Listas

Listas são coleções de **elementos mutáveis**, ou seja, podem ser modificadas após a criação (adicionar, remover ou modificar elementos).

Sua sintaxe é representada por colchetes.

Exemplos:

```
lista = [1, 2, 3]

lista_mista = [1, "texto", 3.14, True, [1, 2, 3]]
```

Listas são flexíveis e podem conter qualquer tipo de dado, misturando-os conforme necessário.

Tuplas

Tuplas são coleções **imutáveis**, o que significa que, uma vez criadas, não podem ser alteradas.

Sua sintaxe é representada por parênteses.

Exemplos:

```
tupla = (1, 2, 3)

tupla_mista = (1, "texto", 3.14, False, [1, 2, 3])
```

Embora tuplas sejam imutáveis, elas podem conter qualquer tipo de dado, incluindo listas ou outras tuplas.

Operações comuns: acesso por índice

Tanto em listas quanto em tuplas, os elementos são acessados utilizando índices entre colchetes. O índice começa em 0.

```
lista = [10, 20, 30]
print(lista[1])



tupla = (11, 42, 64)
print(tupla[1])
```

Acessa o elemento na posição 1 da lista, que é 20.

Acessa o elemento na posição 1 da tupla, que é 42.

Operações comuns: slicing

O slicing permite extrair partes de uma lista ou tupla, criando uma subestrutura a partir de um intervalo de índices.

```
lista = [10, 20, 30, 40]
print(lista[1:3])
```

Extrai os elementos do índice 1 até o 2 (não inclui o índice 3), resultando em [20, 30].

```
tupla = (10, 20, 30, 40)
print(tupla[1:3])
```

Extrai os elementos do índice 1 até o 2 (não inclui o índice 3), resultando em (20, 30).

Operações comuns: operador in

O operador in permite verificar se um elemento está presente em uma lista ou tupla, retornando True caso o elemento esteja presente na coleção ou False caso contrário.

```
lista = [10, 20, 30]
print(20 in lista)
print(40 in lista)


tupla = (10, 20, 30)
print(20 in tupla)
print(40 in tupla)
```

Verifica se 20 está presente na lista, retorna True

Verifica se 40 está presente na lista, retorna False

Verifica se 20 está presente na tupla, retorna True

Verifica se 40 está presente na tupla, retorna False

Manipulação de listas

A manipulação de listas oferece uma série de métodos para adicionar, remover e ordenar elementos.

Método	Descrição	Exemplo	Resultado da lista
.append()	Adiciona um elemento ao final da lista.	lista = [1, 2, 3] lista.append(4)	[1, 2, 3, 4]
.insert()	Insere um elemento em uma posição específica.	lista = [1, 2, 3] lista.insert(1, 5)	[1, 5, 2, 3]
.remove()	Remove o primeiro elemento encontrado com o valor especificado.	lista = [1, 2, 3, 2] lista.remove(2)	[1, 3, 2]
.sort()	Ordena os elementos da lista em ordem crescente.	lista = [3, 1, 2] lista.sort()	[1, 2, 3]
.reverse()	Reverte a ordem dos elementos da lista.	lista = [1, 2, 3] lista.reverse()	[3, 2, 1]

Manipulação de listas: método pop

O método .pop() remove e retorna o elemento da posição indicada.

```
lista = [1, 2, 3]

print(lista.pop(1))
print(lista)
```

Quando pop(1) é chamado, o número 2 é removido da lista e a lista se torna [1, 3].

O primeiro print imprime o valor removido, que é 2.

O segundo print imprime a lista atualizada, que agora é [1, 3].

Concatenação de tuplas

Uma vez criadas, os elementos da tupla não podem ser alterados. Porém é possível criar uma nova tupla ao combinar tuplas existentes.

```
tupla1 = (1, 2)
tupla2 = (3, 4)
nova_tupla = tupla1 + tupla2
```

A tupla nova_tupla contém todos os elementos de tupla1 seguidos pelos elementos de tupla2, resultando em [1, 2, 3, 4].

Iteração sobre elementos

Tanto tuplas quanto listas podem ser iteradas usando laços de repetição, como o for, para acessar seus elementos um por um.

```python
lista = [1, 2, 3, 4]
for item in lista:
    print(item)


tupla = (1, 2, 3, 4)
for item in tupla:
    print(item)
```

O laço for percorre cada elemento da lista, atribuindo-o à variável item e executando a ação (como imprimir) para cada elemento, um de cada vez.

O mesmo acontece no laço for da tupla.

Desempacotamento

O desempacotamento é uma técnica que permite atribuir os valores de uma tupla ou lista a variáveis de forma mais prática.

Isso é útil quando queremos separar os elementos de uma tupla ou lista de forma direta.

```
tupla = (1, 2, 3)
a, b, c = tupla
print(a, b, c)


lista = [10, 20, 30]
x, y, z = lista
print(x, y, z)
```

Desempacota a tupla em 3 variáveis: a, b, c.

Desempacota a lista em 3 variáveis: x, y, z.

Exemplo: listas

Código que cria uma lista de tarefas, adiciona e remove itens, e imprime a lista atualizada. Como a lista de tarefas pode ser alterada, o uso de listas é ideal.

```python
tarefas = ["Estudar Python", "Fazer compras", "Lavar a louça"]

tarefas.append("Ler um livro")

tarefas.remove("Fazer compras")

for tarefa in tarefas:
    print(tarefa)
```

Exemplo: tuplas

Código onde cada cidade é representada por uma tupla que contém o nome da cidade, a latitude e a longitude. Como as coordenadas não devem ser alteradas, utilizamos tuplas, que são imutáveis.

```python
cidade1 = ("São Paulo", -23.5505, -46.6333)
cidade2 = ("Rio de Janeiro", -22.9068, -43.1729)
cidade3 = ("Brasília", -15.7801, -47.9292)

cidades = [cidade1, cidade2, cidade3]

for cidade in cidades:
    nome, latitude, longitude = cidade
    print(f"Cidade: {nome}, Latitude: {latitude}, Longitude: {longitude}")
```

O laço for percorre a lista de cidades, desempacotando cada tupla para acessar o nome, latitude e longitude de cada cidade e exibir essas informações.

alura

Compartilhe um resumo de seus novos conhecimentos em suas redes sociais.

#aprendizadoalura