Inatel

Minicurso Python

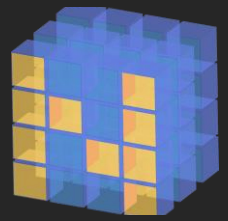
Ferramentas para T319/T320

Conteúdo do Curso

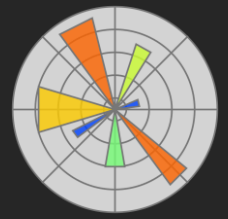
Introdução ao Python

- Um pouco sobre a linguagem
- Introdução ao Colab e Jupyter
- Tipos de dados
- Operadores aritméticos, relacionais e lógicos
- Controle de fluxo
- Laços de repetição
- Coleções de dados
- Funções
- Mensagens de erro

Análise e Processamento de Dados



Geração e análise de dados com a biblioteca Numpy



Visualização de dados com Matplotlib

Notebooks com os exemplos e exercícios

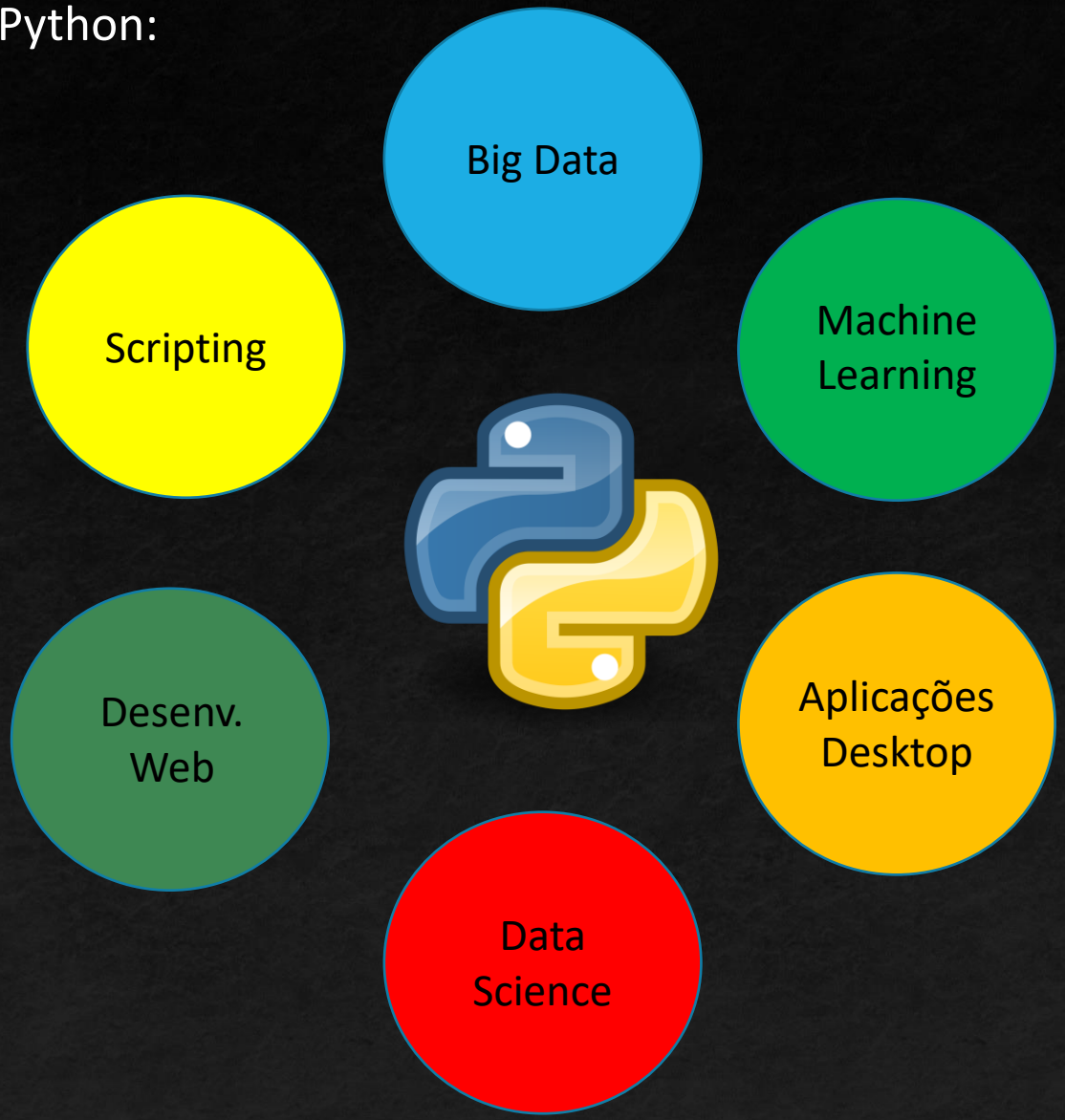
- O curso será dividido em 5 partes:
  - Parte I:
    - Tipos de dados e operadores
    - Notebook: ParteI_Aula
  - Parte II:
    - Controle de fluxo, laços de repetição e coleções de dados
    - Notebook: ParteII_Aula
  - Parte III:
    - Funções e mensagens de erro
    - Notebook: ParteIII_Aula
  - Parte IV:
    - Geração e análise de dados com a biblioteca Numpy
    - Notebook:
  - Parte V:
    - Visualização de dados com Matplotlib
    - Notebook:
- Link do repositório no GitHub: MinursoPythonGitHub

Introdução ao Python

Um pouco sobre a linguagem

- Criada em 1989 pelo matemático holandês Guido Van Rossum.
- Guido desenvolveu o Python enquanto trabalhava no instituto nacional de pesquisa em matemática e ciência da computação na Holanda - Centrum Wiskunde en Informatica(CWI) onde desenvolvia projetos com a linguagem ABC.
- Python é uma linguagem de programação:
  - Alto nível
  - Multi-paradigma
  - Multiplataforma
  - Dinâmica
  - Interpretada
  - Gratuita
  - Código aberto

- Aplicações com Python:



- Algumas empresas que utilizam Python:

Introdução ao Colab e Jupyter

- Google Colab e Jupyter são “ambientes computacionais interativos baseados em aplicações web para a criação de documentos virtuais”.
- Os notebooks do Colab/Jupyter são documentos que permitem combinar código executável e rich text, além de imagens, HTML, LaTeX e etc.
- “Jupyter is the open-source project on which Colab is based. Colab allows you to use and share Jupyter notebooks with others without having to download, install, or run anything.”
- Links:
  - Site do Colab: https://colab.research.google.com/
  - Tutorial Colab: TutorialColab
  - Site Jupyter: https://jupyter.org/
  - Tutorial de instalação Jupyter: InstalacaoJupyter (prof. Felipe Augusto)

VS

Colab	Jupyter
▪ Linguagem Python	▪ Python, C++, Julia, R entre outras
▪ Executado somente na nuvem	▪ Pode ser executado na nuvem ou localmente
▪ Maior número de servidores	▪ Menor número de servidores
▪ Pode se conectar com o Google Drive	
▪ Acesso a GPUs e TPUs	

- Elementos de um notebook:

Tipos de dados

- Python é uma linguagem dinâmica, portanto o tipo de dado é inferido em tempo de execução.
- Outra característica é de uma linguagem fortemente tipada, assim o interpretador não converte automaticamente dados incompatíveis em operações. Ex: não permite a soma de uma string e um valor int.
- Função type() retorna o tipo de dado.

Tipos de dados em Python		
Categoria	Nome	Descrição
Numérica	int	Inteiros
	float	Ponto flutuante
	complex	Número complexo
	bool	Booleano
Sequencial	str	String
	list	Lista
	tuple	Tupla
	range	Intervalo de valores
Conjunto	set	Conjunto
	frozenset	Conjunto imutável
Mapeamento	dict	Dicionário
Nula	NoneType	Valor Nulo

Operadores Aritméticos

- Utilizados para realizar operações matemáticas.

Operador	Nome	Descrição	Exemplo	Resultado
+	Adição	Realiza a soma	2 + 4	6
-	Subtração	Realiza a subtração	5 – 3	2
*	Multiplicação	Realiza a multiplicação	4 * 2	8
/	Divisão	Realiza a divisão	10 / 5	2.0
//	Divisão inteira	Retorna a parte inteira da divisão	10 // 6	1
%	Módulo	Retorna o resto da divisão	5 % 2	1
**	Exponenciação	Retorno um número elevado à potência de outro	2 ** 3	8

Operadores Relacionais ou de Comparação

- Utilizados para comparar valores.

Operador	Nome	Descrição	Exemplo	Resultado
==	Igual a	Verifica se um valor é igual ao outro	10 == 10	True
!=	Diferente de	Verifica se os valores são diferentes	2 != 6	True
>	Maior que	Verifica se um valor é maior que o outro	5 > 8	False
<	Menor que	Verifica se um valor é menor que o outro	6 < 12	True
>=	Maior ou igual a	Verifica se um valor é maior ou igual ao outro	3 >= 3	True
<=	Menor ou igual a	Verifica se um valor é menor ou igual ao outro	7 <= 1	False

Operadores Lógicos

- Utilizados para realizar operações com valores booleanos.

Operador	Nome	Descrição	Exemplo	Resultado
and	E lógico	Retorna True se todas as condições forem verdadeiras e False caso contrário	True and False	False
or	Ou lógico	Retorna True se pelo menos uma condição for verdadeira e False caso contrário	False or True	True
not	Negação lógica	Retorna o valor negado	not False	True

Ordem de precedência dos operadores

Precedência	Nome	Operadores	Tipo
Alta  Baixa	Parênteses	()	
	Expoente	**	Operadores aritméticos
	Multiplicação, divisão, divisão inteira e resto da divisão	* / // %	
	Adição e subtração	+ -	
	Menor ou igual, menor, maior e maior ou igual	<= < > >=	Operadores relacionais
	Igual a e diferente de	== !=	
	Negação lógica	not	Operadores lógicos
	E lógico	and	
	Ou lógico	or	

- Operadores com o mesmo nível de precedência são aplicados da esquerda para direita na ordem em que aparecem na expressão.
  - Ex.:

```
1 resultado = 3 * 9 / 2 % 2
2 print('Resultado = ', resultado)
```

```
Resultado =  1.5
```

- As expressões de potenciação são realizadas da direita para esquerda.
  - Ex.:

```
1 expressao1 = 2 ** 1 ** 2
2 expressao2 = (2 ** 1) ** 2
3
4 print('Resultado sem parênteses = ', expressao1)
5 print('Resultado com parênteses = ', expressao2)
```

```
Resultado sem parênteses =  2
Resultado com parênteses =  4
```

Controle de fluxo

- Estruturas condicionais ou de controle de fluxo são usadas para verificar o resultado de expressões para executar ou não uma sequência de comandos.

```
# Verifique se a condição1 é verdadeira
if <condição1>:
  # Se for verdadeira, execute esse bloco de código
  <bloco de código1>
# Senão, verifique a condição2
elif condição2:
  # Se condição2 for verdadeira, execute esse bloco
  <bloco de código2>
# Se nenhuma das condições anteriores for verdadeira
else:
  # Execute esse bloco
  <bloco de código3>
```

- <condição> : é a expressão a ser avaliada como verdadeira ou falsa
- <bloco de código> : linhas de comando a serem executadas
- elif é uma abreviatura de else if
- Importante:
  - Não se esquecer dos dois pontos ‘:’ depois da condição e do else.
  - Atente-se a indentação!

- Exemplo:

```
1 valor = 500 # Valor de um produto em reais
2
3 if valor > 200:
4   print("Produto caro demais!!!")
5 elif valor >= 150:
6   print("Ainda está caro!")
7 elif valor >= 100:
8   print("O preço é bom, mas pode melhorar!")
9 else:
10  print("Ótimo preço!!!")
```

```
Produto caro demais!!!
```

Laços de repetição

- São estruturas de repetição usadas para percorrer e processar elementos de uma lista, uma tupla, letras de uma string, linhas de um arquivo e etc.
- Em Python há dois tipos: for e while.
- for:
  - Estrutura em que número de repetições é explicitamente definido.

```
1 for <variável> in <objeto iterável>:
2   <bloco de código>
```

  - <objeto iterável> : objeto a ser percorrido
  - <variável> : armazena os elementos a cada iteração/repetição do código
  - <bloco de código> : linhas de comando
  - Para interromper o laço antes de percorrer todos os elementos, adiciona-se o comando ‘break’

- while:
  - Executa um bloco de código enquanto uma condição for verdadeira.

```
1 while <condição>:
2   <bloco de código>
```

  - <condição> : expressão a ser verificada
  - <bloco de código> : linhas de comando
  - Assim como no for, é possível interromper o laço com o comando ‘break’
  - Importante:
    - Loop infinito: se não for implementada uma parte do código em que a condição se torne falsa, o programa ficará preso nessa laço ou loop para sempre.

Coleções de dados

- São estruturas de dados que armazenam vários objetos/elementos, do mesmo ou de tipos diferentes.
- Em Python, as principais coleções de dados são:
  - Lista
  - Tupla
  - Dicionário
  - Conjunto
- Neste minicurso, somente as listas e tuplas serão estudadas.

Listas

- São estruturas:
  - Ordenadas: os elementos possuem uma ordem
  - Heterogêneas: é possível armazenar diferentes tipos de dados em uma mesma lista
  - Mutáveis: os elementos podem ser alterados
  - Indexáveis: é possível acessar os valores por meio de um índice
  - Podem ser fatiadas: é possível dividir as listas em subconjuntos
- Como criar uma lista:
  - As listas em Python são representadas por colchetes em que os elementos vem separados por vírgulas
  - Exemplo:

```
1 nomes = ['Ana', 'João', 'Maria', 'José', 'Joaquim'] # Lista de nomes
2 print(nomes) # Conteúdo da lista
3 print(type(nomes)) # Tipo da variável
```

```
['Ana', 'João', 'Maria', 'José', 'Joaquim']
<class 'list'>
```

Tuplas

- São estruturas:
  - Ordenadas: os elementos possuem uma ordem
  - Heterogêneas: é possível armazenar diferentes tipos de dados em uma mesma lista
  - Imutáveis: os elementos não podem ser alterados. A alteração de um elemento gera uma nova tupla
  - Indexáveis: é possível acessar os valores por meio de um índices
- Como criar uma tupla:
  - As tuplas em Python são representadas por parênteses em que os elementos vem separados por vírgulas
  - Exemplo:

```
1 # Tupla simples
2 tupla = (1, 2, 3, 4, 5)
3 print(tupla) # Conteudo da tupla
4 print(type(tupla)) # Tipo da variável
```

```
(1, 2, 3, 4, 5)
<class 'tuple'>
```

Funções

- Funções são blocos de códigos que realizam tarefas específicas.
- Utilizadas quando uma tarefa é realizada repetidas vezes e em diversas parte do programa.
- Como definir uma função:

```
1 def nomeFuncao(parametro1, parametro2, ....):
2   """Escreva aqui qual a tarefa a função executa"""
3   <bloco de código>
```

  - Utiliza-se def para definir a função, seguido do nome dessa função e entre parênteses, os parâmetros.
  - Não se esqueça dos dois pontos ‘:’
  - É possível adicionar um texto logo após a definição para descrever a função. Esse texto é chamado de docstring, é colocado entre 3 aspas simples ou duplas e pode ser acessado com o seguinte comando:

```
1 nomeFuncao.__doc__
```

```
'Escreva aqui qual a tarefa a função executa'
```

Tipos de funções

- Sem parâmetro e sem retorno:

```
1 def hello():
2   print('Hello world!')
3
4 hello()
```

```
Hello world!
```

- Com um único parâmetro e sem retorno:

```
1 def somaUm(x):
2   """ Função imprime o valor de x somado a 1"""
3   print("Resultado = ", x + 1)
4
5 # Verificando o que a função faz
6 print(somaUm.__doc__)
7
8 # Chamando a função
9 somaUm(10)
```

```
 Função imprime o valor de x somado a 1
Resultado =  11
```

- Com múltiplos parâmetros e com retorno:
  - Para retornar uma valor, coloque-o depois de return

```
1 def novoSalario(salario, aumento):
2   novo_salario = salario + salario*aumento
3   return novo_salario
4
5 meu_salario = novoSalario(2000, 0.1)
6 print('Meu novo salário é igual a R$ ', meu_salario)
```

```
Meu novo salário é igual a R$  2200.0
```

- Chamando a função indicando o nome e o valor do parâmetro:
  - Para deixar explícito o parâmetro a ser definido, coloque nome=valor

```
1 meu_salario = novoSalario(salario=1000, aumento=0.2)
2 print('Meu novo salário é igual a R$ ', meu_salario)
```

```
Meu novo salário é igual a R$  1200.0
```

- Com retorno de múltiplos valores:
  - Para retornar múltiplos valores, coloque-os entre vírgulas depois de return

```python
# Função retornando mais de um valor
def operacoes(x, y):
  soma = x + y
  media = (x + y)/2
  produto = x*y

  return soma, media, produto

soma, media, produto = operacoes(2, 4)
print('Soma = ', soma)
print('Média = ', media)
print('Produto = ', produto)
```

```
Soma =  6
Média =  3.0
Produto =  8
```

- Parâmetros com valor padrão/default:
  - Nesse exemplo, o parâmetro expoente foi definido com valor padrão igual a 2. Portanto, não será obrigatório defini-lo ao chamar a função.

```
1 def potenciacao(base, expoente=2):
2   return base**expoente
3
4 print('Resultado com apenas 1 parâmetro definido = ', potenciacao(4))
5 print('Resultado com os dois parâmetros definidos = ', potenciacao(4, 3))
```

```
Resultado 1 =  16
Resultado 2 =  64
```

  - Importante:
    - É necessário definir os parâmetros com valor padrão após os que não possuem. Caso contrário, será apresentado um erro.

- Com número arbitrário de parâmetros:
  - Para definir uma função com a quantidade de argumentos indefinida, basta colocar um asterisco(*) antes do parâmetro.
  - Os valores serão passados com uma tupla para a função.

```
1 def programadores(*nomes):
2   print("O nome dos programadores são: ", nomes)
3   print("O último programador é: ", nomes[-1])
4
5 # Chamando a função com 3 argumentos
6 print('Função com 3 argumentos')
7 programadores('Marcos', 'Ana', 'Priscila')
8
9 # Chamando a função com 2 argumentos
10 print('\nFunção com 2 argumentos')
11 programadores('Pedro', 'Rafael')
```

```
Função com 3 argumentos
O nome dos programadores são:  ('Marcos', 'Ana', 'Priscila')
O último programador é:  Priscila

Função com 2 argumentos
O nome dos programadores são:  ('Pedro', 'Rafael')
O último programador é:  Rafael
```

- Com recursão:
  - É uma função que chama a si mesma.

```python
def fatorial(n):
  if n == 0:
    return 1
  else:
    return n*fatorial(n-1)

a = 6
print(f'{a}! = ', fatorial(a))
```

```
6! =  720
```

- Funções built-in:
  - Funções que já vêm incorporadas na linguagem.
  - Estão disponíveis a qualquer momento e não necessitam de nenhuma importação.
  - Segue um link com a lista dessas funções: https://docs.python.org/pt-br/3.6/library/functions.html

Mensagens de erro

- As mensagens de erro são importantes pois indicam possíveis falhas no algoritmo. Elas indicam o que está ocorrendo e onde, por isso é necessário ler com calma.
- Pode-se dizer que há dois tipos de erros, os de sintaxe e as exceções.
- Erros de sintaxe:
  - Erro bastante comum.
  - Ocorre quando um símbolo foi esquecido ou colocado de forma incorreta.
  - É indicado por uma seta.

```
1 print('Exemplo erro de sintaxe)
```

```
  File "<ipython-input-35-58e6e5809fee>", line 1
    print('Exemplo erro de sintaxe)
                                   ^
SyntaxError: EOL while scanning string literal
```

SEARCH STACK OVERFLOW

- Exceções:
  - Erros que ocorrem em tempo de execução.
  - Exemplos:
    - ZeroDivisionError: divisão por zero.

```
1 print(10 / (2%2))
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-39-c169372e5c7a> in <module>()
----> 1 print(10 / (2%2))

ZeroDivisionError: division by zero
```

SEARCH STACK OVERFLOW

    - TypeError: operações com tipos de dados incompatíveis.

```
1 a = '10'
2 b = 10
3
4 soma = a + b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-17c1c1478836> in <module>()
      2 b = 10
      3
----> 4 soma = a + b

TypeError: can only concatenate str (not "int") to str
```

SEARCH STACK OVERFLOW

- NameError: variável não declarada.

```
1 print('Olá ', pessoa)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-40-e674d0c813b1> in <module>()
----> 1 print('Olá ', pessoa)

NameError: name 'pessoa' is not defined
```

SEARCH STACK OVERFLOW

- IdentationError: erro de indentação.

```
1 def soma(a, b):
2 return a + b
```

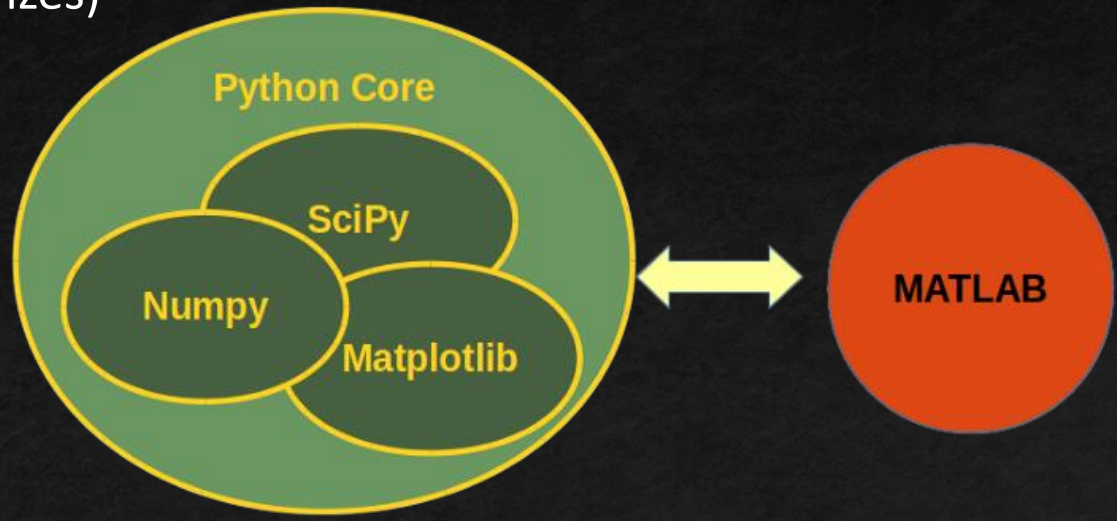
```
  File "<ipython-input-37-042a3ccb7b6e>", line 2
    return a + b
         ^
IndentationError: expected an indented block
```

SEARCH STACK OVERFLOW

Análise e Processamento de Dados

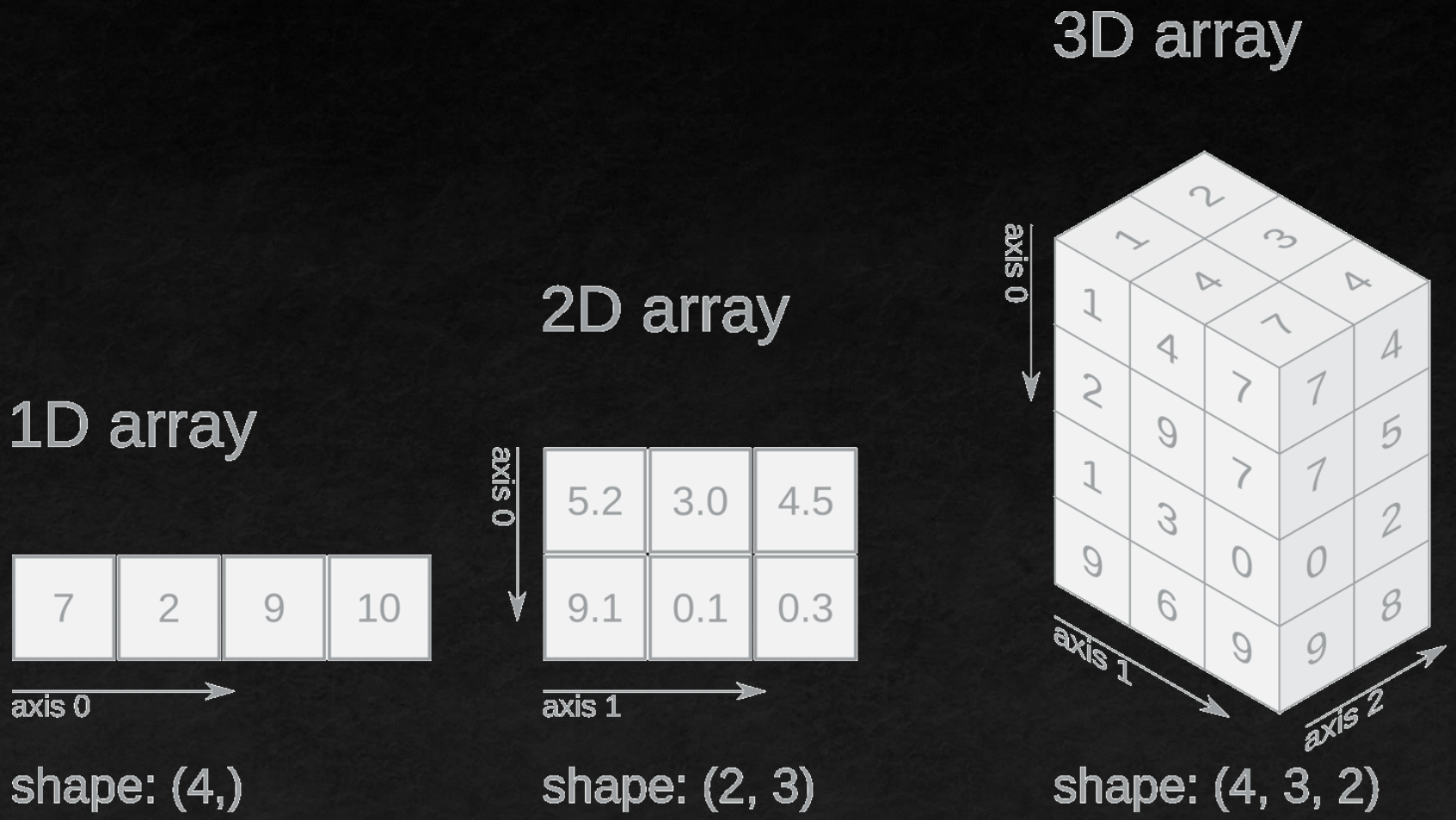
Geração e análise de dados com a biblioteca Numpy

- Numpy é uma biblioteca com funções para se trabalhar com computação numérica.
- Seu principal objeto são os arrays(como vetores e matrizes) multidimensionais.
- Usada em diversas tarefas como:
  - No treinamento de modelos de machine learning
  - Processamento de imagem e computação gráfica
  - Álgebra linear
  - Geração de números aleatórios



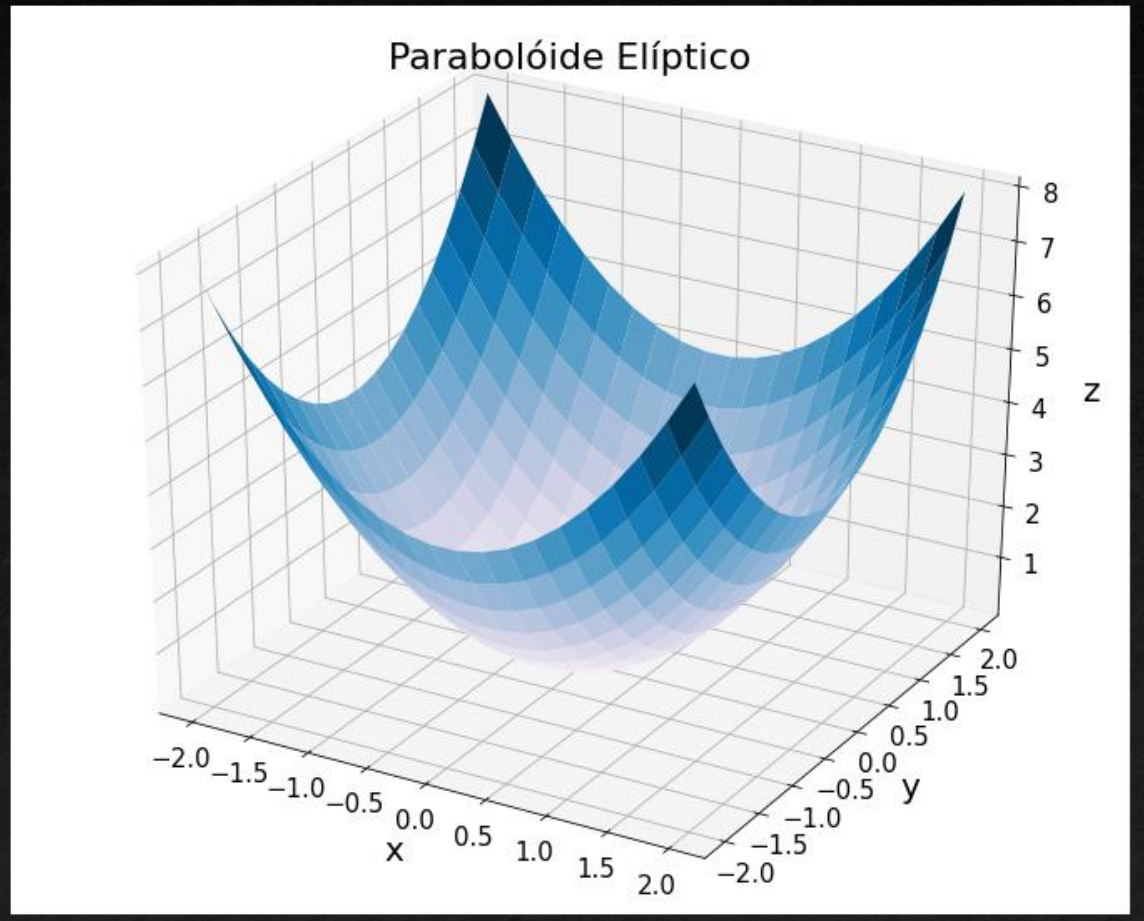
- Documentação: https://numpy.org/doc/stable/reference/index.html

- O que são os arrays?

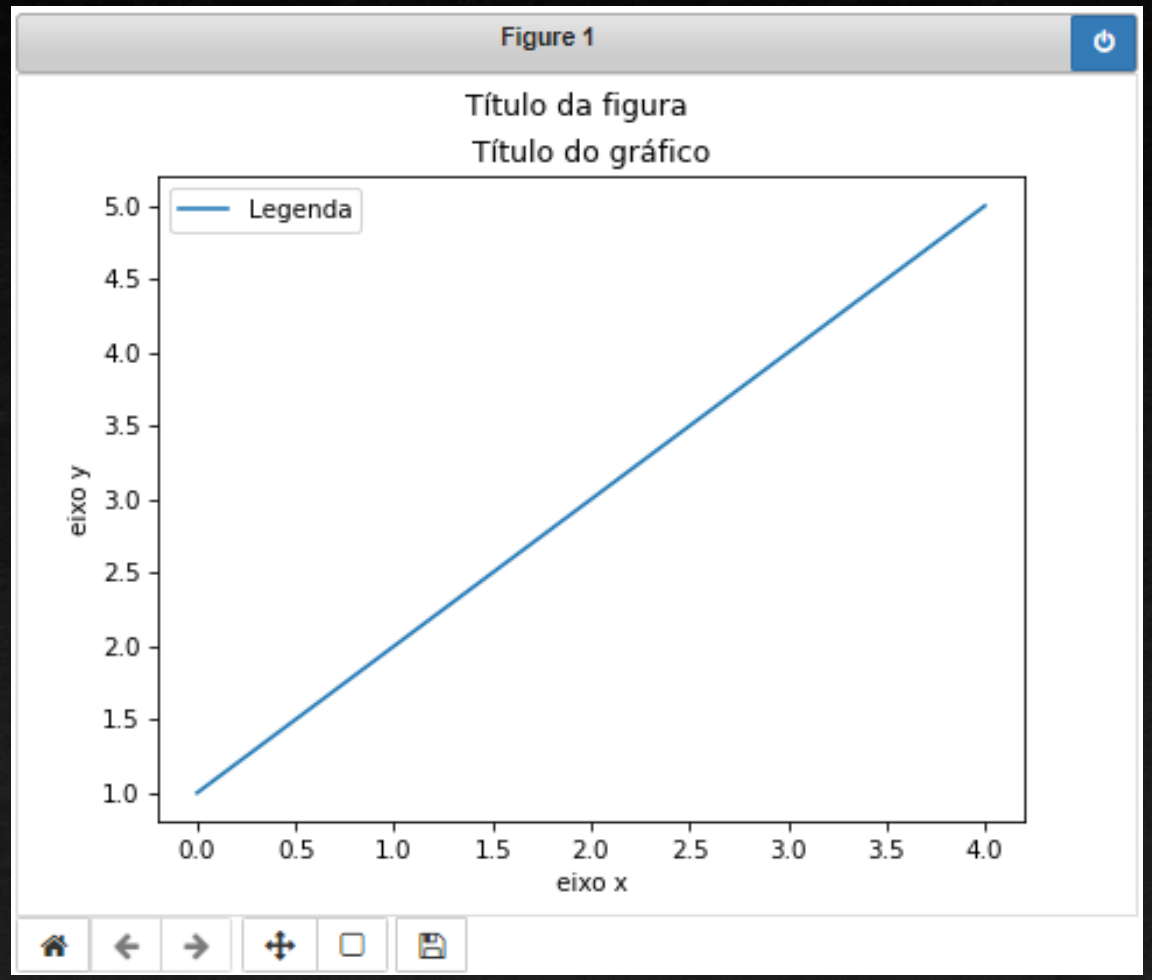


Visualização de dados com Matplotlib

- O Matplotlib é uma biblioteca para construção de gráficos em Python.
- É possível construir:
  - Gráficos 2D
  - Gráficos 3D
  - Histogramas
  - Gráficos de pizza
  - Superfícies de contorno e etc.
- Documentação: https://matplotlib.org/



- Principais componentes de um gráfico:



Agradeço pela atenção!

Bruna de Souza Ribeiro
bruna.br@gea.inatel.br
www.linkedin.com/in/bruna-sribeiro