código.py

GALÁXIA 3

Numpy

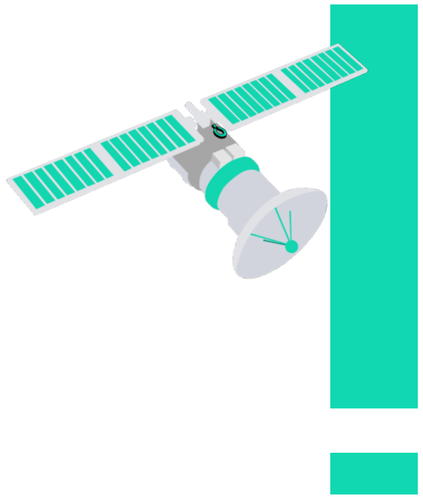
Introdução

Seja bem-vindo à galáxia 3! Nas galáxias anteriores focamos nos assuntos que já vinham nativos do Python, mas agora avançaremos um pouco e abordaremos uma das bibliotecas mais utilizadas do Python, o NumPy. Esta biblioteca é mais uma das ferramentas que você deve dominar para trabalhar com dados e é importante que você entenda suas funcionalidades, usufruindo do máximo de coisas que o NumPy pode fazer por você.

Mundo 1

1. O que é o NumPy?

NumPy é uma biblioteca que serve para trabalhar com vetores dentro da programação. Uma lista de números pode ser um vetor, por exemplo. Por mais que a biblioteca NumPy seja feita para trabalhar dentro do Python, sua estrutura é construída pela linguagem de programação C, que é considerada uma das linguagens mais rápidas do mundo.



2. Para que serve o NumPy?

Esta biblioteca foi criada para fazer operações matemáticas com vetores e, por ter sua estrutura formada na linguagem C, ela também pode servir para otimizar o código. Por mais que as listas, que vêm por padrão no Python, sejam parecidas com a estrutura do array, que são objetos dentro do NumPy, veremos que os arrays são bem mais rápidos.

Mundo 2

1. Importando o módulo

1.1. Escolhendo o nome

Ao importar uma biblioteca, podemos usar o comando “as” para definir o nome que a biblioteca importada será chamada dentro do programa. No exemplo abaixo, nós importamos a biblioteca NumPy usando o comando import numpy as np e nomear como np é uma forma padrão muito utilizada.

```
import numpy as np
```

2. Criando um array a partir de uma lista ou tupla

2.1. Função np.array()

Essa função recebe como parâmetro uma lista ou uma tupla e retorna um array. Lembre-se sempre que ao utilizar uma função de uma biblioteca, precisamos especificar qual biblioteca estamos utilizando, no nosso caso é a NumPy.

Exemplo:

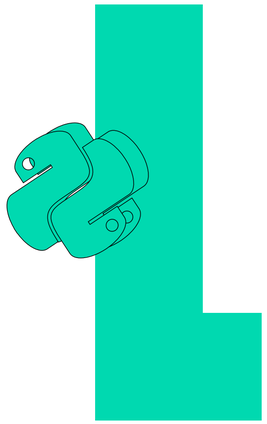
```
import numpy as np
#a partir de uma lista ou tupla
lista = [1, 2, 3, 4]
tupla = (1, 2, 3, 4)

array_a_partir_de_lista = np.array(lista)
array_a_partir_de_tupla = np.array(tupla)

print(array_a_partir_de_tupla)
print(array_a_partir_de_lista)
```

Resposta:

```
>> [1 2 3 4]
>> [1 2 3 4]
```



3. Criando um array a partir de um range

3.1. Função np.arange()

Essa função recebe como parâmetro obrigatório o final do intervalo. Caso seja o único parâmetro, a função definirá que o começo é no "0" com intervalos de "1". Pode receber também o início e o intervalo desejado do range.

Exemplo

```
import numpy as np
#usando um range
array_simples = np.arange(10) # Tamanho do intervalo começando no 0
array_sequencial = np.arange(10, 100, 10) # Início, Final e o intervalo

print("Array simples: ", array_simples)
print("Array sequencial: ", array_sequencial)
```

Resposta:

```
>> Array simples:  [0 1 2 3 4 5 6 7 8 9]
>> Array sequencial:  [10 20 30 40 50 60 70 80 90]
```

4. Criando um vetor de zeros e uns

4.1. Função np.zeros()

Essa função recebe como parâmetro obrigatório o tamanho do array e retorna um array de zeros do tamanho escolhido.

4.2. Função np.ones()

Essa função recebe como parâmetro obrigatório o tamanho do array e retorna um array de uns do tamanho escolhido.

Exemplo:

```
import numpy as np
#criando arrays de zeros ou ums
vetor_zero = np.zeros(5)
vetor_um = np.ones(5)

print("Array com zeros ", vetor_zero)
print("Array com uns ", vetor_um)
```

Resposta:

```
>> Array com zeros  [0. 0. 0. 0. 0.]
>> Array com uns  [1. 1. 1. 1. 1.]
```

5. Gerando números aleatórios

5.1. Função np.random.rand()

Essa função recebe como parâmetro obrigatório um único número, podendo receber mais, que será a quantidade de números a serem gerados entre 0 e 1. Se você definir um número, a função gerará um vetor, se dois, dois vetores, três, três vetores e assim por diante.

Exemplo:

```
import numpy as np
vetor_1d = np.random.rand(2)
vetor_2d = np.random.rand(2,2)

print("Array com uma dimensão e dois números: ", vetor_1d)
print("Array com duas dimensões sendo cada uma delas com 2 números: ", vetor_2d)
```

Resposta:

```
>> Array com uma dimensão e dois números:  [0.21454504 0.68350515]
>> Array com duas dimensões sendo cada uma delas com 2 números:  [[0.02556629 0.77664973]
                                                                  [0.01126889 0.41078045]]
```

5.2. Função np.random.randint()

Essa função gera números inteiros do 0 até o número definido por você. Sendo:

low = Menor possível número do intervalo

high = Maior número possível do intervalo

size = Quantidade de números aleatórios a serem gerados em um vetor

Exemplo:

```
import numpy as np

vetor = np.random.randint(low=5, high=15, size=4)
print("Vetor de números inteiros: ", vetor)
```

Resposta:

```
>> Vetor de números inteiros:  [ 7 14 13 13]
```

5.3. Função np.random.uniform()

Essa função gera números de acordo com a distribuição uniforme. Que funciona da seguinte forma:

$$p(x) = \frac{1}{b-a}$$

low = Menor possível número do intervalo

high = Maior número possível do intervalo

size = Quantidade de números aleatórios a serem gerados em um vetor.

Exemplo:

```
import numpy as np

vetor = np.random.uniform(low=-10, high=3, size=4)
print("Vetor de números inteiros: ", vetor)
```

Resposta:

```
>> Vetor de números:  [-8.89268867 -9.95171943 -3.23764557  0.47642707]
```

6. Criando arrays com mais de uma dimensão

6.1. Função np.array([lista1, lista2, ...])

Essa função recebe como parâmetro obrigatório uma ou mais listas entre chaves. Dizemos que um array é bidimensional quando possui outros 2 arrays dentro dele e tridimensional quando possui 3 arrays dentro dele.

Exemplo:

```
import numpy as np
#criando arrays bi ou tridimensionais

lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista3 = [7, 8, 9]

array2d = np.array([lista1, lista2])
array3d = np.array([lista1, lista2, lista3])

print("Array bidimensional: ", array2d)
print("Array tridimensional: ", array3d)
```

Resposta:

```
>>  Array bidimensional:  [[1 2 3]
                           [4 5 6]]

>>  Array tridimensional:  [[1 2 3]
                            [4 5 6]
                            [7 8 9]]
```

7. Criando uma matriz

7.1. Função np.matriz([lista1, lista2, ...])

Essa função recebe como parâmetro obrigatório uma ou mais listas entre chaves e retorna uma matriz com números de linhas igual ao número de listas.

Exemplo:

```
import numpy as np
#crando matrizes

lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista3 = [7, 8, 9]

matriz = np.matrix([lista1, lista2, lista3])
print(f"Matriz com {len(matriz)} linhas: ", matriz)
```

Resposta:

```
>> Matriz com 3 linhas:  [[ 1  2  3]
                          [ 4  5  6]
                          [ 7  8  9]]
```

8. Diferença entre listas e arrays

Arrays são mais rápidos no processamento de dados do que as listas. Um dos motivos é o fato de a biblioteca NumPy ter partes construídas a partir da linguagem de programação “C”, que costuma ser uma linguagem mais rápida do que o “Python”

8.1. Soma de listas e arrays

Podemos observar essa diferença importante de formato entre as duas ao realizar a operação de soma. Enquanto as listas geram uma concatenação, os arrays irão gerar um vetor com a soma de seus itens.

Exemplo:

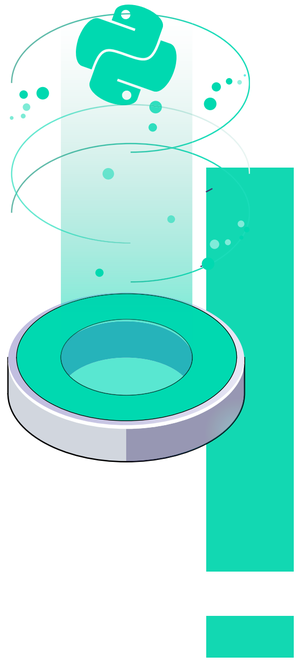
```
import numpy as np
#listas x numpy

lista_1 = [1, 1, 1, 1]
lista_2 = [1, 1, 1, 1]

print("Soma de listas: ", lista_1 + lista_2)
print("Soma de arrays: ",np.array(lista_1) + np.array(lista_2))
```

Resposta:

```
>> Soma de listas:  [1, 1, 1, 1, 1, 1, 1, 1]
>> Soma de arrays:  [2 2 2 2]
```



Mundo 3

1. Métodos e atributos

1.1 Métodos

São funções que mudam o estado de algum objeto. Por exemplo, set() , sort() e int() são alguns exemplos de métodos.

1.2. Atributos

Os atributos são as características dos objetos. Por exemplo, itemsize, ndim e nbytes são alguns desses atributos.

2. Atributos do NumPy

2.1. Atributo .nbytes

Um atributo dos arrays que dá como resposta o número de bytes do array.

2.2. Atributo .itemsize

Um atributo dos arrays que dá como resposta o número de bytes de cada item dentro do array.

Exemplo:

```
import numpy as np

array = np.arange(10)

print( "O array é: ",array)
print("Nº de bites do array é: ",array.nbytes)
print("Nº de bites de cada item dentro do array: ",array.itemsize)
```

Resposta:

```
>> O array é:  [0 1 2 3 4 5 6 7 8 9]
>> Nº de bites do array é:  80
>> Nº de bites de cada item dentro do array:  8
```

2.3. Atributo .ndim

Um atributo dos arrays que dá como resposta o número de dimensões de um array. Se existirem 2 arrays dentro de um array_total, diremos que esse array_total tem 2 dimensões.

2.4. Atributo .shape

Um atributo dos arrays que dá como resposta a formatação do array. Os números de retorno significam quantas linhas e colunas o vetor possui.

Exemplo:

```
import numpy as np

array2d = np.array([[1, 2, 3], [4, 5, 6]])

print("Número de dimensões do array: ",array2d.ndim)
print("Formato do array: ",array2d.shape)
```

Resposta:

```
>> Número de dimensões do array:  2
>> Formato do array:  (2, 3)
```

2.2. Atributo .size

Um atributo dos arrays que dá como resposta a quantidade de itens dentro do array.

Exemplo:

```
import numpy as np

array2d = np.array([[1, 2, 3], [4, 5, 6]])


print("O tamanho do array é: ",array2d.size)
print("A dimensão do array é: ", len(array2d))
```

Resposta:

```
>> O tamanho do array é:  6
>> A dimensão do array é:  2
```

Mundo 4

1. Selecionando itens de um array

Selecionar elementos de um array é igual a selecionar elementos de uma lista.

1.1. Selecionando um item de um array

Escolha a posição desejada e coloque entre as chaves, lembrando que a primeira posição é sempre “0” e a última é “-1”. Para escolher um intervalo, digite a posição inicial e a posição final entre dois pontos “:”, lembrando que a última posição do intervalo não está inclusa.

Exemplo:

```
import numpy as np

vetor = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
#posições
print("Primeira posição do vetor: ", vetor[0])
print("Selecionando um intervalo do vetor: ",vetor[0:2])
print("Última posição do vetor: ",vetor[-1])
```

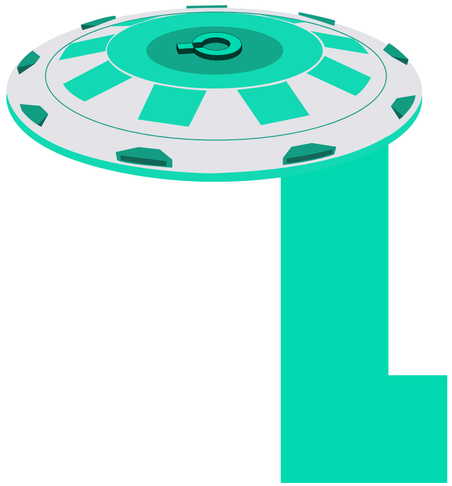
Resposta:

```
>> Primeira posição do vetor: 1
>> Selecionando um intervalo do vetor: [1 2]
>> Última posição do vetor: 9
```

2. Selecionando itens de um array multidimensional

2.1. Selecionando um item de um array multidimensional

Selecionar elementos de um array multidimensional é parecido com selecionar elementos de um outro array ou lista. Você escolherá primeiro a posição do array que deseja e depois escolherá a posição do item dentro do array escolhido. Esse modo é muito parecido com o método de escolher itens dentro de uma tabela no pandas.



Exemplo:

```
import numpy as np
vetor2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print("Escolhendo a primeira posição do terceiro vetor: ",vetor2d[2][0])
print("Escolhendo a segunda posição do primeiro vetor: ",vetor2d[0][1])
```

Resposta:

```
>> Escolhendo a primeira posição do terceiro vetor:  7
>> Escolhendo a segunda posição do primeiro vetor:  2
```

3. Filtrando dados no array

3.1. Filtrando números através de operações “>”, “<” e “==”

Para filtrar dados de um array, podemos usar opções de maior, menor e igualdade. Por exemplo, quando queremos números pares, escolhemos os números que tenham o resto zero na divisão inteira por 2. No Python, esse resto é representado por “%”.

Também vale a pena destacar quando queremos mais de uma condição. Para isso, utilizamos o "&" que tem significado de "e" ou "|" que tem significado de "ou". Nos exemplos abaixo, iremos exemplificar na prática essas operações.

Exemplo:

```
import numpy as np
#filtrando dados

vetor = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

maior_que_5 = vetor[vetor > 5] #maiores que 5
somente_par = vetor[vetor % 2 == 0] #números pares ou divisores inteiros por 2
igual_a_5 = vetor[vetor == 5] #números iguais a 5
maior_que_3_menor_que_8 = vetor[ (vetor > 5) & (vetor < 8)] #números entre 5 e 8

print("Itens do vetor que são maiores que 5: ", maior_que_5)
print("Itens do vetor que são pares: ",somente_par)
print("Itens do vetor que são iguais a 5: ", igual_a_5)
print("Itens do vetor que estão entre 3 e 8: ",maior_que_3_menor_que_8)
```

Resposta:

```
>> Itens do vetor que são maiores que 5:  [6 7 8 9]
>> Itens do vetor que são pares:  [2 4 6 8]
>> Itens do vetor que são iguais a 5:  [5]
>> Itens do vetor que estão entre 3 e 8:  [6 7]
```

Mundo 5

1. Manipulação de arrays

1.1. Função np.concatenate()

Essa função recebe como parâmetro os arrays que devem ser concatenados, ou seja, colocados juntos.

Exemplo:

```
import numpy as np

# juntar dois arrays
array1 = np.arange(20, 25)
array2 = np.arange(15)
array_completo = np.concatenate([array1, array2])

print("Primeiro array: ", array1)
print("Segundo array: ", array2)
print("Array final concatenado: ", array_completo)
```

Resposta:

```
>> Primeiro array:  [20 21 22 23 24]
>> Segundo array:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
>> Array final concatenado:  [20 21 22 23 24  0  1  2  3  4  5  6  7  8
9 10 11 12 13 14]
```

1.2. Função np.split()

Essa função recebe como parâmetro o array a ser separado e a posição onde ocorrerá a separação.

Exemplo:

```
import numpy as np
array_completo = np.arange(9)

#separando em dois arrays
array_separado1, array_separado2 = np.split(array_completo, [5])

print("Array completo: ",array_completo)
print("Array separado 1: ",array_separado1)
print("Array separado 2: ",array_separado2)
```

Resposta:

```
>> Array completo:  [1 2 3 4 5 6 7 8]
>> Array separado 1:  [1 2 3 4 5]
>> Array separado 2:  [6 7 8]
```

1.3. Função np.append()

Essa função recebe como parâmetro o array e o número a ser adicionado ao final.

1.4. Função np.insert()

Essa função recebe como parâmetro o array, a posição onde deseja inserir o número e o número a ser colocado.

Exemplo:

```
import numpy as np

array_completo = np.arange(10)
array_completo1 = np.append(array_completo, 9)
array_completo2 = np.insert(array_completo, 3,  3000)

print("O array completo 1: ", array_completo1)
print("O array completo 2: ", array_completo2)
```

Resposta:

```
>> O array completo 1:  [0 1 2 3 4 5 6 7 8 9 9]
>> O array completo 2:  [   0    1    2 3000    3    4    5    6    7    8    9]
```

1.5. Função np.delete()

Essa função recebe como parâmetro o array e a posição do item a ser excluído.

Exemplo:

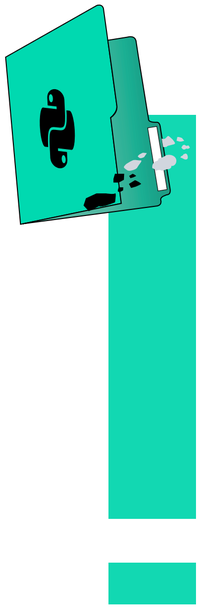
```
import numpy as np

array_completo = np.arange(10)

#deletar dados de um array
print("Array completo antes de deletar: ", array_completo)
array_completo = np.delete(array_completo, -1)
print("Array depois de deletar o último número: ",array_completo)
```

Resposta:

```
>> Array completo antes de deletar:  [0 1 2 3 4 5 6 7 8 9]
>> Array depois de deletar o último número:  [0 1 2 3 4 5 6 7 8]
```



1.6. Função np.reshape()

Essa função recebe como parâmetro o array e o formato que deseja. No caso abaixo, definimos uma matriz com 3 colunas e 3 linhas.

Exemplo:

```
import numpy as np

#reshape
array = np.arange(9)
print("Array com 9 posições: ",array)

array_3x3 = array.reshape(3, 3) #isso aqui é útil na hora de fazer regressão linear
print("Depois do reshape: ",array_3x3)
```

Resposta:

```
>> Array com 9 posições:  [0 1 2 3 4 5 6 7 8]
Depois do reshape:  [[0 1 2]
                     [3 4 5]
                     [6 7 8]]
```

1.7. Trocando números dentro de uma condição

Nesse caso utilizaremos condições para fazer uma troca. No caso abaixo, definimos que os itens do array_inicial menores que 40 serão trocados por zero.

Exemplo:

```
import numpy as np

#trocando números dentro de uma condição

array_inicial = np.array([4, 4, 4, 4, 7, 6, 5, 32, 13, 2, 32, 675, 74, 465])
#como trocar todos os numero menores que 40 por 0?
array_inicial[array_inicial < 40] = 0

print("Array após números serem trocados: ",array_inicial)
```

Resposta:

```
>> Array após números serem trocados:  [  0   0   0   0   0   0   0   0
0   0   0 675  74 465]
```

1.8. Ordenando objetos dentro do array

1.8.1. Função np.sort(array)

Recebe como parâmetro o array a ser ordenado e ordena por ordem crescente.

1.8.2. Função -np.sort(- array)

Recebe como parâmetro o array a ser ordenado e ordena por ordem decrescente.

Exemplo:

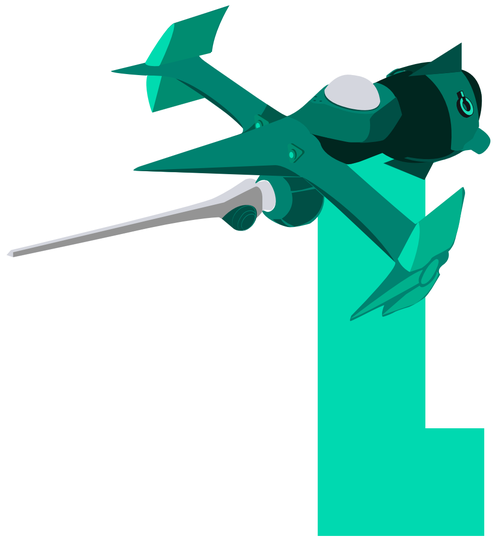
```
import numpy as np

#como ordernar objetos dentro de uma lista?
array_desordenado = np.array([23109, 2349, 349, 20, 3, 5, 12490])
array_crescente = np.sort(array_desordenado)
array_decrescente = -np.sort(-array_desordenado)

print("O array organizado em ordem crescente: ", array_crescente)
print("O array organizado em ordem decrescente: ", array_decrescente)
```

Resposta:

```
>> O array organizado em ordem crescente:  [    3     5    20   349  2349
12490 23109]
>> O array organizado em ordem decrescente:  [23109 12490  2349   349    20
5     3]
```



Mundo 6

1. Operações matemáticas entre arrays

1.1. Somando arrays

No mundo 2, vimos que quando nós somamos listas, o que ocorre na verdade é uma concatenação. Com os vetores é diferente. Ao somar dois vetores, ele somará posições coincidentes de vetores diferentes e, por isso, para que possa ocorrer a soma dos vetores, eles devem ter o mesmo tamanho.

Exemplo:

```python
import numpy as np

#todas as operações funcionam com os vetores.

array1 = np.array([1, 1, 1, 1])
array2 = np.array([4, 2, 6, 2])
array3 = np.array([2, 2])

lista_normal = [1, 1, 1, 1]
lista_normal2 = [4, 2, 6, 2]

print("Soma dos vetores: ", array1 + array2)
print("Soma das listas: ",lista_normal + lista_normal2)

try:
    soma_vetores_tamanhos_diferentes = array1 + array3
except ValueError as error:
    print("Não conseguiu somar, pois... ",error)
```

Resposta:

```
>> Soma dos vetores:  [5 3 7 3]
>> Soma das listas:  [1, 1, 1, 1, 4, 2, 6, 2]
>> Não conseguiu somar, pois...  operands could not be broadcast
together with shapes (4,) (2,)
```

1.2. Somando apenas um número ao array

Função type()

Essa função retornará o tipo da variável passada como parâmetro.

Exemplo:

```
import numpy as np

#E se eu quiser somar um número só?
soma_a_todos_os_elementos = np.array([1, 1, 1, 1]) + 20

print("Array com adição de 20: ",soma_a_todos_os_elementos)
```

Resposta:

```
>> Array com adição de 20:  [21 21 21 21]
```

1.3. Resto das operações matemáticas entre os vetores

Para fazer as demais operações matemáticas nos arrays é só seguir a sintaxe da matemática no python normalmente:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Divisão Inteira
**	Exponenciação

Exemplo:

```
import numpy as np

#todas as operações funcionam normal
array1 = np.array([1, 1, 1, 1])
array2 = np.array([4, 2, 6, 2])

adicao = array1 + array2
subtracao = array1 - array2
multiplicacao = array1 * array2
divisao = (array1/array2).round(2)
divisao_inteira = array1%array2
exponenciacao = array1 ** array2

print("Operação de adição: ",adicao)
print("Operação de subtração: ",subtracao)
print("Operação de multiplicação: ",multiplicacao)
print("Operação de divisão: ",divisao)
print("Operação de divisão inteira: ",divisao_inteira)
print("Operação de exponenciação: ",exponenciacao)
```

Resposta:

```
>> Operação de adição:  [5 3 7 3]
>> Operação de subtração:  [-3 -1 -5 -1]
>> Operação de multiplicação:  [4 2 6 2]
>> Operação de divisão:  [0.25 0.5  0.17 0.5 ]
>> Operação de divisão inteira:  [1 1 1 1]
>> Operação de exponenciação:  [1 1 1 1]
```

2. Operações matemáticas dentro dos arrays

2.1. Função .sum()

Essa função pode ser aplicada no vetor e retorna a soma dos elementos dentro do array.

2.2. Função .max()

Essa função pode ser aplicada no vetor e retorna o maior elemento do array.

2.3. Função .min()

Essa função pode ser aplicada no vetor e retorna o menor elemento do array.

2.4. Função np.abs()

Essa função recebe como parâmetro o próprio vetor e retorna o vetor com todos os números transformados em números absolutos, ou seja, positivos.

Exemplo:

```python
import numpy as np

#Operações dentro do próprio vetor

vetor = np.array([1, 4, -4, -1, 2])

print("Vetor inicial: ",vetor)
print("Soma dos itens do vetor: ",vetor.sum())
print("O maior número do vetor: ",vetor.max())
print("O menor número do vetor: ",vetor.min())
print("Números absolutos do vetor: ",np.abs(vetor))
```

Resposta:

```
>> Vetor inicial:  [ 1  4 -4 -1  2]
>> Soma dos itens do vetor:  2
>> O maior número do vetor:  4
>> O menor número do vetor:  -4
>> Números absolutos do vetor:  [1 4 4 1 2]
```

2.1. Função .mean()

Essa função pode ser aplicada no vetor e retorna a média dos elementos dentro do array.

2.2. Função .median()

Essa função pode ser aplicada no vetor e retorna a mediana dos elementos dentro do array.

2.3. Função .std()

Essa função pode ser aplicada no vetor e retorna o desvio padrão dos elementos dentro do array.

2.1 Função var()

Essa função não recebe nenhum parâmetro e retorna a variância dos elementos dentro do array.

Exemplo:

```
import numpy as np

#Estatisticas descritivas
vetor = np.array([1, 20, -4, -12, 2000])

media = np.mean(vetor)
mediana = np.median(vetor)
desvio_p = np.std(vetor)
var = np.var(vetor)

print("A média dos números do vetor é: ",media) #media
print("A mediana dos números do vetor é: ",mediana) #mediana
print("O desvio padrão dos números do vetor é: ",desvio_p) #desvio padrao
print("A variância dos números do vetor é: ",var) #variancia
```

Resposta:

```
>> A média dos números do vetor é:  401.0
>> A mediana dos números do vetor é:  1.0
>> O desvio padrão dos números do vetor é:  799.56938410622
>> A variância dos números do vetor é:  639311.2
```

2.1 Função np.corrcoef()

Essa função recebe como parâmetro os vetores que desejam fazer a correlação e retorna uma matriz quadrada de correlação.

Exemplo:

```
import numpy as np

#matriz de correlação
vetor = np.array([1, 20, -4, -12, 2000])
vetor2 = np.array([5, 4, 3, -5, 4])
vetor3 = np.array([2, 6, 7, -5, 1])

correlacao = np.corrcoef((vetor, vetor2, vetor3))
print("A matriz correlação entre os 3 vetores é", correlacao.round(2))
```

Resposta:

```
>> A matriz correlação entre os 3 vetores é: [ 1.      0.25   -0.13]
                                             [ 0.25    1.     0.77]
                                             [-0.13    0.77   1.  ]]
```