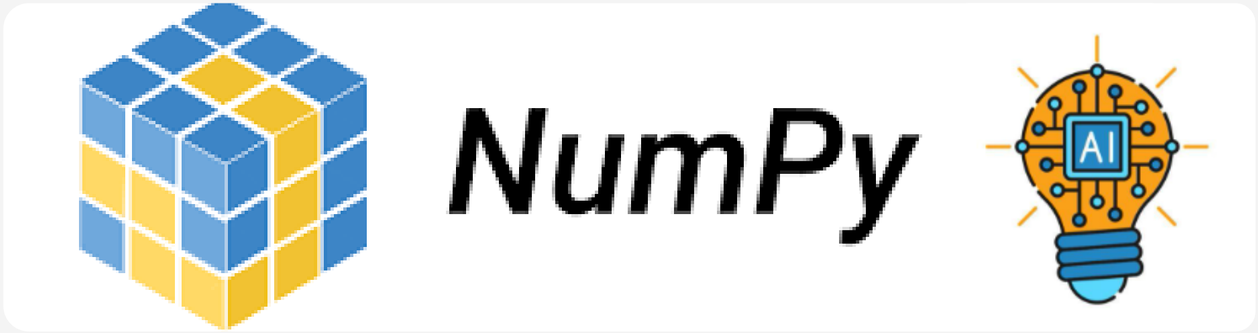


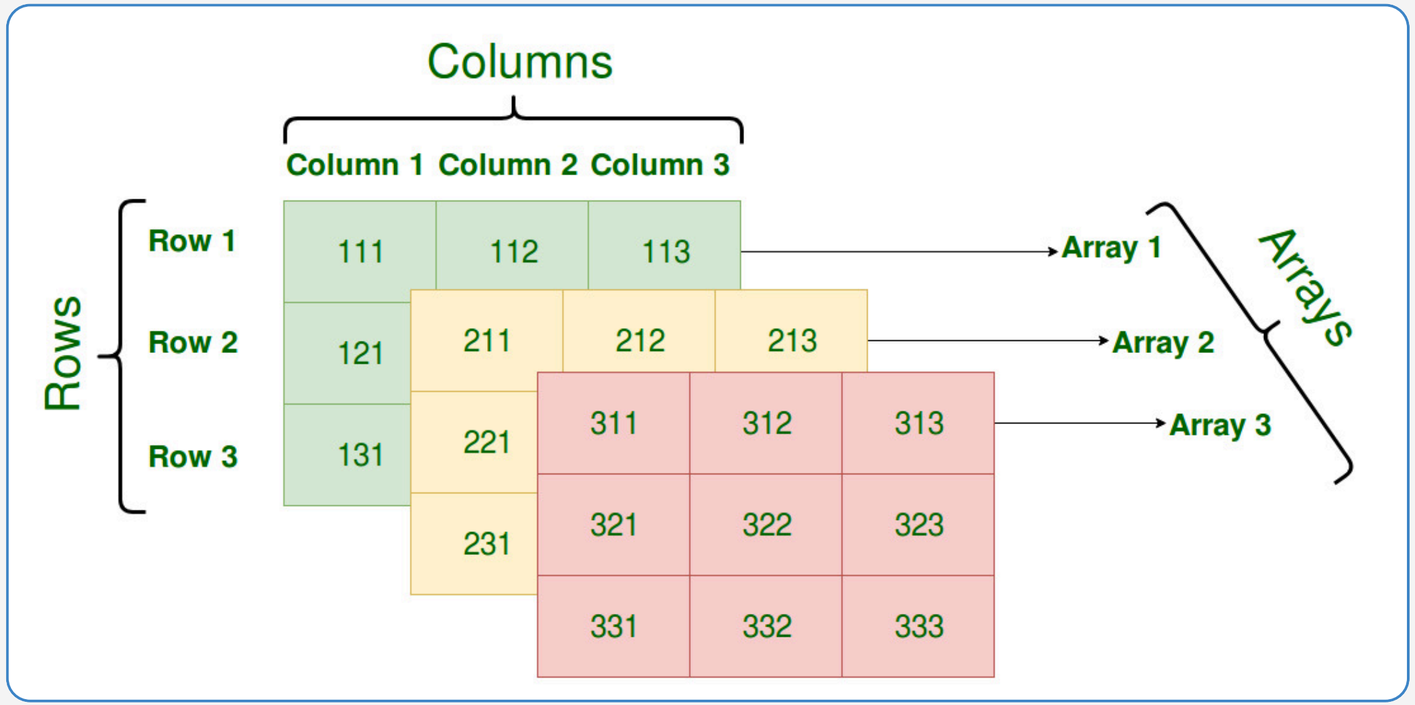
FUNDAMENTOS DA COMPUTAÇÃO NUMÉRICA EM PYTHON

UNIPAC

NumPy

Biblioteca essencial em Python para computação numérica e manipulação de **arrays** multidimensionais.

Ela oferece funcionalidades poderosas para operações matemáticas complexas com **arrays** e **matrizes**, facilitando o trabalho com dados numéricos em larga escala.



Por que usar NumPy?

1. **Eficiência:** NumPy é construído sobre C, o que o torna significativamente mais rápido do que operações similares implementadas em Python puro.
2. **Arrays Multidimensionais:** NumPy introduz o conceito de arrays multidimensionais (ndarrays), que são mais eficientes para armazenar e manipular dados do que as estruturas de dados padrão do Python.
3. **Ampla Gama de Funções:** NumPy oferece uma vasta coleção de funções matemáticas que operam eficientemente em arrays, desde operações básicas até funções avançadas de álgebra linear e transformações de Fourier.

Por que usar NumPy?

1. **Eficiência:** NumPy é construído sobre C, o que o torna significativamente mais rápido do que operações similares implementadas em Python puro.
2. **Arrays Multidimensionais:** NumPy introduz o conceito de arrays multidimensionais (ndarrays), que são mais eficientes para armazenar e manipular dados do que as estruturas de dados padrão do Python.
3. **Ampla Gama de Funções:** NumPy oferece uma vasta coleção de funções matemáticas que operam eficientemente em arrays, desde operações básicas até funções avançadas de álgebra linear e transformações de Fourier.

Criando Arrays em NumPy

A maneira mais básica de criar um array em NumPy é convertendo uma lista ou tupla existente usando **np.array():**

```python
import numpy as np


# Criando um array a partir de uma lista
arr = np.array([1, 2, 3, 4, 5])
print(arr)  # Output: [1 2 3 4 5]
```

```python
# Criando um array 2D (matriz)
matriz = np.array([[1, 2, 3], [4, 5, 6]])
print(matriz)
# Output:
# [[1 2 3]
#  [4 5 6]]
```

UNIPAC

- NumPy facilita operações aritméticas e estatísticas em arrays:

```python
import numpy as np

arr = np.array([1, 2, 3])

# Operações básicas
print(arr + 1)    # Soma escalar: [2 3 4]
print(arr * 2)    # Multiplicação escalar: [2 4 6]
print(np.sqrt(arr))  # Raiz quadrada: [1.  1.41421356 1.73205081]

# Estatísticas simples
print(np.mean(arr))   # Média: 2.0
print(np.std(arr))    # Desvio padrão: 0.816496580927726
```


- Inicialize um array de zeros com **np.zeros**. O comando np.zeros((5,2)) cria um array 5 x 2 de zeros:

```
np.zeros((5,2))

array([[ 0., 0.],
       [ 0., 0.],
       [ 0., 0.],
       [ 0., 0.],
       [ 0., 0.]])
```

```
array = np.ones((3, 3), dtype=np.int)
print(array)

array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

- **Indexação:** Manipulando Dados em Arrays

```python
import numpy as np


arr = np.array([10, 20, 30, 40, 50])
print(arr[0])  # Acessa o primeiro elemento: 10
print(arr[3])  # Acessa o quarto elemento: 40
```

```python
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matriz[0, 1])  # Acessa o elemento na linha 0, coluna 1: 2
print(matriz[2, 2])  # Acessa o elemento na linha 2, coluna 2: 9
```


- **Fatiamento (Slicing):** Fatiamento permite acessar subarrays em NumPy, semelhante ao fatiamento de listas em Python. Você pode especificar intervalos de índices para selecionar múltiplos elementos:

```
arr = np.array([10, 20, 30, 40, 50])

# Seleciona os elementos do índice 1 até o 3 (não inclusivo)
sub_array = arr[1:4]
print(sub_array)  # Output: [20 30 40]
```

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

# Seleciona as duas primeiras linhas e as duas primeiras colunas
sub_matriz = matriz[:2, :2]
print(sub_matriz)
# Output:
# [[1 2]
#  [4 5]]
```

- **Indexação Booleana:** A indexação booleana permite selecionar elementos de um array que atendem a uma determinada condição. É uma ferramenta poderosa para filtrar dados.

```
arr = np.array([10, 20, 30, 40, 50])


# Seleciona elementos maiores que 25
condicao = arr > 25
print(condicao)  # Output: [False False  True  True  True]


# Aplica a condição para filtrar o array
filtrado = arr[condicao]
print(filtrado)  # Output: [30 40 50]
```

- **Indexação Avançada:** Também suporta indexação avançada, que permite selecionar elementos usando arrays de índices ou máscaras booleanas complexas. Isso é útil para operações que exigem maior controle sobre os dados.

```
arr = np.array([10, 20, 30, 40, 50])


# Seleciona elementos nos índices 0, 2 e 4
indices = [0, 2, 4]
print(arr[indices])  # Output: [10 30 50]
```

- **Modificando Arrays com Indexação:** Você pode usar a indexação para modificar elementos específicos em um array. Isso pode ser feito tanto com indexação básica quanto com indexação avançada.

```
arr = np.array([10, 20, 30, 40, 50])

# Modifica o elemento no índice 1
arr[1] = 25
print(arr)  # Output: [10 25 30 40 50]

# Modifica múltiplos elementos usando fatiamento
arr[2:4] = [35, 45]
print(arr)  # Output: [10 25 35 45 50]
```

- **Modificando Arrays com Indexação:** Você pode usar a indexação para modificar elementos específicos em um array. Isso pode ser feito tanto com indexação básica quanto com indexação avançada.

```
arr = np.array([10, 20, 30, 40, 50])

# Modifica o elemento no índice 1
arr[1] = 25
print(arr)  # Output: [10 25 30 40 50]

# Modifica múltiplos elementos usando fatiamento
arr[2:4] = [35, 45]
print(arr)  # Output: [10 25 35 45 50]
```

Carregando Dados de um CSV

	Unnamed: 0	1.2013	2.2013	3.2013	4.2013	5.2013	6.2013	7.2013	8.2013	9.2013	...	6.2019	7.2019	8.2019	9.2019	10.2019	11.2019	12.2019	1.2020
0	Moscow	79.72	81.08	79.68	79.80	80.63	80.80	80.28	78.99	76.77	...	116.91	125.29	123.94	113.03	102.19	97.83	101.07	103.44
1	Kaliningrad	42.67	44.37	44.73	46.75	NaN	51.59	57.80	62.14	56.76	...	79.20	80.85	85.33	75.02	77.95	78.98	76.55	74.89
2	Petersburg	62.55	62.73	63.43	63.83	66.06	69.22	72.07	69.31	65.18	...	115.35	123.03	123.08	109.71	97.22	95.75	97.09	98.18
3	Krasnodar	48.26	51.01	50.91	53.94	61.27	65.44	56.51	53.00	43.87	...	102.01	116.12	92.06	82.70	66.62	68.11	73.48	82.04
4	Ekaterinburg	71.25	71.35	70.90	71.92	72.91	74.39	73.10	70.24	69.12	...	121.68	125.32	123.41	108.48	98.73	96.25	100.12	101.29

5 rows × 88 columns

Carregando Dados de um CSV

A expressão:

np.loadtxt('macas.csv', delimiter=",", usecols=np.arange(1, 88, 1))

está sendo usada para carregar dados de um arquivo CSV chamado **macas.csv**.

Vamos analisar os componentes:

- **'macas.csv':** nome do arquivo a ser carregado.
- **delimiter=",":** Especifica que o delimitador do arquivo CSV é a vírgula**.**
- **usecols=np.arange(1, 88, 1):**
- Especifica quais colunas do arquivo CSV serão carregadas.
- **np.arange(1, 88, 1)** cria um array do NumPy com valores de 1 até 87 (inclusive), incrementando de 1 em 1.