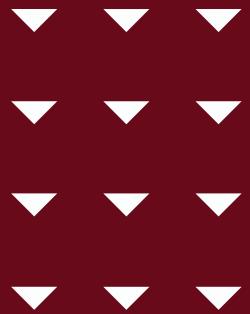
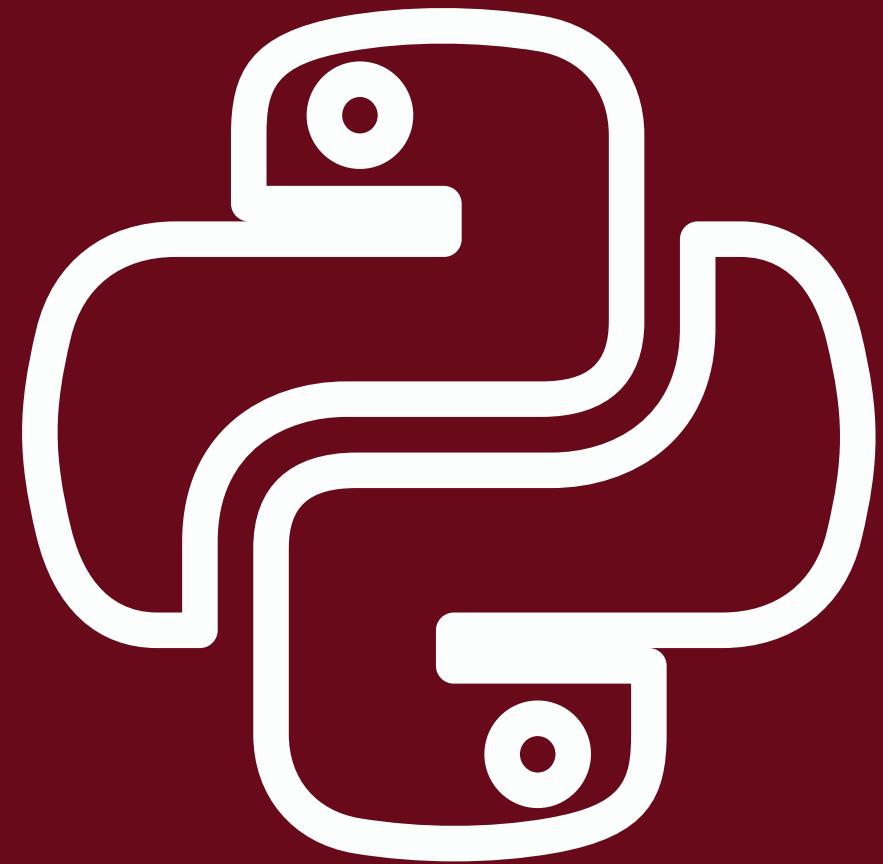
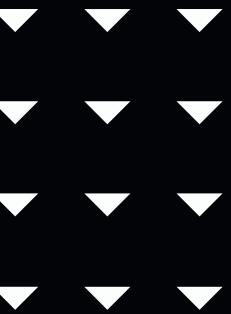


ESTRUTURA DE DADOS





CONTINUANDO

Google Colab - uma solução em nuvem.



Olá, este é o Colaboratory - Colab



← → C ⌘ colab.research.google.com

Navegação anónima

Olá, este é o Colaboratory

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda

Acesso

Login

Compartilhar

Fazer login

Conectar

Índice

Vamos começar

Ciência de dados

Machine learning

Mais recursos

Exemplos em destaque

+ Seção

Conectem-se após o login

Conheça o Colab

(Novidade) Teste a API Gemini

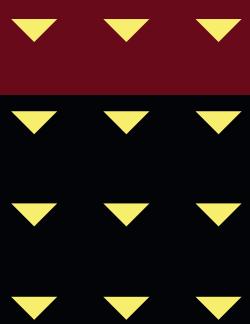
- [Generate a Gemini API key](#)
- [Talk to Gemini with the Speech-to-Text API](#)
- [Compare Gemini with ChatGPT](#)
- [More notebooks](#)

Se você já conhece bem o Colab, confira este vídeo para saber mais sobre as tabelas interativas, a visualização do histórico de código executado e o Palette de comandos.



[]

Mão no código!



Estrutura de repetição

for

Mão no código!

Estruturas de repetição - for

```
1 print(1)
2 print(2)
3 print(3)
4 print(4)
5 print(5)
```

[1] ✓ 0.0s

... 1
2
3
4
5

```
1 for numero in range(1, 6):
2     print(numero)
```

[2] ✓ 0.0s

... 1
2
3
4
5

parou 1 número antes do 6

NESTA CONFIGURAÇÃO. Deve ser um número após o almejado. Exemplo, tem que parar no 5.. então lança um 6.

Mão no código!

```
1 for numero in range(5, 0, -1):  
2     print(numero)  
[3] ✓ 0.0s
```

```
... 5  
4  
3  
2  
1
```

```
1 5 + 4 + 3 + 2 + 1  
[4] ✓ 0.0s
```

```
... 15
```

```
1 soma = 0  
2 for numero in range(1, 6):  
3     soma = soma + numero  
4     # print(soma)  
5  
6 print(soma)  
[5] ✓ 0.0s
```

```
... 15
```

É o parametro do “step”. Por padrão temos uma incrementação (+1), porém podemos arbitrar um passo diferente como esse “-1” por exemplo.

Notem que nesta configuração o número de parada não é um posterior e sim um anterior pois a lógica esta voltada para decrementação.

Mão no código!



```
1 palavra = "sorvete"
2 for letra in palavra:
3     # print(letra)
4     if letra == "v":
5         print("Achou a letra v")
✓ 0.0s
```

Achou a letra v

Mão no código!



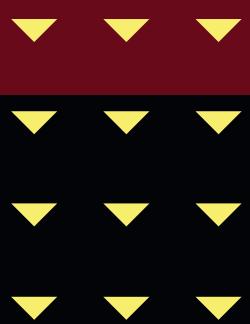
```
▶ ▾ 1 for i in range(0, 5):  
2     print(i)  
3     print("---")  
4     for j in range(0, 3):  
5         print(j)  
6     print()  
[7] ✓ 0.0s
```

```
---  
0  
1  
2  
4  
...  
0  
1  
2
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output

A estrutura acima é muito usada para matrizes!

Mão no código!



Estrutura de repetição

while



^ Estruturas de repetição - while

```
1 numero = 1
2 while numero < 6:
3     print(numero)
4     numero += 1
5 print('---')
```

Quando você trabalha com o **while** você precisa definir uma variável “fora”, colocar uma condição e sempre tem que colocar um incremento se você não colocar esse incremento ele vai entrar num looping infinito.

1
2
3
4
5

Mão no código!



```
1 numero = 5
2 while numero > 0:
3     print(numero)
4     numero -= 1
```

[]

```
... 5
4
3
2
1
```

Mão no código!



```
1 1 + 2 + 3 + 4 + 5
```

```
[ ]
```

```
... 15
```



```
1 soma = 0
2 numero = 1
3 while numero < 6:
4     soma += numero
5     numero += 1
6 print(soma)
```

```
[ ]
```

```
... 15
```

[+ Code](#)[+ Markdown](#)

```
1 numero = 12
2 while numero < 1 or numero > 10:
3     numero = int(input('Digite um número de 1 a 10: '))
```

```
[ ]
```

```
... Digite um número de 1 a 10: 5
```

Mão no código!



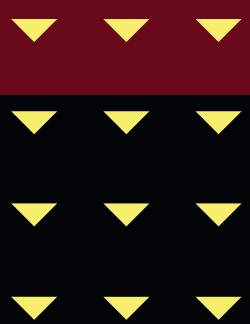
Exercícios

Mão no código!

Implemente cada exercício utilizando tanto o **for** quanto o **while**.

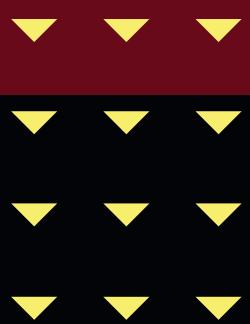
- Ler 5 notas e informar a média.
- Imprimir a tabuada do número 3
 $(3 \times 1 = 3 \text{ até } 3 \times 10 = 30)$

Mão no código!



Resolução

Mão no código!



Exercício 1

Ler 5 notas e informar a média

For

```
1 nota = media = soma = 0
[1] ✓ 0.0s
```

```
1 print(nota, media, soma)
[2] ✓ 0.0s
...
... 0 0 0
```

Mão no código!



```
1 for _ in range(1, 6): # Repete 5 vezes
2 | nota = float(input('Digite a nota '))
3 | soma += nota
[3] ✓ 15.2s
```

```
> 
1 print(soma)
[4] ✓ 0.0s
... 22.0
```

```
1 media = soma / 5
2 print('A média é ', media)
[5] ✓ 0.0s
```

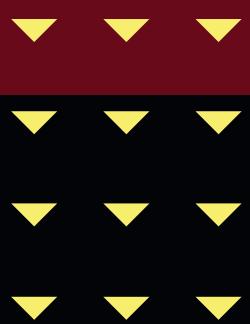
```
... A média é 4.4
```

While

```
1 nota = soma = 0
2 numero = 1
3 while numero <= 5:
4     nota = float(input('Digite a nota:'))
5     soma += nota
6     numero += 1
7 print('A média é ', soma / 5)

6] ✓ 9.8s
```

- A média é 4.4



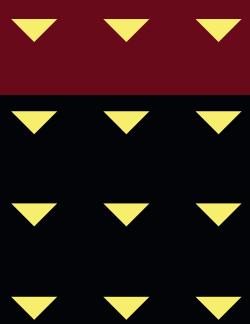
Exercício 2

Imprimir a tabuada do número 3 ($3 \times 1 = 1 - 3 \times 10 = 30$)

For

```
> <
    1 for i in range(1, 11):
    2 | print('3 x {} = {}'.format(i, 3 * i))
7] ✓ 0.0s
..
.. 3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

Mão no código!



While

```
> <
  1 numero = 1
  2 while numero <= 10:
  3     print('3 x {} = {}'.format(numero, 3 * numero))
  4     numero += 1
[8]   ✓  0.0s
```

```
.. 3 x 1 = 3
  3 x 2 = 6
  3 x 3 = 9
  3 x 4 = 12
  3 x 5 = 15
  3 x 6 = 18
  3 x 7 = 21
  3 x 8 = 24
  3 x 9 = 27
  3 x 10 = 30
```

Mão no código!



Coleções

Tuplas e Listas



Coleções

Tuplas

```
▷ ▾ 1 tupla = ("Yuji Itadori", "Megumi Fushiguro", "Nobara Kugisaki")
[104] ✓ 0.0s
```

```
1 tupla
[105] ✓ 0.0s
...
('Yuji Itadori', 'Megumi Fushiguro', 'Nobara Kugisaki')
```

```
1 tupla[0]
[106] ✓ 0.0s
...
'Yuji Itadori'
```

Mão no código!

```
1 tupla[1]
```

[107] ✓ 0.0s

```
... 'Megumi Fushiguro'
```

```
1 tupla[2]
```

[108] ✓ 0.0s

```
... 'Nobara Kugisaki'
```



```
1 tupla.index("Megumi Fushiguro") # Retorna o índice da tupla buscando pela string.
```

[109] ✓ 0.0s

```
... 1
```

Mão no código!

```
1 for elemento in tupla:  
2 |     print(elemento)
```

✓ 0.0s

Yuji Itadori
Megumi Fushiguro
Nobara Kugisaki

Mão no código!

Listas

```
1 l1 = ["Satoru Gojo", "Suguru Geto", "Shoko Ieiri"]
2 l2 = ["Maki Zenin", "Toge Inumaki", "Panda"]
[111] ✓ 0.0s
```

```
1 l3 = l1 + l2
2 print(l3)
[112] ✓ 0.0s
... ['Satoru Gojo', 'Suguru Geto', 'Shoko Ieiri', 'Maki Zenin', 'Toge Inumaki', 'Panda']
```

```
> ^
1 l2
[113] ✓ 0.0s
... ['Maki Zenin', 'Toge Inumaki', 'Panda']
```

Mão no código!

```
1 l2_2 = l2 * 2
2 print(l2_2)
114] ✓ 0.0s
```

```
... ['Maki Zenin', 'Toge Inumaki', 'Panda', 'Maki Zenin', 'Toge Inumaki', 'Panda']
```

```
1 l1[0]
115] ✓ 0.0s
```

```
... 'Satoru Gojo'
```

```
> ^
1 l1
116] ✓ 0.0s
```

```
... ['Satoru Gojo', 'Suguru Geto', 'Shoko Ieiri']
```

+ Code

+ Markdown

```
1 l1[0:2]
117] ✓ 0.0s
```

```
... ['Satoru Gojo', 'Suguru Geto']
```

Mão no código!

```
1 l1.append("Toji Fushiguro")
2 print(l1)
3] ✓ 0.0s
['Satoru Gojo', 'Suguru Geto', 'Shoko Ieiri', 'Toji Fushiguro']
```

```
1 l1.remove("Shoko Ieiri")
2 print(l1)
3] ✓ 0.0s
['Satoru Gojo', 'Suguru Geto', 'Toji Fushiguro']
```

```
1 del l1
3] ✓ 0.0s
```

Mão no código!

```
1 print(l1)
.21] ⚡ 0.0s
```

..

```
NameError
Cell In[121], line 1
----> 1 print(l1)
```

Traceback (most recent call last)

NameError: name 'l1' is not defined

```
> ▾ 1 for item in l2_2:
  2 |   print(item)
]
```

```
.. Maki Zenin
  Toge Inumaki
  Panda
  Maki Zenin
  Toge Inumaki
  Panda
```

Mão no código!



Coleções

Dicionários e Conjuntos (set)

Mão no código!

Dicionários

```
1 rank_forca = {"Ryomen Sukuna": 35, "Satoru Gojo": 32, "Mahito": 14}
```

```
[ ]
```

```
1 rank_forca["Mahito"]
```

```
[ ]
```

```
.. 14
```

Mão no código!

```
1 rank_forca["Mahoraga"] = 25
```

```
[ ]
```

```
1 print(rank_forca)
```

```
[ ]
```

```
… {'Ryomen Sukuna': 35, 'Satoru Gojo': 32, 'Mahito': 14, 'Mahoraga': 25}
```

Mão no código!

```
1 del (rank_forca)["Mahito"]
2 print(rank_forca)
]

.. {'Ryomen Sukuna': 35, 'Satoru Gojo': 32, 'Mahoraga': 25}

1 rank_forca.items()
]

.. dict_items([('Ryomen Sukuna', 35), ('Satoru Gojo', 32), ('Mahoraga', 25)])

> ^
1 rank_forca.keys()
]

.. dict_keys(['Ryomen Sukuna', 'Satoru Gojo', 'Mahoraga'])

1 rank_forca.values()
]

.. dict_values([35, 32, 25])
```

Mão no código!

```
1 rank_forca2 = {"Yuta Okkotsu": 30, "Kento Nanami": 13}  
2 print(rank_forca2)
```

[]

```
... {'Yuta Okkotsu': 30, 'Kento Nanami': 13}
```

```
1 rank_forca.update(rank_forca2)  
2 print(rank_forca)
```

[]

```
... {'Ryomen Sukuna': 35, 'Satoru Gojo': 32, 'Mahoraga': 25, 'Yuta Okkotsu': 30, 'Kento Nanami': 13}
```

▷ ▾

```
1 rank_forca.items()
```

[]

```
... dict_items([('Ryomen Sukuna', 35), ('Satoru Gojo', 32), ('Mahoraga', 25), ('Yuta Okkotsu', 30), ('Kento Nanami', 13)])
```

+ Code

+ Markdown

Mão no código!

```
1 for (
2     personagem,
3     força,
4 ) in rank_força.items(): # quer dizer que vamos percorrer o dicionário acima. Isso significa que dentro de personagem ele
5     | vai copiar o nome do personagem e da variável força ele vai copiar o valor da força do respectivo personagem... chave e
6     | valor do dicionário rank_força.
    print(f"O {personagem}, tem seu nível de força definido em: {força}")
[ ]
...
O Ryomen Sukuna, tem seu nível de força definido em: 35
O Satoru Gojo, tem seu nível de força definido em: 32
O Mahoraga, tem seu nível de força definido em: 25
O Yuta Okkotsu, tem seu nível de força definido em: 30
O Kento Nanami, tem seu nível de força definido em: 13
```



Conjuntos (set)

▷ ▾

```
1 bolo_supresa = (
2     "farinha",
3     "água",
4     "pitada de sal",
5     "ovo",
6     "pitada de sal",
7     "ovo",
8     "ovo",
9     "baunilha",
10    "manteiga",
11 )
```

[]

▷ ▾

```
1 print(bolo_supresa)
```

[]

```
... ('farinha', 'água', 'pitada de sal', 'ovo', 'pitada de sal', 'ovo', 'ovo', 'baunilha', 'manteiga')
```

Mão no código!



```
> <
1 print(
2 |     set(bolo_supresa)
3 ) # set significa conjunto. Quando trabalhamos com o set() ele traz os elementos que não se repetem
4 |
[ ]
... {'ovo', 'manteiga', 'água', 'baunilha', 'farinha', 'pitada de sal'}
```

```
1 c1 = {1, 2, 3, 4, 5} # c1 = conjunto 1
2 c2 = {3, 4, 5, 6, 7} # c2 = conjunto 2
3 c3 = c1.intersection(
4 |     c2
5 ) # c3 = interseção entre o conjunto 1 com o conjunto 2. É teoria dos conjuntos láaa da matemática básica.
[ ]
```

Mão no código!



```
1 print(c3) # interseção  
[ ]  
... {3, 4, 5}
```

```
1 c1.difference(c2) # diferença entre o c1 e c2.  
[ ]  
... {1, 2}
```

```
1 c2.difference(c1)  
[ ]  
... {6, 7}
```

Mão no código!



Matrizes

Mão no código!



Matrizes



[2]

```
1 # pip install numpy  
2 # descomente acima se for necessário caso a biblioteca numpy não estiver instalada.
```

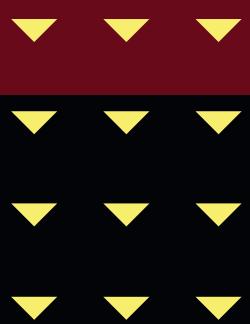
✓ 0.0s

[3]

```
1 import numpy as np
```

✓ 0.1s

Mão no código!



```
1 matriz = np.array(  
2 |   [[2, 3, 1], [4, 5, 7]]  
3 ) # array é como se fosse uma lista um pouco melhorada. Matriz (2x3) criada, 2 linhas e 3 colunas.
```

[4] ✓ 0.0s

```
1 matriz # interessante falar que os indices começam em 0.
```

[5] ✓ 0.0s

```
... array([[2, 3, 1],  
        [4, 5, 7]])
```

▷ ▾

```
1 matriz.shape # shape significa formato que no caso é: (linhas, colunas)
```

[6] ✓ 0.0s

```
... (2, 3)
```

Mão no código!



```
1 matriz[  
2 |   0  
3 ] # acessanando a matriz na posição zero... retornou a primeira linha (índice zero)  
[7] ✓ 0.0s  
... array([2, 3, 1])
```

```
1 matriz[1] # acessanando a matriz na posição um... retornou a segunda linha (índice 1)  
[8] ✓ 0.0s  
... array([4, 5, 7])
```

```
▷ ~  
1 matriz[0][  
2 |   2  
3 ] # para termos o retorno das colunas, precisamos de 2 colchetes. Por exemplo... o primeiro colchete indica LINHA e o  
segundo colchete indica COLUNA. Lembrem-se de trabalhar com índices.  
[9] ✓ 0.0s  
... 1
```

Mão no código!



```
1 matriz[1][2]
```

[10] ✓ 0.0s

... 7

```
1 for i in range(matriz.shape[0]): # shape[0] retorna o número de linhas
2     print(matriz[i])
3     for j in range(matriz.shape[1]): # shape[1] retorna o número de colunas
4         print(matriz[i][j])
```

[13] ✓ 0.0s

... [2 3 1]

2

3

1

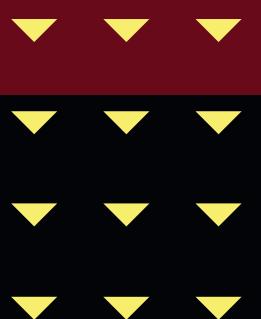
[4 5 7]

4

5

7

Mão no código!



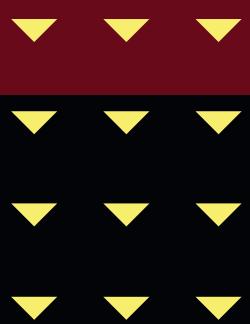
Exercícios

Mão no código!

- 1. Lista:** Crie uma estrutura de repetição para fazer a leitura de 5 números inteiros e os armazene dentro de uma lista. Após a leitura, crie outra estrutura de repetição para somar todos os valores digitados.
- 2. Dicionário:** Crie um dicionário para armazenar o nome e a idade de 3 piratas, fazendo a leitura dos valores por meio de uma estrutura de repetição. Depois, crie uma nova estrutura de repetição para somar todas as idades e retornar a média.
- 2. Matriz:** Dada a matriz abaixo, construa uma estrutura de repetição para percorrer e somar todos os elementos da matriz.

```
1 | matriz = np.array([[3, 4, 1],  
2 |                      [3, 1, 5]])
```

Mão no código!



Resolução

Mão no código!

Exercício 1 - listas

Crie uma estrutura de repetição para fazer a leitura de 5 números inteiros e os armazene dentro de uma lista. Após a leitura, crie outra estrutura de repetição para somar todos os valores digitados

```
1 lista = [] # lista vazia
2 for _ in range(1, 6): # 5 iterações
3     valor = int(input("Digite o valor: ")) # leia o valor
4     lista.append(valor) # adicione o valor a lista
```

[1] ✓ 10.4s

Python

Mão no código!

```
1 lista
```

✓ 0.0s

Py

```
[20, 34, 22, 24, 56]
```

```
1 len(lista) # tamanho da lista ... lembrem-se do índice.
```

✓ 0.0s

Py

5

```
1 soma = 0
2 for i in range(
3 |   len(lista)
4 ): # 0, 1, 2, 3, 4... poderíamos ter colocado o range(1, 6) mas quis usar o len para o script ler
5 |   automagicamente o tamanho da lista.
6 |   # print(lista[i])
7 |   soma += lista[
8 |     i
9 |     ] # acessa cada um dos elementos e variável soma recebe a própria soma como também o item da lista.
9 print("Soma: ", soma)
```

✓ 0.0s

Py

Soma: 156

Mão no código!

```
1 import numpy as np # importe o numpy para o script
2
3 np.array(
4 |     lista
5 ).sum() # Estamos convertendo a lista para o formato do numpy array e com esse formato temos acesso a
       algumas funções matemáticas como soma, média e por ai vai.
6 # tente usar o lista.sum() e perceberá um erro pois não esta usando a biblioteca do numpy.
```

✓ 0.0s

Mão no código!

Exercícios 2 - dicionários

Crie um dicionário para armazenar o nome e a idade de 3 piratas, fazendo a leitura dos valores por meio de uma estrutura de repetição. Depois, crie uma nova estrutura de repetição para somar todas as idades e retornar a média

```
1 piratas = {}
2 for _ in range(1, 4):
3     nome = input("Digite o nome: ")
4     idade = float(input("Digite a idade: "))
5     piratas[nome] = idade
```

✓ 27.7s

Py

```
1 piratas
```

✓ 0.0s

Py

```
· {'Monkey D. Luffy': 19.0, 'Roronoa Zoro': 21.0, 'Nami': 20.0}
```

Mão no código!

```
1 piratas["Nami"]  
:] ✓ 0.0s
```

20.0

```
1 piratas.values()  
:] ✓ 0.0s
```

dict_values([19.0, 21.0, 20.0])

```
1 soma = 0  
2 for idade in piratas.values():  
3     # print(nota)  
4     soma += idade  
5 print("Média: ", soma / 3)  
:] ✓ 0.0s
```

Média: 20.0

Mão no código!

Exercício 3: Matriz

Dada a matriz abaixo, construa uma estrutura de repetição para percorrer e somar todos os elementos da matriz

```
1 matriz = np.array([[3, 4, 1], [3, 1, 5]])
2 matriz
```

```
array([[3, 4, 1],
       [3, 1, 5]])
```

```
1 matriz.shape
```

```
(2, 3)
```

Mão no código!



```
1 soma = 0
2 for i in range(matriz.shape[0]):
3     for j in range(matriz.shape[1]):
4         # print(matriz[i][j])
5         soma += matriz[i][j]
6 print("Soma: ", soma)
]
```

Soma: 17

Mão no código!



Funções



Funções

- Trechos de programa que recebem um determinado nome e podem ser chamados várias vezes durante a execução
- Principais vantagens: reutilização de código, modularidade e facilidade de manutenção do sistema



Função sem parâmetro e sem retorno

```
1 def mensagem():
2     print('Texto da função')
]
✓ 0.0s
```

```
1 mensagem()
2 mensagem()
3 mensagem()
]
✓ 0.0s
```

Texto da função
Texto da função
Texto da função

Mão no código!

Função com passagem de parâmetro

```
1 def mensagem(texto):  
2     print(texto)  
✓ 0.0s
```

```
1 mensagem('texto 1')  
2 mensagem('texto 2')  
3 mensagem('texto 3')  
✓ 0.0s
```

texto 1
texto 2
texto 3

Mão no código!

```
1 def soma(a, b):  
2     print(a + b)  
✓ 0.0s
```

```
1 soma(2, 3)  
✓ 0.0s
```

5

```
1 soma(3, 3)  
2 soma(1, 2)  
✓ 0.0s
```

6

3

Mão no código!

Função com passagem de parâmetros e retorno

```
1 def soma(a, b):  
2     return a + b  
✓ 0.0s
```

```
1 soma(3, 2)  
✓ 0.0s
```

5



```
1 # r = 7  
2 r = soma(3, 2) # a variável r recebe o retorno da função soma  
3 print(r)
```

```
✓ 0.0s
```

5

Mão no código!

```
1 def calcula_energia_potencial_gravitacional(m, h, g = 10):
2     '''
3         Calcula a energia potencial gravitacional
4         Argumentos:
5             m: massa, entrada como uma variável float
6             h: altura, entrada como uma variável float
7
8         Argumento opcional:
9             g: aceleração gravitacional, com valor default de 10
10            '''
11            e = g * m * h
12            return e
```

✓ 0.0s

```
1 calcula_energia_potencial_gravitacional(30, 12)
```

✓ 0.0s

Mão no código!

```
1 calcula_energia_potencial_gravitacional(30, 12, 9.8)
[13] ✓ 0.0s
...
3528.0

> ^
1 help(calcula_energia_potencial_gravitacional)
[14] ✓ 0.0s
...
Help on function calcula_energia_potencial_gravitacional in module __main__:

calcula_energia_potencial_gravitacional(m, h, g=10)
    Calcula a energia potencial gravitacional
    Argumentos:
        m: massa, entrada como uma variável float
        h: altura, entrada como uma variável float

    Argumento opcional:
        g: aceleração gravitacional, com valor default de 10
```

Mão no código!



Exercícios

Mão no código!

1. Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão é $F = (9 * C + 160) / 5$, na qual F é a temperatura em Fahrenheit e C é a temperatura em graus Celsius
 - Função para ler e retorna o valor da temperatura (não recebe parâmetro)
 - Função para fazer o cálculo (recebe como parâmetro a temperatura em graus Celsius)
 - Função para mostrar o resultado, recebendo como parâmetro o valor e fazendo a impressão

Mão no código!

2. Efetuar o cálculo da quantidade de litros de combustível gasto em uma viagem, utilizando um automóvel que faz 12 Km por litro. Para obter o cálculo, o usuário deve fornecer o tempo gasto na viagem e a velocidade média durante ela. Desta forma, será possível obter a distância percorrida com a fórmula $DISTANCIA = TEMPO * VELOCIDADE$. Tendo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem, com a fórmula: $LITROS_USADOS = DISTANCIA / 12$. O programa deve apresentar os valores da velocidade média, tempo gasto na viagem, a distância percorrida e a quantidade de litros utilizada na viagem
1. Função para ler os valores (não recebe parâmetro e retorna os dois valores)
 2. Função para calcular a distância (recebe como parâmetro o tempo e a velocidade e retorna a distância)
 3. Função para calcular a quantidade de litros (recebe como parâmetro a distância e retorna os litros)
 4. Função para apresentar o resultado (recebe como parâmetro os valores e somente imprime o resultado)

Mão no código!



Resolução

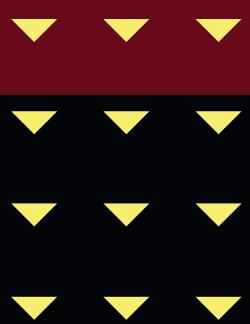


Exercício 1

Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão é $F = (9 * C + 160) / 5$, na qual F é a temperatura em Fahrenheit e C é a temperatura em graus Celsius

- Função para ler e retorna o valor da temperatura (não recebe parâmetro)
- Função para fazer o cálculo (recebe como parâmetro a temperatura em graus Celsius)
- Função para mostrar o resultado, recebendo como parâmetro o valor e fazendo a impressão

Mão no código!



```
1 def ler_temperatura():
2     temperatura = float(input('Digite a temperatura em graus Celsius: '))
3     return temperatura
```

1] ✓ 0.0s

```
1 def converter(temperatura_celsius):
2     temperatura_f = (9 * temperatura_celsius + 160) / 5
3     return temperatura_f
```

2] ✓ 0.0s

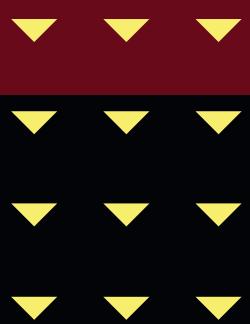
```
1 def mostrar(temperatura_f):
2     print(temperatura_f)
```

3] ✓ 0.0s

```
1 temperatura_c = ler_temperatura()
2 temperatura_f = converter(temperatura_c)
3 mostrar(temperatura_f)
```

4] ✓ 4.1s

• 95.0



Exercício 2

Efetuar o cálculo da quantidade de litros de combustível gasto em uma viagem, utilizando um automóvel que faz 12 Km por litro. Para obter o cálculo, o usuário deve fornecer o tempo gasto na viagem e a velocidade média durante ela. Desta forma, será possível obter a distância percorrida com a fórmula $DISTANCIA = TEMPO * VELOCIDADE$. Tendo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem, com a fórmula: $LITROS_USADOS = DISTANCIA / 12$. O programa deve apresentar os valores da velocidade média, tempo gasto na viagem, a distância percorrida e a quantidade de litros utilizada na viagem

- Função para ler os valores (não recebe parâmetro e retorna os dois valores)
- Função para calcular a distância (recebe como parâmetro o tempo e a velocidade e retorna a distância)
- Função para calcular a quantidade de litros (recebe como parâmetro a distância e retorna os litros)
- Função para apresentar o resultado (recebe como parâmetro os valores e somente imprime o resultado)

Mão no código!

```
1 def leitura():
2     tempo = float(input('Digite o tempo da viagem: '))
3     velocidade = float(input('Digite a velocidade média: '))
4     return tempo, velocidade
```

✓ 0.0s

```
1 def calcula_distancia(tempo, velocidade):
2     return tempo * velocidade
```

✓ 0.0s

```
1 def calcula_litros(distancia):
2     return distancia / 12
```

✓ 0.0s

```
1 def imprime(velocidade, tempo, distancia, litros):
2     print('Velocidade:', velocidade)
3     print('Tempo:', tempo)
4     print('Distância:', distancia)
5     print('Litros:', litros)
```

✓ 0.0s

Mão no código!

```
1 t, v = leitura()
2 d = calcula_distancia(t, v)
3 l = calcula_litros(d)
4 imprime(v, t, d, l)

] ✓ 8.0s
```

Velocidade: 80.0

Tempo: 4.0

Distância: 320.0

Litros: 26.666666666666668

That's all Folks!

ATÉ A PRÓXIMA!