**Sumário**

[Arrays 3](#_Toc42740347)

[Array 3](#_Toc42740348)

[Copiando 3](#_Toc42740349)

[Casting 4](#_Toc42740350)

[NaN (Not a Number) 5](#_Toc42740351)

[Infinito 5](#_Toc42740352)

[Básico de NumPy 6](#_Toc42740353)

[Ranged Data 6](#_Toc42740354)

[Arange() 6](#_Toc42740355)

[Linspace 6](#_Toc42740356)

[Reshaping Data (Remodelando dados) 7](#_Toc42740357)

[Reshape 7](#_Toc42740358)

[Flatten (achatar) 9](#_Toc42740359)

[Transposição 9](#_Toc42740360)

[Transpose 9](#_Toc42740361)

[Tranpose with axes 11](#_Toc42740362)

[Zeros e Uns 12](#_Toc42740363)

[Matemática 14](#_Toc42740364)

[Aritmética 14](#_Toc42740365)

[Funções Não-Lineares 15](#_Toc42740366)

[Exponenciais e logaritmos 15](#_Toc42740367)

[Potência 16](#_Toc42740368)

[Multiplicação de Matrizes 16](#_Toc42740369)

[Random 18](#_Toc42740370)

[Inteiros randômicos 18](#_Toc42740371)

[Funções utilitárias 18](#_Toc42740372)

[Seed 18](#_Toc42740373)

[Shuffle 18](#_Toc42740374)

[Amostragem (Choice) 19](#_Toc42740375)

[Indexação 21](#_Toc42740376)

[Acessando Array 21](#_Toc42740377)

[Slicing (Cortando) 21](#_Toc42740378)

[Array 1-D 21](#_Toc42740379)

[Matriz 22](#_Toc42740380)

[Argmin e Argmax 23](#_Toc42740381)

[Argmin e Argmax without axis 23](#_Toc42740382)

[Argmin e Argmax with axis 24](#_Toc42740383)

[Filtragem 26](#_Toc42740384)

[Filtrando dados 26](#_Toc42740385)

[Is a NaN 27](#_Toc42740386)

[Filtrando no NumPy 27](#_Toc42740387)

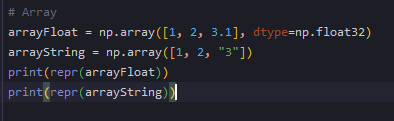
# Arrays

## Array

Os Arrays no NumPy são como as Listas no Python com recursos adicionais. Podemos criar usando o comando: **np.array().**

Esse método possui um argumento que é o **dtype** que recebe um tipo NumPy e converte manualmente a matriz para o tipo especificado.

**Code:**



**Output:**



Quando os tipos são mistos, por exemplo: a lista possui tipos inteiros, float ou até Strings, ela é convertida automaticamente para o tipo de maior nível.

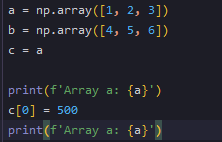
Exemplo:

* Uma lista com inteiros e floats -> Lista de floats
* Uma lista com inteiros, floats e Strings -> Lista de strings

## Copiando

Semelhante ao Python, quando fazemos uma referência a uma lista, na verdade não criamos uma lista. Logo se alterarmos o valor da referência também irá mudar o valor no array original. Podemos contornar isso usando a função **copy()** que retorna o array copiado.

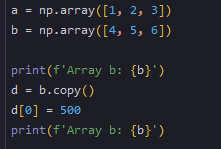
**Code sem .copy():**



**Output:**



**Code com .copy():**



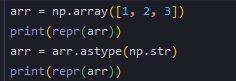
**Output:**



## Casting

Podemos converter um array usando a função **.astype()**, que recebe um tipo NumPy como argumento e retorna o array convertido.

**Code:**



**Output:**



## NaN (Not a Number)

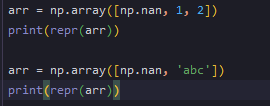
NaN é usado como um espaço reservado para dados ausentes consistentemente em pandas.

**np.nan** permite operações vetorizadas; é um valor flutuante, enquanto **None**, por definição, força o tipo de **objeto**, o que basicamente desativa toda a eficiência em NumPy.

**NaN** pode ser usado como um valor numérico em operações matemáticas, enquanto **None** não pode (ou pelo menos não deveria).

Um uso comum para **np.nan** é como um valor de preenchimento para dados incompletos.

**Code:**



**Output:**



## Infinito

Para representar o infinito no NumPy, usamos a função **np.inf.**

Infinito não pode assumir o tipo inteiro

# Básico de NumPy

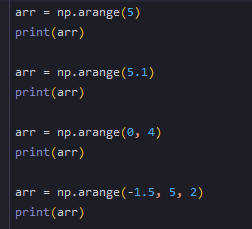
## Ranged Data

### Arange()

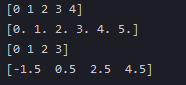
Muito similar com **range()** do Python, no NumPy temos a função **arange()** que retorna um array de 1-dimensão.

* Caso exista somente 1 argumento, será gerado um array 1-D de **0 até n-1**.
* Caso exista 2 argumentos, será gerado um array 1-D de **m até n-1.**
* Caso existam 3 argumentos, será gerado um array 1-D de **m até n-1** com o passo do terceiro argumento.

**Code:**



**Output:**

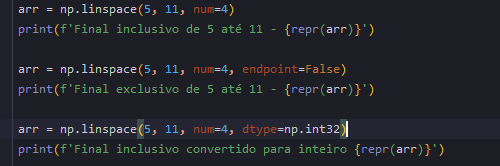


### Linspace

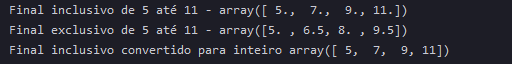
Para especificar o número de elementos que será retornado, usamos **np.linspace().**

Essa função recebe os dois primeiros argumentos necessários, para o início e o fim do intervalo, respectivamente. O final do intervalo é inclusivo para **np.linspace**, a menos que o argumento da palavra-chave **endpoint** esteja definido como **False**. Para especificar o número de elementos, definimos o **num** argumento da palavra - chave (seu valor padrão é 50).

**Code:**



**Output:**



## Reshaping Data (Remodelando dados)

### Reshape

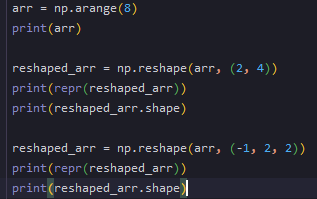
A função que usamos para remodelar os dados no NumPy é **np.reshape**. Ele assume uma matriz e uma nova forma como argumentos necessários.

A nova forma **deve conter exatamente** todos os elementos da matriz de entrada. Por exemplo, podemos remodelar uma matriz com 12 elementos para (4, 3), mas não podemos remodelá-la para (4, 4).

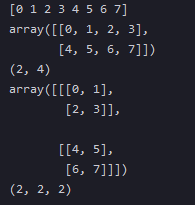
É permitido usar o valor especial de -1 em no máximo uma dimensão da nova forma. A dimensão com -1 assumirá o valor necessário para permitir que a nova forma contenha todos os elementos da matriz.

O primeiro argumento é o próximo array, logo em seguida de uma **tupla** com a forma nova.

**Code:**



**Output:**



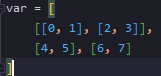
O Array inicialmente era de 1 dimensão [0 1 2 3 4 5 6 7]

Porém virou um array com 2 dimensões – 2 Linhas e 4 colunas (2, 4)

Poderia também virar um array de 4 linhas e 2 colunas (4, 2)

E com o argumento -1, ele assumiu 3 dimensões (2, 2, 2), ou seja, um array que contém 2 arrays que contêm 2 arrays.

O argumento -1 assumiu o valor no qual pudesse caber todos os valores.



Com o atributo **.shape** podemos saber o formato de uma matriz.

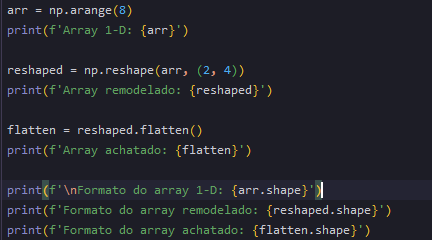
O **shape** de var é (2, 2, 2)

### Flatten (achatar)

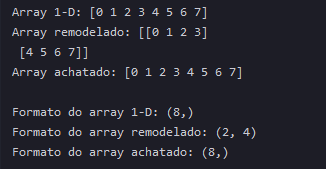
Podemos transformar um array para um array de 1-dimensão, utilizando a função **arr.flatten().**

Ele retorna um array de 1-D.

**Code:**



**Output:**



O Array inicialmente era de formato (8,)

Após remodelado virou um array de formato (2, 4)

E quando achatado voltou a ser um array de formato (8,)

## Transposição

### Transpose

Na álgebra, permutação que troca somente as posições de dois elementos.

Semelhante à forma como é comum remodelar dados, também é comum transpor dados.

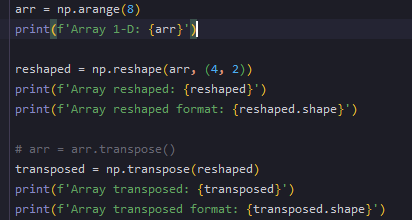
Talvez tenhamos dados que deveriam estar em um formato específico, mas alguns dados novos que obtemos foram reorganizados. Podemos apenas transpor os dados, usando a **np.transpose** função, para convertê-los no formato apropriado.

As linhas da matriz se tornam colunas após a transposição e vice-versa.

(4, 2) -> (2, 4)

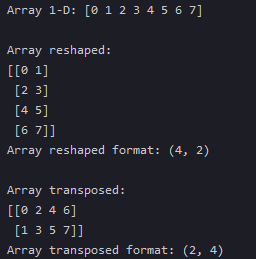
(2, 4) -> (4, 2)

**Code:**



Note que também podemos usar o **array.transpose()** para retornar o array convertido.

**Output:**



### Tranpose with axes

A função np.transpose() além de ter o argumento necessário do array a ser convertido, também temos o argumento de palavra-chave **axes,** que representa a nova permutação das dimensões.

A permutação é um tupla com o mesmo comprimento do formato do array. Então se temos um array de formato (2, 3, 4) a permutação deverá ter 3 números, e cada número será uma posição.

Por exemplo:

(2, 3, 4):

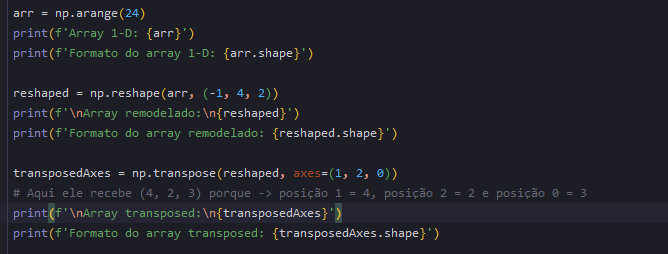
* Posição 0: 2
* Posição 1: 3
* Posição 2: 4

Agora vamos dar um argumento para axes da seguinte maneira: (2, 1, 0). Ao fazer isso a nova forma será **(4, 3, 2).**

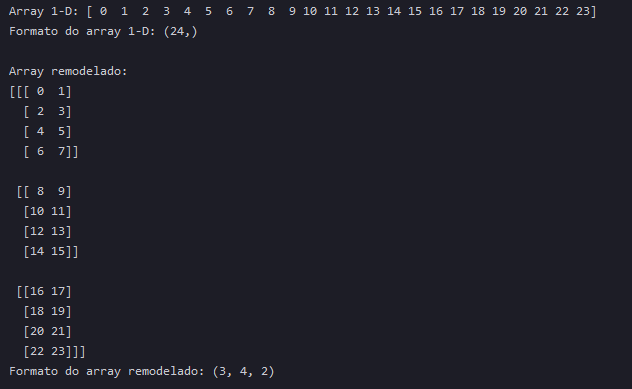
Ou até mesmo: (1, 2, 0), que será **(3, 4, 2).**

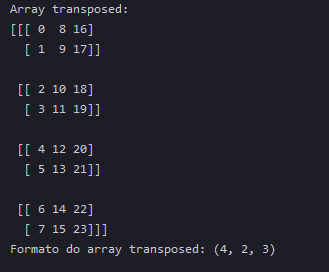
Caso não especificado o argumento axes, ele receberá o valor padrão de inverter o formato.

**Code:**



**Output:**

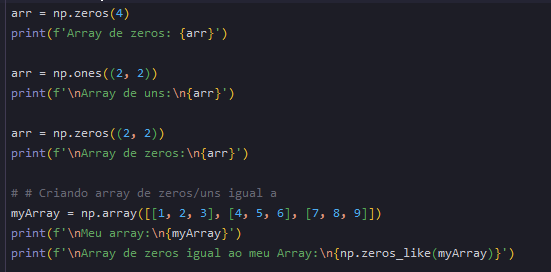




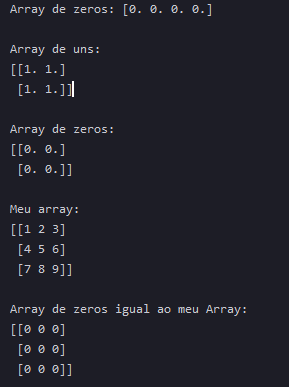
## Zeros e Uns

Às vezes, precisamos criar matrizes preenchidas apenas com 0 ou 1. Por exemplo, como os dados binários são rotulados com 0 e 1, talvez seja necessário criar conjuntos de dados fictícios estritamente de um rótulo. Para criar essas matrizes, o NumPy fornece as funções **np.zeros** e **np.ones.**

**Code:**



**Output:**

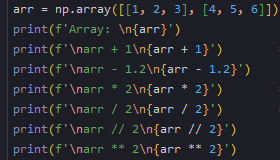


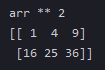
# Matemática

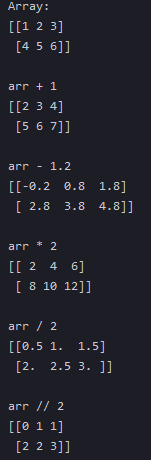
## Aritmética

Usando matrizes NumPy, podemos aplicar aritmética a cada elemento com uma única operação.

**Code:**



**Output:**



É importante observar que executar aritmética em matrizes NumPy não altera a matriz original e, em vez disso, produz uma nova matriz que é o resultado da operação aritmética.

## Funções Não-Lineares

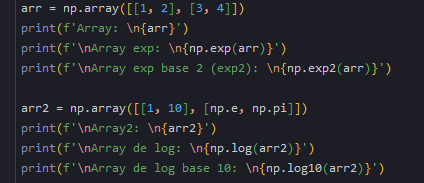
### Exponenciais e logaritmos

Além das operações aritméticas básicas, o NumPy também permite o uso de funções não lineares, como exponenciais e logaritmos.

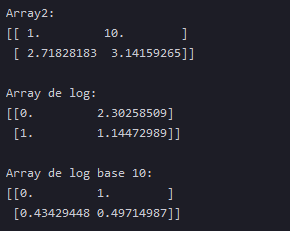
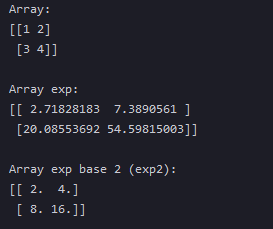
A função **np.exp** executa uma base e exponencial em uma matriz, enquanto a função **np.exp2** executa uma base 2 exponencial.

Da mesma forma, **np.log**, **np.log2**, e **np.log10** todos os logaritmos em realizar uma matriz de entrada, utilizando uma base e, de base 2, e a base 10, respectivamente.

**Code:**



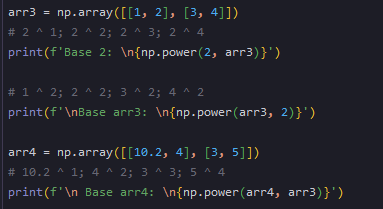
**Output:**



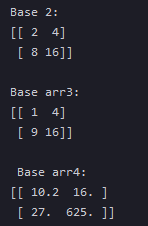
### Potência

O primeiro argumento para a função é a **base**, enquanto o segundo é o **expoente**. Se a base ou expoente for uma matriz em vez de um único número, a operação será aplicada a todos os elementos da matriz.

**Code:**



**Output:**



## Multiplicação de Matrizes

Como matrizes NumPy são basicamente vetores e matrizes, faz sentido que existam funções para produtos de pontos e multiplicação de matrizes. Especificamente, a principal função a ser utilizada é a **np.matmul** que recebe duas matrizes de vetor / matriz como entrada e produz um produto de ponto ou multiplicação de matriz.

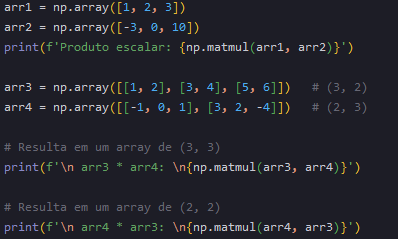
O código abaixo mostra vários exemplos de multiplicação de matrizes. Quando as duas entradas são 1-D, a saída é o **produto escalar**.

Observe que as dimensões das duas matrizes de entrada devem ser válidas para uma multiplicação de matrizes. Especificamente, a segunda dimensão da primeira matriz deve ser igual à primeira dimensão da segunda matriz.

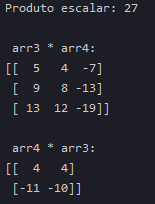
Por exemplo: Matriz de (2, **3**) e uma Matriz de (**3**, 2) são válidas, porque a segunda dimensão do primeiro array é 3 e a primeira dimensão do segundo array também é 3.

Matriz de (2, **3**) e uma matriz de (**2**, 3) não é válida.

**Code:**



**Output:**

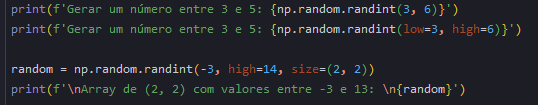


# Random

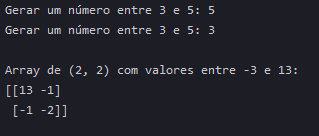
## Inteiros randômicos

Semelhante ao **random,** NumPy também possui um método para geração de números pseudoaleatórios chamado de **np.random.** Para gerar números inteiros pseudoaleatórios, usamos **np.random.randint.**

**Code:**



**Output:**



## Funções utilitárias

### Seed

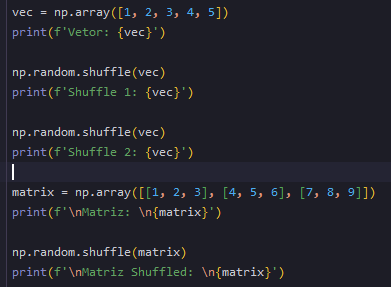
Usamos a função **np.random.seed()** para determinar a semente aleatória que irá nos permitir controlar os resultados dos números pseudoaleatórios. Uma semente aleatória é um valor que irá inicializar um algoritmo gerador de números.

Caso seja definido uma semente, ele sempre irá retornar os mesmos valores gerados, caso contrário ele irá usar o relógio da máquina como uma semente, garantindo assim a aleatoriedade da aplicação.

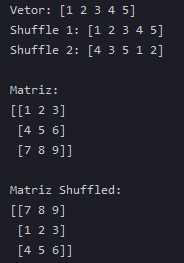
### Shuffle

A **np.random.shuffle** nos permite embaralhar aleatoriamente uma matriz. Observe que o embaralhamento ocorre no lugar (ou seja, sem valor de retorno) e o embaralhamento de matrizes multidimensionais embaralha **apenas a primeira dimensão**.

**Code:**



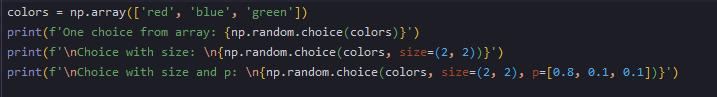
**Output:**



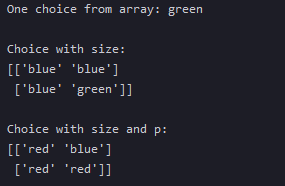
## Amostragem (Choice)

Embora o NumPy forneça distribuições internas para fazer a amostra, também podemos fazer a amostra de uma distribuição personalizada com o método **np.random.choice.**

**Code:**



**Output:**



No primeiro exemplo temos apenas uma amostra do array.

Na segunda temos uma amostra em formato de um array de 2, 2.

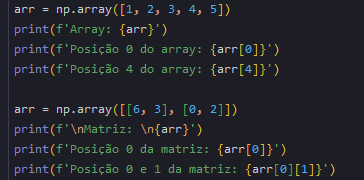
E na última temos o mesmo exemplo anterior, porém com a diferença de ter o argumento de chave **p** que é a probabilidade de cada item do array em aparecer. Nesse caso o item ‘red’ tem 80% de chance de aparecer, enquanto os outros dois tem apenas 10%, somando no fim 100%.

# Indexação

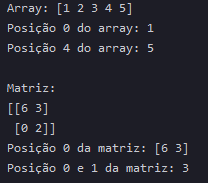
## Acessando Array

O acesso às matrizes NumPy é idêntico ao acesso às listas Python. Para matrizes multidimensionais, é equivalente a acessar listas de listas Python.

**Code:**



**Output:**



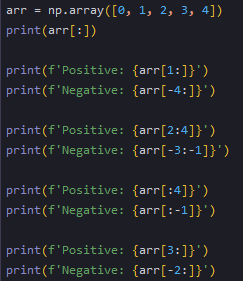
## Slicing (Cortando)

### Array 1-D

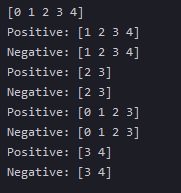
As matrizes NumPy também suportam o fatiamento. Semelhante ao Python, usamos o operador dois pontos ([:]) para fatiar.

Também podemos usar a indexação negativa para cortar na direção inversa.

**Code:**



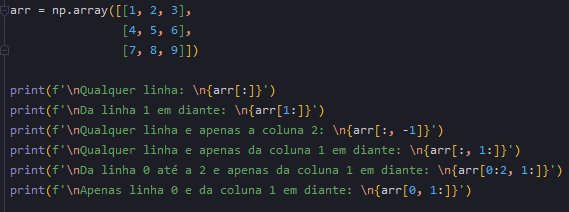
**Output:**



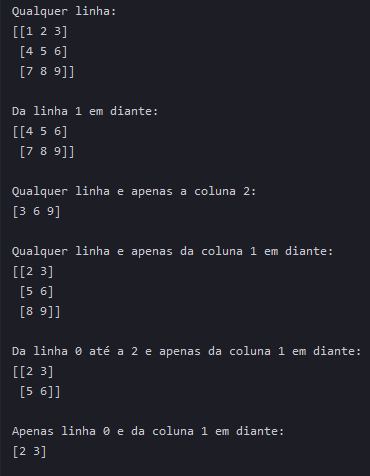
### Matriz

O processo para matrizes é o mesmo, separando por vírgula.

**Code:**



**Output:**

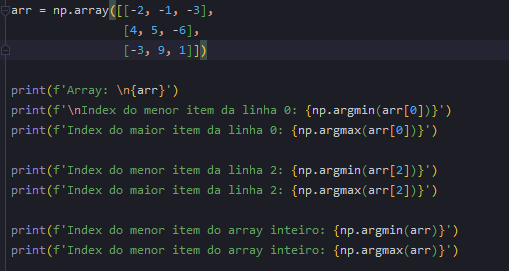


## Argmin e Argmax

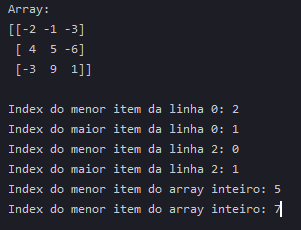
### Argmin e Argmax without axis

Podemos usar as funções **np.argmin** e **np.argmax** para conseguir o índice do menor e maior item de um array.

**Code:**

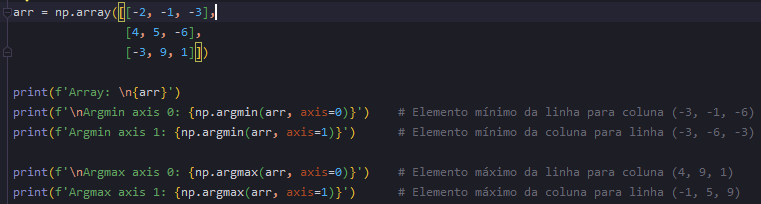


**Output:**

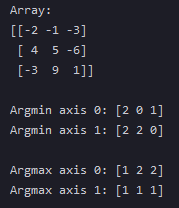


### Argmin e Argmax with axis

**Code:**



**Output:**



# Filtragem

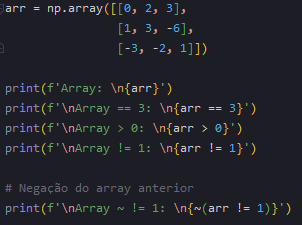
## Filtrando dados

Ao ter vários dados, às vezes queremos filtrar esses valores que tenham uma determinada condição. Por exemplo, o valor de compra mensal dos clientes, desejo filtrar esses dados e me apresentar apenas aqueles que gastam mais de mil reais por mês, com o NumPy não podemos fazer isso.

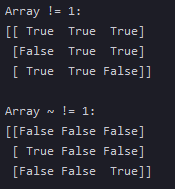
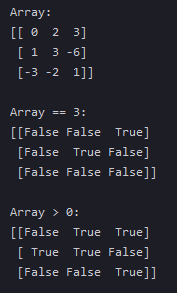
A chave para filtrar dados é através de operações básicas de relação, por exemplo **==**, **>**

A operação **~** representa uma negação booleana, ou seja, **inverte** cada valor da matriz booleana.

**Code:**



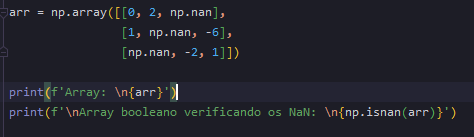
**Output:**



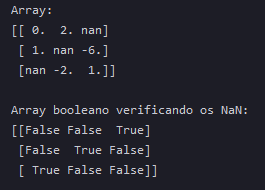
### Is a NaN

Como não é possível usar essas operações de relação com dados NaN, para isso usamos o método **np.isnan(),** que tem como argumento um array e retorna uma matriz booleana, se for um NaN a posição será **TRUE**, caso contrário será **FALSE.**

**Code:**



**Output:**



Cada matriz booleana em nossos exemplos representa a **localização** dos elementos que queremos filtrar. A maneira como realizamos a filtragem em si é através do método **np.where**.

## Filtrando no NumPy