CONJUNTOS DE DADOS MULTIDIMENSIONAIS

Se você considerar que um vetor tem uma dimensão, uma matriz pode ser interpretada como um vetor de vetores ou, ainda, como um array de arrays. Assim, quando você acessar o índice do primeiro array, lá, no seu elemento, estará contido um array inteiro.

Nesse segundo array, você pode realizar todo tipo de interação que pode ser feito em um array unidimensional, como remover ou adicionar elementos, iterar ou descobrir o tamanho, entre outros.

Esse pensamento pode ser propagado por quantas dimensões você desejar, aninhando arrays dentro de arrays. A seguir, temos o exemplo de uma matriz:

	0	1	2	3
0	1	2	3	4
1	2	4	6	8
2	3	6	9	12
3	4	8	12	16

Nessa matriz, o primeiro elemento matriz[0] é o vetor(1,2,3,4). O segundo elemento matriz[1] é o vetor (2,4,6,8), e assim por diante.

Caso você queira acessar o segundo elemento do terceiro vetor, deve escrever matriz[2][1], que corresponde ao elemento de valor 6.

Matrizes em Python

O uso de dados multidimensionais em Python pode ser feito, assim como no caso unidimensional, por meio do uso de listas ou de arrays com a biblioteca numpy.

O jeito mais simples de tratar dados multidimensionais em Python é utilizando listas. Elas são uma abstração fundamental da linguagem Python e podem ser aninhadas diversas vezes, ou seja, você pode fazer listas de listas e assim por diante. Além disso, as listas em Python estão preparadas para ter tipos de dados distintos, o que permite ter um elemento da lista que é uma lista e outro apenas um número ou uma palavra (“String”).

Declarando matrizes como listas

amigos =[["João",25,"Câncer"], ["Maria",23,"Áries"], ["Ana",31,"Aquário"]]

print(amigos)

Essa matriz guarda os amigos em uma lista, na qual cada amigo tem seu nome, idade e signo anotados. Repare que há um par de colchetes mais externo que contém todas as listas, e cada lista interna está separada pelo seu próprio par de colchetes. Os elementos de cada lista são separados por vírgulas.

Inserindo elementos na matriz

Para inserir um elemento na lista de nível mais alto, basta usar o mesmo comando do array unidimensional **append():**

amigos =[['João',25,'Câncer'], ['Maria',23,'Áries'], ['Ana',31,'Aquário']]

amigos.append(['Thiago',30,'Capricórnio'])

print(amigos)

Já para inserir um elemento no segundo nível de lista, você deve usar o colchete com o índice identificando qual elemento da lista de mais alto nível será alterado. Por exemplo, se quiser adicionar a cidade natal de Ana, você deve usar o código a seguir:

amigos[2].append(“Juazeiro”)

Removendo elementos da matriz

Para remover um elemento na lista de nível mais alto, você pode usar a função remove() direto na variável da lista.

amigos =[['João',25,'Câncer'], ['Maria',23,'Áries'], ['Ana',31,'Aquário']]

amigos.remove(['João',25,'Câncer'])

print(amigos)

Já para remover um elemento de segundo nível da lista, você deve utilizar a variável com o índice selecionado entre colchetes.

```
amigos =[['João',25,'Câncer'], ['Maria',23,'Áries'], ['Ana',31,'Aquário']]

amigos[2].remove(31)

print(amigos)
```

Você pode também remover um elemento por meio do seu índice, ao invés do seu valor, usando o comando pop(). A seguir temos o código que remove o primeiro elemento da lista de mais alto nível e depois remove o segundo elemento do segundo elemento restante. Dessa forma, todo o conteúdo de “João“ será removido, e depois a idade de “Ana”:

```
amigos =[['João',25,'Câncer'], ['Maria',23,'Áries'], ['Ana',31,'Aquário']]

amigos.pop(0)

amigos[1].pop(1)

print(amigos)
```

Iterando sobre a matriz

A iteração sobre uma matriz é similar ao caso do vetor, mas a cada nível de aninhamento você pode realizar uma nova iteração. Digamos que você queira apenas imprimir a lista linha a linha, com cada linha contendo os dados de cada amigo. Dessa forma, você usa um iterador para o primeiro nível da lista, como visto a seguir:

```
amigos =[['João',25,'Câncer'], ['Maria',23,'Áries'], ['Ana',31,'Aquário']]

for x in amigos:

   print(x)
```

Agora, para imprimir cada elemento em uma linha, você pode usar dois iteradores aninhados, como a seguir:

```
amigos = [['João',25, 'Câncer'], ['Maria', 23,'Áries'], ['Ana', 31, 'Aquário']]

for x in amigos:

   for y in x:

            print(y)
```

Esse código funciona bem no nosso exemplo porque todo elemento de amigos é uma lista, mas as listas podem ter diversos tipos de elementos. O que acontece com esse último código se, na lista amigos, tiver um elemento que contém apenas um nome, como “Mário”?

```
amigos =[['João',25,'Câncer'], ['Maria',23,'Áries'], ['Ana',31,'Aquário'], ['Mário']]

for x in amigos:

        for y in x:

                print(y)
```

Se você testou o código, viu que o elemento “Mário” foi tratado indevidamente como uma lista. Para resolver esse problema, você pode usar a função **isinstance(x,list)** que retorna verdadeiro se x for um objeto do tipo lista.

```
amigos=[['João',25,'Câncer'], ['Maria',23,'Áries'], ['Ana',31,'Aquário'], ['Mário']]

for x in amigos:

    if isinstance (x,list):

        for y in x:

            print(y)

    else:

        print(x)
```

Repare que esse código só usa o iterador aninhado se o item de alto nível for uma lista, o que é checado usando-se a função isinstance().

Matrizes usando arrays

Da mesma forma que no caso unidimensional, você pode preferir utilizar arrays diretamente para representar dados multidimensionais. É importante ressaltar que, diferentemente das listas, o objeto array deve ser homogêneo, ou seja, deve conter elementos de mesmo tipo.

Portanto, você deve ter um array de inteiros ou array de números de ponto flutuante (float). Desse modo, seu array pode ter muitas dimensões, mas cada dimensão será de arrays de mesmo tamanho.

Para ficar mais fácil entender, vamos começar declarando uma matriz usando array da biblioteca numpy.

Declarando matrizes como arrays

Você pode declarar uma matriz usando o comando np.array da biblioteca numpy, como anteriormente. O importante é saber que cada elemento do array de nível mais alto terá o mesmo tamanho.

```
import numpy as np

matriz1=np.array([[1,2],[3,4],[5,6]])

matriz2=np.array([[1,2],[3],[5,6]])

print(matriz1)

print(matriz2)
```

A matriz1 foi corretamente declarada e salva como um array de arrays (uma matriz). Já a matriz2 foi incorretamente declarada, pois o seu segundo elemento tem tamanho 1 ao invés de 2. Assim, ela foi salva como um array de listas. Experimente.

Para acessar determinado elemento da matriz, você pode usar a variável seguida do índice entre colchetes. Por exemplo, matriz1[0] é o vetor (1,2) já matriz1[1][1] é o elemento 4.

Operações simples com matrizes

import numpy as np

matriz1=np.array([[1,2],[3,4],[5,6]])

matriz2=np.array([[7,8],[9,10],[11,12]])

matriz2=matriz1+matriz2

print(matriz2)

matriz2=matriz2-matriz1

print(matriz2)

Esse código mostra soma e subtração de matrizes. Você pode usar outras funções simples para cálculo direto na matriz. Como exemplos, há **sum()** que soma todos os elementos da matriz, **max()** que retorna o valor máximo, **min()** que retorna o valor mínimo. As funções **mean()** e **std()** retornam a média e o desvio padrão, respectivamente.

```
import numpy as np
matriz = np.array([[1, 2], [3, 4], [5, 6]])
minimo = matriz.min()
maximo = matriz.max()
soma = matriz.sum()
media = matriz.mean()
desvio = matriz.std()
print("Minimo =", minimo)
print("Maximo =", maximo)
print("Soma =", soma)
print("Media =", media)
print("Desvio =", desvio)
```

Assim como no caso unidimensional, no caso multidimensional(matriz) você pode usar a função **enumerate(matriz)** para separar os índices e valores da matriz.