

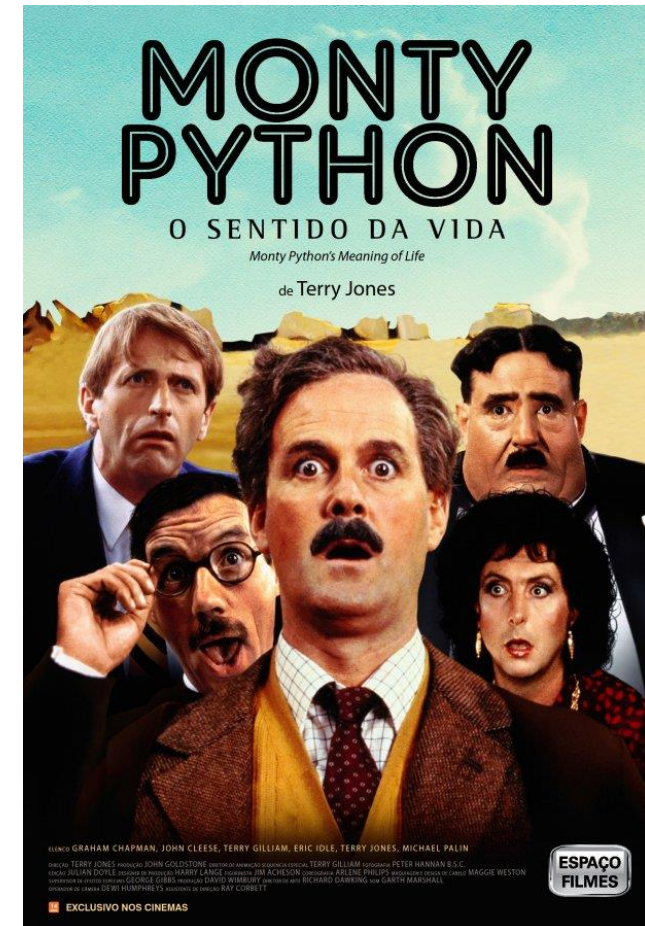
# Introdução a linguagem Python



Pythonidae (Python family)



?



Resolução dos Exercícios da Aula 1

<https://drive.google.com/file/d/16mpT4kJeVPvn8TIPX3zK5bX0SPNMsnda/view?usp=sharing>

Professor: Alex Pereira

# Revisão da Aula 1

2) Crie um pseudocódigo para o cálculo do fatorial de 10.

**PROGRAMA** fatorial10:

fat = 1

n = 1

enquanto n <= 10

    fat = fat \* n

    n = n + 1

**FINALIZAR**

n	fat*n
1	1*1
2	1*2
3	2*3
4	6*4
5	24*5
6	120*6

- Dá pra otimizar? Executar menos iterações?

# Revisão da Aula 1

2) Crie um pseudocódigo para o cálculo do fatorial

- **Aprendizados ?**

- ✓ Matemática, Ponto de Início;
- ✓ Trade-off memória/processamento.

**PROGRAMA** fatorial10:

fat = 1

n = 1

enquanto n <= 10

    fat = fat \* n

    n = n + 1

**FINALIZAR**

```
n = 3
fat = 2
while n <= 10:
    fat = fat * n
    print(n, fat)
    n = n + 1
```

3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

# Revisão da Aula 1

4) Crie um pseudocódigo para o cálculo da multiplicação dos números pares entre 1 e 100. Ou seja:  $2*4*6*...*100$

**PROGRAMA** fatorial10:

mult = 1

n = 1

enquanto n <= 100

    se n % 2 == 0

        mult = mult \* n

    n = n + 1

**FINALIZAR**

- Dá pra otimizar? Executar menos iterações?

n	fat*n
1	--
2	1*2
3	--
4	2*4
5	--
6	8*6

## Revisão da Aula

4) Crie um pseudocódigo para o cálculo da multiplicação de pares entre 1 e 100. Ou seja:  $2 \cdot 4 \cdot 6 \cdot \dots \cdot 100$

- Aprendizados ?

**PROGRAMA** fatorial10:

mult = 1

n = 1

enquanto n <= 100

    se n % 2 == 0

        mult = mult \* n

    n = n + 1

**FINALIZAR**

- Dá pra otimizar mais?

```
n = 2
mult = 1
while n <= 10:
    mult = mult * n
    print(n, mult)
    n = n + 2
```

```
2 2
4 8
6 48
8 384
10 3840
```

## *Revisão da Aula 1*

4) Crie um pseudocódigo para o cálculo da multiplicação dos números pares entre 1 e 100. Ou seja:  $2*4*6*\dots*100$

$$2 * 4 * 6 * 8 * \dots * 100$$

$$= (2*1) * (2*2) * (2*3) * (2*4) * \dots * 2*50$$

$$= 2^{50} * 1 * 2 * 3 * 4 * \dots * 50$$

# Revisão da Aula 1

4) Crie um pseudocódigo para o cálculo da multiplicação dos números pares entre 1 e 100. Ou seja:  $2*4*6*...*100$

- Aprendizados ?

- ✓ Matemática
- ✓ Memorização/Anotação

```
n = 1
mult = 1
while n <= 50:
    mult = mult * n
    n = n + 1
mult = mult * 2**50
print(mult)
```

342432247025119762482

# *Pagar ou Não pagar uma Ferramenta de IA Generativa*

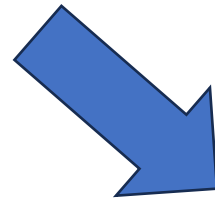
- Transações financeiras recorrentes
  - **Mental Accounting**
    - ✓ Mantemos um sistema mental de dupla entrada para compras, pesando os benefícios contra as desvantagens.
- O benefício é maior do que o seu custo?
  - **A assinatura do ChatGPT Plus está subsidiada**
    - ✓ A assinatura do ChatGPT Pro está gerando U\$200/mês de prejuízo
  - **O nível de automação (complexidade das tarefas)**
    - ✓ Vai aumentar
      - **O modelo o1 é um funcionário digital**
        - Ele iterativamente se corrige até chegar num resultado aceitável
- A variável incerta e influenciável é o benefício
  - **Meta desta disciplina: capacitar a turma gerar mais benefício do que o custo**
    - ✓ Existirão países com/sem massa crítica para alavancar sua produtividade com IA



# 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"]

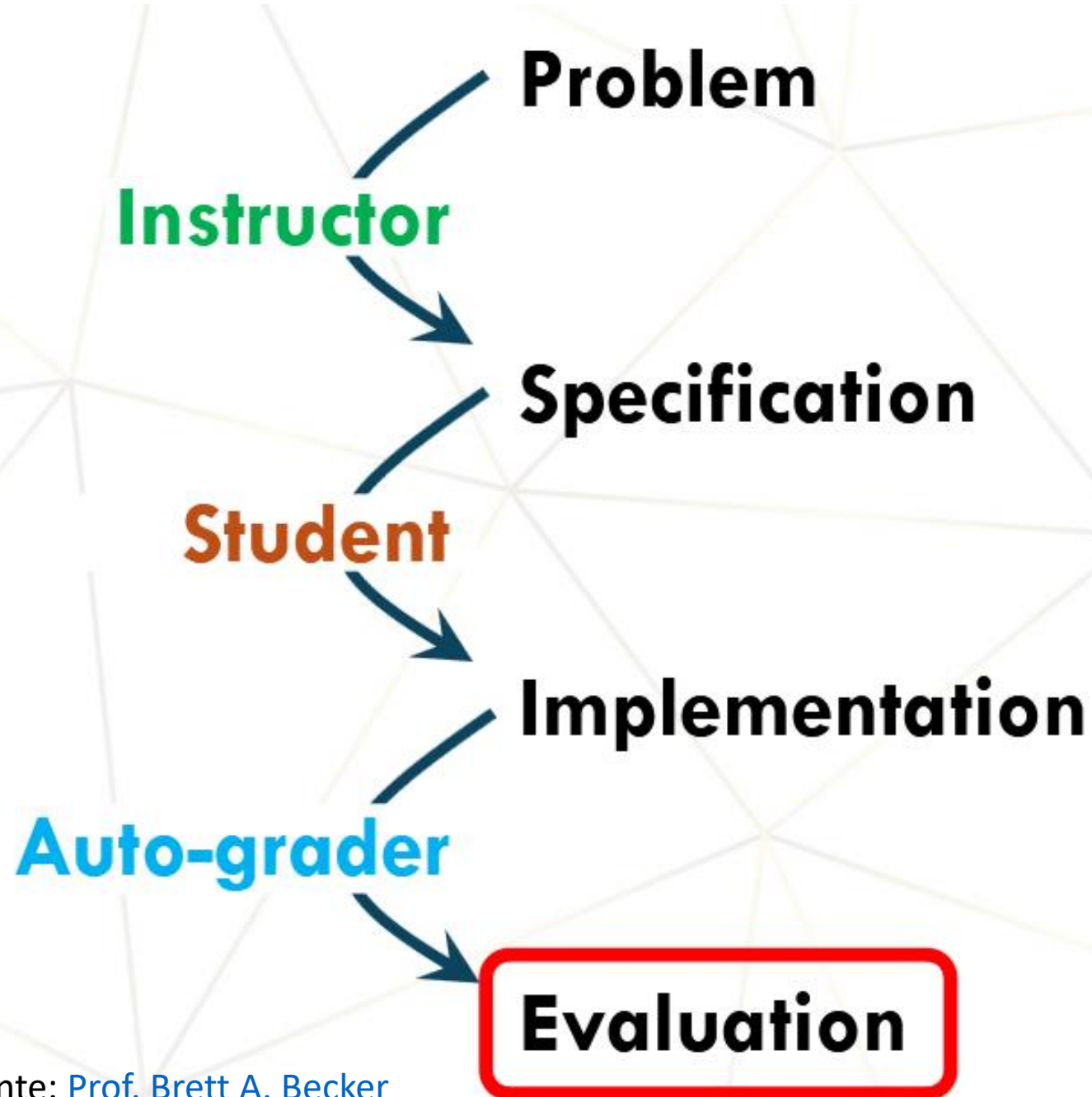


python

 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)
```

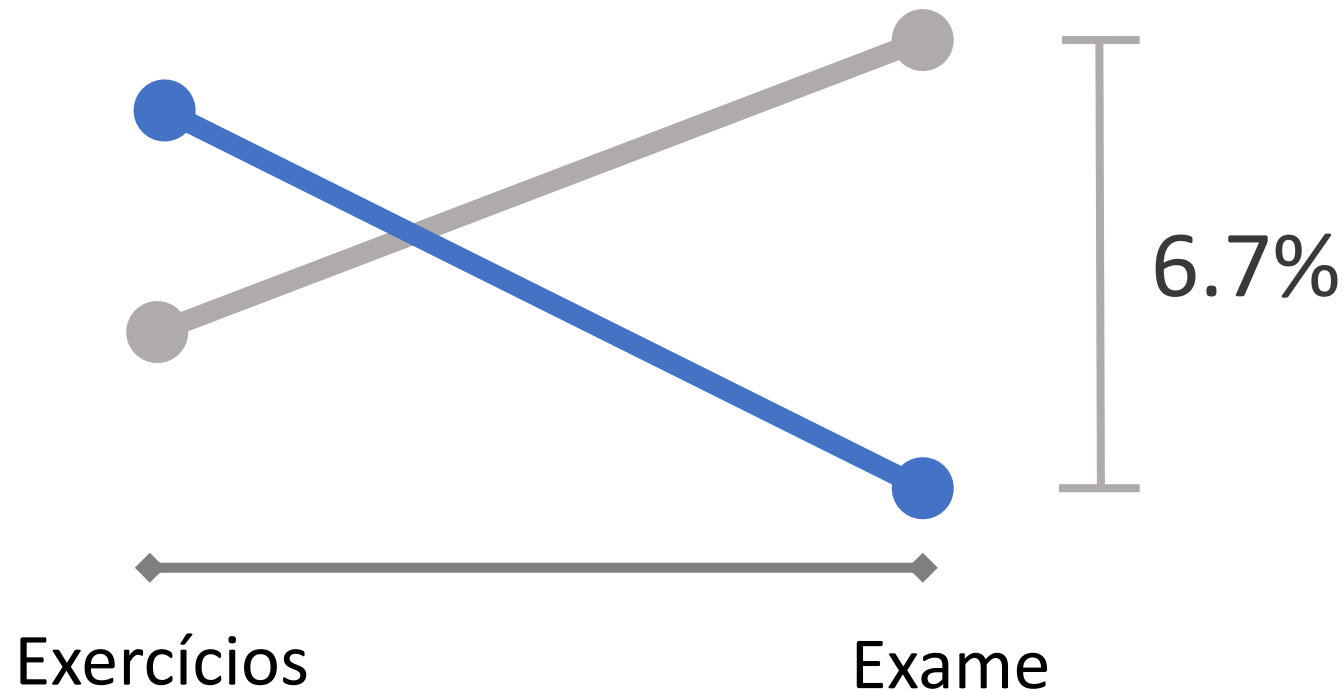


# *Contexto Pós IA Generativa*

- 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

## *Um uso inapropriado de GenAI*

- Num contexto acadêmico, usar sem tentar aprender
  - Apenas copiar e colar o resultado
- Diminui o desempenho num exame sem consulta
  - Em 6.7%
    - ✓ [Fonte](#)



*O futuro é de quem sabe pedir*

Na Era da IA

Quem sabe pedir é quem tem o vocabulário necessário

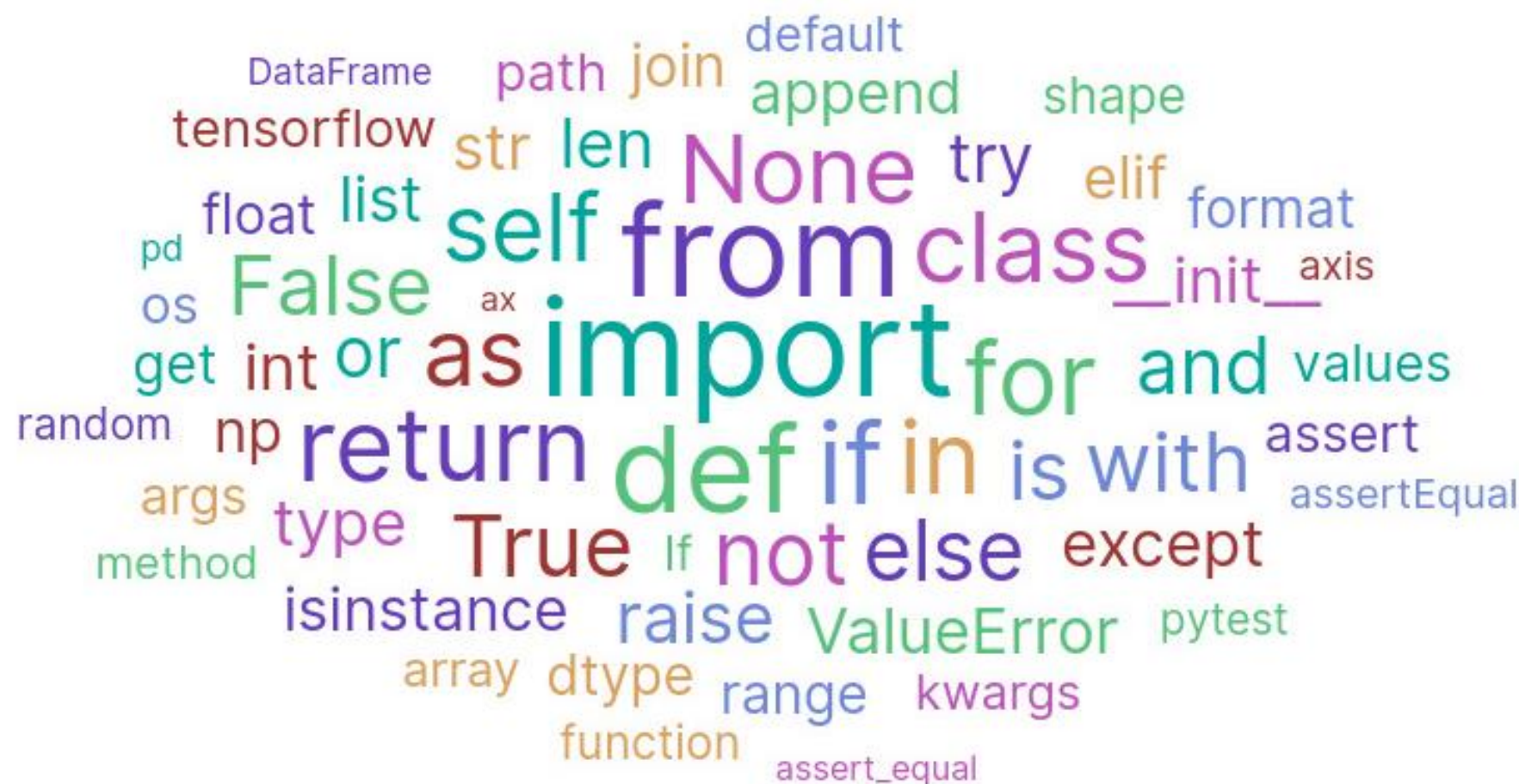
# *A Era do Rei Midas – Vocabulário / Saber Pedir*

- Mitologia Grega
  - Midas pediu ao deus Dioniso para que tudo que tocasse virasse ouro;
  - Quase morreu de fome (comida, cama, água viravam ouro);
  - Pediu ao deus Dioniso para reverter o desejo e foi atendido.
- Se você souber pedir, será atendido com eficiência



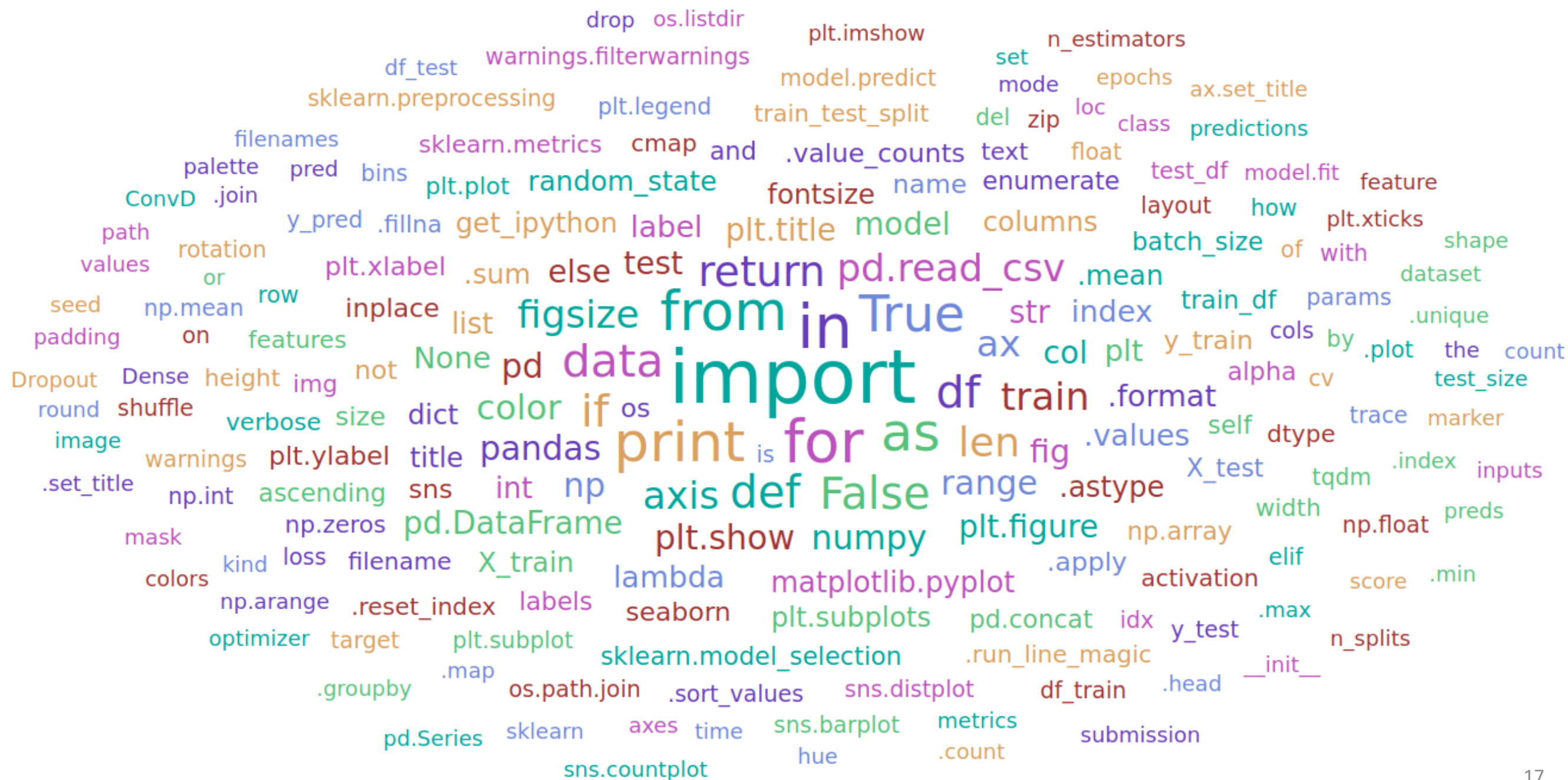
# Nuvem de Palavras do Python

- Os 100 repositórios mais populares do github.com; e
- Os 100 pacotes mais populares do pypi.org.





# *Nuvem de Palavras de Arquivos Python do Kaggle*



## *Atividade Extra para Alunos Experientes*

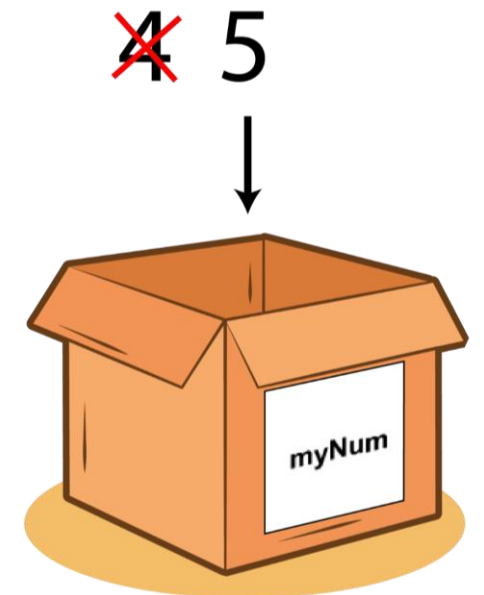
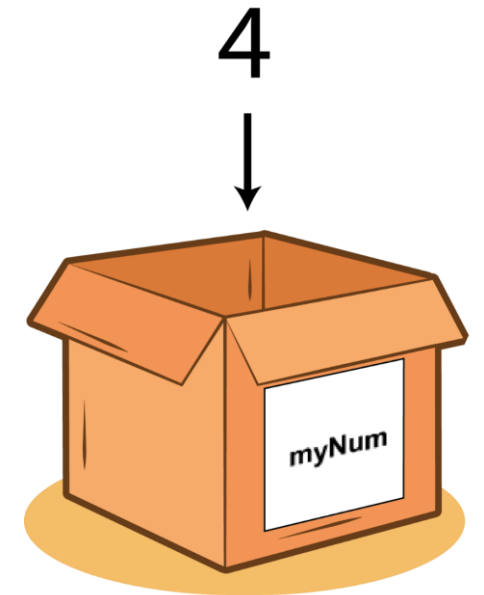
- Usando a IA para produzir código python, contabilizar a porcentagem de serviços digitais
  - Do portal gov.br
    - ✓ Os dados estão dispostos em <https://www.servicos.gov.br/api/v1/servicos/>

# Variáveis

- Label (etiqueta) / Nome e Conteúdo
  - O programador escolhe o label e o conteúdo
    - ✓ Para satisfazer um requisito ou objetivo

```
myNum = 4  
print(myNum)  
myNum = 5  
print(myNum)
```

```
4  
5
```



# Sintaxe do Python

- Dois pontos sinaliza o início de uma sentença composta
  - O conteúdo das sentenças compostas é **aninhado com espaços/tabs**
    - ✓ Sem chaves
- Ponto e vírgula para finalizar uma sentença é opcional

**Erro Comum**  
Esquecer de aninhar  
sentenças compostas

```
x = 2  # Atribuição com sinal de = (igual)
if x > 0:
    # adiciona 1 a x
    x = x + 1;
    print(x)
    print('x maior que zero')
else:
    print('x menor ou igual a zero')
```

```
3
x maior que zero
```

# *Sintaxe do Python*

- Referências para objetos não possuem um tipo associado
  - Não há problema em reatribuir uma referência a outro tipo
    - ✓ Por exemplo:

```
x = 2
print(type(x))
x = 'carro'
print(type(x))
```

```
<class 'int'>
<class 'str'>
```

# Sintaxe do Python

- Funções são declaradas com a palavra-chave **def**
  - Qual a utilidade das funções ?
    - ✓ Reusabilidade / legibilidade
  - Uma função precisa ser carregada em memória
    - ✓ para ser encontrada pelo interpretador
- Chamada / execução de uma função
  - `nome_funcao()`

## Erro Comum

Esquecer de aninhar  
sentenças compostas

```
def imprimir(a):
```

```
    print(a)
```

```
imprimir(2)
```

2

# *return de uma função Python*

- Explícito

- com o keyword **return**

- ✓ seguido de um valor

- Exemplo: `return 10`

```
def soma(m, n):  
    ←→ return m + n  
result = soma(4, 5)  
print(result)
```

9

- Implícito

- sem o keyword **return**, o retorno é **None**

- ✓ vide slide anterior

- Erro comum!

- Precisar de um retorno mas esquecer de escrevê-lo

# Revisão: Definição e execução de uma função

```
def soma(a, b):  
    s = a + b  
    return s
```

The diagram illustrates the definition of a function. It shows the code for a function named `soma` that takes two arguments, `a` and `b`, and returns their sum. The annotations are as follows:   
1. Points to the `def` keyword.   
2. Points to the function name `soma`.   
3. A bracket above the parameters `(a, b)` with a circle containing the number 3.   
4. Points to the colon `:` at the end of the function signature.   
5. A red double-headed arrow between a vertical line and the assignment statement `s = a + b`.   
6. Points to the `return` keyword.   
A vertical blue line is positioned to the left of the function body, and a red double-headed arrow points from this line to the `return` statement.

```
result = soma(1, 2)
```

The diagram illustrates the execution of the function. It shows the code for calling the `soma` function. The annotations are as follows:   
7. Points to the variable `result`.   
8. Points to the assignment operator `=`.   
9. Points to the function name `soma`.   
10. Points to the argument `1`.   
11. Points to the argument `2`.   
A vertical blue line is positioned to the left of the function call, and a red double-headed arrow points from this line to the `return` statement in the function definition above.



# *Dados do tipo Escalar*

- Guardam valores únicos (single) ou simples
- None
  - A representação de valor nulo em Python
- str
  - Conjunto de caracteres. Guarda strings codificadas com UTF-8
- bytes
  - bytes ASCII (ou bytes codificados como Unicode)
- float
  - Número em formato ponto flutuante de 64-bit (Exemplo: 3.1290)
- bool
  - Um valor True ou False
- int
  - Um número inteiro

# Operadores Binários

Operação	Descrição
<b>a + b</b>	Soma a e b
<b>a - b</b>	Subtrai a de b
<b>a * b</b>	Multiplica a por b
<b>a / b</b>	Divide a por b
a // b	Divisão inteira de a por b. Desconsidera o resto.
a ** b	Eleva a à potência b
<b>a &amp; b</b>	<b>True se a e b são True</b>
<b>a   b</b>	<b>True se a ou b são True</b>

\* Em negrito os mais importantes

# Operadores Binários

Operação	Descrição
<b>a == b</b>	<b>True se a é igual a b</b>
<b>a != b</b>	<b>True se a não é igual a b</b>
<b>a &lt; b, a &lt;= b</b>	<b>True se a é menor (ou menor ou igual) a b</b>
<b>a &gt; b, a &gt;= b</b>	<b>True se a é maior (ou maior ou igual) a b</b>
<b>a is b</b>	<b>True se a e b referenciam o mesmo objeto Python</b>
<b>a is not b</b>	<b>True se a e b referenciam objetos Python distintos</b>
<b>a ^ b</b>	Operação de XOR. True se a ou b são True, mas não ambos.

\* Em negrito os mais importantes

# Conversão de Tipos

- `str`, `bool`, `int`, e `float`
  - também são funções para converter valores para esses tipos

```
s = '3.14159'  
fval = float(s)  
ival = int(fval)  
print(fval)  
print(ival)
```

```
3.14159  
3
```

# Laços (loops) do tipo *While*

- Uma maneira de iterar sobre uma coleção
  - **enquanto** uma condição for verdadeira
    - ✓ Curiosidade: porque temos mais dificuldade de montar um laço *while* do que um do tipo *for* ?

```
n = 0
total = 0
while n < 4:
    total = total + n*n
    n = n + 1

print(total)
```

## *Laços (loops) do tipo for*

- Uma maneira de iterar sobre uma coleção
- Usa-se o keyword **in** para referenciar a coleção.
  - **val** foi um termo escolhido pelo programador
  - **break** interrompe a iteração

Atente para a correta  
indentação

```
sequence = [1, 2, 0, 4, 6, 5, 2, 1]
tot_until_5 = 0
for val in sequence:
    if val == 5:
        break
    tot_until_5 = tot_until_5 + val
print(tot_until_5)
```

## *Laço do tipo for com a função range*

- `range(5)`
  - retorna um iterator da sequência 0, 1, 2, 3, 4
- `range(2, 6)`
  - retorna um iterator da sequência 2, 3, 4, 5

```
for val in range(3):  
    print(val*val)
```

```
0  
1  
4
```

## *Formando uma string (texto)*

- Strings podem ser definidas usando aspas simples ou duplas
  - 'aluno' ou “aluno” são válidos

```
a = 4.5
```

```
b = 2
```

```
frase1 = f"a={a}, b={b}"
```

```
frase2 = "a={0}, b={1}".format(a, b)
```

```
print(frase1)
```

```
print(frase2)
```

```
a=4.5, b=2
```

```
a=4.5, b=2
```



## *Funções Populares de String: join*

- Transforma uma lista de strings numa string
  - concatenada por um separador

```
lista_str = ["1", "2", "3", "4"]  
str_concatenado = "-".join(lista_str)  
print(str_concatenado)
```

```
1-2-3-4
```

# *Funções Populares de String: split*

- `str.split(sep=None, maxsplit=-1)`
  - Divide uma string em partes separadas por um caractere separador *sep*
    - ✓ o resultado retornado é uma lista dos sub-elementos

```
texto = "1-2-3-4"  
lista = texto.split("-")  
print(lista)
```

```
['1', '2', '3', '4']
```

# *Outras operações com strings*

- Um character \* numero

- Ex.: "\*" \* 10

```
"*" * 10
```

```
'*****'
```

- Concatenação de strings

```
"hello " + "world!"
```

```
'hello world!'
```

# ***Estruturas de Dados do Python (Nativas)***

# *list (Lista)*

- Sequência de tamanho variável e conteúdo mutável (alterável)
  - Pode conter objetos de vários tipos
  - Define-se uma lista com colchetes [ ]
  - `append` (inserir elementos), `pop` (remover elementos pelo índice)

```
al = [2, 4, 0, None]
print(al)
al.append(9)
print(al)
al.pop(1) # Remover pelo índice
print(al)
al.remove(0) # Use remove para remover pelo valor
print(al)
```

```
[2, 4, 0, None]
[2, 4, 0, None, 9]
[2, 0, None, 9]
[2, None, 9]
```

# *Combinando Listas*

- Use o operador + ou a função extend
  - A função extend é mais rápida do que o operador +

```
al = [2, 4, 0, None]  
al_plus = al + ['a', 'b']  
al.extend(['a', 'b'])  
comp = al_plus == al  
print(comp)
```

True

## *Slicing (fatiar)*

- Use intervalos entre colchetes para fatiar sequências

```
al = [2, 4, 0, 3, 7, 10, 4, 5]
print(al[0:3])    # Do índice zero até o 3 (não incluso)
print(al[:4])     # Do índice zero ao 4
print(al[2:])     # Do índice 2 até o último
print(al[-1])    # O último elemento
print(al[::2])    # A cada dois elementos a partir do zero
print(al[::-1])  # Reverter/espelhar os elementos
al[1:3] = [8, 8]
print(al)
```

## *Tamanho de uma lista: `len(lista)`*

- o método **len** retorna o tamanho de uma lista

```
al = [2, 4, 0, 3, 7, 10, 4, 5]  
print(len(al))
```

```
8
```



# *Importar (import) módulos*

- Módulo
  - arquivo de extensão .py
    - ✓ Contendo definições e funções/métodos
- Package (pacote)
  - diretório/pasta marcado com a presença de um arquivo `__init__.py`
    - ✓ Serve para armazenar módulos
      - Instalar um pacote: **pip install** <nome\_pacote> (Ex.: **pip install numpy**)
        - No Google colab: **!pip install** <nome\_pacote> (Ex.: **!pip install numpy**)

```
import os  
  
print(os.getcwd())
```

C:\Users\alex\Google Drive\Aulas\Enap...

```
from os import getcwd  
  
print(getcwd())
```

C:\Users\alex\Google Drive\Aulas\Enap...

# *Be resourceful*

- Being **resourceful**
  - is the ability to find and use available resources to solve problems and achieve goals.
- Leia as mensagens de erro / log
- Aprenda os termos/palavras/vocabulários
- Aprenda a fazer buscas iterativas (na perplexity / SeachGPT)
  - O resultado de uma busca deve melhorar/refinar a sua próxima busca
- Buscar diretamente no stackoverflow / google
  - Já não é a solução mais eficiente

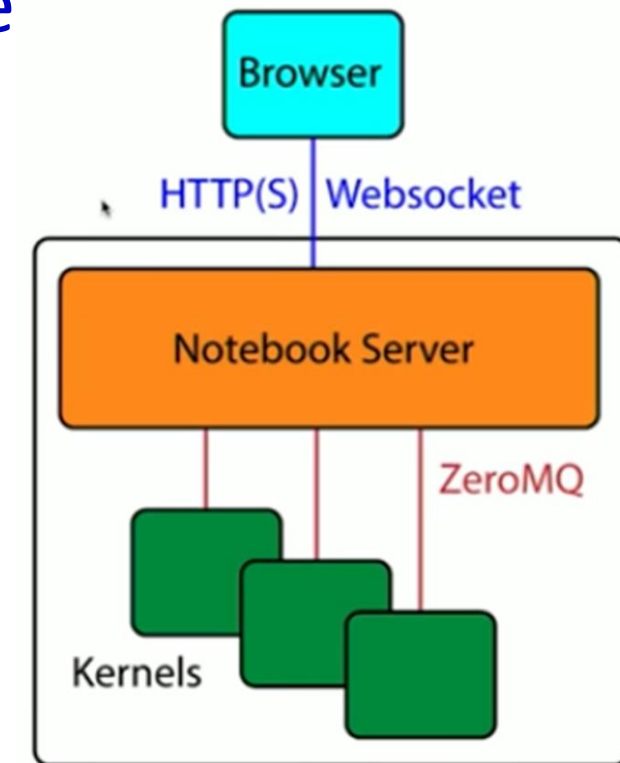
# Concentração na Aula

- Commitment device
  - ferramenta estratégica ajuda indivíduos com seus objetivos de longo prazo,
    - ✓ Ao criar restrições ou consequências para seus eus futuros
- Intention action gap
  - ocorre quando os valores, atitudes ou intenções de alguém não correspondem às suas ações.
- Multiplicação e memorização
- “Estamos num barco turístico”
  - Vou conduzir o barco e apresentar o que há de mais interessante
    - ✓ Não pule dele pra conduzir seu colete salva-vidas.



# *Introdução ao Colab / ipython (notebook)*

- O que é
  - Ferramenta de programação no navegador;
  - Códigos, instruções e resultados são mostrados juntos;
  - Útil para escrever códigos que contam uma história; e
  - Utilizado por estudantes, cientistas e pesquisadores.
- Como é implementado
  - É um servidor web local.
  - Abre uma página no navegador.
  - Suporta diversas linguagens de programação
    - ✓ Entre elas o Python.
- Boa prática de programação no Colab notebook
  - Programar iterativamente: validando cada pequeno resultado

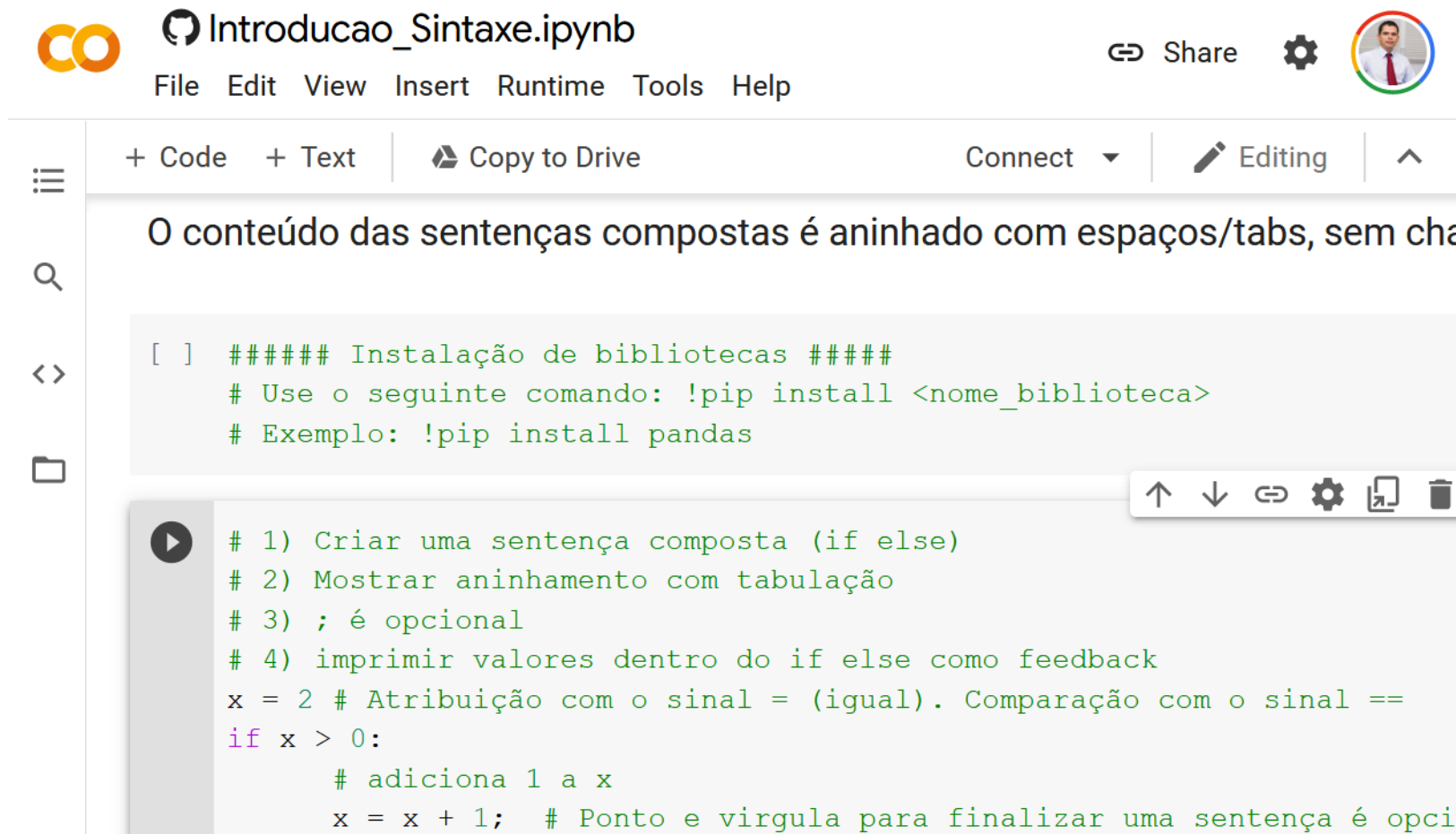




# Notebook (Caderno) no Google Colab

- Google Colab

- Um ambiente gratuito na nuvem do google para execução de código
  - ✓ De cadernos em formato equivalente ao Jupyter Notebook



# Acesso aos cadernos Colab desta Aula

- No Colab <https://colab.research.google.com/>
  - File -> Open notebook

Open notebook

Examples >

Recent >

Google Drive >

**GitHub** 1 >

Upload >

Enter a GitHub URL or search by organization or user

alexlopespereira 2

Repository: alexlopespereira/mba\_enap 3

Branch: main

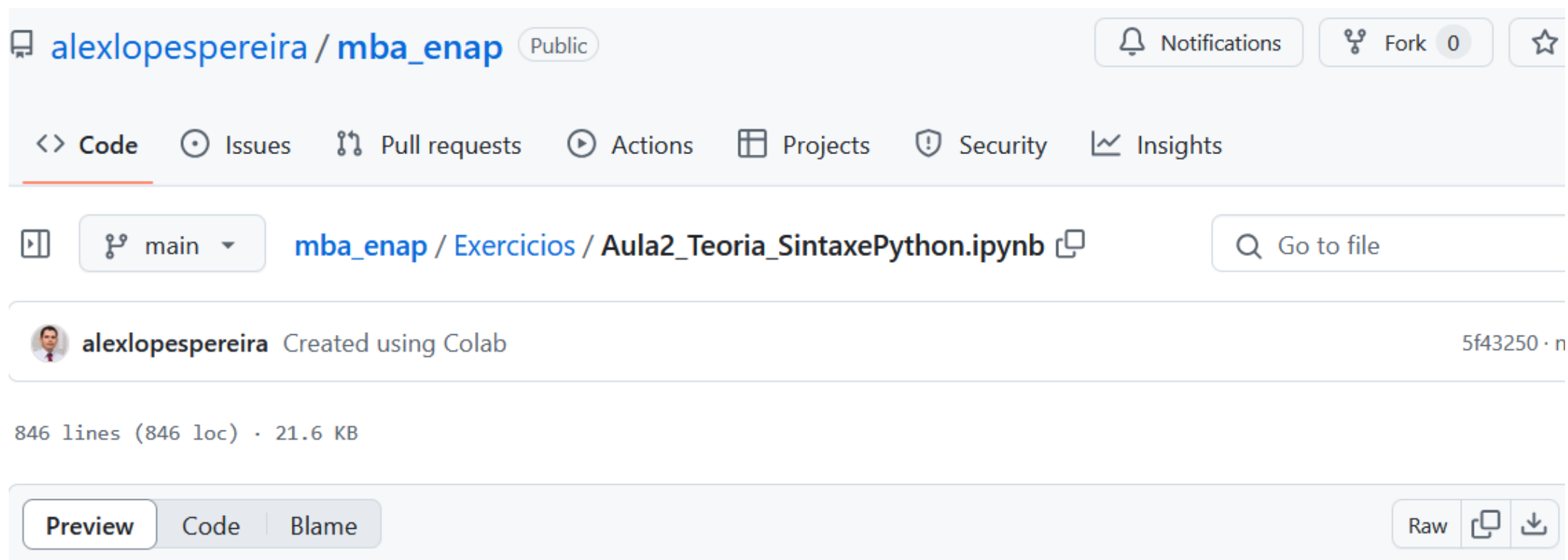
Path

Exercicios/Aula2\_Exercicio.ipynb 4

Repositório desta disciplina: [https://github.com/alexlopespereira/mba\\_enap](https://github.com/alexlopespereira/mba_enap)

# Acessando diretamente do Github

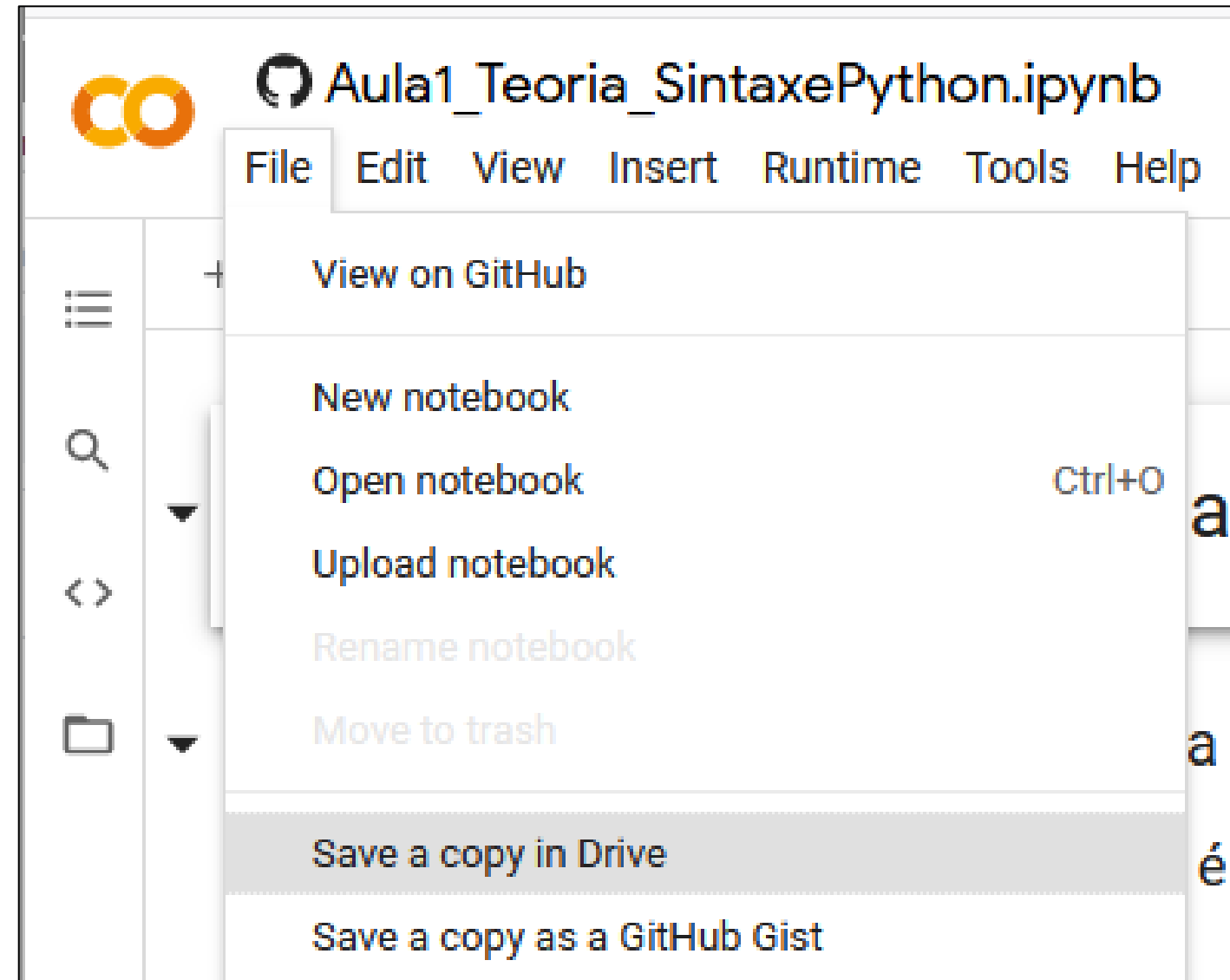
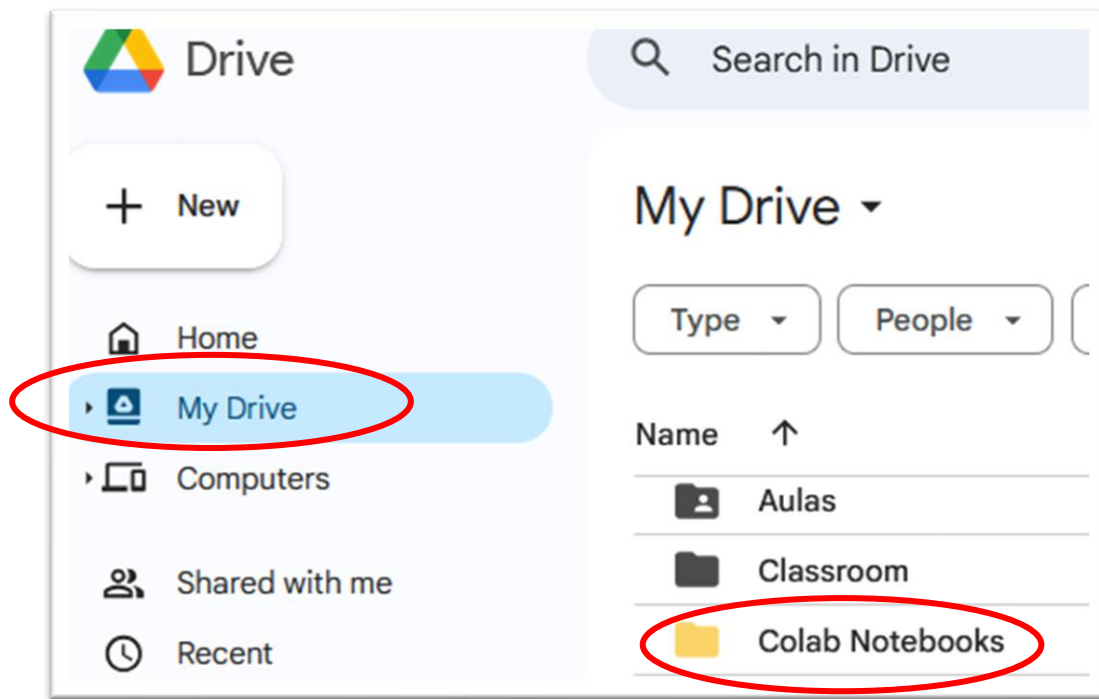
- Repositorio: [https://github.com/alexlopespereira/mba\\_enap](https://github.com/alexlopespereira/mba_enap)



Aula 2 - Explicações sobre a sintaxe da linguagem Python


# Acesso aos cadernos Colab deste Curso

- Salvar uma cópia no google drive
  - O arquivo será salvo em
    - ✓ Meu Drive > Colab Notebooks





# Colab Notebook: Como usar?

- **Preste atenção** na demonstração (*live coding*) do professor.
  - Você terá tempo para praticar sozinho.
- Interação básica com o Colab Notebook
- Clicar em *Play* ou **tecle SHIFT+ENTER** para executar uma célula
  - Os números entre colchetes indicam a ordem de execução dos comandos.
  - O ícone de Play indica que o código está sendo executado.
- Se você **reiniciar** o notebook o conteúdo das variáveis é **perdido**.
- **Leia** as mensagens de **log de erro** (elas são úteis ).

# *Atalhos de Teclado muito úteis*

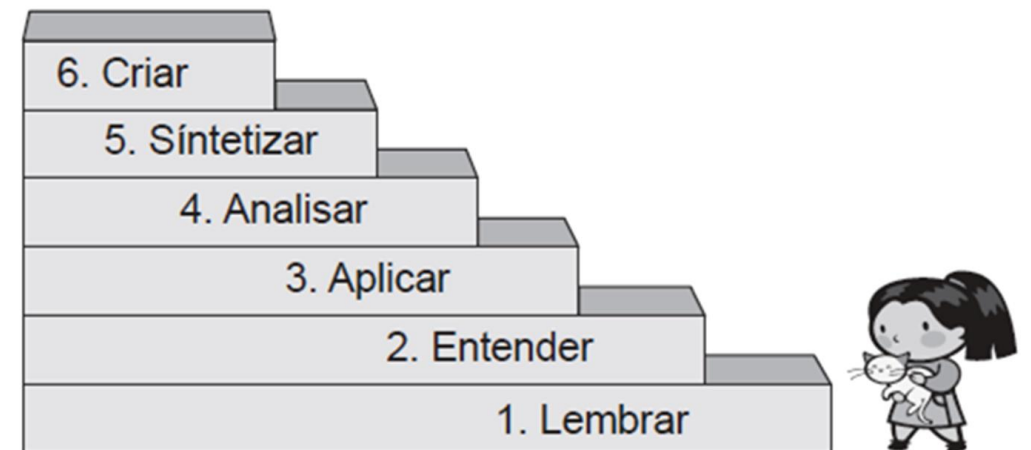
- ESC
  - Sai do modo de edição e entra no modo de comandos
    - ✓ Pode-se sair do modo de edição clicando fora das células
- SHIFT+ENTER
  - Executa a célula atual e passa o cursor para a próxima célula;
- CTRL+ENTER
  - Executa a célula atual e mantém o cursor na mesma célula;
- B (**B**elow) / A (**A**bove) – no modo de comando
  - Adiciona uma célula abaixo/acima da célula selecionada
- ↓ / ↑
  - Move células para baixo / cima
- Acessar documentação/manual

# *Metodologia das Aulas Práticas*

- Grupos de 4 alunos
  - Ajudar e ser ajudado pelos próprios colegas (*peer instruction*);
- Conectados numa sala de vídeo conferência;
- Desenvolvendo os exercícios **individualmente**
  - e tirando dúvidas entre si, se necessário;
- O monitor/professor entra na sala para sanar uma dúvida
  - registrada no canal duvidas do slack.
    - ✓ Informe sua sala na dúvida registrada no Slack
  - depois de **15min sem progresso**, é hora de chamar um monitor!

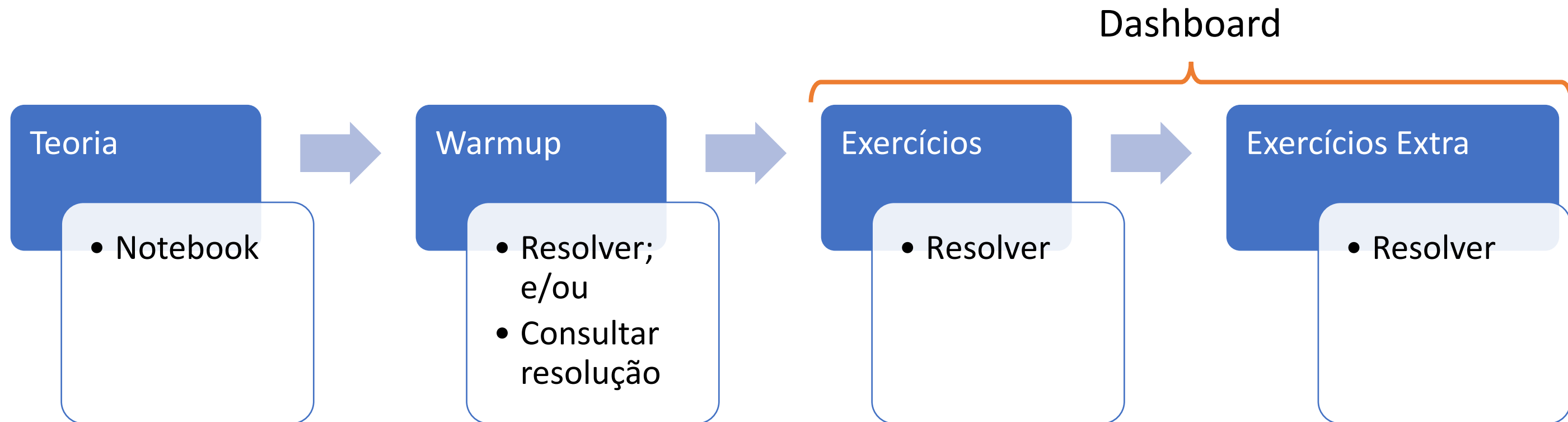
# *Cadernos (notebooks) Jupyter desse curso*

- AulaX\_Teoria\_ZZZ
    - Códigos apresentados nos slides
  - AulaX\_Warmup(\_Solucoes)
    - Exercícios básicos (elementares) de aquecimento
  - AulaX\_Exercicio
    - Exercícios práticos
  - AulaX\_Exercicio\_Extra(\_Solucoes)
    - Exercícios extra
  - Google Colab
    - Init Cell / validate()
- 1. Lembrar / 2. Entender
- 1. Lembrar / 2. Entender
- 3. Aplicar
- 3. Aplicar



# *Comece por onde lhe convier*

- 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.



# *Conceitos de Python abordados na aula*

- def
- If
- else
- while
- for
- in
- break
- return
- import
- from

