Evaluación Módulo de Web Scraping

Amelia Cristina Herrera Briceño

melinnicri@gmail.com

14/07/2024 – 17/07/2024

La evaluación consistirá en 6 preguntas en las cuales deberá resolver enviando el código Python en cualquiera de los dos formatos requeridos (*.py o *.ipynb). Para las justificaciones podrán hacer uso de este mismo archivo Word siempre que mantengan coherencia en la entrega de sus respuestas. Tendrán plazo de entrega hasta el día jueves 17 de Julio a las 12 pm. Trate en lo posible de no dejar los problemas para último momento. El correo es: garcia.moncada2521@gmail.com

1.- Realice el Web scraping de la siguiente página usando selenium “https://subslikescript.com/” cree un dataframe (en *.csv) en donde extraiga las películas y sus correspondientes links. (6 puntos).

*Realizado en Colab de Google. Subido a Github: melinnicri/Scrapeo_web: Cómo scrapear una página web (github.com)

Archivos: notebook “TareaAmeliaCris.ipynb”, archivo final .csv "peliculas_scraped.csv".

Notebook ”Pedido.ipynb”, archivo final .csv "pedido.csv".

2.- Utilizando la misma página anterior realice la extracción del guion de una película (en *.csv o *.txt). (3 puntos).

* Realizado en Colab de Google, subido a melinnicri/Scrapeo_web: Cómo scrapear una página web (github.com)

Archivos: notebook “Guion02.ipynb”, archivo final .txt: si es el resumen inicial (“guion_texto.txt"); si es a continuación del resumen (“guion_extraido.txt").

3.- Realice el Web scraping de la siguiente página “https://en.wikipedia.org/wiki/%C3%8Dndice_de_Precio_Selectivo_de_Acciones” cree un dataframe (en *.csv) en donde extraiga la información de la tabla. (5 puntos)

* Realizado en Colab de Google, subido a melinnicri/Scrapeo_web: Cómo scrapear una página web (github.com)

Archivos: notebook “TablaWiki.ipynb.ipynb”, archivo final .cvs: “ipsa.csv”.

4.- ¿En qué situaciones se hace necesario usar las sentencias try-except en Web Scraping? (2 puntos).

Usos Esenciales de las Sentencias Try-Except en Web Scraping:

Las sentencias try-except son cruciales en el Web Scraping para manejar errores y excepciones que puedan surgir durante el proceso de extracción de datos. Implementarlas correctamente permite:

1. Manejar Errores de Red y Conexión:

- **Errores de Conexión:** El código puede fallar debido a problemas de conexión a Internet, tiempo de espera agotado o sitios web inaccesibles.
- **Errores HTTP:** Errores como 404 (Página no encontrada), 500 (Error del servidor interno) o 503 (Servicio no disponible) pueden ocurrir.

Ejemplo:

Python

```
try:
    # Hacer solicitud HTTP
    response = requests.get(url)
except requests.exceptions.RequestException as e:
    # Manejar error de red o conexión
    print(f"Error de red o conexión: {e}")
```

2. Manejar Errores de Análisis de HTML/DOM:

- **Estructuras HTML Inesperadas:** El HTML de la página puede cambiar o no coincidir con lo esperado, causando errores de análisis.
- **Elementos No Encontrados:** El código puede buscar elementos que no existen en la página, generando excepciones.

Ejemplo:

Python

```
try:
    # Buscar elemento HTML
    element = driver.find_element(By.CSS_SELECTOR, ".selector-
especifico")
except selenium.common.exceptions.NoSuchElementException as e:
    # Manejar error de elemento no encontrado
    print(f"Elemento no encontrado: {e}")
```

3. Manejar Errores de Datos Inválidos:

- **Formatos de Datos Inesperados:** Los datos extraídos pueden tener formatos incorrectos o valores no válidos.
- **Inconsistencias en los Datos:** Los datos pueden ser inconsistentes o incompletos, causando problemas de procesamiento.

Ejemplo:

Python

```
try:
    # Convertir dato a tipo deseado
    precio_texto = element.text
    precio_numerico = float(precio_texto)
except ValueError as e:
    # Manejar error de formato de dato
    print(f"Error de formato de precio: {e}")
```

4. Manejar Errores Específicos del Sitio Web:

- **Protección contra Bots:** Algunos sitios web detectan y bloquean el scraping automático.
- **Limites de Tasas de Rastreo:** Los sitios web pueden limitar la cantidad de solicitudes que se pueden realizar en un tiempo determinado.

Ejemplo:

Python

```
try:
    # Hacer múltiples solicitudes con retardo
    for page_number in range(1, 10):
        # Scrapear página y aplicar retardo
        scrape_page(page_number)
        time.sleep(2)  # Ajustar el retardo según sea necesario
except Exception as e:
    # Manejar error específico del sitio web (p. ej., bloqueo)
    print(f"Error específico del sitio web: {e}")
```

Beneficios de Usar Try-Except:

- **Robustez del Código:** Permite que el código se recupere de errores y continúe procesando otras páginas.
- **Manejo de Excepciones Personalizado:** Se pueden implementar acciones específicas para cada tipo de error.
- **Mejora de la Experiencia del Usuario:** Evita que el script se bloquee por completo y proporcione información útil sobre los errores.

Recomendaciones:

- Identificar los posibles errores que puedan surgir en el escenario de scraping.
- Implementar bloques try-except específicos para cada tipo de error anticipado.
- Registrar los errores para análisis y depuración posteriores.
- Ajustar los retardos y tiempos de espera según las políticas del sitio web.

En resumen, las sentencias try-except son herramientas esenciales para el Web Scraping que permiten manejar errores de manera robusta y flexible, asegurando la confiabilidad y eficiencia del código.

5.- ¿En qué situaciones se hace más cómodo utilizar Xpath en Web Scraping? (2 puntos).

Situaciones Donde XPath Resulta Útil en Web Scraping con Python:

XPath (XML Path Language) es una herramienta poderosa para navegar y extraer datos de estructuras HTML y XML. Su uso en Web Scraping con Python presenta ventajas en diversas situaciones:

1. Estructuras HTML Complejas y Jerárquicas:

- **Árbol DOM Profundo:** Cuando la estructura HTML es profunda y anidada, XPath permite navegar por el árbol DOM de manera precisa y eficiente.
- **Elementos Dinámicos:** XPath puede seleccionar elementos dinámicos generados por JavaScript, lo que lo hace útil para sitios web con contenido dinámico.
- **Atributos y Relaciones Complejas:** XPath permite filtrar y seleccionar elementos en función de atributos, relaciones entre nodos y posiciones dentro del árbol DOM.

Ejemplo:

Python

```
# Seleccionar todos los títulos de artículos dentro de secciones
con clase "articulo"
titulos_articulos = driver.find_elements(By.XPATH,
"//section[@class='articulo']/h2")
```

2. Estructuras HTML Incoherentes o No Estandarizadas:

- **Falta de IDs Únicos:** Cuando los elementos HTML no tienen IDs únicos, XPath puede usar atributos, clases, relaciones de nodos y texto para identificarlos de manera confiable.
- **Patrones de Nombres Incoherentes:** Si los nombres de las clases o IDs no son consistentes, XPath permite usar expresiones más flexibles para localizar elementos.
- **Contenido Dinámico y Generado por JavaScript:** XPath puede seleccionar elementos dinámicos generados por JavaScript, incluso si no tienen IDs o clases estáticas.

Ejemplo:

Python

```
# Seleccionar todos los enlaces que contienen la palabra
"comprar" dentro de botones
enlaces_comprar = driver.find_elements(By.XPATH,
"//button//a[contains(text(), 'comprar')]")
```

3. Extracción de Datos Específicos y Granulares:

- **Seleccionar Atributos Específicos:** XPath permite extraer atributos específicos de elementos HTML, como valores de data-attributes o propiedades CSS.
- **Filtrar Resultados con Condiciones:** Las expresiones XPath pueden incluir condiciones para filtrar resultados, como seleccionar elementos con ciertos valores de atributo o ubicaciones dentro del árbol DOM.
- **Combinar Datos de Múltiples Elementos:** XPath puede combinar datos de diferentes elementos relacionados en la estructura HTML.

Ejemplo:

Python

```
# Extraer el título, precio e imagen de cada producto en una
lista de productos
productos = []
for producto_element in driver.find_elements(By.XPATH,
"//li[@class='producto']"):
    titulo = producto_element.find_element(By.XPATH,
".//h3").text
    precio = producto_element.find_element(By.XPATH,
".//span[@class='precio']").text
    imagen = producto_element.find_element(By.XPATH,
".//img").get_attribute("src")
    productos.append({
        "titulo": titulo,
        "precio": precio,
        "imagen": imagen
    })
```

4. Mantenimiento y Actualización del Código:

- **Cambios en la Estructura HTML:** Si la estructura HTML del sitio web cambia, las expresiones XPath pueden adaptarse con mayor facilidad que los selectores CSS simples.
- **Escalabilidad para Múltiples Sitios Web:** XPath comparte una sintaxis similar entre diferentes sitios web, lo que facilita la reutilización del código para scraping en diversos sitios.
- **Lectura y Comprensión del Código:** Las expresiones XPath suelen ser más descriptivas y fáciles de entender que los selectores CSS complejos, mejorando la legibilidad del código.

En resumen, XPath es una herramienta valiosa para Web Scraping con Python, especialmente cuando se enfrentan a estructuras HTML complejas, dinámicas o incoherentes, se requiere una extracción de datos precisa y granular, o se busca mantener y actualizar el código de manera eficiente.

Sin embargo, es importante considerar que XPath puede ser más lento que los selectores CSS simples en algunos casos.

Hay que recordar que la mejor herramienta para Web Scraping depende de las características específicas del sitio web y los datos que se desean extraer.

6.- El Web Scraping dinámico presenta una serie de ventajas con respecto al Web Scraping estático, mencione 3 de esas ventajas. (2 puntos).

Ventajas Clave del Web Scraping Dinámico en Python:

El Web Scraping dinámico, que utiliza herramientas como Selenium para interactuar con el sitio web como un usuario real, ofrece varias ventajas sobre el Web Scraping estático basado en el análisis de HTML estático:

1. Extracción de Datos Dinámicos y Generados por JavaScript:

- **Contenido Interactivo:** Permite acceder a contenido dinámico generado por JavaScript, como menús desplegables, contenido cargado mediante AJAX o elementos renderizados en el cliente.
- **Datos Actualizados:** Obtiene datos actualizados que se generan en tiempo real al interactuar con el sitio web, lo que no es posible con el HTML estático.
- **Información Completa:** Captura la información completa que se muestra al usuario, incluyendo elementos dinámicos y JavaScript.

Ejemplo:

- Extraer comentarios generados por JavaScript que se cargan al hacer clic en un botón "Mostrar más comentarios".
- Obtener precios actualizados de productos que se calculan en función de opciones seleccionadas.
- Scrapear información de chats en tiempo real o mensajes dinámicos.

2. Simulación de Interacciones del Usuario:

- **Navegación Interactiva:** Permite navegar por el sitio web como un usuario real, siguiendo enlaces, rellenando formularios y realizando acciones como clics o envíos.
- **Autentificación y Registro:** Puede automatizar procesos de autenticación e inicio de sesión para acceder a contenido restringido o áreas de miembros.
- **Interacciones Complejas:** Simula interacciones complejas con el sitio web, como cargar archivos, usar selectores de fecha o realizar acciones en múltiples páginas.

Ejemplo:

- Iniciar sesión en una cuenta de usuario para acceder a datos personalizados.
- Completar formularios de registro o solicitud con datos dinámicos.
- Navegar por una tienda online agregando productos al carrito y completando la compra.

3. Manejo de Sitios Web con Protecciones contra Scraping:

- **Evasión de Protecciones contra Bots:** Puede sortear algunas protecciones contra scraping que detectan y bloquean el acceso automatizado.
- **Adaptación a Cambios en el Sitio Web:** Al interactuar con el sitio web como un usuario, es menos probable que se vea afectado por cambios menores en la estructura HTML o JavaScript.
- **Scraping Robusto y Confiable:** Permite obtener datos de manera más robusta y confiable, incluso en sitios web con medidas anti-scraping.

Ejemplo:

- Extraer datos de sitios web que bloquean el acceso a través de User-Agent o análisis de patrones de acceso.
- Scrapear información de sitios web que utilizan técnicas de ofuscación de JavaScript para dificultar el scraping estático.
- Obtener datos de sitios web que cambian dinámicamente su contenido o estructura en función del comportamiento del usuario.

En resumen, el Web Scraping dinámico con Selenium ofrece una forma más flexible, robusta y completa de extraer datos de sitios web modernos que utilizan JavaScript y presentan contenido dinámico.

Sin embargo, es importante tener en cuenta que el Web Scraping dinámico puede ser más lento y complejo que el Web Scraping estático, y debe usarse con responsabilidad y respetando las políticas de los sitios web.//.