Casos Especiais e Regressão

Aula 02 — O teste passou. E daí?

Aula 02 · 05/11/2026 · Indicadores 1, 3

UniSenac Centro Universitário RS
Senac
Fecomércio
IFEP
Sesc
Sindicatos Empresariais

O mapa de hoje

- **Bloco 1** — uma funcionalidade nova, e o teste antigo que reclamou
- **Bloco 2** — a beirada: fronteira, lista vazia e o erro que tem de acontecer
- **Bloco 3** — cuidar dos testes para eles não virarem um peso
- **Bloco 4** — mão na massa: as regras do balcão da locadora e do sistema de vocês

> Abram a pasta `uc3` de ontem, com o `relatorio.py` já consertado e o `test_relatorio.py`.
> Hoje a gente continua em cima deles.

Retomando a Aula 01

- O `inf` no relatório: era o defeito ou a falha?
- Por que um teste por classe, e não cinco da mesma?
- Lista vazia de aluguéis: o que o teste de vocês decidiu?

Bloco 1

Uma funcionalidade nova

Os três aluguéis mais caros

O dono agora quer ver também os **três maiores** aluguéis do mês. Acrescentem no fim do `avaliar`:

```
        valores = sorted([a.valor for a in alugueis], reverse=True)
        self.tres_maiores = [valores[i] for i in range(3)]
```

E no `__init__`, junto com o `maior` e o `menor`:

```
        self.tres_maiores = []
```

GLOSSÁRIO

`sorted(..., reverse=True)`	ordenado, invertido — do maior para o menor

O teste da funcionalidade nova

```python
def test_encontra_os_tres_maiores():
    alugueis = [Aluguel("Ana", "FIFA 25", 10.0),
                Aluguel("Bruno", "Zelda", 20.0),
                Aluguel("Carla", "Mario Kart", 30.0),
                Aluguel("Ana", "Zelda", 40.0)]
    relatorio = RelatorioDoMes()
    relatorio.avaliar(alugueis)
    assert len(relatorio.tres_maiores) == 3
```

Palpite, antes de rodar: este teste passa? E os quatro testes de ontem, que continuam no mesmo arquivo — passam?

⚠ Rodei a suíte inteira

```
PASSED test_entende_alugueis_em_ordem_crescente
PASSED test_entende_alugueis_em_ordem_decrescente
PASSED test_entende_alugueis_fora_de_ordem
FAILED test_um_aluguel_so_e_o_maior_e_o_menor
       IndexError: list index out of range
PASSED test_encontra_os_tres_maiores
```

- O teste **novo** passou.
- Quem ficou vermelho foi um teste de **ontem** — de uma coisa que ninguém mexeu.

Regressão

- Com **um** aluguel só, `valores` tem um elemento. O `valores[1]` não existe — e o `avaliar` inteiro quebra, até o maior e o menor que funcionavam.
- Mexer numa parte e quebrar outra que já funcionava tem nome: **regressão**.
- Quem pegou não foi ninguém olhando. Foi o teste de ontem, que continuava lá, conferindo.

GLOSSÁRIO

`regression`	regressão — o que funcionava e voltou a quebrar
`test suite`	suíte de testes — todos os testes do projeto, rodados juntos

Consertando — com uma pegadinha

```
        valores = sorted([a.valor for a in alugueis], reverse=True)
        self.tres_maiores = valores[:3]     # fatia: pega até 3, sem estourar
```

- Agora tudo verde. Os cinco.
- Mas façam um teste: **apaguem o** `reverse=True` e rodem de novo.
- Palpite: o `test_encontra_os_tres_maiores` fica vermelho?

⚠ Erro comum — conferir o tamanho e não o conteúdo

Sem o `reverse`, o resultado é `[10.0, 20.0, 30.0]` — os três **menores**. São três, então `len(...) == 3` continua verde. O teste confere uma coisa que nunca estava em dúvida.

```
    assert relatorio.tres_maiores == [40.0, 30.0, 20.0]
```

- Confiram o **conteúdo** — e na ordem certa.
- Vários `assert` no mesmo teste não é problema, se todos conferem **o mesmo resultado**. Dois resultados diferentes são dois testes.

Checagem do Bloco 1

O código ordena **sem** o `reverse`, e o teste é este:

```
relatorio.avaliar(alugueis_de_10_20_30_40)
assert len(relatorio.tres_maiores) == 3
```

- O teste passa ou falha? Por quê?
- Reescreva o `assert` para ele pegar o defeito
- Você mexeu no `avaliar` e o teste de `total_do_cliente`, que ninguém tocou, ficou vermelho. Que nome tem isso?



forms.gle/tqcL9D6L6eEjXaAj8

Intervalo

10 minutos

Bloco 2

A beirada

"Até 2 dias, eu perdoo"

O dono da locadora: *"Atrasou até 2 dias, eu perdoo. A partir do 3º dia, cobro R$ 5,00 por dia de atraso."*

```python
def multa(dias_atraso):
    if dias_atraso > 3:
        return dias_atraso * 5.00
    return 0.0
```

Palpite: vocês só podem escrever **três** testes para essa função. Que valores de `dias_atraso` vocês escolhem? Escrevam no papel.

Onde o código decide

Dias de atraso	O dono disse	O código faz	
2	perdoa: R$ 0	R$ 0	certo
3	**cobra: R$ 15**	**R$ 0**	**falha**
4	cobra: R$ 20	R$ 20	certo

- Todo `if` com `>` ou `>=` tem uma **fronteira**. O defeito mora em cima dela, quase sempre.
- `>` e `>=` são fáceis de trocar — e só o teste **em cima** da fronteira mostra a diferença.
- Por isso a fronteira pede três testes: antes, em cima e depois.

E o mês sem aluguel nenhum?

```
relatorio = RelatorioDoMes()
relatorio.avaliar([])
print(relatorio.maior, relatorio.menor)    # -inf inf
```

- Nenhum erro na tela. O relatório imprime `-inf` e `inf`, e o dono acha que é um valor.
- O certo não é devolver um número qualquer: é **recusar**, e dizer por quê.

```
        if not alugueis:
            raise ValueError("não há aluguéis no mês")
```

`pytest.raises` — testando o erro que tem de acontecer

```
import pytest
def test_mes_sem_aluguel_e_recusado():
    relatorio = RelatorioDoMes()
    with pytest.raises(ValueError):
        relatorio.avaliar([])
```

- Levantou o `ValueError`? Passa. Seguiu em frente sem erro? **Falha**: `DID NOT RAISE`.

GLOSSÁRIO

`raise` / `raises`	levantar / levanta — o código avisa que não pode continuar
`ValueError`	erro de valor — recebi um valor que não faz sentido
`with`	com — "dentro deste bloco, eu espero o erro"

Os casos especiais de sempre

Se a função recebe…	Testem também…
uma lista	vazia · com um item · com vários
um número com regra	antes, em cima e depois da fronteira · zero · negativo
um texto	vazio · com espaço · com acento ou aspas
algo que pode não existir	o id que não está no banco

O erro comum é ir direto ao caso complicado e esquecer os pequenos. O defeito de ontem estava no "um aluguel só"; o de hoje, no "lista vazia".

Checagem do Bloco 2

O dono: *"Quem aluga por 5 dias ou mais ganha 10% de desconto. Aluguel de zero dias, ou menos, não existe."*

- Quais **três** valores de dias você testa na fronteira? E o desconto esperado de cada um?
- Qual entrada tem de ser **recusada**?
- Escreva o teste dessa recusa, com `pytest.raises`



forms.gle/ScEJ6Tyg3MRHabHH8

Intervalo

10 minutos

Bloco 3

Cuidando dos testes

Seis testes, seis `RelatorioDoMes()`

O dono pediu: o relatório agora precisa saber **de que mês** ele é.

```
class RelatorioDoMes:
    def __init__(self, mes):
        ...
```

- Quantos lugares da suíte quebram com essa mudança?
- O código de produção mudou numa linha. Os testes vão mudar em seis.

`fixture` — o que se prepara antes de cada teste

```python
@pytest.fixture
def relatorio():
    return RelatorioDoMes()

def test_um_aluguel_so_e_o_maior_e_o_menor(relatorio):
    relatorio.avaliar(alugueis_com_valores(30.0))
    assert relatorio.maior == 30.0
    assert relatorio.menor == 30.0
```

- O `pytest` vê o parâmetro `relatorio`, roda a fixture e entrega **um relatório novo para cada teste**. O construtor mudou? Muda num lugar só.

GLOSSÁRIO

`fixture`	preparação — o cenário pronto que o teste recebe

Montar o cenário numa linha

```python
def alugueis_com_valores(*valores):
    """Um aluguel para cada valor, na ordem dada."""
    return [Aluguel("Ana", "FIFA 25", valor) for valor in valores]
```

```python
relatorio.avaliar(alugueis_com_valores(25.0, 30.0, 40.0))
```

- O teste fica com o que **importa** para ele: os valores e a ordem. Cliente e jogo não mudam nada no `avaliar`.
- Função que monta cenário de teste tem nome: **construtor de dados de teste** (*test data builder*).

E 100% de cobertura?

- **Cobertura** é quanto do código foi executado por pelo menos um teste.
- O defeito de `total_do_cliente` passava com cobertura **completa**: o teste de um aluguel executava a linha do `=` — e ficava verde.
- Executar a linha não é conferir o resultado. Cobertura mostra o que **ninguém** testou; não prova que o testado está certo.
- Onde gastar teste primeiro: na regra complicada e na regra que dá prejuízo se errar.

GLOSSÁRIO

`coverage`	cobertura — quanto do código os testes executaram

Checagem do Bloco 3

```python
def test_decrescente():
    relatorio = RelatorioDoMes()
    relatorio.avaliar(alugueis_com_valores(40.0, 25.0))
    assert relatorio.menor == 25.0

def test_um_so():
    relatorio = RelatorioDoMes()
    relatorio.avaliar(alugueis_com_valores(7.0))
    assert relatorio.maior == 7.0
```

- Reescreva os dois usando a fixture `relatorio`
- Um colega diz: "temos 100% de cobertura, então não tem defeito". O que você responde?

forms.gle/XVVdarKzMtLk7LvP8

Intervalo

10 minutos

MÃO NA MASSA

As regras do balcão

O `multa.py` do site tem as duas regras do dono, escritas em comentário no topo, e o código que alguém fez a partir delas.

1. **No papel:** a fronteira de cada regra, os casos especiais, e o que tem de ser **recusado**
2. Os testes no `test_multa.py` — antes, em cima e depois da fronteira, e um `pytest.raises`
3. Para cada teste vermelho: achem o **defeito**, consertem, rodem a suíte inteira e anotem o que mudou
4. **Sistema de vocês:** na regra que o grupo escreveu ontem, a fronteira com três testes e um caso que tem de ser recusado

MÃO NA MASSA

O combinado desta entrega

GLOSSÁRIO

`test_grupoNN_aulaNN.py` `test_grupo03_aula02.py`

O que eu confiro: Cada fronteira com teste **antes, em cima e depois** · Pelo menos um `pytest.raises` · O conteúdo conferido, **não só o tamanho** · No fim do arquivo, em comentário, o **registro**: que teste achou que defeito, e o que foi mudado para consertar · A suíte inteira rodada **depois** de cada conserto

> Eu passo de mesa em mesa. O registro é o que eu leio primeiro: ele é a prova de que o conserto veio de um teste, e não de um palpite.

PARA CASA · OPCIONAL

Antes da próxima aula

Testar em casa: passe a fixture `relatorio` e o `alugueis_com_valores` para todos os testes do `test_relatorio.py`. Rode a suíte: o número de testes tem de continuar o mesmo.

Tarefa opcional: o `multa(-1)` — devolução **adiantada** — devolve o quê? E deveria devolver o quê? Escreva o teste do que você acha certo.

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

Para ver acontecer: demonstrações da Aula 02 — escolher os três dias que testam a multa e ver o código errar justo no 3º, e o teste de ontem pegar a regressão — e oito perguntas com resposta na hora.

Para a próxima aula

Hoje o código veio pronto, e o teste veio depois para achar o defeito. E se o teste viesse **antes** do código?

UniSenac Centro Universitário RS
Senac
Fecomércio
IFEP
Sesc
Sindicatos Empresariais