Tratamento de Exceções

Prof. Gabriel Rodrigues Caldas de Aquino

gabrielaquino@ic.ufrj.br
Instituto de Computação - Universidade Federal do Rio de Janeiro

Compilado em:
November 24, 2025

Afinal qual o motivo de tratarmos exceções?

Antes de começarmos o nosso assunto, uma reflexão...

Qual o efeito de um código que não funciona corretamente?

Veja o caso to THERAC-25

Erro de Sintaxe Vs. Exceção

Pontos de observação

- **Erros de Sintaxe**:
  - Detectados antes da execução
  - Exemplo: esquecer dois-pontos (`:`) em um `if`
  - Mensagem mostra o local aproximado do erro antes de executar o código
- **Exceções**:
  - Ocorrem durante a execução do código
  - Representam erros lógicos ou condições inesperadas
    - Exemplos: divisão por zero, tipo incorreto
  - Mostram tipo da exceção e (ex: `ZeroDivisionError`) encerram a execução do código

Erro de Sintaxe Vs. Exceção

Principais Diferenças

- **Erro de Sintaxe**:
  - Impede a execução do código
  - Ou seja o código não roda
  - Precisa ser corrigido antes de executar
- **Exceção**:
  - Ocorre **durante** a execução do código
  - Pode ser tratada com `try-except`
  - Caso seja tratada, o funcionamento do código pode ser recuperado

Exemplos de Erros de Sintaxe

Exemplos de Erros de Sintaxe

- **Esquecer os dois-pontos**:

```
if x > 5   # ERRO: falta ':'
    print("Maior que 5")
```

- **Parênteses não fechados**:

```
print("Olá, mundo"   # ERRO: falta ')'
```

Exemplos de Exceções em Python

Exemplos Práticos

- **ZeroDivisionError**:

```
10 / 0   # Tenta dividir por zero
```

- **NameError**:

```
print(var_inexistente)  # Variável não definida
```

- **TypeError**:

```
"2" + 2  # Concatenação de tipos incompatíveis
```

- **IndexError**:

```
lista = [1, 2]
lista[3]  # Acesso a índice inexistente
```

Mas o que é uma exceção?

Definição

Uma exceção é um acontecimento inesperado ou incomum no fluxo normal do código.

Características principais

- Situações que fogem do comportamento esperado do código
  - Podemos prever ou não
- Códigos podem lançar exceções intencionalmente
- `ZeroDivisionError` e `ValueError` são exemplos de exceções

Então, exceção é isso!

Pontos principais

- **Interrompem** no fluxo normal do programa
- Ocorrem quando algo **inesperado** acontece
- Exemplos:
  - Tentar abrir um arquivo que não existe
  - Dividir um número por zero
  - Acessar uma posição inválida em uma lista

Por que tratar exceção é importante?

- Permitem **recuperar** o programa de erros
- Evitam que o programa **trave** completamente
- Oferecem **feedback** útil para depuração

Entendendo as Mensagens de Erro

Mensagem de Erro

- **Traceback**:
  - Histórico que levou ao erro
- **Localização**:
  - `path/to/file.py`
  - Linha 3: `c=a/b`
- **Tipo da Exceção**:
  - Classe do erro
  - ex: `ZeroDivisionError`
- **Descrição**:
  - Explicação legível do problema
  - ex: "division by zero"

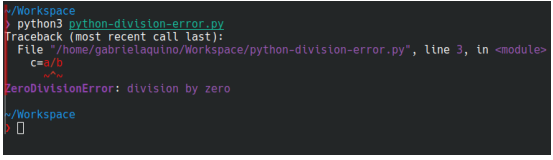


Figure: Exemplo de mensagem

Try/Except

Código Original

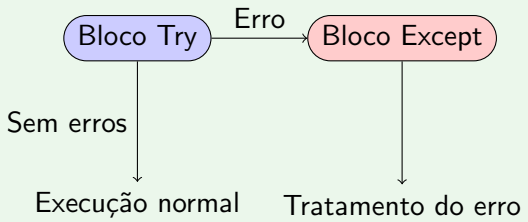
```
a = 10
b = 0
c = a / b  # Problema aqui
print(c)
```

Versão Corrigida com Try/Except

```
a = 10
b = 0

try:
    c = a / b
except ZeroDivisionError:
    print("Erro: Divisão por zero")
    c = float('inf') # Valor padrão
```

Fluxo de Execução

Fluxo de Execução



Como podemos fazer o tratamento?

- Especificar o tipo de exceção
- Registrar que ocorreu um problema
- Definir valores padrão caso tenha um problema

Exemplo de tratamento

Exemplo de tratamento

```
try:
    print("Início do bloco try")
    x = 10 / 0  # O problema ocorre aqui
    print("Esta linha NUNCA será executada")
except ZeroDivisionError:
    print("Erro capturado - divisão por zero")
```

Qual problema temos aqui?

Código Vulnerável

```
idade = int(input("Digite sua idade: "))
if idade >= 18:
    print("Maior de idade")
else:
    print("Menor de idade")
```

O que pode dar errado?

Porque esse código é vulnerável ?

Problema com Entrada do Usuário

Código Vulnerável

```
idade = int(input("Digite sua idade: "))
if idade >= 18:
    print("Maior de idade")
else:
    print("Menor de idade")
```

O que pode dar errado?

- ValueError: Se usuário digitar "dez" em vez de 10

Solução com Tratamento de Exceções

Código com o tratamento de exceção

```
try:
    idade = int(input("Digite sua idade: "))
    if idade >= 18:
        print("Maior de idade")
    else:
        print("Menor de idade")
except ValueError:
    print("Por favor, digite apenas números!")
```

O que pode dar errado neste codigo?

Discutam o código abaixo

```
num = int(input("Número: "))
print(f"Resultado: {100 / num}")
```

Riscos levantados

Pontos de risco

```
num = int(input("Número: "))          # Risco 1: Valor não inteiro
print(f"Resultado: {100 / num}")      # Risco 2: Divisão por zero
```

- ValueError:
  - Entradas como "vinte", "a"...
- ZeroDivisionError:
  - Divisão por zero caso num=0

Solução de Tratamento

Código com Tratamento de Erros

```
try:
    num = int(input("Número: "))
    print(f"Resultado: {100 / num}")
except ValueError:
    print("Digite um número inteiro válido!")
except ZeroDivisionError:
    print("Não pode ser zero!")
```

Exemplo de Execuções

- Entrada "10" → Resultado: 10.0
- Entrada "0" → "Não pode ser zero!"
- Entrada "abc" → "Digite um número inteiro válido!"

O Bloco `else` no Tratamento de Exceções

Quando usar?

O bloco `else` executa **somente se**:

- O bloco `try` for concluído **sem erros**
- Nenhuma exceção foi levantada

Diferença entre fluxos

```
try:
    # Código que pode falhar
except MinhaExcecao:
    # Executa SE ocorrer erro
else:
    # Executa SE NÃO ocorrer erro
finally:
    # Executa SEMPRE
```

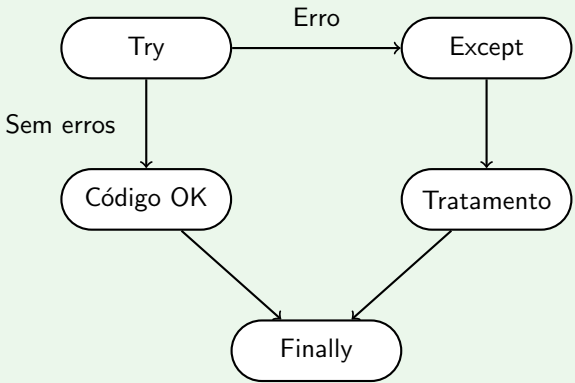
O bloco `finally`

Características do `finally`

- Sempre executa, independentemente:
  - Se ocorrer erro **ou não**
  - Se o erro foi tratado **ou não**
  - Se houve `return` no bloco
- Uso típico para:
  - Liberar recursos (arquivos, conexões)
  - Fazer limpeza
  - Registrar/logging de operações

Fluxo Try-Except-Finally

Fluxo de Execução com Tratamento de Exceções com o finally



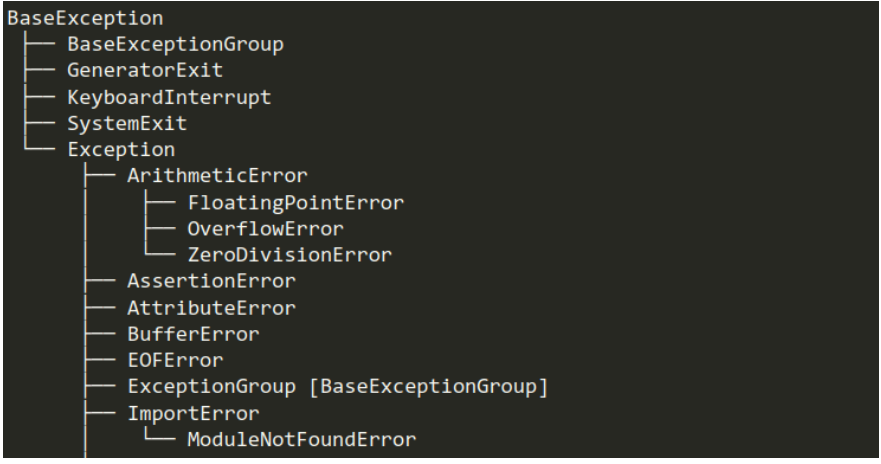
O Bloco `finally` em Python

Funcionamento Básico

```
arquivo = None
try:
    arquivo = open("dados2.txt", "r")
    # Operações com o arquivo
except FileNotFoundError:
    print("Arquivo não encontrado!")
finally:
    print("Sempre executa")
    if arquivo != None:
        arquivo.close()  # Garante o fechamento
    else:
        print(arquivo)
```

Hierarquia de Exceções em Python

Todas as exceções herdam de `BaseException`. Veja "Exception hierarchy" em `https://docs.python.org/3/library/exceptions.html`



Capturando Exceções Genéricas

Exemplo Prático

```
try:
    num = int(input("Digite um número: "))
    resultado = 100 / num
    print(f"Resultado: {resultado}")

except Exception as err:
    print(f"Ocorreu um erro: {type(err).__name__}")
    print(f"Mensagem: {str(err)}")
    print(f"Detalhes completo: {err}")
```

Tratando Diferentes Tipos de Exceções

Exemplo Completo

```
try:
    num = int(input("Digite um número (não zero): "))
    resultado = 100 / num
    print(f"Resultado: {resultado:.2f}")

except ZeroDivisionError:
    print("Erro: Não é possível dividir por zero!")
except ValueError:
    print("Erro: Digite apenas números inteiros!")

except BaseException as err:
    print()
    print(f"Ops! {type(err).__name__}")
    print(f"Fim!")
```

Exceções Personalizadas

Como criar uma exceção básica

```
class MeuErroCustomizado(Exception):
    pass

raise MeuErroCustomizado("Mensagem de erro especial")
```

Exemplo Prático

```
try:
    raise MeuErroCustomizado("Algo deu errado!")
except MeuErroCustomizado as erro:
    print(f"Erro capturado: {erro}")
```

Saída:

```
Erro capturado:  Algo deu errado!
```

Verificação de Triângulo Equilátero

Código que verifica se os lados formam um triângulo equilátero

```
def verifica_equilatero(triangulo):
    if triangulo[0] == triangulo[1] == triangulo[2]:
        return True
    else:
        raise ValueError("Não é um triângulo equilátero!")

try:
    lados = [5, 5, 5]
    if verifica_equilatero(lados):
        print("É um triângulo equilátero!")
except ValueError as e:
    print(f"Erro: {e}")
```

Exceções - Criando exceções personalizadas

Quando usar?

Usada quando precisamos definir nossos próprios tipos de erro para tornar o código mais legível e facilitar o tratamento de erros específicos.

Passo a passo para criação

1. Criar uma classe que herda de `Exception`
2. Definir um construtor (`__init__`) para personalizar a exceção
3. Levantar a exceção (`raise`) no código
4. Capturar a exceção (`except`) e tratá-la

Exceções - Criando exceções personalizadas

1. Criar uma classe que herda de Exception

Cada exceção personalizada deve ser uma classe que herda da classe Exception. Isso garante que ela tenha o comportamento de uma exceção normal.

Exemplo Básico

```
class SaldoInsuficienteError(Exception):
    """Exceção para indicar que o saldo da conta é insuficiente."""
    pass
```

`SaldoInsuficienteError` já funciona como uma exceção, mas ainda não tem uma mensagem personalizada.

Exceções - Criando exceções personalizadas

2. Definir um construtor (__init__) para personalizar a exceção

Podemos adicionar um construtor para aceitar parâmetros e definir uma mensagem de erro.

Exemplo com construtor personalizado

```
class SaldoInsuficienteError(Exception):
    """Exceção para saldo insuficiente."""
    def __init__(self, saldo, valor,
                 mensagem="Saldo insuficiente para a operação."):
        self.saldo = saldo
        self.valor = valor
        self.mensagem = f"{mensagem} Saldo atual: R${saldo:.2f}, valor
solicitado: R${valor:.2f}."
        super().__init__(self.mensagem)
```

3. Levantar a exceção (`raise`) no código

Agora podemos usar `raise` para lançar essa exceção em uma função.

Exemplo de uso com `raise`

```
def sacar(saldo, valor):
    if valor > saldo:
        raise SaldoInsuficienteError(saldo, valor)
    saldo -= valor
    return saldo
```

4. Capturar a exceção (except) e tratá-la no código

Como qualquer outra exceção, podemos capturá-la com `try-except`

Exemplo completo de tratamento

```
try:
    saldo_atual = 100.0
    novo_saldo = sacar(saldo_atual, 200.0)
    print(f"Saque realizado! Novo saldo: R${novo_saldo:.2f}")
except SaldoInsuficienteError as e:
    print(f"Erro: {e}")
```

Criando Exceções Personalizadas

```
class SaldoInsuficienteError(Exception):
    def __init__(self, saldo, valor, mensagem="Saldo insuficiente para a
    operação."):
        self.saldo = saldo
        self.valor = valor
        self.mensagem = f"{mensagem} Saldo atual: R${saldo:.2f},
        valor solicitado: R${valor:.2f}."
        super().__init__(self.mensagem)
def sacar(saldo, valor):
    if valor > saldo:
        raise SaldoInsuficienteError(saldo, valor)
    saldo -= valor
    return saldo
try:
    saldo_atual = 100.0
    novo_saldo = sacar(saldo_atual, 20.0)
    print(f"Saque realizado! Novo saldo: R${novo_saldo:.2f}")
except SaldoInsuficienteError as e:
```

Dicas e melhores práticas

- Não use *except Exception* de forma genérica.
- Não use *except* sem nenhuma exceção.
- Para pegar uma exceção e tratar, precisa que o código que você queira testar esteja dentro do *try*
- Não sabe qual exceção pegar? Leia a documentação do python
  - Google é seu amigo, procure por *"python exception list"*
  - ou vá na URL: `https://docs.python.org/3/library/exceptions.html`

Persistência de Dados

Prof. Gabriel Rodrigues Caldas de Aquino

gabrielaquino@ic.ufrj.br
Instituto de Computação - Universidade Federal do Rio de Janeiro

Compilado em:
November 24, 2025

Persistência de Dados

Problema

Os dados gerados em uma execução do código são perdidos quando o programa é encerrado.

Solução

Se precisamos usar os dados no futuro, é necessário armazená-los de forma persistente.

Mecanismos de Persistência de Dados:

- Arquivos (TXT, CSV, JSON, XML)
- Bancos de dados (SQLite, PostgreSQL, MySQL)

Persistência em Arquivos de Texto

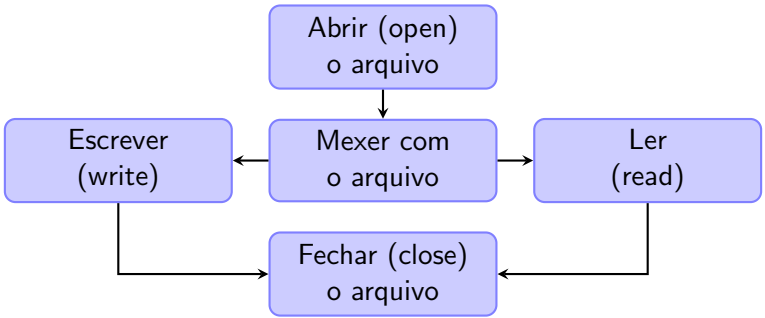
Características

- Formatos comuns: `.txt`, `.csv`, `.json`
- Armazenamento disco rígido
- Manipulação via objetos da classe `File`

Para manipular temos um "protocolo" de Uso

Três etapas: **Abrir**, **Manipular** e **Fechar**

Fluxo para se fazer a manipulação de arquivos



Abrindo Arquivos em Python

Método `open()`

Usado para abrir um arquivo. Requer dois parâmetros principais:

```
arquivo = open("nome_do_arquivo.txt", "modo_de_abertura")
```

Parâmetros

- **Nome do arquivo**:
  - Caminho completo ou relativo
- **Modo de abertura**:
  - `"r"` - Leitura
  - `"w"` - Escrita

Exemplos

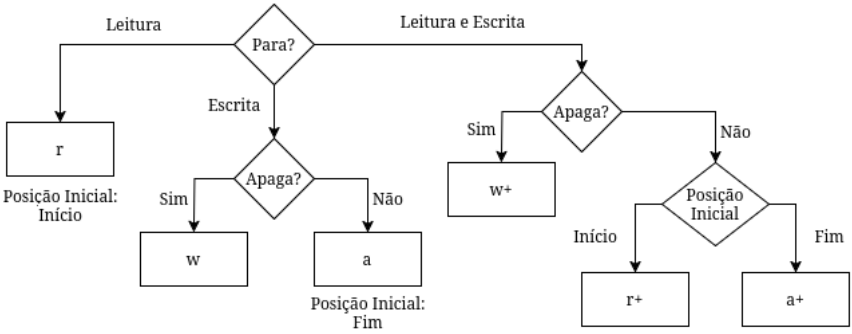
```
# Para escrita
arq = open("dados.txt", "w")

# Para leitura
arq = open("dados.txt", "r")
```

Modos de Abertura de Arquivos

O modo de abertura define como interagiremos com o arquivo:

Modo	**Descrição**	**Existência do Arquivo**
`r`	Leitura apenas	Deve existir
`w`	Escrita	Cria se não existir
`a`	Escrita no final	Cria se não existir
`r+`	Leitura e escrita	Deve existir
`w+`	Leitura e escrita	Cria se não existir
`a+`	Leitura e escrita no final	Cria se não existir

Modos de Abertura de Arquivo



Abrindo Arquivos - Modo de abertura 'r'

- Método read: ler dados de um arquivo que foi previamente aberto para leitura
- Para abrir um arquivo para leitura:
  - Precisamos dar o nome de um arquivo existente
  - Em seguida indicar o modo de leitura *r* no momento da abertura

Abrindo o arquivo dados.txt

```
arquivo = open("dados.txt", "r")
conteudo = arquivo.read()
print(conteudo)
arquivo.close()
```

Lendo linha por linha

- Em arquivos com múltiplas linhas podemos usar for loop para facilitar o tratamento linha por linha

Exemplo de leitura linha por linha

```
arquivo="dados.txt"
arq = open(arquivo, "r")
numero_da_linha = 1
for linha in arq:
    print(f"Linha {numero_da_linha}: {linha}")
    numero_da_linha = numero_da_linha + 1
arq.close()
```

Escrever em Arquivo: write() - Modo de abertura 'w'

Método write()

```
arq.write("conteúdo que será escrito no arquivo")
```

Passos para escrita em arquivo

1. Abrir o arquivo em modo de escrita (`'w'`)
2. Passar o conteúdo como string para `write()`
3. Fechar o arquivo

Escrita Básica em Arquivos

Exemplo de escrita em arquivo

```
arq = open("nomes.txt", "w")
arq.write("Gabriel, Pedro, Manoel")
arq.close()
```

Importante!

- `'w'` = write (escrita)
- Sempre feche o arquivo após escrever
- Cada `write()` grava o conteúdo exato
- Se o arquivo "nomes.txt" **já existe**, ele será **sobrescrito**

Arquivos com mais de uma linha

Leitura de Arquivos Linha por Linha

- Arquivos com múltiplas linhas contêm `\n` escondido

Exemplo Prático

Pedro\n
Antonio\n
Maria\n
Jose\n

Pulando linha na escrita em arquivos

Código de exemplo

```
arq = open("nomes.txt", "w")
arq.write("Gabriel\nPedro\nManoel")
arq.close()
```

O que acontece?

- `\n` é o **caractere especial** para quebra de linha
- Quando escrito no arquivo, ele:
  - Finaliza a linha atual
  - Move o cursor para a próxima linha

Fechando o Arquivo: close()

Método close()

```
arq.close()  # Fecha o arquivo após uso
```

Por que fechar arquivos?

- **Libera recursos do sistema**: Arquivos abertos consomem memória
- **Garante a escrita completa**: Dados podem ficar em buffer
- **Evita corrupção**: Previne acesso concorrente indevido
- **Libera o arquivo**: Permite que outros programas o acessem

Modo de Abertura 'a' (Append)

Funcionamento do modo "a"

```
arquivo = open("dados.txt", "a")  # Modo append
arquivo.write("Novo conteúdo\n")
arquivo.close()
```

Características

- **Abre para escrita** no final do arquivo
- Ponteiro no **fim do arquivo** (só escreve no final)

Gerenciamento de Arquivos com `with open`

Sintaxe Básica

```
with open("arquivo.txt", "modo") as arquivo:
    # Bloco de código
```

Exemplo Prático

```
with open("dados.txt", "r") as arq:
    conteudo = arq.read()
    print(conteudo)
```

Facilidade

- O `with` deixa arquivo aberto.
- Ao sair do bloco, o `close()` é automático.

Controlando a Posição com `seek()`

O que é `seek()`?

Método que permite mover o "cursor" de leitura/escrita para qualquer posição no arquivo

```
arquivo.seek(offset)
```

Parâmetros

- **offset**: Número de bytes para mover

Exemplos

```
# Ir para o byte 10
arquivo.seek(10)
```

Encontrando um conteúdo no arquivo

Como fazer para achar um conteúdo no arquivo?

Podemos achar um conteúdo com usando o **find**

```
f = open("vinyl_sales.txt", "r+")
conteudo = f.read()
pos = conteudo.find("achado")
print(pos)
f.seek(pos)
f.write("Encontrei")
f.close()
```

Nesse caso

Nós lemos o arquivo, depois encontramos a palavra desejada com o find e temos a posição correta na variável **pos**

Modo de Abertura r+

Funcionamento do modo "r+"

```
with open("arquivo.txt", "r+") as arquivo:
    # Operações de leitura E escrita
    conteudo = arquivo.read()  # Lê
    arquivo.write("novo texto")  # Escreve
```

Características

- Permite **leitura e escrita** no mesmo arquivo
- Não apaga o arquivo na hora de abrir
- Posição inicial: **início do arquivo**

Modo de Abertura w+ (Escrita e Leitura)

Funcionamento do modo "w+"

```
with open("arquivo.txt", "w+") as arquivo:
    arquivo.write("Conteúdo inicial\n")
    arquivo.seek(0)  # Volta ao início para leitura
    conteudo = arquivo.read()
    print(conteudo)
```

Características Principais

- **Apaga conteúdo existente**
- Posição inicial: **início do arquivo**

Cuidado!

Sempre use `seek()` antes de ler após escrever

Modo de Abertura a+ (Append e Leitura)

Funcionamento do modo "a+"

```
with open("arquivo-teste.txt", "a+") as arquivo:
    arquivo.write("Nova entrada\n")  # Escreve no FINAL
    arquivo.seek(0)                  # Volta ao início
    texto = arquivo.read()       # Lê todo conteúdo
    print(texto)
```

Características

- **Não apaga** conteúdo existente
- Posição inicial: **Final do arquivo**

Tratamento de Exceções com Arquivos

Por que tratar exceções?

Lidar com arquivos pode lançar exceções inesperadas que devem ser gerenciadas.

Principais exceções com arquivos

- `FileNotFoundError`: Arquivo não existe ou caminho incorreto
- `IOError`: Erros gerais de entrada/saída (disco cheio, permissões)

Tratamento de Exceções com Arquivos

Código de Exemplo

```
try:
    f = open("arquivo-inexistente.txt", "r")
    conteudo = f.read()
    f.close()
    print(conteudo)
except FileNotFoundError:
    print("Arquivo não existente")
```

Demonstração: Tratando Arquivo Corrompido

Código de Tratamento

```
try:
    f = open("arquivo-corrompido.txt", "r")
    conteudo = f.read()
    f.close()
    print(conteudo)
except IOError:
    print("Erro: Problema ao ler o arquivo")
```

Biblioteca Numpy

Prof. Gabriel Rodrigues Caldas de Aquino

gabrielaquino@ic.ufrj.br
Instituto de Computação - Universidade Federal do Rio de Janeiro

Compilado em:
November 24, 2025

NumPy - Introdução

O que é NumPy?

- Pacote fundamental para **computação científica** em Python

Objeto principal: `numpy.ndarray`

- Vetor **n-dimensional** (arrays multidimensionais)
- Características fundamentais:
  - **Tamanho fixo** (definido na criação)
  - **Indexado** por tuplas de inteiros positivos
  - **Homogêneo** - todos elementos do mesmo tipo

Documentação Oficial

Há diversas métodos e atributos disponíveis no NumPy, os quais podem ser consultados na documentação oficial no endereço: `https://numpy.org/doc/stable/index.html`

NumPy - Arrays

Importe o módulo e crie arrays a partir de listas

```
import numpy as np
np.array(lista)
```

Exemplos

```
# Array 1D (vetor)
x = np.array([1, 2, 3])


# Array 2D (matriz)
y = np.array([[1., 0., 0.],
              [0., 1., 0.]])
```

Saída dos Exemplos

```
#Saída:
>>> x
array([1, 2, 3])

>>> y
array([[1., 0., 0.],
       [0., 1., 0.]])
```

Tipos de Numpy array

```
>>> type(x)
numpy.ndarray

>>> type(y)
 numpy.ndarray
```

Características

- `x` é um array **1-dimensional** (vetor)
- `y` é um array **2-dimensional** (matriz)
- Ambos são do tipo `numpy.ndarray`

NumPy - O Atributo dtype

Definição

O dtype define o tipo dos elementos armazenados no array NumPy

Exemplo Inicial

```
minhalista = [1, 2, 3, 4, 5]
arr = np.array(minhalista)
print(arr)          # array([1, 2, 3, 4, 5])
print(type(arr))    # <class 'numpy.ndarray'>
print(arr.dtype)    # dtype('int64')
```

Com Número Decimal

```
minhalista = [1, 2, 3, 4, 5.5]
arr = np.array(minhalista)
print(arr.dtype)  # dtype('float64')
```

Com String

```
minhalista = [1, 2, 3, 4, "ola"]
arr = np.array(minhalista)
print(arr.dtype)  # dtype('<U21')
```

NumPy - Propriedades de Arrays

Criação de Array 2D

```
arr2 = np.array([[1, 2],
                 [3, 4]])
```

Propriedades Fundamentais

- `.ndim` - Número de dimensões
- `.shape` - Tupla com tamanho em cada dimensão
- `.size` - Número total de elementos
- `.dtype` - Tipo dos dados

Aplicado ao Exemplo

```
>>> arr2.ndim
2
>>> arr2.shape
(2, 2)
>>> arr2.size
4
>>> arr2.dtype
dtype('int64')
```

NumPy - Propriedades Básicas de Arrays

Propriedade	**Descrição**
`ndarray.ndim`	Número de eixos (dimensões) do array.
`ndarray.shape`	Dimensões do array. Uma tupla de inteiros indicando o tamanho em cada dimensão. Para uma matriz com n linhas e m colunas, shape será (n, m). O comprimento da tupla shape é igual ao número de dimensões (`ndim`).
`ndarray.size`	Número total de elementos do array.
`ndarray.dtype`	Objeto que descreve o tipo dos elementos no array. Podem ser usados tipos padrão do Python ou tipos específicos do NumPy como `numpy.int32`, `numpy.int16` e `numpy.float64`.

NumPy – Arrays com Valores Pré-definidos

Por que usar valores pré-definidos?

- Facilita a **inicialização** de matrizes antes do preenchimento com dados
- Evita **erros** na alocação de memória para grandes arrays
- Útil para **cálculos numéricos** e simulações

Funções de Criação

Função	**Descrição**
`np.zeros(shape)`	Cria array preenchido com **zeros**
`np.ones(shape)`	Cria array preenchido com **uns**
`np.empty(shape)`	Cria array com **valores aleatórios**

NumPy – Arrays 1D com Valores Pré-definidos

Exemplos 1D

```
# Array de 5 zeros
np.zeros(5)

# Array de 3 uns
np.ones(3)

# Array vazio 4 posições
np.empty(4)
```

Dica Importante

Especifique sempre o `dtype` para controle preciso do tipo numérico:
`np.zeros(5, dtype=np.float32)`

NumPy – Arrays 2D com Valores Pré-definidos

Exemplos 2D

```
# Matriz 2x3 de zeros
np.zeros((2,3))

# Matriz 3x3 de uns
np.ones((3,3))

# Matriz 2x2 vazia
np.empty((2,2))
```

Dica Importante

Ao criar arrays multidimensionais, lembre-se de usar dois parênteses:
o primeiro envolve a tupla com as dimensões, o segundo é da chamada da função.

Exemplo correto: `np.ones((3,3), dtype=int)`

Exemplo: Criação de Arrays com Zeros

```
# importando o módulo
import numpy as np

# criando um array com cinco elementos sendo zeros
array_zeros = np.zeros(5)
print(array_zeros)
# Saída: [0. 0. 0. 0. 0.]

# criando uma matriz 3x4 preenchida com zeros
matriz_zeros = np.zeros((3, 4))
print(matriz_zeros)
# Saída: [[0. 0. 0. 0.]
          [0. 0. 0. 0.]
          [0. 0. 0. 0.]]
```

Exemplo: Criação de Arrays com np.empty

```
# importando o módulo
import numpy as np

# criando uma matriz 3x3 sem inicialização garantida
array_empty = np.empty((3, 3))
print(array_empty)
# Saída: [[4.67296746e-307 1.69121096e-306 1.37959131e-306]
          [1.11261162e-306 1.11260619e-306 9.34609790e-307]
          [8.45559303e-307 9.34600963e-307 1.37959740e-306]]
```

Observação

Os valores podem ser aleatórios, pois `np.empty` não inicializa os elementos, apenas aloca espaço na memória.

Criando Arrays com `np.arange()`

Array 1D (Vetor)

```
# Vetor de 0 a 8
v = np.arange(9)
print(v)
# [0 1 2 3 4 5 6 7 8]

# Vetor de 0 a 8 pulando de 2 em 2
v2 = np.arange(0, 9, 2)
print(v2)
# [0 2 4 6 8]
```

Dica

`np.arange(início, fim, passo)` cria um vetor com espaçamento definido.
O valor final **não é incluído**. O passo pode ser positivo ou negativo!

Criando Matrizes com `reshape()`

Matriz 2x2

```
# Criar e redimensionar
m2x2 = np.arange(1,5).reshape(2,2)
print(m2x2)
# [[1 2]
#  [3 4]]
```

Matriz 3x3

```
# Criar e redimensionar
m3x3 = np.arange(9).reshape(3,3)
print(m3x3)
# [[0 1 2]
#  [3 4 5]
#  [6 7 8]]
```

Dica

- `reshape(linhas, colunas)` para transformar um array 1D em matriz.
- **O número total de elementos deve ser compatível**

NumPy – Entendendo Eixos (Axes)

O que são eixos em NumPy?

Em arrays multidimensionais, os eixos representam as direções ao longo das quais as operações podem ser aplicadas.

- `axis=0`: opera coluna por coluna
  Saída: `[coluna1, coluna2, coluna3]`
- `axis=1`: opera linha por linha
  Saída: `[linha1, linha2, linha3]`

Dica Visual

Pense que o `axis` é a dimensão que será "reduzida".
`axis=0` colapsa cada coluna.
`axis=1` colapsa cada linha.

NumPy – Exemplo Prático com Eixos

Soma com axis=0 e axis=1

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

# Soma por colunas (axis=0)
print(np.sum(matriz, axis=0))
# Saída: [12 15 18]

# Soma por linhas (axis=1)
print(np.sum(matriz, axis=1))
# Saída: [ 6 15 24]
```

Operações ao Longo dos Eixos em NumPy

Operação	**axis = 0 (colunas)**	**axis = 1 (linhas)**
`np.sum()`	Soma por coluna	Soma por linha
`np.mean()`	Média por coluna	Média por linha
`np.max()`	Máximo por coluna	Máximo por linha
`np.min()`	Mínimo por coluna	Mínimo por linha

Exemplo: Matriz de vendas

- Linhas representam lojas
- Colunas representam produtos (Lápis, borracha, caderno)

Codigo

```
vendas = np.array([
    [10, 20, 30],  # Loja 1
    [15, 25, 35],  # Loja 2
    [12, 18, 28],  # Loja 3
    [8, 22, 26],   # Loja 4
])
np.sum(vendas, axis=0) # Soma por coluna (produtos)
# [45 85 119]
np.sum(vendas, axis=1) # Soma por linha (lojas)
# [60 75 58 56]
np.mean(vendas, axis=0) # Média por produto (em todas as lojas)
# [11.25 21.25 29.75]
```

Operações Aritméticas em NumPy – Adição Escalar (Vetor)

Adição Escalar

```
import numpy as np

A = np.arange(1, 10)
print(A)
# [1 2 3 4 5 6 7 8 9]

print(A + 5)  # Adição escalar
# [ 6  7  8  9 10 11 12 13 14]
```

Operações Aritméticas em NumPy – Matriz + Escalar

Adição Escalar em Matrizes

```
import numpy as np

B = np.random.randint(0, 10, (3,3))
print(B)
# [[4 3 1]
#  [6 0 7]
#  [9 3 3]]

print(B + 5)
# [[ 9  8  6]
#  [11  5 12]
#  [14  8  8]]
```

Exemplo: Estoque em Papelarias

- Cada linha representa uma loja;
- Cada coluna representa um item: lápis, borracha e caderno.

Código

```
import numpy as np
# Quantidade atual em 3 lojas
estoque = np.array([[10, 5, 2],
                    [3,  8, 4],
                    [6,  2, 7]])

novo_estoque = estoque + 5 # Reposição de 5 unidades em cada item

print(novo_estoque)
# [[15 10  7]
#  [ 8 13  9]
#  [11  7 12]]
```

Operações entre Vetores em NumPy - Soma

Vetor A

```
import numpy as np

A = np.array([1, 2, 3, 4, 5])
print(A)
# [1 2 3 4 5]
```

Vetor B

```
B = np.array([10, 20, 30, 40, 50])
print(B)
# [10 20 30 40 50]
```

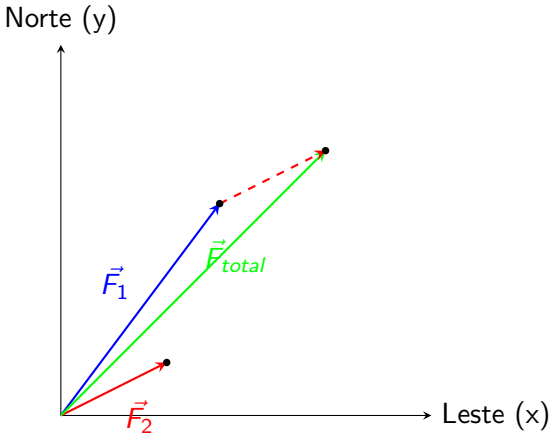
Soma A + B

```
print(A + B)
# [11 22 33 44 55]
```

Soma de Vetores com NumPy – Exemplo

Forças

```
import numpy as np

F1 = np.array([3, 4])
F2 = np.array([2, 1])

F_total = F1 + F2
print(F_total)
# [5 5]
```



Operações entre Matrizes em NumPy - Soma

Matriz B

```
B = np.random.randint(0, 10, (3,3))
print(B)
# [[4 3 1]
#  [6 0 7]
#  [9 3 3]]
```

Matriz C

```
C = np.random.randint(0, 10, (3,3))
print(C)
# [[3 7 1]
#  [6 4 5]
#  [0 1 7]]
```

Soma B + C

```
print(B + C)
# [[ 7 10  2]
#  [12  4 12]
#  [ 9  4 10]]
```

Operações entre Matrizes – Compras em uma Loja

Cada linha é um cliente e cada coluna um item comprado (lápis, caneta, caderno).

Compras no Dia 1

```
dia1 = np.array([
    [1, 2, 0],  # Cliente 1
    [0, 1, 3],  # Cliente 2
    [2, 0, 1]   # Cliente 3
])
```

Compras no Dia 2

```
dia2 = np.array([
    [0, 1, 2],
    [1, 0, 1],
    [1, 2, 1]
])
```

Total de Compras (Dia 1 + Dia 2)

```
total = dia1 + dia2
print(total)
# [[1 3 2]
#  [1 1 4]
#  [3 2 2]]
```

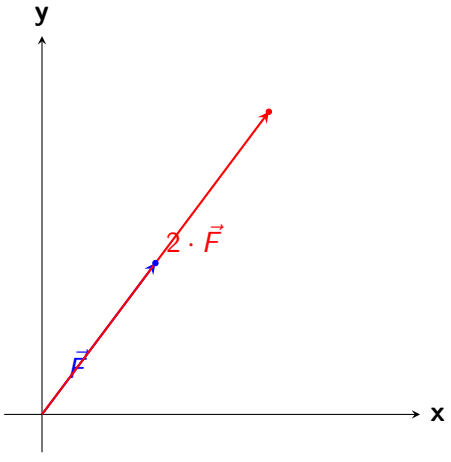
Multiplicação Escalar de Vetor

Código NumPy – Multiplicação Escalar

```
import numpy as np

F = np.array([3, 4])
print(F)
#[3 4]

# Multiplicando por 2
print(2 * F)
#[6 8]
```

Vetores

$$\vec{F} = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \quad 2 \cdot \vec{F} = \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$



NumPy - Multiplicação Element-wise

Exemplo: Receita por produto (quantidade × preço)

```
import numpy as np

Quantidade_produtos = np.array([10, 5, 2])
Preco_produtos = np.array([1, 0.5, 20])

# Gasto por produto
print(Quantidade_produtos * Preco_produtos)
# [10.   2.5 40. ]
```

Multiplicação de cada elemento

Cenário

Cada linha representa uma loja.Cada coluna representa um produto (Lápis, Borracha, Caderno). Queremos saber a receita por produto em cada loja (quantidade $\times$ preço unitário).

Quantidade Vendida (B)

```
B = np.array([[4, 3, 1],
              [6, 0, 7],
              [9, 3, 3]])
```

Preço Unitário (C)

```
C = np.array([[3, 7, 1],
              [6, 4, 5],
              [0, 1, 7]])
```

Receita por Loja e Produto (B * C)

```
print(B * C)
# [[12 21  1]
#  [36  0 35]
#  [ 0  3 21]]
```

Multiplicação Matricial com `np.dot()`

Matriz B (3x3)

```
B = np.array([[4, 3, 1],
              [6, 0, 7],
              [9, 3, 3]])
```

Matriz C (3x3)

```
C = np.array([[3, 7, 1],
              [6, 4, 5],
              [0, 1, 7]])
```

Resultado `np.dot(B, C)`

```
print(np.dot(B, C))
# [[ 30  41  26]
#  [ 18  55  41]
#  [ 45  78  45]]
```

Diferença Fundamental

- `B * C`: Multiplicação elemento a elemento
- `np.dot(B,C)`: Multiplicação de matrizes

Explicação: Multiplicação Matricial com `np.dot()`

Dado que temos duas matrizes quadradas **B** e **C**, ambas de dimensão 3×3. Estamos comparando duas formas distintas de multiplicação em NumPy: a multiplicação elemento a elemento (`*`) e a multiplicação matricial (`np.dot()`).

A diferença das duas operações:
Diferente da multiplicação elemento a elemento (que faz $B[i][j] \times C[i][j]$), a multiplicação matricial segue a regra:
Para cada elemento da matriz resultante, fazemos o produto escalar da linha i de B pela coluna j de C.

Exemplo prático:
Para calcular o valor na posição (0,0) do resultado:

$$4 \times 3 + 3 \times 6 + 1 \times 0 = 12 + 18 + 0 = 30$$

Esse processo se repete para cada posição da matriz resultante.

Transposição de Matrizes em NumPy

Matriz Original

```
B = np.array([[4, 3, 1],
              [6, 0, 7],
              [9, 3, 3]])
```

Método 1: Atributo .T

```
print(B.T)
# [[4 6 9]
#  [3 0 3]
#  [1 7 3]]
```

Método 2: Função transpose()

```
print(B.transpose())
# [[4 6 9]
#  [3 0 3]
#  [1 7 3]]
```

Resolver Sistema Linear (np.linalg.solve(a,b))

Considere o sistema linear:

$$\begin{cases} 2x + 3y - z = 5 \\ 4x + y + 2z = 6 \\ -3x + 2y + z = -4 \end{cases}$$

Representamos na forma matricial $Ax = b$:

$$A = \begin{bmatrix} 2 & 3 & -1 \\ 4 & 1 & 2 \\ -3 & 2 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 6 \\ -4 \end{bmatrix}$$

Código Python para resolver usando NumPy:

```
import numpy as np
A = np.array([[2, 3, -1],
              [4, 1, 2],
              [-3, 2, 1]])
b = np.array([5, 6, -4])
x = np.linalg.solve(A, b)
print("Solução: ", x)
```

Resolva

Considere o seguinte sistema linear:

$$\begin{cases} 3x_1 - 2x_2 + 4x_3 + x_4 - x_5 + 2x_6 = 7 \\ -2x_1 + x_2 - x_3 + 3x_4 + 5x_5 - x_6 = -3 \\ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 10 \\ 4x_1 - x_2 + 2x_3 - x_4 + 3x_5 - 2x_6 = 2 \\ -3x_1 + 5x_2 - x_3 + 2x_4 - 4x_5 + x_6 = 5 \\ 2x_1 + 3x_2 - 2x_3 + x_4 + x_5 - 3x_6 = -1 \end{cases}$$

Forma Matricial

Representamos o sistema como $Ax = b$:

$$A = \begin{bmatrix} 3 & -2 & 4 & 1 & -1 & 2 \\ -2 & 1 & -1 & 3 & 5 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & -1 & 2 & -1 & 3 & -2 \\ -3 & 5 & -1 & 2 & -4 & 1 \\ 2 & 3 & -2 & 1 & 1 & -3 \end{bmatrix}, \quad b = \begin{bmatrix} 7 \\ -3 \\ 10 \\ 2 \\ 5 \\ -1 \end{bmatrix}$$

Resolvendo com NumPy

Código Python para resolver o sistema com NumPy:

```
import numpy as np

A = np.array([
    [3, -2, 4, 1, -1, 2],
    [-2, 1, -1, 3, 5, -1],
    [1, 1, 1, 1, 1, 1],
    [4, -1, 2, -1, 3, -2],
    [-3, 5, -1, 2, -4, 1],
    [2, 3, -2, 1, 1, -3]
])

b = np.array([7, -3, 10, 2, 5, -1])

x = np.linalg.solve(A, b)
print("Solução:", x)
```

Biblioteca Matplotlib

Prof. Gabriel Rodrigues Caldas de Aquino

gabrielaquino@ic.ufrj.br
Instituto de Computação - Universidade Federal do Rio de Janeiro

Compilado em:
November 24, 2025

Origem e Popularidade do `Matplotlib`

- Criado por John D. Hunter em 2003 como uma biblioteca de gráficos em Python.
- Inspirado no `MATLAB`, com o objetivo de permitir a criação de visualizações de alta qualidade por cientistas e engenheiros.
- Atualmente, é amplamente utilizado em áreas como:
  - Ciência de dados,
  - Engenharia,
  - Aprendizado de máquina.

Para que Serve o `Matplotlib`?

- O `matplotlib` é usado para:
  - Criar gráficos e visualizações de dados em Python.
  - Analisar visualmente comportamentos, tendências e padrões.
  - Produzir figuras de alta qualidade para relatórios, artigos e apresentações.
- Com ele, é possível gerar:
  - Gráficos de linha, barra, pizza, dispersão (scatter), histogramas, entre outros.
- É uma ferramenta essencial em:
  - Ciência de dados,
  - Engenharia,
  - Pesquisa científica,
  - Educação.

Introdução ao pyplot

- `matplotlib.pyplot` é uma coleção de funções que faz o `matplotlib` funcionar de forma semelhante ao `MATLAB`.
- Cada função do `pyplot` realiza uma alteração na figura: cria uma figura, define uma área de plotagem, desenha linhas, adiciona rótulos, etc.
- O `pyplot` mantém estados entre chamadas de função, acompanhando a figura e a área de plotagem atual.
- **Axes** são componentes centrais da figura.

Criando Visualizações com pyplot

- Gerar visualizações com pyplot é rápido e simples.
- Exemplo básico:

Exemplo em Python

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4])
plt.ylabel('alguns números')
plt.title('Titulo')
plt.show()
```

Explicação do Código

- `import matplotlib.pyplot as plt`
  Importa a biblioteca `pyplot` do `matplotlib` como `plt` para facilitar o uso.
- `plt.plot([1, 2, 3, 4])`
  Cria um gráfico de linha simples com os valores fornecidos no eixo y;
  O eixo x é automático (índices 0 a 3).
- `plt.ylabel('alguns números')`
  Define o rótulo do eixo y com o texto "alguns números".
- `plt.title('Titulo')`
  Adiciona um título ao gráfico.
- `plt.show()`
  Exibe a figura gerada em uma janela ou no ambiente gráfico.

Plotando x versus y com pyplot

Para plotar valores de x contra y, você pode modificar o código anterior assim:

Exemplo em Python

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('valores de x')
plt.ylabel('alguns números')
plt.show()
```

Adicionando Anotações com `plt.annotate()`

- O método `plt.annotate()` permite destacar pontos importantes no gráfico com texto e setas.
- Parâmetros usados:
  - `xy=(2, 4)`: coordenada do ponto a ser anotado.
  - `xytext=(3, 6)`: posição do texto da anotação.
  - `arrowprops`: define propriedades da seta (ex: cor).

Exemplo em Python

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3], [1, 4, 9])
plt.annotate('ponto importante', xy=(2, 4),
             xytext=(3, 6),
             arrowprops=dict(facecolor='black'))
plt.show()
```

Exemplo de Marcadores e Estilo de Linha em pyplot

- O parâmetro `markersize` ajusta o tamanho dos marcadores nos pontos do gráfico.
- O estilo `'ko:'` significa:
  - `k`: cor preta (black).
  - `o`: marcador circular.
  - `:`: linha pontilhada.

Exemplo em Python

```
import matplotlib.pyplot as plt

plt.plot([2, 1], [3, 1], 'ko:', markersize=10)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

- `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html`

Adicionando Grid ao Gráfico

- O método `plt.grid(True)` adiciona uma grade (grid) ao gráfico, facilitando a leitura dos valores.
- É especialmente útil para visualizações com muitos dados ou comparações visuais.

Exemplo em Python

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'bo-')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.show()
```

Definindo os Limites dos Eixos com `plt.axis`

- O método `plt.axis()` permite definir manualmente os limites dos eixos do gráfico.
- A sintaxe `plt.axis((xmin, xmax, ymin, ymax))` define:
  - `xmin = 0`, `xmax = 6`
  - `ymin = 0`, `ymax = 20`
- Isso pode ser útil para controlar o zoom do gráfico ou garantir uma escala fixa.

Exemplo em Python

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro-')
plt.axis((0, 6, 0, 20))
plt.grid(True)
plt.show()
```

Gráfico do crescimento populacional

Vá na página da Wikipédia que contém dados reais de crescimento populacional mundial `https://pt.wikipedia.org/wiki/Crescimento_populacional` e crie um gráfico com o crescimento populacional ao longo dos anos

Evitando Notação Científica no Eixo Y

- Quando os valores do eixo Y são muito grandes, o `matplotlib` ativa a notação científica automaticamente (ex: `1e6`).
- Para desativar essa notação e mostrar os números completos, usamos:
  - `plt.ticklabel_format(style='plain', axis='y')`

Exemplo em Python

```
ano = [1750, 1800, 1850, 1900, 1950, 1955, 1960, 1965,
       1970, 1975, 1980, 1985, 1990, 1995, 2000, 2005]
populacao = [791000, 978000, 1262000, 1650000, 2518629,
             2755823, 3021475, 3334874, 3692492, 4063587,
             4434682, 4830979, 5263593, 5674380, 6070581, 6453628]
plt.plot(ano, populacao)
plt.ticklabel_format(style='plain', axis='y')
plt.show()
```

Gráfico de Barras: Crescimento Populacional

- Podemos usar `plt.bar()` para representar os dados como um gráfico de barras.
- Útil para destacar visualmente a variação entre os anos.

Exemplo com `plt.bar()`

```
plt.bar(ano, populacao)
plt.xlabel('Ano')
plt.ylabel('População')
plt.ticklabel_format(style='plain', axis='y')
plt.grid(True)
plt.title('Crescimento Populacional Mundial')
plt.show()
```

Gráfico de Barras - Tamanho das barras

- Podemos usar `plt.bar()` para representar os dados como um gráfico de barras.
- O parâmetro `width` controla a largura das barras.

Exemplo com `plt.bar(..., width=20)`

```
plt.bar(ano, populacao, width=20)
...
plt.show()
```

plot() vs scatter()

- Tanto `plt.plot(..., marker='o')` quanto `plt.scatter(...)` podem ser usados para representar pontos.
- `plot()` com marcador desenha uma linha conectando os pontos (a menos que especificado para não fazê-lo).
- `scatter()` desenha apenas os pontos individuais, sem conectá-los.
- `scatter()` também permite personalizar tamanho, cor e transparência ponto a ponto.

Exemplo em Python

```
a = np.array([0, 2, 4, 0, 2, 2, 3, 2])
b = np.array([0, 1, 0, 0, 0, 2, 1, 1.2])

plt.scatter(a, b, marker='o')  # ou plt.plot(a, b)
plt.grid(True)
plt.show()
```

Plotando Múltiplas Curvas

- Podemos usar `plt.plot()` várias vezes para desenhar múltiplas curvas no mesmo gráfico.
- Cada chamada adiciona uma nova série de dados com estilo independente.

Exemplo em Python

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'bo-', label='y = x²')
plt.plot([1, 2, 3, 4], [1, 2, 3, 4], 'rs--', label='y = x')

plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.show()
```

Posicionando a Legenda com `loc`

- Use o parâmetro `loc` para escolher onde a legenda aparece.
- Exemplo simples:

Código Exemplo

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3], label='Linha 1')
plt.plot([3, 2, 1], label='Linha 2')

plt.legend(loc='upper right')  # legenda no canto superior direito
plt.show()
```

Opções de Localização da Legenda (`loc`)

- A posição da legenda pode ser controlada com o parâmetro `loc` da função `plt.legend()`.
- A tabela abaixo mostra 5 posições comuns:

Código	Descrição
`'upper right'`	Canto superior direito
`'upper left'`	Canto superior esquerdo
`'lower left'`	Canto inferior esquerdo
`'lower right'`	Canto inferior direito
`'center'`	Centro da área de plotagem

- Também é possível usar `loc='best'` para posicionamento automático.

Plotando com np.linspace e np.zeros_like

- `np.linspace(0, 2, 100)` cria 100 pontos igualmente espaçados entre 0 e 2.
- `np.zeros_like(x)` gera um array de zeros com o mesmo formato de `x`.
- Podemos usar ambos para construir múltiplas curvas no mesmo gráfico.

Exemplo em Python

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2, 100)
y = np.zeros_like(x)

plt.plot(x, y, 'g-')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.show()
```

Mais de uma curva

- Podemos construir múltiplas curvas no mesmo gráfico.

Exemplo em Python

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2, 100)
y1 = x**2
y2 = np.zeros_like(x)
plt.plot(x, y1, 'g-', label='y = x²')
plt.plot(x, y2, 'r--', label='y = 0')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.show()
```

Cuidado com a Escala dos Dados

- Funções com crescimento rápido (como `x^20`) .. (ou também outros dados) .. podem gerar distorções visuais ou comprimir outras curvas.
- No exemplo abaixo, a curva `y = x²` fica quase invisível perto de `y = x^20`.

Exemplo em Python

```
x = np.linspace(-2, 2, 100)
y1 = x**2
y2 = x**20
plt.plot(x, y1, 'g-', label='y = x²')
plt.plot(x, y2, 'r--', label='y = x^20')
plt.xlabel('x')
plt.ylabel('y')
plt.ticklabel_format(style='plain', axis='y')
plt.grid(True)
plt.legend()
plt.show()
```

Cuidado com o Recorte da Visualização

- Agora, considere o seguinte trecho: `plt.axis((-2, 2, 0, 4))`
- Pergunta: O que acontece com a curva `y = x^20` ao aplicar esse recorte?

Código de exemplo

```
x = np.linspace(-2, 2, 100)
y1 = x**2
y2 = x**20
plt.plot(x, y1, 'g-', label='y = x²')
plt.plot(x, y2, 'r--', label='y = x^20')
plt.xlabel('x')
plt.ylabel('y')
plt.ticklabel_format(style='plain', axis='y')
plt.axis((-2, 2, 0, 4))
plt.grid(True)
plt.legend()
plt.show()
```

O que acontece com `y = x^20`?

- A função `y = x^20` cresce rapidamente para `x` próximos de -2 e 2.
- Ao limitar o eixo y com `plt.axis((-2, 2, 0, 4))`, estamos forçando o gráfico a exibir apenas valores de y entre 0 e 4.
- Como `y = x^20` atinge valores muito maiores que 4 perto das extremidades, esses valores ficam **fora da área visível** e não aparecem no gráfico.
- O resultado é que a curva `y = x^20` parece **"ir pra fora"** da visualização.

Conclusão

Sempre verifique se os limites definidos com `plt.axis()` não estão escondendo partes importantes dos dados.

Usando Funções com `def` e `return`

- Definir funções matemáticas em Python com `def` para organizar melhor o código.
- Isso facilita a reutilização da função em diferentes contextos.

Exemplo em Python

```
def f(x):
    return x**2 + 2*x + 1  # função quadrática

x = np.linspace(-3, 3, 100)
y = f(x)
plt.plot(x, y, label='f(x) = x² + 2x + 1')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gráfico de uma Função')
plt.grid(True)
plt.legend()
plt.show()
```

Salvando Gráficos com `savefig`

- Podemos salvar o gráfico diretamente como uma imagem, sem exibi-lo na tela.
- O comando `plt.savefig("grafico.png", dpi=300)` salva a figura no arquivo `grafico.png`.
- O parâmetro `dpi` (dots per inch) define a resolução da imagem.
- É importante chamar `savefig()` **antes** de `plt.show()`.

Exemplo em Python

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gráfico de exemplo')
plt.savefig("grafico.png", dpi=300)  # Salva a figura
plt.show()  # Exibe a figura
```

Gráfico de Pizza com `plt.pie()`

- O método `plt.pie()` cria um gráfico de setores (pizza).
- A lista `sizes` define os tamanhos relativos de cada fatia.
- A lista `labels` define os rótulos das fatias.
- O parâmetro `autopct='%1.1f%%'` exibe os valores percentuais com uma casa decimal.
- `plt.axis('equal')` garante que o gráfico fique como um círculo (e não ovalado).

Exemplo em Python

```
import matplotlib.pyplot as plt

labels = ['A', 'B', 'C']
sizes = [40, 35, 25]

plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.axis('equal')  # Mantém formato circular
plt.show()
```

Cuidados com Gráficos de Pizza

- O que acontece aqui?

Código

```
labels = ['A', 'B', 'C']
sizes = [4312, 35, 25]

plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.axis('equal')
plt.show()
```

- Soma total: $4312 + 35 + 25 = 4372$
- Porcentagens:
  - A: $\frac{4312}{4372} \approx 98.6\%$
  - B: $\frac{35}{4372} \approx 0.8\%$
  - C: $\frac{25}{4372} \approx 0.6\%$
- As fatias B e C são praticamente invisíveis.

`plot()` vs `hist()`: Diferença Conceitual

- `plt.plot(...)` é usado para criar gráficos de linha, que mostram a relação entre pares de valores (como x e y).
  - Exibe como os valores mudam ao longo de um eixo.
  - Útil para funções matemáticas ou séries temporais.
- `plt.hist(...)` cria um histograma, que mostra a distribuição de frequências de um conjunto de dados.
  - Divide os dados em faixas (bins) e conta quantos valores caem em cada faixa.
  - Mostra **frequências absolutas** (ou relativas, com `density=True`).
- Ou seja:
  - `plot()` $\rightarrow$ representação de valores organizados.
  - `hist()` $\rightarrow$ distribuição dos dados brutos.

Criando um Histograma com `plt.hist()`

- Histograma mostra a distribuição de uma variável contínua dividida em *bins*.
- Exemplo: histograma com dados gerados aleatoriamente de uma distribuição normal.

Código em Python

```
data = np.random.normal(0, 1, 1000)
plt.hist(data, bins=30, edgecolor='black')
plt.xlabel('Valor')
plt.ylabel('Frequência')
plt.title('Histograma de Dados Aleatórios')
plt.grid(True)
plt.show()
```

- `np.random.normal(0, 1, 1000)` gera 1000 valores com média 0 e desvio padrão 1.
- `bins=30` define o número de intervalos.
- `edgecolor='black'` desenha contornos pretos nas barras para melhor visualização.

Histograma com Distribuição Exponencial

- Podemos visualizar dados de uma distribuição exponencial com `plt.hist()`.
- A distribuição exponencial é assimétrica e decresce rapidamente.

Código em Python

```
data = np.random.exponential(scale=1.0, size=1000)
plt.hist(data, bins=30, edgecolor='black')
plt.xlabel('Valor')
plt.ylabel('Frequência')
plt.title('Histograma de Distribuição Exponencial')
plt.grid(True)
plt.show()
```

- `np.random.exponential(scale=1.0, size=1000)` gera 1000 valores exponenciais com média 1.
- A maioria dos valores está concentrada próximo de 0, com uma cauda longa à direita.

Explore Dados Reais no `Kaggle`

- O **Kaggle** é uma plataforma online voltada para:
  - Ciência de dados,
  - Machine learning,
  - Competição com conjuntos de dados reais.
- É um excelente local para:
  - Encontrar conjuntos de dados interessantes,
  - Ver exemplos de visualização com `matplotlib`, `seaborn`, `pandas` etc.,
  - Praticar suas habilidades com notebooks Python.
- Acesse: `https://www.kaggle.com`

Dica

Navegue pelas abas `Datasets` e `Code` para se inspirar e experimentar!