Selenium Cheat Sheet

Table of Contents

Handling UI Elements

Static Dropdown

Definition: A static dropdown is an independent drop down that holds a list of values that is constant. Every time you click on a static dropdown, it will show the same values. The values are loaded when the whole page is loaded.

Info: When Spying on a web element, the tagname should be ‘select’ & all elements within this select should have tagname as ‘option’

Answer: Instantiate Select object and point it to the UI element you wish to interact with. (don’t forget to import the convenient library)
Ex: Select locationDropdown = new Select(driver.findElement(By.id(“insert id of element here”)));

Now every time you call locationDropdown, you can access all the interactions allowed for Select object.
Ex:

- locationDropdown.selectByValue(“insert value here”);
- locationDropdown.selectByIndex(“insert index here”);
- locationDropdown.selectByVisibleText(“insert visible text here”);

conclusion: The Select object only works on dropdown with tagnames select containing option tagnames as children.

Dynamic Dropdown

Definition: A dynamic dropdown is a dependent drop down that holds a list of values that is not constant. The list is usually dependent on another value pre-selected from a different dropdown or any web element. The values populated in a dynamic drop down vary based on the preselected value. The values in a dynamic dropdown are only loaded once the value it depends on is chosen.

Info: When Spying on a web element, the tagname will mostly not show 'select' tagname. It is often treated like normal UI element. (div/li as oppose to select/option). One popular issue with dynamic dropdowns is that two or more elements(values) inside the dropdowns often share the same xpath. In this case, indexing is a must.

Answer:

- Option 1: Using indexes
  Since we are not using the Select object, there is no need to instantiate this object. To get around dynamic dropdown, move just like you do with any visible web element and rely on indexes.

Ex:

- driver.findElement(By.xpath("Insert xpath here")).click() *// first element*
- driver.findElement(By.xpath("Insert xpath here")).click() *// Second xpath*
- The problem here is that the second xpath is also shared with another element thus your test will fail. In this case, Identify the index of the element you want and specify it in your code as such:
- driver.findElement(By.xpath("(Insert xpath here)[2]")) *// Second xpath that has index 2*

Conclusion: All elements are scanned from top left down; this is how UI elements are selected and identified. When one or more element shares the same xpath/css/value… Only the first element (top left →down) will always be selected. To resolve this issue and get around it, we use indexes.

- Option 2: Rely on Parent/ Child relation
  Since we are not using the Select object, there is no need to instantiate this object. To get around dynamic dropdown, move just like you do with any visible web element and rely on parent/Child relation.

Ex:

- driver.findElement(By.xpath("Insert xpath here")).click() *// first element*
- driver.findElement(By.xpath("Insert xpath here")).click() *// Second xpath*
- The problem here is that the second xpath is also shared with another element thus your test will fail. In this case, Identify the parent of this element, start with it to get to your target. Specify it in your code as such:
- driver.findElement(By.xpath("(Insert parent/ child xpath here) ")) *// first xpath*
  - driver.findElement(By.xpath("(//div[@id= 'parentXpath1'] *//a[@value='childXpath1'])*)
- driver.findElement(By.xpath("(Insert parent/child xpath here) "))// second xpath
  - driver.findElement(By.xpath("(//div[@id= 'parentXpath1'] *//a[@value='childXpath2'])*)

Conclusion: All elements are scanned from top left down; this is how UI elements are selected and identified. When one or more element shares the same xpath/css/value… Only the first element (top left →down) will always be selected. To resolve this issue and get around it, we use indexes. **This method is not very useful when elements sharing the same locator are under the same parent**

Auto Suggestive Dropdown

Definition: A suggestive dropdown is an independent dropdown that holds a list of values that is constant. A suggestive list is populated in response to every character entered in the dropdown input box.

It is usually used to facilitate the selection process for the user, by having to type few letters and pressing on enter once the desired value is highlighted by the system

Info: When Spying on a web element, the tagname will mostly not show 'select' tagname. It is often treated like normal UI element. (div/li as oppose to select/option). Think about it from a user experience perspective. The key here is to identify the dropdown element, send few characters, click on enter or navigate through the suggested list by pressing key down & key up before hitting enter.

Answer: implicate your move to interact with the suggestive drop down in code making each step a line of code. Identify the object, enter the data in that object, hit enter button, Or move between the suggested list then hit enter.
Ex:

- driver.findElement(By.id("insert id here")).clear(); *// Object identified, and existing data is cleared, in case any is prepopulated when pages load*
- driver.findElement(By.id("insert id here")).sendKeys("insert input data"); *//Input data sent*
- driver.findElement(By.id("insert id here")).sendKeys(Keys.ARROW_DOWN);
    - Omit this step if you wish to select the first highlighted suggested value
- driver.findElement(By.id("insert id here")).sendKeys(Keys.ENTER) *// Hit Enter button*

Conclusion: It all depends on what you are trying to test when dealing with suggestive dropdown button. It could be one of many tests. Trying out that key up & key down are functional in a suggestive list, or maybe the sorting of the values suggested, maybe even test that when hitting enter, the highlighted value is correctly displayed in the dropdown… Really just be creative when creating your test dealing with suggestive dropdowns and try to mimic your behavior as a user and turn it into scenarios through your code.

Checkboxes

Definition: A checkbox, is just like any other visible element on the UI. It is most commonly of type 'checkbox'. This element unlike the radio button allows the user to select multiple UI elements of type checkbox at once. It acts as a Boolean value, it is either checked or unchecked, and to perform such an action, we use the click button.

Info: When spying on a web element, the tagname will mostly show the value 'input'. It is often treated like normal UI element. Think about it from a user experience perspective. The key here is to identify the checkbox element and perform a click action to either check or uncheck.

Answer: implicate your move to interact with the checkbox and translate them to code. There is always a default state for a checkbox. It is either checked or unchecked which is why you need to be careful when writing the code to support your testcase. It is commonly used with an if condition (to determine current state) before you can perform the actual action upon it.
On a side note, one interesting test is to check how many checkboxes in total there is on the page and print their size. To do so, you must find one common element for all the checkboxes and get its size. Hint, you can rely on the type attribute and in this case, make sure to use driver.findElements instead of driver.findElement; *Notice the letter s at the end*.
Ex:

- driver.findElement(By.id("insert id here")).isSelected(); *// If the checkbox current state at this point is unselected, this will hold the value false*
    - Put the above code in a condition to rely on it in your next line of code
- driver.findElement(By.id("insert id here")).Click(); *// This will perform the click action on the checkbox. If the current state is checked, this action will perform an uncheck action, and vise versa*
    - Rely on the condition mentioned above to decide whether you want to click to uncheck or click to check.

Calendar UI

Definition: A Calendar UI is usually represented through two forms of input, 'From Date' & 'To Date'. In most recent practice, few are the ones who develop this UI from scratch.

Info: Something common about the calendar UI is that it has a built-in function to always identify the current date. This could be used to your advantage when testing through selenium. If you're attentive enough you can easily identify that special UI element to interact with it based on your test. Another thing to be aware of is that some dates could be disabled, and this can be done through numerous ways. It is important to try and understand how the enable/ disable mode is implemented to test correctly.

Answer: Interacting with Calendar UI elements is just like interacting with any UI element. Any element can be reached by identifying it correctly. Current dates usually have a special UI element. Understanding how the disable/ enable mechanism is implemented allows you to easily identify it and test it through code.

Handling Java Alerts (browser alerts)

Definition: A Java Alert is a browser popup alert just like the ones you might get when you attempt to exit a page without saving. It is important to mention that those alerts do not take any input values and are known to only have two forms of interactions, either to accept or dismiss. Those alerts can be handled, or in fact interacted with through selenium once you teach the latter to switch executing commands to the popup instead of the browser

Info: Unlike web elements, you cannot spy on buttons in the java alerts. Instead selenium allows users to use a built-in function that allows you to switch to the popup to interact with it.

Answer: Interacting with the java alert is done yet through WebDriver built in functions and it allows you to either agree with the content or dismiss the content. WebDriver knows exactly which is what and performs actions accordingly.

Ex:

- Driver.switchTo().alert().accept()
- Driver.switchTo().alert().dismiss()

Window Authentication Popups

Unlike Java alerts, window authentication popups take a username and password input, thus they are handled differently

Working with Multiple Windows

Definition: Sometimes when you click a button, or a hyperlink or any visible element on the web page, another window or tab is opened, and the content is loaded. You will then have more than one window. The initial window is called parent window, the newly opened window (as a result of clicking on a hyperlink on parent window for example) is called child window. This is a common scenario when automating with Selenium

Info: It is very important to mention that when the above scenario happens, Selenium will keep the parent window in action by default and will try to execute the rest of the commands on the parent window. If you wish to access the newly opened window, you need to tell Selenium to switch windows before executing any command.

Answer: To move to from parent window to child window or any other window, first you need to gather all available window ids using getWindowHandles() function in order to explicitly switch to that window using driver.switchTo().window("insert windo id here");

Ex: A good way to store all windows ids is to use Set object and then iterate over this set to define and access each id

- Set<String>windowIds= driver.getWindowHandles();
- Iterator<String> it= windowIds.iterator();
- String parentid= it.next(;)
- String childid= it.next());
- driver.switchTo().window(childid);// here we successfully switch to the child window and can perform actions on that window; to move back to parent window, just call the same function and pass parentid as a parameter instead.

Working with Actions Class

Definition: Actions is a class by selenium that allows user like behavior and interactions with a web page. For instance:

- Hover the mouse over on object with Selenium
- Performing Mouse and keyboard interactions with Selenium (*clicks or holding keyboard buttons*)
- Context click non element (*Right click on the mouse*)
- Double clicking on element
- Drag and drop element (*explained in frames section*)

Moreover, this class allows the user to perform composite actions (one line of code to support multiple actions on the web driver; for example: *Send value into search bar, double click on the value entered to highlight it, move the mouse and over support button*)

Info: To make use of the Actions class, you need to instantiate an object of Actions class and pass chrome driver as a parameter in order to play around with all action class methods (*Actions a= new Actions(driver). Note that driver parameter is the WebElement holding ChromeDriver*).
It is very important to mention that you should explicitly build and invoke the composite action once you are done writing the line of code for all the desired actions. To do so, you must enter call the build function and perform function at the end of the line. (*.build().perform()*)

Ex:

- a.moveToElement(driver.findElement(By.id("insert id here"))).click().keyDown(keys.SHIFT).sendKeys("insert key to send here").doubleClick().build().perform
  - a is the new object of Actions class
  - moveToElement() is a method in Actions class, used to move the cursor towards a target on the web page
  - click() is a method to perform the action of clicking
  - keyDown(Keys.SHIFT) is a method used to hold the button shift (*To write in uppercase letters*)
  - .sendKeys() is a method used to input data into a web element
  - .doubleClick() is a method in Actions class to perform the action of double clicking on the mouse
  - .build() is a method in Actions class used to build all composite action to perform
  - .perform() is a method in Actions class to perform the built composite action
- a.moveToElement(driver.findElement(By.id("insert id here"))).contextClick().build().perform()
  - contextClick() is a method in Actions class to perform a right click on the mouse

Working with Frames

Definition: In the context of a web browser, a frame is a part of a web page or browser window which displays content independent of its container, with the ability to load content independently. The HTML or media elements that go in a frame may or may not come from the same web site as the other elements of content on display.
Info: Trying to access an element in a frame is possible, just not similarly as if you are accessing a web element on a web page straight forward by finding the element by specifying the locator (Xpath, CSS, id, etc.…). You need to move into the frame for your code to work anytime you wish to access an element within a frame. This is easily done by using the selenium switchTo() method.

You can move between frames either by specifying the ID, WebElement, or index. When ID is not present, it is best to refer to WebElement over index only to avoid maintenance in case anew frame is added in the future. When using index, the index is not a constant when new frames are added to the web page, thus you might access the wrong frame and your test might fail.

Bear in mind that if you wish to execute some actions outside the frame after, you need to switch back to the default content. This is done by writing driver.switchTo().defaultContent();

Ex:

- driver.switchTo().frame(“insert name of frame”);
    - Switching to frame by name
- driver.switchTo().frame(“insert id of frame”);
    - Switching to frame by id
- driver.switchTo().frame(driver.findElement(By.cssSelector(“insert CSS selector here”)))
    - Switching to frame by WebElement
- driver.switchTo().frame(0);
    - Switching to frame by index

  NB: As a common practice, you could get the number of frames present on the webpage when using index as a locator in order to navigate your way. This is done through the following syntax: Int size = driver.findElements(By.tagName("iframe")).size();

More on drag and drop function in Actions class: dragging and dropping an element is most commonly associated with frames. Therefore, it is essential to specify the frame first before delving into the action of dragging and dropping an element. Now that you know how to switch to a specific frame, you can access the Actions class after declaring its object and use dragAndDrop() method

Ex:

- Actions a = new Actions(driver);
- a.dragAndDrop(source, target);
    - This method takes two WebElements parameters where source is the WebElement of the element you wish to use, target is the WebElement of the location you wish to drop the draggable item.
        - a.dragAndDrop(driver.findElement(By.id(“insert id here”)), driver.findElement(By.id(“insert id here”))).build().perform();

Miscellaneous

Interview Questions

What is the difference between Relative and absolute xpath?

- Relative- Doesnot depend on parent nodes
- Parent/child- Absolute xpath –

How to traverse to sibling element using xpath?

- .//*[@id='tablist1-tab1']/following-sibling::li[2]

How to traverse back to Parent element from Child element using Xpath?

- .//*[@id='tablist1-tab1']/parent::ul

Facts

- Every object may not have ID, className or name- Xpath and CSS Preferred
- Alpha numeric id may vary on every refresh- check
- Confirm the link object with anchor "a" tag
- Classes should not have spaces- Compound classes cannot be accepted
- Double quotes inside double quotes are not accepted
- Xpath/CSS can be defined in n number of ways
- Right click copy on blue highlighted html code to generate xpath
- When xpath starts with html – It is not reliable; It is better to switch browser to get another one
- There is no direct way to get CSS in chrome. You will find it in tool bar
- Degrade browser to less firefox 55 to get Firepath
- $("") - for css , $x("") or xpath; When validating in Console tab in the developer tool
- //tagName[@attribute='value'] - xpath syntax
- tagName[attribute='value'] -CSS tagName#id- CSS tagname.classname- CSS
- //tagName[contains(@attribute,'value')] - xpath regular expression
- tagName[Atrribute*='value'] - Css regular expression
- .classname
    - Only with css, another way to identify an element by class name is to use dot '.' before the actual classname. If classname has a space, replace space with another dot '.'
- #id
    - Only with css, another way to identify an element by class id is to use hash '#' before the actual id