Hands-On Web UI Testing

Andrew Knight
PyOhio 2019

I’m Pandy.
I love testing.

Twitter:
@AutomationPanda

Developers?

Testers?

Data Scientists?

Other Roles?

Web UI testing can be hard.
Let’s make it easy.
We have **2 hours**.

Agenda

1.	Test Project Setup	(Independent)
2.	Web UI Testing Overview	(Lecture)
3.	Writing Our First Test	(Guided)
4.	Defining Page Objects	(Guided)
5.	Setting Up Selenium WebDriver	(Guided)
6.	Making WebDriver Calls	(Guided)
7.	Improving the Solution	(Lecture)
8.	Writing More Tests	(Independent)

Test Project Setup

Clone the test project and follow the README’s setup instructions:

```
git clone https://github.com/AndyLPK247/pyohio-2019-web-ui-testing.git
```

Requirements:

- Git
- Python 3.6 or higher
- Pipenv (“pip install pipenv”)
- Google Chrome (latest version)
- ChromeDriver (matching version; on system path)

Web UI Testing Overview

How would you define “testing”?

Testing Types

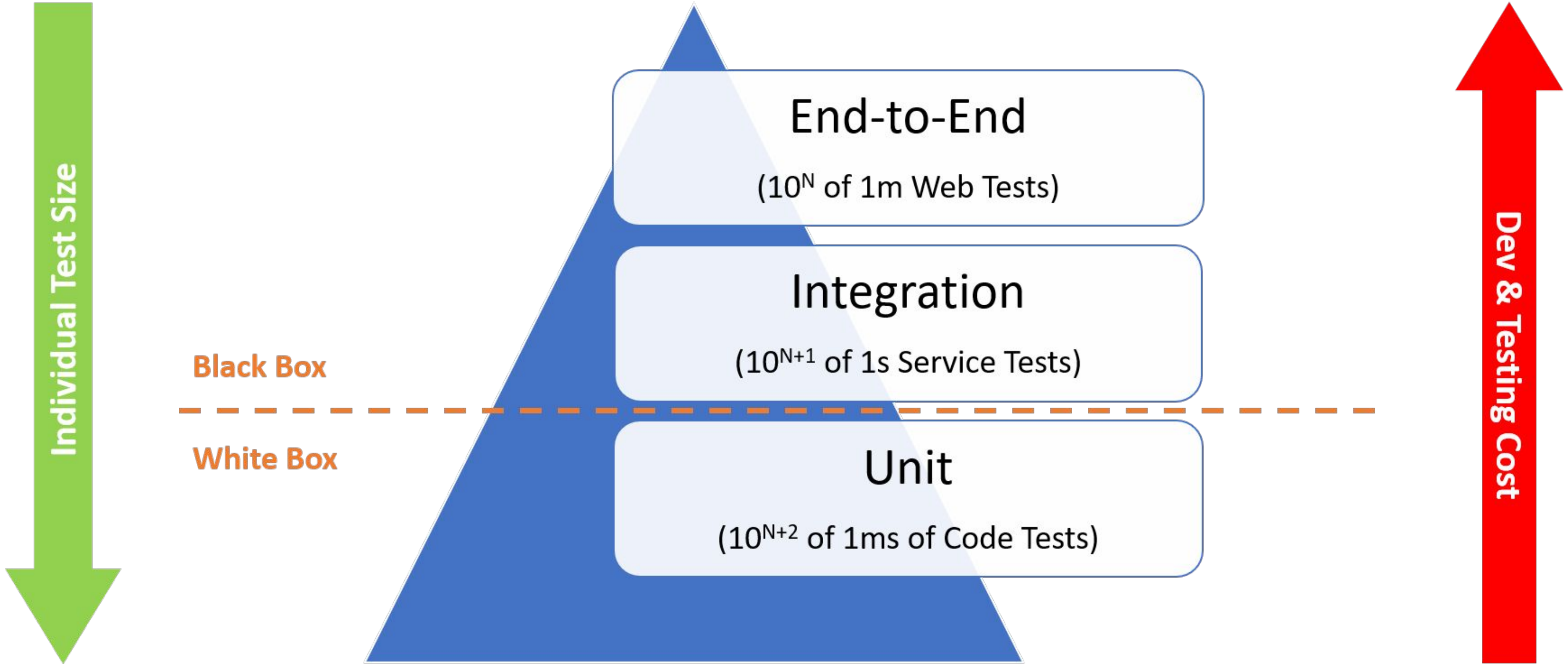
Code Testing

- Covers *code*
- White box - has direct access to source code
- Includes unit testing and subcutaneous testing
- Verifies that individual “units” of code work correctly

Feature Testing

- Covers *features*
- Black box - does not have direct access to source code
- Includes integration and end-to-end testing
- Verifies that live product features work correctly

The Testing Pyramid

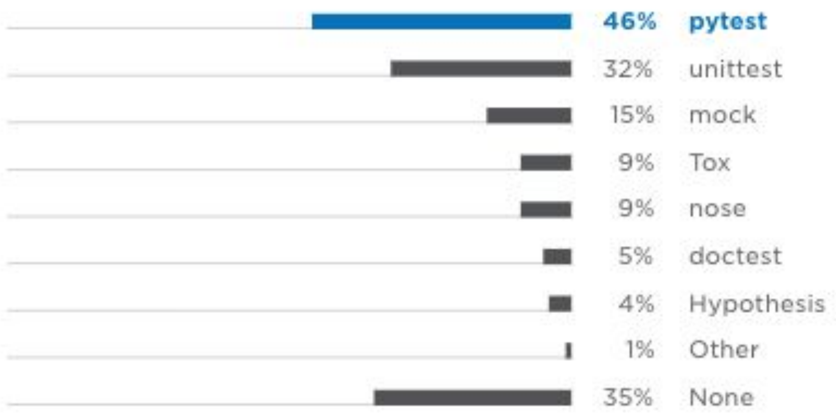


Major misconception:

`unit testing ==` *`all`* `testing`

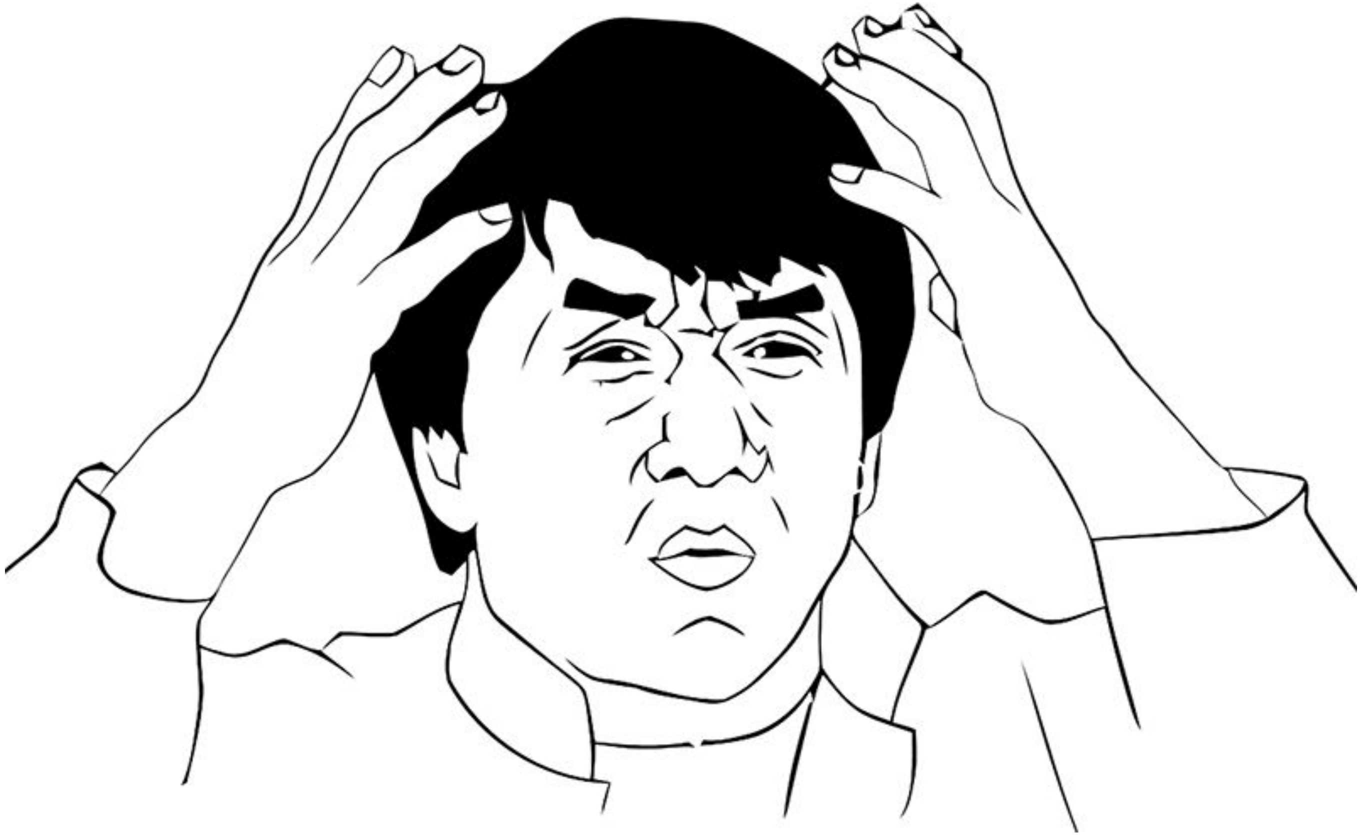
Python Developers Survey 2018:

Testing Frameworks



The leading unit-testing framework is pytest followed by unittest. The other unit testing frameworks are far less popular. It's quite surprising that 35% of Python users don't use any testing frameworks and are presumably not testing their code. In the "Tools to create isolated Python environments" section we identified that around 1 in 5 Python users don't use Python isolation which is another best practice.

Source: https://www.jetbrains.com/research/python-developers-survey-2018/



What is Web UI Testing?

Web UI testing is black box testing of a Web app through a browser.

- It is **feature testing** because it tests the app like a user.
- It is **end-to-end** because all parts are exercised together.

Modern Web apps can have many parts:

- Web UI front-end that displays in a browser (HTML, CSS, JavaScript)
- A service layer (like REST APIs)
- A persistence layer (like databases)
- Web servers and load balancers (like NGINX)
- Queues and workers (for heavy jobs)

Web UI Testing Pros and Cons

Pros

- End-to-end coverage
- Test like a user
- Visible results
- Catch obvious problems

Cons

- Complex to automate
- Slow to execute
- Prone to flakiness
- Root cause analysis is harder

What Makes a “Good” Web UI Test?

- It focuses on one main behavior
- It has a clear, step-by-step procedure
- It covers an important, core feature
- It sticks to a “happy” path or a basic error case
- It avoids redundant, pointless, or unimportant variations
- It cannot be covered by a lower-level test (unit, integration, API)

If the test fails, will people panic?
And will they know what broke?

Since Web UI testing is expensive, focus on **ROI**.

Solution Sketch

Test automation is a special domain of software development.

Language	**Python**
Core Framework	**pytest**
UI Interactions	**Page Object Pattern**
Browser Automation	**Selenium WebDriver**

Solution Diagram

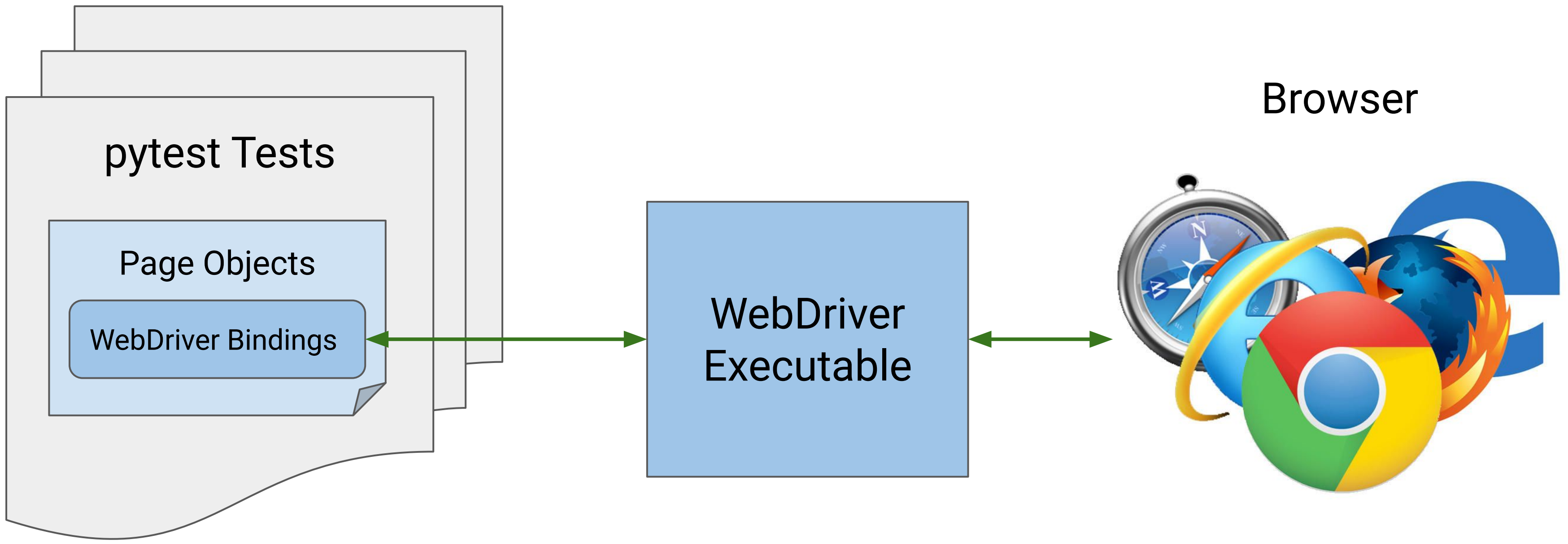


Image Source:
https://www.zdnet.com/article/which-browser-is-most-popular-on-each-major-operating-system/

Why Not Use Django Testing Tools?

Django provides an *excellent* testing client with a temporary database.

However, the Django test client has limitations:

1. It cannot do *feature* testing - it can only do *code* testing.
2. It cannot test apps in a real browser.
3. It can be used only with Django, not with other types of Web apps.

Our solution can do feature testing in real browsers against any Web app!

Why Not Use Codeless Tools?

“Codeless” test automation tools enable users to automate tests without programming. They typically offer forms for steps and locators or record-and-playback scripting. Many include AI for predicting or fixing failures.

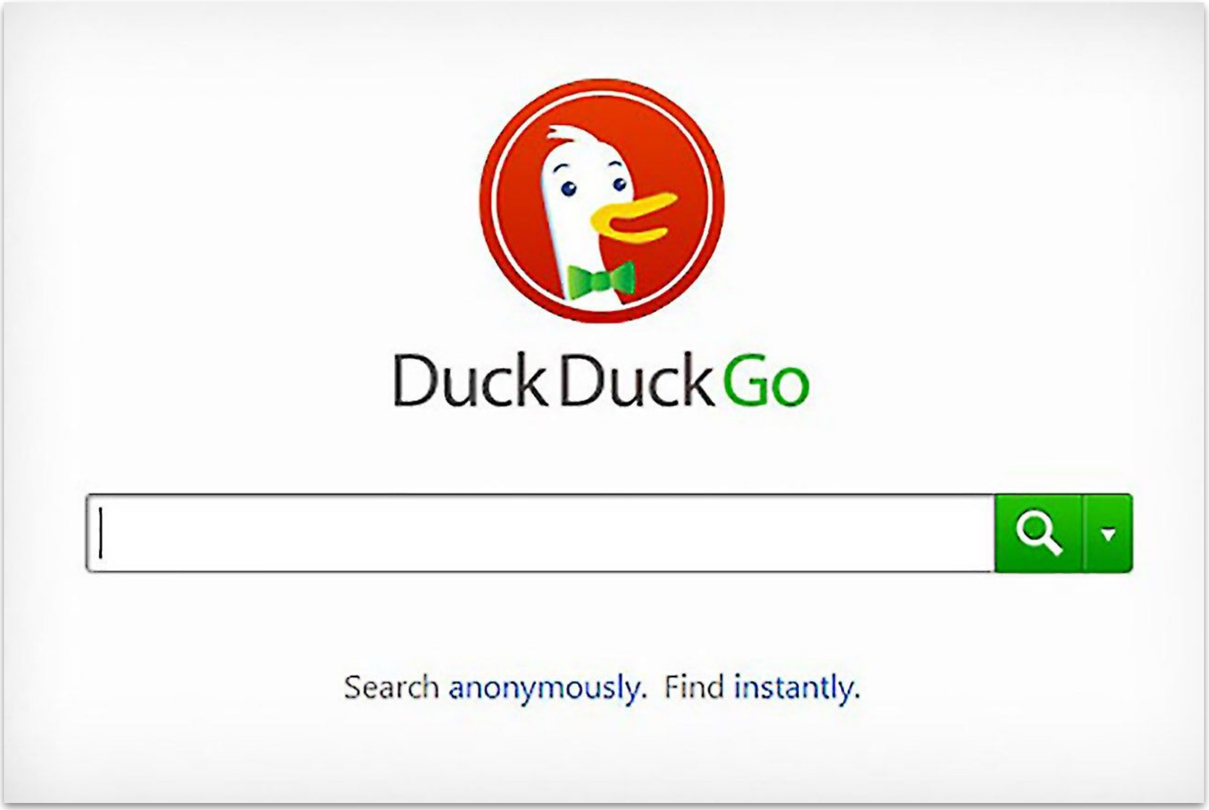
Codeless tools are great for testers who can’t code. However:

- The tools can feel slow and clunky.
- The tests are not very customizable.
- Licenses typically cost a lot of money.
- Vendor lock-in happens.

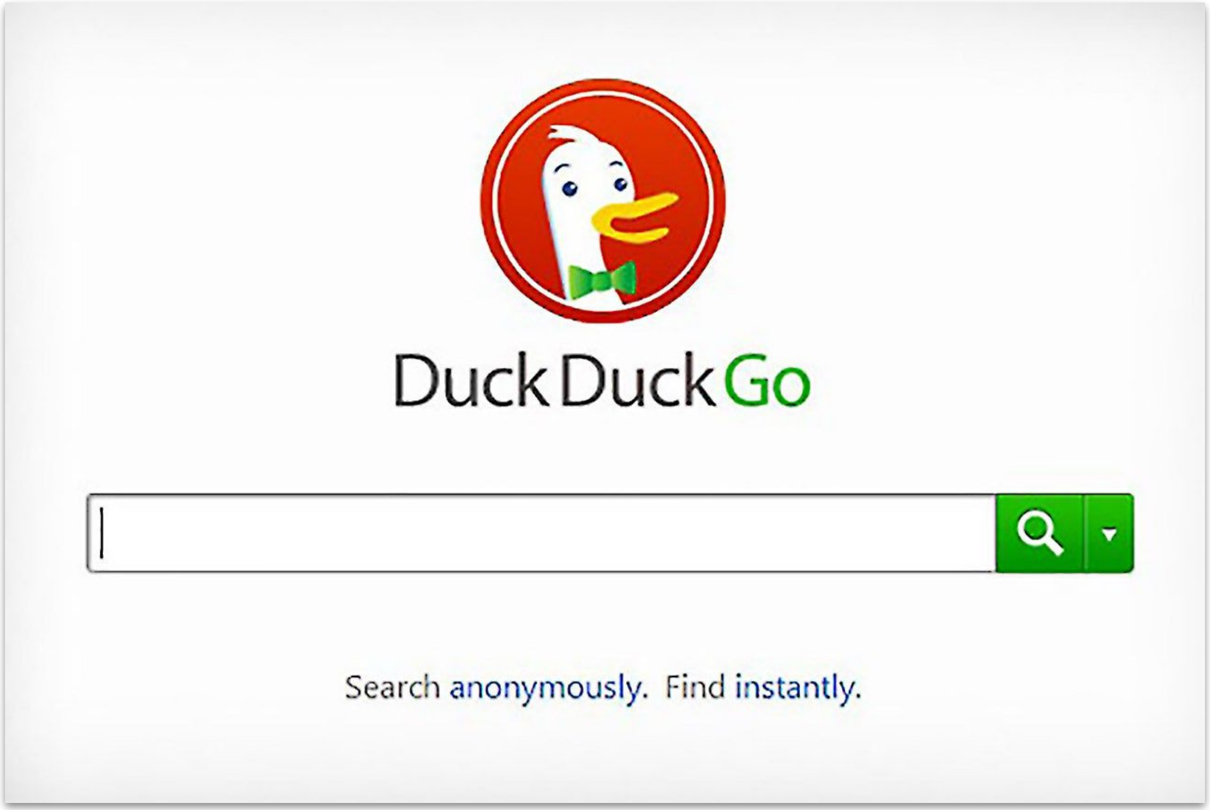
Coded tools (like our solution) are a better alternative for those who can code!

Writing Our First Test

Our Web App to Test



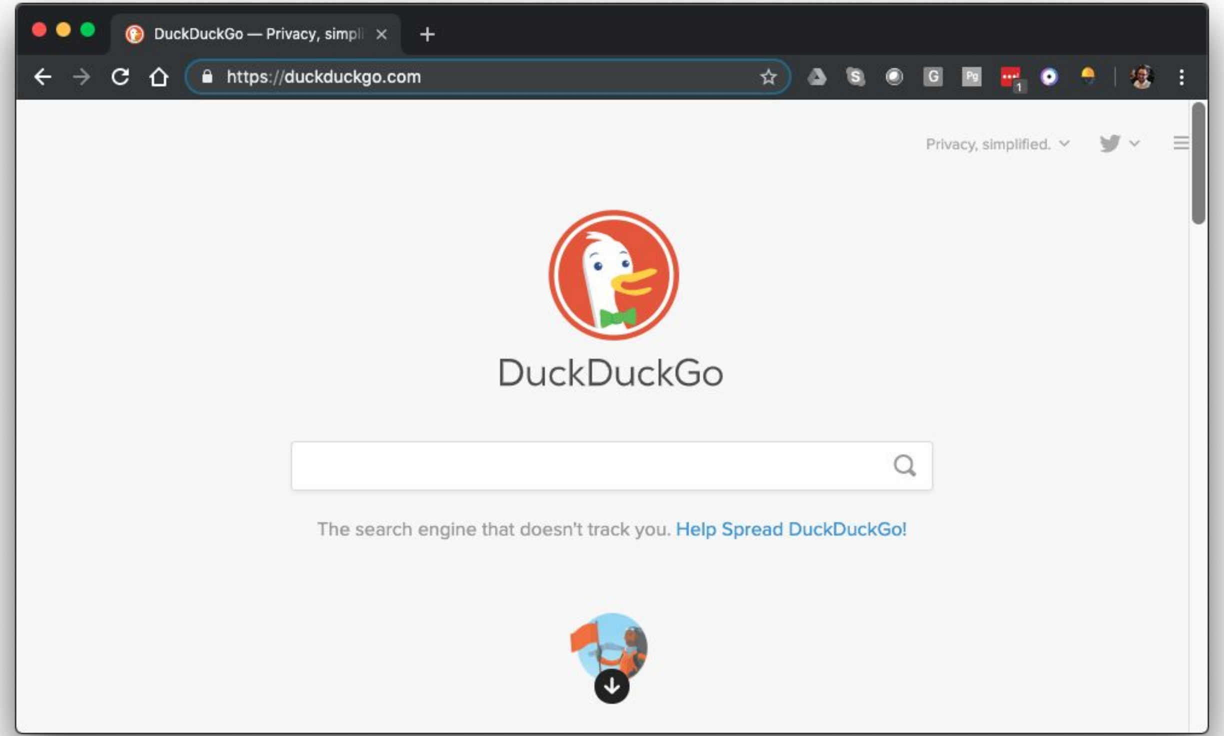
Everyone Do a Search!



Rule #1:
Write test steps
before test code.

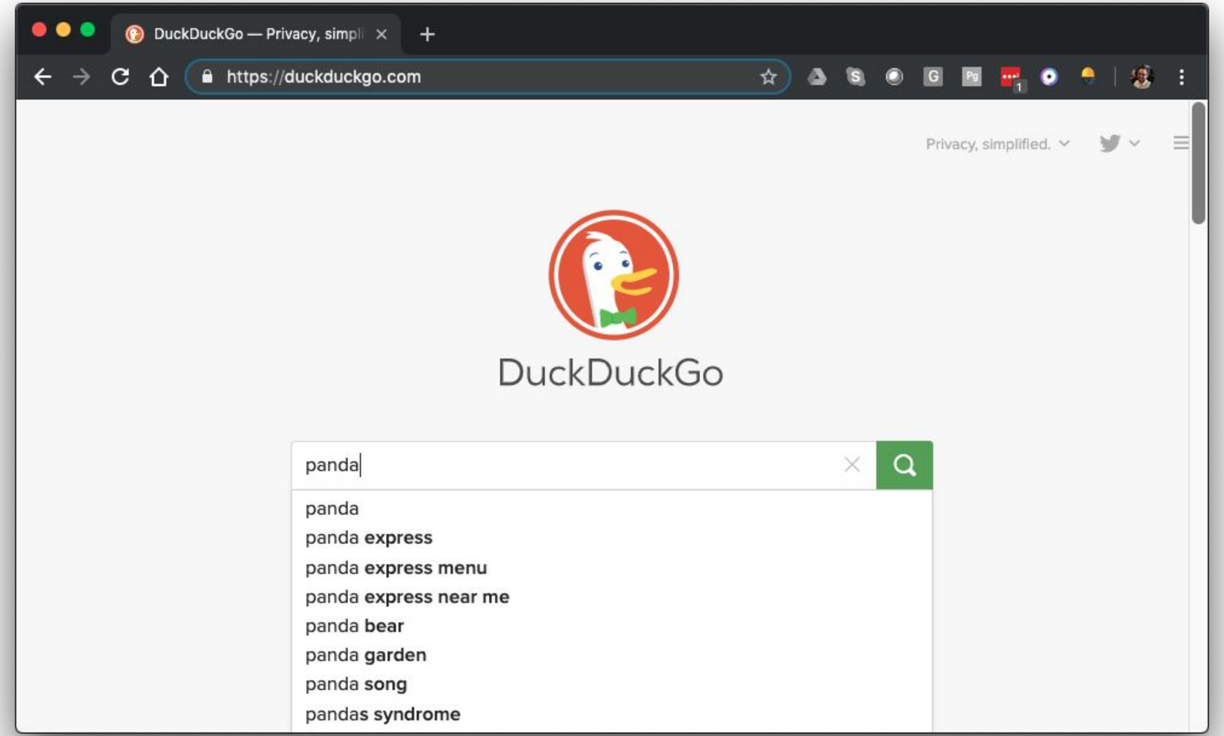
Let’s write a basic search test together!

Step 1: Navigate to DuckDuckGo



The search engine that doesn't track you. Help Spread DuckDuckGo!

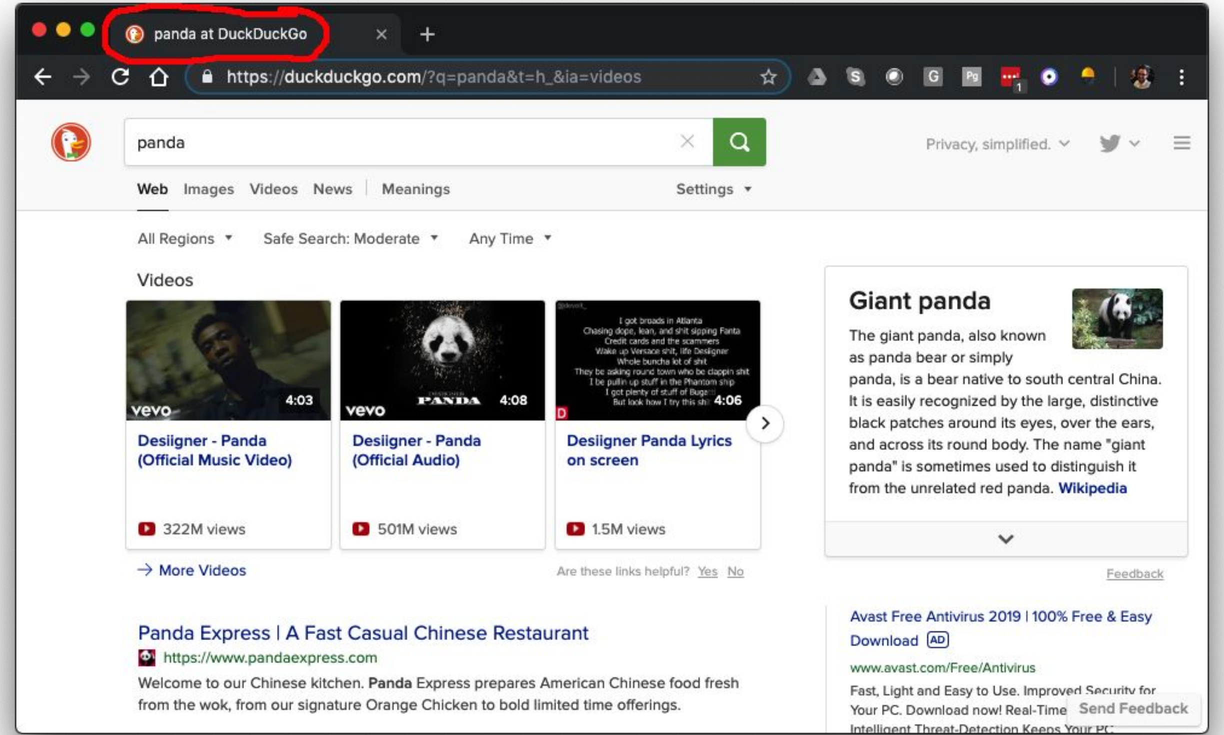
Step 2: Enter a search phrase



panda

panda
panda express
panda express menu
panda express near me
panda bear
panda garden
panda song
pandas syndrome

Step 3: Verify query in title

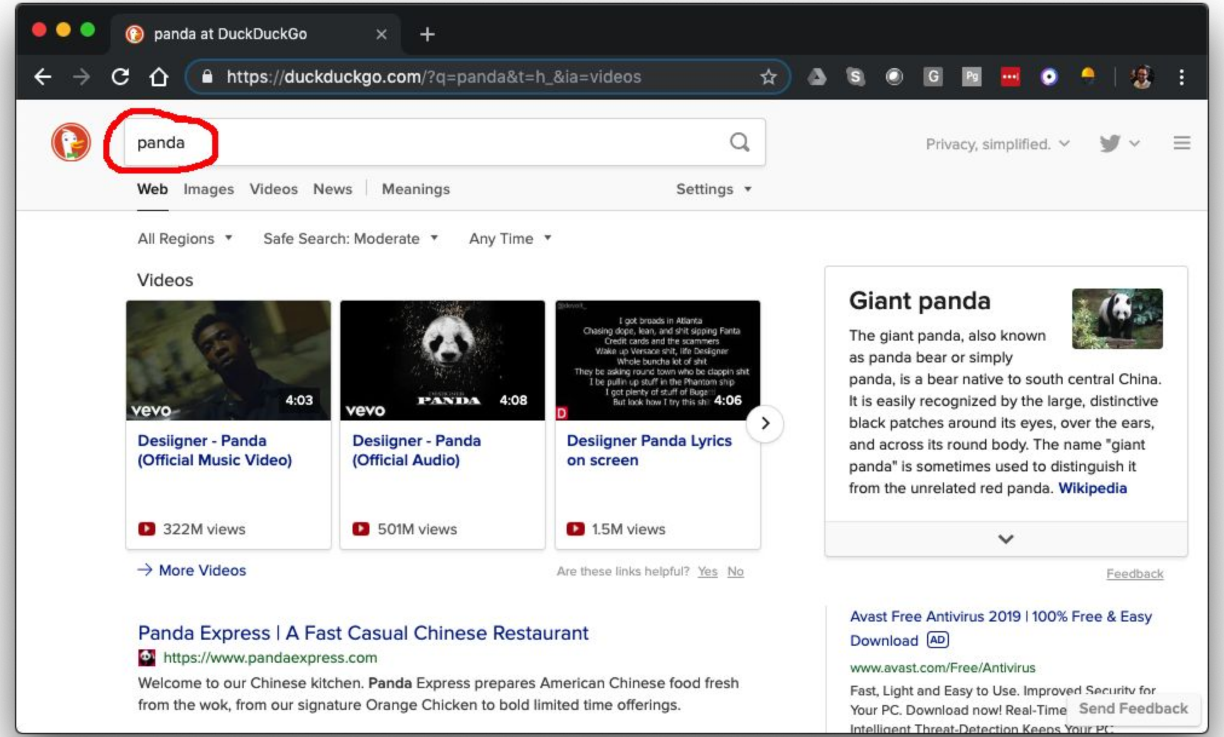


The giant panda, also known as panda bear or simply panda, is a bear native to south central China. It is easily recognized by the large, distinctive black patches around its eyes, over the ears, and across its round body. The name "giant panda" is sometimes used to distinguish it from the unrelated red panda. Wikipedia

Panda Express | A Fast Casual Chinese Restaurant
https://www.pandaexpress.com
Welcome to our Chinese kitchen. **Panda** Express prepares American Chinese food fresh from the wok, from our signature Orange Chicken to bold limited time offerings.

Avast Free Antivirus 2019 | 100% Free & Easy Download AD
www.avast.com/Free/Antivirus
Fast, Light and Easy to Use. Improved Security for Your PC. Download now! Real-Time Intelligent Threat-Detection Keeps Your PC

Step 4: Verify query on results page



Desiigner - Panda (Official Music Video)
322M views

Desiigner - Panda (Official Audio)
501M views

Desiigner Panda Lyrics on screen
1.5M views

More Videos

Are these links helpful? Yes No

Panda Express | A Fast Casual Chinese Restaurant
https://www.pandaexpress.com
Welcome to our Chinese kitchen. **Panda** Express prepares American Chinese food fresh from the wok, from our signature Orange Chicken to bold limited time offerings.

Giant panda

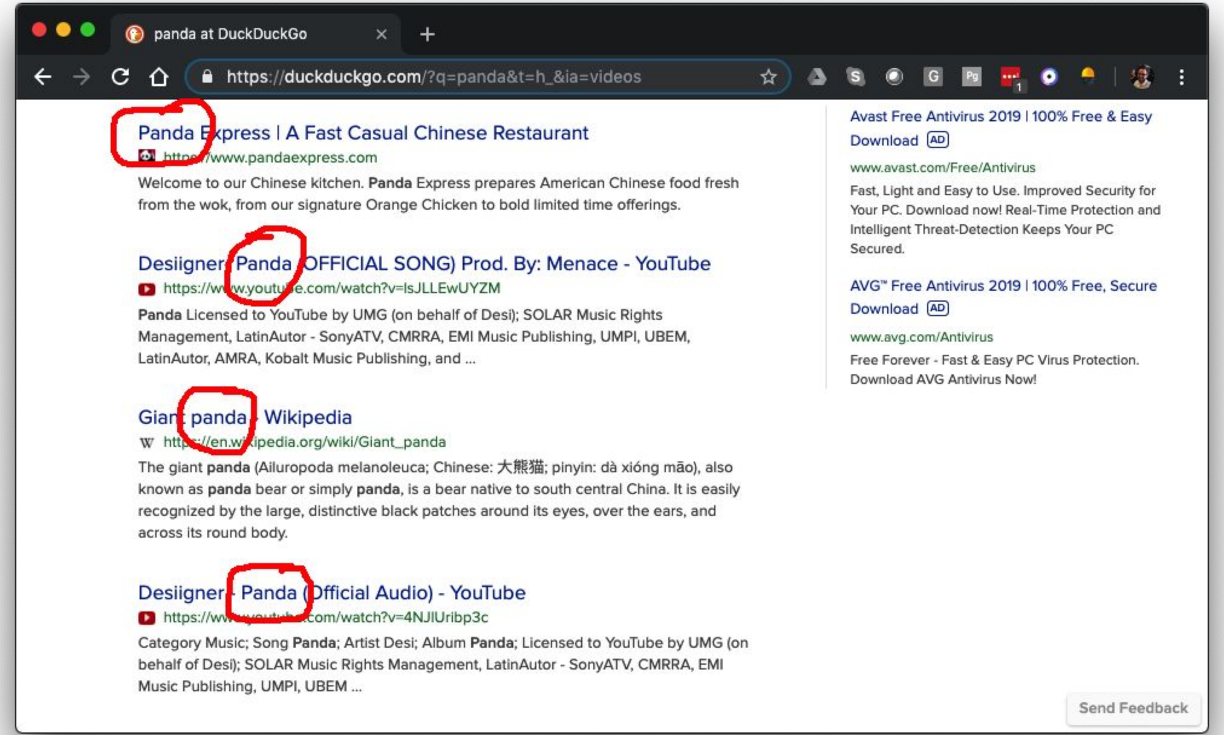
The giant panda, also known as panda bear or simply panda, is a bear native to south central China. It is easily recognized by the large, distinctive black patches around its eyes, over the ears, and across its round body. The name "giant panda" is sometimes used to distinguish it from the unrelated red panda. Wikipedia

Feedback

Avast Free Antivirus 2019 | 100% Free & Easy Download AD
www.avast.com/Free/Antivirus
Fast, Light and Easy to Use. Improved Security for Your PC. Download now! Real-Time Intelligent Threat-Detection Keeps Your PC

Send Feedback

Step 5: Verify results match query



Panda Express | A Fast Casual Chinese Restaurant
https://www.pandaexpress.com
Welcome to our Chinese kitchen. **Panda** Express prepares American Chinese food fresh from the wok, from our signature Orange Chicken to bold limited time offerings.

Desiigner - Panda (OFFICIAL SONG) Prod. By: Menace - YouTube
https://www.youtube.com/watch?v=lsJLLEwUYZM
Panda Licensed to YouTube by UMG (on behalf of Desi); SOLAR Music Rights Management, LatinAutor - SonyATV, CMRRA, EMI Music Publishing, UMPI, UBEM, LatinAutor, AMRA, Kobalt Music Publishing, and ...

Giant panda - Wikipedia
https://en.wikipedia.org/wiki/Giant_panda
The giant **panda** (Ailuropoda melanoleuca; Chinese: 大熊猫; pinyin: dà xióng māo), also known as **panda** bear or simply **panda**, is a bear native to south central China. It is easily recognized by the large, distinctive black patches around its eyes, over the ears, and across its round body.

Desiigner - Panda (Official Audio) - YouTube
https://www.youtube.com/watch?v=4NJlUribp3c
Category Music; Song **Panda**; Artist Desi; Album **Panda**; Licensed to YouTube by UMG (on behalf of Desi); SOLAR Music Rights Management, LatinAutor - SonyATV, CMRRA, EMI Music Publishing, UMPI, UBEM ...

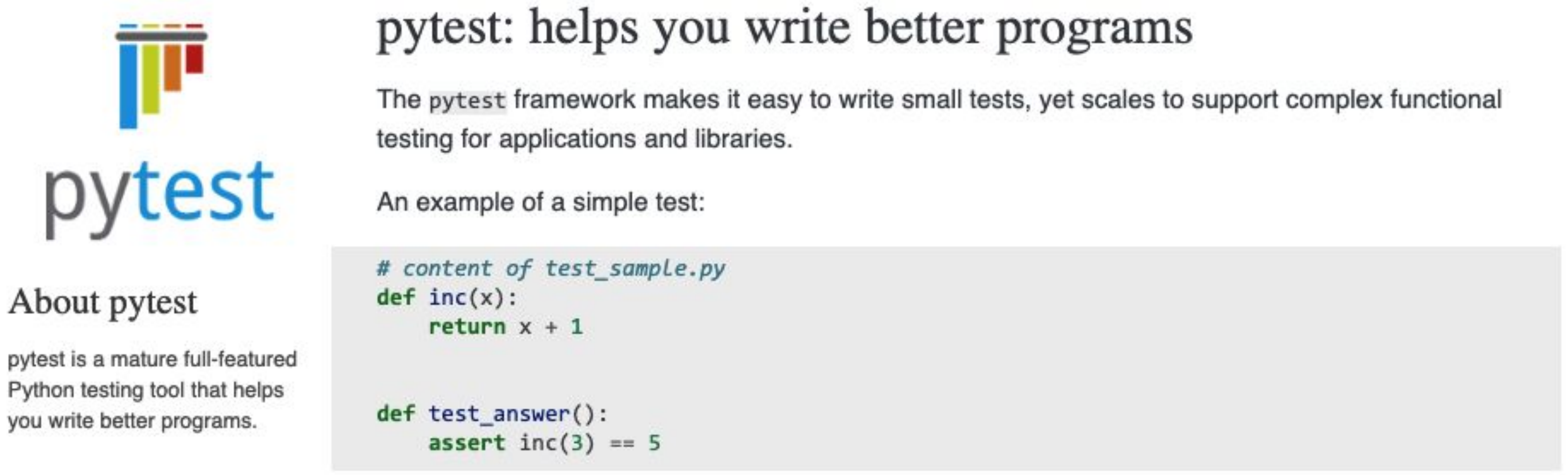
Avast Free Antivirus 2019 | 100% Free & Easy Download AD
www.avast.com/Free/Antivirus
Fast, Light and Easy to Use. Improved Security for Your PC. Download now! Real-Time Protection and Intelligent Threat-Detection Keeps Your PC Secured.

AVG™ Free Antivirus 2019 | 100% Free, Secure Download AD
www.avg.com/Antivirus
Free Forever - Fast & Easy PC Virus Protection. Download AVG Antivirus Now!

Our First Test Case

```
Scenario: Basic DuckDuckGo Search

    Given the DuckDuckGo home page is displayed

    When the user searches for “panda”

    Then the search result title contains “panda”

    And the search result query is “panda”

    And the search result links pertain to “panda”
```

Let’s put this test into **pytest**.



pytest

About pytest

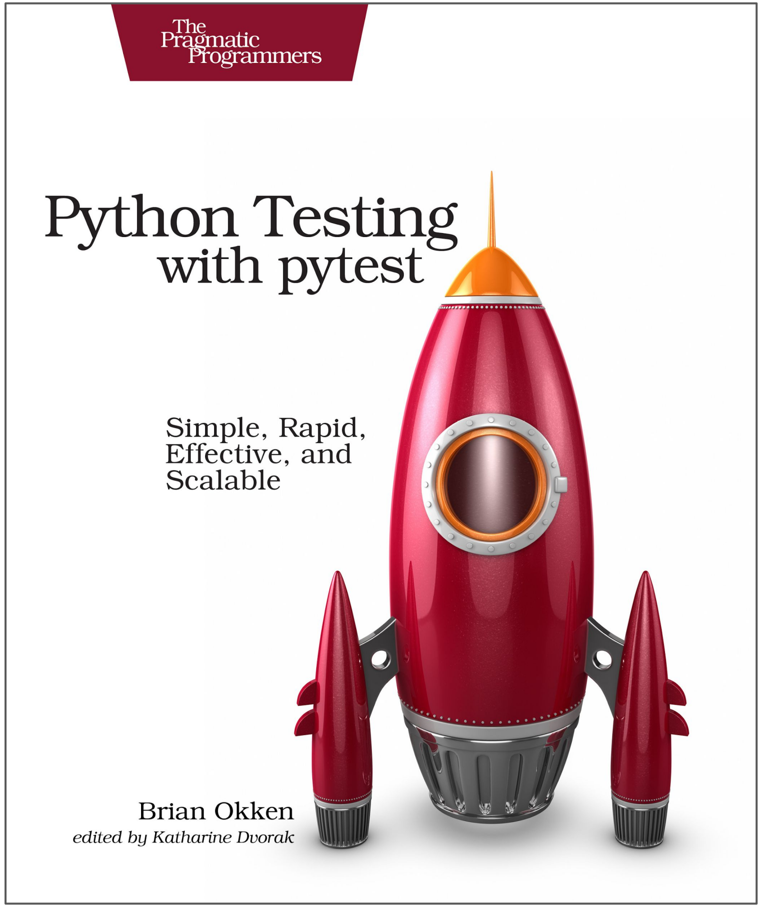
pytest is a mature full-featured Python testing tool that helps you write better programs.

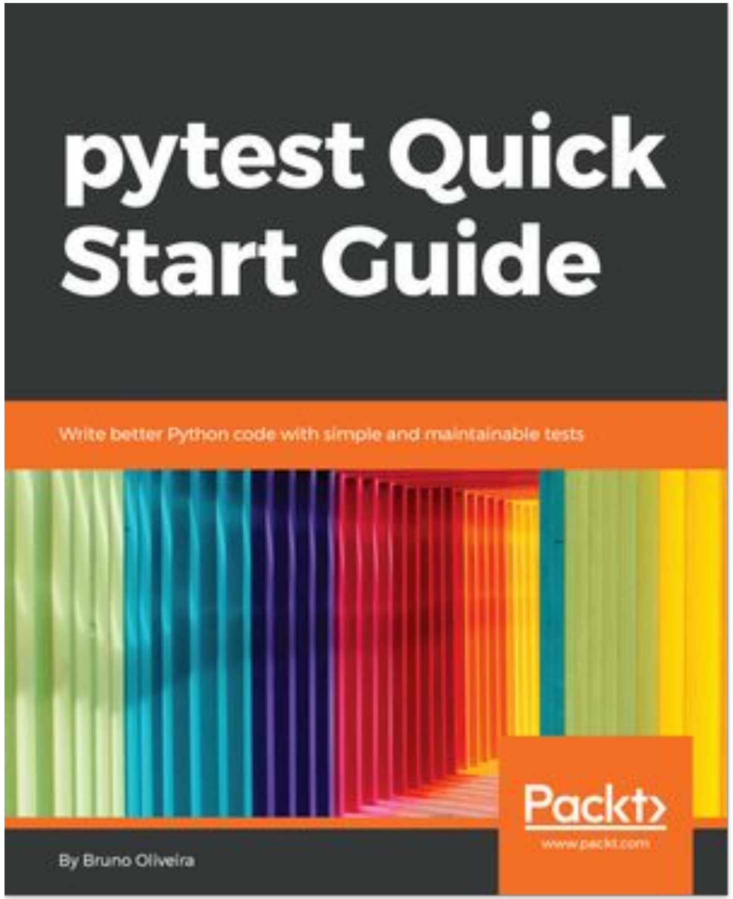
pytest: helps you write better programs

The `pytest` framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

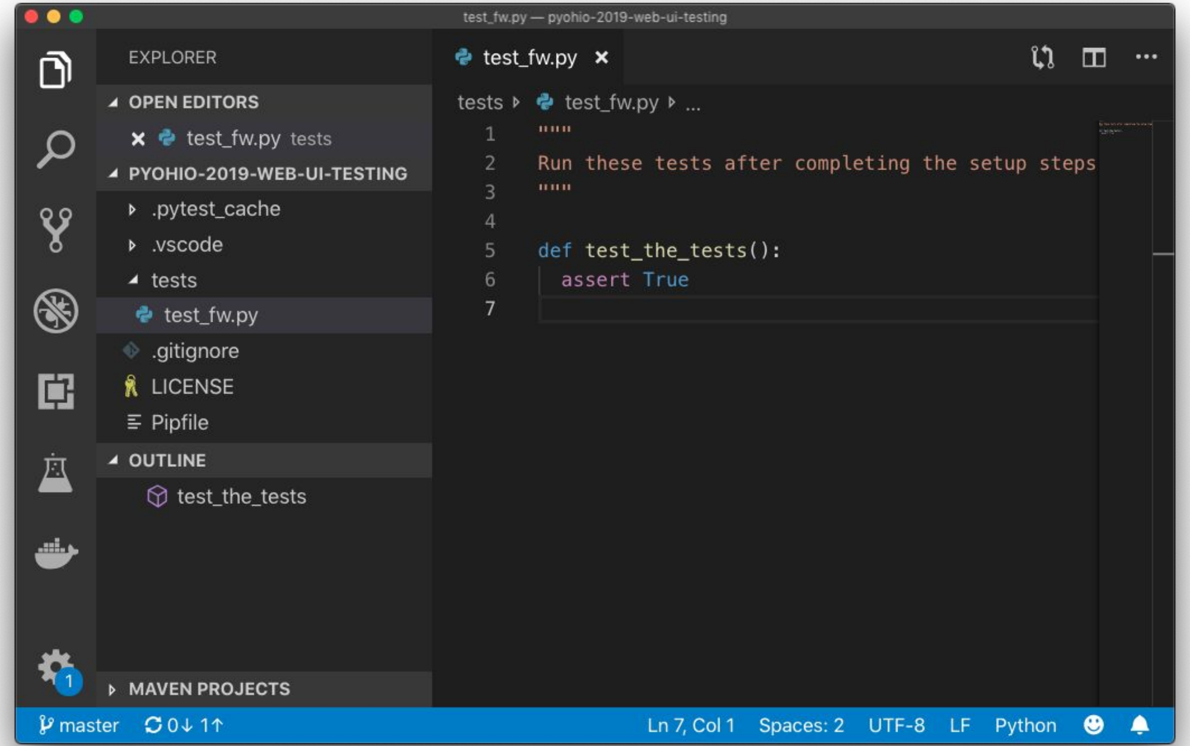
An example of a simple test:

```python
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```


The Pragmatic Programmers
Python Testing with pytest
Simple, Rapid, Effective, and Scalable
Brian Okken
edited by Katharine Dvorak

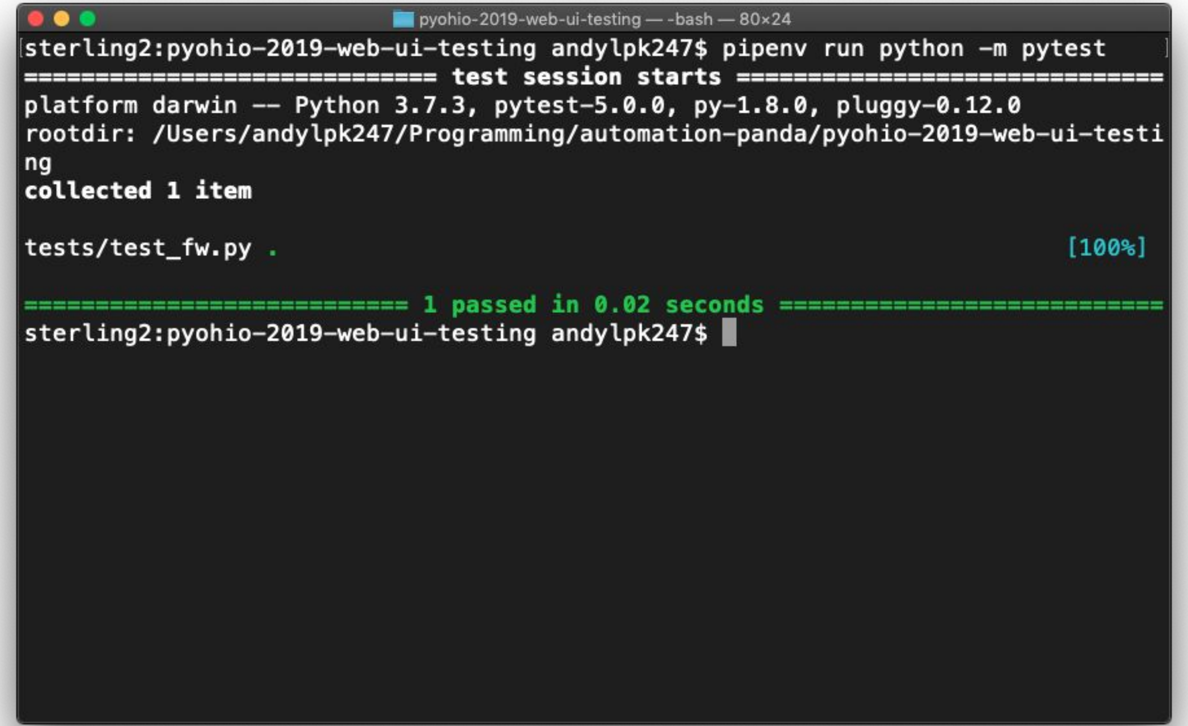

pytest Quick Start Guide
Write better Python code with simple and maintainable tests
By Bruno Oliveira
Packt>
www.packt.com

pytest in Our Project



```python
"""
Run these tests after completing the setup steps
"""

def test_the_tests():
  assert True

```

Running pytest Tests



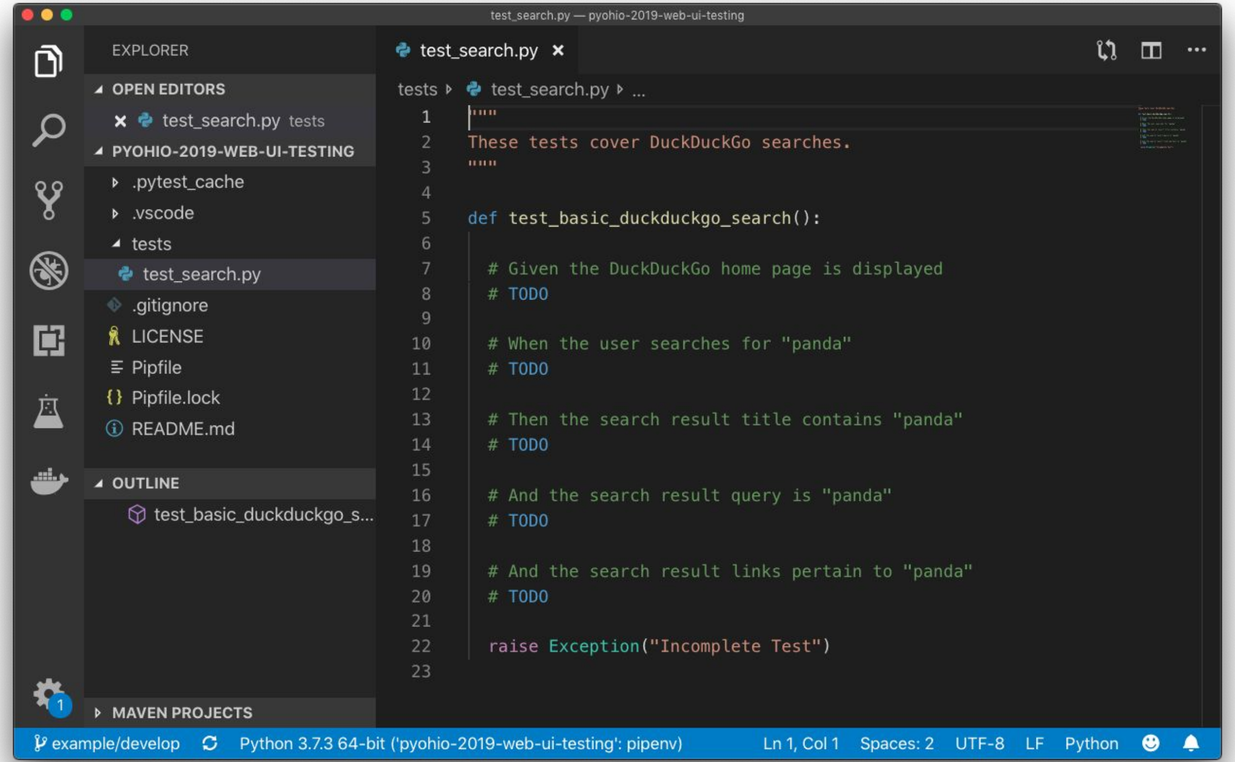
```
sterling2:pyohio-2019-web-ui-testing andylpk247$ pipenv run python -m pytest
============================= test session starts ==============================
platform darwin -- Python 3.7.3, pytest-5.0.0, py-1.8.0, pluggy-0.12.0
rootdir: /Users/andylpk247/Programming/automation-panda/pyohio-2019-web-ui-testi
ng
collected 1 item

tests/test_fw.py .                                                       [100%]

=========================== 1 passed in 0.02 seconds ===========================
sterling2:pyohio-2019-web-ui-testing andylpk247$
```

Hands-On Time!

Finish the setup steps for the tutorial project.
Then, complete **Tutorial Instructions Part 1** in the README.
Take *4 minutes*.

https://github.com/AndyLPK247/pyohio-2019-web-ui-testing
https://bit.ly/2XkgN7w

Our First Test in Comments



test_search.py — pyohio-2019-web-ui-testing

```python
"""
These tests cover DuckDuckGo searches.
"""

def test_basic_duckduckgo_search():

  # Given the DuckDuckGo home page is displayed
  # TODO

  # When the user searches for "panda"
  # TODO

  # Then the search result title contains "panda"
  # TODO

  # And the search result query is "panda"
  # TODO

  # And the search result links pertain to "panda"
  # TODO

  raise Exception("Incomplete Test")

```

Defining Page Objects

What is a Page Object?

A **page object** is an object representing a Web page or component.

- It has *locators* for finding elements on the page.
- It has *interaction methods* that interact with the page under test.

Each Web page or component under test should have a page object class.

- Page objects encapsulate low-level Selenium WebDriver calls.
- That way, tests can make short, readable calls instead of complicated ones.

Our Pages Under Test

DuckDuckGo Search Page

- Load the page
- Search a phrase

DuckDuckGo Result Page

- Get the result count
- Get the search query
- Get the title

Page Object Class Stubs

```python
class DuckDuckGoSearchPage:

  def load(self):
    pass

  def search(self, phrase):
    pass
```

```python
class DuckDuckGoResultPage:

  def result_count_for_phrase(
    self, phrase):
    return 0

  def search_input_value(self):
    return ""

  def title(self):
    return ""
```

Add Page Object Calls to the Test

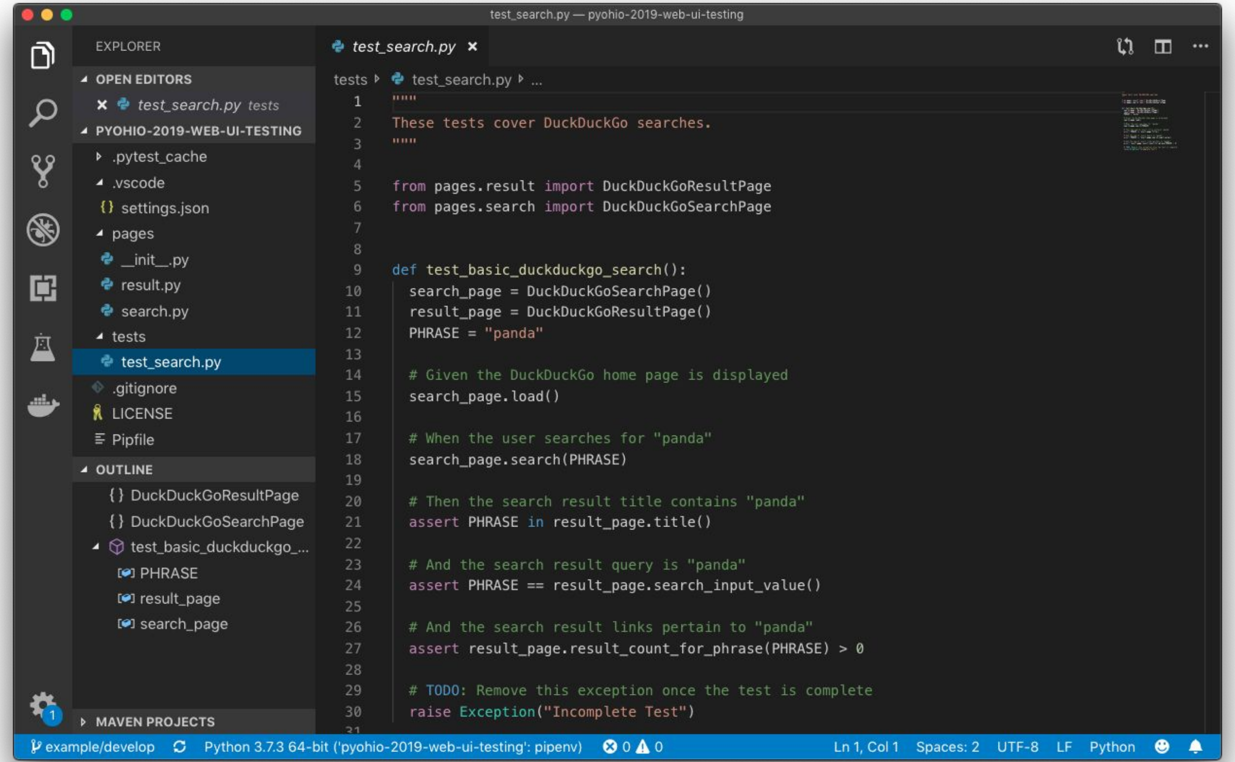
```python
def test_basic_duckduckgo_search():

  # Given the DuckDuckGo home page is displayed
  search_page = DuckDuckGoSearchPage()
  search_page.load()

  # When the user searches for "panda"
  search_page.search("panda")
```

Hands-On Time!

Complete **Tutorial Instructions Part 2** in the README.
Take *8 minutes*.

https://github.com/AndyLPK247/pyohio-2019-web-ui-testing
https://bit.ly/2XkgN7w

Our First Test with Page Objects



```python
"""
These tests cover DuckDuckGo searches.
"""

from pages.result import DuckDuckGoResultPage
from pages.search import DuckDuckGoSearchPage


def test_basic_duckduckgo_search():
  search_page = DuckDuckGoSearchPage()
  result_page = DuckDuckGoResultPage()
  PHRASE = "panda"

  # Given the DuckDuckGo home page is displayed
  search_page.load()

  # When the user searches for "panda"
  search_page.search(PHRASE)

  # Then the search result title contains "panda"
  assert PHRASE in result_page.title()

  # And the search result query is "panda"
  assert PHRASE == result_page.search_input_value()

  # And the search result links pertain to "panda"
  assert result_page.result_count_for_phrase(PHRASE) > 0

  # TODO: Remove this exception once the test is complete
  raise Exception("Incomplete Test")
```

Setting Up Selenium WebDriver

Selenium WebDriver

The **`selenium`** package is the Selenium WebDriver implementation for Python.

It sends Web UI commands from test automation code to a browser.

WebDriver can handle *every* type of Web UI interaction.

The best practice is to make all WebDriver calls from page object methods.

```
Full API Documentation:
https://selenium-python.readthedocs.io/api.html
```

```
pipenv install selenium
```

WebDriver Instances

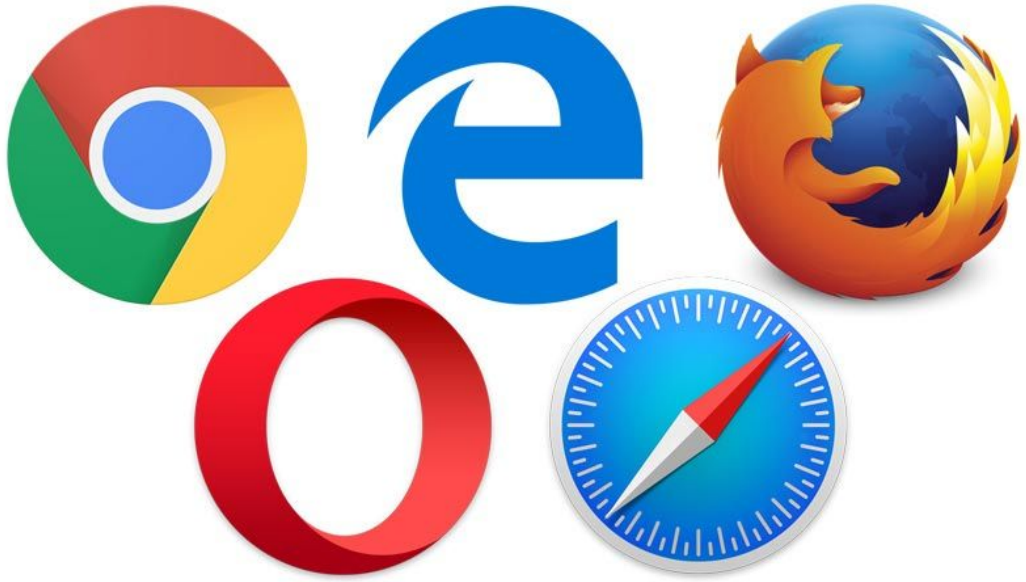
Every test case should have its own WebDriver instance.

- One test → one WebDriver → one browser
- Test case independence

WebDriver initialization and quitting should be handled with a pytest fixture.

- Any test can use a fixture for setup and cleanup
- Always *quit* the WebDriver (not *close*)
- Otherwise, drivers and browsers can become zombie processes!

Which Browser Type?



Source: https://www.color-management-guide.com/images/icc-profile-internet/intro-web-browsers.jpg

WebDriver Fixture

```
import pytest
import selenium.webdriver

@pytest.fixture
def browser():
  b = selenium.webdriver.Chrome()
  b.implicitly_wait(10)
  yield b
  b.quit()
```

Using the Fixture

```
def test_basic_duckduckgo_search(browser):

  search_page = DuckDuckGoSearchPage(browser)
  result_page = DuckDuckGoResultPage(browser)
```

Updating Page Objects

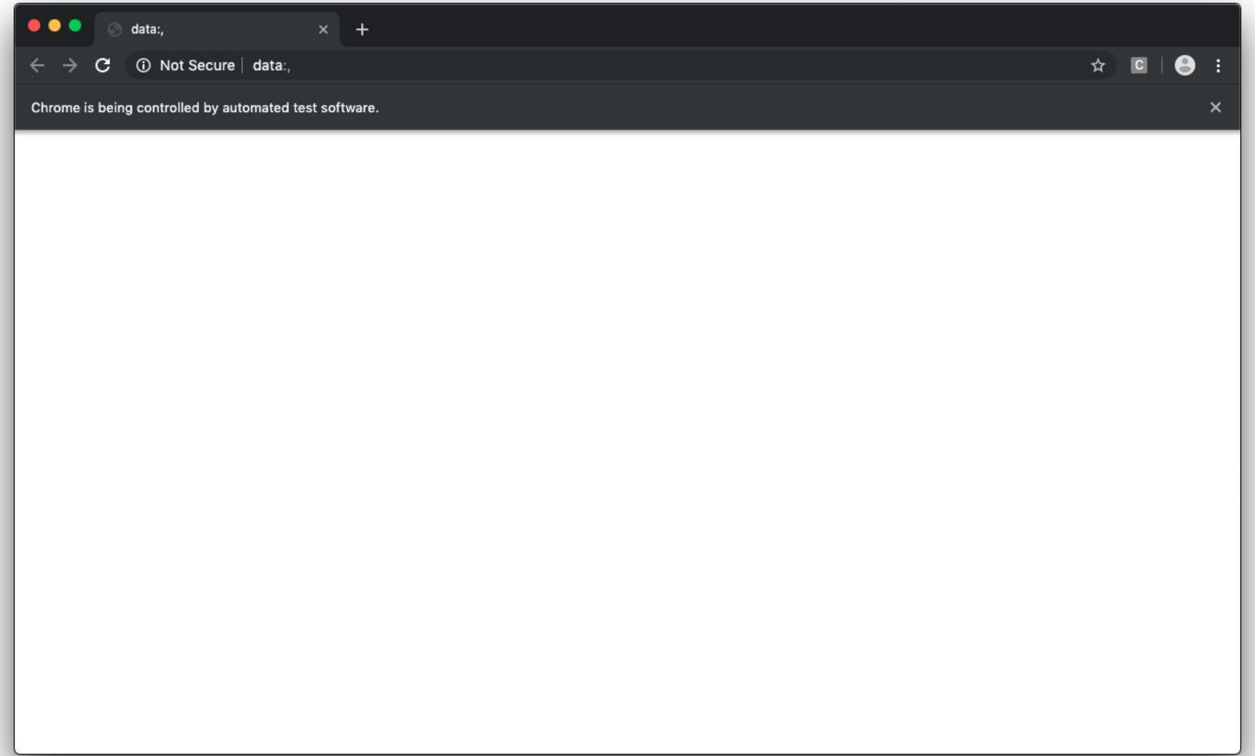
```python
class DuckDuckGoSearchPage:

  def __init__(self, browser):
    self.browser = browser
```

Hands-On Time!

Complete **Tutorial Instructions Part 3** in the README.
Take *8 minutes*.

https://github.com/AndyLPK247/pyohio-2019-web-ui-testing
https://bit.ly/2XkgN7w

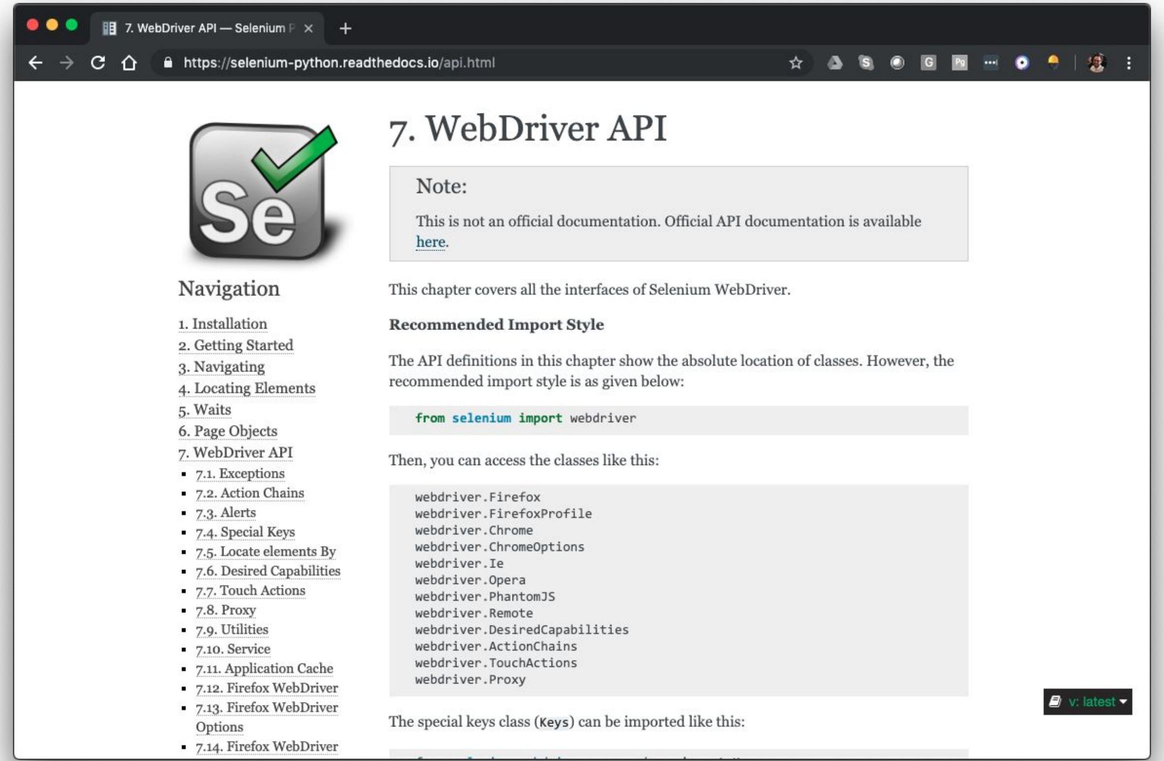
WebDriver-Controlled Chrome



Chrome is being controlled by automated test software.

Making WebDriver Calls

The Docs



7. WebDriver API — Selenium P

https://selenium-python.readthedocs.io/api.html

7. WebDriver API

Note:

This is not an official documentation. Official API documentation is available here.

This chapter covers all the interfaces of Selenium WebDriver.

Recommended Import Style

The API definitions in this chapter show the absolute location of classes. However, the recommended import style is as given below:

```
from selenium import webdriver
```

Then, you can access the classes like this:

```
webdriver.Firefox
webdriver.FirefoxProfile
webdriver.Chrome
webdriver.ChromeOptions
webdriver.Ie
webdriver.Opera
webdriver.PhantomJS
webdriver.Remote
webdriver.DesiredCapabilities
webdriver.ActionChains
webdriver.TouchActions
webdriver.Proxy
```

The special keys class (`Keys`) can be imported like this:

v: latest

Some calls are simple.

Navigating to a Page

```
class DuckDuckGoSearchPage:

  URL = 'https://www.duckduckgo.com'

  def load(self):
    self.browser.get(self.URL)
```

Getting a Page’s Title

```
class DuckDuckGoResultPage:

  def title(self):
    return self.browser.title
```

Many calls interact with **elements**.

Entering a Search Phrase

```python
class DuckDuckGoSearchPage:

  # The "locator" is a query for finding an element
  SEARCH_INPUT = (By.NAME, 'q')

  def search(self, phrase):

    # The element must be found using the locator
    search_input = self.browser.find_element(*self.SEARCH_INPUT)

    # Interactions are set to the element object
    search_input.send_keys(phrase + Keys.RETURN)
```

Get the Input Field’s Value

```
class DuckDuckGoResultPage:

  SEARCH_INPUT = (By.NAME, 'q')

  def search_input_value(self):
    search_input = self.browser.find_element(*self.SEARCH_INPUT)
    return search_input.get_attribute('value')
```

Locators

Locators are queries that find elements on a page.

There are many types:

- By.ID
- By.NAME
- By.CLASS_NAME
- By.CSS_SELECTOR
- By.XPATH
- By.LINK_TEXT
- By.PARTIAL_LINK_TEXT
- By.TAG_NAME

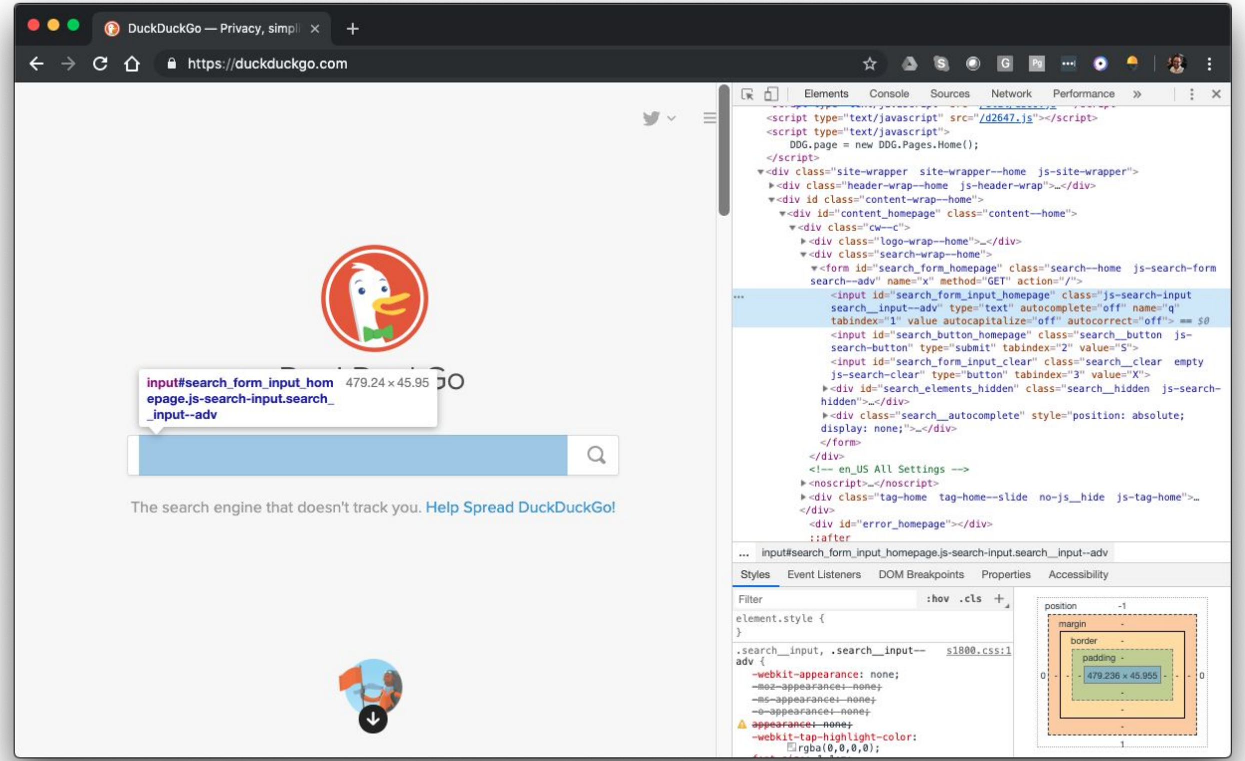
Want to learn more?
Take a free course online!

Test Automation University:
Web Element Locator Strategies

Finding Elements to Write Locators

Use Chrome DevTools!

Learn more from TAU!



```
<script type="text/javascript" src="/d2647.js"></script>
<script type="text/javascript">
  DDG.page = new DDG.Pages.Home();
</script>
▼<div class="site-wrapper  site-wrapper--home  js-site-wrapper">
 ▶<div class="header-wrap--home  js-header-wrap">…</div>
 ▼<div id class="content-wrap--home">
  ▼<div id="content_homepage" class="content--home">
   ▼<div class="cw--c">
    ▶<div class="logo-wrap--home">…</div>
    ▼<div class="search-wrap--home">
     ▼<form id="search_form_homepage" class="search--home  js-search-form
     search--adv" name="x" method="GET" action="/">
…      <input id="search_form_input_homepage" class="js-search-input
       search__input--adv" type="text" autocomplete="off" name="q"
       tabindex="1" value autocapitalize="off" autocorrect="off"> == $0
       <input id="search_button_homepage" class="search__button  js-
       search-button" type="submit" tabindex="2" value="S">
       <input id="search_form_input_clear" class="search__clear  empty
       js-search-clear" type="button" tabindex="3" value="X">
      ▶<div id="search_elements_hidden" class="search__hidden  js-search-
      hidden">…</div>
      ▶<div class="search__autocomplete" style="position: absolute;
      display: none;">…</div>
      </form>
    </div>
    <!-- en_US All Settings -->
   ▶<noscript>…</noscript>
   ▶<div class="tag-home  tag-home--slide  no-js__hide  js-tag-home">…
   </div>
    <div id="error_homepage"></div>
    ::after
```

```
…  input#search_form_input_homepage.js-search-input.search__input--adv
```

```
element.style {
}
.search__input, .search__input--          s1800.css:1
adv {
   -webkit-appearance: none;
   ~~-moz-appearance: none;~~
   ~~-ms-appearance: none;~~
   ~~-o-appearance: none;~~
⚠  ~~appearance: none;~~
   -webkit-tap-highlight-color:
        rgba(0,0,0,0);
```

Common WebDriver Calls

For WebDriver:

- `current_url`
- `find_element`
- `find_elements`
- `find_element_by_*`
- `get`
- `maximize_window`
- `quit`
- `refresh`
- `save_screenshot`
- `title`

For Elements:

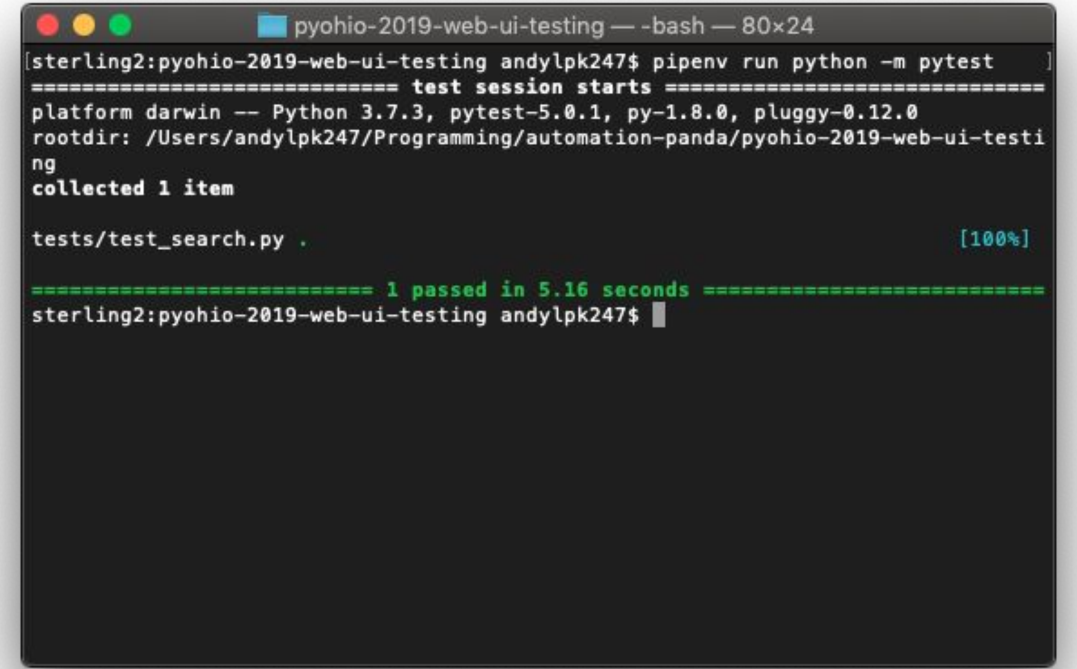
- `clear`
- `click`
- `find_element*`
- `get_attribute`
- `get_property`
- `is_displayed`
- `location`
- `send_keys`
- `size`
- `text`

Hands-On Time!

Complete **Tutorial Instructions Part 4** in the README.
Take *16 minutes*.

https://github.com/AndyLPK247/pyohio-2019-web-ui-testing
https://bit.ly/2XkgN7w

A Successful Test Run



pyohio-2019-web-ui-testing — -bash — 80×24

```
sterling2:pyohio-2019-web-ui-testing andylpk247$ pipenv run python -m pytest
============================= test session starts ==============================
platform darwin -- Python 3.7.3, pytest-5.0.1, py-1.8.0, pluggy-0.12.0
rootdir: /Users/andylpk247/Programming/automation-panda/pyohio-2019-web-ui-testi
ng
collected 1 item

tests/test_search.py .                                                   [100%]

=========================== 1 passed in 5.16 seconds ===========================
sterling2:pyohio-2019-web-ui-testing andylpk247$ 
```

Improving the Solution

Multiple Browsers

Web UI tests should run on *any* browser.

Browser choice should be an input.

Put inputs into a config file.

Read the config file in a fixture.

```python
@pytest.fixture
def browser():
  with open('tests/config.json') as config_file:
    config = json.load(config_file)

  if config['browser'] == 'Chrome':
    b = selenium.webdriver.Chrome()
  elif config['browser'] == 'Firefox':
    b = selenium.webdriver.Firefox()
  # ...
```

Parallel Execution

Web UI tests are *slow*.

Running tests in parallel can drastically reduce runtime.

pytest-xdist is a pytest plugin for parallel execution.

Selenium Grid provides a distributed environment for “remote” WebDrivers. It can also handle different browser, OS, and version combinations.

Explicit Waits

Implicitly waiting up to 10 seconds for every interaction may not be best.

Explicit waits can be applied to each interaction for precise times and conditions.

Most interactions need the target element to *exist* in the DOM.

Some interactions (like clicking) need the element to *appear* (exist + displayed).

Page object methods can put waits together with WebDriver calls.

Better Page Objects

Our page object classes were rudimentary.

A more sophisticated implementation could have:

- A super class for page objects
- Helper methods for common operations
- Logging

An even better evolution would be the **Screenplay Pattern**.

$1M Question:

Should it be a Web UI test?

Congrats!

You finished the tutorial.

Homework:

Do the *Independent Exercises*.

Resources

- Test Automation University
  - Web Element Locator Strategies
  - Behavior-Driven Development with pytest-bdd
  - Setting a Foundation for Successful Test Automation
- TestProject blog
  - Tutorial: Web Testing Made Easy with Python, Pytest and Selenium WebDriver
- Automation Panda blog
  - Testing page
  - Python page