# Introdução à Biblioteca NumPy

## Os Pilares da Computação Científica em Python

https://numpy.org/

# Agenda

### Parte 1: Introdução e o `ndarray`

- O que é NumPy, por que usá-lo e seu papel no ecossistema Python.
- A anatomia do `ndarray`: eixos, forma e atributos essenciais.

### Parte 2: Criação e Manipulação de Arrays

- Funções para criar e inicializar arrays.
- Técnicas para remodelar, combinar e dividir arrays.

### Parte 3: Indexação, Operações e Broadcasting

- Acessando dados com indexação básica, booleana e avançada.
- O paradigma `View` vs. `Copy`, operações matemáticas e estatísticas.
- ***Broadcasting***.

# Parte 1

# Introdução e o `ndarray`

## O que é NumPy?

**NumPy** (Numerical Python) é a biblioteca fundamental para computação científica em Python.

- Seu principal objeto é o `ndarray` (array N-dimensional), uma grade de valores do **mesmo tipo** (homogêneo).
- É a base para análise de dados, machine learning e engenharia.

## Por que NumPy? A Vantagem da Vetorização

- **Performance:** Operações numéricas em grandes volumes de dados são ordens de magnitude mais rápidas que com listas Python.
- **Vetorização:** Permite que operações sejam executadas em arrays inteiros de uma só vez, sem a necessidade de laços `for` em Python.
- **Código Limpo:** O código vetorizado é mais conciso, legível e se assemelha à notação matemática padrão.

As operações são, na verdade, implementadas em código C ou Fortran, o que explica o ganho massivo de velocidade.

## Exemplo: Listas Python vs. Arrays NumPy

Calculando o produto escalar de dois vetores com 1.000.000 de elementos.

```
import numpy as np

# Com listas
def produto_interno_lista(u, v):
  prod = 0
  for i in range(len(u)):
  prod += u[i] * v[i]
  return prod

array_u = np.arange(1000000)
array_v = np.arange(1000000)
np.dot(array_u, array_v)
```

## O Ecossistema Científico Python

NumPy é a infraestrutura sobre a qual quase todo o ecossistema de ciência de dados em Python foi construído.

- **Pandas:** Usa arrays NumPy para suas estruturas `Series` e `DataFrame`.
- **SciPy:** Expande o NumPy com algoritmos científicos avançados.
- **Matplotlib:** Aceita arrays NumPy diretamente para criar gráficos.
- **Scikit-learn:** Utiliza arrays NumPy como o formato de entrada universal para dados.

O `ndarray` funciona como a **lingua franca** para a troca de dados numéricos entre essas bibliotecas.

## Tabela Comparativa: Listas vs. Arrays

| Característica | Listas Python | Arrays NumPy (`ndarray`) |
|---|---|---|
| **Tipo de Dados** | Heterogêneo | Homogêneo (todos do mesmo tipo) |
| **Performance** | Lenta para operações numéricas | Rápida devido à vetorização |
| **Uso de Memória** | Maior | Menor e mais compacto |
| **Funcionalidade** | Genérica | Vasta gama de funções matemáticas |
| **Casos de Uso** | Dados heterogêneos | Computação científica, machine learning |

## Anatomia de um `ndarray`

É uma grade multidimensional de elementos. As dimensões são chamadas de **eixos** ( `axes` ).

- **Array 1D:** 1 eixo (vetor).
- **Array 2D:** 2 eixos (matriz).
  - Eixo 0: linhas
  - Eixo 1: colunas
- **Array 3D:** 3 eixos (cubo de dados).

## Atributos Essenciais do Array

Permitem inspecionar a estrutura de um `ndarray`:

- `ndarray.ndim`: Número de dimensões (eixos).
- `ndarray.shape`: Tupla com o tamanho de cada dimensão.
- `ndarray.size`: Número total de elementos.
- `ndarray.dtype`: Tipo de dados dos elementos (ex: `int64`, `float64`).

```
arr2d = np.array([[1, 2, 3, 4],
                  [5, 6, 7, 8]]) # 2 linhas, 4 colunas

print(f"ndim: {arr2d.ndim}")     # Saída: 2
print(f"shape: {arr2d.shape}")   # Saída: (2, 4)
print(f"size: {arr2d.size}")     # Saída: 8
print(f"dtype: {arr2d.dtype}")   # Saída: int64
```

# Parte 2

# Criação e Manipulação de Arrays

## Criação de Arrays

- **A partir de estruturas Python:**
  `np.array([9, 7, 20])`
- **Funções especializadas (mais eficientes):**
  - `np.zeros((2, 3))`: Preenche com zeros.
  - `np.ones((2, 3))`: Preenche com uns.
  - `np.full((2, 3), 7)`: Preenche com um valor específico.
  - `np.empty((2, 2))`: Array "vazio", mais rápido para inicializar quando todos os valores serão sobrescritos.

## Geração de Sequências: `arange` vs. `linspace`

- `np.arange([start,] stop, [step,])`: Define o **passo** entre os valores. Análogo ao `range` do Python.
- `np.linspace(start, stop, num)`: Define o **número de pontos** desejado no intervalo.

| **Característica** | **`np.arange()`** | **`np.linspace()`** |
| --- | --- | --- |
| **Parâmetro Principal** | `step` (tamanho do passo) | `num` (número de pontos) |
| **Uso com Floats** | Desaconselhado; pode gerar imprecisão [31, 32] | **Recomendado**; garante precisão [31] |
| **Caso de Uso Ideal** | Sequências com passo inteiro. | Amostras para funções e gráficos. |

## Manipulação de Forma: `reshape`, `ravel`, `flatten`

- `arr.reshape((2, 4))`: Altera a forma do array. Retorna uma ***view*** se possível.
- `arr.ravel()`: "Achata" o array para 1D. Retorna uma ***view*** sempre que possível (mais rápido).
- `arr.flatten()`: "Achata" o array para 1D. **Sempre** retorna uma ***cópia*** (mais seguro, porém mais lento).

| Função | Retorna Cópia ou View? | Performance |
|---|---|---|
| `ravel()` | View, sempre que possível | Mais rápida, eficiente em memória |
| `flatten()` | Sempre uma cópia | Mais lenta, consome mais memória |
| `reshape(-1)` | View, sempre que possível | Rápida, comparável a `ravel` |

## Combinando e Dividindo Arrays

### Combinar:

- `np.concatenate()`: Junta arrays ao longo de um eixo **existente**.
- `np.stack()`: Empilha arrays ao longo de um eixo **novo**.
- `np.vstack()` e `np.hstack()` são atalhos para concatenação vertical e horizontal.

### Dividir:

- `np.split()`: Divide um array em sub-arrays de tamanho igual.
- `np.array_split()`: Permite divisões desiguais.
- `np.vsplit()` e `np.hsplit()` são atalhos para divisão vertical e horizontal.

# Parte 3

# Indexação, Operações e Broadcasting

## Métodos de Indexação

- **Básica:** `arr[linha, coluna]` para acessar elementos. `arr[start:stop:step]` para fatiar.
- **Booleana (Mascaramento):** Filtra dados com base em condições.
    i. Crie uma máscara: `mascara = arr > 10`.
    ii. Aplique a máscara: `arr[mascara]`.
    iii. Combine condições com `&` (E) e `|` (OU).
- **Avançada (Fancy Indexing):** Use listas ou arrays de inteiros para selecionar elementos em qualquer ordem: `arr`.

## O Paradigma de `View` vs. `Copy`

- **View (Visão):**
  - Aponta para os **mesmos dados** na memória. Rápida e eficiente.
  - Modificar a view **altera o array original**.
  - Retornada por **fatiamento básico**.
- **Copy (Cópia):**
  - Cria um novo array com **dados duplicados**. Mais lenta e consome mais memória.
  - Modificar a cópia **não altera o original**.
  - Retornada por **indexação booleana e avançada**.

Para verificar, use o atributo `.base`. Se for `None`, é uma cópia. Se for outro array, é uma view.

## Funções Universais (ufuncs)

São o coração das operações matemáticas no NumPy.

- Operam **elemento a elemento** de forma vetorizada.
- Implementadas em C para máxima performance.
- **Exemplos:**
  - Aritméticas: `np.add`, `np.multiply` (ou `+`, `*`).
  - Trigonométricas: `np.sin`, `np.cos`.
  - Exponenciais e Logarítmicas: `np.exp`, `np.log`, `np.sqrt`.

## Agregação e Estatística

Funções que resumem os dados de um array.

- `np.sum`, `np.mean`, `np.std`, `np.min`, `np.max`, `np.median`.
- O parâmetro `axis` define o eixo da operação (`axis=0` para colunas, `axis=1` para linhas).
- `np.argmin()` e `np.argmax()` retornam o **índice** do valor mínimo/máximo.

```
matriz = np.arange(12).reshape(3, 4)
# Soma de cada coluna
soma_colunas = matriz.sum(axis=0)
# Máximo de cada linha
max_linhas = matriz.max(axis=1)
```

## O Poder do Broadcasting

Mecanismo que permite operações em arrays de formas diferentes.

- O array menor é "esticado" virtualmente para corresponder à forma do maior, **sem criar cópias** na memória.
- **Regras (da direita para a esquerda):**
    i. Dimensões são compatíveis se forem **iguais** ou se uma delas for **1**.
    ii. Se o número de eixos for diferente, `1`s são adicionados à esquerda da forma do array menor.

## Exemplo de Broadcasting

Adicionar um vetor a cada linha de uma matriz.

```
matriz = np.ones((3, 4))      # shape (3, 4)
vetor_linha = np.arange(4)  # shape (4,)

# Análise das regras:
# matriz.shape -> (3, 4)
# vetor_linha.shape -> (4,) -> Regra 1: (1, 4)
# Regra 2: A dimensão 1 do vetor é esticada para 3.
# Formas compatíveis: (3, 4) e (3, 4)

resultado = matriz + vetor_linha
print(resultado)
# [[1. 2. 3. 4.]
#  [1. 2. 3. 4.]
#  [1. 2. 3. 4.]]
```

# Parte 4

# Álgebra Linear e Aplicações

## Geração de Números Aleatórios

- **Abordagem Moderna (Recomendada):**
  - `rng = np.random.default_rng(seed=42)`.
  - Cria um gerador **local e isolado**, evitando problemas com estado global.
  - Métodos: `rng.random()`, `rng.integers()`, `rng.normal()`.
- **Semente (`seed`):**
  - Garante a **reprodutibilidade** dos resultados, crucial para experimentos e depuração de código.

## Álgebra Linear com `numpy.linalg`

- **Produto de Matrizes:**
  - `*` -> Multiplicação **elemento a elemento**.
  - `@` ou `np.matmul()` -> Produto de matrizes **padrão** (recomendado).
- **Funções Essenciais:**
  - `linalg.inv()`: Calcula a inversa da matriz.
  - `linalg.det()`: Calcula o determinante.
  - `linalg.solve(A, b)`: Resolve o sistema de equações lineares $Ax = b$. Mais estável e rápido que `inv(A) @ b`.
  - `linalg.eig()`: Calcula autovalores e autovetores.

## Aplicação: Integração com Pandas

Pandas é construído sobre NumPy, a integração é direta e eficiente.

- **De Pandas para NumPy:**
  - `df.to_numpy()` converte um DataFrame para um `ndarray`.
- **De NumPy para Pandas:**
  - `pd.DataFrame(meu_array)` cria um DataFrame a partir de um `ndarray`.

```
import pandas as pd
dados_np = np.array([, , ])
df = pd.DataFrame(dados_np, columns=)
print(df)
```

## Aplicação: Visualização com Matplotlib

Matplotlib aceita `ndarrays` diretamente como entrada para gráficos.

```
import matplotlib.pyplot as plt

# Gera 100 pontos para o eixo x de 0 a 2*pi
x = np.linspace(0, 2 * np.pi, 100)
# Calcula o seno de cada ponto x
y = np.sin(x)

plt.plot(x, y)
plt.title('Gráfico da Função Seno')
plt.grid(True)
plt.show()
```

## Aplicação: Processamento de Imagens

Imagens digitais são, fundamentalmente, `ndarrays`.[12, 93, 94]

- **Tons de cinza:** Matriz 2D (altura, largura).
- **Colorida (RGB):** Array 3D (altura, largura, 3).[95, 96]

Todo o poder do NumPy (fatiamento, broadcasting, ufuncs) pode ser usado para cortar, ajustar brilho/contraste e aplicar filtros em imagens.[95, 97]

# Conclusão

- **NumPy** é a base da computação numérica em Python, oferecendo performance através do `ndarray` e da **vetorização**.
- Dominar seus componentes, desde a **criação e manipulação** de arrays até a **indexação**, **operações matemáticas** e **broadcasting**, é essencial.
- É a **"lingua franca"** que unifica o ecossistema de Data Science (Pandas, Matplotlib, Scikit-learn), permitindo um fluxo de trabalho coeso e eficiente.

# Perguntas?

# Referências


- https://numpy.org/doc/stable/
- https://scipy-lectures.org/intro/numpy/index.html
- https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html
- https://www.datacamp.com/tutorial/python-numpy-tutorial
- https://realpython.com/numpy-tutorial/
- https://www.w3resource.com/python-exercises/numpy/