2. Getting Started

2.1. Simple Usage

If you have installed Selenium Python bindings, you can start using it from Python like this.

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

driver = webdriver.Firefox()
driver.get("http://www.python.org")
assert "Python" in driver.title
elem = driver.find_element_by_name("q")
elem.clear()
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
assert "No results found." not in driver.page_source
driver.close()
```

The above script can be saved into a file (eg:- *python_org_search.py*), then it can be run like this:

```
python python_org_search.py
```

The *python* which you are running should have the *selenium* module installed.

2.2. Example Explained

The *selenium.webdriver* module provides all the WebDriver implementations. Currently supported WebDriver implementations are Firefox, Chrome, IE and Remote. The *Keys* class provide keys in the keyboard like RETURN, F1, ALT etc.

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

Next, the instance of Firefox WebDriver is created.

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

The *driver.get* method will navigate to a page given by the URL. WebDriver will wait until the page has fully loaded (that is, the “onload” event has fired) before returning control to your test or script. *Be aware that if your page uses a lot of AJAX on load then WebDriver may not know when it has completely loaded*:

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

The next line is an assertion to confirm that title has “Python” word in it:

```
assert "Python" in driver.title
```

WebDriver offers a number of ways to find elements using one of the *find_element_by_** methods. For example, the input text element can be located by its *name* attribute using *find_element_by_name* method. A detailed explanation of finding elements is available in the Locating Elements chapter:

v: latest

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

Next, we are sending keys, this is similar to entering keys using your keyboard. Special keys can be sent using *Keys* class imported from *selenium.webdriver.common.keys*. To be safe, we'll first clear any pre-populated text in the input field (e.g. "Search") so it doesn't affect our search results:

```
elem.clear()
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
```

After submission of the page, you should get the result if there is any. To ensure that some results are found, make an assertion:

```
assert "No results found." not in driver.page_source
```

Finally, the browser window is closed. You can also call *quit* method instead of *close*. The *quit* will exit entire browser whereas *close* will close one tab, but if just one tab was open, by default most browser will exit entirely.:

```
driver.close()
```

2.3. 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.

In this chapter, we use *unittest* as the framework of choice. 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()
```

You can run the above test case from a shell like this:

v: latest

```
python test_python_org_search.py
.
----------------------------------------------------------------------
Ran 1 test in 15.566s

OK
```

The above result shows that the test has been successfully completed.

Note: To run the above test in IPython or Jupyter, you should pass a couple of arguments to the *main* function as shown below:

```
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

2.4. Walkthrough of the example

Initially, all the basic modules required are imported. The unittest module is a built-in Python based on Java's JUnit. This module provides the framework for organizing the test cases. The *selenium.webdriver* module provides all the WebDriver implementations. Currently supported WebDriver implementations are Firefox, Chrome, IE and Remote. The *Keys* class provides keys in the keyboard like RETURN, F1, ALT etc.

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

The test case class is inherited from *unittest.TestCase*. Inheriting from *TestCase* class is the way to tell *unittest* module that this is a test case:

```
class PythonOrgSearch(unittest.TestCase):
```

The *setUp* is part of initialization, this method will get called before every test function which you are going to write in this test case class. Here you are creating the instance of Firefox WebDriver.

```
def setUp(self):
    self.driver = webdriver.Firefox()
```

This is the test case method. The test case method should always start with characters *test*. The first line inside this method create a local reference to the driver object created in *setUp* method.

```
def test_search_in_python_org(self):
    driver = self.driver
```

The *driver.get* method will navigate to a page given by the URL. WebDriver will wait until the page has fully loaded (that is, the "onload" event has fired) before returning control to your test or script. *Be aware that if your page uses a lot of AJAX on load then WebDriver may not know when it has completely loaded*:

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

The next line is an assertion to confirm that title has "Python" word in it:

```
self.assertIn("Python", driver.title)
```

v: latest

WebDriver offers a number of ways to find elements using one of the *find_element_by_** methods. For example, the input text element can be located by its *name* attribute using *find_element_by_name* method. Detailed explanation of finding elements is available in the Locating Elements chapter:

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

Next, we are sending keys, this is similar to entering keys using your keyboard. Special keys can be send using *Keys* class imported from *selenium.webdriver.common.keys*:

```
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
```

After submission of the page, you should get the result as per search if there is any. To ensure that some results are found, make an assertion:

```
assert "No results found." not in driver.page_source
```

The *tearDown* method will get called after every test method. This is a place to do all cleanup actions. In the current method, the browser window is closed. You can also call *quit* method instead of *close*. The *quit* will exit the entire browser, whereas *close* will close a tab, but if it is the only tab opened, by default most browser will exit entirely.:

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

Final lines are some boiler plate code to run the test suite:

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

2.5. Using Selenium with remote WebDriver

To use the remote WebDriver, you should have Selenium server running. To run the server, use this command:

```
java -jar selenium-server-standalone-2.x.x.jar
```

While running the Selenium server, you could see a message looking like this:

```
15:43:07.541 INFO - RemoteWebDriver instances should connect to: http://127.0.0.1:4444/wd/hu
```

The above line says that you can use this URL for connecting to remote WebDriver. Here are some examples:

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

driver = webdriver.Remote(
   command_executor='http://127.0.0.1:4444/wd/hub',
   desired_capabilities=DesiredCapabilities.CHROME)

driver = webdriver.Remote(
   command_executor='http://127.0.0.1:4444/wd/hub',
   desired_capabilities=DesiredCapabilities.OPERA)
```

v: latest

```
driver = webdriver.Remote(
   command_executor='http://127.0.0.1:4444/wd/hub',
   desired_capabilities=DesiredCapabilities.HTMLUNITWITHJS)
```

The desired capabilities is a dictionary, so instead of using the default dictionaries, you can specify the values explicitly:

```
driver = webdriver.Remote(
   command_executor='http://127.0.0.1:4444/wd/hub',
   desired_capabilities={'browserName': 'htmlunit',
                         'version': '2',
                        'javascriptEnabled': True})
```

v: latest