Assignment-2

Locators In Selenium

URL: http://www.automationpractice.pl/index.php

Question:1

i)Locate a search bar on the page using its ID. Enter the term “search box” and submit the search.

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

driver = webdriver.Chrome()
driver.get("http://www.automationpractice.pl/index.php")
time.sleep(2)

driver.find_element(By.ID,"search_query_top").send_keys("search box")
time.sleep(2)
driver.find_element(By.XPATH,"//*[@id='searchbox']/button").click()
time.sleep(3)

driver.close()
```

ii) What are the advantages of using an ID locator? When might it not be the best choice?

Advantages of Using an ID Locator:

- **Uniqueness**: IDs are unique to elements, making them easy to locate.
- **Speed**: ID-based lookups are fast because they are optimized by browsers.
- **Stability**: IDs are less likely to change compared to other attributes.
- **Simplicity**: It is more straightforward and human-readable.

When It Might Not Be the Best Choice:

- **Not Always Available**: Not all elements have an ID.
- **ID Conflicts**: Multiple elements may share the same ID by mistake.
- **Dynamic IDs**: If IDs change dynamically (e.g., session-based), they become unreliable.

Question:2

i) Locate a Search button or input field using the name attribute. Perform an action (e.g., clicking or entering text).

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

driver = webdriver.Chrome()
driver.get("http://www.automationpractice.pl/index.php")
time.sleep(2)

driver.find_element(By.NAME,"search_query").send_keys("search box")
time.sleep(2)
driver.find_element(By.XPATH,"//*[@id='searchbox']/button").click()
time.sleep(3)

driver.close()
```

ii) How does Selenium handle multiple elements with the same name attribute?

Selenium handles multiple elements with the same name attribute as follows:

- If there are **multiple elements** with the same name, find_element(By.NAME, ...) will return **the first matching element** it finds in the document.
- To interact with **all elements** with the same name, you can use find_elements(By.NAME, ...), which returns a **list of matching elements**. You can then loop through them or perform actions on specific elements.

Question:3

i) Locate an element (like a "Sign In" button) using its class name and perform a click action.

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

driver = webdriver.Chrome()
driver.get("http://www.automationpractice.pl/index.php")
time.sleep(2)
```

```
driver.find_element(By.CLASS_NAME,"login").click()
time.sleep(2)
driver.close()
```

ii) What limitations might you encounter when using class name as a locator?

While using class name as a locator can be simple and effective, there are several limitations:

1. **Non-unique Class Names**: Many elements on a webpage may share the same class name, leading to ambiguity. If multiple elements have the same class, find_element will return only the first one, and find_elements will return a list. This can make it hard to select a specific element.
2. **Dynamic Classes**: Class names are often used dynamically or generated by JavaScript (e.g., by adding or removing classes based on user actions). This could result in the class name changing, making the locator unreliable.
3. **Long or Complex Class Names**: Some websites use long, complex, or compound class names (e.g., a combination of multiple classes), which may be difficult to manage in automation scripts.
4. **Dependency on CSS Changes**: If the website's CSS changes and the class name is modified, your automation script will break, requiring an update to the class name in the script.

Question:4

Find an element by its tag name (e.g., finding the first <h1> on the page) and print its text content.
In what scenarios is tag name most useful as a locator?

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

driver = webdriver.Chrome()
driver.get("http://www.automationpractice.pl/index.php")
h1_tag = driver.find_element(By.TAG_NAME,"h1").text
print(h1_tag)
driver.close()
```

Tag name is most useful as a locator in the following scenarios:

1. **When you need to interact with all elements of a specific type** (e.g., all <p>, <h1>, or <button> tags).
2. **When the element is unique or semantically important** (e.g., a single <h1> tag or <form> tag).
3. **On simple, static pages** where the structure is straightforward and elements are easily identifiable by their tags.
4. **When other locators (ID, class, XPath) are not available or too complex**.
5. **For basic HTML elements** like headings, lists, and links.

Question:5

Locate a specific link using its visible text (e.g., a link with text "Contact Us") and click on it.
When is it advantageous to use link text over other locators?

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("http://www.automationpractice.pl/index.php")
time.sleep(2)

driver.find_element(By.LINK_TEXT, "Contact us").click()
time.sleep(3)

driver.quit()
```

Using link text is advantageous in the following scenarios:

When the link text is unique: If the link's visible text is unique on the page, it is a straightforward and easy-to-read way to locate a link.

Simplicity and Readability: Link text is human-readable, which makes the locator easy to understand and maintain, especially in cases where the link text is clear and descriptive.

When XPath or CSS Selectors are Too Complex: If the structure of the page is complex and writing a precise XPath or CSS selector becomes cumbersome, using the visible text of the link can be simpler and more efficient.

Stable Text Content: When the visible text of the link is less likely to change (i.e., it is not dynamically generated or changed based on user actions), link text can be more reliable.

Question:6

Locate a link using part of its visible text ("Contact us ") and click on it.
How does partial link text differ in behavior from link text?

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

driver = webdriver.Chrome()

driver.get("http://www.automationpractice.pl/index.php")
time.sleep(2)

driver.find_element(By.PARTIAL_LINK_TEXT, "Contact").click()
time.sleep(3)

driver.quit()

"""
Partial Link Text allows you to locate a link by matching only a part of its visible text, making it
more flexible. It works if you know just a portion of the link text.

Link Text, on the other hand, requires an exact match of the full visible text of the link.

In summary, partial link text is more flexible but can lead to ambiguity if multiple links contain the
same partial text, whereas link text is more precise but requires the full text to match exactly.
"""
```

Question:7

Use a CSS Selector to locate a button by its class and ID and perform a click action.
Why might you choose CSS Selectors over XPath?

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

driver = webdriver.Chrome()
```

```
driver.get("http://www.automationpractice.pl/index.php")
time.sleep(2)

driver.find_element(By.CSS_SELECTOR, "button.btn").click()
time.sleep(3)

driver.quit()

"""
CSS Selectors are often preferred over XPath because:

Faster performance: CSS selectors are typically faster than XPath, especially in browsers like Chrome.
Simpler syntax: They are easier to write and understand, particularly for selecting elements by class, ID, or attributes.
Better browser compatibility: CSS selectors are well-supported and reliable across modern browsers.
No need for complex functions: Unlike XPath, CSS selectors don’t require functions like contains() or text(), making them less error-prone.
"""
```

Question:8

Use XPath to locate an element(Follow us on Facebook) with a specific attribute and perform a click action.
How can relative XPath be beneficial over absolute XPath?

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

driver = webdriver.Chrome()
driver.get("http://www.automationpractice.pl/index.php")
time.sleep(2)
driver.find_element(By.XPATH, "//a[@href='http://www.facebook.com/']").click()
time.sleep(3)

driver.quit()

"""
Less prone to changes: Absolute XPath requires specifying the full path from the root (e.g., /html/body/div[1]/div[2]/a), which can break if any part of the page structure changes. Relative XPath, however, starts the path from a specific element (e.g., //a[@href='http://www.facebook.com/']), making it more robust to changes in the HTML structure.
```

Shorter and more maintainable: Relative XPath is more concise and easier to write and maintain, as it doesn’t rely on the full document structure. It focuses on locating elements based on attributes or text content.

Easier to debug: Since relative XPath targets elements from a specific point, it's easier to troubleshoot if something goes wrong (e.g., if an element is moved or added).

Improved readability: Relative XPath is generally more readable, especially when targeting specific attributes (like id, class, or href), making the script easier to understand and modify.
"""

Question:9

Use a combination of locators (e.g., CSS selector for a parent element and then locate a child element using XPath) to find a specific element.
In what situations would combining locators be helpful?

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

driver = webdriver.Chrome()

driver.get("http://www.automationpractice.pl/index.php")
time.sleep(2)
parent_element = driver.find_element(By.CSS_SELECTOR, "div#homefeatured")
child_element = parent_element.find_element(By.XPATH, ".//a[@title='Add to cart']")
child_element.click()

time.sleep(3)

driver.quit()
```

Combining locators is helpful in the following situations:

1. **Narrowing search scope**: It helps to focus on a specific section of the page, reducing the chance of selecting the wrong element.
2. **Dynamic content**: In dynamic pages with changing structures, combining locators ensures you accurately target elements within specific containers.
3. **Improving accuracy**: It avoids ambiguity, especially when there are many similar elements on the page.