

# 4 - Prática: Principais Bibliotecas e Ferramentas Python para Aprendizado de Máquina (I)

## ▼ Instancia virtual e Jupyter notebook

Havendo instalação do Anaconda na maquina, será necessário somente indicar o comando:

Jupyter notebook

Via CMD da maquina. Isto inicializará o anacondas com caminho raiz do Windows.

A partir disso, basta acessar a pasta que tem necessidade de programar.

Ao criar um arquivo, é possível realizar processo de restart do kernel, do qual caso esteja travado o kernel, o mesmo irá perder variáveis e zerar.

Através do tipo de escrita Markdown é possível realizar escrita na página, diferente do campo Code que inicializa uma linha para programar

Python e Anacondas permitem instalações personalizadas do python, das quais podem ter versões diferentes. Isto é realizado através de ambientes virtuais.

O **VIRTUALENV** que é utilizado para isto.

Para instalação da versão alternativa, o vídeo indica usar CMD, no meu caso utilizei o Anaconda Prompt

**conda create —name “Nome\_do\_perfil\_sem\_aspas”**

```
(python2) C:\Users\leofe>conda create --name versao_teste
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 23.7.4
  latest version: 23.11.0

Please update conda by running

    $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

    conda install conda=23.11.0

## Package Plan ##

  environment location: C:\Users\leofe\.conda\envs\versao_teste

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate versao_teste
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Deste modo ele estará criando um “perfil” referente a esta instalação.

Após isto, será necessário realizar a entrada deste perfil, através do comando activate:

**activate “Nome\_do\_perfil\_sem\_aspas”**

```
(python2) C:\Users\leofe>activate versao_teste
(python2) C:\Users\leofe>conda.bat activate versao_teste
```

Feito isto, realizar instalação da versão do python que tem necessidade:

**conda install python=3.6**

```

(versao_teste) C:\Users\leofe>conda install python=3.6
Collecting package metadata (current_repodata.json): done
Solving environment: unsuccessful initial attempt using frozen solve. Retrying with flexible solve.
Solving environment: unsuccessful attempt using repodata from current_repodata.json, retrying with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 23.7.4
  latest version: 23.11.0

Please update conda by running

  $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

  conda install conda=23.11.0

## Package Plan ##

environment location: C:\Users\leofe\.conda\envs\versao_teste

added / updated specs:
- python=3.6

The following packages will be downloaded:



| package           | build          |         |
|-------------------|----------------|---------|
| certifi-2021.5.30 | py36haa95532_0 | 140 KB  |
| pip-21.2.2        | py36haa95532_0 | 1.9 MB  |
| python-3.6.13     | h3758d61_0     | 14.6 MB |
| setuptools-58.0.4 | py36haa95532_0 | 776 KB  |
| wheel-0.37.1      | pyhd3eb1b0_0   | 33 KB   |
| wincertstore-0.2  | py36h7fe50ca_0 | 14 KB   |
| Total:            |                | 17.4 MB |



The following NEW packages will be INSTALLED:

certifi pkgs/main/win-64::certifi-2021.5.30-py36haa95532_0
pip pkgs/main/win-64::pip-21.2.2-py36haa95532_0
python pkgs/main/win-64::python-3.6.13-h3758d61_0
setuptools pkgs/main/win-64::setuptools-58.0.4-py36haa95532_0
sqlite pkgs/main/win-64::sqlite-3.41.2-h2bbff1b_0
vc pkgs/main/win-64::vc-14.2-h21ff451_1
vs2015_runtime pkgs/main/win-64::vs2015_runtime-14.27.29016-h5e58377_2
wheel pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
wincertstore pkgs/main/win-64::wincertstore-0.2-py36h7fe50ca_0

Proceed ([y]/n)? y

Downloading and Extracting Packages

Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(versao_teste) C:\Users\leofe>python -version
Unknown option: -e
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Try 'python -h' for more information.

(versao_teste) C:\Users\leofe>python --version
Python 3.6.13 :: Anaconda, Inc.

(versao_teste) C:\Users\leofe>

```

Para confirmar instalação é possível indicar o comando:

`python --version`

É possível realizar isto através do comando:

**conda create -n "Nome\_perfil" python='Versão\_do\_Python'**  
**scipy='Versão\_scipy' "Bibliotecas" "bibliotecas" "Bibliotecas"**

Para validar os perfis instalados, basta utilizar o seguinte comando:

**Conda info —envs**

#### ▼ Numpy

Biblioteca de álgebra linear;

Bloco de construção de todas as outras bibliotecas de análise de dados.

É rápida pois métodos foram compilados em C.

#### ▼ Criação de matrizes

##### ▼ Array

Basta definir do seguinte modo:

```
# Variavel = ([a, b, c], [d, e, f], [g, h, i])  
  
lista = ([1, 2, 3], [4, 5, 6], [7, 8, 9])
```

Feito isto, haverá dois modos de chamar esta lista em questão, através do jeito convencional, apenas printando esse valor, ou chamando como um array(função do Numpy)

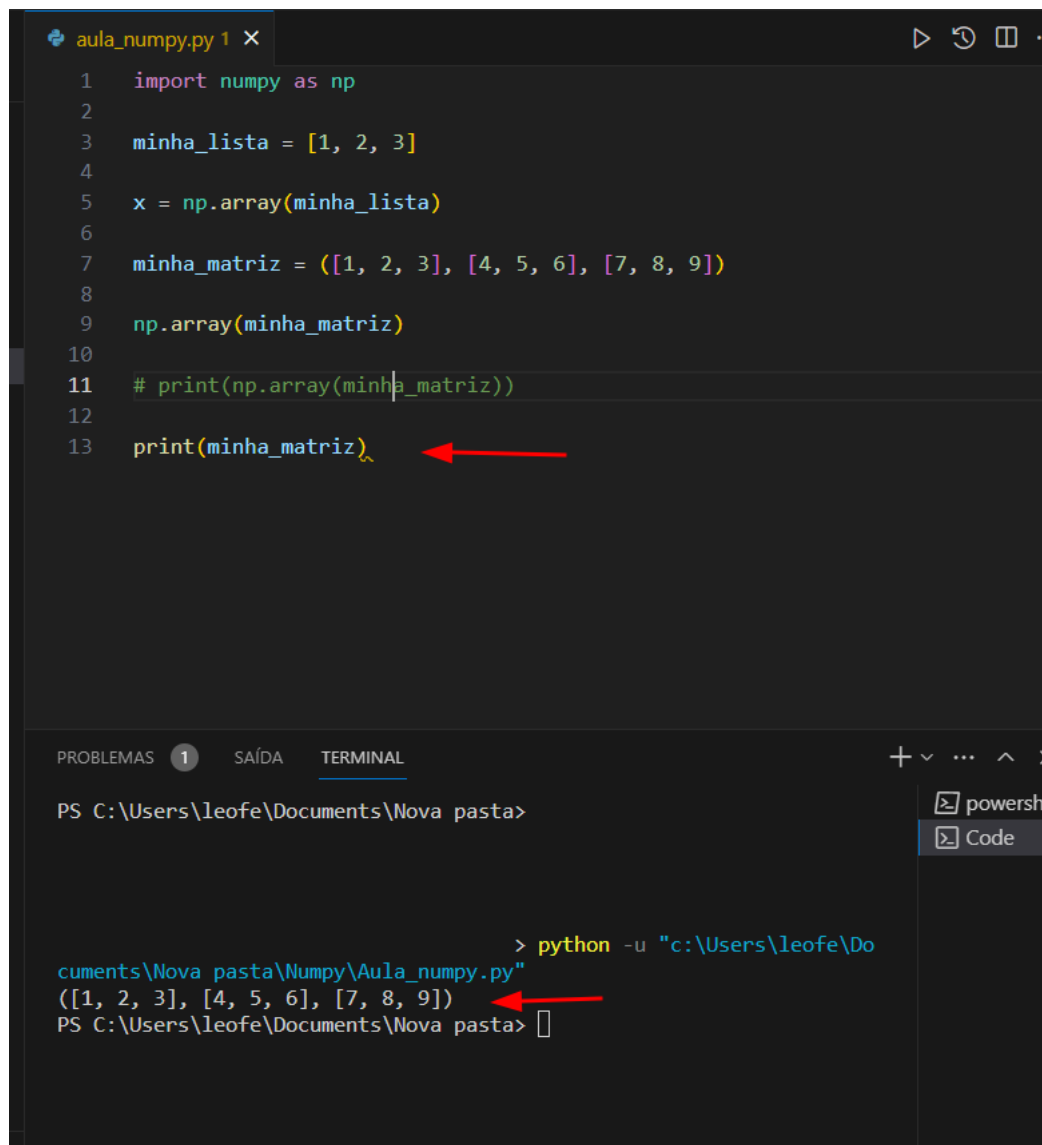
Chamando pela função array, essa lista será considerada uma matriz:

```
aula_numpy.py 1 X
1 import numpy as np
2
3 minha_lista = [1, 2, 3]
4
5 x = np.array(minha_lista)
6
7 minha_matriz = ([1, 2, 3], [4, 5, 6], [7, 8, 9])
8
9 np.array(minha_matriz)
10
11 print(np.array(minha_matriz))
12
13 # print(minha_matriz)
```

PROBLEMAS 1 SAÍDA TERMINAL

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[[1 2 3]
 [4 5 6]
 [7 8 9]]
PS C:\Users\leofe\Documents\Nova pasta>
```

Enquanto se utilizar o print sem função retorna como uma lista:



```
aula_numpy.py 1 X
1 import numpy as np
2
3 minha_lista = [1, 2, 3]
4
5 x = np.array(minha_lista)
6
7 minha_matriz = ([1, 2, 3], [4, 5, 6], [7, 8, 9])
8
9 np.array(minha_matriz)
10
11 # print(np.array(minha_matriz))
12
13 print(minha_matriz)
```

PROBLEMAS 1 SAÍDA TERMINAL

PS C:\Users\leofe\Documents\Nova pasta>

> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula\_numpy.py"

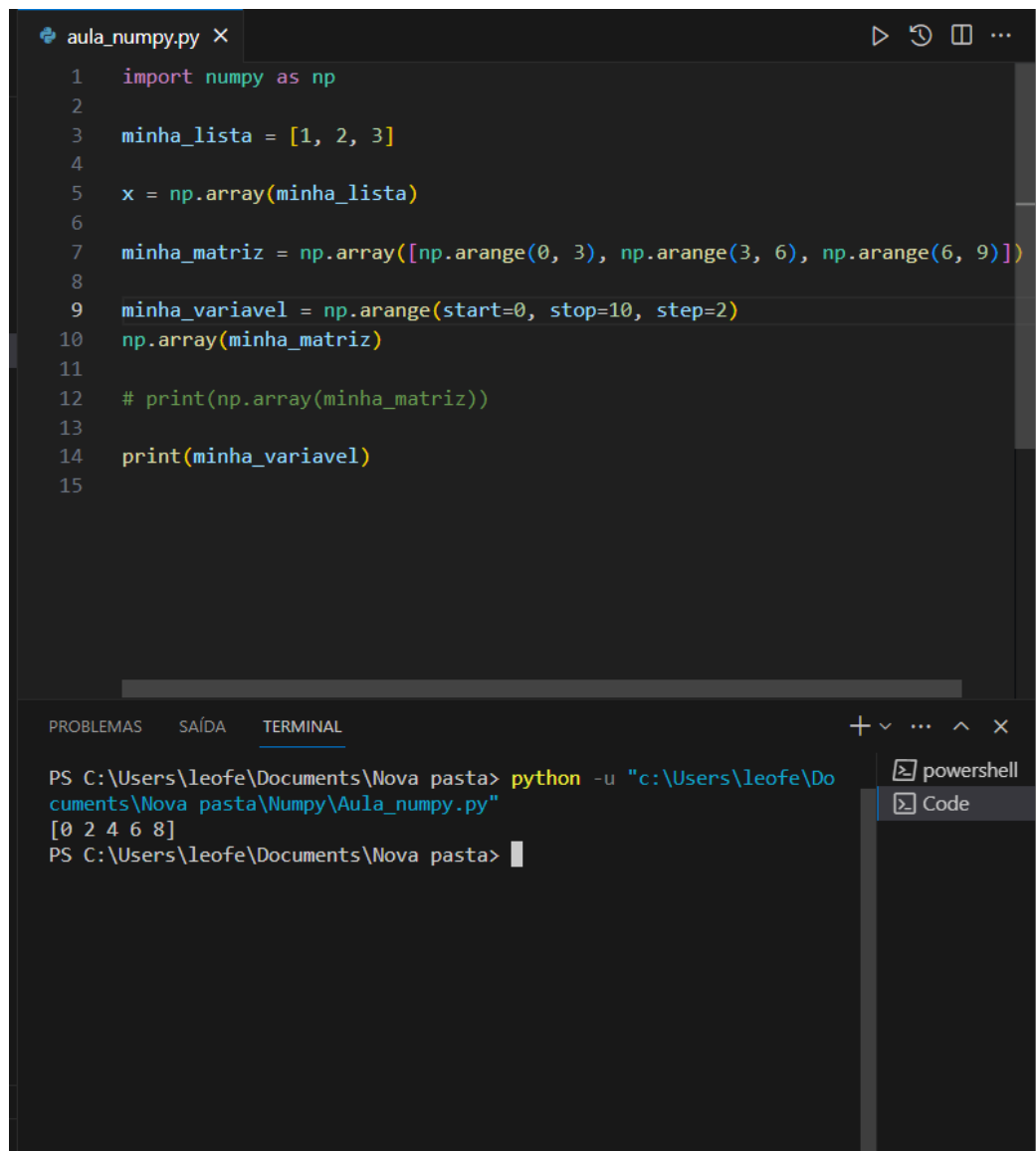
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

PS C:\Users\leofe\Documents\Nova pasta>

### ▼ Arange

Através do `arange`, é possível definir valores para um array, sem necessidade de declarar eles anteriormente, apenas seguindo um padrão definido na função:

```
minha_variavel = np.arange(start = 0, stop = 10, ste
```



The image shows a Visual Studio Code editor window with a file named `aula_numpy.py`. The code in the editor is as follows:

```
1 import numpy as np
2
3 minha_lista = [1, 2, 3]
4
5 x = np.array(minha_lista)
6
7 minha_matriz = np.array([np.arange(0, 3), np.arange(3, 6), np.arange(6, 9)])
8
9 minha_variavel = np.arange(start=0, stop=10, step=2)
10 np.array(minha_matriz)
11
12 # print(np.array(minha_matriz))
13
14 print(minha_variavel)
15
```

Below the editor is a terminal window with the following output:

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Aula_numpy.py"
[0 2 4 6 8]
PS C:\Users\leofe\Documents\Nova pasta>
```

On the right side of the terminal, there are buttons for `powershell` and `Code`.

Sendo possível também gerar uma matriz com estes valores

```
minha_matriz = np.array([
    np.arange(0, 3), np.arange(3, 6), np.arange(6, 9
])
```

```
aula_numpy.py X
1  import numpy as np
2
3  minha_lista = [1, 2, 3]
4
5  x = np.array(minha_lista)
6
7  minha_matriz = np.array([
8      np.arange(0, 3), np.arange(3, 6), np.arange(6, 9)
9  ])
10
11  np.array(minha_matriz)
12
13  # print(np.array(minha_matriz))
14
15  print(minha_matriz)
16
```

PROBLEMAS SAÍDA TERMINAL

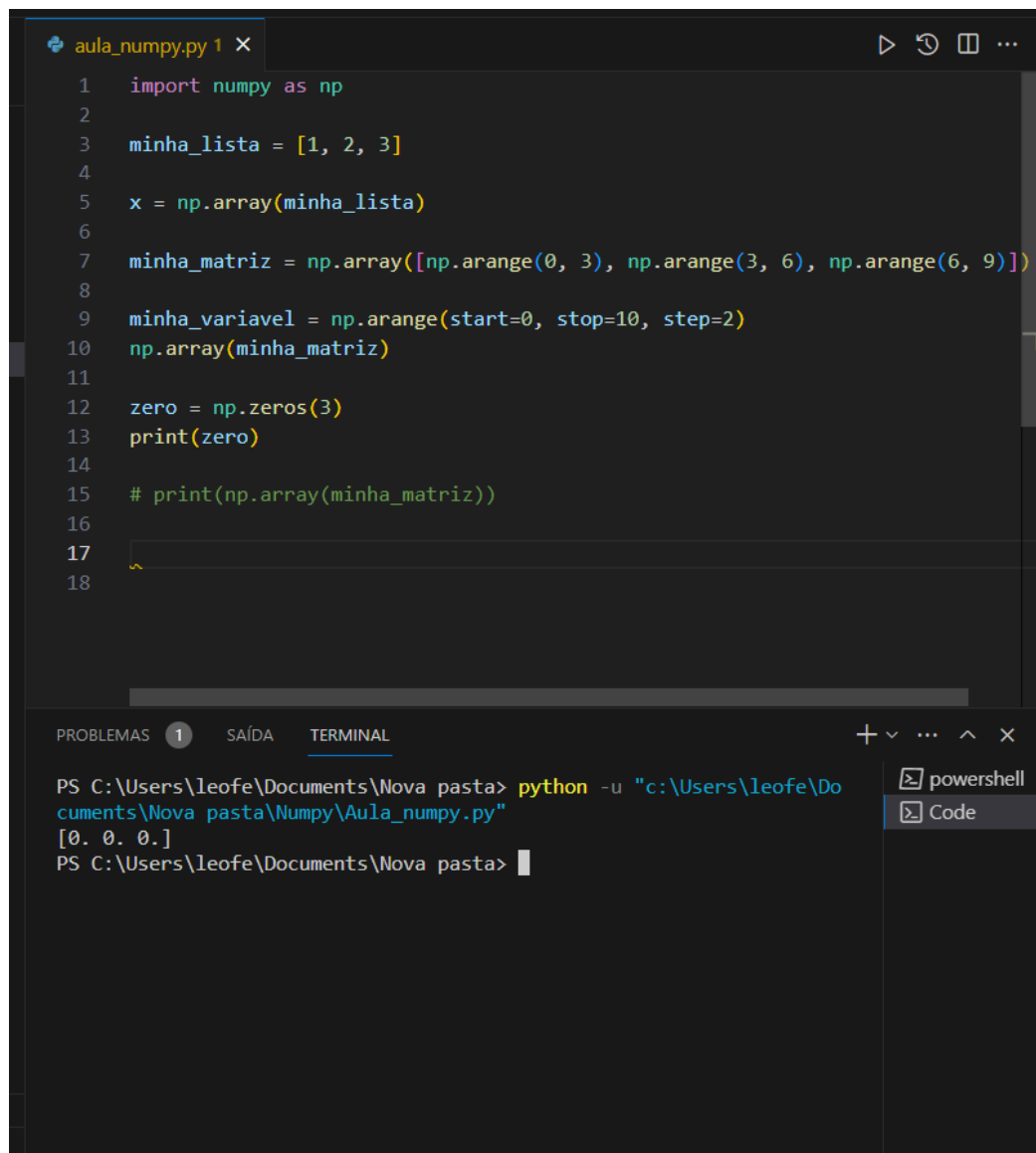
```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[[0 1 2]
 [3 4 5]
 [6 7 8]]
PS C:\Users\leofe\Documents\Nova pasta>
```

### ▼ Zeros

Criar uma matriz com 3 valores zerados:

```
zero = np.zeros(3)
print(zero)
```





```
1 import numpy as np
2
3 minha_lista = [1, 2, 3]
4
5 x = np.array(minha_lista)
6
7 minha_matriz = np.array([np.arange(0, 3), np.arange(3, 6), np.arange(6, 9)])
8
9 minha_variavel = np.arange(start=0, stop=10, step=2)
10 np.array(minha_matriz)
11
12 zero = np.zeros(3)
13 print(zero)
14
15 # print(np.array(minha_matriz))
16
17
18
```

PROBLEMAS 1 SAÍDA TERMINAL

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[0. 0. 0.]
PS C:\Users\leofe\Documents\Nova pasta>
```

**Obs:** Para criação de qualquer matriz, você pode também estar acrescentando um parênteses a mais na chamada de algumas funções específicas para criação de uma matriz, porém a mesma precisa ser uma tupla:

```
minha_variavel = np.zeros((0,10,2))
```

#### ▼ Ones

Criar uma matriz somente com números 1:

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

```
aula_numpy.py 2 X
1 import numpy as np
2
3 # minha_lista = [1, 2, 3]
4
5 # x = np.array(minha_lista)
6
7 # minha_matriz = np.array([np.arange(0, 3), np.arange(3, 6), np.arange(6, 9)])
8
9 # minha_variavel = tuple(np.arange((5, 5)))
10 # np.array(minha_matriz)
11
12 # zero = np.zeros(3)
13 # print(minha_variavel)
14
15 # print(np.array(minha_matriz))
16
17
18 um = np.ones((3,3))
19 print(um)
```

PROBLEMAS 2 SAÍDA TERMINAL

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
PS C:\Users\leofe\Documents\Nova pasta>
```

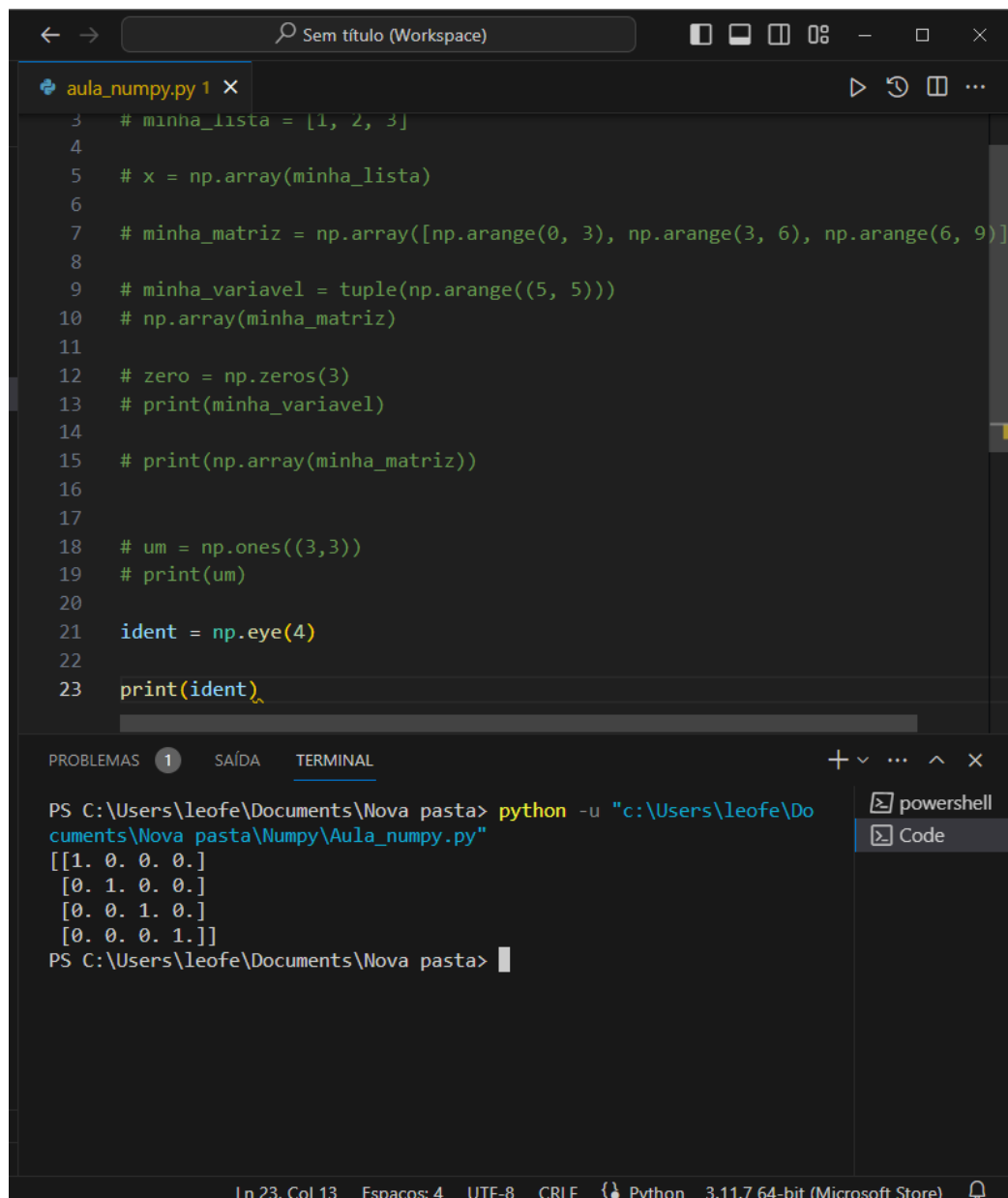
powerShell  
Code

#### ▼ Eye

Cria uma matriz identidade:

```
ident = np.eye(4)

print(ident)
```



The image shows a Visual Studio Code window with a Python file named `aula_numpy.py` open. The code defines several NumPy arrays and prints them. Below the code editor, the `TERMINAL` tab is active, showing the command `python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"` and its output, which displays a 4x4 identity matrix.

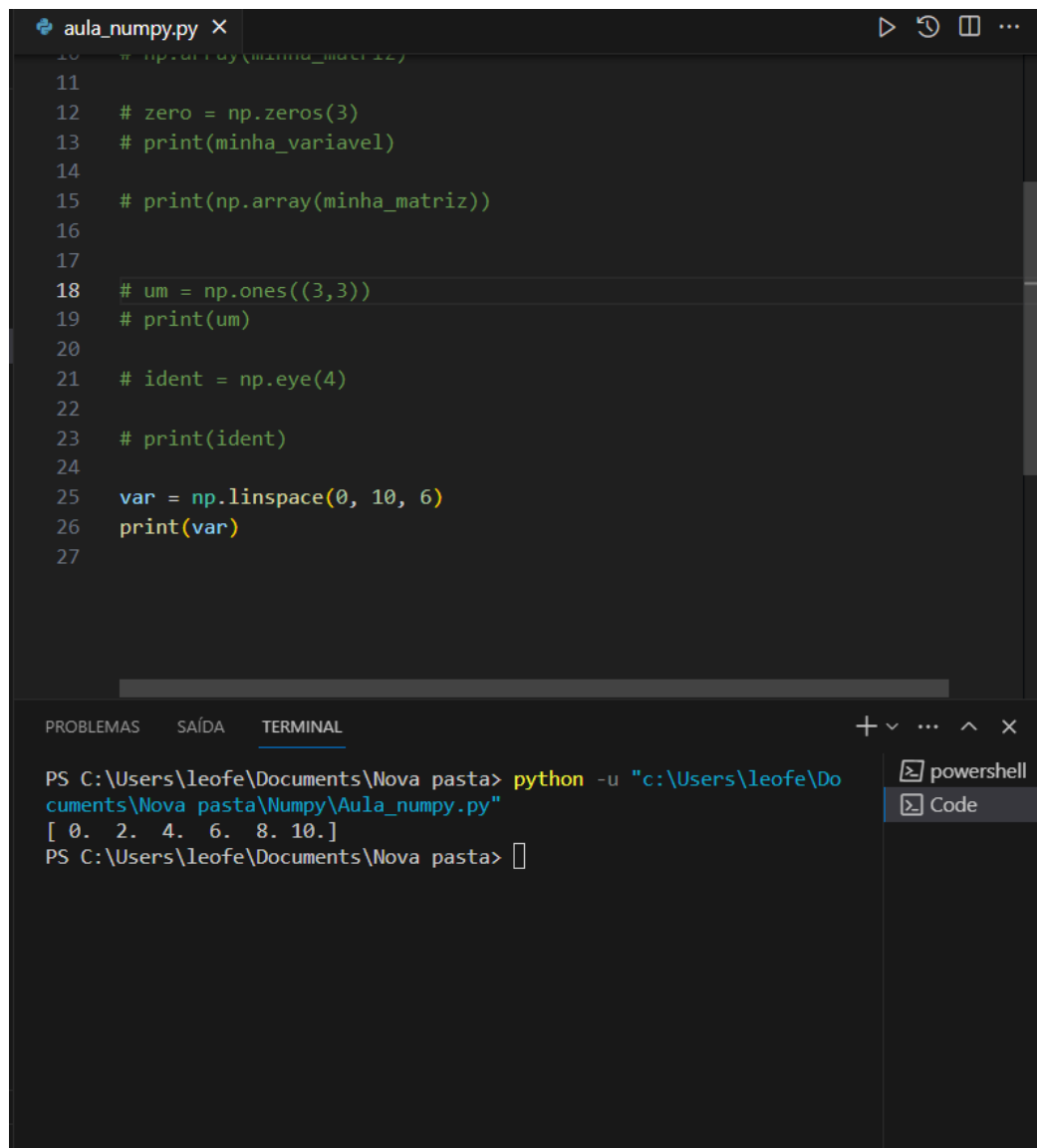
```
3 # minha_lista = [1, 2, 3]
4
5 # x = np.array(minha_lista)
6
7 # minha_matriz = np.array([np.arange(0, 3), np.arange(3, 6), np.arange(6, 9)])
8
9 # minha_variavel = tuple(np.arange((5, 5)))
10 # np.array(minha_matriz)
11
12 # zero = np.zeros(3)
13 # print(minha_variavel)
14
15 # print(np.array(minha_matriz))
16
17
18 # um = np.ones((3,3))
19 # print(um)
20
21 ident = np.eye(4)
22
23 print(ident)
```

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
PS C:\Users\leofe\Documents\Nova pasta>
```

### ▼ Linspace

Através dele você pode especificar quantos valores você quer entre o ponto inicial e final:

```
var = np.linspace(0, 10, 6)
print(var)
```



The image shows a Visual Studio Code editor window with a file named `aula_numpy.py`. The code in the editor is as follows:

```
10 # np.array(minha_matriz)
11
12 # zero = np.zeros(3)
13 # print(minha_variavel)
14
15 # print(np.array(minha_matriz))
16
17
18 # um = np.ones((3,3))
19 # print(um)
20
21 # ident = np.eye(4)
22
23 # print(ident)
24
25 var = np.linspace(0, 10, 6)
26 print(var)
27
```

Below the editor is a terminal window. The terminal shows the command to run the script and its output:

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[ 0.  2.  4.  6.  8. 10.]
PS C:\Users\leofe\Documents\Nova pasta>
```

The terminal window also shows tabs for `powerShell` and `Code`.

```
var = np.linspace(0,10,100)
print(var)
```

```
aula_numpy.py X
10 # print(np.zeros(minha_variavel))
11
12 # zero = np.zeros(3)
13 # print(minha_variavel)
14
15 # print(np.array(minha_matriz))
16
17
18 # um = np.ones((3,3))
19 # print(um)
20
21 # ident = np.eye(4)
22
23 # print(ident)
24
25 var = np.linspace(0, 10, 100)
26 print(var)
27

PROBLEMAS SAÍDA TERMINAL
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[ 0. 0.1010101 0.2020202 0.3030303 0.4040404 0.50505051
0.60606061 0.70707071 0.80808081 0.90909091 1.01010101 1.11111111
1.21212121 1.31313131 1.41414141 1.51515152 1.61616162 1.71717172
1.81818182 1.91919192 2.02020202 2.12121212 2.22222222 2.32323232
2.42424242 2.52525253 2.62626263 2.72727273 2.82828283 2.92929293
3.03030303 3.13131313 3.23232323 3.33333333 3.43434343 3.53535354
3.63636364 3.73737374 3.83838384 3.93939394 4.04040404 4.14141414
4.24242424 4.34343434 4.44444444 4.54545455 4.64646465 4.74747475
4.84848485 4.94949495 5.05050505 5.15151515 5.25252525 5.35353535
5.45454545 5.55555556 5.65656566 5.75757576 5.85858586 5.95959596
6.06060606 6.16161616 6.26262626 6.36363636 6.46464646 6.56565657
6.66666667 6.76767677 6.86868687 6.96969697 7.07070707 7.17171717
7.27272727 7.37373737 7.47474747 7.57575758 7.67676768 7.77777778
7.87878788 7.97979798 8.08080808 8.18181818 8.28282828 8.38383838
8.48484848 8.58585859 8.68686869 8.78787879 8.88888889 8.98989899
9.09090909 9.19191919 9.29292929 9.39393939 9.49494949 9.59595959
9.6969697 9.7979798 9.8989899 10. ]
PS C:\Users\leofe\Documents\Nova pasta> █
```

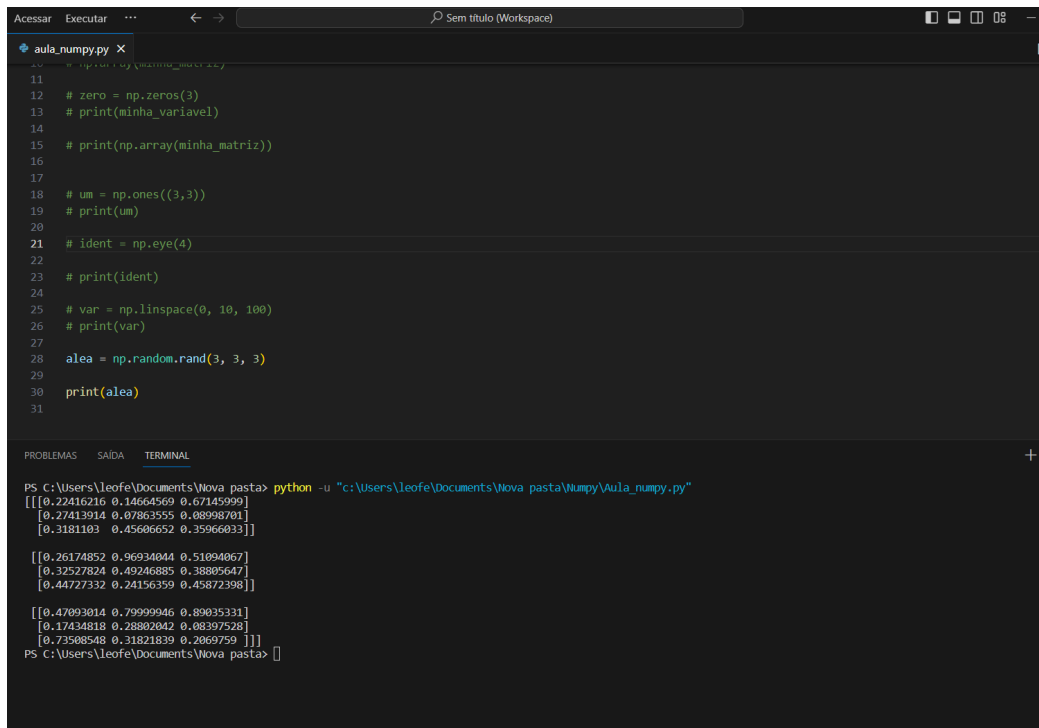
## ▼ Random

Retorna números “Aleatórios” em um intervalo

### Modelo uniforme

```
# teste = np.random.rand(quantidade_de_blocos_de_mat
# Colunas_da_matriz)

alea = np.random.rand(3,3,3)
print(alea)
```



```
10 # np.random.randn(3,3)
11
12 # zero = np.zeros(3)
13 # print(minha_variavel)
14
15 # print(np.array(minha_matriz))
16
17
18 # um = np.ones((3,3))
19 # print(um)
20
21 # ident = np.eye(4)
22
23 # print(ident)
24
25 # var = np.linspace(0, 10, 100)
26 # print(var)
27
28 alea = np.random.rand(3, 3, 3)
29
30 print(alea)
31
```

PROBLEMAS SAÍDA TERMINAL

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[[[0.22416216 0.14664569 0.67145999]
 [0.27413914 0.07863555 0.08998701]
 [0.3181103 0.45606652 0.35966033]]

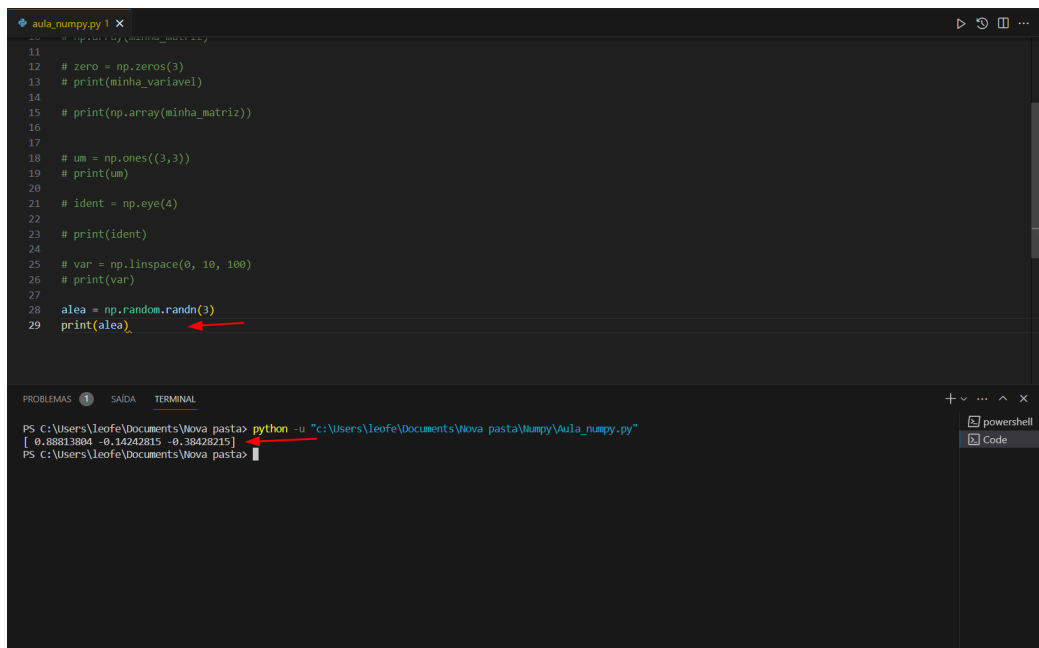
 [[0.26174852 0.96934044 0.51094067]
 [0.32527824 0.49246885 0.38805647]
 [0.44727332 0.24156359 0.45872398]]

 [[0.47093014 0.79999946 0.89035331]
 [0.17434818 0.28802042 0.08397528]
 [0.73508548 0.31821839 0.2069759 ]]]]
PS C:\Users\leofe\Documents\Nova pasta>
```

## ▼ Randn

Gera números aleatórios de uma distribuição normal padrão com média zero e desvio padrão um ( $N(0,1)$ ):

```
alea = np.random.randn(3)
print(alea)
```



```
11
12 # zero = np.zeros(3)
13 # print(minha_variavel)
14
15 # print(np.array(minha_matriz))
16
17
18 # um = np.ones((3,3))
19 # print(um)
20
21 # ident = np.eye(4)
22
23 # print(ident)
24
25 # var = np.linspace(0, 10, 100)
26 # print(var)
27
28 alea = np.random.randn(3)
29 print(alea)
```

PROBLEMAS 1 SAÍDA TERMINAL

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_numpy.py"
[ 0.88813804 -0.14242815 -0.38428215]
PS C:\Users\leofe\Documents\Nova pasta>
```

### ▼ Randint

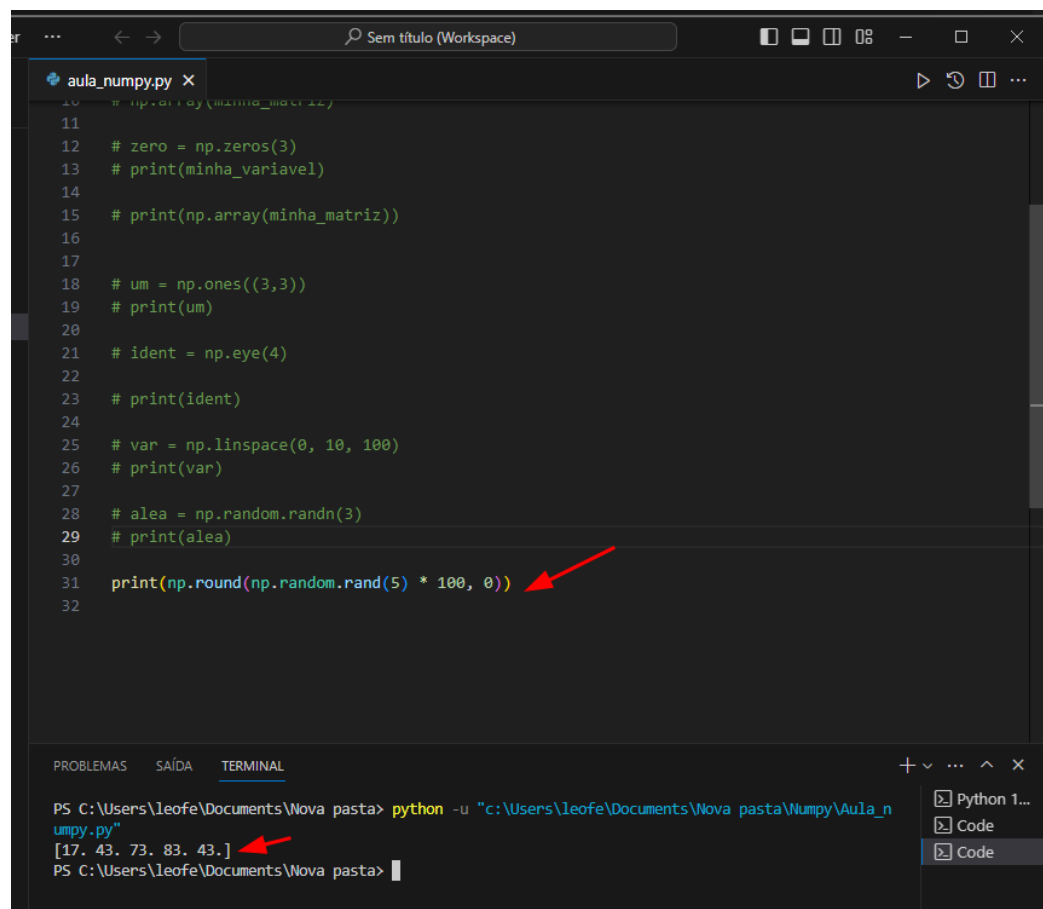
Utilizando função Randint é possível criar número “Aleatórios” como inteiros:

```
numero = np.random.randint(0, 100, 10)
print (numero)
```

### ▼ Rand -> Deixar como inteiro

É possível criar uma função semelhante ao randint:

```
np.round(np.random.rand(5)*100,0)
```



```
11 # np.random.rand(5) * 100
12 # zero = np.zeros(3)
13 # print(minha_variavel)
14
15 # print(np.array(minha_matriz))
16
17
18 # um = np.ones((3,3))
19 # print(um)
20
21 # ident = np.eye(4)
22
23 # print(ident)
24
25 # var = np.linspace(0, 10, 100)
26 # print(var)
27
28 # alea = np.random.randn(3)
29 # print(alea)
30
31 print(np.round(np.random.rand(5) * 100, 0))
32
```

PROBLEMAS SAÍDA TERMINAL

PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula\_numpy.py"

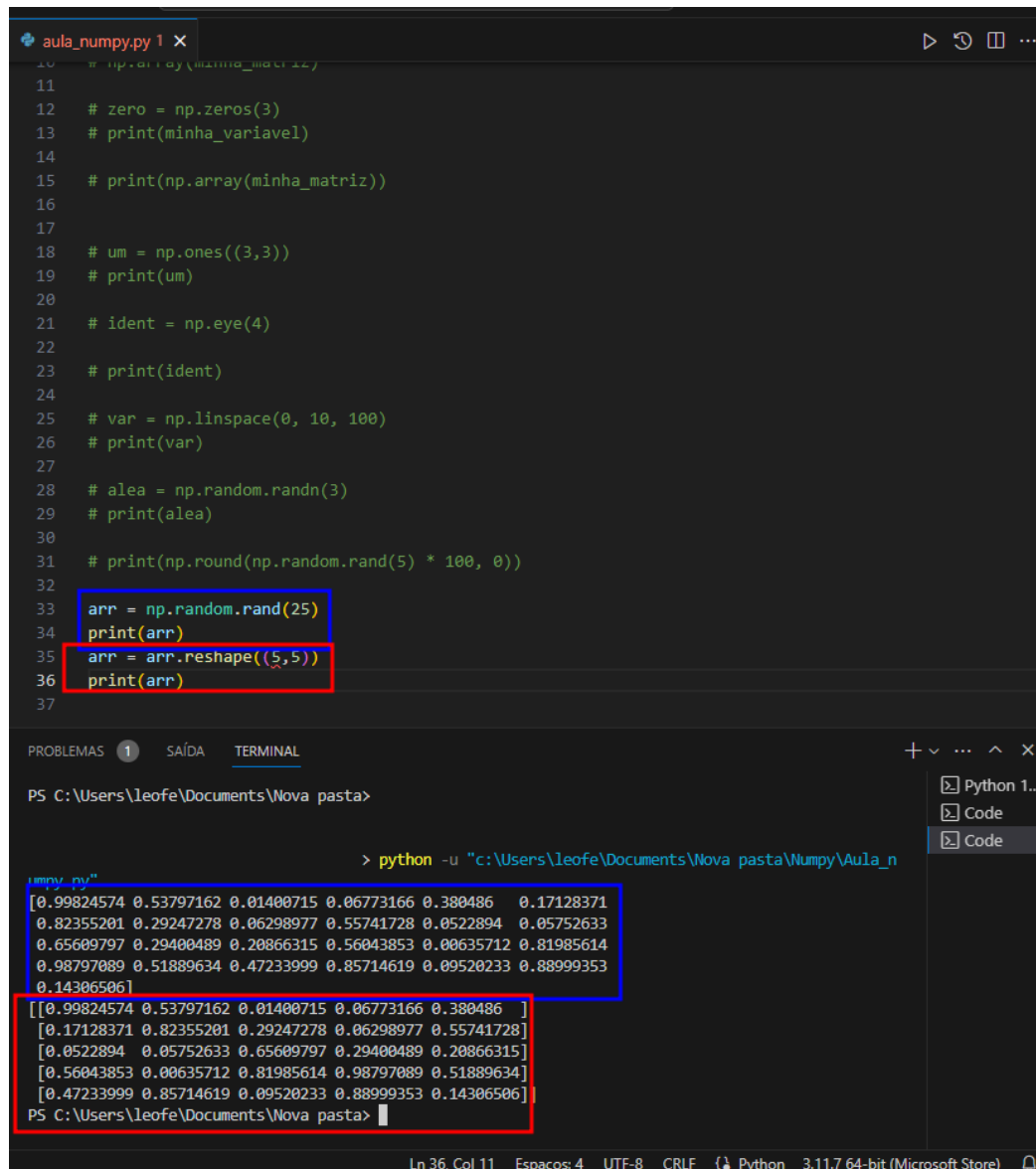
[17. 43. 73. 83. 43.]

PS C:\Users\leofe\Documents\Nova pasta>

### ▼ Reshape

É possível transformar um array:

```
arr = np.random.rand(25)
print(arr)
arr = arr.reshape((5,5))
print(arr)
```



The screenshot shows a Python IDE with a file named `aula_numpy.py`. The code defines several NumPy arrays and prints them. Lines 33-36 are highlighted with a blue box, and lines 34-35 are highlighted with a red box. The terminal output shows the result of the code execution, with the first 25 elements of the array printed on one line, and the reshaped 5x5 array printed on the next line.

```
20 # np.array(minha_matriz)
21
22 # zero = np.zeros(3)
23 # print(minha_variavel)
24
25 # print(np.array(minha_matriz))
26
27
28 # um = np.ones((3,3))
29 # print(um)
30
31 # ident = np.eye(4)
32 # print(ident)
33
34 # var = np.linspace(0, 10, 100)
35 # print(var)
36
37 # alea = np.random.randn(3)
38 # print(alea)
39
40 # print(np.round(np.random.rand(5) * 100, 0))
41
42
43 arr = np.random.rand(25)
44 print(arr)
45 arr = arr.reshape((5,5))
46 print(arr)
```

```
PS C:\Users\leofe\Documents\Nova pasta>
> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\Aula_n
umpy.py"
[0.99824574 0.53797162 0.01400715 0.06773166 0.380486 0.17128371
0.82355201 0.29247278 0.06298977 0.55741728 0.0522894 0.05752633
0.65609797 0.29400489 0.20866315 0.56043853 0.00635712 0.81985614
0.98797089 0.51889634 0.47233999 0.85714619 0.09520233 0.88999353
0.14306506]
[[0.99824574 0.53797162 0.01400715 0.06773166 0.380486 ]
[0.17128371 0.82355201 0.29247278 0.06298977 0.55741728]
[0.0522894 0.05752633 0.65609797 0.29400489 0.20866315]
[0.56043853 0.00635712 0.81985614 0.98797089 0.51889634]
[0.47233999 0.85714619 0.09520233 0.88999353 0.14306506]]
PS C:\Users\leofe\Documents\Nova pasta>
```

## ▼ Shape

Indica tamanho de um array

```
arr = np.random.rand(5,5)

arr.shape
```



The screenshot shows a Jupyter Notebook workspace with a file named 'aula\_numpy.py'. The code in the notebook includes several NumPy operations: creating a 3x3 ones matrix, a 4x4 identity matrix, a 1D linear space, random values, and a 5x5 random array. The last line of code is `arr.shape`, which is highlighted with a red box. To the right, the interactive console shows the output of the previous line, which is `(5, 5)`, also highlighted with a red box. The console indicates it is connected to Python 3.11.7.

#### ▼ Max / Min

Max = Retorna maior valor da matriz

Min = Retorna menor valor da matriz

#### ▼ Argmax / Argmin

Retorna o índice do maior/menor elemento

#### ▼ Indexação

Ao criar 2 arrays e definir os valores de um no outro, python faz um ponteiro do valor “copiado”, ou seja, se houver qualquer alteração em ambos arrays, o outro também compartilhará destas alterações.

A alternativa para isto, é utilizar a função Copy:

```
arr = np.arange(50).reshape((5, 10))
arr2 = arr[:3].copy
```

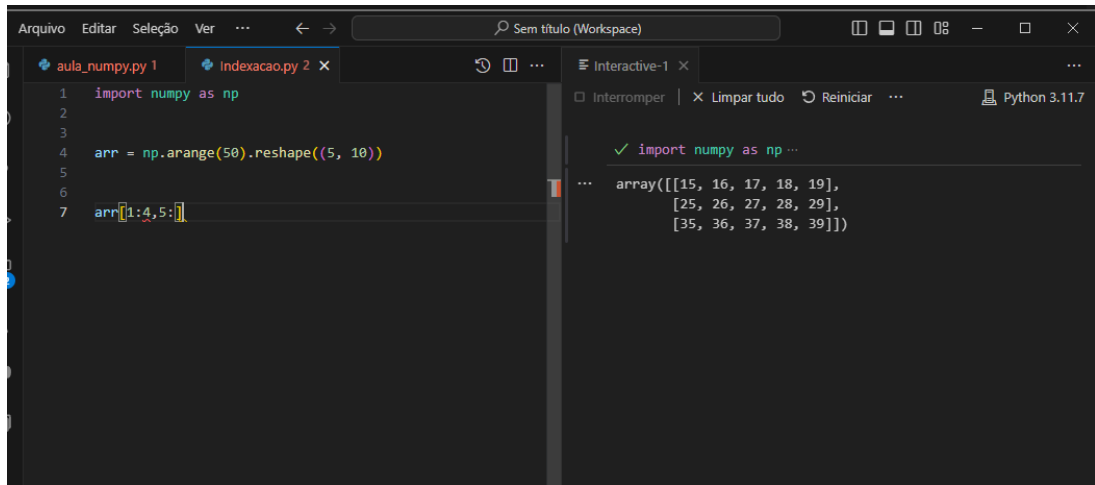
#### ▼ Slices

Antes vírgula linhas, após linhas colunas

Identificar partes destes dados:

```
arr = np.arange(50).reshape((5, 10))
```

```
arr[1:4,5:]
```



The screenshot shows a Jupyter Notebook interface. On the left, a code editor displays a Python script in a file named 'aula\_numpy.py'. The script imports numpy as np, creates an array 'arr' of size (5, 10) using np.arange(50).reshape((5, 10)), and then indexes it with arr[1:4,5:]. On the right, the interactive console shows the output of the script, which is a 3x5 array of integers: [[15, 16, 17, 18, 19], [25, 26, 27, 28, 29], [35, 36, 37, 38, 39]].

```
Arquivo Editar Seleção Ver ... Sem título (Workspace)
```

```
1 import numpy as np
2
3
4 arr = np.arange(50).reshape((5, 10))
5
6
7 arr[1:4,5:]
```

```
Interactive-1
Interromper | Limpar tudo Reiniciar Python 3.11.7
```

```
✓ import numpy as np ...
... array([[15, 16, 17, 18, 19],
          [25, 26, 27, 28, 29],
          [35, 36, 37, 38, 39]])
```

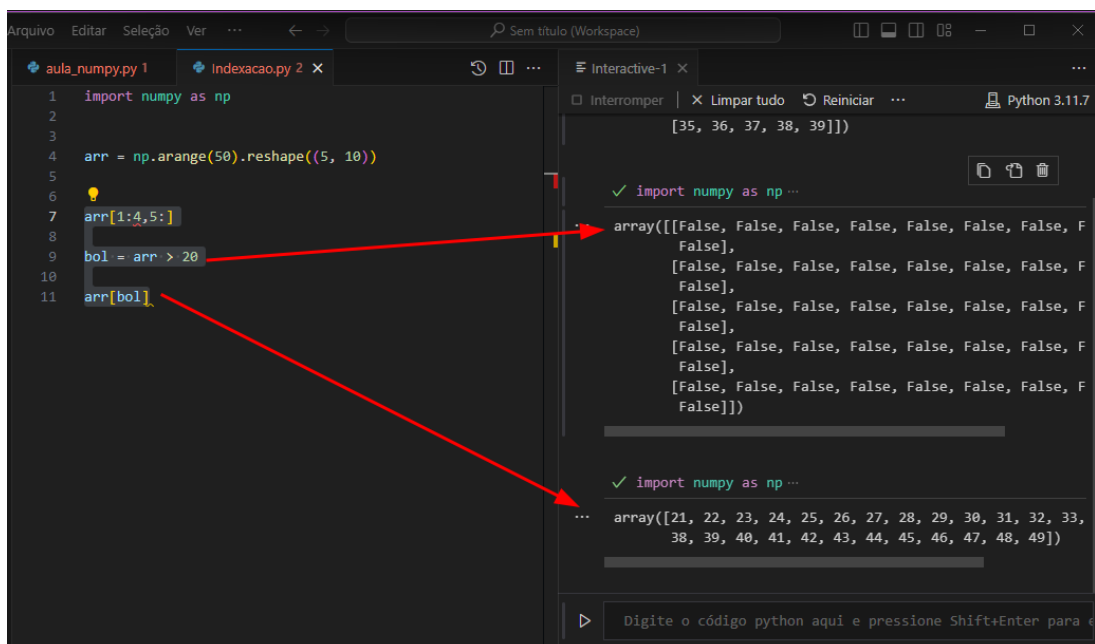
#### ▼ Anotação Booleanos

É possível realizar operações lógicas com matriz, afim de identificar determinados valores:

```
arr[1:4,5:]
```

```
bol = arr > 20
```

```
arr[bol]
```



The screenshot shows a Jupyter Notebook interface. On the left, a code editor displays a Python script in a file named 'aula\_numpy.py'. The script imports numpy as np, creates an array 'arr' of size (5, 10) using np.arange(50).reshape((5, 10)), and then indexes it with arr[1:4,5:]. Below this, it creates a boolean array 'bol' using bol = arr > 20, and finally indexes the original array with arr[bol]. Red arrows point from the 'bol' variable in the code to the corresponding output in the interactive console. On the right, the interactive console shows the output of the script, which is a 3x5 array of integers: [[15, 16, 17, 18, 19], [25, 26, 27, 28, 29], [35, 36, 37, 38, 39]]. Below this, it shows the output of the boolean indexing operation, which is a 3x5 array of booleans: [[False, False, False, False, False, False, False, False, False, False], [True, True, True, True, True, True, True, True, True, True], [True, True, True, True, True, True, True, True, True, True]].

```
Arquivo Editar Seleção Ver ... Sem título (Workspace)
```

```
1 import numpy as np
2
3
4 arr = np.arange(50).reshape((5, 10))
5
6
7 arr[1:4,5:]
8
9 bol = arr > 20
10
11 arr[bol]
```

```
Interactive-1
Interromper | Limpar tudo Reiniciar Python 3.11.7
```

```
✓ import numpy as np ...
... array([[15, 16, 17, 18, 19],
          [25, 26, 27, 28, 29],
          [35, 36, 37, 38, 39]])
```

```
✓ import numpy as np ...
... array([[False, False, False, False, False, False, False, False, False, False],
          [True, True, True, True, True, True, True, True, True, True],
          [True, True, True, True, True, True, True, True, True, True]])
```

▼ Selecionar determinados índices em um array:

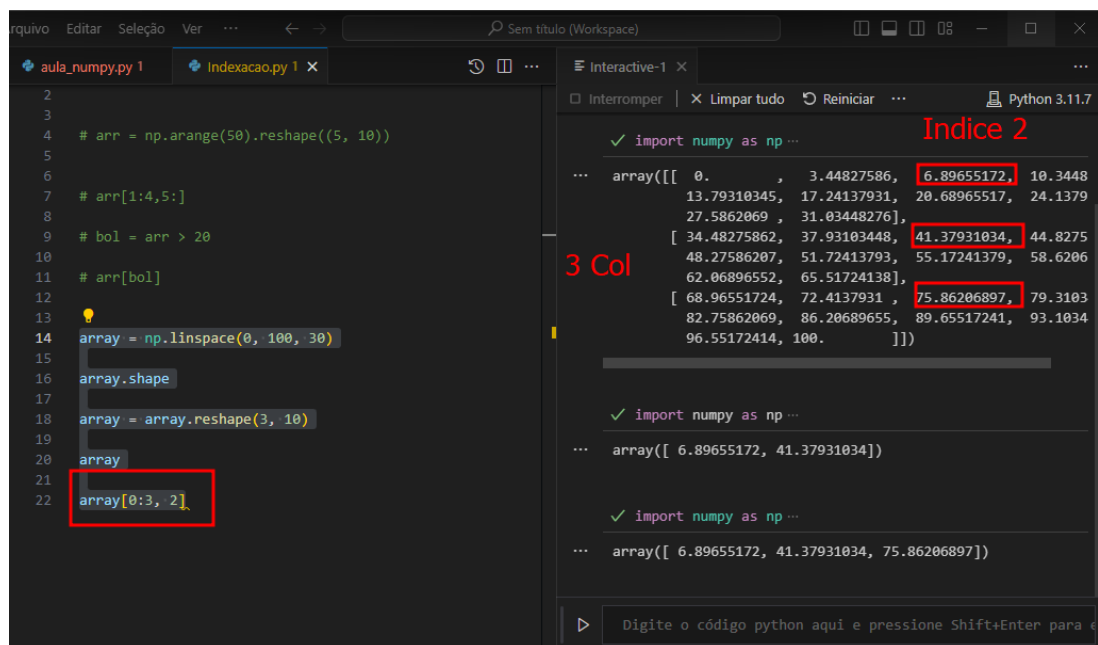
```
array = np.linspace(0, 100, 30)

array.shape

array = array.reshape(3, 10)

array

array[0:3, 2]
```



The screenshot shows a Jupyter Notebook with two tabs: 'aula\_numpy.py 1' and 'Indexacao.py 1 x'. The code in the first tab includes comments and operations on a NumPy array. The second tab shows the output of the code, which is a 3x10 array. The output is displayed as a table with 3 rows and 10 columns. The first row is highlighted in red, and the second column is highlighted in blue. The output is as follows:

	0.	3.44827586,	6.89655172,	10.3448					
...	13.79310345,	17.24137931,	20.68965517,	24.1379					
	27.5862069,	31.03448276,							
[	34.48275862,	37.93103448,	41.37931034,	44.8275					
	48.27586207,	51.72413793,	55.17241379,	58.6206					
	62.06896552,	65.51724138,							
[	68.96551724,	72.4137931,	75.86206897,	79.3103					
	82.75862069,	86.20689655,	89.65517241,	93.1034					
	96.55172414,	100.	]]						

▼ Operadores com arrays

É possível realizar operações aritméticas com 2 arrays(Ou o mesmo), através do seguinte método:

```
# Defina o array

arr = np.arange(1,17)

arr + arr

arr - arr
```

```
arr * arr
```

```
arr / arr
```

```
10 / arr
```

```
arr + 100
```

```
arr * 200
```

The screenshot displays a Jupyter Notebook environment with a dark theme. On the left, a code editor shows a file named `operadores.py` with the following Python code:

```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6 arr
7
8
9 arr + arr
10
11 arr - arr
12
13 arr * arr
14
15 arr / arr
16
17 10 / arr
18
19 arr + 100
20
21 arr * 200
```

On the right, the interactive console shows the results of each line of code, connected to Python 3.11.7:

- `import numpy as np`: Successful.
- `arr`: `array([ 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30])`
- `arr - arr`: Successful.
- `arr * arr`: `array([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])`
- `arr / arr`: Successful.
- `10 / arr`: `array([10. , 5. , 3.33333333, 2.5 , 1.66666667, 1.42857143, 1.25 , 1.11111111, 0.90909091, 0.83333333, 0.76923077, 0.71428571, 0.625 ])`
- `arr + 100`: `array([101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115])`
- `arr * 200`: `array([ 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400, 2600, 2800, 3000])`

At the bottom, a terminal window shows the command used to run the notebook:

```
PS C:\Users\leofe\Documents\Nova pasta> python -u "c:\Users\leofe\Documents\Nova pasta\Numpy\operadores.py"
```

Ele estará realizando operação índice a índice (Caso de escalar), no caso de 2 arrays, mesmo sendo diferentes, é necessário que tenham o mesmo índice para dar certo.

#### ▼ Funções numpy

`sqrt(array)`

Tira raiz quadrada desse array

```
np.sqrt(arr)
```

```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6
7
8 np.sqrt(arr)
9
```

Conectado a Python 3.11.7

```
✓ import numpy as np ...
... array([1.         , 1.41421356, 1.73205081, 2.         , 2.23606798,
2.44948974, 2.64575131, 2.82842712, 3.         , 3.16227766,
3.31662479, 3.46410162, 3.60555128, 3.74165739, 3.8268343,
3.95959177, 4.         ])
```

`exp(array)`

exponenciação

```
np.exp(arr)
```

```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6
7 np.exp(arr)
8
```

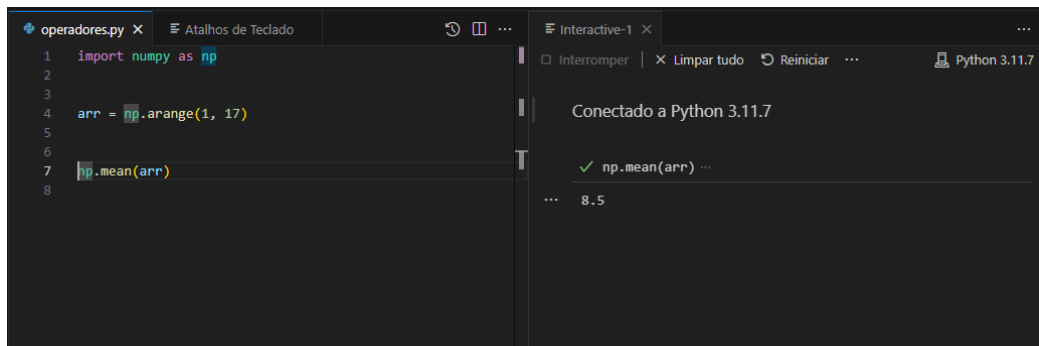
Conectado a Python 3.11.7

```
✓ import numpy as np ...
... array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.48811636e+01,
1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
8.10308393e+03, 2.20264658e+04, 5.98741417e+04, 1.64703130e+05,
4.42413392e+05, 1.20260428e+06, 3.26901737e+06, 8.86179715e+06])
```

mean(array)

Média de todos elementos

```
np.mean(arr)
```



The screenshot shows a Jupyter Notebook interface. On the left, a code editor displays the following Python code:

```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6
7 np.mean(arr)
8
```

On the right, the interactive console shows the output of the last cell:

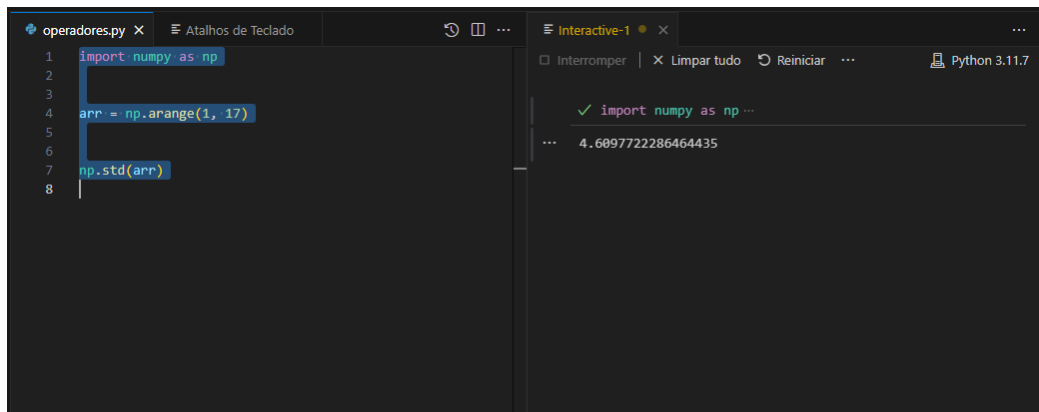
```
Conectado a Python 3.11.7

✓ np.mean(arr) ...
... 8.5
```

std(array)

Padrão de todos elementos

```
np.std(arr)
```



The screenshot shows a Jupyter Notebook interface. On the left, a code editor displays the following Python code:

```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6
7 np.std(arr)
8
```

On the right, the interactive console shows the output of the last cell:

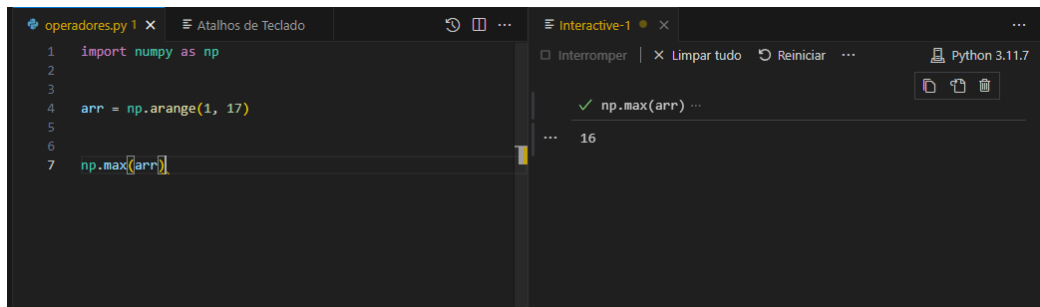
```
Conectado a Python 3.11.7

✓ import numpy as np ...
... 4.6097722286464435
```

max(array)

Maior elemento

```
np.max(arr)
```



The screenshot shows a Jupyter Notebook interface. The left pane contains a Python file named 'operadores.py' with the following code:

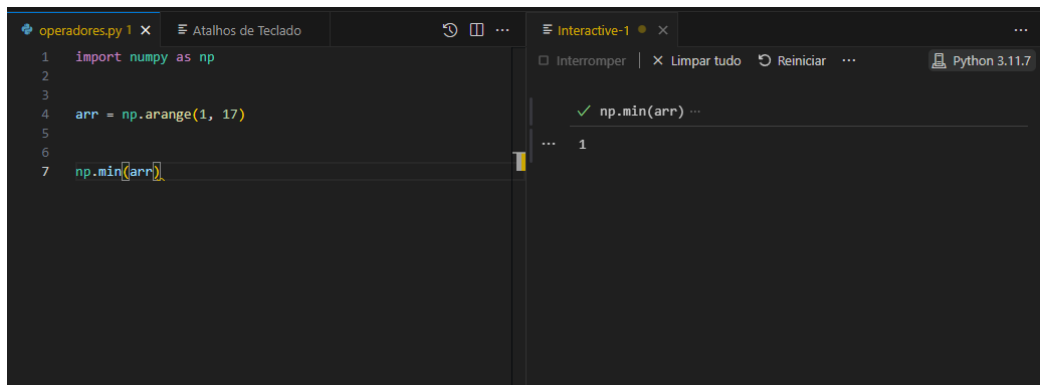
```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6
7 np.max(arr)
```

The right pane shows the execution of the code, with a green checkmark indicating success and the output '16'.

`min(array)`

Menor elemento

```
np.min(arr)
```



The screenshot shows a Jupyter Notebook interface. The left pane contains a Python file named 'operadores.py' with the following code:

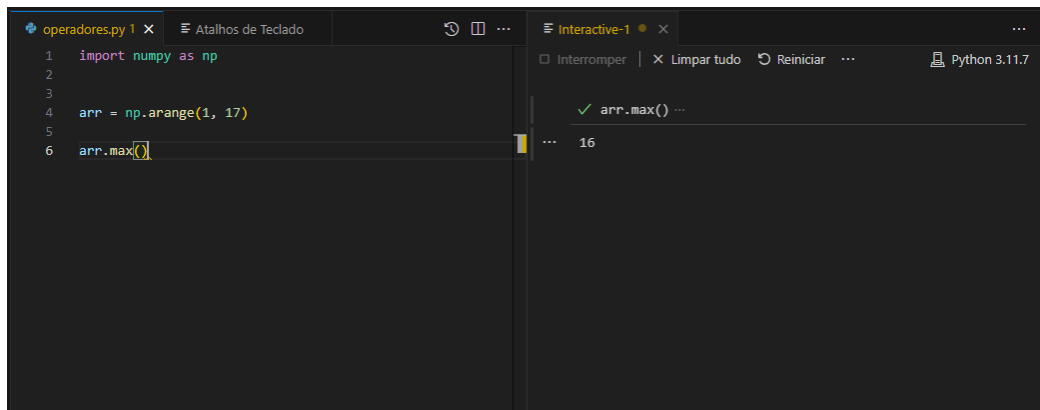
```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6
7 np.min(arr)
```

The right pane shows the execution of the code, with a green checkmark indicating success and the output '1'.

`array.max`

Chamar função sob o objeto array

```
arr.max()
```



The screenshot shows a Jupyter Notebook interface. The left pane contains a Python file named 'operadores.py' with the following code:

```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6 arr.max()
```

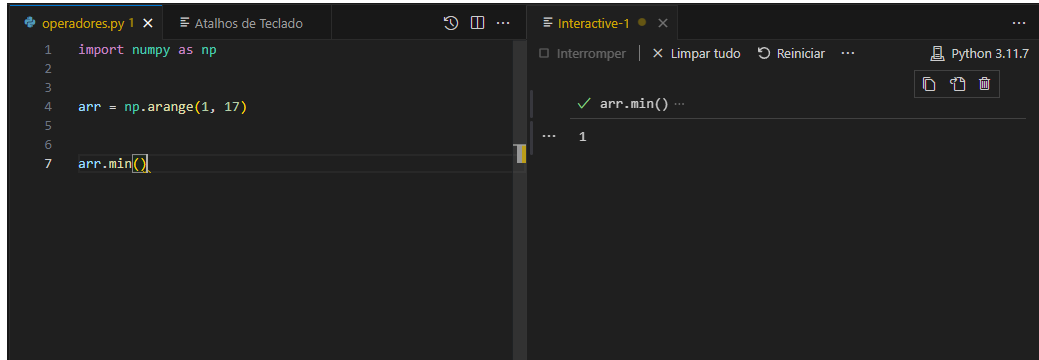
The right pane shows the execution of the code, with a green checkmark indicating success and the output '16'.

`array.min`



Chamar função sob o objeto array

```
arr.min()
```



The screenshot shows a Jupyter Notebook interface. On the left, a code editor displays the following Python code:

```
1 import numpy as np
2
3
4 arr = np.arange(1, 17)
5
6
7 arr.min()
```

On the right, the interactive console shows the execution of the code, resulting in the output:

```
✓ arr.min() ...
... 1
```

#### ▼ Atividades:

Ocorram alguns divergências nas respostas. Segue na sequência as mesmas:



The image shows two side-by-side Jupyter Notebook screenshots. The left notebook is titled 'Crie uma matriz de 10 cincos' and shows the code:

```
np.linspace(5, 5, 10)
```

The output is:

```
array([5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

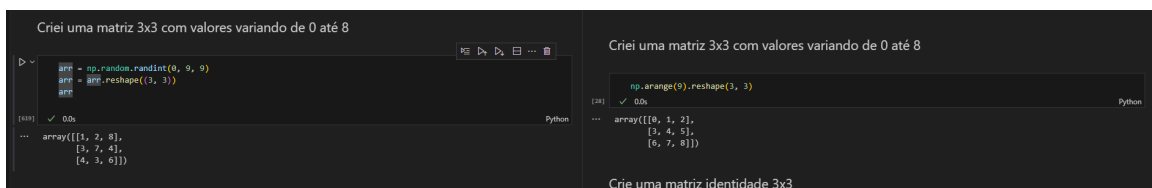
The right notebook is also titled 'Crie uma matriz de 10 cincos' and shows the code:

```
np.ones(10) * 5
```

The output is:

```
array([ 5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.])
```

Aqui busquei gerar através da função linspace, já na resolução, professor preferiu utilizar uma operação encima de uma função



The image shows two side-by-side Jupyter Notebook screenshots. The left notebook is titled 'Crie uma matriz 3x3 com valores variando de 0 até 8' and shows the code:

```
arr = np.random.randint(0, 9, 9)
arr = arr.reshape(3, 3)
```

The output is:

```
array([[1, 2, 8],
       [3, 7, 4],
       [4, 3, 6]])
```

The right notebook is also titled 'Crie uma matriz 3x3 com valores variando de 0 até 8' and shows the code:

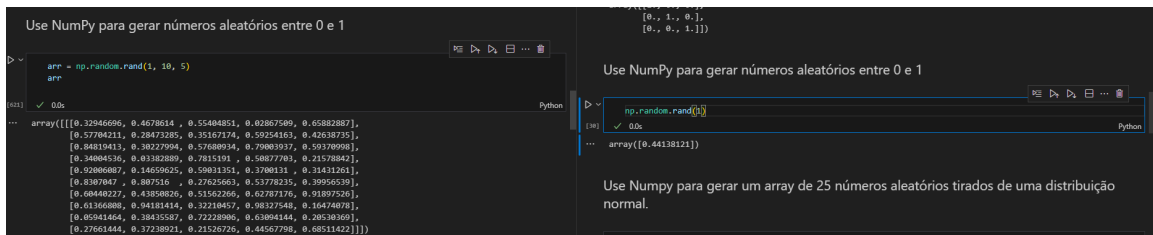
```
np.arange(9).reshape(3, 3)
```

The output is:

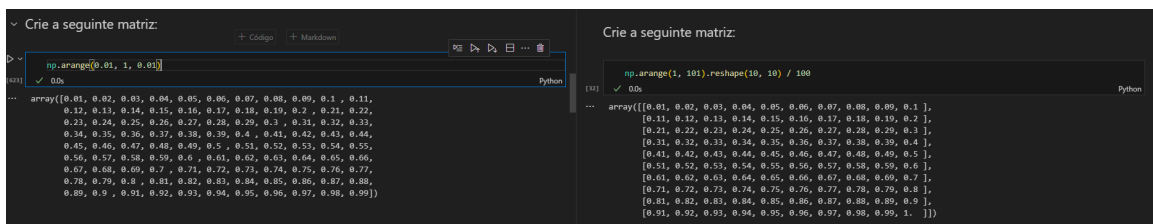
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Tivemos também um diferença quanto criação de matriz com valores variáveis de 0-8. Não me dei conta fazendo, que poderia utilizar o array criado anteriormente para realizar o random.

Deste modo, defini um novo array de 0-9 contendo 9 valores ao todo, e dei um reshape, afim de ordenar eles.

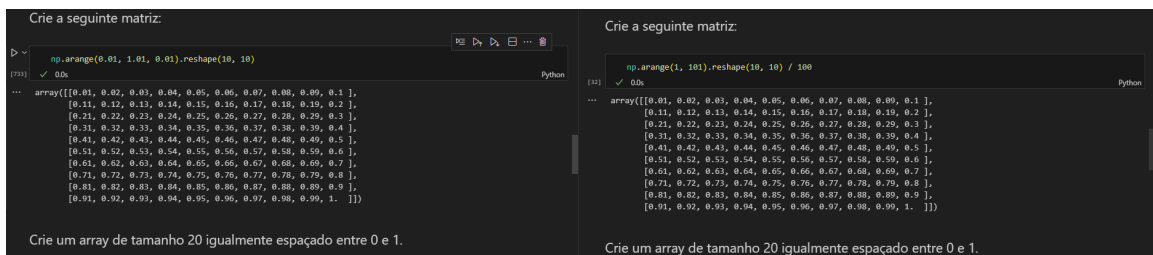


Outra diferença foi a questão de número aleatórios entre 0 e 1. Na questão indica números, mas resposta dele só conta um. De qualquer modo, seria somente uma questão de interpretação.



Averiguando, havia errado também seguinte questão.

A principio seria somente uma questão de ordenar as matrizes, adicionando um reshape.



Outras diferenças menores, foram quanto uso de funções para matriz, do qual, utilizei o próprio numpy para fazer, enquanto ele utilizou de funções de matriz

