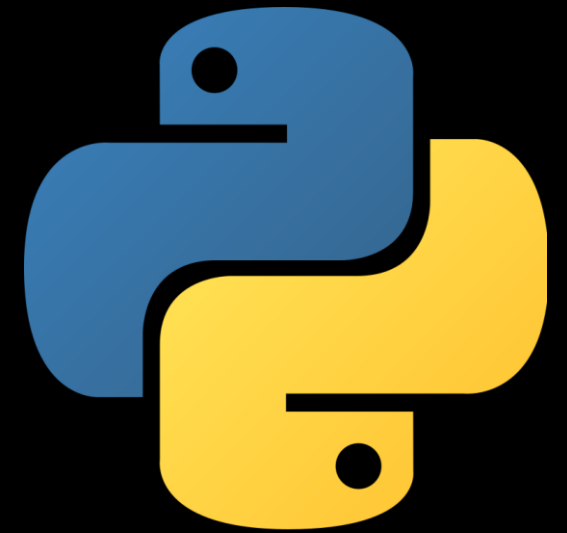


PYTHON – NOTAS DE ESTUDO



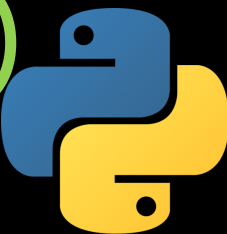
Por Thiago Barros

Github - <https://github.com/Tbarros1996>



Numpy é uma biblioteca de funções do Python que serve para trabalharmos com matemática (Algébrica, Matricial). Tem como objetivo de ser usual com a área científica.

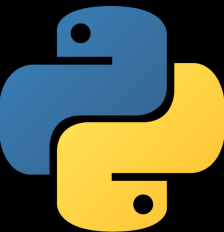
Numpy deve trabalhar em conjunto com outras bibliotecas para obtenção de resultados, plotagem de gráficos entre outros problemas (Matplotlib, Scipy, Math, Cmath, Pandas..)





Fonte de Documentação e Informação:

<https://numpy.org>

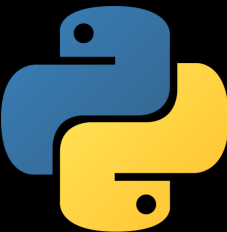




Parte 1: Instalação da Biblioteca

IDE: Pycharm

Repositório: PyPi (Python Package Index)





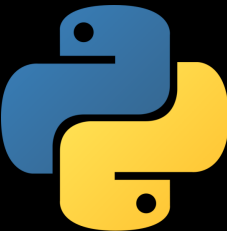
Revisão: Estruturas de Dados do Python

TIPOS DE DADOS

NUMERICOS
TEXTUAIS
LOGICOS

TIPOS DE ESTRUTURAS BÁSICAS DE DADOS DO PYTHON

LISTAS []
SETS {}
TUPLAS ()
DICIONÁRIOS {}
(Entre Outras)

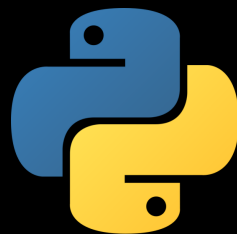




Sets ---- “Tipo de Lista” que não pode ter elementos repetidos. Parcialmente imutável, mas possível de ser inserido informação com função.

```
> z = {1, 2, 3, 4, 5, 56}  
> z_2 = set([1, 2, 3, 4, 5, 65, 56])
```

```
▼ z = {set: 6} {1, 2, 3, 4, 5, 56}  
  01 {int} 1  
  01 {int} 2  
  01 {int} 3  
  01 {int} 4  
  01 {int} 5  
  01 {int} 56  
  01 __len__ = {int} 6  
▼ z_2 = {set: 7} {1, 2, 3, 4, 5, 65, 56}  
  01 {int} 1  
  01 {int} 2  
  01 {int} 3  
  01 {int} 4  
  01 {int} 5  
  01 {int} 65  
  01 {int} 56  
  01 __len__ = {int} 7
```





Listas ---- Conjunto de elementos de qualquer tipo ordenado, ou não, mas apenas em uma linha


```
lista_1 = [1, 2, 3, 4, "batata", "Morte ao Arroz com Uva Passa"]
```

```
1 lista_1 = {list: 6} [1, 2, 3, 4, 'batata', 'Morte ao Arroz com Uva Passa']
01 0 = {int} 1
01 1 = {int} 2
01 2 = {int} 3
01 3 = {int} 4
01 4 = {str} 'batata'
01 5 = {str} 'Morte ao Arroz com Uva Passa'
01 __len__ = {int} 6
```

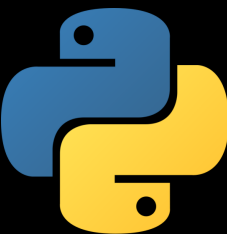




Tupla ---- “Lista” Que é imutável. Uma vez declarada, será do jeito que está sempre

```
▼  tupla_do_feijão = {tuple: 3} ('Feijão Preto', 'Feijão Carioca', 'Feijão Mulato')  
  01 0 = {str} 'Feijão Preto'  
  01 1 = {str} 'Feijão Carioca'  
  01 2 = {str} 'Feijão Mulato'  
  01 __len__ = {int} 3
```

```
>>> tupla_do_feijão = ("Feijão Preto", "Feijão Carioca", "Feijão Mulato")  
>>> type(tupla_do_feijão)
```





Dicionário ---- “Tipo de Lista” que pode ser trabalhada com índices(Que são Inteiros), e que pode ser usada com referência.

```
>>> lista_de_compras = {0:"Batata",1:"Feijão",2:"Carne"}  
>>> type(lista_de_compras)  
<class 'dict'>
```

```
lista_de_compras = {dict: 3} {0: 'Batata', 1: 'Feijão', 2: 'Carne'}  
0 = {str} 'Batata'  
1 = {str} 'Feijão'  
2 = {str} 'Carne'  
__len__ = {int} 3
```



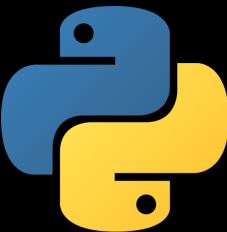


```
<class 'numpy.ndarray'>
```

Tipo de Objeto do Numpy (Classe) : *ndarray*

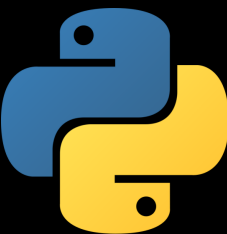
O Python não possui funções para trabalhar com matrizes (Apesar das listas, operações matriciais não funcionam para elas)

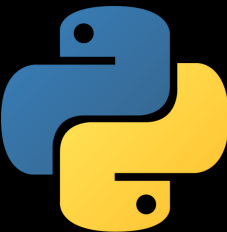
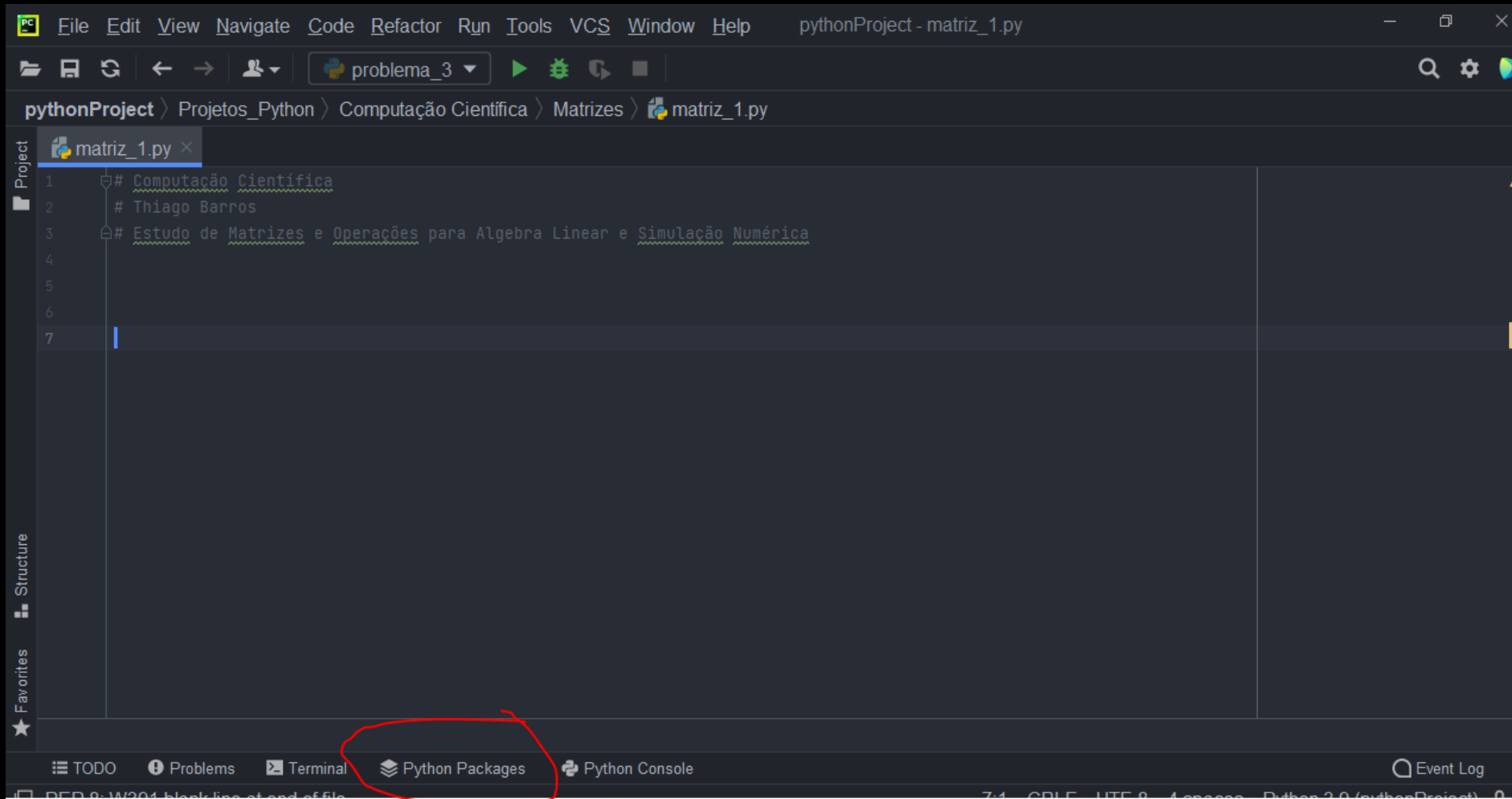
Numpy NÃO torna o Python um MATLAB, mas o propósito é trabalhar com Matrizes tão bem como software da Mathworks





List	Array
Can consist of elements belonging to different data types	Only consists of elements belonging to the same data type
No need to explicitly import a module for declaration	Need to explicitly import a module for declaration
Cannot directly handle arithmetic operations	Can directly handle arithmetic operations
Can be nested to contain different type of elements	Must contain either all nested elements of same size
Preferred for shorter sequence of data items	Preferred for longer sequence of data items
Greater flexibility allows easy modification (addition, deletion) of data	Less flexibility since addition, deletion has to be done element wise
The entire list can be printed without any explicit looping	A loop has to be formed to print or access the components of array
Consume larger memory for easy addition of elements	Comparatively more compact in memory size







Python Packages - pythonProject

Python Packages

Search for more packages » Add Package

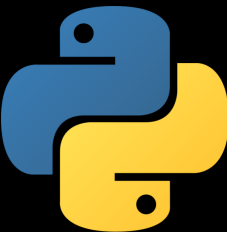
▼ Installed

sympy	1.8
six	1.16.0
simupy	1.0.0
setuptools	57.4.0
scipy	1.7.0
pytz	2021.1
python-dateutil	2.8.2
pyserial	3.5
pyparsing	2.4.7
pip	21.2.1
Pillow	8.3.1
pandas	1.3.1
numpy	1.21.1
mpmath	1.2.1
matplotlib	3.4.2
kiwisolver	1.3.1
ikpy	3.2.2
cycler	0.10.0

▼ PyPI repository

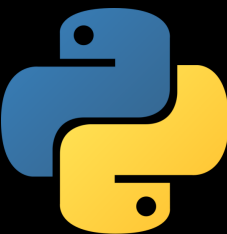
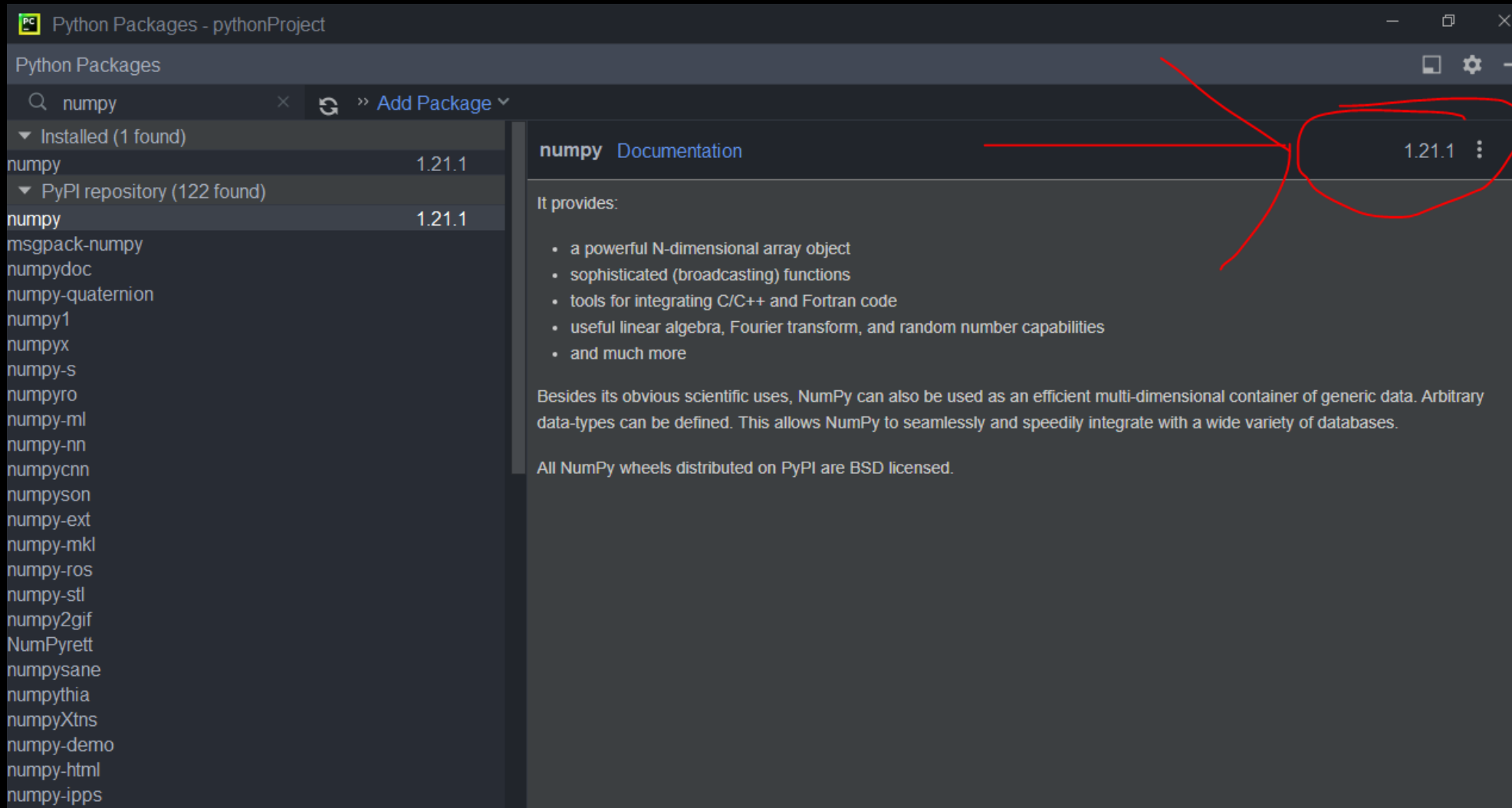
urllib3	
six	1.16.0
botocore	
setuptools	57.4.0
requests	
certifi	
idna	

Select a package to view documentation





Vai aparecer um botão de instalação, caso contrário ele já foi instalado e vai mostrar qual versão está instalada no sistema.





Parte 2: Importando para o Texto, o Numpy

```
import numpy as np
```

(as) – Palavra chave do Python que serve para colocar um apelido na biblioteca (Ajuda a Deixar o Código mais Simples)





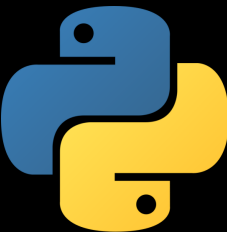
Parte 3: Criando Uma Matriz Linha

1	2
3	4
5	6

NOTAÇÃO
MATRICIAL

$$m \times n$$

ESSA MATRIZ TEM 3 LINHAS E 2 COLUNAS





```
numpy.array(object, dtype=None, *, copy=True, order='K', subok=False, ndmin=0,  
            like=None)
```

1º Argumento – Obrigatório – Matriz

`np.array` ([1ª Linha], [2ª Linha], [3ª Linha].... [Linha m],
[dtype=formato dos dados])

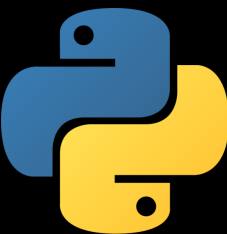




1º Forma = Com uma Variável

```
import numpy as np  
  
x = np.array([[1, 2], [3, 4], [5, 6]])  
print(x)
```

(np.array) ← -----
Sempre vai ser usada
para se criar matriz



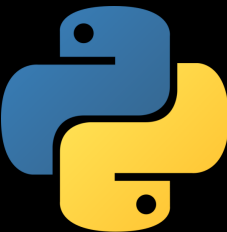


2º Forma = Convertendo uma Lista
no formato matricial em Matriz

```
import numpy as np

x = ([[1, 2], [3, 4], [5, 6]])
print(x)

y = np.array(x)
print(y)
```



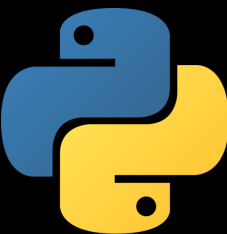


```
import numpy as np

x = [[1, 2, 3], [4, 5, 6]]
print(x)

y = np.array(x, dtype=float)
print(y)
```

```
[[1, 2, 3], [4, 5, 6]]
[[1. 2. 3.]
 [4. 5. 6.]]
```





Exercício – Crie as Seguintes Matrizes

1) $\begin{bmatrix} 3 & 5 & -1 \\ 0 & \frac{4}{5} & \sqrt{2} \end{bmatrix}$ é matriz 2×3

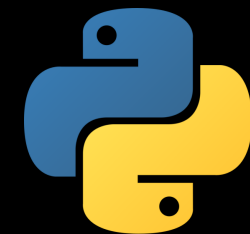
2) $\begin{bmatrix} 4 & -3 \\ \frac{3}{7} & 2 \\ 4 & 1 \end{bmatrix}$ é matriz 3×2

3) $\begin{bmatrix} 0 & -9 & -1 & 7 \end{bmatrix}$ é matriz 1×4

4) $\begin{bmatrix} 5 \\ 1 \\ -3 \end{bmatrix}$ é matriz 3×1

5) $\begin{bmatrix} 1 & 2 \\ 3 & 7 \end{bmatrix}$ é matriz 2×2

$$c = \begin{bmatrix} 1.1 & -3.2 & 3.4 & 0.6 \\ 0.6 & 1.1 & -0.6 & 3.1 \\ 1.3 & 0.6 & 5.5 & 0.0 \end{bmatrix}$$

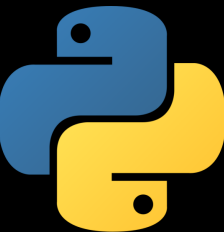




```
import numpy as np
import math as mt

matriz_1 = np.array([[3, 5, -1], [0, (4 / 5), mt.sqrt(2)]])
print(matriz_1)
```

```
[[ 3.          5.         -1.          ]
 [ 0.          0.8        1.41421356]]
```

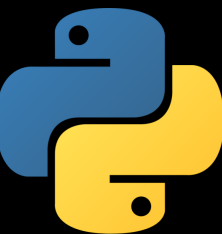




```
matriz_2 = np.array([[4, -3], [(3 / 7), 2], [4, 1]])  
print(matriz_2)
```

```
[[ 4.         -3.         ]  
 [ 0.42857143  2.         ]  
 [ 4.          1.         ]]
```

```
matriz_2 = np.array([[4, -3], [(3 / 7), 2], [4, 1]], dtype=float)  
print(matriz_2)
```





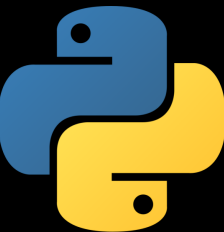
```
print(matrix_5)
matrix_5 = np.array([[1.1, (-3.2), 3.4, 0.6], [0.6, 1.1, (-0.6), 3.1], [1.3, 0.6, 5.5, 0]], dtype=float)
print(matrix_5)
```

```
[[ 1.1 -3.2  3.4  0.6]
 [ 0.6  1.1 -0.6  3.1]
 [ 1.3  0.6  5.5  0. ]]
```





Nota Inicial – Sempre que formos fazer referencia a um elemento ou uma quantidade de **linhaxColuna** a serem geradas, usamos uma tupla **(m,n)**





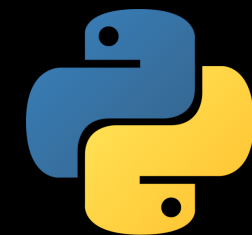
Parte 4: Matrizes Especiais – Matriz Nula – `np.zeros()`

0	0	0
0	0	0
0	0	0

```
import numpy as np
import math as mt

##### Matriz Nula#####

x = np.zeros((3,3))
print(x)
```





Parte 4: Matrizes Especiais – Matriz de UNS – `np.ones()`

1	1	1
1	1	1
1	1	1

```
y = np.ones((3, 3))  
print(y)
```





Parte 4: Matrizes Especiais – Matriz Identidade – `np.identity()`

1	0	0
0	1	0
0	0	1

```
z = np.identity(3)  
print(z)
```

```
[[1.  0.  0.]  
 [0.  1.  0.]  
 [0.  0.  1.]]
```





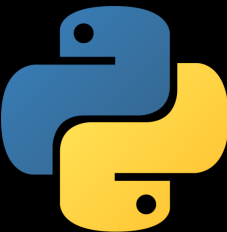
Parte 4: Matrizes Especiais – Matriz Identidade – `np.eye()` [MATLAB]

`numpy.eye`

```
numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C', *, Like=None)
```

Return a 2-D array with ones on the diagonal and zeros elsewhere.

[\[source\]](#)



numpy.identity

`numpy.identity(n, dtype=None, *, like=None)`

Return the identity array.

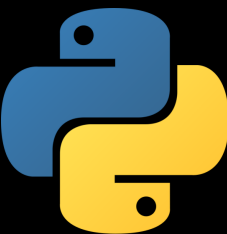
The identity array is a square array with ones on the main diagonal.

Parameters: *n* : *int*

Number of rows (and columns) in $n \times n$ output.

dtype : *data-type, optional*

Data-type of the output. Defaults to `float`.





```
##### Matriz Identidade#####
```

```
z = np.identity(3)
```

```
print(z)
```

```
##### Matriz de Zeros e Uns#####
```

```
sad = np.eye(3)
```

```
print(sad)
```

```
[[1.  0.  0.]
```

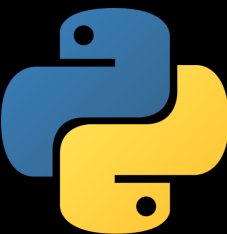
```
 [0.  1.  0.]
```

```
 [0.  0.  1.]]
```

```
[[1.  0.  0.]
```

```
 [0.  1.  0.]
```

```
 [0.  0.  1.]]
```





Parameters: N : *int*

Number of rows in the output.

M : *int, optional*

Number of columns in the output. If None, defaults to N .

k : *int, optional*

Index of the diagonal: 0 (the default) refers to the main diagonal, a positive value refers to an upper diagonal, and a negative value to a lower diagonal.

dtype : *data-type, optional*

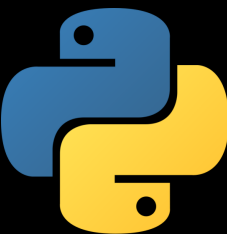
Data-type of the returned array.





```
#💡### Matriz de Zeros e Uns####  
sad = np.eye(3, 2)  
print(sad)
```

```
[[1.  0.]  
 [0.  1.]  
 [0.  0.]]
```





Tanto a função *numpy.eye()* e a função *numpy.identity()* geram matrizes com zeros e uns, mas a primeira gera uma matriz de qualquer dimensão com zeros e uns enquanto a segunda sempre vai gerar uma matriz identidade que tem que ser quadrada para se ter uma diagonal principal.



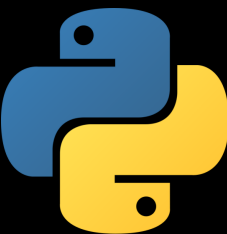


Parte 5: Operações com Matrizes – Parte 1 – Soma Matricial Vetor*Escalar

$$1^{\circ}) \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 4 & -1 & 1 \\ -4 & 0 & -6 \end{bmatrix} = \begin{bmatrix} 1+4 & 2-1 & 3+1 \\ 4-4 & 5+0 & 6-6 \end{bmatrix} =$$
$$= \begin{bmatrix} 5 & 1 & 4 \\ 0 & 5 & 0 \end{bmatrix}$$

$$2^{\circ}) \begin{bmatrix} 7 & 8 \\ 9 & 9 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 7+0 & 8+1 \\ 9+2 & 9+3 \end{bmatrix} = \begin{bmatrix} 7 & 9 \\ 11 & 12 \end{bmatrix}$$

$$3^{\circ}) \begin{bmatrix} 5 \\ 11 \\ \frac{3}{4} \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix} = \begin{bmatrix} 5+1 \\ 11-2 \\ \frac{3}{4}+3 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \\ \frac{15}{4} \end{bmatrix}$$





$$a = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \quad b = \begin{bmatrix} -1 & 2 \\ 0 & 1 \end{bmatrix}$$
$$c = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad d = 5$$

2.8 Scalar and Array Operations

What is the result of each of the following expressions?

- | | |
|-----------------|-----------------|
| (a) $a + b$ | (e) $a + c$ |
| (b) $a \cdot b$ | (f) $a + d$ |
| (c) $a * b$ | (g) $a \cdot d$ |
| (d) $a * c$ | (h) $a * d$ |

```
import numpy as np
import math as mth

a = np.array([[1, 0], [2, 1]])
b = np.array([[-1, 2], [0, 1]])
c = np.array([3], [2])
d = 5

#####Exercicio#####

# Soma das Matrizes

print(a+b)
print(b+a)
print(a+c)
print(c+a)
print(b+c)
print(c+b)
```





```
[[0 2]
 [2 2]]
[[0 2]
 [2 2]]
[[4 3]
 [4 3]]
[[4 3]
 [4 3]]
[[2 5]
 [2 3]]
[[2 5]
 [2 3]]
```





```
# Operações Constante Matriz
```

```
print(a*d)
```

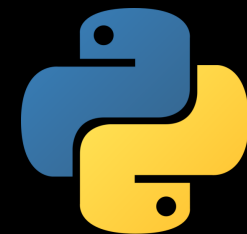
```
print(b*d)
```

```
print(c*d)
```

43. Exemplos

$$1^{\circ}) \quad 3 \cdot \begin{bmatrix} 1 & 7 & 2 \\ 5 & -1 & -2 \end{bmatrix} = \begin{bmatrix} 3 & 21 & 6 \\ 15 & -3 & -6 \end{bmatrix}$$

$$2^{\circ}) \quad \frac{1}{2} \cdot \begin{bmatrix} 0 & 2 & 4 \\ 8 & 6 & 4 \\ 10 & 12 & -6 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 4 & 3 & 2 \\ 5 & 6 & -3 \end{bmatrix}$$





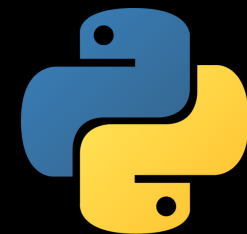
Parte 5: Operações com Matrizes – Parte 2 – Produto de Matrizes com Matrizes

OBSERVAÇÃO: o Numpy não faz produto de matrizes diretamente feito o MATLAB (Usando apenas `*`) se fizer isso será um produto de arranjo elemento a elemento o que está errado





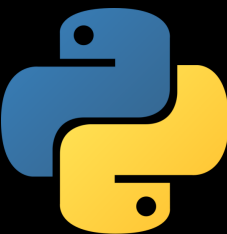
OBSERVAÇÃO 2: Matrizes e Arranjos (Álgebra Linear) seriam na prática mesma coisa, (Uma matriz seria um arranjo bidimensional e um tensor um tridimensional) mas o Numpy possui duas classes separadas (Até a versão 1.21) e uma específica para matriz. A documentação recomenda o uso dessa classe APENAS em caso de trabalho com álgebra linear, e essa classe pode sair biblioteca no futuro.



Matrix objects

Note

It is strongly advised *not* to use the matrix subclass. As described below, it makes writing functions that deal consistently with matrices and regular arrays very difficult. Currently, they are mainly used for interacting with `scipy.sparse`. We hope to provide an alternative for this use, however, and eventually remove the `matrix` subclass.





Método 1

Função `numpy.matmul()`

`numpy.matmul` ¶

```
numpy.matmul(x1, x2, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis]) = <ufunc 'matmul'>
```

Matrix product of two arrays.

Parameters: `x1, x2` : *array_like*

Input arrays, scalars not allowed.

`out` : *ndarray, optional*

A location into which the result is stored. If provided, it must have a shape that matches the signature $(n,k),(k,m) \rightarrow (n,m)$. If not provided or `None`, a freshly-allocated array is returned.

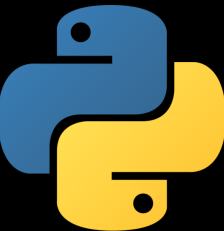




matriz1@matriz2

a @ b

matrix multiply





```
import numpy as np

matriz_A = np.array([[2, 3, 1], [(-1), 0, 2]])
matriz_B = np.array([[1, (-2)], [0, 5], [4, 1]])

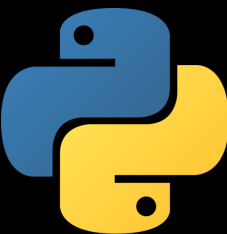
matriz_produto = np.matmul(matriz_A, matriz_B)
|

print(matriz_produto)

matriz_produto2 = matriz_A @ matriz_B
print(matriz_produto2)
```

```
[[ 6 12]
 [ 7  4]]

[[ 6 12]
 [ 7  4]]
```



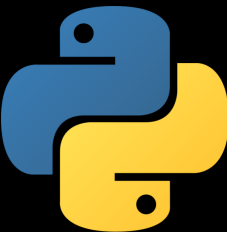


```
import numpy as np

matriz_A = np.array([[2, 3, 1], [(-1), 0, 2]])
matriz_B = np.array([[1, (-2)], [0, 5], [4, 1]])

matriz_produto = np.matmul(matriz_A, matriz_B)

print(matriz_produto)
```





$$\begin{pmatrix} 2 & 3 & 1 \\ -1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & -3 \\ 0 & 5 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 6 & 10 \\ 7 & 5 \end{pmatrix}$$

▼ Подробности (Умножение матриц)

Умножение матриц: строки первой матрицы умножаются на столбцы второй.

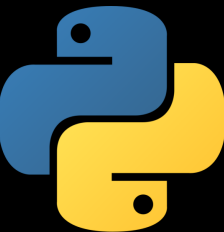
$$\begin{pmatrix} 2 & 3 & 1 \\ -1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & -3 \\ 0 & 5 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 + 3 \cdot 0 + 1 \cdot 4 & 2 \cdot (-3) + 3 \cdot 5 + 1 \cdot 1 \\ -1 \cdot 1 + 0 \cdot 0 + 2 \cdot 4 & -1 \cdot (-3) + 0 \cdot 5 + 2 \cdot 1 \end{pmatrix} = \begin{pmatrix} 6 & 10 \\ 7 & 5 \end{pmatrix}$$

https://matrixcalc.org/#%7B%7B2,3,1%7D,%7B-1,0,2%7D%7D*%7B%7B1,-3%7D,%7B0,5%7D,%7B4,1%7D%7D





Parte 6: Análise de Dimensões – Básico





Dimensão das Matrizes -1D/2D/3D `np.ndim()`

```
>>> primeira_matriz = np.array([[1, 2, 3], [4, 5, 6]])  
... segunda_matriz = np.array([[1, 2, 3], [4, 5, 8]])
```

```
>>> np.ndim(primeira_matriz)  
2
```





Número de Elementos da Matriz

`np.size()`

```
>>> np.size(primeira_matriz)
6
```

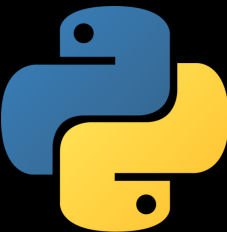




Numero de Linhas e Colunas

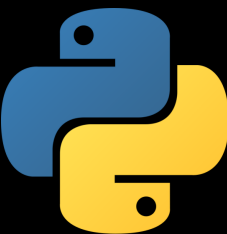
`np.shape()`

```
>>> np.shape(primeira_matriz)
(2, 3)
```





```
>>> b = np.arange(12).reshape(4, 3)      # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>>
>>> c = np.arange(24).reshape(2, 3, 4)   # 3d array
>>> print(c)
[[[ 0  1  2  3]
```

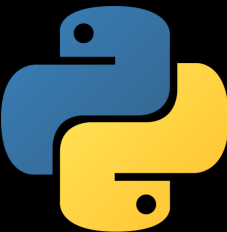




Criar Matrizes Espaçadas

`np.linspace()`

```
>>> np.linspace(1,10)
array([ 1.          ,  1.18367347,  1.36734694,  1.55102041,  1.73469388,
        1.91836735,  2.10204082,  2.28571429,  2.46938776,  2.65306122,
        2.83673469,  3.02040816,  3.20408163,  3.3877551 ,  3.57142857,
        3.75510204,  3.93877551,  4.12244898,  4.30612245,  4.48979592,
        4.67346939,  4.85714286,  5.04081633,  5.2244898 ,  5.40816327,
        5.59183673,  5.7755102 ,  5.95918367,  6.14285714,  6.32653061,
        6.51020408,  6.69387755,  6.87755102,  7.06122449,  7.24489796,
        7.42857143,  7.6122449 ,  7.79591837,  7.97959184,  8.16326531,
        8.34693878,  8.53061224,  8.71428571,  8.89795918,  9.08163265,
        9.26530612,  9.44897959,  9.63265306,  9.81632653, 10.          ])
```

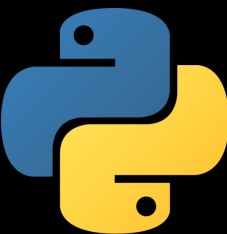




Parte 7: Análise Lógica/Comparativa – Parte 1

```
m logspace()  
logaddexp  
log1p  
logaddexp2  
logical_and  
logical_not  
logical_or  
logical_xor  
>>> arr  
Press Enter to insert, Guia to replace Next Tip  
>>> np.log
```

Operador	Significado
==	Igual a
!=	Diferente de
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a





LOGIC OPS:

`&` or `|` in NumPy is bitwise **AND/OR**, while in MATLAB `&` and `|` are logical **AND/OR**. The two can appear to work the same, but there are important differences. If you would have used MATLAB's `&` or `|` operators, you should use the NumPy ufuncs `logical_and`/`logical_or`.

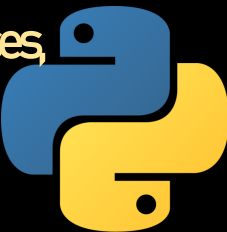
The notable differences between MATLAB's and NumPy's `&` and `|` operators are:

Non-logical {0,1} inputs: NumPy's output is the bitwise AND of the inputs. MATLAB treats any non-zero value as 1 and returns the logical AND.

For example `(3 & 4)` in NumPy is 0, while in MATLAB both 3 and 4 are considered logical true and `(3 & 4)` returns 1.

Precedence: NumPy's `&` operator is higher precedence than logical operators like `<` and `>`; MATLAB's is the reverse.

If you know you have boolean arguments, you can get away with using NumPy's bitwise operators, but be careful with parentheses, like this: `z = (x > 1) & (x < 2)`. The absence of NumPy operator forms 0

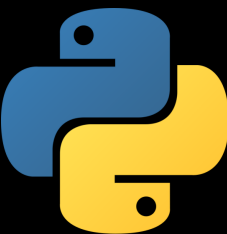




Igualdade de Matrizes

Com base na documentação a comparação pode ocorrer elemento a elemento ou com todas as matrizes

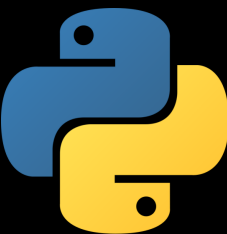
Como resultado pode ser **True** ou **False**





Caso 1 – Matrizes de Dimensões Diferentes

```
matriz_A = np.array([[2, 3, 1], [(-1), 0, 2]])  
matriz_B = np.array([[1, (-2)], [0, 5], [4, 1]])
```





```
False
```

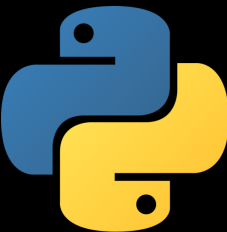
```
F:\pythonProject\Projetos_Python\Computação Científica\Matrizes\operacoes_matrizes_produto.py:20: DeprecationWarning: elementwise comparison failed;  
  x = matriz_A == matriz_B
```





Caso 2 – Matrizes Com Dimensões Iguais e Elementos Iguais

```
primeira_matriz = np.array([[1, 2, 3], [4, 5, 6]])  
segunda_matriz = np.array([[1, 2, 3], [4, 5, 6]])  
💡  
x = primeira_matriz == segunda_matriz  
print(x)
```





```
[[ True  True  True]
 [ True  True  True]]
```



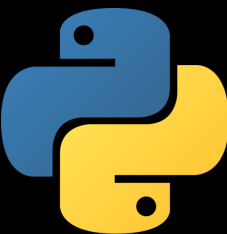


Caso 3 – Matrizes Com Dimensões Iguais e Elementos Diferentes

```
import numpy as np
import math as mt

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

x = primeira_matriz==segunda_matriz
print(x)
```





```
[[False False  True]
 [ True False  True]]
```





Caso 4 – Comparando toda a Matriz

Método 1 – Usando a Função `all()`





```
import numpy as np
import math as mt

"""
primeira_matriz = np.array([[1, 2, 3], [4, 5, 6]])
segunda_matriz = np.array([[2, 3, 3], [4, 4, 6]])

x = primeira_matriz==segunda_matriz
print(x)
"""

#Comparando a matriz usando a função all()

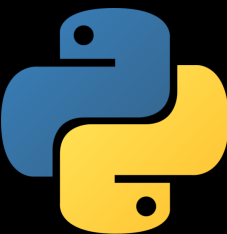
primeira_matriz = np.array([[1,2,3],[3,2,1]])
segunda_matriz = np.array([[32,45,99],[7,3,8]])
matriz_comparacao = primeira_matriz==segunda_matriz
resultado_comparacao = matriz_comparacao.all()
print(resultado_comparacao)
```

PC Run - pythonProject

Run: condicionais_aplicada_matriz

F:\pythonProject\venv\Scripts
False

Process finished with exit





```
matriz_c = np.array([[1, 2, 3], [1, 2, 3]])  
matriz_d = np.array([[1, 2, 3], [1, 2, 3]])  
  
mc2 = matriz_c == matriz_d  
  
print(mc2.all())
```



F:\pythonProject\venv\Script
True

Process finished with exit





pythonProject > Projetos_Python > Computação Científica > Matrizes > operacoes

operacoes_matrizes_produto.py × condicionais_aplicada_matrizes.py ×

```
8
9 # Exemplo
10
11 import numpy as np
12
13 matriz_A = np.array([[2, 3, 1], [(-1), 0, 2]])
14 matriz_B = np.array([[1, (-2)], [0, 5], [4, 1]])
15
16 comparacao = matriz_A == matriz_BS
17 print(comparacao)
18
19
20
21
22 matriz_c = np.array([[1, 2, 3], [1, 2, 3]])
23 matriz_d = np.array([[1, 2, 3], [1, 2, 3]])
24
25 mc2 = matriz_c == matriz_d
26
27 print(mc2.all())
```

```
1. (pythonProject\Projetos_Python\Computação Científica
    comparacao = matriz_A == matriz_B
False
True

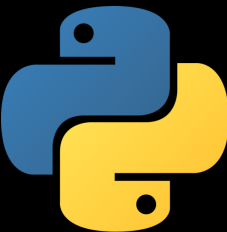
Process finished with exit code 0
```





A comparação de matrizes de dimensões diferentes sempre gerará uma mensagem de erro e as de dimensões iguais uma outra matriz comparando elemento a elemento

A função `all()` funciona para outras estruturas já estudadas (Exemplo: Lista) visto que são objetos iteráveis. Funciona com a lógica do Bitwise AND. Se todos os Elementos são True, ou False retorna um resultado. Se alguns são false ou true retorna false





Método 2 – Usando a Função `numpy.array_equal()`

```
numpy.array_equal(a1, a2, equal_nan=False)
```

[\[source\]](#)

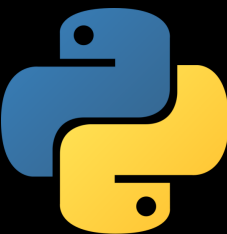
True if two arrays have the same shape and elements, False otherwise.

Parameters: `a1, a2 : array_like`

`equal_nan : bool`

Whether to compare NaN's as equal. If the dtype of `a1` and `a2` is complex, values will be considered equal if either the real or the imaginary component of a given value is `nan`.

New in version 1.19.0.





```
import numpy as np

matriz_A = np.array([[2, 3, 1], [(-1), 0, 2]])
matriz_B = np.array([[1, (-2)], [0, 5], [4, 1]])
matriz_c = np.array([[1, 2, 3], [1, 2, 3]])
matriz_d = np.array([[1, 2, 3], [1, 2, 3]])
primeira_matriz = np.array([[1, 2, 3], [3, 2, 1]])
segunda_matriz = np.array([[32, 45, 99], [7, 3, 8]])
primeira_matriz2 = np.array([[1, 2, 3], [4, 5, 6]])
segunda_matriz2 = np.array([[2, 3, 3], [4, 4, 6]])

#Comparação 1

print(np.array_equal(matriz_A, matriz_A))#Matriz igual A Ela Mesma
print(np.array_equal(matriz_A, matriz_B))
print(np.array_equal(matriz_A, matriz_c))
print(np.array_equal(matriz_c, matriz_A))
print(np.array_equal(matriz_c, matriz_d))
print(np.array_equal(primeira_matriz, primeira_matriz2))
```

True

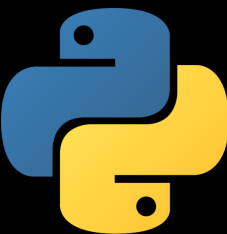
False

False

False

True

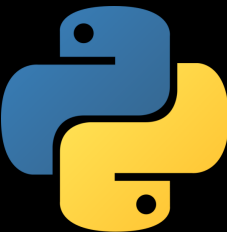
False





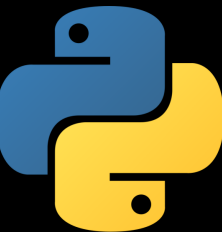
Usando a Função `numpy.array_equal()`

an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally NumPy provides types of its own. `numpy.int32`, `numpy.int16`, and `numpy.float64` are some examples.





Extras





1. Descobrimos Versão de Qualquer Pacote `__version__`

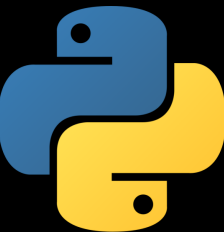
```
>>> import bzrlib  
>>> bzrlib.__version__  
'2.3.0'
```





1. Descobrimos Versão de Qualquer Pacote Externo que contenha versão `__version__`

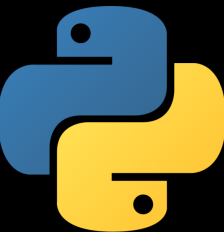
```
>>> import bzrlib  
>>> bzrlib.__version__  
'2.3.0'
```





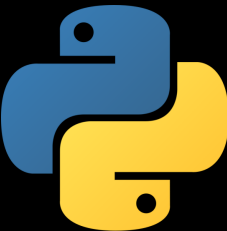
2. Descobrimos a configuração do Numpy instalado

```
np.show_config()
```





```
>>> np.show_config()
blas_mkl_info:
  NOT AVAILABLE
blis_info:
  NOT AVAILABLE
openblas_info:
  library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_info']
  libraries = ['openblas_info']
  language = f77
  define_macros = [('HAVE_CBLAS', None)]
blas_opt_info:
  library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_info']
  libraries = ['openblas_info']
  language = f77
  define_macros = [('HAVE_CBLAS', None)]
lapack_mkl_info:
  NOT AVAILABLE
openblas_lapack_info:
  library_dirs = ['D:\\a\\1\\s\\numpy\\build\\openblas_lapack_info']
  libraries = ['openblas_lapack_info']
  language = f77
  define_macros = [('HAVE_CBLAS', None)]
lapack_opt_info:
```

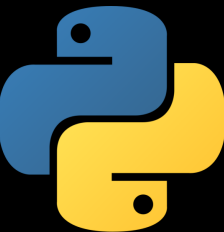




3. Criando Arranjos sequenciais

No Numpy temos uma função que usa o mesmo principio da função range do for:

```
np.arange()
```





Parte 7: Exercícios





1. Import the numpy package under the name `np` (☆☆☆)

```
Automatize_tarefas.py ^  questao_1.py ^  
# Lista de exercícios com Problemas de Python utilizando o Módulo Numpy  
  
# Import the numpy Package under the name np ===== Level Beginner  
  
# Para importar qualquer bloco de funções devemos usar o método import e o nome do pacote/bloco  
# Para Modificar o nome do pacote no algoritmo usamos o método as (como) e escrevemos o nome que queremos referenciar  
  
import numpy as np
```





2. Print the numpy version and the configuration (☆☆☆)

```
# Computação Científica

# Lista de exercícios com Problemas de Python utilizando o Módulo Numpy

# Lista 2 - Descubra a Versão do Módulo Numpy e A sua configuração

#para imprimir a versão do numpy use os metodos __version__

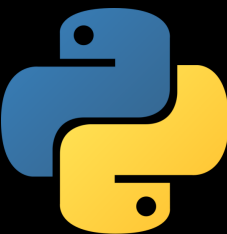
import numpy as np

versao = np.__version__

print(versao)

#para imprimir a configuração do numpy instalado usamos a função np.show.config()

np.show_config()
|
```





3. Create a null vector of size 10 (★☆☆)





```
# Computação Científica

# Lista de exercícios com Problemas de Python utilizando o Módulo Numpy

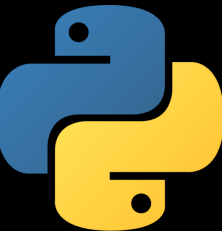
# Lista 3 - Imprima um vetor 10 nulo

# como sabemos, existe uma função que gera uma função nula que é a função zeros

import numpy as np

nula = np.zeros(10)
print(nula)

nula_2d = np.zeros((10, 10),
                    dtype=int) # Imprimir uma matrix 10x10 nula _----- Lembre-se que dimensões são representadas por tuplas
print(nula_2d)
```





Exercício 4 – Encontre o número de elementos da matriz e mostre as dimensões dela

```
# Computação Científica

# Lista de exercícios com Problemas de Python utilizando o Módulo Numpy

# Questão 4: Como descobrir o tamanho de uma matriz ou arranjo no numpy:

# O numpy possui uma função que retorna as dimensões mxn de um arranjo ou matriz

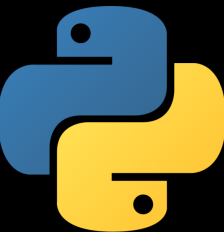
import numpy as np

matriz = np.eye(10, dtype=int)
print(matriz)

quantidade_elemento = np.size(matriz) # No Numpy retorna os número de elementos da matriz
print(quantidade_elemento)

|

dimensoes = np.shape(matriz) # Shape retorna o número de linhas e colunas da matriz
print(dimensoes)
```





6. Create a null vector of size 10 but the fifth value which is 1 (☆☆☆)

7. Create a vector with values ranging from 10 to 49 (☆☆☆)

github.com/rougier/numpy-100/blob/master/100_Numpy_exercises.md

2021 numpy-100/100_Numpy_exercises.md at master · rougier/numpy-100

8. Reverse a vector (first element becomes last) (☆☆☆)

9. Create a 3x3 matrix with values ranging from 0 to 8 (☆☆☆)

10. Find indices of non-zero elements from [1,2,0,0,4,0] (☆☆☆)



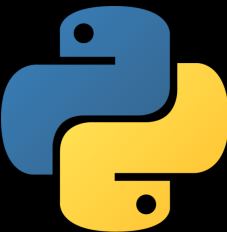


```
PythonProject > Computação Científica > Matrizes > Resolução_100_Problemas > questao_5.py
Open... Matize_Tarefas.py x questao_3.py x questao_4.py x questao_5.py x
1 # Computação Científica Usando o Módulo Numpy
2
3
4 # Questão 5: Crie um vetor nulo de tamanho 10 mas com o quinto elemento seja 1
5
6 # Primeiro Passo: Importação do Módulo Numpy
7
8 import numpy as np
9
10 # Segundo Passo: criar um vetor nulo com 10 elementos
11
12 nulo = np.zeros((10, 1), dtype=int)
13 print(nulo)
14 # Segundo passo: substituir o o quinto elemento do vetor por 1
15
16 nulo[4] = 1
17 print(nulo)
18
```

Run: questao_5 x

F:\pythonProject\ve

```
[[0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]]
[[0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]]
```

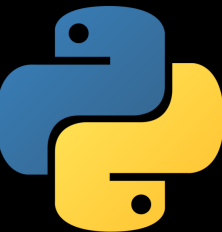




7. Create a vector with values ranging from 10 to 49 (☆☆☆)

```
sequencia = np.arange(10, 50)  
print(sequencia)
```

```
[0]  
[0]  
[0]  
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33  
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
```

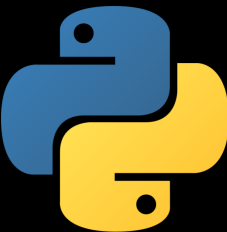




8. Reverse a vector (first element becomes last) (☆☆☆)

```
sequencia = np.arange(10, 50).reshape((40, 1))  
print(sequencia)  
sequencia_inversa = sequencia[::-1]  
print(sequencia_inversa)
```

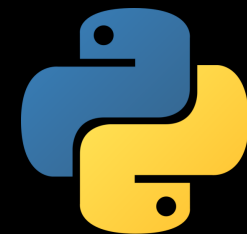
```
[0]  
[0]  
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33  
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]  
[[10]
```





9. Create a 3x3 matrix with values ranging from 0 to 8 (☆☆☆)

```
# Questão 9: Crie uma matriz que vá de 0 a 8 sendo 3x3  
# podemos usar a função arange juntamente com a função reshape (redimensiona o arranjo)  
  
questao7 = np.arange(9).reshape(3, 3)  
print(questao7)
```





10. Find indices of non-zero elements from [1,2,0,0,4,0] (☆☆☆)

```
# Jeito manual

import numpy as np

vetor_linha = np.array([1, 2, 0, 0, 4, 0])

valor1 = vetor_linha[0]
valor2 = vetor_linha[1]
valor3 = vetor_linha[4]
print(valor1, valor2, valor3)

# Jeito automatizado

i = 0

tamanho = np.size(vetor_linha)
```





```
# Jeito automatizado

i = 0

tamanho = np.size(vetor_linha)

for i in range(tamanho):

    if vetor_linha[i] == 0:
        i += 1
        print("Valor Nulo")

    elif vetor_linha[i] > 0:
        print("Indice: ", i, "Valor", vetor_linha[i])
        i += 1

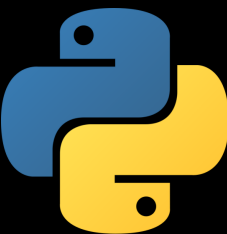
    else:
        print("Indice: ", i, "Valor", vetor_linha[i])
        i += 1

if i > tamanho:
    print("Fim do Arranjo")
```

Run: questao_10 x

F:\pythonProject\venv\Script
1 2 4
Indice: 0 Valor 1
Indice: 1 Valor 2
Valor Nulo
Valor Nulo
Indice: 4 Valor 4
Valor Nulo

Process finished with exit code 0





```
i = 0

tamanho = np.size(vetor_linha)

while i < (tamanho-1):

    if vetor_linha[i] == 0:
        i += 1
        print("Valor Nulo")

    elif vetor_linha[i] > 0:
        print("Indice: ", i, "Valor", vetor_linha[i])
        i += 1

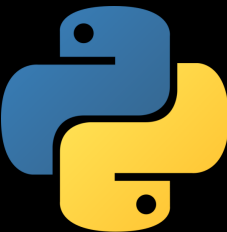
    else:
        print("Indice: ", i, "Valor", vetor_linha[i])
        i += 1

if i == (tamanho-1):
    print("Fim do Arranjo")
    break
```

Output:

```
F:\pythonProject\venv\Scripts\python.e
1 2 4
Indice:  0 Valor 1
Indice:  1 Valor 2
Valor Nulo
Valor Nulo
Indice:  4 Valor 4
Valor Nulo
Indice:  0 Valor 1
Indice:  1 Valor 2
Valor Nulo
Valor Nulo
Indice:  4 Valor 4
Fim do Arranjo

Process finished with exit code 0
```



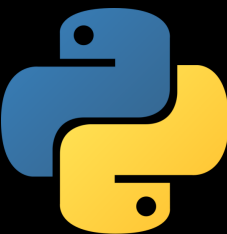
11. Create a 3x3 identity matrix (★☆☆)

```
# Questão 11 - Crie uma matriz identidade 3x3
```

```
identidade = np.identity(3, dtype=int)
print(identidade)
```

```
[[1 0 0]
 [0 1 0]
 [0 0 1]]
```

```
Process finished
```



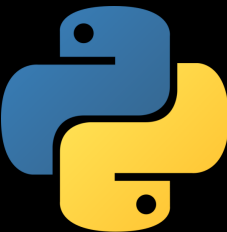


Exercício – Dada uma matriz, some todos os elementos dela

```
import numpy as np

matriz1 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]], dtype=int)
print(matriz1)

soma_metodo1 = np.sum(matriz1)
print(soma_metodo1)
```





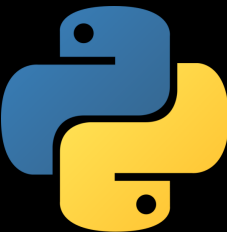
Parte 8: Elementos de Matrizes





13. Create a 10x10 array with random values and find the minimum and maximum values (☆☆☆)

Existem varias formas de analisar um valor máximo e mínimo
Em um arranjo, ou eixo(**coluna**)





numpy.amin

```
numpy.amin(a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)
```

[\[source\]](#)

Return the minimum of an array or minimum along an axis.

Parameters: *a* : *array_like*

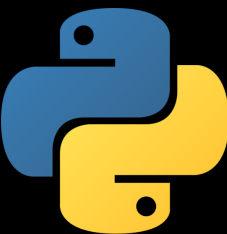
Input data.

axis : *None or int or tuple of ints, optional*

Axis or axes along which to operate. By default, flattened input is used.

New in version 1.7.0.

If this is a tuple of ints, the minimum is selected over multiple axes, instead of a single axis or all the axes as before.





numpy.amax

```
numpy.amax(a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)
```

[source]

Return the maximum of an array or maximum along an axis.

Parameters: *a* : *array_like*

Input data.

axis : *None or int or tuple of ints, optional*

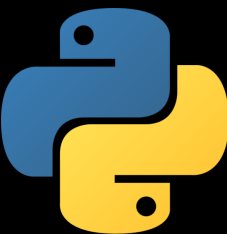
Axis or axes along which to operate. By default, flattened input is used.

New in version 1.7.0.

If this is a tuple of ints, the maximum is selected over multiple axes, instead of a single axis or all the axes as before.

out : *ndarray, optional*

Alternative output array in which to place the result. Must be of the same shape and buffer length as the expected output. See [Output type determination](#) for more details.





```
# Desafio 100 Questões de Numpy

# Problema 13 - Crie uma matriz 10x10 com valores aleatórios e retorne o maior valor deles e o menor
from random import random

import numpy as np

matriz = np.array([[1, 3, 683, 554], [754, 0, 694, 54]])
print(matriz)

maximo = np.amax(matriz)
minimo = np.amin(matriz)

print(maximo, minimo)
```





18. Create a 5x5 matrix with values 1,2,3,4 just below the diagonal (☆☆☆)

```
>>> x = np.identity((5))*range(1,6)
>>> print(x)
[[1.  0.  0.  0.  0.]
 [0.  2.  0.  0.  0.]
 [0.  0.  3.  0.  0.]
 [0.  0.  0.  4.  0.]
 [0.  0.  0.  0.  5.]]
```

