**08 - Faça como eu fiz: Exceptions e Marks**

Chegou a hora de você executar o que foi visto na aula!

Nessa aula, aprendemos o que são *exceptions* em Python e como criar testes que levem em consideração a sua utilização no código. Também aprendemos como utilizar os markers para otimizar e organizar a execução dos testes.

E aí? Já colocou a mão na massa?

Chegou a hora de entender como trabalhar com *exception* e organizar os seus testes com *markers*!

Caso tenha dúvidas, confira o andamento do seu projeto clicando na **Opinião do Instrutor**

VER OPINIÃO DO INSTRUTOR

**Opinião do instrutor**

1) A Dominique recebeu a demanda de que uma das funcionalidades da classe Funcionario, o método calcular\_bonus() deve retornar uma *exception* caso a condição pré-estabelecida pelo sua lógica não seja atendida;

2) Voltamos nossa atenção para o método calcular\_bonus() e entendemos que o propósito dele é verificar se 10% do salário de um funcionário está abaixo de R$1000,00, retornando um bônus equivalente a 10% do salário.

Caso contrário deve retornar 0. Antes da dar continuidade e alterar o método para incluir uma *exception*, precisamos criar um teste para 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 # thenCOPIAR CÓDIGO

3) Aprendemos brevemente o que é uma *exception* em Python e alteramos o método calcular\_bonus() para incluir uma *exception* personalizada;

def calcular\_bonus(self):

valor = self.\_salario \* 0.1

if valor > 1000:

raise Exception('O salário é muito alto para receber um bônus')

return valorCOPIAR CÓDIGO

4) Agora, precisamos de um teste que nos certifique que se um funcionário não possuir os requisitos para obter um bônus, o método calcular\_bonus() irá retornar a *exception* personalizada que criamos. Começamos importando o Pytest dentro do arquivo test\_bytebank.py;

import pytestCOPIAR CÓDIGO

5) Em seguida, criamos o teste que lida com uma *exception* em calcular\_bonus();

def test\_quando\_calcular\_bonus\_recebe\_100000000\_deve\_retornar\_exception(self):

with pytest.raises(Exception):

entrada = 100000000 # given

funconario\_teste = Funcionario('teste', '11/11/2000', entrada)

resultado = funconario\_teste.calcular\_bonus() # when

assert resultado # thenCOPIAR CÓDIGO

6) Percebemos que podemos ter mais de um teste para o mesmo trecho de código e não necessariamente um teste invalida o outro;

7) Em seguida, aprendemos que podemos organizar melhor os nossos testes utilizando uma ferramenta do Pytest chamada **Markers**. Podemos categorizar nossos testes utilizando *tags* personalizadas para cada um;

8) Vamos criar uma *tag* personalizada para os testes que envolvem o método calcular\_bonus(). Para isso, vamos importar a biblioteca de markers no inicio do arquivo test\_bytebank.py;

from pytest import markCOPIAR CÓDIGO

9) Podemos criar essas *tags* através da utilização de decoradores, uma linha antes de cada teste;

@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\_exception(self):

with pytest.raises(Exception):

entrada = 100000000 # given

funconario\_teste = Funcionario('teste', '11/11/2000', entrada)

resultado = funconario\_teste.calcular\_bonus() # when

assert resultado # thenCOPIAR CÓDIGO

10) Acabamos de criar o mark calcular\_bonus, porém, ao executar o Pytest no terminal/prompt de comando, obtemos um aviso do Pytest. Isso acontece porque o mark que estamos utilizando se trata de um mark personalizado e não está registrado nas configurações do Pytest.

11) Podemos consertar isso criando um arquivo de configurações do Pytest chamado pytest.ini. Nesse arquivo, podemos alterar algumas das configurações padrão do Pytest, incluindo a possibilidade da inserção de marks personalizados;

[pytest]

markers =

calcular\_bonus: Teste para o metodo calcular\_bonus