T.2.2. WEB SCRAPING

Juan Carlos Pérez González

ÍNDICE

Consideraciones Generales

Tarea 1. BeautifulSoup

Tarea 2. Selenium

Tarea 3. Scrapy

1. Consideraciones Generales

a. Este trabajo tendrá mucho de investigación ya que los objetivos que se pretenden están muy abiertos a vuestra elección.
b. Asegúrate de revisar y cumplir con los términos de servicio del sitio web al realizar *scraping*, es decir, si lo permite o no. Probablemente si no lo permite ya no te dejará hacer *scraping*.
c. No olvidemos instalar los paquetes necesarios, así como el driver o plugin para el navegador en el caso de Selenium.
d. Adapta las etiquetas a la estructura del sitio web. Ten en cuenta que no todos los sitios tienen el mismo DOM
e. Como siempre, no dudéis en utilizar todas las herramientas y ayudas que consideréis oportunas, pero recordad, **entended lo que estáis haciendo en el código**.
f. Si encontráis una solución en la web, adaptarla a otro sitio web.
g. Documentar el código así fijareis los conceptos y me permite comprobar
h. Procura manejar los errores con excepciones.
i. La entrega como siempre, códigos en ficheros ***.ipnyb*** o ***.py***. y nombre vuestros apellidos.
j. No dudéis en consultar aquellas dudas que tengáis.

TAREA 1. BeautifulSoup

En este ejercicio, se propone crear un script Python utilizando ***Beautiful Soup*** para realizar scraping web y extraer información de productos de un **sitio de compras en línea**. La aplicación que construirás será un simple scraper que extraerá detalles como mínimo el nombre del producto, el precio y el enlace de compra aunque son susceptibles de ampliación.

Los pasos que debes seguir son:

1. Selecciona un sitio web. Debes proponer aquel sitio web que desees, siempre y cuando dicho sitio permite el scraping que como ya hemos visto no siempre ocurre.
2. No nos olvidemos: ***pip install beautifulsoup***
3. Scraping de los datos. Identifica aquellas etiquetas HTML que contienen la información relevante con los datos que se solicitan.
4. Almacenamiento de los datos. Debe ser en CSV-Excel y JSON. Como mínimo uno al menos para poder visualizar los resultados.

Tarea 2. Selenium[1]

Este ejercicio plantea un script con ***Selenium*** en una página de buscador de vuelos, como siempre puedes elegir la que quieras. Importante, hacer búsquedas en anónimo para evitar confirmar un vuelo y que no sean automatizadas para evitar que bloqueen la búsqueda.

La idea es realizar el seguimiento de los precios de un destino, **solo ida**, el que queráis durante algunos días y guardarlo en un fichero.

Seleccionar la página o páginas donde realizar búsqueda de vuelos.
No nos olvidemos: ***pip install selenium***
Configurar ***Selenium*** con el driver de acuerdo al navegador que vais a utilizar.
Buscar los id necesarios para realizar la búsqueda para:

Origen
Destino
Fecha

Volcar los datos en un fichero CSV-Excel o JSON

Anexo.

En el tema de los drivers de selenium tenéis que tener en cuenta que:

1. si usáis el ***geckodrive*** de Mozilla:
    a. **Descargar GeckoDriver:**
        i. Sitio oficial de GeckoDriver en https://github.com/mozilla/geckodriver/releases
        ii. Descarga la versión adecuada para tu sistema operativo
    b. **Extraer el Archivo:**
        i. Al descomprimir btendrás un archivo ejecutable llamado ***geckodriver.exe.***
    c. **Agregar GeckoDriver al PATH (opcional pero recomendado):**
        i. Para ejecutar GeckoDriver desde cualquier ubicación en tu sistema, es recomendable agregar la ubicación del archivo ejecutable al PATH de Windows. Se puede hacer esto agregando la ubicación a la variable de entorno PATH o moviendo el archivo a un directorio que ya esté en el PATH.
    d. **Usar GeckoDriver con Selenium en Python:**
        i. Recuerda establecer la ruta correcta.

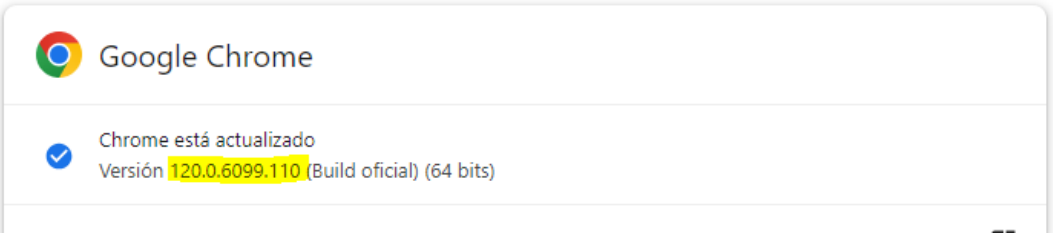
[1] Me parece interesante:
https://www.tutorialselenium.com/2017/07/20/como-usar-localizadores-en-selenium-ide/

```
from selenium import webdriver

# Especifica la ruta al ejecutable de GeckoDriver
geckodriver_path = r'C:\ruta\a\geckodriver.exe'
# /home/loquesea/geckdodriver

# Inicializa el navegador Firefox con Selenium
driver = webdriver.Firefox(executable_path=geckodriver_path)

# Realiza acciones con el navegador...
# Por ejemplo, navega a una URL
driver.get('https://www.ejemplo.com')

# Cierra el navegador al finalizar
driver.quit()
```

---

2. **Si usamos el de Chrome**

    a. **Descargar** (ver apuntes) y verificar la versión correcta en *chrome://settings/help:*



Buscas aquí la versión: https://googlechromelabs.github.io/chrome-for-testing/

    b. El **resto sería igual** que geckodrive y, en cuanto al código inicial sería:

```
from selenium import webdriver

# Especifica la ruta al ejecutable de ChromeDriver

chromedriver_path = r'C:\ruta\a\chromedriver.exe'

# Configura las opciones del navegador (opcional probablemente
no necesario)

chrome_options = webdriver.ChromeOptions()

# Inicializa el navegador Chrome con Selenium

driver = webdriver.Chrome(executable_path=chromedriver_path,
options=chrome_options)

# Por ejemplo, navega a una URL

driver.get('https://www.ejemplo.com')

# Cierra el navegador al finalizar

driver.quit()
```

TAREA 3. Scrapy

Este es un ejemplo sencillo de una araña de Scrapy que extrae citas de http://quotes.toscrape.com, web diseñada para practicar la extracción de datos.

A. Primero instalamos la librería

```
pip install scrapy
```

B. Lanzamos un proyecto Scrapy

```
scrapy startproject quotes_scraper
```

C. Creamos un Spider o *araña* dentro del directorio creado, p.e., ***quotes_scraper.py*** dentro del directorio ***quotes_scraper***

```
import scrapy

class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'texto': quote.css('span.text::text').get(),
                'autor': quote.css('span small::text').get(),
            }

        next_page = response.css('li.next a::attr(href)').get()
        if next_page is not None:
            yield response.follow(next_page, self.parse)
```

Esta “araña” llamada **quotes** que extraerá citas de la web *quotes.toscrape.com*

En la siguiente web, tenemos otro ejemplo, aún más completo, de un scrapy sobre la web de Zara.

https://elmundodelosdatos.com/extraccion-datos-sitios-web-productos-zara/

Práctica.

Teniendo en cuenta lo anterior y toda aquella información relativa al uso de Scrapy que consideres oportuno utilizar, en esta práctica se solicitará información sobre aspectos de la ludopatía y cuya información puedes sacar en uno (no es necesario buscar en todos) de los siguientes foros:

- *Ludopatia.org*. Por la rehabilitación de jugadores patológicos y otras adicciones. Foro de discusión.

https://www.ludopatia.org/forum/default.asp

- *Ludopatía*. Foro de discusión.

https://www.forolinternas.com/viewtopic.php?f=16&t=17505

- *Ludopatía, adicción y problemas con el juego*. Foro de discusión

http://foroapuestas.forobet.com/ludopatia-adiccion-y-problemas-con-el-juego/

La idea es sacar unos cientos de ***post o entradas*** relacionadas con un determinado juego de apuestas, podéis elegir uno cualquiera: bingo, tragaperras, etc…

También podéis pedir que el script nos pida un juego y ejecutarlo en función de la variable entrada.

Otra opción interesante adicional sería ver el número de post que contiene un determinado juego y compararlos entre ellos para analizar o darnos una idea de cuál tiene mayor problemática de adicción.

Puede que la búsqueda duré algún tiempo si el foro es muy grande (solo las salidas en castellano)

Volcar los datos en un fichero ***.json***