#1) get()

Command using get() to open a URL in the current browser.
The command below will open the specified URL, 'https://www.softwaretestinghelp.com' in the browser.

Syntax:
driver.get("https://www.softwaretestinghelp.com");

Explanation:

- Navigate to the URL https://www.softwaretestinghelp.com

#2) getCurrentUrl()

Command using getCurrentUrl() to check if the URL is correct.
The below command gets the current URL in the string format.

Syntax:
driver.getCurrentUrl();

We usually use this method in commands to check if we have navigated to the right page as expected. For that, we have to use Assert as shown in the below **Example**.
Syntax:
Assert.assertEquals(expectedUrl, driver.getCurrentUrl());

Where expectedUrl is the URL that is expected in the string format.

Explanation:

- Check and verify that the URL loaded remains the same and the correct page is loaded.

#3) findElement(By, by) and click()

findElement(By, by) and click() to Click on any element of the webpage.
The findElement(By, by) method searches and locates the first element on the current page, which matches the criteria given as a parameter. This method is usually used in commands to simulate user actions like click, submit, type etc.

The below command searches and locates the first element in the webpage with id”submit1” and clicks on it if it is not covered.

Syntax:
driver.findElement(By.id("submit1")).click();

The element can be located using **ID**, **Name**, **Class Name**, **Tag Name**, **Link Text & Partial Link Text**, **CSS Selector** and **X Path**.
Explanation:

- Look for the required Submit button.
- Click on the button.

The command below selects an item from the list box.

Syntax:
WebElement roleDropdown = driver.findElement(By.id("name1");

roleDropdown.click();

Explanation:

- Search and locate the list item by id “name1”.
- Click on that item.

#4) isEnabled()

isEnabled() to Check Whether the Element is Enabled Or Disabled in the Selenium WebDriver.
In order to check if a particular element is enabled in a web page, we use isEnabled() method.

Syntax:
boolean textBox =
driver.findElement(By.xpath("//input[@name='textbox1']")).isEnabled();

Explanation:

- Finds the element in the webpage according to the xpath and checks if the element is enabled.

#5) findElement(By, by) with sendKeys()

findElement(By, by) with sendKeys() to type in the form fields.
Form validation checks by entering the different user inputs that are often required in automation testing. We use findElement(By, by) to locate the fields and sendKeys() to type some content into an editable field.

The below command uses the Name locator to find the form field and types “Aaron” in it.

Syntax:

```
driver.findElement(By.name("name")).sendkeys("Aaron");
```

Explanation:

- Look for the required name field in the form.
- Enter the value “Aaron” in it.

#6) findElement(By, by) with getText()

findElement(By, by) with getText() to store value of targeted web element.
The getText() is a method that gets you the inner text of the web element. Get text is the text inside the HTML tags.

The below code finds the Element with tagName “select” and gets the text inside the tag and stores it in a variable drop-down. Now the String dropDown can be used for further actions inside the program.

Syntax:

```
String dropDown = driver.findElement(By.tagName("dropdown1")).getText();
```

Explanation:

- Look for the required field in the form which has the tagName “dropdown1”.
- Take the text inside its HTML tag.
- Store the text in String object ‘DropDown’.

#7) Submit()

Submit() to submit a web form.
The click() method that we discussed above can be used to click on any links or buttons. Submit() is a better alternative to click() if the element to be clicked is a submit button. The submit button is inside the HTML 'form' tag and the type of button is 'submit'(not 'button').

The submit() makes life easier by automatically finding the button and the method that can be appended to any other field like name or email address. In the case of click, we have to use findElement(By, by) method and specify the correct locators.

In some scenarios where the action is done through elements other than a button, submit() works and click() won't.

Syntax:
driver.findElement(By.xpath("//input[@name='comments']")).submit();

Explanation:

- Find element in the given x path with name 'comments'.
- Submit the form.

#8) findElements(By, by)

findElements(By, by) to get the list of web elements.
Sometimes we might want to print or do an action on a list of web elements like links or input fields in a webpage. In such a case, we use findElements(By, by).

Syntax:
List<WebElement> allChoices =
dropDown.findElements(By.xpath(".//fruitoption"));

Explanation:

- A list of all the web elements with specified xpath is stored in the webelement list allChoices.

#9) findElements(By, by) with size()

findElements(By, by) with size() to verify if an element is present.
findElements(By, by) can be used to verify if an element is actually present in the webpage.

The command below is used if we want to verify that an element with particular locator is present in a webpage. If size() != 0 then the element is present.

Syntax:
Boolean checkIfElementPresent=
driver.findElements(By.xpath("//input[@id='checkbox2']")).size()!= 0;

Explanation:

- Find element is specified in xpath with id 'checkbox2'.
- According to the size of the element list, the Boolean checkIfElementPresent will be set to TRUE or FALSE.

#10) pageLoadTimeout(time,unit)

pageLoadTimeout(time,unit) to set the time for a page to load.
Sometimes due to server issues or network delays, a page might take more than usual time to load. This might throw an error in the program. In order to avoid this, we set a wait time and pageLoadTimeout() is one of such method. This will usually follow a get() command.

Syntax:
driver.manage().timeouts().pageLoadTimeout(500, SECONDS);

Explanation:

- Wait for 500 seconds for a page to load.

#11) implicitlyWait()

implicitlyWait() to set a wait time before searching and locating a web element.
What happens if the Webdriver tries to locate an element before the webpage loads and the element appears? NoSuchElementExeption will be thrown. In order to avoid this, we can add a command to implicitly wait for a certain amount of time before locating the element.

Syntax:
driver.manage().timeouts().implicitlyWait(1000, TimeUnit.SECONDS);

Explanation:

- Implicitly wait for 1000 seconds before executing the next line in the code.

#12) untill() and visibilityOfElementLocated()

untill() from WebdriverWait and visibilityOfElementLocated() from ExpectedConditions to wait explicitly till an element is visible in the webpage.
To handle cases where an element takes too much time to be visible on the software web page applying implicit wait becomes tricky. In this case, we can write a comment to wait until the element appears on the webpage. This command uses a combination of until() method from the WebdriverWait Class and visibilityOfElementLocated() method from the ExpectedConditions class.

Syntax:
WebDriverWait wait = new WebDriverWait(driver, 10);

WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated (By.xpath("//input[@id='name']")));

Explanation:

- The first line says how much time to wait which is 10 seconds.
- The second condition says an expected condition to wait for. Here it is an element with id'name' in the mentioned xpath.

#13) untill() and alertIsPresent()

untill() from WebdriverWait and alertIsPresent() from ExpectedConditions to wait explicitly till an alert appears.
In some scenarios, we have to wait for alerts to continue the test. In this case, we use a command using until() method from the WebdriverWait class and alertIsPresent() method from the ExpectedConditions class.

Please see the command below:
WebDriverWait wait = new WebDriverWait(driver, 10);

```
WebElement element = wait.until(ExpectedConditions.alertIsPresent()

);
```

Explanation:

- The first line says how much time to wait – that is 10 seconds.
- The second condition says an expected condition to wait for. Here it is an alert pop up.

#14) getTitle()

getTitle() to get page title in the Selenium webdriver.
Syntax:

```
String title = driver.getTitle();

System.out.println(title);
```

This is usually used to print the title in the output logs.

Explanation:

- Get the title of the webpage and store it in the String object title.
- Print the value stored in the title to the output logs.

#15) Select

Select class for selecting and deselecting values from the drop-down in Selenium WebDriver.
We often have dropdown related scenarios. Methods from the Select class is used to handle this. We can use selectByVisibleText(),selectByValue() or selectByIndex() according to the scenario.

Syntax:

```
WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

dropdown.selectByVisibleText("Apple");
```

Explanation:

- Find Drop down using it's id "select".
- Select the visible text "Apple" from the dropdown.

```
WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

Dropdown.selectByValue("Apple")
```

Explanation:

- Find the Drop down using it's id "select".
- Select the text with value "Apple" from the dropdown.

```
WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

listbox.selectByIndex(1);
```

Explanation:

- Find the Drop down using it's id "select".
- Select the drop-down item with index value '1' from the drop-down (Second item).

Similar to the select, we can deselect values from the drop-down using similar commands.

Please check the commands:

```
WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

dropdown.deselectByVisibleText("Apple");
```

Explanation:

- Find the Drop down using it's id "select".
- Deselect the visible text "Apple" from the drop-down.

```
WebElement mySelectedElement = driver.findElement(By.id("select"));
```

```
Select dropdown= new Select(mySelectedElement);

Dropdown.deselectByValue("Apple");
```

Explanation:

- Find the Drop down using it's id "select".
- De-select the text with value "Apple" from the drop-down.

```
WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

listbox.deselectByIndex(1);
```

Explanation:

- Find the Drop down using it's id "select".
- De-select the drop-down item with the index value '1' from the drop-down (Second item).

#16) navigate()

navigate() to navigate between the URLs.

We often see scenarios were we might want to navigate from the landing URL and then go back or forward. In such cases, instead of using get(), we can use navigate(). In Navigate we can use back() and forward() methods without specifying the URLs.

Syntax:

```
driver.navigate().to("https://www.softwaretestinghelp.com");

driver.navigate().back();

driver.navigate().forward();
```

Explanation:

- Navigate to https://www.softwaretestinghelp.com
- Navigate back.
- Navigate forward.

#17) getScreenshotAs()

getScreenshotAs() to Capture the entire page screenshot in Selenium WebDriver.
This one is often required to save your work details or sometimes to manually check the outputs. The below command is used to take a screenshot and save in an output file.

Syntax:

```
File shot = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(shot, new File("D:\\ shot1.jpg"));
```

Explanation:

- Take a screenshot and save the file in object shot.
- Save the file in D drive as shot1.png.

#18) moveToElement()

moveToElement() from the Actions class to simulate mouse hover effect.
There are scenarios where we need to hover over web elements like over menu to see submenu, links to see color changes etc. In these cases, we use Actions class. Take a look at the below syntax for Action class.

Syntax:

```
Actions actions = new Actions(driver);

WebElement mouseHover =
driver.findElement(By.xpath("//div[@id='mainmenu1']/div"));

actions.moveToElement(mouseHover);

actions.perform();
```

Explanation

- Find and Locate the web element with div id ‘mainmenu1’.
- Move the mouse pointer to the element.

#19) dragAndDrop()

dragAndDrop() from Actions class to drag an element and drop it on another element.
In some scenarios, we might want to drag elements. **For Example,** drag an image to the stage. In this case, we can use the Actions class.
In the dragAndDrop method, we pass the two parameters, Source locator- the element we want to drag and Destination locator- the element to which we want to drop.

Syntax:

```
WebElement sourceLocator =
driver.findElement(By.xpath("//*[@id='image1']/a"));

WebElement destinationLocator =
driver.findElement(By.xpath("//*[@id='stage']/li"));

Actions actions=new Actions(driver);

actions.dragAndDrop(sourceLocator, destinationLocator).build().perform();
```

Explanation:

- Find and Locate the source web element.
- Find and Locate the destination web element.
- Drag and drop the source element on the destination element.

#20) switchTo() and accept(), dismiss() and sendKeys()

switchTo() and accept(), dismiss() and sendKeys() methods from Alert class to switch to popup alerts and handle them.
To switch to alerts, popups and handle them, we use a combination the of **switchTo()** and **accept(), dismiss()** methods from the Alert class.
Syntax:

```
Alert alert = driver.switchTo().alert();

alert.sendKeys("This Is Softwaretestinghelp");
```

alert.accept()

Explanation:

- Switch to the alert window.
- Type “This Is Softwaretestinghelp” inside the alert.
- Accept the alert and close it.

alert.dismiss() can be used to dismiss the alert.

#21) getWindowHandle() and getWindowHandles()

getWindowHandle() and getWindowHandles() to handle Multiple Windows in Selenium WebDriver.

There are many cases where web applications have many frames or windows.

Those are mostly advertisements or information popup windows. We can handle multiple windows using Windows Handlers. Webdriver stores a unique window id for each window. We make use of this id to handle them.

Syntax:

String handle= driver.getWindowHandle();

Set<String> handle= driver.getWindowHandles();

The above commands are used to get window ids of the current window and all the windows respectively. Please see the loop below to see how can we go to each window through for loop.

```
for (String handle : driver.getWindowHandles()){

driver.switchTo().window(handle);

}
```

Explanation:

- For each window handle id from driver.getWindowHandles(), switch to that window id.

#22) getConnection()

getConnection() from DriverManager to start Database Connection.
In order to start a database connection, we use getConnection from DriverManager class.

Syntax:
DriverManager.getConnection(URL, "username", "password")

Explanation:

- Connect to the Database through URL and credentials.

#23) POI

POI to read from the excel files.
In data driven testing, we often save inputs in excel file and read it. In order to do this in WebDriver, we import POI package and then use the below command.

Syntax:
Workbook workbook = WorkbookFactory.create(new FileInputStream(file));

Sheet sheet = workbook.getSheetAt(0);

Explanation:

- Create a reader file.
- Read the file.

#24) assertEquals(),assertNotEquals(), assertTrue() and assertFalse()

Asserts using assertEquals(),assertNotEquals(), assertTrue() and assertFalse() to compare the results.
Assertions are used to compare the expected and actual results. Pass or fail of a test is usually decided from the result of assertions. Different types of assert are used in automation.

Syntax:
Assert.assertEquals(message, “This text”);

Assert.assertNotEquals(message, “This text”);

Assert.assertTrue(result<0);

Assert.assertFalse(result<0);

Explanation:

- In the first command, whenever the expected and actual values are same, the assertion passes with no exception. i.e., if the message is “This text”, then the assertion passes.
- In the second command, whenever the expected and actual values are same, the assertion fails with an exception. i.e., if the message is “This text”, then the assertion fails.
- In the third command, if the condition passes, assertion passes. i.e., if result<0, then the assertion passes.
- In the fourth command, if the condition passes, the assertion fails. i.e., if result<0, then the assertion fails.

#25) close() and quit()

close() and quit() to close windows and driver instances.
These commands are used at the end of every automation program.

Syntax:
driver.close()

driver.quit()

Explanation:
The first command closes the current window.

The second command quits this driver instance, closing every associated window, which is opened.