08

Do as I did: Exceptions and Marks

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

In this lesson, we learned what *exceptions* are in Python and how to create tests that take into account their use in the code. We also learned how to use markers to optimize and organize test execution.

So, have you gotten your hands dirty yet?

It's time to understand how to work with *exceptions* and organize your tests with *markers* !

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

Instructor's opinion

1) Dominique received the demand that one of the class's functionalities `Funcionario` , the method, `calcular_bonus()` must return an *exception* if the condition pre-established by its logic is not met;

2) We turn our attention to the method `calcular_bonus()` and understand that its purpose is to check if 10% of an employee's salary is below R$1000.00, returning a bonus equivalent to 10% of the salary.

Otherwise it should return 0. Before we continue and change the method to include an *exception* , we need to create a test for `calcular_bonus()` ;

```
def test_quando_calcular_bonus_recebe_1000_deve_retornar_100(self):
    entrada = 1000  # given
    esperado = 100

    funconario_teste = Funcionario('teste', '11/11/2000', entrada)
    resultado = funconario_teste.calcular_bonus() # when

    assert resultado == esperado  # then
```

COPY CODE

3) We briefly learned what an *exception* is in Python and changed the method `calcular_bonus()` to include a custom *exception ;*

```
def calcular_bonus(self):
    valor = self._salario * 0.1
    if valor > 1000:
        raise Exception('O salário é muito alto para receber um bônus
    return valor
```

COPY CODE

4) Now, we need a test that makes sure that if an employee does not meet the requirements to get a bonus, the method `calcular_bonus()` will return the custom *exception* we created. We start by importing Pytest into the file `test_bytebank.py` ;

```
import pytest
```

COPY CODE

5) Next, we create the test that handles an *exception* in `calcular_bonus()` ;

```
def test_quando_calcular_bonus_recebe_100000000_deve_retornar_exception(s
        with pytest.raises(Exception):
            entrada = 100000000  # given

            funconario_teste = Funcionario('teste', '11/11/2000', entrada
            resultado = funconario_teste.calcular_bonus()  # when

            assert resultado  # then
```

COPY CODE

6) We realized that we can have more than one test for the same piece of code and one test does not necessarily invalidate the other;

7) Next, we learned that we can better organize our tests using a Pytest tool called **Markers** . We can categorize our tests using custom *tags for each one;*

8) Let's create a custom *tag* for the tests that involve the method `calcular_bonus()` . To do this, we will import the markers library at the beginning of the file `test_bytebank.py` ;

```
    from pytest import mark
```

COPY CODE

9) We can create these *tags* using decorators, one line before each test;

```
   @mark.calcular_bonus
    def test_quando_calcular_bonus_recebe_1000_deve_retornar_100(self):
        entrada = 1000  # given
        esperado = 100

        funconario_teste = Funcionario('teste', '11/11/2000', entrada)
        resultado = funconario_teste.calcular_bonus() # when

        assert resultado == esperado  # then

   @mark.calcular_bonus
    def test_quando_calcular_bonus_recebe_100000000_deve_retornar_excepti
        with pytest.raises(Exception):
            entrada = 100000000  # given

            funconario_teste = Funcionario('teste', '11/11/2000', entrada
            resultado = funconario_teste.calcular_bonus()  # when

            assert resultado  # then
```

COPY CODE

10) We have just created the mark `calcular_bonus` , however, when running Pytest in the terminal/command prompt, we get a warning from Pytest. This happens because the mark we are using is a custom mark and is not registered in the Pytest settings.

11) We can fix this by creating a Pytest configuration file called `pytest.ini` . In this file, we can change some of Pytest's default settings, including the ability to insert custom marks;

```
    [pytest]
markers =
    calcular_bonus: Teste para o metodo calcular_bonus
```

COPY CODE