Repetindo Ações com while: Cadastro de Compras Iterativo

While – Objetivos da aula

- Você será capaz de:
  - Entender e aplicar laços while para repetição.
  - Controlar fluxos interativos com menus e sentinelas.
  - Validar entradas do usuário continuamente.
  - Testar fluxos repetitivos usando TDD.
  - Construir um menu interativo no AraCoins.

Introdução aos loops

- Laços permitem repetir blocos de código até uma condição ser satisfeita.
- **while** repete enquanto a condição for **True**.
- Útil para entrada contínua de dados, menus interativos e fluxos cíclicos.

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

Introdução aos loops

- No exemplo abaixo, repete-se o bloco enquanto contador < 5.
- É necessário alterar a condição dentro do laço para evitar um loop infinito.

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

Conceito de sentinela

- Uma sentinela é um valor especial que indica quando parar o loop.
- Exemplo: digitar "sair" para encerrar entrada de compras.

```
compra = ""
while compra != "sair":
    compra = input("Digite o produto ou 'sair': ")
    print(f"Produto registrado: {compra}")
```

- Continua pedindo produtos até que o usuário digite "sair".

Validação de entrada

- Sempre verifique se o usuário digitou valores válidos.
- Evita erros e garante integridade dos dados.

```
valor = input("Digite o valor da compra: ")
while not valor.isdigit():
    valor = input("Valor inválido. Digite novamente: ")
valor = float(valor)
```

- O loop garante que o valor seja numérico.

Fluxo completo de entrada

- **while True** simula um loop infinito (até que a condição de parada seja atendida).
- **break** sai do loop e **continue** reinicia o loop imediatamente.

```
while True:
    produto = input("Produto: ")
    if produto == "sair":
        break
    valor = input("Valor: ")
    if not valor.isdigit():
        print("Valor inválido")
        continue
    print(f"{produto} registrado com valor {valor}")
```

Fluxo completo de entrada

- Importante: Python é **case-sensitive**
  - Python diferencia maiúsculas e minúsculas em palavras-chave, nomes de variáveis, funções e classes.
  - Ex.: **while** funciona, mas WHILE gera erro.

```
while True:
    print("Loop funcionando")  # correto

WHILE True:
    print("Erro!")             # NameError
```

Menu interativo básico

- Estrutura de menu usando while + sentinela.

```
opcao = ""
while opcao != "sair":
    print("1 - Adicionar compra")
    print("2 - Ver total")
    print("sair - Encerrar")
    opcao = input("Escolha uma opção: ")
```

Controle de fluxo no menu

- Uso de if/elif/else dentro do loop.

```
if opcao == "1":
    print("Adicionar compra selecionado")
elif opcao == "2":
    print("Ver total selecionado")
elif opcao != "sair":
    print("Opção inválida")
```

Menu completo

```
opcao = ""
while opcao != "sair":
    print("1 - Adicionar compra")
    print("2 - Ver total")
    print("sair - Encerrar")
    opcao = input("Escolha uma opção: ")
    if opcao == "1":
        print("Adicionar compra selecionado")
    elif opcao == "2":
        print("Ver total selecionado")
    elif opcao != "sair":
        print("Opção inválida")
```

Testando menus

- Separar lógica de exibição e lógica de cálculo facilita testes.
- Podemos criar funções que retornam strings em vez de usar input/print.

```python
def test_processar_opcao():
    assert processar_opcao("1") == "Adicionar compra"
    assert processar_opcao("2") == "Ver total"
    assert processar_opcao("xyz") == "Opção inválida"
```

```python
def processar_opcao(opcao):
    if opcao == "1":
        return "Adicionar compra"
    elif opcao == "2":
        return "Ver total"
    else:
        return "Opção inválida"
```

Loop com contagem de compras

- Contabiliza o número de entradas.

```
total_compras = 0
while True:
    produto = input("Produto: ")
    if produto == "sair":
        break
    total_compras += 1
print(f"Total de compras: {total_compras}")
```

Testando contagem

- Simula o loop com uma lista de produtos.

```python
def contar_compras(lista_produtos):
    return len(lista_produtos)

def test_contar_compras():
    assert contar_compras(["Caneta", "Caderno"]) == 2
```

Uso de Flags

- Uma **flag** é uma variável booleana que controla o loop.
- Útil para encerrar o loop por múltiplas condições.

```
continuar = True
while continuar:
    # código
    continuar = input("Continuar? (s/n) ") == "s"
```

Testando loops com flags

- Esta função verifica se o usuário deseja continuar o loop.
- Retorna True apenas se a resposta for "s" (sim) e False caso contrário.
- Permite testar o fluxo de repetição sem depender de input() real, mantendo a lógica do loop testável.

```python
def deve_continuar(resposta):
    return resposta.lower() == "s"

def test_deve_continuar():
    assert deve_continuar("s") is True
    assert deve_continuar("n") is False
```

Validação de valores numéricos

- Validando valores numéricos com while.
- Evita erros de conversão com int() ou float().

```
while True:
    valor = input("Valor: ")
    if valor.replace(".", "").isdigit():
        valor = float(valor)
        break
    print("Valor inválido")
```

Contando produtos com while

- Conta quantos produtos foram adicionados antes do comando "sair".
- Simples, direto e totalmente testável com **assert**.

```python
def test_contar_produtos():
    assert contar_produtos(["Caneta", "Caderno", "sair", "Lápis"]) == 2
    assert contar_produtos(["Produto1", "Produto2", "sair"]) == 2
    assert contar_produtos(["sair"]) == 0
```

```python
def contar_produtos(entradas):
    total = 0
    i = 0
    while i < len(entradas):
        if entradas[i] == "sair":
            break
        total += 1
        i += 1
    return total
```

Função de loop do AraCoins

- Permite simular entrada de compras em sequência.
- O loop para ao receber "sair", garantindo controle do fluxo.
- Cada compra gera 10% do valor em pontos, aplicando a regra do sistema.

Função de loop do AraCoins

```python
def test_processar_compras():
    assert processar_compras( [100, 200, "sair", 50] ) == 30.0
    assert processar_compras( [50, 150, "sair"] ) == 20.0
    assert processar_compras( ["sair"] )  == 0
```

```python
def processar_compras(entradas):
    total_pontos = 0
    i = 0

    while i < len(entradas):
        compra = entradas[i]
        i += 1

        if compra == "sair":
            break

        # Regra: 10% do valor em pontos
        total_pontos += compra * 0.10

    return total_pontos
```

Dicas importantes

- Sempre evitar loop infinito: controlar com flag ou **break**.
- Separar lógica do input/output para facilitar testes.
- Validar entrada do usuário antes de usar valores.
- Escrever testes para fluxos repetitivos garante segurança do código.

Encerramento

- Você aprendeu:
  - Usar laços while para repetição e menus interativos.
  - Controlar o fluxo com sentinelas e flags.
  - Validar entradas contínuas do usuário.
  - Testar fluxos repetitivos com TDD.
  - Construir um menu interativo completo no AraCoins.

PUC Minas