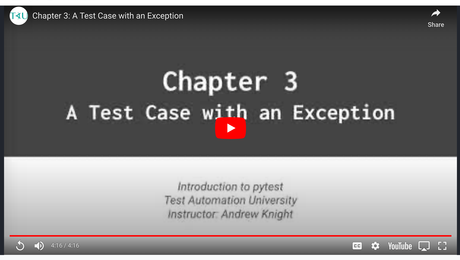


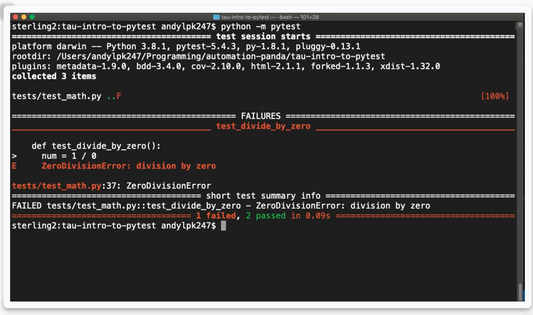
▼ Transcripted Summary

In the previous chapter, we learned that any raised exception that is not handled within a test case function will cause that test case to report a failure. So, this begs an important question. How do we verify that a piece of code successfully raises an exception?

For example, let's verify that dividing by zero raises an exception. In our "test_math.py" module, add a new test case function named "test_divide_by_zero" and implement it as follows:

```py
def test_divide_by_zero():
  num = 1 / 0
```

This single line alone should raise an exception. Let's see what happens when we run our new test case.



As expected, the test fails. Dividing by zero raises a "ZeroDivisionError" exception with the message, "division by zero". The math operation correctly raised the exception, but letting it rise outside of the test case function caused the test to fail. Thankfully, pytest safely catches any and all unhandled exceptions, performs any cleanup, and moves on to the next test case. Exceptions for one test case won't affect other tests.

So, how can we properly handle and verify exceptions inside of a test case? We could write our own try/except block, but thankfully, pytest actually provides a construct for handling expected exceptions: "pytest.raises".

pytest.raises

To use it, first import pytest into the test module.

```py
import pytest
```

Then add the following lines to our existing test case:

```py
def test_divide_by_zero():
  with pytest.raises(ZeroDivisionError) as e:
    num = 1 / 0

  assert 'division by zero' in str(e.value)
```

In Python, `with` is a special statement for automatically handling extra "enter" and "exit" logic for a caller. It is most commonly used for file input and output.

The "enter" logic opens the file, the body reads or writes, and the "exit" logic closes the file. For "pytest.raises", the "enter" logic makes the code catch any exceptions, and the "exit" logic asserts if the desired exception type was actually raised.

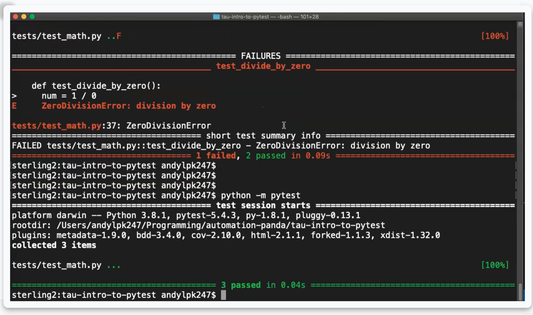
In our case, one divided by zero should raise a "ZeroDivisionError". So "pytest.raises" should catch the exception and keep running the test as if there were no problem.

Furthermore, "pytest.raises" looks for an exception of a specific *type*. If the steps within the statement's body do not raise the desired exception, then it will raise an assertion error to fail the test. Using "pytest.raises" makes the test code more concise and avoids repetitive try/except blocks.

In addition to verifying that the code raises an exception, we can also verify attributes of the raised exception. Notice how the "with" statement stores the exception object as a variable named 'e'. We can verify the exception's message with the following line:

```py
  assert 'division by zero' in str(e.value)
```

Let's run our updated tests.



Boom! This time it runs flawlessly. The test catches the expected exception and verifies its message.

Resources

- Assertions about expected exceptions
- Python errors and exceptions
- Python code style
- Python's "with" statement

Hide my answers

1. What is the purpose of "pytest.raises"?

- (•) To verify that the code under test raises expected exceptions.
- ( ) To catch exceptions raised by the pytest framework.
- ( ) To raise new exceptions when test failures happen.
- ( ) To change the order of test cases.

2. "pytest.raises" must be imported from the pytest module.

- (•) true
- ( ) false

3. "pytest.raises" does not look for exceptions based on type.

- ( ) true
- (•) false

4. Why does "pytest.raises" need to be used with a "with" statement?

- ( ) The "with" statement adds additional logs to the test report.
- ( ) False - "pytest.raises" can be used without a "with" statement.
- (•) The "with" statement executes additional "enter" and "exit" logic for handling raised exceptions.
- ( ) The "with" statement enables the test case to run functions from other modules.

5. How can "pytest.raises" store the raised exception object for further assertions?

- ( ) This is not possible.
- ( ) Include a variable name as the second argument in the "pytest.raises" call.
- (•) Add an "as" clause to the end of the "with" statement to store the exception into a variable.
- ( ) Use the "exception" keyword to reference the most recent exception.

Note: 100 credits is for successful completion on the first try; 50 credits for the second try, and 25 credits thereafter

Prev Chapter Next Chapter