«Talento Tech»

Automation Testing

Clase 8

Buenos Aires Ciudad

Clase N° 8: Localización de Elementos y Acciones en Selenium

Temario

- **¿Por qué y cómo localizar elementos?** (ID, Name, CSS, XPath)
- **Interacciones básicas**: `click`, `send_keys`, `clear`, `text`, `get_attribute`
- **Manejo de formularios (login en Sauce Demo)**
- **Esperas implícitas vs. explícitas** – sincronización real
- **Contexto TalentoLab** – tarea solicitada por Matías y Silvia
- Introducción al **Proyecto de Pre-Entrega**

Objetivos de la clase

En esta sesión profundizaremos en la importancia de localizar con precisión los elementos dentro de una página web, analizando los posibles errores y consecuencias que pueden surgir si esta tarea no se realiza correctamente. Practicaremos diversas estrategias de localización y aprenderemos a elegir la más adecuada según el contexto, lo que nos permitirá ejecutar acciones reales como clics, escritura y lectura de elementos, comprendiendo en cada caso qué valida exactamente la automatización. Como ejercicio práctico, completaremos el formulario de login de Sauce Demo de principio a fin, incorporando también técnicas de sincronización mediante esperas para evitar resultados erróneos o inconsistentes.

¿Por qué necesitamos localizar elementos?

Cuando automatizamos navegadores, **“ver”** un elemento es el primer paso para **controlarlo** o **verificarlo**:

1. **Interactuar** – Sin encontrar el botón *Login*, no podemos presionarlo.
2. **Validar** – Para afirmar que aparece el mensaje “Productos”, debemos identificar su `<div>`.
3. **Sincronizar** – Saber que un elemento existe o está visible nos indica que la página cargó.
4. **Mantenimiento** – Selectores claros facilitan actualizar scripts cuando cambia el DOM.

> Piensa en Selenium como un robot que necesita "coordenadas"; los selectores son su mapa.

Estrategia	Ejemplo HTML	Selector	¿Cuándo preferirla?
ID	`<input id="user-name">`	`By.ID, 'user-name'`	Atributo único y estable
Name	`<input name="password">`	`By.NAME, 'password'`	Formularios clásicos
CSS	`<input type="submit" class="btn_primary">`	`input.btn_primary`	Combinar etiqueta-clase-atributo
XPath	`<div class="error">…`	`//div[@class='error']`	Ausencia de ID/Name, búsquedas por texto

Regla de oro: intenta primero **ID** → **Name** → **CSS** corto → **XPath** solo cuando lo anterior no sea posible.

Interacciones básicas — *¿Para qué las usamos?*

Una vez localizado el elemento, **queremos imitar lo que haría un usuario**. Cada método se corresponde con un gesto humano y con una *prueba* concreta.

Método	¿Qué hace?	Situación de ejemplo
`send_keys()`	Escribe texto como el teclado	Llenar "Username" con standard_user
`clear()`	Borra el contenido del input	Reiniciar el campo "Search" antes de un nuevo término
`click()`	Presiona el elemento	Pulsar Add to cart en el primer ítem
`.text`	Lee el texto visible	Verificar que el título sea Products después del login
`get_attribute()`	Obtiene el valor de un atributo HTML	Comprobar si un checkbox tiene checked="true"

Ejemplo práctico

```
boton_login = driver.find_element(By.CSS_SELECTOR, 'input[type="submit"]')
boton_login.click()  # 1) Acción: click

# 2) Verificación inmediata
assert "/inventory.html" in driver.current_url

# 3) Lectura de texto
header = driver.find_element(By.CSS_SELECTOR, '.title').text
assert header == 'Products'
```

Cada línea responde a una **pregunta de prueba**: ¿se hizo clic? ¿cambió la URL? ¿apareció el título correcto?

Manejo del formulario de login

¿Qué estamos testeando?

- **Funcionalidad**: que un usuario válido pueda acceder al inventario.
- **Flujo**: que la aplicación redirija a `/inventory.html` y cambie el título a "Swag Labs".
- **Feedback**: que, si fallara, aparezca un mensaje de error (lo veremos más adelante).

Resultado esperado: después de enviar `standard_user / secret_sauce`, el navegador debe mostrar la lista de productos sin errores.

¿Por qué lo hacemos?
El login es la *puerta de entrada*; si falla, todo el resto queda bloqueado. Validarlo temprano evita perder tiempo en pasos posteriores.

👉Acceso al repositorio:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import os

# Configuración del driver de Chrome
def setup_driver():
    """Configura y devuelve una instancia del WebDriver de Chrome."""
    chrome_options = Options()
    # Opciones para ejecución en entornos CI o sin interfaz gráfica
     # chrome_options.add_argument("--headless")  # Descomenta para modo
headless
    chrome_options.add_argument("--no-sandbox")
    chrome_options.add_argument("--disable-dev-shm-usage")

    # Crear el servicio de Chrome
    # Especifica la ruta al chromedriver si es necesario
    # service = Service('/ruta/a/tu/chromedriver')
    service = Service()

    # Inicializar el driver
    driver = webdriver.Chrome(service=service, options=chrome_options)
    driver.maximize_window()
```

```
    return driver

def test_login_saucedemo():
    """Prueba el inicio de sesión en SauceDemo y verifica el carrito de compras."""
    # Inicializar el driver
    driver = setup_driver()

    try:
        # Abrir la página de inicio de sesión
        driver.get("https://www.saucedemo.com/")

        # Esperar a que se cargue el formulario de login
        username_input = WebDriverWait(driver, 10).until(
            EC.visibility_of_element_located((By.ID, "user-name"))
        )

        # Ingresar credenciales
        username_input.send_keys("standard_user")
        driver.find_element(By.ID, "password").send_keys("secret_sauce")

        # Hacer clic en el botón de login
        driver.find_element(By.ID, "login-button").click()

         # Verificar que el login fue exitoso comprobando que estamos en la página de productos
        WebDriverWait(driver, 10).until(
                        EC.visibility_of_element_located((By.CLASS_NAME, "inventory_item"))
        )
        print("✅ Login exitoso!")

        # Añadir un producto al carrito
            driver.find_element(By.XPATH,  "//button[contains(@data-test, 'add-to-cart')]").click()

          # Esperar y verificar que aparezca el badge del carrito de compras
        badge = WebDriverWait(driver, 10).until(
                        EC.visibility_of_element_located((By.CLASS_NAME, "shopping_cart_badge"))
        )
```

```
        # Verificar que el contador del carrito muestra 1
            assert badge.text == "1", f"El contador del carrito debería mostrar 1, pero muestra {badge.text}"
        print("✅ Producto añadido al carrito correctamente!")

         # Opcional: hacer clic en el carrito y verificar que contiene el producto
        driver.find_element(By.CLASS_NAME, "shopping_cart_link").click()
        cart_item = WebDriverWait(driver, 10).until(
                        EC.visibility_of_element_located((By.CLASS_NAME, "cart_item"))
        )
        print("✅ Producto visible en el carrito!")

        # Cerrar sesión
        driver.find_element(By.ID, "react-burger-menu-btn").click()
        WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.ID, "logout_sidebar_link"))
        ).click()

        # Verificar que volvimos a la página de login
        WebDriverWait(driver, 10).until(
            EC.visibility_of_element_located((By.ID, "login-button"))
        )
        print("✅ Cierre de sesión exitoso!")

        print("✅ ¡Todas las pruebas pasaron con éxito!")
        return True

    except Exception as e:
        print(f"❌ Error durante la prueba: {e}")
        return False
    finally:
        # Cerrar el navegador al finalizar
        print("Cerrando el navegador...")
        driver.quit()

if __name__ == "__main__":
    test_login_saucedemo()
```

Recuerda que para ejecutar la prueba debemos escribir en consola:
`python3 test_login.py`

Esperas implícitas vs. explícitas

La sincronización es crítica: **no todas las páginas cargan a la misma velocidad**.

Espera implícita (*global)*

`driver.implicitly_wait(5)` **# segundos**

- Se configura **una sola vez**.
- WebDriver reintenta cualquier `find_element` cada 0,5 s hasta 5 s.
- **Ideal** como red de seguridad en scripts sencillos o al inicio del proyecto.

Espera explícita

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

wait = WebDriverWait(driver, 10)
badge = wait.until(EC.visibility_of_element_located((By.CLASS_NAME,
'shopping_cart_badge')))
```

- Se aplica **solo donde hace falta** y con condiciones precisas (visibilidad, clickeable, texto, URL).
- **Ideal** para pasos críticos o cuando sabemos que cierto elemento tarda en aparecer.

Uso recomendado	Implicita	Explicita
Primeros scripts – evitar flakiness general	✔️	❌ (no indispensable)
Paso que depende de AJAX/lento	❌	✔️
Proyecto grande con muchas páginas	⚠️ (puede alargar todo)	✔️

No mezcles demasiadas implícitas y explícitas con tiempos altos; pueden acumularse y frenar el test.

Próximos pasos

En la **Clase 9** abordaremos **Page Object Model** para escalar nuestros tests y evitar duplicación de código. ¡Practica hasta que tus scripts sean estables y fáciles de leer!

Creciendo en TalentoLab

Después de varias clases aprendiendo sobre selectores, interacción y automatización, el equipo de desarrollo de TalentoLab liberó una nueva versión del portal. Antes de avanzar con pruebas más complejas, tenemos que asegurarnos de que lo esencial —el login y la navegación básica— **funcione perfectamente de forma automatizada**.

Daily del lunes: tareas asignadas

Silvia (PO):

“Quiero validar el flujo de login, la lista de productos y que se pueda agregar al carrito correctamente en SauceDemo. Si falla algo ahí, todo lo demás pierde sentido.”

Matías (Automation Lead):

“Usá localización por `ID`, `Name` y `CSS` para esos tres elementos clave. Agregá al menos una espera **explícita** (con WebDriverWait) para el badge del carrito. Si todo sale bien, mostrá `Test OK` en consola.”

Ejercicios Prácticos

Tu misión es crear **un solo script completo** en Python con Selenium que cumpla el siguiente flujo sobre https://www.saucedemo.com:

¿Qué debe hacer nuestro Script?:

1. **Abrir la página de login.**
2. **Completar usuario y contraseña**:
    - Usuario: `standard_user`
    - Contraseña: `secret_sauce`
3. **Hacer clic en el botón de login.**
4. **Validar que el login fue exitoso**:
    - Que la URL contenga `/inventory.html`
    - Que el título sea `Swag Labs` o que aparezca la palabra `Products`

5. **Agregar el primer producto al carrito**.
6. **Esperar explícitamente** a que aparezca el **badge** del carrito (`.shopping_cart_badge`) y confirmar que dice `1`.
7. (Opcional pero recomendado): ingresar al carrito y verificar que el producto añadido esté en la lista.
8. **Mostrar `Test OK` en consola si todo se completó con éxito.**

Consignas de Pre-Entrega de Proyecto.

Automatización de Login y Navegación Básica

El objetivo del Proyecto de Pre-Entrega es que apliques los conocimientos adquiridos hasta la Clase 8 del curso, demostrando tu capacidad para automatizar flujos básicos de navegación web utilizando Selenium WebDriver y Python. Este proyecto te permitirá poner en práctica lo aprendido sobre interacción con elementos web, estrategias de localización y validación de estados en una página. El sitio objetivo para esta automatización será saucedemo.com, una aplicación web demo especialmente diseñada para prácticas de testing.

Requerimientos Específicos:

Tecnologías a Utilizar:

- Python como lenguaje de programación
- Pytest como framework de testing
- Selenium WebDriver para automatización de interfaces web
- Git para control de versiones
- GitHub como repositorio de código

Organización del Código:

- Organiza tu proyecto con al menos 2 archivos separados (por ejemplo, un archivo para pruebas y otro para funciones auxiliares)
- Incluye comentarios descriptivos en el código para facilitar su comprensión
- Utiliza nombres significativos para variables, métodos y clases

Funcionalidades Específicas:

1. Automatización de Login:

- **Caso de Prueba de Login**: **(Clases 6 a 8)**
    - Navegar a la página de login de saucedemo.com
    - Ingresar credenciales válidas (usuario: "standard_user", contraseña: "secret_sauce")
    - Validar login exitoso verificando que se haya redirigido a la página de inventario

2. Navegación y Verificación del Catálogo: (Clases 6 a 8)

- **Caso de Prueba de Navegación**:
    - Verificar que el título de la página de inventario sea correcto
    - Comprobar que existan productos visibles en la página (al menos verificar la presencia de uno)

    - Validar que elementos importantes de la interfaz estén presentes (menú, filtros, etc.)

3. Interacción con Productos: (Clase 8)

- **Caso de Prueba de Carrito**:
    - Añadir un producto al carrito haciendo clic en el botón correspondiente
    - Verificar que el contador del carrito se incremente correctamente
    - Navegar al carrito de compras
    - Comprobar que el producto añadido aparezca correctamente en el carrito

Control de Versiones y Documentación: (Clase 3)

Repositorio en GitHub:

- Sube el proyecto a un repositorio en GitHub
- Realiza commits frecuentes y con mensajes descriptivos que muestren el progreso del proyecto

README.md:

- Incluye un archivo README.md que explique:
    - El propósito del proyecto
    - Las tecnologías utilizadas
    - Cómo instalar las dependencias
    - Cómo ejecutar las pruebas

Generar reporte en HTML de las pruebas realizadas:

pytest pre-entrega-final/test_saucedemo.py -v --html=reporte.html

Funcionalidad Esperada:

- Los casos de prueba deben ejecutarse correctamente en el sitio saucedemo.com
- Las validaciones deben ser claras y específicas para cada paso
- El código debe ser legible y estar bien organizado
- Los tests deben ser independientes entre sí (la falla de uno no debe afectar a los demás)

Entregables:

- Repositorio público en GitHub con todo el código del proyecto.
- Archivo `README.md` que incluya:
    - Propósito del proyecto
    - Tecnologías utilizadas
    - Instrucciones de instalación de dependencias
    - Comando para ejecutar las pruebas (por ejemplo: `pytest -v --html=reporte.html`)

- Reporte HTML generado por Pytest con resultados de la ejecución.
- Evidencias adicionales: capturas de pantalla automáticas en caso de fallos y logs de ejecución.

Formato de Entrega

- Nombre del repositorio: `pre-entrega-automation-testing-[nombre-apellido]`.
- Estructura mínima de carpetas:
  - `tests/`
  - `utils/` (funciones auxiliares)
  - `datos/` (si aplica datos externos como CSV/JSON)
  - `reports/` (reportes HTML y capturas)
- Compartir enlace al repositorio en el aula virtual antes de la fecha límite.


Buenos Aires
aprende
Agencia de Habilidades para el Futuro
Buenos
Aires
Ciudad