**Getting started with Selenium with Python**

There are a few automation testing tools for web applications out there. [Selenium](https://selenium.dev/) is one of them and is probably most widely-used. It is the most famous and the oldest, but still relevant, automation testing tool. It is open-source and supports many different major programming languages, such as Java, C#, PHP, Python, Perl and Ruby.

For automation tests, I have been using [Cypress](https://www.cypress.io/), which uses TypeScript. Cypress is great because it is easy to learn (writing tests feel like writing regular JavaScript unit tests apart from needing to learn Cypress APIs). It is easy to debug as the browser stays open after the test runs and we can rewind to any previous steps.

One of the drawback for using Cypress is that it uses the older version of chrome and no other browser support is available at the moment.

Anyway, I decided to get out of my comfort zone and give it a go at Selenium. A few people suggested me that it was a pretty cool tool for web application automation tests. Data people have been using it to scrape web or automate file download actions to get data and they really love the tool. Because we can write those scripts in Python, the entry barrier for data people is low.

Learning Selenium seems to open more possibilities than just writing web automation tests. The best thing is that we can do them all in Python!

This post focuses on setting up Selenium and run a simple test for a fictitious user sign in page. Let’s get started.

**Setting up**

Assuming you have Python (both 2 and 3 work) installed, you need to install selenium, web driver, pytest and pytest-html. You can check out the [selenium-python documentation](https://selenium-python.readthedocs.io/installation.html#introduction) for further information. This documentation’s examples use [unittest](https://docs.python.org/3/library/unittest.html). I prefer using [pytest](https://docs.pytest.org/en/latest/). So, let’s install pytest for a testing framework and also install pytest-html to output the test results in a html format.

1. **Install Selenium**

<pre>

pip install selenium

</pre>

1. **Install driver**

By downloading different drivers, you can run tests in different browsers. Drivers are supplied by the browser vendors. You can get the links to download drivers from the 1.3 Driver section [here](https://selenium-python.readthedocs.io/installation.html#introduction). In this example, let’s get the [driver for chrome](https://sites.google.com/a/chromium.org/chromedriver/downloads). You need to make sure the version of the driver matches the version of the Chrome installed on your computer.

The driver is an executable file. You can put it in any folder and the path to that driver can be referred in Python code like this: driver = webdriver.Chrome(‘./drivers/chromedriver.exe’).

1. **Install pytest**

[pytest](https://docs.pytest.org/en/latest/) is the best testing framework for Python. It is used widely for large scale Python projects. It has pretty much everything you ask for a testing framework.

<pre>

pip install pytest

</pre>

1. **Install pytest-html**

This is for outputting the test results in html format. By default, it creates report.html in the same folder as the test script is located.

<pre>

pip install pytest-html

</pre>

**Running test**

The test is executed by pytest. To run a test, we can use this command, assuming the test script is called exampleTest.py and you are in the same folder where the script is.

<pre>

py.test --html=report.html exampleTest.py

</pre>

For the first time, I recommend to run [this example](https://blog.testproject.io/2019/07/16/web-test-using-selenium-webdriver-python-chrome/). The test is test\_basic\_duckduckgo\_search. You can copy and paste the entire example and execute it with py.test –html=report.html firstTest.py.

**Test Example**

Now, here is the actual test example. I have a website with a registration page. Each user has to have a unique email. Therefore, I have utility function to create unique user each time the test runs.

The selecting element is fairly simple. As with any automation test, we need to wait till the website loads. You can use the WebDriverWait function for that.

For some reason, element\_to\_be\_clickable didn’t work for me. The work around is to use execute\_script function to call JavaScript function. It works fine.

We need to initialise the driver in the fixture. The rest is pretty straight forward.

<script src="https://gist.github.com/mydatahack/ec91e9d43008d1a12afbbbcc44ef8a76.js"></script>

After having written tests with JavaScript for a long time, it is fun and liberating to use Python. You should give it a go, too!