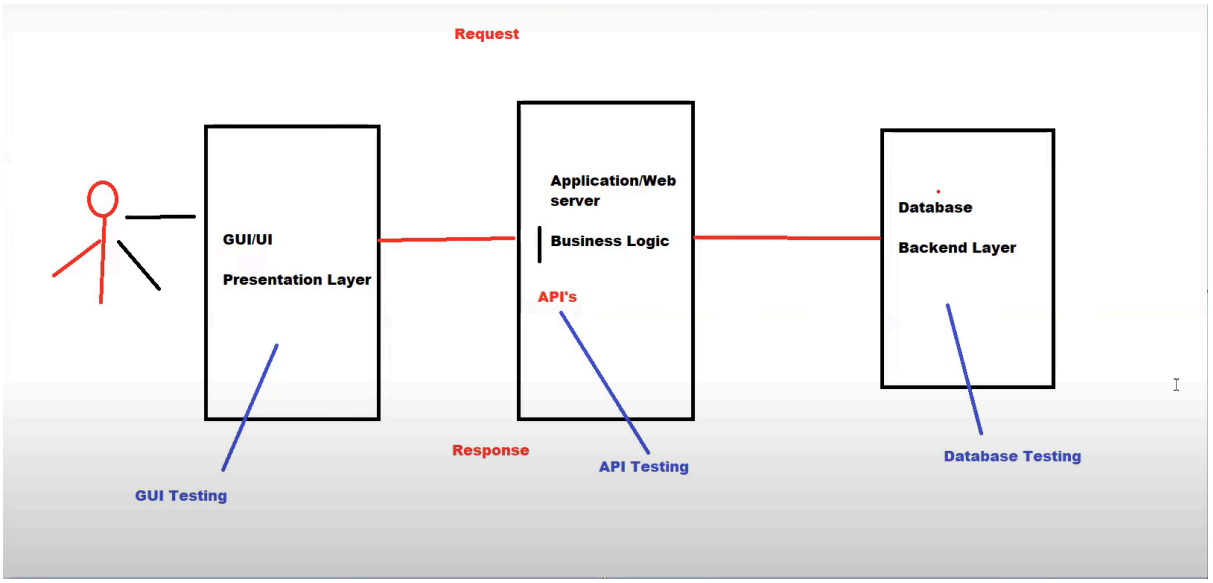
Selenium WebDriver

Course URL

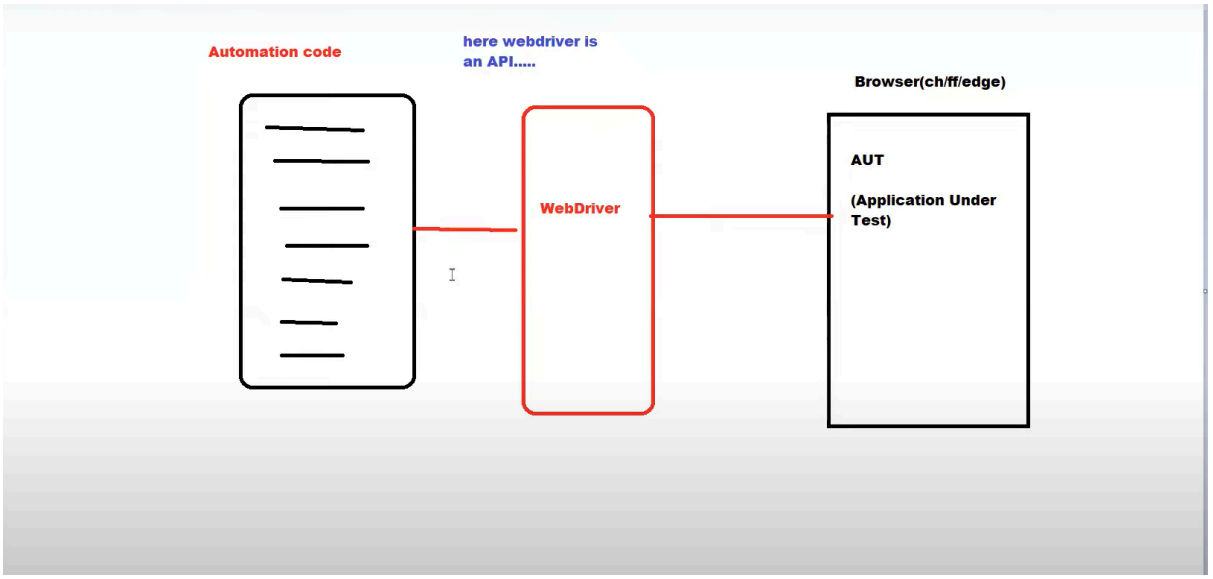
What is WebDriver?
→ WebDriver is a module in python.
** There are so many things in selenium in that webdriver is one part.
→ WebDriver is one of the component in selenium.

These are the different classes which are used to run selenium on different browsers.
Firefox browser ------> Firefox()
Chrome browser ------> Chrome()
Edge ------> Edge()



→ WebDriver is an API (Application Programming Interface)

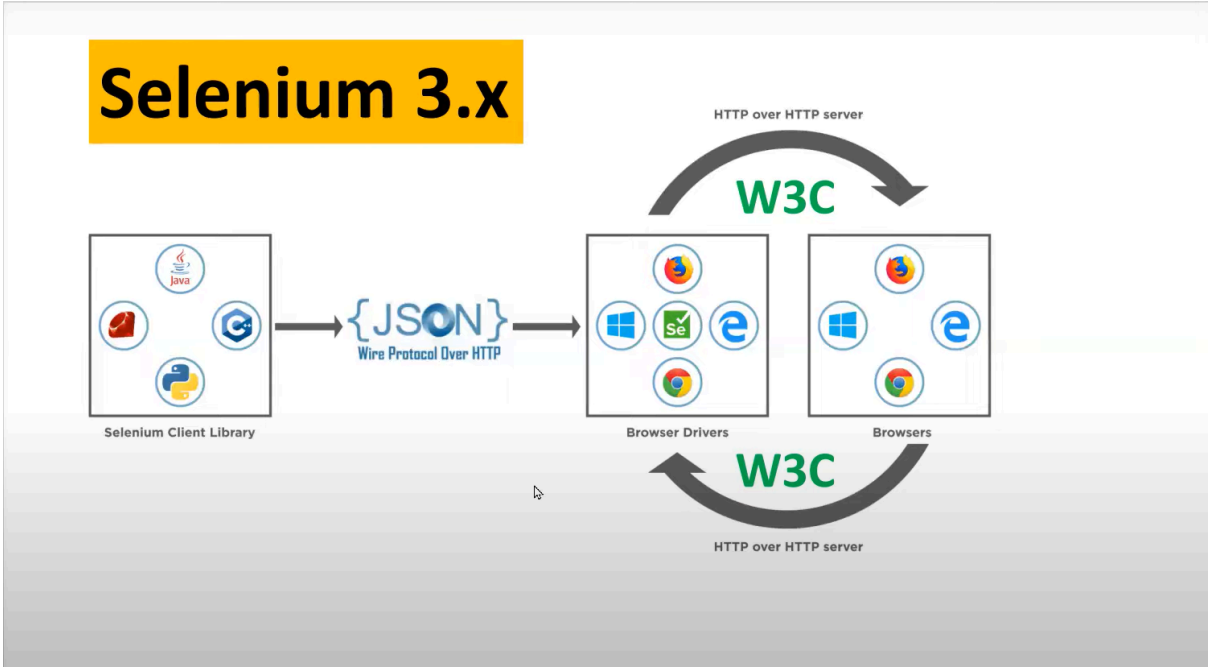
- Here the webdriver acts as an API(bridge) between our Automation code and Browser.
- It will read and execute our Automation code programs and perform the action on the browser (GUI interface).



Selenium Architecture:

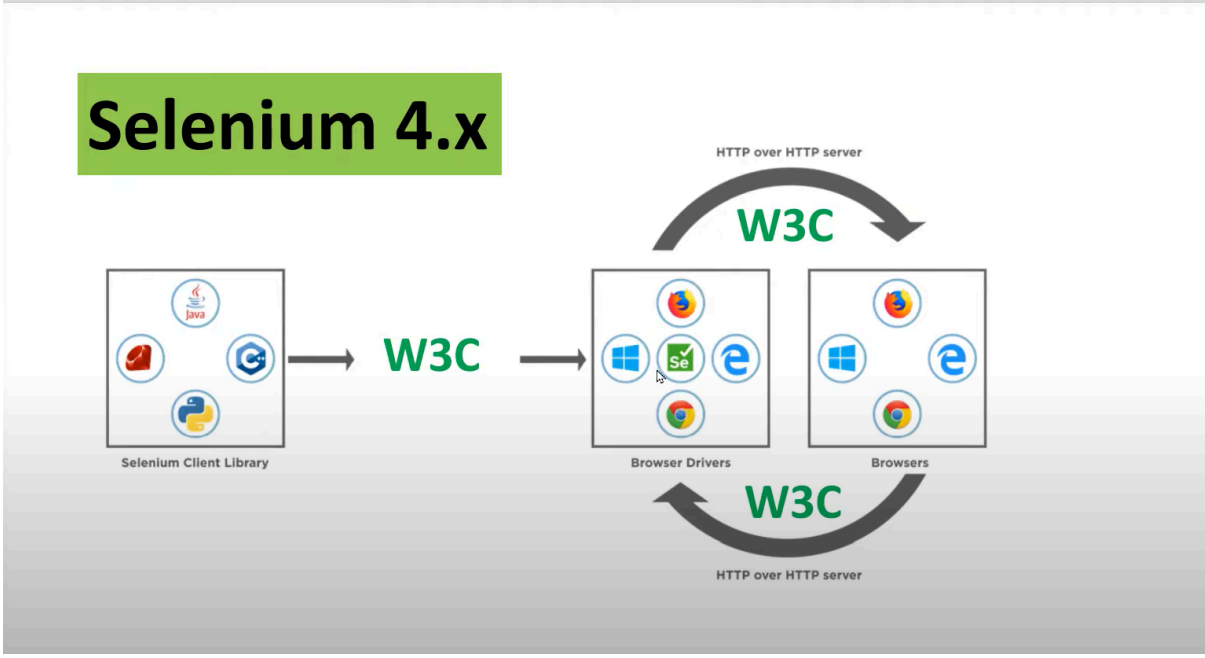
There are two types of selenium versions Selenium 3.x & Selenium 4.x

- Selenuim 3.x



  - Selenium Client Library : Here client library means the package in our programming language like (import selenium) package.
  - Browser Drivers : Browser driver is just a .exe file which is used to communicate with our browser automatically using our automation code.
  - Browser : It's a normal browser in our pc, this Browser and Browser Driver communicate each other using W3C protocols.
  - JSON : The selenium Client library and our Browser can't communicate directly there is a protocol which is called JSON(Wire Protocol Driver HTTP) which acts like a bridge between them.

Selenium 4.x



- As there are two types of protocols involved JSON and W3C in selenium 3.x , many issues like code will stop working suddenly came into picture. So to reduce these issues selenium 4.x was developed where the Selenium

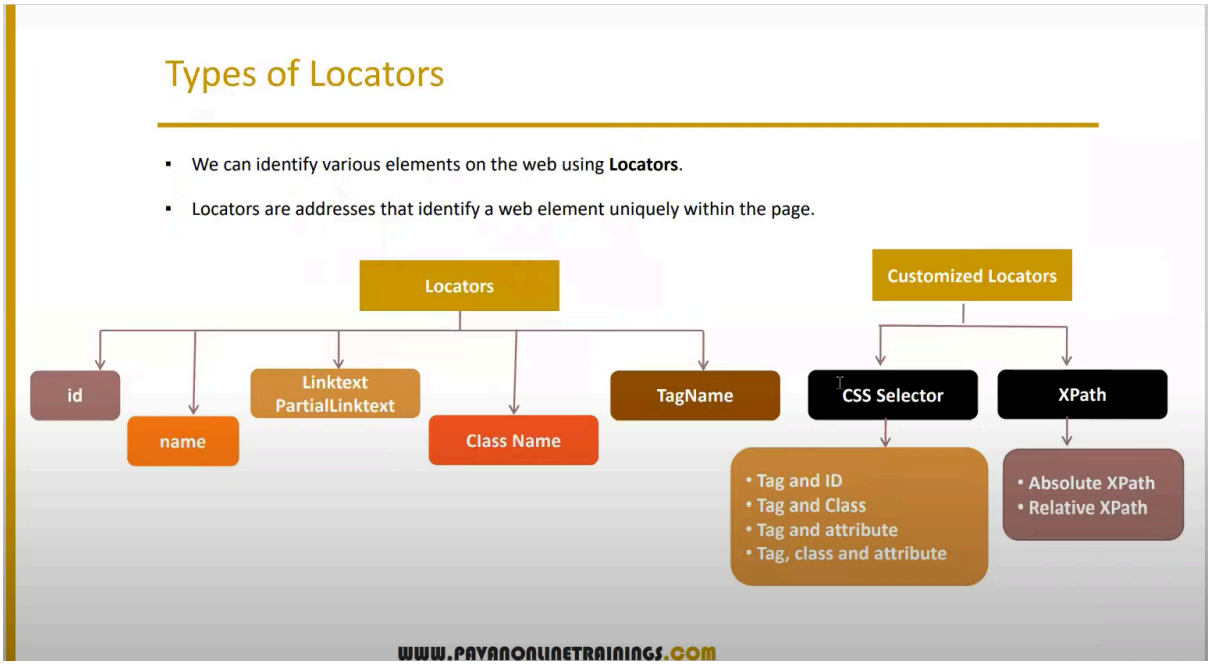
- Client Library — ” W3C” – WebDriver — ” W3C” – WebBrowser uses same protocol “W3C”.

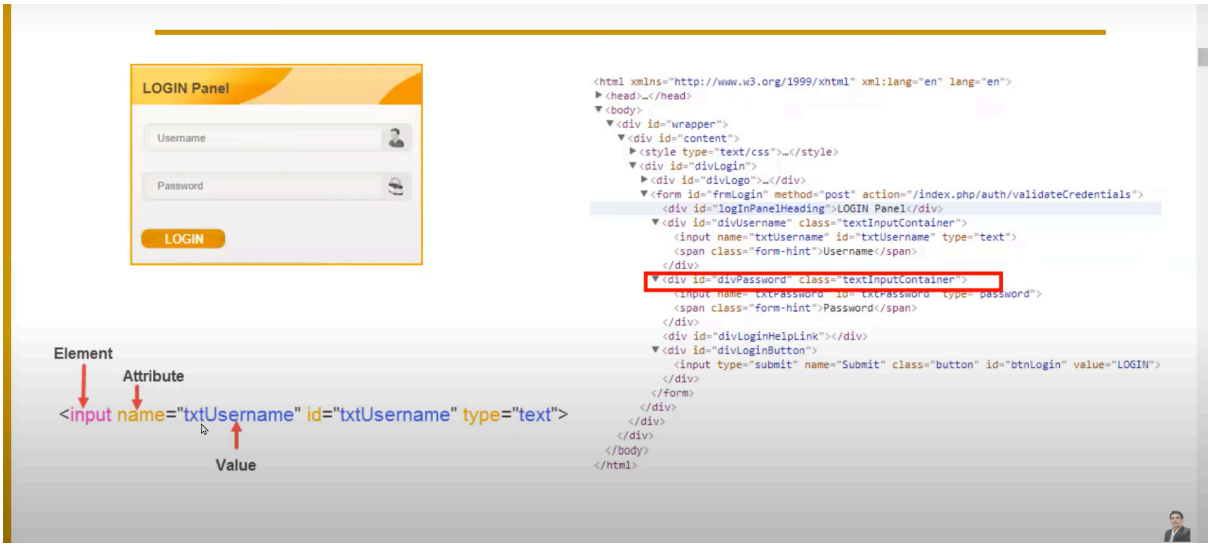
Episode-1:

Refer → FirstTestcase.py file in "SeleniumUpdatedcourse project"

1) Open web Browser.
2) open url https://opensource-demo.orangehrmlive.com
3) Enter username (Admin)
4) Enter password (admin123)
5) Click on login
6) Capture the title of the home page. (Actual title)
7) Verify the title of the page: OrangeHRM (Expected)
8) Close browser

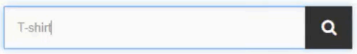
Day - 2 : Locators





ID

http://automationpractice.com/index.php

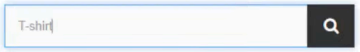



```
<input class="search_query form-control ac_input" type="text" id=
"search_query_top" name="search_query" placeholder="Search" value
autocomplete="off"> == $0
```

driver.find_element(By.ID,"search_query_top").send_keys("T-shirt")

Name

http://automationpractice.com/index.php




```
▶<button type="submit" name="submit_search" class="btn btn-default
button-search">…</button> == $0
```

driver.find_element(By.NAME,"submit_search").click()


```
<a class="product-name" href="http://automationpractice.com/
index.php?id_product=7&controller=product" title>
                    Printed Chiffon Dress
                </a> == $0
```

driver.find_element(By.LINK_TEXT,"Printed Chiffon Dress").click()
driver.find_element(By.PARTIAL_LINK_TEXT,"Chiffon Dress").click()

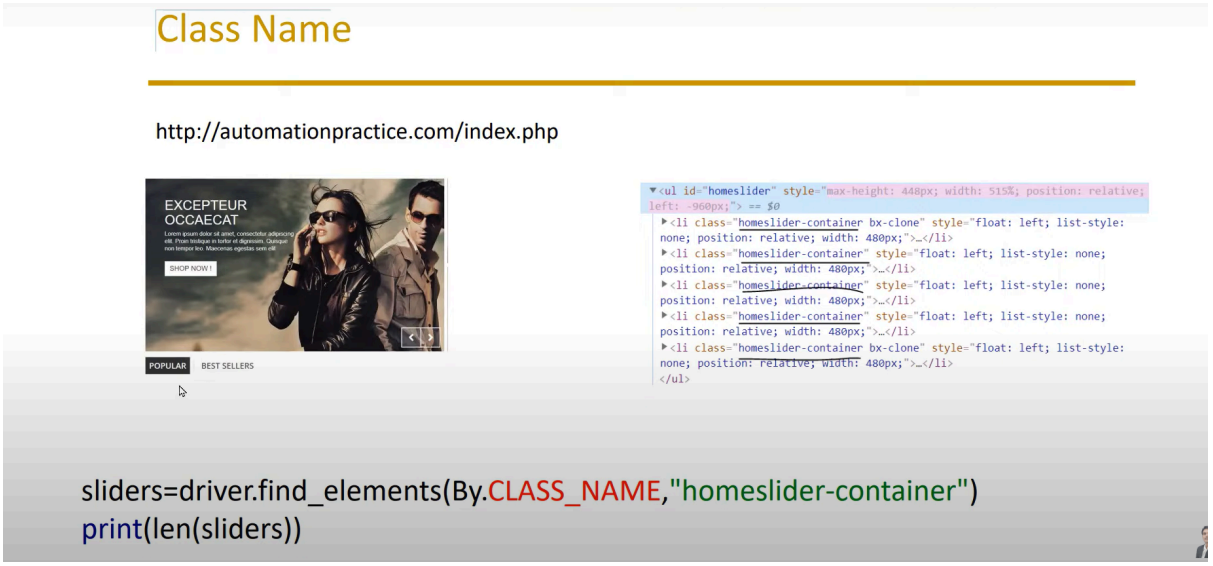
Interview Question: What is the difference between LINK_TEXT and PARTIAL_LINK_TEXT?

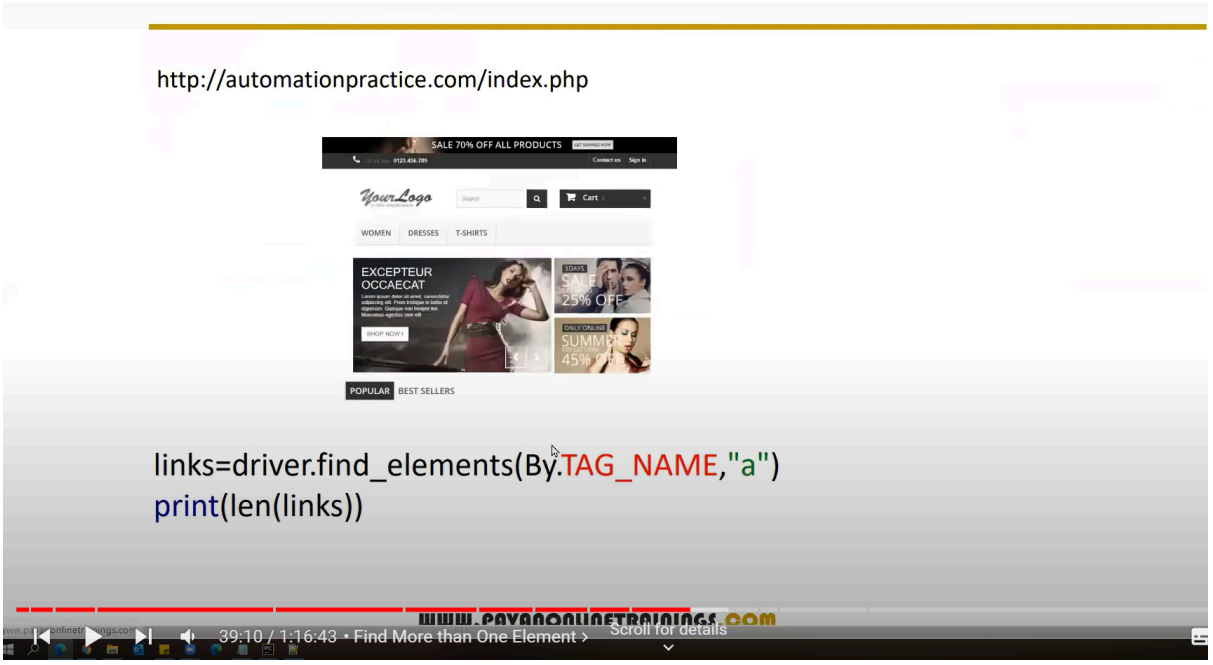
In LINK_TEXT we have to give the proper link text which is in between <a>potato vegitable tag.
In PARTIAL_LINK_TEXT we have to give the partial link text like <a>potato.

Id
Name
Link text
Partial link text (All these elements will contain single values)

→ IF WE WANT TO GET MULTIPLE VALUES FROM THE WEBSITE THEN WE HAVE TO USE

Classname
Tagname





Interview Question: How can you find the multiple elements in a webpage?

We have to use the find_elements(By.CLASS_NAME) or find_elements(By.TAG_NAME) locators to find multiple elements in a page.

CUSTOMIZED LOCATORS:

1. CSS selector
2. XPATH

1 . CSS Selectors types: Syntax EXAMPLE

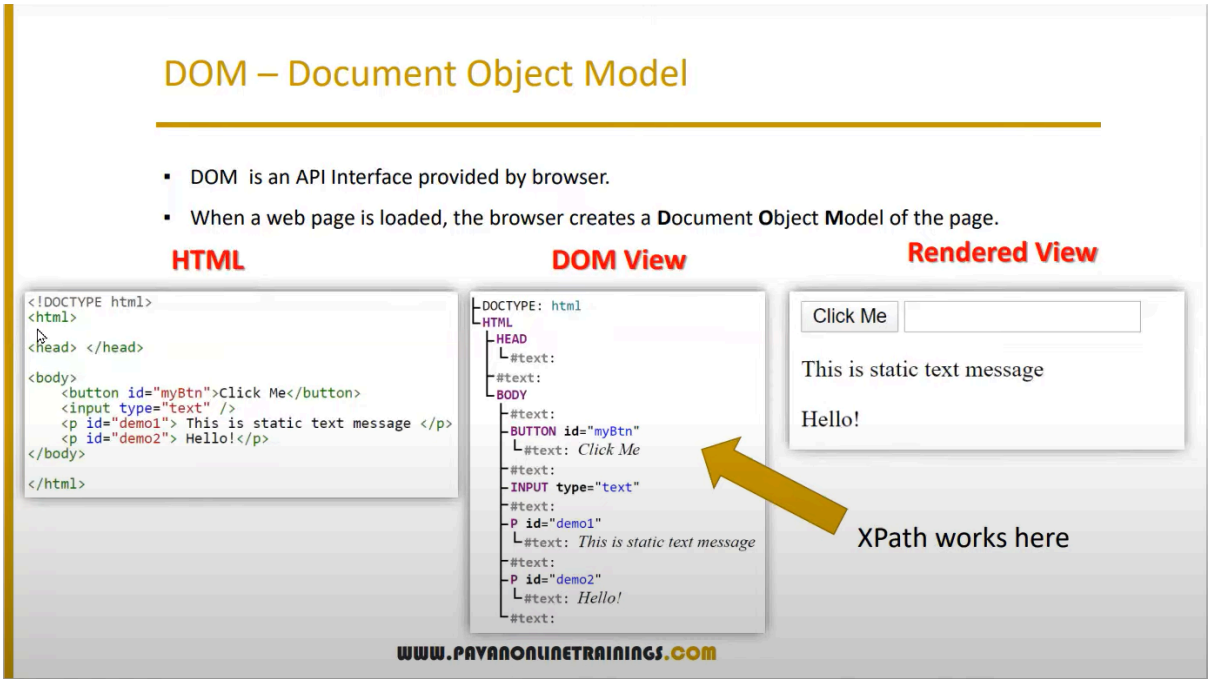
1. Tag - ID tagname#Id EX : “input#email”
2. Tag - CLASS name tagname.valueOfClass EX : “input.`inputtext`
3. Tag - Attribute tagname[Attribute=value] EX : “input[`data-testid=”royal_email”]`
4. Tag - class Attribute tagname.ValueOfClass[Attribute=value]
   EX : “input.inputtextt[`data-testid=”royal_email]”`

   #### Refer CSS-Locators.py file in the Locators folder of the project.

   ** Note: In class names (tagname.valueOfClass) it will not consider spaces , So you have to delete all letters after first space.

2. XPATH :

- XPath is defined as an XML path.
- It is a syntax or language for finding any element on a webpage using an XML expression.
- XPath is used to find the location of any element on the webpage using DOM (Document Object Model) structure.
- XPath can be used to navigate through elements and attributes in DOM.
- XPath is an address of an element.



TYPES OF XPATH :

1. Absolute / Full XPath.
   Ex : /html/body/nav/div/div[2]/div[1]/div/ul/li[2]/a

2. Relative / Partial XPath.
   Ex : //*[@id="navbarSupportedContent"]/div[1]/div/ul/li[2]/a

Diff between Absolute & Relative XPaths : (**Imp question)

---

1) Absolute xpath starts from the root html node.
   Relative xpath directly jumps to the element on DOM.

2) Absolute xpath starts with /
   Relative xpath starts with //

3) In Absolute xpath we use only tags/nodes.
   In Relative xpath we use attributes.

How to write XPaths manually :

—----------

/html/body/div[1]/div[3]/form/div[1]/div[1]/div[4]/center/input[1] → Absolute XPath

Relative XPath :

//tagname[@atttribute='value']

Ex 1 : //input[@value='Google Search']

For example, if you don't know the tag name then you can write " * " where it will filter out all the elements in the page with the given attribute=value.

Ex 2 : //*[@value="Google Search"]

How to capture XPath automatically :

1) Right-click on the element → inspect → Copy → Copy XPath.
2) Use the "SelectorsHub" browser extension.

Interview question :

Which XPath is preferred to use normally?

Ans: Generally we use Relative XPath only, in rare cases we will use Absolute XPath.

Why is Relative XPath preferable? → Because Relative XPath is more stable than the Absolute Xpath as we use specific attributes to locate an element.

On the other side in Absolute XPath, we have to filter out each and every element in the webpage.

1) If the developer introduces a new element then our old Absolute Xpath will be broken.
2) If the developer changed the location of our desired element then the Absolute XPath will be broken.

Because of these reasons, Relative XPath is preferable.

XPath options :

——--

and

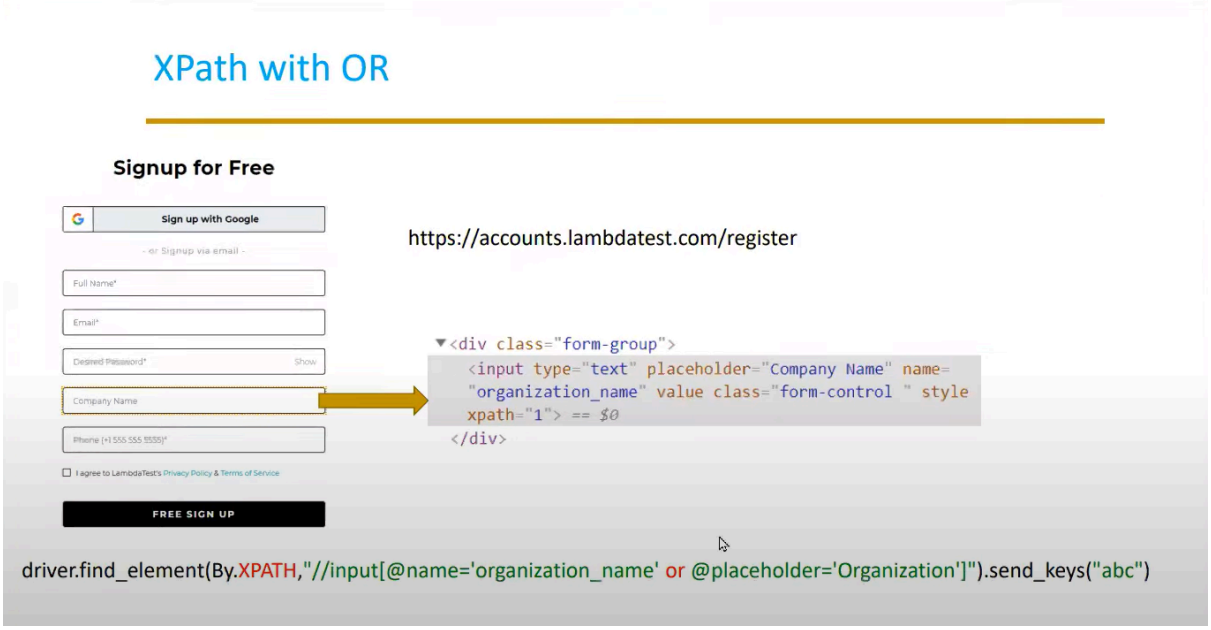
or

contains() → Ex : contains(@id,'email')

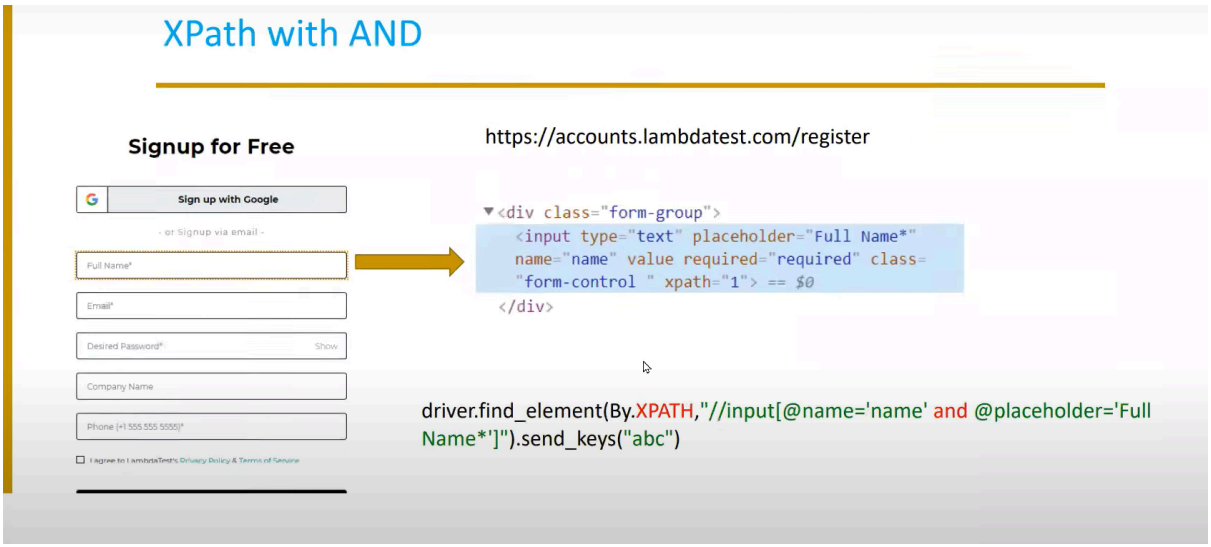
starts-with()

text()

OR :



AND :




```
driver.get("https://accounts.lambdatest.com/register")
```

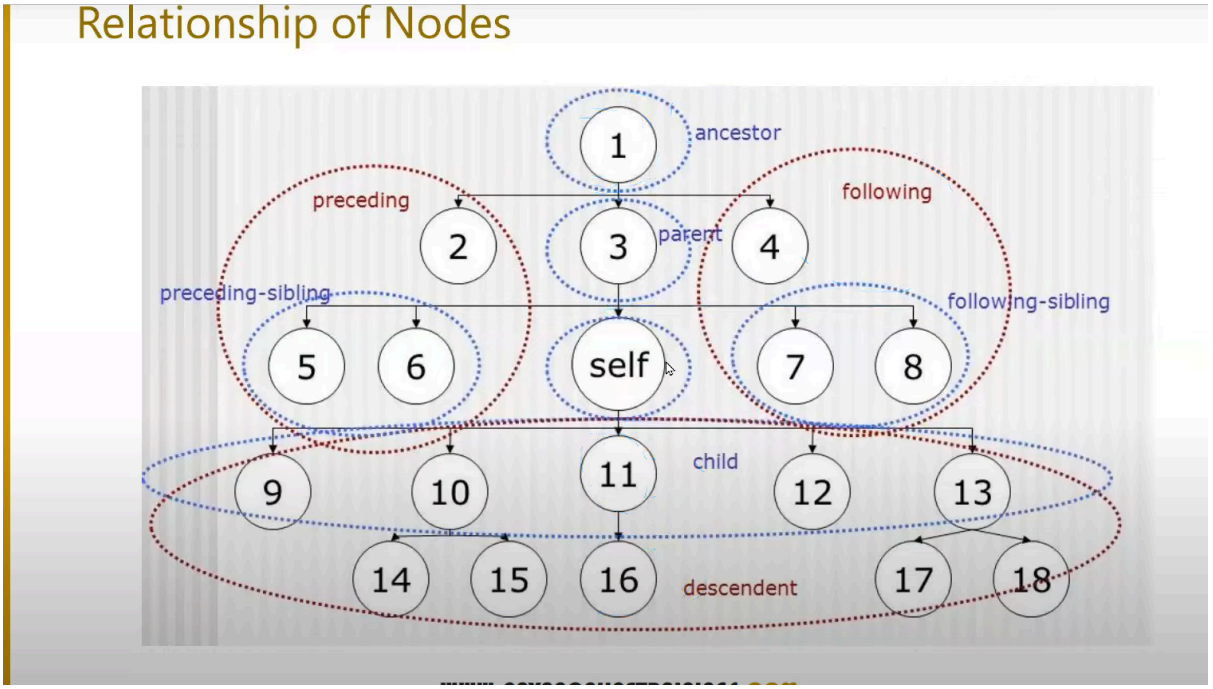
```
#contains()
driver.find_element(By.XPATH,"//*[contains(@id,'email')]").sen
d_keys('Charanteja@gmail.com')

#starts-with()
driver.find_element(By.XPATH,"//*[starts-with(@name,'name')]")
.send_keys('Charanteja')

#text()
driver = webdriver.Chrome()
driver.get("https://www.amazon.in/")
driver.find_element(By.XPATH,"//a[text()='Mobiles']").click()
```

Refer XPath Demo.py file in the pycharm

XPath Axes : (Session 4)



Types :

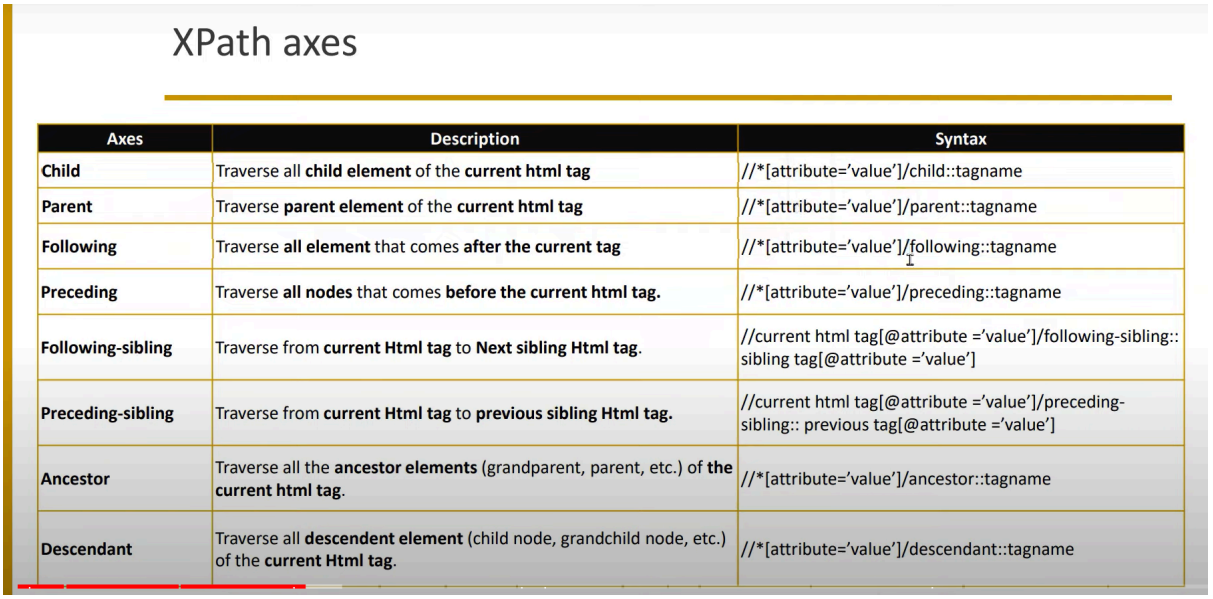
Self
Parent
Ancestor
Descendant

Following
Following-sibling

Preceding
preceding-sibling

**(Note) Self element : `"//a[contains(text(),'APL Apollo Tubes')]"`

We can consider any element in our web page as a self element(center element) (normally we will use a element which has attributes, which helps us to find out the element easily) basing that element in our web page we will find the other elements in our web page by using this parent,child,ancestor relations.



XPath axes

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

The website used for practice → https://money.rediff.com/gainers/bse/daily/groupa

Continue part 5 video from the begining ……..!

```
#self

text_msg =
driver.find_element(By.XPATH,"//a[contains(text(),'APL Apollo
Tubes')]/self::a").text
print(text_msg)
```

**** Different Types Of Commands Used In Selenium :
(Verify → Different_commands.py file)

1) Application commands
2) Conditional commands
3) Browser commands
4) Navigational commands
5) Wait commands

1) Application commands :
---
get() - Opening the application URL
title - To capture the title of the current webpage
current_url - To capture the current URL of the webpage
page_source - To capture the source code of the page

2) Conditional commands :
---
is_displayed()
is_enabled()
is_selected()

3) Browser commands
---
close() - Closes the single browser window only (where the driver focused) - it will close the parent window.
quit() - Closes the multiple browser windows (this will kill the process)

A window handler is used to close the selected browser window.

Diff between close() & quit() – Interview question :

When a user opens a web browser many web browser-related processes are automatically run by our OS.

When you use the close command it will close the window only but the background process still keeps on running.
But when you use the quit command it will kill (close) the entire Chrome web driver and its process that's the difference.

4) Navigational Commands :
---

back()
forward()
refresh()

**** Difference between find_element() Vs find_elements() methods
INTERVIEW QUESTION

(Refer find_element() Vs find_elements().py file)

→ find_elements() will return the list of web elements even though it's pointing to a single web element.
→ It will store in list format so that we can access the elements using indexing
Ex: element[0]

→ In the find_element() method if an element is not found it will return "`NoSuchElementException`" but in the find_elemenst() method it will return zero but not an error.

**** text Vs get_attribute('value') refer :TexttVsAttribute.py file

text ----- > returns the inner text of the element.
get_attribute() → returns the value of any attribute of the web element.

<input id='123' name='xyz'> Email: </input> —> Email: is the inner text
Ex: get_attribute('id') #returns 123

5) Wait Commands :

**** What is a **synchronization** problem?

Normally our automation script will execute faster than our application response and it will lead to throwing an error so to overcome this issue we have to use wait commands in Selenium.

If we use time.sleep() command →

Adv :
---
1) Simple to use

Dis-Adv :
---
1) Performance of the script is very poor.
2) If the element is not available within the time mentioned, still there is a chance of getting an exception.

1) Implicit wait(): If any error occurs in any line it will wait for the mentioned time and then execute the remaining statements or else it will throw an error.

We have to use it just after calling the driver's statement
Ex: `driver.implicitly_wait(10)`

Adv :
---
1) Single statement.
2)Performance will not be reduced(If the element is available within the time it proceeds to execute further statement).

Dis-Adv :
---
1) If the element is not available within the time mentioned, still there is a chance of getting an exception.
2) In this case, we have to use try-except statements.

2) explicit wait() :

Explicit wait works based on a condition.

Adv :
---
1) It will work more effectively because it has an inbuilt Exception handling mechanism.

Dis-Adv :
---
1) we have to implement it in Multiple places so people will find it difficult when we compare it with Implicit wait.

```
mywait =
WebDriverWait(driver,10,poll_frequency=2,ignored_exceptions=[N
oSuchElementException,

ElementNotVisibleException,

ElementNotSelectableException,
                                                    Exception]
                        )
```

** We have to use it just after calling the driver's statement, here 10 is the timeout, and “Poll_frequency” is the number of times the wait command has to fetch the element before the actual timeout.

Note: “Poll_frequency” should be less than the actual timeout seconds.

** Then we will declare our customized wait condition until we find a particular element.

```
element =
mywait.until(EC.presence_of_element_located((By.XPATH,"//a[tex
t()='orange HRM']")))

element.click()
```

** How to work with different elements in a webpage :

1) Checkboxes → refer handleCheckBox.py

2) Links :

   Internal links: Navigates in the same webpage. Refer handleLinks.py
   External links: Navigate to some other webpage.
   Broken link: Link that doesn't have any target. Refer handleBrokenLinks.py

3) Dropdown :

   Practice website: https://testautomationpractice.blogspot.com/

Alerts / Popups : refer Alerts.py file

Myalert = driver.switch_to_alert()

Myalert.text
Myalert.accept()
Myalert.sendkeys('Charan')
Myalert.dismiss()

**Interview qn: How to handle alerts?

By using the " **switch_to_alert()** " command we can navigate to the alert and by using the accept() and dismiss() methods we can click on " OK " or " Cancel " in the alerts.

Authenticated popups:
---

1) In some high-security websites companies will develop websites that will ask for your username and password in the alert window, when you try to open the website.
2) As the username and password are in the alert window we can't enter the input fields using the " send keys () " function, even if we switch to the alert using switch_to_alert() we can enter only one field.
3) So in this case we just have to bypass the alert window by injecting the username and password into the URL of the website.

Actual-URL: https://the-internet.herokuapp.com/basic_auth

Syntax: https:// username : password @abcdt.com

Updated - URL:
`https://admin:admin@the-internet.herokuapp.com/basic_auth`

** Interview Qn: **How to handle the Authenticated popups?**

→ We have to inject our username and password through the URL of the webpage and then we can bypass it.

Frames/iframes: (refer frames.py file)

Frames: A Frame is a particular section in a webpage that displays data from another webpage.

- Sometimes our webpage contains many sections (frames) that are from other sources, **for example**: a Restaurant website contains a map frame which is embedded in it.

- So we can't access the elements present in the frames directly, first, we have to navigate into the frame then we have to identify the element in that particular frame.

We can use frames in these four scenarios.

1. switch_to.frame(name of the frame)
2. switch_to.frame(id of the frame)
3. switch_to.frame(webelement)
4. switch_to.frame(0)

#To come back from frame Switch_to.default_content()

You can use any of these tags as frames

- Frame
- iframe
- form

Switch between browser windows :

- Refer Window_handles.py file in the updated Selenium course.

- notifications.py: Sometimes we will get some alert link “Allow notification Access” which is sent by our browser.

  So we can’t use either the switch to commands or bypass the alert by sending the credentials in the URL for this kind of alert, we have to disable the Chrome options. Refer to notification_popup.py file.

Web tables :

- Refer **Webtable.py** file to learn how to get data from tables

+ To loop the data from a table in the selenium we have to pass the variable into the Xpath in a different syntax as the XPath will not allow direct variables.

Example 1:

```
data =
driver.find_element(By.XPATH,"//table[@name='BookTable']//tr[3
]/td[3]").text
```

Example 2 (Actual Xpath to loop the elements) :

```
d =
driver.find_element(By.XPATH,"//table[@name='BookTable']//tr["
+str(r)+"]/td["+str(c)+"]").text
```

Date Picker :

There are two types of elements in a webpage.

1) Standard elements (buttons, links, etc.,).
2) Non-standard elements (Customized elements like date pickers).

For the Non-standard kind of date pickers, we have to build our logic to pick the date from the date picker which is actively displayed in the browser.

Refer **datepicker.py** file for more reference

1) Here we have to compare our date, time, and years with the actual date which is shown in the datepicker in the browser.
2) If our date does not match, then we have to keep clicking the next button until we find our required date.

Mouse Operations :

- **Action Chains** **(**Imp for interview)**

1. Mouse hover → `move_to_element(element)`
2. Right click → `context_click(element)`
3. Double click → `double_click(element)`
4. Drag and drop → `drag_and_drop(source,target)`

5. **Sliders : (VvImp for interview)**

- To learn about sliders first, we have to know about the X-axis and Y-axis.

- Every element in our webpage has a location that includes both the axis, we can also find the location of the element using the **(.location)** method.
- So, to move the slider first we have to find the actual X-axis location and then we have to increase the axis location by using -
- `action.drag_and_drop_by_offset(min_slider,100,0).perform()`

Now we have to move the sliders by increasing/Decreasing the X-axis location,
as we are moving the slider only in the X-axis,
here we can ignore the Y-axis and give the value 0 in the offset-method.

Slider → **`action.drag_and_drop_by_offset(min_slider,X,Y).perform()`**

Scrolling in a window :

Sometimes the element that we need is somewhere below the page, to identify that we have to scroll the webpage automatically using selenium.

- There are 3 different ways to scroll the page in Selenium:

1) Scroll down the page using pixel.

```
driver.execute_script("window.scrollBy(0,2000)","")
```

2) Scroll down the page till the element is visible.

```
flag =
driver.find_element(By.XPATH,"//a[contains(text(),'India'
)]")
driver.execute_script("arguments[0].scrollIntoView();",fl
ag)
```

3) Scroll down the page till the end.

```
4) driver.execute_script("window.scrollBy(0,document.body.sc
   rollHeight)")
```

Action chains using Keyboard shortcuts:

- Refer KeyboardActions.py file in the course.

TO capture screenshot of a webpage :

Use `driver.save_screenshot(folderpath CWD + image name)` method to take screenshot of a webpage.

How to open a link in a new tab in Selenium:

Refer Tabsandwindows.py file in the course

How to Handle Cookies:

Cookie: Cookies are the temporary files that are created by our browser when we search for something.

For example:

- If we log in to an e-commerce website and are given our email and password, it will suggest you to save our email and password in the browser to log in next time directly.

- These credentials are saved in our browser in the form of cookies, also if we search for something in the e-commerce website next time it will suggest the same kind of products i.e our browser is using the cookies to save your data which will be useful for next visit to the same website.

Refer HandleCookies.py file in the course

Headless mode testing: **VImp** Refer to **HeadlessTesting.py** file
---

- Headless mode means without getting any UI we are still able to execute our test cases.
  There are a few advantages and disadvantages as well
  Advantages :
  We can run our browser and script in the background without interrupting the desktop which will also save us time.
  Disadvantages :
  As we are running it in the background we are unable to see the UI of the webpage and not able to identify the actual error properly.

DataDriven Testing :

Sometimes our test cases need some data to execute the test case **like** entering the details of the student in a college website, where all the data is already saved in an Excel sheet we just have to use the same data to store in our website, these process we can call as DataDriven testing.

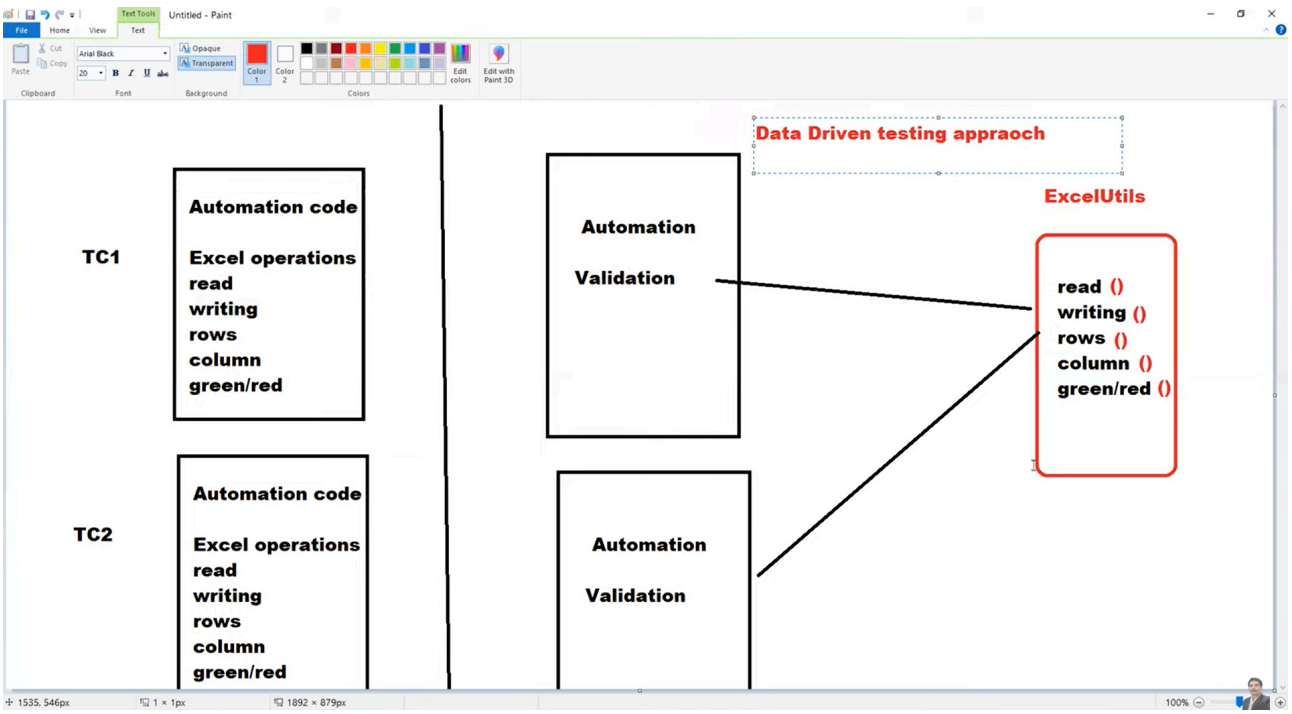
To open an Excel sheet we need to install the “openpyxl” module in Python.

To read the data from the Excel sheet:

- **sheet.cell(r,c).value**

To write the data into the Excel sheet:

- **sheet.cell(r,c).value= ”Charan”**

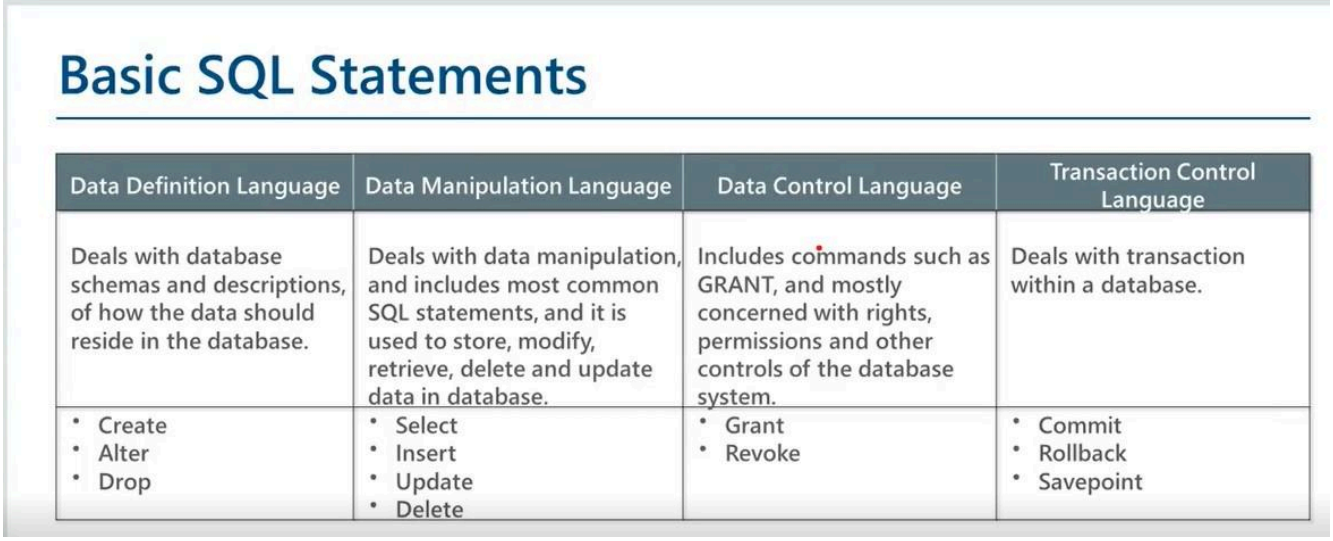


Refer “Excel” directory in the course Vv-imp

DataDriven Testing : (using a database)

Selenium doesn't support directly interacting with SQL, for that, we have to install an external package called **mysql - connector - python** to get data into our Python program. (Just like the openpyxl module to work with Excel)

- Database Server commands:



Basic SQL Statements

Data Definition Language	Data Manipulation Language	Data Control Language	Transaction Control Language
Deals with database schemas and descriptions, of how the data should reside in the database.	Deals with data manipulation, and includes most common SQL statements, and it is used to store, modify, retrieve, delete and update data in database.	Includes commands such as GRANT, and mostly concerned with rights, permissions and other controls of the database system.	Deals with transaction within a database.
• Create • Alter • Drop	• Select • Insert • Update • Delete	• Grant • Revoke	• Commit • Rollback • Savepoint

Database Server: Server means the place where exactly the database is stored, it can be our computer, or in some cases the database will be saved in a virtual machine in our workplace and all the employees will connect to that virtual machine to connect to the database.

Database Client: Client means software that is used to connect with our database.
For Example: In a company, a database is saved in a virtual machine and all the employees will connect to the same server using the client software.

What is SQL: SQL is a language, by using SQL we can communicate with the database.

SQL contains so many sub-languages:

1. DDL (Data Definition language)
2. DML (Data Manipulation language)
3. DRL (Data Retrieval Language) → SELECT
4. DCL (Data Control language)
5. TCL (Transaction Control language)

First, we need to connect our database with our Python code by using mysql-connector-python module and then give all the credentials like hostname, username, password, etc.,

Then we have to create a Cursor object which is used to execute the SQL queries in Python.

- Refer DOB_Operations.py file in SQL_Selenium folder

+ Our date will execute flow will be
  Query in Python code → Cursor → Database

+ Again the returned data from the database through any SELECT command is also saved in the cursor first, and then we have to return it from the cursor to our Python program

  To return the data from the cursor we have to use a loop statement to get all the data from the cursor.
  - Refer Select_DB_Operation.py file in SQL_Selenium folder

- To perform Data Driven Testing using selenium refer to **FixedDepositCalculator_Using_Database.py** where you will get the complete info on how to get data from the DB, insert the same data into a website, and validate the data.

How to write test cases in Python?

- The selenium alone doesn't provide a testing tool/framework, one can use the unittest framework of Python to write test cases.

- The other options for a tool/framework are py.test and nose

  Refer to the unit testing folder in our course.

- Refer creating testsuite folder and selenium logs folder in the course