



Tópico Avançado: Consultas, Ordenações e Análises na Lista



Objetivos da aula

- Ao final desta aula, você será capaz de:
 - Ordenar listas de tuplas usando **sorted()**
 - Utilizar **key** e **lambda** para extrair valores
 - Encontrar itens com **max()** e **min()**
 - Realizar consultas avançadas na lista de compras
 - Testar ordenações e análises com **TDD**
 - Integrar consultas ao menu do AraCoin

Por que ordenar listas?

- Ordenar ajuda a visualizar dados de forma mais clara, encontrar produtos mais baratos/caros e organizar informações.
- Ordenar também facilita comparações rápidas, melhora a navegação pelos dados e torna a análise mais eficiente em sistemas interativos.

O que é `sorted()`?

- `sorted()` retorna uma nova lista ordenada, sem modificar a lista original.
- `sorted()` em listas simples:
 - O código abaixo ordena números em ordem crescente.



```
numeros = [ 4, 1, 8, 2]  
ordenado = sorted(numeros)  
assert ordenado == [1, 2, 4, 8]
```

sorted() com lista de tuplas

- Ordena alfabeticamente pelo nome do produto.



```
compras = [("Caneta", 10), ("Caderno", 25)]  
ordenado = sorted(compras)
```

A importância do `key=`

- `key=` diz ao Python como comparar os elementos durante a ordenação.
- Ordenando pelo valor (índice 1)
 - A ordenação considera apenas o valor da compra.



```
compras = [("Caneta", 10), ("Caderno", 25)]  
ordenado = sorted(compras, key=lambda item: item[1])  
assert ordenado == [("Caneta", 10), ("Caderno", 25)]
```

Ordem decrescente

- **reverse=True** inverte a ordem.



```
ordenado_desc = sorted(compras, key=lambda item: item[1], reverse=True)
```

Teste: ordenando valores

- Teste simples de ordenação crescente.



```
def test_ordenacao_valores():  
    compras = [("Borracha", 5), ("Livro", 40)]  
    ordenado = sorted(compras, key=lambda x: x[1])  
    assert ordenado == [("Borracha", 5), ("Livro", 40)]
```

max() para encontrar o item mais caro

- **max()** usa a mesma lógica de key.



```
compras = [("Caneta", 10), ("Caderno", 25)]  
mais_caro = max(compras, key=lambda item: item[1])  
assert mais_caro == ("Caderno", 25)
```

min() para item mais barato

- Encontra o menor valor.



```
mais_barato = min(compras, key=lambda item: item[1])  
assert mais_barato == ("Caneta", 10)
```

Testando max() e min()

- Teste 1 garante o item de maior valor.
- Teste 2 garante o item de menor valor.



```
def test_mais_caro():  
    compras = [("Lápis", 3), ("Caneta", 10)]  
    assert max(compras, key=lambda x: x[1]) == ("Caneta", 10)
```



```
def test_mais_barato():  
    compras = [("Caneta", 10), ("Lápis", 3)]  
    assert min(compras, key=lambda x: x[1]) == ("Lápis", 3)
```

Exibindo lista ordenada

- Função para ordenar compras por nome.
- No teste, o primeiro item deve ser “Borracha”.



```
def ordenar_por_nome(compras):  
    return sorted(compras, key=lambda x: x[0])
```



```
def test_ordenar_por_nome():  
    compras = [("Caderno", 20), ("Borracha", 5)]  
    r = ordenar_por_nome(compras)  
    assert r[0][0] == "Borracha"
```

Ordenar por valor – função

- Ordena pela segunda posição da tupla.
- No teste, o menor valor aparece primeiro.



```
def ordenar_por_valor(compras):  
    return sorted(compras, key=lambda x: x[1])
```



```
def test_ordenar_por_valor():  
    compras = [("Caneta", 10), ("Lápis", 3)]  
    r = ordenar_por_valor(compras)  
    assert r == [("Lápis", 3), ("Caneta", 10)]
```

Função para maior compra

- Muito útil para relatórios.



```
def maior_compra(compras):  
    return max(compras, key=lambda x: x[1])
```



```
def test_maior_compra():  
    compras = [("Caneta", 10), ("Lápis", 3)]  
    assert maior_compra( compras ) == ("Caneta", 10)
```

Função para menor compra

- Muito útil para relatórios.



```
def menor_compra(compras):  
    return min(compras, key=lambda x: x[1])
```



```
def test_menor_compra():  
    compras = [("Caneta", 10), ("Lápis", 3)]  
    assert menor_compra( compras ) == ("Lápis", 3)
```

Filtrar compras acima de um valor

- Cria consulta específica.
- Só retorna itens maiores que 20.



```
def filtrar_acima(compras, limite):  
    return [ item for item in compras if item[1] > limite]
```



```
def test_filtrar_acima():  
    compras = [("Caneta", 10), ("Bolsa", 50)]  
    r = filtrar_acima(compras, 20)  
    assert r == [("Bolsa", 50)]
```

Ordenação personalizada

- lambda complexa.
- Ordena por valor, depois nome.



```
ordenado = sorted(compras, key=lambda x: (x[1], x[0]))
```



```
def test_ordenacao_dupla():  
    compras = [("Caderno", 10), ("Caneta", 10)]  
    r = sorted(compras, key=lambda x: (x[1], x[0]))  
    assert r == [("Caneta", 10), ("Caderno", 10)]
```

A função anônima lambda

- **Lambda** para transformar valores em uma lista



```
precos = [ 10, 20, 30]
dobrados = list(map(lambda x: x * 2, precos))
assert dobrados == [20, 40, 60]
```

- Lambda permite criar pequenas funções anônimas para transformar itens rapidamente sem declarar uma função completa.

A função anônima lambda

- **Lambda** para filtrar itens de acordo com uma condição



```
compras = [("Caneta", 10), ("Caderno", 25), ("Lápis", 5)]  
baratos = list(filter(lambda x: x[1] < 15, compras))  
assert baratos == [("Caneta", 10), ("Lápis", 5)]
```

- Lambda facilita expressar condições curtas para filtrar elementos de uma lista de forma simples e direta.

Testes com lambda

- Filtrar produtos acima de um valor mínimo usando lambda com filter()



```
def test_filtrar_produtos_caros_lambda():  
    compras = [("Caneta", 5), ("Caderno", 30), ("Mochila", 120)]  
    caros = list(filter(lambda x: x[1] > 20, compras))  
    assert caros == [("Caderno", 30), ("Mochila", 120)]
```

- A lambda retorna apenas os itens cujo valor é maior que 20, permitindo filtrar a lista.

Testes com lambda

- Ordenar por valor usando lambda



```
def test_ordenar_por_valor_lambda():  
    compras = [("Caneta", 15), ("Lápis", 5), ("Borracha", 8)]  
    resultado = sorted(compras, key=lambda x: x[1])  
    assert resultado == [("Lápis", 5), ("Borracha", 8), ("Caneta", 15)]
```

- Esta lambda extrai o valor numérico de cada tupla para permitir que a lista seja ordenada do mais barato para o mais caro.

Encerramento

- Nesta aula você aprendeu:
 - Como ordenar listas com **sorted()**
 - Como usar **key** e **lambda**
 - Como obter máximo e mínimo com **max/min**
 - Como criar consultas avançadas
 - Como testar todas essas operações usando **TDD**
 - Como integrar essas funcionalidades ao menu do AraCoins



PUC Minas

© **PUC Minas** • Todos os direitos reservados, de acordo com o art. 184 do Código Penal e com a lei 9.610 de 19 de fevereiro de 1998.

Proibidas a reprodução, a distribuição, a difusão, a execução pública, a locação e quaisquer outras modalidades de utilização sem a devida autorização da Pontifícia Universidade Católica de Minas Gerais.