Automation Testing:

Using Python and Selenium

Automation Testing is a software testing technique that uses automated tools and scripts to perform tests on a software application. The main goal is to ensure that the software behaves as expected and to catch any defects or errors early in the development process. Automation testing is particularly useful for repetitive test cases, regression testing, and scenarios with large datasets.

Types of Automation Testing:

1. **Unit Testing:**
    - Focuses on testing individual units or components of the software in isolation.
    - Written and executed by developers to ensure that each unit of the software performs as designed.
2. **Integration Testing:**
    - Verifies the interactions between different modules or components of the software.
    - Ensures that integrated components work together as expected.
3. **Functional Testing:**
    - Evaluates the software's functionality by testing its features and capabilities.
    - Includes test cases to validate the system against specified requirements.
4. **Regression Testing:**
    - Ensures that new changes or additions to the software do not adversely affect existing features.
    - Automates previously executed test cases to catch regressions.
5. **Performance Testing:**
    - Measures the system's performance under various conditions such as load, stress, and scalability.
    - Ensures the application can handle a specified level of load without degradation.
6. **Load Testing:**
    - Examines how well the system performs under specific load conditions.
    - Identifies performance bottlenecks and potential issues related to system resources.
7. **End-to-End Testing:**
    - Validates the entire application flow from start to finish.
    - Ensures that all integrated components work together seamlessly.

Now, let's delve into automation testing using Selenium and Python.

Automation Testing with Selenium and Python:

Selenium: Selenium is a powerful open-source tool widely used for automating web applications for testing purposes. It supports multiple programming languages, including Python. Selenium provides a way for interacting with web elements, performing actions like clicking, typing, and validating, and capturing screenshots.

Tools and Packages Required:

1. **Python:**

- Make sure Python is installed on your machine. You can download it from python.org (https://www.python.org/).

2. **Selenium WebDriver:**

   - Install the Selenium WebDriver for Python using the following command:

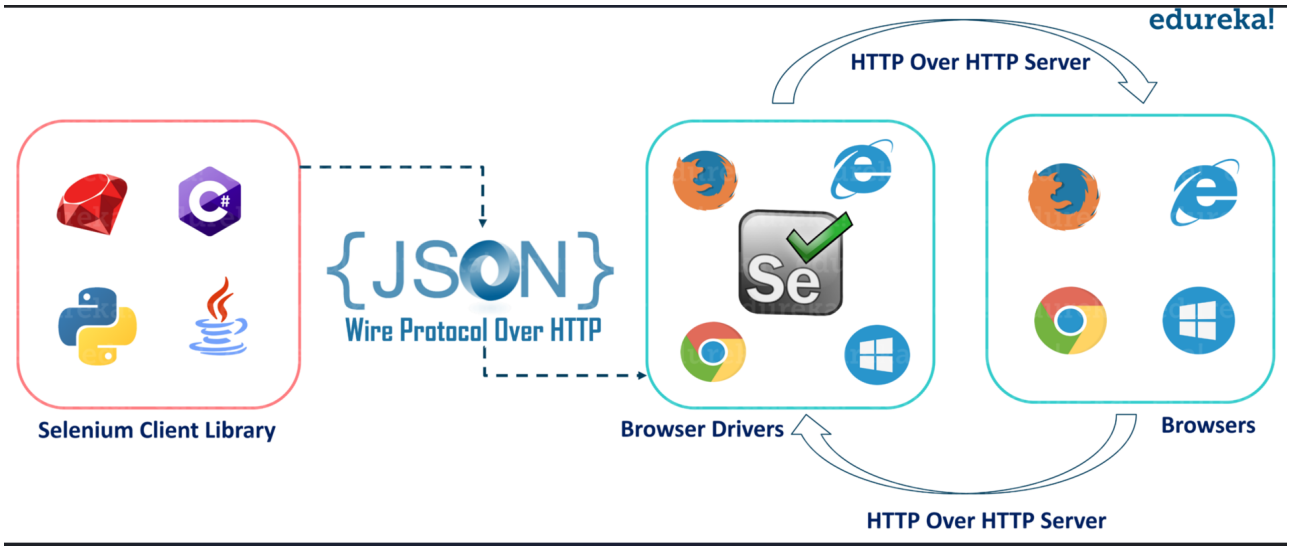
     ```
     pip install selenium
     ```

3. **Web Browser Driver:**

   - Download the web driver for the browser you want to automate (e.g., ChromeDriver for Google Chrome, GeckoDriver for Mozilla Firefox). Ensure it's in your system's PATH or provide the path in your script.

Webdriver

A WebDriver, in the context of Selenium, is a tool or an interface that allows you to interact with web browsers and automate the testing of web applications. It provides a programming interface to control and manipulate a browser's behavior, enabling automated testing of web applications across different browsers and platforms.



Here are key points about WebDriver and its use:

1. **Automation Interface:**

   - WebDriver serves as an automation interface that allows you to script interactions with web browsers using programming languages like Python, Java, C#, Ruby, etc.

2. **Cross-browser Testing:**

   - WebDriver enables cross-browser testing, allowing you to run your tests on different browsers like Chrome, Firefox, Safari, and others. This is crucial for ensuring that your web application behaves consistently across various browsers.

3. **Interacting with Web Elements:**

   - WebDriver provides methods to interact with various web elements such as buttons, text fields, checkboxes, and more. You can perform actions like clicking, typing, submitting forms, and validating content.

4. **Navigation:**

   - You can use WebDriver to navigate through different pages of a website by opening URLs, clicking links, or using back and forward navigation.

5. **JavaScript Execution:**

- WebDriver allows the execution of JavaScript code within the context of a web page. This is useful for handling dynamic elements and performing complex interactions.

6. **Screenshot Capture:**
   - WebDriver provides the capability to capture screenshots during test execution. This is useful for visually verifying the state of the application at different points in your test.
7. **Headless Browsing:**
   - WebDriver supports headless browser testing, which means you can run tests without a graphical user interface. This is useful for running tests in environments without a display server.
8. **Parallel Test Execution:**
   - WebDriver can be used to run tests in parallel, improving the efficiency of test execution and reducing the overall test run time.
9. **Integration with Testing Frameworks:**
   - WebDriver is often used in conjunction with testing frameworks like JUnit, TestNG, or pytest to structure and organize test code, manage test data, and report test results.
10. **Open Source:**

- Selenium WebDriver is open source, making it widely adopted and supported by a large community. This community-driven development ensures continuous improvement and compatibility with the latest web technologies.

In summary, WebDriver is a crucial component in Selenium that facilitates the automation of interactions with web browsers, enabling efficient and reliable testing of web applications across different browsers and scenarios. It provides a standardized way to control browsers programmatically and is a fundamental tool for anyone involved in web application testing or automation.

Let's Start, Open Browser using Selenium

In [4]:
```
# !pip install selenium
```

In [6]:
```
# Import the webdriver module from the selenium library
from selenium import webdriver

# Opening Chrome web browser
chrome_driver = webdriver.Chrome()  # Create an object for the Chrome webdriver
chrome_driver.maximize_window()  # Maximize the Chrome browser window

# Opening Firefox browser
firefox_driver = webdriver.Firefox()  # Create an object for the Firefox webdriv
firefox_driver.maximize_window()  # Maximize the Firefox browser window

# Opening Edge browser
edge_driver = webdriver.Edge()  # Create an object for the Edge webdriver
edge_driver.maximize_window()  # Maximize the Edge browser window
```

Code Explanation:

1. `from selenium import webdriver` : Import the `webdriver` module from the `selenium` library. This module provides a way to automate web browsers.
2. `chrome_driver = webdriver.Chrome()` : Create an instance of the Chrome webdriver, which will open the Chrome browser with a blank tab.

3. `chrome_driver.maximize_window()` : Maximize the Chrome browser window to make it full-screen.
4. `firefox_driver = webdriver.Firefox()` : Create an instance of the Firefox webdriver, which will open the Firefox browser.
5. `firefox_driver.maximize_window()` : Maximize the Firefox browser window.
6. `edge_driver = webdriver.Edge()` : Create an instance of the Edge webdriver, which will open the Edge browser.
7. `edge_driver.maximize_window()` : Maximize the Edge browser window.

Make sure you have the required browser drivers (chromedriver.exe, geckodriver.exe, and msedgedriver.exe) available in your system PATH or provide their paths explicitly in the webdriver instantiation if they are not in the PATH.

- **Note: In Seleniuim4.0 and later versions drivers not reuqired to download as it's already download from server, even no need to specify the driver Path**

Closing Opened Browser

In Selenium, there are two methods to close a browser window: `close()` and `quit()` .

1. **`close()` :**

    - The `close()` method is used to close the current browser window or tab that the WebDriver is currently handling.
    - It leaves other open browser windows or tabs unaffected.
    - It is generally used when you want to close a specific browser window or tab, but keep the WebDriver session running.

    ```
    # Example using close()
    chrome_driver.close()  # Close the Chrome browser window
    ```

2. **`quit()` :**

    - The `quit()` method is used to exit the entire WebDriver session, closing all open browser windows or tabs associated with that WebDriver instance.
    - It not only closes the browser windows but also releases the associated driver executable process and resources.
    - It is recommended to use `quit()` at the end of your script to ensure all resources are properly released.

    ```
    # Example using quit()
    chrome_driver.quit()  # Quit the entire Chrome WebDriver session
    ```

Why Close and Quit?

Closing and quitting browser windows is essential for several reasons:

1. **Resource Management:**

    - Browsers consume system resources. If you have multiple browser windows or tabs open, they can collectively use a significant amount of memory. Closing unnecessary windows helps manage resources efficiently.
2. **Clean Test Environment:**

    - In automated testing, it's crucial to start with a clean and consistent state for each test. Closing or quitting browsers ensures that each test starts with a fresh browser instance.
3. **Preventing Memory Leaks:**

- Closing or quitting the browser helps prevent memory leaks. Not closing browser instances properly may lead to memory buildup over time, causing instability or unexpected behavior.

4. **Driver Cleanup:**

   - The `quit()` method releases the WebDriver executable process and resources. This is important, especially if your script spawns multiple browser instances during its execution. Failing to quit the WebDriver could result in lingering processes.

In summary, while `close()` is used to close a specific browser window, `quit()` is used to terminate the entire WebDriver session, closing all associated windows and releasing resources. Properly managing browser instances ensures efficient resource utilization and a clean testing environment. It's a good practice to use `quit()` at the end of your script to ensure proper cleanup.

In [10]:

```python
# Import the webdriver module from the selenium library
from selenium import webdriver
import time

# Opening browsers
chrome_driver = webdriver.Chrome()
print("Openning Chrom browser")
firefox_driver = webdriver.Firefox()
print("Openning Firefox browser")
edge_driver = webdriver.Edge()
print("Openning Edge browser")


time.sleep(3) # taling 3 secs pause
# closing browsers using close() method
chrome_driver.close()
print("Chrom Closed using close() method")
firefox_driver.close()
print("Firefox Closed using close() method")
edge_driver.close()
print("Edge Closed using close() method")

print("--------------------")

time.sleep(4)
# Again, Opening browsers
chrome_driver = webdriver.Chrome()
print("Openning Chrom browser")
firefox_driver = webdriver.Firefox()
print("Openning Firefox browser")
edge_driver = webdriver.Edge()
print("Openning Edge browser")


time.sleep(3)
# closing browsers using quit() method
chrome_driver.quit()
print("Chrom Closed using quit() method")
firefox_driver.quit()
print("Firefox Closed using quit() method")
edge_driver.quit()
print("Edge Closed using quit() method")
```

```
Openning Chrom browser
Openning Firefox browser
Openning Edge browser
Chrom Closed using close() method
Firefox Closed using close() method
Edge Closed using close() method
Openning Chrom browser
Openning Firefox browser
Openning Edge browser
Chrom Closed using quit() method
Firefox Closed using quit() method
Edge Closed using quit() method
```

Open a website

- Open any website or URL using `driver.get(_website_link_here_)`

In [12]:

```python
# Import the webdriver module from the selenium library
from selenium import webdriver

# Import the time module for introducing delays in the script
import time

# Create a new instance of the Chrome webdriver
driver = webdriver.Chrome()

# Open a website (replace the URL with the one you want to open)
website_url = "https://www.youtube.com"
driver.get(website_url)

# Optionally, maximize the browser window for a better view
driver.maximize_window()

# Do any other interactions or tests on the website as needed

# Introduce a delay of 5 seconds (useful for observing the browser interaction)
time.sleep(5)

# Close the browser window
driver.quit()
```

In [13]:

```python
# Open Same website in all browsers
from selenium import webdriver
import time

website = "https://www.youtube.com/"

# Opening Chrome web browser
chrome_driver = webdriver.Chrome()  # Create an object for the Chrome webdriver
chrome_driver.maximize_window()  # Maximize the Chrome browser window
chrome_driver.get(website) # Open website in chrome

# Opening Firefox browser
firefox_driver = webdriver.Firefox()  # Create an object for the Firefox webdriv
firefox_driver.maximize_window()  # Maximize the Firefox browser window
firefox_driver.get(website) # Open website in firefox

# Opening Edge browser
edge_driver = webdriver.Edge()  # Create an object for the Edge webdriver
edge_driver.maximize_window()  # Maximize the Edge browser window
edge_driver.get(website) # Open website in edge

time.sleep(5)
chrome_driver.quit()
firefox_driver.quit()
edge_driver.quit()
```

Important Keywords and Methods

Selenium is a powerful tool for automating web browser interactions, and it provides a variety of methods, keywords, functions, and classes. Here are some important ones:

Methods:

1. **`get(url)` :** Navigates to the specified URL.

```
driver.get("https://www.example.com")
```

2. **`find_element(by, value)` and `find_elements(by, value)` :** Locates a single or multiple elements on the page.

```
element = driver.find_element("id", "username")
```

3. **`send_keys(value)` :** Simulates typing into the element.

```
element.send_keys("Hello, Selenium!")
```

4. **`click()` :** Clicks the element.

```
element.click()
```

5. **`clear()` :** Clears the text from an input field.

```
element.clear()
```

6. **`get_attribute(name)` :** Gets the value of the specified attribute of the element.

```
value = element.get_attribute("href")
```

7. **`text` :** Gets the visible text of the element.

```
text = element.text
```

8. **`is_displayed()` , `is_enabled()` , `is_selected()` :** Checks the visibility, enablement, and selection status of an element, respectively.

```
visible = element.is_displayed()
```

9. **`title` and `current_url` :** Get the title and current URL of the page.

```
title = driver.title
current_url = driver.current_url
```

Keywords and Functions:

1. **`WebDriverWait` and `expected_conditions` :** Used for waiting until certain conditions are met.

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

element = WebDriverWait(driver, 10).until(EC.presence_of_element_located
((By.ID, "username")))
```

2. **`By` class:** Represents the mechanism to locate elements.

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

element = driver.find_element(By.XPATH, "//input[@id='username']")
```

Classes:

1. **`webdriver.WebDriver` :** The main class for WebDriver that provides methods for browser interactions.

```
from selenium import webdriver

driver = webdriver.Chrome()
```

2. **`webdriver.WebElement`** : Represents an HTML element and provides methods for interacting with it.

```
element = driver.find_element("id", "username")
```

3. **`webdriver.ActionChains`** : Class for handling advanced user interactions like drag-and-drop and keypress.

```
from selenium.webdriver.common.action_chains import ActionChains

actions = ActionChains(driver)
actions.move_to_element(element).perform()
```

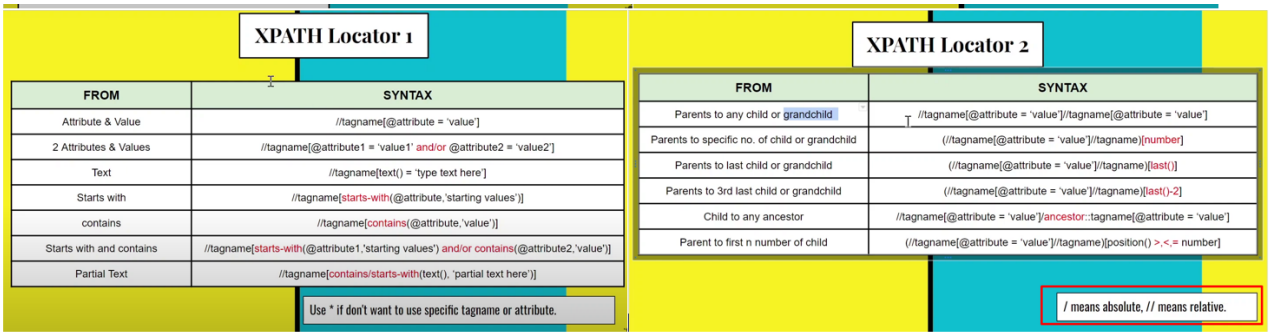
4. **`webdriver.DesiredCapabilities`** : Class for specifying desired capabilities when initializing a browser.

```
from selenium import webdriver

capabilities = webdriver.DesiredCapabilities.CHROME
driver = webdriver.Chrome(desired_capabilities=capabilities)
```

These are just a few examples of the many methods, keywords, functions, and classes available in Selenium. Depending on your testing needs, you may explore more advanced features and utilities provided by the Selenium library. Always refer to the official [Selenium documentation](https://www.selenium.dev/documentation/en/) for the most up-to-date information.

Element

In the context of web automation using Selenium, an "element" refers to an HTML element on a web page. HTML elements are the building blocks of a web page, and they include various types such as buttons, text fields, checkboxes, links, and more. Selenium allows you to interact with these elements programmatically.



How to Find Elements:

Selenium provides several methods to locate and interact with elements on a web page. The primary mechanism for finding elements is the `find_element` method. Here are some common strategies to locate elements:

1. **By ID:**

   - Locate an element using its unique `id` attribute.

```
element = driver.find_element(By.ID, "element_id")
```

2. **By Name:**
   - Locate an element using its `name` attribute.

     ```
     element = driver.find_element(By.NAME, "element_name")
     ```

3. **By Class Name:**
   - Locate an element using its class name.

     ```
     element = driver.find_element(By.CLASS_NAME, "element_class")
     ```

4. **By Tag Name:**
   - Locate an element using its HTML tag name.

     ```
     element = driver.find_element(By.TAG_NAME, "input")
     ```

5. **By Link Text:**
   - Locate a link element by its visible text.

     ```
     element = driver.find_element(By.LINK_TEXT, "Click me")
     ```

6. **By Partial Link Text:**
   - Locate a link element by a portion of its visible text.

     ```
     element = driver.find_element(By.PARTIAL_LINK_TEXT, "Click")
     ```

7. **By XPath:**
   - Locate an element using XPath expressions.

     ```
     element = driver.find_element(By.XPATH, "//input[@id='username']")
     ```

8. **By CSS Selector:**
   - Locate an element using a CSS selector.

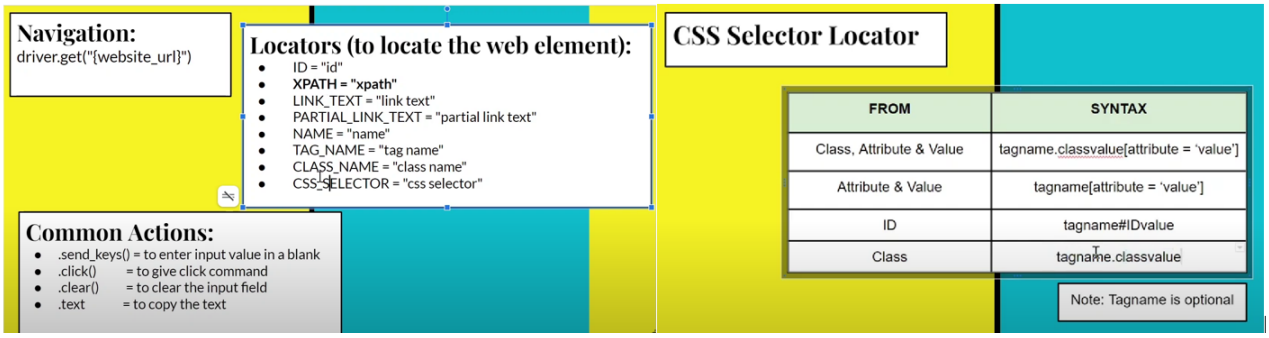
     ```
     element = driver.find_element(By.CSS_SELECTOR, "input#username")
     ```

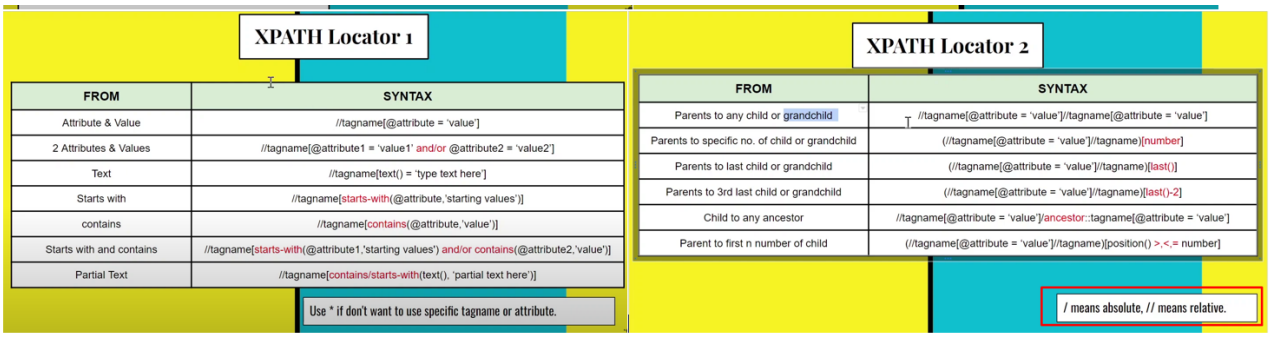
Remember to import `By` from `selenium.webdriver.common.by` in your code.

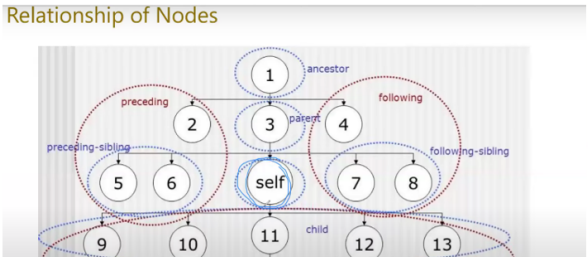
After finding an element, you can perform various actions on it, such as sending keys, clicking, retrieving attributes, or checking its visibility. For example:

```
# Typing into an input field
element.send_keys("Hello, Selenium!")

# Clicking a button
element.click()

# Getting the text of an element
text = element.text
```

Choose the appropriate locator strategy based on the structure of the HTML and the specific attributes of the elements you want to interact with. It's often helpful to use browser developer tools to inspect the HTML structure of a webpage and identify suitable locators for elements.







XPath axes

Axes	Description	Syntax
Child	Traverse all **child element** of the **current html tag**	//*[attribute='value']/child::tagname
Parent	Traverse **parent element** of the **current html tag**	//*[attribute='value']/parent::tagname
Following	Traverse **all element** that comes **after the current tag**	//*[attribute='value']/following::tagname
Preceding	Traverse **all nodes** that comes **before the current html tag.**	//*[attribute='value']/preceding::tagname
Following-sibling	Traverse from **current Html tag** to **Next sibling Html tag.**	//current html tag[@attribute ='value']/following-sibling:: sibling tag[@attribute ='value']
Preceding-sibling	Traverse from **current Html tag** to **previous sibling Html tag.**	//current html tag[@attribute ='value']/preceding-sibling:: previous tag[@attribute ='value']
Ancestor	Traverse all the **ancestor elements** (grandparent, parent, etc.) of the	//*[attribute='value']/ancestor::tagname

In [ ]:
```
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get("https://www.youtube.com/")
driver.find_element(By.ID, "search")
```

In [58]:
```
# If browser close automatically then follow below code
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options

options = Options()
options.add_experimental_option('detach', True) # disable auto close browser
```

In [1]:
```
# some demo websites for automation testing
rahul = "https://rahulshettyacademy.com/AutomationPractice/"
nopcom = "https://demo.nopcommerce.com/"
orange = "https://opensource-demo.orangehrmlive.com/web/index.php/auth/login"
heroku = "https://the-internet.herokuapp.com/"
testauto = "https://testautomationpractice.blogspot.com/"
click_tets = "https://skill-test.net/mouse-double-click"
fb = "https://www.facebook.com/"
google = "https://www.google.com/"
```

Wait Conditions

- In Selenium, waiting is a crucial aspect of automating web interactions, as it allows the script to pause execution until a certain condition is met or a specific element becomes available. Waiting is essential because web pages may take some time to load, and elements may appear or change dynamically.

There are two types of waits in Selenium:

1. **Implicit Wait:**

   - An implicit wait tells the WebDriver to **wait for a certain amount of time** before throwing an exception if an element is not immediately available.
   - It is **set once for the entire session** and remains effective until the WebDriver is closed.

In [72]:
```
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.implicitly_wait(10)
# Set implicit wait to 10 seconds, it declare only once at top
# Now, any subsequent element lookups will wait up to 10 seconds before raising

driver.get(rahul)

text = driver.find_element(By.ID, "autocomplete")
text.send_keys("Hello, Rohit!")
print(text.get_attribute("value"))

driver.quit()
```

```
Hello, Rohit!
```

2. **Explicit Wait:**

   - An explicit wait is a more granular way of waiting for certain conditions before proceeding with the test.
   - It allows you to **wait for a specific condition to be met** before proceeding further in the code.
   - It is defined for a certain condition and a maximum time limit.
   - Example in Python using WebDriverWait:

In [7]:

```python
# Basic use of Explicit Wait
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

driver = webdriver.Chrome()
driver.maximize_window()

mywait = WebDriverWait(driver, 10)  # Explicit wait declaration

driver.get(google)

search = driver.find_element(By.NAME, "q")
search.send_keys("Selenium")
search.submit()

# use of explicit wait
search_link = mywait.until(EC.presence_of_element_located((By.XPATH, "//a[contai
search_link.click()


# 2nd use of explicit wait
open_link = mywait.until(EC.presence_of_element_located((By.CLASS_NAME, "seleniu
print(open_link.text)

driver.quit()
```

```
Selenium WebDriver
```

`WebDriverWait` in Selenium is a class that provides explicit wait conditions for waiting until certain conditions are met before proceeding with the execution of the script. It is a part of the Selenium WebDriver's expected conditions framework.

Here's the signature of the `WebDriverWait` class:

```
class WebDriverWait(driver, timeout, poll_frequency=0.5, ignored_exceptions=
None)
```

Let's break down the parameters:

1. **`driver`** : This is the WebDriver instance (e.g., `webdriver.Chrome()` , `webdriver.Firefox()`) on which the wait will be applied.
2. **`timeout`** : This is the maximum amount of time, in seconds, to wait for a condition to be met. If the condition is not met within this time, a `TimeoutException` will be raised.
3. **`poll_frequency`** : This is the time, in seconds, that `WebDriverWait` will sleep between each call to the method being waited for. By default, it's set to 0.5 seconds. It determines how often the expected condition is checked.
4. **`ignored_exceptions`** : This is an optional parameter that allows you to specify a tuple of exceptions that should be ignored during the wait process. If any of the specified exceptions occur, they will be ignored, and the wait will continue.

Here's an example of how you might use `WebDriverWait` :

```python
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

# Create a WebDriver instance
driver = webdriver.Chrome()

# Navigate to a webpage
driver.get("https://example.com")

# Use WebDriverWait to wait for an element to be present on the page
element = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "some_element_id"))
)

# Perform actions after the element is found
element.click()

# Close the browser window
driver.quit()
```

In this example, `WebDriverWait` is used to wait for the presence of an element with the ID

Both implicit and explicit waits are essential for handling dynamic content, AJAX requests, and ensuring that the WebDriver does not attempt to interact with elements before they are available on the page. Explicit waits are generally preferred for their flexibility and precision in handling specific conditions.

Explicit Wait Exception Handling

- Explicit wait can handle exception
- In above example exception cannot handle as we didn't declare which exception has be handled

In [16]:

```python
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
from selenium.common.exceptions import NoSuchElementException, ElementNotSelecta

driver = webdriver.Chrome()
driver.maximize_window()

# Wait declaration with exceptions that's to be handled
mywait = WebDriverWait(driver, 10, ignored_exceptions=[NoSuchElementException, E

# Define the URL
google = "https://www.google.com"

try:
    driver.get(google)

    search = driver.find_element(By.NAME, "q")
    search.send_keys("Selenium")
    search.submit()

    # use of explicit wait
    search_link = mywait.until(EC.presence_of_element_located((By.XPATH, "//a[co
    search_link.click()
    print("Search clicked")

    # Correct the class name
    open_link = mywait.until(EC.presence_of_element_located((By.CLASS_NAME, "sle
    print(open_link.text)

except TimeoutException:
    print("Timeout occurred during the explicit wait. Element not found within t

finally:
    driver.quit()

```

```
Search clicked
Timeout occurred during the explicit wait. Element not found within the specified t
ime.
```

Alerts/Pop-ups Handling

Handling alerts (pop-ups) in Selenium involves using the `Alert` interface provided by Selenium. This interface provides methods to interact with JavaScript alerts, confirms, and prompts. Here's a basic overview of how you can handle different types of alerts:

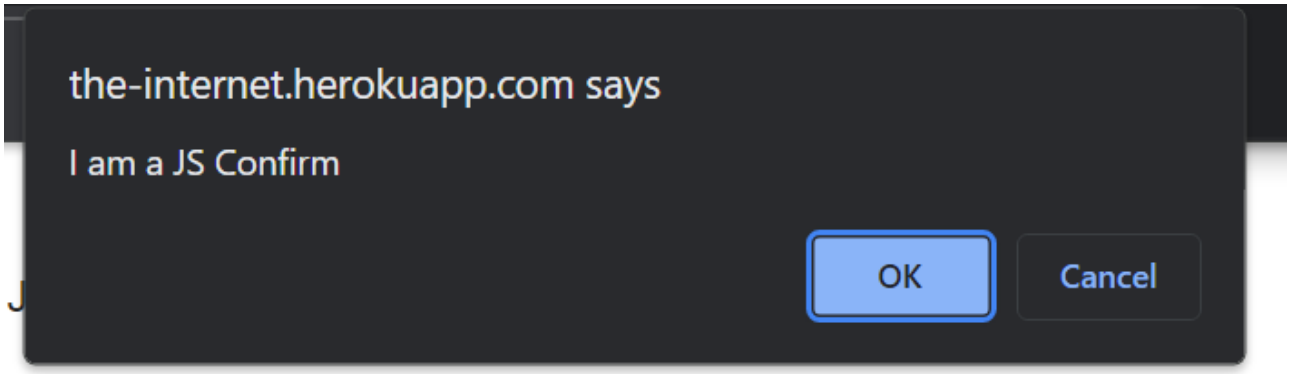
1. Handling JavaScript Alerts:

In [57]:

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

heroku = "https://the-internet.herokuapp.com/"
driver = webdriver.Chrome()
driver.get(heroku)
driver.maximize_window()

# Opening Alert page
driver.find_element(By.LINK_TEXT, "JavaScript Alerts").click()
driver.find_element(By.XPATH, "//button[@onclick='jsAlert()']").click()


# Switch to the alert
alert = driver.switch_to.alert

# Get the text of the alert
alert_text = alert.text
print("Alert Text:", alert_text)

time.sleep(2)
# Accept the alert (click OK)
alert.accept()

# Dismiss the alert (click Cancel)
# alert.dismiss()

# switch driver focus to html page/ default
driver.switch_to.default_content()
print("Result: ", driver.find_element(By.ID, "result").text)

time.sleep(2)
driver.quit()
```

```
Alert Text: I am a JS Alert
Result:  You successfully clicked an alert
```

2. Handling JavaScript Confirms:

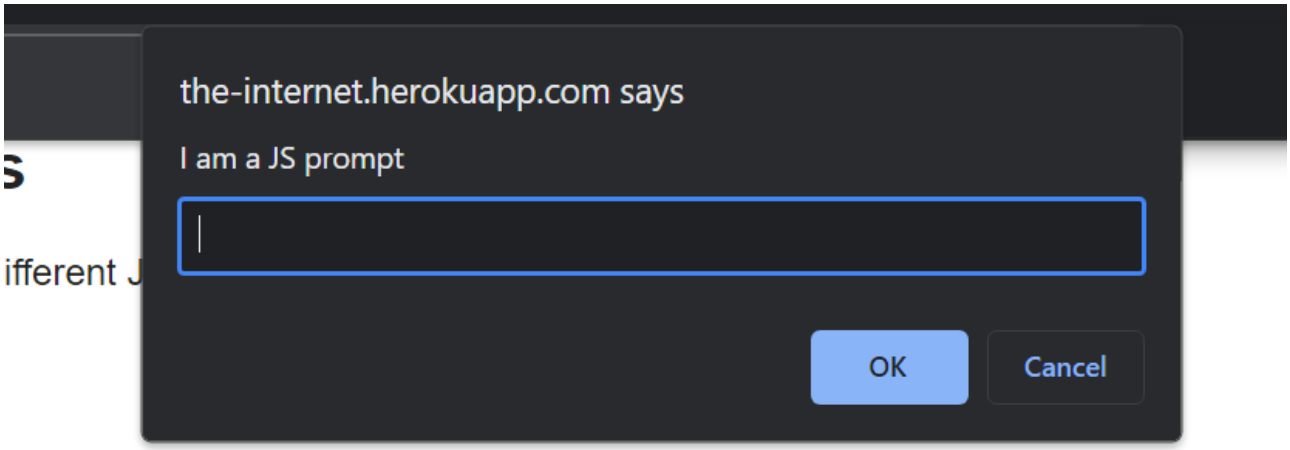


In [61]:

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

heroku = "https://the-internet.herokuapp.com/"
driver = webdriver.Chrome()
driver.get(heroku)
driver.maximize_window()

# Opening Alert page
driver.find_element(By.LINK_TEXT, "JavaScript Alerts").click()
driver.find_element(By.XPATH, "//button[@onclick='jsConfirm()']").click()

# Switch to the confirm
confirm = driver.switch_to.alert

# Get the text of the confirm
confirm_text = confirm.text
print("Confirm Text:", confirm_text)

time.sleep(2)
# Accept the confirm (click OK)
# confirm.accept()

# Dismiss the confirm (click Cancel)
confirm.dismiss()

# switch driver focus to html page/ default
driver.switch_to.default_content()
print("Result: ", driver.find_element(By.ID, "result").text)

time.sleep(2)
driver.quit()
```

```
Confirm Text: I am a JS Confirm
Result:  You clicked: Cancel
```

3. Handling JavaScript Prompts:



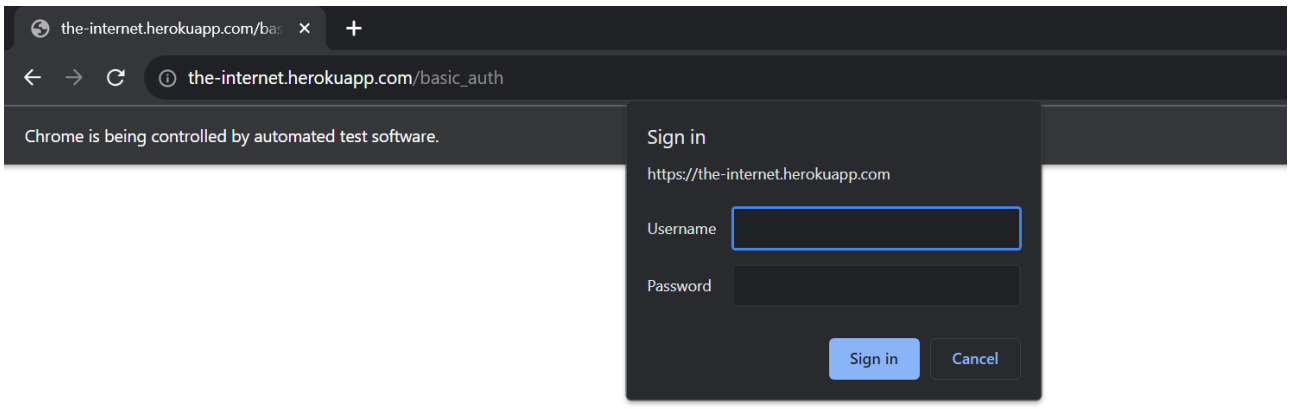
In [62]:

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

heroku = "https://the-internet.herokuapp.com/"
driver = webdriver.Chrome()
driver.get(heroku)
driver.maximize_window()

# Opening Alert page
driver.find_element(By.LINK_TEXT, "JavaScript Alerts").click()
driver.find_element(By.XPATH, "//button[@onclick='jsPrompt()']").click()

# Switch to the prompt
prompt = driver.switch_to.alert

# Get the text of the prompt
prompt_text = prompt.text
print("Prompt Text:", prompt_text)

time.sleep(2)
# Enter text into the prompt
prompt.send_keys("Selenium is awesome!")

# Accept the prompt (click OK)
prompt.accept()

# Dismiss the prompt (click Cancel)
# prompt.dismiss()

# switch driver focus to html page/ default
driver.switch_to.default_content()
print("Result: ", driver.find_element(By.ID, "result").text)

time.sleep(2)
driver.quit()
```

```
Prompt Text: I am a JS prompt
Result:  You entered: Selenium is awesome!
```

Authentication Pop-up

- Handling authentication pop-ups in Selenium involves using a combination of your browser's built-in features and Selenium commands. Authentication pop-ups are browser-level dialogs that typically appear when accessing a webpage that requires username and password authentication.



In [66]:

```python
from selenium import webdriver
import time
# Provide authentication credentials
username = "admin"
password = "admin"

# Set up Chrome options with authentication credentials
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--disable-extensions')
chrome_options.add_argument('--disable-infobars')
chrome_options.add_argument('--ignore-certificate-errors')
chrome_options.add_argument(f'--user-data-dir={username}:{password}@httpbin.org'

# Create a WebDriver instance with Chrome options
driver = webdriver.Chrome(options=chrome_options)

# Navigate to a webpage that requires authentication
driver.get("https://the-internet.herokuapp.com/")

# Continue interacting with the page as needed

time.sleep(2)
# Close the browser window
driver.quit()

```

In [69]:

```python
from selenium import webdriver
import time

# Provide authentication credentials
username = "admin"
password = "admin"

# Create a WebDriver instance
driver = webdriver.Chrome()

# Construct the URL with authentication credentials
url_with_credentials = f"https://{username}:{password}@the-internet.herokuapp.co

# Navigate to the URL
driver.get(url_with_credentials)
print("Page Title: ",driver.title)
print("Page URL: ", driver.current_url)

# Continue interacting with the page as needed

time.sleep(3)
# Close the browser window
driver.quit()

```

```
Page Title:  The Internet
Page URL:  https://admin:admin@the-internet.herokuapp.com/basic_auth (https://admi
n:admin@the-internet.herokuapp.com/basic_auth)
```

Remember to adapt the XPath or other locators based on the structure of the webpage you are working with. Additionally, uncomment the lines related to accepting or dismissing alerts, confirms, or prompts based on your specific use case.

Frames/iFrames

In HTML, frames and iframes are used to divide a web page into multiple sections or to embed external content within a webpage. They allow developers to create a layout where different parts of the webpage can load content independently.

Frames:

Frames are an older HTML feature that divides a webpage into multiple independent sections, each with its own HTML document. The `<frame>` tag is used to define each frame, and the `<frameset>` tag is used to define the overall structure of frames on the page. Each frame can load a separate HTML document.

Example of using frames in HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Frames Example</title>
</head>
<frameset cols="25%,75%">
  <frame src="frame1.html" name="frame1">
  <frame src="frame2.html" name="frame2">
</frameset>
</html>
```

IFrames (Inline Frames):

IFrames, short for "inline frames," are a more modern and flexible way of achieving similar functionality to frames. The `<iframe>` tag is used to embed another HTML document within the current HTML document. IFrames are commonly used to include external content (such as ads or embedded videos) within a webpage.

Example of using iframes in HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>IFrame Example</title>
</head>
<body>
  <h2>Main Content</h2>
  <iframe src="external_content.html" width="600" height="400" title="Extern
al Content"></iframe>
</body>
</html>
```

Key points about frames and iframes:

1. **Frameset vs. Iframe**: Frames are typically defined using the `<frameset>` tag, while iframes are defined using the `<iframe>` tag.
2. **Independence**: Each frame in a frameset has its own HTML document, whereas an iframe embeds content within the main HTML document.

3. **Communication**: Communication between frames can be more challenging and often requires the use of JavaScript. In iframes, communication is often handled more seamlessly.
4. **Browser Support**: IFrames are more widely supported in modern browsers, while frames are considered outdated and are not commonly used in contemporary web development.
5. **Responsive Design**: IFrames can be more easily integrated into responsive web design compared to frames.

When developing web applications, it is recommended to use iframes over frames for better compatibility and maintainability. However, the use of frames and iframes should be done judiciously,

In [15]:
```python
# Iframe Switch
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get("https://yourwebsite.com")

# Locate the iframe element
iframe = driver.find_element(By.ID, "myiframe")

# Switch to the iframe context
driver.switch_to.frame(iframe)

# Now you are inside the iframe and can interact with its elements
element_inside_iframe = driver.find_element(By.ID, "element_inside_iframe")
element_inside_iframe.send_keys("Text inside iframe")

# Switch back to the default content
driver.switch_to.default_content()

# Perform actions outside the iframe
element_outside_iframe = driver.find_element(By.ID, "element_outside_iframe")
element_outside_iframe.click()

driver.quit()
```

Browser Windows Handling

Browser window handling in Selenium is crucial when dealing with scenarios where a web application opens multiple browser windows or tabs. Selenium provides methods to switch between these windows. Here's a basic guide on how to handle multiple browser windows:

1. Opening New Windows or Tabs:

To open a new window or tab, you can use JavaScript or execute specific actions in your application that trigger the opening of a new window or tab. For example, clicking a link with the attribute `target="_blank"` might open a new window.

- open new window: `driver.switch_to.new_window("window")`
- open new tab: `driver.switch_to.new_window("tab")`

2. Getting Window Handles:

Selenium maintains a list of window handles. Each handle represents a unique browser window or tab. You can use `driver.window_handles` to get a list of all the window handles.

```
# Get the handles of all open windows
window_handles = driver.window_handles
```

3. Switching Between Windows:

Use `driver.switch_to.window(handle)` to switch between different windows. Pass the handle of the window you want to switch to as an argument.

```
# Switch to the first window
driver.switch_to.window(window_handles[0])
```

In Selenium, the **window handles (often referred to as "window IDs") are not static.** They are dynamically assigned by the browser and can change during the execution of a test script. Each window or tab opened by the browser is assigned a unique identifier (handle), and these identifiers are subject to change based on the order in which windows are opened or closed.

When you call `driver.window_handles`, Selenium returns a list of current window handles. The order of handles in this list may change if new windows are opened or existing windows are closed.

It's important to note the following:

1. **Order of Handles**: The order of window handles in the list corresponds to the order in which the windows were opened, with the first handle representing the main window.
2. **Dynamic Nature**: Since window handles are dynamic, it's crucial to retrieve the handles when needed and not rely on hard-coded indices, as they may change.
3. **Switching Between Windows**: When switching between windows using `driver.switch_to.window(handle)`, always use the actual handle obtained from `driver.window_handles`.

Example:

```
# Get the handles of all open windows
window_handles = driver.window_handles

# Switch to the second window
driver.switch_to.window(window_handles[1])
```

Remember to close the windows or tabs appropriately to avoid leaving unnecessary browser instances running. You can use `driver.close()` to close the current window and `driver.quit()` to close the entire browser.

This example assumes that you have opened a new window during the execution of your test script. If your application opens new windows in response to user actions, you may need to adjust the script

In [122]:

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

driver = webdriver.Chrome()
driver.maximize_window()
driver.get(testauto)
driver.implicitly_wait(5)

wiki = driver.find_element(By.ID, "Wikipedia1_wikipedia-search-input")
wiki.send_keys("Cricket")
wiki.submit()

# Opening new window tabs using the attribute target="_blank"
wiki_links = driver.find_elements(By.XPATH, "//div[@id='wikipedia-search-result-
for link in wiki_links:
    link.click()

# printing id of current tab
current_window_id = driver.current_window_handle
print("Id of current window: ", current_window_id)

# Id of all opened tabs
all_opened_windows = driver.window_handles
print("List of All Opened Windows ID: ", all_opened_windows)

# switching windows and printing title with window id
for i in all_opened_windows:
    driver.switch_to.window(i)
    print(f"Window Tab Title: {driver.title.ljust(30)}, Window ID: {i}")

# switching driver to first window
driver.switch_to.window(all_opened_windows[0])

# confirm the title of first window
print("Current Window Title: ", driver.title)

# window close
# driver.close () # close current window only
# driver.quit() # close all windows at once

for i in all_opened_windows:
    driver.switch_to.window(i)
    print(f"{driver.title.ljust(30)} : Closed")
    driver.close()
    time.sleep(2)
```

```
Id of current window:  2C072DC543F6D268848B8AE9ADE9446F
List of All Opened Windows ID:  ['2C072DC543F6D268848B8AE9ADE9446F', '6C8014ADAB41A
C8C53853C7BE2AE583B', 'A411C329EF48AC8DA5C4015A9D042A46', 'F1E43A7DCE21143351A53990
87382DD8', 'C7C2662CEB0B1DE593909CA4179A7052', '0A198008C8CD48365570D1FE690F0CFF']
Window Tab Title: Automation Testing Practice   , Window ID: 2C072DC543F6D268848B8A
E9ADE9446F
Window Tab Title: Cricket in India - Wikipedia  , Window ID: 6C8014ADAB41AC8C53853C
7BE2AE583B
Window Tab Title: Cricket pitch - Wikipedia     , Window ID: A411C329EF48AC8DA5C401
5A9D042A46
Window Tab Title: Cricket (insect) - Wikipedia  , Window ID: F1E43A7DCE21143351A539
9087382DD8
Window Tab Title: Cricket - Wikipedia           , Window ID: C7C2662CEB0B1DE593909C
A4179A7052
Window Tab Title: Cricket World Cup - Wikipedia , Window ID: 0A198008C8CD48365570D1
FE690F0CFF
Current Window Title:  Automation Testing Practice
Automation Testing Practice    : Closed
Cricket in India - Wikipedia   : Closed
Cricket pitch - Wikipedia      : Closed
Cricket (insect) - Wikipedia   : Closed
Cricket - Wikipedia            : Closed
Cricket World Cup - Wikipedia  : Closed
```

Open New Window/Tab

- Open new window or tab
- open url in new window or tab
- to open new window follow this code: `driver.switch_to.new_window("window")`
- to open new tab follow this code : `driver.switch_to.new_window("tab")`

In [54]:

```python
import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By


options = Options()
# options.add_argument("--headless") # this is must for taking entire page scree
driver = webdriver.Chrome(options=options)
driver.maximize_window()

driver.get(nopcom)

time.sleep(2)
driver.switch_to.new_window("tab") # open new tab
time.sleep(2)
driver.get(google) # open this url in newly opened tab

time.sleep(2)
driver.switch_to.new_window("window") # open new window
time.sleep(2)
driver.get(fb) # open this url in newly opened window

time.sleep(2)
driver.switch_to.new_window("tab") # this new tab will be opened in previous win
time.sleep(2)
driver.get(testauto) # this url will be opened in above tab

for i in driver.window_handles:
    driver.switch_to.window(i)
    print("Title: ", driver.title, "ID: ", i)

print("Current Tab title: ", driver.title)

time.sleep(3)
driver.quit()
```

```
Title:  nopCommerce demo store ID:  9A795E0CB300DB22D2A50743AE15C57A
Title:  Google ID:  9302954CC33C82491416D47C017CB455
Title:  Facebook – log in or sign up ID:  39455A80F41626D71B8607576D8F9633
Title:  Automation Testing Practice ID:  0217E5FA494EF09E658E92A549C0E16D
Current Tab title:  Automation Testing Practice
```

Browser Level Settings

Browser level settings in Selenium refer to various configuration options and capabilities that can be set when initializing a browser session. These settings allow you to customize the behavior of the WebDriver and influence how the browser interacts with your automation scripts. Here are some common browser-level settings and how to use them:

1. Setting Browser Window Size:

You can set the initial size of the browser window using the `window-size` capability.

```
from selenium import webdriver

chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--window-size=1200,800")

driver = webdriver.Chrome(chrome_options=chrome_options)
```

2. Disabling Browser Extensions:

You can disable browser extensions using the `disable-extensions` capability.

```
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--disable-extensions")

driver = webdriver.Chrome(chrome_options=chrome_options)
```

3. Headless Mode:

You can run the browser in headless mode, which means without a graphical user interface. This is useful for running tests in the background without displaying the browser window.

```
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--headless")

driver = webdriver.Chrome(chrome_options=chrome_options)
```

4. Ignoring Certificate Errors:

To ignore SSL certificate errors, you can use the `ignore-certificate-errors` capability.

```
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--ignore-certificate-errors")

driver = webdriver.Chrome(chrome_options=chrome_options)
```

5. Setting Browser Language:

You can set the language of the browser using the `--lang` option.

```
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--lang=en-US")

driver = webdriver.Chrome(chrome_options=chrome_options)
```

6. Incognito Mode:

To open the browser in incognito (private browsing) mode, use the `--incognito` option.

```
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--incognito")

driver = webdriver.Chrome(chrome_options=chrome_options)
```

7. Setting Proxy:

You can set a proxy for your browser session using the `--proxy-server` option.

```
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--proxy-server=http://your-proxy-server:port")

driver = webdriver.Chrome(chrome_options=chrome_options)
```

8. Notifications/Alert Settings

To disable notifications in a Chrome browser controlled by Selenium, you can use the `--disable-notifications` argument. Here's an example in Python:

```
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--disable-notifications")

driver = webdriver.Chrome(chrome_options=chrome_options)
```

In [146]:

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
import time

options = Options()
options.add_argument("--incognito")
# options.add_argument("--disable-notifications")
options.add_argument("--window-size=1000,800")

driver = webdriver.Chrome(options=options)
driver.implicitly_wait(10)
# driver.maximize_window()

driver.get("https://whatmylocation.com/")
print(driver.title)
time.sleep(5)
driver.quit()
```

```
My Location Now - What is My Current Location?
```

Web Table

Working with web tables in Selenium involves locating the table, navigating its rows and columns, and extracting or interacting with the data. Let's go through some common tasks with web tables using Selenium in Python.

- `table` - tag use to define a **table** in html
- `thead` - use to define **table heading**
- `tr` - use to define **table row**
- `th` - actual **table heading** name
- `tbody` - define **table body** means data body below heading
- `td` - actual **table data**

Task 1: Get Table Content

Assuming you have an HTML table like this:

```
<table id="example">
    <thead>
        <tr>
            <th>Name</th>
            <th>Age</th>
            <th>Country</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>John</td>
            <td>25</td>
            <td>USA</td>
        </tr>
        <tr>
            <td>Jane</td>
            <td>30</td>
            <td>Canada</td>
        </tr>
        <!-- More rows... -->
    </tbody>
</table>
```

You can use Selenium to get the table content:

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://yourwebsite.com")

# Locate the table
table = driver.find_element_by_id("example")

# Extract data from the table
for row in table.find_elements_by_css_selector("tbody tr"):
    columns = row.find_elements_by_tag_name("td")
    data = [column.text for column in columns]
    print(data)

driver.quit()
```

Task 2: Perform Actions on the Table

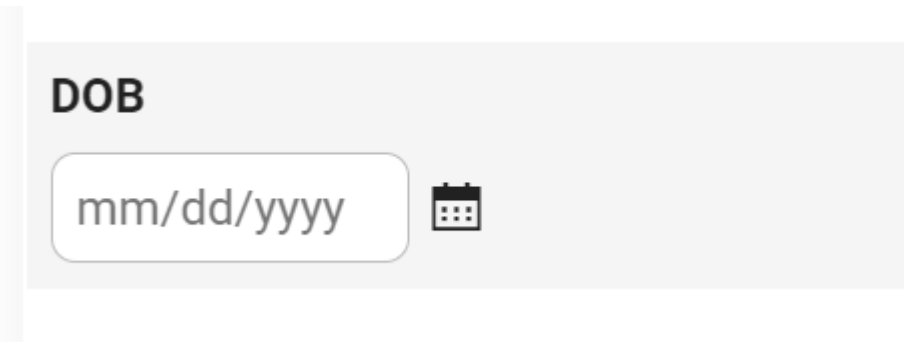
Let's say you want to click a button in a specific row:

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

driver = webdriver.Chrome()
driver.get("https://yourwebsite.com")

# Locate the table
table = driver.find_element_by_id("example")

# Find the row where Name is "John"
john_row = table.find_element(By.XPATH, "//tr[td='John']")
```

In [212]:

```python
# Get Table content
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

testauto = "https://testautomationpractice.blogspot.com/"
driver = webdriver.Chrome()
driver.maximize_window()
driver.get(testauto)

# no of rows
table_row = driver.find_elements(By.XPATH, "//table[@name='BookTable']//tr")
print("No of Rows in Table including heading: ", len(table_row))

# No of columns
table_col = driver.find_elements(By.XPATH, "//table[@name='BookTable']//th")
print("No of Columns in Table: ", len(table_col))

# table data
for row in range(2, len(table_row)+1):
    for col in range(1, len(table_col)+1):
        data = driver.find_element(By.XPATH, f"//table[@name='BookTable']//tr[{r
        print(data.ljust(20), end=" ")
    print("")

# table data based on conditions(list of books whose author is Mukesh)
books_ls = []
for row in range(2, len(table_row)+1):
    author = driver.find_element(By.XPATH, f"//table[@name='BookTable']//tr[{row
    if author == "Mukesh":
        books = driver.find_element(By.XPATH, f"//table[@name='BookTable']//tr[{
        books_ls.append(books)
print("Books Written by Mukesh: ", books_ls)


# table data based on conditions(list of books those price is => 1000)
books_ls = []
for row in range(2, len(table_row)+1):
    price = driver.find_element(By.XPATH, f"//table[@name='BookTable']//tr[{row}
    if int(price) >= 1000:
        books = driver.find_element(By.XPATH, f"//table[@name='BookTable']//tr[{
        books_ls.append(books)
print("Books Price greater then 1000: ", books_ls)

time.sleep(5)
driver.quit()
```

Date Picker



In [268]:
```
1
```

In [250]:
```
1  from selenium import webdriver
2  from selenium.webdriver.common.by import By
3  import time
4
5  testauto = "https://testautomationpractice.blogspot.com/"
6  driver = webdriver.Chrome()
7  driver.maximize_window()
8  driver.get(testauto)
9
10 iframe = driver.find_element(By.ID, "frame-one796456169")
11
12 driver.switch_to.frame(iframe)
13
14 # date element
15 date = driver.find_element(By.ID, "RESULT_TextField-2")
16 print(type(date.text), date.text)
17 print(date.get_attribute("placeholder"))
18
19 time.sleep(5)
20 date.send_keys("09/12/2023")
21 time.sleep(2)
22 print(date.get_attribute("value"))
23 date.clear()
24 time.sleep(2)
25 date.send_keys("11/09/2020")
26 time.sleep(2)
27 print(date.get_attribute("value"))
28 date.clear()
29
30 time.sleep(4)
31 driver.quit()
```

```
<class 'str'>
mm/dd/yyyy
09/12/2023
11/09/2020
```

Mouse Operations

Selenium provides a class called `ActionChains` that allows you to perform various mouse and keyboard actions. Here are some common mouse operations along with examples using Selenium in Python:

1. Click

Perform a left-click on an element:

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

driver = webdriver.Chrome()
driver.get("https://yourwebsite.com")
```

In [ ]:
```
1
```

2. Double Click

Perform a double-click on an element:

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

driver = webdriver.Chrome()
driver.get("https://yourwebsite.com")

element_to_double_click = driver.find_element(By.ID, "elementId")
ActionChains(driver).double_click(element_to_double_click).perform()

driver.quit()
```

In [325]:
```
1  from selenium import webdriver
2  from selenium.webdriver.common.action_chains import ActionChains
3  from selenium.webdriver.common.by import By
4
5  driver = webdriver.Chrome()
6  driver.implicitly_wait(10)
7  driver.maximize_window()
8  driver.get("https://skill-test.net/mouse-double-click")
9
10 act = ActionChains(driver)
11 double_click_it = driver.find_element(By.ID, "clicker")
12
13 act.double_click(double_click_it).perform()
```

3. Right Click (Context Click)

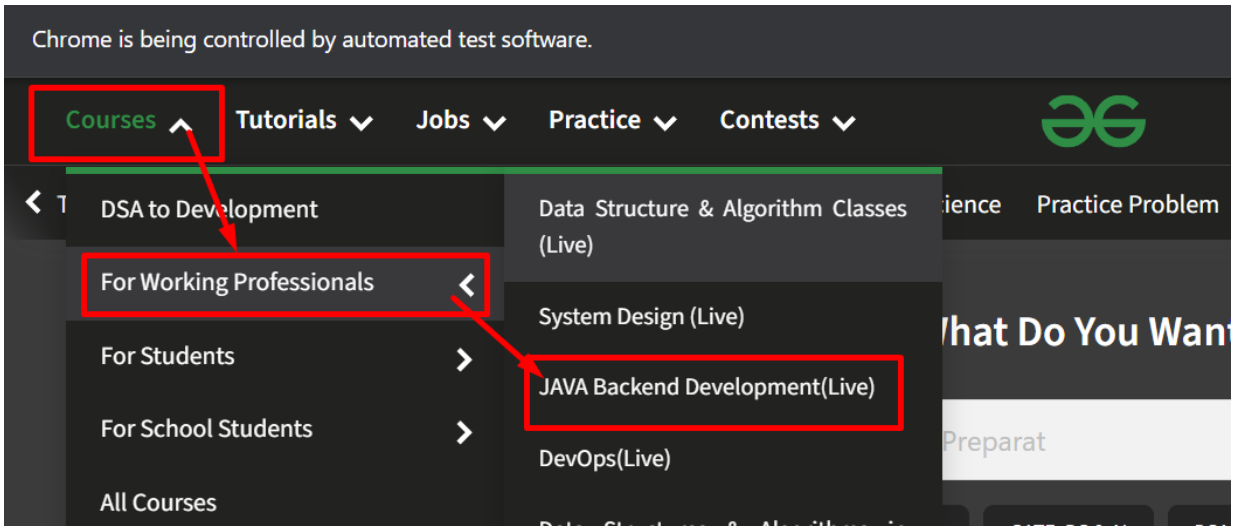
Perform a right-click on an element:

In [302]:

```python
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()
driver.get("https://swisnl.github.io/jQuery-contextMenu/demo.html")

right_click_btn = driver.find_element(By.CLASS_NAME, "btn-neutral")

# Right click action(context click)
act = ActionChains(driver)
act.context_click(right_click_btn).perform()

time.sleep(2)
clk = driver.find_element(By.CLASS_NAME, "context-menu-icon-edit")

# single click
act.click(clk).perform()

time.sleep(3)
driver.quit()
```

4. Hover (Move To Element)

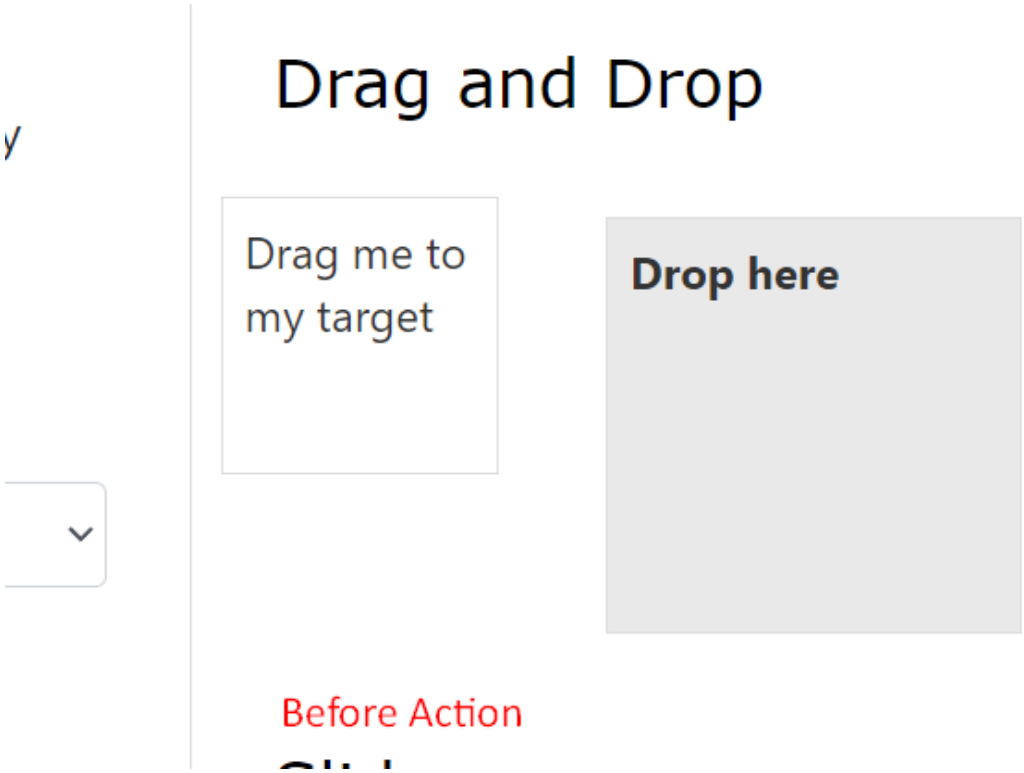
- Move the mouse to a specific element: method used to nevigate `move_to_element()`

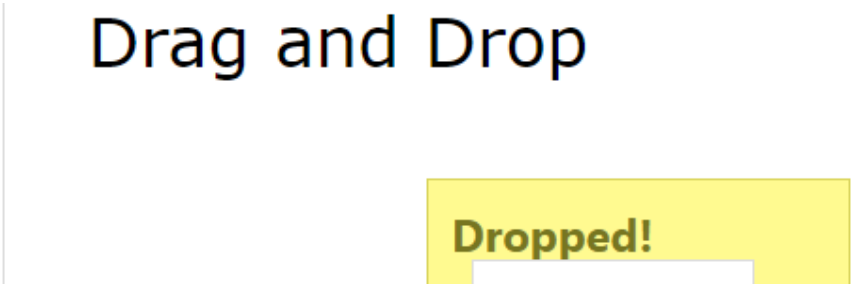


In [323]:

```python
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()
driver.get("https://www.geeksforgeeks.org/")

main_menu = driver.find_element(By.CLASS_NAME, "header-main__list-item")
sub_menu = driver.find_element(By.XPATH, "/html/body/nav/div/div[1]/ul[1]/li[1]/
item = driver.find_element(By.XPATH, "/html/body/nav/div/div[1]/ul[1]/li[1]/ul/l

time.sleep(2) # just to see the action

# Hover actions, 3 moves
act = ActionChains(driver)
act.move_to_element(main_menu).move_to_element(sub_menu).move_to_element(item).c

time.sleep(3)
driver.quit()
```

5. Drag and Drop





In [327]:

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

driver = webdriver.Chrome()

driver.implicitly_wait(10)
driver.maximize_window()

driver.get("https://testautomationpractice.blogspot.com/")

source = driver.find_element(By.ID, "draggable")
target = driver.find_element(By.ID, "droppable")

act = ActionChains(driver)
act.drag_and_drop(source,target).perform()

time.sleep(5)
driver.quit()
```

Slider

In [357]:

```python
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.by import By
import time

driver = webdriver.Chrome()

driver.implicitly_wait(10)
driver.maximize_window()

driver.get("https://testautomationpractice.blogspot.com/")

slider = driver.find_element(By.XPATH, "//div[@id='slider']//span")
print(slider.get_attribute("style"))
print("Location: ", slider.location)
act = ActionChains(driver)

time.sleep(4)
act.drag_and_drop_by_offset(slider,200,0).perform()

slider = driver.find_element(By.XPATH, "//div[@id='slider']//span")
print(slider.get_attribute("style"))
print("Location: ", slider.location)

time.sleep(3)
driver.quit()
```

```
left: 0%;
Location:  {'x': 762, 'y': 1096}
left: 63%;
Location:  {'x': 959, 'y': 1096}
```

Scroll Page

To scroll a page in Selenium using Python, you can use the `execute_script` method to execute JavaScript code that performs the scrolling. Here are a few examples:

1. Scroll Down

In [371]:
```python
from selenium import webdriver
import time
driver = webdriver.Chrome()
driver.maximize_window()
driver.get(heroku)

# Scroll down by a certain number of pixels (e.g., 500)
# driver.execute_script(f"window.scrollBy(0, 500);")

for i in range(0,10):
    driver.execute_script(f"window.scrollBy(0, {10*i});")
    value = driver.execute_script("return window.pageYOffset;")
    print("No of Pixels moved: ", value)
    time.sleep(2)

time.sleep(5)
driver.quit()
```

```
No of Pixels moved:  0
No of Pixels moved:  10
No of Pixels moved:  30
No of Pixels moved:  60
No of Pixels moved:  100
No of Pixels moved:  150
No of Pixels moved:  210
No of Pixels moved:  280
No of Pixels moved:  360
No of Pixels moved:  450
```

In [377]:
```python
# Scroll down to element position
from selenium import webdriver
import time
driver = webdriver.Chrome()
driver.maximize_window()
driver.get("https://www.orangehrm.com/")

time.sleep(3)
target = driver.find_element(By.XPATH, "//div[@class='homepage-product-content '
driver.execute_script("arguments[0].scrollIntoView();", target) # scroll to elem

time.sleep(5)
driver.quit()
```

2. Scroll Up

In [379]:
```python
from selenium import webdriver

driver = webdriver.Chrome()
driver.get(heroku)

time.sleep(3)

driver.execute_script(f"window.scrollBy(0, 500);")
time.sleep(3)
# Scroll up by a certain number of pixels (e.g., -500)
driver.execute_script("window.scrollBy(0, -500);")

time.sleep(5)
driver.quit()
```

3. Scroll to the Bottom

In [381]:
```python
from selenium import webdriver

driver = webdriver.Chrome()
driver.get(heroku)

time.sleep(3)
# Scroll to the bottom of the page
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")

time.sleep(5)
driver.quit()
```

4. Scroll to the Top

In [382]:
```python
from selenium import webdriver

driver = webdriver.Chrome()
driver.get(heroku)

time.sleep(3)
# Scroll to the bottom of the page
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")

time.sleep(3)

# Scroll to the top of the page
driver.execute_script("window.scrollTo(0, 0);")

time.sleep(5)
driver.quit()
```

These examples use the `execute_script` method to execute JavaScript code that manipulates the `window` object. Adjust the scroll values or customize the scrolling behavior based on your specific requirements.

Keep in mind that some websites may use dynamic loading or infinite scrolling, so you might need to

Keyboard Actions

Selenium provides the `Keys` class to simulate keyboard actions. Here are some common keyboard actions you can perform using Selenium in Python:

1. Typing Text

In [384]:

```python
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()
driver.get("https://testautomationpractice.blogspot.com/")

# Locate the input field
input_field = driver.find_element(By.ID, "textarea")

time.sleep(3)
# scrolling to the textarea element
driver.execute_script("arguments[0].scrollIntoView();", input_field)

time.sleep(3)
# Type text into the input field
input_field.send_keys("Hello, Selenium!")

time.sleep(5)
driver.quit()

```

2. Pressing Keys

In [390]:

```python
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.action_chains import ActionChains
import time

driver = webdriver.Chrome()
driver.maximize_window()
driver.implicitly_wait(10)
driver.get(google)
act = ActionChains(driver)

time.sleep(3)
driver.find_element(By.NAME, "q").send_keys("Selenium Python Documentation")
time.sleep(3)

# Press the Enter key
act.send_keys(Keys.ENTER).perform()

time.sleep(3)

# Press Tab key multiple times
for i in range(35):
    if i > 15:
        time.sleep(1)
    act.send_keys(Keys.TAB).perform()
    i += 1

time.sleep(5)
driver.quit()
```

3. Combining Keys

In [397]:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.get(google)
driver.maximize_window()

act = ActionChains(driver)

time.sleep(3)
driver.find_element(By.NAME, "q").send_keys("Selenium Python Documentation")

time.sleep(3)

# Press the Enter key
act.send_keys(Keys.ENTER).perform()

time.sleep(3)

# Combining keys (pressing Ctrl+A to select all text)
act.key_down(Keys.CONTROL).send_keys('a').key_up(Keys.CONTROL).perform()

time.sleep(2)
# Combining keys (pressing Ctrl+C to select all text)
act.key_down(Keys.CONTROL).send_keys('c').key_up(Keys.CONTROL).perform()

time.sleep(2)
driver.get("https://www.rapidtables.com/tools/notepad.html")

time.sleep(2)
text_area = driver.find_element(By.ID, "area")
text_area.send_keys("Written by Selenium: ")
print(text_area.get_attribute("class"))

time.sleep(2)
# Paste : Ctrl + V
act.key_down(Keys.CONTROL).send_keys('v').key_up(Keys.CONTROL).perform()

time.sleep(5)
driver.quit()

```

notes

4. Special Keys

In [394]:

```python
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Chrome()
driver.get(google)
driver.maximize_window()

act = ActionChains(driver)

time.sleep(3)
driver.find_element(By.NAME, "q").send_keys("Selenium Python Documentation")

time.sleep(3)

# Press the Enter key
act.send_keys(Keys.ENTER).perform()

time.sleep(3)

# press Down_Arrow key 5 times
for i in range(5):
    time.sleep(2)
    act.send_keys(Keys.ARROW_DOWN).perform()
    i += 1

    # press Up_Arrow key 5 times
for i in range(5):
    time.sleep(2)
    act.send_keys(Keys.ARROW_UP).perform()
    i += 1

time.sleep(3)
driver.quit()
```

These are just a few examples of keyboard actions you can perform with Selenium. The `Keys` class provides various keys you can use, and you can combine them to create more complex interactions. Adjust the examples based on your specific requirements and the structure of the webpage you are working with.

Download and Upload Files

Downloading and uploading files using Selenium involves interacting with file input elements and system dialogs. Here are examples for both scenarios:

1: Download Files

In [112]:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.chrome.options import Options
import time

opt = Options()
opt.add_argument("--disable-extensions")
opt.add_argument("--start-maximized")
opt.add_argument("--disable-popup-blocking")
opt.add_experimental_option('detach', True)
opt.add_experimental_option('prefs', {"safebrowsing.enabled": 1})

# Add preferences to enable or disable safe browsing
# 0: Safe Browsing disabled, 1: Safe Browsing enabled

driver = webdriver.Chrome(options=opt)
driver.implicitly_wait(10)
driver.get("https://www.sublimetext.com/download")
driver.maximize_window()

act = ActionChains(driver)

time.sleep(3)

download_file = driver.find_element(By.XPATH, "//*[@id='dl_win_64']/a[1]")
download_file.click()
print("Downloading start...")
time.sleep(10)
driver.quit()

```

```
Downloading start...
```

In [113]:

```python
# Detailed video for Download and Upload with many browser settings
from IPython.display import HTML
html_code = """
<iframe width="560" height="315" src="https://www.youtube.com/embed/NA1uXmBV_BQ?
"""
display(HTML(html_code))

```

Upload File

In [116]:

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

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

# Locate the file input element
file_input = driver.find_element(By.ID, "file-input")

# Provide the path to the file you want to upload
file_path = "C:/Selenium/Projects/img_10.png"
file_input.send_keys(file_path)
print("File uploaded")

# Perform other actions as needed
time.sleep(10)
driver.quit()

```

Bootstrap Dropdown


```python
driver.get("https://www.dummyticket.com/dummy-ticket-for-visa-application/")
driver.maximize_window()

driver.find_element(By.XPATH,"//span[@id='select2-billing_country-container']").click()

coutrieslist=driver.find_elements(By.XPATH,"//ul[@id='select2-billing_country-results']/li")
print(len(coutrieslist))

for country in coutrieslist:
    if country.text=="India":
        country.click()
        break
```

Screenshot

Taking screenshots in Selenium is a useful technique for debugging, capturing webpage states, or validating test results. Here's how you can take screenshots using Selenium in Python:

1. Full Screen Screenshot

In [7]:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
import time
# Create a webdriver instance (e.g., Chrome)
driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()

# Navigate to a webpage
driver.get(nopcom)

time.sleep(5)
# Take a screenshot of the entire page
driver.save_screenshot("full_page_screenshot2.png")
print("saved")

# Close the browser
driver.quit()

```

```
saved
```

2. Specific Element Screenshot

In [29]:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
import time
# Create a webdriver instance (e.g., Chrome)
driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()

# Navigate to a webpage
driver.get(nopcom)

time.sleep(5)

# take a screenshot of a specific element
element = driver.find_element(By.XPATH, "/html/body/div[6]/div[3]/div/div/div/di
element.screenshot("element_screenshot.png")

print("Saved")
driver.quit()
```

```
Saved
```

3. Scrolling down and capturing multiple screenshots

In [9]:
```python
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.implicitly_wait(5)
driver.get(nopcom)

# Scroll down the page (adjust the range and step based on your needs)
for scroll in range(0, 1000, 200):
    driver.execute_script(f"window.scrollTo(0, {scroll});")
    time.sleep(1)  # Add a short delay to allow the page to render

    # Take a screenshot at each scroll position
    driver.save_screenshot(f"screenshot_scroll_{scroll}.png")

driver.quit()

```

4. Entire Page Screenshot

In [28]:
```python
# Entire Page Screenshot
import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By


options = Options()
options.add_argument("--headless") # this is must for taking entire page screens
driver = webdriver.Chrome(options=options)
driver.maximize_window()

URL = nopcom
driver.get(URL)

width = 1920
height = driver.execute_script("return Math.max(document.body.scrollHeight, docu
print(width,height)

driver.set_window_size(width,height)
full_page = driver.find_element(By.TAG_NAME, "body")

full_page.screenshot("Full_Page_Screenshot.png")
print("Saved")
time.sleep(3)
driver.quit()
```

```
3883
```

Handle Cookies

Cookies in Web Development:

Cookies are small pieces of data stored on a user's computer by the web browser while browsing a website. They are commonly used to remember user preferences, login sessions, and other information that enhances the user experience. Cookies are sent between the client (browser) and the server with each HTTP request.

Types of Cookies:

1. **Session Cookies:** Temporary cookies that are deleted when the browser is closed.
2. **Persistent Cookies:** Stored on the user's device for a specified period, even after the browser is closed.

Handling Cookies in Selenium:

Selenium provides methods to interact with cookies. Here are some common operations:

1. **Get All Cookies:**

```
# Get all cookies
cookies = driver.get_cookies()
print(cookies)
```

2. **Get a Specific Cookie:**

```
# Get a specific cookie by name
cookie = driver.get_cookie("cookie_name")
print(cookie)
```

3. **Add a Cookie:**

```
# Add a new cookie
new_cookie = {"name": "example_cookie", "value": "example_value"}
driver.add_cookie(new_cookie)
```

4. **Delete a Cookie:**

```
# Delete a specific cookie by name
driver.delete_cookie("cookie_name")
```

5. **Delete All Cookies:**

```
# Delete all cookies
driver.delete_all_cookies()
```

In [94]:

```python
# Get and add cookies
from selenium import webdriver

# Create a webdriver instance (e.g., Chrome)
driver = webdriver.Chrome()

# Navigate to a website
driver.get(nopcom)

# Get All Cookies
cookies = driver.get_cookies()

type(cookies)

# First Cookie details
cookies[1]

for cookie in cookies:
    print(cookie,"\n")

# Get one cookie by cookie name
cookie1 = driver.get_cookie(".Nop.Culture")
print(cookie1)

print("Domain: ", cookie1.get("domain"))

for key in cookie1:
    print(f"{key.ljust(10)} : {cookie[key]}")

print("No of cookies present: ", len(cookies))

# Add a new cookie
driver.add_cookie({"name":"credential","value":"username"})

updated_cookies = driver.get_cookies()
print("No of cookies present: ", len(updated_cookies))

for key in updated_cookies:
    print("Cookie name: ",key['name'])

driver.quit()
```

{'domain': 'demo.nopcommerce.com', 'expiry': 1733801761, 'httpOnly': False, 'name': '.Nop.Culture', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value': 'c%3Den-US%7Cuic%3Den-US'}

{'domain': 'demo.nopcommerce.com', 'httpOnly': True, 'name': '.Nop.Antiforgery', 'path': '/', 'sameSite': 'Strict', 'secure': True, 'value': 'CfDJ8MN0ldDInNtCgdE20r5hSU4dW2lD6jo2DTJQe_Jote_1b7nCucAokEqIQ5j52CTqV2ObC5kkjglyY5NtxKzM1LdDROn1m2yi16hAZw5e1sbUyFK2p6H6nhgNMRw7Ky1Bu3F9hiN5sp_0biqzsgGaAT4'}

{'domain': 'demo.nopcommerce.com', 'expiry': 1733715361, 'httpOnly': True, 'name': '.Nop.Customer', 'path': '/', 'sameSite': 'Lax', 'secure': True, 'value': 'ddc5363f-33d5-4fdf-aa09-6fb69cd4adba'}

{'domain': 'demo.nopcommerce.com', 'expiry': 1733801761, 'httpOnly': False, 'name': '.Nop.Culture', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value': 'c%3Den-US%7Cuic%3Den-US'}
Domain: demo.nopcommerce.com
domain   : demo.nopcommerce.com
expiry   : 1733715361
httpOnly   : True
name   : .Nop.Customer
path   : /
sameSite   : Lax
secure   : True
value   : ddc5363f-33d5-4fdf-aa09-6fb69cd4adba
No of cookies present: 3
No of cookies present: 4
Cookie name: .Nop.Culture
Cookie name: credential
Cookie name: .Nop.Antiforgery
Cookie name: .Nop.Customer

In [109]:

```python
# Delete cookies
from selenium import webdriver

driver = webdriver.Chrome()
driver.get(nopcom)

# Get All Cookies
cookies = driver.get_cookies()
print("Total No of Cookies", len(cookies))
for i in cookies:
    print(i, "\n")


# delete a cookie by name
cookie_name = '.Nop.Customer'
driver.delete_cookie(cookie_name)
print(f"Cookie {cookie_name} deleted")

print("Updated  Cookies: ", len(driver.get_cookies()))

driver.delete_all_cookies()
print("Delete all cookies")

print("Updated  Cookies: ", len(driver.get_cookies()))

driver.quit()
```

```
Total No of Cookies 3
{'domain': 'demo.nopcommerce.com', 'expiry': 1733802209, 'httpOnly': False, 'name':
'.Nop.Culture', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value': 'c%3Den-U
S%7Cuic%3Den-US'}

{'domain': 'demo.nopcommerce.com', 'httpOnly': True, 'name': '.Nop.Antiforgery', 'p
ath': '/', 'sameSite': 'Strict', 'secure': True, 'value': 'CfDJ8MN0ldDInNtCgdE20r5h
SU6t_4OW9p2PLJ2wBfn9epQP_GshL275b4Bi55KxoOuyVZXZ5yHtkRva49Mc1TvkIcCor1QplnTjls2xPw0
sPuNGWTBor1nAFDB9dCuyE5gallZoiF_QprBgfwv6w2Gezjs'}

{'domain': 'demo.nopcommerce.com', 'expiry': 1733715809, 'httpOnly': True, 'name':
'.Nop.Customer', 'path': '/', 'sameSite': 'Lax', 'secure': True, 'value': '9d4e7658
-bd97-49bc-bc3f-2b63abb96e37'}

Cookie .Nop.Customer deleted
Updated  Cookies:  2
Delete all cookies
Updated  Cookies:  0
```

In [3]:

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

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.maximize_window()
driver.get("https://www.linkedin.com/in/sivakumarbabujiofficial/recent-activity/

email = driver.find_element(By.ID, "email-or-phone")
password = driver.find_element(By.ID, "password")
email.send_keys("robinhud299@gmail.com")
password.send_keys("Santy#9887")

driver.find_element(By.ID, "join-form-submit").click()

```

In [ ]:

```

```