Learning Objectives: Testing

Learners will be able to...

- **Differentiate the four different Django test classes**
- **Test status codes and URL patterns with `assertEquals`**
- **Test templates with `assertTemplateUsed`**
- **Test content with `assertContains`**

info

Make Sure You Know

You have some familiarity with Python's testing framework or Java's JUnit.

Limitations

The testing covered here is enough for a basic test of our website. This is not a deep-dive into Django's testing framework.

Testing in Django

Django Testing Framework

Start by activating the `django` virtual environment.

```
conda activate django
```

Testing is an important part of web development with Django. In fact, every time you create a Django app, a test file is included. The expectation is that you test all of the code you write, even if it is a simple website.

Django's testing framework extends Python's built-in testing framework. The most common test cases in Django are:

- `SimpleTestCase` - used to test functionality that does not rely on the database
- `TestCase` - used when testing the database
- `TransactionTestCase` - used to test database transactions
- `LiveServerTestCase` - launches a live server so you can run other browser-based testing tools

▼ **Did you notice?**

> Django test cases use camel case (`TestCase`) for their naming convention as opposed to snake case (`Test_Case`) you typically see in Python.

Since our website does not make use of the database, our testing will make use of the `SimpleTestCase` class.

Testing HTTP Response Status

When a client (your browser) makes a request to a server (like loading a page), the server also sends a response code. You are probably most familiar with the "404 Not Found" response code when a page does not load. We fully expect our homepage, about page, and three programming pages to load as expected. The response code for this would be 200, which is what we will test for first. Open the `tests.py` file.

The file on the left is the `test.py` file found in the `languages` directory. This file will contain all the tests for the languages Django app. Alter the first line of the file to import `SimpleTestCase` as we are not using the database. Create the `HomepageTests` class that inherits from `SimpleTestCase`. Then define the method `test_url_by_pattern`. Yes, this name is long, but the method names should be completely clear as to what it is the method is testing. Pass `self` as an argument to the method.

```
from django.test import SimpleTestCase

class HomepageTests(SimpleTestCase):
  def test_url_by_pattern(self):
```

Create the variable `response` which contains the response from requesting the URL pattern `'/'`, which is the pattern for home page. Django's (and Python's) testing framework relies on assertion statements. If the assertion is true, the test cases passes. If not, the test case fails. In this example, we are going to use `assertEqual` which verifies if two values are equal. We want to know if the status code from requesting the homepage is equal to 200. Pass `response.status_code` and `200` to the `self.assertEqual` method.

```
from django.test import SimpleTestCase

class HomepageTests(SimpleTestCase):
  def test_url_by_pattern(self):
    response = self.client.get('/')
    self.assertEqual(response.status_code, 200)
```

Creating the test for the about page is going to be very similar. Create a test class and inherit from `SimpleTestCase`. Create method to test for the 200 HTTP response. Be sure, however, that you are using the appropriate URL pattern. In particular, notice how there is a slash before and after the word `about`. If you do not have both slashes, the test will fail.

```
class AboutpageTests(SimpleTestCase):
  def test_url_by_pattern(self):
    response = self.client.get('/about/')
    self.assertEqual(response.status_code, 200)
```

Now create similar tests for the Python, F#, and Racket pages. Be sure to use the appropriate names for the test classes and URL patterns.

```
class PythonpageTests(SimpleTestCase):
  def test_url_by_pattern(self):
    response = self.client.get('/python/')
    self.assertEqual(response.status_code, 200)

class FsharppageTests(SimpleTestCase):
  def test_url_by_pattern(self):
    response = self.client.get('/fsharp/')
    self.assertEqual(response.status_code, 200)

class RacketpageTests(SimpleTestCase):
  def test_url_by_pattern(self):
    response = self.client.get('/racket/')
    self.assertEqual(response.status_code, 200)
```

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black languages/tests.py
```

Once all the tests are done, run the test file with the command below.

```
python manage.py test
```

If all of your tests pass, you should see the following output:

```
Found 5 test(s).
System check identified no issues (0 silenced).
.....
----------------------------------------------------------------------
------
Ran 5 tests in 0.026s

OK
```

Deactivate the virtual environment.

```
conda deactivate
```

▼ **Code**

Your test file should look like this:

```
from django.test import SimpleTestCase


class HomepageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/")
        self.assertEqual(response.status_code, 200)


class AboutpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/about/")
        self.assertEqual(response.status_code, 200)


class PythonpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/python/")
        self.assertEqual(response.status_code, 200)


class FsharppageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/fsharp/")
        self.assertEqual(response.status_code, 200)


class RacketpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/racket/")
        self.assertEqual(response.status_code, 200)
```

URL Names

Django Reverse

Start by activating the `django` virtual environment.

```
conda activate django
```

On the previous page, we tested our website by using the URL patterns. In this test, we are going to evaluate the website by the name given to each page in the `urls.py` file. In order to do this, we are going to make use of the `reverse` function from Django. Open the `tests.py` file and import `reverse`.

```
from django.test import SimpleTestCase
from django.urls import reverse
```

Testing the Name

We have aleady created the `HomepageTests` class. Any further tests for the homepage will be a method in this class. Create the `test_url_by_name` method which has `self` as an argument. Just like the last test case, we are going to generate a response with the `get()` method. However, we are going to use `reverse('home')` as the parameter.

```
class HomepageTests(SimpleTestCase):
  def test_url_by_pattern(self):
    response = self.client.get('/')
    self.assertEqual(response.status_code, 200)

  def test_url_by_name(self):
    response = self.client.get(reverse('home'))
    self.assertEqual(response.status_code, 200)
```

Create the `test_url_by_name` method for each of the test classes. Be sure to replace `'home'` with the URL name for each test case.

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black languages/tests.py
```

Run the test file with the command below.

```
python manage.py test
```

If all of your tests pass, you should see the following output:

```
Found 10 test(s).
System check identified no issues (0 silenced).
..........
----------------------------------------------------------------------
------
Ran 10 tests in 0.032s

OK
```

Deactivate the virtual environment.

```
conda deactivate
```

▼ **Code**

> Your test file should look like this:

```
from django.test import SimpleTestCase
from django.urls import reverse


class HomepageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("home"))
        self.assertEqual(response.status_code, 200)


class AboutpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/about/")
        self.assertEqual(response.status_code, 200)
```

```
    def test_url_by_name(self):
        response = self.client.get(reverse("about"))
        self.assertEqual(response.status_code, 200)


class PythonpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/python/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("python"))
        self.assertEqual(response.status_code, 200)


class FsharppageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/fsharp/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("fsharp"))
        self.assertEqual(response.status_code, 200)


class RacketpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/racket/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("racket"))
        self.assertEqual(response.status_code, 200)
```

Templates

Testing Templates

Start by activating the `django` virtual environment.

```
conda activate django
```

Django provides the `assertTemplateUsed` method to check if the proper template is being used. Create the `test_correct_template` method inside the `HomepageTests` class. Just like in the previous test, use the `reverse` function to get information about the URL pattern. This time, use `assertTemplateUsed` and `'home.html'` as the two values needed for comparison.

```
class HomepageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse('home'))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse('home'))
        self.assertTemplateUsed(response, 'home.html')
```

Create the `test_correct_template` method for the other test classes in the file. Replace `'home'` and `'home.html'` with their respective new values.

Running the Tests

Once the new test cases have been added, open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black languages/tests.py
```

Run the test file with the command below.

```
python manage.py test
```

If all of your tests pass, you should see the following output:

```
Found 15 test(s).
System check identified no issues (0 silenced).
...............
----------------------------------------------------------------------
------
Ran 15 tests in 0.037s

OK
```

▼ **Code**

> Your test file should look like this:

```
from django.test import SimpleTestCase
from django.urls import reverse


class HomepageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("home"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("home"))
        self.assertTemplateUsed(response, "home.html")


class AboutpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/about/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("about"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("about"))
```

```
        self.assertTemplateUsed(response, "about.html")


class PythonpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/python/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("python"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("python"))
        self.assertTemplateUsed(response, "python.html")


class FsharppageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/fsharp/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("fsharp"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("fsharp"))
        self.assertTemplateUsed(response, "fsharp.html")


class RacketpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/racket/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("racket"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("racket"))
        self.assertTemplateUsed(response, "racket.html")
```

Content

Testing Content

Start by activating the `django` virtual environment.

```
conda activate django
```

Homepage

The final tests for our website will verify the content on the page is correct. The `assertContains` method allows for you to check for the presence of specific text on a page. You can even specify the number of times you expect to see the text. When it comes to testing for content, you can test to verify that every single word appears on the page. That is a bit much for our purposes. Instead, we will test for important things and not focus on the details. Keep in mind that test can (and probably should) be more rigorous that what we are doing here.

Let's start by testing the homepage. Open the `tests.py` file and add a test that checks for the presence of a title. We will make use of the `reverse` method one more time. We want to test that the tile has the same text and HTML tags.

```
class HomepageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse('home'))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse('home'))
        self.assertTemplateUsed(response, 'home.html')

    def test_for_title(self):
        response = self.client.get(reverse('home'))
        self.assertContains(response, '<h1>Programming
        Languages</h1>')
```

Our homepage also includes the three Bootstrap card components. These are a key feature of the page, so we will test for them as well. **Note**, we will not be testing for the layout. We cannot test if the template has the content `{% include "pythonCard.html" %}` because Django substitutes the HTML of the included file for the template tag. Instead, we are going to test for `<div class="card">`. Since there are three cards, we should see three `<div>` tags with class `card`. Add a `count=3` to our `assertContains` test.

```
class HomepageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse('home'))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse('home'))
        self.assertTemplateUsed(response, 'home.html')

    def test_for_title(self):
        response = self.client.get(reverse('home'))
        self.assertContains(response, '<h1>Programming
        Languages</h1>')

    def test_for_cards(self):
        response = self.client.get(reverse("home"))
        self.assertContains(response, '<div class="card">',
        count=3)
```

▼ **Did you notice?**

> This last test has three different `self.assertContains` statement. In order for this test to pass, all three assertations must be true.

About Page

Just like the homepage, we are going to test for the title on the about page. Create a similar method called `test_for_tile` but substitute the name and HTML for our about page.

```
class AboutpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get('/about/')
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse('about'))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse('about'))
        self.assertTemplateUsed(response, 'about.html')

    def test_for_title(self):
        response = self.client.get(reverse('about'))
        self.assertContains(response, '<h1>About Page</h1>')
```

Since the about page is so simple, we will test for the contents of the text on the page.

```
class AboutpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get('/about/')
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse('about'))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse('about'))
        self.assertTemplateUsed(response, 'about.html')

    def test_for_title(self):
        response = self.client.get(reverse('about'))
        self.assertContains(response, '<h1>About Page</h1>')

    def test_for_content(self):
        response = self.client.get(reverse('about'))
        self.assertContains(response, '<p>This page is an
        exercise in learning how Django displays templates
        through URL pattern matching.</p>')
```

Programming Language Pages

Finally, we are going to test the contents of the programming language pages. For the sake of brevity, we are not going to check for the exact wording for the entire page (though it can be done). However, ever, we are going to check for the title (`<h1>` tag) and subtitles (`<h2>` tags). Create the `test_for_title` and `test_for_subtitles` methods.

```
class PythonpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get('/python/')
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse('python'))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse('python'))
        self.assertTemplateUsed(response, 'python.html')

    def test_for_title(self):
        response = self.client.get(reverse('python'))
        self.assertContains(response, '<h1>Python</h1>')

    def test_for_subtitles(self):
        response = self.client.get(reverse('python'))
        self.assertContains(response, '<h2>History</h2>')
        self.assertContains(response, '<h2>Uses</h2>')
```

Add the `test_for_title` and `test_for_subtitles` methods to the test classes for F# and Racket as well. You can copy the above methods and make the appropriate substitutions as needed.

Running the Tests

Once all the tests are written, open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black languages/tests.py
```

Run the test file with the command below.

```
python manage.py test
```

If all of your tests pass, you should see the following output:

```
Found 25 test(s).
System check identified no issues (0 silenced).
.........................
----------------------------------------------------------------------
------
Ran 25 tests in 0.048s

OK
```

Deactivate the virtual environment.

```
conda deactivate
```

▼ **Code**

> Your test file should look like this:

```
from django.test import SimpleTestCase
from django.urls import reverse


class HomepageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("home"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("home"))
        self.assertTemplateUsed(response, "home.html")

    def test_for_title(self):
        response = self.client.get(reverse("home"))
        self.assertContains(response, "<h1>Programming
        Languages</h1>")

    def test_for_cards(self):
        response = self.client.get(reverse("home"))
        self.assertContains(response, '<div class="card">',
        count=3)


class AboutpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
```

```
        response = self.client.get("/about/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("about"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("about"))
        self.assertTemplateUsed(response, "about.html")

    def test_for_title(self):
        response = self.client.get(reverse("about"))
        self.assertContains(response, "<h1>About
        Page</h1>")

    def test_for_content(self):
        response = self.client.get(reverse("about"))
        self.assertContains(
            response,
            "<p>This page is an exercise in learning how
        Django displays templates through URL pattern
        matching.</p>",
        )


class PythonpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/python/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("python"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("python"))
        self.assertTemplateUsed(response, "python.html")

    def test_for_title(self):
        response = self.client.get(reverse("python"))
        self.assertContains(response, "<h1>Python</h1>")

    def test_for_subtitles(self):
        response = self.client.get(reverse("python"))
        self.assertContains(response, "<h2>History</h2>")
        self.assertContains(response, "<h2>Uses</h2>")
```

```
class FsharppageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/fsharp/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("fsharp"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("fsharp"))
        self.assertTemplateUsed(response, "fsharp.html")

    def test_for_title(self):
        response = self.client.get(reverse("fsharp"))
        self.assertContains(response, "<h1>F#</h1>")

    def test_for_subtitles(self):
        response = self.client.get(reverse("fsharp"))
        self.assertContains(response, "<h2>History</h2>")
        self.assertContains(response, "<h2>Uses</h2>")


class RacketpageTests(SimpleTestCase):
    def test_url_by_pattern(self):
        response = self.client.get("/racket/")
        self.assertEqual(response.status_code, 200)

    def test_url_by_name(self):
        response = self.client.get(reverse("racket"))
        self.assertEqual(response.status_code, 200)

    def test_correct_template(self):
        response = self.client.get(reverse("racket"))
        self.assertTemplateUsed(response, "racket.html")

    def test_for_title(self):
        response = self.client.get(reverse("racket"))
        self.assertContains(response, "<h1>Racket</h1>")

    def test_for_subtitles(self):
        response = self.client.get(reverse("racket"))
        self.assertContains(response, "<h2>History</h2>")
        self.assertContains(response, "<h2>Uses</h2>")
```