05 – Organizando testes com Markers

Nesta aula, vamos estudar algumas das funcionalidades do Pytest que nos permitem explorar a execução dos testes.

Sabemos que o comando pytest -v roda todos os testes, no entanto, a Dominique gostaria de verificar o resultado de apenas um teste específico. Anteriormente, reparamos que há uma seta verde à esquerda do nome de cada teste. Ao clicar sobre ela com o botão direito, podemos selecionar "*Run*" para executar o teste individualmente (com uma interface diferente), porém essa solução dependente da IDE. Seria interessante ter um recurso que podemos executar em qualquer tipo de ambiente, pelo terminal.

Uma opção para selecionar determinado teste e executá-lo é usar a ***flag*** -k, que realizará a filtragem a partir de uma palavra do nome do método. Por exemplo, para executar apenas o teste test\_quando\_idade\_recebe\_13\_03\_2000\_deve\_retornar\_22, vamos rodar o comando pytest -k idade, pois este é o único teste com a palavra "idade".

No resultado, seremos informados que 1 teste passou e 4 foram desselecionados. Isto é, eles não falharam, apenas não os executamos. Para obter o retorno de modo verboso, podemos usar mais de uma *flag* ao mesmo tempo: pytest -v -k idade.

Por enquanto, nosso projeto tem poucos testes, então conseguimos lembrar os nomes de cada um deles, mas a tendência é que esse arquivo fique mais extenso. Dependendo do tamanho do projeto, talvez até tenhamos mais de um arquivo de testes. Com o tempo, filtrar testes pelo nome se tornará uma prática inviável, pois não teremos certeza se uma palavra está presente apenas em um teste. Como os nomes devem ser verbosos e explicativos, é provável que muitas palavras se repitam. No nosso caso, todos os testes repetem "quando" e "deve", por exemplo.

Então, outra opção para selecionar a execução de testes específicos é por meio dos ***marks***! Para entendermos o que são os *marks*, primeiramente vamos importá-los no início do arquivo:

from codigo.bytebank import Funcionario

import pytest

from pytest import mark

Vamos supor que queremos executar apenas os testes referentes ao método calcular\_bonus — no caso, os dois últimos testes que desenvolvemos neste curso. Para filtrá-los, criaremos um ***decorator*** chamado calcular\_bonus, que será um *mark*:

@mark.calcular\_bonus

def test\_quando\_calcular\_bonus\_recebe\_1000\_deve\_retornar\_100(self):

entrada = 1000 # given

esperado = 100

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

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

assert resultado == esperado # then

**Em resumo, o *mark* é uma *tag* que associamos a determinados testes.** Ele tem o formato de um *decorator* do Python: inicia-se com o símbolo de arroba, seguido da nomenclatura mark, um ponto e o nome.

Na segunda linha do arquivo, estamos importando o pytest. Na linha seguinte, importamos o mark de pytest. Poderíamos usar o *mark* com `@pytest.mark.calcular\_bonus`, mas optamos pela segunda importação para manter o código mais organizado.

Vamos usar a mesma *tag* no outro teste referente ao calcular\_bonus:

@mark.calcular\_bonus

def test\_quando\_calcular\_bonus\_recebe\_1000\_deve\_retornar\_100(self):

entrada = 1000 # given

esperado = 100

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

resultado = funcionario\_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

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

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

assert resultado # then

Para executar apenas esses dois testes, basta chamar a *tag* calcular\_bonus com a *flag* -m (de *mark*):

pytest -v -m calcular\_bonus

Analisando o resultado no terminal, notaremos que 2 testes foram executados e passaram; e 3 testes foram desselecionados, como esperado.

Além disso, teremos dois *warnings* (avisos), indicando que pytest.mark.calcular\_bonus é um *mark* **fora dos padrões**. O Pytest não sabe distinguir se é um erro de digitação (*typo*) ou um *mark* personalizado.

Os *marks* não são apenas *tags* que podemos utilizar como referências para certos testes, eles têm outras utilidades e seguem alguns padrões. Para verificar quais tipos de *marks* existem no Pytest, vamos executar pytest --markers no terminal. Dessa forma, teremos a documentação de todos os *markers* que podemos utilizar. Por exemplo, ao colocar `@pytest.mark.skip` antes de um teste, executaremos todos os testes e "pularemos" apenas este. Vamos testá-lo.

Primeiramente, vamos comentar os *markers* `@mark.calcular\_bonusque inserimos nas linhas 35 e 45. Em seguida, vamos incluir@mark.skipna linha imediatamente antes do teste de decréscimo de salário. No terminal, rodaremospytest -v` para executar todos os testes. Como resultado, notaremos que foram executados 4 testes e 1 foi ignorado ou "pulado" (*skipped*).

Portanto, os *marks* oferecem a possibilidade de organizarmos melhor quais testes queremos executar ou até mesmo a forma como queremos executá-los. Vamos estudar mais alguns exemplos, executando pytest --markers novamente.

Além do `@pytest.mark.skipque "pula" um teste, temos o@pytest.mark.skipifque "pulará" o teste se uma condição for obedecida. Existe também o@pytest.mark.xfail` que determina que um teste deve falhar, caso certa condição seja atendida. São diversos *markers* à disposição, é interessante consultarmos [a documentação do Pytest](https://cursos.alura.com.br/course/python-tdd-explorando-testes-unitarios/task/docs.pytest.org/en/7.1.x/how-to/mark.html) para encontrar formas de personalizar nossos códigos conforme nossas necessidades.

Por ora, vamos voltar ao projeto da Dominique. Removeremos o `@mark.skipque incluímos na linha 24 e descomentar os@mark.calcular\_bonus` das linhas 35 e 45.

Constatamos que, ao rodar pytest -v -m calcular\_bonus, receberemos 2 *warnings*. Para retirar esses avisos, precisamos gerar um arquivo, onde incluiremos esses *marks* personalizados. Clicaremos com o botão direito sobre a pasta bytebank-tdd no painel à esquerda, selecionaremos "New > File" e nomearemos o novo arquivo pytest.ini.

Este é um arquivo de configuração do Pytest e ele precede a configuração original do Pytest. É preciso tomar cuidado com o que colocamos nele, pois, uma vez o Pytest é executado, ele dará prioridade ao que está neste arquivo e, depois, considerará a configuração padrão do Pytest.

Na primeira linha, digitaremos [pytest]. A seguir, vamos incluir o nome do nosso *marker* personalizado, bem como uma descrição:

[pytest]

markers =

calcular\_bonus: Teste para o metodo calcular\_bonus

Vamos salvar. Em seguida, executaremos pytest -v -m calcular\_bonus no terminal mais uma vez, para rodar apenas os testes marcados com a *tag* calcular\_bonus. Dessa vez, 2 testes passarão, 3 serão desselecionados e não teremos *warnings*, pois acrescentamos os *markers* personalizados no arquivo pytest.ini.