Tomando Decisões com if:
Regras de Recompensa no Sistema

Ifs – Objetivos da aula

- Ao final desta aula, você será capaz de:
  - Entender **if**, **elif** e **else** para decisões no código
  - Implementar regras de recompensa simbólicas por faixa de compra
  - Testar condições com **assert**
  - Trabalhar com decisões encadeadas e limites de valor
  - Testar exceções usando **with** pytest.raises

- Aprenderemos a implementar regras de negócio utilizando estruturas condicionais e TDD.

Por que usar decisões condicionais?

- Recompensas variam conforme o valor da compra
- Estruturas condicionais permitem alterar o comportamento do programa de forma clara e lógica
- O TDD ajudará a validar cada regra implementada

Operadores relacionais

- Para comparar valores usamos:
  - **>** maior que
  - **<** menor que
  - **>=** maior ou igual
  - **<=** menor ou igual
  - **==** igual
  - **!=** diferente

* Com esses operadores, iremos definir as faixas de compras que acionam diferentes recompensas.

Estrutura básica if

- O trecho abaixo executa um bloco de código apenas se a condição for verdadeira.

```
valor = 600
if valor > 500:
    print("Compra grande")
```

Introduzindo elif

- Permite testar uma segunda condição se a primeira falhar.

```
valor = 200
if valor > 500:
    print("Compra grande")
elif valor > 100:
    print("Compra média")
```

Usando else

- Captura todos os casos que não se encaixam nos ifs ou elifs.

```
valor = 50
if valor > 500:
    print("Compra grande")
elif valor > 100:
    print("Compra média")
else:
    print("Compra pequena")
```

Implementando decisão com if

- A regra de bônus foi adicionada na função **calcula_pontos** usando um if:

```python
def calcula_pontos(valor):
    pontos = valor * 0.10
    if valor > 500:
        pontos += 50
    return pontos
```

```python
def test_pontos_bonus():
    assert calcula_pontos(600) == 60 + 50
```

Operadores Lógicos em Decisões

- Operadores lógicos permitem combinar condições dentro de um if ou elif.
  - 1. **and** → Todas as condições precisam ser verdadeiras
  - Executa o bloco apenas se valor > 100 E usuario_compras > 3 (fidelidade).

```
valor = 200
usuario_compras = 4

if valor > 100 and usuario_compras > 3:
    print("Bônus especial")
```

Operadores Lógicos em Decisões

- Operadores lógicos permitem combinar condições dentro de um if ou elif.
  - 2. **or** → Basta uma das condições ser verdadeira
  - Executa o bloco se valor < 20 OU valor > 500.

```
valor = 10

if valor < 20 or valor > 500:
    print("Oferta especial!")
```

Operadores Lógicos em Decisões

- Operadores lógicos permitem combinar condições dentro de um if ou elif.
  - 3. **not** → Inverte uma condição
  - Executa o bloco se valor não for maior que 0.

```
valor = -10

if not valor > 0:
    print("Valor inválido")
```

Exemplo TDD + Ifs + AND e OR

- Teste:

```python
# Testes com assert para validar a regra
def test_recompensa():
    assert recompensa(30, 0) == (3, "Compra pequena")                  # compra menor que 50
    assert recompensa(150, 0) == (15, "Compra média")                  # compra entre 101 e 200
    assert recompensa(300, 0) == (30, "Compra grande ou cliente VIP")  # compra > 200
    assert recompensa(600, 6) == (60, "Compra grande e frequente!")    # compra > 500 e >5 compras anteriores
    assert recompensa(100, 11) == (10, "Compra grande ou cliente VIP") # cliente VIP
```

Exemplo TDD + Ifs + AND e OR

- Implementação:

```python
def recompensa(valor, compras_anteriores):
    # Calcula 10% do valor como pontos
    pontos = valor * 0.10

    # Define mensagem simbólica baseada na faixa da compra
    if valor > 500 and compras_anteriores > 5:
        mensagem = "Compra grande e frequente!"
    elif valor > 200 or compras_anteriores > 10:
        mensagem = "Compra grande ou cliente VIP"
    elif valor > 100:
        mensagem = "Compra média"
    else:
        mensagem = "Compra pequena"

    return pontos, mensagem
```

Exemplo status do usuário

- Teste:

```python
# Função de teste
def test_status_usuario():
    assert status_usuario(600, 6) == "VIP Gold"      # muitos pontos e compras
    assert status_usuario(350, 2) == "VIP Silver"    # pontos altos ou poucas compras
    assert status_usuario(100, 3) == "VIP Silver"    # pontos baixos mas compras >=3
    assert status_usuario(0, 0) == "Sem pontos"      # nenhum ponto
    assert status_usuario(50, 1) == "Cliente padrão" # caso padrão
```

Exemplo status do usuário

- Implementação:

```python
def status_usuario(pontos_totais, compras_mes):
    """
    Retorna o status do usuário baseado nos pontos acumulados
    e no número de compras no mês.
    """
    if pontos_totais > 500 and compras_mes >= 5:
        return "VIP Gold"
    elif pontos_totais > 300 or compras_mes >= 3:
        return "VIP Silver"
    elif not  pontos_totais > 0):
        retu(n "Sem pontos"
    else:
        return "Cliente padrão"
```

Ifs aninhados ou encadeados

- Mensagem e bônus de acordo com pontos e frequência de compras:

```
def recompensa_avancada(pontos, compras_mes):
    if pontos > 100:
        if compras_mes > 5:
            return "VIP Gold: Bônus extra!"
        else:
            return "VIP Gold"
    else:
        if pontos > 50:
            return "VIP Silver"
        else:
            if pontos > 0:
                return "Cliente Bronze"
            else:
                return "Sem pontos"
```

Ifs aninhados ou encadeados

- Atenção:
  - Python usa a indentação para definir blocos de código.
  - Não existem chaves {} como em outras linguagens.
  - Cada nível de indentação indica um bloco de código dentro de if, for, while, funções, classes, etc.
  - Indentação incorreta causa erros de sintaxe (IndentationError) ou comportamento inesperado.

Ifs aninhados ou encadeados

- Exemplo correto:

```
if True:
    print("Bloco correto")
    print("Tudo dentro do if")
```

- Exemplo incorreto:

```
if True:
print("Erro de indentação!")  # Indentação ausente
```

Ifs compacto

- Python permite escrever decisões simples em uma linha usando um operador ternário para isto:

```
variavel = valor_se_verdadeiro if condicao else valor_se_falso
```

```
def status_usuario(pontos):
    return "VIP" if pontos > 500 else "Cliente padrão"

print(status_usuario(600))  # VIP
print(status_usuario(200))  # Cliente padrão
```

Python não tem switch/case

- Diferente de algumas linguagens como C, Java ou JavaScript, Python não possui a estrutura switch/case nativa.

- Para tomar decisões múltiplas, usamos:
  - if / elif / else
  - Dicionários com funções ou valores (como alternativa)

Testando Exceções no TDD

- No TDD, não testamos só valores corretos, mas também se a função trata entradas inválidas corretamente.
- Se a exceção não ocorrer, o teste falha automaticamente, mostrando que a função não está tratando o erro corretamente.
- Esse tipo de teste é muito útil para garantir a robustez do sistema e que ele não aceita entradas inválidas.

Testando Exceções no TDD

- A função **validar_compra** lança um erro (**ValueError**) se o valor da compra for negativo.
- O **with** pytest.raises(ValueError): é usado para verificar se o erro esperado realmente ocorre. with cria um contexto onde o **pytest** espera que uma exceção seja levantada.

```python
def validar_compra(valor):
    if valor < 0:
        raise ValueError("Valor inválido")
    return True
```

```python
import pytest

def test_validar_compra():
    with pytest.raises(ValueError):
        validar_compra(-10)
```

Testando Exceções no TDD

- **validar_compra(-10)** deve lançar **ValueError**
- **pytest.raises** confirma que a exceção ocorreu
- Teste passa se a exceção for levantada, falha caso contrário

```python
def validar_compra(valor):
    if valor < 0:
        raise ValueError("Valor inválido")
    return True
```

```python
import pytest

def test_validar_compra():
    with pytest.raises(ValueError):
        validar_compra(-10)
```

Exemplo clientes inativos (ValueError)

- Teste de bloqueio de compras para clientes inativos

```python
class Usuario:
    def __init__(self, ativo=True):
        self.ativo = ativo
        self.compras = []

    def add_compra(self, valor):
        if not self.ativo:
            raise ValueError("Usuário inativo não pode comprar")
        self.compras.append(valor)
```

```python
import pytest

def test_usuario_inativo():
    u = Usuario(ativo=False)
    with pytest.raises(ValueError):
        u.add_compra(100)
```

Exemplo valor numérico (TypeError)

- Lança um **TypeError** se o valor da compra não for numérico

```python
def calcula_pontos(valor):
    if not isinstance(valor, (int, float)):
        raise TypeError("Valor da compra deve ser numérico")
    return valor * 0.10
```

```python
import pytest

def test_calcula_pontos_tipo():
    with pytest.raises(TypeError):
        calcula_pontos("cem")  # string em vez de número
    with pytest.raises(TypeError):
        calcula_pontos(None)   # None não é permitido
```

Exemplo valor numérico (TypeError)

- O teste garante que o sistema **não aceita valores não numéricos**.
- **with** pytest.raises(TypeError) captura a exceção correta.
- **not isinstance**(valor, (int, float)) verifica se valor não é um número inteiro nem um número de ponto flutuante.

```python
import pytest

def test_calcula_pontos_tipo():
    with pytest.raises(TypeError):
        calcula_pontos("cem")  # string em vez de número
    with pytest.raises(TypeError):
        calcula_pontos(None)   # None não é permitido
```

Exemplo dicionário (KeyError)

```python
import pytest

# Teste usando pytest
def test_nivel_usuario_keyerror():
    usuario_valido =  {"nome": "João", "nivel": "VIP"}
    usuario_invalido =  {"nome": "Maria"}  # sem a chave "nivel"

    # Testa caso válido
    assert nivel_usuario(usuario_valido) == "VIP"

    # Testa exceção quando a chave não existe
    with pytest.raises(KeyError):
        nivel_usuario(usuario_invalido)
```

```python
# Função que retorna o nível do usuário a partir de um dicionário
def nivel_usuario(usuario):
    # Se a chave "nivel" não existir, KeyError será lançado
    return usuario["nivel"]
```

Exemplo dicionário (KeyError)

- A exceção **KeyError** acontece quando tentamos acessar uma chave que não existe no dicionário.
- **with** pytest.raises(KeyError) permite testar a exceção de forma segura.
- Quando você acessa usuario["nivel"] em um dicionário e a chave "nivel" não existe, o Python já lança automaticamente um **KeyError**.

Encerramento

- Você aprendeu:
  - if, elif e else para decisões no código
  - Implementar regras de recompensa simbólicas por faixa de compra
  - Testar condições com **assert**
  - Trabalhar com decisões encadeadas e limites de valor
  - Testar exceções usando **with** pytest.raises: **ValueError**, **TypeError** e **KeyError**

PUC Minas