

# Desenvolvimento Aberto



**Testes automatizados**

Igor dos Santos Montagner ( [igorsm1@insper.edu.br](mailto:igorsm1@insper.edu.br) )

# Projeto profissional

- Qualidade de código
  - Linting - formatação e erros comuns
- Documentação
  - Usuário
  - Desenvolvedor

# Projeto profissional

- Qualidade de código
  - Linting - formatação e erros comuns
  - **O código funciona?**
- Documentação
  - Usuário
  - Desenvolvedor

# Meu programa funciona?

- Sob quais condições?
- Em quais plataformas?
- Quais operações são suportadas?
- Consigo conferir o resultado de uma execução? Se sim, existe um valor de referência?

# Testes automatizados

**Ideia:** escrever um programa que verifica se um outro programa responde como esperado

- Definir situações a serem testadas ...
- e o resultado esperado em cada situação

# Testes automatizados

## Não ajudam:

- a revelar novos bugs
- a garantir que um software é livre de bugs

## Ajudam

- a evitar que bugs descobertos voltem
- a evitar que mudanças não intencionais quebrem código que estava funcionando.
- a documentar em quais situações o software funciona.

# Testes automatizados

1. Testes de unidade
2. Testes de integração
3. Testes de interface de usuário

# Testes unitários

**Ideia:** dada uma função, verificar se ela devolve o valor esperado para um certo conjunto de parâmetros.

- Testa as funções de maneira **isolada**
- **Cobertura:** porcentagem das linhas de código que é executada durante os testes de unidade.
- Serve como documentação da função



# Testes unitários - pytest

```
# content of test_sample.py
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
```

That's it. You can now execute the test function:

```
$ pytest
===== test session starts =====
platform linux -- Python 3.x.y, pytest-3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile:
collected 1 item

test_sample.py F [100%]

===== FAILURES =====
_____ test_answer _____

    def test_answer():
>         assert func(3) == 5
E         assert 4 == 5
E         + where 4 = func(3)

test_sample.py:5: AssertionError
===== 1 failed in 0.12 seconds =====
```

# Testes de integração

**Ideia:** dados um conjunto de classes com interdependências, verificar se elas funcionam bem **em conjunto**.

- Testa interação entre em objetos
- Possibilidade de criar *mocks*, que são objetos falsos feitos para simular a interação entre vários objetos.

# Testes de interface de usuário

**Ideia:** simula ações do usuário (cliques, entrada de dados, etc) e confere se a saída esperada é mostrada na tela

- Menos específico possível
- Mais fiel ao uso real de um usuário

# Testes de interface de usuário

## Selenium

Permite fazer scripts que interagem com uma página web, realizando entrada de dados, rolagem de tela e cliques. Cada `assert` pode ser feito com o conteúdo de um objeto da página.

# O quê eu preciso testar?

# O quê eu preciso testar?

**Ninguém sabe....**

# Atividade prática: Testado e aprovado



**Objetivo:** Primeira experiência com testes automatizado de código.

```
"metadata": {"url": "repo-servidor-desagios", "group": ["igual", "6"]}
```

# Atividade prática: Testado e aprovado

1. Criar testes de interface de usuário usando *Selenium*
  - [tutorial](#)
  - ideia é reproduzir os mesmos testes apresentados no manual do usuário
  - testar usando Firefox e Chrome
2. Criar testes para a função que executa a função submetida pelo usuário ( `lambda_handler` ) usando *pytest*
  - [tutorial](#)
  - ideia é verificar se alguns programas bem e mal formatados dão o resultado esperado

Os scripts devem ser colocados dentro da pasta *test* nos fontes de vocês.