Testes de Unidade

Aula 01 — O teste que o print não fez

UniSenac Centro Universitário RS
Senac
Fecomércio
IFEP
Sesc
Sindicatos Empresariais

A UC3 numa tela

Indicador	O que vocês vão mostrar que sabem fazer
1	Usar diferentes formas de testar, de acordo com o que o sistema faz
2	Tabular o resultado do que foi testado
3	Corrigir e ajustar, a partir do que o teste mostrou

- **Aulas 01 a 06** — testes, com o livro do Maurício Aniche traduzido para Python
- **Aulas 07 a 09** — Git, manutenção e a entrega da UC
- O modelo de avaliação é o mesmo da UC2: entrega do grupo por aula, e a checagem de bloco **conta por ter sido feita**

O mapa de hoje

- **Bloco 1** — o código que roda, imprime certo e está errado
- **Bloco 2** — o teste que confere sozinho: `assert` e `pytest`
- **Bloco 3** — quantos testes bastam?
- **Bloco 4** — mão na massa: testar o relatório da locadora e uma regra do sistema de vocês

> Hoje ninguém mexe no banco. O código de hoje é Python puro, com regra de negócio da locadora — o CRUD de vocês volta na Aula 05.

Retomando a UC2

- Que caso de falha vocês escreveram para o CRUD do grupo?
- Na Aula 18, como vocês **sabiam** que o CRUD funcionava?
- Alguém achou um erro no próprio CRUD **depois** de entregar?

> Anotem no papel os casos de falha que surgirem agora. Eles voltam no Bloco 4 de hoje e viram plano de teste na Aula 06.

Bloco 1

Funciona na minha máquina

O relatório do mês

O dono da locadora quer saber qual foi o aluguel **mais caro** e o **mais barato** do mês.

```
class RelatorioDoMes:
    def __init__(self):
        self.maior = float("-inf")   # começa no "menos infinito"
        self.menor = float("inf")    # começa no "mais infinito"

    def avaliar(self, alugueis):
        for aluguel in alugueis:
            if aluguel.valor > self.maior:
                self.maior = aluguel.valor
            elif aluguel.valor < self.menor:
                self.menor = aluguel.valor
```

Leiam com calma. Tem algo errado aí?

Rodei. Imprimiu certo.

```
alugueis = [
    Aluguel("Ana",   "FIFA 25",    30.0),
    Aluguel("Bruno", "Zelda",      40.0),
    Aluguel("Carla", "Mario Kart", 25.0),
]
relatorio = RelatorioDoMes()
relatorio.avaliar(alugueis)
print(relatorio.maior)   # 40.0
print(relatorio.menor)   # 25.0
```

- O maior saiu 40. O menor saiu 25. Os dois certos.
- Está testado? Pode ir para o computador da locadora?

Palpite — e se a ordem for outra?

Os **mesmos três** aluguéis, só que agora em ordem crescente de valor:

```
alugueis = [
    Aluguel("Carla", "Mario Kart", 25.0),
    Aluguel("Ana",   "FIFA 25",    30.0),
    Aluguel("Bruno", "Zelda",      40.0),
]
```

Antes de eu rodar: escrevam no papel o que vai aparecer no `maior` e no `menor`. Os valores são os mesmos de antes — muda só a ordem.

⚠ O que saiu na tela

```
40.0
inf
```

- O maior continua certo.
- O menor saiu `inf` — infinito. Nenhum aluguel custou isso.
- Mesmos valores, mesmo código. O que mudou foi só a ordem.

Linha a linha

Aluguel	Maior que o maior?	Entra no elif?	maior	menor
início	—	—	`-inf`	`inf`
Carla, 25	sim	**não**	25	`inf`
Ana, 30	sim	**não**	30	`inf`
Bruno, 40	sim	**não**	40	`inf`

Em ordem crescente, todo aluguel é maior que o anterior. O `if` sempre entra, então o `elif` **nunca roda** — e o `menor` nunca sai do infinito.

Engano, defeito, falha

Engano
na cabeça de quem escreveu: "um valor não pode ser o maior e o menor ao mesmo tempo"

Defeito
no código: o `elif` onde devia haver um segundo `if`

Falha
na tela: o relatório mostra `inf` como menor aluguel

GLOSSÁRIO

`error` / `mistake`	engano — o erro humano
`defect` / `bug`	defeito — o que ficou errado no código
`failure`	falha — o que o usuário vê dar errado

Checagem do Bloco 1

Sem rodar, escreva o `maior` e o `menor` que o `RelatorioDoMes` do slide mostra para cada lista de valores:

- `40, 30, 25`
- `50`
- `30, 20, 35`

Qual das três revela a falha? Por quê?

`forms.gle/UEiDJqvhYpHamzM4A`

Intervalo

10 minutos

Bloco 2

Quem confere a tela?

Todo teste tem três partes

Vocês já testaram à mão o cadastro de cliente do CRUD. Foi assim:

1. **Cenário** — abrir o cadastro e deixar o telefone em branco
2. **Ação** — mandar gravar
3. **Validação** — olhar se o sistema recusou e disse por quê

- O `print` do relatório já fez o cenário e a ação **sozinho**.
- A validação ainda é o olho de vocês, lendo `40.0` e `25.0` na tela.
- E se fossem 200 relatórios para conferir, toda vez que alguém mexe no código?

`assert` — a validação vira código

```python
from relatorio import Aluguel, RelatorioDoMes
def test_entende_alugueis_em_ordem_crescente():
    alugueis = [Aluguel("Carla", "Mario Kart", 25.0),   # cenário
                Aluguel("Ana", "FIFA 25", 30.0),
                Aluguel("Bruno", "Zelda", 40.0)]
    relatorio = RelatorioDoMes()                        # ação
    relatorio.avaliar(alugueis)
    assert relatorio.maior == 40.0                      # validação
    assert relatorio.menor == 25.0
```

GLOSSÁRIO

`assert`	afirmar — se não for verdade, o teste para ali e fica vermelho
`test`	teste — no `pytest`, toda função cujo nome começa com `test_`

Rodando o `pytest` — sem terminal

1. Criem a pasta `uc3`, ponham nela o `relatorio.py` do site, e abram a pasta no VS Code (**File → Open Folder**)
2. Num notebook novo nessa pasta, rodem uma célula com `%pip install pytest` — o mesmo jeito do `psycopg2` na Aula 12
3. Criem o arquivo `test_relatorio.py` e colem o teste do slide anterior
4. Na barra da esquerda, abram o painel **Testing** (o frasco) → **Configure Python Tests** → **pytest** → **Root directory** (a própria pasta)
5. Cliquem no ▶ ao lado do teste. Verde passou, vermelho falhou

⚠ Vermelho

Palpite antes: qual das duas linhas de `assert` vai acusar?

```
    assert relatorio.maior == 40.0
>   assert relatorio.menor == 25.0
E   assert inf == 25.0

FAILED test_relatorio.py::test_entende_alugueis_em_ordem_crescente
```

- O `>` aponta a linha que falhou. O `E` mostra o que veio e o que se esperava.
- Ninguém precisou ler a tela e comparar de cabeça: quem comparou foi o `assert`.

Consertando o defeito

```
    def avaliar(self, alugueis):
        for aluguel in alugueis:
            if aluguel.valor > self.maior:
                self.maior = aluguel.valor
            if aluguel.valor < self.menor:      # era elif
                self.menor = aluguel.valor
```

- Um `if` independente para cada pergunta: o mesmo aluguel **pode** ser o maior e o menor.
- ▶ de novo. Verde, em milissegundos.
- E o teste fica no arquivo: da próxima vez que alguém mexer no `avaliar`, ele confere de novo, sem ninguém pedir.

⚠ Erro comum — o teste que nunca rodou

> O `pytest` só acha o que segue o combinado de nome. Fora dele, o teste não falha: ele **some**. O painel fica vazio, ou mostra um teste a menos, e a turma acha que está tudo verde.

Assim ele acha	Assim ele ignora, em silêncio
arquivo `test_relatorio.py`	`relatorio_teste.py`, `testes.py`
função `def test_entende_...():`	`def entende_...():`, `def Test_...():`

Conferência rápida: o número de testes no painel bate com o número de funções que vocês escreveram?

Checagem do Bloco 2

O arquivo se chama `relatorio_teste.py`:

```
def test_ordem_decrescente():
    alugueis = [Aluguel("Bruno", "Zelda", 40.0),
                Aluguel("Carla", "Mario Kart", 25.0)]
    relatorio = RelatorioDoMes()
    relatorio.avaliar(alugueis)
```

- Marque o cenário e a ação
- Escreva a validação que falta
- O `pytest` vai rodar este teste? Por quê?



forms.gle/Y33iC2TNdxSbw8eN9

Intervalo

10 minutos

Bloco 3

Quantos testes bastam?

Mais um teste? E mais um?

```
def test_crescente_1():   # 25, 30, 40
def test_crescente_2():   # 100, 200, 300
def test_crescente_3():   # 1, 2, 3
def test_crescente_4():   # 10, 20, 30, 40, 50
```

- Quatro testes. Se o defeito do `elif` voltar, quantos deles ficam vermelhos?
- Algum deles pega um defeito que os outros não pegam?

Classes de equivalência

Casos que exercitam o código **do mesmo jeito** formam uma classe. Basta um teste por classe.

Classe	Exemplo	O caminho que ela faz no código
crescente	25, 30, 40	o `if` sempre entra
decrescente	40, 30, 25	o `if` só entra na primeira volta
fora de ordem	30, 40, 25	os dois caminhos se alternam
um aluguel só	30	o mesmo valor é o maior **e** o menor

GLOSSÁRIO

`equivalence class` — classe de equivalência — casos que testam a mesma coisa

⚠ Erro comum — o nome que não diz nada

Assim, não

```
def test1():
def test_relatorio():
def test_ok():
```

Assim, sim

```
def test_entende_alugueis_em_ordem_crescente():
def test_um_aluguel_so_e_o_maior_e_o_menor():
```

Quando o teste ficar vermelho daqui a um mês, o nome é a primeira coisa que alguém lê. `test1` falhou — e daí? Já `test_um_aluguel_so_e_o_maior_e_o_menor` falhou diz o que quebrou antes de abrir o arquivo.

Primeiro a lista, depois o código

Antes de escrever qualquer teste, escrevam no papel:

1. O que a função **deve** fazer, em português
2. As **classes** de entrada que mudam o caminho dentro do código
3. Um **exemplo** de cada classe, com o resultado esperado
4. Só então, um teste por linha da lista

> Essa lista é o começo de um **plano de teste**. Os casos de falha que vocês trouxeram da Aula 18 já são linhas dela.

Checagem do Bloco 3

Seis testes para o `RelatorioDoMes`, cada um com uma lista de valores:

- `10, 20, 30`
- `5, 15, 25`
- `30, 20, 10`
- `20, 10, 30`
- `99`
- `7`

Agrupe em classes e risque os que não trazem nada de novo. Com quantos testes você fica?



`forms.gle/vNF7tomdSNkPn7tB6`

Intervalo

10 minutos

MÃO NA MASSA

Testando o relatório da locadora

No `relatorio.py` há uma segunda função, `total_do_cliente(alugueis, cliente)`: quanto um cliente gastou no mês.

1. **No papel:** as classes de entrada de `total_do_cliente` e um exemplo de cada, com o resultado esperado
2. Um teste por classe no `test_relatorio.py`, com nome que diz o que testa
3. Rodem. Se algum ficar vermelho, achem o **defeito**, consertem, e rodem tudo de novo
4. **Sistema de vocês:** escolham uma regra do minimundo do grupo (um preço, um prazo, um desconto), escrevam a função e de 3 a 4 testes de classes diferentes

MÃO NA MASSA

O combinado desta entrega

GLOSSÁRIO

`test_grupoNN_aulaNN.py` `test_grupo03_aula01.py`

Por que começa com `test_`? Mudem o nome e olhem o painel Testing.

O que eu confiro: No topo do arquivo, em comentário, a **lista de classes** que vocês escreveram no papel · **Um teste por classe**, sem repetir classe · Nomes de teste que dizem **o que** testam · O defeito de `total_do_cliente` achado **por um teste**, não no olho · A regra do sistema de vocês com pelo menos **3 testes**

> Eu passo de mesa em mesa. Tenham na tela o painel Testing com os testes rodados — verde e vermelho, os dois contam.

PARA CASA · OPCIONAL

Antes da próxima aula

Testar em casa: volte à lista de casos de falha do CRUD que você trouxe da Aula 18. Para cada um, escreva a classe a que ele pertence.

Tarefa opcional: o que o `RelatorioDoMes` deveria fazer com uma lista **vazia** de aluguéis? Escreva o teste do que você acha certo e rode.

Para treinar: o `relatorio.py` está no site, na página da aula.

Para ver acontecer: demonstrações da Aula 01 — palpitar o menor, ver o `elif` virar infinito e montar a menor suíte que pega o defeito — e oito perguntas com resposta na hora.

Para a próxima aula

Os testes de hoje ficaram verdes.
Isso **prova** que o código está certo?

UniSenac Centro Universitário RS
Senac
Fecomércio
IFEP
Sesc
Sindicatos Empresariais