MAKE SCHOOL

Testing with PyTest

WEB 1.1

Agenda

MAKE SCHOOL

- Learning Outcomes
- Why write tests?
- How do we write unit tests?
- Edge Cases
- **BREAK**
- Route Tests
- Lab Time
- Wrap-Up

Learning Outcomes

MAKE SCHOOL

By the end of today, you should be able to...

1. **Explain** why it is important to write unit tests.
2. **Write** unit tests for a Python function for multiple scenarios.
3. **Identify** edge cases for a given function.
4. **Write** route tests for Flask routes.

Videos

MAKE SCHOOL

Watch the Videos!

MAKE SCHOOL

Why write unit tests?

Reading (15 minutes)

MAKE SCHOOL

Read this article. Then, with a group of 3, answer the following questions:

1. Do you agree with the 5 reasons given for writing unit tests? Why or why not?
2. What other reasons can you think of for writing tests?
3. According to the article, how much test coverage is necessary? Do you agree with their assessment?

MAKE SCHOOL

How do we Unit Test?

Terminology

MAKE SCHOOL

An **automated test** is code that runs your program, and verifies that it works correctly.

A **unit test** verifies that a *single unit* of code (usually a single function) works correctly under a *single scenario*.

In this class, we'll be writing **route tests**, which can be slightly broader (sometimes they are testing multiple units).

How to Unit Test

MAKE SCHOOL

Writing a unit test is all about:

Expectation vs. Reality

What do we expect to happen when we call the function? What really happens? Do those two things match?

Expectation Vs. Reality

Expectation Vs. Reality

Expectation Vs. Reality

Expectation Vs. Reality

✔️ Test Passes!

What is a Unit Test?

MAKE SCHOOL

Any unit test is, fundamentally, answering the following question:

"Does my expectation of this code's output match the reality?"

To do this, the unit test must do the following:

1. Choose a ***scenario*** - that is, a set of inputs to the function.
2. Establish what we ***expect*** the function to produce for that scenario.
3. Call the function to determine its ***actual*** output for the scenario.
4. ***Compare*** the expected & actual results. If they match, the test passes! Otherwise, the test fails.

Example

Let's say I have a function that, given someone's name, returns a greeting:

```
def greet_by_name(name):
    """Returns a greeting to the given person."""
    greeting = "Hello, " + name + "!"
    return greeting
```

1. What *scenario* do we want to test? Let's pick one.
    a. name = "Ducky"
2. What do we *expect* as the output?
    a. "Hello, Ducky!"

Example

Here's what the test function might look like:

```
def test_greeting_Ducky(self):
    """Test for greet_by_name"""
    expected = 'Hello, Ducky!'

    actual = greet_by_name('Ducky')

    assert actual == expected
```

1. Choose a scenario - we're testing for name "Ducky"

2. Decide what the expected output is

3. Call the function under the scenario to get its actual output

4. Assert that they are the same. If the assertion is false, the test will fail.

What is an assertion?

MAKE SCHOOL

An **assertion** is a line of code that "asserts" that a certain condition is true.

The **assert** keyword, followed by a boolean condition, will perform an assertion.

If the condition evaluates to **False**, the test case will immediately fail and will not continue on to the next line.

If the condition evaluates to **True**, the test case will continue running. If all assertions pass, then the test case will pass.

Example

MAKE SCHOOL

Sometimes, we condense this code into one line:

```
def test_greeting_jeremy(self):
    """Test for greet_by_name"""
    assert greet_by_name('Jeremy') == 'Hello, Jeremy!'
```

Can you identify all 4 parts?

Demo

MAKE SCHOOL

Watch as your instructor demonstrates how to write and run a unit test with Pytest.

Activity (20 minutes)

MAKE SCHOOL

Clone the Flask Testing Starter code for today's activity.

With a partner (pair programming style), follow the instructions in the README for Part 1.

MAKE SCHOOL

Edge Cases

What is an Edge Case?

MAKE SCHOOL

An **edge case** is a scenario that only occurs when a parameter is at an extreme (maximum or minimum).

For parameters that are **numbers**, **1 and 0** are edge cases.

For parameters that are **strings**, **the empty string ''** is an edge case.

It is important to write unit tests for edge cases so that we can catch any possible bugs.

Activity (15 minutes)

MAKE SCHOOL

In the activity starter code, the last string function contains a bug: It throws an error when the given string is empty.

With your partner, write a test to catch the bug. Your test should fail.

Then, fix the bug by modifying the function code. Run your test again - it should now pass.

Then, write edge case tests for each of the other functions.

MAKE SCHOOL

Break - 10 min

Route Tests

Testing a Route

MAKE SCHOOL

Testing a route function is pretty similar, with a few differences:

1. The **scenario** is usually determined by the URL, as well as any data passed in via a form.
2. The **expectation** is what we expect to see rendered on the page.
3. The **actual result** is determined by performing a GET or POST action to the URL.
4. In addition to the text on the page, we also want to check that the **status code** of the request matches our expectation.

Example

MAKE SCHOOL

Here's an example of how it works:

```
def test_index():
    """Test that the index page shows "Hello, World!" """
    res = app.test_client().get('/')
    assert res.status_code == 200

    result_page_text = res.get_data(as_text=True)
    expected_page_text = "Hello, World!"
    assert expected_page_text == result_page_text
```

Can you identify the scenario, expectation, actual, and comparison?

Sending Form Data - GET

When we are testing a route that uses a form, we want to make sure that it will show the correct result for a given set of form inputs.

For a GET form, we can send data by modifying the URL query parameters.

E.g. the URL

```
/color_results?color=blue
```

will send the value of "blue" for the input field with name "color":

```
<input type="text" name="color">
```

Sending Form Data - GET

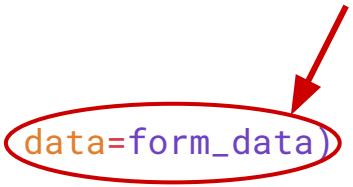
Here's an example:

```
def test_color_results_blue():
    """Test the /color_results page with input of 'blue'."""
    result = app.test_client().get('/color_results?color=blue')

    assert result.status_code == 200

    result_page_text = result.get_data(as_text=True)
    expected_page_text = 'Wow, blue is my favorite color, too!'
    assert expected_page_text in result_page_text
```

Sending Form Data - POST

For a POST form, we can't send the data via the URL. So, you can pass a **named parameter**, **data**, to the POST function.

```
def test_reverse_message_results_helloworld():
    form_data = {
        'message': 'Hello World'
    }
    res = app.test_client().post('/message_results', data=form_data)
    assert res.status_code == 200

    result_page_text = res.get_data(as_text=True)
    assert 'dlroW olleH' in result_page_text
```



Activity

MAKE SCHOOL

With your partner, complete the Part 3 challenges for the activity by completing the TODOs for the app tests. (Some of these were taken from Homework 2, so they should look familiar!)

There are a lot of test cases to fill out, but after a while they become really quick to write :).

MAKE SCHOOL

Lab Time

Homework

Instead of a full Homework 5, you'll submit the activity code from today's class.

Make sure you complete all of the TODOs!

This should give you more time to work on your final project!