <https://www.python.org/>
HTML almacenado desde: <https://www.python.org/>
Agregado a la cola de rastreo: <https://python.org/#content>
Agregado a la cola de rastreo: <https://python.org/>
Agregado a la cola de rastreo: <https://www.python.org/psf/>
Agregado a la cola de rastreo: <https://docs.python.org>
Agregado a la cola de rastreo: <https://python.org/jobs/>
Agregado a la cola de rastreo: <https://python.org/community/>
Actualmente rastreando: <https://python.org/#content>
HTML almacenado desde: <https://python.org/#content>
Actualmente rastreando: <https://python.org/>
HTML almacenado desde: <https://python.org/>
Actualmente rastreando: <https://www.python.org/psf/>
HTML almacenado desde: <https://www.python.org/psf/>
Actualmente rastreando: <https://docs.python.org>
HTML almacenado desde: <https://docs.python.org>
Agregado a la cola de rastreo: <https://python.org/download.html>
Agregado a la cola de rastreo: <https://docs.python.org/3.14/>
Agregado a la cola de rastreo: <https://docs.python.org/3.13/>
Agregado a la cola de rastreo: <https://docs.python.org/3.12/>
Agregado a la cola de rastreo: <https://docs.python.org/3.11/>
Agregado a la cola de rastreo: <https://docs.python.org/3.10/>
Agregado a la cola de rastreo: <https://docs.python.org/3.9/>
Agregado a la cola de rastreo: <https://docs.python.org/3.8/>
Agregado a la cola de rastreo: <https://docs.python.org/3.7/>
Actualmente rastreando: <https://python.org/jobs/>
HTML almacenado desde: <https://python.org/jobs/>
Actualmente rastreando: <https://python.org/community/>
HTML almacenado desde: <https://python.org/community/>
Actualmente rastreando: <https://python.org/download.html>
Se ha encontrado un error HTTP: HTTP Error 404: Not Found
Actualmente rastreando: <https://docs.python.org/3.14/>
HTML almacenado desde: <https://docs.python.org/3.14/>
Actualmente rastreando: <https://docs.python.org/3.13/>
HTML almacenado desde: <https://docs.python.org/3.13/>
Actualmente rastreando: <https://docs.python.org/3.12/>
HTML almacenado desde: <https://docs.python.org/3.12/>
Actualmente rastreando: <https://docs.python.org/3.11/>
HTML almacenado desde: <https://docs.python.org/3.11/>
Actualmente rastreando: <https://docs.python.org/3.10/>
HTML almacenado desde: <https://docs.python.org/3.10/>
Actualmente rastreando: <https://docs.python.org/3.9/>
HTML almacenado desde: <https://docs.python.org/3.9/>
Actualmente rastreando: <https://docs.python.org/3.8/>
HTML almacenado desde: <https://docs.python.org/3.8/>
Actualmente rastreando: <https://docs.python.org/3.7/>
HTML almacenado desde: <https://docs.python.org/3.7/>
Agregado a la cola de rastreo: <https://python.org/3/index.html>
Agregado a la cola de rastreo: <https://python.org/genindex.html>
Agregado a la cola de rastreo: <https://python.org/py-modindex.html>

```
Agregado a la cola de rastreo: https://python.org/#
Agregado a la cola de rastreo: https://python.org/whatsnew/index.html
Agregado a la cola de rastreo: https://www.python.org/dev/peps/
Agregado a la cola de rastreo:
https://wiki.python.org/moin/BeginnersGuide
Agregado a la cola de rastreo:
https://wiki.python.org/moin/PythonBooks
Actualmente rastreando: https://python.org/3/index.html
Se ha encontrado un error HTTP: HTTP Error 404: Not Found
Actualmente rastreando: https://python.org/genindex.html
Se ha encontrado un error HTTP: HTTP Error 404: Not Found
Actualmente rastreando: https://python.org/py-modindex.html
Se ha encontrado un error HTTP: HTTP Error 404: Not Found
Actualmente rastreando: https://python.org/#
HTML almacenado desde: https://python.org/#
Actualmente rastreando: https://python.org/whatsnew/index.html
Se ha encontrado un error HTTP: HTTP Error 404: Not Found
Actualmente rastreando: https://www.python.org/dev/peps/
HTML almacenado desde: https://www.python.org/dev/peps/
Actualmente rastreando: https://wiki.python.org/moin/BeginnersGuide
HTML almacenado desde: https://wiki.python.org/moin/BeginnersGuide
Agregado a la cola de rastreo: https://python.org/moin/FrontPage
Agregado a la cola de rastreo: http://www.python.org
Agregado a la cola de rastreo: https://python.org/moin/BeginnersGuide
Agregado a la cola de rastreo: https://python.org/moin/RecentChanges
Agregado a la cola de rastreo: https://python.org/moin/FindPage
Agregado a la cola de rastreo: https://python.org/moin/HelpContents
Actualmente rastreando: https://wiki.python.org/moin/PythonBooks
HTML almacenado desde: https://wiki.python.org/moin/PythonBooks
Agregado a la cola de rastreo: https://python.org/moin/PythonBooks
Actualmente rastreando: https://python.org/moin/FrontPage
Se ha encontrado un error HTTP: HTTP Error 404: Not Found
Actualmente rastreando: http://www.python.org
HTML almacenado desde: http://www.python.org
```

Conclusión

Este proyecto de calentamiento ayuda a aprender cómo construir una araña web básica en Python, y cubre aspectos fundamentales del web scraping, como extraer enlaces, rastrear páginas web, y almacenar la información recuperada. Además, proporciona una base sólida para explorar aplicaciones más avanzadas de la minería de datos y el análisis web.

Resumen de la Clase Spider para Rastreo Web

La clase `Spider` es una implementación que facilita el proceso de rastreo web automatizado, explorando sistemáticamente un dominio para recopilar información de diversas páginas web. A continuación se detalla su funcionamiento, desglosando cada uno de sus componentes y métodos.

Inicialización de la Araña Web

La clase `Spider` comienza su trabajo con la inicialización de tres parámetros clave:

- **starting_url**: La URL inicial desde la cual la araña comenzará a rastrear.
- **crawl_domain**: El dominio que la araña debe seguir y explorar. Esto asegura que los enlaces rastreados pertenezcan a un dominio específico.
- **max_iter**: El número máximo de páginas web que la araña visitará, evitando un rastreo infinito.

Cuando se crea una instancia de la clase, se configura un punto de inicio y se prepara una estructura para almacenar los enlaces por rastrear (`links_to_crawl`) y los enlaces ya visitados (`links_visited`).

Proceso de Rastreo Web

El método principal para realizar el rastreo es `run()`. Este método gestiona el flujo general de la araña:

- Comienza con la URL inicial y agrega la página a la cola de rastreo.
- Continúa procesando cada enlace en la cola de rastreo, extrayendo contenido y buscando nuevos enlaces hasta que:
 - La cola de enlaces se vacíe.
 - Se haya alcanzado el límite de `max_iter`, controlando así el número de páginas visitadas.

Recuperación de Contenido HTML

El método `retrieveHtml()` es responsable de acceder al enlace, obtener el contenido HTML de la página y convertirlo a un formato manejable. Este proceso incluye:

- Intentar abrir el enlace utilizando la función `urlopen()`.
- Leer el contenido de la página y decodificarlo en una cadena de texto.
- Manejar errores como problemas de conexión o errores HTTP, garantizando la robustez del rastreador.

Extracción y Validación de Enlaces

Una vez que se recupera el contenido HTML de la página, el método `retrieveAndValidateLinks()` entra en acción:

- Extrae los enlaces de la página utilizando la función `getLinks()`, que busca las etiquetas `` en el HTML.
- Los enlaces extraídos se validan para asegurarse de que pertenezcan al dominio especificado en `crawl_domain`.
- Los enlaces válidos y que no han sido visitados anteriormente se agregan a la cola de rastreo, asegurando que la araña continúe explorando nuevas páginas dentro del dominio.

Almacenamiento de Datos Recolectados

Cada vez que la araña visita una página, el contenido HTML y la URL de la página se guardan en la colección `collection` utilizando el método `storeHtml()`:

- La URL de la página se guarda junto con la fecha de cuando se rastreó y el contenido HTML de la página.
- Esta colección se utiliza para análisis posteriores, almacenamiento o procesamiento de la información extraída.

El almacenamiento permite hacer un seguimiento de las páginas visitadas y sus contenidos, lo que es útil para tareas de análisis, auditoría o minería de datos.

Validación del Rastreador Web

Ahora validemos el funcionamiento de la araña web utilizando un ejemplo práctico. A continuación, se muestra cómo se inicializa y ejecuta el rastreador para recopilar contenido de un sitio web.

Ejemplo de Uso de la Clase Spider:

Supongamos que estamos interesados en rastrear un sitio de libros llamado "Books to Scrape". Inicializamos el rastreador con la URL inicial, el dominio que queremos explorar y el número máximo de páginas a rastrear:

```
# Asumiendo que la clase Spider está definida previamente con la
función getLinks implementada

# Crear una instancia de la clase Spider con la URL inicial, el
dominio a rastrear y el número máximo de iteraciones
spider = Spider('https://books.toscrape.com/', 'books.toscrape.com',
20)

# Iniciar el proceso de rastreo.
spider.run()

# Después de ejecutar el rastreo, `spider.collection` contendrá el
HTML de hasta 20 páginas de 'books.toscrape.com'.
# Cada entrada en la colección incluirá la URL, la fecha en que se
rastreó y el contenido HTML de la página.

Actualmente rastreando: https://books.toscrape.com/
HTML almacenado desde: https://books.toscrape.com/
Agregado a la cola de rastreo: https://books.toscrape.com/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/catalogue/category/books_1/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/catalogue/category/books/travel_2/index.htm
l
Agregado a la cola de rastreo:
```


https://books.toscrape.com/catalogue/category/books/mystery_3/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/catalogue/category/books/historical-fiction_4/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/catalogue/category/books/sequential-art_5/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/catalogue/category/books/classics_6/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/catalogue/category/books/philosophy_7/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/catalogue/category/books/romance_8/index.html
Actualmente rastreando: <https://books.toscrape.com/index.html>
HTML almacenado desde: <https://books.toscrape.com/index.html>
Actualmente rastreando:
https://books.toscrape.com/catalogue/category/books_1/index.html
HTML almacenado desde:
https://books.toscrape.com/catalogue/category/books_1/index.html
Agregado a la cola de rastreo:
<https://books.toscrape.com/../../index.html>
Agregado a la cola de rastreo:
https://books.toscrape.com/../../books/travel_2/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../books/mystery_3/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../books/historical-fiction_4/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../books/sequential-art_5/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../books/classics_6/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../books/philosophy_7/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../books/romance_8/index.html
Actualmente rastreando:
https://books.toscrape.com/catalogue/category/books/travel_2/index.html
HTML almacenado desde:
https://books.toscrape.com/catalogue/category/books/travel_2/index.html
Agregado a la cola de rastreo:
<https://books.toscrape.com/../../index.html>
Agregado a la cola de rastreo:
https://books.toscrape.com/../../books_1/index.html

Agregado a la cola de rastreo:
https://books.toscrape.com/../../mystery_3/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../historical-fiction_4/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../sequential-art_5/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../classics_6/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../philosophy_7/index.html
Actualmente rastreando:
https://books.toscrape.com/catalogue/category/books/mystery_3/index.html
HTML almacenado desde:
https://books.toscrape.com/catalogue/category/books/mystery_3/index.html
Agregado a la cola de rastreo:
https://books.toscrape.com/../../travel_2/index.html
Actualmente rastreando:
https://books.toscrape.com/catalogue/category/books/historical-fiction_4/index.html
HTML almacenado desde:
https://books.toscrape.com/catalogue/category/books/historical-fiction_4/index.html
Actualmente rastreando:
https://books.toscrape.com/catalogue/category/books/sequential-art_5/index.html
HTML almacenado desde:
https://books.toscrape.com/catalogue/category/books/sequential-art_5/index.html
Actualmente rastreando:
https://books.toscrape.com/catalogue/category/books/classics_6/index.html
HTML almacenado desde:
https://books.toscrape.com/catalogue/category/books/classics_6/index.html
Actualmente rastreando:
https://books.toscrape.com/catalogue/category/books/philosophy_7/index.html
HTML almacenado desde:
https://books.toscrape.com/catalogue/category/books/philosophy_7/index.html
Actualmente rastreando:
https://books.toscrape.com/catalogue/category/books/romance_8/index.html
HTML almacenado desde:
https://books.toscrape.com/catalogue/category/books/romance_8/index.html
Actualmente rastreando: <https://books.toscrape.com/../../../../../index.html>

Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../books/travel_2/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../books/mystery_3/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../books/historical-fiction_4/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../books/sequential-art_5/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../books/classics_6/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../books/philosophy_7/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../books/romance_8/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
<https://books.toscrape.com/../../../index.html>
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../../books_1/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../mystery_3/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando: https://books.toscrape.com/../../historical-fiction_4/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando: https://books.toscrape.com/../../sequential-art_5/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../classics_6/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../philosophy_7/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request
Actualmente rastreando:
https://books.toscrape.com/../../travel_2/index.html
Se ha encontrado un error HTTP: HTTP Error 400: Bad Request

Consultar el Contenido Recopilado

Una vez que se ha ejecutado el rastreo, puedes consultar la cantidad de elementos en la colección con el siguiente código:

```
# ¿Cuántos elementos tiene nuestra colección?
len(spider.collection)

10
```

También puedes consultar el contenido de un elemento específico en la colección:

```
# Consultar el contenido de la primera entrada en la colección
# spider.collection[0]
```

Si deseas ver todas las URLs que la araña ha visitado, puedes enumerar las URLs de la colección de la siguiente manera:

```
# Enumerar las URLs recuperadas
[spider.collection[i]['url'] for i in range(len(spider.collection))]

['https://books.toscrape.com/',
 'https://books.toscrape.com/index.html',
 'https://books.toscrape.com/catalogue/category/books_1/index.html',
 'https://books.toscrape.com/catalogue/category/books/travel_2/index.html',
 'https://books.toscrape.com/catalogue/category/books/mystery_3/index.html',
 'https://books.toscrape.com/catalogue/category/books/historical-fiction_4/index.html',
 'https://books.toscrape.com/catalogue/category/books/sequential-art_5/index.html',
 'https://books.toscrape.com/catalogue/category/books/classics_6/index.html',
 'https://books.toscrape.com/catalogue/category/books/philosophy_7/index.html',
 'https://books.toscrape.com/catalogue/category/books/romance_8/index.html']
```

Resumen Final

Esta clase `Spider` proporciona una forma eficiente de rastrear páginas web, extrayendo enlaces y almacenando contenido de forma automatizada. Al implementar la araña web de esta manera, se facilita la recolección de datos de una manera estructurada, optimizando el proceso de exploración y almacenamiento.

Desafío para Pequeñas Empresas: Extracción de Datos de Productos de una Tienda en Línea Ficticia

Este desafío tiene como objetivo familiarizarte con el proceso de web scraping y análisis de datos utilizando una tienda en línea simulada llamada "Fake Store API". Se espera que puedas recolectar, almacenar y analizar datos de productos, lo que te proporcionará experiencia práctica en la recopilación y manejo de grandes cantidades de datos.

Objetivo

La tarea principal es realizar un análisis de productos extrayendo datos de una tienda en línea simulada, como "Fake Store API" (<https://fakestoreapi.com/>). Este entorno seguro permite practicar técnicas de web scraping de manera ética y eficiente.

Pasos

1. Recopilación de Datos:

- Utiliza la clase `Spider` para rastrear el sitio web de "Fake Store API", la cual proporciona un punto de acceso a información sobre productos, categorías y precios.
- Recopila datos sobre productos que incluyan:
 - **Nombre del producto.**
 - **Categoría del producto.**
 - **Precio.**
 - **Descripción.**
- Realiza el scraping de todas las páginas de productos disponibles en el sitio.

2. Almacenamiento de Datos:

- Una vez que hayas recopilado los datos, es esencial almacenarlos en un formato estructurado, como un archivo CSV o en una base de datos relacional.
- Asegúrate de estructurar los datos de manera que sea fácil analizarlos posteriormente, utilizando un formato adecuado para el análisis de datos.

3. Análisis:

- Realiza un análisis exploratorio de los datos para extraer patrones y tendencias.
- Examina la distribución de los productos entre diferentes categorías, compara los precios y detecta cualquier tendencia emergente en los productos.
- Utiliza herramientas como `pandas` para explorar los datos de manera más eficiente y obtener insights clave.

4. Informe:

- Prepara un informe que resuma tus hallazgos, incluyendo:
 - **Tendencias de productos:** Identifica qué categorías tienen mayor demanda.
 - **Estrategias de precios:** Compara precios entre diferentes categorías o tipos de productos.

- **Popularidad de categorías:** Observa qué categorías tienen más productos disponibles y qué categorías parecen ser más populares entre los consumidores.
- Asegúrate de incluir gráficos y estadísticas para reforzar tus conclusiones.

Tareas para los Estudiantes

- Trabaja en parejas para planificar y ejecutar el proceso de web scraping.
- Asegúrate de seguir buenas prácticas de scraping, como:
 - Consultar el archivo `robots.txt` del sitio web para verificar las restricciones de scraping.
 - Limitar la tasa de solicitudes para evitar sobrecargar el servidor de la API.
- Realiza un análisis exhaustivo de los datos recopilados utilizando herramientas como `pandas` o visualizaciones con `matplotlib`.
- Colabora para crear un informe comprensivo, resumiendo tus hallazgos y las metodologías utilizadas.
- Presenta tus resultados a la clase, destacando las perspectivas clave que se derivan del análisis.

Resultado Esperado

Este desafío proporcionará experiencia práctica en web scraping, análisis de datos y la presentación de hallazgos en un contexto empresarial. Al completar este proyecto, habrás mejorado tu comprensión sobre cómo funciona el mercado de venta al por menor en línea y cómo las pequeñas empresas pueden aprovechar estos datos para tomar decisiones informadas.

Ejemplo de Uso de Web Scraping y Almacenamiento de Datos

A continuación se presentan ejemplos de cómo puedes usar la API de "Fake Store API" para realizar web scraping, almacenar los productos en un archivo CSV, y luego leer esos datos desde el archivo CSV.

```
import requests
import csv

class APIClient:
    def __init__(self, base_url):
        self.base_url = base_url
    def get_products(self):
        response = requests.get(f"{self.base_url}/products")
        if response.status_code == 200:
            return response.json()
        else:
            response.raise_for_status()

# Uso de la clase
```

```

client = APIClient("https://fakestoreapi.com")
products = client.get_products()

# Guardamos los productos en un archivo CSV
def save_to_csv(products, filename="products.csv"):
    keys = products[0].keys() # Se usan las claves del primer
    producto como encabezado
    with open(filename, 'w', newline='', encoding='utf-8') as
    output_file:
        dict_writer = csv.DictWriter(output_file, fieldnames=keys)
        dict_writer.writeheader() # Escribir encabezado
        dict_writer.writerows(products) # Escribir todos los
    productos

# Guardar los productos en el archivo CSV
save_to_csv(products)

```

¿Cómo cargar y mostrar los productos desde un archivo CSV?

```

import csv

# Leer el archivo CSV y cargar los productos
csv_filename = "fake_store_products.csv"
products_list = []

with open(csv_filename, mode='r', newline='', encoding='utf-8') as
file:
    reader = csv.DictReader(file)
    for row in reader:
        products_list.append(row)

# Imprimir los productos almacenados
for product in products_list:
    print(product)

{'id': '1', 'title': 'Fjallraven - Foldsack No. 1 Backpack, Fits 15
Laptops', 'price': '109.95', 'description': 'Your perfect pack for
everyday use and walks in the forest. Stash your laptop (up to 15
inches) in the padded sleeve, your everyday', 'category': "men's
clothing", 'image': 'https://fakestoreapi.com/img/81fPKd-
2AYL.AC_SL1500.jpg', 'rating_rate': '3.9', 'rating_count': '120'}
{'id': '2', 'title': 'Mens Casual Premium Slim Fit T-Shirts ',
'price': '22.3', 'description': 'Slim-fitting style, contrast raglan
long sleeve, three-button henley placket, light weight & soft fabric
for breathable and comfortable wearing. And Solid stitched shirts with
round neck made for durability and a great fit for casual fashion wear
and diehard baseball fans. The Henley style round neckline includes a
three-button placket.', 'category': "men's clothing", 'image':
'https://fakestoreapi.com/img/71-
3HjGNDUL.AC_SY879._SX._UX._SY._UY_.jpg', 'rating_rate': '4.1',

```

```
'rating_count': '259'}
{'id': '3', 'title': 'Mens Cotton Jacket', 'price': '55.99',
 'description': 'great outerwear jackets for Spring/Autumn/Winter,
 suitable for many occasions, such as working, hiking, camping,
 mountain/rock climbing, cycling, traveling or other outdoors. Good
 gift choice for you or your family member. A warm hearted love to
 Father, husband or son in this thanksgiving or Christmas Day.',
 'category': "men's clothing", 'image':
 'https://fakestoreapi.com/img/71li-ujtlUL._AC_UX679_.jpg',
 'rating_rate': '4.7', 'rating_count': '500'}
{'id': '4', 'title': 'Mens Casual Slim Fit', 'price': '15.99',
 'description': 'The color could be slightly different between on the
 screen and in practice. / Please note that body builds vary by person,
 therefore, detailed size information should be reviewed below on the
 product description.', 'category': "men's clothing", 'image':
 'https://fakestoreapi.com/img/71YXze0uslL._AC_UY879_.jpg',
 'rating_rate': '2.1', 'rating_count': '430'}
{'id': '5', 'title': "John Hardy Women's Legends Naga Gold & Silver
 Dragon Station Chain Bracelet", 'price': '695', 'description': "From
 our Legends Collection, the Naga was inspired by the mythical water
 dragon that protects the ocean's pearl. Wear facing inward to be
 bestowed with love and abundance, or outward for protection.",
 'category': 'jewelery', 'image':
 'https://fakestoreapi.com/img/71pWzhdJNwL._AC_UL640_QL65_ML3_.jpg',
 'rating_rate': '4.6', 'rating_count': '400'}
{'id': '6', 'title': 'Solid Gold Petite Micropave ', 'price': '168',
 'description': 'Satisfaction Guaranteed. Return or exchange any order
 within 30 days.Designed and sold by Hafeez Center in the United
 States. Satisfaction Guaranteed. Return or exchange any order within
 30 days.', 'category': 'jewelery', 'image':
 'https://fakestoreapi.com/img/61sbMiUnoGL._AC_UL640_QL65_ML3_.jpg',
 'rating_rate': '3.9', 'rating_count': '70'}
{'id': '7', 'title': 'White Gold Plated Princess', 'price': '9.99',
 'description': "Classic Created Wedding Engagement Solitaire Diamond
 Promise Ring for Her. Gifts to spoil your love more for Engagement,
 Wedding, Anniversary, Valentine's Day...", 'category': 'jewelery',
 'image':
 'https://fakestoreapi.com/img/71YAIFU48IL._AC_UL640_QL65_ML3_.jpg',
 'rating_rate': '3', 'rating_count': '400'}
{'id': '8', 'title': 'Pierced Owl Rose Gold Plated Stainless Steel
 Double', 'price': '10.99', 'description': 'Rose Gold Plated Double
 Flared Tunnel Plug Earrings. Made of 316L Stainless Steel',
 'category': 'jewelery', 'image':
 'https://fakestoreapi.com/img/51UDEzMJVpL._AC_UL640_QL65_ML3_.jpg',
 'rating_rate': '1.9', 'rating_count': '100'}
{'id': '9', 'title': 'WD 2TB Elements Portable External Hard Drive -
 USB 3.0 ', 'price': '64', 'description': 'USB 3.0 and USB 2.0
 Compatibility Fast data transfers Improve PC Performance High
 Capacity; Compatibility Formatted NTFS for Windows 10, Windows 8.1,
```


Windows 7; Reformatting may be required for other operating systems; Compatibility may vary depending on user's hardware configuration and operating system', 'category': 'electronics', 'image': 'https://fakestoreapi.com/img/61IBBVJvSDL_AC_SY879_.jpg', 'rating_rate': '3.3', 'rating_count': '203'}

{'id': '10', 'title': 'SanDisk SSD PLUS 1TB Internal SSD - SATA III 6 Gb/s', 'price': '109', 'description': 'Easy upgrade for faster boot up, shutdown, application load and response (As compared to 5400 RPM SATA 2.5" hard drive; Based on published specifications and internal benchmarking tests using PCMark vantage scores) Boosts burst write performance, making it ideal for typical PC workloads The perfect balance of performance and reliability Read/write speeds of up to 535MB/s/450MB/s (Based on internal testing; Performance may vary depending upon drive capacity, host device, OS and application.)', 'category': 'electronics', 'image': 'https://fakestoreapi.com/img/61U7T1koQqL_AC_SX679_.jpg', 'rating_rate': '2.9', 'rating_count': '470'}

{'id': '11', 'title': 'Silicon Power 256GB SSD 3D NAND A55 SLC Cache Performance Boost SATA III 2.5', 'price': '109', 'description': '3D NAND flash are applied to deliver high transfer speeds Remarkable transfer speeds that enable faster bootup and improved overall system performance. The advanced SLC Cache Technology allows performance boost and longer lifespan 7mm slim design suitable for Ultrabooks and Ultra-slim notebooks. Supports TRIM command, Garbage Collection technology, RAID, and ECC (Error Checking & Correction) to provide the optimized performance and enhanced reliability.', 'category': 'electronics', 'image': 'https://fakestoreapi.com/img/71kWymZ+c+L_AC_SX679_.jpg', 'rating_rate': '4.8', 'rating_count': '319'}

{'id': '12', 'title': 'WD 4TB Gaming Drive Works with Playstation 4 Portable External Hard Drive', 'price': '114', 'description': "Expand your PS4 gaming experience, Play anywhere Fast and easy, setup Sleek design with high capacity, 3-year manufacturer's limited warranty", 'category': 'electronics', 'image': 'https://fakestoreapi.com/img/61mtL65D4cL_AC_SX679_.jpg', 'rating_rate': '4.8', 'rating_count': '400'}

{'id': '13', 'title': 'Acer SB220Q bi 21.5 inches Full HD (1920 x 1080) IPS Ultra-Thin', 'price': '599', 'description': '21.5 inches Full HD (1920 x 1080) widescreen IPS display And Radeon free Sync technology. No compatibility for VESA Mount Refresh Rate: 75Hz - Using HDMI port Zero-frame design | ultra-thin | 4ms response time | IPS panel Aspect ratio - 16: 9. Color Supported - 16. 7 million colors. Brightness - 250 nit Tilt angle -5 degree to 15 degree. Horizontal viewing angle-178 degree. Vertical viewing angle-178 degree 75 hertz', 'category': 'electronics', 'image': 'https://fakestoreapi.com/img/81QpkIctqPL_AC_SX679_.jpg', 'rating_rate': '2.9', 'rating_count': '250'}

{'id': '14', 'title': 'Samsung 49-Inch CHG90 144Hz Curved Gaming Monitor (LC49HG90DMNXZA) – Super Ultrawide Screen QLED ', 'price':

'999.99', 'description': '49 INCH SUPER ULTRA WIDE 32:9 CURVED GAMING MONITOR with dual 27 inch screen side by side QUANTUM DOT (QLED) TECHNOLOGY, HDR support and factory calibration provides stunningly realistic and accurate color and contrast 144HZ HIGH REFRESH RATE and 1ms ultra fast response time work to eliminate motion blur, ghosting, and reduce input lag', 'category': 'electronics', 'image': 'https://fakestoreapi.com/img/81Zt42ioCgL._AC_SX679_.jpg', 'rating_rate': '2.2', 'rating_count': '140'}

{'id': '15', 'title': "BIYLACLESEN Women's 3-in-1 Snowboard Jacket Winter Coats", 'price': '56.99', 'description': 'Note:The Jackets is US standard size, Please choose size as your usual wear Material: 100% Polyester; Detachable Liner Fabric: Warm Fleece. Detachable Functional Liner: Skin Friendly, Lightweight and Warm. Stand Collar Liner jacket, keep you warm in cold weather. Zippered Pockets: 2 Zippered Hand Pockets, 2 Zippered Pockets on Chest (enough to keep cards or keys) and 1 Hidden Pocket Inside. Zippered Hand Pockets and Hidden Pocket keep your things secure. Humanized Design: Adjustable and Detachable Hood and Adjustable cuff to prevent the wind and water, for a comfortable fit. 3 in 1 Detachable Design provide more convenience, you can separate the coat and inner as needed, or wear it together. It is suitable for different season and help you adapt to different climates', 'category': "women's clothing", 'image': 'https://fakestoreapi.com/img/51Y5NI-I5jL._AC_UX679_.jpg', 'rating_rate': '2.6', 'rating_count': '235'}

{'id': '16', 'title': "Lock and Love Women's Removable Hooded Faux Leather Moto Biker Jacket", 'price': '29.95', 'description': '100% POLYURETHANE(shell) 100% POLYESTER(lining) 75% POLYESTER 25% COTTON (SWEATER), Faux leather material for style and comfort / 2 pockets of front, 2-For-One Hooded denim style faux leather jacket, Button detail on waist / Detail stitching at sides, HAND WASH ONLY / DO NOT BLEACH / LINE DRY / DO NOT IRON', 'category': "women's clothing", 'image': 'https://fakestoreapi.com/img/81XH0e8fefL._AC_UY879_.jpg', 'rating_rate': '2.9', 'rating_count': '340'}

{'id': '17', 'title': 'Rain Jacket Women Windbreaker Striped Climbing Raincoats', 'price': '39.99', 'description': "Lightweight perfect for trip or casual wear---Long sleeve with hooded, adjustable drawstring waist design. Button and zipper front closure raincoat, fully stripes Lined and The Raincoat has 2 side pockets are a good size to hold all kinds of things, it covers the hips, and the hood is generous but doesn't overdo it. Attached Cotton Lined Hood with Adjustable Drawstrings give it a real styled look.", 'category': "women's clothing", 'image': 'https://fakestoreapi.com/img/71HblAHs5xL._AC_UY879_-2.jpg', 'rating_rate': '3.8', 'rating_count': '679'}

{'id': '18', 'title': "MBJ Women's Solid Short Sleeve Boat Neck V ", 'price': '9.85', 'description': '95% RAYON 5% SPANDEX, Made in USA or Imported, Do Not Bleach, Lightweight fabric with great stretch for comfort, Ribbed on sleeves and neckline / Double stitching on bottom hem', 'category': "women's clothing", 'image':

```
'https://fakestoreapi.com/img/71z3kpMAYsL._AC_UY879_.jpg',  
'rating_rate': '4.7', 'rating_count': '130'}  
{'id': '19', 'title': "Opna Women's Short Sleeve Moisture", 'price':  
'7.95', 'description': '100% Polyester, Machine wash, 100% cationic  
polyester interlock, Machine Wash & Pre Shrunk for a Great Fit,  
Lightweight, roomy and highly breathable with moisture wicking fabric  
which helps to keep moisture away, Soft Lightweight Fabric with  
comfortable V-neck collar and a slimmer fit, delivers a sleek, more  
feminine silhouette and Added Comfort', 'category': "women's  
clothing", 'image':  
'https://fakestoreapi.com/img/51leg55uWmdL._AC_UX679_.jpg',  
'rating_rate': '4.5', 'rating_count': '146'}  
{'id': '20', 'title': 'DANVOUY Womens T Shirt Casual Cotton Short',  
'price': '12.99', 'description': '95%Cotton,5%Spandex, Features:  
Casual, Short Sleeve, Letter Print,V-Neck,Fashion Tees, The fabric is  
soft and has some stretch., Occasion:  
Casual/Office/Beach/School/Home/Street. Season:  
Spring,Summer,Autumn,Winter.', 'category': "women's clothing",  
'image': 'https://fakestoreapi.com/img/61pHAEJ4NML._AC_UX679_.jpg',  
'rating_rate': '3.6', 'rating_count': '145'}
```

Uso de APIs para la Recuperación de Datos

Entendiendo el Panorama General

En muchos casos, los sitios web proporcionan APIs (Interfaz de Programación de Aplicaciones) como una forma estructurada y eficiente de acceder a los datos. Las APIs permiten obtener información específica sin tener que procesar todo el HTML de una página web, lo que hace que el proceso sea mucho más eficiente.

La Ventaja de las APIs

Las APIs ofrecen varios beneficios importantes en comparación con el web scraping tradicional:

- **Acceso más estructurado:** Los datos vienen en un formato fácilmente legible como JSON.
- **Estabilidad:** A diferencia del scraping, que puede romperse si el diseño de la página cambia, las APIs están diseñadas para ofrecer acceso consistente a los datos.
- **Mayor eficiencia:** Las APIs generalmente proporcionan solo los datos necesarios, lo que reduce el esfuerzo de procesar grandes cantidades de HTML.

Uso de APIs con Autenticación

En algunas APIs, es necesario autenticar tus solicitudes. Esto generalmente se hace enviando un token de API con la solicitud, lo que permite acceder a los datos de manera segura y controlada. Consulta la documentación de la API para obtener detalles sobre cómo manejar la autenticación.

Resumen

Utilizar APIs para obtener datos es más eficiente, predecible y ético en comparación con el web scraping. Cuando una API está disponible, generalmente es el método preferido para obtener datos de un sitio web. El uso de APIs también asegura que sigas las políticas de acceso del sitio web y minimizas el riesgo de sobrecargar sus servidores.

Ejemplo: Obtener Datos del Clima Usando la API de OpenWeatherMap en Python

Este ejemplo demuestra cómo usar la API de OpenWeatherMap para obtener datos meteorológicos en tiempo real para una ciudad específica.

Prerrequisitos

- Obtener una clave API gratuita de OpenWeatherMap (<https://openweathermap.org/api>).
- Asegurarse de tener la biblioteca `requests` instalada (`pip install requests`).

Script de Python para Recuperar Datos Meteorológicos

```
import requests

def get_weather(api_key, city):
    base_url = "http://api.openweathermap.org/data/2.5/weather?"
    city_name = city
    complete_url = f"{base_url}appid={api_key}&q={city_name}"
    response = requests.get(complete_url)
    return response.json()

# Reemplazar con tu clave API y el nombre de la ciudad
api_key = '9da0f0082cf35d67520a646b8f2ff3a7'
city_name = 'Barcelona'
weather_data = get_weather(api_key, city_name)

print(f"Clima en {city_name}:")
print(weather_data)

Clima en Barcelona:
{'coord': {'lon': 2.159, 'lat': 41.3888}, 'weather': [{'id': 800,
'main': 'Clear', 'description': 'clear sky', 'icon': '01d'}], 'base':
'stations', 'main': {'temp': 285.6, 'feels_like': 285.02, 'temp_min':
283.78, 'temp_max': 287.48, 'pressure': 1016, 'humidity': 81,
'sea_level': 1016, 'grnd_level': 1008}, 'visibility': 10000, 'wind':
{'speed': 3.6, 'deg': 340}, 'clouds': {'all': 0}, 'dt': 1732000199,
'sys': {'type': 2, 'id': 2003688, 'country': 'ES', 'sunrise':
1731998702, 'sunset': 1732033737}, 'timezone': 3600, 'id': 3128760,
'name': 'Barcelona', 'cod': 200}
```

Expected Output

El script devolverá datos sobre el clima en formato JSON, que incluyen la temperatura, humedad, descripción del clima, etc.

Nota

- Asegúrate de reemplazar 'YOUR_API_KEY' y 'CITY_NAME' con tu clave API y la ciudad deseada.
 - Explora la documentación de OpenWeatherMap para obtener más detalles sobre cómo personalizar las solicitudes y manejar diferentes tipos de datos.
-

Desafío: Analizando Tendencias de Hashtags en Instagram con Instaloader

Objetivo

Aprovechar `Instaloader`, una biblioteca de Python, para descargar publicaciones asociadas con un hashtag específico en Instagram. Analiza los datos recopilados para identificar tendencias, contenido popular y compromiso del usuario.

Pasos

1. Instalar Instaloader

- Asegúrate de que Python esté instalado en tu sistema.
- Instala `Instaloader` usando pip: `pip install instaloader`

2. Recolección de Datos

- Elige un hashtag relevante para un tema de interés (por ejemplo, #naturaleza, #viaje, #comida).
- Usa `Instaloader` para descargar publicaciones etiquetadas con el hashtag elegido. Considera limitaciones como el número de publicaciones para evitar sobrecargar la API.

```
import instaloader

L = instaloader.Instaloader()
posts = instaloader.Hashtag.from_name(L.context,
'TU_HASHTAG').get_posts()

for post in posts:
    # Agrega código para procesar y almacenar detalles de la
    publicación
```

3. Análisis de Datos

Analiza los datos descargados para:

- Tendencias populares en el hashtag.
- Temas o sujetos comunes en imágenes o subtítulos.
- Niveles de compromiso del usuario (me gusta, comentarios).

4. Reporte

- Compila tus hallazgos en un informe.
- Incluye representaciones visuales (gráficos, nubes de palabras) para ilustrar las tendencias clave.

Notas Importantes

- Respeta los términos de servicio de Instagram y las directrices éticas en el scraping de datos.
- Ten en cuenta la privacidad y el consentimiento, especialmente con el contenido generado por el usuario.
- El alcance de la recolección de datos debe ser limitado para fines educativos.

Resultado Esperado

Este desafío tiene como objetivo proporcionar experiencia práctica con Instaloader, desarrollar habilidades de análisis de datos y ofrecer perspectivas sobre tendencias en redes sociales y comportamiento del usuario.

```
# import instaloader

# # Crear una instancia de Instaloader
# L = instaloader.Instaloader()

# # Descargar publicaciones asociadas con un perfil
# profile_name = 'nasa' # Cambiar según el perfil que quieras
# analizar
# profile = instaloader.Profile.from_username(L.context, profile_name)

# # Limitar el número de publicaciones a descargar
# max_posts = 10
# posts_info = []

# # Recorrer las publicaciones y almacenar la información
# for index, post in enumerate(profile.get_posts()):
#     if index >= max_posts:
#         break
#     post_info = {
#         'id': post.shortcode,
#         'likes': post.likes,
#         'comments': post.comments,
```

```

#         'caption': post.caption,
#         'timestamp': post.date_utc,
#         'url': f"https://www.instagram.com/p/{post.shortcode}/"
#     }
#     posts_info.append(post_info)

# # Imprimir la información de las publicaciones descargadas
# print("Publicaciones del perfil:")
# for info in posts_info:
#     print(info)

```

```

-----
-----
NameError                                Traceback (most recent call
last)

```

```

Cell In[42], line 16
      13 posts_info = []
      15 # Recorrer las publicaciones y almacenar la información
--> 16 for index, post in enumerate(profile.get_posts()):
      17     if index >= max_posts:
      18         break

```

NameError: name 'profile' is not defined

```

# Función para filtrar publicaciones que contienen un hashtag
# específico en el texto
def filtrar_hashtag(posts_info, hashtag):
    hashtag_posts = [post for post in posts_info if hashtag.lower() in
(post['caption'] or '').lower()]
    return hashtag_posts

```

```

# Hashtag a buscar (sin incluir el símbolo #)
hashtag = 'Moon'

```

```

# Filtrar publicaciones que contienen el hashtag en su texto
# hashtag_posts = filtrar_hashtag(posts_info, hashtag)

```

```

# Imprimir publicaciones que contienen el hashtag
# print(f"\nPublicaciones con el hashtag #{hashtag}:")
# for info in hashtag_posts:
#     print(info)

```