Hoja informativa: Selenium

Selenium WebDriver es un controlador de navegador. Puedes usarlo para implementar código que controle al navegador y para emular las acciones de usuario.

Selenium es capaz de emular la mayoría de las acciones de usuario en el navegador:

- Hacer clic en un elemento: un botón, un botón de opción, una casilla de verificación, una lista desplegable.
- Rellenar campos de entrada.
- Navegar entre páginas: retroceder y avanzar, abrir URL.
- Navegar por la página: actualizar la página o desplazarse.

Abrir y cerrar páginas

Cada prueba automatizada de Selenium comienza con dos pasos: crear un controlador e importar los paquetes necesarios.

Importar paquetes

Para usar los comandos de Selenium, debes importar el paquete Selenium WebDriver:

```
from selenium import webdriver
```

Crear un controlador

Se crea un controlador específicamente para el navegador en el que quieres ejecutar las pruebas automatizadas. Por ejemplo, este código crea un controlador para Google Chrome, Firefox e Internet Explorer:

```
driver = webdriver.Chrome() # Google Chrome
driver = webdriver.Firefox() # Firefox
driver = webdriver.Ie() # Internet Explorer
```

Ajustes especiales

Junto con el controlador, puedes agregar ajustes personalizados. Esto es opcional, pero útil.

```
chrome_options = webdriver.ChromeOptions() # Crea un objeto para la configuración
chrome_options.add_argument('--headless') # Ejecuta el navegador desde la terminal sin una interfaz gráfica
chrome_options.add_argument('--window-size=640,480') # Ajusta el tamaño de la ventana a 640 x 480 pixeles
driver = webdriver.Chrome(options=chrome_options) # Crea un controlador y pasa la configuración de los ajustes establecidos
```

Estas son algunas de las opciones:

- `-headless` inicia el navegador sin mostrar la ventana. Esto es útil cuando inicias el navegador en sistemas sin un componente gráfico.

- `-window-size` inicia el navegador con el tamaño de ventana especificado. Esta opción es útil cuando pruebas interfaces en sistemas con una resolución de pantalla en particular.

Abrir páginas

Para abrir una página en el navegador, necesitas el método `get()`. Pásale la página que quieres abrir como argumento:

```
# Import Selenium WebDriver
from selenium import webdriver

# Crea un controlador
driver = webdriver.Chrome()

# Abrir una página
driver.get('https://google.com/')
```

Cerrar páginas

Una vez finalizada la prueba, es necesario cerrar el navegador. Si no lo haces, es posible que algunos procesos en segundo plano no se cierren correctamente. Esto puede provocar una fuga de datos o un error de acceso.

Así es como se cierra el navegador:

```
driver.quit()
```

Cómo obtener una URL de un sitio

A veces, necesitas comprobar la URL de la página en la que te encuentras actualmente. El primer paso es obtener la URL de la página. Para hacerlo, utiliza este método:

```
current_url = driver.current_url
```

La URL se ha guardado en `current_url`.

Para comprobar que esta es la URL que necesitas, usa `assert`. Por ejemplo, así es como compruebas que estás actualmente en la página `[ google.com ]`:

```
assert current_url == 'https://google.com/'
```

Si necesitas probar una página solo una vez durante todo el programa, no tienes que guardar su URL en una variable:

```
assert driver.current_url == 'https://google.com/'
```

También puedes comprobar solo una parte de una URL. Por ejemplo, puedes comprobar que la URL contiene `google.com` mediante el operador `in`:

```
assert 'google.com' in driver.current_url
```

Búsqueda de elementos

Selenium ofrece dos métodos para buscar elementos, `findElement()` y `findElements()`:

- `findElement()` se utiliza para encontrar un solo elemento. Si el método encuentra varios elementos, devolverá solo el primero. El método `find_element()` devuelve un objeto `WebElement`.
- `findElements()` se usa para encontrar varios elementos. Devuelve los elementos como una lista: `List<WebElement>`.

Necesitarás argumentos para ambos métodos. Entre paréntesis, especifica el punto de referencia que utilizará Selenium al buscar un elemento. Podría ser el ID de un botón u otros atributos, como XPath o una etiqueta HTML. La clase `By` será útil aquí.

Class By

Esta clase ayuda a especificar los criterios de búsqueda de los elementos:

```
By.CLASS_NAME # Buscar por nombre de la clase
By.CSS_SELECTOR # Buscar por selector CSS
By.ID # Buscar por atributo ID
By.LINK_TEXT # Buscar por el texto del enlace (no el enlace https:// en sí, sino el texto dentro del objeto del enlace;
 más sobre eso en la próxima lección)
By.PARTIAL_LINK_TEXT # Buscar por una parte del texto del enlace, similar a By.linkText(text)
By.NAME # Buscar por el atributo name
By.TAG_NAME # Buscar por la etiqueta HTML
By.XPATH # Buscar por XPath
```

Para usar los métodos de la clase `By`, primero debes importarla:

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

Esto es lo que obtendrás:

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

driver = webdriver.Chrome()
driver.get("https://www.saucedemo.com/v1/")

# Para buscar un elemento
driver.find_element(By.XPATH, ".//img")

# Para buscar un grupo de elementos
driver.find_elements(By.XPATH, ".//input")
```

Hacer clic en elementos

Selenium tiene un método especial para hacer clic llamado `click()`. Para hacer clic en un elemento, primero tienes que encontrarlo.

Así es como se ve en el código:

```
driver.find_element(By.LINK_TEXT, "Iniciar sesión").click()
```

Puedes dividir esta sentencia en dos partes. Primero, encuentras el elemento, luego haces clic en él:

```
element = driver.find_element(By.LINK_TEXT, "Iniciar sesión")
element.click()
```

El uso de las esperas

Selenium ofrece **esperas** (waits): segmentos de código que interrumpen la prueba durante un tiempo. Las esperas pueden ser explícitas o implícitas.

Las **esperas explícitas** detienen la prueba durante un período de tiempo exacto. Puedes establecer esperas explícitas utilizando la clase `WebDriverWait`. Primero, importa esta clase:

```
from selenium.webdriver.support.wait import WebDriverWait
```

A continuación, simplemente escribe el nombre de la clase seguido de paréntesis con el controlador y el tiempo de espera:

```
from selenium.webdriver.support.wait import WebDriverWait
# Pausar la prueba durante 3 segundos
WebDriverWait(driver, 3)
```

Para personalizar tus esperas, puedes establecer condiciones de espera específicas. Puedes establecer una condición utilizando la clase `expected_conditions`. Especifica primero la clase, seguida de un punto, y luego la condición en sí. Estas son las condiciones más utilizadas:

- `element_to_be_clickable`: se puede hacer clic en el elemento.
- `presence_of_element_located`: el elemento está presente en la página.
- `visibility_of_element_located`: el elemento está presente en la página y es visible.

Por ejemplo, "hasta que se pueda hacer clic en el elemento": `expected_conditions.element_to_be_clickable`.

Se establece una condición de espera usando el método `until()`:

```
# Esperar a que se pueda hacer clic en el botón durante no más de 3 segundos
WebDriverWait(driver, 3).until(expected_conditions.element_to_be_clickable((By.TAG_NAME, "button")))
```

Utiliza esperas explícitas cuando busques elementos específicos y necesites esperar a que se carguen.

Selenium también cuenta con **esperas implícitas**.

Se establecen esperas explícitas cada vez que es necesario detener la prueba. Las esperas implícitas se emplean solo una vez, utilizando el método `implicitly_wait()`. Ahora, todos los comandos que sigan este método irán acompañados de una espera:

```
# Espera implícita de 3 segundos
driver.implicitly_wait(3)
driver.find_element(By.TAG_NAME, "button")
```

Selenium buscará el elemento durante no más de 3 segundos: `implicitly_wait(3)`. La espera se agrega una vez y luego se aplica automáticamente. Si escribes otro método, también se ejecutará con una espera.

Si el elemento no se encuentra en 3 segundos, el programa generará un error: `NoSuchElementException: Message: no such element: Unable to locate element` (Mensaje: no existe tal elemento: No se puede localizar el elemento).

Es mejor utilizar esperas explícitas, ya que son más fáciles de gestionar. Usa esperas implícitas cuando se espera que el elemento aparezca en el DOM.

> ❗ Evita mezclar esperas implícitas y explícitas en tu código, ya que esto puede generar condiciones conflictivas.

Rellenar campos de entrada

Para completar un campo, utiliza el método `send_keys()`. Especifica el valor de entrada entre paréntesis: `send_keys("Comprar un hámster")`.

```
driver.find_element(By.TAG_NAME, "textarea").send_keys("Python")
```

A veces, primero es necesario borrar el campo antes de ingresar algo. Para hacerlo, usa el método `clear()`. Tienes que encontrar el elemento y agregar `.clear()`.

```
driver.find_element(By.TAG_NAME, "textarea").send_keys("Python")
driver.find_element(By.TAG_NAME, "textarea").clear()
driver.find_element(By.TAG_NAME, "textarea").send_keys("python")
```

Obtener el texto de un elemento

Para obtener el texto de un elemento, utiliza el método `text()`.

```
driver.find_element(By.CLASS_NAME, "card-section").text
```

Desplazar elementos a la vista

Si Selenium tiene que hacer algo con un elemento en particular, desplaza el elemento a la pantalla de visualización.

Si este desplazamiento automático no es suficiente, puedes utilizar el método que desplaza el elemento a la vista. El método `execute_script()` te ayudará en este caso.

```
element = driver.find_element(By.ID, "root")
driver.execute_script("arguments[0].scrollIntoView();", element)
```

Vamos a desglosar esto:

- Busca el elemento de destino: `find_element(By.ID, "root")`. Guárdalo en la variable `element`.
- Aplica `execute_script()` al objeto `driver`.
- Entre paréntesis, escribe un **script** que realice el desplazamiento: `"arguments[0].scrollIntoView();”`.

- El segundo argumento entre paréntesis es el elemento de destino: `element`.