David Figueroa

DOCUMENTACIÓN

Proyecto: Web Scrapp

Descripción: Este es el instructivo o documentación sobre el uso de las funciones creadas para facilitar el WebScraping para los Bots que actualmente están hechos, así como consejos prácticos

Introducción

En el siguiente Documento se explicará a detalle las funciones creadas para poder facilitar el scrapeo web basado en la librería selenium.

Archivos

El código donde está la librería es el siguiente:

Scrapeo_HDI.py

Web Scrap

El WebScrap es el finísimo arte de extraer información de sitios web de forma masiva, de esta forma una tarea que resultaría repetitiva se hace de forma automática. Se usa principalmente para la obtención de los precios de las competidoras, los datos que requerimos son las combinaciones por carrocería y por zona para cada aseguradora disponible en el portal.

Cabe aclarar que no todos los sitios web son iguales y dentro de estos hay unos más complicados que otros, algunos elementos se comportan distinto que otros, pero lo importante es tener en claro los conceptos base.

Iniciando Web Scrap

Selenium

La librería o paquetería más importante que usaremos es Selenium, esta librería nos permite interactuar con la página, escoger elementos, darle clic, abrir una lista desplegable, insertar caracteres a cajas de texto, etc.

Hoy en día se recomienda la instalación de la versión 4.11.2 ya que esta es la que mejor se adapta al Chromedriver.

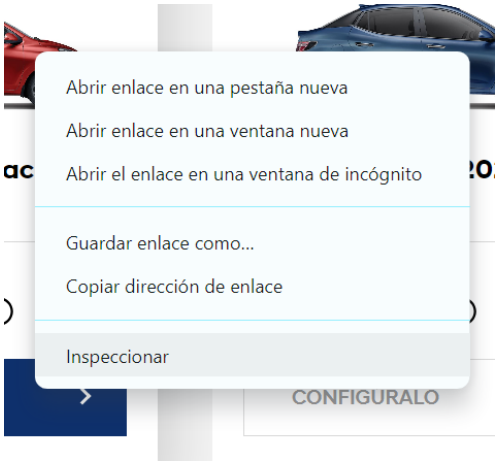
```
import selenium

selenium_version =  selenium.__version__

if selenium_version != '4.11.2':
    %pip install selenium==4.11.2
```

El Chromedriver es un archivo que puedes descargar de Chrome que te permite hacer la conexión con Python y Chrome, este archivo se encuentra en esta carpeta, cabe recalcar que hay distintas versiones de este driver, el mismo Python te dirá cuando esta versión sea diferente y que tienes que actualizarlo.

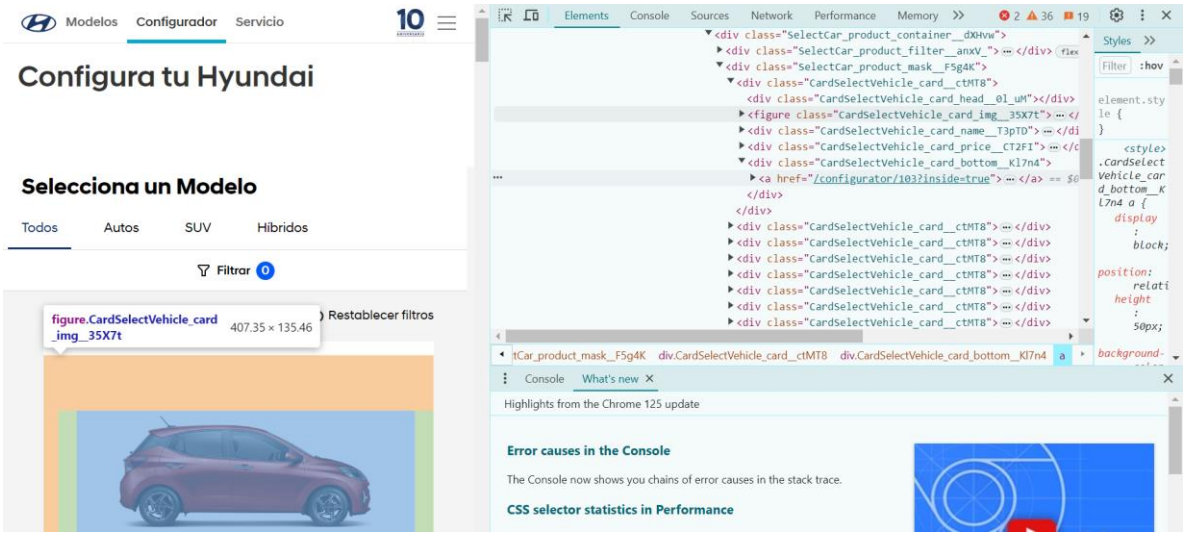
Para empezar el WebScrap lo primero es definir nuestro driver, el driver es el objeto que se conecta a Chrome y direccionando a este objeto haremos todas las funciones necesarias.

```
CO = webdriver.ChromeOptions()
CO.add_experimental_option('excludeSwitches', ['enable-logging'])
CO.add_argument('--ignore-certificate-errors')
CO.add_argument('--start-maximized')
driver = webdriver.Chrome(options = CO)
driver.get("https://www.hyundai.com.mx/configurador-modelos/")
```

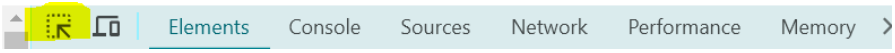
Una vez ya teniendo el driver definido y que veamos que se abrió una nueva pestaña de Chrome en nuestro equipo podemos empezar a interactuar con los objetos de dicha página. Para hacerlo solo basta en darle click derecho a la pagina y seleccionar la opción **Inspeccionar**



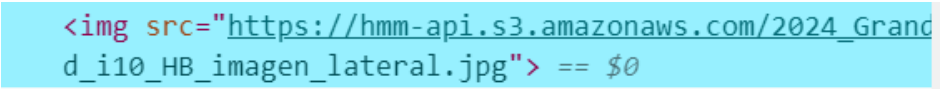
Cuando hagamos esto se abrirá el código fuente de la página:



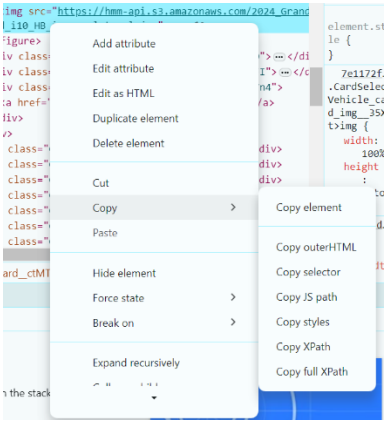
Una vez que el código este visible lo que sigue es seleccionar el elemento con el que buscamos interactuar.



En la esquina superior derecha veremos el símbolo de una flecha, al seleccionarlo entraremos a un “modo de referencia” donde el cursor del equipo servirá como guía, bastara seleccionar el elemento que querramos para que nos de el código al que se referencia este elemento.



Ya que tenemos la parte del código que hace referencia al elemento, le daremos click derecho para obtener la descripción del elemento o un xpath.



Al tener el xpath o un elemento nosotros tendremos la dirección del código que hace referencia la elemento, hay distintas formas de referenciar un elemento según la pagina web (id, class, xpath, cssselector) pero la más común es por xpath, cada una tiene sus ventajas y desventajas y también depende de como este programada la página.

Clases

Limpieza

Definimos clases para la limpieza de los datos que aparecen en el sitio web que posteriormente nosotros tenemos que limpiar, se le pueden agregar más o menos, pero estos son los más comunes.

```python
class Limpieza():
    def __init__(self,dato):
        self.dato = dato
    def precio(self):
        lista = ['PRECIO',':',' ',',','MXN','$',')','(']
        y = self.dato.upper()
        for i in lista:
            y = y.replace(i,'').strip()
        return y
    def cp(self):
        y = (5-len(self.dato))*'0'+self.dato
        return y
    def auto(self):
        y = self.dato.upper()
        palabras = ['SEDAN','COUPE','CONVERTIBLE']
        for i in palabras:
            y = y.replace(i,'').strip()
        return y
    def moneda(self):
        return locale.currency(float(self.dato),grouping=True)

    def Normalizar(self):
        acentos_dict = {"Á":"A","É":"E","Í":"I","Ó":"O","Ú":"U",'á':'a','é':'e','í':'i','ó':'o','ú':'u'}
        y = self.dato
        for con_acento, sin_acento in acentos_dict.items():
            y = y.replace(con_acento, sin_acento)
        return y
```

Wait_Scrap

Unas veces se necesita esperar a la presencia de un elemento (a que la página esté cargada y o el elemento termine de cargarse), este método funciona a veces según la página.

```python
class Wait_Scrap():
    def __init__(self,driver,seconds):
        self.driver = driver
        self.seconds = seconds
        self.wait = WebDriverWait(self.driver, self.seconds)
    def xpath(self,ruta):
        wait = self.wait
        element = wait.until(EC.presence_of_element_located((By.XPATH, ruta)))
        return element
    def class_name(self,ruta):
        wait = self.wait
        element = wait.until(EC.presence_of_element_located((By.CLASS_NAME, ruta)))
        return element
    def css_selector(self,ruta):
        wait = self.wait
        element = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, ruta)))
        return element
```

Find_Scrap

Este método se utiliza para encontrar a un elemento dentro de la página con los métodos mencionados anteriormente.

```python
class Find_Scrap():
    def __init__(self,driver):
        self.driver = driver
    def xpath_p(self,ruta):
        elements  = self.driver.find_elements(By.XPATH,ruta)
        return elements
    def xpath_s(self,ruta):
        element  = self.driver.find_element(By.XPATH,ruta)
        return element
    def css_p(self,ruta):
        elements  = self.driver.find_elements(By.CSS_SELECTOR,ruta)
        return elements
    def css_s(self,ruta):
        element  = self.driver.find_element(By.CSS_SELECTOR,ruta)
        return element
    def class_p(self,ruta):
        elements  = self.driver.find_elements(By.CLASS_NAME,ruta)
        return elements
    def class_s(self,ruta):
        element  = self.driver.find_element(By.CLASS_NAME,ruta)
        return element
    def id_p(self,ruta):
        elements  = self.driver.find_elements(By.ID,ruta)
        return elements
    def id_s(self,ruta):
        element  = self.driver.find_element(By.ID,ruta)
        return element
```

Este método es el más importante ya que es con lo que harás referencia a los objetos para después interactuar con estos.

```python
find_scraper = Find_Scrap(driver)
```

Empezamos definiendo el objeto y después ya sea por xpath o alguna de las otras formas poner el xpath, clase, etc. El prefijo **_s** o **_p** se refiere a uno o muchos (singular/plural) dependiendo el tipo de objeto al que hagamos referencia, usualmente se usa el _s, pero _p es útil cuando quieres interactuar con objetos que sean similares (lista de coches, lista estados, etc) en este caso es más conveniente usar **.class_p,** ya que la búsqueda por clase es más efectiva.

```python
cerrar_cookies = find_scraper.xpath_s('//*[@id="configura-tu-vehículo-|-hyundai-méxico"
cerrar_cookies.click()
```

Find_Wait_Scrap

Esta clase es igual que Find_Scrap, con la diferencia que intentara encontrar el objeto durante un lapso de tiempo hasta que este disponible, es útil con páginas que tienden a ser lentas de forma variable, donde hay veces que tardan un segundo y otras que tardan 10, con esto se soluciona el problema.

```
class Find_Wait_Scrap():
    def __init__(self,driver):
        self.driver = driver
    def xpath_p(self,ruta):
        elements = 0
        for second in range(1,25):
            time.sleep(1)
            try:
                elements = self.driver.find_elements(By.XPATH,ruta)
                break
            except:
                continue
        return elements
```

De Igual forma lo definimos antes de usar:

```
find_wait_scraper =Find_Wait_Scrap(driver)
```

```
versiones = find_wait_scraper.class_p('Configurator_group__v_rv7')
version_0 = [v for v in versiones if version in v.text][0]
```

Click_Wait

De igual forma hay elementos que aunque estén disponibles tardan en poder ser clickeables

```
def Click_Wait(element):
    for s in range(0,25):
        try:
            element.click()
            break
        except:
            time.sleep(1)
```

Otras funciones

Mover el Scroll

Usualmente para interactuar con un objeto este tiene que estar “visible”, por lo que tendremos que hacer scroll hasta que este sea visible.

```
def Mover_ScrollV(p):
    driver.execute_script(f"window.scrollTo(0,{p})")
```

Cambio de frame

Hay veces en las que los elementos con los que querremos interactuar no están visibles del todo, esto se debe a que están dentro de un iframe, lo que haremos será entrar a ese iframe, realizar las operaciones necesarias y después salir del iframe

```
iframe_vehiculos = find_scraper.xpath_s('//*[@id="iFrameResizer0"]')
driver.switch_to.frame(iframe_vehiculos)
# Interactuar con los elementos del iframe
driver.switch_to.default_content()
```

Consideraciones Importantes

- No todos los Bots se hicieron usando esta metodología, pero todos están basados en Selenium, esto se debe a que estas buenas prácticas surgieron después. Los códigos que pueden servir de ejemplo son entre otros:
    - Scrapeo_Huyndai.ipynb
    - Chirey_Fin.ipynb
    - Bot_Autocompara.ipynb
- La mayoría de los códigos tendrían que hacer uso de la librería **os**, ya que esta saca la dirección de cada equipo y con esto mantener un buen orden de las carpetas para que en caso de que se tenga que correr el bot en distintos equipos no tendrán que estar cambiando las direcciones.
- En caso de que haya recaptchas, tendrán que usar nuevos métodos como pyautogui