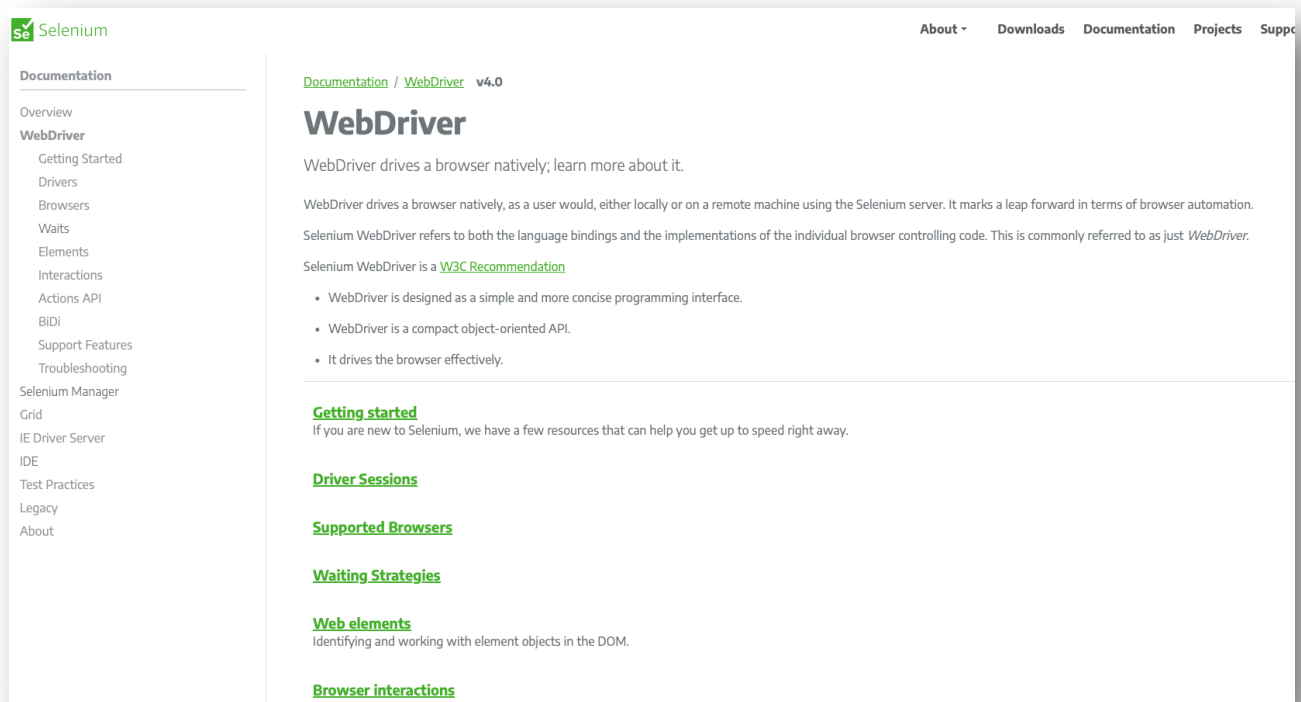


Adquisición datos con Selenium

En ocasiones, hacer WebScrapping sobre páginas web que tengan los datos que nos interesan requiere de interactuar con el navegador para autenticarse mediante usuario y contraseña, rellenar un formulario o desplegar menús. Selenium es la herramienta que usaremos cuando queramos controlar un navegador desde Python.

Siempre es bueno tener a mano la web con la documentación oficial para consultar (<https://www.selenium.dev/documentation/>) porque en muchas ocasiones las herramientas con “ChatGPT” dan soluciones muy complicadas o que no tienen un funcionamiento como se espera.



The screenshot shows the Selenium WebDriver documentation page. The left sidebar contains a navigation menu with links like Overview, WebDriver, Getting Started, Drivers, Browsers, Waits, Elements, Interactions, Actions API, BIDI, Support Features, Troubleshooting, Selenium Manager, Grid, IE Driver Server, IDE, Test Practices, Legacy, and About. The main content area is titled 'WebDriver' and includes an introduction, a list of features (designed as a simple interface, compact API, effective browser control), and links to 'Getting started', 'Driver Sessions', 'Supported Browsers', 'Waiting Strategies', 'Web elements', and 'Browser interactions'.

Instalación

Instalar Selenium es muy sencillo. Usaremos "pip" o "conda" para instalar el conjunto de archivos que conforma la librería y, gracias a estar usando uno de estos gestores de paquetes, mantenerlo actualizado.

```
pip install selenium
```

Recordemos que Selenium usa un navegador para interactuar con las páginas web así que también necesitaremos un "driver" específico de cada navegador. La manera más sencilla de tener disponibles drivers consiste en instalar el "webdriver-manager" que será el encargado de descargar y configurarlos.

```
pip install webdriver-manager
```

De esta manera, para usar un navegador con motor "Chrome" añadiremos las siguientes líneas a nuestro proyecto Python:

```
1. from selenium import webdriver
2. from selenium.webdriver.chrome.service import Service
3. from webdriver_manager.chrome import ChromeDriverManager
4.
5. driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
6.
```

Si preferimos usar un navegador con motor "Gecko" como el usado en el Firefox debemos añadir las siguientes líneas al proyector Python:

```
1. from selenium import webdriver
2. from selenium.webdriver.firefox.service import Service
3. from webdriver_manager.firefox import GeckoDriverManager
4.
5. driver = webdriver.Firefox(service=Service(GeckoDriverManager().install()))
6.
```

De manera idéntica podríamos usar navegadores como los motores de "Edge" u "Opera". El navegador "Safari" no necesita "driver" porque MacOS ya lo incluye en el sistema operativo.

Hola mundo!

Con la instalación de Selenium y el gestor de los drivers para los navegadores ya podemos implementar nuestro primer y más sencillo ejemplo.

Abriremos un navegador y cargaremos la página web de nuestra aula virtual (<https://fpadistancia.edu.xunta.gal/login/index.php>) en un navegador de tipo "Chrome"

```
1. from selenium import webdriver
2. from selenium.webdriver.chrome.service import Service
```

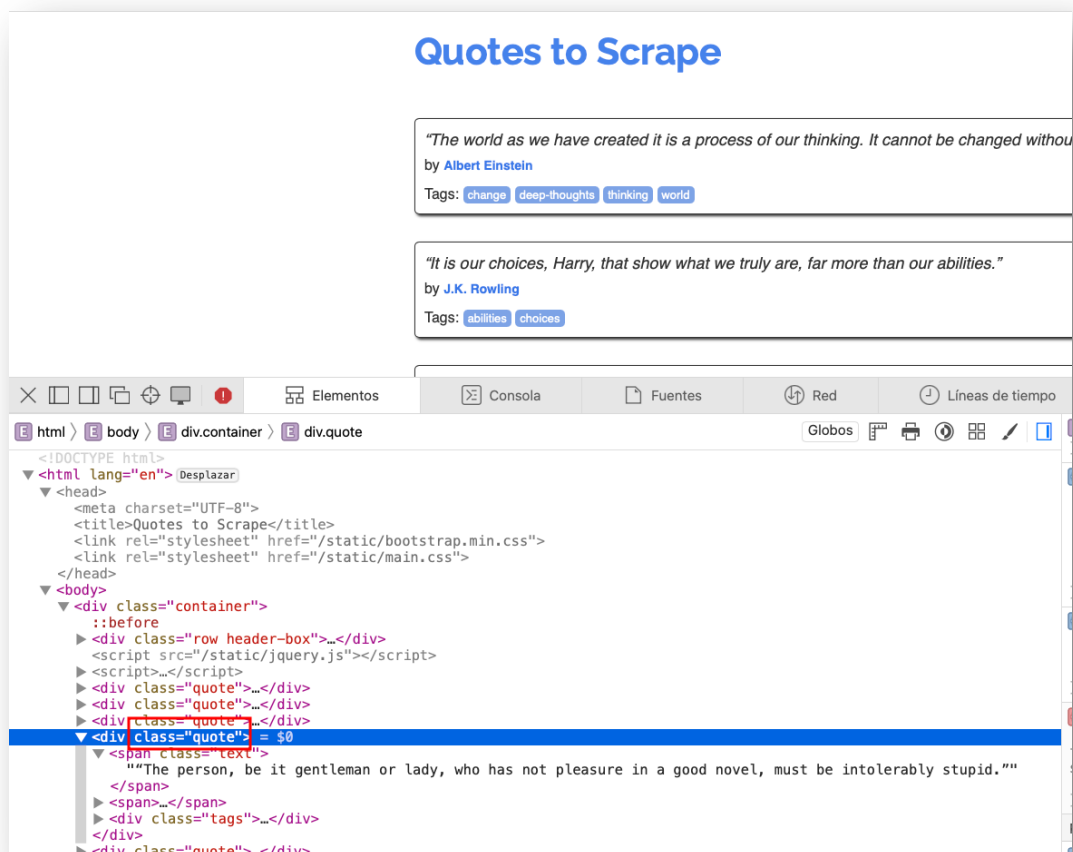
```
3. from webdriver_manager.chrome import ChromeDriverManager
4.
5. driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
6. URL = "https://fpadistancia.edu.xunta.gal/login/index.php"
7. driver.get(URL)
8.
```

Este código simplemente abre un navegador y accede a la URL indicada, algo muy similar a lo que ya hacíamos con la librería "requests". Para saber cuándo utilizar "requests" o "selenium" debemos comprender qué es lo que hace cada una.

"Requests" devuelve el HTML que responde el servidor pero no ejecuta JavaScript ni hace cargas dinámicas. Idealmente usaremos "Requests" para páginas muy estáticas.

"Selenium" simula ser un navegador por lo que es más pesado pero puede interactuar con la web haciendo clics o scrolls y además carga todo el JavaScript por lo que obtenemos una versión "final" de la página.

Por ejemplo, <https://quotes.toscrape.com/js/> es una web que solo muestra citas de personajes famosos pero lo hace cargando un JavaScript con el que va rellenando etiquetas con clase "quote". Si usamos "request" se descargará una web con un esqueleto prácticamente vacío sin ninguna etiqueta con esta clase. En cambio, al usar Selenium, esperaremos a cargar y ejecutar en el navegador todo el JavaScript.



Login en una web

El siguiente paso en Selenium consiste en interactuar con la web haciendo un “login” en alguna web, por ejemplo en nuestra aula virtual. Para ello, además de abrir la web con un determinado navegador tendremos que ser capaz de movernos a través de ella y escribir las credenciales.

Necesitaremos importar la clase “By” desde “selenium.webdriver.common.by” con la línea

```
from selenium.webdriver.common.by import By
```

Con esta librería podemos encontrar un elemento en el HTML de la web usando el método “find_element” del controlador pasándole como parámetros la opción y el valor que buscamos.

Analicemos este código:

```
1. from selenium import webdriver
2. from selenium.webdriver.common.by import By
3. from selenium.webdriver.chrome.service import Service
4. from webdriver_manager.chrome import ChromeDriverManager
5.
6. URL = "https://fpadistancia.edu.xunta.gal/login/index.php"
7. USUARIO = "miusuario"
8. CONTRASEÑA = "micontraseña"
9.
10. driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
11.
12. driver.get(URL"https://fpadistancia.edu.xunta.gal/login/index.php")
13.
14. caja_usuario = driver.find_element(By.ID, "username")
15. caja_usuario.send_keys(USUARIO)
16.
17. caja_contraseña = driver.find_element(By.ID, "password")
18. caja_contraseña.send_keys(CONTRASEÑA)
19.
20. boton_acceder = driver.find_element(By.ID, "loginbtn")
21. boton_acceder.click()
22.
23. driver.quit()
24.
```

En la parte de importaciones de otras librerías encontramos las necesarias para instalar y usar el “WebDriver” para el motor “Chrome” y añadimos “By” para poder seleccionar etiquetas por su “id”.

En la parte de declaraciones simplemente referenciamos la URL y guardamos usuario y contraseña.

En la línea 12 realizamos una petición “GET” de la URL de nuestra aula virtual.

Las líneas 14 y 15 colocan el foco en la etiqueta cuyo “id” es “username”. Previamente he tenido que estudiar la web e inspeccionarla para averiguar esta información. A continuación escribo el nombre de usuario guardado donde el cursor “caja_usuario” apunta.

Las líneas 17 y 18 hacen lo mismo para la contraseña.

En las líneas 20 y 21, localizo el botón de “Enviar” y llamo al método click() sobre él.



En este ejemplo hemos usado la opción de "By.ID" para buscar etiquetas según su "id" pero la clase "By" importada nos proporciona más opciones como:

- By.ID para buscar por "id"
- By.NAME para buscar por "name"
- By.CLASS_NAME para buscar por clase CSS
- By.TAG_NAME para buscar por tipo de etiqueta HTML
- By.LINK_TEXT para buscar enlaces que muestren exactamente un texto
- By.XPATH para buscar usando una ruta XPATH.

En ocasiones no será suficiente con escribir en cajas de texto y hacer clic en botones, también tendremos que interactuar con otros elementos. Normalmente un usuario pincharía en algún lugar y recorrería las opciones usando las flechas. Para este y otros casos similares tenemos la clase "Keys" que podemos importar con:

```
from selenium.webdriver.common.keys import Keys
```

Esta importación nos dará acceso a poder enviar teclas especiales como:

- Keys.ALT envía la tecla "alt"
- Keys.ARROW_UP para enviar una tecla de flecha arriba. Abajo, izquierda y derecha funcionan con su nombre significativo en inglés exactamente igual.
- Keys.BACKSPACE para borrar un carácter a la izquierda.
- Keys.CONTROL envía la tecla "control"
- Keys.ESCAPE para el "Esc"
- Keys.RETURN para un "Enter"
- Keys.SHIFT tecla "shift"
- Keys.SPACE para la barra espaciadora
- Keys.TAB para el tabulador
- Y muchas otras otra que hay que consultar en la documentación oficial.

Además estas teclas especiales se pueden combinar como por ejemplo:

```
1. Elem.send_keys(Keys.CONTROL, 'a')
2. Elem.send_keys(Keys.CONTROL, 'c')
```

En el código anterior estoy enviando un Ctl+a para seleccionar todo el contenido y un Ctl+c para copiar ese contenido en el portapapeles.

Interactuar con desplegables

Pondremos un ejemplo de cómo interactuar con un desplegable usando la web de pruebas (<https://the-internet.herokuapp.com/dropdown>). En este caso concreto de los desplegables, Selenium proporciona herramientas específicas para trabajar con las etiquetas "select" de los desplegables.

La primera y menos aconsejada alternativa es interactuar con el desplegable mediante clics, flechas e Intros. Esto se debe a que no todos los drivers de navegadores funcionan de la misma manera y es altamente probable que falle cuando nos intentamos mover a través de las opciones pulsando flechas.

La segunda opción es usar "find_elements" (fijémonos que es un plural por lo que nos devuelve un array) para obtener todas las opciones y finalmente hacemos clic en la que nos interesa.

```
1. from selenium import webdriver
2. from selenium.webdriver.common.by import By
3. from selenium.webdriver.common.keys import Keys
4. from selenium.webdriver.chrome.service import Service
5. from webdriver_manager.chrome import ChromeDriverManager
6.
7. driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
8. driver.get("https://the-internet.herokuapp.com/dropdown")
9.
10. dropdown = driver.find_element(By.ID, "dropdown")
11.
12. dropdown.click()
13.
14. opciones = driver.find_elements(By.TAG_NAME, "option")
15.
16. opciones[2].click()
17.
18. driver.quit()
19.
```

La tercera y mejor opción consiste en utilizar la clase "Select" que proporcionan los webdrivers que nos permitirá seleccionar la opción que más interese usando métodos como:

- Select_by_index() para elegir la opción ordenada por su índice.
- Select_by_visible_text() en la que elegimos aquella cuyo texto coincida
- Select_by_value() donde nos quedamos con aquella cuyo valor del atributo "value" coincida.

```
1. from selenium import webdriver
2. from selenium.webdriver.common.by import By
3. from selenium.webdriver.support.ui import Select
4. from selenium.webdriver.chrome.service import Service
5. from webdriver_manager.chrome import ChromeDriverManager
6.
7. driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
8. driver.get("https://the-internet.herokuapp.com/dropdown")
9.
10. dropdown = driver.find_element(By.ID, "dropdown")
11.
12. select = Select(dropdown)
13. select.select_by_index(2)
14.
15. driver.quit()
16.
```

Selenium avanzado

Los métodos que proporciona la clase "webdriver" permiten navegar entre páginas con el método "get(URL)" que ya hemos visto pero también con otros como "back()", "forward()" o "refresh()".

También hay métodos para interactuar con la página como "find_element(by, value)", "find_elements(by, value)", "click()" o "send_keys(keys)" que ya hemos visto.

Otros métodos proporcionan información de la página como "get_title()" que nos devuelve el título de la web o "page_source()" que nos devuelve el código fuente final en HTML

Hay métodos para movernos entre ventanas o pestañas como "switch_to_frame(id)" para entrar en un "iframe" concreto dentro de la web.

Otros métodos sirven para el manejo de ventanas emergentes como "switch_to_alert()" que pone el foco en la ventana de alerta o "alert.accept()" que hace un equivalente a pulsar en "Aceptar"

Hay más métodos para manejar cookies o el propio navegador, esperas para todos los elementos estén presentes o se cumpla una condición, y también para realizar acciones táctiles como por ejemplo desplazar pero todas ellas quedan fuera del alcance introductorio de esta guía. La documentación completa está en <https://www.selenium.dev/documentation/>

Selenium + BeautifulSoup

Aunque hay métodos más "elegantes" para combinar ambas herramientas, en esta guía vamos a aplicar un enfoque más didáctico. La idea es usar Selenium para llegar hasta la web que publica los datos cuando este acceso requiere cierta participación por parte del usuario, principalmente respondiendo a algún desafío como un usuario y contraseña o tal vez tener que rellenar algún formulario y enviarlo.

Con selenium debemos asegurarnos de tener la web con la información. Para ello podemos hacer un "sleep()" durante unos segundos para asegurarnos que la web se ha cargado. También podríamos llamar al método "WebDriverWait(driver,time).until(condición)" para forzar la espera hasta que se aparezca un elemento específico de la web final.

Una vez tengamos claro que la web se ha cargado correctamente descargamos el código HTML con "driver.page_source()" y seguimos con BeautifulSoup desde ahí como si acabáremos de hacer la petición de la web.

```
1. from selenium import webdriver
2. from selenium.webdriver.common.by import By
3. from selenium.webdriver.common.keys import Keys
```



```

4. from selenium.webdriver.chrome.service import Service
5. from webdriver_manager.chrome import ChromeDriverManager
6. from selenium.webdriver.support.ui import WebDriverWait
7. from selenium.webdriver.support import expected_conditions as EC
8. import time
9. from bs4 import BeautifulSoup
10.
11. # Iniciar el navegador con Selenium
12. driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
13.
14. # Abrir la página de inicio de sesión
15. url = "https://ejemplo.com/login" # Cambia la URL por la de tu sitio
16. driver.get(url)
17.
18. # Localizar los campos de usuario y contraseña, y el botón de inicio de sesión
19. username_field = driver.find_element(By.ID, "username") # Cambia el ID por el de tu página
20. password_field = driver.find_element(By.ID, "password") # Cambia el ID por el de tu página
21. login_button = driver.find_element(By.ID, "login_button") # Cambia el ID por el de tu página
22.
23. # Introducir las credenciales
24. username_field.send_keys("tu_usuario")
25. password_field.send_keys("tu_contraseña")
26.
27. # Hacer clic en el botón de inicio de sesión
28. login_button.click()
29.
30. # Esperar a que la página de destino cargue después de iniciar sesión
31. WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "elemento_de_post_login")))
32. # Cambia por un elemento visible en la página de destino
33.
34. # Obtener el HTML de la página actual
35. page_html = driver.page_source
36.
37. # Usar BeautifulSoup para analizar el HTML
38. soup = BeautifulSoup(page_html, "html.parser")
39.
40. # Ahora puedes analizar el contenido de la página
41. print(soup.prettify()) # Muestra el HTML con una mejor estructura
42.
43. # Aquí puedes extraer lo que necesites con BeautifulSoup, por ejemplo:
44. contenido = soup.find_all("div", class_="contenido") # Ejemplo de extracción de contenido
45. for item in contenido:
46.     print(item.text)
47.
48. # Cerrar el navegador cuando hayas terminado
49. driver.quit()
50.

```

Este es el resultado de un código de "ChatGPT" bajo el prompt de "Escribe un código sencillo que combine Selenium y BeautifulSoup". Evidentemente el código da por supuesto URL y estructura de las páginas, es decir, no es un ejemplo aplicable en el mundo real pero sirve para ver la estructura del programa y además es fácilmente adaptable a nuestros propios proyectos.

Empezamos en la línea 12 creando el controlador para simular estar usando un navegador con motor Chrome. Toda esa línea nos garantiza que si no tenemos el driver instalado nos lo descargará de internet y lo configurará. A efectos prácticos podemos decir que esa variable apunta a nuestro navegador virtual.

La línea 16 abre la web indicada en nuestro navegador virtual referenciado por la variable "driver".

Las líneas 19, 20 y 21 localizan en la web las etiquetas relacionadas con los cuadros de texto donde debemos poner usuario y contraseña, así como el botón de enviar. Todas ellas las está localizando

mediante un "id" propio pero recordemos que debemos inspeccionar y analizar la web para determinar cómo podemos llegar hasta esos elementos. Básicamente será mediante una ruta XPATH que nos indique el orden en el que vamos desplegando elementos hasta llegar a lo que nos interese o bien, como en este caso, algún atributo de la etiqueta.

En la línea 24 y 25 simplemente rellenamos las cajas de texto con las cadenas de ejemplo para, en la línea 28 hacer clic en el botón.

La línea 31 hace esperar explícitamente hasta 10 segundos como máximo en espera de se cumpla la condición de que en la nueva página aparezca la etiqueta con "id" " igual a "elemento_de_post_login".

Por último, en la línea 35, Selenium devuelve el HTML de la nueva web completa. A partir de aquí entramos en los dominios de BeautifulSoup para la gestión de ese HTML empezando por indicarle que se trata de un código HTML en la línea 38 al instanciar la variable "soup" con la que ya seguiremos trabajando en BeautifulSoup.