



SDJ INTERNATIONAL
COLLEGE

Selenium Python

Unit 3

– By Tushar Sir



– By Tushar Sir

Selenium Webdriver Packages

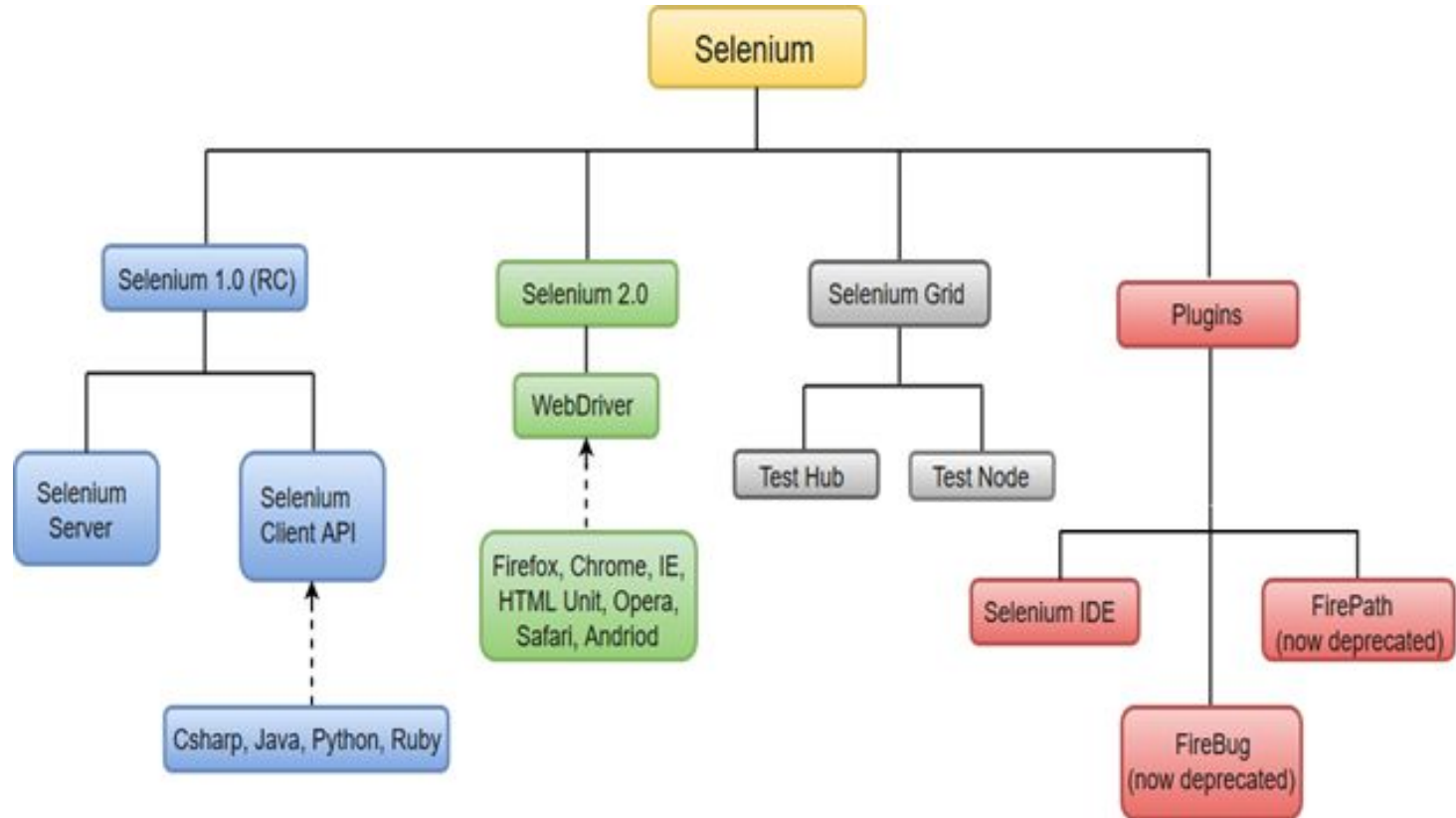
- Selenium WebDriver is a tool that automates the front-end testing of web applications.
- It uses drivers provided by various browsers to simulate user actions.
- Selenium WebDriver has the following drivers:

ChromeDriver, FirefoxDriver, InternetExplorerDriver, SafariDriver, OperaDriver, EdgeDriver, EventFiringWebDriver, RemoteWebDriver.

- Selenium WebDriver was first introduced as a part of Selenium v2.0. The initial version of Selenium i.e Selenium v1 consisted of only IDE, RC and Grid.
- However, with the release of Selenium v3, RC has been deprecated and moved to legacy package.
- Selenium WebDriver performs much faster as compared to Selenium RC because it makes direct calls to the web browsers. RC on the other hand needs an RC server to interact with the browser.

– By Tushar Sir

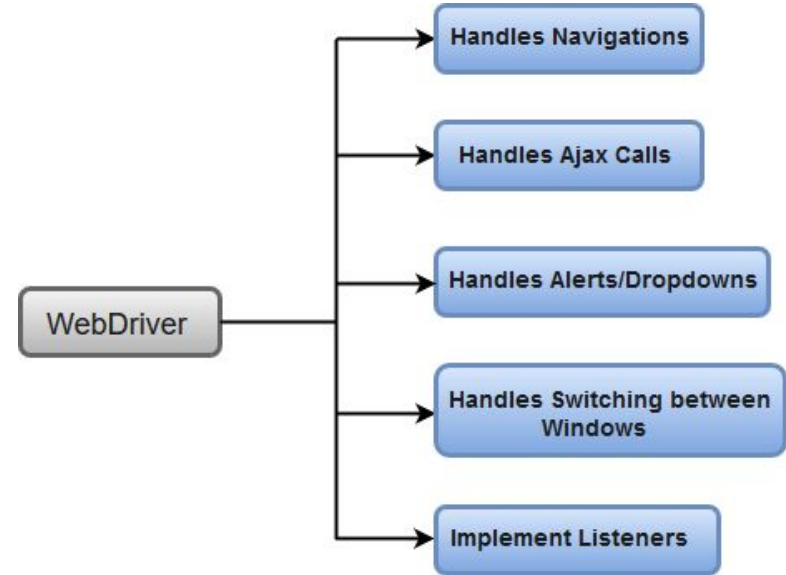
Selenium Webdriver Packages



– By Tushar Sir

Selenium Webdriver Packages

- WebDriver has a built-in implementation of Firefox driver (Gecko Driver). For other browsers, you need to plug-in their browser specific drivers to communicate and run the test.
- Most of the commands used in Selenium WebDriver are easy to implement.
- WebDriver provides multiple solutions to cope with some potential challenges in automation testing.
- WebDriver also allows testers to deal with complex types of web elements such as checkboxes, dropdowns and alerts through dynamic finders.



webdriver.support.ui package

- The package `org.openqa.selenium.support.ui` is used in Selenium. Selenium WebDriver is an interface that defines a set of methods. The implementation is provided by browser-specific classes, such as:

AndroidDriver, ChromeDriver, FirefoxDriver, InternetExplorerDriver, IPhoneDriver, SafariDriver.

- The Selenium.Support package contains .NET support utilities and classes. It supports automating multiple browser platforms.
- Selenium WebDriver allows you to address more complex scenarios when there is a need for fully automated UI tests (e.g. automated regression tests) and / or to parallelize their executions across different browsers (Selenium Grid).

Initialize Browser, Navigate to any website.

- First, import the WebDriver and Keys classes from Selenium.

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

- The WebDriver class will connect you to a browser's instance, which we will shortly cover. The Keys class lets you emulate the stroke of keyboard keys, including special keys like “Shift” and “Return”.
- Next, create an instance of Chrome with the path of the driver you downloaded through the websites of the respective browser. This example assumes the driver is in the same directory as the Python script you will execute.

driver = webdriver.Chrome('./chromedriver')

Initialize Browser, Navigate to any website.

- If you are testing on your local machine, this opens an instance of Chrome locally. This command lets you perform tests on it until you use the `.close()` method to end the connection to the browser.
- Next, use the `.get()` method of the driver to load a website. You may also load a local development site as this process is equivalent to opening a window of Chrome on your local machine, typing a URL, and hitting Enter. The `.get()` method starts loading a website and waits for it to render completely before moving on to the next step.

`driver.get("https://www.python.org")`

Initialize Browser, Navigate to any website.

- Once the page loads successfully, you can use the .title attribute to access the textual title of the webpage. If you wish to check whether the title contains a particular substring, use the assert or if statements. For simplicity, let us print the title of the page.

```
print(driver.title)
```

- The output is the following text:

Welcome to Python.org

If you run the test on a Python interpreter, you notice that the Chrome browser window is still active. Also, a message on Chrome states that automated software is currently controlling it.

– *By Tushar Sir*

Initialize Browser, Navigate to any website.

- Next, let us submit a query in the search bar. First, select the element from the HTML DOM, enter a value into it, and submit the form by emulating the Return key press.
- You can select the element using its CSS class, ID, name attribute, or tag name. If you check the source of the query search bar, you notice that the name attribute of this DOM element is “q”.
- Therefore, you can use the `.find_element_by_name()` method as follows to select the element.

```
search_bar = driver.find_element_by_name("q")
```

Initialize Browser, Navigate to any website.

- Once the DOM element is selected, you first need to clear its contents using the `.clear()` method, enter a string as its value using the `.send_keys()` method and finally, emulate the press of the Return key using `Keys.RETURN`.

`search_bar.clear()`

`search_bar.send_keys("getting started with python")`

`search_bar.send_keys(Keys.RETURN)`

- You notice in the window that these actions trigger a change in the URL with the search results in the window. To confirm the current URL of the window, you can use the following command.

`print(driver.current_url)`

Initialize Browser, Navigate to any website.

- The following string is displayed:

'https://www.python.org/search/?q=getting+started+with+python&submit='

- To close the current session, use the `.close()` method. It also disconnects the link with the browser.

`driver.close()`

In a fully automated flow, you will run multiple tests sequentially and may not be able to view each step as they occur.

Get login page of the website, fetch user_id, password

Create a Selenium WebDriver instance

- To launch the website in the desired browser, set the system properties to the path of the driver for the required browser. This example will use ChromeDriver for Login Automation using Selenium Webdriver. The syntax for the same will be:

Webdriver driver = new ChromeDriver();

System.setProperty("webdriver.chrome.driver", "Path of the chrome driver");

Get login page of the website, fetch user_id, password

Configure the Web browser

- Usually, the web page will be in a minimized format when the test case is run. Maximize the browser for a clear picture of the test cases executed. Use the command below to do the same:

`driver.manage.window.maximize();`

Navigate to the web URL

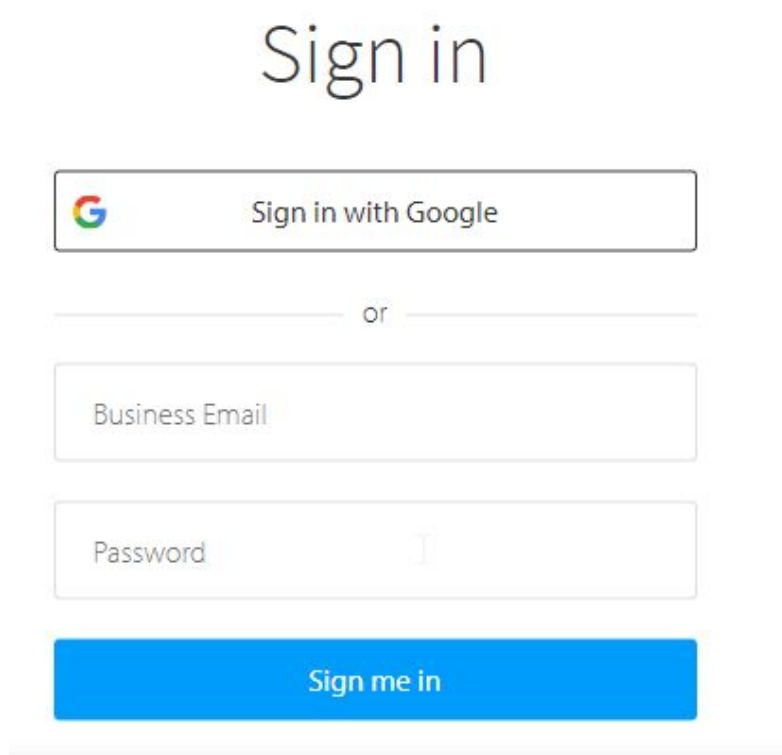
- Open the browser with the desired URL. Use the command below to open the URL in the desired instantiated browser:

`driver.get("https://www.browserstack.com/users/sign_in");`

Get login page of the website, fetch user_id, password

Locating the Web Element

- Locators are an essential part of every Selenium script as they identify the elements that the test script will interact with to replicate user actions.
- For example, let's try to locate the email and password field of the login form of Browserstack sign in page.



– *By Tushar Sir*

Get login page of the website, fetch user_id, password

- Inspect the email field to locate the element using the ID locator, as shown in the image below:

```
</h1>
▶ <p class="sub-text sub-text--sign-up ">...</p>
  <br class="auth-br">
▶ <div class="input-placeholder col-lg-6 col-center">...</div>
▶ <div class="input-placeholder col-lg-6 col-center">...</div>
▶ <div class="input-placeholder col-lg-6 col-center user-full-name-input-placeholder hide">...</div>
▼ <div class="input-placeholder col-lg-6 col-center">
  ▼ <div class="input-wrapper">
    ...
    <input autocomplete="on" class="input-lg text user_email_ajax" id="user_email_login" name="user[login]" placeholder="Business Email"
      tabindex="2" type="text" value> == $0
    ▶ <div class="error-msg hide">...</div>
    </div>
  </div>
▶ <div class="input-placeholder col-lg-6 col-center pass-input-placeholder">...</div>
▶ <div class="input-placeholder col-lg-6 col-center user-persona-container hide">...</div>
▶ <div class="input-placeholder col-lg-6 col-center user-persona-container hide">...</div>
▶ <div class="input-placeholder col-lg-6 col-center terms-input hide ">...</div>
▶ <div class="input-placeholder col-lg-6 col-center google-login-button hide">...</div>
... container.container-fluid div.row div.auth-form-tablet.text-center div.input-placeholder.col-lg-6.col-center div.input-wrapper input#user_email_login.input-lg.text.user_email_ajax ...
```


Get login page of the website, fetch user_id, password

- Locate it via the ID locator in Selenium WebDriver:

`driver.findElement(By.id("user_email_login"));`

- Since this returns a webelement, store it in webelement variable with:

`WebElement username=driver.findElement(By.id("user_email_login"));`

Get login page of the website, fetch user_id, password

- Repeat the same steps for the password field.



```
<p class="sub-text sub-text--sign-up ">...</p>
<br class="auth-br">
<div class="input-placeholder col-lg-6 col-center">...</div>
<div class="input-placeholder col-lg-6 col-center">...</div>
<div class="input-placeholder col-lg-6 col-center user-full-name-input-placeholder hide">...</div>
<div class="input-placeholder col-lg-6 col-center">...</div>
<div class="input-placeholder col-lg-6 col-center pass-input-placeholder">
  <div class="input-wrapper">
    <input autocomplete="on" class="input-lg text" id="user_password" name="user[password]" placeholder="Password" tabindex="3" type=
      "password"> == $0
    <div class="error-msg hide">...</div>
  </div>
</div>
<div class="input-placeholder col-lg-6 col-center user-persona-container hide">...</div>
<div class="input-placeholder col-lg-6 col-center user-persona-container hide">...</div>
<div class="input-placeholder col-lg-6 col-center terms-input hide ">...</div>
<div class="input-placeholder col-lg-6 col-center google-login-button hide">...</div>
<div class="input-placeholder col-lg-6 col-center github-login-button hide">...</div>
```

`driver.findElement(By.id("user_password"));`

`WebElement password=driver.findElement(By.id("user_password"));`

– By Tushar Sir

Get login page of the website, fetch user_id, password

Perform Action on the Located Web Element

- After locating the element, testers need to perform the desired action. In this case, the action is entering text in the email and password fields and hitting the login button. For this, the script uses sendKeys and click methods, as shown below:

```
username.sendKeys("abc@gmail.com");
```

```
password.sendKeys("your_password");
```

```
login.click();
```

Get login page of the website, fetch user_id, password

Verify & Validate The Action

- To validate the results, use assertion. Assertions are important for comparing the expected results to the actual results. If it matches, the test case passes. If not, then the test case fails. The syntax below will help to assert (validate) the outcome from actions by performing login automation:

Assert.assertEquals(String actual, String expected);

- Save the actual URL post-login into a string value, which is:

String actualUrl="https://www.browserstack.com/users/sign_in";

- The Expected URL can be identified by using the method below:

String expectedUrl= driver.getCurrentUrl();

Get login page of the website, fetch user_id, password

- The final assertion would look like:

Assert.assertEquals(actualUrl, expectedUrl);

- If the test case is passed, it will retrieve the same. Else it will return as failed.
- On executing the code, Selenium will navigate to the Chrome browser and open the BrowserStack login page. Then, it will log in using the relevant credentials. It will also check the test case status using Assert and try to match the URL.
- Follow the steps and protocol detailed above to automate the login function of websites with Selenium. Remember to leverage the power of real browsers and devices along with Selenium's many abilities to create immediately effective test scripts that generate desirable results with minimal time and effort.

Webdriver Methods

`maximize_window()`

- The `maximize_window()` method is used to maximize the current browser window. It is a part of the WebDriver interface in Selenium Python.
- To use the `maximize_window()` method, you first need to create a WebDriver object. Once you have created a WebDriver object, you can call the `maximize_window()` method on it.
- The `maximize_window()` method is a very useful method for automating web tests. By maximizing the browser window, you can ensure that all of the elements on the page are visible. This can help to prevent errors in your tests.
- The following code shows how to maximize the current browser window using the `maximize_window()` method:

```
from selenium import webdriver
```

```
driver = webdriver.Chrome()
```

```
driver.maximize_window()
```

– By Tushar Sir

Webdriver Methods

- **get(String url):** This method will launch a new browser and opens the given URL in the browser instance.
- **getWindowHandle():** It is used to handle single window i.e. main window. Its return type is string. It will return browser window handle from focused browser.
- **getCurrentUrl():** Get a string representing the current URL that the browser is looking at.
- **getTitle():** Get the title of the current page.
- **getPageSource():** Get the source of the last loaded page. If the page has been modified after loading (for example, by Javascript) there is no guarantee that the returned text is that of the modified page. Please consult the documentation of the particular driver being used to determine whether the returned text reflects the current state of the page or the text last sent by the web server. The page source returned is a representation of the underlying DOM: do not expect it to be formatted or escaped in the same way as the response sent from the web server. Think of it as an artist's impression.

– *By Tushar Sir*

Webdriver Methods

`find_element_by_name()`

- The `find_element_by_name()` method in Selenium Python is used to find an element on a web page by its name attribute. The syntax for this method is:

`driver.find_element_by_name(name)`

- Where name is the name attribute of the element you want to find.
- For example, to find the element with the name username, you would use the following code:

`username_element = driver.find_element_by_name('username')`

- Once you have found the element, you can use it to perform actions such as clicking it, sending text to it, or getting its text.

– *By Tushar Sir*

Webdriver Methods

- Here is an example of how to use the `find_element_by_name()` method to click on a button:

```
login_button = driver.find_element_by_name('login')
```

```
login_button.click()
```

- Here is an example of how to use the `find_element_by_name()` method to send text to an input field:

```
username_field = driver.find_element_by_name('username')
```

```
username_field.send_keys('my_username')
```

Webdriver Methods

- Here is an example of how to use the `find_element_by_name()` method to get the text from an element:

```
password_field = driver.find_element_by_name('password')
```

```
password = password_field.get_attribute('value')
```

- The `find_element_by_name()` method is a very useful tool for automating tasks on web pages. It is easy to use and can be used to find any element on a page, regardless of its type.

Webdriver Methods

`send_keys()`

- We can press enter in Selenium with the help of **`send_keys()`** function in Selenium Python.
- `send_keys()` function takes different keys as its parameter. Hence we need to import keys before using this function.
- We can perform all the keyboard operations with the help of keys in Selenium. Class `selenium.webdriver.common.keys` comes with various methods that one can use for this purpose.

Webdriver Methods

- For pressing enter we need to pass Keys.ENTER as parameter to send_keys() method as below:

```
from selenium.webdriver.common.keys import Keys
```

```
driver.find_element_by_name("Value").send_keys(Keys.ENTER)
```

Sending a key without an Element

- For sending keys without specifying an element in Python we can use the ActionChains class as follows –

```
from selenium.webdriver.common.action_chains import ActionChains
```

```
value = "Test"
```

```
actions = ActionChains(driver)
```

```
actions.send_keys(value)
```

```
actions.perform()
```

– By Tushar Sir

Webdriver Methods

Selenium Press Enter without Element Python

- For example, we want to send text to a username field on a login page. The login page has already loaded and the username field is in focus as soon as the login page loads.
- Here we can directly send the username value as the username text box is in focus using the `send_keys` method, then press tab to navigate to the password field and press enter on the login button.

Webdriver Methods

```
from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Firefox(executable_path="C:\\geckodriver.exe")

driver.get("url")

actions = ActionChains(driver)

actions.send_keys(value=username)

actions.send_keys(keys.TAB)

actions.send_keys(value=password)

actions.send_keys(keys.ENTER)

actions.perform()

driver.quit()
```

– *By Tushar Sir*

Webdriver Methods

close()

- **driver.close()** closes only the current window on which Selenium is running automated tests.
- On the other hand, the **driver.quit()** method closes all browser windows and ends the WebDriver session.
- close() is a webdriver command that closes the browser window currently in focus.
- During the automation process, if there is more than one browser window opened, then the close() command will close only the current browser window, which is having focus at that time.
- The remaining browser windows will not be closed.

Import Keys class from Selenium.webdriver.common.keys

from selenium.webdriver.common.keys import Keys

- This code imports the Keys class from the selenium.webdriver.common.keys module.
- The Keys class contains a list of all the special keys that can be pressed on a keyboard.
- This can be used to automate tasks such as pressing the enter key or the backspace key.

Import Keys class from Selenium.webdriver.common.keys

```
# import webdriver
```

```
from selenium import webdriver
```

```
# import Action chains
```

```
from selenium.webdriver.common.action_chains import ActionChains
```

```
# import KEYS
```

```
from selenium.webdriver.common.keys import Keys
```

```
# create webdriver object
```

```
driver = webdriver.Firefox()
```

– By Tushar Sir

Import Keys class from Selenium.webdriver.common.keys

```
# get geeksforgeeks.org
```

```
driver.get("https://www.geeksforgeeks.org/")
```

```
# create action chain object
```

```
action = ActionChains(driver)
```

```
# perform the operation
```

```
action.key_down(Keys.CONTROL).send_keys('F').key_up(Keys.CONTROL).perform()
```

Thank You

- By Tushar Sir