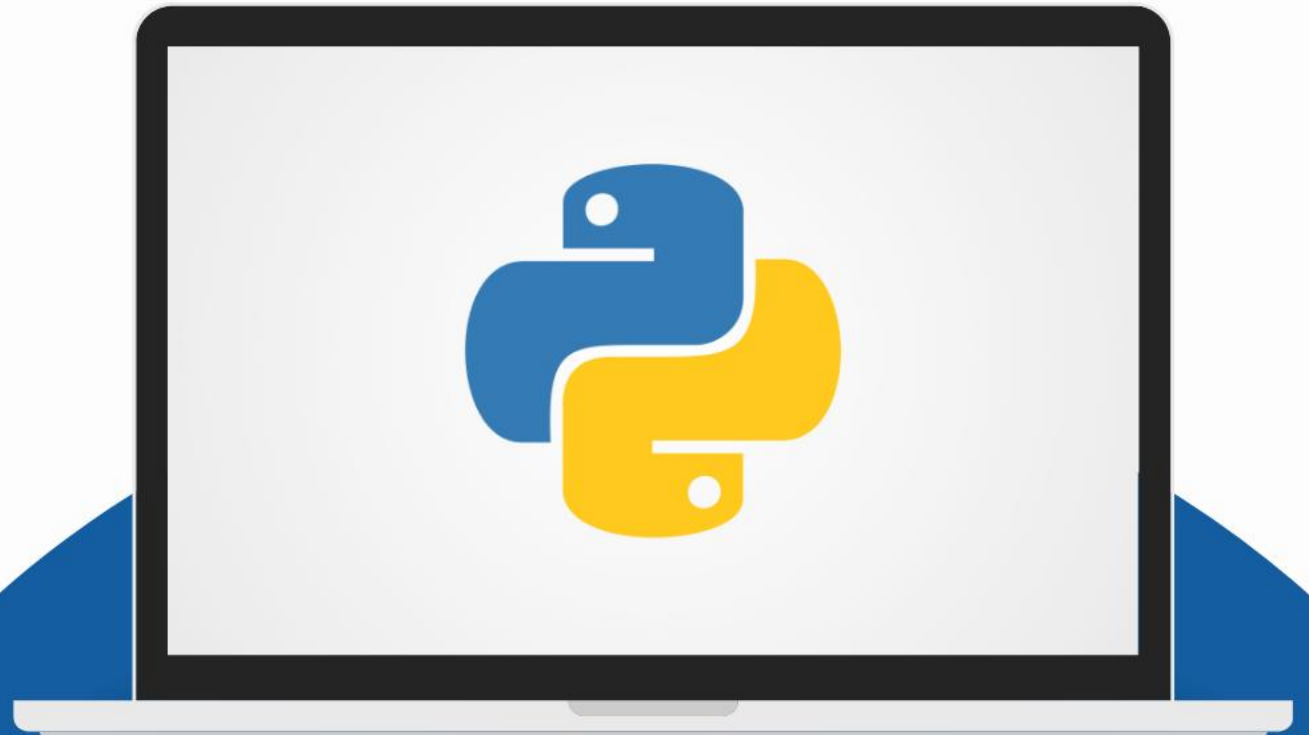


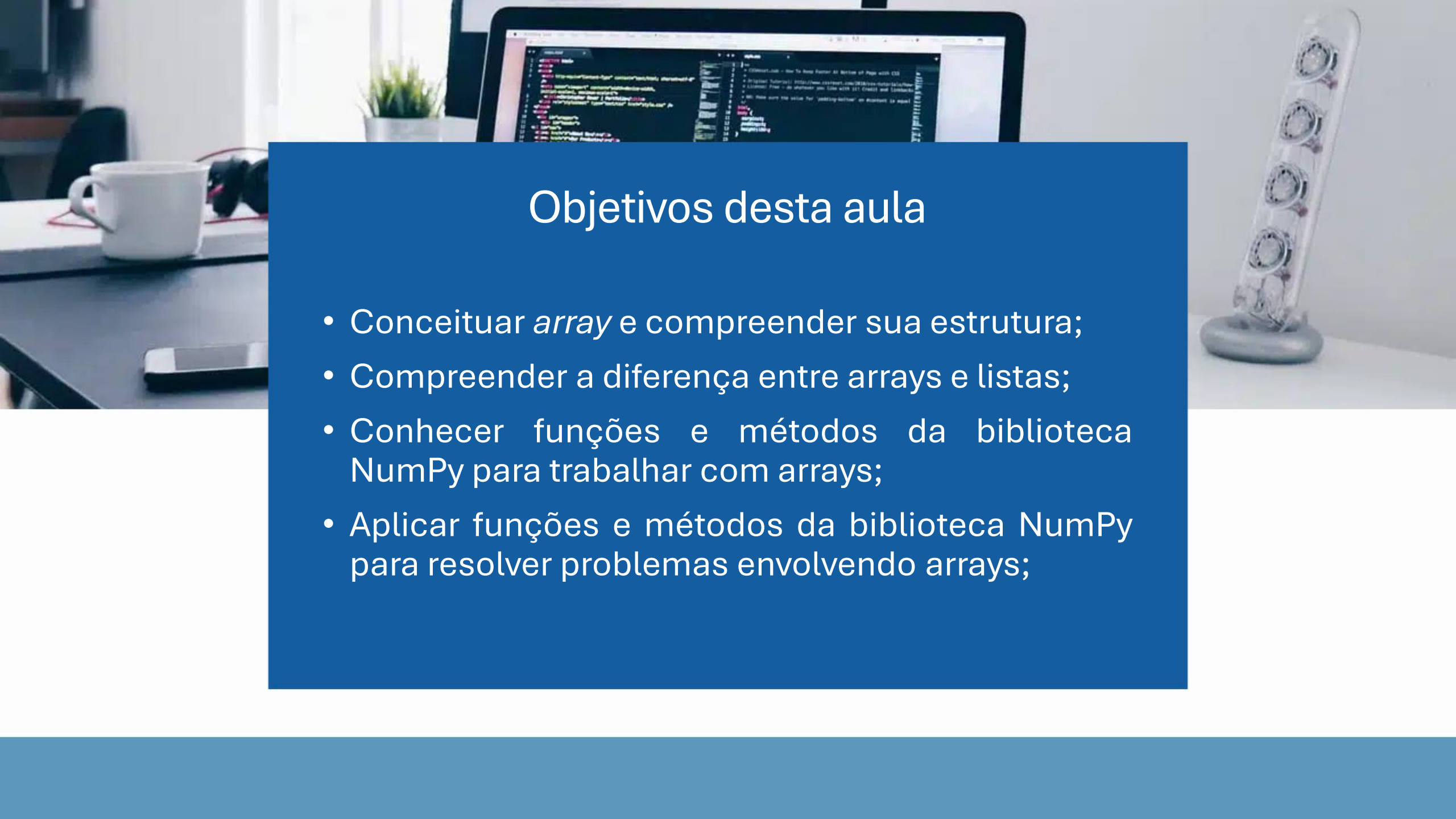


Aula #9:

Arrays: funções e métodos

Prof. Dr. Luiz Álvaro de Oliveira Júnior





Objetivos desta aula

- Conceituar *array* e compreender sua estrutura;
- Compreender a diferença entre arrays e listas;
- Conhecer funções e métodos da biblioteca NumPy para trabalhar com arrays;
- Aplicar funções e métodos da biblioteca NumPy para resolver problemas envolvendo arrays;

Sumário

▶ Arrays	04
▶ Métodos para arrays	06
▶ Funções para arrays	14
▶ Exercícios	36



Arrays

Um ***array*** é uma estrutura de dados da biblioteca NumPy, capaz de armazenar diferentes valores em si mesma, permitindo a execução de cálculos simples e complexos.

Se assemelham às listas, mas **armazenam elementos de um único tipo e oferecem operações matemáticas mais eficientes** que elas, pois as operações valem para todos os elementos do array.

Arrays

Usamos arrays (do NumPy) **se precisamos** trabalhar com números e **realizar operações matemáticas** simples ou complexas **de forma eficiente**.

Assim como as listas, os **arrays são** estruturas de dados **iteráveis**, ou seja, podem ter seus elementos percorridos por alguma estrutura de repetição.

Os arrays são menos versáteis que as listas, que aceitam dados de tipos diferentes, mas são superiores a elas no que se refere à performance numérica.

Métodos para arrays

Os principais métodos para arrays são mostrados a seguir:

concatenate(): une dois ou mais arrays em um só.

Sintaxe: **concatenate** ((**array1**, **array2**), **axis**)

Exemplo:

```
import numpy as np
a = np.array([1, 2]) #1D
b = np.array([3, 4]) #1D
c = np.concatenate((a, b))
print(c)
```

Saída

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

Observação: omitindo axis, os arrays 1D (linha única) terão os elementos unidos em sequência.

Métodos para arrays

Podemos usar o parâmetro `axis` para definir como a concatenação ocorrerá.

Se **axis = 0**, empilha-se ao longo das linhas (verticalmente).

```
a = np.array([[1, 2], [3, 4]])  
b = np.array([[5, 6], [7, 8]])  
c = np.concatenate((a, b), axis=0)  
print(c)
```

Saída

```
[[1 2]  
 [3 4]  
 [5 6]  
 [7 8]]
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Para concatenar arrays 2D com `axis = 0`, o número de colunas deve ser igual.

Métodos para arrays

Se **axis = 1**, empilha-se ao longo das colunas (horizontalmente).

```
a = np.array([[1, 2], [3, 4]])  
b = np.array([[5, 6], [7, 8]])  
c = np.concatenate((a, b), axis=1)  
print(c)
```

Saída

```
[[1 2 5 6]  
 [3 4 7 8]]
```

$$\begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{bmatrix}$$

Para concatenar arrays 2D com `axis = 1`, o número de linhas deve ser igual.

Métodos para arrays

reshape(): muda a forma do array preservando os elementos.

Sintaxe: **reshape(array, forma)**

Exemplo:

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

Saída

```
[1 2 3 4 5 6]

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

Observação: A forma do novo array é definida pelo número de linhas e de colunas, entre parênteses. No exemplo, (3,2) representa 3 linhas e 2 colunas.

Métodos para arrays

resize(): muda a forma do array preenchendo posições vazias.

Sintaxe: **resize(array, forma)**

Exemplo:

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

Saída

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

Observação: Se o array original tem menos elementos, o método `resize` repete os números nas posições vazias até completar.

Métodos para arrays

flatten(): achata o array tornando-o unidimensional (linha única).

Sintaxe: **flatten()**

Exemplo:

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

Saída

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

Métodos para arrays

split(): divide o array em partes menores.

Sintaxe: **split(array, partes)**

Exemplo:



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



Saída

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

Métodos para arrays

transpose(): transpõe o array (troca linhas por colunas).

Sintaxe: **transpose()** ou mais comumente **T**

Exemplo:

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

Observação: print(a.T) também funciona!

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

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

Funções para arrays

As principais funções para arrays são mostradas a seguir:

array(): cria um array a partir de uma lista de elementos informados.

Sintaxe: **np.array(lista)**

Exemplo:

```
import numpy as np
a = np.array([1, 2])
print(a)
```

Saída

```
[1 2]
```

Observação: os elementos são colocados entre [] e separados por vírgulas. Para arrays 2D, os elementos de uma mesma linha ficam entre [].

Funções para arrays

empty(): cria um array vazios com forma definida.

Sintaxe: **empty(forma)**

Exemplo:

```
import numpy as np
a = np.empty([2,3])
print(a)
```

Saída

```
[[3.834e-315  0.000e+000  6.859e-310]
 [3.834e-315  5.838e-317  6.859e-310]]
```

Observação: a função **reserva espaço na memória**, mas não inicializa as posições com zeros. O conteúdo inicial será **aleatório** ou **residual** a depender da memória.

Funções para arrays

zeros(): cria um array preenchido com zeros.

Sintaxe: **zeros(forma)**

Exemplo:

```
import numpy as np
a = np.zeros([2,3])
print(a)
```

Saída

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

Observação: para 1D, informe o número de elementos, para 2D, informe os números de linhas e de colunas, para 3D, informe três inteiros)

Funções para arrays

ones(): cria um array preenchido com 1.

Sintaxe: **ones(forma)**

Exemplo:

```
import numpy as np
a = np.ones([2,2])
print(a)
```

Saída

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

Observação: para 1D, informe o número de elementos, para 2D, informe os números de linhas e de colunas, para 3D, informe três inteiros)

Funções para arrays

eye(): cria uma matriz quadrada com 1 na diagonal principal.

Sintaxe: **eye(ordem)**

Exemplo:

```
import numpy as np
a = np.eye(2)
print(a)
```

Saída

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

Observação: ordem se refere à ordem da matriz. É um número inteiro.

Funções para arrays

arange(): gera uma sequência numérica com valores espaçados.

Sintaxe: **arange(início, fim, passo)**

Exemplo:

```
import numpy as np
a = np.arange(0,10,2)
print(a)
```

Saída

```
[0 2 4 6 8]
```

Observação: início, fim e passo são números inteiros. Início é includente, fim é excludente.

Funções para arrays

sum(): soma os elementos do array.

Sintaxe: **sum(array)**

Exemplo:

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([[2, 4], [6, 8]])
somaA = np.sum(a)
print("somaA =", somaA)
print("somaB =", np.sum(b))
```

● ● ● Saída

```
somaA = 10
somaB = 20
```

Funções para arrays

max(): retorna o maior valor de um array.

Sintaxe: **max(array)**

Exemplo:

```
import numpy as np
a = np.array([[10, 2],[-3, 104]])
maior = np.max(a)
print("maior =", maior)
```

Saída

```
maior = 104
```

Funções para arrays

min(): retorna o menor valor de um array.

Sintaxe: **min(array)**

Exemplo:

```
import numpy as np
a = np.array([[10, 2], [-3, 104]])
menor = np.min(a)
print("menor =", menor)
```

Saída

```
menor = -3
```

Funções para arrays

mean(): retorna a média de um array.

Sintaxe: **mean(array)**

Exemplo:

```
import numpy as np
a = np.array([10, 20, 30, 40])
media = np.mean(a)
print("media =", media)
```

● ● ● Saída

```
media = 25
```

Funções para arrays

intersect1d(): retorna os termos comuns entre dois arrays.

Sintaxe: **intersect1d(array)**

Exemplo:

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([[3, 4, 5], [6, 8, 2]])
comum = np.intersect1d(a, b)
print("comum = ", comum)
```

Saída

```
comum = [2 3 4]
```

Observação: ordena automaticamente, exige tipos compatíveis mas não a mesma dimensão e ignora duplicatas.

Funções para arrays

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([[0.5, 3.0, 5.0],[6.0, 8.0, 2.0]])
c = np.array(["0.5", "3", "5", "0"])
d = np.array(["dois","três","quatro"])
comum_ab = np.intersect1d(a, b)
comum_ac = np.intersect1d(a, c)
comum_ad = np.intersect1d(a, d)
print("comum ab = ", comum_ab)
print("comum ac = ", comum_ac)
print("comum ad = ", comum_ad)
```

Saída

```
comum ab = [2.0 3.0]
comum ac = ['3']
comum ad = [ ]
```

Observação: os arrays a, b, c e d são de tipos diferentes, mas a, b e c são compatíveis entre si. Por outro lado d é incompatível com a, b e c.

Funções para arrays

sort(): retorna uma cópia ordenada do array.

Sintaxe: **sort(array)**

Exemplo:

```
import numpy as np
a = np.array([10, 2, -3, 104])
b = np.sort(a)
print("a =", a)
print("b =", b)
```

● ● ● Saída

```
a = [10, 2, -3, 104]
b = [-3, 2, 10, 104]
```

Funções para arrays

column_stack(): empilha arrays lado a lado (em colunas).

Sintaxe: **column_stack**((**array_1**, **array_2**, ..., **array_n**))

Exemplo:

```
import numpy as np
```

```
Num_A = np.array([7.5, 8.0, 6.2, 7.8])
```

```
Num_B = np.array([5.5, 9.0, 4.3, 9.1])
```

```
array = np.column_stack((Num_A, Num_B))  
print(array)
```

Saída

```
[[7.5 5.5]  
 [8.0 9.0]  
 [6.2 4.3]  
 [7.8 9.1]]
```

Funções para arrays

round(): arredonda os termos do array por um número especificado de casas decimais.

Sintaxe: **round(array, casas)**

Exemplo:

```
import numpy as np
a = np.array([3.1415, 2.7182])
print(np.round(a, 2))
```

● ● ● Saída

```
a = [3.14 2.72]
```

Observação: a função round segue as regras de arredondamento.

Funções para arrays

floor(): arredonda para baixo (menor inteiro mais próximo).

Sintaxe: **floor(array)**

Exemplo:

```
import numpy as np
a = np.array([3.1415, 2.7182])
print(np.floor(a))
```

Saída

```
a = [3.0 2.0]
```

Funções para arrays

ceil(): arredonda para cima (maior inteiro mais próximo).

Sintaxe: **ceil(array)**

Exemplo:



```
import numpy as np
a = np.array([3.1415, 2.7182])
print(np.ceil(a))
```



Saída

```
a = [4.0 3.0]
```

Funções para arrays

trunc(): remove as casas decimais sem arredondar (truncamento).

Sintaxe: **trunc(array)**

Exemplo:

```
import numpy as np
a = np.array([3.1415, 2.7182])
print(np.trunc(a))
```

Saída

```
a = [3.0 2.0]
```

Funções para arrays

std(): calcula o desvio padrão populacional de um array.

Sintaxe: **std(array)**

Exemplo:

```
import numpy as np
notas = np.array([7.5, 8.0, 9.0, 6.5, 8.5])
desvio = np.std(notas)
print("Desvio:", np.round(desvio,2))
```

Saída

```
desvio = 0.86
```

Observação: para calcular o desvio padrão amostral, coloque uma vírgula após o array e informe ddof = 1 como parâmetro.

Funções para arrays

A biblioteca NumPy tem ainda um módulo chamado random que permite a geração de números aleatórios. Nesse módulo há duas funções bastante úteis: random e randint.

random(): gera números aleatórios entre 0 e 1.

Sintaxe: **random(quantidade)**

```
import numpy as np
a = np.random.random(2)
print("a =", a)
```

Saída

```
a = [0.010214856 0.614894097]
```

Funções para arrays

randint(): gera números aleatórios em um intervalo especificado.

Sintaxe: **randint(início, fim, quantidade)**

Exemplo:

```
import numpy as np
a = np.random.randint(1,20,4)
print("a =", a)
```

Saída

```
a = [15  8  6 12]
```

Observação: início, fim e quantidade são números inteiros. Início é inclusivo, fim é exclusivo e se não informar quantidade, apenas um número aleatório será gerado no intervalo especificado.

Funções para arrays

uniform(): gera números aleatórios decimais em um intervalo.

Sintaxe: **uniform**(**início**, **fim**, **quantidade**)

Exemplo:

```
import numpy as np
a = np.random.uniform(0,10,4)
print("a =", a)
```

Saída

```
a = [6.21 9.48 7.73 8.25]
```

Observação: início, fim e quantidade são números inteiros. Início é inclusivo, fim é exclusivo e se não informar quantidade, apenas um número aleatório será gerado no intervalo especificado.

Exercícios

Exercício #1

Escreva um programa em Python que solicite ao professor as notas das turmas A01/1 e A01/2 de Programação Estruturada. Em seguida faça:

- a) Junte as informações das duas turmas em um único array;
- b) Ordene as notas;
- c) Organize o array em duas colunas, de forma que as notas da turma A01/1 estejam na primeira coluna e as notas da turma A01/2 estejam na segunda coluna.

```
import numpy as np

dados = int(input("Digite o número de alunos: "))
notas1 = np.empty(dados)
notas2 = np.empty(dados)

print("Digite as notas da turma A01/1")
for i in range(dados):
    notas1[i] = float(input(f"Digite a nota do aluno {i+1}: "))

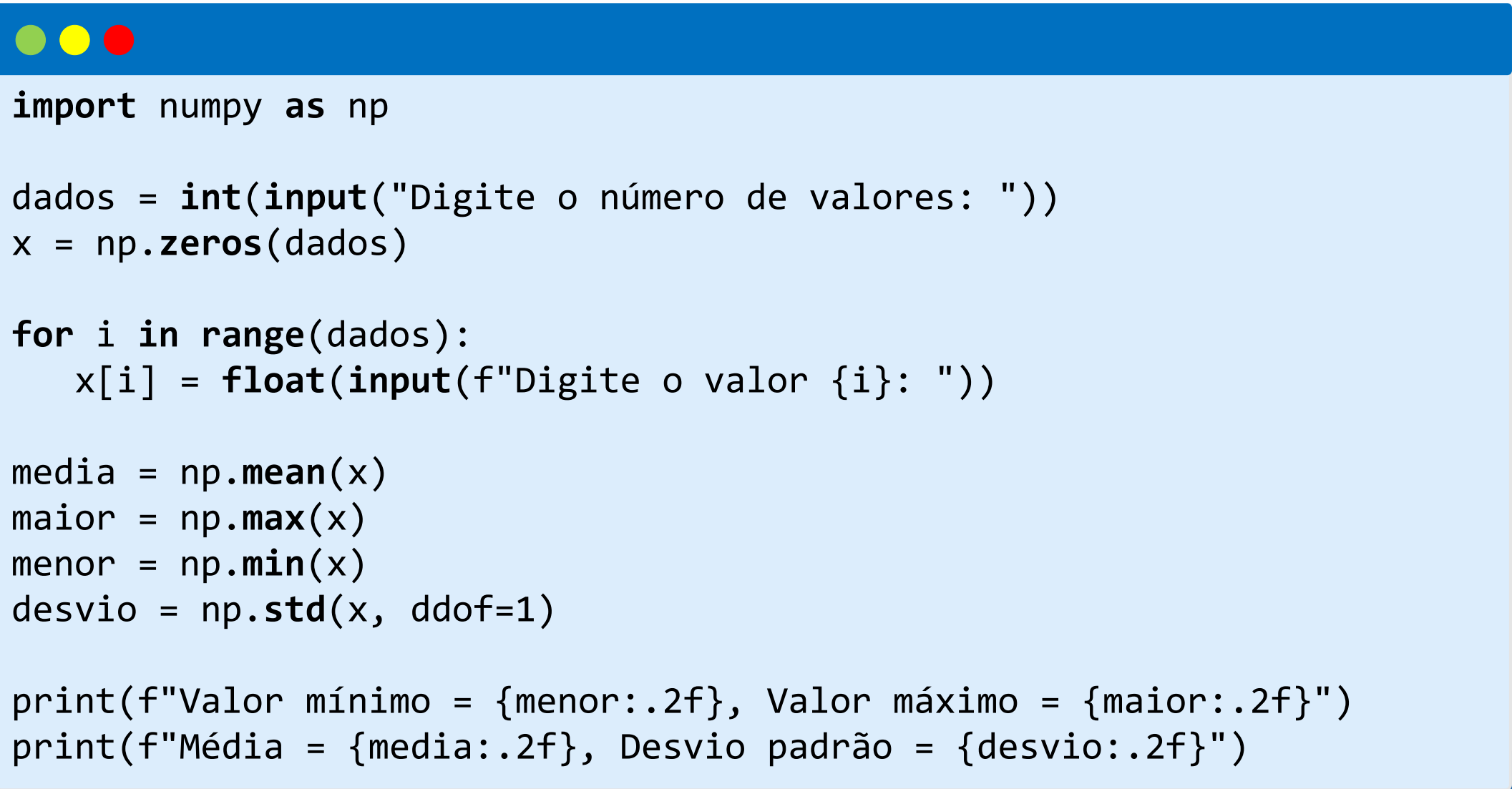
print("Digite as notas da turma A01/2")
for i in range(dados):
    notas2[i] = float(input(f"Digite a nota do aluno {i+1}: "))

notasA01 = np.concatenate((notas1,notas2))           #Letra (a)
print("Notas A01: ",notasA01)
ranking = np.sort(notasA01)                           #Letra (b)
print("Ranking: ",ranking)
tabela_notas = np.column_stack((notas1,notas2))       #Letra (c)
print(tabela_notas)
```

Exercício #2

Escreva um programa em Python que calcule a média, o maior valor, o menor valor e o desvio padrão dos dados obtidos em um experimento realizado no laboratório. O programa deve:

- Solicitar que o usuário informe os dados individualmente;
- Calcular o maior valor e o menor valor da amostra;
- Calcular a média aritmética;
- Calcular o desvio padrão amostral;
- Utilizar funções da biblioteca numpy para resolver o problema.



```
import numpy as np

dados = int(input("Digite o número de valores: "))
x = np.zeros(dados)

for i in range(dados):
    x[i] = float(input(f"Digite o valor {i}: "))

media = np.mean(x)
maior = np.max(x)
menor = np.min(x)
desvio = np.std(x, ddof=1)

print(f"Valor mínimo = {menor:.2f}, Valor máximo = {maior:.2f}")
print(f"Média = {media:.2f}, Desvio padrão = {desvio:.2f}")
```