09

Do as I did: Pytest

It's time for you to implement what you saw in class!

In this lesson, you were introduced to one of the most widely used Python testing frameworks, Pytest. With it, we "translated" the test previously created into the Pytest model and at the same time advanced in the logic of creating tests using the agile ***Given-When-Then*** method . Finally, we created a new test from scratch using ***Given-When-Then*** within the Pytest environment.

So, have you gotten your hands dirty yet?

It's time for you to create a test with Pytest!

If you have any questions, check the progress of your project by clicking on **Instructor's Opinion** .

Instructor's opinion

Check the progress of your project according to the instructions:

1) We start by learning a little about the Pytest testing framework and that its use is the most recommended for creating unit and automated tests.

2) We install Pytest using the following command in the IDE terminal:

```
pip install pytest==7.1.2
```

COPY CODE

3) We continue with good practice in using the virtual environment and create the file `requirements.txt` so that all packages used are registered within it and facilitate future reproduction of the project on other machines.

```
pip freeze > requirements.txt
```

COPY CODE

4) Before using Pytest, we need to organize the directory structure that we have in the project and create the file where the tests will be stored. We start by creating the directory `tests` inside the main directory.

5) Then we create the file `test_bytebank.py` inside the directory `tests` .

> **IMPORTANT** : the naming of directories and files interferes with the functioning of Pytest. In order for it to recognize the tests written within the file `test_bytebank.py` , the name of the directory must be `tests` and the prefix of the test file itself must always be **test_** .

6) With the test environment ready, we begin to "translate" the test done in `main.py` to the Pytest standard within the file `test_bytebank.py` .

7) We learned the importance of test naming. Every test name must begin with the prefix **test_** and all must be as verbose as possible, that is, they must explicitly explain what they are testing.

8) Next we saw that there is an agile test construction methodology that can help us with the logic of the test code, the *Given-When-Then* methodology .

9) We build the first test inside Pytest:

```
from codigo.bytebank import Funcionario

class TestClass:
    def test_quando_idade_recebe_13_03_2000_deve_retornar_22(self):
        entrada = '13/03/2000' # Given-Contexto
        esperado = 22

        funcionario_teste = Funcionario('Teste', entrada, 1111)
        resultado = funcionario_teste.idade() # When-ação
        assert resultado == esperado  # Then-desfecho
```

COPY CODE

10) Right after the "translation" of this test that we had done in the file `main.py` , a new demand arises for the creation of a method within the class `Funcionario` . The new method will be called `sobrenome()` and must return the surname of the employees.

11) We modified the file `bytebank.py` and helped Dominique by creating the new method:

```
def sobrenome(self):
        nome_completo = self.nome.strip()
        nome_quebrado = nome_completo.split(' ')
        return nome_quebrado[-1]
```

COPY CODE

12) After creating this method we realized that we needed to test it too, so we created a new test `test_bytebank.py` just for it.

```
def test_quando_sobrenome_recebe_Lucas_Carvalho_deve_retornar_Carvalho(
entrada = ' Lucas Carvalho ' # Given
esperado = 'Carvalho'

lucas = Funcionario(entrada, '11/11/2000', 1111)
resultado = lucas.sobrenome() # When

assert resultado == esperado
```

COPY CODE