



SOFTWARE QUALITY

Software Quality Assurance

Lecture 12

Agenda

- Selenium
- Selenium webdriver
- Selenium IDE

WebDriver also enables you to **use a programming language** in creating your test scripts (not possible in Selenium IDE).

Following programming languages are supported by WebDriver

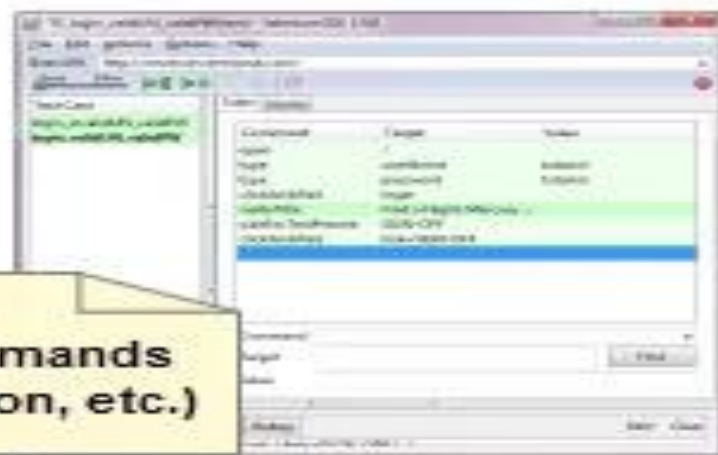
- Java
- .Net
- PHP
- Python
- Perl
- Ruby

You do not have to know all of them. You just need to be knowledgeable in one

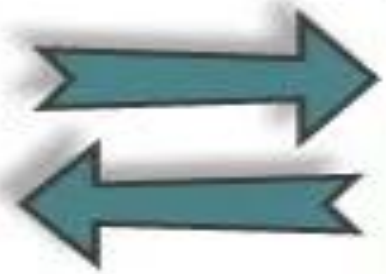
WebDriver's architecture is simpler than Selenium RC (Remote control)'s.

- It controls the browser from the OS level
- All you need are your programming language's IDE (which contains your Selenium commands) and a browser.

**Selenium Commands
(Java, .Net, Python, etc.)**



Web Server



- **WebDriver interacts with page elements in a more realistic way.** For example, if you have a disabled text box on a page you were testing, WebDriver really cannot enter any value in it just as how a real person cannot.
- **WebDriver interacts with page elements in a more realistic way.** For example, if you have a disabled text box on a page you were testing, WebDriver really cannot enter any value in it just as how a real person cannot.

Selenium Server is written in Java, and you need to have JRE 1.6 or above to install it on your server. It is available on Selenium's download page.

How to run your automated test using Selenium and Python?

Once you have completed the pre-requisites section, you are ready to start your first test in Selenium with the Python programming language!

1. 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. The Keys class lets you emulate the stroke of keyboard keys, including special keys like "Shift" and "Return".

2. Next, create an instance of Chrome with the path of the driver that you downloaded through the websites of the respective browser. In this example, we assume that the driver is in the same directory as the Python script that you will execute.

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

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.

3. 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 not only starts loading a website but also waits for it to render completely before moving on to the next step.

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

4. 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, you can 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 are running 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 controlling it at the moment.

Inspect python.org

```
▶ <div id="top" class="top-bar do-not-print">...</div>
  <!-- Header elements -->
▼ <header class="main-header" role="banner">
  • ▼ <div class="container"> == $0
    ▶ <h1 class="site-headline">...</h1>
    ▼ <div class="options-bar-container do-not-print">
      <a href="https://psfmember.org/civicrm/contribute/transact?reset=1&id=2" class="donate-button">Donate</a>
    ▼ <div class="options-bar">
      ▶ <a id="site-map-link" class="jump-to-menu" href="#site-map">...</a>
      ▼ <form class="search-the-site" action="/search/" method="get">
        ▼ <fieldset title="Search Python.org">
          ▶ <span aria-hidden="true" class="icon-search">...</span>
          <label class="screen-reader-text" for="id-search-field">Search This Site</label>
          <input id="id-search-field" name="q" type="search" role="textbox" class="search-field"
            placeholder="Search" value tabindex="1">
          <button type="submit" name="submit" id="submit" class="search-button" title="Submit this Search"
            tabindex="3"> GO </button>
          <!--[if IE]><input type="text" style="display: none;" disabled="disabled" size="1"
            tabindex="4"><![endif]-->
        </fieldset>
```

5. Next, let us submit a query in the search bar. First, select the element from the HTML DOM and 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, its name attribute, or even the 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")
```

6. 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)
```

```
from selenium import webdriver

from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome("C:\Program
Files\Google\Chrome\Application\chromedriver.exe")

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

search_bar = driver.find_element_by_name("q")

search_bar.clear()

search_bar.send_keys("getting started with python")

search_bar.send_keys(Keys.RETURN)
```

Using Selenium to write tests

- Selenium is mostly used for writing test cases. The selenium package itself doesn't provide a testing tool/framework. You can write test cases using Python's unittest module. The other options for a tool/framework are pytest and nose.
- Here is the modified example which uses unittest module. This is a test for python.org search functionality:

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
class PythonOrgSearch(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
    def test_search_in_python_org(self):
        driver = self.driver
        driver.get("http://www.python.org")
        self.assertIn("Python", driver.title)
        elem = driver.find_element_by_name("q")
        elem.send_keys("pycon")
        elem.send_keys(Keys.RETURN)
        assert "No results found." not in driver.page_source
    def tearDown(self):
        self.driver.close()
if __name__ == "__main__":
    unittest.main()
```


Navigating

The first thing you'll want to do with WebDriver is navigate to a link. The normal way to do this is by calling `get` method:

```
driver.get("http://www.google.com")
```

Interacting with the page

To interact with the pages, or, more specifically, the HTML elements within a page. First of all, we need to find one. WebDriver offers a number of ways to find elements. For example, given an element defined as:

```
<input type="text" name="passwd" id="passwd-id" />
```

you could find it using any of:

```
element = driver.find_element_by_id("passwd-id")
```

```
element = driver.find_element_by_name("passwd")
```

```
element = driver.find_element_by_xpath("//input[@id='passwd-id']")
```

```
element = driver.find_element_by_css_selector("input#passwd-id")
```

First of all, you may want to enter some text into a text field:

```
element.send_keys("some text")
```

You can simulate pressing the arrow keys by using the “Keys” class:

```
element.send_keys(" and some", Keys.ARROW_DOWN)
```

Locating

1. Locating by Id

Use this when you know the id attribute of an element. With this strategy, the first element with a matching id attribute will be returned. If no element has a matching id attribute, a `NoSuchElementException` will be raised.

For instance, consider this page source:

```
<html>
<body>
  <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
  </form>
</body>
</html>
```

The form element can be located like this:

```
login_form = driver.find_element_by_id('loginForm')
```

2. Locating by Name

Use this when you know the name attribute of an element. With this strategy, the first element with a matching name attribute will be returned. If no element has a matching name attribute, a `NoSuchElementException` will be raised.

For instance, consider this page source:

```
<html>
<body>
  <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
  </form>
</body>
</html>
```

The username & password elements can be located like this:

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

3. Locating by XPath

- XPath is the language used for locating nodes in an XML document. As HTML can be an implementation of XML (XHTML), Selenium users can leverage this powerful language to target elements in their web applications.
- XPath supports the simple methods of locating by id or name attributes and extends them by opening up all sorts of new possibilities such as locating the third checkbox on the page.
- One of the main reasons for using XPath is when you don't have a suitable id or name attribute for the element you wish to locate.
- You can use XPath to either locate the element in absolute terms (not advised), or relative to an element that does have an id or name attribute. XPath locators can also be used to specify elements via attributes other than id and name.

For instance, consider this page source:

```
<html>
<body>
  <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
  </form>
</body>
</html>
```


The form elements can be located like this:

```
login_form = driver.find_element_by_xpath("/html/body/form[1]")
```

```
login_form = driver.find_element_by_xpath("//form[1]")
```

```
login_form = driver.find_element_by_xpath("//form[@id='loginForm']")
```

The username element can be located like this:

```
username = driver.find_element_by_xpath("//form[input/@name='username']")
```

```
username = driver.find_element_by_xpath("//form[@id='loginForm']/input[1]")
```

```
username = driver.find_element_by_xpath("//input[@name='username']")
```

Filling in forms

```
element = driver.find_element_by_xpath("//select[@name='name']")

all_options = element.find_elements_by_tag_name("option")

for option in all_options:

    print("Value is: %s" % option.get_attribute("value"))

    option.click()
```

This will find the first “SELECT” element on the page, and cycle through each of its OPTIONS in turn, printing out their values, and selecting each in turn.

As you can see, this isn't the most efficient way of dealing with SELECT elements. WebDriver's support classes include one called a "Select", which provides useful methods for interacting with these:

```
from selenium.webdriver.support.ui import Select

select = Select(driver.find_element_by_name('name'))

select.select_by_index(index)

select.select_by_visible_text("text")

select.select_by_value(value)
```

WebDriver also provides features for deselecting all the selected options:

```
select = Select(driver.find_element_by_id('id'))  
  
select.deselect_all()
```

This will deselect all **OPTIONS** from that particular **SELECT** on the page.

To move backward and forward in your browser's history:

```
driver.forward()  
  
driver.back()
```

Explicit waits in Selenium Python

- When a page is loaded by the browser, the elements within that page may load at different time intervals.
- This makes locating elements difficult: if an element is not yet present in the DOM, a locate function will raise an `ElementNotVisibleException` exception. Using waits, we can solve this issue.
- Waiting provides some slack between actions performed – mostly locating an element or any other operation with the element. Selenium Webdriver provides two types of waits – implicit & explicit.

Explicit Waits

- An explicit wait is a code you define to wait for a certain condition to occur before proceeding further in the code.
- The extreme case of this is `time.sleep()`, which sets the condition to an exact time period to wait.
- There are some convenience methods provided that help you write code that will wait only as long as required.
- Explicit waits are achieved by using `webdriverWait` class in combination with `expected_conditions`.

Let's consider an example –

```
# import necessary classes
```

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

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

```
from selenium.webdriver.support import expected_conditions
```

```
as EC
```

```
# create driver object
```

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

```
# A URL that delays loading
```

```
driver.get("http://somedomain / url_that_delays_loading")
```

```
try:
```

```
    # wait 10 seconds before looking for element
```

```
    element = WebDriverWait(driver, 10).until(
```

```
        EC.presence_of_element_located((By.ID,
```

```
        "myDynamicElement"))
```

```
    )
```

```
finally:
```

```
    # else quit
```

```
    driver.quit()
```

This waits up to 10 seconds before throwing a `TimeoutException` unless it finds the element to return within 10 seconds. `WebDriverWait` by default calls the `ExpectedCondition` every 500 milliseconds until it returns successfully.

Expected Conditions –

There are some common conditions that are frequently of use when automating web browsers. For example, `presence_of_element_located`, `title_is`, and so on. Some of them are –

- `title_is`
- `title_contains`
- `presence_of_element_located`
- `visibility_of_element_located`
- `visibility_of`
- `presence_of_all_elements_located`
- `element_located_to_be_selected`
- `element_selection_state_to_be`
- `element_located_selection_state_to_be`
- `alert_is_present`

How to create an Explicit wait in Selenium Python ?

Explicit wait as defined would be the combination of WebDriverWait and Expected conditions. Let's implement this on <https://www.geeksforgeeks.org/> and wait 10 seconds before locating an element.

```
# import webdriver
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
# create webdriver object
driver = webdriver.Firefox()

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

# get element after explicitly waiting for 10 seconds
element = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.link_text, "Courses")))
# click the element
element.click()
```

Implicit Waits

An implicit wait tells WebDriver to poll the DOM for a certain amount of time when trying to find any element (or elements) not immediately available. The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object. Let's consider an example –

```
# import webdriver
from selenium import webdriver
driver = webdriver.Firefox()
# set implicit wait time
driver.implicitly_wait(10) # seconds

# get url
driver.get("http://somedomain / url_that_delays_loading")

# get element after 10 seconds
myDynamicElement = driver.find_element_by_id("myDynamicElement")
```

This waits up to 10 seconds before throwing a `TimeoutException` unless it finds the element to return within 10 seconds.

How to create an Implicit wait in Selenium Python ?

Implicit wait as defined would be the set using `implicitly_wait` method of driver. Let's implement this on <https://www.geeksforgeeks.org/> and wait 10 seconds before locating an element.

```
# import webdriver
from selenium import webdriver
# create webdriver object
driver = webdriver.Firefox()

# set implicit wait time
driver.implicitly_wait(10) # seconds

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

# get element after 10 seconds
element = driver.find_element_by_link_text("Courses")
# click element
element.click()
```

Action Chains in Selenium Python

ActionChains are a way to automate low-level interactions such as mouse movements, mouse button actions, keypress

This is useful for doing more complex actions like hover over and drag and drop. Action chain methods are used by advanced scripts where we need to drag an element, click an element

How to create an Action Chain Object ?

To create object of Action Chain, import Action chain class from docs and pass driver as the key argument. After this one can use this object to perform all the operations of action chains.

```
# import webdriver
```

```
from selenium import webdriver
```

```
# import Action chains
```

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

```
# create webdriver object
```

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

```
# create action chain object
```

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

How to use Action Chains in Selenium ?

After one has created an object of Action chain, open a webpage, and perform various other methods using below syntax and examples. Action chains can be used in a chain pattern as below –

```
menu = driver.find_element_by_css_selector(".nav")
hidden_submenu = driver.find_element_by_css_selector(".nav # submenu1")
```

```
ActionChains(driver).move_to_element(menu).click(hidden_submenu).perform()
```

Or actions can be queued up one by one, then performed.:

```
menu = driver.find_element_by_css_selector(".nav")
hidden_submenu = driver.find_element_by_css_selector(".nav # submenu1")
```

```
actions = ActionChains(driver)
actions.move_to_element(menu)
actions.click(hidden_submenu)
actions.perform()
```

<u>Method</u>	<u>Description</u>
<u>click</u>	<u>Clicks an element.</u>
<u>click and hold</u>	<u>Holds down the left mouse button on an element.</u>
<u>context click</u>	<u>Performs a context-click (right click) on an element.</u>
<u>double click</u>	<u>Double-clicks an element.</u>
<u>drag and drop</u>	<u>Holds down the left mouse button on the source element, then moves to the target element and releases the mouse button.</u>
<u>drag and drop by offset</u>	<u>Holds down the left mouse button on the source element, then moves to the target offset and releases the mouse button.</u>

<u>key_down</u>	<u>Sends a key press only, without releasing it.</u>
<u>key_up</u>	<u>Releases a modifier key.</u>
<u>move_by_offset</u>	<u>Moving the mouse to an offset from current mouse position.</u>
<u>move_to_element</u>	<u>Moving the mouse to the middle of an element.</u>
<u>move_to_element_with_offset</u>	<u>Move the mouse by an offset of the specified element, Offsets are relative to the top-left corner of the element.</u>
<u>perform</u>	<u>Performs all stored actions.</u>
<u>pause</u>	<u>Pause all inputs for the specified duration in seconds</u>
<u>release</u>	<u>Releasing a held mouse button on an element.</u>
<u>reset_actions</u>	<u>Clears actions that are already stored locally and on the remote end</u>
<u>send_keys</u>	<u>Sends keys to current focused element.</u>

Syntax –

click(on_element=None)

Args –

on_element – The element to click. If None, clicks on current mouse position.

Example –

```
<input type="text" name="passwd" id="passwd-id" />
```

To find an element one needs to use one of the locating strategies, For example,

```
element = driver.find_element_by_id("passwd-id")
```

```
element = driver.find_element_by_name("passwd")
```

Now one can use click method as an Action chain as below –

```
click(on_element=element)
```

How to use click Action Chain method in Selenium Python ?

To demonstrate, click method of Action Chains in Selenium Python. Let' s visit <https://www.geeksforgeeks.org/> and operate on an element.

Program –

```
# import webdriver
from selenium import webdriver

# import Action chains
from selenium.webdriver.common.action_chains import ActionChains

# create webdriver object
driver = webdriver.Firefox()
# get geeksforgeeks.org
driver.get("https://www.geeksforgeeks.org/")
# get element
element = driver.find_element_by_link_text("Courses")
# create action chain object
action = ActionChains(driver)
# click the item
action.click(on_element = element)
# perform the operation
action.perform()
```

```
# import webdriver
from selenium import webdriver
```

```
# import Action chains
from selenium.webdriver.common.action_chains import
ActionChains
```

```
# create webdriver object
driver = webdriver.Firefox()
```

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

```
# get element
element = driver.find_element_by_link_text("Courses")
```

```
# create action chain object
action = ActionChains(driver)
```

```
# click the item
action.click(on_element = element)
```

```
# perform the operation
action.perform()
```

Syntax –

click_and_hold(on_element=None)

Args –

on_element: The element to mouse down. If None, clicks on current mouse position.

Example –

```
<input type = "text" name = "passwd" id = "passwd-id" />
```

To find an element one needs to use one of the locating strategies, For example,

```
element = driver.find_element_by_id("passwd-id")
```

```
element = driver.find_element_by_name("passwd")
```

Now one can use click_and_hold method as an Action chain as below –

```
click_and_hold(on_element=element)
```

How to use double_click Action Chain method in Selenium Python ?

Program –

```
# import webdriver
```

```
from selenium import webdriver
```

```
# import Action chains
```

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

```
# create webdriver object
```

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

```
# get geeksforgeeks.org
```

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

```
# get element
```

```
element = driver.find_element_by_link_text("Courses")
```

```
# create action chain object
```

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

```
# double click the item
```

```
action.double_click(on_element = element)
```

```
# perform the operation
```

```
action.perform()
```

Syntax –

drag_and_drop(source, target)

Args –

- source: The element to mouse down.
- target: The element to mouse up.

Example –

```
<input type="text" name="passwd" id="passwd-id" />
```

To find an element one needs to use one of the locating strategies, For example,

```
element = driver.find_element_by_id("passwd-id")
```

```
element = driver.find_element_by_name("passwd")
```

Now one can use drag_and_drop method as an Action chain as below –

```
drag_and_drop(source, target)
```

How to use drag_and_drop Action Chain method in Selenium Python ?

```
# import webdriver
from selenium import webdriver
# import Action chains
from selenium.webdriver.common.action_chains import
ActionChains

# create webdriver object
driver = webdriver.Firefox()

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

# get source element
source_element =
driver.find_element_by_link_text("Courses")

# get target element
target_element = driver.find_element_by_link_text("Hard")

# create action chain object
action = ActionChains(driver)

# drag and drop the item
action.drag_and_drop(source_element, target_element)

# perform the operation
action.perform()
```

Selenium IDE

Selenium IDE Commands

Each Selenium IDE test step can chiefly be split into the following three components:

- Command
- Target
- Value

Command	Target	Value
type	id=Passwd	TestSelenium
Action needs to be performed	The web element to interact with	String that needs to be entered in the web element

Types of Selenium IDE commands

There are three flavors of Selenium IDE commands. Each of the test steps in Selenium IDE falls under any of the following categories.

- Actions
- Accessors
- Assertions

Actions

Actions are those commands which interact directly with the application by either altering its state or by pouring some test data.

For Example, “type” command lets the user interact directly with the web elements like a text box. It allows them to enter a specific value in the text box and as when the value is entered; it is showed on the UI as well.

Another example is the “click” command. The “click” command lets the user manipulate with the state of the application.

In case of failure of an action type command, the test script execution halts and rest of the test steps would not be executed.

Accessors

Accessors are those commands which allow the user to store certain values to a user-defined variable. These stored values can be later on used to create assertions and verifications.

For example, “storeAllLinks” reads and stores all the hyperlinks available within a web page into a user-defined variable. Remember the variable is of array type if there are multiple values to store.

Assertions

Assertions are very similar to Accessors as they do not interact with the application directly. Assertions are used to verify the current state of the application with an expected state.

Assertions are used to verify the current state of the application with an expected state.

Forms of Assertions:

#1. assert – the “assert” command makes sure that the test execution is terminated in case of failure.

#2. verify – the “verify” command lets the Selenium IDE carry on with the test script execution even if the verification is failed.

#3. wait for – the “waitFor” command waits for a certain condition to be met before executing the next test step. The conditions are like the page to be loaded, element to be present. It allows the test execution to proceed even if the condition is not met within the stipulated waiting period.

Command	Description	#Arguments
open	Opens a specified URL in the browser.	1
assertTitle, VerifyTitle	Returns the current page title and compares it with the specified title	1
assertElementPresent, verifyElementPresent	Verify / Asserts the presence of an element on a web page.	1
assertTextPresent, verifyTextPresent	Verify / Asserts the presence of a text within the web page.	1
type, typeKeys, sendKeys	Enters a value (String) in the specified web element.	2
Click, clickAt, clickAndWait	Clicks on a specified web element within a web page.	1
waitForPageToLoad	Sleeps the execution and waits until the page is loaded completely.	1
waitForElement Present	Sleeps the execution and waits until the specified element is present	1
chooseOkOnNext Confirmation, chooseCancelOn NextConfirmation	Click on "OK" or "Cancel" button when next confirmation box appears.	0

Commonly used Selenium **IDE commands**

assertTitle gets the title of a website and checks it against the provided text.

- Assert and verify commands are both useful for verifying condition match or not.
- The difference is that verify command will verify the condition and if it does not match, it will give an error message in the Log area and the macro continues to run.
- With the assert command, if the condition does not match then it will stop remaining macro execution in the selenium IDE software testing tool.

assertText Example

Command	Target	Pattern/Text
open	https://ui.vision/	
assertTitle	Open-Source RPA and Web Automation Tools for macOS, Linux and Windows	

Executing ▾



✓ a*

Run all tests Ctrl+Shift+R ium.dev ▾

Command

Target

Value

1

✓ open

<https://ui.vision/>

2

✓ assert title

Open-Source RPA and Web Automation Tools for macOS, Linux and Windows

Command

//



Target



Value

Description

Runs: 1 Failures: 0

Executing ▾

⏮

▶

🔄

⌚ ▾

🚫

⏸

❌

X a*

https://www.selenium.dev

	Command	Target	Value
1	✓ open	https://ui.vision/	
2	X assert title	Open-mSource RPA and Web Automation Tools for macOS, Linux and Wi...	

Command

assert title ▾

//

🔗

Target

Open-mSource RPA and Web Automation Tools for ma

🔍

🔍

Value

Description

Runs: 1 Failures: 1

assertText and **verifyText** both get the text of an element (as defined by the locator) and check if it meets the requirement of the pattern. This works for any element that contains text.

omayo (QAFox.com) x +

← → ↻ ⓘ Not secure | omayo.blogspot.com

Text Box with Preloaded Text

Selenium WebDriver

Opens In New Window Link

SeleniumTutorial

Enabled Button

Button2

Disabled Button

Button1

Disabled Text Box

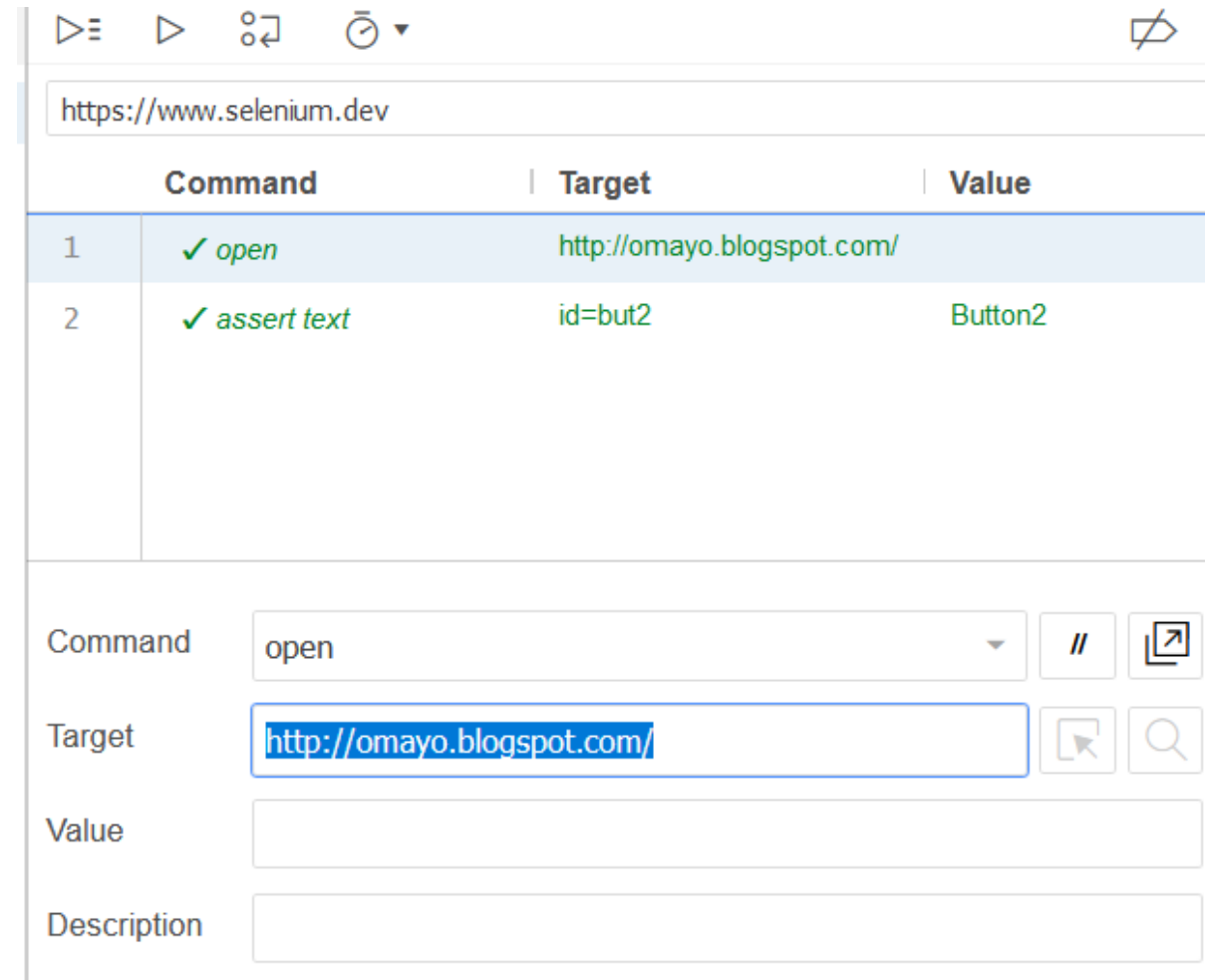
Table

Name	Age	Place
------	-----	-------

Checking the text on this button is a per our requirements

‘**assert text**’ to the Command box field, enter the id locating strategy **id=but2** into the Target box field and the text ‘Button2’ which should be between the HTML tags of the located UI element

Change open to : <http://omayo.blogspot.com/>



The screenshot displays the Selenium IDE interface. At the top, the address bar shows 'https://www.selenium.dev'. Below it is a table of test steps:

	Command	Target	Value
1	✓ open	http://omayo.blogspot.com/	
2	✓ assert text	id=but2	Button2

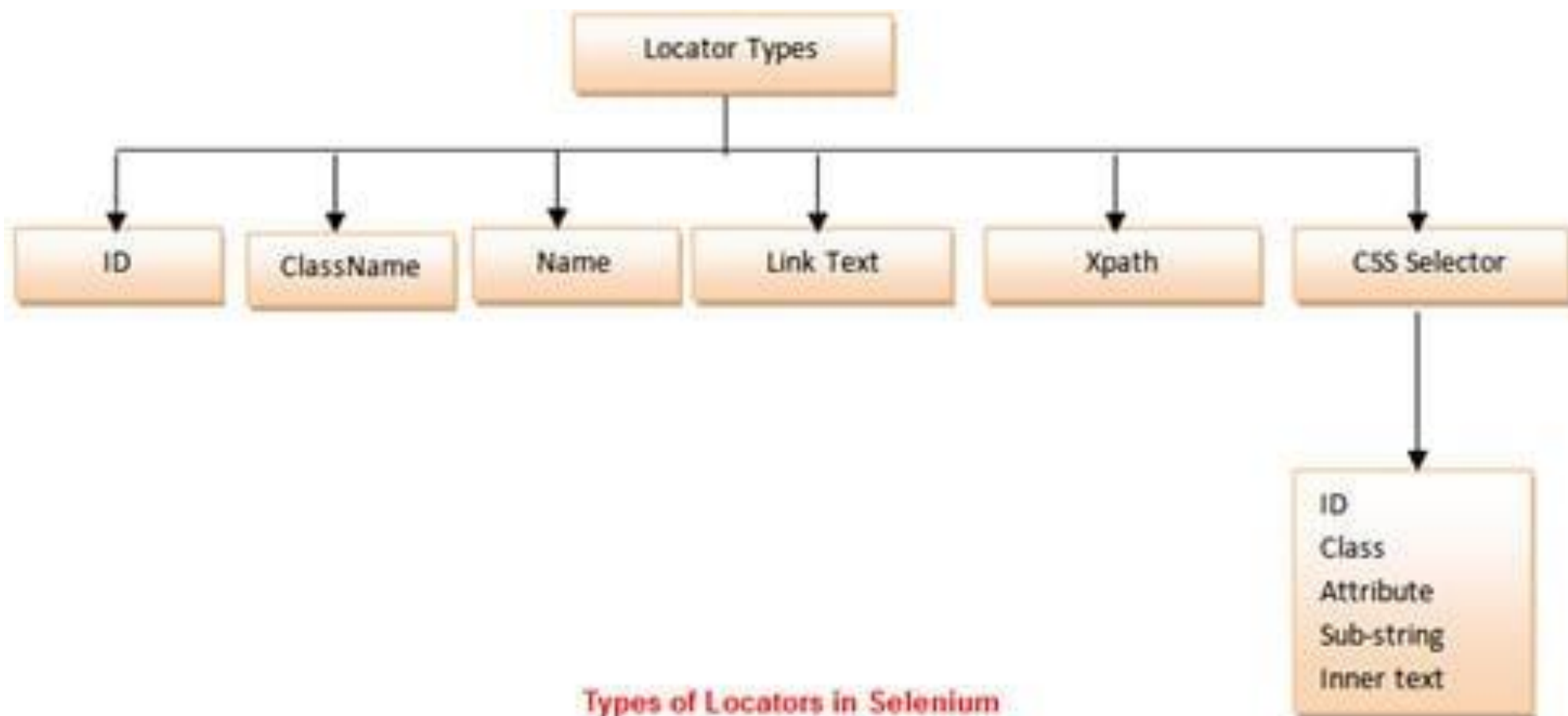
Below the table, the 'Command' field is set to 'open'. The 'Target' field contains 'http://omayo.blogspot.com/' and is highlighted with a blue selection box. The 'Value' and 'Description' fields are currently empty.

There is a diverse range of web elements. **The most common amongst them are:**

- . Text box
- . Button
- . Drop Down
- . Hyperlink
- . Check Box
- . Radio Button

Types of Locators

Identifying these elements has always been a very tricky subject and thus it requires an accurate and effective approach. Thereby, we can assert that more effective the locator, more stable will be the automation script. Essentially every Selenium command requires locators to find the web elements. Thus, to identify these web elements accurately and precisely we have different types of locators.



Types of Locators in Selenium

Using ID as a Locator

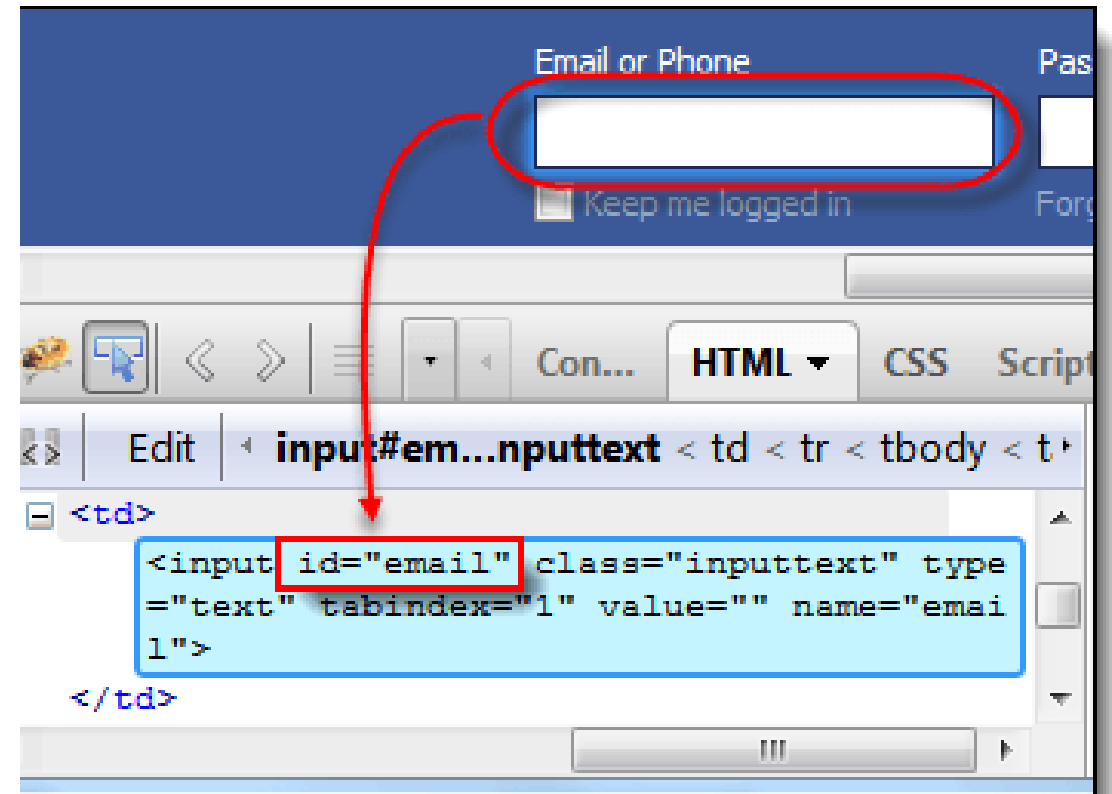
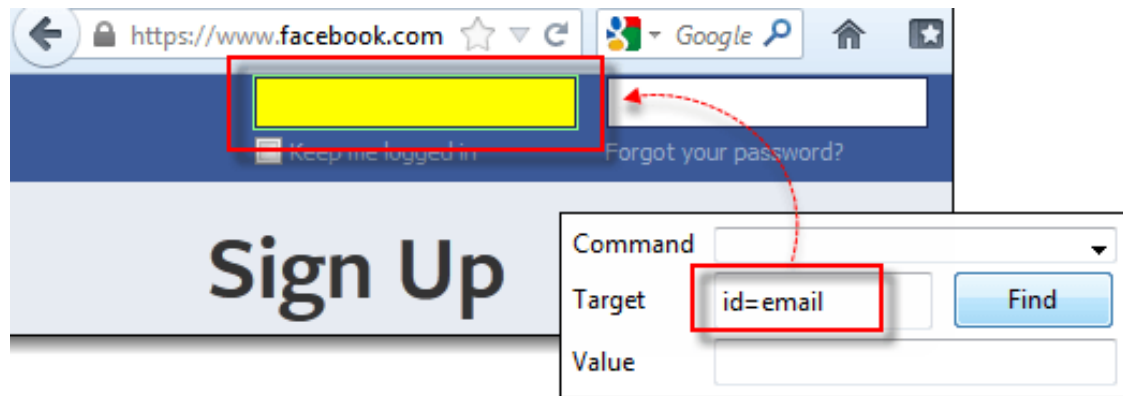
The best and the most popular method to identify web element is to use ID. The ID of each element is alleged to be unique.



Application under test: <http://demo.guru99.com/test/facebook.html>

This is the most common way of locating elements since ID's are supposed to be unique for each element.

Target Format: `id=id of the element`



Locating by CSS Selector – Tag and Class

Syntax	Description
css=tag.class	<ul style="list-style-type: none">• tag = the HTML tag of the element being accessed• . = the dot sign. This should always be present when using a CSS Selector with class• class = the class of the element being accessed

Email or Phone Password

☐ Keep me logged in [Forgot your password?](#)

```
<td>  
<input id="email" class="inputtext" type="text" tabindex="1" value="" name="email" style="background-color: rgb(255, 255, 255);">  
</td>
```

Email or Phone

☐ Keep me logged in

Command	
Target	css=input.inputtext
Value	

Find

Open <http://omayo.blogspot.com/>

click at `css=input.gsc-search-button` 10,20

Locating by Name

Locating elements by name are very similar to locating by ID, except that we use the “**name=**” prefix instead.

Target Format: `name=name of the element`

The screenshot shows a web form with a yellow header "Find A Flight". Below the header is a text area with the message: "Registered users can **sign-in** here to find the lowest fare on participating airlines." Below this are two input fields: "User Name:" and "Password:". A yellow "Sign-In" button is positioned below the "Password:" field. A red arrow originates from the "User Name:" input field and points to the HTML code in the DOM inspector below. The DOM inspector shows the following code:

```
<td width="112">  
<input type="text" size="10" name="userName">  
</td>
```

The diagram illustrates the process of locating an element by name. It shows a web form with a yellow header "Find A Flight". Below the header is a text area with the message: "Registered users can **sign-in** here to find the lowest fare on participating airlines." Below this are two input fields: "User Name:" and "Password:". A yellow "Sign-In" button is positioned below the "Password:" field. A red arrow originates from the "User Name:" input field and points to the "name=userName" text in the "Target" field of a tool interface. The tool interface has a "Command" dropdown, a "Target" field containing "name=userName", and a "Value" field. A "Find" button is located to the right of the "Target" field.

Locating by Link Text

This type of CSS locator in Selenium applies only to hyperlink texts. We access the link by prefixing our target with “link=” and then followed by the hyperlink text.



```
'mouseover')" onmouseout="changeStyle(th
is, 'mouseout')">
<a href="mercuryregister.php">REGISTER<
/a>
</td>
+ <td class="mouseout" width="73" height="
```

Command	Target	Value
	link=REGISTER	

Command	<input type="text"/>	
Target	<input type="text" value="link=REGISTER"/>	<input type="button" value="Find"/>
Value	<input type="text"/>	

Locating by CSS Selector

CSS Selectors in Selenium are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes. Locating by CSS Selectors in Selenium is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.

CSS Selectors in Selenium have many formats, but we will only focus on the most common ones.

- Tag and ID
- Tag and class
- Tag and attribute
- Tag, class, and attribute
- Inner text

Locating by CSS Selector – Tag and ID

Again, we will use Facebook's Email text box in this example. As you can remember, it has an ID of "email," and we have already accessed it in the "Locating by ID" section. This time, we will use a Selenium CSS Selector with ID in accessing that very same element.

Syntax	Description
css=tag#id	<ul style="list-style-type: none">• tag = the HTML tag of the element being accessed• # = the hash sign. This should always be present when using a Selenium CSS Selector with ID<ul style="list-style-type: none">• id = the ID of the element being accessed

Email or Phone Password

☐ Keep me logged in [Forgot your password?](#)

```
<td>  
<input id="email" class="inputtext" type="text"  
tabindex="1" value="" name="email">  
</td>
```

Email or Phone

☐ Keep me logged in

Command	
Target	css=input#email Find
Value	

Locating by CSS Selector – Tag and Class

Locating by CSS Selector in Selenium using an HTML tag and a class name is similar to using a tag and ID, but in this case, a dot (.) is used instead of a hash sign.

Syntax	Description
css=tag.class	<ul style="list-style-type: none">• tag = the HTML tag of the element being accessed• . = the dot sign. This should always be present when using a CSS Selector with class• class = the class of the element being accessed

Email or Phone Password

☐ Keep me logged in [Forgot your password?](#)

```
<td>  
<input id="email" class="inputtext" type="text" tabindex="1" value="" name="email" style="background-color: rgb(255, 255, 255);">  
</td>
```

Email or Phone

☐ Keep me logged in

Command

Target Find

Value

Locating by CSS Selector – Tag and Attribute

This strategy uses the HTML tag and a specific attribute of the element to be accessed.

Syntax	Description
css=tag[attribute=value]	<ul style="list-style-type: none">• tag = the HTML tag of the element being accessed• [and] = square brackets within which a specific attribute and its corresponding value will be placed• attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID.• value = the corresponding value of the chosen attribute.

Contact Information

First Name:

Last Name:

Phone:

```
<td>  
  <input size="20" name="lastName" maxleng  
  th="60">  
</td>
```

First Name:

Last Name:

Phone:

Command	<input type="text"/>
Target	<input type="text" value="css=input[name=lastName]"/>
Value	<input type="text"/>

Find

Locating by CSS Selector – tag, class, and attribute

Syntax	Description
<code>css=tag.class[attribute=value]</code>	<ul style="list-style-type: none">• tag = the HTML tag of the element being accessed• . = the dot sign. This should always be present when using a CSS Selector with class• class = the class of the element being accessed• [and] = square brackets within which a specific attribute and its corresponding value will be placed• attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID.• value = the corresponding value of the chosen attribute.

Email or Phone	Password
<input type="text"/>	<input type="password"/>
<input type="checkbox"/> Keep me logged in	Forgot your password?

```
<td>
  <input id="email" class="inputtext" type=
    ="text" tabindex="1" value="" name="emai
    1">
</td>
<td>
  <input id="pass" class="inputtext" type=
    ="password" tabindex="2" name="pass">
</td>
```


Email or Phone	Password
<input type="text"/>	<input type="password"/>

Command	<input type="text"/>	
Target	<input type="text" value="css=input.inputtext[tabindex=1]"/>	<input type="button" value="Find"/>
Value	<input type="text"/>	

Locating by CSS Selector – inner text

As you may have noticed, HTML labels are seldom given id, name, or class attributes. So, how do we access them? The answer is through the use of their inner texts. **Inner texts are the actual string patterns that the HTML label shows on the page.**

Syntax	Description
<code>css=tag:contains("inner text")</code>	<ul style="list-style-type: none">• tag = the HTML tag of the element being accessed• inner text = the inner text of the element

```
<td align="right">  
  <font size="2" face="Arial, Helvetica,  
  sans-serif">Password: </font>  
</td>
```

This is the inner text

User

Name:

Password:

User

Name:

Password:

Command	<input type="text"/>	▼
Target	css=font:contains("Password:")	Find
Value	<input type="text"/>	

Back to commands....

type is one of the commands in Selenium IDE.

The purpose of **type** command in Selenium IDE, is to type any given text into the text fields in the application.

Open <http://omayo.blogspot.com/>

type name=q SomeText

click is one of the commands in Selenium IDE.

The purpose of **click** command in Selenium IDE, is to click on any UI element in the application.

Open <http://omayo.blogspot.com/>

click id=checkbox2

click at is one of the commands in Selenium IDE.

The purpose of **click at** command in Selenium IDE, is to click on any UI element at the given x & y coordinate position of the UI element.

set window as is one of the commands in Selenium IDE.

The purpose of **set window size** command in Selenium IDE, is to resize the browser window.

command set window size

target 300X200

close is one of the commands in Selenium IDE.

The purpose of **close** command in Selenium IDE, is to close the browser window.

select is one of the commands in Selenium IDE.

The purpose of **select** command in Selenium IDE, is to select an option from the dropdown field.

```
open    http://omayo.blogspot.com/
```

```
select  id=drop1      label=doc 4
```

add selection is one of the commands in Selenium IDE.

The purpose of **add selection** command in Selenium IDE, is to select options from the Multi-selection box field.

```
add selection id=multiselect1    Volvo
```

remove selection is one of the commands in Selenium IDE.

The purpose of the **remove selection** command is to remove the selection of selected options in the multi-selection box field.

Project: OmayoProj

Tests ▾

+

Search tests...



TestOne



http://omayo.blogspot.com/

Command

Target

Value

1

open

http://omayo.blog
spot.com/

2

add selection

id=multiselect1

Volvo

3

add selection

id=multiselect1

Swift

4

add selection

id=multiselect1

Hyundai

5

add selection

id=multiselect1

Audi

6

remove selection

id=multiselect1

Volvo



check is one of the commands in Selenium IDE.

The purpose of **check** command in Selenium IDE, is to select the radio button.

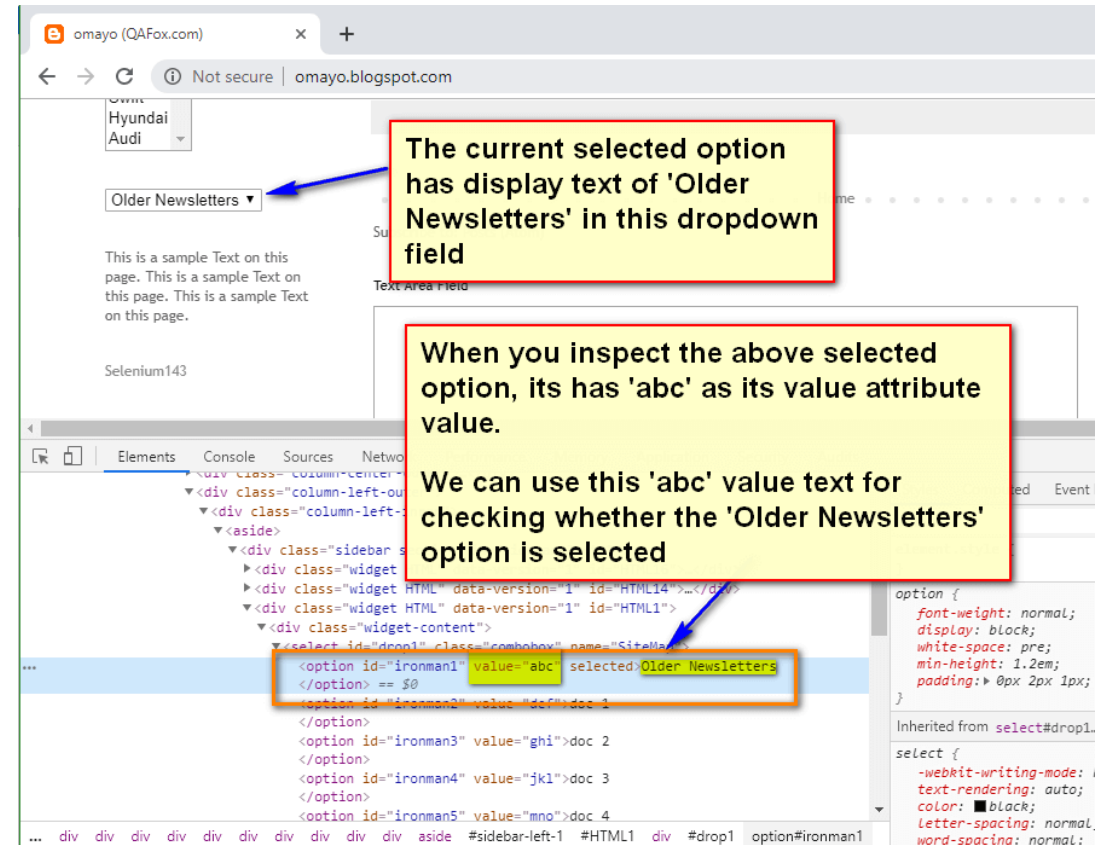
check id=radio1

similarly, uncheck.

assert selected value is one of the commands in Selenium IDE.

The purpose of **assert selected value** command in Selenium IDE, is to check whether the given option is selected in the dropdown field.

assert selected value id=drop1 abc



assert selected label is one of the commands in Selenium IDE.

The purpose of **assert selected label** command in Selenium IDE, is to check whether the given option is selected in the dropdown field using its label text or display text.

assert selected value id=drop1 abc Older Newsletters

assert checked is one of the commands in Selenium IDE.

The purpose of **assert checked** command in Selenium IDE, is to check whether the given checkbox is in selected state.

assert checked id=checkbox1

assert editable is one of the commands in Selenium IDE.

The purpose of **assert editable** command in Selenium IDE, is to check whether the given field is in an editable state.

assert editable name=q

assert element present is one of the commands in Selenium IDE.

The purpose of **assert element present** command in Selenium IDE, is to check whether the given UI element is present on the page.

```
assert element present id=but2
```

assert alert is one of the commands in Selenium IDE.

The purpose of **assert alert** command in Selenium IDE, is to check whether the required alert is displayed on the page.

```
click id=alert1
```

```
assert alert Hello
```

assert confirmation is one of the commands in Selenium IDE.

The purpose of **assert confirmation** command in Selenium IDE, is to check whether the required confirmation dialog is displayed on the page.

```
click    id=confirm
```

```
assert confirm    Press OK to confirm
```

assert prompt is one of the commands in Selenium IDE.

The purpose of **assert prompt** command in Selenium IDE, is to check whether the prompt dialog is displayed during execution.

```
open    http://omayo.blogspot.com/
```

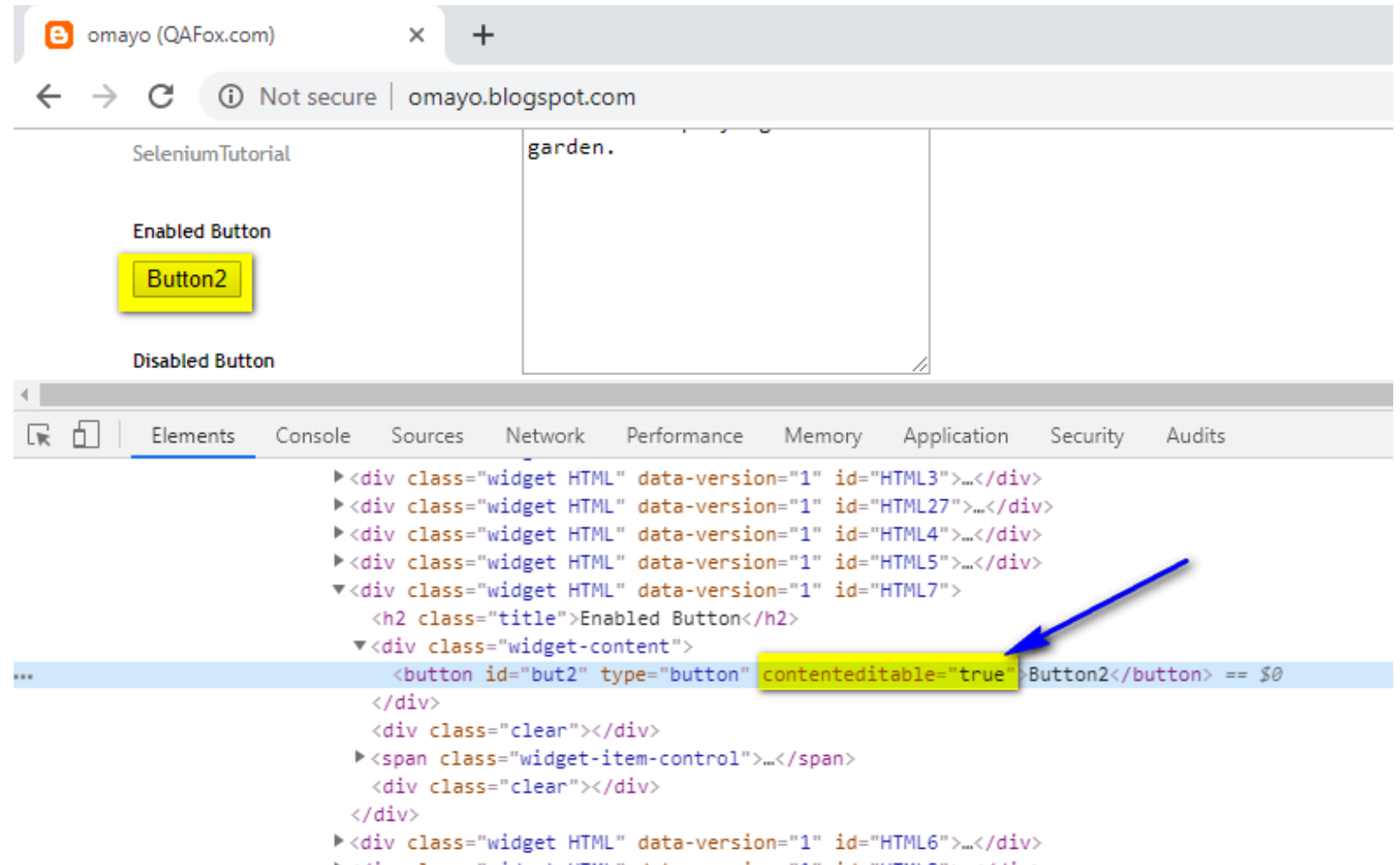
```
click    id=prompt
```

```
assert prompt    What is your name?
```

edit content is one of the commands in Selenium IDE.

The purpose of **edit content** command in Selenium IDE, is to change the value of the given UI element.

Note: This option only with the UI elements who's **contenteditable** attribute value is set to 'true' as shown below:



By default, most of the UI element will have the above attribute 'contenteditable' set to 'false' and hence this Selenium IDE command will give error on execution.

edit content id=but2 QAFox

execute script is one of the commands in Selenium IDE.

The purpose of **execute script** command in Selenium IDE, is to execute the JavaScript code in Selenium IDE.

Execute script **alert("Hello World!")**

- mouse down, mouse move at and mouse up are commands in Selenium IDE.
- The purpose of the mouse down command is to perform mouse left click operation, mouse move at command is to move the holder UI element to the target element and the mouse up command is to release the mouse click to release the so far held UI element to the desired element in Selenium IDE.

open	https://jqueryui.com/droppable/	
select frame	index=0	
mouse down	id=draggable	
mouse move at	id=droppable	
mouse up	id=droppable	

submit is one of the commands in Selenium IDE.

The purpose of the **submit** command is to submit any form on the page say Login, Signup forms, etc.

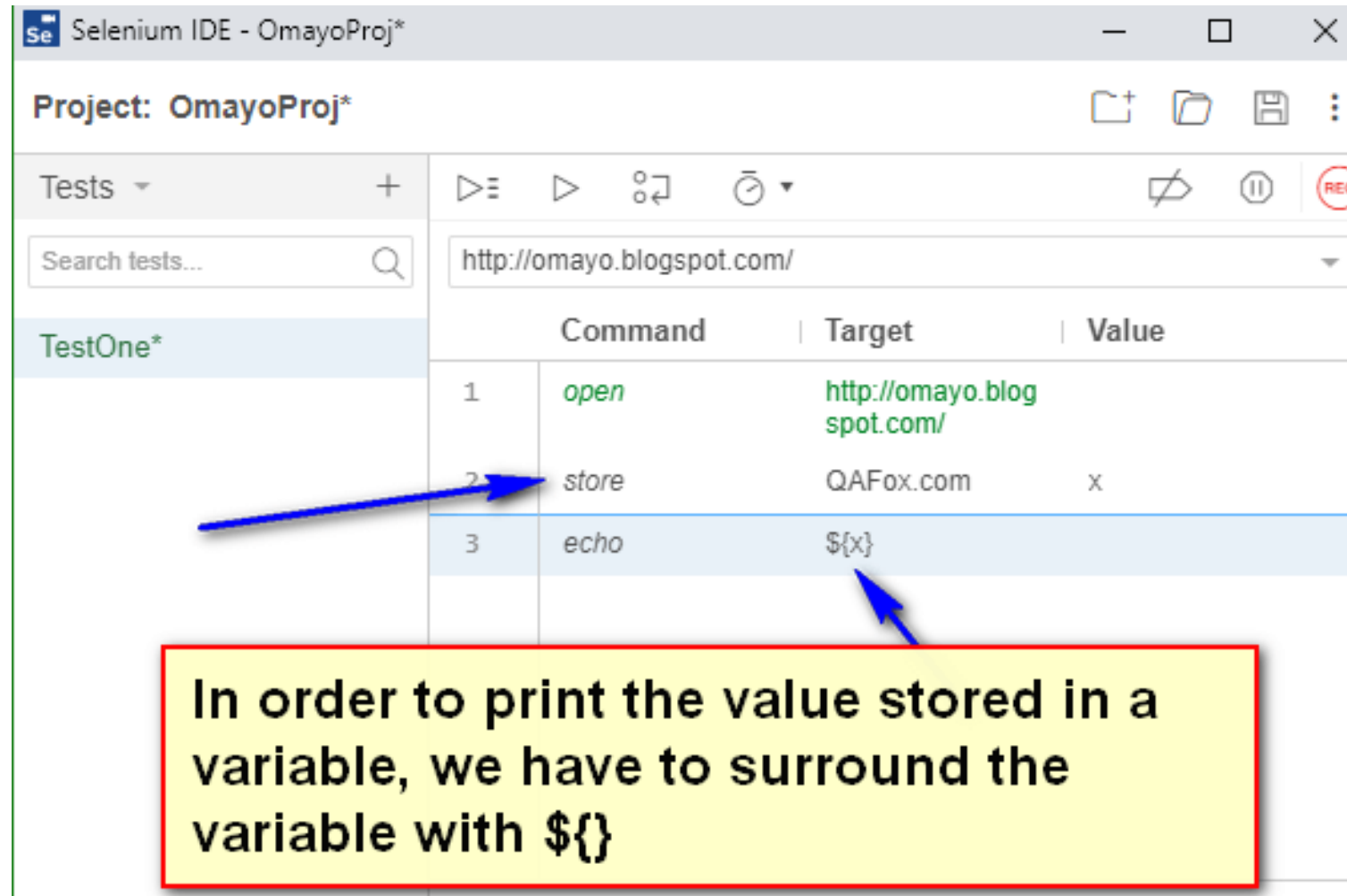
open	http://tutorialsninja.com/demo/index.php?route=account/login	
submit	css=form[action\$='login']	

store is one of the commands in Selenium IDE.

The purpose of the **store** command is to store any text into a variable in Selenium IDE.

open	http://omayo.blogspot.com/	
store	QAFox.com	x
echo	\${x}	

After incorporating the commands in the above table, Our Selenium IDE should look like below:



The screenshot shows the Selenium IDE interface for a project named 'OmayoProj*'. The 'Tests' panel on the left shows 'TestOne*'. The main area displays a table of commands for the selected test. The table has columns for 'Command', 'Target', and 'Value'. The commands are: 1. 'open' with target 'http://omayo.blogspot.com/' and no value; 2. 'store' with target 'QAFOX.com' and value 'x'; 3. 'echo' with target '\$ {x}' and no value. A blue arrow points to the 'store' command, and another blue arrow points to the '\$ {x}' target in the 'echo' command. A yellow text box at the bottom contains the following text:

In order to print the value stored in a variable, we have to surround the variable with \$ {}

Run the test and observe that the test will be passed and the value stored in the variable 'x' will be printed in the 'Log' tag as shown below:

Selenium IDE - OmayoProj*

Project: OmayoProj*

Tests +

Search tests...

TestOne*

http://omayo.blogspot.com/

	Command	Target	Value
1	open	http://omayo.blog spot.com/	
2	store	QAfox.com	x
3	echo	\${x}	

Command

Target

Value

Description

Log Reference

Running 'TestOne' 14:32:06

1. open on http://omayo.blogspot.com/ OK 14:32:06

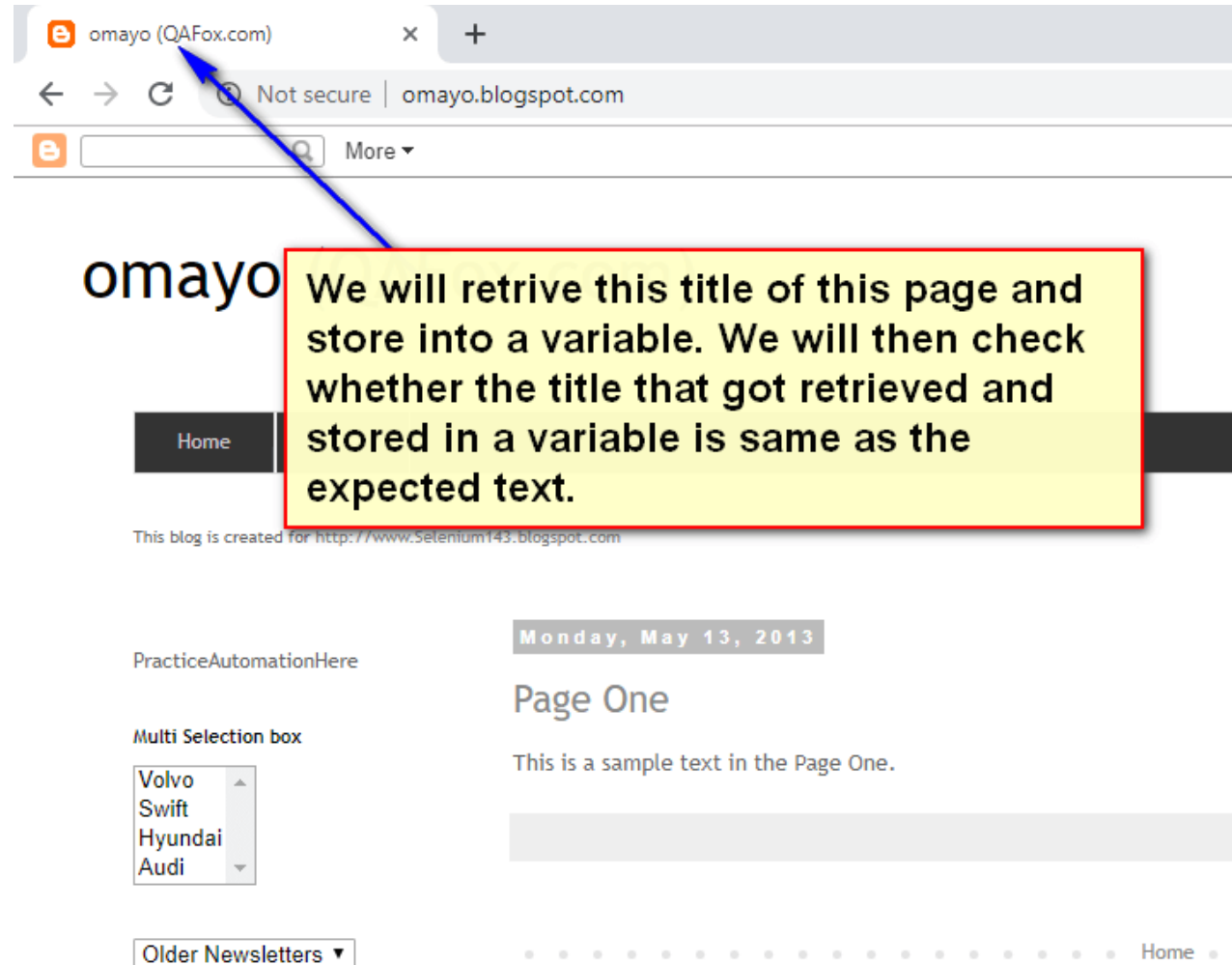
2. store on QAfox.com with value x OK 14:32:06

echo: QAfox.com 14:32:06

'TestOne' completed successfully 14:32:07

assert is one of the commands in Selenium IDE.

The purpose of the **assert** command is to check the value stored in a variable is according to the expected result



Command	Target	Value
open	http://omayo.blogspot.com/	
store title		x
assert	x	omayo (QAFox.com)

wait for element editable is one of the commands in Selenium IDE.

The purpose of the **wait for element editable** command is to wait for the element to get editable. i.e. If any required element on the application is by default in non-editable state and gets editable in some time, we can use **wait for element editable** command to wait for the element to get editable, before editing the element.

omayo (QAFox.com) x +

← → ↻ ⓘ Not secure | omayo.blogspot.com

Button3

Phoenix to San Francisco

Disable Enable Button

My Button

Click the button try it button to disable the button after 3 seconds.

Try it

Button X Button Y

Double click Here

Mr Option: ☐

The above Mr Option will be enabled in 10 seconds after clicking on below Check this button

Check this

LoginSection

Login page

Simple Login Page

Username Password Login Cancel

www.selenium-by-arun.blogspot.com

This option is by default in non-editable state and gets enabled after 10 seconds after clicking the below 'Check this' button.

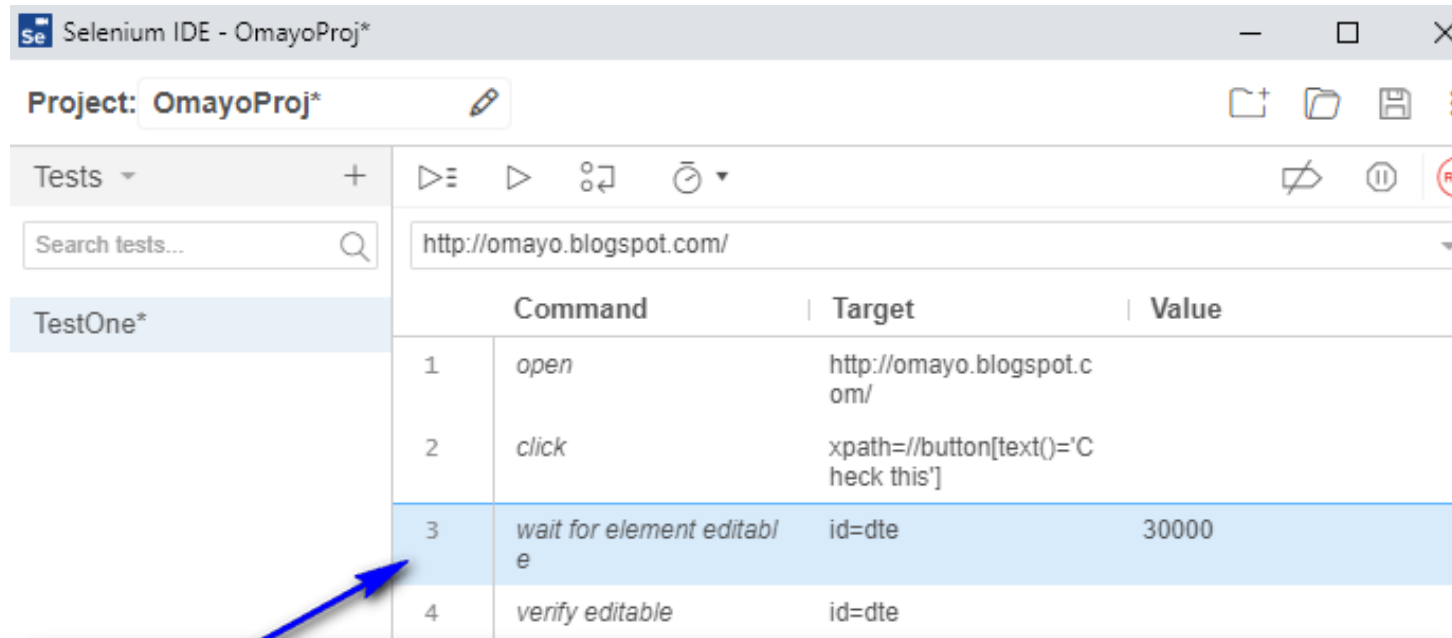
We will use 'wait for element editable' command for waiting for this checkbox option to get enabled.

Follow the below steps for practicing **wait for element editable** command in Selenium IDE

1) Let's write the below code in Selenium IDE to wait for the 'checkbox' option to get editable after clicking on the 'Check this' button:

Command	Target	Value
open	http://omayo.blogspot.com/	
click	xpath=//button[text()='Check this']	
wait for element editable	id=dte	30000
verify editable	id=dte	

After incorporating the commands in the above table, Our Selenium IDE should look like below:



The screenshot shows the Selenium IDE interface for a project named 'OmayoProj*'. The URL bar is set to 'http://omayo.blogspot.com/'. The test 'TestOne*' contains the following commands:

	Command	Target	Value
1	open	http://omayo.blogspot.com/	
2	click	xpath=//button[text()='Check this']	
3	wait for element editable	id=dte	30000
4	verify editable	id=dte	

A blue arrow points to the third command, 'wait for element editable'.

This will wait for the maximum of 30000 milli seconds (i.e. 30 seconds) for the required checkbox option to get enabled.

As the element gets enabled in 10 seconds, this command only waits for 10 seconds instead of 30 seconds.

2) Run the test and observe that Selenium IDE waits for the checkbox option to get enabled and then verifies whether the checkbox option is enabled state as shown below:

The screenshot displays the Selenium IDE interface with a browser window on the left and the IDE control panel on the right. The browser window shows a web page with several buttons: 'Button3', 'My Button', 'Try it', 'Button X', 'Button Y', 'Double click Here', and 'Check this'. A blue arrow points to the 'Mr Option: ☐' checkbox, which is currently unchecked. The IDE control panel shows the 'TestOne*' test with the following commands:

	Command	Target	Value
1	open	http://omayo.blogspot.com/	
2	click	xpath=//button[text()='Check this']	
3	wait for element editable	id=dte	30000
4	verify editable	id=dte	

The bottom of the IDE shows the 'Log' tab with the following execution details:

```
Running 'TestOne' 15:37:39
1. open on http://omayo.blogspot.com/ OK 15:37:40
2. click on xpath=//button[text()='Check this'] OK 15:37:40
3. waitForElementEditable on id=dte with value 30000 OK 15:37:42
4. verifyEditable on id=dte OK 15:37:52
'TestOne' completed successfully 15:37:53
```

wait for element visible is one of the commands in Selenium IDE.

The purpose of the **wait for element visible** command is to wait for the element to get displayed on the page i.e. If any required element on the application is by default not displayed and gets displayed in some time, we can use **wait for element visible** command to wait for the element to get displayed before performing any operation on the element

Command	Target	Value
open	http://omayo.blogspot.com/	
wait for element visible	id=delayedText	30000
store text	id=delayedText	x
echo	\${x}	

Selenium IDE - OmayoProj*

Project: OmayoProj*

Tests +

Search tests...

TestOne*

http://omayo.blogspot.com/

	Command	Target	Value
1	open	http://omayo.blogspot.com/	
2	wait for element visible	id=delayedText	30000
3	store text	id=delayedText	x
4	echo	\${x}	

This will wait for the maximum of 30000 milliseconds (i.e. 30 seconds) for the required text to get displayed on the page.

As the required text gets displayed in 10 seconds after page load, this command only waits for 10 seconds instead of 30 seconds.

store json is one of the commands in Selenium IDE.

The purpose of the **store json** command is to retrieve and store json content into a variable in Selenium IDE.

Command	Target	Value
open	http://omayo.blogspot.com/	
store json	{ "glossary": { "title": "example glossary", "GlossDiv": { "title": "S", "GlossList": { "GlossEntry": { "ID": "SGML", "SortAs": "SGML", "GlossTerm": "Standard Generalized Markup Language", "Acronym": "SGML", "Abbrev": "ISO 8879:1986", "GlossDef": { "para": "A meta-markup language, used to create markup languages such as DocBook.", "GlossSeeAlso": ["GML", "XML"] }, "GlossSee": "markup" } } } } }	x
echo	\${x}	

send keys is one of the commands in Selenium IDE.

The purpose of the **send keys** command is to press the required keyboard key using any of the below-specified keyboard keycodes:

[su_table alternate="no"]

Key Code	Function
KEY_LEFT	Navigation Left
KEY_UP	Navigation Up
KEY_RIGHT	Navigation Right
KEY_DOWN	Navigation Down
KEY_PGUP or KEY_PAGE_UP	Page up
KEY_PGDN or KEY_PAGE_DOWN	Page down
KEY_BKSP or KEY_BACKSPACE	Backspace
KEY_DEL or KEY_DELETE	Delete
KEY_ENTER	Enter
KEY_TAB	Tab

We will enter 'qafox.com' text into this search box field and press 'Enter' keyboard key to perform the Search operation on this page.



Search This Blog

qafox.com

Search

Radio options

☐ Male

☐ Female

AlertDemo

ClickToGetAlert

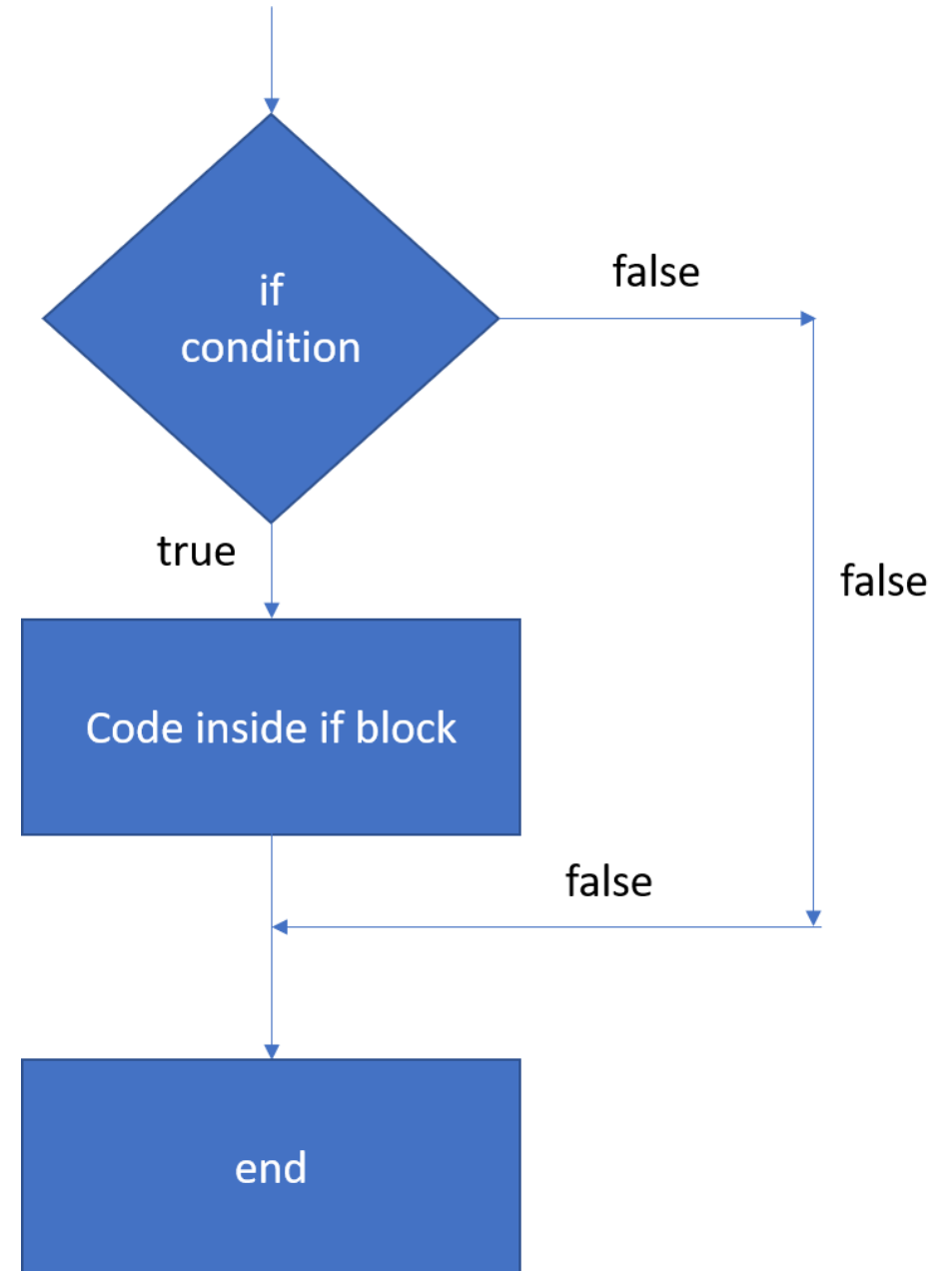
Command	Target	Value
open	http://omayo.blogspot.com/	
type	name=q	qafox.com
send keys	name=q	\${KEY_ENTER}

if is one of the commands in Selenium IDE and we need to end it with **end** command.

if and **end** are two commands in Selenium IDE which can be used together.

The purpose of the **if** command is to check whether the given condition is true or false. If the condition results in true, the Selenium IDE statements inside the **if** and **end** commands will be executed. If the condition results in false, the Selenium IDE statements inside the **if** and **end** commands won't be executed (i.e. skipped from execution).

The **if** and **end** commands will work as depicted below:



Let's create a test in Selenium IDE for retrieving the title of the page, adding **if** condition to check whether the retrieved title is equal to the expected title text and printing the result using the below Selenium IDE code:

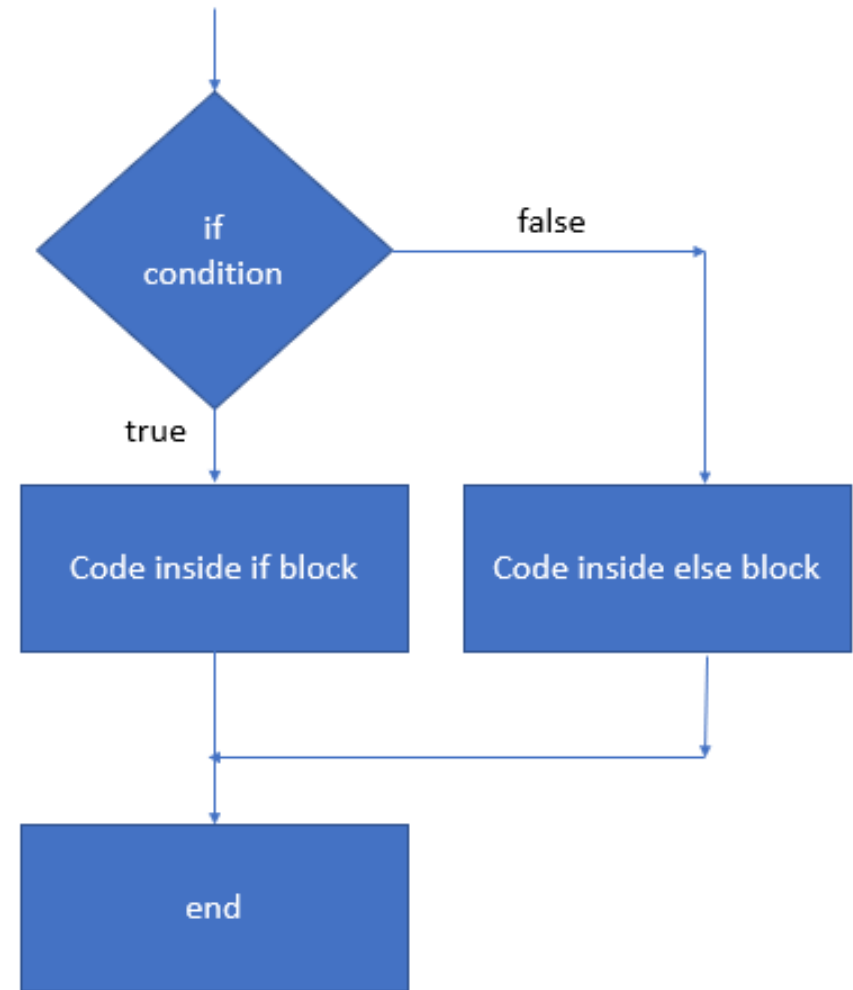
Command	Target	Value
open	http://omayo.blogspot.com/	
store title		x
if	\${x} === "omayo (QAFox.com)"	
echo	Correct title is displayed on the web page	
end		

else is one of the commands in Selenium IDE and we need to use it with **if** and **end** commands

else is the command in Selenium IDE will be executed when the **if** condition results in false.

The purpose of the **if** command is to check whether the given condition is true or false. If the condition results in true, the Selenium IDE statements inside the **if** block will be executed. If the condition results in false, the Selenium IDE statements inside the **else** block will be executed.

The **else** command will work as depicted below:



1) Let's create a test in Selenium IDE for retrieving the title of the page, adding **if** condition to check whether the retrieved title is equal to the expected title text, **else** to execute its block when the retrieved title is not equal to the expected title text and printing the result using the below Selenium IDE code:

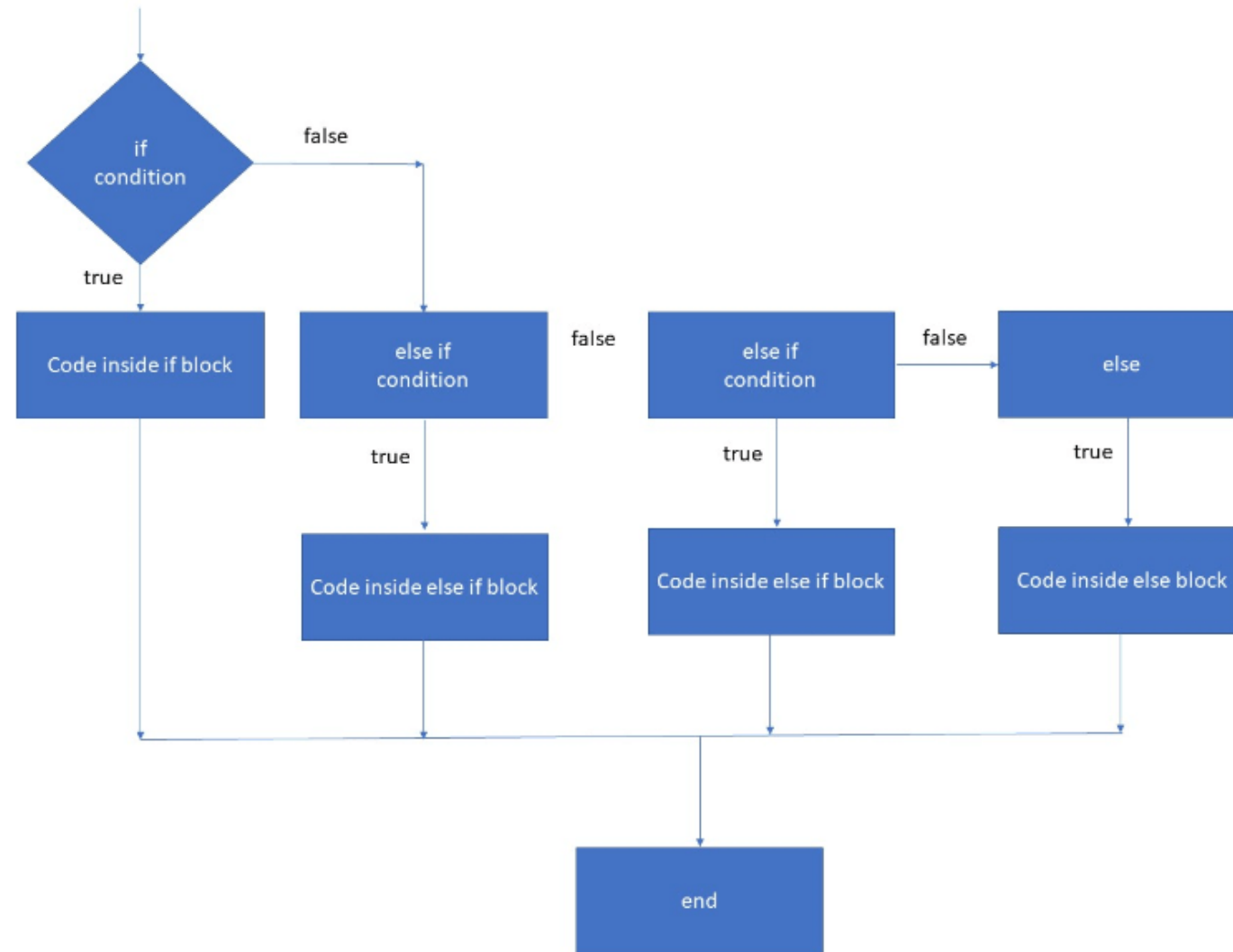
Command	Target	Value
open	http://omayo.blogspot.com/	
store title		x
if	\${x} === "xyz"	
echo	Correct title is displayed on the web page	
else		
echo	Wrong title is displayed on the web page	
end		

else if is one of the commands in Selenium IDE and we need to use it with **if**, **else** and **end** commands

else if is the command in Selenium IDE will be executed when the **if** condition results in false and we have another condition to be validated before going to the **else** block.

The purpose of the **if** command is to check whether the given condition is true or false. If the condition results in true, the Selenium IDE statements inside the **if** block will be executed. If the condition results in false, the **if** block will be skipped and the next **else if** condition will be checked for true or false. If the **else if** condition is true, the statements inside the **else if** block will be executed. And if the **else if** condition is false, the **else if** block will be skipped and the next **else if** condition or **else** block will be executed.

The **else if** command will work as depicted below:



1) Let's create a test in Selenium IDE for retrieving the title of the page, adding **if** condition to check whether the retrieved title is equal to the expected title one text, **else if** condition to check whether the retrieved title is equal to the expected title two text, **else** to execute its block when the retrieved title is not equal to the expected title one and title two texts and printing the result using the below Selenium IDE code:

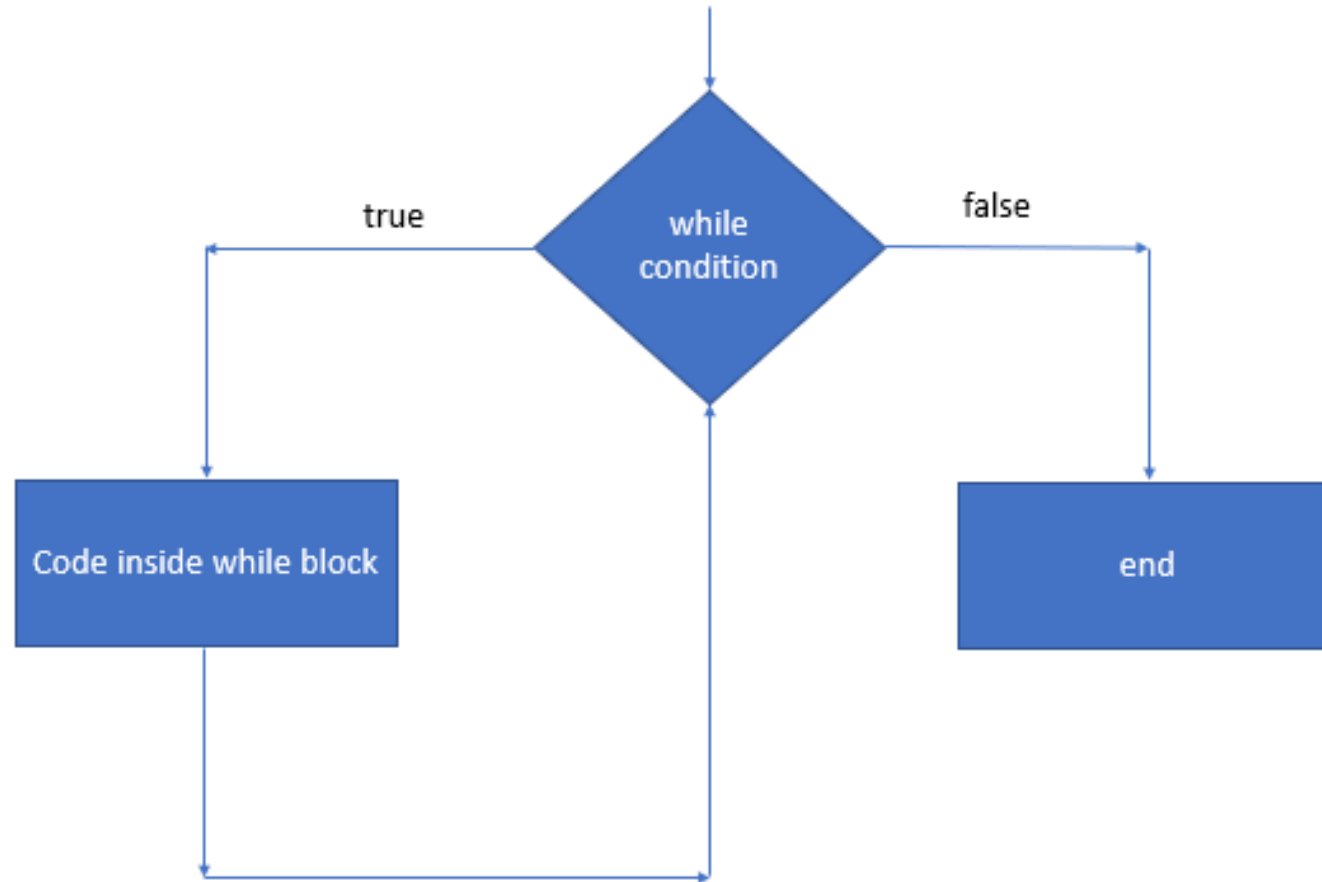
Command	Target	Value
open	http://omayo.blogspot.com/	
store title		x
if	\${x} == "xyz"	
echo	Correct title one is displayed on the web page	
else if	\${x} == "omayo (QAFox.com)"	
echo	Correct title two is displayed on the web page	
else		
echo	Wrong title is displayed on the web page	
end		

while is one of the commands in Selenium IDE and we can end it with **end** command.

while is the command in Selenium IDE used for executing a set of same statements multiple times until the **while** condition becomes false.

We will execute a set of statements inside the **while** block multiple times until the **while** condition becomes false. i.e. We will execute the same set of statements inside the **while** block until the value inside the variable x is smaller than 5. Initially, we will assign the value 1 to the variable x and thereafter increment the value stored in variable x by 1 until the while condition becomes false (i.e. when the value of x becomes 5, the **while** condition $x < 5$ will become false).

The **while** command will work as depicted below:



As you can see in the above flow diagram, the block of code inside the **while** block will be executed multiple times until the while condition become false.

1) Let's create a test in Selenium IDE for printing the value stored in the variable x until the **while** condition become false as shown below:

Command	Target	Value
open	http://omayo.blogspot.com/	
execute script	return 1	x
while	\${x}<5	
echo	\${x}	
execute script	return \${x}+1	x
end		

- do and repeat if are the commands in Selenium IDE, which needs to be used together.
- do and repeat if are the command in Selenium IDE used for executing a set of same statements multiple times until the repeat if condition becomes false.
- Difference between while and do commands
- In while command, the condition to be checked for entering into the iterative loop block will be at the beginning. Whereas in do command, we will check the condition at the end of the iterative loop block using repeat if command.

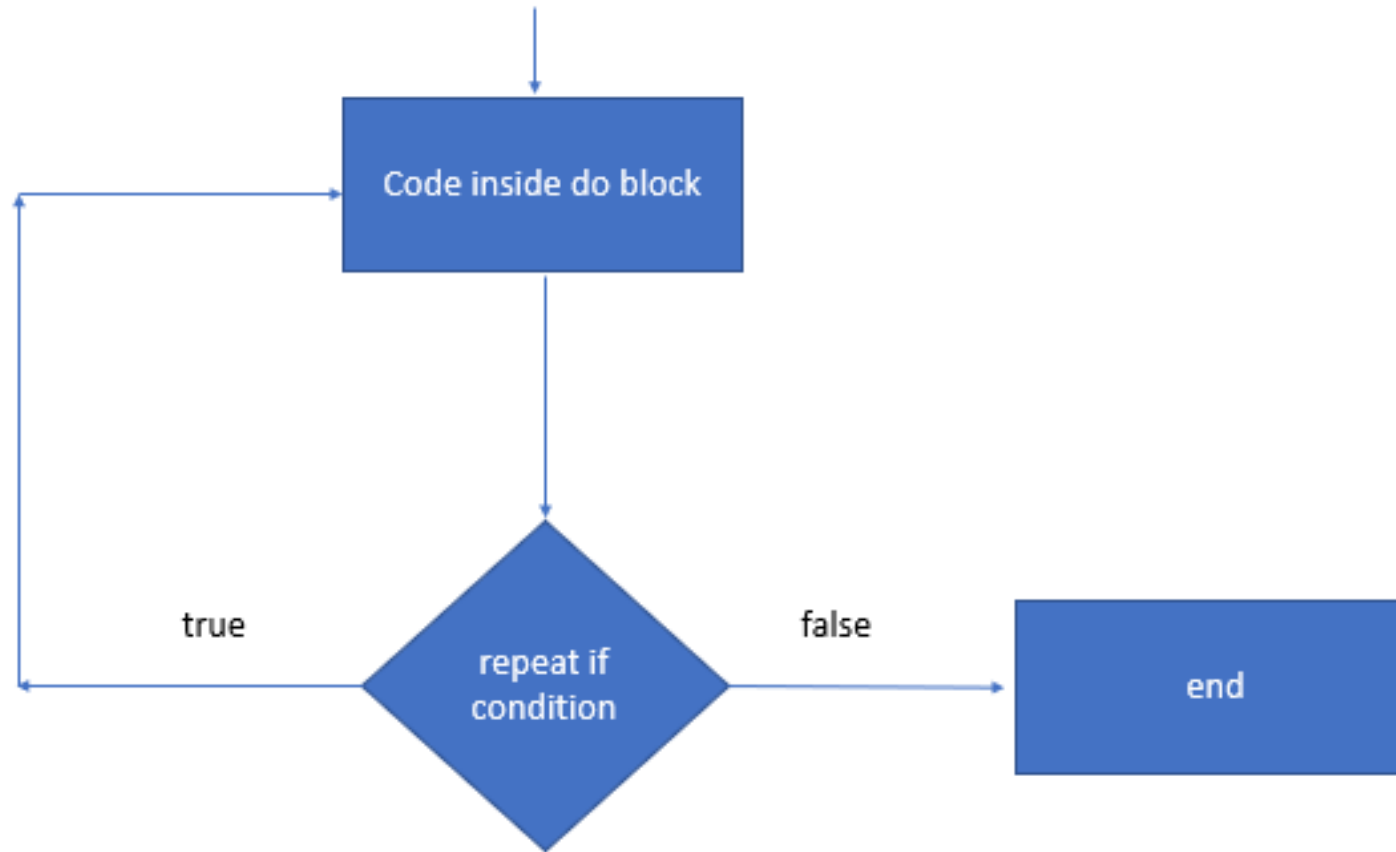
- ---

while condition
statements inside the while block
end

do
statements inside the do block
repeat if condition

We will execute a set of statements inside the **do** block multiple times until the **repeat if** condition becomes false. i.e. we will execute the same set of statements inside the **do** block until the value inside the variable x is smaller than 5. Initially, we will assign the value 1 to the variable x and thereafter increment the value stored in variable x by 1 until the **repeat if** condition becomes false (i.e. when the value of x becomes 5, the **repeat if** condition $x < 5$ will become false).

The **do** command will work as depicted below:



As you can see in the above flow diagram, the block of code inside the **do** block will be executed multiple times until the **repeat if** condition becomes false.

1) Let's create a test in Selenium IDE for printing the value stored in the variable x until the **repeat if** condition of **do** block become false as shown below:

Command	Target	Value
open	http://omayo.blogspot.com/	
execute script	return 1	x
do		
echo	\${x}	
execute script	return \${x}+1	x
repeat if	\${x}<5	

Project: OmayoProj*

Tests ▾



Search tests...



http://omayo.blogspot.com/

TestOne*

Command

Target

Value

1

open<http://omayo.blogspot.com/>

2

execute script

return 1

x

3

do

4

echo $\${x}$

5

*execute script*return $\${x}+1$

x

6

repeat if $\${x} < 5$

2) Click on 'Run current test' option and observe that the test got successfully executed and the value stored in the variable x will be printed multiple times under the Log tab.

Note: The initial value stored in the variable x is 1 (i.e. we have executed JavaScript statement 'return 1' as shown above using [execute script](#) command, for storing the value 1 into the variable x)

And also, we have incremented the value stored in the variable x by 1 every time we enter into the while block. (i.e. we have executed JavaScript statement 'return $\$ \{x\} + 1$ ' as shown above using [execute script](#) command, for incrementing the value stored in the variable by 1 every time we enter to the **do** block)

Selenium IDE - OmayoProj*

Project: OmayoProj*

Tests +

Search tests...

http://omayo.blogspot.com/

	Command	Target	Value
1	open	http://omayo.blogspot.com/	
2	execute script	return 1	x
3	do		
4	echo	\${x}	
5	execute script	return \${x}+1	x
6	repeat if	\${x} < 5	

Command

Target

Value

Description

Log Reference

3. do OK 16:54:36

echo: 1 16:54:36

5. executeScript on return \${x}+1 with value x OK 16:54:36

6. repeatIf on \${x} < 5 OK 16:54:37

echo: 2 16:54:37

echo: 3 16:54:37

echo: 4 16:54:38

'TestOne' completed successfully 16:54:38

for each is one of the commands in Selenium IDE.

for each is the command in Selenium IDE used for executing a set of same statements multiple times until all the values in the given array (i.e. multiple set of values stored in a single variable) are completed.

Command	Target	Value
open	http://omayo.blogspot.com/	
execute script	return ["Audi","Volvo","BMW"]	x
for each	x	itr
echo	\${itr}	
end		