Test Automation using:

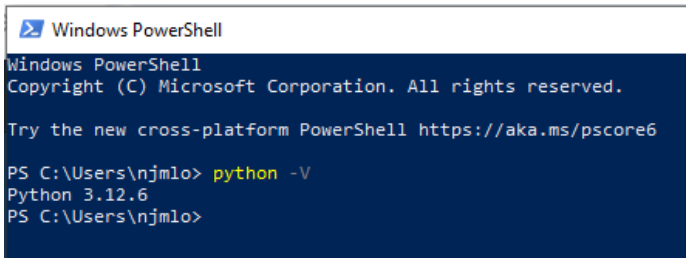
- Python
- Selenium IDE and Webdriver
- PyCharm
- PyTest
- Libraries (ie., time)

Document highlight:

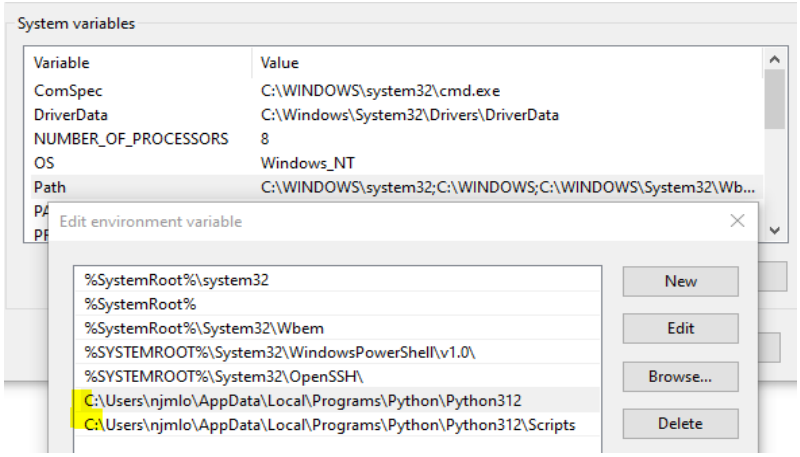
- Tools installation and setup
- Run a post installation test
- Interacting with elements in a web page
- Record, Playback, and Export (Selenium IDE)
- Pytest with Selenium and Python (generate html reports)
- More Selenium, Python, Pytest sampling codes and test run
    - Pytest fixtures, set up, and tear down
    - Parameterization
    - Run a test case using the command line/terminal
- Page Object Model (POM) sampling codes and test run

1. Tools installation and setup

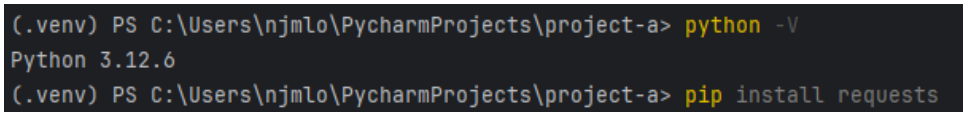
Python (version at the time of documentation):



Add Python environment variable under System Variable Path:



Determine Python version, install “requests” library:



Pip version (version at the time of documentation):

```
C:\Windows\System32>python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\njmlo\appdata\local\programs\python\python312\lib\site-packages (24.2)
Collecting pip
  Using cached pip-24.3.1-py3-none-any.whl.metadata (3.7 kB)
Using cached pip-24.3.1-py3-none-any.whl (1.8 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.2
    Uninstalling pip-24.2:
      Successfully uninstalled pip-24.2
Successfully installed pip-24.3.1
```

```
C:\Windows\System32>pip --version
pip 24.3.1 from C:\Users\njmlo\AppData\Local\Programs\Python\Python312\Lib\site-packages\pip (python 3.12)
```

PyCharm Community Edition (IDE version at the time of documentation):

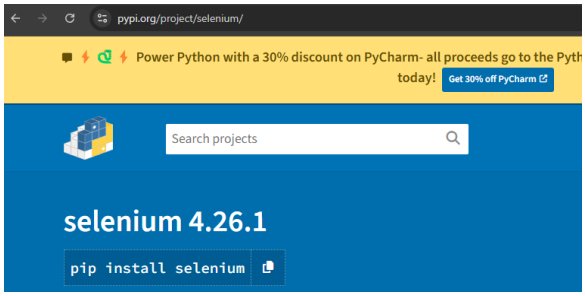
Download:

https://www.jetbrains.com/pycharm/download

Selenium install (version at the time of documentation):

```
C:\Windows\System32>pip install -U selenium
Collecting selenium
  Downloading selenium-4.26.1-py3-none-any.whl.metadata (7.1 kB)
```

```
Installing collected packages: sortedcontainers, websocket-client, typing_extensions, sniffio, pysocks, pycparser, h11, attrs, wsproto, outcome, cffi, trio, trio-websocket, selenium
Successfully installed attrs-24.2.0 cffi-1.17.1 h11-0.14.0 outcome-1.3.0.post0 pycparser-2.22 pysocks-1.7.1 selenium-4.26.1 sniffio-1.3.1 sortedcontainers-2.4.0 trio-0.27.0 trio-websocket-0.11.1 typing_extensions-4.12.2 websocket-client-1.8.0 wsproto-1.2.0

C:\Windows\System32>
```

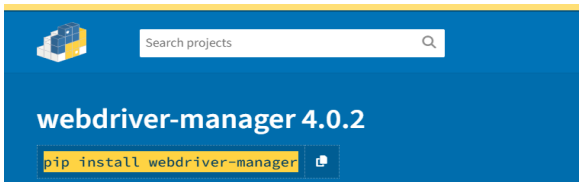


```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>pip check selenium
No broken requirements found.

C:\Windows\System32>pip show selenium
Name: selenium
Version: 4.26.1
Summary: Official Python bindings for Selenium WebDriver
Home-page: https://www.selenium.dev
Author:
Author-email:
License: Apache 2.0
Location: C:\Users\njmlo\AppData\Local\Programs\Python\Python312\Lib\site-packages
Requires: certifi, trio, trio-websocket, typing_extensions, urllib3, websocket-client
Required-by:

C:\Windows\System32>
```

Webdriver install (version at the time of documentation):

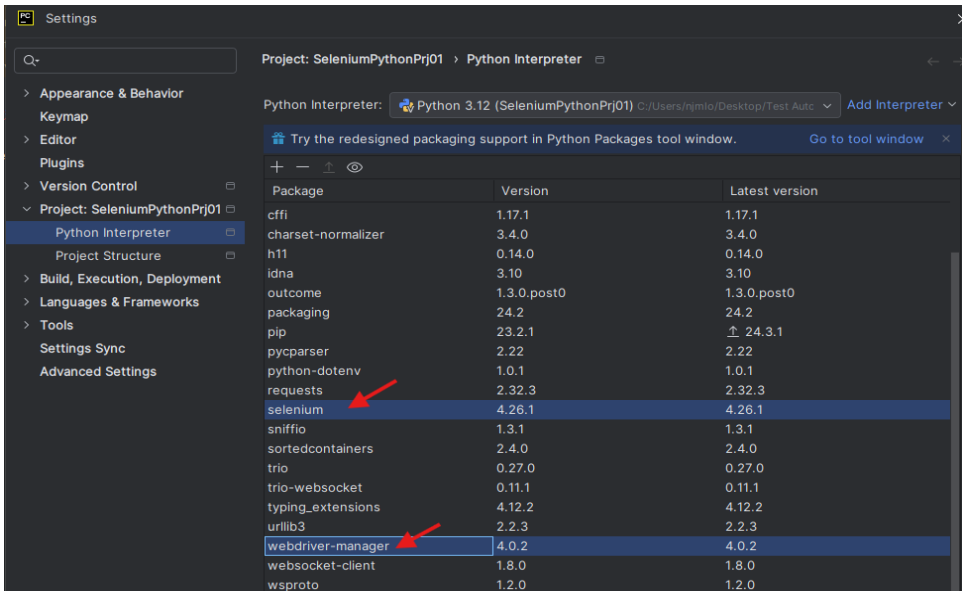
```
C:\Windows\System32>pip install webdriver-manager
Collecting webdriver-manager
  Downloading webdriver_manager-4.0.2-py2.py3-none-any.whl.metadata (12 kB)
```

```
Installing collected packages: python-dotenv, webdriver-manager
Successfully installed python-dotenv-1.0.1 webdriver-manager-4.0.2

C:\Windows\System32>
```

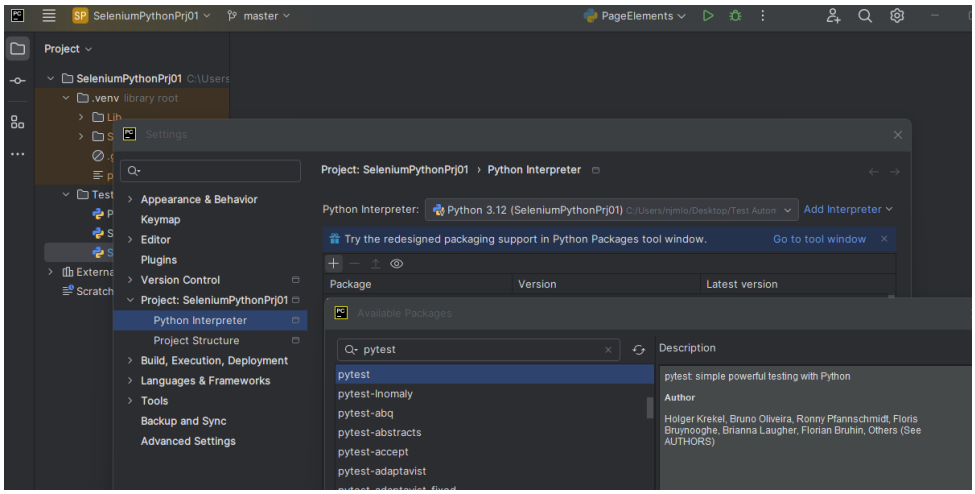


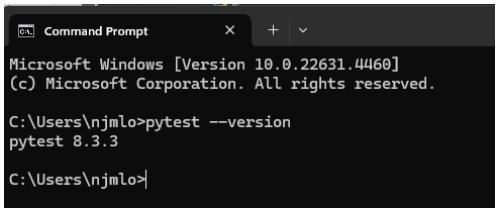

```
C:\Windows\System32>pip check webdriver-manager
No broken requirements found.

C:\Windows\System32>pip show webdriver-manager
Name: webdriver-manager
Version: 4.0.2
Summary: Library provides the way to automatically manage drivers for different browsers
Home-page: https://github.com/SergeyPirogov/webdriver_manager
Author: Sergey Pirogov
Author-email: automationremarks@gmail.com
License:
Location: C:\Users\njmlo\AppData\Local\Programs\Python\Python312\Lib\site-packages
Requires: packaging, python-dotenv, requests
Required-by:

C:\Windows\System32>
```

In Pycharm, validate Python interpreter:

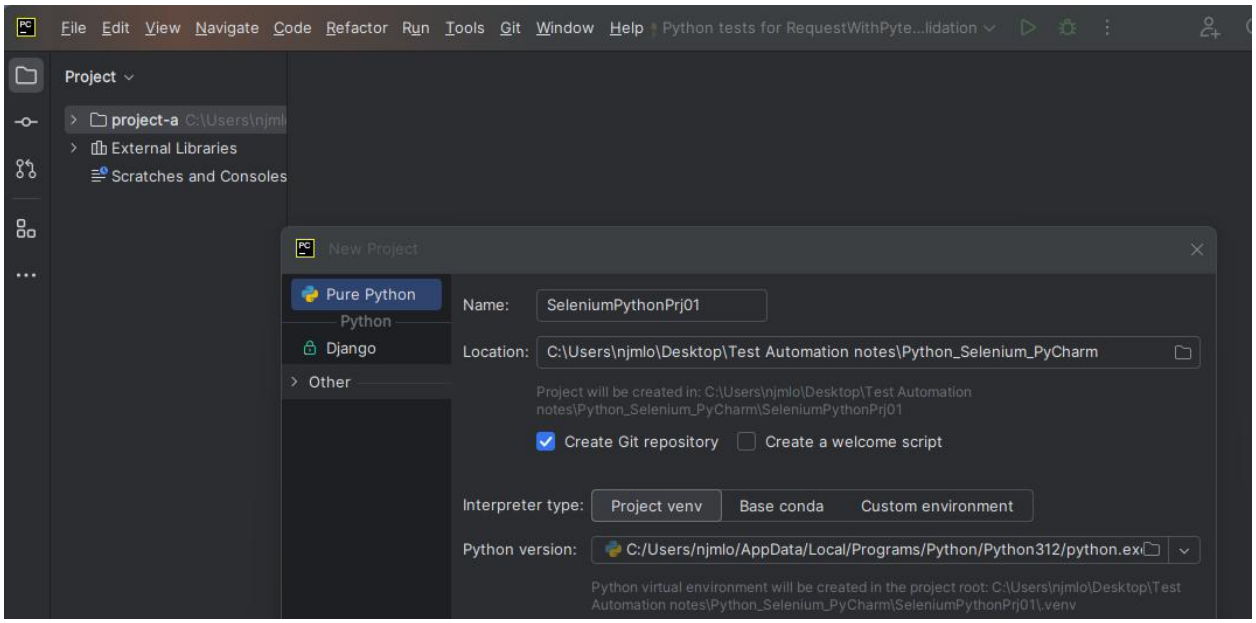


Add Pytest in the package:

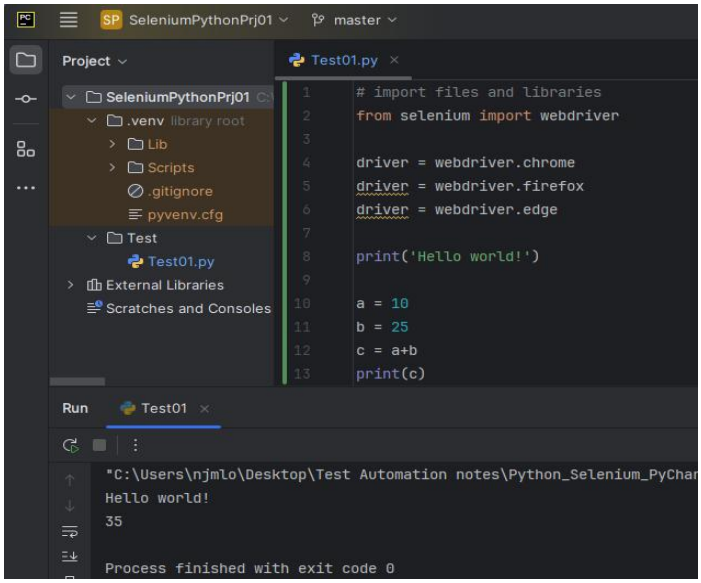




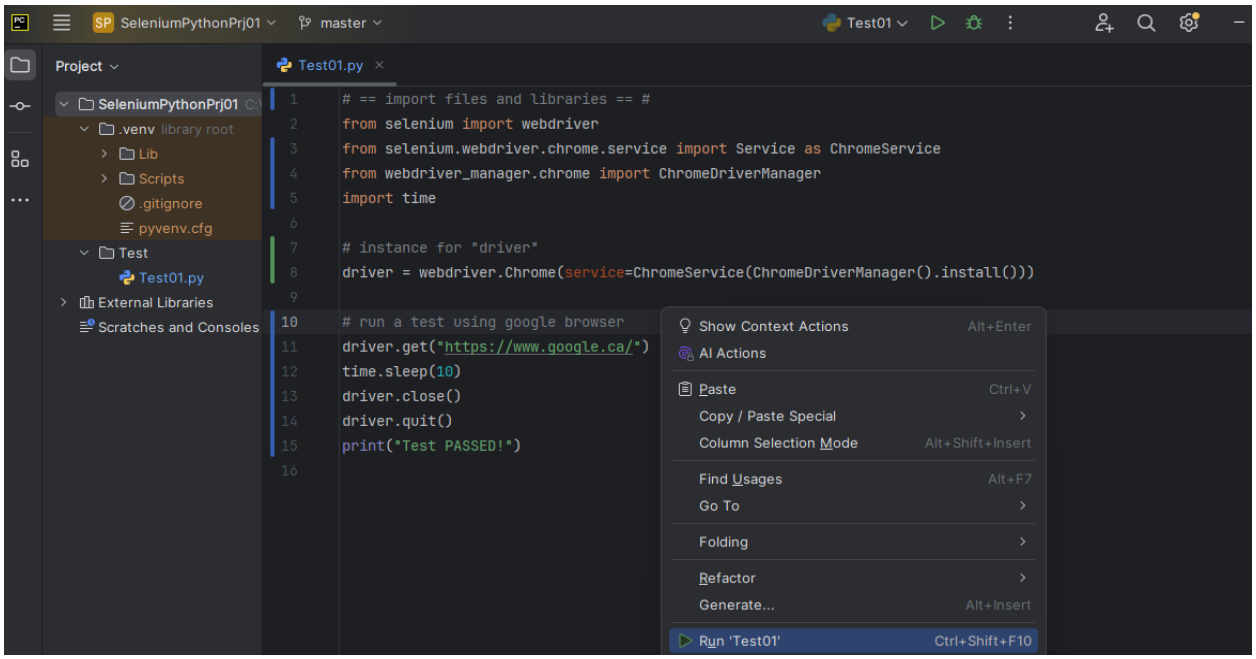
Create a New Project:

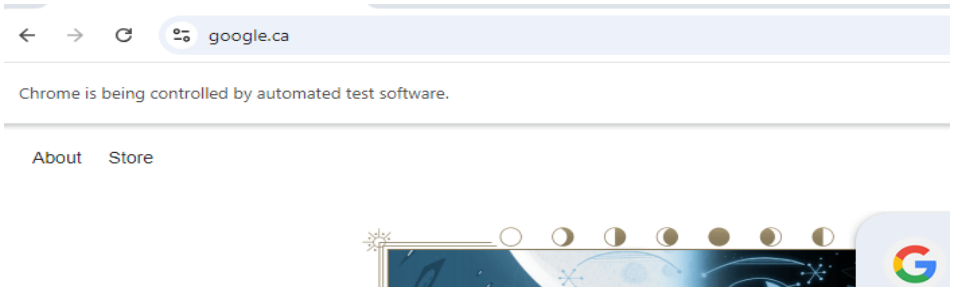


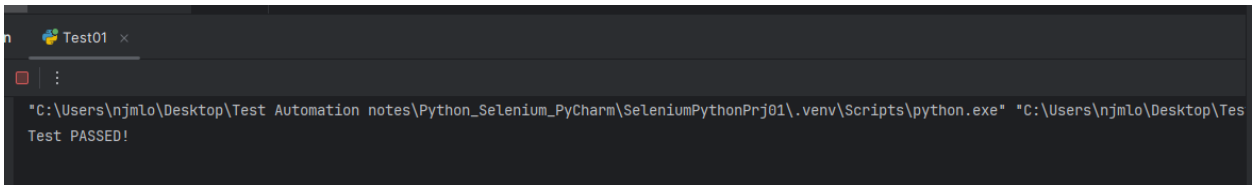
Create a folder and python file, run a quick test (this is to ensure initial components are setup properly):



2. Run a simple test using Google search page







3. Interacting with elements in a web page

Chrome…

PageElements.py

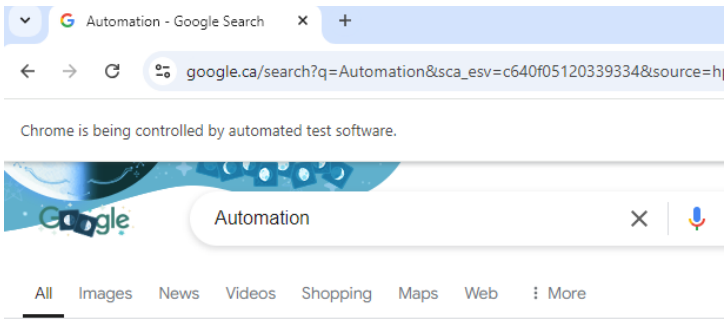
```python
# == import files and libraries == #
from selenium import webdriver
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.support.wait import WebDriverWait
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as ec
import time

# == extra import files and libraries == #
# from selenium.webdriver import Keys

# instance for CHROME Browser...
chromeBrowser = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))
# run a test using browser
chromeBrowser.get('https://www.google.ca')

# find the Search box...
googleSearchBox = chromeBrowser.find_element(By.ID, value: "APjFqb")
# insert a text...
googleSearchBox.send_keys("Automation")
# click the Google Search button...
waitChrome = WebDriverWait(chromeBrowser, timeout: 5)
elementChrome = waitChrome.until(ec.element_to_be_clickable((By.NAME,"btnK")))
chromeBrowser.find_element(By.NAME, value: "btnK").click()
# another way to test the search button (mirrors the action of hitting the enter button after typing what to search
# >> googleSearchBox.send_keys(Keys.RETURN)

# additional further actions...
time.sleep(5)
chromeBrowser.close()
chromeBrowser.quit()
# display a confirmation of test result...
print("Test Chrome PASSED!")
```

Edge…

```
# instance for EDGE Browser...
#   if __name__ == '__main__':
edgeBrowser = webdriver.Edge()
# run a test using browser
edgeBrowser.get('https://www.google.ca')

# find the Search box...
googleSearchBox = edgeBrowser.find_element(By.ID, value: "APjFqb")
# insert a text...
googleSearchBox.send_keys("Automation")
# click the Google Search button...
waitEdge = WebDriverWait(edgeBrowser, timeout: 5)
elementEdge = waitEdge.until(ec.element_to_be_clickable((By.NAME,"btnK")))
edgeBrowser.find_element(By.NAME, value: "btnK").click()
# another way to test the search button (mirrors the action of hitting the enter button after typing what to search
# >> googleSearchBox.send_keys(Keys.RETURN)

# additional further actions...
time.sleep(5)
edgeBrowser.close()
edgeBrowser.quit()
# display a confirmation of test result...
print("Test Edge PASSED!")
```

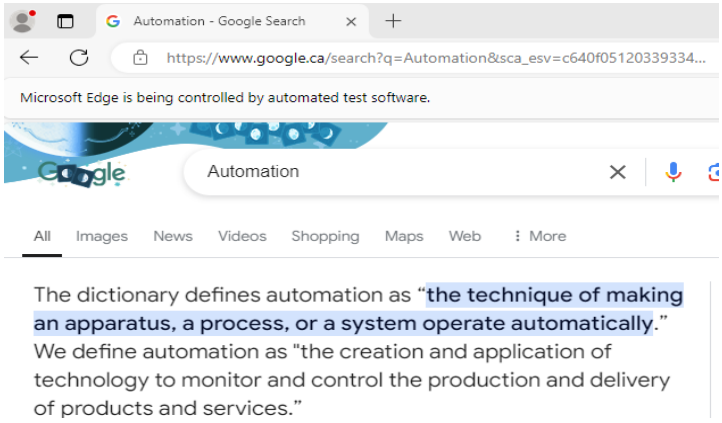
```
Run   PageElements
"C:\Users\njmlo\Desktop\Test Automation notes\Python_Selenium_PyCharm\SeleniumPythonPrj01\.venv\Scripts\python.exe" "C:\Users\njmlo\Desktop\Tes
Test Chrome PASSED!
Test Edge PASSED!

Process finished with exit code 0
```

Chrome…



The dictionary defines automation as "the technique of making an apparatus, a process, or a system operate automatically." We define automation as "the creation and application of technology to monitor and control the production and delivery of products and services."

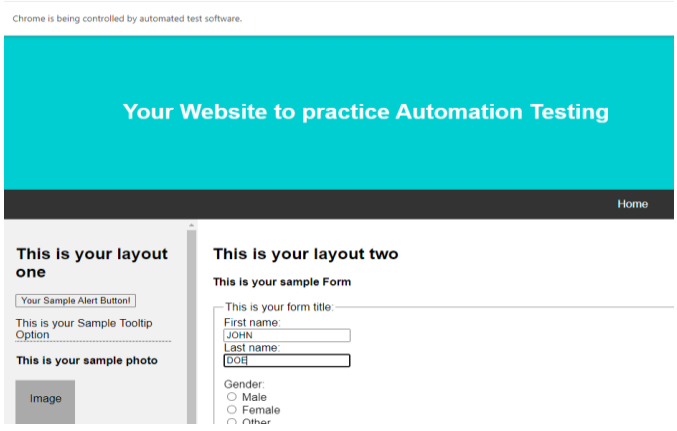
Edge…



Run another test calling a different test page…

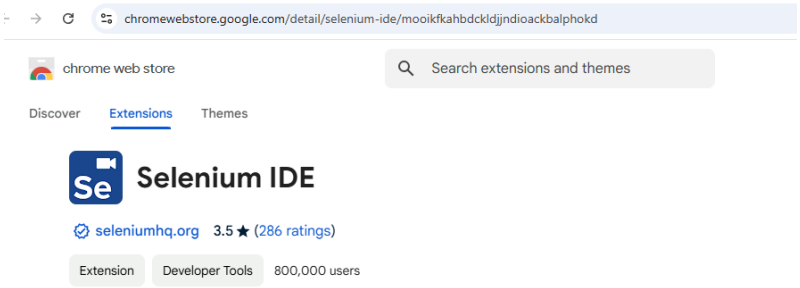
```
# another instance for CHROME Browser...
chromeBBrowser = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))
# run another  test using browser
chromeBBrowser.get('https://trytestingthis.netlify.app')
# insert text(s)...
time.sleep(5)
chromeBBrowser.find_element(By.ID,  value: "fname").send_keys("JOHN")
chromeBBrowser.find_element(By.ID,  value: "lname").send_keys("DOE")
time.sleep(5)
chromeBBrowser.find_element(By.XPATH,  value: "//button[@class = 'btn btn-success']").click()

# additional further actions...
time.sleep(5)
chromeBBrowser.close()
chromeBBrowser.quit()
# display a confirmation of test result...
print("Test Chrome (B) PASSED!")
```

```
"C:\Users\njmlo\Desktop\Test Automation notes\Python_Selenium_PyCharm\SeleniumPythonPrj01\.venv\Scripts\python.exe" "C:\Users\njmlo\Desktop\T
Test Chrome (A) PASSED!
Test Chrome (B) PASSED!
Test Edge PASSED!

Process finished with exit code 0
```

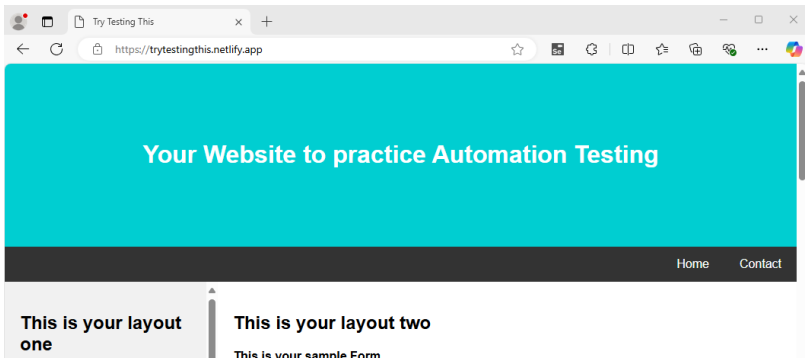


4. Record, Playback, and Export (Selenium IDE)

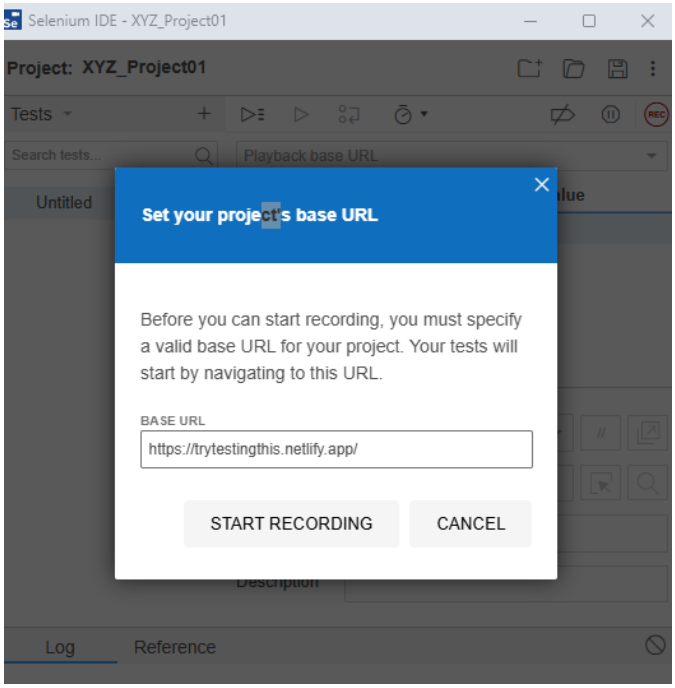
Download Selenium IDE and add to Chrome:
https://www.selenium.dev/selenium-ide/



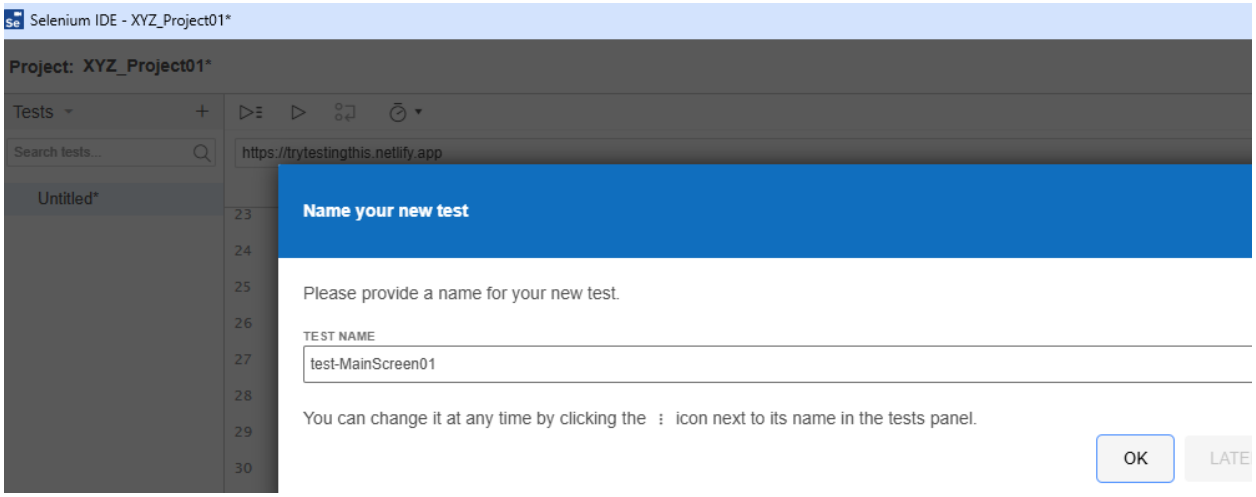
Test web site:
https://trytestingthis.netlify.app/

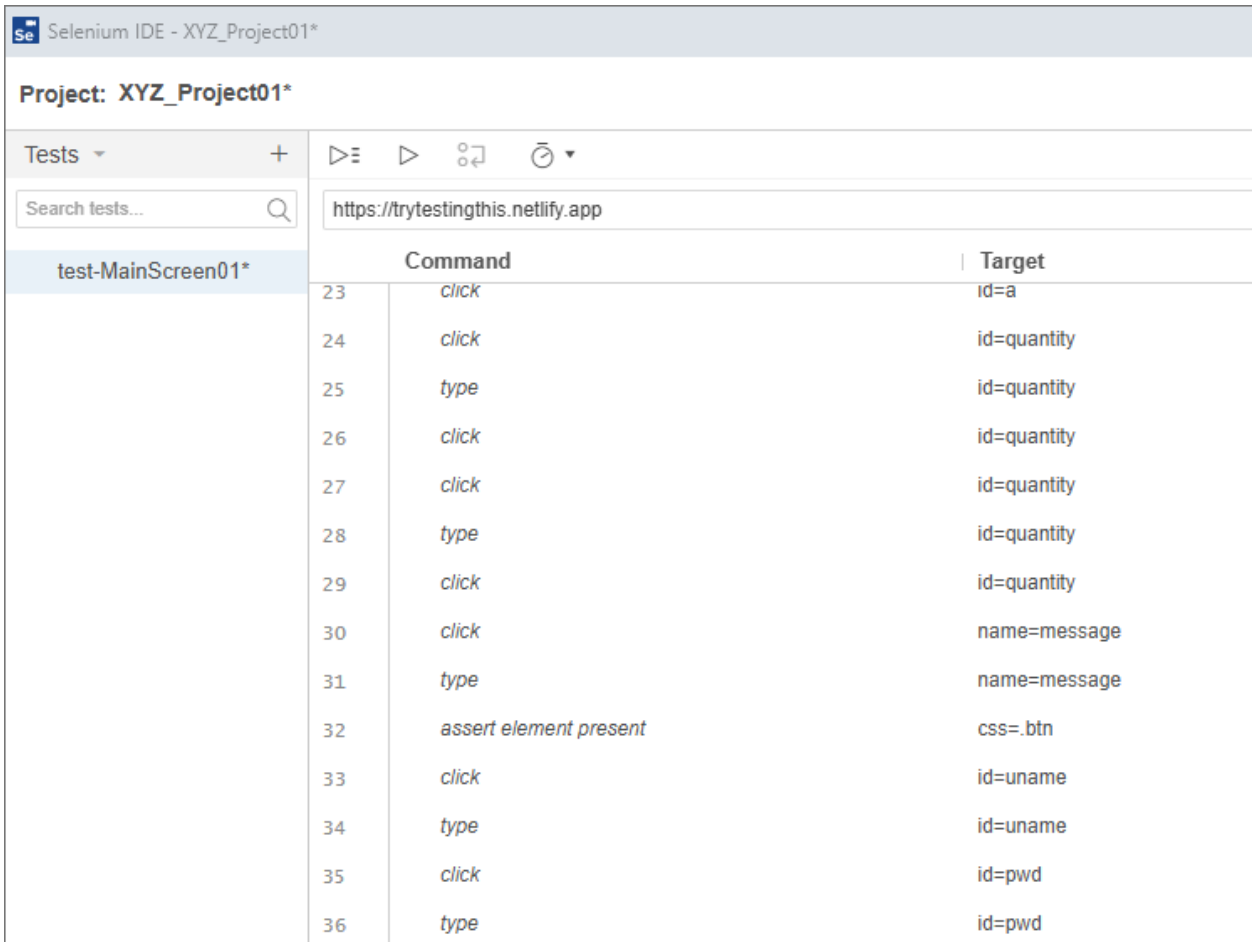


Record the test…

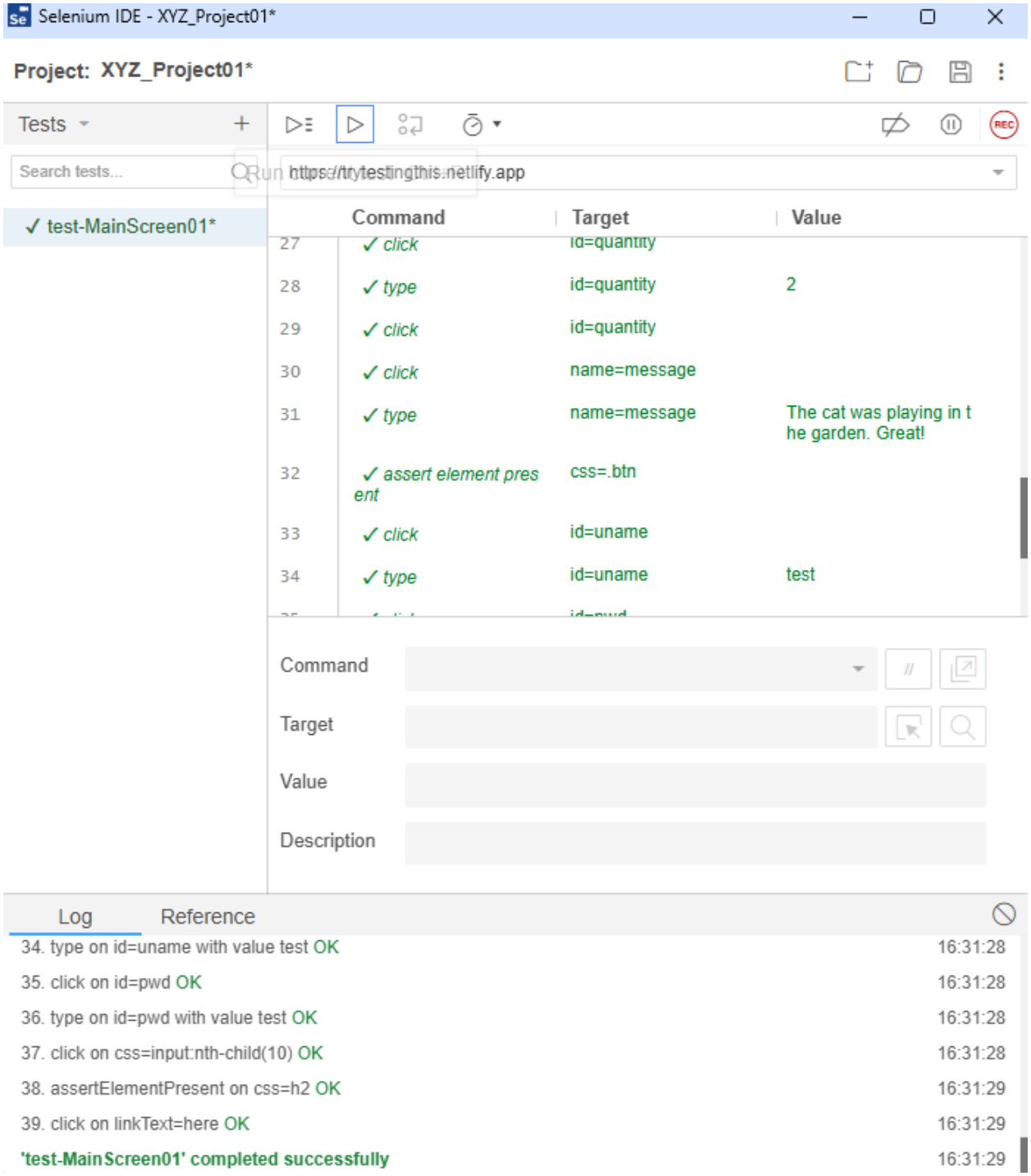


Stop recording…

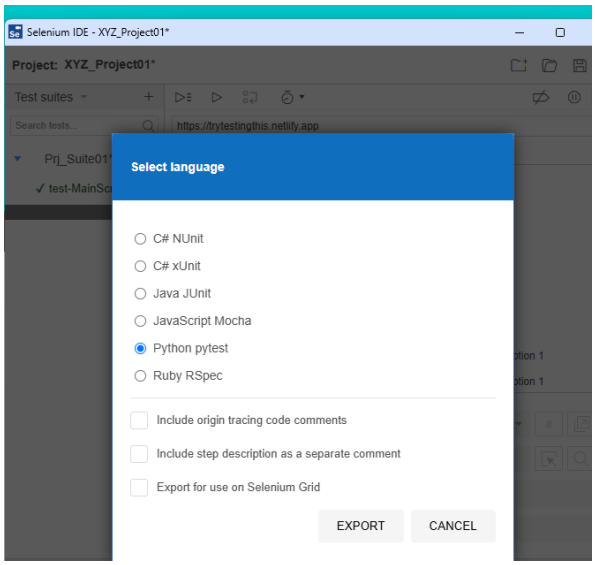




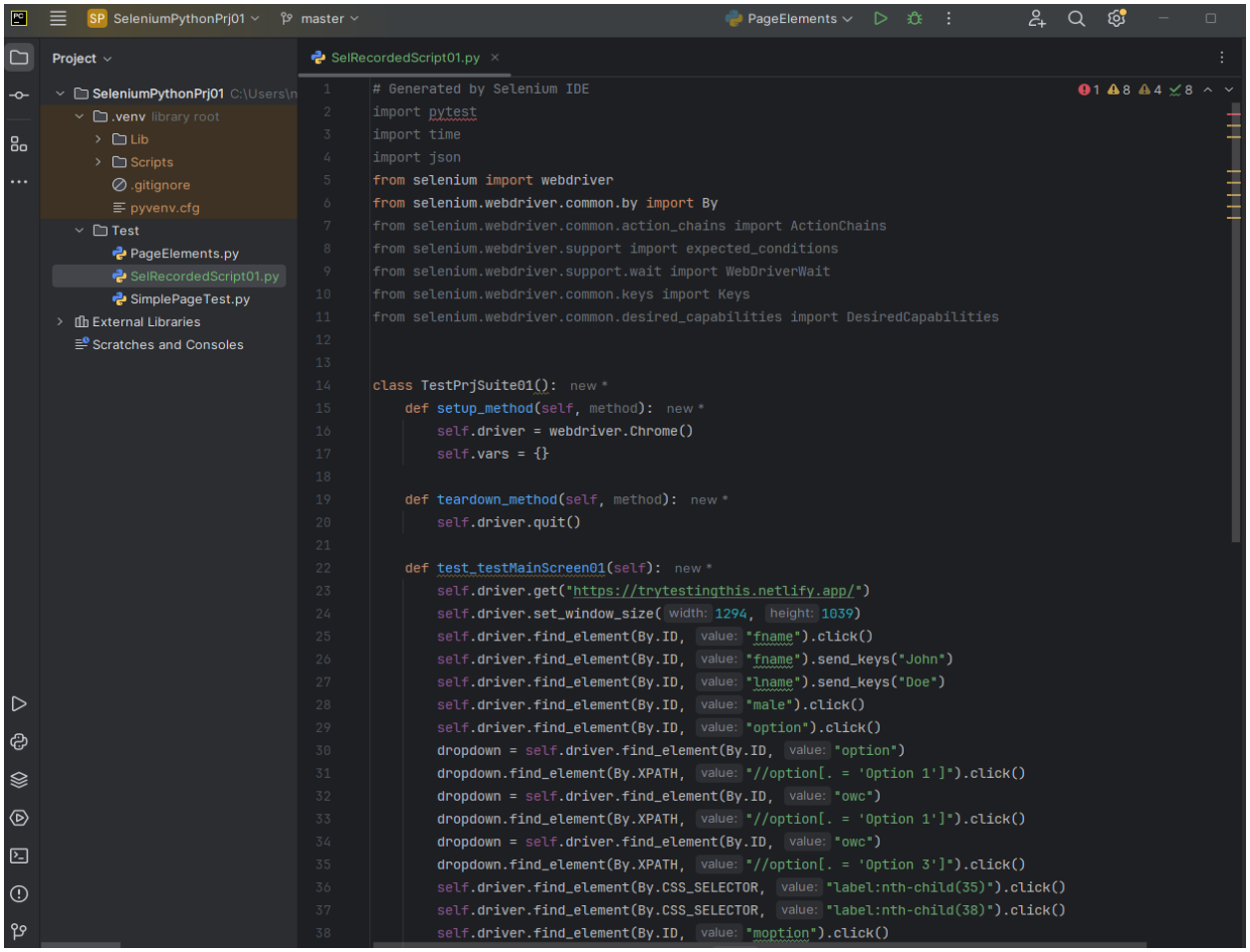
(Re)run the test…



Export suite…



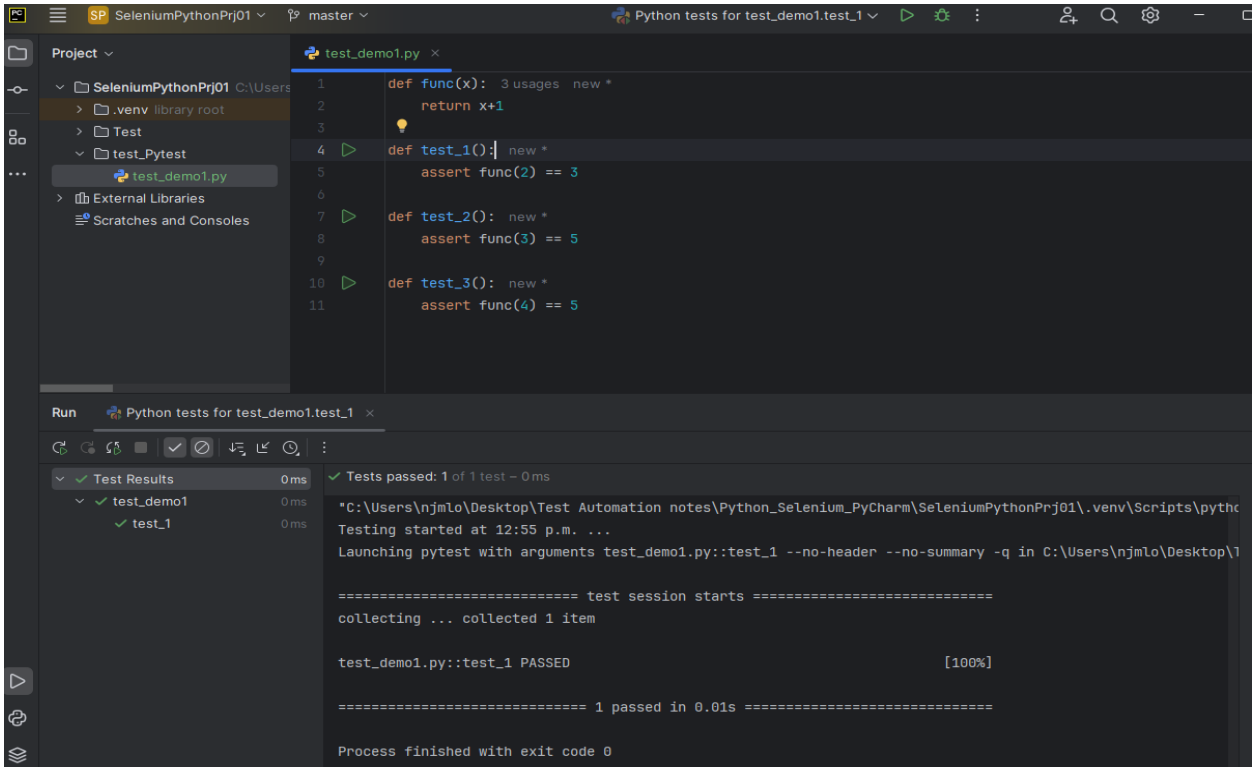
Create a python file in editor and dump the (recorded) code…

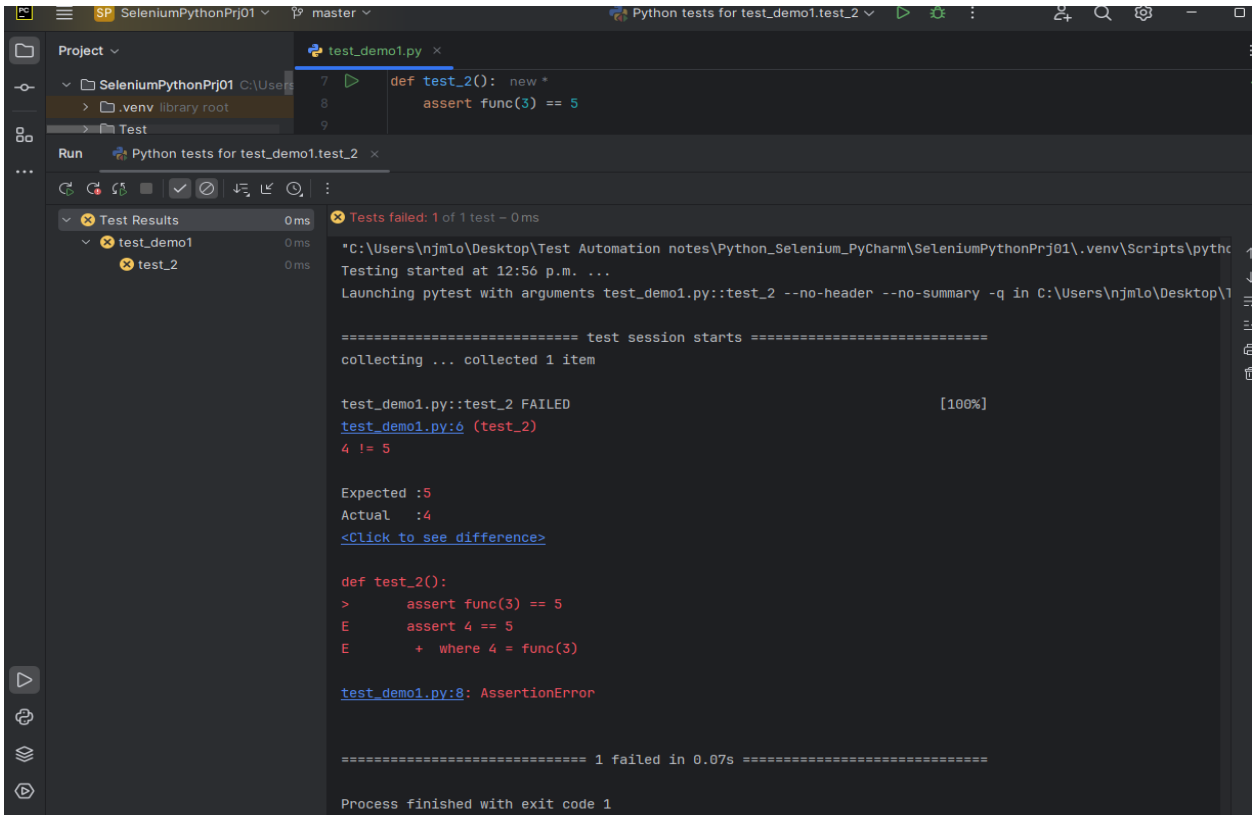


Note: this is one way to help in identifying what unique attribute can be used for element under test

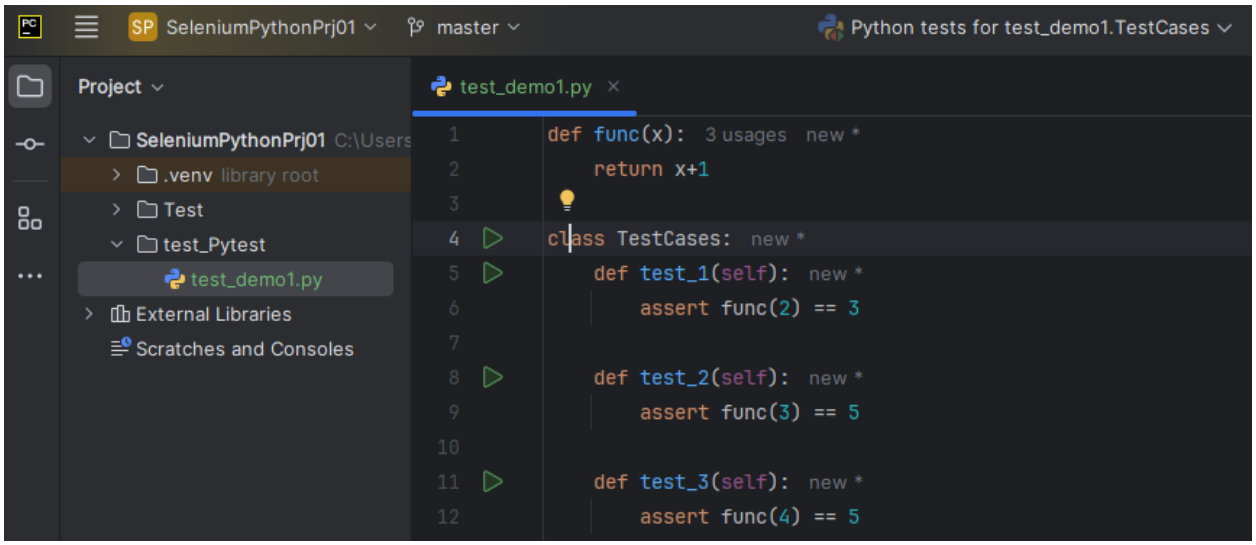
5. Using Pytest with Selenium and Python

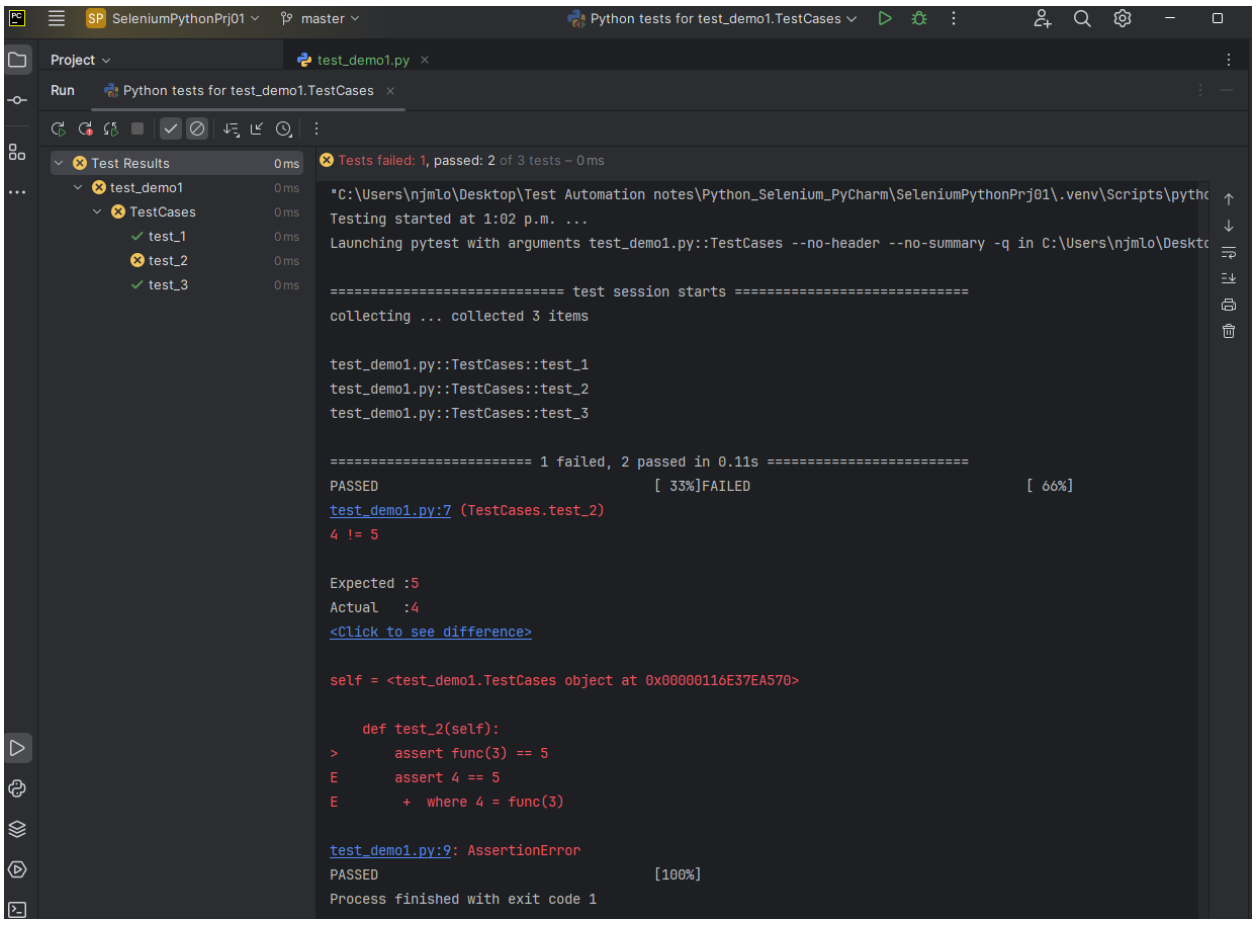
Quick test run of passed and failed cases…



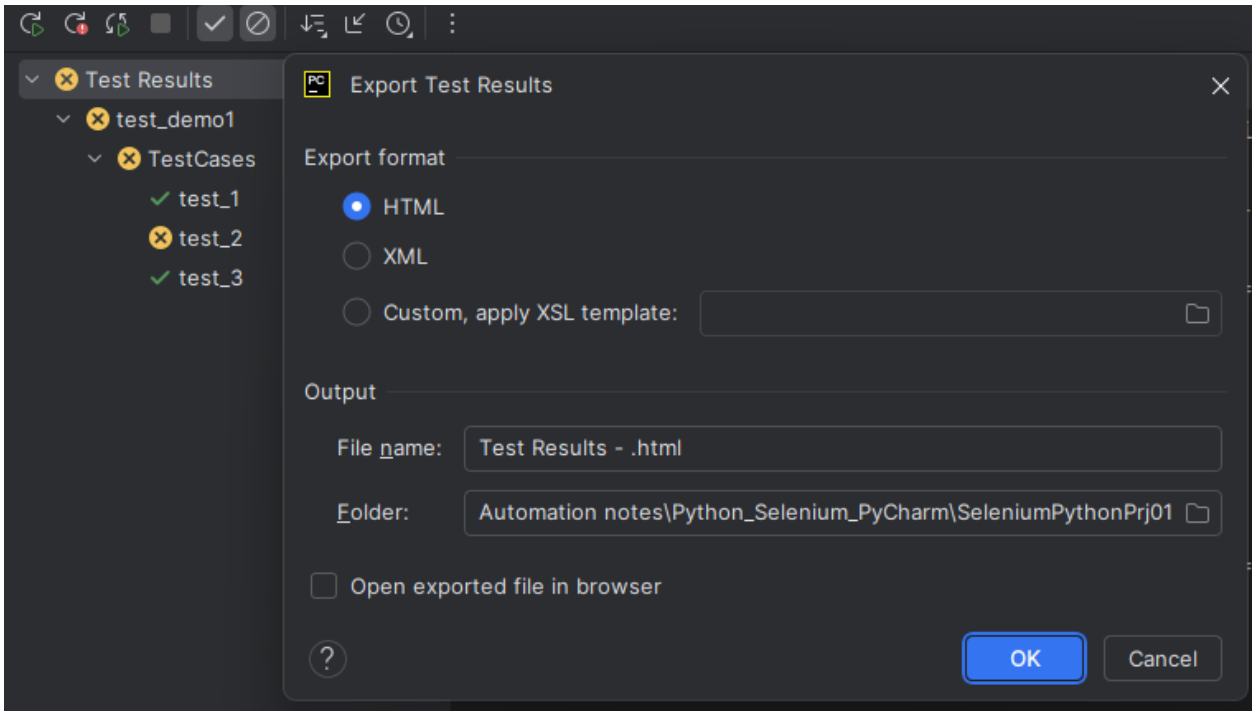


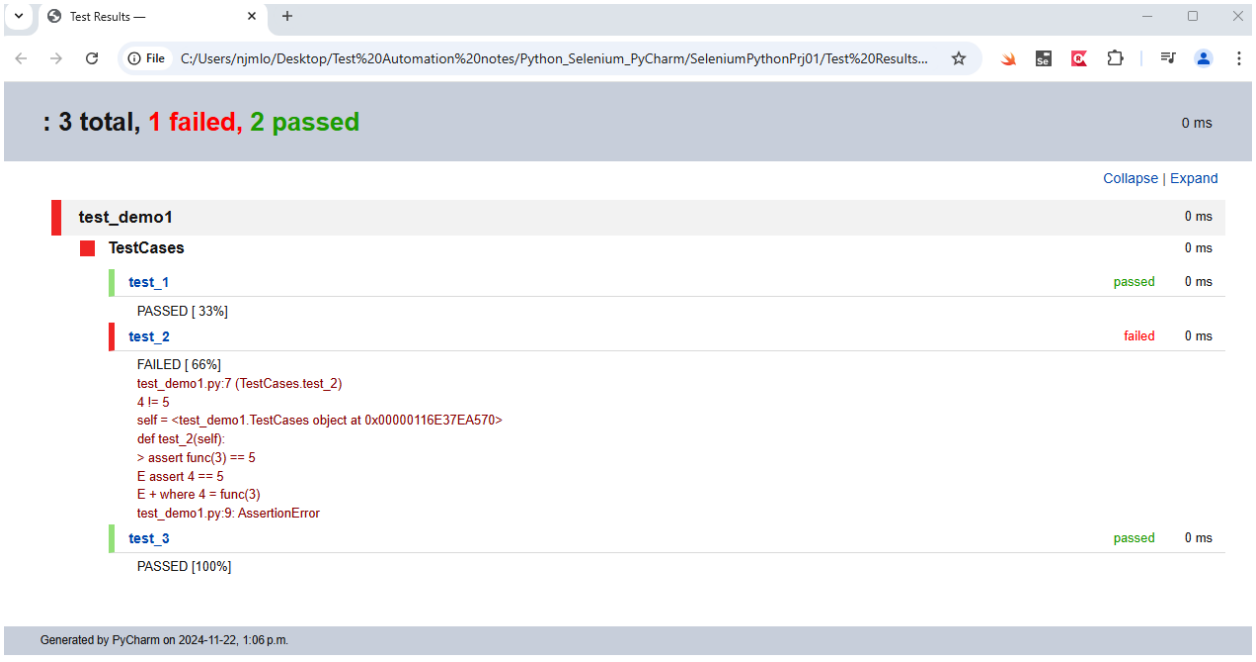
Create and run a class for the set of test cases…



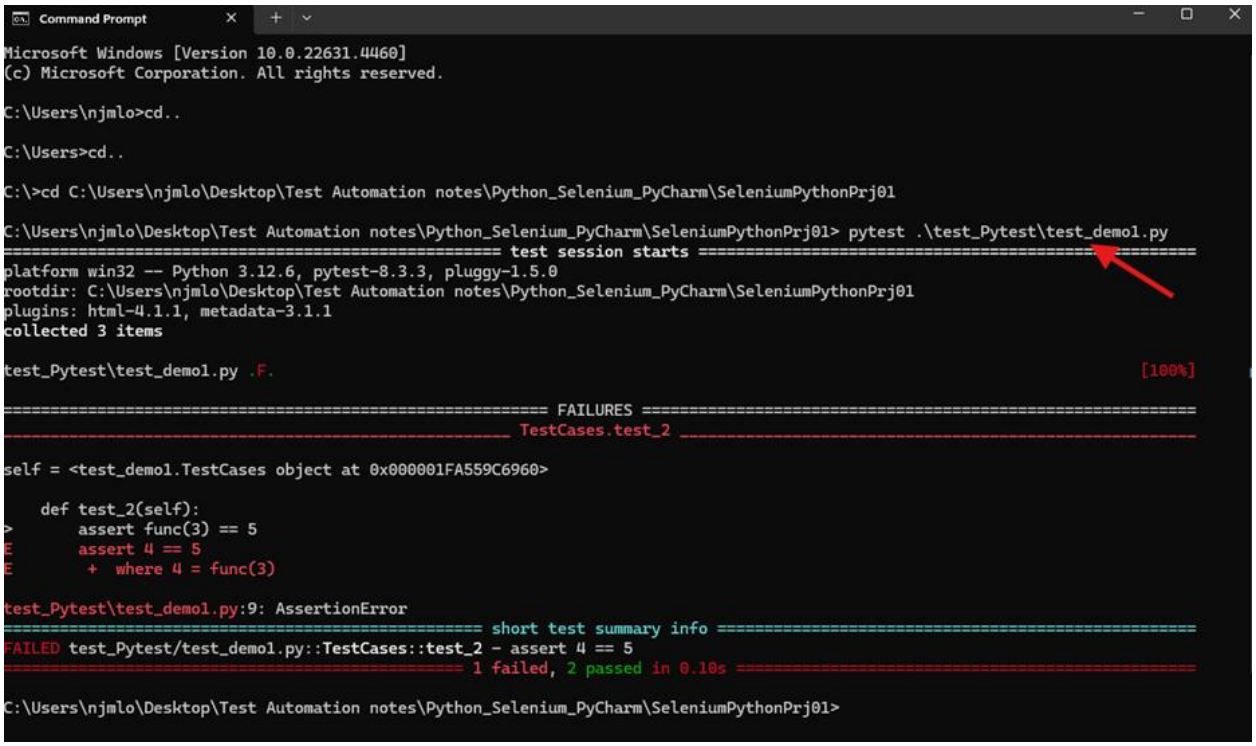


Export the test result...

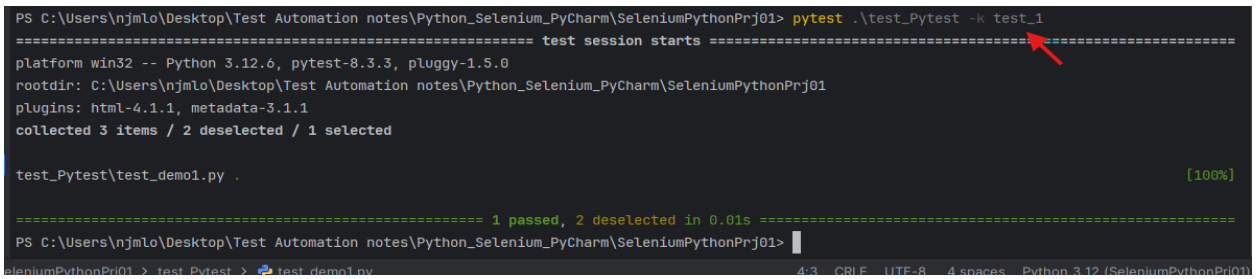


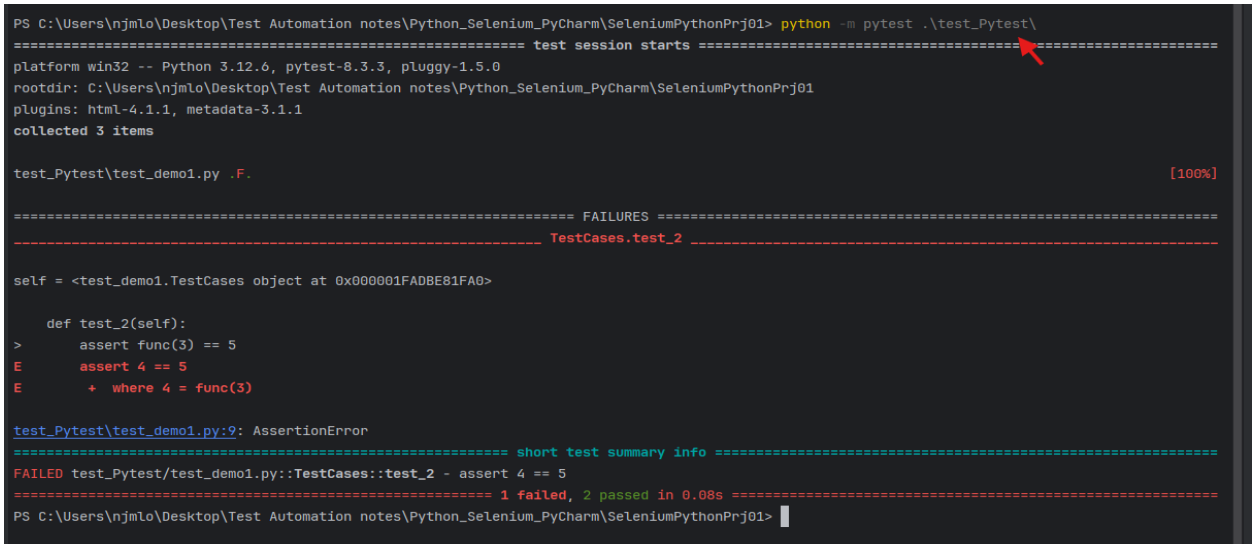


Run Pytest from the cmd line or IDE terminal…

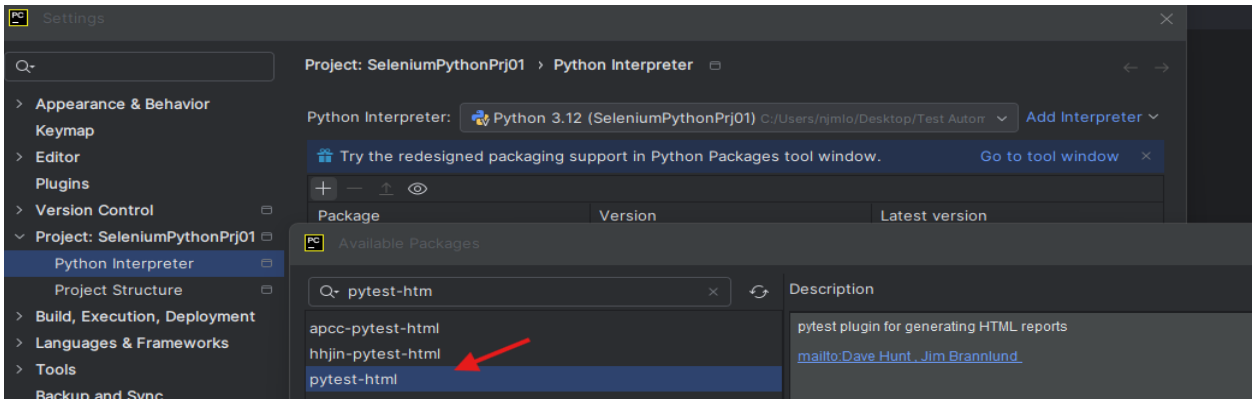


Using the cmd line to find a keyword in the test case to run…

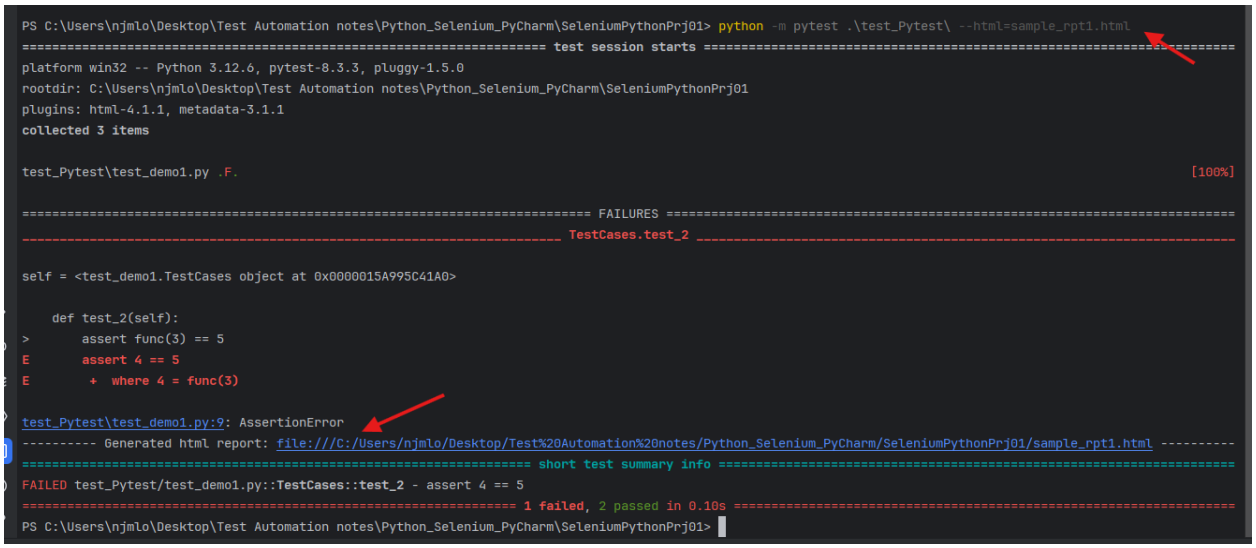




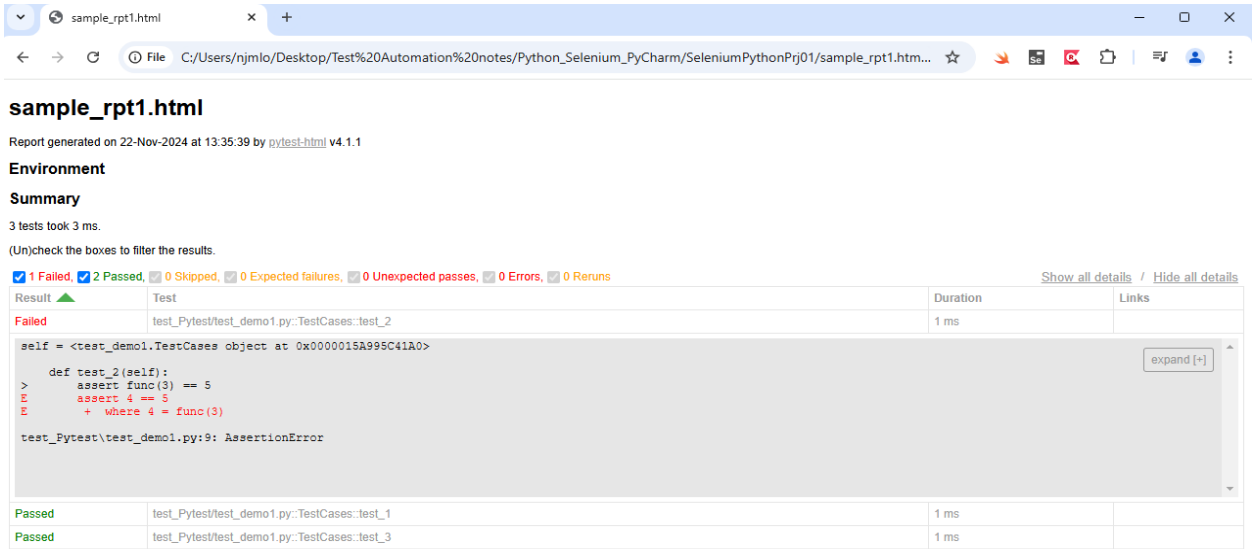
Add another type of reporting tool…



Using the cmd line or terminal…

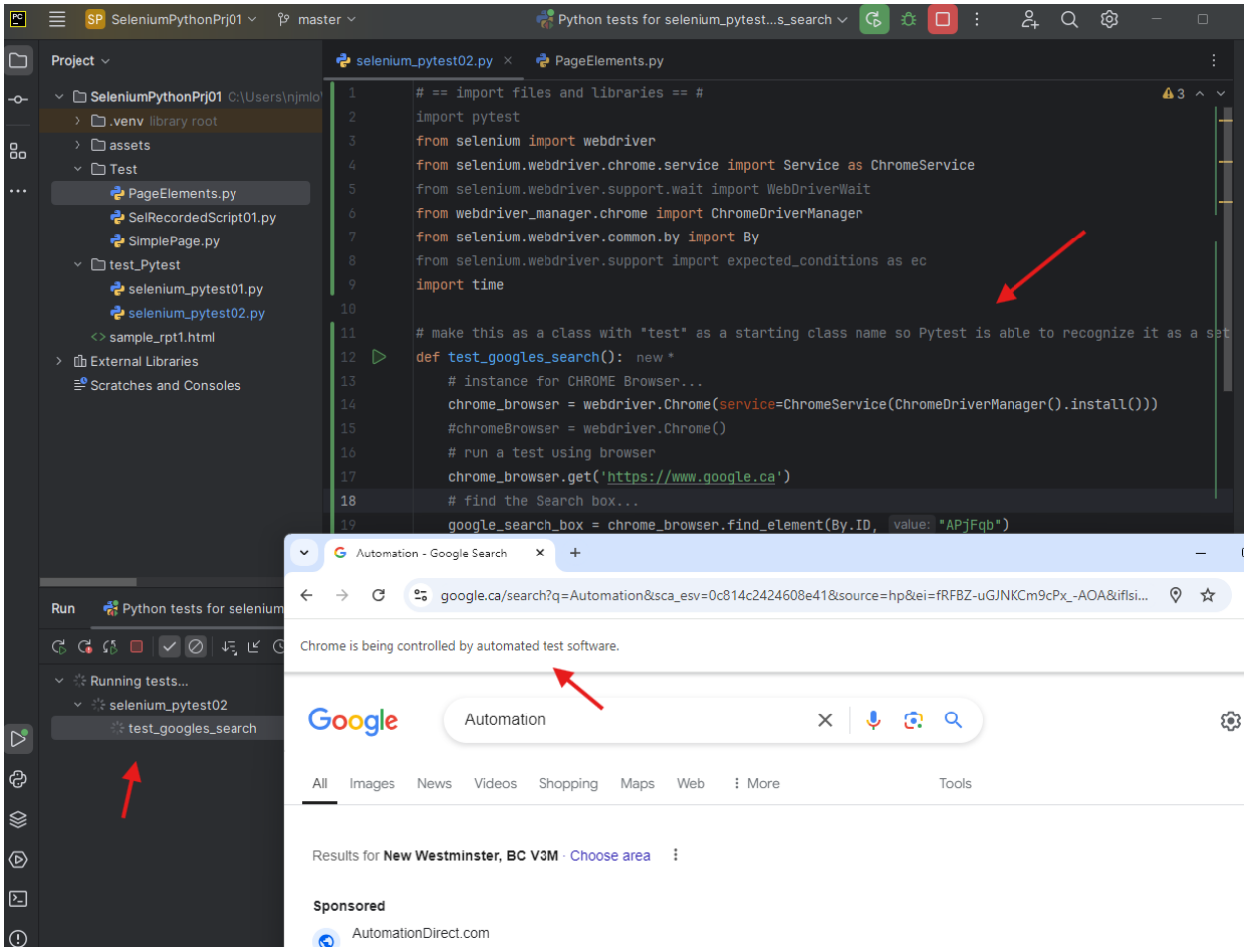


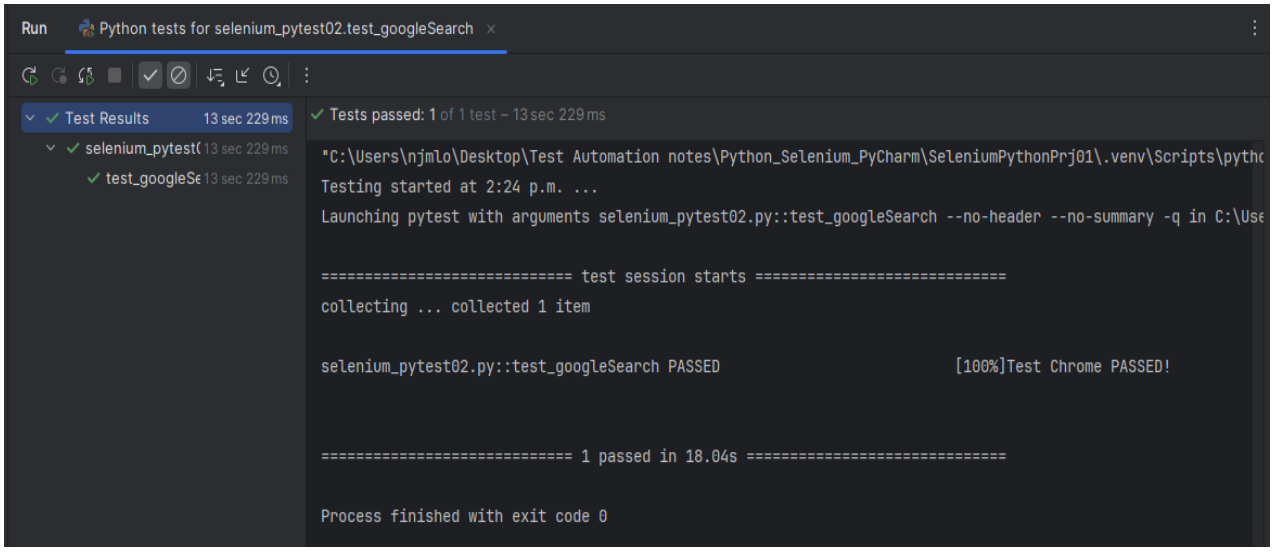
View the generated report…



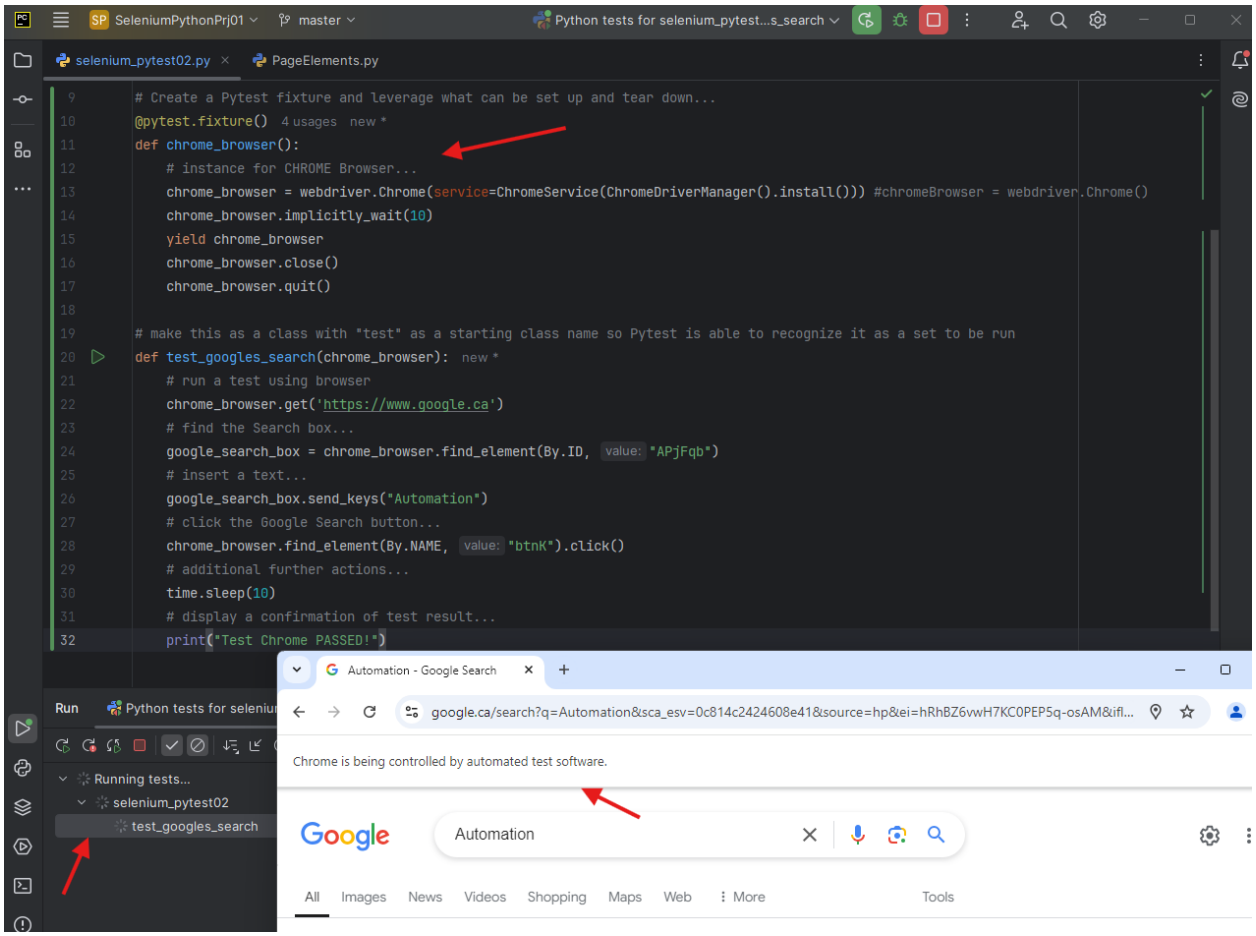
6. More Selenium, Python, Pytest sampling codes and test run

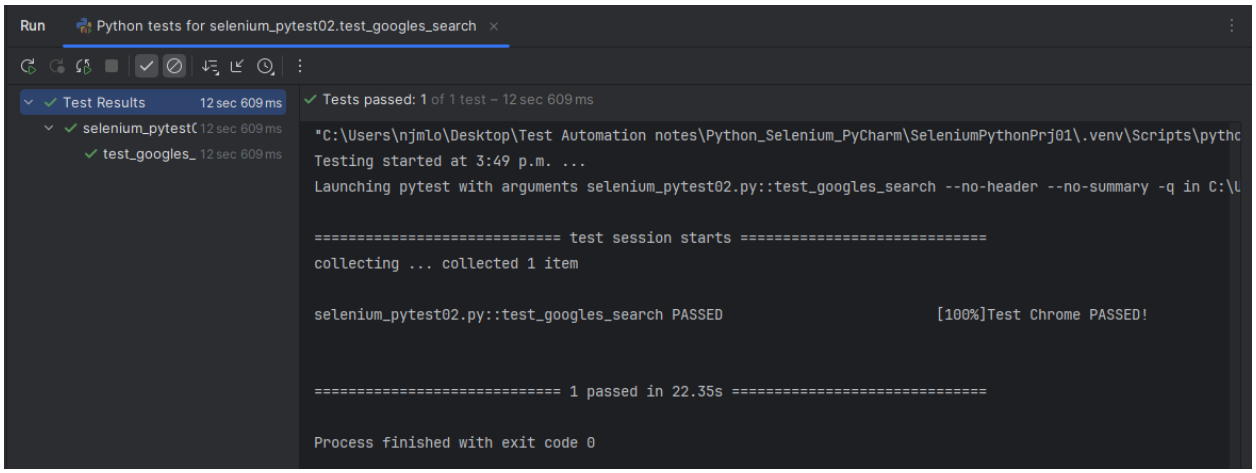
Run a google search test…





Create a Pytest fixtures and leverage what can be set up and tear down…

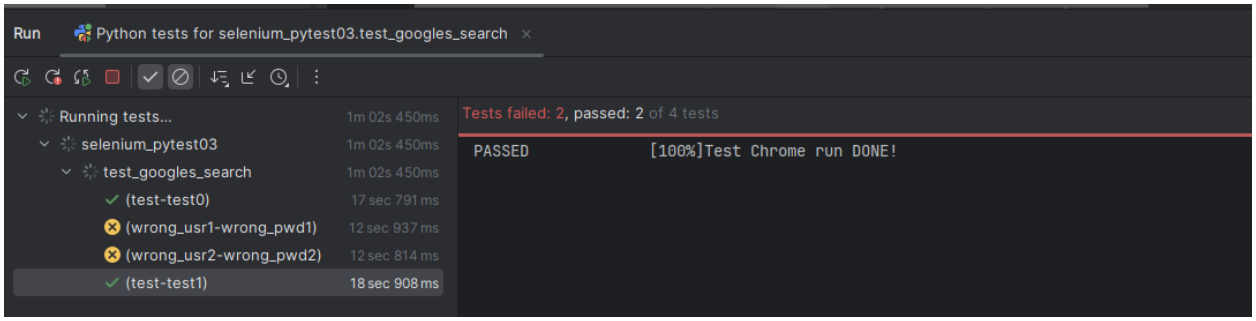




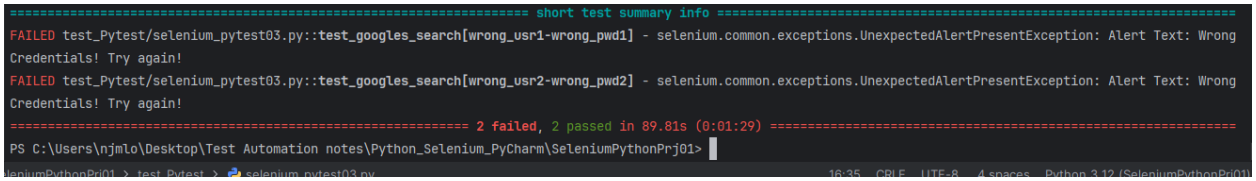
Parameterization in Pytest…

selenium_pytest03.py

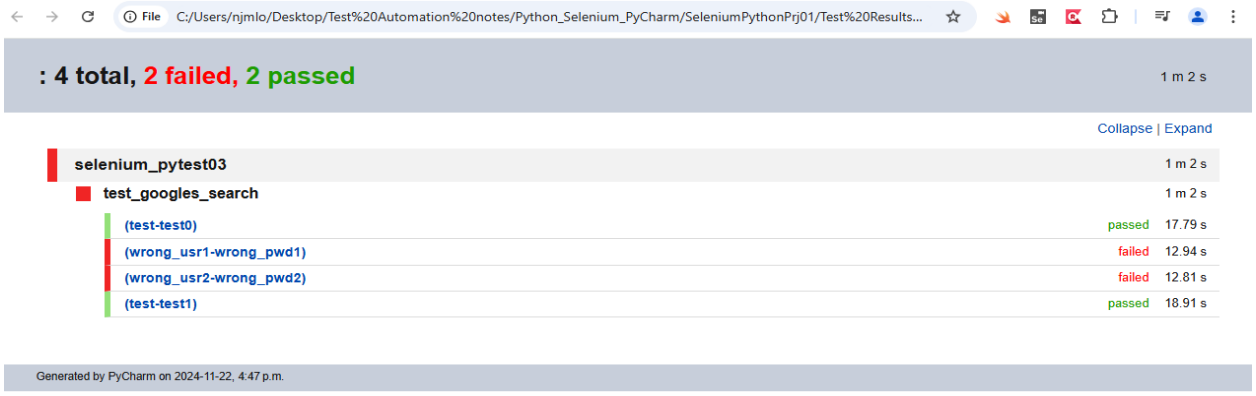
```python
# Create a Pytest fixture and leverage what can be set up and tear down...
@pytest.fixture()  6 usages
def chrome_browser():
    # instance for CHROME Browser...
    service = ChromeService(ChromeDriverManager().install())
    chrome_browser = webdriver.Chrome(service=service)
    # chrome_browser = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install())) #chromeBrowser = webdrive
    # additional further actions...
    chrome_browser.implicitly_wait(5)
    yield chrome_browser
    chrome_browser.close()
    chrome_browser.quit()

# Parameterization in Pytest...
@pytest.mark.parametrize('username, password',[
    ('test','test'),
    ('wrong_usr1', 'wrong_pwd1'),
    ('wrong_usr2', 'wrong_pwd2'),
    ('test', 'test'),
])

# make this as a class with "test" as a starting class name so Pytest is able to recognize it as a set to be run
def test_googles_search(chrome_browser, username, password):
    # run another  test using browser
    chrome_browser.get('https://trytestingthis.netlify.app')
    time.sleep(10)
    chrome_browser.find_element(By.ID,  value: "uname").send_keys(username)
    chrome_browser.find_element(By.ID,  value: "pwd").send_keys(password)
    time.sleep(2)
    chrome_browser.find_element(By.XPATH,  value: "//input[@value = 'Login']").click()
    # additional further actions...
    assert "Successful" in chrome_browser.page_source
    time.sleep(5)
    # display a confirmation of test result...
    print("Test Chrome run DONE!")
```

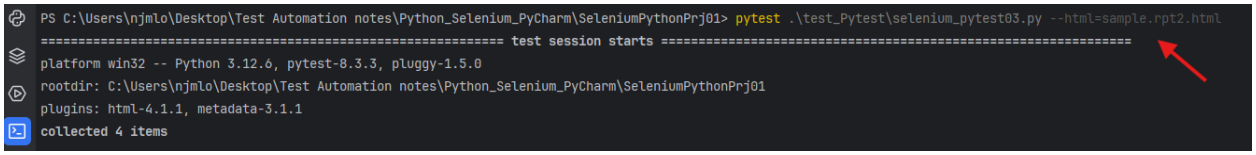


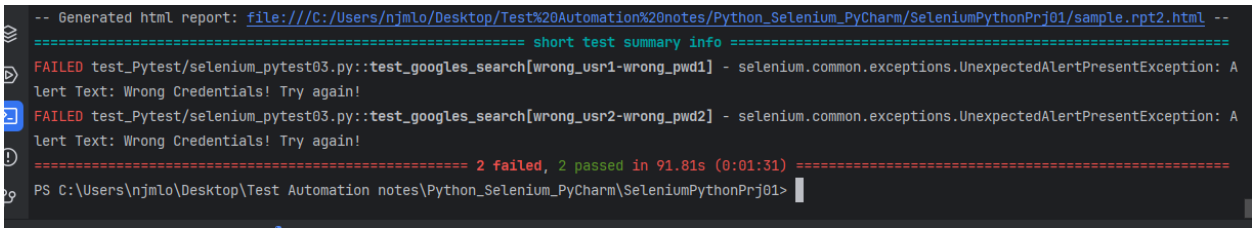
Using the cmd line or terminal…

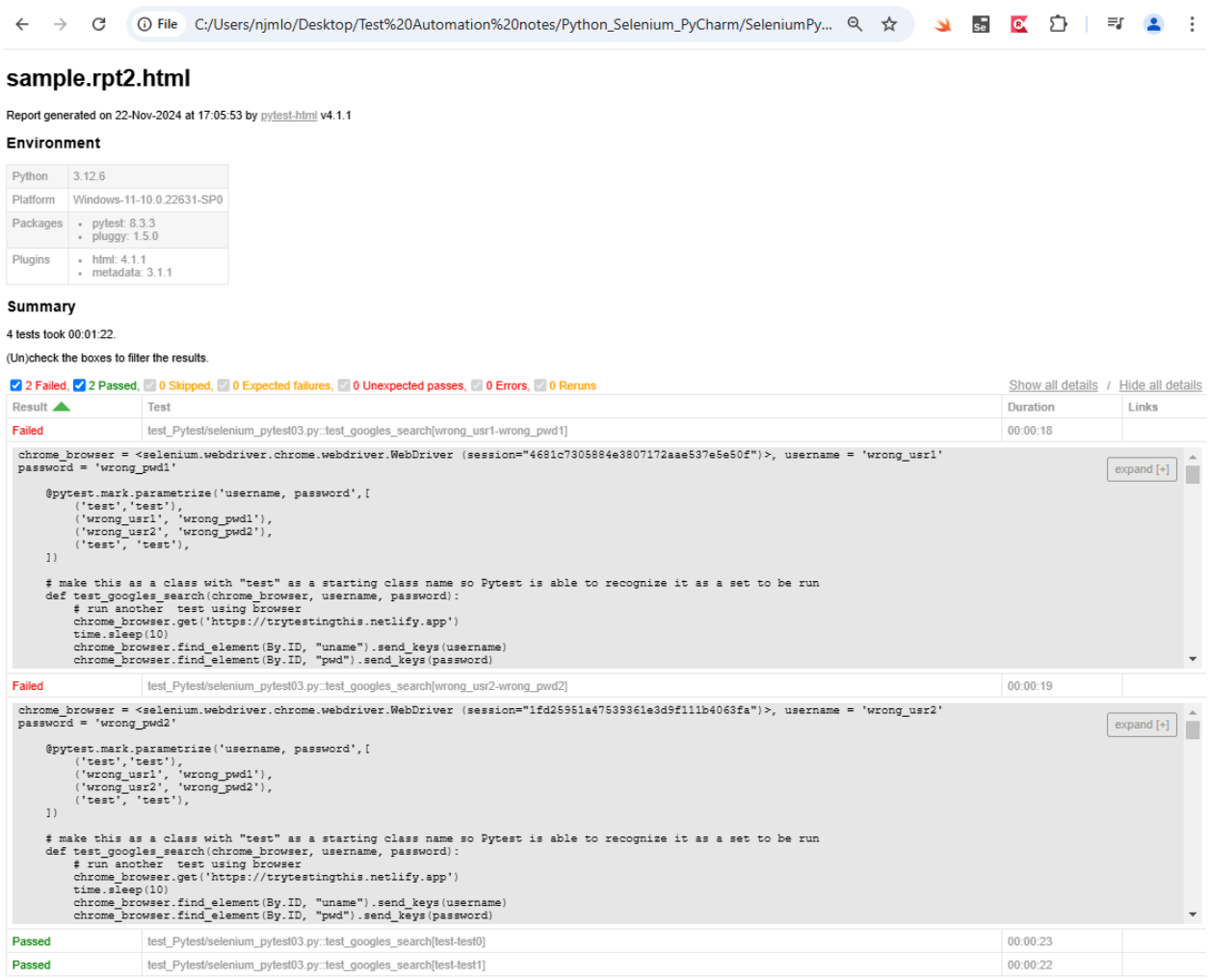


Export the test result…

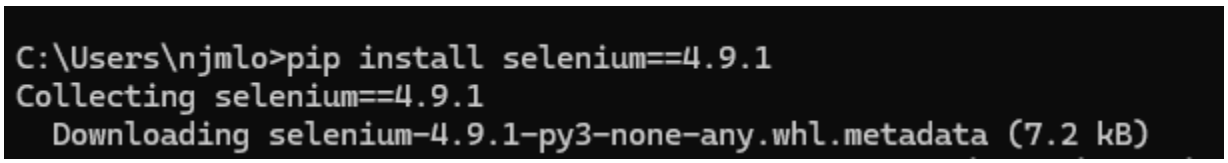






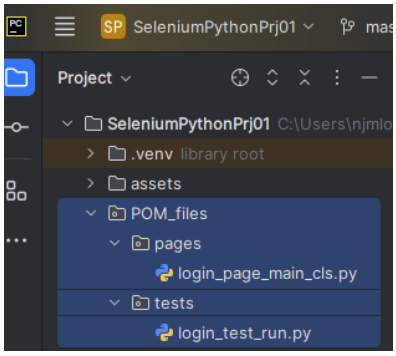


Note: if running the test in the command line is throwing issues, downgrade (for now) the Selenium to version 4.9.1…



7. Page Object Model (POM)

Create folders and files…



Web elements and methods (in login_page_main_cls.py file)…

login_test_run.py | login_page_main_cls.py

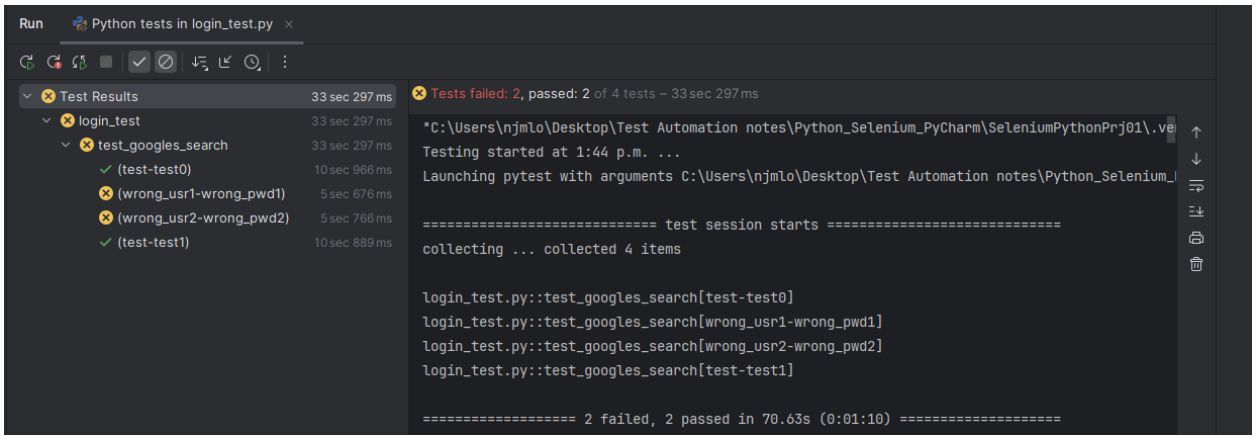
```python
# == import files and libraries == #
from selenium.webdriver.common.by import By

# define the class...
class LoginPage:  2 usages  njmlopez17

    # define the constructor function that takes the driver parameter...
    def __init__(self, driver):  njmlopez17
        self.driver = driver
        # identify the page element locators...
        self.username_textbox = (By.ID, "uname")
        self.password_textbox = (By.ID,"pwd")
        self.login_button = (By.XPATH, "//input[@value = 'Login']")

    # define the action function...
    def login_actions (self, url, username, password):  1 usage  njmlopez17
        # open the page...
        self.driver.get(url)
        # enter username...
        self.driver.find_element(*self.username_textbox).send_keys(username)
        # enter password...
        self.driver.find_element(*self.password_textbox).send_keys(password)
        # click button...
        self.driver.find_element(*self.login_button).click()
        # additional further actions...
        assert "Successful" in self.driver.page_source
```

Test methods (in login_test_run.py file)...

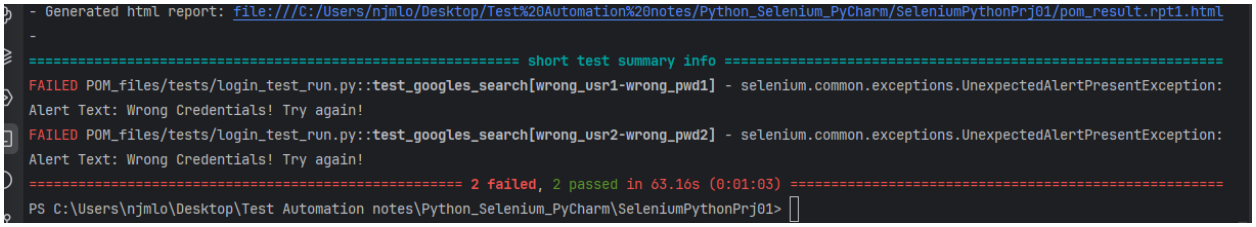
login_test_run.py | login_page_main_cls.py

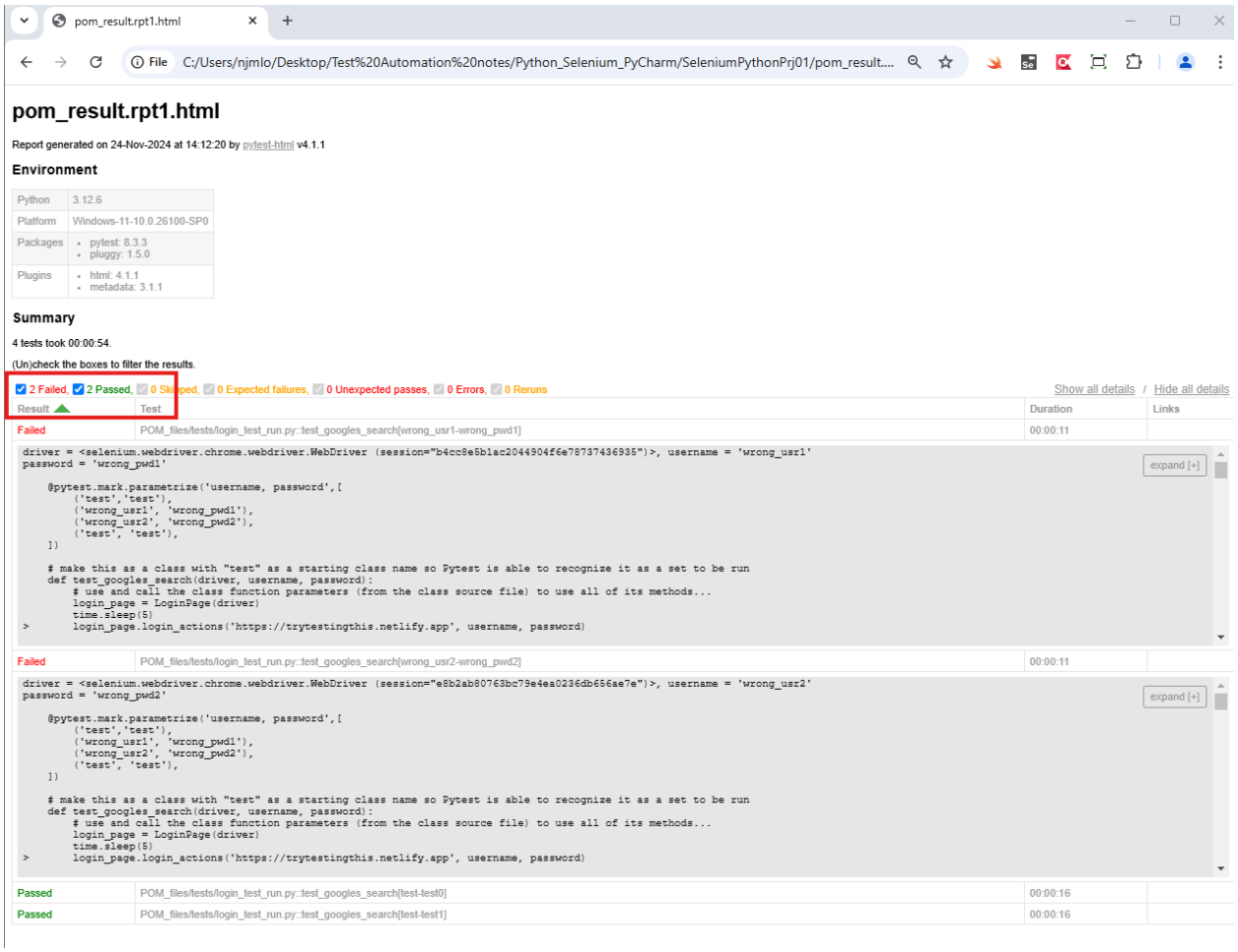
```python
# == import files and libraries == #
import pytest
from selenium import webdriver
from selenium.webdriver.chrome.service import Service as ChromeService
from webdriver_manager.chrome import ChromeDriverManager
import time

# import the login page class so it can be used in this set of test run...
from POM_files.pages.login_page_main_cls import LoginPage

# Create a Pytest fixture and leverage what can be set up and tear down...
@pytest.fixture()  2 usages  njmlopez17
def driver():
    # instance for CHROME Browser...
    service = ChromeService(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service)
    # additional further actions...
    driver.implicitly_wait(5)
    yield driver
    driver.close()
    driver.quit()

# Parameterization in Pytest...
@pytest.mark.parametrize('username, password',[  njmlopez17
    ('test','test'),
    ('wrong_usr1', 'wrong_pwd1'),
    ('wrong_usr2', 'wrong_pwd2'),
    ('test', 'test'),
])

# make this as a class with "test" as a starting class name so Pytest is able to recognize it as a set to be run
def test_googles_search(driver, username, password):
    # use and call the class function parameters (from the class source file) to use all of its methods...
    login_page = LoginPage(driver)
    time.sleep(5)
    login_page.login_actions( url: 'https://trytestingthis.netlify.app', username, password)
    time.sleep(5)
    # display a confirmation of test result...
    print("Test Chrome (login page) DONE!")
```

Test results...



Test report...




pom_result.rpt1.html
File C:/Users/njmlo/Desktop/Test%20Automation%20notes/Python_Selenium_PyCharm/SeleniumPythonPrj01/pom_result....
pom_result.rpt1.html
Report generated on 24-Nov-2024 at 14:12:20 by pytest-html v4.1.1
Environment
Python 3.12.6
Platform Windows-11-10.0.26100-SP0
Packages pytest: 8.3.3 pluggy: 1.5.0
Plugins html: 4.1.1 metadata: 3.1.1
Summary
4 tests took 00:00:54.
(Un)check the boxes to filter the results.
2 Failed, 2 Passed, 0 Skipped, 0 Expected failures, 0 Unexpected passes, 0 Errors, 0 Reruns
Show all details / Hide all details
Result Test Duration Links
Failed POM_files/tests/login_test_run.py::test_googles_search[wrong_usr1-wrong_pwd1] 00:00:11
driver = <selenium.webdriver.chrome.webdriver.WebDriver (session="b4cc8e5b1ac2044904f6e787374369 35")>, username = 'wrong_usr1'
password = 'wrong_pwd1'
@pytest.mark.parametrize('username, password',[
('test','test'),
('wrong_usr1', 'wrong_pwd1'),
('wrong_usr2', 'wrong_pwd2'),
('test', 'test'),
])
# make this as a class with "test" as a starting class name so Pytest is able to recognize it as a set to be run
def test_googles_search(driver, username, password):
# use and call the class function parameters (from the class source file) to use all of its methods...
login_page = LoginPage(driver)
time.sleep(5)
> login_page.login_actions('https://trytestingthis.netlify.app', username, password)
expand [+]
Failed POM_files/tests/login_test_run.py::test_googles_search[wrong_usr2-wrong_pwd2] 00:00:11
driver = <selenium.webdriver.chrome.webdriver.WebDriver (session="e8b2ab80763bc79e4ea0236db656ae7e")>, username = 'wrong_usr2'
password = 'wrong_pwd2'
@pytest.mark.parametrize('username, password',[
('test','test'),
('wrong_usr1', 'wrong_pwd1'),
('wrong_usr2', 'wrong_pwd2'),
('test', 'test'),
])
# make this as a class with "test" as a starting class name so Pytest is able to recognize it as a set to be run
def test_googles_search(driver, username, password):
# use and call the class function parameters (from the class source file) to use all of its methods...
login_page = LoginPage(driver)
time.sleep(5)
> login_page.login_actions('https://trytestingthis.netlify.app', username, password)
expand [+]
Passed POM_files/tests/login_test_run.py::test_googles_search[test-test0] 00:00:16
Passed POM_files/tests/login_test_run.py::test_googles_search[test-test1] 00:00:16

**********END**********