ARRAYS(VETORES) EM PYTHON

Vetores ou arrays

Em um programa, normalmente, você precisa manipular dados durante sua execução. Assim, usamos variáveis para representar um espaço na memória no qual um dado pode ser armazenado e acessado.

Mas o que fazemos quando a quantidade de dados começa a crescer? Será que precisamos gerar uma variável para cada dado? Os vetores, ou arrays, servem para resolver esse problema.

Um array é um tipo especial de variável capaz de agregar um conjunto de valores em um espaço contíguo de memória. Se você possuir uma lista de elementos, pode salvá-los todos em um array e acessá-los diretamente.

Por exemplo, se você possuir uma lista de números **[257,42,28730]** e organizá-los em um array chamado **Lista Final**, essa variável ficará disposta na memória de forma contígua.

Deslocamento	Endereço de memória	Valor
2	**OxFF750164**	**28730**
1	**OxFF750162**	**42**
0	**OxFF750160**	**257**

Elementos de um array

Um array tem dois elementos principais:

1. Seu nome, pelo qual nos referimos à variável e essencialmente guarda o valor do endereço de memória base do array.

2. Seus componentes, representados pelo seu valor e pelo seu deslocamento (também chamado índice). O deslocamento começa sua contagem em 0 e é incrementado em 1 unidade a cada novo valor adicionado.

No nosso exemplo anterior, você teria:

Lista_Final[0] valendo 257;

Lista_Final[1] valendo 42 ;

Lista_Final[2] valendo 28730;

Lembre-se de que os **índices são contados a partir do 0**, pois representam seu deslocamento em relação ao endereço base da memória.

Tamanho do array

Se o array é uma variável especial que pode guardar diversos valores, como sabemos que tamanho ocupará? Para isso, precisamos saber a quantidade de índices que o array terá e qual tipo de variável será salvo em cada índice. Multiplicando o tamanho da variável salva pelo número de índices, você terá o tamanho do array.

Por exemplo, um array de inteiros (cada um ocupando 2 bytes) com 10 índices precisará de 20 bytes de memória separado para seu uso.

Endereço relativo e endereço real

E como acessamos o terceiro elemento do array? Ao escrevermos vetor[2], o endereço base de vetor será acessado (por exemplo endereço de memória 16), depois será multiplicado o deslocamento (2) pelo tamanho de cada componente do array (2 bytes), e isso será somado ao endereço base. Com isso, o endereço real de vetor[2] será 20. Você pode ver um exemplo do vetor na tabela a seguir!

Exemplo de endereço do array

Deslocamento	Endereço de memória	Valor
4	**24**	**32**
3	**22**	**54**
2	**20**	**28730**
1	**18**	**42**
0	**16**	**257**

Usando arrays em Python

Usando listas como array

Agora que você já entendeu o que é e para que serve um array, vamos usá-lo de forma prática na linguagem Python.

Python não tem arrays de forma nativa, mas você pode utilizar o elemento list para substituí-las de forma muito simples.

Para declarar um vetor com três nomes João, Maria e Ana, basta você usar:

nomes = array["João", "Maria", "Ana"]

print(nomes)

Acessando um elemento

Para acessar um elemento diretamente, basta você escrever o nome da variável e o índice do elemento.

nomes = ['João', 'Maria', 'Ana']

x =nomes[2]

print(x)

O resultado será “Ana”, pois é o terceiro valor da lista. Aqui cabe um aviso: ao usar listas, se você tentar acessar um índice fora dos existentes na lista, seu programa causará um erro, pois você estaria tentando acessar um espaço de memória que não está alocado para aquela variável.

Descobrindo o tamanho da lista

Se você quiser saber o tamanho de uma lista em Python, pode usar o comando **len()**(que vem de length, ou comprimento em inglês).

nomes = ['João', 'Maria', 'Ana']

x =len(nomes)

print(x)

Esse código mostra como você pode acessar o tamanho de uma lista, no caso x terá o valor de 3, pois a lista tem 3 elementos.

Iterando sobre a lista

Outra necessidade comum em um array é a de percorrê-lo elemento por elemento. Em Python, você pode iterar sobre a lista (percorrer elemento por elemento)

nomes = ['João', 'Maria', 'Ana']

for x in nomes:

    print(x)

Alterando a lista

Como você pode manipular uma lista depois de criá-la? Que tipo de operações você pode querer fazer? As operações mais simples que podem ser feitas em qualquer

estrutura de dados, incluindo listas, são a adição de um elemento e a remoção de um elemento.

Adicionando um elemento

Em Python, para adicionar um elemento a uma lista já existente, basta você usar o comando **append().**

```
nomes = ['João', 'Maria', 'Ana']

print(nomes)

nomes.append('José')

print(nomes)
```

Removendo um elemento

Em Python, para remover um elemento conhecido de uma lista já existente, você pode usar o comando remove().

```
nomes = ['João', 'Maria', 'Ana']

print(nomes)

nomes.remove('João')

print(nomes)
```

Se, ao invés de remover um item por seu valor, você quiser remover pelo índice, poderá usar o comando **pop()**

```
nomes = ['João', 'Maria', 'Ana']

print(nomes)

nomes.pop(1)

print(nomes)
```

Usando arrays diretamente

Para usar arrays em Python de forma direta, você precisa importar a biblioteca **numpy** e utilizar o elemento array.

Criando um array

```
import numpy as np

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

print(arr)
```

Acessando um elemento

```
import numpy as np

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

print(arr[0])
```

Você pode também acessar os elementos contando de trás para frente, usando índices negativos. Por exemplo, arr[-1] terá o valor de 5.

Tipo de dado armazenado

Quando um array é criado, ele contém valores de determinado tipo. Se você criar um array com números inteiros, ele será de tipo inteiro. Da mesma forma, se você criar com strings, ele será automaticamente criado com strings. Você pode verificar o tipo de array com o comando **arr.dtype**

```
import numpy as np

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

print(arr.dtype)
```

Quando você cria o array, pode assinalar um tipo específico usando o comando dtype.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5], dtype='S')

print(arr.dtype)
```

Usando copy e view

Ao usar o comando copy, você cria uma cópia do array. Qualquer alteração feita na cópia não afetará o valor do array original.

```
import numpy as np

arr = np.array([1, 2, 3, 4])

x=arr.copy()

x[0]=42

print(arr)

print(x)
```

Quando você usa o comando view, está usando outra variável para se referenciar ao mesmo array. Qualquer alteração feita em uma variável alterará o valor do array original.

```
import numpy as np

arr = np.array([1, 2, 3, 4])

y=arr.view()

y[0]=43

print(arr)

print(y)
```

Iterando sobre um array

Quando você precisa percorrer o array, pode usar o laço for para iterar sobre a lista, ou seja, caminhar pela lista elemento a elemento sequencialmente.

```
import numpy as np

arr = np.array([1, 2, 3, 4])

for x in arr:

    print (x)
```

Você pode também utilizar a função enumerate() se precisar acessar um elemento por meio do seu índice. A função enumerate(array) vai criar uma lista duplicada, onde o primeiro elemento contém o índice e o segundo elemento contém o valor guardado no array.

```
import numpy as np

arr = np.array([1, 2, 3, 4])

for indice,valor in enumerate(arr):

    print (indice,valor)
```

Esse código imprimirá o valor de cada índice seguido do valor armazenado.