Iterando com for:
Exibindo Compras e Total

For – Objetivos da aula

- Entender o laço for para iterar sobre listas e tuplas.
- Somar valores e exibir produtos usando iteração.
- Testar o comportamento esperado com assert e TDD.
- Explorar variações de escrita do for.

O que é o For

- Estrutura de repetição que percorre cada elemento de uma coleção.
- Mais simples e seguro que usar índices manualmente.
- Ideal para listas, tuplas e strings.

Sintaxe básica

- Itera sobre cada elemento da lista compras.

```
compras = [("Caneta", 10), ("Caderno", 20)]
for item in compras:
    print(item)
```

Acessando elementos da tupla

- Desempacota a tupla diretamente dentro do for.

```
for produto, valor in compras:
    print(f"Produto: {produto}, Valor: {valor}")
```

Exemplo de soma acumulada

- Soma todos os valores usando _ para ignorar o produto.

```
total = 0
for _, valor in compras:
    total += valor
print(total)
```

Testando soma com assert

- Mantém a lógica e o teste separados.

```python
def test_calcular_total():
    compras = [("Caneta", 10), ("Caderno", 25)]
    assert calcular_total(compras) == 35
```

```python
def calcular_total(compras):
    total = 0
    for _, valor in compras:
        total += valor
    return total
```

Iteração com índice usando range

- Útil quando precisamos do índice, mas não recomendado se o elemento já é suficiente.

```
for i in range(len(compras)):
    print(compras[i])
```

Testando elementos com índice

- Esse teste verifica se cada elemento da lista compras é uma tupla com o nome do produto como **string** e o valor como **inteiro**, garantindo a integridade dos dados antes de qualquer processamento.

```python
def test_produto_posicao():
    compras = [("Caneta", 10), ("Caderno", 25)]
    for i in range(len(compras)):
        assert isinstance(compras[i][0], str)
        assert isinstance(compras[i][1], int)
```

Iterando somente produtos

- _ indica que ignoramos o valor, mantendo o código limpo.

```
for produto, _ in compras:
    print(produto)
```

Iterando somente valores

- _ ignora o produto.

```
for _, valor in compras:
    print(valor)
```

Iteração reversa

- Mostra como percorrer a lista de trás para frente.

```
for i in range(len(compras)-1, -1, -1):
    print(compras[i])
```

Iteração reversa

- No for i in range(len(compras)-1, -1, -1): os três valores têm funções específicas:
  - len(compras)-1 → é o índice inicial, ou seja, começa no último elemento da lista.
  - -1 → é o índice final exclusivo; como o range não inclui o valor final, colocamos -1 para que o índice 0 seja incluído.
  - -1 → é o passo, ou seja, o loop decrementa de 1 em 1, percorrendo a lista de trás para frente.

```
for i in range(len(compras)-1, -1, -1):
    print(compras[i])
```

Usando enumerate

- **enumerate** fornece índice e elemento ao mesmo tempo.

```
for i, (produto, valor) in enumerate(compras):
    print(i, produto, valor)
```

```
def test_enumerate():
    compras = [("Caneta", 10), ("Caderno", 25)]
    for i, (produto, valor) in enumerate(compras):
        assert i < len(compras)
        assert isinstance(produto, str)
        assert isinstance(valor, int)
```

Somando e exibindo

```python
total = 0
for produto, valor in compras:
    print(f"{produto}: {valor}")
    total += valor
print(f"Total: {total}")
```

```python
def test_exibir_total():
    compras = [("Caneta", 10), ("Caderno", 25)]
    total = 0
    for _, valor in compras:
        total += valor
    assert total == 35
```

Filtrando dentro do for

- Mostra itens que atendem a uma condição.

```
for produto, valor in compras:
    if valor > 15:
        print(produto, valor)
```

```
def test_filtrar_valor():
    compras = [("Caneta", 10), ("Caderno", 25)]
    for produto, valor in compras:
        if valor > 15:
            assert valor > 15
```

Iterando strings

- For também funciona em strings, percorrendo cada caractere.

```
for letra in "AraCoins":
    print(letra)
```

```
def test_string():
    palavra = "AraCoins"
    for letra in palavra:
        assert isinstance(letra, str)
```

Loop aninhado

- Exemplo de for dentro de for (aninhamento).

```
compras = [ ("Caneta", 10), ("Caderno", 25)]
categorias = [ "Papelaria", "Escritório"]

for produto, valor in compras:
    for categoria in categorias:
        print(produto, categoria)
```

```
def test_loops_aninhados():
    compras = [ ("Caneta", 10), ("Caderno", 25)]
    categorias = [ "Papelaria", "Escritório"]
    for produto, valor in compras:
        for categoria in categorias:
            assert isinstance(produto, str)
            assert isinstance(categoria, str)
```

Iterando com range(start, stop, step)

- step define o incremento.

```python
for i in range(0, 10, 2):
    print(i)
```

```python
def test_range():
    numeros = []
    for i in range(0, 10, 2):
        numeros.append(i)
    assert numeros == [0, 2, 4, 6, 8]
```

Iteração com break

- **break** interrompe o loop antecipadamente.

```
for produto, valor in compras:
    if valor > 20:
        break
    print(produto)
```

Iteração com continue

- **continue** pula a iteração atual sem encerrar o loop.

```
for produto, valor in compras:
    if valor < 15:
        continue
    print(produto)
```

Testando break e continue

```python
def test_break_continue():
    compras = [("Caneta", 10), ("Caderno", 25)]
    resultados = []
    for produto, valor in compras:
        if valor < 15:
            continue
        resultados.append(produto)
    assert resultados == ["Caderno"]
```

Iterando sobre tupla diretamente

- for funciona tanto em listas quanto em tuplas.

```
tupla_compras = (("Caneta", 10), ("Caderno", 25))
for produto, valor in tupla_compras:
    print(produto, valor)
```

```
def test_tupla():
    tupla_compras = (("Caneta", 10), ("Caderno", 25))
    for produto, valor in tupla_compras:
        assert isinstance(produto, str)
        assert isinstance(valor, int)
```

Encerramento

- Você aprendeu:
  - Como usar for para percorrer **listas** e **tuplas**.
  - Iteração com **range**, **enumerate** e for **aninhados**.
  - Como aplicar **break**, continue e filtros.
  - Testes com **assert** para somar, filtrar e validar dados.
  - Aplicação prática no AraCoins para exibir produtos e calcular totais.

PUC Minas