

---

---

# Inteligência Artificial

— Prof. MSc. Bruno Santos —  
email: [bruno.santos@saojudas.br](mailto:bruno.santos@saojudas.br)

---

---

# OBJETIVOS da AULA

- Apresentação do projeto de desempenho;
- Fundamentos de lógica de programação; e
- Introdução ao Python.

# PROJETO DE DESEMPENHO

- COMITÊ DE CLASSIFICADORES



# PROJETO DE DESEMPENHO

Inferência: raciocínio concluído a partir de premissas



# PROJETO DE DESEMPENHO

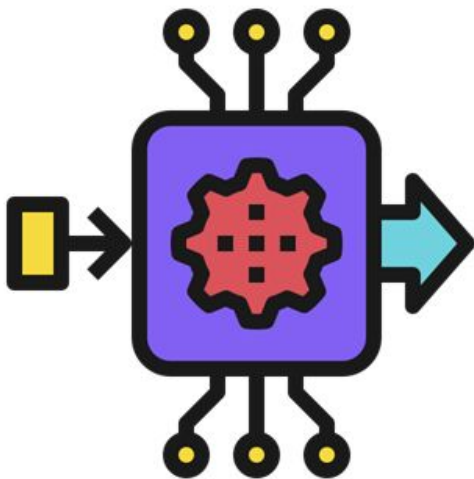
## Tópicos da base de dados Census

- age;
- workclass;
- final-weight;
- education;
- education-num;
- marital-status;
- occupation;
- relationship;
- race;
- sex;
- capital-gain;
- capital-loss;
- hours-per-week;
- native-country; e
- income.

# PROJETO DE DESEMPENHO

Podemos inferir um atributo específico com base nos demais atributos.

- age;
- workclass;
- final-weight;
- education;
- ...



- income

# PROJETO DE DESEMPENHO

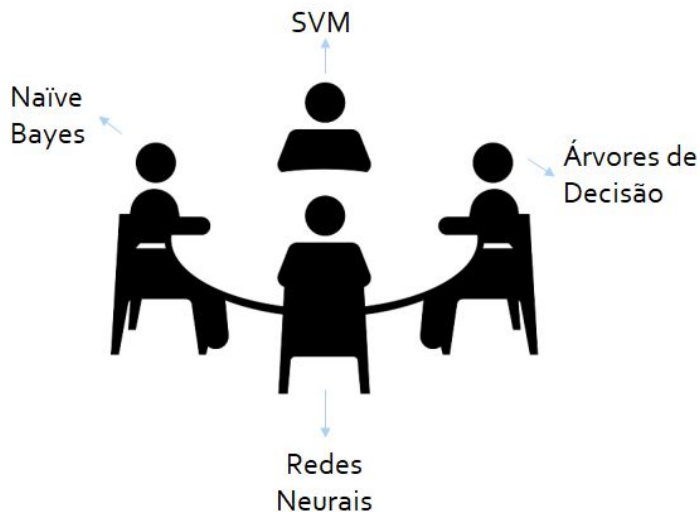
Como fazer a inferência com IA?



- Fuzzy;
- Naive Bayes;
- Árvores de Decisão;
- Redes Neurais;
- ...

# PROJETO DE DESEMPENHO

## O que é um comitê?



### Resultado: <50k

	<50k	>= 50k
SVM	✓	
Naïve Bayes		✓
Árvores de Decisão	✓	
Redes Neurais	✓	



# DÚVIDAS



# Lógica é “a arte de pensar”

A ideia do **pensamento lógico** é um pensamento corrigido, organizado e correto – **pensar de forma racional**, considerando o que é certo e errado, quais ações são as melhores e quais são as piores para alcançar determinado objetivo.

**Lógica da programação** é aplicar a lógica para organizar e estruturar programas de computador (algoritmos).

**Algoritmo** pode ser definido como uma sequência de ações para alcançar um determinado objetivo.

# Lógica é “a arte de pensar”

A lógica é necessária para desenvolver algoritmos porque estes precisam ser ordenados, precisos, com ações claras e objetivas, de forma que todo o seu desenvolvimento e seu resultado sejam previsíveis; ou seja, os **algoritmos** devem seguir uma **lógica**.

# DÚVIDAS



# ESTRUTURAS DE LÓGICA DA PROGRAMAÇÃO

Pode parecer irrelevante, mas, quando vamos tratar de coisas mais complicadas, os algoritmos são fundamentais para, não só organizar o programa, como organizar **nosso próprio pensamento**.

Podemos ter vários algoritmos para um mesmo objetivo – na verdade, é quase impossível que exista apenas um algoritmo para um objetivo.

# Primitivos

- **Tipo Inteiro** (código/comando: int) – Compreende os valores do conjunto matemático dos números inteiros, por exemplo -1, 0, 1, -50, 30;
- **Tipo Real ou Ponto Flutuante** (código/comando: float) – São os valores do conjunto dos números reais, ou, grosso modo, os que são escritos com vírgula, por exemplo “5.3”, “-0,3359”, “10.00” (por ter vírgula, mesmo que o número depois da vírgula seja apenas zero, ainda é um ponto flutuante);

# Primitivos

- **Tipo Caractere** (código/comando: char) – O tipo char é usado para representar caracteres, “a”, “b”, “x”. Os números podem ser considerados também, mas, neste caso, eles não possuem valor matemático;
- **Tipo Lógico ou Booleano** (código/comando: bool) – São os dados que possuem apenas dois valores distintos: verdadeiro ou falso.

Para armazenar os valores dos tipos (float, int, char, bool), usamos uma coisa chamada variável, que é o elemento que o computador usa para **armazenar os valores**. O nome é “variável”, porque o valor que ele **guarda** pode **mudar** ao longo da **execução do código** – O SEU VALOR, e não o seu tipo.

Reformulando nosso pseudocódigo, temos algo assim:

```
int preço do produto;  
int total de dinheiro do jogador;  
PegaPreço(preço do produto);  
PegaDinheiro(total de dinheiro do jogador);  
  
Se (total de dinheiro do jogador < preço do produto) -> aqui ocorre um caso booleano, apenas dois valores existentes  
    Exibir (Jogador não tem dinheiro suficiente);  
Se não  
    Total de dinheiro do jogador = total de dinheiro do jogador - preço do produto;  
    Exibir (Produto “x” adquirido);
```



# Variáveis

As **variáveis** armazenam e fazem operações apenas com valores compatíveis ao seu tipo.

Traduzindo, a adição de um valor de uma variável do tipo int com uma variável do tipo float, ocorrerá sem problemas (como uma subtração, caso o resultado tenha número com vírgula, se o resultado for armazenado em uma variável int a linguagem irá arredondar o valor, se for float o resultado permanece inalterado).

Agora, se for feita essa operação com um tipo char, não sairá como esperado, sendo difícil prever o que acontecerá.

# Operadores

As operações mais comuns entre as linguagens de programação, são separados em:

- Operadores Aritméticos- as operações matemáticas conhecidas: adição (+), subtração (-), multiplicação (\*) e divisão (/); e
- Operadores Relacionais- são operadores usados para comparação entre os valores armazenados pelas variáveis. Por definição o resultado dos operadores é lógico (verdadeiro ou falso).

# Operadores relacionais

Símbolo	Nome do Operador	Exemplo	Significado
>	Maior que	$x > y$	x é maior que y?
>=	Maior ou igual	$x \geq y$	x é maior ou igual a y ?
<	Menor que	$x < y$	x é menor que y?
<=	Menor ou igual	$x \leq y$	x é menor ou igual a y ?
==	Igualdade	$x == y$	x é igual a y?
!=	Diferente de	$x != y$	x é diferente de y?

# Operadores lógicos

Servem para combinar expressões, geralmente formadas pelos operadores relacionais ( $x < y$  E  $x > z$ ).

Operador	Ação
&&	And (E)
	Or (Ou)
!	Not (Não)

# DÚVIDAS



# PYTHON PARA IA

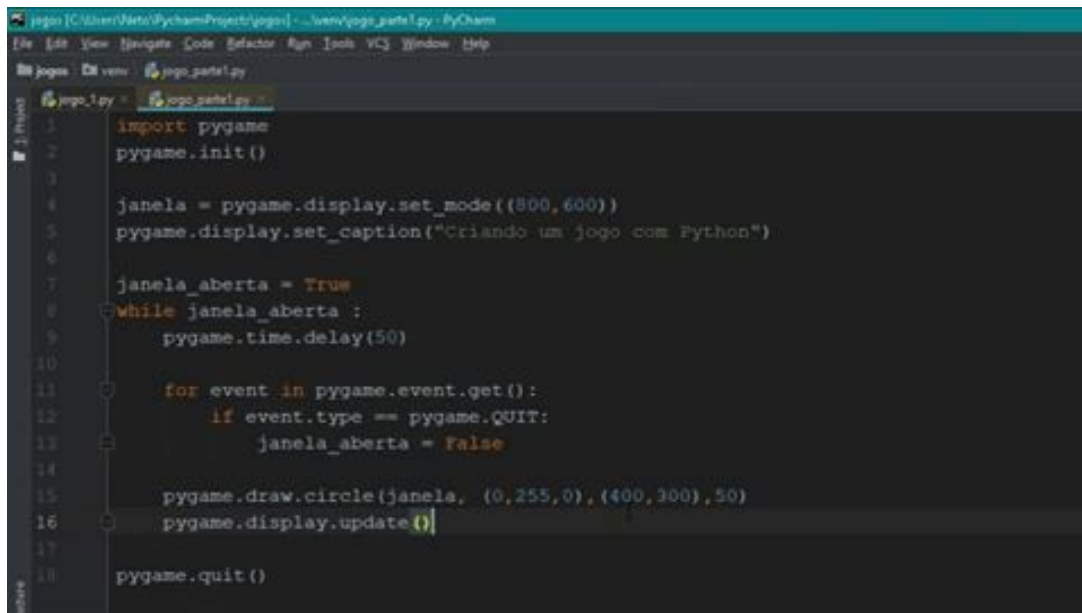
Python é uma linguagem de propósito geral usada para as mais diversas aplicações. Atualmente Python é uma das linguagens mais utilizadas no mundo dada a sua diversidade de uso e o número enorme de bibliotecas disponíveis para as mais diversas áreas.

Uma das área em que Python é muito utilizado é a Inteligência Artificial (IA) principalmente no contexto da Aprendizagem de Máquina e Ciência de Dados.

# PYTHON PARA IA

Nosso objetivo aqui no laboratório não vai ser destrinchar todos os conhecimentos sobre Python, mas sim apresentar os conceitos importantes de Python que os permitam utilizar todo o ferramental de IA disponível principalmente no que diz respeito à manipulação tabelas e arquivos e estruturas de dados de maneira que você consiga manipular o conjunto de dados que você quer analisar.

# IDE de PYTHON: COLAOB do Google



The image shows a screenshot of the PyCharm IDE interface. The title bar indicates the project is 'jogo' and the file being edited is 'jogo\_parte1.py'. The menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The toolbar shows icons for saving, running, and other IDE functions. The main editor area displays the following Python code:

```
1 import pygame
2 pygame.init()
3
4 janela = pygame.display.set_mode((800,600))
5 pygame.display.set_caption("Criando um jogo com Python")
6
7 janela_aberta = True
8 while janela_aberta :
9     pygame.time.delay(50)
10
11     for event in pygame.event.get():
12         if event.type == pygame.QUIT:
13             janela_aberta = False
14
15     pygame.draw.circle(janela, (0,255,0), (400,300),50)
16     pygame.display.update()
17
18 pygame.quit()
```

[LINK](#)





# ATIVIDADE 1 - Operadores aritméticos

```
1 print("hello world")
2
3 x = 3
4 print(type(x)) # Mostra "<class 'int'>"
5 print(x) # Mostra "3"
6 print(x + 1) # Adiciona e mostra "4"
7 print(x - 1) # Subtrai e mostra "2"
8 print(x * 2) # Multiplica e mostra "6"
9 print(x ** 2) # Faz a exponenciação e mostra "9"
10 x += 1
11 print(x) # Mostra "4"
12 x *= 2
13 print(x) # Mostra "8"
14 y = 2.5
15 print(type(y)) # Mostra "<class 'float'>"
16 print(y, y + 1, y * 2, y ** 2) # Mostra "2.5 3.5 5.0 6.25"
```

```
1 print("hello world")
2
3 x = 3
4 print(type(x)) # Mostra "<class 'int'>"
5 print(x) # Mostra "3"
6 print(x + 1) # Adiciona e mostra "4"
7 print(x - 1) # Subtrai e mostra "2"
8 print(x * 2) # Multiplica e mostra "6"
9 print(x ** 2) # Faz a exponenciação e mostra "9"
10 x += 1
11 print(x) # Mostra "4"
12 x *= 2
13 print(x) # Mostra "8"
14 y = 2.5
15 print(type(y)) # Mostra "<class 'float'>"
16 print(y, y + 1, y * 2, y ** 2) # Mostra "2.5 3.5 5.0 6.25"
```



## ATIVIDADE 3 - *Strings*

```
1 nome = 'Hanamichi Sakuragi'
2 print(nome)
3 n = 'Sakuragi'
4 sn = 'Hanamichi'
5 nc = n + ' ' + sn
6 print(nc)
```



```
1  nome = 'Hanamichi Sakuragi'
2  print(nome)
3  n = 'Sakuragi'
4  sn = 'Hanamichi'
5  nc = n + ' ' + sn
6  print(nc)
```

# DÚVIDAS





# FUNÇÕES

O uso de funções permite incluir na análise de dados estruturas de condição e iteração.

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

OU

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

OU

```
1 soma = lambda a,b:a+b
```

---

```
3 resposta = soma(35,4)  
4 print(resposta)
```



# ENTRADAS DE DADOS

Um dos recursos que as vezes precisamos considerar é o de receber valores de quem está usando a aplicação.

```
1 def leitura():  
2     nome = input("Digite seu nome:")  
3     sobrenome = input("Digite seu sobrenome:")  
4     nomecompleto = nome+" "+sobrenome  
5     print("seu nome completo eh: ",nomecompleto)
```

---

```
7 leitura()
```



# Entrada de dados - int

```
1 def potencia(a,b):  
2     c=a**b  
3     return c
```

---

```
5 x = int(input("Digite um valor:"))  
6 y = int(input("Digite o valor de potência:"))  
7 print(potencia(x,y))
```





# Entrada de dados - float

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

---

```
5 x = float(input("Digite um valor:"))  
6 y = float(input("Digite um valor:"))  
7 print(soma(x,y))
```

# ESTRUTURAS DE DECISÃO (CONDICIONAIS)

As estruturas de decisão permitem alterar o fluxo de execução de um programa, percorrendo um ou outro conjunto de instruções de acordo com o valor (Verdadeiro/Falso) de um teste lógico.

Em Python temos as estruturas de decisão “se” (if), “se/senão” (if..else) e “se/senão se/senão” (if..elif..else).

# Condicionais - IF

A instrução if é utilizado quando precisamos decidir se um trecho do programa deve ou não ser executado. Ele é associado a uma condição, e o trecho de código será executado se o valor da condição for verdadeiro.

Sintaxe:

```
if <condição>:  
    <instruções>
```

# Condicionais - IF



```
1 nota = float(input("Digite sua nota na UC:"))
```

```
3 if nota >= 7.0:  
4     print('Aprovado!')
```

# Condicionais - IF..ELSE

Na instrução if..else um trecho de código será executado se a condição for verdadeira e outro se a condição for falsa.

Sintaxe:

```
if <condição1>:  
    <instruções1>  
else :  
    <instruções2>
```

# Condicionais - IF..ELSE



```
1 nota = float(input("Digite sua nota na UC:"))
```

```
3 if nota >= 7.0:  
4     print('Aprovado!')  
5 else:  
6     print('Reprovado!')
```

# Condicionais - IF..ELIF..ELSE

A instrução if..elif..else é usada quando houver diversas condições, cada uma associada a um trecho de código.

Sintaxe:

```
if <condição1>:  
    <instruções1>  
elif <condição2>:  
    <instruções2>  
elif <condição3>:  
    <instruções3>  
...  
else :  
    <instruçõesN>
```

# Condicionais - IF..ELIF..ELSE



```
1  idade = int(input("Digite sua idade:"))
```

```
3  if idade < 3:  
4      print('Bebê')
```

```
6  elif idade < 10:  
7      print('Infantil')  
8  elif idade < 14:  
9      print('Junior')  
10 elif idade < 18:  
11     print('Adolescente')  
12 elif idade < 30:  
13     print('Jovem')
```

```
15 else:  
16     print('Adulto')
```



# ATIVIDADE 1



```
1 def categoria(idade):
2     if idade < 3:
3         ctg = 'Bebê'
4     elif idade < 10:
5         ctg = 'Infantil'
6     elif idade < 14:
7         ctg = 'Junior'
8     elif idade < 18:
9         ctg = 'Adolescente'
10    elif idade < 30:
11        ctg = 'Jovem'
12    else:
13        ctg = 'Adulto'
14    return ctg
```

```
16 idd = int(input("Digite sua idade:"))
17 print("Sua categoria é:", categoria(idd))
```

# ESTRUTURAS DE REPETIÇÃO

A Estrutura de repetição também conhecida como “loop” é utilizada para executar uma sequência de comandos por várias vezes.

A repetição está associada ou a uma condição, que indica se deve continuar a repetição, ou a uma sequência de valores, que determina quantas vezes a sequência deve ser repetida.

# Laço WHILE

No laço while, o trecho de código da repetição está associado a uma condição. Enquanto a condição tiver valor verdadeiro, o trecho é executado. Quando a condição passa a ter valor falso, a repetição termina.

Sintaxe:

```
while <condição>:  
    <instruções>
```

# Laço WHILE



```
1 senha = "54321"
```

```
3 leitura = " "
```

```
5 while (leitura != senha):  
6     leitura = input("Digite a senha: ")  
7     if leitura == senha : print('Acesso liberado ')  
8     else: print('Senha incorreta. Tente novamente')
```

# Laço WHILE



```
1 contador = 0
2 somador = 0
```

```
4 while contador < 5:
5     contador = contador + 1
6     valor = float(input('Digite o ' + str(contador) + 'º valor: '))
7     somador = somador + valor
```

```
9 print('Soma = ', somador)
```

# Laço FOR

O laço for é a estrutura de repetição mais utilizada em Python. Pode ser utilizado com uma sequência numérica (gerada com o comando range) ou associado a uma lista. O trecho de código da repetição é executado para cada valor da sequência numérica ou da lista.

Sintaxe:

```
for <variável> in range (<início>, <limite>, <passo>):  
    <instruções>
```

ou

```
for <variável> in <lista>:  
    <instruções>
```

# Laço FOR



```
1  # Encontrar a soma S = 1+4+7+10+13+16+19
```

```
3  S=0
```

```
5  for x in range(1,20,3):  
6      S = S+x
```

```
8  print('Soma = ',S)
```

# Laço FOR



```
1  #Montar lista
```

```
3  def Leitura(lista,qtd):  
4      for i in range(1,qtd+1):  
5          lista.append(int(input("Digite o valor do elemento ")))
```

```
7  ListaNotas= []  
8  Leitura(ListaNotas,5)  
9  print(ListaNotas)
```



# Laço FOR



```
1 # As notas de um aluno estão armazenadas em uma lista.  
2 # Calcular a média dessas notas.
```

```
4 Lista_notas= [3.4,6.6,8,9,10,9.5,8.8,4.3]  
5 soma=0
```

```
7 for nota in Lista_notas:  
8     soma = soma + nota
```

```
10 média = soma/len(Lista_notas)
```

```
12 print('Média = ', '{:.4f}'.format(média))
```



## ATIVIDADE 2

```
1 # As notas de um aluno estão armazenadas em uma lista.  
2 # Calcular a média dessas notas.
```

```
4 def CalcMedia(lista):  
5     soma=0  
6     for nota in lista:  
7         soma = soma + nota  
8     return soma/len(lista)
```

```
10 Lista_notas= [3.4,6.6,8,9,10,9.5,8.8,4.3]
```

```
12 media = CalcMedia(Lista_notas)
```

```
14 print('Média = ', '{:.4f}'.format(media))
```

# DÚVIDAS





# BIBLIOTECA NUMPY - Vetor

```
1 import numpy as np
```

```
3 a = np.array([12,34,26,18,10]) # cria uma matriz unidimensional
```

```
5 print(a)
```

```
6 print(a.dtype)
```

```
[12 34 26 18 10]  
int64
```



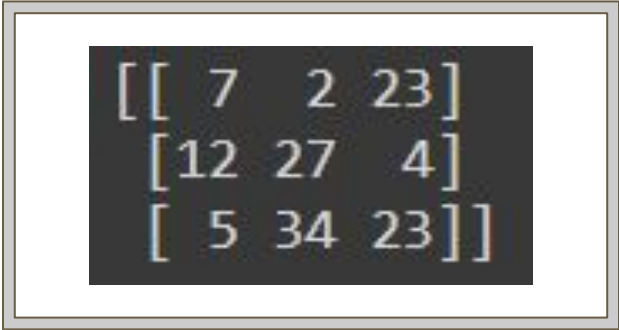
# BIBLIOTECA NUMPY - Matriz

```
1 import numpy as np
```

```
3 # cria um matriz bidimensional
```

```
4 b = np.array([[7,2,23],[12,27,4],[5,34,23]])
```

```
6 print(b)
```



```
[[ 7  2 23]  
 [12 27  4]  
 [ 5 34 23]]
```



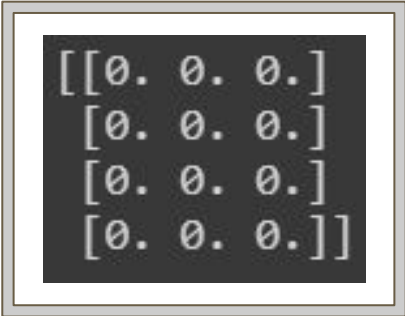
# BIBLIOTECA NUMPY - Matriz

```
1 import numpy as np
```

```
3 # cria uma matriz 4x3 com valores zero
```

```
4 d = np.zeros([4,3])
```

```
6 print(c)
```

A 4x3 matrix of zeros, displayed as a list of lists. The matrix is enclosed in a double-bordered box.

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



# BIBLIOTECA NUMPY - Matriz

```
1 import numpy as np
```

```
3 # cria uma matriz de 5x7 com valores 1  
4 e = np.ones([5,7])
```

```
6 print(e)
```

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



# BIBLIOTECA NUMPY - Matriz

```
1 import numpy as np
```

```
3 # Carregar um array com valores aleatórios.
```

```
4 g = np.random.random((5, 2))
```

```
6 print(g)
```

```
[[0.67216489 0.11556796]
 [0.46711967 0.43568234]
 [0.07573738 0.0559876 ]
 [0.16976327 0.35589302]
 [0.55497006 0.99364991]]
```





# BIBLIOTECA NUMPY - Matriz

```
1 import numpy as np

3 # Carregar um array com valores aleatórios com negativo.
4 g2= np.random.randn((5),(2))

6 print(g2)
```

```
[[ 0.6350792 -1.14191243]
 [ 0.30870985  0.20569628]
 [-0.13096161  1.4407617 ]
 [ 0.42598957  0.83590921]
 [ 0.36755948 -0.01270296]]
```



# BIBLIOTECA NUMPY - Matriz

```
1 import numpy as np
```

```
3 # O método "random" para criar arrays multidimensionais com ganhos definidos
4 h = (10*np.random.random((3,4)))
5 print(h)
```

```
7 # retornar o valor absoluto
8 h = np.floor(h)
9 print(h)
```

```
[[5.85397789 9.97273226 8.26247896 6.69999324]
 [6.72221126 8.23934455 6.40427515 6.62700814]
 [3.24751005 2.71017698 3.41130433 2.26394094]]
[[5. 9. 8. 6.]
 [6. 8. 6. 6.]
 [3. 2. 3. 2.]]
```

# BIBLIOTECA NUMPY - Where



```
1  import numpy as np

3  # criando matriz com valores aleatórios positivos e negativos
4  v = np.random.randn(4, 4)
5  print(v)

7  # criando matriz com valores booleanos baseado no array v
8  x = (v > 0)
9  print(x)

11 # criando matriz com valores -1 e 1 baseado nos valores do array x
12 z = np.where(x > 0, 1, 0)
13 print(z)
```

# BIBLIOTECA NUMPY - Where



```
[[ -0.54540342  1.21060329 -1.35828984  1.20804078]
 [  0.24722995 -0.43484467 -0.06924833 -1.15007648]
 [-0.18768103 -1.7230034  -1.92313773  0.56098353]
 [  0.54426559  1.31460213  0.12107292 -0.02367841]]

[[False  True False  True]
 [ True False False False]
 [False False False  True]
 [ True  True  True False]]

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



# BIBLIOTECA NUMPY - Operações

```
1 import numpy as np

3 # cria a matriz bidimensional k
4 k = np.array([[17,22,43],[27,25,14],[15,24,32]])

6 print(k) # Mostra a matriz k
7 print(k[0][1]) # Mostra um elemento específico da matriz k
8 print(k.shape) # Mostra o tamanho das dimensões da matriz k
9 print(k.max()) # Mostra o maior valor da matriz k
10 print(k.min()) # Mostra o menor valor da matriz k
11 print((k.sum())) # Mostra a soma dos valores da matriz k
12 print(k.mean()) # Mostra o valor da média dos valores da matriz k
13 print(k.std()) # Mostra o valor do desvio padrão dos valores da matriz k
```



# BIBLIOTECA NUMPY - Operações

```
[[17 22 43]
 [27 25 14]
 [15 24 32]]
22
(3, 3)
43
14
219
24.333333333333332
8.615231988880057
```

# BIBLIOTECA NUMPY - Operações



```
1 import numpy as np
```

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

```
4 o = np.array([[1, 1], [1, 1]])
```

```
6 # Soma de matrizes
```

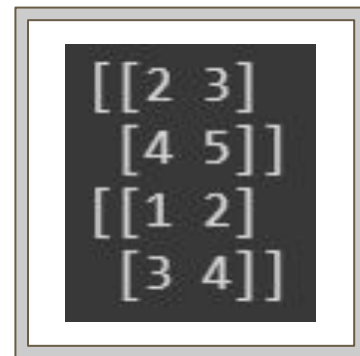
```
7 res1 = n+o
```

```
8 print(res1)
```

```
10 # Multiplicação de matrizes
```

```
11 res2 = n*o
```

```
12 print(res2)
```

A diagram showing the result of adding two 2x2 matrices. It consists of a dark gray square containing the text  $\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$  and  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ , which is enclosed within a white border.

# BIBLIOTECA MATPLOTLIB

```
1 import matplotlib.pyplot as plt
```





# BIBLIOTECA MATPLOTLIB

```
1 import matplotlib.pyplot as plt
```

```
3 x = [1,2,3,4,5,6,7,8,9,10]
```

```
4 y = []
```

```
6 for i in x:
```

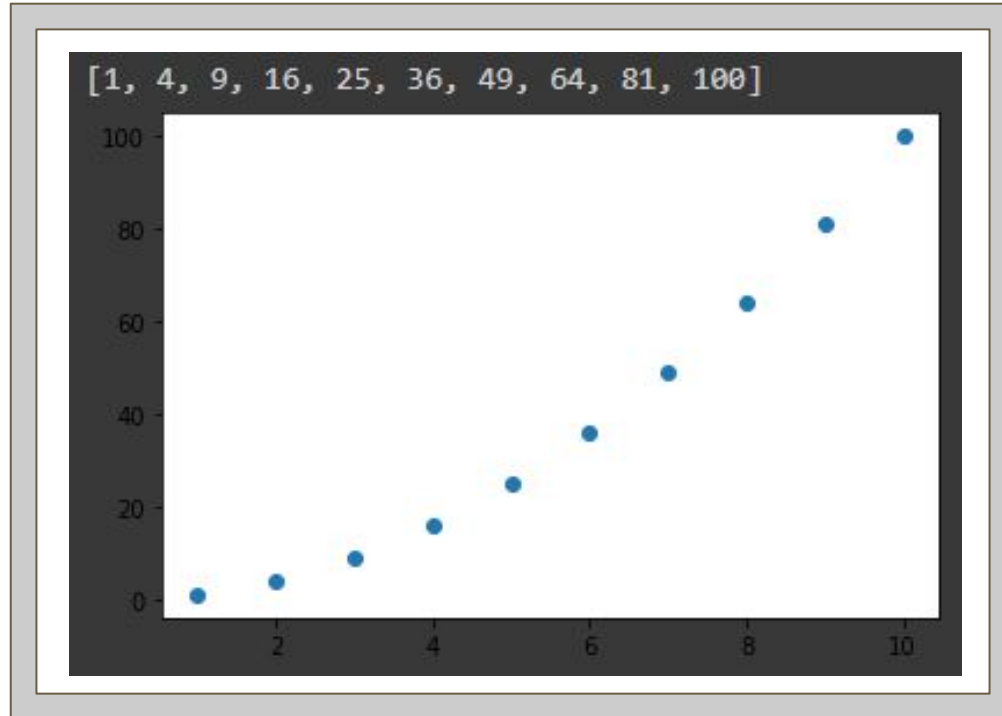
```
7     y.append(i**2)
```

```
8 print(y)
```

```
10 plt.scatter(x,y)
```

```
11 plt.show()
```

# BIBLIOTECA MATPLOTLIB





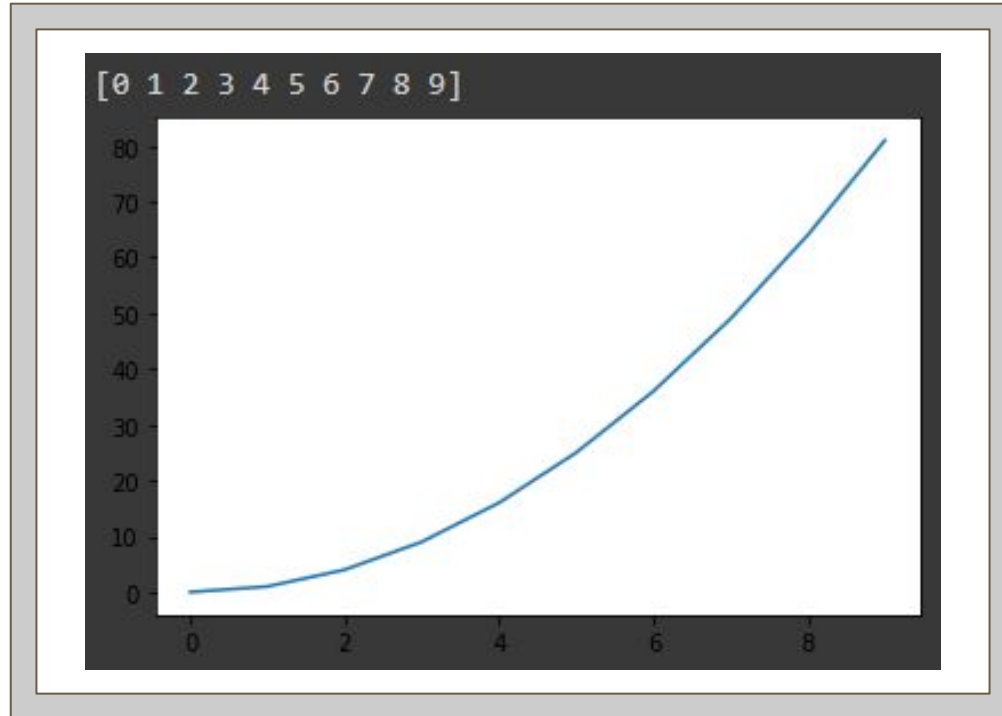
# BIBLIOTECA MATPLOTLIB

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

```
4 x1 = np.arange(0,10,1)
5 print(x1)
```

```
7 plt.plot(x1,x1**2)
8 plt.show()
```

# BIBLIOTECA MATPLOTLIB





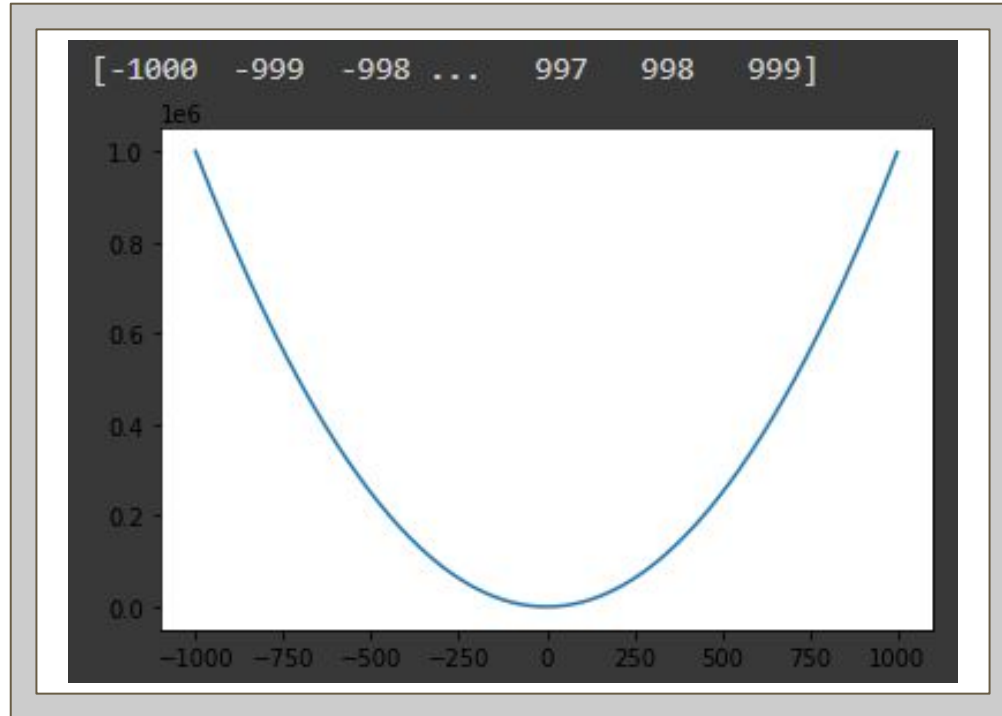
# BIBLIOTECA MATPLOTLIB

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

```
4 x2 = np.arange(-1000,1000,1)
5 print(x2)
```

```
7 plt.plot(x2,x2**2)
8 plt.show()
```

# BIBLIOTECA MATPLOTLIB



# DÚVIDAS



# BIBLIOTECA PANDAS

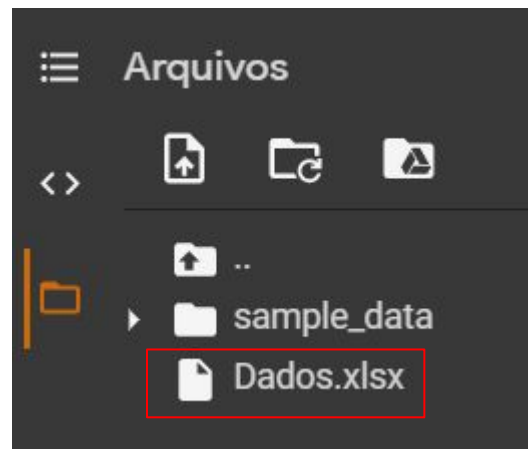
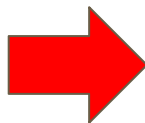
```
1 import pandas as pd
```





# BIBLIOTECA PANDAS - Upload files

		Dados	☆	📁	☁
		Arquivo	Editar	Ver	Inserir
		↶	↷	🖨	📋
				100%	RS
	<i>fx</i>				
		A	B		
1		Números	Letras		
2		1	a		
3		2	b		
4		3	c		
5		4	d		
6		5	e		
7		6	f		
8		7	g		
9		8	h		
10		9	i		
11		10	j		
12					



# BIBLIOTECA PANDAS - Upload files



<https://pandas.pydata.org/>

<https://pandas.pydata.org/docs/pandas.pdf>

```
6 Datos1 = pd.read_excel('Datos.xlsx')
```

```
8 Datos1.shape (10, 2)
```

```
10 Datos1.head() — 10 Datos1.head(10)
```

	Números	Letras
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e

	Números	Letras
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e
5	6	f
6	7	g
7	8	h
8	9	i
9	10	j