# Exercício: Testes com Pytest e CI com GitHub Actions

Este exercício tem como objetivo apresentar exemplos simples de uso do framework Pytest para testes unitários em Python e a configuração de um pipeline de integração contínua (CI) utilizando GitHub Actions.

## Exemplo 1: Operações matemáticas simples

1. Crie um arquivo chamado `math\_utils.py` com as seguintes funções:

def add(a, b):  
 return a + b  
  
def subtract(a, b):  
 return a - b

2. Crie um arquivo `test\_math\_utils.py` com os testes:

from math\_utils import add, subtract  
  
def test\_add():  
 assert add(2, 3) == 5  
  
def test\_subtract():  
 assert subtract(5, 2) == 3

## Exemplo 2: Multiplicação com verificação de tipo

1. Altere `math\_utils.py` para adicionar a função:

def multiply(a, b):  
 if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):  
 raise ValueError("Both arguments must be numbers")  
 return a \* b

2. Adicione o teste correspondente em `test\_math\_utils.py`:

def test\_multiply():  
 assert multiply(4, 3) == 12  
  
def test\_multiply\_invalid\_type():  
 try:  
 multiply("a", 3)  
 except ValueError:  
 assert True

## Configuração do GitHub Actions

Crie o arquivo `.github/workflows/python-app.yml` com o seguinte conteúdo:

name: Python application  
  
on: [push, pull\_request]  
  
jobs:  
 build:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
 - name: Set up Python  
 uses: actions/setup-python@v4  
 with:  
 python-version: '3.10'  
 - name: Install dependencies  
 run: |  
 python -m pip install --upgrade pip  
 pip install pytest  
 - name: Run tests  
 run: |  
 pytest

## Exemplo 3: Projeto com Godot-Python (gdpython)

Este exemplo demonstra como configurar um projeto simples usando o módulo Python no Godot (via Godot-Python), com testes unitários automatizados e deploy de um pacote com GitHub Actions.

1. Crie o arquivo `player.py` com a lógica básica:

class Player:  
 def \_\_init\_\_(self, name, hp):  
 self.name = name  
 self.hp = hp  
  
 def take\_damage(self, amount):  
 self.hp -= amount  
 if self.hp < 0:  
 self.hp = 0

2. Crie o arquivo `test\_player.py` para testar o comportamento:

from player import Player  
  
def test\_player\_damage():  
 player = Player("Hero", 100)  
 player.take\_damage(30)  
 assert player.hp == 70  
  
def test\_player\_hp\_cannot\_be\_negative():  
 player = Player("Hero", 10)  
 player.take\_damage(50)  
 assert player.hp == 0

3. Crie um `setup.py` básico para empacotar o código:

from setuptools import setup, find\_packages  
  
setup(  
 name="godot\_player",  
 version="0.1",  
 packages=find\_packages(),  
 install\_requires=[],  
)

4. Crie o workflow `.github/workflows/deploy.yml`:

name: Test and Deploy Python Godot Module  
  
on:  
 push:  
 branches: [main]  
  
jobs:  
 build:  
 runs-on: ubuntu-latest  
  
 steps:  
 - uses: actions/checkout@v3  
  
 - name: Set up Python  
 uses: actions/setup-python@v4  
 with:  
 python-version: '3.10'  
  
 - name: Install dependencies  
 run: |  
 pip install pytest build  
  
 - name: Run tests  
 run: pytest  
  
 - name: Build package  
 run: python -m build  
  
 - name: Upload artifact  
 uses: actions/upload-artifact@v3  
 with:  
 name: godot\_player\_package  
 path: dist/\*

Este fluxo realiza os testes com `pytest`, empacota o projeto com `build`, e armazena o pacote na aba 'Artifacts' do GitHub Actions, pronto para ser instalado com pip.