Automation

By Shaumik Daityari, Community Contributor and Pradeep Krishnakumar, Manager - February 21, 2021

Automation Testing Selenium Selenium Webdriver

In an agile environment, developers emphasize pushing changes quickly. With every change that requires a modification to the front end, they need to run appropriate cross browser tests. While a small project may choose manual testing, an increasing number of browsers make a case for automation testing. In this post, we provide a step-by-step tutorial of web automation testing through Selenium and Python.

Selenium allows you to define tests and automatically detect the results of these tests on a pre-decided browser. A suite of Selenium functions enables you to create step-by-step interactions with a webpage and assess the response of a browser to various changes. You can then decide if the response of the browser is in line with what you expect.

This post assumes that you do not have prior knowledge of Selenium. However, basic knowledge of front-end concepts like DOM and familiarity with Python is expected.

Pre-requisites for running Selenium tests with

While the installation of Selenium makes the functionality available to you, you need additional drivers for it to be able to interface with a chosen web browser. The download links for the drivers are available here: Chrome, Edge, Firefox, and Safari. For the remainder of this tutorial, we will use the Chromedriver. Follow the link for the browser of your choice and download the driver for the compatible version.

People also read: How to run Selenium tests on Chrome using ChromeDriver

---

If you only plan to locally test Selenium, downloading the package and drivers should suffice. However, if you would like to set Selenium up on a remote server, you would additionally need to install the Selenium Server. 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.

Featured Articles

Start Selenium Testing with Python: Automated Testing of a User Signup Form

How to Switch Tabs in Selenium For Python

Setting up BrowserStack Automate

Automation Testing Selenium Selenium Webdriver

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, which we will shortly cover. 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.

In case, you want to try Local Testing using our BrowserStack Automate, check out this documentation.

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")
```

Featured Articles

Start Selenium Testing with Python: Automated Testing of a User Signup Form

How to Switch Tabs in Selenium For Python

Setting up BrowserStack Automate

Automation Testing Selenium Selenium Webdriver

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.

Copied

```
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.

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")
```

```
search_bar.send_keys(Keys.RETURN)
```

Also read: Want to understand other use cases of SendKeys in Selenium? Check it out.

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)
```

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 this example, we have looked at the steps involved in running our first test using Selenium and Python. Do note that we kept the window open during all stages of the test, to ensure you knew what went on in the background as you run each command.

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

To summarise the discussion, here is your **first Selenium test on Python**. You may save it in the file *selenium_test.py* and run python *selenium_test.py* to run the test.

```
from selenium import webdriver
```

```
search_bar.send_keys(Keys.RETURN)
print(driver.current_url)
driver.close()
```

Navigate through HTML DOM Elements

Now that you have successfully run your first test in Selenium with Python, let us look at various options to select DOM elements and interact with them. In the example, we selected the search bar and queried for a string. Let us explore the selection further. Here is the HTML of the search bar.


```
<input id="id-search-field" name="q" type="search" role="textbox" class="search-field" p
```

In the example, we used the *.find_element_by_name()* method, which searches for the attribute name within the input HTML tag. We can also search for this term using other methods.

- CSS ID: *.find_element_by_id("id-search-field")*
- DOM Path: *.find_element_by_xpath("//input[@id='id-search-field']")*
- CSS class: *.find_element_by_class_name("search-field")*

While the CSS ID is unique to every element by design, you may find multiple results when searching through the class name. Further, when you search through the DOM path of the element, you can be certain of what you are searching for.

Did you know: Difference between *findElement* and *findElements*? **Find out**.

Navigate through Windows and Frames

Your web application may require you to work with multiple windows and frames. Common use cases of working on new windows are social logins and file uploads. The *.switch_to_window()* method of the driver will help you to change the active window and work on different actions in a new window. The code that switches focus to a new window is:

```
print(driver.window_handles)
```

Similarly, you can switch focus to a frame within a window through the *.switch_to_frame()* method. To switch back to the primary window after completing relevant actions, run the following.

```
driver.switch_to_default_content()
```

Note that all actions within this section change the state of the current window and not anything. Therefore, we do not store the values in a variable and instead call the methods.

Work with Idle Time During a Test

While we have looked at various tests in static web applications, a single-page application may require you to wait for a specific time until you perform an action.

There are two types of waits in Selenium: *implicit* and *explicit* waits. An explicit wait makes your driver wait for a specific action to be completed (like content load using AJAX). An implicit wait makes the driver wait for a particular time.

For an explicit wait, you need to use a try-finally block because it can potentially make your test stuck in the worst-case scenario. Essentially, you instruct the driver to wait for a certain element for a specified time before letting go.

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

try:
element = WebDriverWait(driver, 5).until(
EC.presence_of_element_located((By.ID, "id-of-new-element"))
)
finally:
driver.quit()
```

First, use the *WebDriverWait()* function to tell the driver to wait for an amount of five seconds. You then

```
driver.implicitly_wait(5)
element = driver.find_element_by_id("id-of-new-element")
```

How to integrate Selenium with Python Unit Tests

Let us try to understand how to integrate Selenium tests into Python unit tests. For this purpose, we will use the unit test module in Python.

Featured Articles

Start Selenium Testing with Python: Automated Testing of a User Signup Form

How to Switch Tabs in Selenium For Python

Setting up BrowserStack Automate

Automation Testing Selenium Selenium Webdriver

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

class ChromeSearch(unittest.TestCase):

def setUp(self):
self.driver = webdriver.Chrome('./chromedriver')

def test_search_in_python_org(self):
driver = self.driver
driver.get("https://www.python.org")
self.assertIn("Python", driver.title)
elem = driver.find_element_by_name("q")
elem.send_keys("getting started with python")
elem.send_keys(Keys.RETURN)
assert "https://www.python.org/search/?q=getting+started+with+python&submit=" == driver.current

def tearDown(self):
self.driver.close()

if __name__ == "__main__":
unittest.main()
```

In this example, you need to set up the driver object when initializing the unit test class through the *.Chrome()* method. In the single test that we demonstrate, the same text is put on the search bar and the resultant change in URL is compared to the URL that was seen earlier. You may additionally write a different test for a different browser and reuse the same functionality.

To run Selenium on real devices through BrowserStack, you need to register on BrowserStack first. *You get 100 minutes of free testing under the free plan, after which you need to subscribe to a monthly plan.*

On logging in, select "BrowserStack Automate" and set the device-browser combination on which you would like to run a test. You are then shown the sample code to copy over and run from your terminal to run your test.

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

desired_cap = {
'browserName': 'android',
'device': 'Samsung Galaxy Note 9',
'realMobile': 'true',
'os_version': '8.1',
'name': 'Bstack-[Python] Sample Test'
}

driver = webdriver.Remote(
command_executor='https://<<user>>:<<user.keys>>@hub.browserstack.com:80/wd/hub',
desired_capabilities=desired_cap)

driver.get("https://www.google.com")
if not "Google" in driver.title:
raise Exception("Unable to load google page!")
elem = driver.find_element_by_name("q")
elem.send_keys("BrowserStack")
elem.submit()
print (driver.title)
driver.quit()
```


Let us take a moment to see the differences from what we have discussed so far in this article.

- First, you will notice that the test is conducted on a remote server and not on your local machine. Hence, you are using the *.Remote()* method to call a specific URL with the settings that you need. Selenium Server is installed on the BrowserStack cloud, which takes care of the initialization of the relevant browser and device for you to test on. Once the driver is initiated, you are familiar with the

The following is the output of the code, as expected. BrowserStack allows you to view a video of the test being performed on the device in real-time from your dashboard too.

```
BrowserStack - Google Search
```

Limitations with Selenium Python tests

While Selenium helps you in automating your tests and saving precious time, it has its limitations. Even with such a robust testing suite, you will often find yourself in an awkward position due to the ever-changing nature of front-end technologies.

Here are the top five challenges that one faces when automating the testing process with Selenium.

Final Thoughts

In this article, we covered various techniques of automating your cross browser testing process through Selenium using the Python programming language. While we have discussed the nuances of browser support, DOM navigation, waits, and unit tests. Finally, we covered how to perform remote testing on BrowserStack.

Even with all the knowledge of how the Selenium framework works, your testing framework is only as robust as the tests you design. Automating the testing process saves a lot of time during the test, so you should ensure that you spend significant time on designing the tests to capture all possible scenarios. It's always better to catch an error in the testing phase rather than leading to a customer complaint.

Featured Articles

Start Selenium Testing with Python: Automated Testing of a User Signup Form

How to Switch Tabs in Selenium For Python

Setting up BrowserStack Automate

Automation Testing Selenium Selenium Webdriver

Start Selenium Testing with Python: Automated Testing of a User Signup Form

A quick and easy example of how to use Selenium to automate signup process with Python. Learn to get...

Learn More

This guidepost demonstrates how to switch between multiple tabs in Selenium using Python.

Learn More

Setting up BrowserStack Automate

Any website, once developed, needs to be tested across multiple platforms and browsers. This way, yo...

Learn More