Apostila: Tuplas, Conjuntos e Listas em Python

Introdução: Tipos de Coleção em Python

Em Python, as **listas** são flexíveis, mas não são a única opção. Para tarefas específicas, como proteger dados de alterações ou garantir que todos os itens são únicos, usamos as **tuplas** e os **conjuntos**. Cada tipo de dado tem uma função clara e é essencial no dia a dia de um programador.

---

1. Tuplas (Tuples): A Estrutura Imutável

Uma tupla é uma coleção ordenada de itens, mas com uma regra crucial: ela é **imutável**. Isso significa que, após a criação, você não pode alterar seus itens, nem adicionar ou remover elementos. A imutabilidade protege os dados e, por ser mais simples, as tuplas são levemente mais eficientes que as listas em certas situações.

Como Criar e Acessar uma Tupla

A sintaxe usa parênteses (). Para acessar um item, usamos o índice, assim como nas listas.

Python

```
# Criando uma tupla para coordenadas GPS
coordenadas_gps = (-23.55052, -46.633308)

# Acessando o primeiro item
print(f"Latitude: {coordenadas_gps[0]}")
```

```python
# Tentando alterar um item, o que causa um erro
try:
    coordenadas_gps[0] = -23.5
except TypeError as e:
    print(f"\nErro: {e}")
```

Métodos da Tupla: Apenas para Informação

Por serem imutáveis, as tuplas têm apenas dois métodos principais, usados para buscar informações sobre seus elementos.

- **.count(valor)**: Conta quantas vezes um valor aparece na tupla.
- **.index(valor)**: Retorna o índice da **primeira** ocorrência de um valor. Se o valor não for encontrado, um erro ValueError é gerado.

Python

```python
# Usando os métodos da tupla
dados = ("João", 25, "SP", "SP")
contagem = dados.count("SP")
print(f"\nO valor 'SP' aparece {contagem} vez(es) na tupla.")
```

Casos de Uso Reais para Tuplas

- **Coordenadas Geográficas e Pontos:** Representar um ponto em um mapa (latitude, longitude). Como a posição de um local é fixa, uma tupla é perfeita.
- **Registros de Banco de Dados:** Quando você lê uma linha de dados de um banco, ela geralmente é um registro que não deve ser modificado. Uma tupla é a escolha ideal para armazenar esse registro, como ('id_do_produto', 'nome_do_produto', 'preço').
- **Chaves de Dicionário:** Como as chaves de dicionários precisam ser imutáveis, você pode usar tuplas para chaves compostas, como ('primeiro_nome', 'sobrenome').

---

2. Conjuntos (Sets): A Estrutura de Itens Únicos

Um conjunto é uma coleção de itens **sem ordem** e **sem duplicatas**. Ele é otimizado para operações de verificação e matemática de conjuntos, como encontrar elementos em comum ou exclusivos.

Como Criar e Usar um Conjunto

A sintaxe principal usa chaves {}. Se você criar um conjunto com itens duplicados, eles serão automaticamente removidos. Lembre-se, não é possível acessar itens por índice, já que a ordem não é garantida.

Python

```
# Criando um conjunto de números com duplicatas
numeros_sorteio = {15, 8, 22, 15, 30}
print(f"Conjunto de números únicos: {numeros_sorteio}")

# Adicionando um novo item
numeros_sorteio.add(5)
print(f"Após adicionar o número 5: {numeros_sorteio}")
```

Métodos do Conjunto: Manipulação e Operações

Os conjuntos são mutáveis e oferecem métodos para gerenciar elementos e realizar operações poderosas.

Métodos de Manipulação:

- **.add(elem)**: Adiciona um elemento.
- **.remove(elem)**: Remove um elemento. Causa um erro se não existir.
- **.discard(elem)**: Remove um elemento sem causar erro se ele não estiver presente.

Métodos de Operações Matemáticas:

- **.union(outro_set)** ou |: Retorna todos os elementos de ambos os conjuntos.
- **.intersection(outro_set)** ou &: Retorna os elementos que estão em ambos.
- **.difference(outro_set)** ou -: Retorna os elementos que estão no primeiro conjunto, mas não no segundo.

Python

```
clientes_A = {"Ana", "Pedro", "João"}
clientes_B = {"João", "Maria", "Pedro"}

# Encontrando clientes em comum (interseção)
em_comum = clientes_A.intersection(clientes_B)
print(f"\nClientes em comum: {em_comum}")

# Encontrando todos os clientes únicos (união)
todos_clientes = clientes_A.union(clientes_B)
print(f"Todos os clientes: {todos_clientes}")
```

Casos de Uso Reais para Conjuntos

- **Remoção de Duplicatas:** Processar uma lista de e-mails para garantir que cada endereço seja único antes de enviar uma campanha.
- **Verificação de Pertinência:** Verificar se um item está em uma coleção de forma extremamente rápida. Por exemplo, checar se um usuário pertence a um grupo de permissões.
- **Análise de Dados:** Encontrar a interseção (elementos em comum) ou a diferença entre dois grupos de dados. Por exemplo, encontrar os clientes que compraram o produto A mas não o produto B.

---

3. Combinando Tuplas, Conjuntos e Listas

No mundo real, essas estruturas de dados raramente são usadas isoladamente. Elas se

complementam. Uma **lista de tuplas** pode ser usada para representar registros, e um **conjunto** pode ser usado para analisar esses registros de forma eficiente.

Exemplo: Analisando pedidos de uma loja. Cada pedido é uma tupla (ID_pedido, produto, status). Queremos encontrar todos os produtos únicos que foram cancelados.

Python

```
# Lista de tuplas representando os pedidos
pedidos = [
    (101, "Teclado", "Concluído"),
    (102, "Mouse", "Cancelado"),
    (103, "Monitor", "Concluído"),
    (104, "Teclado", "Cancelado")
]

# Usamos um conjunto para armazenar os produtos cancelados, garantindo unicidade
produtos_cancelados = set()

# Iteramos sobre a lista de pedidos
for pedido in pedidos:
    id_pedido, produto, status = pedido  # Desempacotamento da tupla
    if status == "Cancelado":
        produtos_cancelados.add(produto)

print(f"Produtos únicos de pedidos cancelados: {produtos_cancelados}")
```

---

4. Desafios Intermediários e Soluções Detalhadas

Desafio 1: Vendas e Descontos

Você tem uma lista de vendas (cada venda é uma tupla (produto, valor)) e uma lista de

produtos com 10% de desconto.

1. Calcule o valor total de todas as vendas.
2. Calcule o valor total das vendas dos produtos com desconto.
3. Crie uma lista com os produtos **sem desconto** que foram vendidos.

Dados:
vendas = [("Mesa", 200), ("Cadeira", 120), ("Mesa", 200), ("Luminária", 50)]
produtos_com_desconto = ["Mesa", "Luminária"]
Solução Passo a Passo:

Python

```
vendas = [("Mesa", 200), ("Cadeira", 120), ("Mesa", 200), ("Luminária", 50)]
produtos_com_desconto = ["Mesa", "Luminária"]

# Passo 1: Converter a lista de produtos com desconto para um conjunto
# Isso torna a busca por produtos com desconto mais rápida e eficiente.
descontos_set = set(produtos_com_desconto)

# Passo 2: Calcular o valor total de todas as vendas
total_vendas = 0
for produto, valor in vendas:
    total_vendas += valor
print(f"Valor total de todas as vendas: R$ {total_vendas}")

# Passo 3: Calcular o valor total das vendas com desconto
total_desconto = 0
for produto, valor in vendas:
    # Usamos o conjunto para verificar se o produto está na lista de descontos
    if produto in descontos_set:
        total_desconto += valor
print(f"Valor total das vendas com desconto: R$ {total_desconto}")

# Passo 4: Encontrar os produtos vendidos sem desconto
# Usamos um conjunto auxiliar para evitar duplicatas na lista final
produtos_sem_desconto_vendidos = []
produtos_ja_vistos = set()
for produto, _ in vendas:
    # Verifica se o produto não tem desconto e se já não foi adicionado à lista
    if produto not in descontos_set and produto not in produtos_ja_vistos:
```

```
        produtos_sem_desconto_vendidos.append(produto)
        produtos_ja_vistos.add(produto)
print(f"Produtos vendidos sem desconto: {produtos_sem_desconto_vendidos}")
```

Desafio 2: Análise de Acessos de Usuários

Você tem uma lista de acessos, onde cada item é uma tupla (usuario, recurso_acessado).

1. Crie um conjunto de todos os usuários únicos que fizeram acessos.
2. Crie um conjunto de todos os recursos únicos acessados.
3. Crie uma lista de tuplas de usuários e recursos, mas sem duplicação de acessos.

Dados:
acessos = [("Ana", "Login"), ("Pedro", "Dashboard"), ("Ana", "Login"), ("Ana", "Perfil"), ("Pedro", "Configurações")]
Solução Passo a Passo:

Python

```
acessos = [("Ana", "Login"), ("Pedro", "Dashboard"), ("Ana", "Login"), ("Ana", "Perfil"), ("Pedro", "Configurações")]

# Passo 1: Criar um conjunto de todos os usuários únicos
usuarios_unicos = set()
for usuario, _ in acessos:
    usuarios_unicos.add(usuario)
print(f"Usuários únicos: {usuarios_unicos}")

# Passo 2: Criar um conjunto de todos os recursos únicos
recursos_unicos = set()
for _, recurso in acessos:
    recursos_unicos.add(recurso)
print(f"Recursos únicos: {recursos_unicos}")

# Passo 3: Criar uma lista de tuplas de acessos únicos
# O jeito mais simples é converter a lista de tuplas para um conjunto de tuplas
# A tupla é um tipo imutável, então pode ser usada em um conjunto.
acessos_unicos = list(set(acessos))
```

```
print(f"Lista de acessos únicos: {acessos_unicos}")
```

Desafio 3: Comparação de Registros

Você tem uma lista de registros de vendas e uma lista de registros de envio. Cada registro é uma tupla (id_registro, produto).

1. Encontre os registros que estão **em ambas as listas** (vendas concluídas e já enviadas).
2. Encontre os registros que estão **na lista de vendas mas não na de envio**.

Dados:
vendas_concluidas = [("id_001", "p_a"), ("id_002", "p_b"), ("id_003", "p_c")]
registros_envio = [("id_001", "p_a"), ("id_004", "p_d"), ("id_002", "p_b")]
Solução Passo a Passo:

Python

```
vendas_concluidas = [("id_001", "p_a"), ("id_002", "p_b"), ("id_003", "p_c")]
registros_envio = [("id_001", "p_a"), ("id_004", "p_d"), ("id_002", "p_b")]

# Passo 1: Converter as listas de tuplas para conjuntos
# A tupla é um tipo de dado imutável e pode ser colocada em um conjunto.
# Isso nos permite usar os métodos de interseção e diferença.
set_vendas = set(vendas_concluidas)
set_envios = set(registros_envio)

# Passo 2: Encontrar registros em ambas as listas (interseção)
registros_em_comum = set_vendas.intersection(set_envios)
print(f"Registros de vendas concluídas e enviadas: {registros_em_comum}")

# Passo 3: Encontrar registros na lista de vendas que não estão na de envio (diferença)
pendentes_envio = set_vendas.difference(set_envios)
print(f"Registros de vendas pendentes de envio: {pendentes_envio}")
```