September 6, 2020

#Python | #pytest | #blog

Hello, World!

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

In this course, we will be working on a mobile phone number validation application.

The application:

- Accepts a number as input

- For every number in the list
  - Normalize the number
  - Check cache if this number was validated before
  - If it's not in cache call external service REST API to validate the number
  - print the normalized number and the result of validation

Let's start with the Normalize step. Numbers can contain hyphens, spaces, brackets and can start with a plus sign. Normalize function should leave plus sign and remove all others.

First we need to install pytest. Most obvious option is to use pip.

Copy

```
pip install pytest
```

Let's write first unit test cases describing these requirements. All of them will be inside `test_normalize.py` file.

First test case for spaces. Test case is just a function:

Copy

```
from normalize import normalize

def test_spaces():
    number = '555 12 34' # input number
    normalized_number = normalize(number) # our code we will write later
    assert normalized_number == '5551234' # expected result
```

As an input we have a number with spaces, and as an output we assume that the number won't have them. Normalized number is a result of our normalized function. And we put an assertion at the end of the test that we get what we expect.

Basically assert is the main part of the test - we compare expected and actual results of some calculation and if they are not equal assert fail with an exception and pytest will show that test as failed. In more complex cases than that one it's wise to have only one assertion in the end of the test case.

Next test case for hyphens:

Copy

```
def test_hyphens():
    number = '555-12-34'
```

```
    normalized_number = normalize(number)
    assert normalized_number == '5551234'
```

Then for brackets:

Copy

```
def test_brackets():
    number = '31(0)5551234'
    normalized_number = normalize(number)
    assert normalized_number == '3105551234'
```

Now let run the tests. Pytest will look for files starting with test in current directory and its subdirectories. It will find the functions starting with `test_` and recognize them as an individual test cases.

Copy

```
~ pytest .

=== ERRORS ===
___ ERROR collecting test_normalize.py ___
ImportError while importing test module 'testing_python_with_pytest/test_normalize.py'
Hint: make sure your test modules/packages have valid Python names.
Traceback:
test_normalize.py:3: in <module>
    from normalize import normalize
E   ModuleNotFoundError: No module named 'normalize'
=== short test summary info ===
ERROR test_normalize.py
!!! Interrupted: 1 error during collection !!!
=== 1 error in 0.24s ===
```

Now we are on a RED step of TDD - write a test that doesn't work.

Let's create dummy function in `normalize.py`

Copy

```
def normalize(number):
    return '1123455'
```

The output has changed - now we see that all tests failed (red Fs) and in the FAILURES section there is a description of every failure.

Copy

```
test_normalize.py FFF
```

```
=== FAILURES ===
___ test_spaces  ___

    def test_spaces():
        number = '555 12 34'
        normalized_number = normalize(number)
>       assert normalized_number == '5551234'
E       AssertionError: assert '1123455' == '5551234'
E         - 5551234
E         + 1123455
....
test_normalize.py:21: AssertionError
=== short test summary info ===
FAILED test_normalize.py::test_spaces - AssertionError: assert '1123455' == '5551234'
FAILED test_normalize.py::test_hyphens - AssertionError: assert '1123455' == '5551234'
FAILED test_normalize.py::test_brackets - AssertionError: assert '1123455' == '3105551
=== 3 failed in 0.17s ===
```

Now I'm going to implement a normalize partially

Copy

```
import re

def normalize(number):
    return re.sub(r'[ \-()]', '', number)
```

Now all the tests are green

Copy

```
pytest .
==== test session starts ===
collected 3 items

test_normalize.py ...

=== 3 passed in 0.04s ===
```

There seems to be some duplication - we have 3 cases that does the same. Pytest has a parametrization feature to combine them in one case to avoid duplication

Copy

```
import pytest

@pytest.mark.parametrize("test_input,expected", # 1
```

```
                          [('555 12 34', '5551234'), # 2
                           ('31(0)5551234', '3105551234'),
                           ('31(0)5551234', '3105551234')])
def test_normalize(test_input, expected): # 3
    assert expected == normalize(test_input)
```

1. parametrize decorator accepts comma separated list of parameters - input number and expected normalized version as a first argument
2. second argument is a list of tuples, representing cases, in each tuple we have an input number and the expected result
3. test case function definition should contain 2 input arguments - same as we have in #1

In this unit you've learned how to launch pytest and create test cases. In the next unit we'll cover how to select test and test exceptions.

Similar articles:

- Advanced fixtures with pytest
- Pytest plugins
- Selecting tests with pytest
- Test driven Development
- Testing HTTP client with pytest