Hands-On Web UI Testing

Andrew Knight
DjangoCon 2019

I'm Pandy.
I love testing.

@AutomationPanda

AutomationPanda.com

Quick Poll:

Developers?

Testers?

Other Roles?

Web UI testing can be hard.
Let’s make it easy.
We have **3.5 hours**.

Agenda

1. Learning About Web UI Testing
2. Writing Our First Test
3. Initializing Selenium WebDriver
4. Defining Page Objects
5. Finding Locators for Elements
6. Making WebDriver Calls
7. Configuring Multiple Browsers
8. Handling Race Conditions
9. Running Tests in Parallel
10. Adding More Tests

Test Project Setup

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

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

Requirements:

- Git
- Python 3.7 or higher
- Pipenv (“pip install pipenv”)
- Google Chrome and ChromeDriver
- Mozilla Firefox and geckodriver

Learning About Web UI Testing

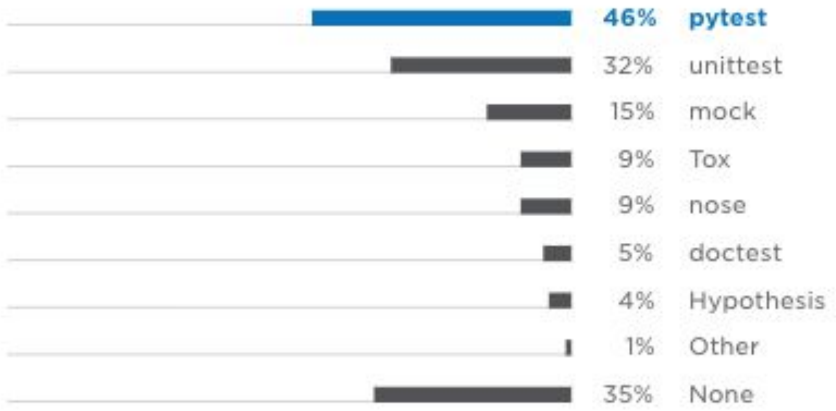
How would you define “testing”?

Major misconception:

unit testing == *all* testing

Python Developers Survey 2018:

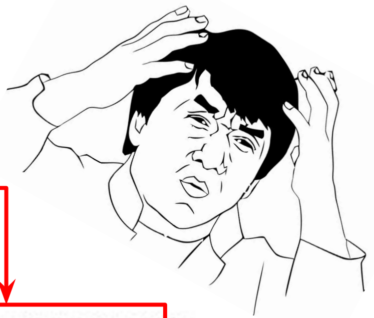
Testing Frameworks



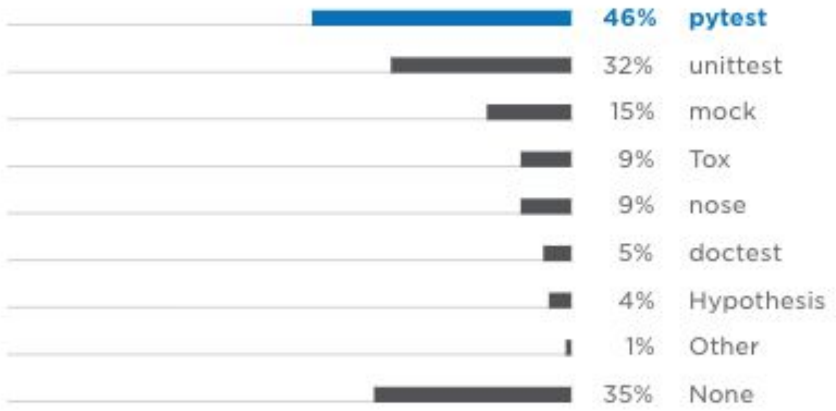
The leading unit-testing framework is pytest followed by unittest. The other unit testing frameworks are far less popular. It's quite surprising that 35% of Python users don't use any testing frameworks and are presumably not testing their code. In the "Tools to create isolated Python environments" section 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/

Python Developers Survey 2018:



Testing Frameworks



46%	pytest
32%	unittest
15%	mock
9%	Tox
9%	nose
5%	doctest
4%	Hypothesis
1%	Other
35%	None

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/

There is a difference between testing **code** and testing **features**.

There is a difference between testing **code** and testing **features**.

WHITE BOX

Is the code written to do expected things?

“Unit” or “Subcutaneous”

There is a difference between testing **code** and testing **features**.

WHITE BOX

Is the code written to do expected things?

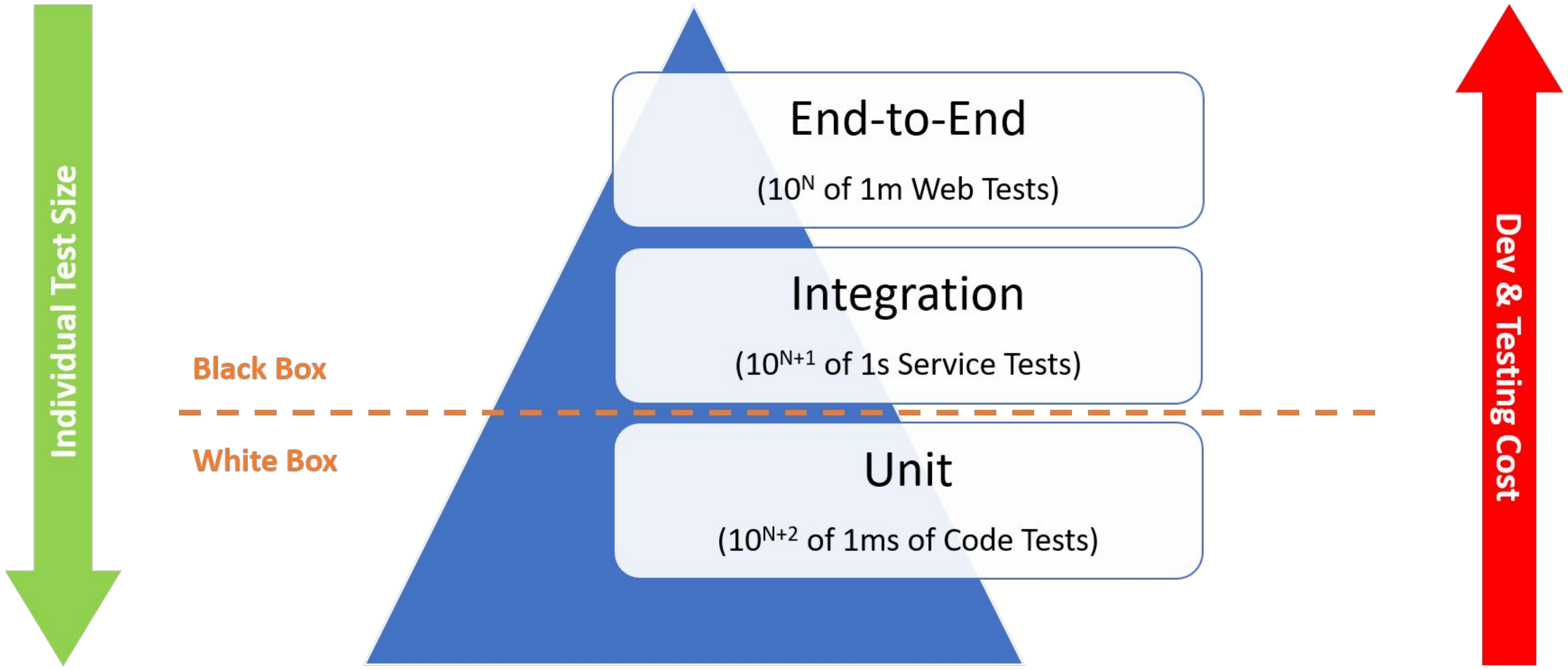
“Unit” or “Subcutaneous”

BLACK BOX

Does the product meet the requirements?

“Integration” or “End-to-End”

The Testing Pyramid



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

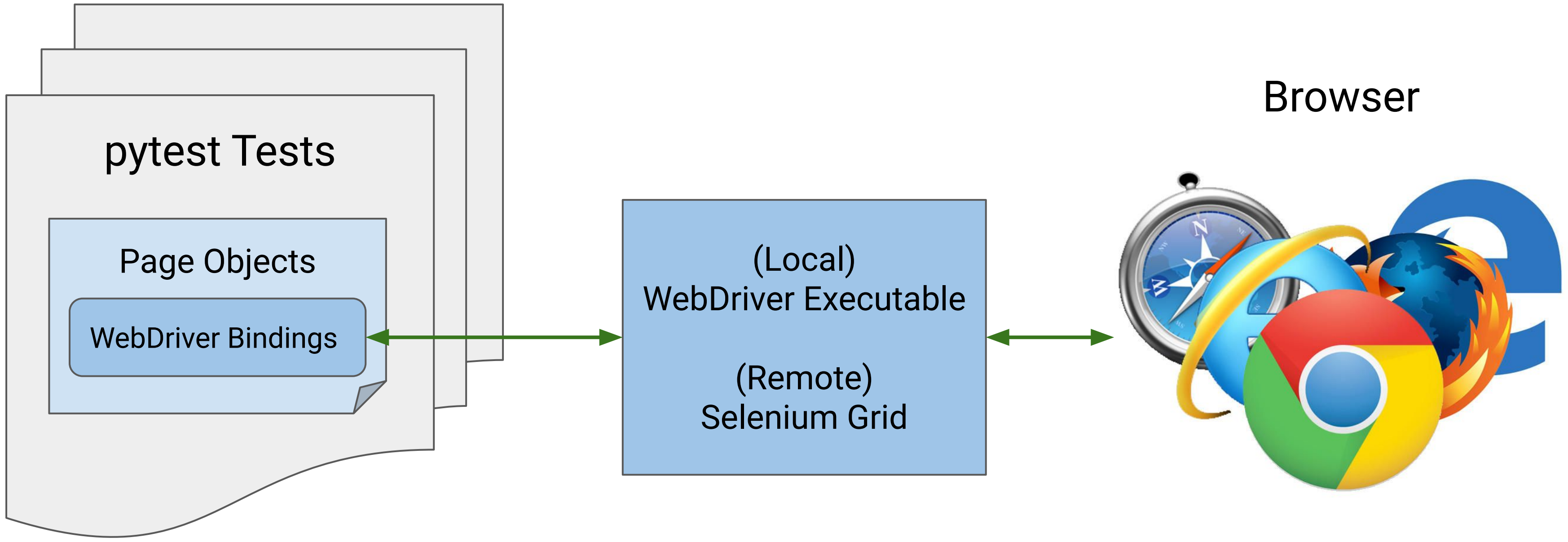


Image Source:
https://www.zdnet.com/article/which-browser-is-most-popular-on-each-major-operating-system/

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.

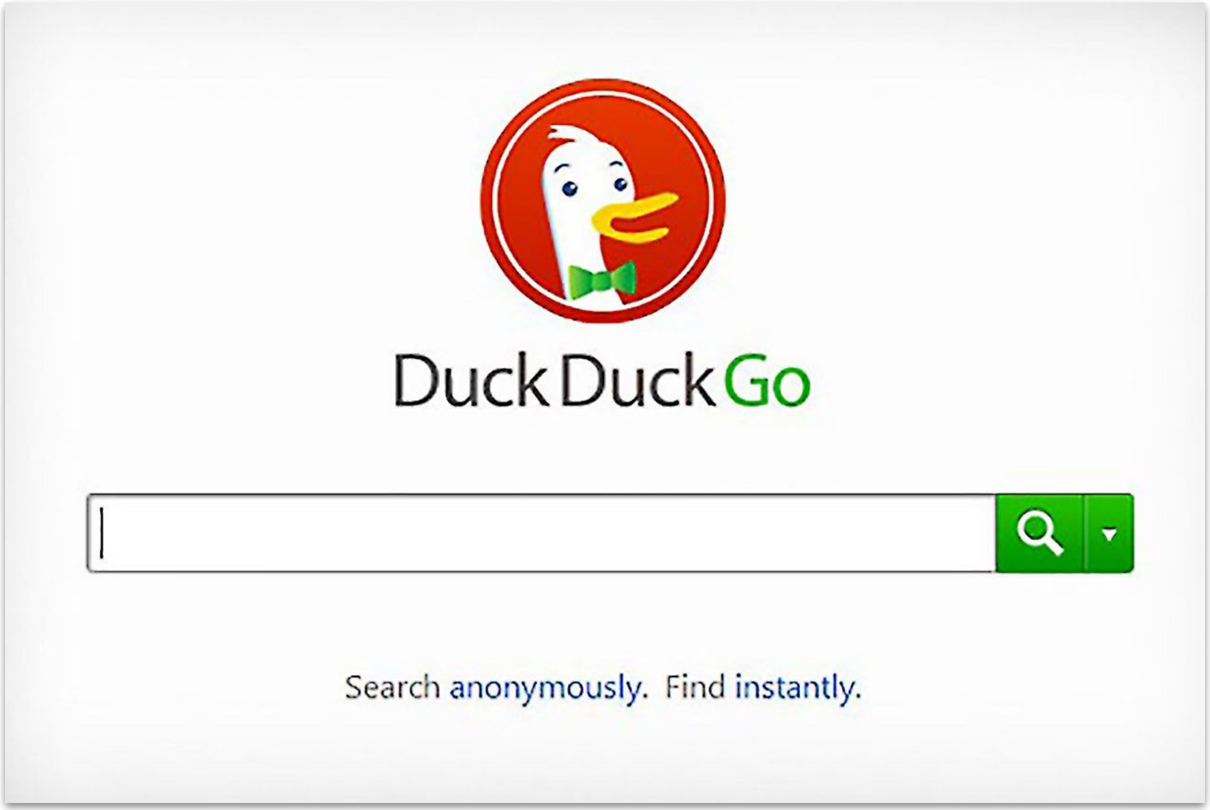
Codeless tools are great for testers who can’t code. However:

- 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

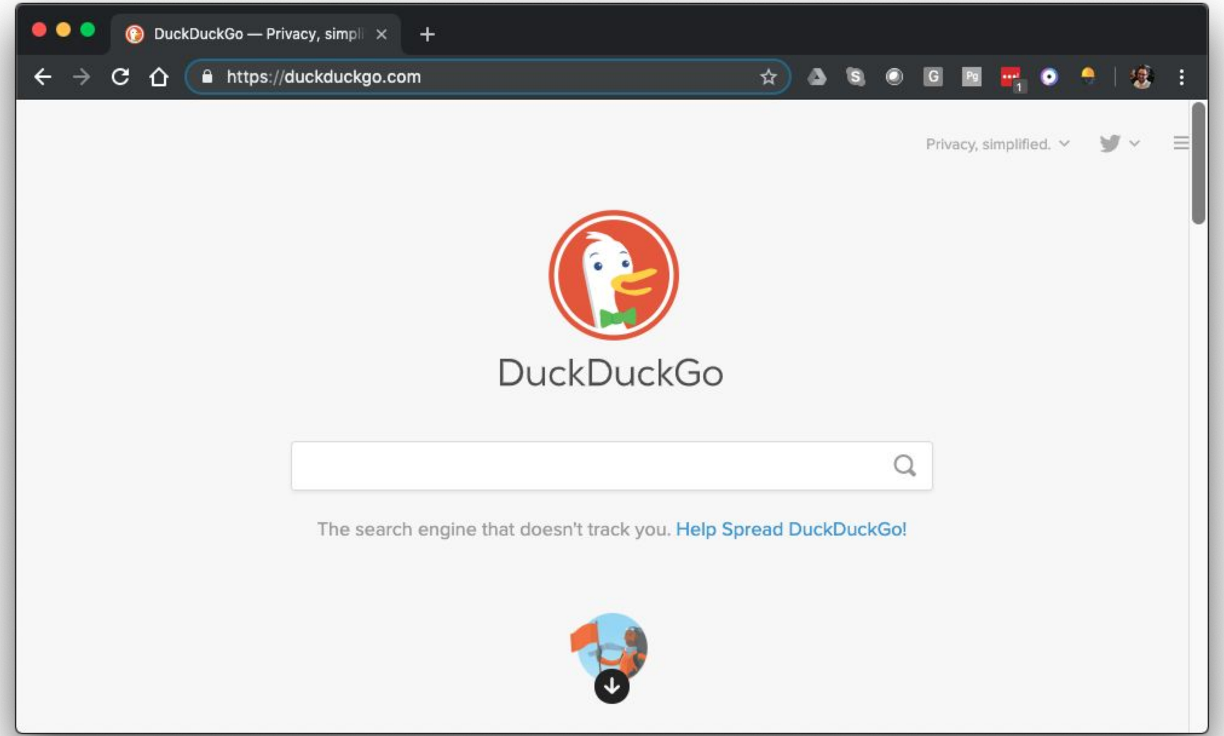


Everyone Do a Search!

Let’s write a basic Web test together!

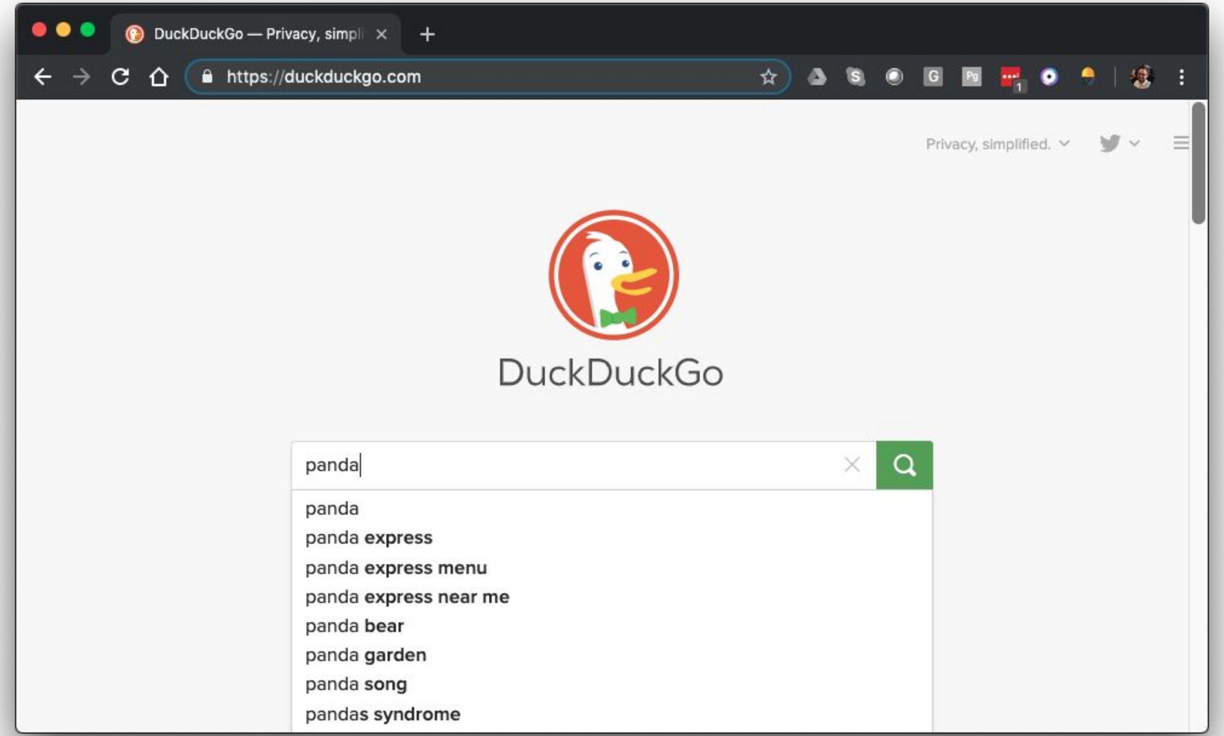
Always write test steps <u>before</u> test code.

Step 1: Navigate to DuckDuckGo



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

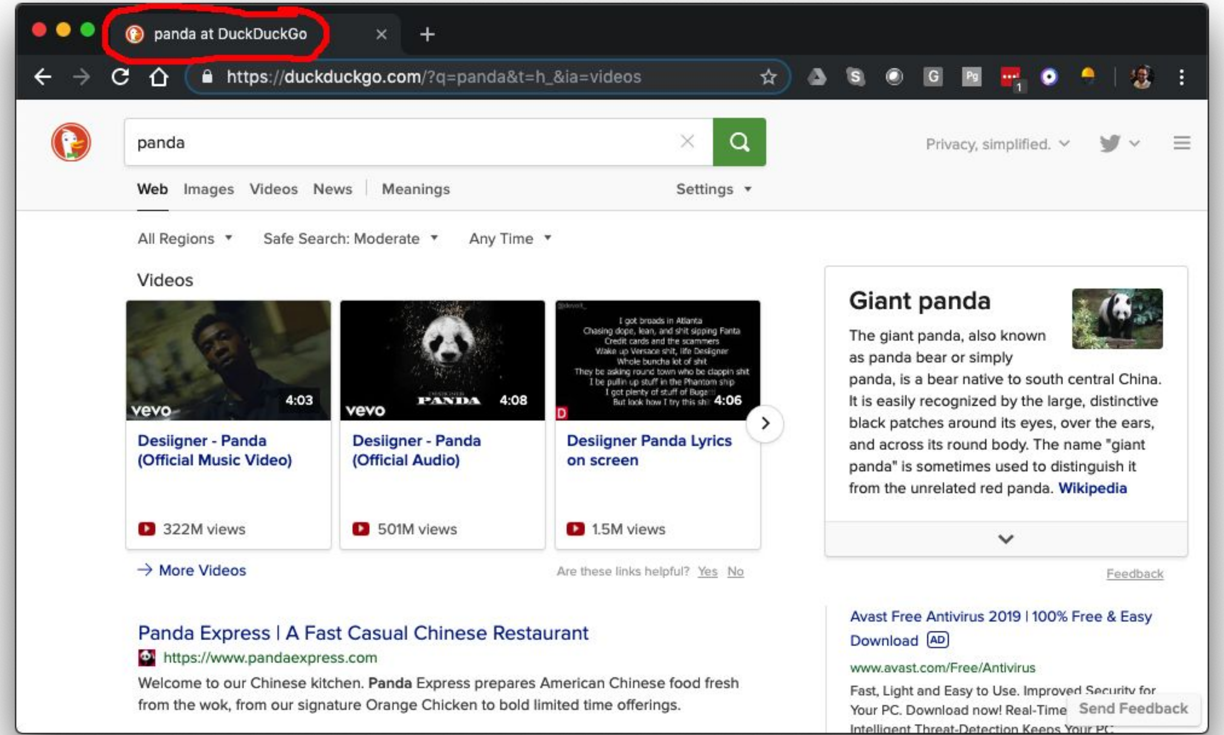
Step 2: Enter a search phrase



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



panda at DuckDuckGo

https://duckduckgo.com/?q=panda&t=h_&ia=videos

panda

Web Images Videos News Meanings Settings
Privacy, simplified.
All Regions Safe Search: Moderate Any Time

Videos

Desiigner - Panda (Official Music Video) 4:03 322M views

Desiigner - Panda (Official Audio) 4:08 501M views

Desiigner Panda Lyrics on screen 4:06 1.5M views

More Videos

Are these links helpful? Yes No

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

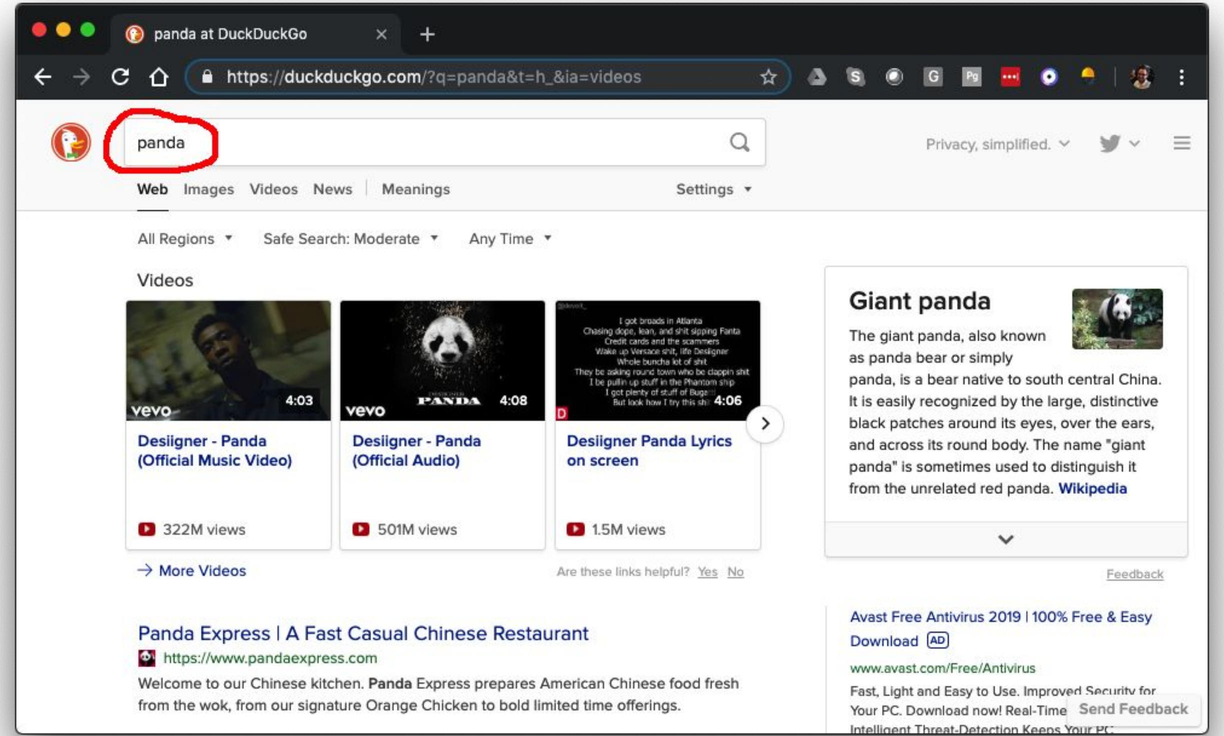
Feedback

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

Send Feedback

Step 4: Verify query on results page

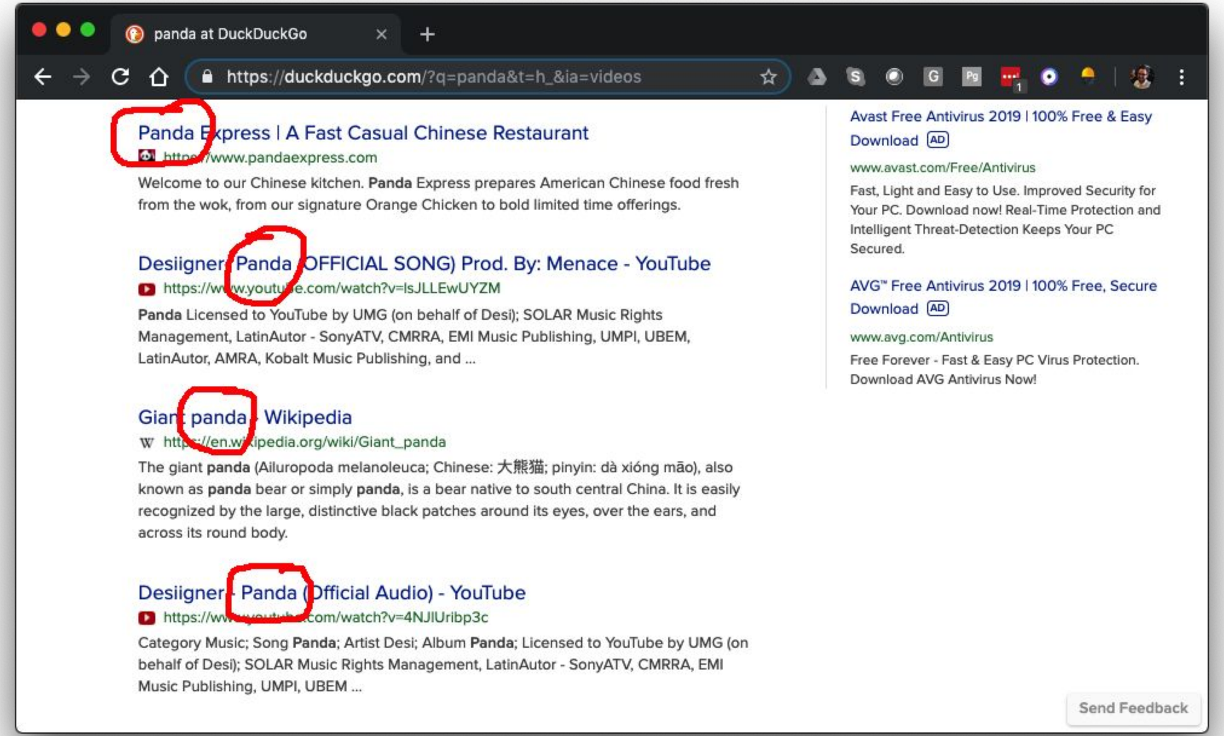


The giant panda, also known as panda bear or simply panda, is a bear native to south central China. It is easily recognized by the large, distinctive black patches around its eyes, over the ears, and across its round body. The name "giant panda" is sometimes used to distinguish it from the unrelated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

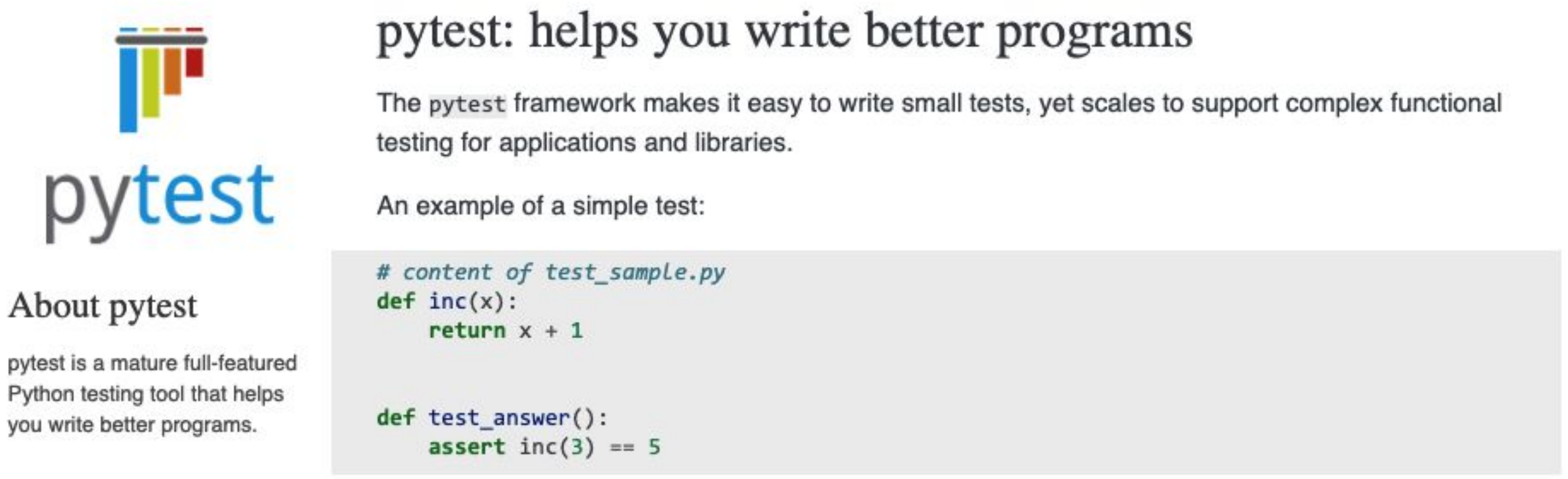
AVG™ Free Antivirus 2019 | 100% Free, Secure Download AD
www.avg.com/Antivirus
Free Forever - Fast & Easy PC Virus Protection. Download AVG Antivirus Now!

Send Feedback

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: helps you write better programs

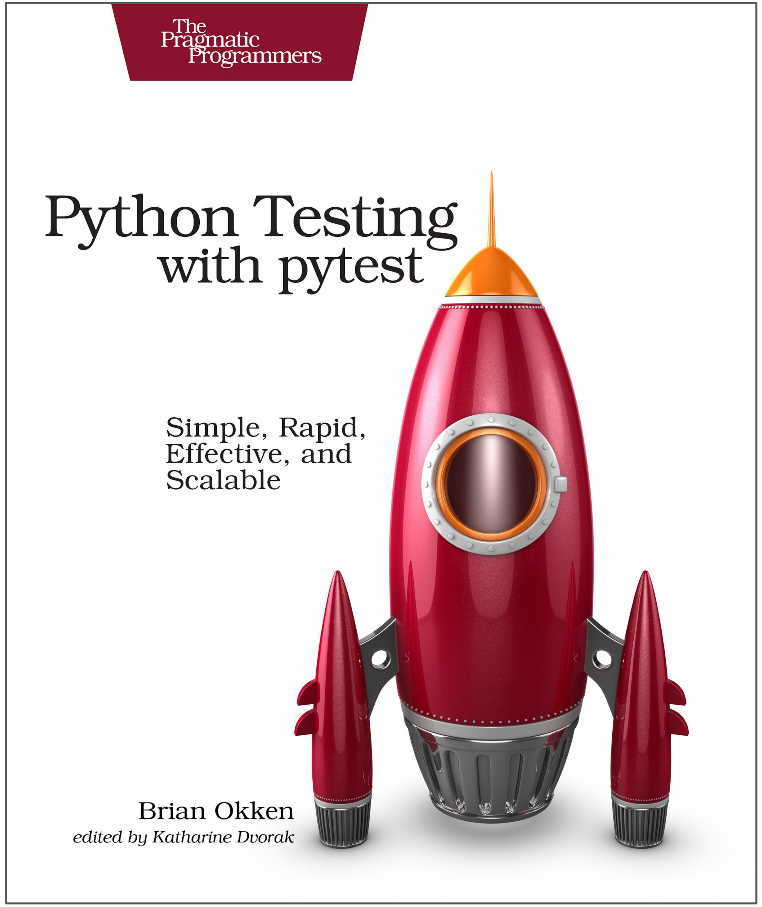
The `pytest` framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

An example of a simple test:

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


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

About pytest

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



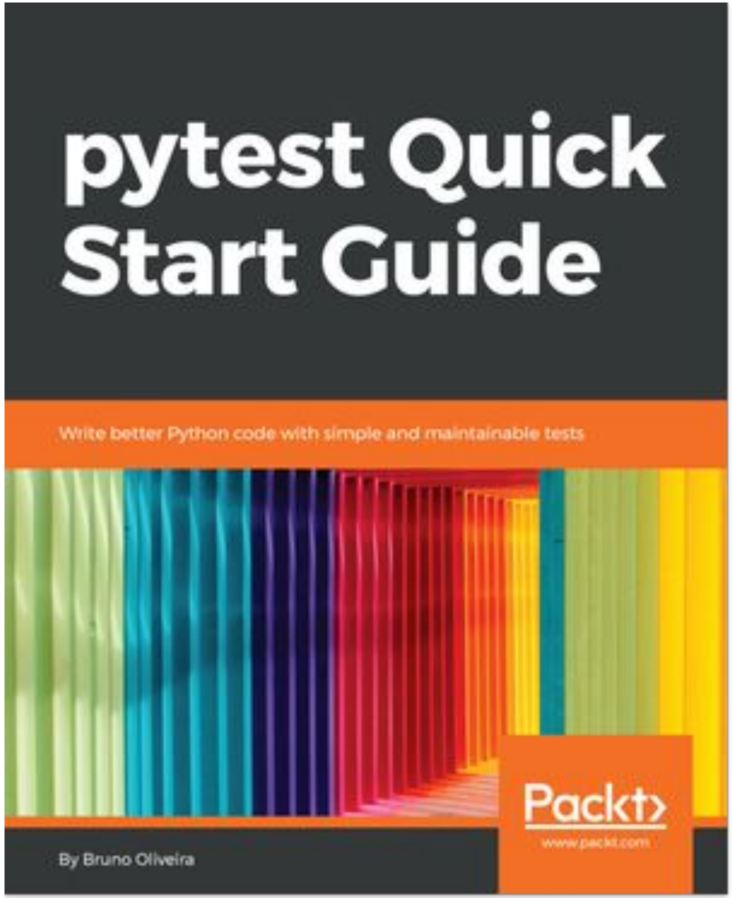
The Pragmatic Programmers

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

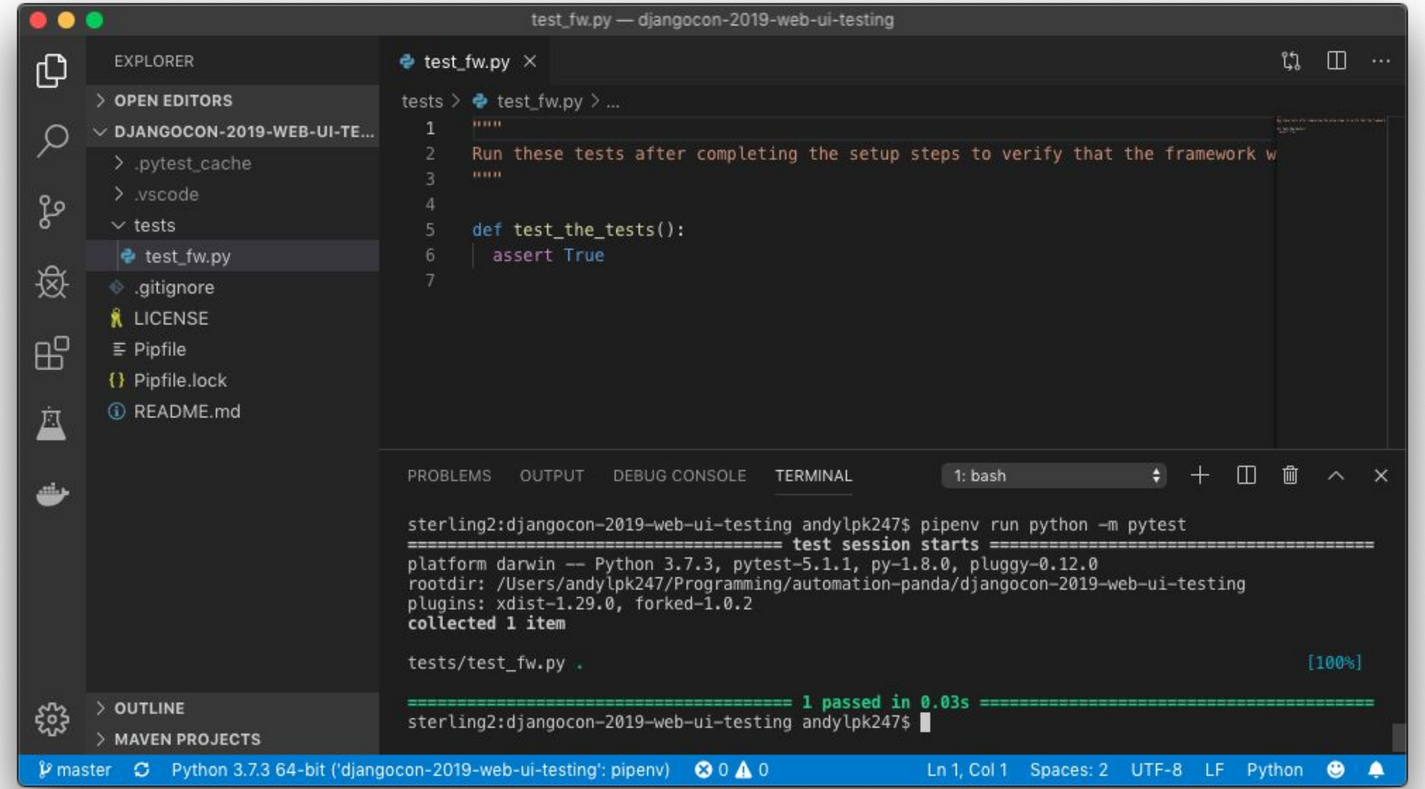
Packt>

www.packt.com

By Bruno Oliveira

```
pipenv install pytest
```

pytest in Our Project



```python
"""
Run these tests after completing the setup steps to verify that the framework w
"""


def test_the_tests():
  assert True
```

```
sterling2:djangocon-2019-web-ui-testing andylpk247$ pipenv run python -m pytest
======================================= test session starts ========================================
platform darwin -- Python 3.7.3, pytest-5.1.1, py-1.8.0, pluggy-0.12.0
rootdir: /Users/andylpk247/Programming/automation-panda/djangocon-2019-web-ui-testing
plugins: xdist-1.29.0, forked-1.0.2
collected 1 item

tests/test_fw.py .                                                                           [100%]

======================================= 1 passed in 0.03s ==========================================
sterling2:djangocon-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 *5 minutes*.

https://github.com/AndyLPK247/djangocon-2019-web-ui-testing

Our First Test in Comments

```python
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")
```

```
Project Tree:
.
└── tests
    └── test_search.py
```

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

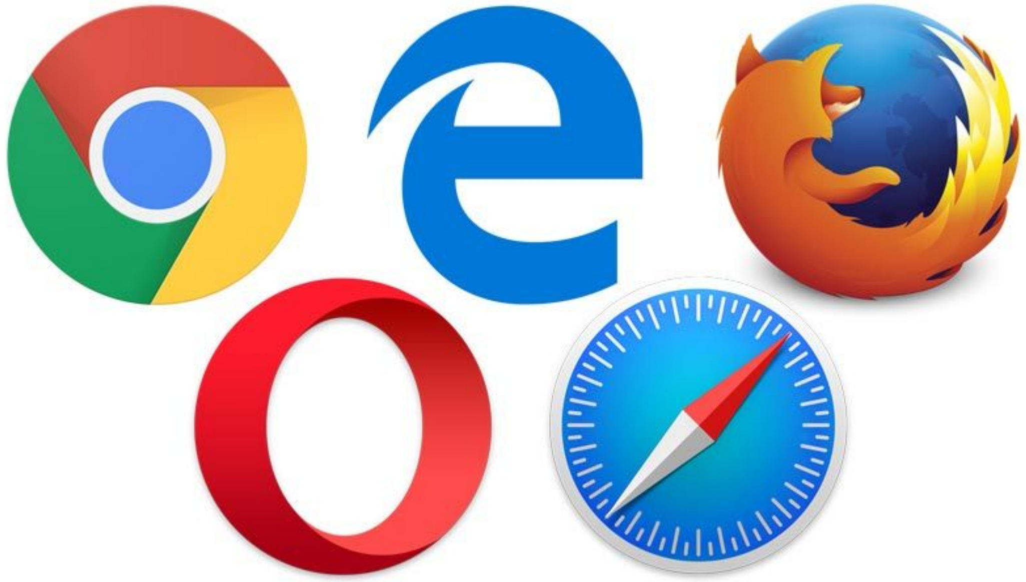
Every test case should have its own WebDriver instance.

- 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

```python
import pytest
import selenium.webdriver

@pytest.fixture
def browser():
  # This browser will be local
  # ChromeDriver must be on the system PATH
  b = selenium.webdriver.Chrome()
  b.implicitly_wait(10)
  yield b
  b.quit()
```

```
Project Tree:
.
└── tests
    ├── conftest.py
    └── test_search.py
```

Using the Fixture

```python
def test_basic_duckduckgo_search(browser):

  # 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")
```

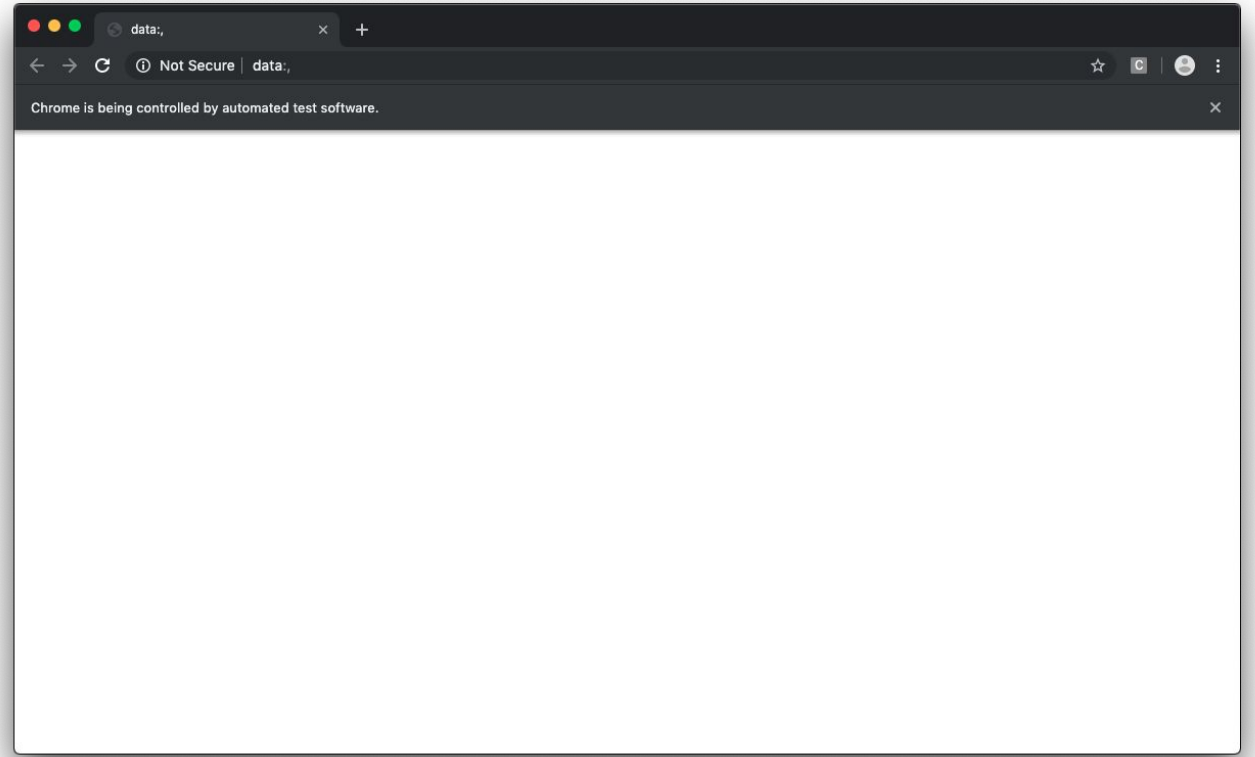
```
Project Tree:
.
└── tests
    ├── conftest.py
    └── test_search.py
```

Hands-On Time!

Complete **Tutorial Instructions Part 2** in the README.
Take *5 minutes*.

```
https://github.com/AndyLPK247/djangocon-2019-web-ui-testing
```

WebDriver-Controlled Chrome



Chrome is being controlled by automated test software.

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 link titles
- Get the search input value
- Get the title

Page Object Class Stubs

```python
class DuckDuckGoSearchPage:

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

  def load(self):
    pass

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

```python
class DuckDuckGoResultPage:

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

  def result_link_titles(self):
    return []

  def search_input_value(self):
    return ""

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

Adding Page Object Calls to the Test

```python
def test_basic_duckduckgo_search(browser):
  search_page = DuckDuckGoSearchPage(browser)
  result_page = DuckDuckGoResultPage(browser)

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

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

  # Then the search result title contains "panda"
  assert "panda" in result_page.title()
  # And the search result query is "panda"
  assert "panda" == result_page.search_input_value()
  # And the search result links pertain to "panda"
  for title in result_page.result_link_titles():
    assert "panda" in title.lower()
```

```
Project Tree:
.
├── pages
│   ├── __init__.py
│   ├── result.py
│   └── search.py
└── tests
    ├── conftest.py
    └── test_search.py
```

Hands-On Time!

Complete **Tutorial Instructions Part 3** in the README.
Take *10 minutes*.

https://github.com/AndyLPK247/djangocon-2019-web-ui-testing

Finding Locators for Elements

Web Elements

An *element* is a “thing” on a Web page, like a button, label, or text input.

Tests interact with elements in three steps:

1. Wait for the target element to appear
2. Get an object representing the target element
3. Send commands to the element object

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

Example ID Locator

Element:

```
<button id="django_ok">OK</button>
```

Locator:

```
(By.ID, "django_ok")
```

Example CSS Selector Locator

Element:

```
<button class="django_ok">OK</button>
```

Locator:

```
(By.CSS_SELECTOR, "button.django_ok")
```

Example XPath Locator

Element:

```
<button>OK</button>
```

Locator:

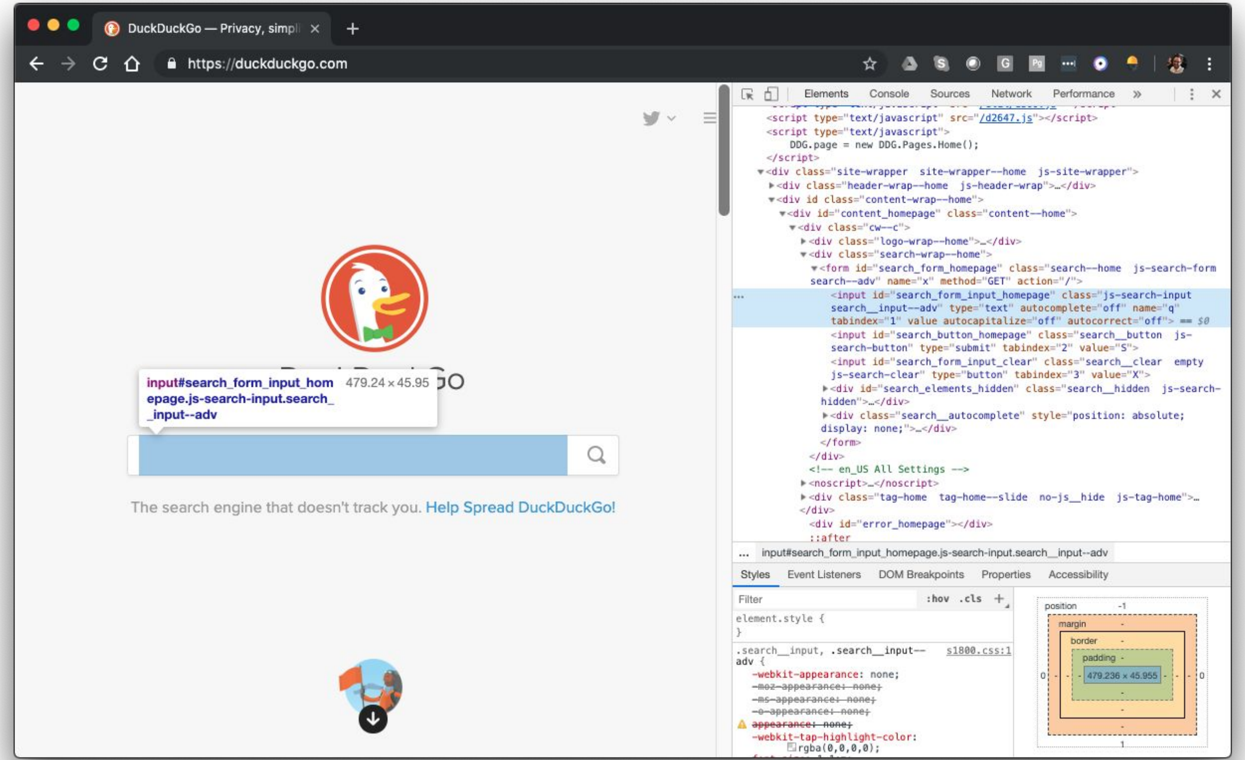
```
(By.XPATH, "//button[text()='OK']")
```

Elements Needed for our Test

1. The search input on the DuckDuckGo search page
2. The search input on the DuckDuckGo results page
3. The result links on the DuckDuckGo results page

Finding Elements to Write Locators

Use Chrome DevTools!



```
<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);
```

Adding Locators to Page Objects

```python
from selenium.webdriver.common.by import By

class DuckDuckGoSearchPage:

  SEARCH_INPUT = (By.ID, 'search_form_input_homepage')

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

  def load(self):
    pass

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

```
Project Tree:
.
├── pages
│   ├── __init__.py
│   ├── result.py
│   └── search.py
└── tests
    ├── conftest.py
    └── test_search.py
```

Hands-On Time!

Complete **Tutorial Instructions Part 4** in the README.
Take *15 minutes*.

https://github.com/AndyLPK247/djangocon-2019-web-ui-testing

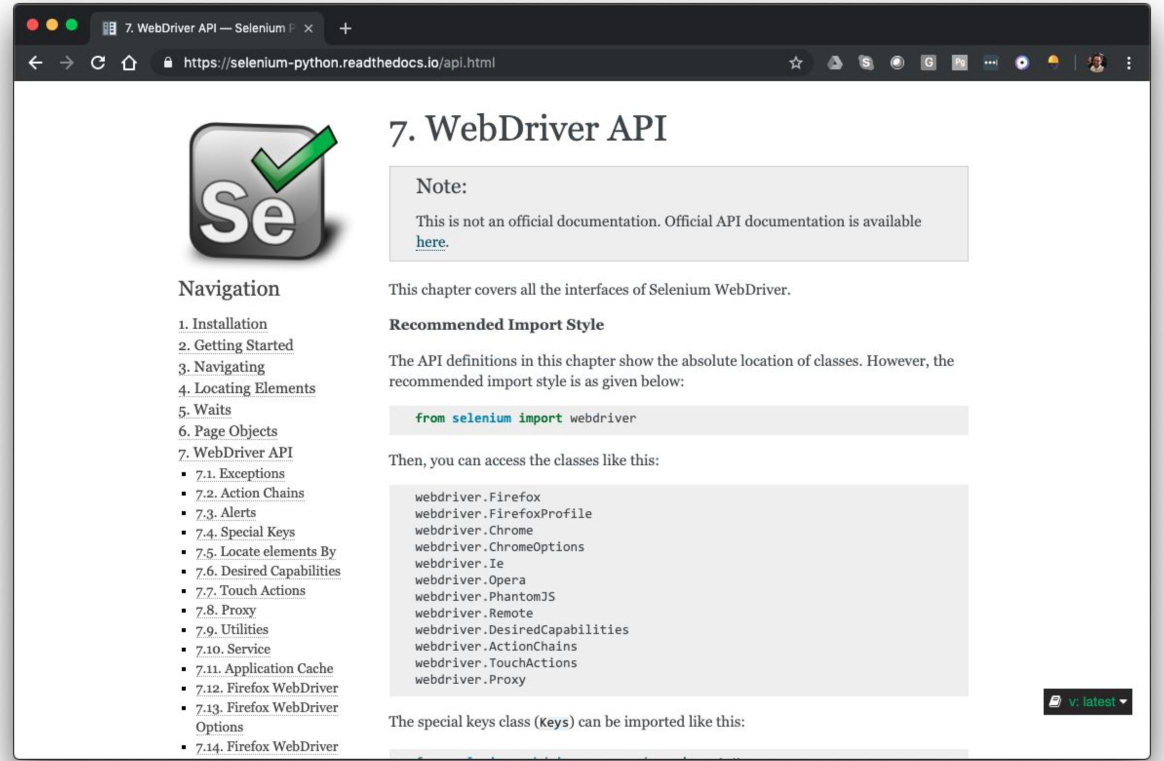
The Result Page Object

```python
from selenium.webdriver.common.by import By

class DuckDuckGoResultPage:

  RESULT_LINKS = (By.CSS_SELECTOR, 'a.result__a')
  SEARCH_INPUT = (By.ID, 'search_form_input')

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

  def result_link_titles(self):
    return []

  def search_input_value(self):
    return ""

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

```
Project Tree:

.
├── pages
│   ├── __init__.py
│   ├── result.py
│   └── search.py
└── tests
    ├── conftest.py
    └── test_search.py
```

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.

Getting a Page's Title

```python
class DuckDuckGoResultPage:

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

```
Project Tree:

.
├── pages
│   ├── __init__.py
│   ├── result.py
│   └── search.py
└── tests
    ├── conftest.py
    └── test_search.py
```

Many calls interact with **elements**.

Entering a Search Phrase

```
class DuckDuckGoSearchPage:

  # The locator
  SEARCH_INPUT = (By.ID, 'search_form_input_homepage')

  def search(self, phrase):

    # Finding the target element
    search_input = self.browser.find_element(
        *self.SEARCH_INPUT)

    # Sending a command to the element
    search_input.send_keys(phrase + Keys.RETURN)
```

```
Project Tree:
.
├── pages
│   ├── __init__.py
│   ├── result.py
│   └── search.py
└── tests
    ├── conftest.py
    └── test_search.py
```

Common WebDriver Calls

For WebDriver:

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

For Elements:

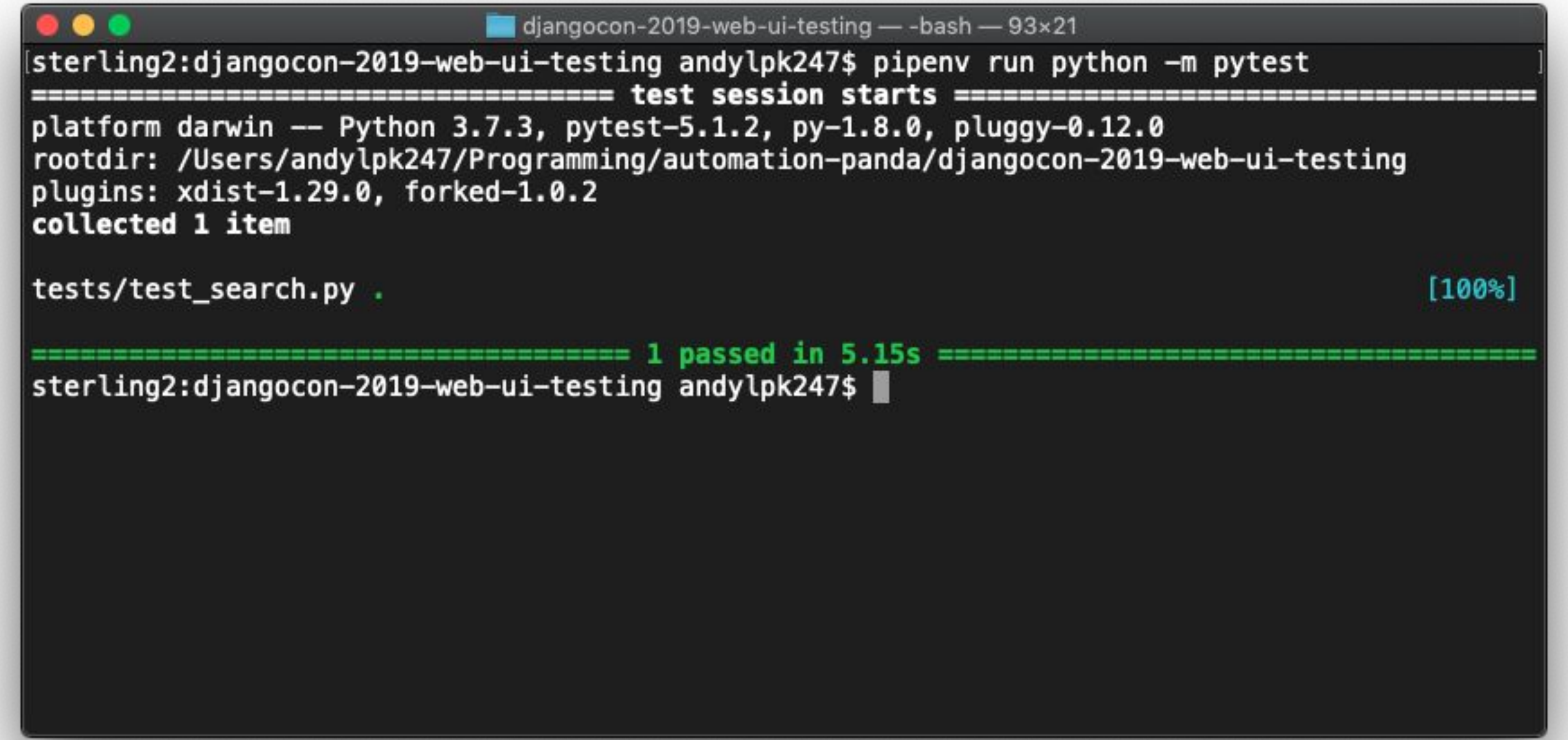
- `clear`
- `click`
- `find_element*`
- `get_attribute`
- `get_property`
- `is_displayed`
- `location`
- `send_keys`
- `size`
- `text`

Hands-On Time!

Complete **Tutorial Instructions Part 5** in the README.
Take *15 minutes*.

https://github.com/AndyLPK247/djangocon-2019-web-ui-testing

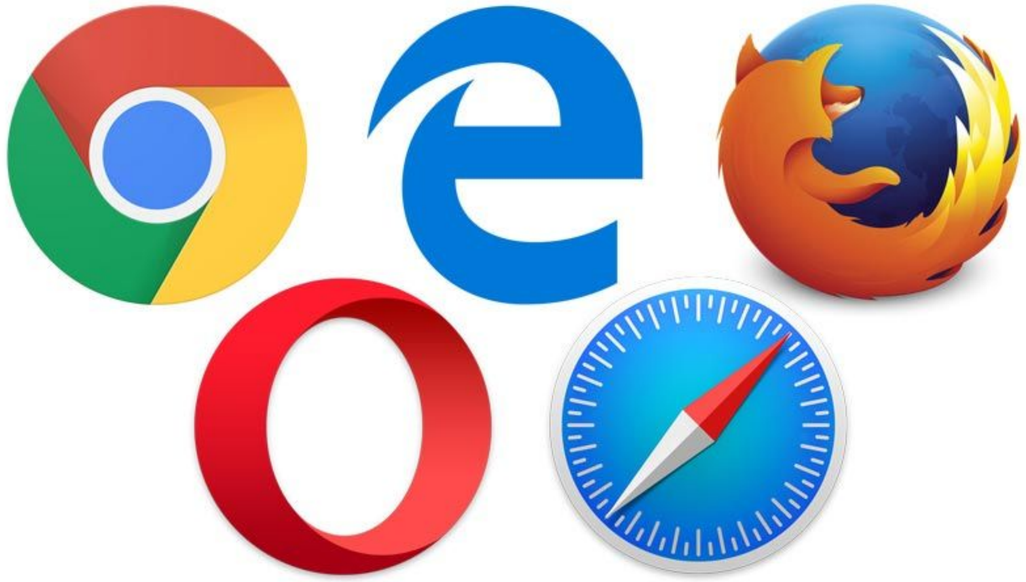
A Successful Test Run



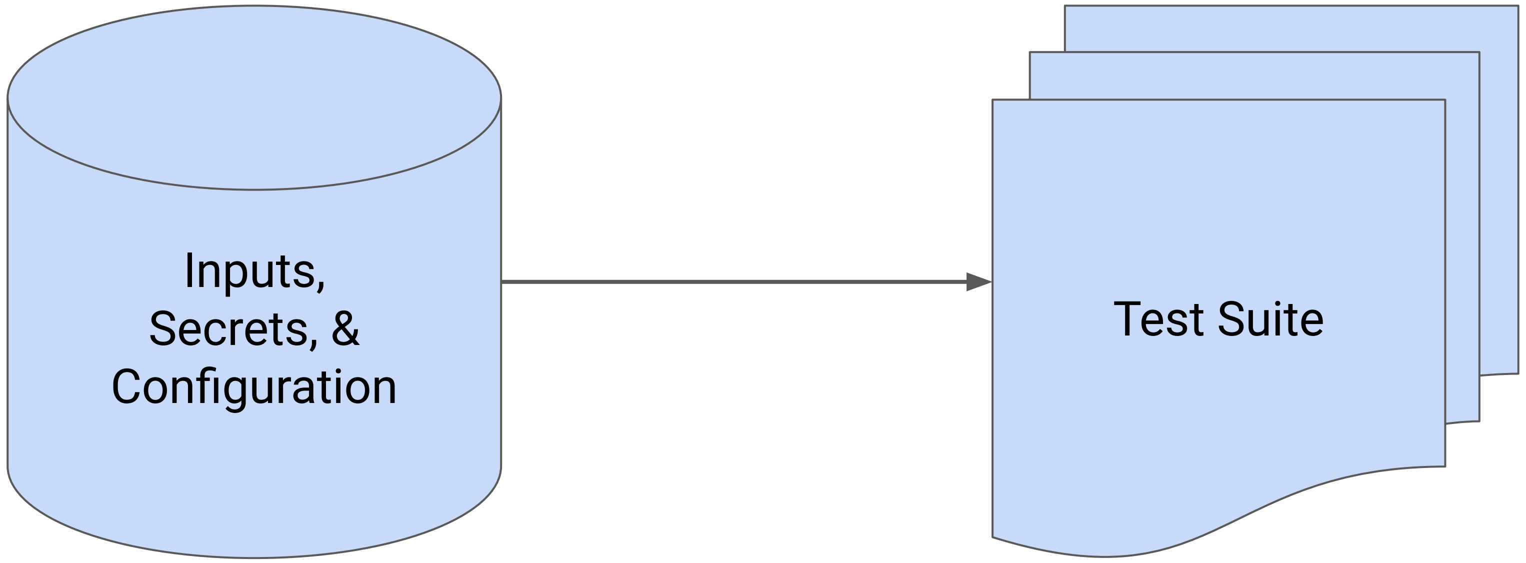
```
sterling2:djangocon-2019-web-ui-testing andylpk247$ pipenv run python -m pytest
==================================== test session starts =====================================
platform darwin -- Python 3.7.3, pytest-5.1.2, py-1.8.0, pluggy-0.12.0
rootdir: /Users/andylpk247/Programming/automation-panda/djangocon-2019-web-ui-testing
plugins: xdist-1.29.0, forked-1.0.2
collected 1 item

tests/test_search.py .                                                                  [100%]

===================================== 1 passed in 5.15s ======================================
sterling2:djangocon-2019-web-ui-testing andylpk247$ 
```

Configuring Multiple Browsers

Again: Which Browser Type?



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



Don’t hard-code inputs, secrets, or config into code!
Don’t commit secrets to source control, either!

Reading Inputs

Ways to pass inputs into test automation:

- Config files (JSON, YAML, INI, CSV, etc.)
- Encrypted files
- Environment variables
- Key management service (AWS KMS, Azure KeyVault)
- ~~custom command line arguments for pytest~~

For Python, JSON config files are the easiest solution.

- JSON is readable and hierarchical
- The `json` module is part of the standard library
- The `json` module can parse a .json file into a dictionary *in one line*

Why Not Parametrize Browser Choice?

In pytest, a test can be parametrized with sets of inputs. A test could be parametrized to run once for each in a list of browser types.

However, **browser choice is an aspect of testing**.

- In theory, every test should run on every supported browser.
- Parametrizing every test case would be duplicative.
- To run different browsers, simply launch multiple test suite runs.

Our Config File

```json
{
  "browser": "Chrome",
  "implicit_wait": 10
}
```

```
Project Tree:
.
├── pages
│   ├── __init__.py
│   ├── result.py
│   └── search.py
├── tests
│   ├── conftest.py
│   └── test_search.py
└── config.json
```

Reading the Config File

```python
import json

@pytest.fixture
def config(scope='session'):

  # Read the file
  with open('config.json') as config_file:
    config = json.load(config_file)

  # Assert values are acceptable
  supported = ['Firefox', 'Chrome', 'Headless Chrome']
  assert config['browser'] in supported
  assert isinstance(config['implicit_wait'], int)
  assert config['implicit_wait'] > 0

  # Return config so it can be used
  return config
```

```
Project Tree:
.
├── pages
│   ├── __init__.py
│   ├── result.py
│   └── search.py
├── tests
│   ├── conftest.py
│   └── test_search.py
└── config.json
```

Updating the Browser Fixture

```python
@pytest.fixture
def browser(config):

  if config['browser'] == 'Firefox':
    b = selenium.webdriver.Firefox()
  elif config['browser'] == 'Chrome':
    b = selenium.webdriver.Chrome()
  elif config['browser'] == 'Headless Chrome':
    opts = selenium.webdriver.ChromeOptions()
    opts.add_argument('headless')
    b = selenium.webdriver.Chrome(options=opts)
```

```
Project Tree:
.
├── pages
│   ├── __init__.py
│   ├── result.py
│   └── search.py
├── tests
│   ├── conftest.py
│   └── test_search.py
└── config.json
```

Headless browsers are great for test automation.

Hands-On Time!

Complete **Tutorial Instructions Part 6** in the README.
Take *15 minutes*.

https://github.com/AndyLPK247/djangocon-2019-web-ui-testing

Handling Race Conditions

How many people had test failures for **Firefox**?

Race Condition

When actors access the same resource at the same time without following an order of operations.

Race conditions can cause *flakiness* in any black box test.

Web UI Race Conditions

The most common race condition in Web UI test automation is **attempting to access an element before it appears on the page**.

To mitigate this race condition, always wait for the target element to appear before calling `find_element*` methods. The implicit wait setting in the browser fixture handles most cases automatically.

Our Test's Race Condition

```
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"
```

Hands-On Time!

Complete **Tutorial Instructions Part 7** in the README.
Take *15 minutes*.

https://github.com/AndyLPK247/djangocon-2019-web-ui-testing

Our Corrected Test Steps

```
Scenario: Basic DuckDuckGo Search

    Given the DuckDuckGo home page is displayed

    When the user searches for “panda”

    Then the search result query is “panda”

    And the search result links pertain to “panda”

    And the search result title contains “panda”
```

Running Tests in Parallel

Web UI tests are **slow**.

1 min

(typical Web UI test execution time)

Parallel Testing

Parallel testing is a great way to speed up test suites.

- Use `pytest-xdist` to add parallel testing to pytest.
- Avoid *collisions* (when tests access shared state).
- Locally limit the thread count to one Web UI test per processor/core.
- Use Selenium Grid to massively scale out parallel testing.

```
pipenv install pytest-xdist
```

```
python -m pytest -n 3
```

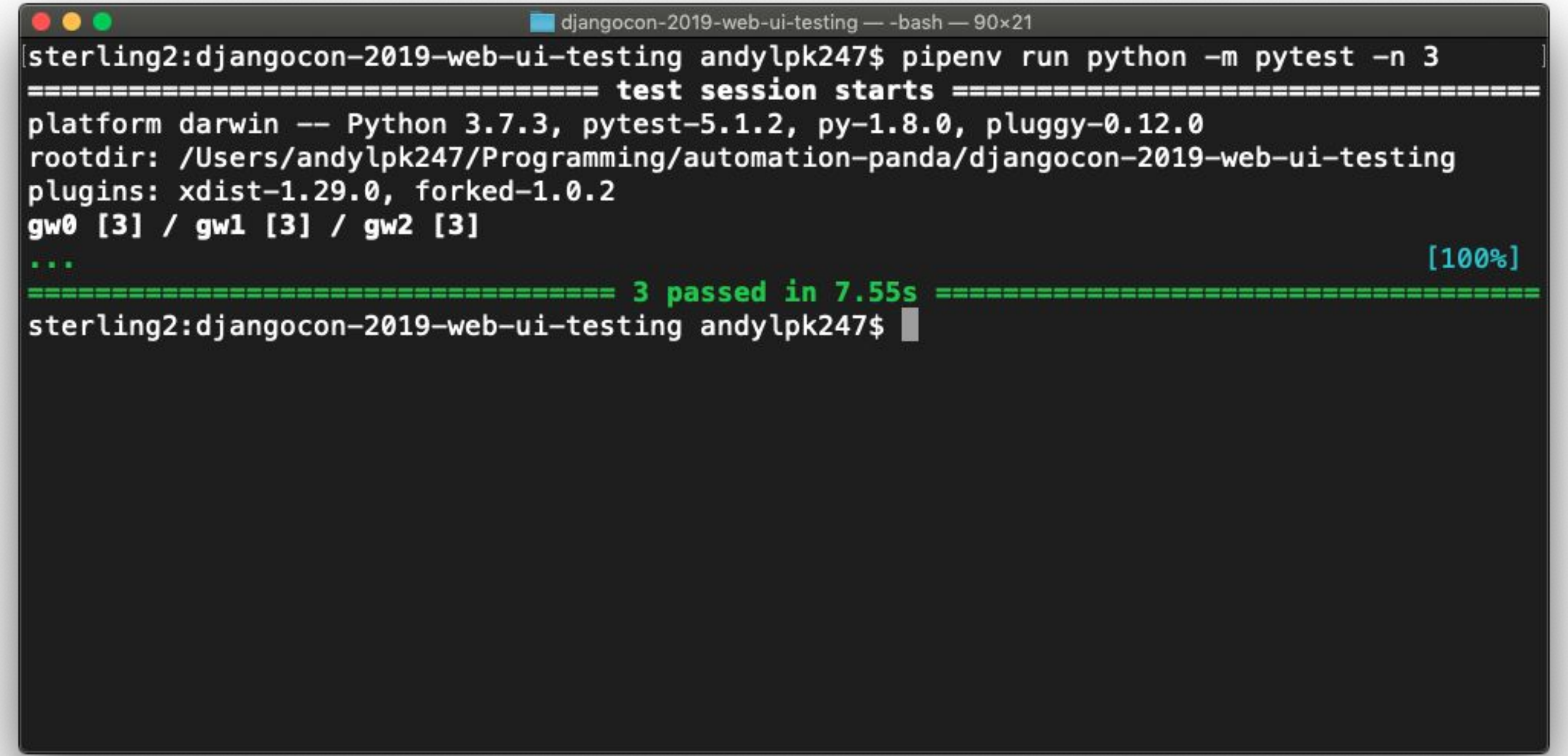
Hands-On Time!

Complete **Tutorial Instructions Part 8** in the README.

Take *15 minutes*.

https://github.com/AndyLPK247/djangocon-2019-web-ui-testing

Successful Parallel Tests



```
sterling2:djangocon-2019-web-ui-testing andylpk247$ pipenv run python -m pytest -n 3
================================== test session starts ===================================
platform darwin -- Python 3.7.3, pytest-5.1.2, py-1.8.0, pluggy-0.12.0
rootdir: /Users/andylpk247/Programming/automation-panda/djangocon-2019-web-ui-testing
plugins: xdist-1.29.0, forked-1.0.2
gw0 [3] / gw1 [3] / gw2 [3]
...                                                                                 [100%]
=================================== 3 passed in 7.55s ====================================
sterling2:djangocon-2019-web-ui-testing andylpk247$
```

Congrats!

You finished the tutorial.

Homework:

Do the *Independent Exercises*.

Write more tests!

Resources

- Test Automation University
    - Web Element Locator Strategies
    - Behavior-Driven Development with pytest-bdd
    - Setting a Foundation for Successful Test Automation
- Tutorials
    - TestProject: *Web Testing Made Easy with Python, Pytest and Selenium WebDriver*
    - SmartBear CrossBrowserTesting: *Hands-On UI Testing with Python*
    - PyOhio 2019: https://github.com/AndyLPK247/pyohio-2019-web-ui-testing
- Automation Panda blog
    - Testing page
    - Python page