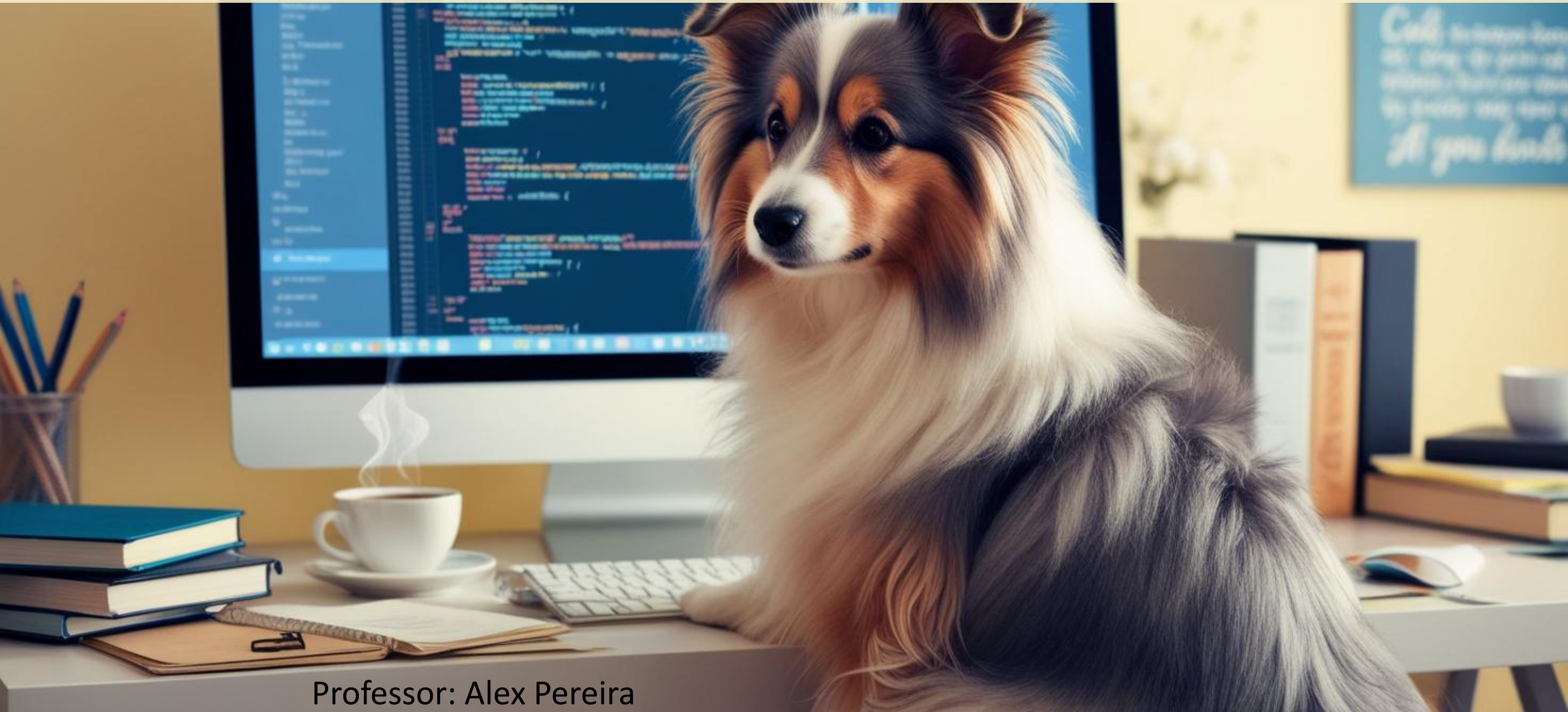


# *Introdução a Programação em Python*



Professor: Alex Pereira

# ***Apresentação Pessoal***



**2000 a 2004 – Graduação em Engenharia de Computação no ITA**



**2005 a 2008 – Mestrado em Eng. de Computação e Eletrônica no ITA**



**2009 a 2015 – Doutorado em Eng. de Computação e Eletrônica no ITA**



**2004 a 2010 – Empreendedor, sócio em empresa de base tecnológica**



**2013 a 2017 – Censipam / Ministério da Defesa**



**2016 – Professor**




**2017 – Ministério do Planejamento**



**2024 – Presidente da Associação dos Engenheiros do ITA**

Right now (23/12/24), there are a handful of people who have realized that

# *Computer Science **as we know** it is dead*

AI has written every line of code that I have worked on in the last two months, and I have heard the same from many people I respect. And the vast majority of the world hasn't caught up, and has no idea that this is even possible. **Plan accordingly!** 

[Fonte](#)

# *Faz sentido aprender a programar na era da IA Generativa ?*

- Analogia com a calculadora
  - Faz sentido aprender as 4 operações básicas
    - ✓ Na era da calculadora?
      - Por que?
- Não faz mais sentido você mesmo produzir cada linha de código
  - A maior produtividade está em especificar trechos/módulos
    - ✓ Suficientemente não ambíguos

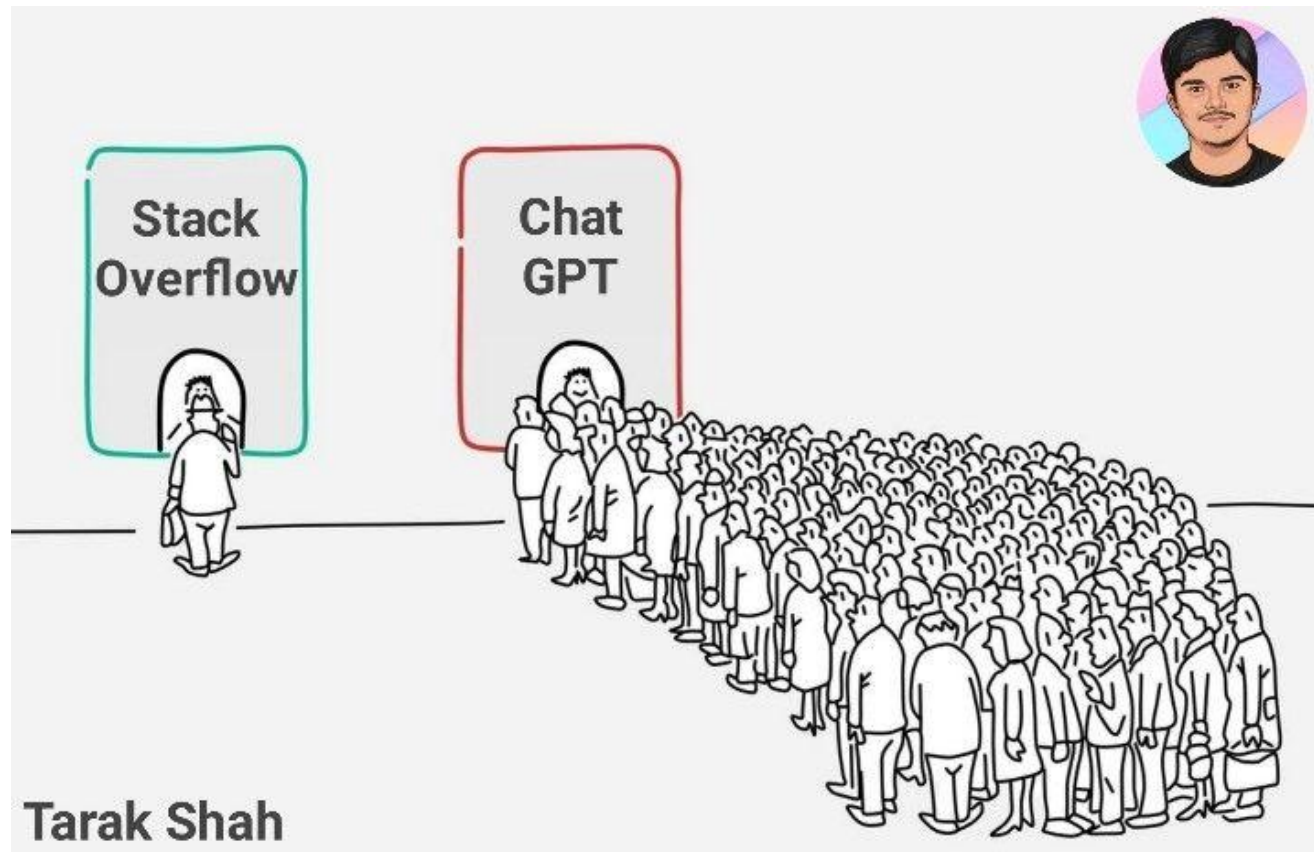
***O futuro é de quem sabe pedir***

Na Era da IA

Quem sabe pedir é quem tem o vocabulário necessário

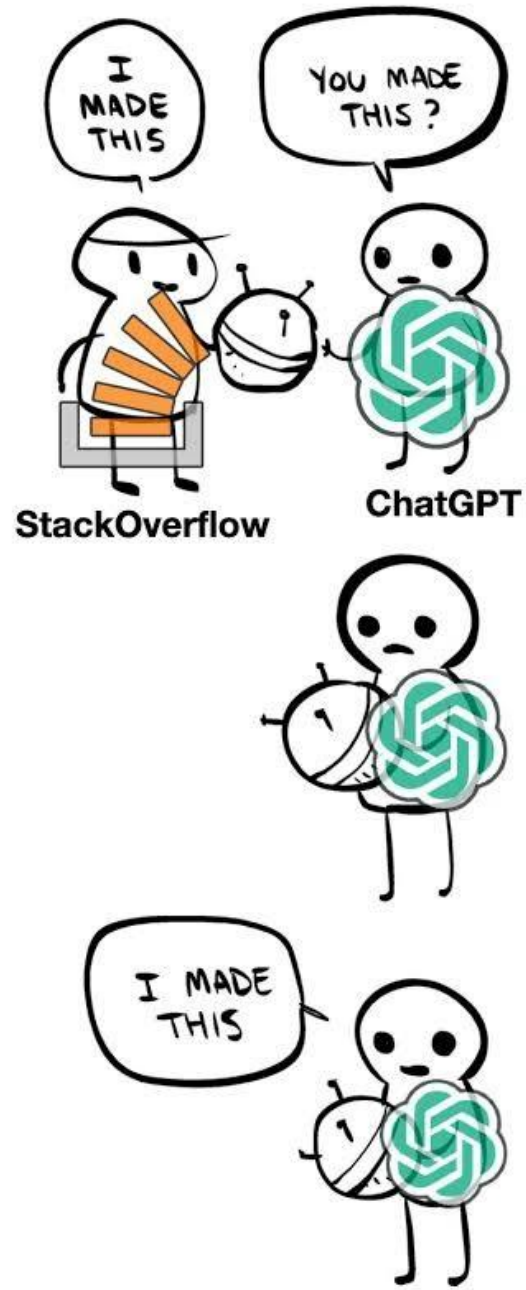


# Stack Overflow vs Gen AI (Novo vs Antigo)





# *Stack Overflow vs Gen AI (Novo vs Antigo)*



## *Links Importantes*

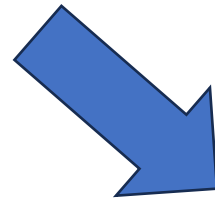
- Google Colab (Notebooks)
  - <https://colab.research.google.com> (Conta no Google/Gmail)
- Link da Aula no Zoom
- Repositório da Disciplina no Github
- [Dashboard de Notas](#) (entre com seu email pra ver seu desempenho e comparar com o desempenho da turma)
- Cadastro de usuários autorizados (Conta no Google/Gmail)



# Resolvendo Problemas com o ChatGPT

Bob	Jane	Ivy	John	Laura	Greg
5	10	30	0	35	40

keys\_in\_range(<sup>↑</sup> , 9, 50) =  
["Jane", "Ivy", "Laura", "Greg"]

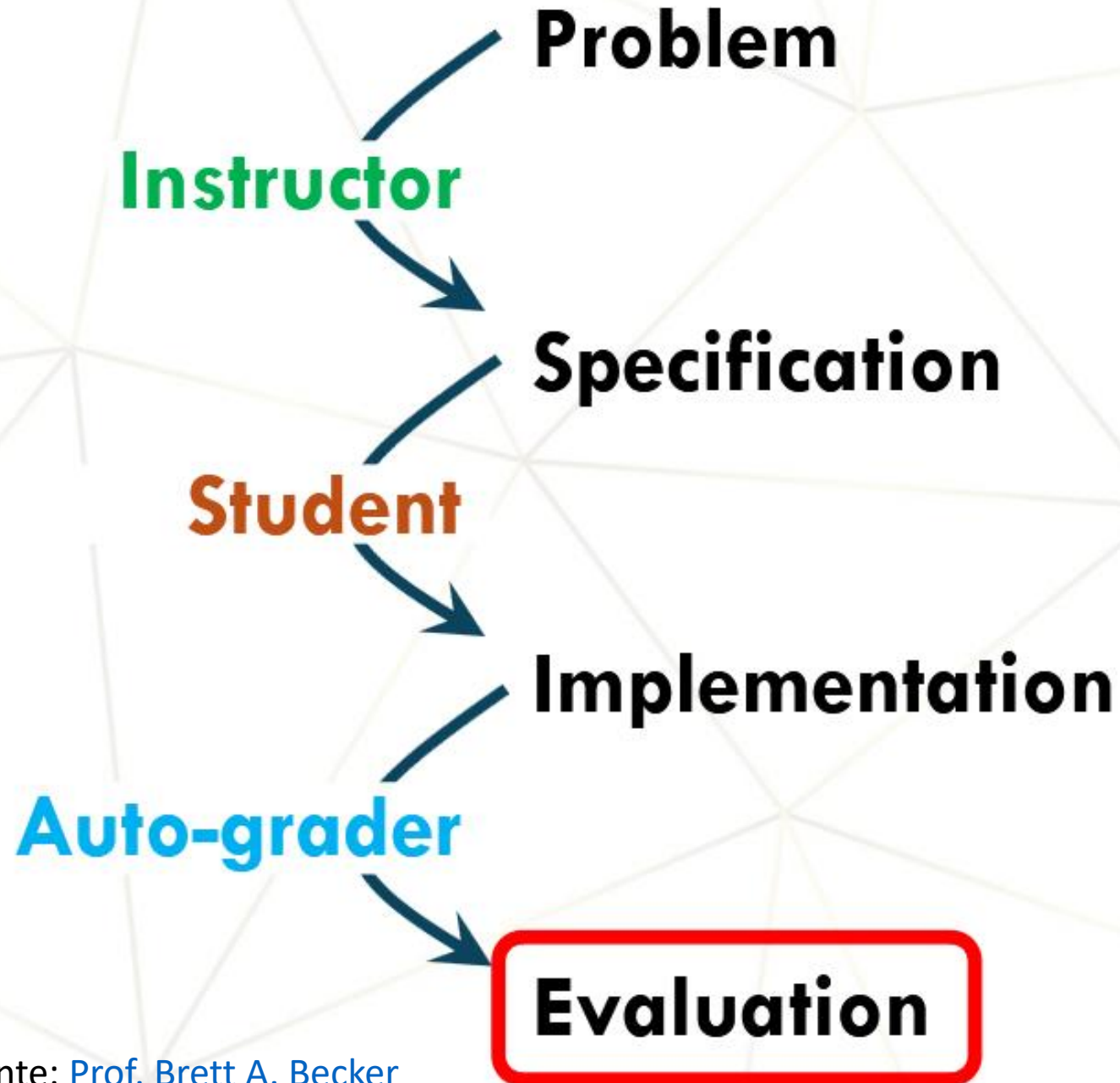


python

 Copy code

```
def keys_in_range(data, lower, upper):  
    return [key for key, value in data.items() if lower <= value <= upper]  
  
data = {"Bob": 5, "Jane": 10, "Ivy": 30, "John": 0, "Laura": 35, "Greg": 40}  
result = keys_in_range(data, 9, 50)  
print(result) # Output: ['Jane', 'Ivy', 'Laura', 'Greg']
```

# Ensino de Programação

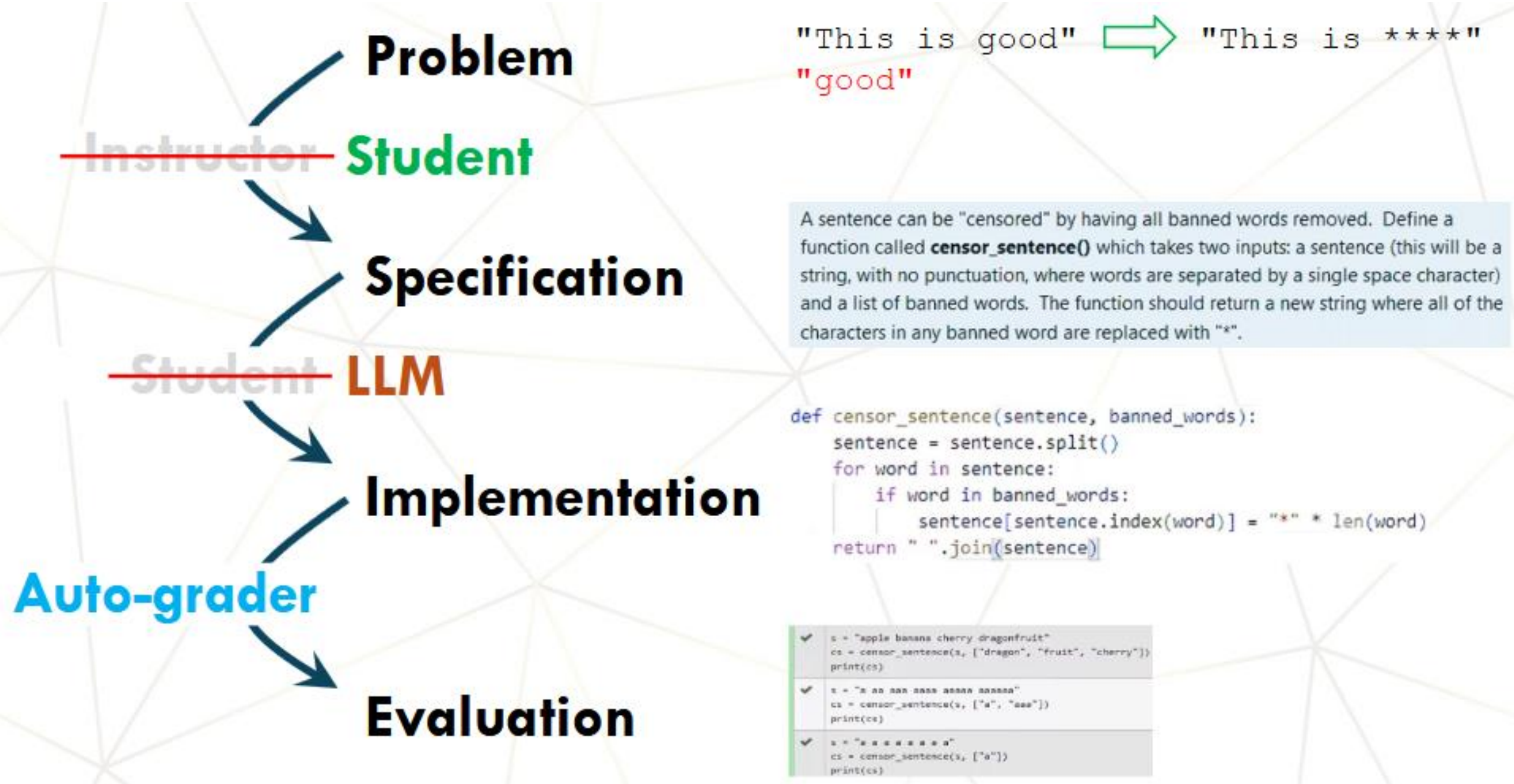


"This is good"  "This is \*\*\*\*"  
"good"

A sentence can be "censored" by having all banned words removed. Define a function called **sensor\_sentence()** which takes two inputs: a sentence (this will be a string, with no punctuation, where words are separated by a single space character) and a list of banned words. The function should return a new string where all of the characters in any banned word are replaced with "\*".

```
def sensor_sentence(sentence, banned_words):  
    sentence = sentence.split()  
    for word in sentence:  
        if word in banned_words:  
            sentence[sentence.index(word)] = "*" * len(word)  
    return " ".join(sentence)
```

```
✓ s = "apple banana cherry dragonfruit"  
  cs = sensor_sentence(s, ["dragon", "fruit", "cherry"])  
  print(cs)  
  
✓ s = "a aa aaa aaaa aaaaa aaaaaa"  
  cs = sensor_sentence(s, ["a", "aaa"])  
  print(cs)  
  
✓ s = "a a o o o o o o o"  
  cs = sensor_sentence(s, ["a"])  
  print(cs)
```





# *Contexto Pós IA Generativa*

- Não faz mais sentido avaliar Código do Aluno
  - Avaliação incentiva/promove a prática de atividades (foco)
    - ✓ Essas atividades devem ser
      - Compatíveis/equivalentes com a prática no trabalho do dia a dia
      - Cognitivamente atrativas/desafiadoras
- O que faz sentido agora é avaliar os prompts
  - Avaliar se o aluno produziu um prompt
    - ✓ que um LLM transforma em código e resolve o problema
- Nossos exercícios serão de produção de prompt

# *Google Colab*

Comandos no Terminal e funcionalidades úteis

# *Comandos de Terminal*

- cd
  - Para entrar num diretório
- ls
  - Para listar o conteúdo de um diretório
- cp
  - Para copiar um arquivo ou uma pasta
- curl
  - Para fazer uma requisição http



# *Funcionalidades Úteis do Colab*

- **IA Generativa de Código** ★★ ★
- Upload de arquivo
- Salvar arquivo no github
- Secrets
- Variáveis
- Terminal
- Atalhos (SHIFT+ENTER e CTRL+ENTER)
- Diff notebooks



- Poucas funções/keywords (ferramentas)
- Funções/Keywords abrangentes
- Mais fácil memorizar
- Demanda mais criatividade para combinar Keywords para resolver um problema



- Muitas funções (ferramentas)
- Funções específicas
- Mais difícil memorizar
- Demanda menos criatividade e mais leitura/consulta

<https://numpy.org/doc/stable/search.html>

# *Recursos do Numpy*

- Computação eficiente com array multi-dimensional
  - armazena dados numa região contínua de memória;
  - As funções numpy escritas em C podem operar diretamente na memória.
- Funções matemáticas eficientes/rápidas em arrays
  - sem a necessidade de escrever loops (laços)
- Possui funções de
  - Álgebra linear, geração de números aleatórios, entre outros



# *Criando ndarrays (array de N dimensões)*

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
print(arr1)

data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
print(arr2)

print(np.zeros(5)) # Função que cria um array de zeros
```

```
[6. 7.5 8. 0. 1.]
[[1 2 3 4]
 [5 6 7 8]]
[0. 0. 0. 0. 0.]
```

## *Alguns atributos dos ndarrays*

```
# 0 tipo do dado é inferido. Mas também pode ser especificado
arr1 = np.array([6, 7.5, 8, 0, 1])
print(arr1.dtype) # tipo do dado
arr2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]], dtype=np.int32)
print(arr2.shape) # Forma/dimensões do array
print(arr2.ndim) # Quantidade de dimensões
```

```
float64
(2L, 4L)
2
```

# *Alguns tipos de dados Numpy*

Tipo básico	Tipo Numpy disponív.	Comentários
Boolean	bool	Tamanho de 1 byte
Integer	int8, int16, int32, int64, int128, int	int tem o tamanho do int padrão da plataforma C
Unsigned Integer	uint8, uint16, uint32, uint64, uint128, uint	uint tem o tamanho do uint padrão da plataforma C
Float	float32, float64, float, longfloat	Float é sempre de precisão dupla (64 bits). longfloat é um float maior cujo tamanho depende da plataforma.
String	str, Unicode	Unicode é sempre UTF32



## *Conversão de tipos (cast)*

```
num_str = np.array(['1.25', '-9.6', '42'], dtype=np.string_)  
arr_float = num_str.astype(float) # Converte para float  
print(arr_float)  
  
arr1 = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])  
arr1_int = arr1.astype(np.int32) # Converte para int32  
print(arr1_int)
```

```
[ 1.25 -9.6  42. ]  
[ 3 -1 -2  0 12 10]
```

## *Array bi-dimensional*

```
data = np.array([[ 0.78, 1.87, 0.82], [-1.60, -0.01, -0.02]])  
print(data)  
print(data.shape) # Forma/dimensões do array  
print(data.ndim) # Quantidade de dimensões
```

```
[[ 0.78  1.87  0.82]  
 [-1.6  -0.01 -0.02]]  
(2, 3)  
2
```

Curiosidade: Qual o dado bi-dimensional mais comum na computação ?

# *Operações vetorizadas em ndarrays*

- Funções e operações em várias dimensões sem loops
  - Mais eficientes (preferíveis)

```
data = np.array([[ 3, 4, 8], [10, -4, -2]])  
print(data)  
print(data * 10)  
print(data + data)
```

```
[[ 3  4  8]  
 [10 -4 -2]]  
[[ 30 40 80]  
 [100 -40 -20]]  
[[ 6  8 16]  
 [20 -8 -4]]
```

# *Aritmética com NumPy Arrays*

```
arr = np.array([1, 2, 3])  
arr2 = np.array([4, 5, 6])  
print(arr * arr2)  
print(arr - arr2)  
print(1 / arr)  
print(arr ** 0.5)  
print(arr2 > arr)
```

```
[ 4 10 18]  
[-3 -3 -3]  
[1.    0.5   0.33333333]  
[1.    1.41421356 1.73205081]  
[ True  True  True]
```

## *Outros métodos de criação de array*

```
print(np.arange(10)) #cria array de números sequenciais  
print(np.arange(2, 10, dtype=float))  
print(np.zeros(5))  
print(np.ones((2,3)))
```

```
[0 1 2 3 4 5 6 7 8 9]  
[2. 3. 4. 5. 6. 7. 8. 9.]  
[0. 0. 0. 0. 0.]  
[[1. 1. 1.]  
 [1. 1. 1.]]
```



# *Comparação de Desempenho: Numpy array vs list*

```
my_arr = np.arange(100000000) #cria array de números sequenciais
my_list = list(range(100000000))
# %time mede o tempo tomado pela execução da linha
%time my_arr2 = my_arr * 2
%time my_list2 = [x * 2 for x in my_list]
print(my_arr2[1:5])
print(my_list2[1:5])
```

Wall time: 32 ms

Wall time: 1.15 s

[2 4 6 8]

[2, 4, 6, 8]

# *Slicing (fatiar) NumPy Arrays*

- Nas listas os slices (cortes) são cópias;
- Nos arrays numpy os slices são views (visualizações)
  - para copiar, use a função: `.copy()` , exemplo: `arr[1:4].copy()`

```
arr = np.arange(10)
print(arr)
arr_slice = arr[5:8]
arr_slice[1] = 12345
print(arr)

li2 = list(range(10)) # 0 equivalente com uma lista
list_slice = li2[5:8]
list_slice[1] = 12345
print(li2)
```

# *Slicing (fatiar) de arrays bidimensionais*

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(arr2d)  
print(arr2d[2])  
print(arr2d[0][2]) # Tanto faz  
print(arr2d[0, 2]) # Tanto faz  
print(arr2d[:, 1])
```

```
[[1 2 3]  
 [4 5 6]  
 [7 8 9]]  
[7 8 9]  
3  
3  
[2 5 8]
```

		axis 1		
		0	1	2
axis 0	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

# Outras funções numpy

- `np.sum()` – Soma os elementos de um array
  - Todos os elementos ou os elementos em cada um dos eixos
    - ✓ `axis=0`, soma ao longo das colunas
    - ✓ `axis=1`, soma ao longo das linhas

```
1 ar = np.array([[0, 1], [0, 5]])
2 print(ar)
3 print(np.sum(ar))
4 print(np.sum(ar, axis=0))
5 print(np.sum(ar, axis=1))
```

```
[[0 1]
 [0 5]]
```

```
6
```

```
[0 6]
[1 5]
```

## *Outras funções numpy*

- `np.mean()` – Calcula a media dos elementos de um array
- `np.std()` – Calcula o desvio padrão dos elementos de um array
- `np.any()` – Retorna um booleano (ou um array de booleanos) informando se há pelo menos um elemento True.

```
1 arr2 = np.array([[True, False], [True, True]])
2 print(arr2)
3 print(np.any(arr2))
4 print(np.any(arr2, axis=0))
```

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



## *Outras funções numpy*

- `np.argmax()` – Retorna os índices dos valores máximos ao longo de um eixo
  - Analogamente `np.argmin()` retorna os índices dos valores mínimos

```
1 a = np.array([[10, 11, 12],
2               [13, 14, 15]])
3 print(a)
4 print(np.argmax(a))
5 print(np.argmax(a, axis=0))
6 print(np.argmax(a, axis=1))
```

```
[[10 11 12]
 [13 14 15]]
```

```
5
```

```
[1 1 1]
```

```
[2 2]
```

# *Filtrar um Array baseado numa condição*

- A condição que você deseja filtrar (especificação de requisito)
  - é colocada entre colchetes

```
1 arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
2 print(arr)
3 print(arr < 4)
4 print(arr[arr < 4])
```

```
[0 1 2 3 4 5 6 7 8]
```

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

```
[0 1 2 3]
```

# Operações lógicas com Array Numpy

- & faz a operação AND e | faz a operação OR
  - **sempre** use os parenteses para fazer operações lógicas com arrays

```
1 arr1 = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8])
2 selecao = (arr1 > 2) & (arr1 < 5)
3 print(selecao)
4 print(arr[selecao])
```

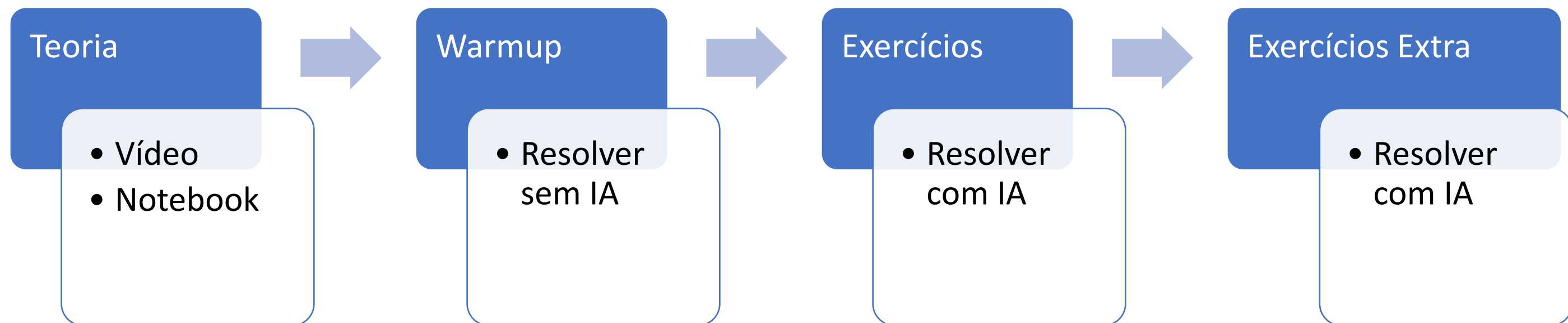
```
[False False False  True  True False False False False]
[3 4]
```

```
1 selecao2 = (arr1 < 2) | (arr1 > 5)
2 print(selecao2)
3 print(arr1[selecao2])
```

```
[ True  True False False False False  True  True  True]
[0 1 6 7 8]
```

# Prática no Colab Notebook

- Escolham por onde começar: Teoria, Warmup ou Exercícios;
  - As soluções dos warmups já estão publicadas;
  - As soluções dos exercícios extra serão disponibilizadas ao final do dia;
- É esperado que não terminem todos os exercícios durante a aula;
  - Façam o restante ao longo da semana.
  - <https://numpy.org/doc/stable/search.html>



# *Se entender a piada é porque já está falando a língua dos nerds*

