**09 - Faça como eu fiz: Pytest**

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

Nessa aula, você foi apresentado a um dos frameworks de testes mais utilizados do Python, o Pytest. Com ele nós “traduzimos” o teste feito anteriormente para o modelo do Pytest e ao mesmo tempo avançamos na lógica de criação de testes através do método ágil ***Given-When-Then***. Por fim, criamos um novo teste do zero utilizando o ***Given-When-Then*** dentro do ambiente do Pytest.

E aí? Já colocou a mão na massa?

Chegou a hora de você criar um teste com o Pytest!

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

VER OPINIÃO DO INSTRUTOR

**Opinião do instrutor**

Confira o andamento do seu projeto de acordo com o que foi instruído:

1) Começamos aprendendo um pouco sobre o framework de testes Pytest e que a sua utilização é a mais recomendada para a criação de testes unitários e automatizados.

2) Instalamos o Pytest através do seguinte comando no terminal da IDE:

pip install pytest==7.1.2COPIAR CÓDIGO

3) Seguimos com uma boa prática de utilização do ambiente virtual e criamos o arquivo requirements.txt para que todos os pacotes utilizados sejam registrados dentro dele e facilitar a reprodução futura do projeto em outras máquinas.

pip freeze > requirements.txtCOPIAR CÓDIGO

4) Antes da utilização do Pytest, precisamos organizar a estrutura de diretórios que temos no projeto e criar o arquivo onde os testes vão ficar. Começamos por criar o diretório tests dentro do diretório principal.

5) Em seguida, criamos o arquivo test\_bytebank.py dentro do diretório tests.

**IMPORTANTE**: a nomenclatura de diretórios e arquivos interfere no funcionamento do Pytest. Para que ele reconheça os testes escritos dentro do arquivo test\_bytebank.py, o nome do diretório deve ser tests e o prefixo do próprio arquivo de testes sempre deve ser **test\_**.

6) Com o ambiente de testes pronto, começamos a “traduzir” o teste feito em main.py para o padrão do Pytest dentro do arquivo test\_bytebank.py.

7) Aprendemos a importância da nomenclatura dos testes. Todo nome de teste deve começar com o prefixo **test\_** e todos devem ser o mais verbosos o possível, ou seja, devem explicar de forma explícita o que estão testando.

8) Em seguida vimos que existe uma metodologia ágil de construção de testes que pode nos ajudar na lógica do código de teste, a metodologia ***Given-When-Then***.

9) Construímos o primeiro teste dentro do 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-desfechoCOPIAR CÓDIGO

10) Logo após a “tradução” desse teste que havíamos feito no arquivo main.py, surge uma nova demanda para a criação de um método dentro da classe Funcionario. O novo método se chamará sobrenome() e deve retornar o sobrenome dos funcionários.

11) Modificamos o arquivo bytebank.py e ajudamos a Dominique criando o novo método:

def sobrenome(self):

nome\_completo = self.nome.strip()

nome\_quebrado = nome\_completo.split(' ')

return nome\_quebrado[-1]COPIAR CÓDIGO

12) Após a criação desse método percebemos que precisamos testá-lo também, então criamos um novo teste em test\_bytebank.py apenas para ele.

def test\_quando\_sobrenome\_recebe\_Lucas\_Carvalho\_deve\_retornar\_Carvalho(self):

entrada = ' Lucas Carvalho ' # Given

esperado = 'Carvalho'

lucas = Funcionario(entrada, '11/11/2000', 1111)

resultado = lucas.sobrenome() # When

assert resultado == esperado