Raciocínio Computacional

UNIDADE 04

Listas

Esta unidade tem por objetivo apresentar o conceito de listas em Python, atuando como estruturas de dados, tanto homogêneas quanto heterogêneas. A partir do uso de listas, será possível armazenar valores sequenciais sem a necessidade de criar um número exato de variáveis, uma vez que podem ser armazenados diversos valores acessados por índices. O uso de listas amplia exponencialmente as possibilidades em programação, como será visto a seguir.

Listas

Em linguagens de programação clássicas, como, por exemplo, Java, existe uma divisão clara entre estruturas de dados homogêneas e heterogêneas. As homogêneas são aquelas que permitem armazenamento de dados sequencialmente, todos de um mesmo tipo; vetores e matrizes em Java são exemplos clássicos deste tipo de estrutura. As heterogêneas, por sua vez, podem armazenar qualquer tipo de dado sequencialmente; *structs* do C/C++ podem ser mencionados como representantes deste tipo.

A linguagem Python possui alguns tipos de estrutura de dados, tais como: ***list*** (listas), ***tuple*** (tuplas), ***set*** e ***dictionary*** (dicionários). Contudo, nenhum deles é, necessariamente, uma estrutura de dados homogênea, ou seja, na prática, Python não apresenta o conceito de vetores e matrizes separado das estruturas-padrão da linguagem. Existe, sim, uma convenção em que o comportamento de vetores e matrizes (estruturas homogêneas) é implementado usando listas, enquanto as demais estruturas trabalham com múltiplos tipos de dados. Entretanto, cabe deixar claro que uma lista é uma estrutura de dados heterogênea, ou seja, pode armazenar qualquer tipo de dado.

Listas com comportamento de vetor

Como visto anteriormente, vetores possuem comportamento homogêneo de armazenamento de dados, ou seja, todos os dados devem ser do mesmo tipo. Para ter esse comportamento, em Python deve-se usar uma lista armazenando o mesmo tipo de dado, a qual pode ser criada com elementos iniciais colocados entre colchetes. Seguem alguns exemplos de listas simulando comportamento de vetor:

```
nums = [1, 4, 7, 2, 4, 9]
vogais = ['a', 'e', 'i', 'o', 'u']
booleanos = [True, True, False, True, False]
```

O comportamento de vetor é desejável para interagir de forma análoga sobre os dados da lista, o que normalmente é feito dentro de um laço finito.

Exemplo de aplicação 1: Elabore um programa que solicite ao usuário cinco números, armazene-os em uma lista e exiba como resultado os dados obtidos.

```
1. nums = [0, 0, 0, 0, 0]
2. for i in range(5):
3.     num = int(input("Digite um número: "))
4.     nums[i] = num
5. print(nums)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic01.py
Digite um número: 10
Digite um número: 20
Digite um número: 30
Digite um número: 40
Digite um número: 50
[10, 20, 30, 40, 50]


Process finished with exit code 0
```

Neste exemplo, na linha 01, é criada uma lista com cinco elementos zerados, os quais são substituídos dentro do laço em todas as iterações da linha 04. Contudo, o que aconteceria se no laço fossem solicitados seis números, em vez de cinco?

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic01.py
Digite um número: 10
Digite um número: 20
Digite um número: 30
Digite um número: 40
Digite um número: 50
Digite um número: 60
Traceback (most recent call last):
  File "/Users/user/projects/untitled1/First.py", line 4,
in <module>
    nums[i] = num
IndexError: list assignment index out of range


Process finished with exit code 1
```

Este é o comportamento-padrão de vetor, com os valores devendo ser inicializados para que sejam usados. Neste ponto, pode ser usado o comportamento de listas para não ocorrer esse tipo de problema, fazendo uso do método ***append*** da lista para adicionar elementos a ela.

```
1. nums = []
2. for i in range(5):
3.     num = int(input("Digite um número: "))
4.     nums.append(num)
5. print(nums)
```

Dessa forma, independentemente da quantidade de elementos, a lista é corretamente populada, uma vez que uma lista, por definição, pode conter infinitos elementos.

Exemplo de aplicação 2: Elabore um programa que solicite ao usuário cinco números e exiba duas listas, uma de números pares e outra de números ímpares.

```
1. pares = []
2. impares = []
3. for i in range(5):
4.     num = int(input("Digite um número: "))
5.     if num % 2 == 0:
6.         pares.append(num)
7.     else:
8.         impares.append(num)
9. print("Números pares:", pares, "- Números ímpares:",
   impares)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic02.py
Digite um número: 3
Digite um número: 8
Digite um número: 6
Digite um número: 11
Digite um número: 22
Números pares: [8, 6, 22] - Números ímpares: [3, 11]


Process finished with exit code 0
```

Acessando dados de uma lista

Os dados em uma lista podem ser acessados a partir de seu nome, seguido do índice do elemento entre colchetes, sendo que o índice começa em 0 e vai até o número de elementos menos 1.

Exemplo de aplicação 3: Dada a lista de dados *nums* = [1, 4, 23, 11, 8], corra a lista usando um objeto *range* e imprima cada elemento em uma linha.

```
1. nums = [1, 4, 23, 11, 8]
2. for i in range(len(nums)):
3.     print(nums[i])
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic03.py
1
4
23
11
8


Process finished with exit code 0
```

Na linha 02, para identificar o tamanho da lista, é utilizado o comando ***len***, abreviação do termo "*length*" (comprimento), ou seja, o índice vai de 0 até o comprimento da lista menos 1.

Outra forma de obter o mesmo resultado é utilizar o comando ***for*** interagindo com cada elemento da lista, como segue:

```
1. nums = [1, 4, 23, 11, 8]
2. for num in nums:
3.     print(num)
```

Exemplo de aplicação 4: Dada a lista de dados *nums* = [17, 33, 23, 11, 8, 15, 9], corra a lista e identifique o maior e o menor número.

```
1. nums = [17, 33, 23, 11, 8, 15, 9]
```

```
2. maior = nums[0]
3. menor = nums[0]
4. for num in nums:
5.     if num > maior:
6.         maior = num;
7.     if num < menor:
8.         menor = num
9. print("O maior número da lista é", maior, "e o menor é",
   menor)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic04.py
O maior número da lista é 33 e o menor é 8


Process finished with exit code 0
```

Algoritmo de ordenação

Embora uma lista em Python possua a capacidade de ordenar-se a partir de seus próprios métodos, é interessante entender como utilizar um algoritmo de ordenação de um vetor.

Para ordenar um vetor, deve-se comparar um elemento com o seguinte, verificando se atende ou não ao critério de ordenação. Em caso positivo, deve ser efetuada a troca de posição entre os elementos do vetor. A fim de correr todo o vetor, devem ser efetuadas n - 1 iterações, como segue:

```
1. for i in range(len(nums) - 1):
2.     if nums[i] > nums[i+1]:
3.         aux = nums[i]
4.         nums[i] = nums[i+1]
5.         nums[i+1] = aux
```

Neste caso, sempre é comparado um elemento com o imediatamente posterior, por isso as iterações devem ser na ordem de n - 1, para não estourar os limites do vetor. Para a troca de posições dentro do vetor, deve-se fazer uso de uma variável auxiliar,

uma vez que, quando um elemento recebe nova atribuição, não é mais possível acessar seu valor anterior.

Após a iteração sobre, por exemplo, o vetor do exemplo de aplicação 4:

```
nums = [17, 33, 23, 11, 8, 15, 9]
```

O resultado será:

```
[17, 23, 11, 8, 15, 9, 33]
```

Fazendo uso dessa iteração, é possível perceber que, na ordenação crescente, caso o maior número esteja na primeira posição do vetor, ele será deslocado para o final. Assim sendo, para ordenar esse vetor, se faz necessário realizar o processo n - 1 vezes, fazendo uso de laços aninhados.

Exemplo de aplicação 5: Dada a lista de dados *nums* = [17, 33, 23, 11, 8, 15, 9], ordene-a e mostre o resultado ao usuário.

```
1. nums = [17, 33, 23, 11, 8, 15, 9]
2. aux = 0
3. for _ in range(len(nums) - 1):
4.     for i in range(len(nums) - 1):
5.         if nums[i] > nums[i+1]:
6.             aux = nums[i]
7.             nums[i] = nums[i+1]
8.             nums[i+1] = aux
9. print(nums)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic05.py
[8, 9, 11, 15, 17, 23, 33]


Process finished with exit code 0
```

É importante observar na linha 03 que, se a variável contadora não for utilizada dentro do bloco de códigos do laço, não será necessário declará-la, fazendo uso do símbolo de ***underscore*** (_).

Listas com comportamento de matriz

Matrizes possuem comportamento análogo ao dos vetores, porém com capacidade de armazenamento multidimensional. Para os propósitos desta unidade, serão utilizadas apenas matrizes bidimensionais, ou seja, com duas dimensões.

O vetor, como visto anteriormente, possui vários elementos, sendo que cada elemento ocupa uma posição. Analogamente, uma matriz bidimensional possui vários elementos dispostos como um vetor, porém cada posição possui um vetor de elementos.

Disposição em vetor:

```
v = [17, 23, 11, 8, 15, 9, 33]
```

Disposição em matriz:

```
m = [[3, 6, 8], [11, 5, 2], [1, 1, 21]]
```

Neste caso, basicamente, tem-se um vetor **m** com os seguintes elementos:

```
m[0] = [3, 6, 8]     m[1] = [11, 5, 2]     m[2] = [1, 1, 21]
```

Exemplo de aplicação 6: Solicite ao usuário que digite três coordenadas (x, y), armazenando-as em uma matriz bidimensional.

```
1. coordenadas = []
2. for i in range(3):
3.     x = int(input("Insira um valor de x: "))
4.     y = int(input("Insira um valor de y: "))
5.     coordenadas.append([x, y])
6. print(coordenadas)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic06.py
Insira um valor de x: 1
Insira um valor de y: 1
Insira um valor de x: 2
Insira um valor de y: 2
Insira um valor de x: 3
Insira um valor de y: 3
[[1, 1], [2, 2], [3, 3]]


Process finished with exit code 0
```

Exemplo de aplicação 7: Solicite ao usuário que entre com diversos nomes, informando de forma separada nome e último sobrenome. Ao final, mostre uma lista de nomes no formato: sobrenome, nome. Para encerrar a entrada de dados, basta o usuário inserir no nome o texto “sair”.

```
1. nomes = []
2. while True:
3.     nome = input("Digite o nome: ")
4.     if nome == "sair":
5.         break
6.     sobrenome = input("Digite o sobrenome: ")
7.     nome_completo = [nome, sobrenome]
8.     nomes.append(nome_completo)
9.
10. for nome in nomes:
11.     print(nome[1] + ", " + nome[0])
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic07.py
Digite o nome: Alcir
Digite o sobrenome: Damaceno
Digite o nome: Maria da Silva
Digite o sobrenome: Chaves
Digite o nome: Henrique
Digite o sobrenome: Maciel
Digite o nome: sair
Damaceno, Alcir
Chaves, Maria da Silva
Maciel, Henrique


Process finished with exit code 0
```

É importante ressaltar que, nesses exemplos, foi simulado o comportamento de matrizes, que são estruturas homogêneas, mas as listas também possuem a capacidade de armazenar diferentes tipos de dado, como, por exemplo, um nome, uma idade etc. Entretanto, a melhor estrutura para trabalhar com registros em Python é o dicionário, que será visto na próxima unidade.

Acesso a elementos específicos de uma matriz

Assim como ocorre com vetores, que possuem a capacidade de acesso a um elemento específico a partir do seu índice, os elementos de uma matriz podem ser acessados a partir do índice de cada dimensão. No caso de uma matriz de dimensão 3x3, o elemento central poderia ser acessado a partir de:

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

Exemplo de aplicação 8: Dada a matriz m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]], efetue a soma de todos os seus elementos e exiba o resultado.

```
1. m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
2. soma = 0
3. for i in range(3):
4.     for j in range(3):
```

```
5.          soma += m[i][j]
6. print("A soma dos elementos da matriz m é igual a", soma)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic08.py
A soma dos elementos da matriz m é igual a 45


Process finished with exit code 0
```

Listas – métodos e funções built in

Métodos

Uma lista é um objeto em Python, ou seja, não faz parte dos tipos primitivos. Dessa forma, possui métodos, que são ações que podem ser realizadas. No caso, pode-se: adicionar elementos, excluir elementos, entre outros. Segue a lista de métodos que podem ser aplicados sobre listas em Python:

Listas – métodos

Método	Descrição
append()	Adiciona um elemento no fim da lista.
extend()	Adiciona todos os elementos de uma lista em outra.
insert()	Insere um elemento em determinado índice da lista.
remove()	Remove um elemento da lista.
pop()	Remove e retorna determinado elemento de um índice da

	lista.
clear()	Remove todos os elementos da lista.
index()	Retorna o índice do primeiro elemento localizado na lista.
count()	Retorna a quantidade de um elemento na lista passado como argumento.
sort()	Ordena os elementos de uma lista em ordem ascendente.
reverse()	Inverte a ordem dos elementos de uma lista.
copy()	Retorna uma cópia da lista.

Seguem exemplos de aplicação de métodos sobre listas:

```
1. nums = [17, 33, 8, 11, 8, 15, 9]
2. print(nums)
3. # adiciona o elemento 10 ao final da lista
4. nums.append(10)
5. print(nums)
6. # extend nums com os elementos de nums2
7. num2 = [3, 7]
8. nums.extend(num2)
9. print(nums)
10. # insere o elemento 0 na posição 2 da lista
11. nums.insert(2, 0)
12. print(nums)
13. # remove o elemento 11 da lista
14. nums.remove(11)
15. print(nums)
16. # remove e retorna o elemento na posição 7 da lista
17. print(nums.pop(7))
18. print(nums)
19. # cria uma cópia da nums em numsCpy
20. numsCpy = nums.copy()
21. print(numsCpy)
22. # remove todos os elementos de nums
23. nums.clear()
24. print(nums)
25. # conta quantos elementos 8 existem na lista
```

```
26. nums = numsCpy.copy()
27. print(nums.count(8))
28. # ordena nums em ordem ascendente
29. nums.sort()
30. print(nums)
31. # inverte os elementos de nums
32. nums.reverse()
33. print(nums)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/listas_metodos.py
[17, 33, 8, 11, 8, 15, 9]
[17, 33, 8, 11, 8, 15, 9, 10]
[17, 33, 8, 11, 8, 15, 9, 10, 3, 7]
[17, 33, 0, 8, 11, 8, 15, 9, 10, 3, 7]
[17, 33, 0, 8, 8, 15, 9, 10, 3, 7]
10
[17, 33, 0, 8, 8, 15, 9, 3, 7]
[17, 33, 0, 8, 8, 15, 9, 3, 7]
[]
2
[0, 3, 7, 8, 8, 9, 15, 17, 33]
[33, 17, 15, 9, 8, 8, 7, 3, 0]


Process finished with exit code 0
```

Funções built in sobre listas

Funções ***built in*** são funções internas da linguagem que podem ser utilizadas sobre determinado objeto. No caso do Python, existem algumas funções que podem ser aplicadas sobre listas, facilitando a execução de alguns algoritmos que teriam de ser desenvolvidos pelo programador para realizar as mesmas funções.

Python len()

Conforme visto anteriormente, a função ***len*** retorna a quantidade de elementos de uma lista.

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
len(nums) => 10
```

A função também é muito usada para saber quantos caracteres possui uma *string*.

```
nome = "Python"
len(python) => 6
```

Python max()

A função **max** retorna o maior valor dentro de uma lista.

```
nums = [17, 33, 8, 11, 8, 15, 9]
max(nums) => 33
```

Caso os elementos da lista sejam *strings*, a comparação é feita tomando por base a ordem alfabética.

```
heróis = ["Zorro", "Capitão América", "Hulk", "Super-Homem"]
max(heróis) => "Zorro"
```

Python min()

A função **min** retorna o menor valor dentro de uma lista.

```
nums = [17, 33, 8, 11, 8, 15, 9]
min(nums) => 8
```

Da mesma forma que a função **max**, caso os elementos da lista sejam *strings*, a função **min** efetua a comparação tomando por base a ordem alfabética.

```
heróis = ["Zorro", "Capitão América", "Hulk", "Super-Homem"]
max(heróis) => "Capitão América"
```

Python sorted()

A função ***sorted*** retorna a lista passada em ordem ascendente.

```
nums = [17, 33, 8, 11, 8, 15, 9]
sorted(nums) => [8, 8, 9, 11, 15, 17, 33]
```

Também é possível fazer a ordenação em ordem descendente, por meio do parâmetro ***reverse***.

```
nums = [17, 33, 8, 11, 8, 15, 9]
sorted(nums, reverse=True) => [33, 17, 15, 11, 9, 8, 8]
```

Ainda, pode-se aplicar a função ***sorted*** sobre uma *string*, retornando uma lista não editável, no caso, uma tupla, como será visto na próxima unidade.

```
nome = "José dos Santos"
```

```
sorted(nome) => [' ', ' ', 'J', 'S', 'a', 'd', 'n', 'o', 'o',
'o', 's', 's', 's', 't', 'é']
```

Python sum()

A função ***sum*** retorna a soma de todos os elementos de uma lista.

```
nums = [17, 33, 8, 11, 8, 15, 9]
sums(nums) => 101
```

Exemplo de aplicação 9: Em uma competição de saltos ornamentais, são obtidas dos jurados sete notas, sendo eliminadas a maior e a menor. A valoração do salto é feita pela soma das demais notas. Crie um programa que receba as notas dos juízes, remova a maior e menor nota e some as demais, fazendo uso de métodos de listas e funções *built in*.

```
1. notas = []
2. for _ in range(7):
3.     nota = float(input("Entre com uma das notas: "))
4.     notas.append(nota)
5. menor = min(notas)
6. if notas.count(menor) == 1:
7.     notas.remove(menor)
8. else:
9.     indice = -1
10.    for i in range(len(notas)):
11.        if notas[i] == menor:
12.            indice = i
13.            break
14.    notas.pop(indice)
15. maior = max(notas)
16. if notas.count(maior) == 1:
17.    notas.remove(maior)
18. else:
19.    indice = -1
20.    for i in range(len(notas)):
21.        if notas[i] == maior:
22.            indice = i
23.            break
24.    notas.pop(indice)
25. soma = sum(notas)
26. print(f"A pontuação final do salto foi {soma:.1f}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic9.py
Entre com uma das notas: 7.2
Entre com uma das notas: 7.6
Entre com uma das notas: 6.3
Entre com uma das notas: 9.2
Entre com uma das notas: 7.8
Entre com uma das notas: 9.2
Entre com uma das notas: 7.8
A pontuação final do salto foi 39.6


Process finished with exit code 0
```

Uma funcionalidade importante deste exemplo é verificar se a maior ou menor nota aparece mais de uma vez na lista, uma vez que o método ***remove*** apaga todas as ocorrências daquele elemento. Assim, se faz necessário correr toda a lista, buscando a primeira ocorrência da respectiva nota para ser removida pela função ***pop***, a qual usa o índice do elemento para isso.

EXERCÍCIO

Listas com comportamento de vetor

Exercício de fixação 1: Crie um programa que solicite ao usuário seis números, calculando a média, e mostre uma lista com os números iguais ou acima da média e uma lista com os números abaixo da média.

Resolução do exercício

Resolução exercício 1

```
1. nums = []
2. maiores = []
3. menores = []
4. soma = 0
5. for _ in range(6):
```

```
    num = int(input("Informe um dos 6 números: "))
    nums.append(num)
    soma += num
media = soma / 6
for num in nums:
    if num >= media:
        maiores.append(num)
    else:
        menores.append(num)
print("A média dos 6 números é", media)
print("Os números iguais ou maiores que a média são", maiores)
print("Os números menores que a média são", menores)
```

C:\

```
/Users/user/untitled1/bin/python /Users/user/projects/untitled1/ExFix01.py
Informe um dos 6 números: 6
Informe um dos 6 números: 9
Informe um dos 6 números: 2
Informe um dos 6 números: 4
Informe um dos 6 números: 3
Informe um dos 6 números: 7
A média dos 6 números é 5.166666666666667
Os números iguais ou maiores que a média são [6, 9, 7]
Os números menores que a média são [2, 4, 3]

Process finished with exit code 0
```

Exercício de fixação 2: Crie um programa que solicite ao usuário duas listas com cinco elementos cada e, como resultado, crie uma terceira lista que intercale os elementos das anteriores.

Resolução do exercício

Resolução exercício 2

```
lista1 = []
lista2 = []
intercalados = []
for _ in range(5):
    num = int(input("Informe um elemento da
lista 1: "))
    lista1.append(num)
for _ in range(5):
    num = int(input("Informe um elemento da
lista 2: "))
    lista2.append(num)
for i in range(5):
    intercalados.append(lista1[i])
    intercalados.append(lista2[i])
print("Elementos intercalados:",
intercalados)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix02.py
Informe um elemento da lista 1: 2
Informe um elemento da lista 1: 6
Informe um elemento da lista 1: 4
Informe um elemento da lista 1: 9
Informe um elemento da lista 1: 3
Informe um elemento da lista 2: 6
Informe um elemento da lista 2: 4
Informe um elemento da lista 2: 1
Informe um elemento da lista 2: 1
Informe um elemento da lista 2: 1
Elementos intercalados: [2, 6, 6, 4, 4, 1,
9, 1, 3, 1]


Process finished with exit code 0
```

Exercício de fixação 3: Crie um programa que leia as temperaturas médias de cada mês do ano e as armazene em uma lista; use outro vetor para guardar e exibir, quando necessário, o nome dos meses do ano; calcule a média anual de temperatura e informe quais meses ficaram acima da média anual.

Resolução exercício 3

```
meses = [
    "janeiro", "fevereiro", "março", "abril",
    "maio", "junho", "julho", "agosto",
    "setembro", "outubro", "novembro",
"dezembro"
]
temperaturas = []
soma = 0
for mes in meses:
    temperatura = float(input("Temperatura
média em " + mes + ": "))
    temperaturas.append(temperatura)
    soma += temperatura
media = soma / 12
print(f"Meses com temperatura acima da média
de {media:.1f} graus:")
for i in range(12):
    if temperaturas[i] > media:
        print(f"{meses[i]}:
{temperaturas[i]:.1f} graus")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix03.py
Temperatura média em janeiro: 30
Temperatura média em fevereiro: 28
Temperatura média em março: 26
Temperatura média em abril: 24
Temperatura média em maio: 24
Temperatura média em junho: 21
Temperatura média em julho: 22
Temperatura média em agosto: 24
Temperatura média em setembro: 23
Temperatura média em outubro: 25
Temperatura média em novembro: 27
Temperatura média em dezembro: 28
Meses com temperatura acima da média de
25.2 graus:
janeiro: 30.0 graus
fevereiro: 28.0 graus
março: 26.0 graus
novembro: 27.0 graus
dezembro: 28.0 graus


Process finished with exit code 0
```

Listas com comportamento de matriz

Exercício de fixação 4: Crie um programa que efetue a soma de duas matrizes 3x3, sabendo que a soma desse tipo de matriz é uma nova matriz 3x3, sendo cada elemento resultado da soma dos elementos das matrizes na mesma posição. Exemplo:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 3 & 2 & 3 \\ 1 & 3 & 3 \\ 0 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 & 6 \\ 5 & 8 & 9 \\ 7 & 10 & 11 \end{bmatrix}$$

#ParaTodosVerem Abre colchetes linha com 1 2 3, linha abaixo com 4 5 6, linha abaixo com 7 8 9, fecha colchetes. Sinal de adição e abre novo colchetes. Linha com 3 2 3, linha abaixo com 1 3 3,

linha abaixo com 0 2 2, fecha colchetes. Sinal de igual e abre novo colchetes com linha com 4 4 6, linha abaixo com 5 8 9, linha abaixo com 7 10 11, fecha colchetes.

 Resolução do exercício 

Resolução exercício 4

```
matrizA = []
matrizB = []
matrizSoma = []
for i in range(3):
    matrizA.append([])
    for _ in range(3):
        num = int(input("Elemento da matriz A: "))
        matrizA[i].append(num)
for i in range(3):
    matrizB.append([])
    for _ in range(3):
        num = int(input("Elemento da matriz B: "))
        matrizB[i].append(num)
for i in range(3):
    matrizSoma.append([])
    for j in range(3):
        num = matrizA[i][j] + matrizB[i][j]
        matrizSoma[i].append(num)
print("Matriz A:", matrizA)
print("Matriz A:", matrizB)
print("Matriz A:", matrizSoma)
```

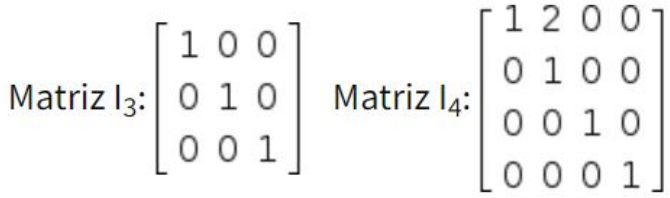
C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix04.py
Elemento da matriz A: 1
Elemento da matriz A: 2
Elemento da matriz A: 3
Elemento da matriz A: 4
Elemento da matriz A: 5
Elemento da matriz A: 6
Elemento da matriz A: 7
Elemento da matriz A: 8
Elemento da matriz A: 9
Elemento da matriz B: 3
Elemento da matriz B: 2
Elemento da matriz B: 3
Elemento da matriz B: 1
Elemento da matriz B: 3
Elemento da matriz B: 3
Elemento da matriz B: 0
Elemento da matriz B: 2
Elemento da matriz B: 2
Matriz A: [[1, 2, 3], [4, 5, 6], [7, 8,
9]]
Matriz A: [[3, 2, 3], [1, 3, 3], [0, 2,
2]]
Matriz A: [[4, 4, 6], [5, 8, 9], [7, 10,
11]]


Process finished with exit code 0
```

Exercício de fixação 5:

A matriz-identidade é uma matriz com a mesma quantidade de linhas e colunas, apresentando todos os elementos da diagonal principal (de cima para baixo, da esquerda para a direita) iguais a 1 e os demais elementos iguais a 0. Crie um programa que solicite o tamanho da matriz desejada, gerando a matriz-identidade. Exemplo:



#ParaTodosVerem Matriz I_3: abre colchetes, linha com 1 0 0, linha abaixo com 0 1 0, linha abaixo com 0 0 1. Ao lado, Matriz I_4: abre colchetes, linha com 1 2 0 0, linha abaixo com 0 1 0 0, linha abaixo com 0 0 1 0, linha abaixo com 0 0 0 1, fecha colchetes.

Resolução do exercício

Resolução exercício 5

```
identidade = []
tamanho = int(input("Qual é o tamanho da
matriz-identidade desejada? "))
for i in range(tamanho):
    identidade.append([])
    for j in range(tamanho):
        if i == j:
            identidade[i].append(1)
        else:
            identidade[i].append(0)
for linha in identidade:
    print(linha)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix05.py
Qual é o tamanho da matriz-identidade
desejada? 5
[1, 0, 0, 0, 0]
[0, 1, 0, 0, 0]
[0, 0, 1, 0, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 0, 1]


Process finished with exit code 0
```

Listas – métodos e funções built in

Exercício de fixação 6: Dado o vetor *nums* = [3, 7, 2, 9, 5, 6], crie um programa que, em uma linha, calcule a média dos seus elementos.

 Resolução do exercício 

Resolução exercício 6

```
1. nums = [3, 7, 2, 9, 5, 6]
2. media = sum(nums)/len(nums)
3. print("A média dos elementos do vetor é",
   media)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix06.py
A média dos elementos do vetor é
5.333333333333333


Process finished with exit code 0
```

Exercício de fixação 7: Dado o vetor *nums* = [3, 11, 6, 32, 15, 22, 4, 10, 5], crie uma matriz 3x3 com os três maiores elementos na primeira linha, os três elementos intermediários na segunda linha e os elementos menores na terceira linha.

Resolução do exercício

Resolução exercício 7

```
1. nums = [3, 11, 6, 32, 15, 22, 4, 10, 5]
2. linha1 = []
3. linha3 = []
4. matriz = []
5. for _ in range(3):
6.     linha1.append(nums.pop(nums.index(max(nums)))
   )
7.     linha1.sort()
8. for _ in range(3):
9.     linha3.append(nums.pop(nums.index(min(nums)))
   )
10.     linha3.sort()
11. nums.sort()
12. matriz.append(linha1)
13. matriz.append(nums)
14. matriz.append(linha3)
15. print(matriz)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix07.py
[[15, 22, 32], [6, 10, 11], [3, 4, 5]]


Process finished with exit code 0
```

Desafio

Exercício de fixação 8: Crie um programa que solicite o nome de quatro times de futebol e simule partidas de forma que cada time jogue uma vez com os outros três. Na partida, deve perguntar quantos gols fez cada time e somar as devidas pontuações. Ao final, deve dizer quais times foram campeões, uma vez que pode haver empate em pontuação. Observação: vitória vale 3 pontos para o vencedor, empate vale 1 ponto para cada time e derrota não soma pontos.

 Resolução do exercício 

Resolução exercício 8

```
 1. times = []
 2. tabela = [0, 0, 0, 0]
 3.
 4. for _ in range(4):
 5.     time = input("Digite o nome de um dos
    times: ")
 6.     times.append(time)
 7. for i in range(3):
 8.     for j in range(i+1,4):
 9.         print("Jogo:", times[i], "x",
    times[j])
10.         gols1 = int(input("Gols do " +
    times[i] + ": "))
11.         gols2 = int(input("Gols do " +
    times[j] + ": "))
```

```
        if gols1 > gols2:
            tabela[i] += 3
        elif gols1 < gols2:
            tabela[j] += 3
        else:
            tabela[i] += 1
            tabela[j] += 1
print("*** Pontuação dos Times ***")
for i in range(4):
    print(times[i] + ": ", tabela[i])
maior = max(tabela)
print("****** Time Campeão ******")
for i in range(4):
    if tabela[i] == maior:
        print(times[i])
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix08.py
Digite o nome de um dos times: Força FC
Digite o nome de um dos times: Raça FC
Digite o nome de um dos times: Habilidade
FC
Digite o nome de um dos times: Dedicação
FC
Jogo: Força FC x Raça FC
Gols do Força FC: 2
Gols do Raça FC: 1
Jogo: Força FC x Habilidade FC
Gols do Força FC: 2
Gols do Habilidade FC: 5
Jogo: Força FC x Dedicação FC
Gols do Força FC: 0
Gols do Dedicação FC: 0
Jogo: Raça FC x Habilidade FC
Gols do Raça FC: 2
Gols do Habilidade FC: 2
Jogo: Raça FC x Dedicação FC
Gols do Raça FC: 1
Gols do Dedicação FC: 3
Jogo: Habilidade FC x Dedicação FC
Gols do Habilidade FC: 3
Gols do Dedicação FC: 0
*** Pontuação dos Times ***
Força FC:  4
Raça FC:  1
Habilidade FC:  7
Dedicação FC:  4
****** Time Campeão ******
Habilidade FC


Process finished with exit code 0
```

Neste vídeo, veremos o uso das estruturas de lista em conjunto com a função *enumerate()*.

Conclusão

A respeito do uso de listas na programação em Python, podemos concluir que:

- As listas são estruturas de dados heterogêneas.
- Entretanto, são normalmente usadas como estruturas de dados homogêneas.
- A linguagem Python não possui formalmente estruturas de vetor e matriz.
- Com listas, é possível simular o comportamento de vetores e matrizes.
- Os elementos de uma lista podem ser acessados a partir do nome dela, seguido do índice do elemento entre colchetes.
- Uma matriz é basicamente um vetor que apresenta em cada um de seus elementos um novo vetor.
- Os elementos de uma matriz podem ser acessados pelo nome dela, seguido dos índices de cada dimensão colocados individualmente entre colchetes.
- Listas podem ser modificadas a partir de métodos próprios e funções ***built in***.

Referências

BANIN, S. L. **Python 3**: conceitos e aplicações uma abordagem didática. São Paulo: Érica, 2018.

PYTHON SOFTWARE FOUNDATION. **Documentação Python 3.9.0**. Disponível em: https://docs.python.org/pt-br/3/. Acesso em: 27 out. 2020.