Web Automation Selenium 4.x
Notes - TheTestingAcademy (Pramod Sir)

📗 Mastering Web Automation with Selenium

IDE

1. Pycharm
2. Visual Studio Code

About Selenium

- Selenium Automates Web Browsers.

What is Selenium?

Selenium is an open-source suite.

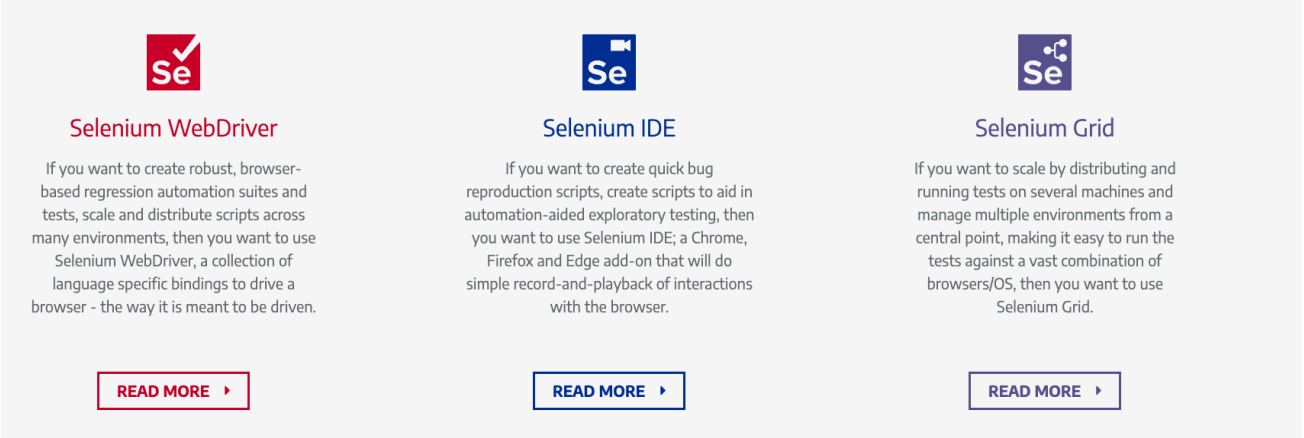
Birth of WebDriver

Simon Stewart created WebDriver circa 2006 when browsers and web applications were becoming more powerful and more restrictive with JavaScript programs like Selenium Core. It was the first cross-platform testing framework that could control the browser from the OS level.

which can automate browsers

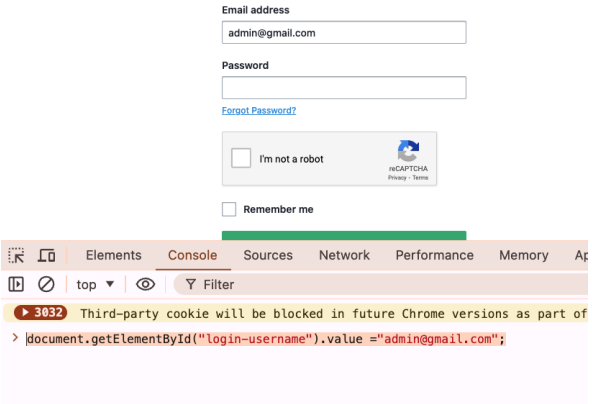
Selenium automates browsers. That's it!

What you do with that power is entirely up to you.

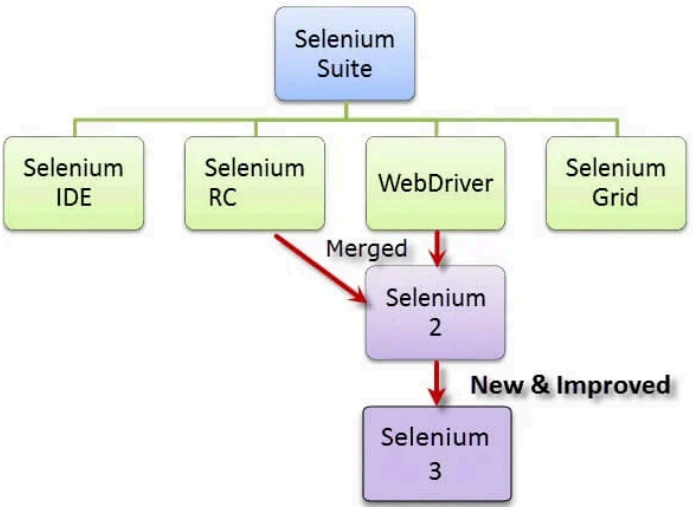


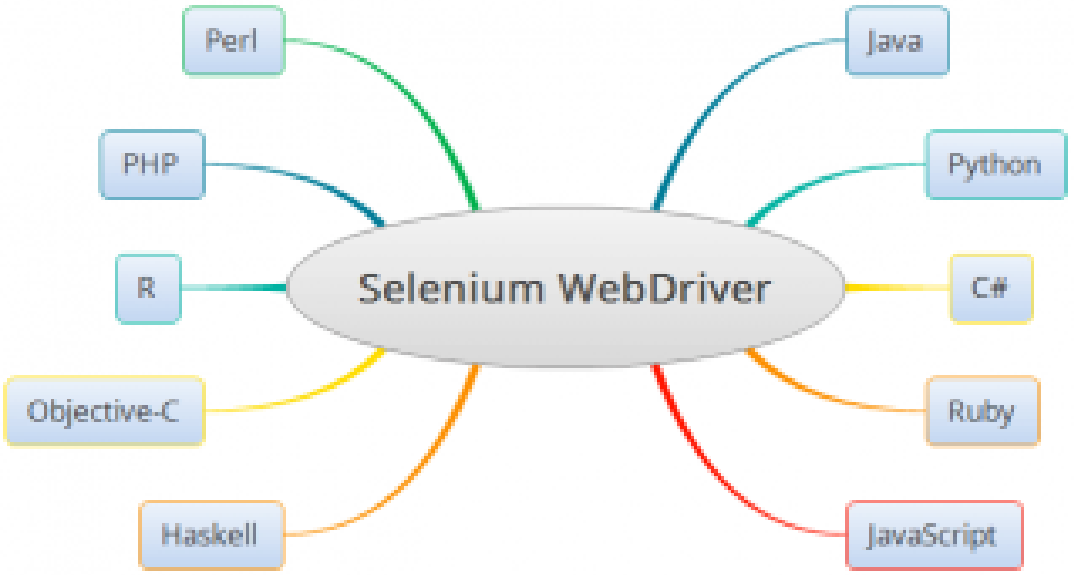
[History -]

Selenium RC executed tests by injecting JavaScript code into the web browser being automated. RC deprecated.



Webdriver - Find the elements,





Selenium vs Playwright vs Cypress

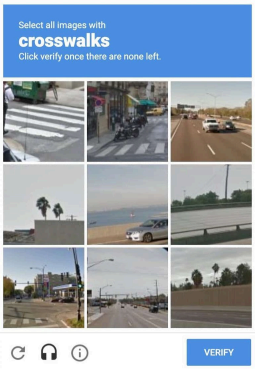
Compare Results - https://blog.checklyhq.com/cypress-vs-selenium-vs-playwright-vs-puppeteer-speed-comparison/

Requirement/Tool	Selenium	Cypress	Playwright
Established solution	Yes	Yes	Yes
Has own runner/reporting	No	Yes	Yes
Compatible with other runners (Junit, Jest, ...)	Yes	No	Yes
Cross Domain Support	Yes	No	Yes
Multi tab support	Yes	No	Yes
Performance	Good	Good	Best
Drag and Drop support	Yes	Yes	Yes
Dynamic waits	No	Yes	Yes
Static waits	Yes	Yes	No
Element selector support	css, xpath, text	css, xpath, text	css, xpath, text
Parallel test execution	Yes	Yes (pro version or Sorry Cypress project)	Yes
Video recording, screenshots	Yes (videos only with 3rd party tools)	Yes	Yes
Possibility to use as AWS Lambda	Yes	Yes	Yes
Support for Java, JavaScript and Python	Yes	No	Yes

Don’t use Selenium here

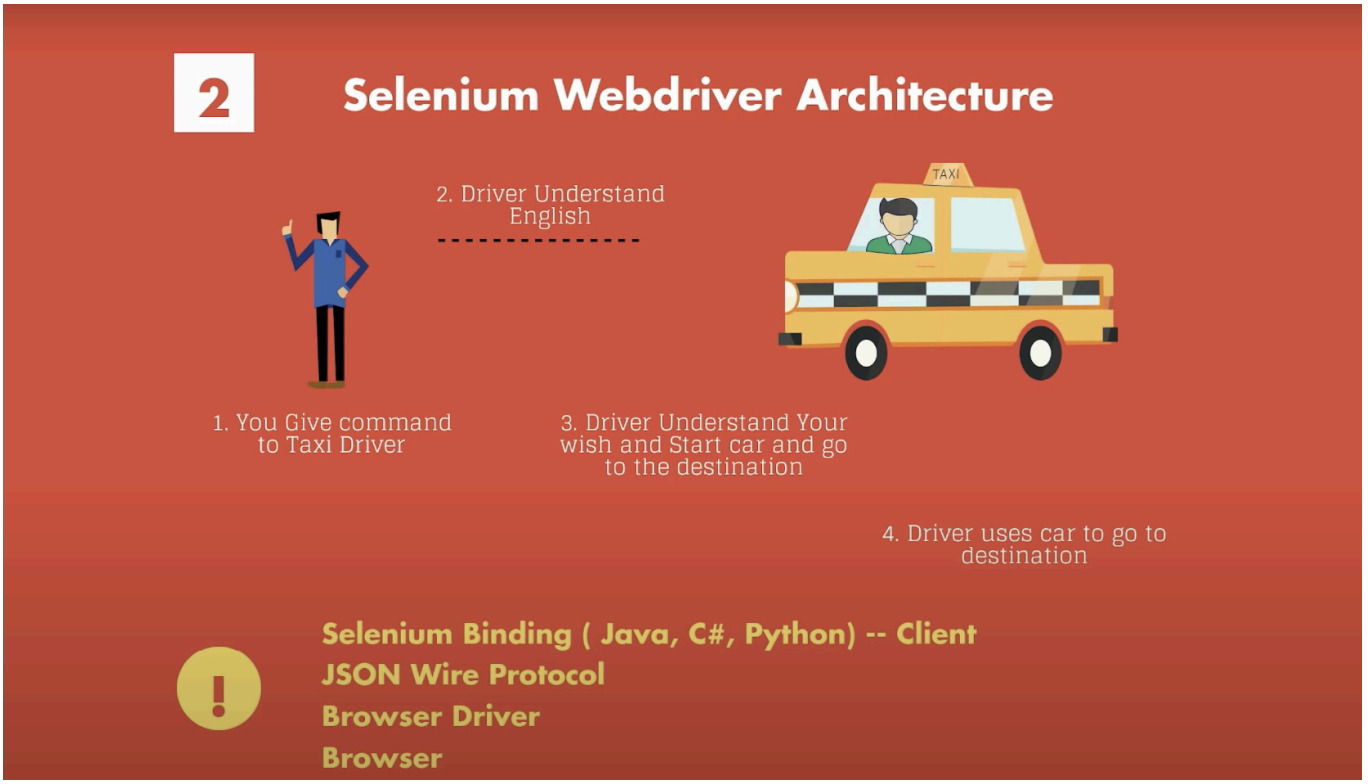
Few situations where you might not want to use Selenium for testing.

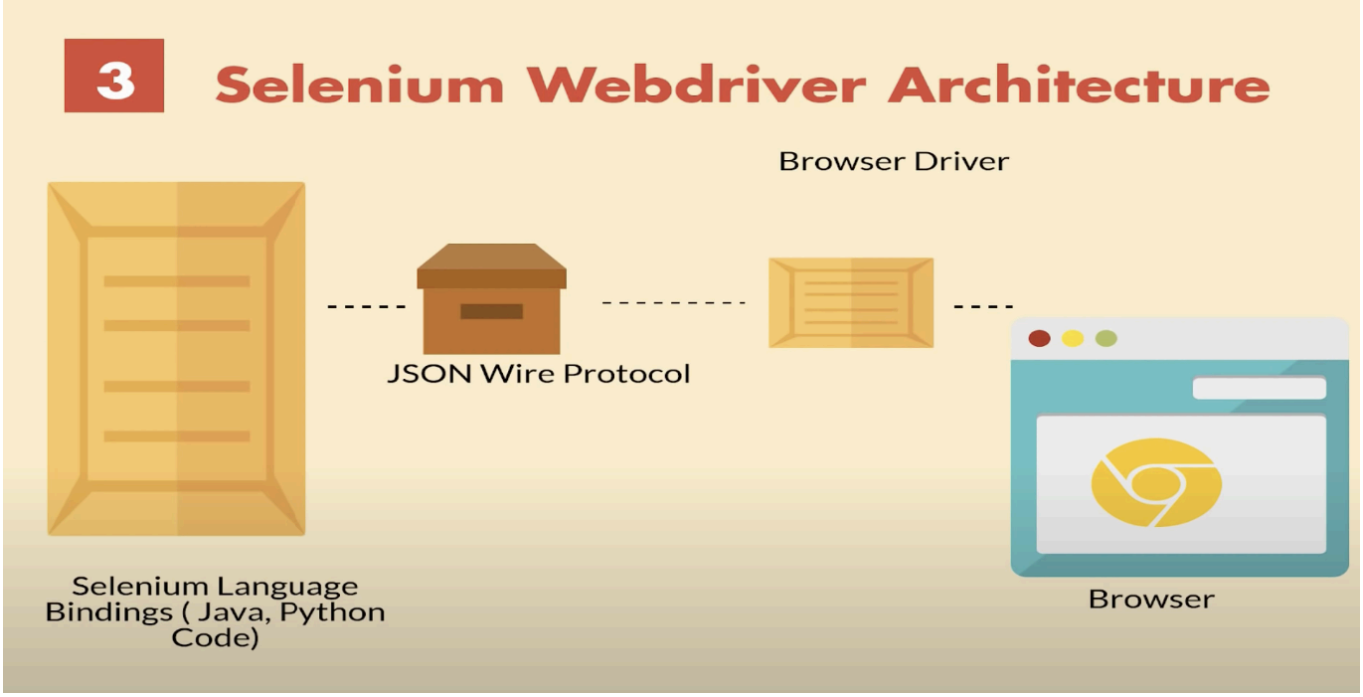
- Selenium is not well-suited for performance or load testing because it is resource-intensive and can slow down the system under test.
- When you need to test native mobile apps.
- Selenium may have difficulty interacting with custom controls or non-standard UI elements.
- Captcha / TWO-FACTOR AUTHENTICATION (2FA)
- FILE DOWNLOADS & VERIFICATION.
- AUDIO OR VIDEO STREAMING
- Security Testing
- API TESTING, mobile Appium is recommended.



WebDriver Architecture

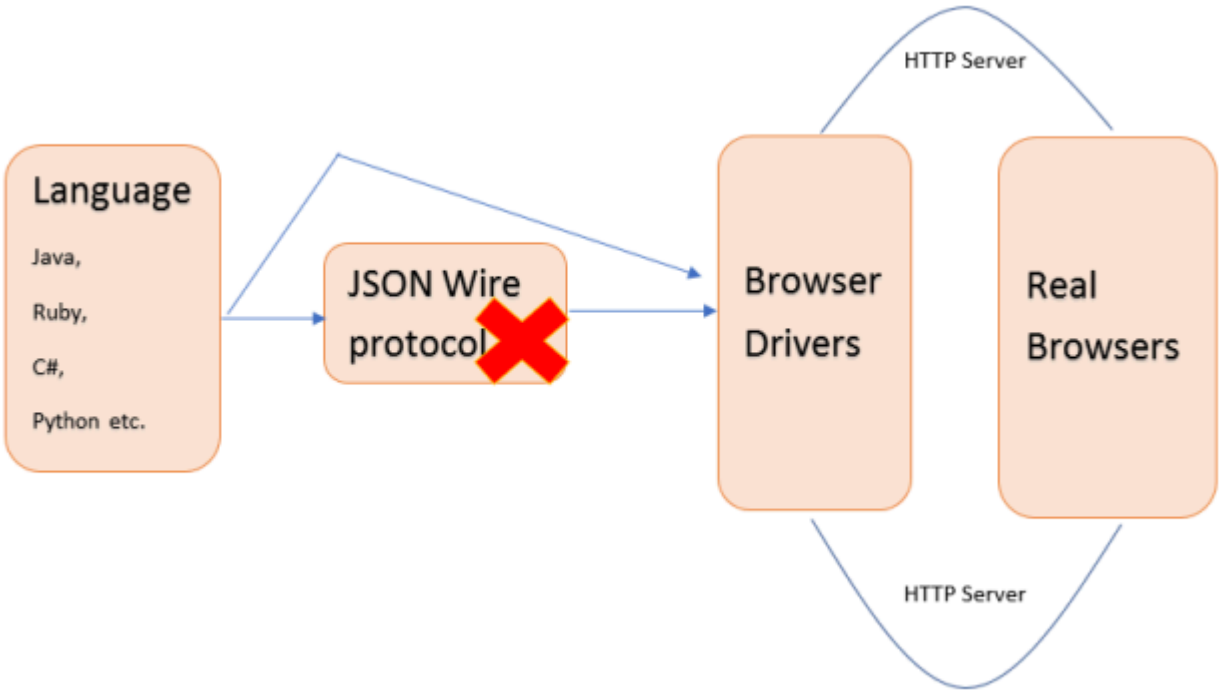
Before Selenium 4





After Selenium 4.x (w3c)

They remove the JSON wire protocol

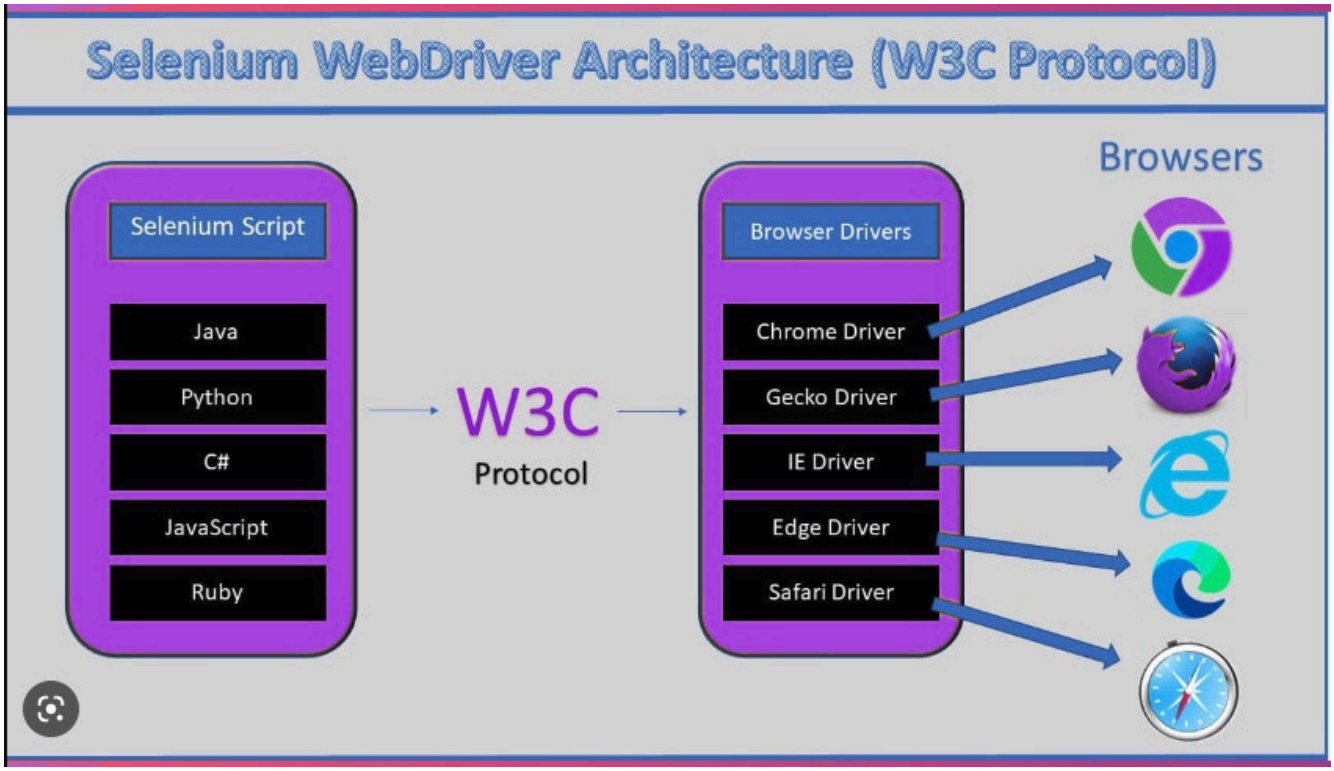


Now communicate directly to Browser via Browser Drivers.

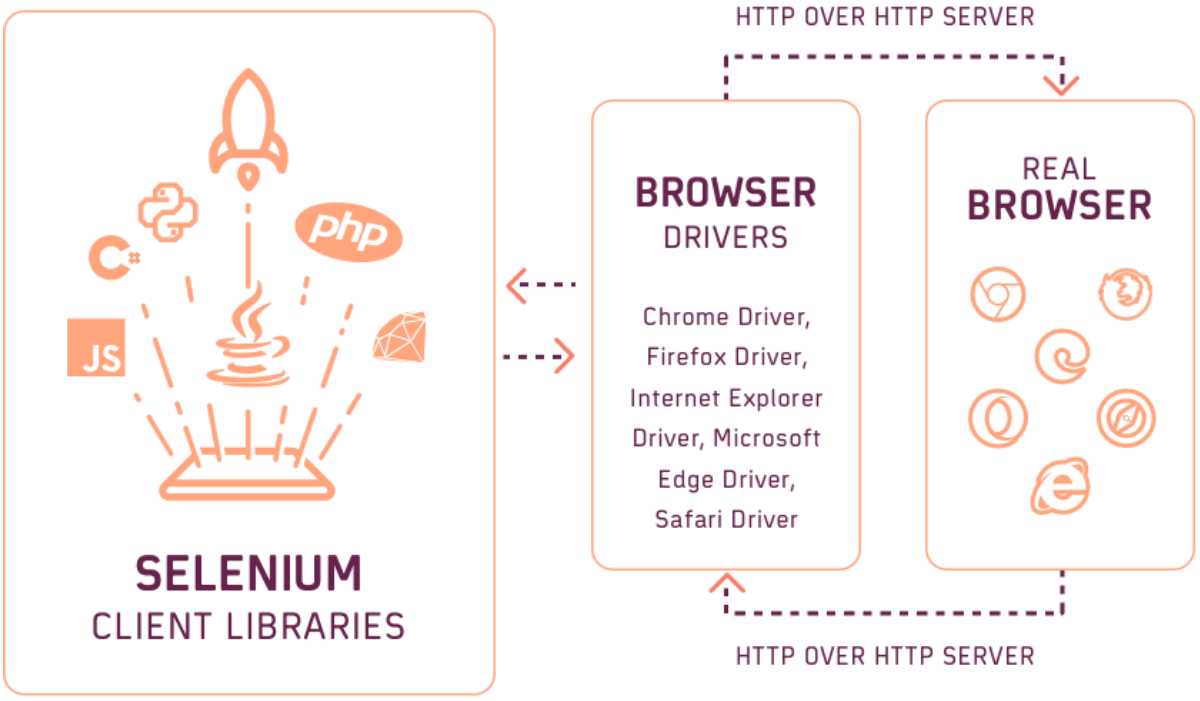
Install Browser Drivers

Quick Reference

Browser	Supported OS	Maintained by
Chromium/Chrome	Windows/macOS/Linux	Google
Firefox	Windows/macOS/Linux	Mozilla
Edge	Windows/macOS/Linux	Microsoft
Internet Explorer	Windows	Selenium Project
Safari	macOS High Sierra and newer	Apple



Postman Collection -
https://api.postman.com/collections/611814-37f62772-042a-4e0d-bde8-488e26bf8be7?access_key=PMAT-01H5E8RJNFEVG8BE7CAB049ZFZ



HTML elements

<input type="email" class="text-input W(100%)" name="username" id="login-username" data-qa="hocewoqisi" pramod="dutta">

HTML Tag - input
Attribute = value

https://www.w3schools.com/html/html_elements.asp

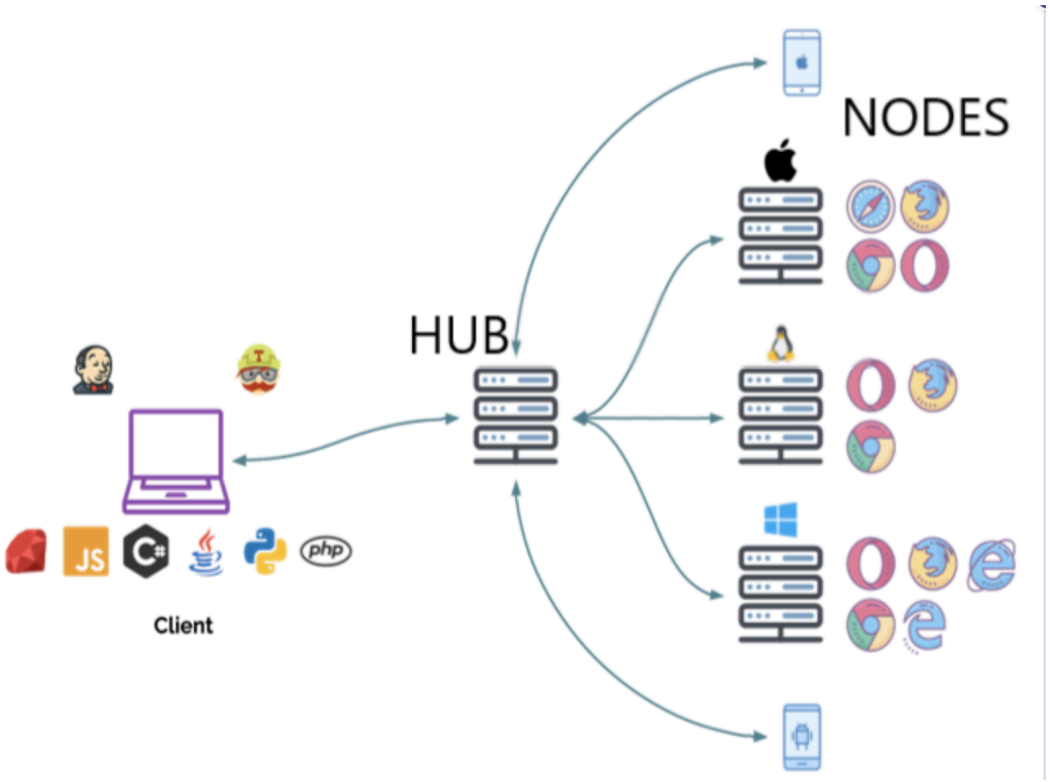
Selenium IDE

- Open source record and playback test automation for the web
- Selenide Language
- Installation
- Launch the IDE
- Recording test
    - Suite
    - test
- Command-line Runner
- npm install -g selenium-side-runner

- npm install -g chromedriver
- npm install -g geckodriver
- Control Flow
- Code Export

Selenium Grid (3,4.x)

- Smart proxy server that makes it easy to run tests in parallel on multiple machines.
- Major components of Selenium Grid.
- Hub is a server that accepts the access requests from the WebDriver client, routing the JSON test commands to the remote drives on nodes. It takes instructions from the client and executes them remotely on the various nodes in parallel
- Note device that consists of a native OS and a remote WebDriver. It receives requests from the hub in the form of JSON test commands and executes them using WebDriver



When to use Selenium Grid

- Multiple browsers and their versions.
- Reduce the time that a test suite takes to complete a test.
- Cross Browser Testing.

How to Start Selenium Grid

```
java -jar selenium-server-standalone-<version>.jar -role hub

java -jar selenium-server-standalone-<version>.jar -role node -hub
https://localhost:4444/grid/register
```

capability.setBrowserName();
capability.setPlatform();
capability.setVersion()
capability.setCapability(,);

Selenium Grid 4

- Router - Takes care of forwarding the request to the correct component.
- Distributor - Its main role is to receive a new session request and find a suitable Node where the session can be created.
- Node - Each Node takes care of managing the slots for the available browsers of the machine where it is running.
- Session Map - Keeps the information of the session id and the Node where the session is running
- Event Bus - Event Bus serves as a communication path between the Nodes
- The Grid does most of its internal communication through messages, avoiding expensive HTTP calls

Different Grid Types

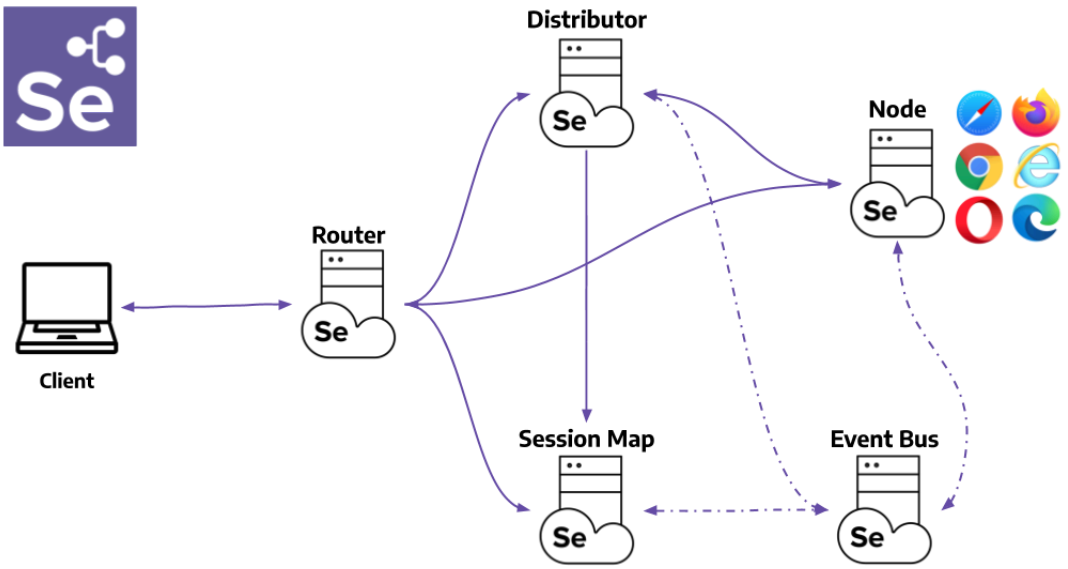
1. Standalone Mode
2. Classical Grid (Hub and Node like earlier versions)
3. Fully Distributed (Router, Distributor, Session, and Node)

Run Grid 4

```
Running in Standalone Mode  ( vs Distributed Mode)
java -jar selenium-server-4.0.0-alpha-6.jar standalone

Start the Hub:
java -jar selenium-server-4.0.0-alpha-6.jar hub
```

```
Register a Node:
java -jar selenium-server-4.0.0-alpha-6.jar node --detect-drivers
```

https://www.selenium.dev/documentation/en/grid/grid_4/setting_up_your_own_grid/



Run Selenium on Docker

```
https://github.com/SeleniumHQ/docker-selenium


docker run -d -p 4444:4444 -v /dev/shm:/dev/shm
selenium/standalone-chrome:4.0.0-alpha-7-prerelease-20201009
```

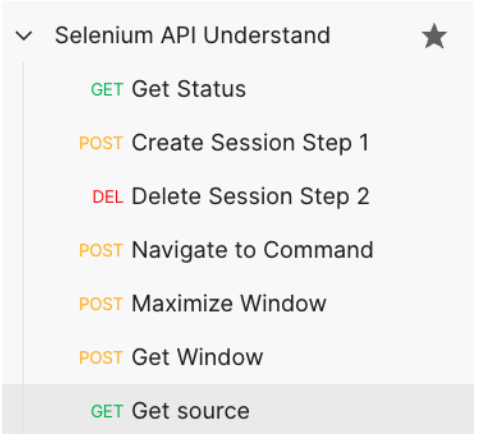
Run on Cloud Service Providers - BrowserStack

https://www.browserstack.com/

Understanding Selenium API

https://www.w3.org/TR/webdriver/

https://drive.google.com/drive/folders/1tYeAaBbvE6WptutWSz_pU6DOpuaf22pp?usp=sharing



Virtual Env

- Isolate the Python Environments
- Main Parent - Global
    - Prject basec Python version activate or deactivate the virtualenv
- How to Install
    - Pip install virtualenv
    - Virtualenv – help
    - Optional - python -m pip install --user virtualenv
    -
- Create the environment (creates a folder in your current directory)
    - virtualenv env_name
- In Linux or Mac, activate the new python environment
    - source env_name/bin/activate
- Or in Windows
    - .\env_name\Scripts\activate
- Confirm that the env is successfully selected
    - which python3
- Deactivate
    - deactivate

Logging with Pytest

1. Add by importing the import logging.
2. Add the format in which the logs you want in two
    a. Pytest.ini
    b. Pyproject.toml
        i. Add the format
3. Use the logger by
    a. logging.getLogger(__name__)
    b. logger.info(“String message”)

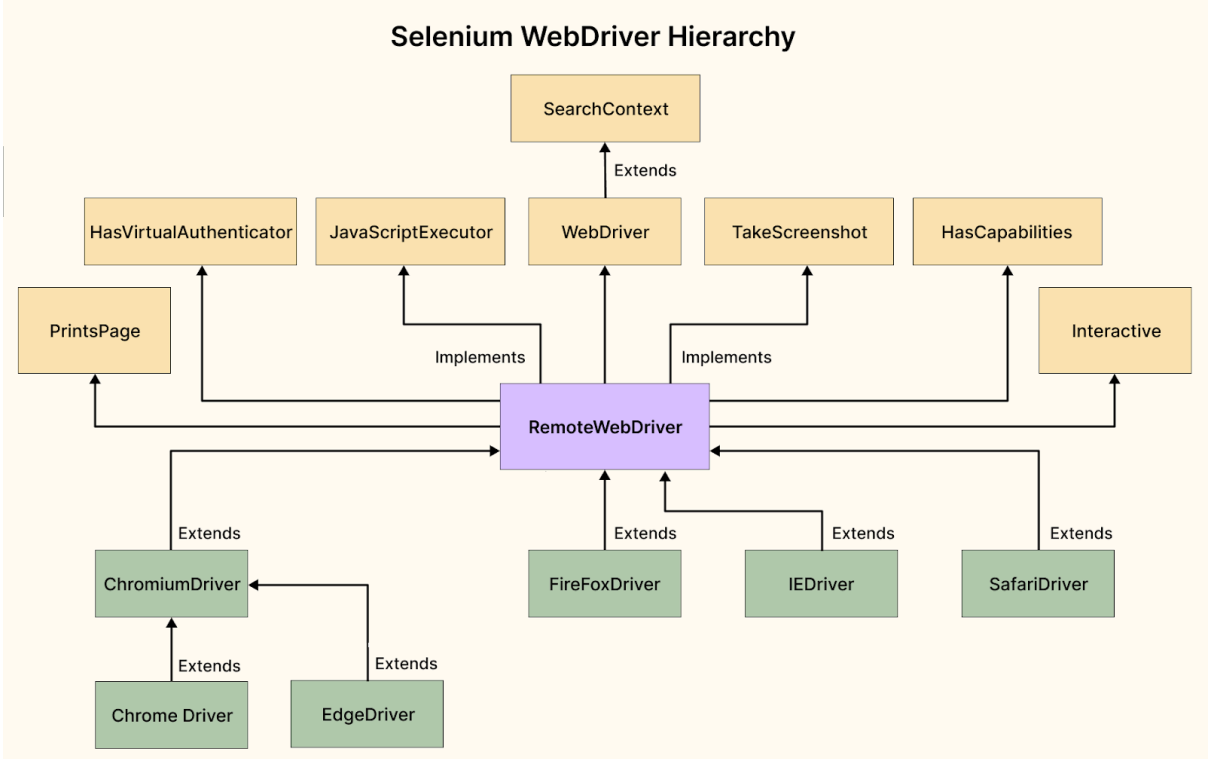
```
import logging
```

```
LOGGER = logging.getLogger(__name__)
driver.get("https://www.google.com")
LOGGER.info('eggs info')
LOGGER.warning('eggs warning')
```

```
LOGGER.error('eggs error')
LOGGER.critical('eggs critical')
```

Webdriver Hierarchy

WebDriver API – The API is a set of classes and methods that allow you to interact with the browser through code. The API allows running the tests on different browsers like Chrome, Firefox, MS Edge, etc.



API gives you access to browser controls like the navigation bar, back button, tabs, windows, etc. You can also get information from the browser, such as the current URL and page source.

Also, different actions like typing in a textbox and working with WebElements like checkboxes, radio buttons, and dropdowns can be performed using WebDriver API.

ChromeDriver

The **ChromeDriver** class provides a number of methods for interacting with the Chrome browser, such as get() for navigating to a specific URL, findElement() for locating elements on a page, and click() for simulating a mouse click on an element. You can use these methods to automate a variety of actions on the Chrome browser.

ChromeOptions

https://gist.github.com/ntamvl/4f93bbb7c9b4829c601104a2d2f91fe5

import the ChromeOptions class from the `org.openqa.selenium.chrome` **package**

```
from selenium import webdriver

def test_login():
    chrome_options = webdriver.ChromeOptions()
    chrome_options.add_argument("--start-maximized")
    driver = webdriver.Chrome(chrome_options)
    driver.get("https://app.vwo.com")
    print(driver.title)
    driver.quit()
```

```
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument("--start-maximized")
```

pageLoadStrategy

Strategy	Ready State	Notes
normal	complete	Used by default, waits for all resources to download
eager	interactive	DOM access is ready, but other resources like images may still be loading
none	Any	Does not block WebDriver at all

The `document.readyState` property of a document describes the loading state of the current document.

Proxy

A proxy server acts as an intermediary for requests between a client and a server. In simple, the traffic flows through the proxy server on its way to the address you requested and back.

```
from selenium import webdriver


def test_login():
   chrome_options = webdriver.ChromeOptions()
   chrome_options.add_argument("--start-maximized")
   # Set PageLoadStrategy to 'none' (Not a built-in option,
   # but we can use it for reference)
   # Add the proxy to ChromeOptions
   chrome_options.add_argument("--page-load-strategy=none")

   # Add the proxy to ChromeOptions
   proxy_server = "http://your_proxy_ip:your_proxy_port"
   chrome_options.add_argument('--proxy-server=' + proxy_server)

   driver = webdriver.Chrome(options=chrome_options)
   driver.get("https://app.vwo.com")
   print(driver.title)
   driver.quit()
```

Remote WebDriver

Remote WebDriver consists of a server and a client. The server is a component that listens on a port for various requests from a Remote WebDriver client.

```
from selenium import webdriver
from selenium.webdriver.common.desired_capabilities import
DesiredCapabilities

# Create ChromeOptions instance
chrome_options = webdriver.ChromeOptions()

# Add options to ChromeOptions (same as shown in the previous example)
chrome_options.add_argument('--headless')
chrome_options.add_argument('--window-size=1366x768')

# Set desired capabilities with ChromeOptions
desired_capabilities = DesiredCapabilities.CHROME.copy()
desired_capabilities['platform'] = 'ANY'  # Platform can be 'WINDOWS',
'LINUX', etc.
desired_capabilities['version'] = ''  # Version can be empty or specific
version like '91.0'

# URL of the Remote WebDriver server
remote_server_url =
"http://<remote_server_ip>:<remote_server_port>/wd/hub"

# Create Remote WebDriver instance
driver = webdriver.Remote(command_executor=remote_server_url,
desired_capabilities=desired_capabilities, options=chrome_options)

# Navigate to the desired URL
driver.get("https://example.com")

# Now you can interact with the web page using the specified options on
the Remote WebDriver
```

Remote WebDriver is commonly used in **conjunction with a cloud-based testing service**, which allows for distributed testing across multiple machines and environments.

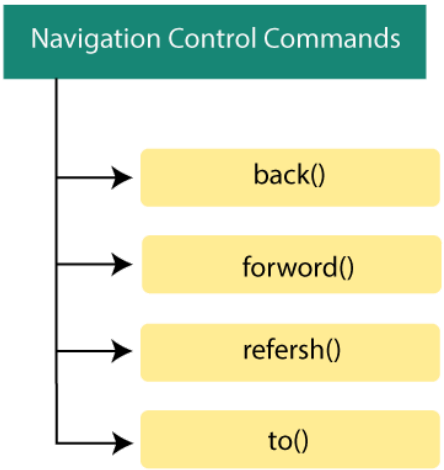
Difference Between Quit and Close in Selenium

	Quit	Close
Definition	Closes all browser windows and ends the WebDriver session	Closes the current browser window
Effect	Ends the WebDriver session completely	Leaves the WebDriver session open
Usage	Typically used at the end of a test suite or test case	Typically used at the end of a single test case
Example	`driver.quit()`	`driver.close()`

driver.close(); // Closed the window, Session id != null, Error - Invalid session Id
driver.quit(); // Closed All the window and Session = null, Error - Session ID is null

🧭Navigation in Selenium

- Refresh, forward, back
- driver.get()



Navigation commands in Selenium

get(String url) - This command is used to open a specific URL in the browser.

driver.get("https://www.example.com");
navigate().to(String url) - **Not Exist in python**

- Refresh
- Back

- forward

```
# Navigation Command
driver.back()
driver.get("https://www.bing.com")
print(driver.title)

driver.forward()
print(driver.title)

driver.back()
print(driver.title)
driver.refresh()

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

🔎 Locators in Selenium

A locator is a way of identifying an element on a web page so that it can be interacted with.

There are several different types of locators that can be used, including:

- ID: This locator type uses the unique ID attribute of an element to locate it on the page.
- Name: This locator type uses the name attribute of an element to locate it on the page.
- Class name: This locator type uses the class attribute of an element to locate it on the page.
- Tag name: This locator type uses the HTML tag name of an element to locate it on the page.
- Link text: This locator type uses the text of a link to locate it on the page.
- Partial link text: This locator type uses part of the text of a link to locate it on the page.

- **CSS selector**: This locator type uses a CSS selector to locate an element on the page.
- **Xpath:** This locator type uses an XPath expression to locate an element on the page.
- When writing test scripts with Selenium, you can use a combination of these locator types to accurately and reliably locate elements on the page.

- find_element_by_id: Finds an element by its unique id attribute.
- find_element_by_name: Finds an element by its name attribute.
- find_element_by_xpath: Finds an element using an XPath expression.
- find_element_by_link_text: Finds an anchor element (a) by its visible text.
- find_element_by_partial_link_text: Finds an anchor element (a) by a partial match of

its visible text.

- find_element_by_tag_name: Finds an element by its HTML tag name (e.g., "div", "input", "a", etc.).
- find_element_by_class_name: Finds an element by its CSS class name.
- find_element_by_css_selector: Finds an element using a CSS selector.

For multiple elements, you can use the plural versions of these functions.

- (e.g., find_elements_by_id, find_elements_by_name, etc.), which will return a list of matching WebElement objects.

It uses "locators" to identify and manipulate elements on a web page. There are several types of locators that can be used in Selenium, including:

Make Appointment

1. **ID**: This locator uses the unique id attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: ... , you can use the ID locator "#some-id" to find this element.
2. Name: This locator uses the name attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: <input name="username">, you can use the Name locator "username" to find this element.
3. Class Name: This locator uses the class attribute of an element to locate it. For example, if the HTML for an element on the page looks like this: ... , you can use the Class Name locator ".some-class" to find this element.
4. Link Text: This locator uses the visible text of a link element to locate it. For example, if the HTML for a link on the page looks like this: VWO, you can use the Link Text locator "VWO" to find this element.
5. Partial Link Text: This locator is similar to the Link Text locator, but it only matches a portion of the link text. For example, using the Partial Link Text locator "VWO" would match a link with the text "Welcome to VWO".
6. CSS Selector: This locator uses a CSS selector to locate an element. CSS selectors are strings that specify how to find an element on a page based on its HTML structure. For example, if the HTML for an element on the page looks like this: ... , you can use the CSS selector "div.some-class#some-id" to find this element.
7. XPath: This locator uses an XPath expression to locate an element. XPath is a language for navigating and selecting elements in an XML document (including HTML documents). It allows you to specify complex, hierarchical patterns for locating elements on a page. For example, if you want to find all the <p> elements that are descendants of the element with the ID "some-id", you could use the XPath expression "//div[@id='some-id']/p" to find these elements.

These are the main types of locators that are used in Selenium. Which one you use will depend on the specific elements you are trying to locate on the page, and the HTML

structure of the page itself.

[Assignment] - Automating the Login Page of VWO.com

1. Fetch the locators - https://app.vwo.com/
2. Create a Maven project and add TestNG.
3. Add the Allure Report (Allure TestNG)
4. Automate the two Test cases of VWO.com
    a. Valid Username and Valid Password
5. Run them and share results.
6. Push the code to github.com
7. Git repo - ReadMe.md a Screen shot of allure.

Understanding Locators and HTML Forms

Tag
Attribute = Value

<input data-qa="hocewoqisi" type="email" class="text-input W(100%)" name="username" id="login-username" >
<input data-qa="hocewoqisi" type="email" class="text-input W(100%)" name="username2" id="login-username" >

data-qa="hocewoqisi"
type="email"
class="text-input W(100%)"
name="username"
id="login-username"

Preference
id -> name -> className -> Link Text / Partial Text(a) -> CSS Selector -> XPath.

XPath - 60%
CSS Selector - 30%
ID, Name, CLASS - 10%

Custom attribute it is not id, name, class -> Custom Attribute -
    # student = "praveen" , roll=123, phone="233", placeholder="dasda"
    #data-qa="dasda" , testID="123"

---

findElement vs findElements

findElement() is a method used to locate a single element on a web page. It takes a locator as an argument, and returns the **first matching element** that it finds. For example:

```
usernameField = driver.findElement(By.ID,"username"));
```

In this example, findElement() is used to locate the element with the ID "username". If it is found, the element is returned and stored in the usernameField variable.

findElements() is similar to findElement(), but it returns a list of all matching elements instead of just the first one. For example:

```
allLinks = driver.findElements(By.TAGNAME("a"));
```

In this example, findElements() is used to locate all <a> elements on the page. These elements are returned in a list and stored in the allLinks variable.

It's important to note that if findElement() is used and no matching element is found, it will throw a NoSuchElementException. On the other hand, if findElements() is used and no matching elements are found, it will return an empty list.

What is an HTML Form?

A HTML form is a section of a web page that contains form elements, such as text fields, checkboxes, and buttons. Forms allow users to enter data and interact with a website.

Forms are created using the <form> HTML tag. This tag defines the start and end of a form, and it can have several attributes that determine how the form behaves.

For example, the action attribute specifies the URL of the server-side script that will process the form data, and the method attribute specifies whether the form data will be sent to the server using the GET or POST method.

```
<form action="http://www.example.com/form-handler.php" method="POST">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username">
  <br>
  <label for="password">Password:</label>
  <input type="password" id="password" name="password">
  <br>
  <input type="submit" value="Log In">
</form>
```

When the user enters their username and password and clicks the "Log In" button, the form data will be sent to the server-side script at the URL specified in the action attribute (`http://www.example.com/form-handler.php`) using the POST method.

The server-side script can then process the form data and perform the desired action, such as checking the user's credentials against a database and logging them in.

text Method

the getText() method is used to retrieve the text of an element on a web page. This method can be called on an element, and it will return the text of the element, including any child elements.

```
WebElement element = driver.findElement(By.id("some-id"));
String elementText = element.getText();
```

getAttribute() Method

the getAttribute() method is used to retrieve the value of an attribute of an element on a web page.

```
element = driver.findElement(By.ID,"some-id");
element.getAttribute("class");
```

sendKeys

the sendKeys() method is used to enter text into a text field or text area on a web page

click()

the click() method is used to simulate a user clicking on an element on a web page

⚒️ SelectorsHub for the Locators

SelectorsHub is a tool that can be used to help identify and generate locators for elements on a web page in Selenium

https://selectorshub.com/

[Assignment] - Invalid error message Capture for the Login Page of VWO.com

8. Fetch the locators - https://app.vwo.com/
9. Create a Python project and add pytest, alloure.
10. Add the Allure Report (Allure pytest)
11. Automate the two Test cases of VWO.com
    a. Invalid Username and Valid Password.
12. Capture the error and pass the test.
13. Run them and share results.

Link Text locator.

the findElement() method is used with the By.linkText() locator to locate a link on the page with the text "VWO". The element is then stored in the vwoLink variable.

The click() method is called on this element, which simulates the user clicking on the link with their mouse. This will navigate the user to the URL specified in the href attribute of the <a> element.

🛣️ Mastering XPath

What is XPath?

XPath is a query language for selecting nodes from an HTML / XML document.
XPath was defined by the World Wide Web Consortium.

Core Logic - **//tagName[@attribute='value']**

<input type="email" class="text-input W(100%)" name="username" id="login-username" data-qa="hocewoqisi">

//input[@data-qa="hocewoqisi"]
//input[@id="login-username"]
//input[@name="username"]

//li[@data-qa="rubehixixu"]/input

//*[@name="username"] - Slow way (* wild card) - search for all the tags with unique name = usernmae

For Button

//button - all buttons - Not good
//button[@type='submit'] - 2 Elements
//button[@id='js-login-btn'] - 1 Element
//button[@data-qa='sibequkica'] - 1 Element

TAG - h1, p, input, a, form, img, video, audio,button, table, ul, li, tr, div, select, span, -> Html Tags

Attrbute - id, class, name, alt, href, src, data-qa, ….srcset ..

- **Relative XPath**
- Absolute XPath

- XPath Functions

Absolute XPath
- **From the root**
- **Big problem - of on future any changes in html page**
- **Abs Xpath will break**
-

/html/body/div[2]/div[1]/div[2]/div/div[1]/div/div/div[3]/form[1]/ul/li[1]/div/input

Why do we need to MASTER Locators?

Probably the first question asked by the interviewer.

- You should always find small and efficient Locators.
- UI Automation is all about finding locators.
- Don't use tools at first.

//input[@id='login-password']/../

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that mat they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Absolute XPath

1. Complete path from the Root Element.
2. If any element is added or deleted, Xpath fails.
3. /html/body/div[2]/div[1]/div[2]/div/div[1]/div/div/div[3]/form[1]/ul/li[1]/div/input

Relative Xpath

1. Short and simple to use.
2. You can simply start by referencing the element you want and go from there

Based on searching an element in DOM.**//*[@id="login-username"].**

//input[@id="login-username"]

Xpath -> //input[@id="txt-username"]

Css - > #txt-username

XPath Functions

- **Known Attribute** - //*[@id='btn-make-appointment']

- **TAG Name** - //a[@id='btn-make-appointment']
- **Xpath Function**
    - **Full Visible text - text()** - //a[text()='Make Appointment'], //*[text()='Make Appointment']
    - **Partial Text() - contains()**
        - //a[contains(text(),'Make Appointment')]
        - //a[contains(text(),'Make')]
        - //a[contains(text(),'Appointment')]
        - //a[contains(text(),'App')] - This may fail if there 1 or more a tag with App.
        - //a[contains(@id,'btn-make-appointment')]
    - //a[starts-with(text(),'Make')]
-

Contains()
//tag_name[contains(@attribute,'value_of_attribute')]

Starts-with()
//tag_name[starts-with(@attribute,'Part_of_Attribute_value')]
Text()
//tag_name[text()='Text of the element']

String functions

concat(string, ...): XPath concat function concatenated number of arguments and return to a concatenated string.

starts-with(string, string): XPath start-with function return True/False. Return True if second argument string is start with first argument.

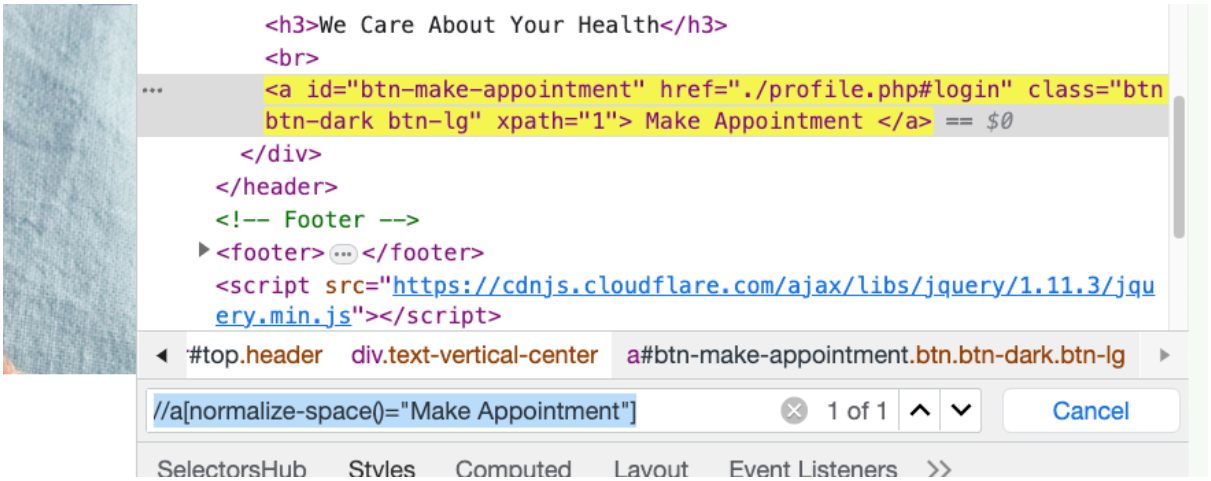
contains(string, string) - XPath contains function return True/False. Return True if second argument string is a contain of first argument.
string-length(string): XPath string-length function return the length of string.

substring-after(string, string): XPath substring-after function return the substring of the first argument string base on first occurrence of the second argument string after all character.

substring-before(string, string): XPath substring-before function return the substring of the first argument string base on first occurrence of the second argument string before all character.

normalize-space(string): XPath normalize-space function sequence of whitespace combine into single normalize space and removing leading and trailing whitespace.

https://katalon-demo-cura.herokuapp.com/



Operators - AND & OR

And Example
//tag_name[@name = 'Name value' and @id = 'ID value']

https://katalon-demo-cura.herokuapp.com/

//a[text()="Make Appointment" and contains(@id,"btn-make-appointment")]

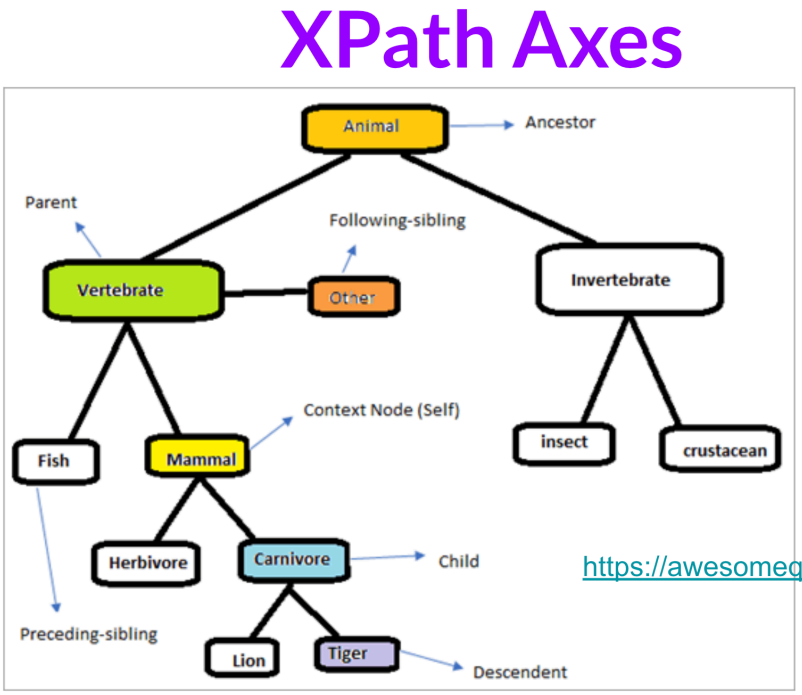
OR Example
//input[@placeholder ='Full Name' or @type = 'text']

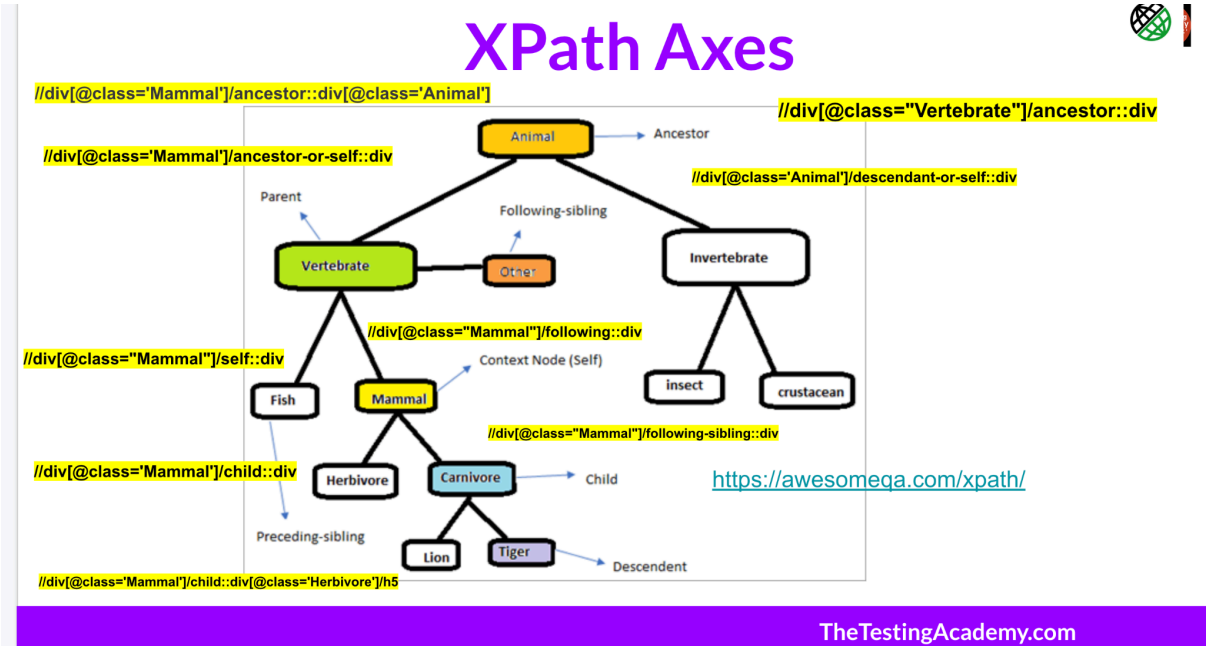
Operator	Description
|	Computes two node-sets
+	Addition
-	Subtraction
*	Multiplication
div	Division
=	Equal
!=	Not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
or	or
and	and
mod	Modulus (division remainder)

XPath Axes

In the XML documents, we have relationships between various nodes to locate those nodes in the DOM structure.

- Ancestor
- Child, parent
- Descendant
- Following, following-sibling
- Self.





AxisName	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

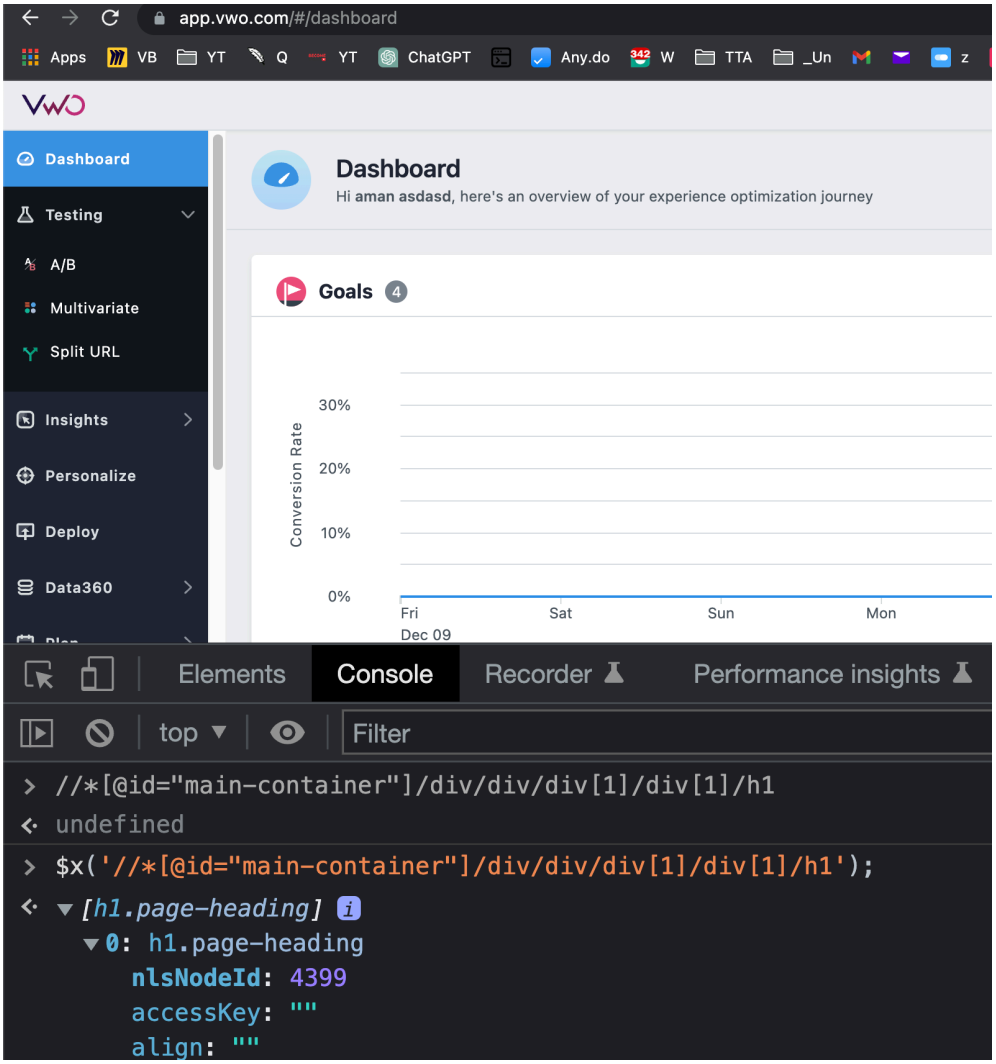
https://www.softwaretestinghelp.com/xpath-axes-tutorial/

//span[text()='Invalid Email']/ancestor::div

//*[@id="main-page"]/div[1]/child::div

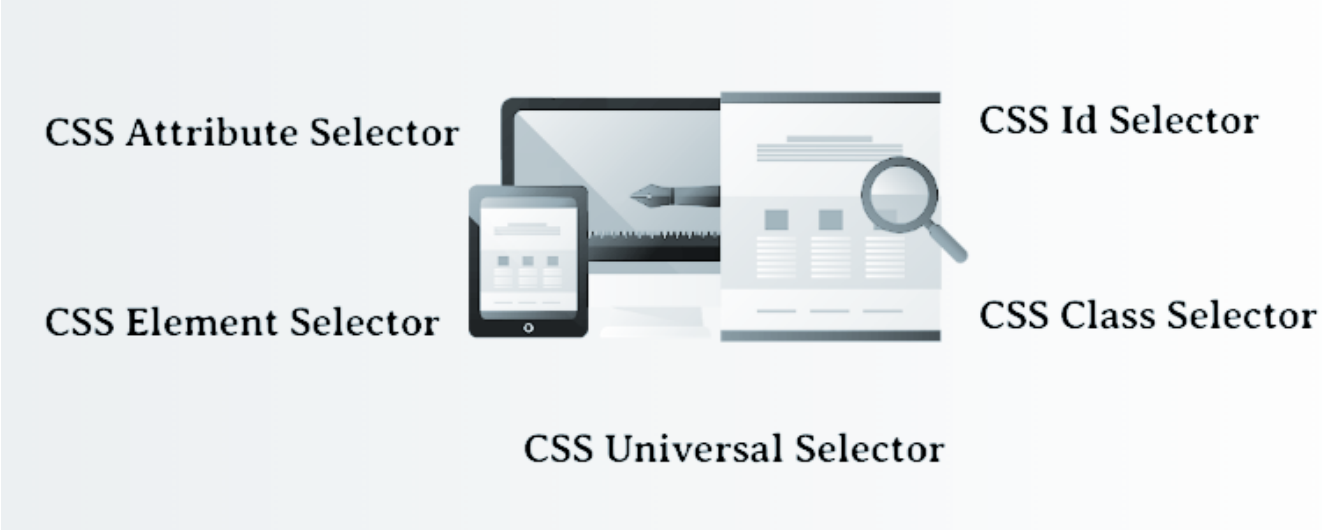
//*[@id="js-main-container-wrap"]/child::div
//*[@id="js-main-container-wrap"]/following::div

https://devhints.io/xpath



⚠️ Master CSS Selectors

CSS selectors are used to select elements in an HTML or XML document in order to apply styles or other manipulations to those elements.



CSS Selectors?

#id
.class
div.first > span

Demo https://awesomeqa.com/css/

div.first > span:nth-child(3)

li:nth-of-type(even)

div.first > span:nth-child(2n+1)

Direct Child Selector > p > span

div.first > span:nth-of-type(2n+1)

Wildcard Selectors (*, ^ and $) in CSS

div.first > span:first-child

[attribute*=”str”] Selector:

- * contains.
- ^ begins with
- $ ends with

div.first > span:last-child

CSS selectors allow you to select elements based on their tag name, id, class, attribute, and other characteristics.

- To select all elements with the tag "p" (paragraph), you could use the following selector: p
- To select an element with the ID "main-heading", you could use the following selector: #main-heading
- To select all elements with the class "error", you could use the following selector: .error
- To select all elements with the attribute "disabled", you could use the following selector: [disabled]
- To select all "a" elements that are descendants of a "nav" element, you could use the following selector: nav a

form#login-form input[type="radio"]

CSS [attribute*=value] Selector

The [attribute*=”str”] selector is used to select those elements whose attribute value contains the specified substring str.

CSS [attribute=value] Selector
The [attribute=value] selector in CSS is used to select those elements whose attribute value is equal to “value”.

CSS [attribute$=value] Selector The [attribute$=”value”] selector is used to select those elements whose attribute value ends with a specified value “value”.
CSS [attribute|=value] Selector This is used to select those elements whose attribute value is equal to “value” or whose attribute value started with “value” immediately followed by hyphen (-).

CSS [attribute~=value] Selector The [attribute~=”value”] selector is used to select those elements whose attribute value contains a specified word.

CSS [attribute^=value] Selector The [attribute^=value] selector is used to select those elements whose attribute value begins with given attribute.

CSS :first-child Selector The :first-child selector is used to select those elements which are the first-child elements.

CSS :last-child Selector The :last-child Selector is used to target the last child element of it’s parent for styling.

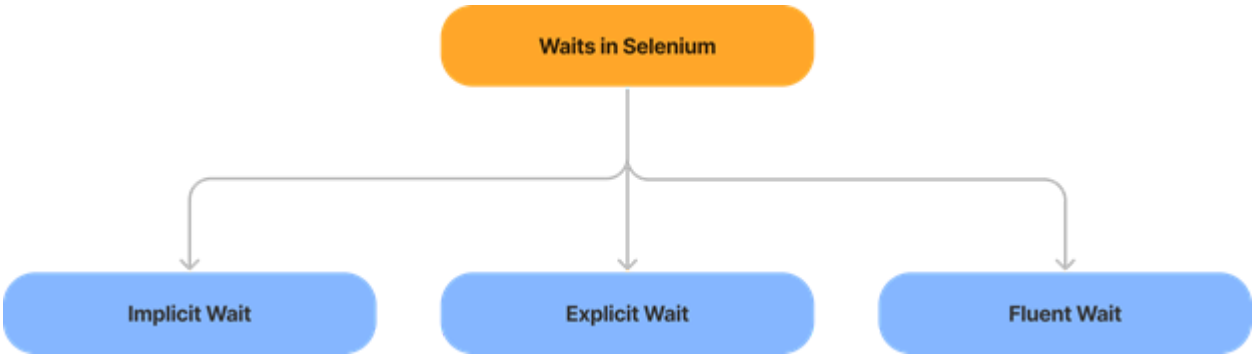
CSS :nth-child() Selector The :nth-child() CSS pseudo-class selector is used to match the elements based on their position in a group of siblings.

CSS :nth-of-type() Selector The :nth-of-type() in css Selector is used to style only those elements which are the nth number of child of its parent element.

⌛ Selenium Waits

Why Do We Need Waits In Selenium?

- Web applications are developed using Ajax and Javascript.
- New JS frameworks are more advanced and use AJax, react, and angular.
- elements which we want to interact with may load at different time intervals.



Implicit Wait

- Selenium Web Driver has borrowed the idea of implicit waits from Watir.
- If the element is not located on the web page within that time frame, it will throw an **exception**.
- WebDriver polls the DOM for a certain duration when trying to find any element.
- Global settings applicable to all elements
- It tells the web driver to wait for the x time before moving to the next command.
- Gives No Such Element Exception.
- Once it is set it is applicable to full automation script.
- Implicit wait is maximum time between the two commands.
- Different from time.sleep - time.sleep() - It will sleep time for script/ Py Int.
- Not good way to use it in script as it's sleep without condition.
- Do not mix implicit and explicit waits. Doing so can cause unpredictable wait times.

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.implicitly_wait(10)  # Wait up to 10 seconds for elements to
appear
```

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

Explicit Wait

Explicit Wait in Selenium is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing "ElementNotVisibleException" exception

- Little intelligent wait, wait for certain conditions.
- They **allow your code to halt program execution, or freeze the thread, until the condition you pass it resolves.**
- The condition is called with a certain frequency until the timeout of the wait is Elapsed.
- This means that for as long as the condition returns a falsy value, it will keep trying and waiting.
- It provides better way to handle the dynamic Ajax elements
- Element not visible exception if element not found.
- Good fit for synchronizing the state between the browser and its DOM, and your.
- Replace Thread.sleep / time.sleep() with explicit wait always

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

# Wait up to 10 seconds until an element with ID 'some-element' becomes
visible - 2 ->
element = WebDriverWait(driver, 10).until(
    EC.visibility_of_element_located((By.ID, 'some-element'))
)
```

The following are the Expected Conditions that can be used in Selenium Explicit Wait

Expected Conditions for Waiting:

You can use various expected conditions with explicit waits, such as:

- visibility_of_element_located: Wait for an element to become visible.
- element_to_be_clickable: Wait for an element to be clickable.

- presence_of_element_located: Wait for an element to be present in the DOM.
- text_to_be_present_in_element: Wait for specific text to be present in an element.
- title_contains: Wait for the page title to contain a specific text.

[Assignment] Fix the VWO login page with the heading page visibility, Use Expected Condition

Fluent Wait

Fluent Wait instance defines the maximum amount of time to wait for a condition as well as the frequency with which to check the condition

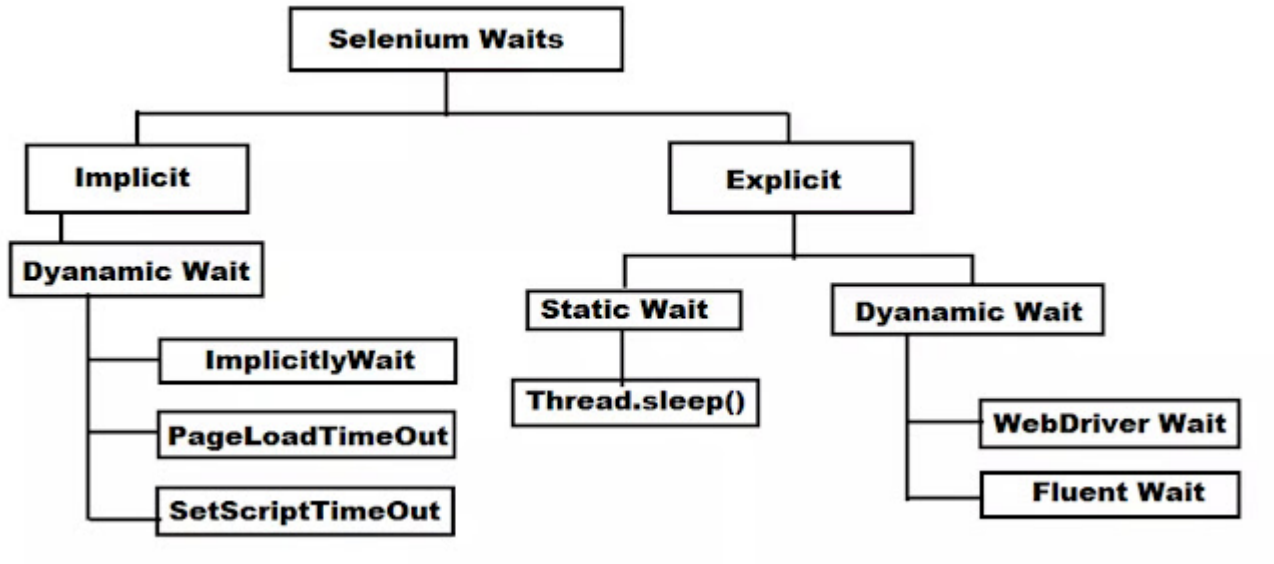
- Exception - NoSuchElementException
- Waiting 30 seconds for an element to be present on the page, checking for its presence once every 5 seconds.

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

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

wait = FluentWait(driver, timeout=30, polling_frequency=5,
ignored_exceptions=[NoSuchElementException])
element = wait.until(EC.presence_of_element_located((By.ID,
'some-element')))
```

https://www.selenium.dev/documentation/en/webdriver/waits/

Ref - https://www.guru99.com/implicit-explicit-waits-selenium.html

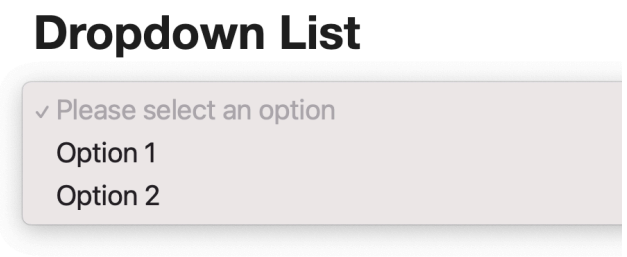
Implicit Wait	Explicit Wait
• Implicit Wait time is applied to all the elements in the script	• Explicit Wait time is applied only to those element by us
• In Implicit Wait, we need **not** specify “ExpectedConditions” on the element to be located	• In Explicit Wait, we need to specify “ExpectedCon element to be located
• It is recommended to use when the elements are located with the time frame specified in Selenium implicit wait	• It is recommended to use when the elements are load and also for verifying the property of the elen like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected)



Select Demo, Static and Dynamic Dropdowns

Handling Static Dropdowns

https://the-internet.herokuapp.com/dropdown



Handling Dynamic Dropdowns

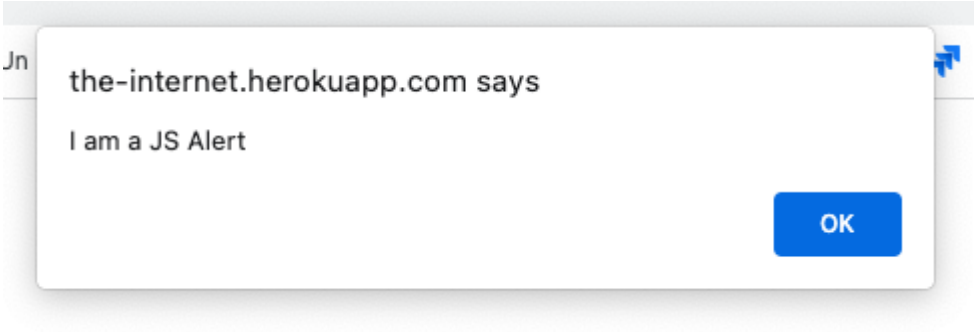
1. We will use the XPath axes.
2. We will use the advanced css selectors for the same.
3. Traditional select classes won't work.

Alert in Selenium

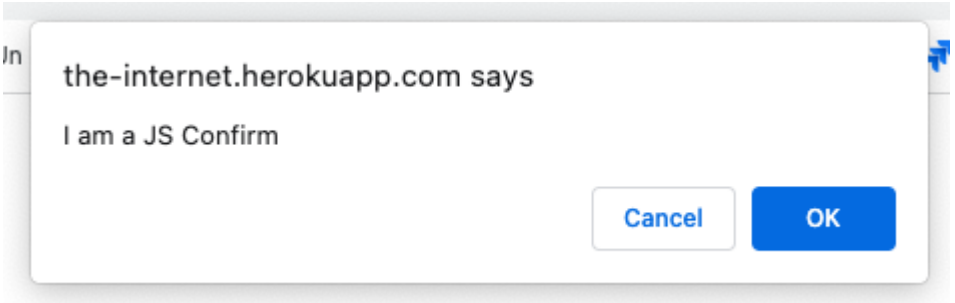
An alert is a small window that appears on top of a web page and displays a message to the user. Most of the time, alerts are used to show important information or ask the user for something.

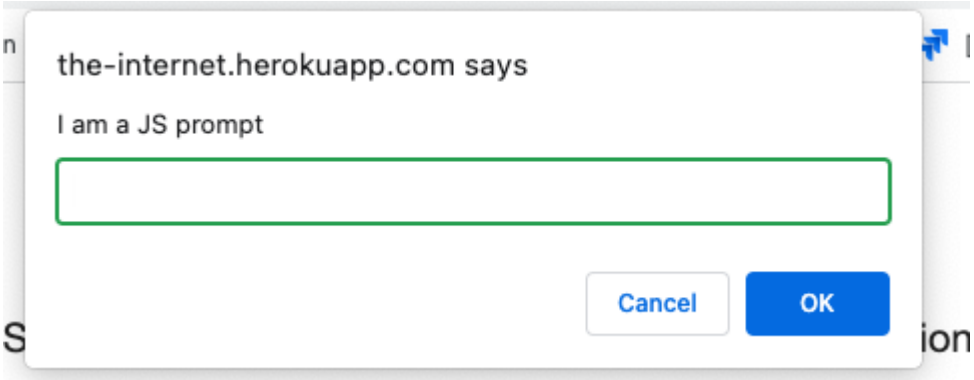
https://the-internet.herokuapp.com/javascript_alerts

Prompt Alert



Confirmation Alert





Handle Alert in Selenium WebDriver

```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support.expected_conditions import
alert_is_present

# Initialize the Chrome WebDriver
driver = webdriver.Chrome()

# Navigate to a web page that displays an alert
driver.get("http://example.com/page-with-alert")

# Wait for the alert to appear
wait = WebDriverWait(driver, 10)
wait.until(alert_is_present())

# Now you can interact with the alert using the driver.switch_to.alert
method
alert = driver.switch_to.alert

# For example, accept the alert
alert.accept()

# Or dismiss the alert
# alert.dismiss()

# After interacting with the alert, you can continue with the rest of
your test script
# ...

# Don't forget to close the browser when you are done
driver.quit()
```

1) **void dismiss() // To click on the ‘Cancel’ button of the alert.**

2) void accept()// To click on the ‘OK’ button of the alert.

3) String getText() // To capture the alert message.

4) void sendKeys(String stringToSend) // To send some data to alert box.

```
from selenium import webdriver
from selenium.webdriver.common.alert import Alert

# Initialize the Chrome WebDriver
driver = webdriver.Chrome()

# Navigate to a web page that displays an alert
driver.get("http://example.com/page-with-alert")

# Switch to the alert
alert = Alert(driver)

# Send keys (input text) to the alert
alert.send_keys("Text")

# Continue with the rest of your test script..
```

Handling Checkboxes and Handling Radio Buttons

Checkboxes and radio buttons are types of form elements that allow users to make multiple selections or choose a single option from a group of options.

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

# Initialize the Chrome WebDriver
driver = webdriver.Chrome()

# Navigate to the web page containing checkboxes
driver.get("http://example.com/page-with-checkboxes")
```

```
# Find all checkbox elements using CSS selector
checkboxes = driver.find_elements(By.CSS_SELECTOR,
"input[type='checkbox']")

# Iterate through the checkbox elements
for checkbox in checkboxes:
    # Check the checkbox if it is not already checked
    if not checkbox.is_selected():
        checkbox.click()

# Continue with the rest of your test script...
```

Web Table in Selenium

What is a Web Table?

A web table is a way of representing data in rows and columns.

"<table>" - It defines a table. You can also say that it's the starting point of a table.

<thead>
<tbody>

"<th>" - It defines a header cell, which means you should define your headings inside th tag.
"<tr>" - It defines a row in a table.
"<td>" - It defines a cell in a table. "td" always lie inside the tr tag.

```
//table[@id="customers"]
```

//table[contains(@id,"cust")]

Example -
https://awesomeqa.com/webtable.html

Company	Contact	Country
Google	Maria Anders	Germany
Meta	Francisco Chang	Mexico
Microsoft	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Adobe	Yoshi Tannamuri	Canada
Amazon	Giovanni Rovelli	Italy

Static Table - Data will not change.

Dynamic Table - No of Col may change.

```
from selenium import webdriver

# Initialize the Firefox driver
driver = webdriver.Firefox()

# URL for the web page
URL = "https://awesomeqa.com/webtable.html"
driver.get(URL)
driver.maximize_window()

# Number of Rows and Columns in the table
row_elements =
driver.find_elements_by_xpath("//table[@id='customers']/tbody/tr")
col_elements =
driver.find_elements_by_xpath("//table[@id='customers']/tbody/tr[2]/td")

row = len(row_elements)
col = len(col_elements)

print(row)
print(col)

first_part = "//table[@id='customers']/tbody/tr["
second_part = "]/td["
third_part = "]"

# Loop through rows and columns to print data
for i in range(2, row + 1):
    for j in range(1, col + 1):
        dynamic_xpath = f"{first_part}{i}{second_part}{j}{third_part}"
        data = driver.find_element_by_xpath(dynamic_xpath).text
        print(data, end=" ")
    print()

# Find Helen Bennett's country
for i in range(2, row + 1):
    for j in range(1, col + 1):
        dynamic_xpath = f"{first_part}{i}{second_part}{j}{third_part}"
        data = driver.find_element_by_xpath(dynamic_xpath).text
        if "Helen Bennett" in data:
```

```
            country_path = f"{dynamic_xpath}/following-sibling::td"
            country_text = driver.find_element_by_xpath(country_path).text
            print("------")
            print(f"Helen Bennett is in - {country_text}")

print(" |||||||||||||||||||||| \n")

# Navigate to another URL
driver.get("https://awesomeqa.com/webtable1.html")

# Get the table
table = driver.find_element_by_xpath("//table[@summary='Sample
Table']/tbody")
rows_table = table.find_elements_by_tag_name("tr")

# Loop through rows and columns to print data
for row_element in rows_table:
    columns_table = row_element.find_elements_by_tag_name("td")
    for element in columns_table:
        print(element.text)

# Quit the driver
driver.quit()
```

Actions, Windows and iframe.

Actions class is an ability provided by Selenium for handling keyboard and mouse events.

- Keyboard Events
- Mouse Events
- Wheel Mouse

```
from selenium.webdriver.common.action_chains import
ActionChains
actions = ActionChains(driver)

actions.key_down(Keys.SHIFT)\
.send_keys_to_element(FIRSTNAME, "the testing academy")\
```

```
.key_up(Keys.SHIFT)\
.perform()
```

Methods of Action Class
Action class is useful mainly for mouse and keyboard actions. In order to perform such actions, Selenium provides various methods.

Mouse Actions in Selenium:

1. **Perform Mouse Hover Action on the Web Element**
2. **moveToElement(live).build().perform();**
3. **doubleClick():** Performs double click on the element
4. **clickAndHold()**: Performs long click on the mouse without releasing it
5. **dragAndDrop():** Drags the element from one point and drops to another
6. **moveToElement():** Shifts the mouse pointer to the center of the element
7. **contextClick():** Performs right-click on the mouse Keyboard Actions in Selenium
8. **sendKeys()**: Sends a series of keys to the element
9. **keyUp():** Performs key release
10. **keyDown():** Performs keypress without release.

```
import pytest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.action_chains import
ActionChains
from selenium.webdriver.common.by import By

@pytest.mark.actions
def test_01_actions():
    # Initialize the Firefox driver
    driver = webdriver.Firefox()

    # URL for the web page
    URL = "https://awesomeqa.com/practice.html"
    driver.get(URL)
    driver.maximize_window()

    FIRSTNAME = driver.find_element(By.NAME, "firstname")

    # Create object of ActionChains class
    actions = ActionChains(driver)
```

```
    # This will type Username in Uppercase as we are typing
using Shift key pressed
    actions.key_down(Keys.SHIFT)\
        .send_keys_to_element(FIRSTNAME, "the testing
academy")\
        .key_up(Keys.SHIFT)\
        .perform()

    date = driver.find_element(By.ID, "datepicker")
    actions.send_keys_to_element(date,
"23/12/2025").perform()

    link = driver.find_element(By.XPATH,
"//a[contains(text(), 'Click here to Download File')]")
    actions.context_click(link).perform()

    # Quit the driver
    driver.quit()

# Run the test function
if __name__ == "__main__":
    pytest.main([__file__])
```

Keyboard Events

KeyDown(KeyCode) - Performs key press without releasing it.

```
ActionChains(driver)\
      .key_down(Keys.SHIFT)\
      .send_keys("abc")\
      .perform()
```

KeyUp(KeyCode) - Performs a key release. It has to be used after keyDown to release the key.

```
 ActionChains(driver)\
      .key_down(Keys.SHIFT)\
      .send_keys("a")\
      .key_up(Keys.SHIFT)\
      .send_keys("b")\
      .perform()
```

Send Keys

```
ActionChains(driver)\
      .send_keys("abc")\
      .perform()
```

Send to Element

```
text_input = driver.find_element(By.ID, "textInput")
   ActionChains(driver)\
      .send_keys_to_element(text_input, "abc")\
      .perform()
```

Copy and Paste

```
cmd_ctrl = Keys.COMMAND if sys.platform == 'darwin' else Keys.CONTROL

    ActionChains(driver)\
        .send_keys("Selenium!")\
        .send_keys(Keys.ARROW_LEFT)\
        .key_down(Keys.SHIFT)\
        .send_keys(Keys.ARROW_UP)\
        .key_up(Keys.SHIFT)\
        .key_down(cmd_ctrl)\
        .send_keys("xvv")\
        .key_up(cmd_ctrl)\
        .perform()
```

Mouse actions

- Click and hold
- Click and release
- Context Click
- Back Click
- Double click
- Move to element
- Move by offset

Drag and Drop

With Action or with Function

```
String URL = "https://the-internet.herokuapp.com/drag_and_drop";
        driver.get(URL);
        driver.manage().window().maximize();
//Actions class method to drag and drop
        Actions builder = new Actions(driver);
        WebElement from = driver.findElement(By.id("column-a"));
        WebElement to = driver.findElement(By.id("column-b"));
//Perform drag and drop
        builder.dragAndDrop(from,to).perform();
```

Scroll wheel actions

- Scroll to element
- Scroll by given amount
- Scroll from an element by a given amount
- Scroll from an element with an offset

File Upload

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

# Set Chrome options
options = Options()
options.page_load_strategy = 'normal'

# Initialize the Chrome driver with options
driver = webdriver.Chrome(options=options)

# URL for the web page
URL = "https://awesomeqa.com/selenium/upload.html"
driver.get(URL)
driver.maximize_window()

upload_file = driver.find_element(By.XPATH,
"//input[@id='fileToUpload']")
upload_file.send_keys("/Users/pramod/Documents/Course/apitesting.jpeg")

driver.find_element(By.NAME, "submit").click()
```

```
# Quit the driver
driver.quit()
```

Window:
In any browser, a window is the main webpage to which the user is directed after clicking on a link or URL. Such a window in Selenium is referred to as the "parent window also known as the main window which opens when the Selenium WebDriver session is created and has all the focus of the WebDriver.

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

@pytest.mark.usefixtures("driver")
def test_01_windows(driver):
    # Open the page
    driver.get("https://the-internet.herokuapp.com/windows")

    # Store the handle of the current window
    main_window_handle = driver.current_window_handle

    # Find the "Click Here" link
    link = driver.find_element(By.LINK_TEXT, "Click Here")

    # Click the link to open a new window
    link.click()

    # Store the handles of all open windows in a list
    window_handles = driver.window_handles

    # Iterate through the list of window handles
    for handle in window_handles:
        # Switch the focus to each window in turn
        driver.switch_to.window(handle)

        # Check if the text "New Window" is present in the window
        if "New Window" in driver.page_source:
            print("The text 'New Window' was found in the new window.")
            break

    # Switch the focus back to the main window
```

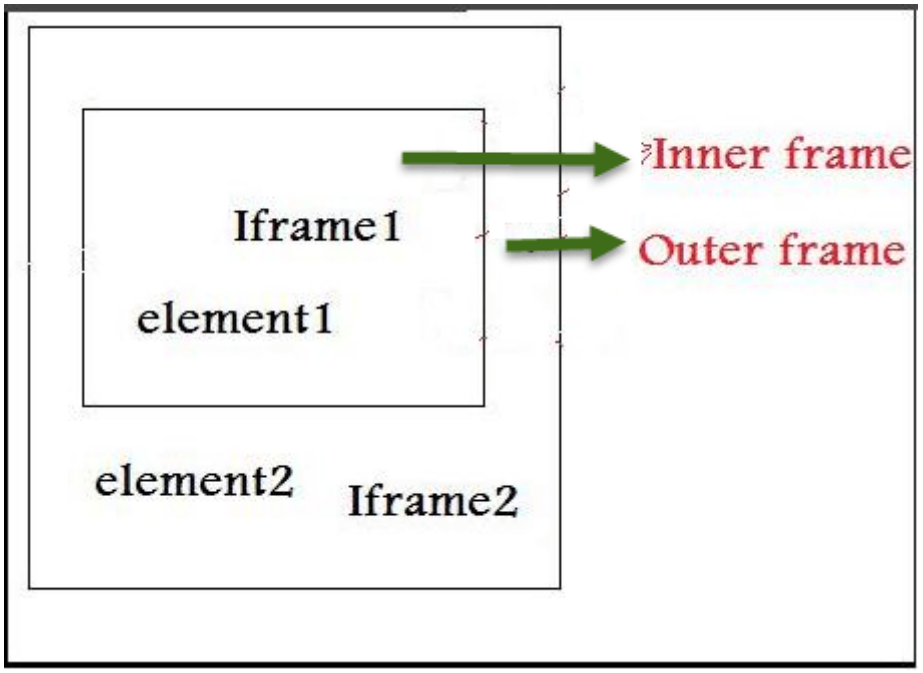
```
    driver.switch_to.window(main_window_handle)

if __name__ == "__main__":
    pytest.main([__file__])
```

IFRAME

An iframe (short for inline frame) is an HTML element that allows you to embed another HTML document within the current document. Iframes are often used to embed videos, advertisements, or other external content on a webpage.

1.
2. By Index
3. By Name or Id
4. By Web Element

```
    # Switch frame by id
driver.switch_to.frame('buttonframe')

    # Now, Click on the button
driver.find_element(By.TAG_NAME, 'button').click()
```



[Assignment] Open HEATMAP of vwo.com and Click on iframe Click map

ActionClass, Iframew, Windows

1. Open this link with webdriver https://app.vwo.com/#/analyze/osa/13/heatmaps/1?token=eyJhY2NvdW50X2lkIjo2NjY0MDAsImV4cGVyaW1lbnRfaWQiOjEzLCJjcmVhdGVkX29uIjoxNjcxMjA1MDUwLCJ0eXBlIjoiY2FtcGFpZ24iLCJ2ZXJzaW9uIjoxLCJoYXNoIjoiY2IwNzBiYTc5MDM1MDI2N2QxNTM5MTBhZDE1MGU1YTUiLCJzY29wZSI6IiIsImZybiI6ZmFsc2V9&isHttpsOnly=1
2. Use Action to MOVE the mouse to View Heatmap and Click on it.
3. Switch the Window and Switch to iframe
4. Click on button Click Map in the iframe of heatmap.

Solution

i

```
import time

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

@pytest.fixture
def driver():
    driver = webdriver.Chrome()
```

```
    yield driver
    # Close the web driver
    driver.quit()

@pytest.mark.usefixtures("driver")
def test_02_windows_actions_complex(driver):
    URL =
"https://app.vwo.com/#/analyze/osa/13/heatmaps/1?token=eyJhY2NvdW50X2lkI
jo2NjY0MDAsImV4cGVyaW1lbnRfaWQiOjEzLCJjcmVhdGVkX29uIjoxNjcxMjA1MDUwLCJ0e
XBlIjoiY2FtcGFpZ24iLCJ2ZXJzaW9uIjoxLCJoYXNoIjoiY2IwNzBiYTc5MDM1MDI2N2QxN
TM5MTBhZDE1MGU1YTUiLCJzY29wZSI6IiIsImZybiI6ZmFsc2V9&isHttpsOnly=1"
    driver.get(URL)
    driver.maximize_window()

    mainWindowHandle = driver.current_window_handle

    ac = ActionChains(driver)
    ac.move_to_element(driver.find_element(By.CSS_SELECTOR,
"[data-qa='yedexafobi']")).click().perform()


    time.sleep(20)


    window_handles = driver.window_handles

    # Here we will check if child window has other child windows and
will fetch the heading of the child window
    for handle in window_handles:
        if mainWindowHandle != handle:
            driver.switch_to.window(handle)
            driver.switch_to.frame("heatmap-iframe")
            driver.find_element(By.CSS_SELECTOR,
"[data-qa='liqokuxuba']").click()



if __name__ == "__main__":
    pytest.main([__file__])
```

JavaScript executor -

The JavaScript Executor is a feature of the Selenium WebDriver that allows you to execute JavaScript code within the context of the current page.
This can be useful for interacting with elements on the page that are not directly accessible through the Selenium API, or for bypassing certain limitations of the Selenium API.

Here are some common functions that you can use with the JavaScript Executor in Selenium:

- **arguments[0].click()**: This function clicks on the element specified as the first argument.
- **arguments[0].scrollIntoView():** This function scrolls the element specified as the first argument into view.
- **arguments[0].setAttribute(arguments[1], arguments[2]):** This function sets the attribute specified by the second argument to the value specified by the third argument for the element specified as the first argument.
- **arguments[0].innerHTML** = arguments[1]: This function sets the inner HTML of the element specified as the first argument to the value specified by the second argument.
- **return arguments[0].value:** This function returns the value of the element specified as the first argument.
- **return arguments[0].style.display:** This function returns the display style of the element specified as the first argument.

What can you do?

- JavaScriptExecutor provides two methods “executescript” & “executeAsyncScript” to handle.
- Executed the JavaScript using Selenium Webdriver.
- Illustrated how to click on an element through JavaScriptExecutor, if selenium fails to click on element due to some issue.
- Generated the ‘Alert’ window using JavaScriptExecutor.
- Navigated to the different page using JavaScriptExecutor.
- Scrolled down the window using JavaScriptExecutor.
- Fetched URL, title, and domain name using JavaScriptExecutor.

Dynamic Elements

let’s say ‘id’ of a username field is ‘uid_123’

Class=”abc-kkj3k2jk3j2”
id=”web-2323sdsdsd”
Use any of the Techq.

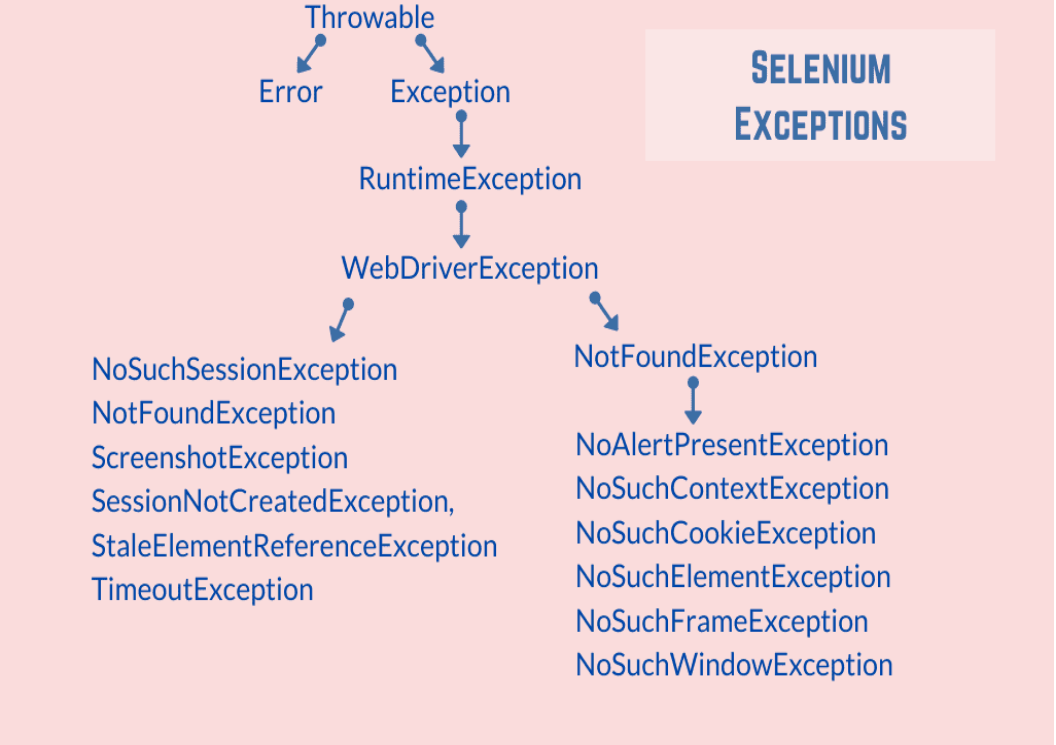
- [contains(@id,’uid’)]”
- */*[starts-with(@id,’uid’)]*
- *Relative Locators*
- *Xpath Axes*

- *Custom CSS Selectors*

Project CRM

1. Login with the Credential - https://opensource-demo.orangehrmlive.com/web/index.php/auth/login
2. Add user https://opensource-demo.orangehrmlive.com/web/index.php/admin/saveSystemUser
3. Search User

Selenium Exception



NoSuchElementException: This exception is thrown when the ***web driver is unable to locate an element*** on the page using the specified search criteria.

NoSuchFrameException: This exception is thrown when the web driver is unable to switch to a specified frame.

NoAlertPresentException: This exception is thrown when the web driver is unable to find an alert box on the page.

ElementNotVisibleException: This exception is thrown when the web driver is unable to interact with an element that is not visible on the page.

ElementNotInteractableException: This exception is thrown when the web driver is unable to interact with an element that is not enabled or not displayed.

StaleElementReferenceException: This exception is thrown when the web driver is unable to interact with an element that has been modified or removed from the DOM after it was located.

TimeoutException: This exception is thrown when the web driver times out while waiting for an element to be located or an action to be performed.

WebDriverException: This is a general exception that is thrown when an error occurs while interacting with the web driver.

- Waits

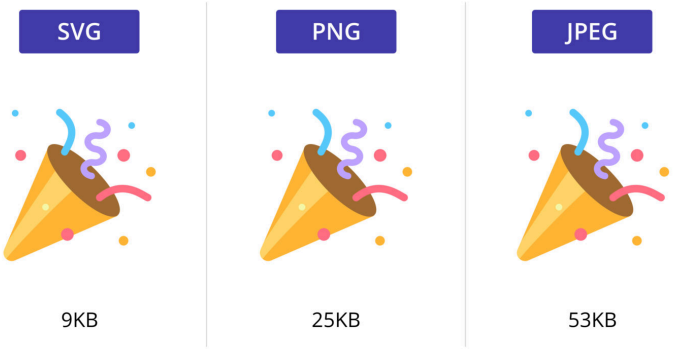
- Try and Except

Misc Scenarios in Selenium

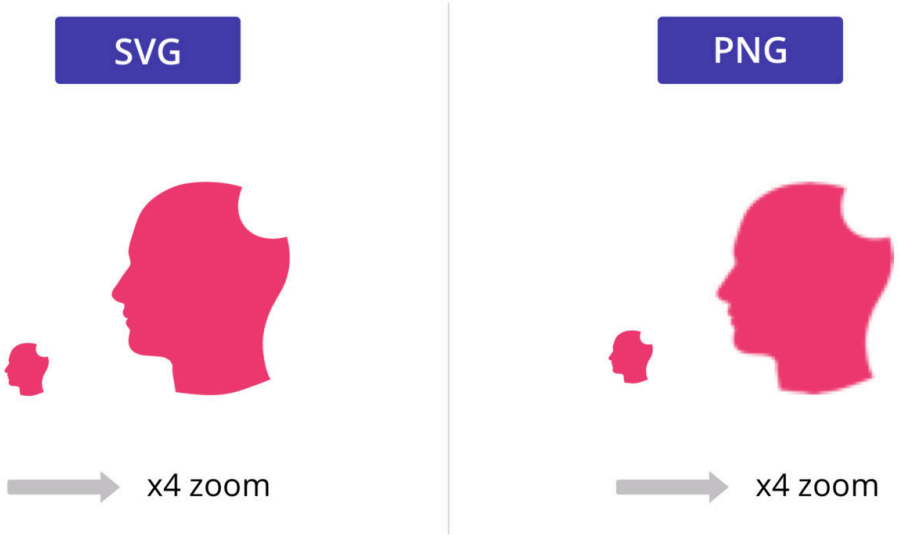
1. Search and Find the text in Web table with pagination.
2. Navigate here - https://codepen.io/templatesio/full/MWqxxog
3. Search “Calvin Golden” on page x.

Handling SVG & Shadow DOM

1. What is SVG - **Scalable Vector Graphics** to define graphics for web.
    a. XML based language to create 2-D graphics/images with animation and interactivity.
    b. Uses geometrical figures to draw an image.



    c. <svg> tag is used as a container for SVG graphics.



2. How to handle SVG Elements in Selenium?
3. How to create XPATH for SVG Elements in HTML DOM?

<svg>
<g>
<circle>
<polygon>
<path>

Your First SVG

https://www.w3schools.com/graphics/tryit.asp?filename=trysvg_myfirst

https://www.amcharts.com/svg-maps/?map=india" - Map is SVG

https://flipkart.com - Search button

SVG Automation Problem - Find the Tripura and Click on It

https://www.amcharts.com/svg-maps/?map=india" - Map is SVG

```
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.action_chains import ActionChains

@pytest.fixture
def driver():
   driver = webdriver.Chrome()
   yield driver
   driver.quit()

def test_svg_demo(driver):
   driver.get("https://flipkart.com")
   driver.maximize_window()
   search_input = driver.find_element(By.NAME, "q")
   search_input.send_keys("AC")

   search_element = driver.find_element(By.XPATH,
"//*[local-name()='svg']/*[local-name()='g' and @fill-rule='evenodd']")
   actions = ActionChains(driver)
   actions.move_to_element(search_element).click().perform()

   driver.get("https://www.amcharts.com/svg-maps/?map=india")
```

```
    states_list = driver.find_elements(By.XPATH,
"//*[name()='svg']/*[name()='g'][7]/*[name()='g']/*[name()='g']/*[name(
)='path']")
    for state in states_list:
        aria_label = state.get_attribute("aria-label")
        print(aria_label)
        if aria_label == "Tripura  ":
            actions.move_to_element(state).click().perform()
            break

if __name__ == "__main__":
    pytest.main(["-v"])
```

Shadow DOM

- Shadow DOM is a web standard that provides encapsulation for DOM and CSS in a web component.
- It allows developers to create encapsulated and reusable UI components.
- Elements inside a Shadow DOM are hidden from the main document's DOM, and the styles defined within a Shadow DOM do not affect the main document's styles, and vice versa.

```
<!DOCTYPE html>
<html>
<head>
    <title>Shadow DOM Example</title>
</head>
<body>
    <my-custom-element>
        #shadow-root
            <p>This is content inside Shadow DOM</p>
    </my-custom-element>
</body>
</html>
```

ScreenShot

- Simple Selenium Method to take screenshot

Relative Locators in Selenium

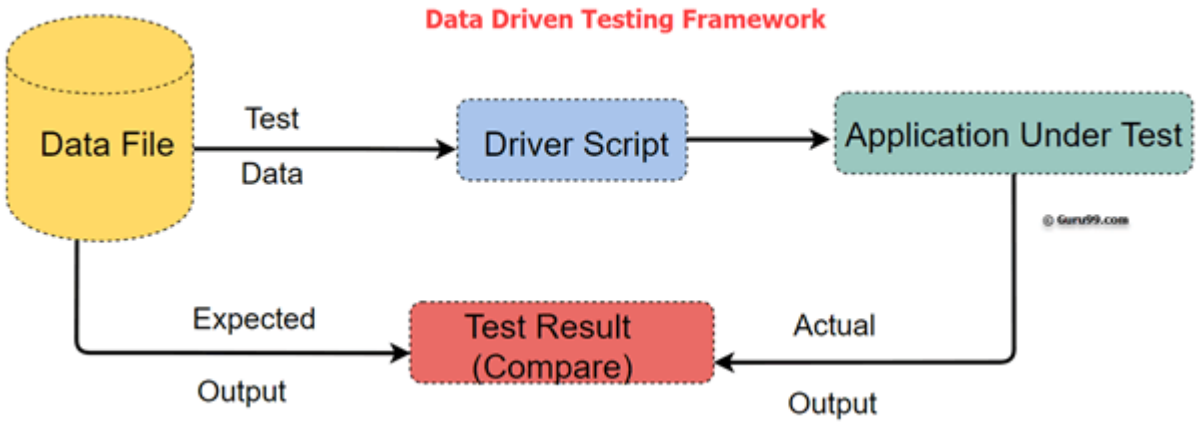
Selenium 4 introduces Relative Locators (previously called Friendly Locators). These locators are helpful when it is not easy to construct a locator for the desired element, but easy to describe spatially where the element is in relation to an element that does have an easily constructed locator.

https://www.selenium.dev/documentation/webdriver/elements/locators/

Relative Locators in Selenium 4
1. above()
2. below()
3. toLeftOf()
4. toRightOf()
5. near()

Data Driven Testing (Apache POI)

- Test data is stored in table or spreadsheet format.
- In data-driven testing, the input data and expected results are created in a table or spreadsheet.
- Data generation can be done by - https://www.mockaroo.com/
- Run your Test cases based on Data.



Valid Email	Valid Password	Valid
Invalid Email	Valid Password	Invalid

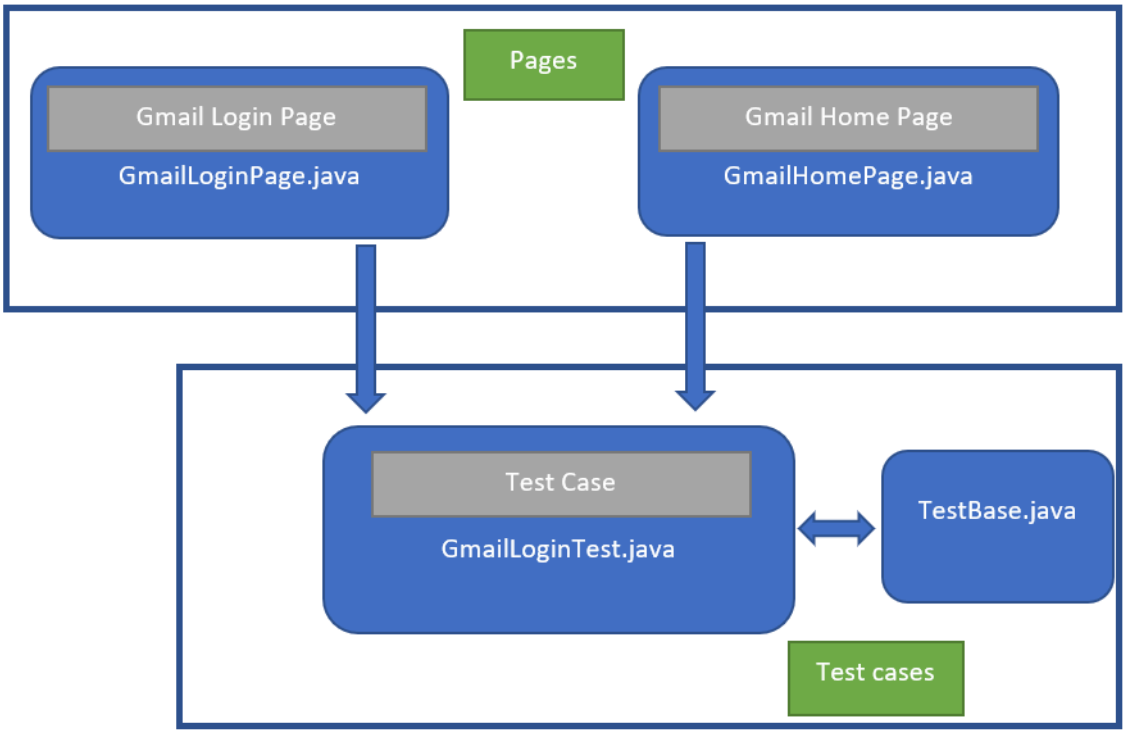
```python
import pytest
from selenium import webdriver
from openpyxl import load_workbook


# pip install pytest openpyxl pytest-excel
def get_test_data():
    workbook = load_workbook("testdata.xlsx")
    sheet = workbook.active
    data = []
    for row in sheet.iter_rows(min_row=2, values_only=True):  # Start from the second row to skip headers
        data.append(row)
    return data

@pytest.fixture
def setup_teardown():
    driver = webdriver.Chrome()
    driver.get("https://app.vwo.com")  # Replace with your website URL
    driver.maximize_window()
    yield driver
    driver.quit()


@pytest.mark.parametrize("username, password", get_test_data())
def test_login(setup_teardown, username, password):
    driver = setup_teardown
    print(username, password)

    # Add your assertions or validation steps here
    # For example, assert that a successful login redirects to the dashboard page

    # Wait for a brief moment to see the action
    driver.implicitly_wait(5)
```

Page Object Model

What is a Page Object Model in Selenium?

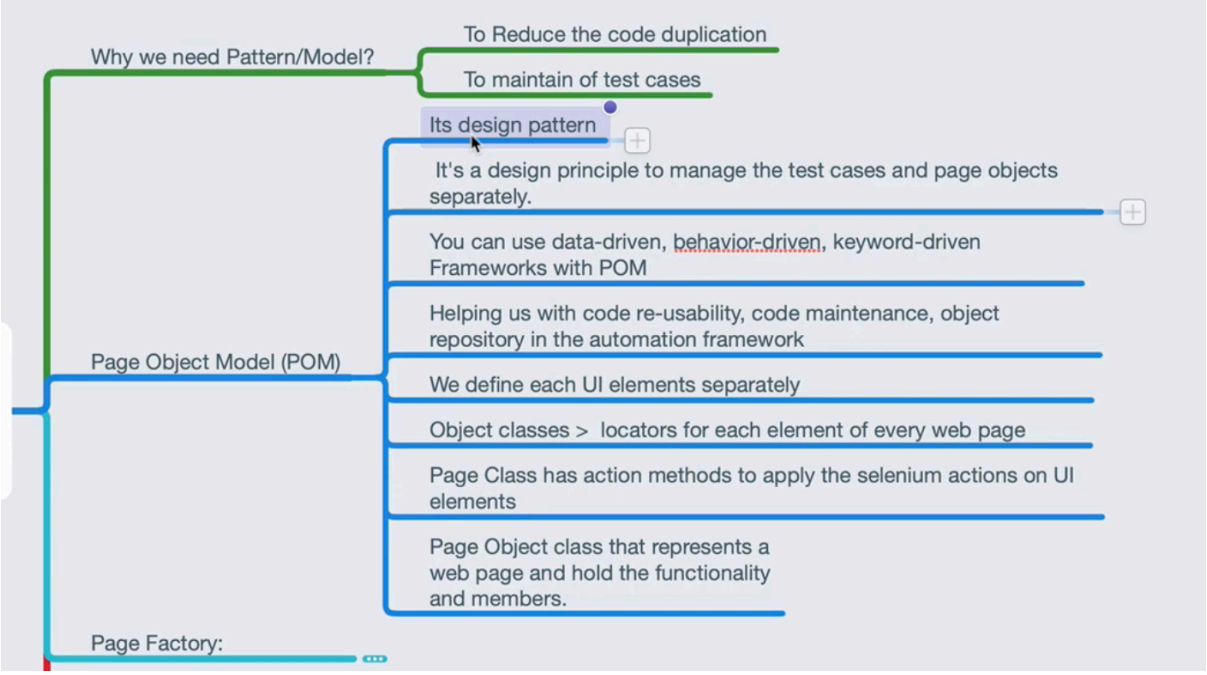
is a design pattern in Selenium that creates an object repository for storing all web elements. It helps reduce code duplication and improves test case maintenance.

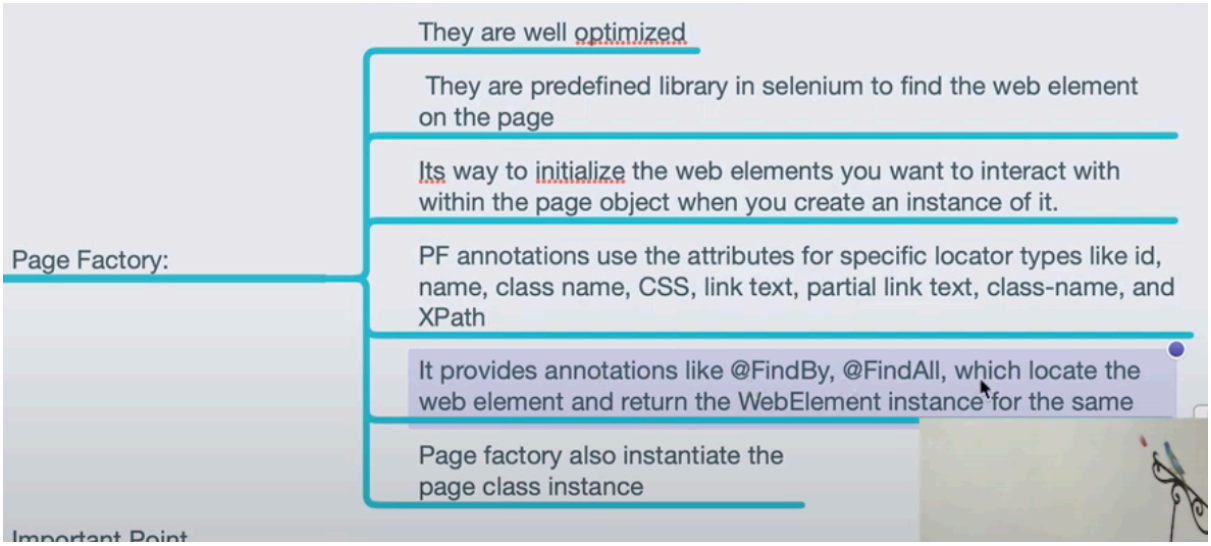
https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/



- Page Object Model in Selenium WebDriver is an Object Repository design pattern.
- Selenium page object model creates our testing code maintainable, reusable.
- Page Factory is an optimized way to create an object repository in the Page Object Model framework concept.
- AjaxElementLocatorFactory is a lazy load concept in Page Factory – page object design pattern to identify WebElements only when they are used in any operation.

PAGE OBJECT MODEL	PAGE FACTORY
It is a class which represents the web page and holds the functionalities	It is a way to initialize the web elements within the page object when the instance is created





POM vs Page Factory

What is the Main difference between Page Object Model and Page Factory in Seleniu…

Selenium Manager (Beta)

- Selenium Manager is a binary generated with Rust that manages driver installation.
- Start the grid with this additional argument: --selenium-manager true
- https://www.selenium.dev/documentation/selenium_manager/

Configuration

Specific values can be overridden by specifying environment variables or by using a config file located by default at `~/.cache/selenium/selenium-manager-config.toml`.

CLI	Env Variable	Config File
–browser chrome	SE_BROWSER=chrome	browser = “chrome”
–driver chromedriver	SE_DRIVER=chromedriver	driver = “chromedriver”
–browser-version 106	SE_BROWSER_VERSION=106	browser-version = “106”
–driver-version 106.05249.61	SE_DRIVER_VERSION=106.0.5249.61	driver-version = “106.0.5249.61”
–browser-path /path/to/chromium	SE_BROWSER_PATH=/path/to/chromium	browser-path = “/path/to/chromium”
	SE_OS=macos	os = “macos”
	SE_ARCH=x64	arch = “x64”
–proxy user@pass:myproxy:8080	SE_PROXY=user@pass:myproxy:8080	proxy = “user@pass:myproxy:8080”
–browser-ttl 0	SE_BROWSER_TTL=0	browser-ttl = 0
–driver-ttl 86400	SE_DRIVER_TTL=86400	driver-ttl = 86400
–clear-cache		

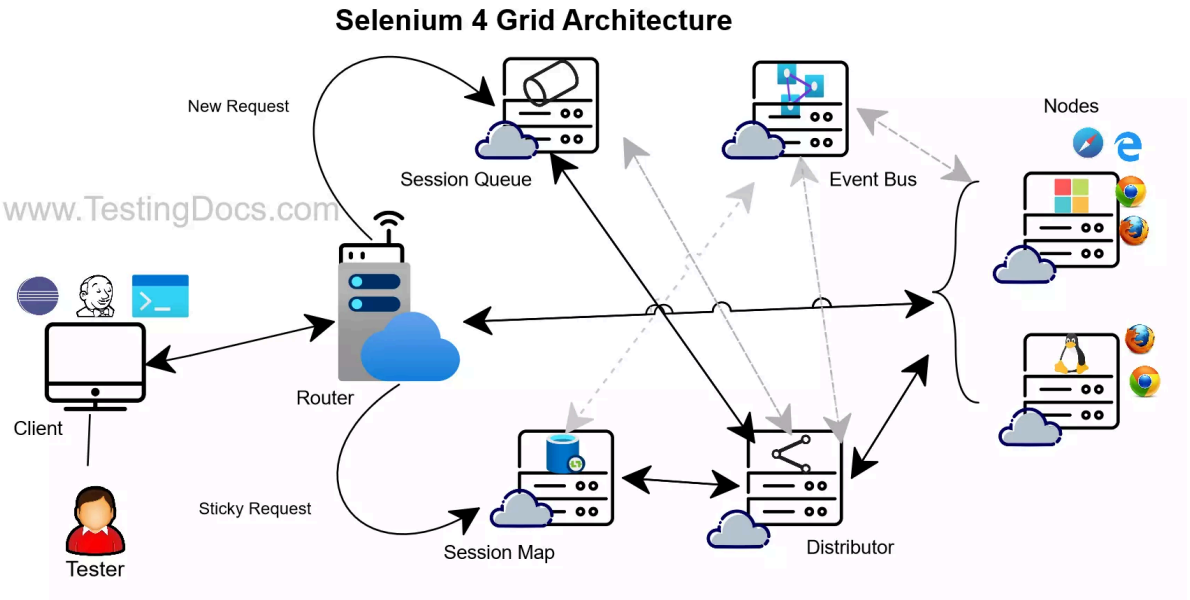
Selenium Grid

- Want to run tests in parallel across multiple machines? Then, Grid is for you.
- Selenium Grid allows the execution of WebDriver scripts on remote machines by routing commands sent by the client to remote browser instances.

Grid aims to:

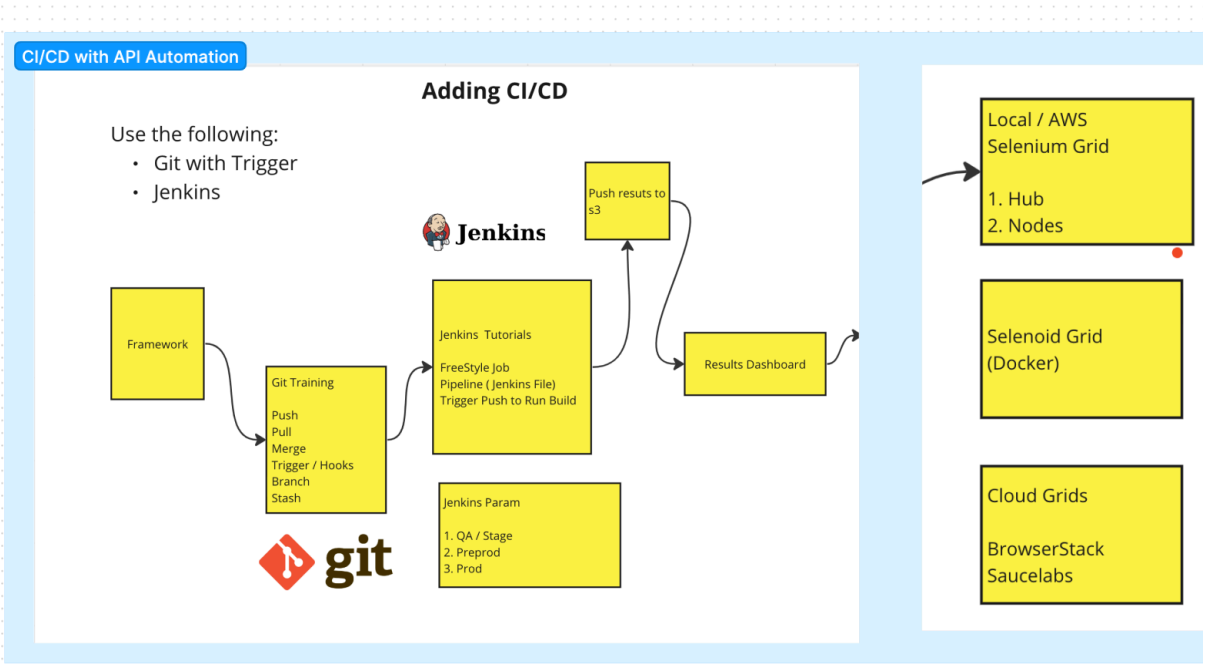
- Provide an easy way to run tests in parallel on multiple machines
- Allow testing on different browser versions
- Enable cross platform testing
- Interested? Go through the following sections to understand how Grid works, and how to set up your own.

Selenium Grid architecture



Selenium Framework

Parallel Test



Reference

1. https://www.geeksforgeeks.org/css-selectors-complete-reference/?ref=lbp
2. https://google.com
3.

Jenkins (Automation Run)

1. Push the code git
2. get the git link - public git
3. install jenkins
4. create a freestyle job - new item
5. SCM -> git link repo
6. Build
7. Report -> Add a plugin Allure report, HTML Report Plugin configration.

On Windows

```
set path="C:\Users\sikhi\AppData\Local\Programs\Python\Python312"
set path="C:\Users\sikhi\AppData\Local\Programs\Python\Python312\Scripts"
pip install -r requirements.txt
pytest tests\integration_test\test_crud.py
```

On Mac
cd "/Users/pramod/.jenkins/workspace/Python 1x Automation"
pip3 install -r requirements.txt
/Library/Frameworks/Python.framework/Versions/3.9/bin/pytest tests/integration_test/test_crud.py -s -v --html=report.html --alluredir=./reports

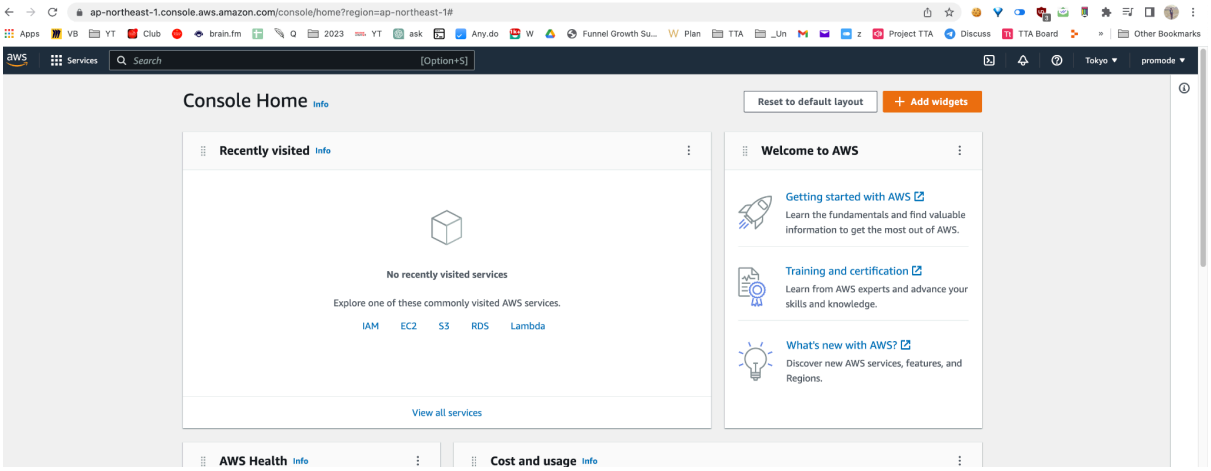
By using Virtual Env

/opt/homebrew/bin/python3 -m venv .
source ./bin/activate
pip install allure-pytest selenium pytest selenium-page-factory pytest-html openpyxl pyyaml faker openpyxl pytest-xdist python-dotenv
pytest tests/test/vwoLoginTests/test_vwo_login_pf.py
deactivate

AWS Basics and Running Selenium Grid

- We will be using AWS for the Jenkins or Selenium Grid Install.
-

1. Create a Free tier account https://aws.amazon.com/free/
2. You will be logged in to the AWS Account



Install Jenkins in AWS

Step - 1 Install Java

Update your system

sudo apt update
Install java
sudo apt install openjdk-11-jre -y
Validate Installation

java -version
It should look something like this

openjdk version "11.0.12" 2021-07-20 OpenJDK Runtime Environment (build 11.0.12+7-post-Debian-2) OpenJDK 64-Bit Server VM (build 11.0.12+7-post-Debian-2, mixed mode, sharing)
Step - 2 Install Jenkins

Just copy these commands and paste them onto your terminal.

curl -fsSL https://pkg.jenkins.io/debian/jenkins.io.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \ https://pkg.jenkins.io/debian binary/ | sudo tee \ /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins

Step -3 Start jenkins

sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins
Step - 4 Open port 8080 from AWS Console:

Edit for Port - /etc/default/jenkins

How to find Java Location?

readlink -f $(which java)

Running Selenium Test cases on Selenium Grid

1. Install Selenoid
2. Install Docker

Sudo su // FOR SU user
Apt-get update
apt install docker.io -y
sudo systemctl status docker
sudo systemctl start docker

sudo wget "https://github.com/aerokube/cm/releases/download/1.8.1/cm_linux_amd64"

sudo chmod +x cm_linux_amd64
sudo ./cm_linux_amd64 selenoid start –vnc

./cm_linux_amd64 selenoid-ui start
./cm_linux_amd64 selenoid-ui stop