Module 19 - AUTOMATION WITH SELENIUM

Welcome to Module 19! Selenium WebDriver is a widely used open-source framework for automating web browsers. It allows you to write scripts that interact with web pages just like a human user would: clicking buttons, filling forms, navigating pages, and extracting information.

Chapter 1: Setting Up and Basic Interactions

This chapter covers the installation of Selenium, how to set up your WebDriver, and perform fundamental tasks like opening URLs and automating simple searches.

1.1 Installation

To use Selenium with Python, you need two main components:

1. **Selenium Python Library:** Install it using pip:

   Bash

   ```
   pip install selenium
   ```

2. **WebDriver Executable:** Selenium automates browsers using browser-specific "WebDrivers." You need to download the WebDriver executable for the browser you want to automate. Common WebDrivers include:
   - **ChromeDriver:** For Google Chrome (most common).
     - Download from: https://chromedriver.chromium.org/downloads
     - **Important:** Download the version that matches your installed Chrome browser version.
   - **GeckoDriver:** For Mozilla Firefox.
     - Download from: https://github.com/mozilla/geckodriver/releases
   - **MSEdgeDriver:** For Microsoft Edge.
     - Download from: https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

   Setup (Crucial!):

   - **Recommended (Simple):** Place the downloaded WebDriver executable (e.g., `chromedriver.exe` for Windows, `chromedriver` for Linux/macOS) in a directory that is included in your system's `PATH` environment variable. This allows Python to find it automatically.
   - **Alternative (Explicit Path):** If you don't want to modify your PATH, you can specify the exact path to the WebDriver executable in your Python code. We'll use this method in our examples for clarity.

1.2 Opening a URL

After setting up, the first step is to launch a browser and navigate to a URL.

- **Steps:**
  1. Import `webdriver` from `selenium`.
  2. Create an instance of the WebDriver (e.g., `webdriver.Chrome()`).
  3. Use the `driver.get()` method to open a URL.
  4. Use `time.sleep()` to pause execution (useful for observing actions, but in real scenarios, use explicit waits).
  5. Always close the browser at the end using `driver.quit()`.
- **Example:**

  Python

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
import time

# --- IMPORTANT ---
# Replace 'path/to/your/chromedriver.exe' with the actual path to
your ChromeDriver executable
# Example: C:/Users/YourUser/Downloads/chromedriver-
win64/chromedriver.exe
# Or: /usr/local/bin/chromedriver on macOS/Linux if you installed it
there
CHROMEDRIVER_PATH = 'path/to/your/chromedriver.exe'

try:
    # Set up the WebDriver service
    service = Service(CHROMEDRIVER_PATH)

    # Initialize the Chrome WebDriver
    driver = webdriver.Chrome(service=service)
    print("Chrome browser launched successfully.")

    # Open Google's website
    driver.get("https://www.google.com")
    print("Navigated to Google.com")

    # Pause for a few seconds to see the page
    time.sleep(3)

    # Close the browser
    driver.quit()
    print("Browser closed.")

except Exception as e:
    print(f"An error occurred: {e}")
    print("Please ensure ChromeDriver path is correct and Chrome
browser is installed.")
    print("Also, check that ChromeDriver version matches your Chrome
browser version.")
```

1.3 Automating Google Search

Now, let's automate a common task: performing a search on Google. This involves finding an HTML element (the search bar), typing text into it, and then submitting the form.

- **Key Concepts:**
  - `find_element()`: Locates a single HTML element on the page.
  - `By` class: Used to specify the method for finding an element (e.g., `By.NAME`, `By.ID`, `By.XPATH`).
  - `send_keys()`: Simulates typing text into an input field.
  - `submit()`: Submits the form the element belongs to (if it's an input within a form).
  - `click()`: Simulates a mouse click on an element.
- **Example:**

  Python

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By # Import By for finding
elements
import time

CHROMEDRIVER_PATH = 'path/to/your/chromedriver.exe'

try:
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)
    driver.get("https://www.google.com")

    # Give some time for the page to load
    time.sleep(2)

    # 1. Find the search input field by its 'name' attribute
    # Inspect Google's search bar: it usually has name="q"
    search_box = driver.find_element(By.NAME, "q")
    print("Found search box.")

    # 2. Type "Selenium Python" into the search box
    search_box.send_keys("Selenium Python")
    print("Typed 'Selenium Python' into search box.")

    # 3. Submit the search query
    # Option A: Use .submit() on the input element
    search_box.submit()
    print("Search submitted.")

    # Option B (Alternative): Find the search button and click it
    # search_button = driver.find_element(By.NAME, "btnK") # 'btnK'
is a common name for Google search button
    # search_button.click()
    # print("Search button clicked.")

    # Pause to view search results
    time.sleep(5)

    driver.quit()
```

```
except Exception as e:
    print(f"An error occurred during Google search automation: {e}")
    print("Ensure the search box 'name' attribute ('q') or button
'name' attribute ('btnK') is correct.")
```

1.4 Automating Navigations

Selenium allows you to navigate browser history just like a user would.

- **Methods:**
  - `driver.back()`: Navigates back to the previous page in the browser history.
  - `driver.forward()`: Navigates forward to the next page in the browser history.
  - `driver.refresh()`: Refreshes the current page.
- **Example:**

  Python

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import time

CHROMEDRIVER_PATH = 'path/to/your/chromedriver.exe'

try:
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)

    # 1. Navigate to the first page
    driver.get("https://www.google.com")
    print("Page 1: Google.com")
    time.sleep(2)

    # 2. Navigate to a second page (e.g., performing a search)
    search_box = driver.find_element(By.NAME, "q")
    search_box.send_keys("Python official website")
    search_box.submit()
    print("Page 2: Google Search Results")
    time.sleep(3)

    # 3. Click on the first search result (e.g., Python.org)
    # We'll use a CSS selector here, often reliable for first link
    first_result_link = driver.find_element(By.CSS_SELECTOR,
"div#search a h3")
    first_result_link.click()
    print("Page 3: Navigated to Python.org (or similar first
result)")
    time.sleep(3)

    # 4. Go back to the previous page (Google Search Results)
    driver.back()
    print("Navigated back to Page 2 (Google Search Results)")
    time.sleep(3)

    # 5. Go forward to the next page (Python.org again)
    driver.forward()
    print("Navigated forward to Page 3 (Python.org)")
```

```
        time.sleep(3)

        # 6. Refresh the current page
        driver.refresh()
        print("Page 3 refreshed.")
        time.sleep(3)

        driver.quit()

    except Exception as e:
        print(f"An error occurred during navigation automation: {e}")
```

Chapter 2: Locating Elements and Extracting Data

This chapter focuses on the various sophisticated methods Selenium provides for finding specific elements on a web page and extracting their data.

2.5 Finding Elements by Different Methods

Selenium offers several strategies to locate elements. Choosing the right method depends on the HTML structure and the uniqueness of the element.

- **`By.ID`:** Locates an element by its `id` attribute. IDs are supposed to be unique on a page.
  - Example: `driver.find_element(By.ID, "loginButton")`
- **`By.NAME`:** Locates an element by its `name` attribute.
  - Example: `driver.find_element(By.NAME, "username")`
- **`By.CLASS_NAME`:** Locates elements by their `class` attribute. Note: if an element has multiple classes (e.g., `class="btn primary large"`), you can only use one class name at a time with this method. For multiple classes, use CSS Selector or XPath.
  - Example: `driver.find_element(By.CLASS_NAME, "product-title")`
- **`By.TAG_NAME`:** Locates elements by their HTML tag name (e.g., `div`, `a`, `input`, `p`).
  - Example: `driver.find_element(By.TAG_NAME, "h1")`
- **`By.LINK_TEXT`:** Locates an `<a>` (anchor) element by its exact visible text.
  - Example: `driver.find_element(By.LINK_TEXT, "Click here for more details")`
- **`By.PARTIAL_LINK_TEXT`:** Locates an `<a>` element by partial matching of its visible text.
  - Example: `driver.find_element(By.PARTIAL_LINK_TEXT, "more details")`
- **`By.CSS_SELECTOR`:** Locates elements using CSS selectors (similar to how CSS styles elements). Very powerful and often preferred for complex selections over XPath due to readability and performance.
  - Examples:
    - `#id_name` (by ID)
    - `.class_name` (by class)
    - `tag_name` (by tag)
    - `tag_name[attribute='value']` (by tag and attribute)
    - `parent > child` (direct child)
    - `ancestor descendant` (any descendant)

    - Example: `driver.find_element(By.CSS_SELECTOR, "div.product-card > h2.title")`
- **`By.XPATH`:** Locates elements using XPath expressions. Extremely powerful and flexible for navigating the HTML DOM structure.
    - `find_element()` returns the first matching element.
    - `find_elements()` returns a list of all matching elements.
- **Example (Demonstrating various `By` methods):** *(We'll use a dummy website for demonstration, or you can inspect elements on any public site like `books.toscrape.com`)*

Python

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import time

CHROMEDRIVER_PATH = 'path/to/your/chromedriver.exe'

try:
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)
    driver.get("http://books.toscrape.com/") # A good site for
practicing
    time.sleep(2)

    print("\n--- Finding Elements ---")

    # By ID (if exists, books.toscrape.com doesn't have many
prominent IDs)
    # Try to find a hypothetical element:
    try:
        # This ID likely won't exist on books.toscrape.com, just for
demo
        non_existent_element = driver.find_element(By.ID,
"nonExistentId")
        print(f"Found by ID (this should fail):
{non_existent_element.tag_name}")
    except Exception:
        print("Element with ID 'nonExistentId' not found (as
expected).")

    # By Class Name: Find the first book's price
    try:
        price_element = driver.find_element(By.CLASS_NAME,
"price_color")
        print(f"Found by CLASS_NAME (first price):
{price_element.text}")
    except Exception as e:
        print(f"Error finding by CLASS_NAME: {e}")

    # By Tag Name: Find all <h3> tags (often book titles)
    h3_tags = driver.find_elements(By.TAG_NAME, "h3")
    print(f"Found {len(h3_tags)} elements by TAG_NAME (h3). First
one: {h3_tags[0].text if h3_tags else 'N/A'}")

    # By Link Text: Find a specific link, e.g., "Poetry" category
    try:
        poetry_link = driver.find_element(By.LINK_TEXT, "Poetry")
        print(f"Found by LINK_TEXT: {poetry_link.text}")
        # poetry_link.click() # Uncomment to click
        # time.sleep(2)
        # driver.back()
    except Exception as e:
        print(f"Error finding by LINK_TEXT: {e}")

    # By Partial Link Text: Find a link containing "Fantasy"
    try:
        fantasy_link_partial =
driver.find_element(By.PARTIAL_LINK_TEXT, "Fantasy")
        print(f"Found by PARTIAL_LINK_TEXT:
{fantasy_link_partial.text}")
    except Exception as e:
        print(f"Error finding by PARTIAL_LINK_TEXT: {e}")

    # By CSS Selector: Find all book titles
    # CSS selector: article with class 'product_pod', then h3, then a
    book_titles_css = driver.find_elements(By.CSS_SELECTOR,
"article.product_pod h3 a")
    print(f"Found {len(book_titles_css)} elements by CSS_SELECTOR
(book titles). First one: {book_titles_css[0].get_attribute('title')
if book_titles_css else 'N/A'}")

    time.sleep(3)
    driver.quit()

except Exception as e:
    print(f"An error occurred: {e}")
```

2.6 Selecting Links

This is a specific application of `find_element()` using `By.LINK_TEXT` or `By.PARTIAL_LINK_TEXT`, followed by `click()`.

- **Example:**

  Python

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import time

CHROMEDRIVER_PATH = 'path/to/your/chromedriver.exe'

try:
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)
    driver.get("http://books.toscrape.com/")
    time.sleep(2)

    # Find the 'Travel' category link and click it
    try:
        travel_link = driver.find_element(By.LINK_TEXT, "Travel")
        print(f"Found link: '{travel_link.text}'. Clicking now...")
        travel_link.click()
        print("Navigated to Travel category.")
        time.sleep(3)
    except Exception as e:
        print(f"Error finding/clicking 'Travel' link: {e}")
        print("The link text 'Travel' might not exist or the page
structure changed.")

    driver.quit()

except Exception as e:
    print(f"An error occurred: {e}")
```

2.7 Refreshing the Page

Already demonstrated in section 1.4, `driver.refresh()` simply reloads the current page in the browser.

- **Example (Simple Refresh):**

  Python

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
import time

CHROMEDRIVER_PATH = 'path/to/your/chromedriver.exe'

try:
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)
    driver.get("https://www.google.com")
    print("Page loaded.")
    time.sleep(3)

    driver.refresh()
    print("Page refreshed.")
    time.sleep(3)

    driver.quit()

except Exception as e:
    print(f"An error occurred: {e}")
```

2.8 Finding Element by XPath

XPath (XML Path Language) is a powerful query language for selecting nodes from an XML document (and HTML documents can be treated as XML). It allows you to navigate through elements and attributes in the DOM structure, making it very flexible for finding elements that might not have unique IDs or classes.

- **XPath Syntax Basics:**
  - `/`: Selects from the root node.
  - `//`: Selects nodes from the current node that match the selection, no matter where they are.
  - `.`: Selects the current node.
  - `..`: Selects the parent of the current node.
  - `@attribute`: Selects an attribute.
  - `tagname`: Selects all nodes with the specified tag name.

    - `tagname[condition]`: Selects tags that meet a certain condition.
        - `[@id='someid']`: Element with a specific ID.
        - `[@class='someclass']`: Element with a specific class.
        - `[contains(@class, 'partial_class')]`: Element whose class attribute contains a substring.
        - `[text()='Some Text']`: Element whose visible text is exactly 'Some Text'.
        - `[contains(text(), 'Partial Text')]`: Element whose visible text contains 'Partial Text'.
        - `[index]`: Selects the element at a specific index (1-based).
- **Example:**

  Python

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import time

CHROMEDRIVER_PATH = 'path/to/your/chromedriver.exe'

try:
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)
    driver.get("http://books.toscrape.com/")
    time.sleep(2)

    print("\n--- Finding Elements by XPath ---")

    # XPath 1: Find the first <h1> tag on the page (absolute path)
    try:
        h1_xpath = driver.find_element(By.XPATH,
"/html/body/div/div/div/div/section/div[2]/ol/li/article/div[2]/h3/a"
)
        # Note: This is an example of an absolute XPath, which is
brittle.
        # A more robust XPath for the first book title might be:
        first_book_title_xpath = driver.find_element(By.XPATH,
"//article[@class='product_pod'][1]/h3/a")
        print(f"Found first book title by XPath:
{first_book_title_xpath.get_attribute('title')}")
    except Exception as e:
        print(f"Error finding first book title by XPath: {e}")


    # XPath 2: Find all book prices (using contains for class, more
robust than exact match)
    all_prices_xpath = driver.find_elements(By.XPATH,
"//p[contains(@class, 'price_color')]")
    print(f"Found {len(all_prices_xpath)} prices by XPath. First one:
{all_prices_xpath[0].text if all_prices_xpath else 'N/A'}")

    # XPath 3: Find a specific book by its title text
    try:
        book_by_text_xpath = driver.find_element(By.XPATH,
"//a[contains(text(), 'A Light in the Attic')]")
        print(f"Found book by partial link text XPath:
{book_by_text_xpath.text}")
```

```
    except Exception as e:
        print(f"Error finding book by text XPath: {e}")

    time.sleep(3)
    driver.quit()

except Exception as e:
    print(f"An error occurred: {e}")
```

- **Tip:** You can often get XPath from your browser's developer tools (Inspect Element -> Right Click -> Copy -> Copy XPath or Copy Full XPath). Be cautious with "Full XPath" as it's often too specific and brittle.

2.9 Getting Data

Once you have located an element, you can extract various types of information from it.

- **`element.text`:** Returns the visible (inner) text of the element, including the text of its sub-elements, excluding any leading/trailing whitespace.
- **`element.get_attribute('attribute_name')`:** Returns the value of a specified HTML attribute (e.g., `href`, `src`, `value`, `title`, `class`, `id`).
- **`element.tag_name`:** Returns the HTML tag name of the element (e.g., `'div'`, `'a'`, `'input'`).
- **`element.is_displayed()`:** Returns `True` if the element is visible on the page, `False` otherwise.
- **`element.is_enabled()`:** Returns `True` if the element is enabled (not disabled), `False` otherwise.
- **`element.is_selected()`:** Returns `True` if the element (e.g., checkbox, radio button, option in a dropdown) is selected, `False` otherwise.
- **Example:**

  Python

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import time

CHROMEDRIVER_PATH = 'path/to/your/chromedriver.exe'

try:
    service = Service(CHROMEDRIVER_PATH)
    driver = webdriver.Chrome(service=service)
    driver.get("http://books.toscrape.com/catalogue/a-light-in-the-
attic_1000/index.html") # Navigate to a specific book page
    time.sleep(2)

    print("\n--- Getting Data from Elements ---")

    # Get the product title
    try:
        product_title_element = driver.find_element(By.TAG_NAME,
"h1")
        product_title = product_title_element.text
        print(f"Product Title: {product_title}")
        print(f"  Tag Name: {product_title_element.tag_name}")
        print(f"  Is Displayed?:
{product_title_element.is_displayed()}")
    except Exception as e:
        print(f"Error getting product title: {e}")

    # Get the price
    try:
        price_element = driver.find_element(By.CLASS_NAME,
"price_color")
        product_price = price_element.text
        print(f"Product Price: {product_price}")
    except Exception as e:
        print(f"Error getting product price: {e}")

    # Get the description (often in a <p> tag after an <h3> or <h4>)
    try:
        # This XPath targets the <p> tag that is a sibling to an <h3>
with text 'Product Description'
        description_element = driver.find_element(By.XPATH,
"//h2[text()='Product Description']/following-sibling::p")
        product_description = description_element.text[:200] + "..."
# Get first 200 chars
        print(f"Product Description (first 200 chars):
{product_description}")
    except Exception as e:
        print(f"Error getting product description: {e}")

    # Get the 'href' attribute of the "Add to basket" button (or any
link)
    try:
        # Example using a button if it had an href, or a link like
'All products'
        # Let's target the 'Add to basket' button which doesn't have
an href for demo
        add_to_basket_button = driver.find_element(By.CLASS_NAME,
"btn-primary")
        # For demonstration, we'll try to get its 'type' attribute
        button_type = add_to_basket_button.get_attribute("type")
        print(f"Add to Basket Button Type: {button_type}")
        print(f"Add to Basket Button Text:
{add_to_basket_button.text}")
    except Exception as e:
        print(f"Error getting button attributes: {e}")

    time.sleep(3)
    driver.quit()

except Exception as e:
    print(f"An error occurred: {e}")
```