

deixar o pycharm sem linhas embaixo

File / Settings / Inspections / Proofreading / Typo

instalando jupyter e anaconda

faça download do anaconda: <https://www.anaconda.com/products/individual>

install for: just me ☒ / se der errado coloque all users ☒ > install

para ver se instalou msm procure no seu computador por: jupyter notebook

abra o jupyter no seu computador > downloads > new folder

untitled folder > renomeie essa pasta para 'jupyter'

new > python3

instale no terminal da anaconda (anaconda terminal):

pip install pyautogui

pip install pyperclip

ctrl + enter executa o programa do jupyter

shift + enter cria uma nova linha de código no jupyter

adicionando novo valor para uma variável

dica: quando atualizar um valor de uma variável é necessário colocar adicionar o cálculo novamente para calcular:

exemplo:

```
faturamento = 1000
```

```
custo = 500
```

```
lucro = faturamento - custo
```

```
print(lucro)
```

```
custo = 700
```

```
lucro = faturamento - custo
```

```
print(lucro)
```

usar para a string

são usados dentro do {} da formatação da string

:< Alinha o texto à esquerda (se tiver espaço na tela para isso)

:> Alinha o texto à direita (se tiver espaço na tela para isso)

:^ Alinha o texto ao centro (se tiver espaço na tela para isso)

:+ Coloca o sinal sempre na frente do número (independente se é positivo ou negativo)

;, Coloca a vírgula como separador de milhar

:_ Coloca o _ como separador de milhar

:e Formato Científico

:f Número com quantidade fixa de casas decimais

:x Formato HEX minúscula (para cores)

:X Formato HEX maiúscula (para cores)

:% Formato Percentual

exemplo de alinhamento

```
email = 'fernanda@gmail.com'
```

```
print('Meu e-mail não é {:.^50}, show?'.format(email))
```

resposta: Meu e-mail não é fernanda@gmail.com , show?

exemplo edição de sinal

```
custo = 500
faturamento = 270
lucro = faturamento - custo
print('Faturamento foi {:+} e lucro foi {:+}'.format(faturamento, lucro))
```

resposta: Faturamento foi +270 e lucro foi -230

exemplo separador de milhar

```
custo = 5000
faturamento = 2700
lucro = faturamento - custo
print('Faturamento foi {:+,} e lucro foi {:+,}'.format(faturamento, lucro))
```

resposta: Faturamento foi +2,700 e lucro foi -2,300

exemplo decimais fixas

```
custo = 500
faturamento = 270
lucro = faturamento - custo
print('Faturamento foi {:.2f} e lucro foi {:.1f}'.format(faturamento, lucro))
```

resposta: Faturamento foi 270.00 e lucro foi -230.0

exemplo percentual

```
custo = 500
faturamento = 1300
lucro = faturamento - custo
margem = lucro / faturamento
print('Margem de lucro foi de {:.2%}'.format(margem))
```

Margem de lucro foi de 61.54%

exemplo combinações de formatos

```
custo = 5000
faturamento = 27000
lucro = faturamento - custo
print('Faturamento foi R${:,.2f} e lucro foi R${:,.2f}'.format(faturamento, lucro))
```

```
#transformando no formato brasileiro
lucro_texto = 'R${:,.2f}'.format(lucro)
print(lucro_texto.replace('.', ',').replace('_', ''))
```

resposta: Faturamento foi R\$27,000.00 e lucro foi R\$22,000.00
R\$22.000,00

exemplo round - serve para arredondar um número

```
imposto = 0.15758
preco = 100
valor_imposto = round(preco * imposto, 1)
print('Imposto sobre o preço é de {}'.format(valor_imposto))
```

Imposto sobre o preço é de 15.8

casefold (deixa em letra minúscula)

```
Uso:
    texto = 'Lira'
    print(texto.casefold())
Resultado:
    'lira'
```

endswith (verifica se um texto termina com certo argumento)

```
Uso:
    texto = 'lira@gmail.com'
    print(texto.endswith('gmail.com'))
Resultado:
    True
```

find (encontra o índice de um argumento)

```
Uso:
    texto = 'lira@gmail.com'
    print(texto.find('@'))
Resultado:
    4
```

Obs: lembrando como funciona a posição nas strings, então o @ está na posição 4

```
l i r a @ g m a i l . c o m
0 1 2 3 4 5 6 7 8 9 10 11 12 13
```

splitlines (separa um texto de outros textos de acordo com os 'enters' do texto)

```
Uso:
    texto = """Olá, bom dia
    Venho por meio desse e-mail lhe informar o faturamento da loja no dia de hoje.
    Faturamento = R$2.500,00
    """
    print(texto.splitlines())
Resultado:
    ['Olá, bom dia', 'Venho por meio desse e-mail lhe informar o faturamento da loja no dia de
    hoje.', 'Faturamento = R$2.500,00']
```

startswith (verifica se um texto começa com uma palavra)




```
Uso:
    texto = 'BEB123453'
    print(texto.startswith('BEB'))
Resultado:
    True
```


criando um ambiente virtual

para criar ambiente virtual:
procure por 'cmd'
digite no terminal:
pip install --upgrade pip
pip install virtualenv virtualenvwrapper-win

procure por 'sistema' > configurações avançadas do sistema > variáveis de ambiente > novo

nome da variável: WORKON_HOME
valor da variável: C:\Users\Maki\Envs

   aqui vai ser o novo nome de usuário
aqui vai ficar o nome do seu usuário do pc

digite no terminal: mkvirtualenv guppe
  nome do ambiente virtual

se aparecer abaixo o nome do ambiente virtual, significa que ele foi criado:
(guppe) + caminho


carta do python

no seu terminal digite:
python
import this

aparecerá uma carta escrito


pep8 - boas práticas


o pep8 são propostas de melhoria para o python


 Camelcase: a primeira letra do nome da classe é maiúscula e se tiver 2 palavras, as duas primeiras letras são maiúsculas e são juntas


```
class Calculadora:  
    pass
```

```
class CalculadoraCientifica  
    pass
```

 para funções, os nomes são todos em minúsculos, sem espaço, se quiser separar as palavras use underline

 para indentação utilize 4 espaços (não utilize tabs)

 depois de criar uma classe pula 2 linhas em branco

 quando for importar uma biblioteca com várias coisas dentro coloque entre parênteses, exemplo:

```
from types import (  
    StringType,  
    ListType,  
    SetType,  
    OutroType  
)
```

📁 os imports vem sempre primeiro

📁 sempre finalize o programa com uma linha em branco

📁 comentários de uma linha: #
comentários de mais de uma linha: """ """

dir e help

dir: apresenta todos os atributos/propriedades e funções e métodos para determinado tipo de dado ou variável

dir(tipo de dado/variável)
👉 exemplo: dir('university')

help: apresenta a documentação de como usá-los
help(tipo de dado/variável.opção escolhida)

👉 exemplo:
help('university'.title)

tipo primitivo

✓ tipo primitivo

+ soma

- subtração

* multiplicação

/ divisão

** potência

// é o resultado de uma divisão sem a vírgula

% é o resto, se o resto for igual a 1, o número é ímpar, se for igual a 0 o número é par

== igual

✓ a ordem de precedência (que precisa fazer o cálculo primeiro) é:

1- () parênteses

2- ** potência

3- * multiplicação, / divisão, // resultado da divisão inteira, % resto da divisão inteira (o que aparecer primeiro é o que vai calcular primeiro)

+ adição e - subtração

🌀 (com o type conseguimos descobrir qual é o tipo primitivo)

num = 5

type(num)

para números muito grandes, para facilitar nossa leitura podemos separar por underline: (o exemplo abaixo é um milhão)

1_000_000

🌀 Para limpar o terminal: Ctrl + L

🌀 Float: Tipo real, decimal

As casas decimais são separadas por ponto

Cuidado ao fazer conversão do tipo float para inteiro, porque perde a precisão

✿ É possível fazer dupla atribuição:
valor1, valor2 = 12, 54
- valor1 = 12
- valor 2 = 54

✿ É possível trabalhar com números complexos, basta adicionar o valor 'j':
variável = 5j
type(variável)

✿ Tipo booleano: True (verdadeiro) ou False (falso)
- Sempre apresentam a inicial maiúscula

✿ Vai mostrar que não está ativo:
ativo = False
print(ativo)

✿ Usando a negação 'not'. Vai mostrar que o usuário está ativo (True):
print(not ativo)

✿ Ou (or): -> Os dois precisam ser falso para ser False, de resto é sempre True
True or True: True
True or False: True
False or True: True
False or False: False

✿ Exemplo (A resposta vai ser True):
ativo = True
logado = False
print(ativo or logado)

✿ E (and): -> Os dois precisam ser verdadeiro para ser True, de resto é sempre False
True and True: True
True and False: False
False and True: False
False and False: False

✿ String: é tudo que estiver entre aspas simples, aspas duplas, aspas simples triplas, aspas simples triplas, aspas duplas triplas

✿ Se você tem o nome: Gina's Bar, por ter aspas simples, ela fica dentro de aspas duplas

```
nome = "Gina's Bar"
```

✿ Para pular uma linha usamos o \n
nome = 'Fernanda \n Maki Hirose'

✿ Para mostrar uma aspa no programa (\\" vai mostrar):
nome = 'Fernanda \\' Maki Hirose'

✿ Deixar letras em maiúsculas:
nome = "Fernanda's Bar"
print(nome.upper())

✿ Deixar letras em minúsculas:
nome = "Fernanda's Bar"
print(nome.lower())

✿ Transforma em uma lista de strings (vai separar cada letras e espaço)
nome = "Fernanda's Bar"
print(nome.split())

✿ Slice de string:
nome = "Fernanda's Bar"
print(nome[0:8]) -> Vai pegar o nome Fernanda, vai do 0 até o 7

print(nome.split()[0]) -> Vai pegar o nome Fernanda's

print(nome[::-1]) -> Vai inverter o valor da variável

✿ Substituir algo:
print(nome.replace('F', 'G'))

✿ saber se é numérico (o resultado vai ser True ou False)
(não coloque o tipo primitivo)
n = input('Digite algo: ')
print(n.isnumeric())

✿ saber se é letra (o resultado vai ser True ou False)
(não coloque o tipo primitivo)
n1 = input('Digite algo: ')
print(n1.isalpha())

✿ saber se é letra e numérico (o resultado vai ser True ou False)
isspace() -> só tem espaços
istitle() -> está capitalizada
isupper() -> está em maiúsculas
isnumeric() -> é numérico
isspace() -> só tem espaços
islower() -> está em minúsculas

biblioteca

📖 a biblioteca 'math' apresenta funcionalidades matemáticas extras:
ceil > arredonda os números para cima
floor > arredonda os números para baixo
trunc > vai tirar da vírgula para frente
pow > semelhante aos 2 asteriscos
sqrt > calcula a raiz quadrada
factorial > calcula o fatorial

📖 para importar a biblioteca math: import math
importar somente uma funcionalidade: from math import sqrt
para importar mais de 1 funcionalidade: from math import sqrt, factorial

cores

entre a barra e o m ficarão os códigos de cores:

😊 estilos: 0 (sem estilo nenhum), 1 (negrito), 4 (sublinhado), 7 (vai inverter)

😊 texto: 30 (branco), 31 (vermelho), 32 (verde), 33 (amarelo), 34 (azul), 35 (roxo), 36 (azul claro), 37 (cinza)

😊 cores de fundo do python: 40 (branco), 41 (vermelho), 42 (verde), 43 (amarelo), 44 (azul), 45 (roxo), 46 (azul claro), 47 (cinza)

exemplo: \33[0:33:44m

escopo de variável

✓ Escopo global: são variáveis que foram definidas e podem ser usadas em todo o programa

✓ Escopo local: são variáveis que não podem ser usadas em todo o programa, porque provavelmente foram definidas dentro de um bloco

👉 Exemplo 1:

```
numero = 42 # Exemplo de variável global
print(numero) # Vai mostrar o número na tela
print(type(numero)) # Vai mostrar o tipo primitivo do número
```

```
if numero > 10:
    novo = numero + 10 # A variável novo está declarada localmente dentro do bloco if. Portanto,
    # é local
    print(novo)
```

condicionais: ✓ if, else, elif

estruturas lógicas:

✓ Estruturas lógicas: and, or, not, is

Operadores unários: not

Operadores binários: and, or, is

Para o and os dois valores precisam ser iguais, para o or um ou outro valor precisa ser true ou false

O not é o booleano invertido

👉 Exemplo 1:

```
ativo = True
logado = True
```

```
if ativo and logado:
    print('Bem-vindo usuário!')
else:
    print('Você precisa ativar a sua conta, chegue seu e-mail')
```

👉 Exemplo 2:

```
ativo = True
if ativo not True:
    print('Você precisa ativar a sua conta, cheque seu e-mail')
else:
    print('Bem-vindo usuário')
```


👍 Exemplo 3:

```
ativo = True  
logado = False
```

```
if ativo:  
    print('Bem-vindo usuário')  
else:  
    print('Você precisa ativar a sua conta, cheque seu e-mail')  
print(ativo is True)
```

loop for

se só tiver o segundo valor, significa que vai de 0 até esse número
se tiver o primeiro valor, significa que vai desse número até o segundo número
pode ter o terceiro valor que é o passo, de quanto em quanto
pode ser na ordem decrescente, o primeiro valor é o valor máximo, o segundo valor é o valor mínimo, e o terceiro valor é o valor negativo de quanto em quanto

lembrando que o primeiro item tem valor 0

o enumerate pega o índice e o valor de uma string, lista, tupla, dicionário, etc

🔗 -- Exemplo 1 - Iterando uma string: --
nome = 'Fernanda'

```
for letra in nome:  
    print(letra, end="")
```

(Vai mostrar as letras do nome 'Fernanda', para não deixar uma embaixo da outra utilizou-se o end="")

🔗 -- Exemplo 2 - Iterando uma lista: --
lista = [1, 3, 5, 7, 9]
numeros = range(1, 10)

```
for numero in lista:  
    print(numero)
```

(Vai mostrar os números da lista um embaixo do outro)

🔗 -- Exemplo 3 - Iterando um range: --
for numero in range(1, 10):
 print(numero)

(Vai mostrar do número 1 até o 9)
range(valor inicial, valor final - 1)

🔗 -- Exemplo 4 - Enumerate para pegar o índice ou o valor --

```
nome = 'Fernanda'  
for i, v in enumerate(nome):  
    print(v)
```

(i = índice
v = valor

vai ver o índice ou o valor de nome
no print pediu para ver o valor, podemos substituir também pelo 'i' para ver o índice)

🔗 -- Exemplo 5 - Enumerate para pegar o valor --

```
nome = 'Fernanda'
for valor in enumerate(nome):
    print(valor)
```

🔗 -- Exemplo 6 - Perguntando quantas vezes quer rodar + calculando a soma dos números informados --

```
quantidade = int(input('Quantas vezes o loop deve rodar? '))
soma = 0
```

```
for n in range(1, quantidade + 1):
    num = int(input(f'Informe o {n}/{quantidade} valor: '))
    soma += 1
print(f'A soma é {soma}')
```

🔗 -- Exemplo 7 - Unicode --

<https://emojipedia.org/>

Original: U+1F605

Modificado: U0001F605 (adicione U000 + o código unicode sem o U+)

```
for _ in range(3):
    for num in range(1, 10):
        print('\U0001F605' * num)
```

- 1- O primeiro for vai repetir 3 vezes
- 2- O segundo for vai de 1 até 9, os quais vão se repetir 3 vezes
- 3- Vai mostrar o emoji modificado vezes o número de vezes do num)

🔗 Exemplo 8 - pegando o índice e o valor com o enumerate

```
lista = ['Fernanda', 'Julia', 'Maria']
for i, v in enumerate(lista):
    print(f'O índice -> {i} : {v}')
```

🔗 -- DICAS/Pycharm --

- 1- Ctrl + clique no print = você vai ver a documentação do print
- 2- Quando não queremos adicionar um valor, podemos ignorar ele com underline

ranges

Ranges são utilizados para gerar sequências numéricas de maneira específica

- 1- Se não especificar onde ele vai começar, ele sempre vai começar em 0
- 2- Se você especificar o número, ele vai começar do número até o número final
- 3- Temos o passo especificado pelo usuário
- 4- Temos a forma inversa, tem o valor final primeiro, depois o valor inicial e o passo negativo
- 5- lembrando que o primeiro número começa na posição 0

🔗 Exemplo 1:

```
for num in range(11):
    print(num)
```

Vai mostrar do número 0 até o 10

🕒 Exemplo 2:
for num in range(1, 11):
 print(num)

Vai mostrar do número 1 até o 10

🕒 Exemplo 3:
for num in range(1, 11, 2):
 print(num)

Vai mostrar do 1 até o 10 de 2 em 2

🕒 Exemplo 4:
for num in range(10, 0, -1):
 print(num)

sair com o break:

Utilizamos o break para sair do loop

⚙️ Exemplo 1 - Com range:

```
for numero in range(1, 11):  
    if numero == 6:  
        break  
    else:  
        print(numero, end=' ')  
print('\nSai do loop')
```

⚙️ Exemplo 2 - Com while:

```
while True:  
    comando = input("Digite 'sair' para sair: ")  
    if comando == 'sair':  
        break
```

listas:

🌐 Podemos adicionar qualquer valor nas listas, as listas aceitam valores repetidos:

```
lista1 = [1, 2, 3, 4, 5] -> Lista de números  
lista2 = ['F', 'e', 'r', 'n', 'a', 'n', 'd', 'a'] -> Lista de strings  
lista3 = [] -> Lista vazia  
lista4 = list(range(11)) -> Adiciona valores de 0 a 10 na lista
```

🌐 Podemos checar se determinado valor está dentro de uma lista:

```
lista1 = [1, 2, 3, 4, 5]  
if 4 in lista1:  
    print('Encontrei o número 4')  
else:  
    print('Não encontrei o número 4')
```

🌐 Podemos ordenar uma lista - Ordena os números, na ordenação das strings as palavras que tiverem a primeira letra maiúscula vem primeiro:

```
lista1 = [2, 9, 3, 1, 5]  
lista1.sort()
```

```
print(lista1)
```

🌐 Ordenar uma lista na ordem decrescente

para colocar um valor na ordem decrescente:

```
valores = [8, 7, 6, 5, 4, 3, 2, 1]
```

```
valores.sort(reverse=True)
```

🌐 Podemos contar quantas vezes um determinado item aparece em uma lista:

```
lista2 = ['F', 'e', 'r', 'n', 'a', 'n', 'd', 'a']
```

```
print(lista2.count('n'))
```

🌐 Podemos adicionar um elemento na lista com o append, obs: só podemos adicionar um elemento por vez:

```
lista2 = ['F', 'e', 'r', 'n', 'a', 'n', 'd', 'a']
```

```
lista2.append('Olá')
```

```
print(lista2)
```

🌐 Para adicionarmos mais de 1 elemento na lista:

```
lista1 = [2, 9, 3, 1, 5]
```

```
lista1.append([99, 100]) -> Adiciona uma lista dentro de uma lista
```

```
print(lista1)
```

🌐 Para perguntarmos se mais de um elemento está na lista utilizamos []:

```
lista1 = [2, 9, 3, 1, 5]
```

```
if [99, 100] in lista1:
```

```
    print('Encontrei os valores')
```

```
else:
```

```
    print('Não encontrei os valores')
```

🌐 Para adicionar mais de 1 elemento na lista sem adicionar uma lista dentro de uma lista, o extend não aceita valor único:

```
lista1 = [2, 9, 3, 1, 5]
```

```
lista1.extend([200, 300])
```

```
print(lista1)
```

🌐 Adicionando valor em uma posição da lista, ele não substitui o valor da posição atual, esse valor se desloca para a direita:

```
lista1 = [2, 9, 3, 1, 5]
```

```
lista1.insert(0, 'Novo valor')
```

```
print(lista1)
```

🌐 Podemos adicionar duas listas ou mais:

```
lista1 = [2, 9, 3, 1, 5]
```

```
lista2 = ['F', 'e', 'r', 'n', 'a', 'n', 'd', 'a']
```

```
lista3 = lista1 + lista2
```

```
print(lista3)
```

🌐 Podemos inverter uma lista:

Forma1:

```
lista1 = [2, 9, 3, 1, 5]
lista1.reverse()
print(lista1)
```

Forma2 - Slice:

```
lista1 = [2, 9, 3, 1, 5]
print(lista1[::-1])
```

🌐 Podemos copiar uma lista:

```
lista1 = [2, 9, 3, 1, 5]
lista2 = lista1.copy()
print(lista2)
```

🌐 Podemos contar quantos elementos tem dentro da lista:

```
lista1 = [2, 9, 3, 1, 5]
print(len(lista1))
```

🌐 Podemos excluir o último elemento de uma lista - ele também retorna o último elemento:

```
lista1 = [2, 9, 3, 1, 5]
print(lista1.pop())
```

🌐 Podemos excluir um valor de uma lista pelo nome

```
lista = ['fernanda', 'maki']
removendo = lista.remove('fernanda')
print(lista)
```

🌐 Podemos remover um elemento pelo índice - se não existir o índice informado vai dar IndexError:

```
lista1 = [2, 9, 3, 1, 5]
print(lista1.pop(0))
print(lista1)
```

🌐 Podemos remover todos os elementos:

```
lista1 = [2, 9, 3, 1, 5]
lista1.clear()
print(lista1)
```

🌐 Podemos repetir elementos em uma lista:

```
lista1 = [2, 9, 3, 1, 5]
lista1 = lista1 * 3
print(lista1)
```

🌐 Podemos converter uma string para uma lista - o split separa as palavras pelo espaço:

```
nome = 'Olá, bom dia'
nome = nome.split()
print(nome)
```

🌐 Podemos converter uma string para uma lista - se não tiver espaços podemos dizer qual é o separador dentro do split:

```
nome = 'Olá,bom dia,tudo,bem?'  
nome = nome.split(',')  
print(nome)
```

🌐 Convertendo uma lista em uma string - vai atribuir espaço entre as palavras e vai juntar as palavras:

```
lista1 = ['Programação', 'em', 'Python']  
curso = ''.join(lista1)  
print(curso)
```

🌐 Se você quiser separar as palavras por outra forma em vez de espaço, exemplo \$:

```
lista1 = ['Programação', 'em', 'Python']  
curso = '$'.join(lista1)  
print(curso)
```

🌐 Iterando sobre listas:

🔗 Exemplo 1 - calculando a soma de todos os elementos numéricos e mostrando os elementos da lista:

```
soma = 0  
lista1 = [2, 9, 3, 1, 5]  
for elemento in lista1:  
    print(elemento)  
    soma += elemento  
print(soma)
```

🔗 Exemplo 2 - calculando a soma de todos os elementos strings e mostrando os elementos da lista:

```
soma = ""  
lista1 = ['Programando em Python']  
for elemento in lista1:  
    print(elemento)  
    soma += elemento  
print(soma)
```

🔗 Exemplo 3:

```
carrinho = []  
usuario = "
```

while True:

```
    usuario = input("Adicione um produto na lista ou digite 'sair' para sair: ")  
    if usuario != 'sair':  
        carrinho.append(usuario)  
    else:  
        for usuario in carrinho:  
            print(usuario)  
        break
```

🔗 Exemplo 4:

```
cores = ['amarelo', 'branco', 'azul', 'roxo']  
print(cores[-1]) -> roxo
```

```
print(cores[-2]) -> azul
print(cores[-3]) -> branco
print(cores[-4]) -> amarelo
```

🌐 Achar o índice de um número - ele pega o índice do primeiro valor encontrado:

```
cores = ['amarelo', 'branco', 'azul', 'roxo']
print(cores.index('amarelo')) -> vai retornar o índice 0
```

🌐 Achar o índice de um número a partir de um índice, caso não tenha o valor na lista vai retornar ValueError:

```
cores = ['amarelo', 'branco', 'azul', 'roxo', 'amarelo']
print(cores.index('amarelo', 1)) -> vai retornar o índice 4
```

🌐 Achar um índice entre um número e outro:

```
lista = [1, 2, 3, 4, 5, 3, 6, 3]
print(lista.index(3, 2, 4)) -> vai encontrar índice de 3 entre 2 e 4, vai retornar o valor 2
```

🌐 Revisão do slicing:

```
lista[início:fim:passo]
range[início:fim:passo]
```

```
lista = [1, 2, 3, 4]
print(lista[4:2]) -> coma do índice 1, até o 3 de 2 em 2
print(lista[1:]) -> vai começar do 2 até o 4
print(lista[:2]) -> vai pegar do 1 até o final de 2 em 2
```

🌐 Realizar a soma de uma lista:

```
lista = [1, 2, 3, 4]
print(sum(lista))
```

🌐 Pegar o maior valor de uma lista:

```
lista = [1, 2, 3, 4]
print(max(lista))
```

🌐 Pegar o menor valor de uma lista:

```
lista = [1, 2, 3, 4]
print(min(lista))
```

🌐 Transformar uma lista em uma tupla:

```
lista = [1, 2, 3, 4]
tupla = tuple(lista)
print(tupla)
```

🌐 Desempacotamento de listas - as variáveis vão receber os valores das listas:

```
lista = [1, 2, 3, 4]
num1, num2, num3, num4 = lista
print(num1)
print(num2)
print(num3)
```

🌐 Forma 1 - Deep copy - modificando uma lista ela não afeta a outra

```
lista = [1, 2, 3]
print(lista)
nova = lista.copy()
print(nova)
nova.append(4)
```

```
print(lista)
print(nova)
```

🌐 Forma 2 - Shallow Copy - modificando uma lista ela afeta a outra

```
lista = [1, 2, 3]
print(lista)
nova = lista
print(nova)
nova.append(4)
print(lista)
print(nova)
```

🌐 para substituir um valor pelo outro na lista com números

```
lista = [1, 2, 3, 4]
lista[0] = 100
print(lista)
```

🌐 para substituir um valor pelo outro com strings - o número 2 mostra quantas vezes você quer que substitua o 'f' pelo 'g', no caso 2 vezes

```
texto = 'ferafda'
texto = texto.replace('f', 'g', 2)
print(texto)
```

🌐 em vez de escrever if e else, é possível escrever try e except:
o try vai tentar executar algo, se falhar vai executar o código do except
caso você não queira escrever nada se falar, adicione o 'pass' depois do except

🌐 juntar duas listas
lista1.extend(lista2)

tuplas

1- Tudo que serve para dicionário serve para lista, com exceção das listas serem imutáveis, é necessário adicionar vírgula para números únicos, usa-se parênteses e não copiamos uma tupla usando o copy():

Esses dois exemplos são duas tuplas:

```
tupla = (1, 2, 3, 4, 5)
tupla = 1, 2, 3, 4, 5
```

2- Tupla com 1 valor - tuplas são definidas pela vírgula

```
tupla = (4) -> isso não é uma tupla
tupla = (4,) -> isso é uma tupla por conter a vírgula
```

3- É possível usar uma tupla com o range

```
tupla = tuple(range(11))
print(tupla)
print(type(tupla))
```

4- Desempacotamento da tupla


```
tupla = ['Fernanda Maki Hirose', '02/06/2003']  
nome, datanasc = tupla  
print(tupla)
```

5- Não pode adicionar valor nem tirar valor de uma tupla, já que elas são imutáveis

```
6- Soma  
tupla = [1, 2, 3]  
print(sum(tupla))
```

```
7- Valor máximo  
tupla = [1, 2, 3]  
print(max(tupla))
```

```
8- Valor mínimo  
tupla = [1, 2, 3]  
print(min(tupla))
```

```
9- Tamanho  
tupla = [1, 2, 3]  
print(len(tupla))
```

```
10- Concatenação de tuplas  
tupla1 = [1, 2, 3]  
tupla2 = [4, 5, 6]  
tupla1 = tupla1 + tupla2  
print(tupla1 + tupla2)
```

11- É possível iterar uma tupla da mesma forma que os dicionários

12- Devemos usar tuplas sempre que não precisamos modificar os dados, exemplo: meses

13- O acesso aos elementos de uma tupla são semelhantes ao acesso aos dicionários

```
meses = ('Janeiro', 'Fevereiro')  
print(meses[0])
```

14- Tuplas são mais rápidas que listas

15- Copiando uma tupla - na tupla não temos o problema de Shallow Copy:

```
tupla = (1, 2, 3)  
nova = tupla
```

```
outra = (4, 5, 6)  
nova = nova + outra
```

```
print(nova)  
print(tupla)
```

dicionários

Em algumas linguagens de programação, os dicionários python são conhecidos como mapas.

Os dicionários são representados por chaves {}

Não podemos ter chaves repetidas, caso haja chaves com nomes repetidos, os valores vão ser substituídos

A chave e o valor são separados por :, podem ser definidas por qualquer tipo de dados

Os dicionários são representados por chave/valor:

👩 Form 1:
países = {'br': 'Brasil', 'eua': 'Estados Unidos', 'py': 'Paraguai'}
print(países)
print(type(países))

👩 Form 2 - Menos comum:
países = dict(br='Brasil', eua='Estados Unidos', py='Paraguai')
print(países)
print(type(países))

👩 Acessando elementos: Forma recomendada
países = {'br': 'Brasil', 'eua': 'Estados Unidos', 'py': 'Paraguai'}
print(países.get('eua'))

👩 Vai pedir para encontrar o país 'ru', se não encontrar vai colocar o valor 'Não encontrado' no lugar
países = {'br': 'Brasil', 'eua': 'Estados Unidos', 'py': 'Paraguai'}
pais = países.get('ru', 'Não encontrado')
print(f'Encontrei o país {pais}')

👩 Verificando se algo está dentro de um elemento:
países = {'br': 'Brasil', 'eua': 'Estados Unidos', 'py': 'Paraguai'}
print('br' in países)

👩 Tuplas são interessantes de serem usadas como chaves de dicionários, porque elas são imutáveis:
localidades = {
 ('São Paulo'): 'SP',
 ('Rio de Janeiro'): 'RJ'
}
print(localidades)

👩 Adicionar elementos em um dicionário:
receita = {'janeiro': 100, 'fevereiro': 150}
receita['março'] = 200
print(receita)

👩 Atualizando elementos:
receita = {'janeiro': 100, 'fevereiro': 150}
receita.update({'janeiro': 111})
print(receita)

👩 Remover dados de um dicionário: (o pop é usado quando o valor é armazenado dentro de uma variável)
receita = {'janeiro': 100, 'fevereiro': 150}
remover = receita.pop('janeiro')
print(receita)

👩 Remover dados de um dicionário: (o del não armazena dentro de uma variável)
receita = {'janeiro': 100, 'fevereiro': 150}
del receita['janeiro']
print(receita)

👩 Remover todos os dados de um dicionário:
receita = {'janeiro': 100, 'fevereiro': 150}

```
receita.clear()
print(receita)
```

👤 ♀ Forma 1 - Copiando um dicionário para outro - Deep Copy:

```
receita = {'janeiro': 100, 'fevereiro': 150}
novo = receita.copy()
novo['abril'] = 3
print(receita)
print(novo)
```

👤 ♀ Forma 2 - Copiando um dicionário para outro - Shallow Copy:

```
novo = d
print(novo)
novo['d'] = 4
print(d)
print(novo)
```

👤 ♀ Método fromkeys - é necessário ter mais de 1 chave, quando tem apenas 1 chave, todas as letras vão receber o valor:

```
veja = {}.fromkeys('teste', 'valor')
print(veja)
```

👤 ♀ Veja o método fromkeys com mais de uma chave:

```
veja = {}.fromkeys(range(0, 11), 'valor range')
print(veja)
```

👤 ♀ Veja o método fromkeys com uma lista dentro:

```
usuário = {}.fromkeys(['chocolate', 'aveia', 'nescau'], 'comidas')
print(usuário)
```

👤 ♀ vai mostrar as chaves
.keys()

👤 ♀ vai mostrar os valores
.values()

👤 ♀ vai mostrar as chaves e os valores
.items()

🔑 Acessando as chaves:

```
receita = {'jan': 100, 'fev': 250, 'mar': 400}
print(receita.keys())
```

```
for chave in receita.keys():
    print(chave)
```

🔑 Acessando os valores:

```
receita = {'jan': 100, 'fev': 250, 'mar': 400}
print(receita.values())
```

```
for valor in receita.values():
    print(valor)
```

🔑 Desempacotamento de dicionários:

```
receita = {'jan': 100, 'fev': 250, 'mar': 400}
print(receita)
```

```
for chave, valor in receita.items():
    print(f'chave: {chave} e valor: {valor}')
```

```
print(receita.items())
```

🔗 Soma, valor máximo, valor mínimo, tamanho:

```
receita = {'jan': 100, 'fev': 250, 'mar': 400}
print(sum(receita.values()))
print(max(receita.values()))
print(min(receita.values()))
print(len(receita))
```

⊗ O tipo None é um tipo sem tipo
Ele sempre é dado com a primeira letra em maiúscula
Certo: None
Errado: none
O tipo None sempre é False

⊗ numeros = None
print(numeros)
print(type(numeros))

⊗ numeros = 1.44, 1.34, 5.67
print(numeros)
print(type(numeros))

⊗ Quando usar o tipo None?

✓ Conjuntos: teoria dos conjuntos da matemática
Os conjuntos são chamados de sets
Não possuem valores duplicados, ordenados
Os elementos não são acessados via índices, não são indexados
Eles são bons quando precisa-se armazenar elementos, sem se importar com a ordenação, chaves, valores e itens duplicados
Eles são referenciados por {}
É possível ter qualquer tipo de dado dentro de um set

✓ Um dicionário tem chave e valor, um conjunto tem apenas valor
Isso é um conjunto:
s = {5, 2, 6, 1, 1}
print(type(s))

✓ Podemos verificar se o elemento está no conjunto:
s = {5, 2, 6, 1, 1}
if 0 in s:
 print('Número existente')
else:
 print('Número não existente')

✓ Vamos ver como as listas, tuplas, dicionários e conjuntos se comportam:
Listas aceitam valores duplicados, então temos 10 elementos:

```
lista = [99, 2, 34, 23, 2, 12, 1, 44, 5, 34]
print(f'Lista: {lista} com {len(lista)} elementos')
```

✓ Tuplas aceitam valores duplicados, então temos 10 elementos:

```
tupla = 99, 2, 34, 23, 2, 12, 1, 44, 5, 34
print(f'Tupla: {tupla} com {len(tupla)} elementos')
```

✓ Dicionários não aceitam chaves duplicadas, então temos 8 elementos:

```
dicionarios = {}.fromkeys([99, 2, 34, 23, 2, 12, 1, 44, 5, 34], 'dict')
print(f'Dicionários: {dicionarios} com {len(dicionarios)} elementos')
```

✓ Conjuntos não aceitam valores duplicados, então temos 8 elementos:

```
conjunto = {99, 2, 34, 23, 2, 12, 1, 44, 5, 34}
print(f'Conjunto: {conjunto} com {len(conjunto)} elementos')
```

✓ Adicionando dados em um conjunto:

```
s = {3, 5, 1}
s.add(6)
print(s)
```

✓ Removendo dados em um conjunto (não retorna o valor):

```
s = {3, 5, 1}
s.discard(1)
```

✓ Removendo dados em um conjunto

```
s = {3, 5, 1}
s.remove(1)
```

✓ Copiando - Deep Copy:

```
s = {3, 5, 1}
novo = s.copy()
novo.add(4)
print(novo)
print(s)
```

✓ Copiando - Shallow Copy:

```
s = {3, 5, 1}
novo = s
novo.add(4)
print(s)
print(novo)
```

✓ Remover todos os itens de um conjunto:

```
s = {3, 5, 1}
s.clear()
print(s)
```

✓ Junta todos os nomes sem repeti-los

```
python = {'Julia', 'Pedro', 'Fernando'}
java = {'Julia', 'Charlie', 'Fernanda'}
unindo = python.union(java)
print(unindo)
```

✓ Mostrando os nomes que se repetem

```
python = {'Julia', 'Pedro', 'Fernando'}  
java = {'Julia', 'Charlie', 'Fernanda'}  
unindo = python.intersection( java)  
print(unindo)
```

✓ Mostrando os nomes que não se repetem

```
python = {'Julia', 'Pedro', 'Fernando'}  
java = {'Julia', 'Charlie', 'Fernanda'}  
unindo = python.difference(java)  
unindo2 = java.difference(python)  
print(unindo)  
print(unindo2)
```

✓ Soma, valor máximo, valor mínimo, tamanho

```
s = {1, 2, 3, 4, 5, 6}  
print(sum(s))  
print(max(s))  
print(min(s))  
(print(len(s)))
```

📁 Módulos Collections - Counter

<https://docs.python.org/3/library/collections.html>

O Counter conta quantas vezes aparece certo valor:

📁 Exemplo 1 - Contando números - Vai contar quantas vezes os números apareceram:

```
from collections import Counter  
lista = [1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 5]  
res = Counter(lista)  
print(type(res))  
print(res)
```

📁 Exemplo 2 - Contando strings - Vai contar quantas vezes as letras da palavra apareceram:


```
from collections import Counter  
tupla = ('Fernanda')  
nome = Counter(tupla)  
print(type(nome))  
print(nome)
```

📁 Exemplo 3 - Vai contar quantas vezes as palavras da lista aparecem:

```
from collections import Counter
```

```
texto = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor  
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation  
ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit  
in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat  
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'
```

```
lista = texto.split()  
contador = Counter(lista)  
print(contador)
```

 Exemplo 4 - Encontrando as 5 palavras com mais ocorrências:

```
from collections import Counter
```

```
texto = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor  
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation  
ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit  
in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat  
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'
```

```
contador = Counter(texto)  
print(contador.most_common(5))
```