Python/Webdriver Automation Associate

Author: Matt Chiang
Publish February 28, 2019
Rev. 2
Updated May 16, 2019

1. Intro - Automation using Python, Webdriver, Unittest.
    a. Have a full step by step tutorial to get started and setup development environment and dependent libraries.

2. Prerequisites
    a. Know page objects automation development pattern.
        i. https://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html
        ii. http://toolsqa.com/selenium-webdriver/page-object-model/
        iii. https://www.swtestacademy.com/page-object-model-java/
        iv. https://medium.com/tech-tajawal/page-object-model-pom-design-pattern-f9588630800b

3. Main sections
    a. Provide links to documentation or blogs that cover each programming concepts. Class, Method, Loops, If/Else, Exception Handling, Collections, Build tools, data provider/driven, action builders, reporting, queues/lists, test annotations, testing framework, assertions/verifies.
    b. Each topic will have a focus on writing a short solution that covers each topic.
    c. Provide a code challenge to demonstrate knowledge of the section.

4. Resources
    a. Links to free resources to help learn the topics.
        i. https://www.w3schools.com/python/default.asp
        ii. https://selenium-python.readthedocs.io/

5. Quick start guide to checking your code into github.
    a. https://docs.google.com/document/d/1DTYoPb4LnJpz3LOGrNBMbhpK70dI9AG-GIwl5orOkPs/edit?usp=sharing

Challenge 1 - Quick Start - Python/Webdriver
Let's begin:
Step 1:
go to https://www.python.org/downloads/, find your install and download the latest 3.x version and install.

For those of you not on windows:
**install of python on Ubuntu.
https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-local-programming-environment-on-ubuntu-16-04

***install of Python on Mac.
http://www.pyladies.com/blog/Get-Your-Mac-Ready-for-Python-Programming/

Step 2:
After you install python, let's open a command prompt or terminal and test it by running "python -V". Do you see a version number? Python is installed if you do. If you see a bunch of other text, use the “ctrl Z” and “Enter” to exit the python. Then try again and make sure you use a upper case “V”.

Step 3:
Download the free PyCharm Community Edition (https://www.jetbrains.com/pycharm/) or your favorite IDE that is capable of building/interpreting python.

Step 4:
Start your PyCharm or IDE.

Step 5:
Create a New Project and name it challenges.

Step 6:
Now open the “Terminal” in PyCharm. This is at the bottom of window. You should now be in the project folder you just created. …\PycharmProject\challenges>

Step 7:
Let's install selenium-webdriver by running "pip install selenium". If you are using python 3 on a Mac, you will enter “pip3 install selenium”

Pip is the python manager to get and install various libs that you will need. At this point, if you view open the location where the \PycharmProject\challenges folder, you'll see a venv folder. Inside this folder, you’ll see a \Lib\site-packages folder. Inside this folder, you’ll see the selenium package you just installed.

to install chromedriver by running "pip install chromedriver" or “pip3 install chromedriver”

Step 8:
By default, python installs unittest framework. We’ll be using this for our testing setup/teardown and assertions.

Step 9:
Now, before you can run your test, you’ll need to download the chromedriver and/or the geckodriver if you want to use Firefox.
You can download these here:
https://sites.google.com/a/chromium.org/chromedriver/downloads
https://github.com/mozilla/geckodriver/releases

click the version/Releases that matches your chrome or firefox or the latest version available. Then download the chromedriver/geckodriver zip file for the OS you have.

Once downloaded, extract the chromedriver.exe/geckodriver.exe to the root of where your project folder is. On windows, it’s typically in the:
c:\users\yourusername\PycharmProjects\challenges

One you have these files extracted, you will see them in Pycharm in the project explorer.

** installing chromedriver on a Mac.
http://jonathansoma.com/lede/foundations-2017/classes/more-scraping/selenium/

Step 10:
Now right click on the Challenges folder in the project explorer and select “New” then “Python Package”. This will create a new folder of challenge1 and a __init__.py file in this folder.

Step 11:
create a .py file for your automation code in the challenge1 folder by right clicking on the challenge1 folder and selecting “python file” in PyCharm. Give the file a name. ie: challenge1.py

Step 12:
Let's create our function and basic setup and tear down structure in the .py file.

```
class Challenge1(unittest.TestCase):

    def setUp(self):
        #code to startup webdriver

    def tearDown(self):
        #code to close webdriver
```

```
    def test_challenge1(self):
        #code for our test steps
```

Step 13:
Now let's import webDriver and declare a new webdriver variable that will be used to perform the actions in Chrome/Firefox. We'll also need to import unittest library.

```
import unittest
from selenium import webdriver
```

Step 14:
It's also a good practice to add a unittest.main() at the bottom of our code so that it can be run. This provides a command-line interface to the test script.

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

Step 15:
Let's build our before method by initializing our driver. You'll need the relative path to where the chrome driver/geckodriver is. Since files are one folder up from where our challenge1.py file is, it'll be:

```
self.driver = webdriver.Chrome("../chromedriver.exe")
```

**** if you are using a Mac/Linux, you won't need the .exe of the file extension.**

Step 16:
Let's add some code to close web browser after we finish so we don't have manually close the browser in the teardown.

```
self.driver.close()
```

Step 17:
Let's write a test function that opens google's page and checks to make sure the page title is what we expected by using the assertIn method.

```
self.driver.get("https://www.google.com")
self.assertIn("Google", self.driver.title)
```

Here's your finished code:

```
import unittest
from selenium import webdriver

class Challenge1(unittest.TestCase):

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

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

    def test_challenge1(self):
        self.driver.get("https://www.google.com")
        self.assertIn("Google", self.driver.title)


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

Step 18:
To launch your automation test, save your test file, then run “python challenge1.py” in your terminal or command prompt.

You should now see something like this:

DevTools listening on
ws://127.0.0.1:61429/devtools/browser/b0e49739-11c1-4790-81f9-67fd4a43e3b4
.

Ran 1 test in 2.896s

OK

That's it! You are now all setup to write more automation using Python/Webdriver.

Now for the fun part of this certification. Below are the concepts that applies to every web app that we deal with. Once you have mastered these concepts, building an automation framework for another client should be a breeze. As you finish each challenge, the code needs to be checked into a public github repo. This will allow our clients to see the knowledge you have gained and the coding patterns used. They can see your mastery of the concepts and how it would apply to their projects. Once the code is checked into github, add your repo to your CDP(Career Development Plan) and notify the Practice Manager (Matt Chiang – matt.chiang@stgconsulting.com). We will checkout your code and run each challenge. Remember to make sure you have all your dependencies in your project folder when you check

it in. It'll be easier to pull your code and run. We will announce your completion and make you famous to all our Account Managers, and potential clients. We will have certification badges that you can also add to your email signatures to show off your accomplishment.

Challenge 2 - Asserts:
Many times, the general thought for an automation script is to go from point A to point B. However, this doesn't measure if the test actually passed or not. There are times when a button click doesn't work or the action handler going to a wrong page like a 404 page. Just because the script worked and you are on a new page does not mean the test passed. That is where asserts come in. In terminology, you can so soft or hard asserts. Soft asserts checks the data but does not stop the script from running. Hard asserts will stop the script when it fails. Soft asserts allows you to build long running automation scripts like one what might span across 5 different pages. When you write a script for shopping for a product, you would typically to go the login page, product page, cart page, checkout summary page, checkout verification page. Rather than writing one script for each of these pages, you can build one test and add in soft asserts and trigger warnings or print out to log files when there is failed scenario in the soft assertion. Then once you get to the final page, or final step, you can do a hard assertion to check for something like an order number.

For this challenge, look through the different ways to do assertions. Then write a script that will go to copart.com, search for exotics and verify porsche is in the list of cars. Use the hard assertion for this challenge.

https://pythontips.com/2013/08/07/the-self-variable-in-python-explained/
https://micropyramid.com/blog/understand-self-and-__init__-method-in-python-class/
https://selenium-python.readthedocs.io/getting-started.html
https://www.tutorialspoint.com/python/assertions_in_python.htm
http://www.mahsumakbas.net/selenium-assertion-with-python-unittest/

Another tutorial on how to use webdriver's getAttributes. Example is in java but works the same in python.
https://www.seleniumeasy.com/selenium-tutorials/how-to-get-attribute-values-using-webdriver

This example is an older video that explains verify vs assert. The exact code is not relevant to our training but it's a good concept video to watch.

https://www.youtube.com/watch?v=iw_NDJsLYt8

Challenge 3 - Loops:
Loops can be used to write your own wait statements. They can also be used to iteration through a list of items.

For this challenge, go to copart and print a list of all the “Popular Items” of vehicle Make/Models on the home page and the URL/href for each type. This list can dynamically change depending on what is authored by the content creator but using a loop will make sure that everything will be displayed regardless of the list size.

https://www.pythonforbeginners.com/loops/for-while-and-nested-loops-in-python
https://www.learnpython.org/en/Loops
https://www.w3schools.com/python/python_for_loops.asp
https://devhints.io/xpath
https://www.w3schools.com/xml/xpath_intro.asp
https://www.red-gate.com/simple-talk/dotnet/.net-framework/xpath,-css,-dom-and-selenium-the-rosetta-stone/

Your output in the console would look like:
IMPREZA - https://www.copart.com/popular/model/impreza
CAMRY - https://www.copart.com/popular/model/camry
ELANTRA - https://www.copart.com/popular/model/elantra
SONATA - https://www.copart.com/popular/model/sonata
COROLLA - https://www.copart.com/popular/model/corolla
ALTIMA - https://www.copart.com/popular/model/altima
FORESTER - https://www.copart.com/popular/model/forester
OPTIMA - https://www.copart.com/popular/model/optima
IMPALA - https://www.copart.com/popular/model/impala
PRIUS - https://www.copart.com/popular/model/prius
FORD - https://www.copart.com/popular/make/ford
TOYOTA - https://www.copart.com/popular/make/toyota
CHEVROLET - https://www.copart.com/popular/make/chevrolet
DODGE - https://www.copart.com/popular/make/dodge
HONDA - https://www.copart.com/popular/make/honda
NISSAN - https://www.copart.com/popular/make/nissan
SUBARU - https://www.copart.com/popular/make/subaru

Challenge 4 - Operators and Functions:
For this challenge, we will be using operators and functions to tinker around w/ string concatenation and tokenization. String concatenation is the process of putting together a message using multiple variables. The application of this can be for logging or capturing variables or using a dynamic variable that is found on a page before it's used to trigger an action. Tokenization is needed for breaking up a sentence or to break up a long string into arrays so that you can example each word. It can also be used to pull variables out of the URLs. There are endless uses.

For this challenge, we are going to write a function that display the fibonacci sequence up to a certain number. If I want the fibonacci for the 9 order of the sequence, I should see 21. Keep your function to calculate the fibonacci sequence separate from the file that has the unittest.main().

However, to add additional challenge to this challenge, instead of displaying the number 21, I want the string representation of twenty one. This will require you to use string concatenation to print out the string.

https://technobeans.com/2012/04/16/5-ways-of-fibonacci-in-python/
https://www.programiz.com/python-programming/examples/fibonacci-recursion
https://www.youtube.com/watch?v=POQIIKb1BZA

Your console output will look something like this.

13 - thirteen
144 - one hundred forty four
7408 - seven thousand four hundred eight

Now that you know how to write a function, copy your previous challenge folder and create a new folder. Now let's refactor your "before" method and make it a one line call out to another function. The reason why we do this is so that we only have one function that does our initialization of the driver. Otherwise, the poor implementation would be to copy multiple lines from script to script. When it comes time to change the "before" method, you will have to change it in multiple places versus one.

**** Do not use someone else's library. You should write your own logic. Keep non related classes separate. ie. Fibnacci class has no relations to convertNumbertoString class ****

Challenge 5 - If/Else/Switch

Decision making is what will make your automation run w/ some AI helping it. Rather than building a script w/ hard coded variables, why not use if/else/switch to do some basic decision making. Let's say, for a shopping scenario, I want to use Visa payment. Then if I want to test MasterCard, should I write two sets of scripts? Or should I write one script that depending on the data/card I want to use, the script will use if/else/switch to select the appropriate payment method for me. This way, I will only have one script to maintain but I can write as many tests as I want that passes in a different set of data/variables that can be tested. If someone changes the payment form, I would only have to update one set of code instead of multiple sets of code. This will help you make your automation more maintainable. Another use in the decision making is to count bits of data.

For this challenge, go to https://www.copart.com and do a search for "porsche" and change the drop down for "Show Entries" to 100 from 20. Count how many different models of porsche is in the results on the first page and return in the terminal how many of each type exists.

Possible values can be "CAYENNE S", "BOXSTER S", etc.

For the 2nd part of this challenge, create a switch statement to count the types of damages. Here's the types:
REAR END
FRONT END
MINOR DENT/SCRATCHES
UNDERCARRIAGE
And any other types can be grouped into one of MISC.

https://www.w3schools.com/python/python_conditions.asp
https://www.techbeamers.com/python-switch-case-statement/
https://softwareengineering.stackexchange.com/questions/206816/clarification-of-avoid-if-else-advice
https://en.wikipedia.org/wiki/Cyclomatic_complexity
https://dzone.com/articles/code-smells-if-statements
https://stackoverflow.com/questions/14555992/how-to-get-rid-of-if-else-statement-clutter

*** make sure you make your code reusable. use class and functions.

Challenge 6 - Error Handling
Using a try/catch block can help you catch certain errors that might exist that you weren't anticipating. It can also give you some type of behavior you want to take when something happens.

Let's say you are running a test script and the 4th step fails. At this point, you can decide what you want it to do. Do you want to try something else, or take a screenshot, or reset your browser of where you are at for the next step in the script?

Taking a screenshot is a common way to use the try catch block. For this challenge, go to copart site, search for nissan, and then for the model, search for "skyline". This is a rare car that might or might not be in the list for models. When the link does not exist to click on, your script will throw an exception. Catch the exception and take a screenshot of the page of what it looks like.

https://selenium-python.readthedocs.io/api.html#module-selenium.common.exceptions
https://www.pythonforbeginners.com/error-handling/python-try-and-except
https://www.w3schools.com/python/python_try_except.asp
https://pythonspot.com/selenium-take-screenshot/
https://www.pythonforbeginners.com/gui/how-to-use-pillow
http://allselenium.info/capture-screenshot-element-using-python-selenium-webdriver/

*** make sure you make your code reusable. use class and functions.

Challenge 7 - Array/Dictionary.

One mistake that many people make is for writing automation is making their script/objects too static. This causes a maintenance nightmare when you have 1000 tests that needs to be updated when a value changes or locators change. This also posed a challenge when there are multiple languages involved but the page layout stays the same or if the list of elements grows or shrinks. Arrays are great to use to create a virtual layout/map on navigation objects like a top menu or multiple links in the footer. Rather than locating only one link, why not build a map of all the links on a certain section. For this challenge, take a look at https://www.copart.com main page. Go to the Makes/Models section of the page. Create a 2 dimensional array that stores all the values displayed on the page along w/ the URL for that link. Once you have this array, you can verify all the elements in the array navigates to the correct page. To get started, inspect the code and notice the section of the page is built using angular. There is no static id or element class that identifies each element in this section. Everything is generic. The only way to build a function/object for this section is to loop through each element. Hint: xpath is easiest.

https://www.w3schools.com/python/python_dictionaries.asp
https://www.w3schools.com/python/python_arrays.asp

*** make sure you make your code reusable. use class and functions.

Node/Webdriver Automation Solution Engineer
Advanced Level 2

Challenge 8 - REST web service.
There are times when a GUI is not available or it's broken. This is typically if you are testing APIs. This is a typical scenario for testing search. The GUI is only built to capture some string or parameter. Then when you click the search button, it passes the string into a web service that returns some type of data.

For this challenge, use the copart web service to search for Toyota Camry and grab the data and output it to a log file.

Here's the search end point.
https://www.copart.com/public/lots/search

The request method would be a POST.

For the form data, you will be passing the "query: toyota camry"

The results returned will look like this:
{"returnCode":1,"returnCodeDesc":"Success","data":{"query":{"query":["toyota camry"],"page":0,"size":20,"start":0,"watchListOnly":false,"freeFormSearch":true,"hideImages":false,"defaultSort":false,"specificRowProvided":false},"results":{"totalElements":5090,"content":[{"lotNumberStr":"31429677","ln":31429677,"mkn":"TOYOTA","lm":"CAM……

Notice one of the items in the data is totalElements. This tells you how many items are in the search results.

Grab that data and output it to a log file.

Then run another query for "nissan skyline". The results might or might not return anything. Output how many in the results are found. Do this w/ 10 different search parameters of your favorite cars.

https://www.udemy.com/api-testing-python/
https://www.grossum.com/blog/beginner-s-guide-to-automating-api-tests-using-python
https://github.com/svanoort/pyresttest#running-a-simple-test
https://toolbelt.readthedocs.io/en/latest/user.html

Advanced Level 2.
Challenge 9 - Objects, JSON, and validating values in a JSON.

https://docs.python.org/3/library/json.html
https://www.w3schools.com/python/python_json.asp
https://www.programiz.com/python-programming/json
https://aboutsimon.com/blog/2018/04/04/Python3-Type-Checking-And-Data-Validation-With-Type-Hints.html
https://pynative.com/python-check-user-input-is-number-or-string/
https://codeyarns.com/2010/01/28/python-checking-type-of-variable/
https://pydantic-docs.helpmanual.io/
https://docs.python.org/3/library/typing.html
https://towardsdatascience.com/flattening-json-objects-in-python-f5343c794b10?gi=b80a822dad9e
https://pandas.pydata.org/
https://docs.python.org/3/library/json.html

For this challenge, it'll be an extension of the previous challenge. What good is a request to an API if all we are testing is a 200 response. There is so much more to testing REST APIs.

Extend the code now and validate every piece of data. Take the data returned and validate a random object in the json.

Here's a sample of one of the objects w/in the return JSON:
{"lotNumberStr":"31095358","ln":31095358,"mkn":"TOYOTA","lm":"CAMRY","lcy":2017,"fv":"4T1BF1FK1HU655063","la":19893.0,"rc":12578.0,"obc":"N","orr":0.0,"ord":"NOT ACTUAL","egn":"2.5L 4","cy":"4","ld":"2017 TOYOTA CAMRY LE","yn":"CO - DENVER CENTRAL","cuc":"USD","tz":"MST","ad":1548356400000,"at":"12:00:00","aan":0,"hb":0.0,"ss":5,"bndc":"","bnp":0.0,"sbf":false,"ts":"CO","stt":"ST","td":"SALVAGE TITLE","tgc":"TITLEGROUP_S","dd":"FRONT END","tims":"https://cs.copart.com/v1/AUTH_svc.pdoc00001/PIX133/3ab42ef9-7518-4f52-bf23-a95020b63968.JPG","lic":["IV","CERT-E"],"gr":"B010","dtc":"FR","adt":"E","ynumb":120,"phynumb":120,"bf":false,"ymin":175,"offFlg":false,"htsmn":"Y","tmtp":"AUTOMATIC","myb":0.0,"lmc":"TOYT","lcc":"CERT-E","sdd":"SIDE","lcd":"ENHANCED VEHICLES","ft":"GAS","hk":"NO","drv":"Front-wheel Drive","ess":"Pure Sale","showSeller":false,"sstpflg":false,"syn":"CO - DENVER CENTRAL","ifs":true,"pbf":false,"crg":0.0,"brand":"COPART"}

Validate lotNumberStr is a string, ln is an integer/float, mkn is a string, etc and return a list of all the data types for each variable.

Advance Level 2:
Challenge 10 - Generate webservice tests using a data provider.
For this challenge, you will read the data from an excel spreadsheet or CSV file. Depending on the rows of data and the columns of data, you will generate tests from this. This is an extension of challenge 8.

If the spreadsheet has the following values.

make	model	year	Vehicle type
toyota			
	camry		
		1997	
			Automobiles
toyota	avalon	2018	Automobiles

You would run 5 tests. Each test would use the web service test you wrote in challenge 8, pass in the value for the query, run the search and validate the data you searched for is in the list of results. If there are 30 lines of data, you would run 30 different tests. However, keep in mind that you will only write one test and loop through the tests with the data.

Advanced Level 2.
Challenge 11 - Building a webcrawler.

Who wants to ever test a website w/ hundreds or thousands of pages. Or who has every wanted to get every piece of data from a website and see if there are misspelled words.

For this challenge, let's build a webcrawler that will navigate to every page on https://www.copart.com. Don't worry about crawling all the data for now as we never want to seem like we are doing a DOS attack.

https://medium.com/python-pandemonium/develop-your-first-web-crawler-in-python-scrapy-6b2ee4baf954

https://www.youtube.com/watch?v=XjNm9bazxn8
https://www.youtube.com/watch?v=sVNJOiTBi_8
https://www.youtube.com/watch?v=pLHejmLB16o

https://www.youtube.com/watch?v=nRW90GASSXE
https://www.youtube.com/watch?v=z_vIWoTZm2E
https://www.youtube.com/watch?v=pjkZCQTfneQ
https://www.youtube.com/watch?v=jCBbxL4BGfU
https://www.youtube.com/watch?v=F2lbS-F0eTQ
https://www.youtube.com/watch?v=udBt0K7gwLc
https://www.youtube.com/watch?v=Eis9vu4XiNI
https://www.youtube.com/watch?v=MpazNSqP4uo
https://www.youtube.com/watch?v=QHWy0CXDBI4
https://www.youtube.com/watch?v=uTdweD5SCag
https://www.youtube.com/watch?v=luYg1qMVSfY
https://www.youtube.com/watch?v=XHIlke_0WnM
https://www.youtube.com/watch?v=rxGUiLcW0cI
https://www.youtube.com/watch?v=PPonGS2RZNc
https://www.youtube.com/watch?v=vKFc3-5Y17U

Advanced Level 2.
Challenge 12 - Final
This will be the final topic in our Python webdriver certification. It's to put all the stuff together as a solution that can be launched using a trigger. This topic will be more like a master thesis where you get to do your own research and build your own solution. Why is this needed? Simple 4 letter acronym. CI/CD (Continuous Integration, Continuous Deployment) pipeline. As part of a CI/CD pipeline, you want all your validation tests to be integrated so that when a new piece of code is checked in by a Feature Developer, and the code is deployed, the automation tests needs to auto trigger, run, and report pass/fail so that the build can be stopped or promoted to the next environment. You can use jenkins, teamcity, or bamboo or whatever other CI/CD tools to accomplish this challenge.

Here's a primer:
https://www.infoworld.com/article/3271126/ci-cd/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

https://stackify.com/jenkins-teamcity-bamboo/

For the reporting portion of this challenge, you can use any reporting framework. Easy ones can be integrated into jenkins. However, in practice, this makes it difficult to see the results if there are hundreds or thousands of tests executed per day or if the same suite of tests is executed multiple times a day. You can use a spreadsheet to document or store your tests. You can email your test results out. Come up w/ your own solution that will make this testing framework completely self executing and self reporting.