

# Topics

Tuesday, November 5, 2024

11:14 PM

## 1. About Selenium

## 2. Getting Started

## 3. Navigating Web Pages

- *Locating Web Elements*
- *Understanding Xpath*
- *Interacting with Web Elements*
- *Dropdown & Multiselect*
- *Scrolling*

## 4. Advanced Web Interaction

- *Explicit Waits*
- *Implicit Waits*
- *Handling Alerts*
- *Frames & iFrames*

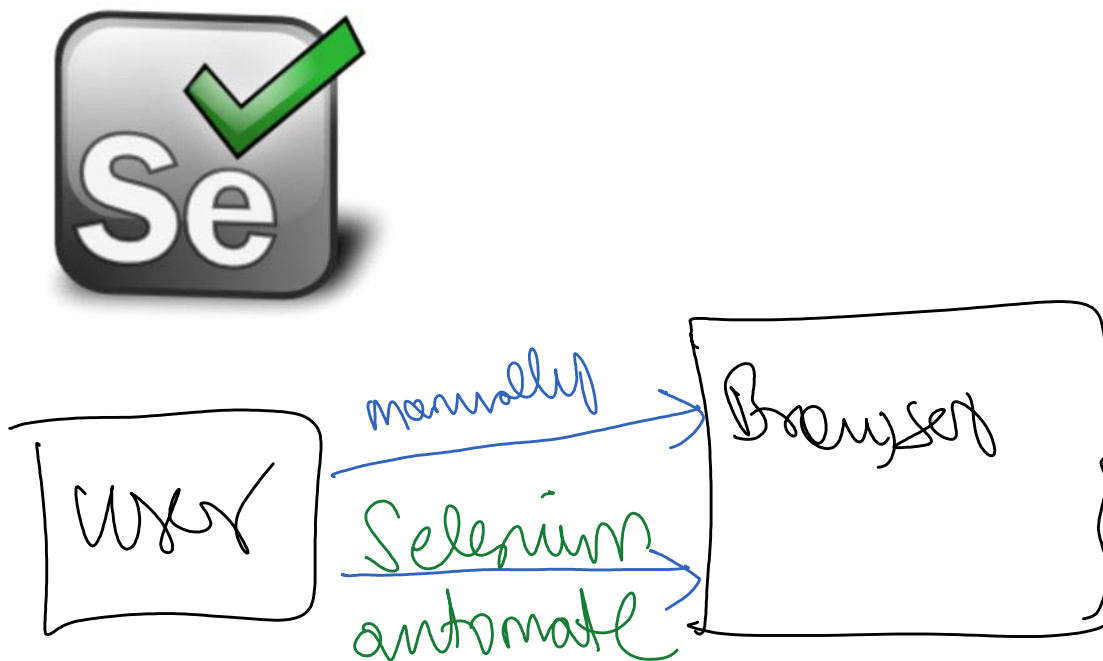
## 5. Best Practices & Optimization

# 1. Definition

Tuesday, November 5, 2024 11:14 PM

## What is Selenium?

- Selenium is a powerful, open-source tool used for automating web browsers.
- It is often utilized for web scraping when interacting with dynamic websites that rely on JavaScript to load content, which static scraping libraries like BeautifulSoup or Requests cannot handle effectively.
- When scraping websites using Python, Selenium acts as a web driver, automating browser actions to interact with web pages like a human user.
- It can navigate to web pages, simulate user interactions (clicks, scrolls, form submissions), and extract data directly from rendered HTML.
- Selenium was originally developed for testing web applications. Over time, it became a popular tool for web scraping due to its ability to handle dynamic, JavaScript-heavy websites.

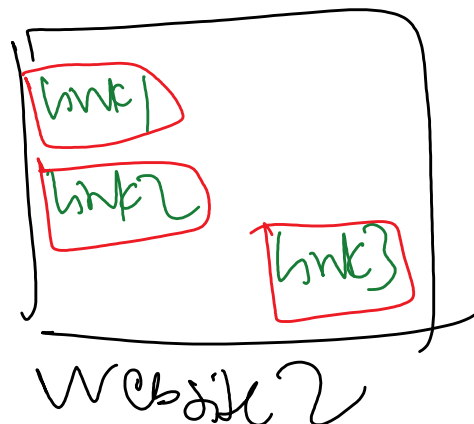
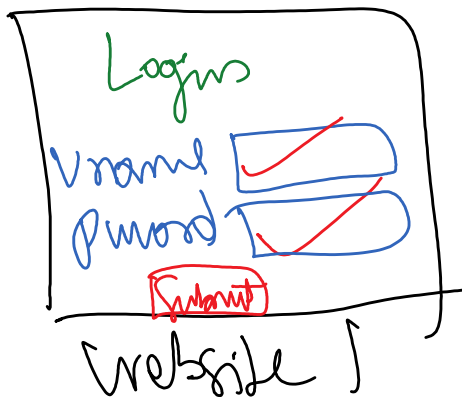


## 2. Key Features

Monday, December 2, 2024 11:41 PM

### Key Features:

- **Dynamic Content Handling:**
  - Can scrape data from JavaScript-heavy sites.
  - Waits for elements to load before interacting or extracting data.
- **Interaction Simulation:**
  - Handles tasks such as clicking buttons, filling forms, selecting dropdowns, and scrolling pages.
  - Useful for scraping data hidden behind user interactions.
- **Cross-Browser Support:**
  - Works with popular browsers like Chrome, Firefox, Edge, and Safari.
- **Customizable Waits:**
  - Implements explicit and implicit waits to ensure elements are fully loaded before actions are performed.



### 3. Comparison

Monday, December 2, 2024 11:41 PM

Comparative Analysis:

FEATURE	SELENIUM	BEAUTIFUL SOUP	REQUESTS
Handles JavaScript	✓	✗	✗
Ease of Use	Moderate	Easy	Easier
Speed	Slower (browser overhead)	Faster	Fastest
Interaction with Elements	✓	✗	✗

## 4. Advantages

Monday, December 2, 2024

11:42 PM

### Advantages:

- Handles JavaScript and AJAX (*Asynchronous JavaScript and XML*)
- Simulates user behavior
- Allows scraping data behind login screens or requiring user interaction

## 5. Disadvantages

Monday, December 2, 2024 11:42 PM

### Disadvantages:

- Slower than static scraping methods since it requires a full browser environment
- Heavy CPU and memory usage
- Websites may block or detect Selenium bots

# 1. First Steps

Monday, December 2, 2024

11:47 PM

1. Activate virtual environment
2. Install selenium
3. Download suitable web driver

## 2. Exercise

Tuesday, December 3, 2024

12:01 AM

### Write a Python script to:

- Initialize a web driver
- Open Google's home page
- Get the below details:
  - *Title of webpage*
  - *URL of webpage*
- Take a screenshot of the page
- Close the browser



# 1. Locating Elements

Tuesday, December 3, 2024 12:35 AM

- Selenium provides the function `find_element` to find and locate elements of a webpage
- There're several strategies to identify an element of a webpage:

## 1. ID:

```
element = driver.find_element("id", "<element_id>")
```

## 2. Name:

```
element = driver.find_element("name", "<element_name>")
```

## 3. Class:

```
element = driver.find_element("class name", "<element class id>")
```

## 4. Tag:

```
element = driver.find_element("tag name", "<element tag>")
```

## 5. XPath:

```
element = driver.find_element("xpath", "<element xpath link>")
```

### Element Location Strategies:

Locator Method	Locator Class	Description
id	By.ID	Finds an element by ID.
name	By.NAME	Finds an element by name attribute.
class name	By.CLASS_NAME	Finds an element by CSS class name.
tag name	By.TAG_NAME	Finds elements by HTML tag.

Locator Method	Locator Class	Description
id	By.ID	Finds an element by ID.
name	By.NAME	Finds an element by name attribute.
class name	By.CLASS_NAME	Finds an element by CSS class name.
tag name	By.TAG_NAME	Finds elements by HTML tag.
css selector	By.CSS_SELECTOR	Finds elements using CSS selectors.
link text	By.LINK_TEXT	Finds links by their exact text.
partial link text	By.PARTIAL_LINK_TEXT	Finds links by partial match of their text.
xpath	By.XPATH	Finds elements using XPath expressions.

```
from selenium.webdriver.common.by import By  
element = driver.find_element(By.ID, "<element_id>")
```

## 2. XPath

Tuesday, December 3, 2024 12:58 AM

### What is XPath?

- XPath (XML Path Language) is a query language used to navigate and locate elements within XML or HTML documents.
- Selenium uses XPath as one of its locator strategies to find elements on a webpage.
- XPath is a powerful tool for locating elements in Selenium, offering unmatched flexibility and precision.
- It's a go-to solution when working with complex web pages or when other locators are insufficient.

### Advantages:

- **Locate Elements Anywhere:**
  - XPath can traverse the entire DOM, allowing you to locate elements in deep nested structures or without unique identifiers.
  - Works for all elements, even those without `id`, `name`, or `class`.
- **Offers Rich Syntax:** XPath supports a variety of conditions and operators, enabling users to
  - Match elements by attributes
  - Locate elements by position
  - Use partial matches
- **Supports Text-Based Matching:**
  - Locate elements based on visible text
- **Supports Relative Paths:** You can locate elements without specifying their full path in the DOM, making XPath expressions robust to changes
  - Relative: `//div[@class='example']`
  - Absolute: `/html/body/div[1]/div[2]`

- **Navigate the DOM Hierarchy**

```
//div[@id='container']/child::p # Direct child
//span[@class='item']/ancestor::div # Ancestor
//div[@id='content']/following-sibling::div # Sibling
```

- **Independent of HTML Structure**
  - XPath can navigate through the DOM and locate elements that might not be directly visible or styled.
  - Elements in the DOM can exist even if they are hidden from the user's view, such as elements with CSS properties like `display: none` or `visibility: hidden`.

### Disadvantages:

- XPath is slower than CSS Selectors because of its ability to traverse the entire DOM.
- XPath expressions can be harder to read and maintain, especially for deeply nested elements.
- Some older browsers may have limited support for advanced XPath queries.



## 3. Interacting with Elements

Tuesday, December 3, 2024 1:16 AM

### 1. Typing Input into Fields:

- This is achieved using the `send_keys` function
- It's used to simulate typing into an element, such as a text input field or a text area.
- It allows users to send keystrokes or input text programmatically as if a user were typing on a keyboard.
- Enables simulation of key presses like Enter, Tab, etc.
  - *This is achieved by using the class `Keys` from the module `selenium.webdriver.common.keys`*

### 2. Clearing Input Fields:

- This is achieved using the `clear` function
- It's used to clear the text content of a text input element on a web page
- It ensures the field is empty before entering new data (which can be done using the `send_keys` function)

### 3. Clicking Buttons:

- This is achieved using the `click` function
- It's used to simulate a mouse click on a web element
- Allows users to interact with clickable elements on a web page, such as buttons, links, checkboxes, or radio buttons.

### 4. Submitting Forms:

- This is achieved using the `submit` function
- Helps to automatically submit a form without explicitly clicking the "Submit" button
- Executes the action associated with the `<form>` tag, such as navigating to a new page or processing data.
- Direct use of `submit` is not common in modern web development:
  - *Most forms today rely on JavaScript events or custom logic*
  - *For such forms, using the `click` function on a "Submit" button or JavaScript execution may be more reliable.*



## 4. Dropdown & Multiselect

Tuesday, December 3, 2024 10:29 PM

### 1. Dropdown:

- Need to identify the dropdown element as usual
- Wrap the identified element under the class `Select`, imported from the module `selenium.webdriver.support.select`
- There are 3 ways to select an option from the dropdown list:
  - **select\_by\_index**: provide the index of the option
  - **select\_by\_value**: provide the name attribute of the option
  - **select\_by\_visible\_text**: provide the actual text value of the option

### 2. Multiselect:

- Similar to dropdown as discussed above
- There are multiple ways to select an option:
  - **select\_by\_index**: provide the index of the option
  - **select\_by\_value**: provide the name attribute of the option
  - **select\_by\_visible\_text**: provide the actual text value of the option
- Similarly, there are multiple ways to deselect an option:
  - **deselect\_by\_index**: provide the index of the option
  - **deselect\_by\_value**: provide the name attribute of the option
  - **deselect\_by\_visible\_text**: provide the actual text value of the option
  - **deselect\_all**: takes no arguments

## 5. Scrolling

Thursday, December 5, 2024 12:19 AM

- There're several ways of scrolling a webpage using Selenium:
  - *Scrolling to a specific element*
  - *Scrolling vertically*
  - *Scrolling horizontally*
  - *Scrolling the page height*
  - *Infinite scrolling*
- Scrolling actions are mainly achieved using the `execute_script` method


```
driver.execute_script(script, *args)
```

- This method is mainly used to execute JavaScript code within the context of the currently loaded webpage
- It allows users to directly interact with and manipulate the Document Object Model (DOM) of the page
- Helps interact with elements that might not be accessible using Selenium's standard methods
- To better handle dynamically loaded content on modern and JavaScript-heavy websites
  - **script**: String containing JavaScript code
  - **args**: Optional arguments to pass to the JavaScript code, usually Web Elements or other variables

### 1. Scrolling to a specific element:

- Identify any element on the webpage
- Use the method `scrollIntoView`

```
element = driver.find_element("id", "specificElementId")  
driver.execute_script("arguments[0].scrollIntoView();", element)
```



### 2. Scrolling Vertically:

- Use the method `scrollBy`
- Specify the number pixels to scroll by
- +ve value indicates scrolling down
- -ve value indicates scrolling up

```
driver.execute_script("window.scrollBy(0, 500);")
```

### 3. Scrolling Horizontally:



- Similar to scrolling vertically
- The no. of pixels are provided as the first argument
- +ve value indicates scrolling to the right

```
driver.execute_script("window.scrollBy(500, 0);")
```

#### 4. Scrolling to Page Height:

- Use the `scrollTo` method
- Pass the value `document.body.scrollHeight` in place of pixels as usual
- It refers to total height of the content in a webpage

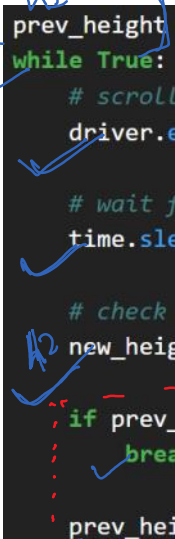
```
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
```

#### 5. Infinite Scrolling:

- Initially, the webpage loads a fixed amount of content.
- As the user scrolls close to the bottom of the page, a JavaScript function triggers a request to load more content dynamically
- The new content is added to the page, and the process repeats.

##### Algorithm:

- Get the height of the currently loaded page (h1)
- Run an infinite loop
- Scroll down the page to h1
- Inside the loop, get the height of the page again (h2)
- If h1 is same as h2, break out of the loop as no new content has been loaded
- If h1 is not same as h2, update h1 as h2 and continue the loop



```
prev_height = driver.execute_script('return document.body.scrollHeight')
while True:
    # scroll to page bottom
    driver.execute_script('window.scrollTo(0, document.body.scrollHeight);')

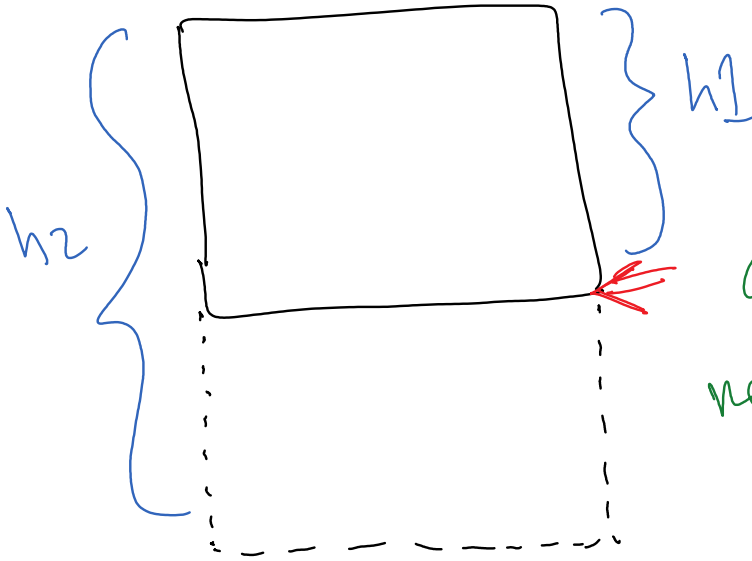
    # wait for content to load
    time.sleep(5)

    # check if content was loaded and page height changed
    new_height = driver.execute_script('return document.body.scrollHeight')

    if prev_height == new_height:
        break

    prev_height = new_height
```

```
break
prev_height = new_height
```



$curr\_height = h1$   
 $new\_height = h2$

$curr\_height == new\_height$   
true  
break  
false  
 $curr\_height = new\_height$



# 1. Intro

Thursday, November 7, 2024 10:37 PM

- This section covers advanced web interactions that go beyond basic navigation and element manipulation.
- By mastering these techniques, developers will be able to handle real-world web scraping and automation challenges, including interacting with dynamic content, handling alerts, and managing iframes.

## Topics:

- *Explicit Waits*
- *Implicit Waits*
- *Handling Alerts*
- *Frames & iFrames*

## 2. Explicit Waits

Friday, December 6, 2024 1:38 AM

### What is Explicit Wait?

- Type of wait that pauses the execution of the script until a specific condition is met or a specified maximum time is reached.
- Useful when dealing with dynamic web elements that take time to appear or become interactable on the page.
- Helps avoid exceptions such as `NoSuchElementException` or `ElementNotInteractableException`
- Improves script reliability by waiting only as long as necessary
- Optimizes test execution time compared to fixed delays (e.g., `time.sleep()`)

### How to Implement?

- Selenium provides the `WebDriverWait` class to implement explicit waits
- It works with expected conditions defined in the `selenium.webdriver.support.expected_conditions` module
- The script checks for the condition at regular intervals (default - 500ms) until it's met or the timeout occurs

### Syntax:

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

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

#### • Parameters:

- ✓ **driver:** The WebDriver instance controlling the browser
  - ✓ **timeout:** Maximum time (in seconds) to wait for the condition to be met
  - ✓ **poll\_frequency:** How often (in seconds) the condition is checked (default: 0.5 seconds)
  - **ignored\_exceptions:** A tuple of exceptions to ignore while waiting (optional)
- `WebDriverWait` often works in conjunction with `Expected Conditions` (EC) to define what to wait for:

CONDITION	DESCRIPTION
<code>presence_of_element_located</code>	Waits for the element to be present in the DOM
<code>visibility_of_element_located</code>	Waits for the element to be visible on the page
<code>element_to_be_clickable</code>	Waits for the element to be visible and enabled (clickable)
<code>text_to_be_present_in_element</code>	Waits for specific text to appear inside an element
<code>alert_is_present</code>	Waits for a browser alert to appear
<code>title_is</code>	Waits for the page title to exactly match a given value
<code>title_contains</code>	Waits for the page title to contain specific text



### 3. Implicit Waits

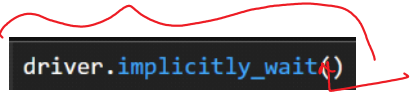
Friday, December 6, 2024 1:38 AM

#### What is Implicit Wait?

- Implicit Waits are a mechanism in Selenium to specify a default waiting time for the WebDriver when searching for elements on a webpage.
- If an element is not immediately found, the WebDriver waits for the specified duration before throwing a `NoSuchElementException`.
- This type of wait applies globally to all element searches in the WebDriver instance.

#### How It Works:

- We can set up the waiting duration using the `implicitly_wait` method of the webdriver instance.
- Makes Selenium scripts resilient to minor delays in the loading of web elements caused by network speed, animations, or dynamic content.
- Once set, it applies to all `find_element` and `find_elements` methods for the lifetime of the WebDriver instance.
- If the element is found before the timeout period, the script proceeds immediately. Otherwise, it waits until the timeout is reached and raises an exception if the element is still not found.



```
driver.implicitly_wait()
```

```
Signature: driver.implicitly_wait(time_to_wait: float) -> None
Docstring:
Sets a sticky timeout to implicitly wait for an element to be found,
or a command to complete. This method only needs to be called one time
per session. To set the timeout for calls to execute_async_script, see
set_script_timeout.

:Args:
- time_to_wait: Amount of time to wait (in seconds)
```

#### Advantages:

- **Simplicity:** Easy to implement and applies globally, avoiding repetitive waits for every element.
- **Resilience:** Handles minor delays in loading dynamically generated elements, reducing script failures.
- **Better Control:** Provides a default buffer for all element searches without the need for explicit handling.

#### Disadvantages:

- Since it applies globally, it may not suit situations where different elements require different wait times.
- Mixing implicit waits with explicit waits can cause unpredictable behavior, as implicit waits can interfere with explicit wait polling mechanisms.
- Implicit waits only handle element visibility or presence and cannot wait for specific conditions like page titles or JavaScript execution.

#### Best Practices:

- Use either implicit waits or explicit waits in your script, but not both, to avoid conflicts.
- Set reasonable timeout durations and not very high implicit wait times (e.g., 60 seconds) as it can unnecessarily delay script execution.
- Implicit waits are suitable for simple scripts without complex wait conditions.

### Implicit vs Explicit Wait:

FEATURE	IMPLICIT WAIT	EXPLICIT WAIT
Scope	Applies globally to all element searches	Applies to specific elements or conditions
Flexibility	Cannot target specific conditions or elements	Can wait for custom conditions like visibility or clickability
Ease of Use	Easier to implement for simple scenarios	Requires additional code for condition handling
Performance	May introduce unnecessary delays globally	More efficient as it targets specific waits
Application	Better for static pages where elements are predictable	Better for more dynamic and complex websites where elements can be unpredictable



## 4. Frames & IFrames

Friday, December 6, 2024 1:38 AM

### What is a Frame/IFrame?

- In web development, **frames** and **iframes** are HTML elements that allow you to embed one HTML document inside another.

### Frame:

- Part of the `<frame>` and `<frameset>` HTML tags, which were used in early web development to divide the browser window into multiple sections, each capable of loading a separate HTML document.
- Now obsolete in HTML5, frames are rarely used. They were replaced by iframes and other modern layout techniques like CSS Grid and Flexbox.

### Iframe (Inline Frame):

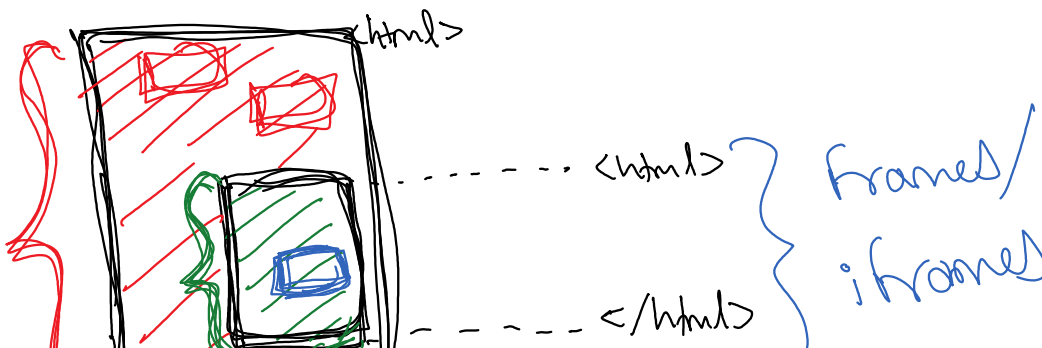
- An `<iframe>` is an HTML element that embeds another HTML document within the current page.
- Changes in the parent page (like CSS or JavaScript) typically do not affect the iframe's content, and vice versa.
- Each iframe has its own DOM (Document Object Model), CSS, and JavaScript scope.
- Interaction between the parent and iframe is restricted if they originate from different domains for security reasons.

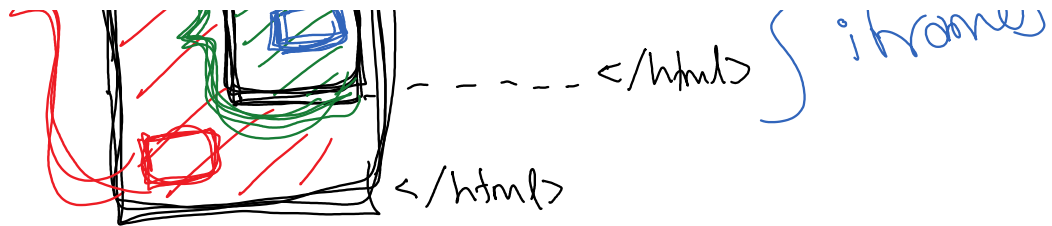
### Working with Selenium:

- To interact with iframe content in Selenium, users must explicitly switch the context to the iframe using `driver.switch_to.frame()`
- Frame content can be identified in many different ways:
  - *ID*
  - *Name*
  - *Index*
  - *Xpath*
  - *CSS Selector*
- After interacting with an iframe, switch back to the parent page using `driver.switch_to.default_content()`

### Best Practices:

- Ensure you know which frame or iframe contains the desired elements by inspecting the page source.
- Whenever possible, avoid switching by index to maintain flexibility if the page structure changes.
- Always switch back to the main content after interacting with a frame.





## 5. Handling Alerts

Friday, December 6, 2024 1:38 AM

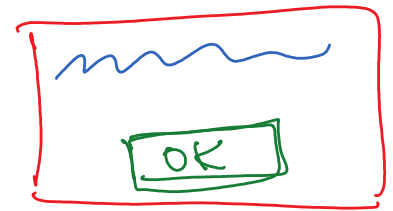
### What are Alerts:

- Alerts in web refer to small, temporary messages or pop-ups that appear in a web browser to communicate information or request user actions.
- They are typically generated by JavaScript or built into the HTML/CSS structure of a webpage.
- Alerts are used for various purposes, including notifying users, obtaining confirmation, or prompting for input.

### Types of Alerts:

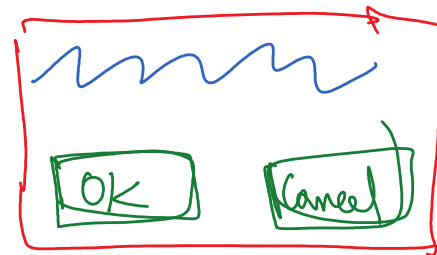
#### 1. JavaScript alerts:

- Created using JavaScript's `alert()` function.
- Displays a simple message to the user with a single "OK" button.
- Blocks user interaction with the page until dismissed.



#### 2. Confirmation Alerts:

- Created using JavaScript's `confirm()` function.
- Asks the user to confirm an action with "OK" and "Cancel" buttons.
- Returns true for "OK" and false for "Cancel"



#### 3. Prompt Alerts:

- Created using JavaScript's `prompt()` function.
- Requests user input and provides an input field along with "OK" and "Cancel" buttons.
- Returns the input value for "OK" or null for "Cancel".

#### 4. Browser-Based Alerts (Authentication Pop-Ups):

- Appear when a website requests HTTP basic authentication.
- Requires entering a username and password

#### 5. Custom HTML Alerts (Modal Dialogs):

- Designed using HTML, CSS, and JavaScript to create custom alert-style messages.
- Offers more flexibility in design and functionality (e.g., styled dialog boxes with multiple buttons or inputs).

## Handling Alerts with Selenium:

- To interact with alerts in Selenium, users must explicitly switch the context to the alert box using `driver.switch_to.alert`
- Use the `accept()` method to click the "OK" button
- Use the `dismiss()` method to click the "Cancel" button on confirmation pop-ups
- Use the `text` attribute to retrieve the message displayed on the alert
- Use the `send_keys()` method to input text into a prompt pop-up

# 1. Intro

Saturday, December 7, 2024 10:39 AM

- This section provides strategies for writing maintainable, efficient, and robust Selenium scripts.
- Adhering to these best practices will improve the performance of test automation or web scraping projects and make the code easier to debug and maintain.

## 2. Write Maintainable Code

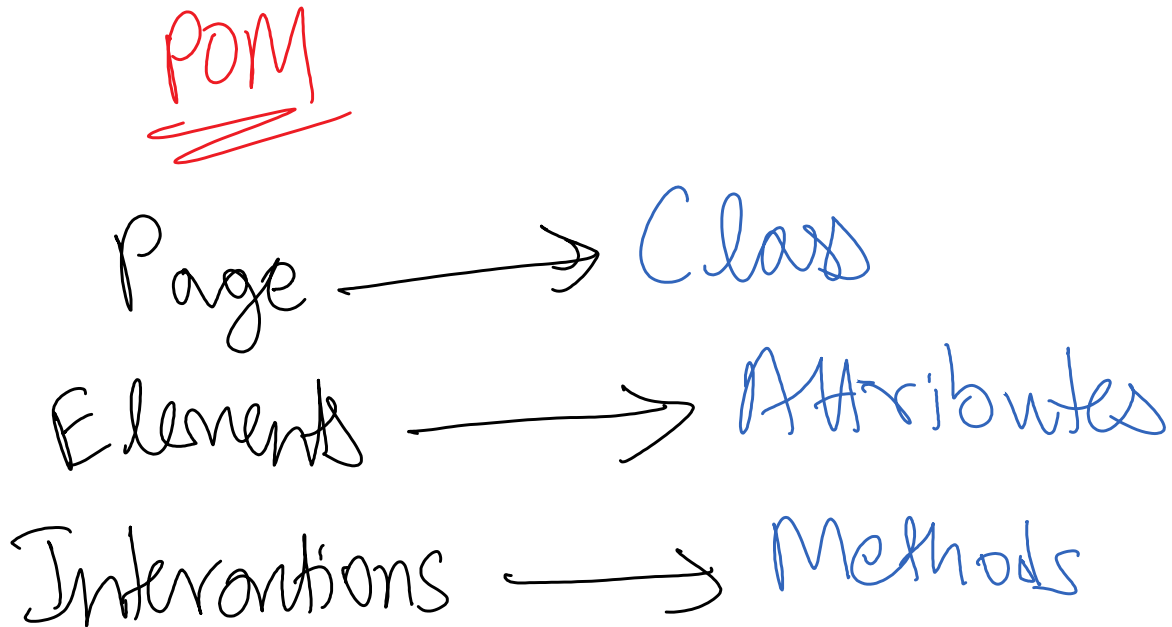
Saturday, December 7, 2024 10:41 AM

### 1. Page Object Model (POM):

- The Page Object Model is a design pattern that separates the code used to locate and interact with web elements.
- This improves code readability and reusability
- Each webpage is represented by a class
- Elements are defined as variables, and interactions (methods) are encapsulated within the class

### 2. Variable Names:

- Avoid overly generic names (ex: element1, button1)
- Use descriptive names to define web elements



## 3. Enhance Performance

Saturday, December 7, 2024 11:29 AM

### 1. Using Appropriate Waits:

- Overuse of `time.sleep()` can slow down scripts unnecessarily
- Look to use Explicit Waits and Implicit Waits for better performance
- Use Explicit Waits for better control

### 2. Optimize Locator Strategies:

- Use ID and NAME where possible as they are the fastest locators
- Avoid using XPath unless necessary, as it can be slower

### 3. Reuse Browser Sessions:

- Instead of launching a new browser for each test, reuse the browser session if applicable
- For scraping, minimize browser interaction by using headless mode

```
from selenium.webdriver.chrome.options import Options

options = Options()
options.add_argument("--headless")
driver = webdriver.Chrome(options=options)
```

## 4. Robustness and Error Handling

Saturday, December 7, 2024

11:38 AM

### 1. Try-Except blocks:

- Wrap code for critical interactions within try-except blocks to manage unexpected failures
- Catch specific exceptions if possible

```
try:  
    element = driver.find_element(By.ID, "non_existent_id")  
except NoSuchElementException:  
    print("Element not found")
```

### 2. Release Resources:

- Always close the browser session at the end of the script to free resources
- This can be achieved using `driver.quit()`



## 5. Logging and Debugging

Saturday, December 7, 2024 11:43 AM

### 1. Logging:

- Avoid printing directly to the console to debug; use logs instead
- Use python's `logging` module for better control

```
import logging

logging.basicConfig(level=logging.INFO)
logging.info("Test started")
```

### 2. Capture Screenshots:

- Save a screenshot for debugging If code fails
- Use the `save_screenshot` method of the webdriver instance

### 3. Developer Tools:

- Use the browser's Developer Tools to inspect element locators and understand dynamic content of the webpage
- Gives a better understanding of the website structure and its respective code

## 6. Security Considerations

Saturday, December 7, 2024

11:49 AM

### 1. Secure Sensitive Data:

- Avoid exposing credentials in scripts
- Use encrypted files or environment variables for storage

### 2. Respect Website Policies:

- Check the website's `robots.txt` and Terms of Service before scraping
- Be ethical in your automation and scraping practices