Practical – Introduction to Selenium – TUTOR

Learning Outcomes

1. Understand the concept and benefits of Selenium.
2. Install Selenium.
3. Use basic Selenium functions

The source code used in this practical can be downloaded or pulled from:
https://gitlab.com/budy.nyp/sweng

What is Selenium?

- Selenium is an open-source automation framework for web applications, enabling testers and developers to automate browser interactions and perform functional testing. With versatile tools like WebDriver, Selenium supports various programming languages and facilitates cross-browser testing, making it a go-to choice for efficient and scalable web automation.
- Selenium is not a single tool but a suite of software to automate web browsers.
- We will be focusing on Selenium WebDriver, the core component.

Why Use Selenium?

- Manual testing of web applications is slow, repetitive, and prone to human error. Imagine manually testing a login form with 100 different username/password combinations every time a developer makes a change.
- To solve it, developers are using automation. Write a script once and run it thousands of times, saving countless hours and ensuring consistency.

Key Benefits:

- Regression Testing: Quickly check that new code hasn't broken existing features.
- Cross-Browser Testing: Run the same test on Chrome, Firefox, Edge, etc.
- Efficiency: Tests can run unattended, even overnight.

Selenium WebDriver Architecture (The "How")

Your script does not talk to the browser directly. It talks to a **WebDriver**.

The WebDriver is a specific executable file (e.g., `chromedriver.exe`) that acts as a translator or a bridge.

1. Your Python Script sends a command (e.g., "click this button").
2. The Selenium library translates this into a standard command.
3. The **WebDriver** (`chromedriver.exe`) receives the command.
4. The WebDriver then gives the instructions to the **actual Browser (Chrome)**.
5. The Browser executes the instruction (the button is clicked).

Installation

On PyCharm

1. **Install Python (if not already)**

2. **Open your Project:** Launch PyCharm and open your Python project. If you don't have one, create a new Python project.

3. **Configure Python Interpreter (if needed):** If you're opening an existing project or haven't configured an interpreter, PyCharm might prompt you to do so. Ensure you have a Python interpreter (preferably latest version) configured for your project.

4. **Install Selenium via Python Package Tool Window:**
   - Go to the **Python Package** tool window. You can usually find this at the bottom of the PyCharm window, or by navigating to `View > Tool Windows > Python Packages`.
   - In the `Search` field, type `selenium`.
   - Locate `selenium` in the list of available packages and click the `Install` button next to it. PyCharm may show the list of `selenium` versions, choose latest version.
5. Verify the installation:In the Project pane on the left, right-click your project folder and choose **New** > **Python File**.
   - Create a new file in your project named `verify_install.py`.
   - Add the same verification code:

```
try:
    from selenium import webdriver
    print("SUCCESS: Selenium is installed correctly.")
except ImportError:
    print("ERROR: Selenium is not installed or not found in this environment.")
```

   - Run `verify_install.py`.
     The output should appear in your terminal, confirming the successful installation.

On VSCode

VS Code is a general-purpose editor, so the process relies more on using the integrated terminal.

1. **Install Python (if not already)**

2. **Open your Project Folder:** Launch VS Code. Create an empty folder for your project and open VS Code in that folder. This folder becomes your "workspace."

3. **Create a Virtual Environment:** It's a best practice to use a virtual environment for your project to manage dependencies.
    a. Open the **Command Palette** (`Ctrl + Shift + P`).
    b. Type `Python: Create Environment` and select the command.
    c. Choose `venv` as the environment type.
    d. If prompted, select `requirements.txt` for installation if you have one, or proceed without it.

4. **Install Selenium:** On terminal, execute this command: `pip install selenium`

5. **Verify the Installation**
    a. Create a new file in your project named `verify_install.py`.
    b. Add the same verification code:

```
try:
    from selenium import webdriver
    print("SUCCESS: Selenium is installed correctly.")
except ImportError:
    print("ERROR: Selenium is not installed or not found in this environment.")
```

    c. Run `verify_install.py`.
    The output should appear in your terminal, confirming the successful installation.

Hands-On Lab: The "Hello, Browser!" Script

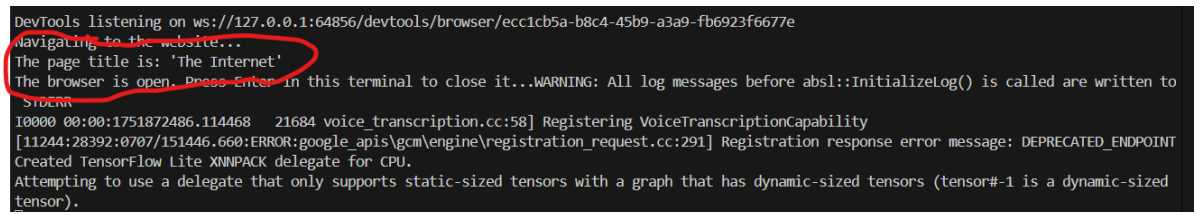
Let's write a script to prove our setup works.

1. Create a new file named `lab1_open_browser.py`.
2. Write the following code:

```
# lab1_open_browser.py

from selenium import webdriver

# This line creates an instance of the Chrome browser.
# Selenium Manager will automatically find or download the correct chromedriver.exe.
driver = webdriver.Chrome()

# 1. Open the target website
print("Navigating to the website...")
driver.get("https://the-internet.herokuapp.com")

# 2. Get and print the title of the page
page_title = driver.title
print(f"The page title is: '{page_title}'")

# 3. A short pause so we can see the browser window before it closes
input("The browser is open. Press Enter in this terminal to close it...")

# 4. Close the browser window and end the WebDriver session completely
driver.quit()

print("Script finished.")
```

3. In your terminal, execute `python lab1_open_browser.py`. A Chrome window should open to the target site.

   The terminal should show something similar to the following. Note that the page title displayed on the terminal.



Concepts: Finding Elements on a Page

You can use Selenium to locate specific element on the HTML page. In Chrome, you can do it using Developer Tools:

1. Open `https://the-internet.herokuapp.com/login` in Chrome.
2. Right-click on the "Username" input box and select **"Inspect"**. This opens the Developer Tools.
3. You can inspect different elements on the HTML page, such as `<input type="text" name="username" id="username">`.

In Selenium, you use locators to locate specific elements.

The most common locators:

- `By.ID`: The best and most reliable. IDs should be unique on a page.
  Example:

```
# Example 1: Find search input
search_input = driver.find_element(By.ID, "searchInput")
# Example 2: Find main navigation
main_nav = driver.find_element(By.ID, "mainNav")
```

- `By.NAME`: Good for form elements that have a `name` attribute.
  Example:

```
# Example 1: Find search query input
search_query = driver.find_element(By.NAME, "search_query")
# Example 2: Find category dropdown
category_select = driver.find_element(By.NAME, "category")
```

- `By.CLASS_NAME`: This is useful for selecting multiple elements that share a class.
  Example:

```
# Example 1: Find all book items
book_items = driver.find_elements(By.CLASS_NAME, "book-item")
# Example 2: Find all quantity inputs
quantity_inputs = driver.find_elements(By.CLASS_NAME, "qty-input")
```

- `By.TAG_NAME`: This can be used to find all elements of a specific type.
  Example:

```
# Example 1: Find all links
all_links = driver.find_elements(By.TAG_NAME, "a")
# Example 2: Find all paragraphs
all_paragraphs = driver.find_elements(By.TAG_NAME, "p")
```

- `By.LINK_TEXT`: This is helpful for finding links.
  Example:

```
# Example 1: Find home link
home_link = driver.find_element(By.LINK_TEXT, "Home Page")
# Example 2: Find terms link
terms_link = driver.find_element(By.LINK_TEXT, "Terms of Service")
```

- `By.PARTIAL_LINK_TEXT`: The Partial Link Text locator allows for partial matching of the anchor text, making it useful for long or dynamic link text.
  Example:

```
# Example 1: Find digital magazines link
magazines_link = driver.find_element(By.PARTIAL_LINK_TEXT,
"Digital")
# Example 2: Find help center link
help_link = driver.find_element(By.PARTIAL_LINK_TEXT, "Help")
```

- `By.CSS_SELECTOR`: A very powerful and flexible way to find elements, using CSS selector syntax.
  Example:

```
# Example 1: Find all reserve buttons
reserve_buttons = driver.find_elements(By.CSS_SELECTOR, ".reserve-
btn")
# Example 2: Find first book's author
first_author = driver.find_element(By.CSS_SELECTOR, "#book1
.author")
```

- `By.XPATH`: The most powerful locator, can navigate the entire page structure. Use it when nothing else works.
  Example:

```
# Example 1: Find all buttons in book items
book_buttons = driver.find_elements(By.XPATH, "//div[@class='book-
item']//button")
# Example 2: Find navigation links
nav_links = driver.find_elements(By.XPATH, "//nav[@id='mainNav']/a")
```

Now, let's combine everything to automate a login process. We will introduce two key actions:

- `send_keys()`: To type into an element.
- `click()`: To click on an element.

We will also introduce the most important concept for stable tests: **Waits**.

The Problem with Timing

Web pages are not instant. Elements might take a few milliseconds or even seconds to appear. If your script tries to find an element before it exists, it will crash.

- **Bad solution:** `time.sleep(5)` - This is a "blind wait". It's inefficient if the element appears in 1 second, and it will fail if the element takes 6 seconds.
- **Good solution: Explicit Waits.** Tell Selenium: "Wait *up to* 10 seconds for this specific element to become available. As soon as you find it, continue."

Hands-On Lab: Automating a Login Form

1. Create a new file named `lab2_login_test.py`.
2. Write the following code.

```
# lab2_login_test.py

import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# --- Setup ---
driver = webdriver.Chrome()
driver.get("https://the-internet.herokuapp.com/login")
print(f"Opened page: '{driver.title}'")

# --- Login Automation ---
try:
    # Step 1: Find username field, wait for it to exist, then type.
    # WebDriverWait(driver, 10) will wait for a maximum of 10 seconds.
    # .until(...) will wait for the specific expected_condition to be
true.
    # EC.presence_of_element_located is the condition to check.
    username_field = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "username"))
    )
    username_field.send_keys("tomsmith")
    print("Entered username.")

    # Step 2: Find password field and type. No wait needed as the page is
already loaded.
    password_field = driver.find_element(By.ID, "password")
    password_field.send_keys("SuperSecretPassword!")
    print("Entered password.")

    # Step 3: Find login button by its CSS selector and click it.
```

```
    login_button = driver.find_element(By.CSS_SELECTOR,
"button[type='submit']")
    login_button.click()
    print("Clicked login button.")

    # --- Verification ---
    # Step 4: On the new page, wait for the success message to be VISIBLE.
    success_message_banner = WebDriverWait(driver, 10).until(
        EC.visibility_of_element_located((By.ID, "flash"))
    )

    # Step 5: Assert that the test was successful.
    # An 'assert' checks if a condition is true. If not, it stops and
raises an error.
    # This is how we know if our test actually passed or failed.
    assert "You logged into a secure area!" in success_message_banner.text
    print("TEST PASSED: Login successful and verified.")

except Exception as e:
    print(f"TEST FAILED: An exception occurred: {e}")

finally:
    # --- Teardown ---
    time.sleep(3) # A brief pause to see the final result on the screen
    driver.quit()
    print("Browser closed.")
```

3. Run `lab2_login_test.py`.

   Observe the browser as it fills the form, logs in, and verifies the success message.

```
DevTools listening on ws://127.0.0.1:57182/devtools/browser/2546c90d-02b4-41bc-aeea-b83b5e3ee851
Opened page: 'The Internet'
Entered username.
Entered password.
Clicked login button.
TEST PASSED: Login successful and verified.
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1751872680.560133    8040 voice_transcription.cc:58] Registering VoiceTranscriptionCapability
[12024:16400:0707/151800.890:ERROR:google_apis\gcm\engine\registration_request.cc:291] Registration response error message: DEPRECATED_ENDPOINT
Browser closed.
```

Hands-On Lab: Interacting with Checkboxes

Let's learn to interact with checkboxes and verify their state (checked or unchecked).

1. Create a new file named `lab3_checkboxes.py`.
2. Write the following code:

```
# lab3_checkboxes.py

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

driver = webdriver.Chrome()
driver.get("https://the-internet.herokuapp.com/checkboxes")
print(f"Opened page: '{driver.title}'")

try:
    # Find all the checkboxes on the page. XPath is good for this.
    # This XPath means: find all 'input' tags that have an attribute 'type' equal to 'checkbox'.
    checkboxes = driver.find_elements(By.XPATH, "//input[@type='checkbox']")

    # Isolate the first and second checkbox from the list
    checkbox1 = checkboxes[0]
    checkbox2 = checkboxes[1]

    # --- Interact with the first checkbox ---
    print(f"Initial state of checkbox 1: is_selected() = {checkbox1.is_selected()}") # Should be False
    print("Clicking checkbox 1...")
    checkbox1.click()
    print(f"New state of checkbox 1: is_selected() = {checkbox1.is_selected()}") # Should be True
    assert checkbox1.is_selected() == True
    print("VERIFIED: Checkbox 1 is now checked.")

    print("-" * 20) # separator

    # --- Interact with the second checkbox ---
```

```
    print(f"Initial state of checkbox 2: is_selected() =
{checkbox2.is_selected()}") # Should be True
    print("Clicking checkbox 2...")
    checkbox2.click()
    print(f"New state of checkbox 2: is_selected() =
{checkbox2.is_selected()}") # Should be False
    assert checkbox2.is_selected() == False
    print("VERIFIED: Checkbox 2 is now unchecked.")

    print("\nTEST PASSED: All checkbox interactions were successful.")

except Exception as e:
    print(f"TEST FAILED: An exception occurred: {e}")

finally:
    time.sleep(3)
    driver.quit()
    print("Browser closed.")
```

3. Run `lab3_checkboxes.py`.

   This script demonstrates finding multiple elements, using `.is_selected()` to verify their state and updating the state.

```
DevTools listening on ws://127.0.0.1:57221/devtools/browser/d606ec91-d723-467f-b17f-644a5c84d218
Opened page: 'The Internet'
Initial state of checkbox 1: is_selected() = False
Clicking checkbox 1...
New state of checkbox 1: is_selected() = True
VERIFIED: Checkbox 1 is now checked.
--------------------
Initial state of checkbox 2: is_selected() = True
Clicking checkbox 2...
New state of checkbox 2: is_selected() = False
VERIFIED: Checkbox 2 is now unchecked.

TEST PASSED: All checkbox interactions were successful.
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1751872752.327724   24696 voice_transcription.cc:58] Registering VoiceTranscriptionCapability
[25892:688:0707/151912.559:ERROR:google_apis\gcm\engine\registration_request.cc:291] Registration response error message: DEPRECATED_ENDPOINT
Browser closed.
```

Exercise: Dynamic Element Handling

This exercise will bring together everything you've learned: navigation, finding elements, clicking, waiting for dynamic content, and using assertions to verify the application's state.

Scenario: You will automate the "Add/Remove Elements" page (https://the-internet.herokuapp.com/add_remove_elements/) . This page has a button that adds a "Delete" button to the page. Each "Delete" button, when clicked, removes itself.

Your Goal: Write a single Python script that performs the following actions and verifications:

1. **Navigate:** Open the "Add/Remove Elements" page.
2. **Verify Initial State:** Assert that there are no "Delete" buttons on the page when it first loads.
3. **Add an Element:** Click the "Add Element" button.
4. **Wait and Verify Addition:** Wait for the new "Delete" button to appear and then assert that exactly one "Delete" button is now visible on the page.
5. **Remove the Element:** Click the "Delete" button.
6. **Wait and Verify Removal:** Wait for the "Delete" button to disappear and assert that it has been successfully removed (i.e., the count of "Delete" buttons is back to zero).

Tips:

- The "Delete" buttons all share the same class name: added-manually.
- Use driver.find_elements() (the plural version) to get a list of all matching elements. You can then check the length of this list (len()) to see how many there are.
- Use WebDriverWait to handle the delays as elements are added and removed.

Solutions

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# A constant for the target class name to avoid typos
DELETE_BUTTON_CLASS = "added-manually"

# --- Setup ---
driver = webdriver.Chrome()
print("Starting the Add/Remove Elements test.")

try:
    # --- Step 1: Navigate ---
    url = "https://the-internet.herokuapp.com/add_remove_elements/"
    driver.get(url)
    print(f"Navigated to {url}")

    # --- Step 2: Verify Initial State ---
    # Use find_elements (plural) which returns a list. If no elements are found,
the list is empty.
    initial_delete_buttons = driver.find_elements(By.CLASS_NAME,
DELETE_BUTTON_CLASS)
    assert len(initial_delete_buttons) == 0
    print("Initial state verified: No 'Delete' buttons are present.")

    # --- Step 3: Add an Element ---
    add_button = driver.find_element(By.CSS_SELECTOR,
"button[onclick='addElement()']")
    add_button.click()
    print("Clicked the 'Add Element' button.")

    # --- Step 4: Wait and Verify Addition ---
    # We will wait for the list of 'Delete' buttons to have exactly one element.
    # We can use a lambda function to create a custom wait condition.
```

```
    WebDriverWait(driver, 10).until(
        lambda d: len(d.find_elements(By.CLASS_NAME, DELETE_BUTTON_CLASS)) == 1
    )

    # Now that we've waited, we can find the buttons again and assert the count.
    added_delete_buttons = driver.find_elements(By.CLASS_NAME,
DELETE_BUTTON_CLASS)
    assert len(added_delete_buttons) == 1
    print("Verification successful: One 'Delete' button has been added.")

    # --- Step 5: Remove the Element ---
    # Since we know there is one button, we can get it from the list (the first
element, index 0)
    added_delete_buttons[0].click()
    print("Clicked the 'Delete' button.")

    # --- Step 6: Wait and Verify Removal ---
    # We use the same lambda trick to wait until the list of 'Delete' buttons is
empty again.
    WebDriverWait(driver, 10).until(
        lambda d: len(d.find_elements(By.CLASS_NAME, DELETE_BUTTON_CLASS)) == 0
    )

    final_delete_buttons = driver.find_elements(By.CLASS_NAME,
DELETE_BUTTON_CLASS)
    assert len(final_delete_buttons) == 0
    print("Verification successful: The 'Delete' button has been removed.")
    print("\n--------------------")
    print("  TEST PASSED!  ")
    print("--------------------")

except AssertionError:
    print("\nTEST FAILED: An assertion failed.")
except Exception as e:
    print(f"\nTEST FAILED: An unexpected error occurred: {e}")
finally:
    # --- Teardown ---
```

```
    time.sleep(3) # A brief pause to see the final browser state
    driver.quit()
    print("Browser closed. Test finished.")
```

Where to Go From Here?

This is the introduction. Professional test automation builds on this with:

- **Test Frameworks (like pytest):** To structure tests cleanly, run them, and generate reports.
- **Page Object Model (POM):** A famous design pattern to organize your locators and make your test suite maintainable.
- **Handling More Complex UI:** Interacting with dropdown menus, handling pop-up alerts, switching between browser tabs, and drag-and-drop actions.
- **Selenium Grid:** A tool for running hundreds of tests in parallel across different browsers and operating systems to get fast feedback.

References

- Wikipedia article on Selenium, https://en.wikipedia.org/wiki/Selenium_(software)