

CC8210 – NCA210

Programação Avançada I

Prof. Reinaldo A. C. Bianchi

Prof. Isaac Jesus da Silva

Prof. Danilo H. Perico

Conteúdo Programático

AULA	DATA	TEORIA
1	10/08/20	Introdução a disciplina. Introdução a Python :Estrutura Sequencial, Condicional e Estruturas de Repetição;
2	17/08/20	Vetores ou Listas;
3	24/08/20	Listas Aninhadas ou Matrizes;
4	31/08/20	Modularização (Funções com/sem passagem de parâmetro, com/sem retorno);
5	14/09/20	Manipulação de Strings
6	21/09/20	Manipulação de Arquivos
7	28/09/20	Dicionários e Tuplas
	05/10/20	P1
8	19/10/20	Introdução à POO - Classes e Objetos
9	26/10/20	Introdução à biblioteca Pandas
10	09/11/20	Introdução à bibliotecas matemáticas e gráficas numpy e matplotlib. Manipulação de gráficos múltiplos e de barras;
11	16/11/20	Interface Gráfica do Usuário (GUI)
12	23/11/20	P2

Relembrando a aula passada: Python

Comando **if**

- Em Python, e em várias outras linguagens de programação, o comando principal para a realização de decisões é o **if**
- Sintaxe do **if** no Python:

```
if <condição>:  
    bloco verdadeiro
```
- **If** nada mais é do que nosso **se**
 - Em português, podemos entender o comando if da seguinte forma:
 - **Se a condição for verdadeira, faça alguma coisa**

Comando **else**

- O comando **else** (senão) é utilizado nos casos em que a segunda condição é simplesmente o **contrário** da primeira.
- Sempre utilizado como uma sequência de um **if**
- Sintaxe:

```
if <condição>:  
    bloco verdadeiro  
else:  
    bloco contrário
```

Comando **while**

- O comando **while** (enquanto) serve para executarmos alguma repetição **enquanto** uma condição for verdadeira (True)
- Sintaxe:

```
while <condição>:  
    #bloco que será repetido enquanto a condição for verdadeira
```

Comando **for**

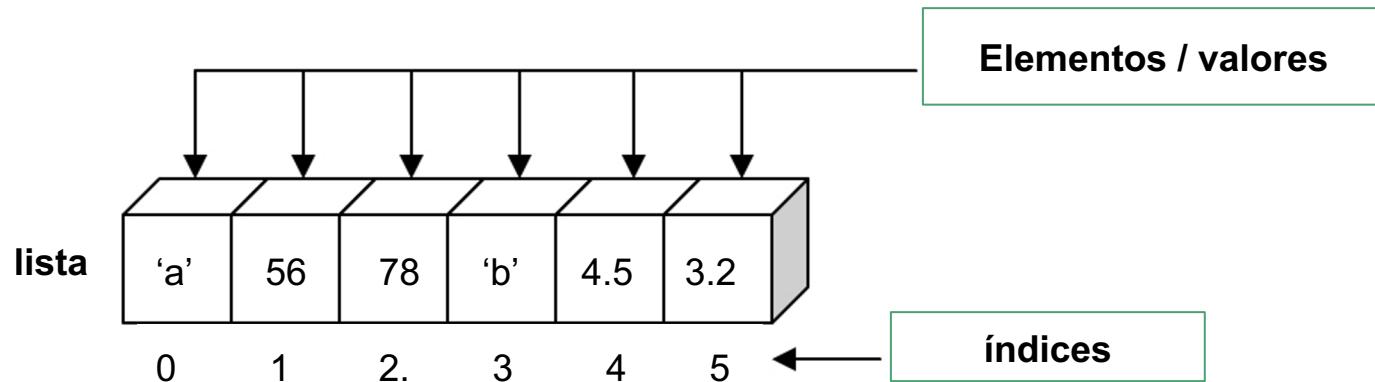
- **for** é a estrutura de repetição mais utilizada
- Sintaxe:

```
for <referência> in <sequência>:  
    #bloco de código que será repetido  
    #a cada iteração
```
- Durante a execução, a cada iteração, a referência aponta para um elemento da sequência.
- Uma vantagem do for com relação ao while é que o contador não precisa ser explícito!

Lista

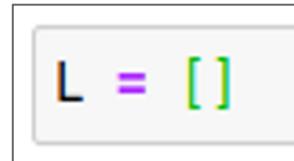
- Uma lista é uma variável que armazena um conjunto de valores.
- Lista é um tipo de variável que permite o armazenamento de valores com tipos homogêneos ou heterogêneos:
 - Homogêneos = do mesmo tipo.
 - Heterogêneo = de tipos diferentes.
- Os valores armazenados em uma lista são acessados por um índice.

Lista heterogênea



Lista

- Para indicar que uma variável é uma lista, o símbolo de colchetes [] é utilizado para delimitar o conjunto
- Sintaxe - criando uma lista chamada L:



- L é uma lista vazia

Lista - Exemplo

- Criando uma lista chamada z com 3 números inteiros

```
z = [5, 7, 1]
print(z)

[5, 7, 1]
```

- Dizemos que z tem tamanho 3

Lista - Adicionando elementos no fim da lista

- Podemos ainda adicionar novos elementos no fim da lista
- Para isto, utilizamos o método ***append(item)***
- Exemplo:

```
z = [32, 7, 1]
print(z)

[32, 7, 1]

z.append("oi")
print(z)

[32, 7, 1, 'oi']
```

Lista - Adicionando elemento em qualquer lugar

- Podemos ainda adicionar novos elementos em qualquer lugar da lista
- Para isto, utilizamos o método *insert(índice, item)*
- Exemplo:

```
z = [32, 7, 1]
print(z)
[32, 7, 1]

z.insert(1,"oi")
print(z)
[32, 'oi', 7, 1]
```

Lista - Removendo da lista pelo índice

- Podemos remover um elemento da lista
- Para isto, utilizamos o método *pop(índice)*
- Exemplo:
- Se nenhum índice for especificado, ele retorna o ultimo elemento.

```
z = ["a", "b", "c", "d", "e"]  
print(z)
```

```
['a', 'b', 'c', 'd', 'e']
```

```
z.pop(1)  
print(z)
```

```
['a', 'c', 'd', 'e']
```

Lista - Removendo da lista pelo elemento

- Podemos remover um elemento da lista
- Para isto, utilizamos o método ***remove(item)***
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]  
print(z)
```

```
['a', 'b', 'c', 'd', 'e']
```

```
z.remove("d")  
print(z)
```

```
['a', 'b', 'c', 'e']
```

```
z = [1,2,3,1,4,5,1]  
z.remove(1)  
print(z)
```

```
[2, 3, 1, 4, 5, 1]
```

Lista - Cópia

- Para criarmos uma cópia independente, utilizamos a sintaxe:

```
z1 = z[ : ]
```

- Outra opção é utilizar o método `list.copy()`.
 - Return a shallow copy of the list.
 - Equivalent to `a[:]`.

```
z = [4,5,3,6]
z1 = z[:]
print("antes da alteração na lista z")
print(z)
print(z1)
z[1] = 98
print("depois da alteração na lista z")
print(z)
print(z1)
```

```
antes da alteração na lista z
[4, 5, 3, 6]
[4, 5, 3, 6]
depois da alteração na lista z
[4, 98, 3, 6]
[4, 5, 3, 6]
```

Lista - Tamanho da lista

- Como temos os métodos para incluir e remover dados das listas, nem sempre sabemos qual é o tamanho exato que a lista tem
- Para descobrirmos o tamanho da lista, utilizamos o método ***len(lista)***
- Exemplo:

```
a = [3, 4, 5]
print(len(a))

a.append(9)
a.append(11)
print(len(a))
```

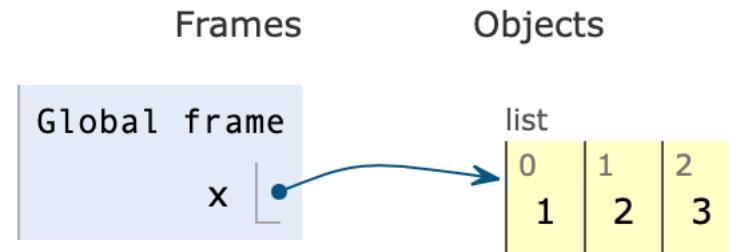
```
3
5
```

Python Tutor

- Podemos usar o Python Tutor para verificar graficamente o comportamento dos nossos programas.
 - Python Tutor helps people overcome a fundamental barrier to learning programming: understanding what happens as the computer runs each line of code.
 - You can use it to write Python, Java, C, C++, JavaScript, and Ruby code in your web browser and see its execution visualized step by step.
- <http://www.pythontutor.com>
 - Ver exemplo “hello” em Python

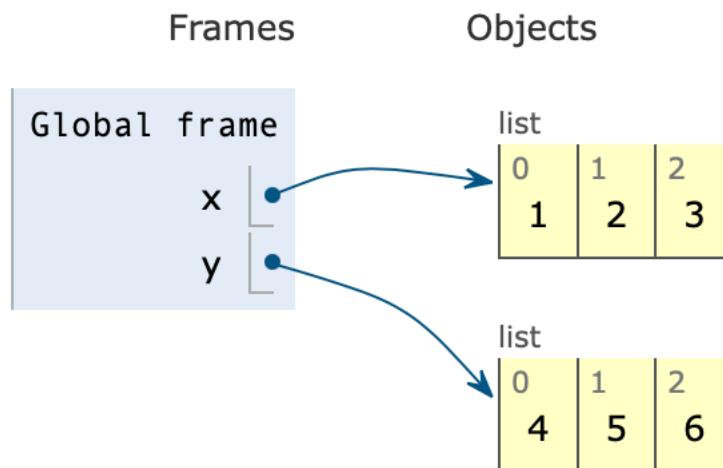
Python Tutor

- `x = [1, 2, 3]`



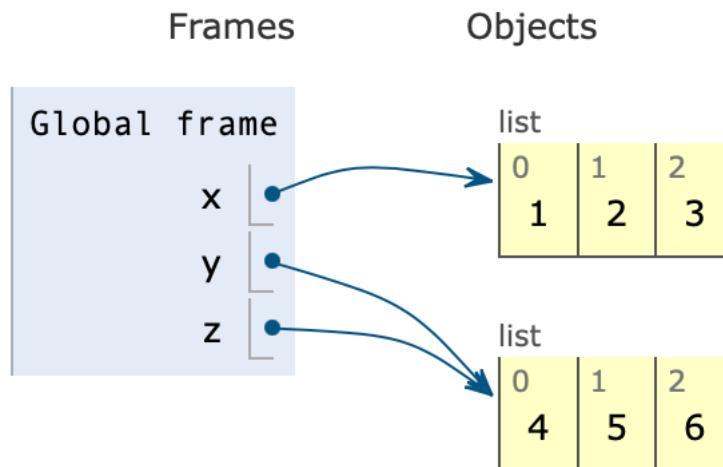
Python Tutor

- $x = [1, 2, 3]$
- $y = [4, 5, 6]$



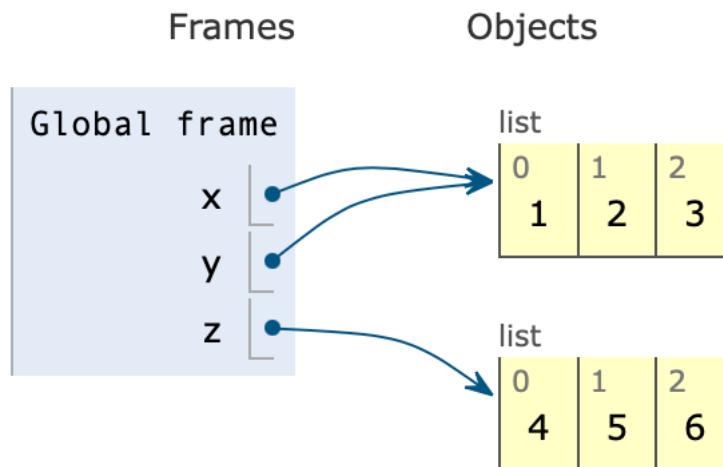
Python Tutor

- $x = [1, 2, 3]$
- $y = [4, 5, 6]$
- $z = y$



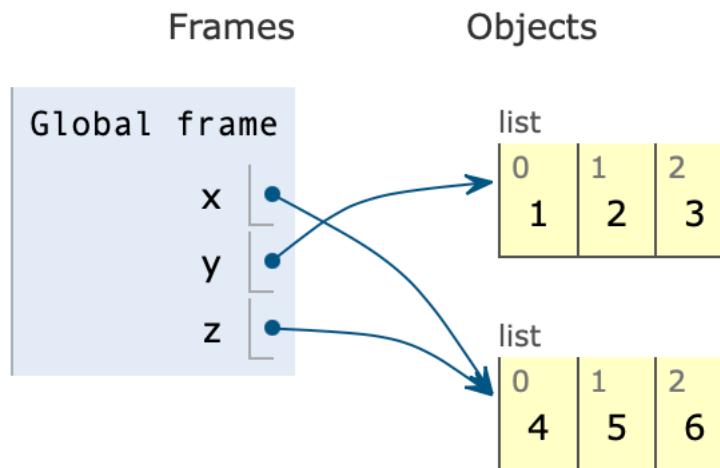
Python Tutor

- $x = [1, 2, 3]$
- $y = [4, 5, 6]$
- $z = y$
- $y = x$



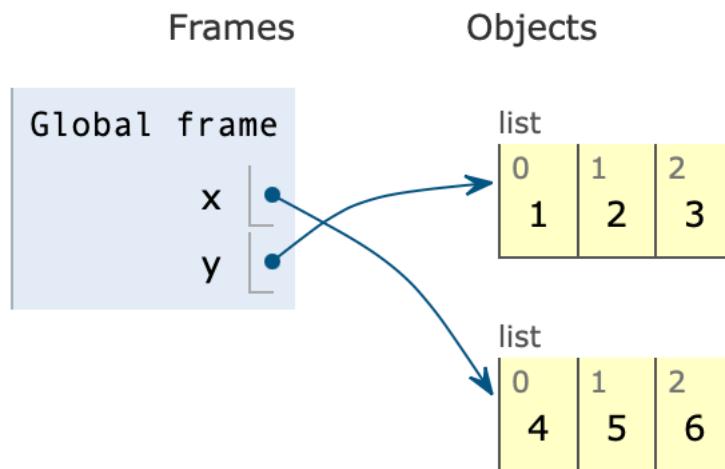
Python Tutor

- $x = [1, 2, 3]$
- $y = [4, 5, 6]$
- $z = y$
- $y = x$
- $x = z$



Python Tutor

- $x = [1, 2, 3]$
 - $y = [4, 5, 6]$
 - $z = y$
 - $y = x$
 - $x = z$
 - $\text{del}(z)$



Listas de Listas ou Listas Aninhadas

Listas Aninhadas

- Uma lista aninhada é uma lista que aparece como um elemento de uma outra lista.
- O quarto elemento da “lista” (índice 3) é uma lista aninhada
`lista = ["hello!", 6.7, 5, [1, 2]]`
- Quando pedimos para imprimir o elemento no índice 3, vemos o seguinte
`print(lista[3])`
[1, 2]

Listas Aninhadas

- Para acessar o número 2, devemos colocar o índice 3, para acessar a lista aninhada e, depois, o índice 1, para acessar o número 2:

```
lista = ["hello!", 6.7, 5, [1, 2]]
```

```
print( lista[3][1] )
```

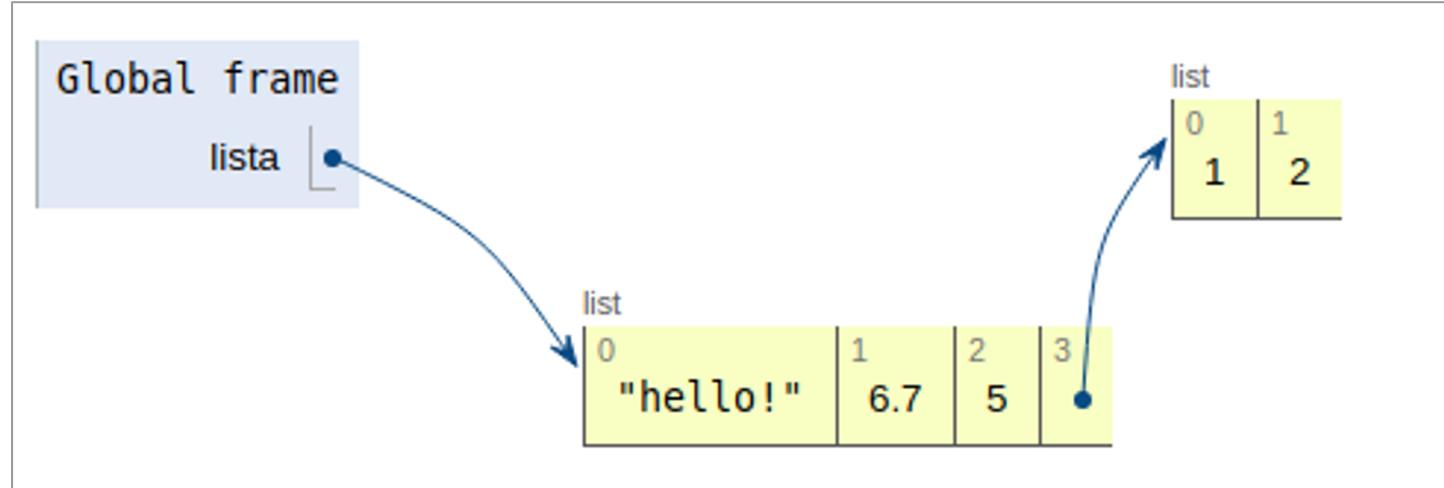
```
2
```

- Os colchetes avaliam a sentença da esquerda para a direita, então o elemento no índice 3 será acessado primeiro.
 - Depois, como o elemento no índice 3 é outra lista, podemos acessar o

Listas Aninhadas

- Podemos usar o Python Tutor para verificar graficamente o comportamento das listas:

```
lista = ["hello!", 6.7, 5, [1, 2]]
```

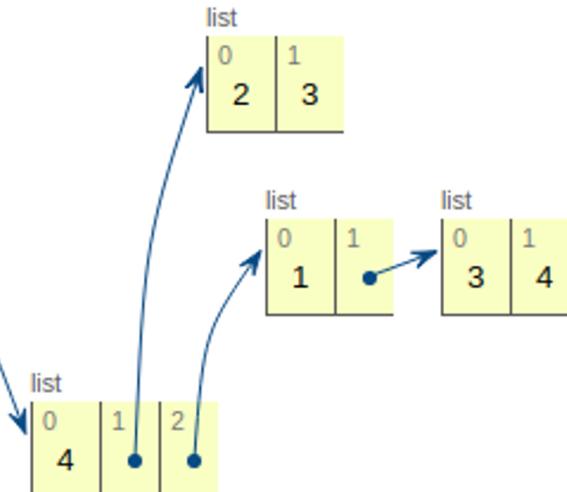


Listas Aninhadas

- Podemos aninhar listas dentro de listas aninhadas

```
lista2 = [4, [2, 3], [1, [3, 4]]]
```

Global frame
lista2



```
print( lista2[2][1] )
```

```
[3, 4]
```

```
print( lista2[2][1][0] )
```

```
3
```

```
print( lista2[2][1][1] )
```

```
4
```

Matrices

Matrizes

- Listas aninhadas podem ser utilizadas para representar matrizes.
- Uma matriz é um caso específico de lista aninhada
- Exemplo - matriz A
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$
- No Python:

```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Matrizes

- Para ficar visualmente mais simples, podemos escrever a matriz A assim:

```
A = [[1, 2, 3],  
     [4, 5, 6],  
     [7, 8, 9]]
```

```
print(A)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
print(A[0])
```

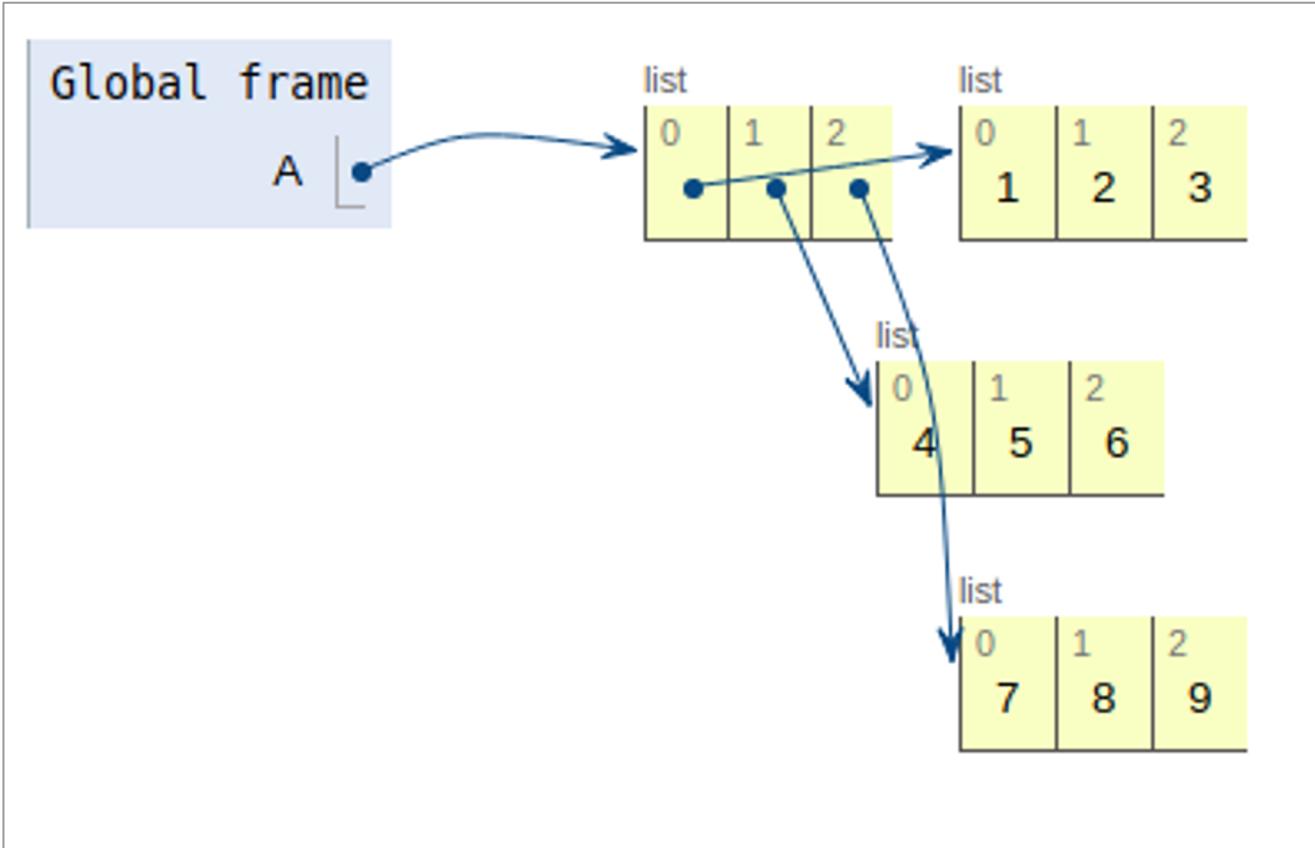
```
[1, 2, 3]
```

```
print(A[0][2])
```

```
3
```

Matrizes

- matriz A:



Matrizes

- Para ler todos os elementos de uma matriz, um de cada vez, utilizamos repetições:

```
for linha in range(len(A)):  
    for coluna in range(len(A[linha])):  
        print(A[linha][coluna], end=" ")  
    print()
```

```
1 2 3  
4 5 6  
7 8 9
```

Exemplo

- Criar uma matriz, M, 10 x 15 cujos elementos são iguais a somatória de sua linha com sua coluna (elemento = linha + coluna).

```
M = []

for num_linha in range(10):
    linha = []
    for num_coluna in range(15):
        linha.append(num_linha+num_coluna)
    M.append(linha)
```

Criando Matrizes

- Exemplo: Exibindo os elementos da matriz M:

```
for linha in range(len(M)):
    for coluna in range(len(M[linha])):
        print("%4d" % M[linha][coluna], end=" ")
    print()
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Matrizes com arrays

Vetores ou Arrays

- Além das listas, existem os vetores, ou arrays em inglês.
- An array is also a data structure that stores a collection of items.
 - Like lists, arrays are ordered, mutable, enclosed in square brackets, and able to store non-unique items.
- The Python array module requires all array elements to be of the same type.
 - To create an array, you'll need to specify a value type.

Vetores ou Arrays

- To use arrays in Python, you need to import either an array module:

```
import array as arr
```

- The code below creates na array of integers:

```
array_1 = arr.array("i", [3, 6, 9, 12])
print(array_1)
print(type(array_1))
```

Códigos para criação dos arrays de diferentes tipos

Type code	Tipo em C	Tipo em Python	Tamanho mínimo em bytes	No-tas
'b'	signed char	int	1	
'B'	unsigned char	int	1	
'u'	Py_UNICODE	Caractere unicode	2	(1)
'h'	signed short	int	2	
'H'	unsigned short	int	2	
'i'	signed int	int	2	
'I'	unsigned int	int	2	
'l'	signed long	int	4	
'L'	unsigned long	int	4	
'q'	signed long long	int	8	
'Q'	unsigned long long	int	8	
'f'	float	float	4	
'd'	double	float	8	

Matrizes com Arrays ?

- Não Existem...
 - 
- Para usar matrizes em estruturas de arrays, u arrays com maiores dimensões, teremos que usar o NumPy.

Conclusão

- Vimos como criar listas de listas
- Vimos como elas podem ser usadas como matrizes
- Vimos como manipular matrizes.
- Não existe um tipo “array” básico para criar matrizes, como existia para vetores.
 - Para matrizes numéricas usaremos o NumPy.

Fim

