September 6, 2020

#Python | #pytest | #blog

Selecting tests with pytest

Other pytest articles:

Why testing is important

Types of tests

Test driven Development

Hello, World!

Selecting tests with pytest

Testing HTTP client with pytest

Testing database with pytest

Advanced fixtures with pytest

Pytest plugins

Let's add another requirement for our normalize function - it will raise an exception if the number contains a letter, or if a plus sign is not at the beginning.

Now let's think a bit about the design of the application. Especially about error handling. I assume that our application should have its own exception called `NumberValidationException`. So the normalize function will raise this exception if a number has a letter.

In order to test if our code raises an exception there is a context manager in pytest that allows that - with `pytest.raises` and exception. And a test case with plus sign in the wrong place. Same here - our code should raise an exception.

Copy

```
def test_number_with_letters():
    number = '+31abc5551234' # 1
    with pytest.raises(NumberValidationException): # 2
        normalize(number) # 3


def test_number_misplaces_plus():
    number = '+31+5551234'
    with pytest.raises(NumberValidationException):
        normalize(number)
```

1. Input number that we expect to raise an exception
2. we instruct pytest to check that our code will raise an exception
3. Code that raises the exception internally

The tests fail now as our current implementation doesn't raises anything. Before fixing that, it's possible for now just skip this test. For that the test case may be decorated with `@pytest.mark.skip`

Copy

```
@pytest.mark.skip
def test_number_with_letters():
    ...
```

The output of pytest will show that we are skipping tests

Copy

```
...
test_normalize.py ...s.
...
=== 4 passed, 1 skipped in 0.04s ===
```

There is a variation of skip - conditional skip. Let's skip the test if a environment variable is set. It's useful when, for example, we skip all integration tests that touch database in an environment without any database available:

Copy

```
@pytest.mark.skipif(os.getenv('SKIP_TEST'), # 1
                    reason='skipping if environment variable is set') # 2
def test_number_with_letters():
    ...
```

1. A skip condition, test is skipped if True
2. Optional explanation for skipping the test

We know that the test fails - so skipping might be a suboptimal option. Another decorator in pytest - xfail can be used to mark failing cases. It can serve as a reminder that there is a bug in our code that needs to be fixed later.

Copy

```
@pytest.mark.xfail
def test_number_with_letters():
    ...
```

Output:

Copy

```
test_normalize.py ...X.
=== 4 passed, 1 xpassed in 0.04s ===
```

Sometimes we want to run a single test of a specific subset of test cases. To run a single test we can pass its name (or it's part) with a -k option.

Copy

```
~ pytest . -k spaces
==== test session starts ===
collected 5 items / 4 deselected / 1 selected

test_normalize.py .
```

```
=== 1 passed, 4 deselected in 0.04s ===
```

If we want to create a subset of test cases we can mark them with a mark decorator:

Copy

```
@pytest.mark.exceptions
def test_number_with_letters():
    ...
```

Then we call pytest with -m option - `pytest . -m exceptions`

Can you finish the normalize function so all the tests are green?

In this unit you've learned how to select tests, use marks and test exceptions. In the next one you'll learn how to test HTTP client and how to use pytest fixtures.

Similar articles:

- Advanced fixtures with pytest
- Hello, World!
- Pytest plugins
- Test driven Development
- Testing HTTP client with pytest