



# Python 1

## Fundamentos

# HISTÓRIA DO PYTHON



O Python foi criado em 1989 por Guido van Rossum com o objetivo de ser simples e legível.

Lançado em 1991, passou por grandes mudanças: em 2000, a versão 2.0 trouxe melhorias, e em 2008, o Python 3.0 foi lançado, mas incompatível com a versão anterior.

A partir de 2010, ganhou popularidade em áreas como ciência de dados e inteligência artificial.

Em 2018, Guido se afastou e a comunidade passou a liderar o projeto. Hoje, o Python é uma das linguagens mais usadas no mundo.

# COISAS BÁSICAS SOBRE O PYTHON

## 1 - O Python é uma linguagem interpretada:

Isso significa que o código fonte não é previamente convertido em código de máquina. Em vez disso, ele é executado linha por linha por um interpretador, em tempo de execução.



### **Exemplo: Tradutor simultâneo (intérprete) vs tradutor de livros (compilador)**

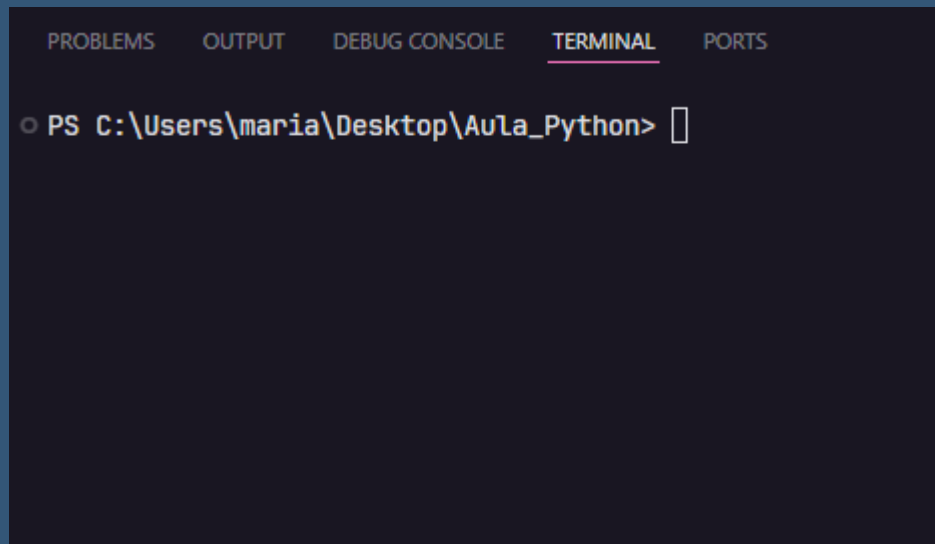
- Imagine que você está em uma conferência onde alguém fala em inglês, mas você só entende português.
  - Um **intérprete** está ao seu lado e traduz **falando ao mesmo tempo que ouve**. Ele **não traduz tudo antes**, ele **interpreta linha por linha, em tempo real** – assim como o Python executa o código.
  - ✓ Isso é como uma **linguagem interpretada** funciona.
- Agora, pense em um tradutor que recebe um **livro inteiro em inglês**, traduz tudo de uma vez e depois entrega o livro completo em português.
  - ✓ Isso seria como uma **linguagem compilada**, que **compila todo o código antes** de você poder usá-lo.

# COISAS BÁSICAS SOBRE O PYTHON

## 2 - O Python é principalmente uma linguagem Back-End:

É mais usada na parte que roda por trás das aplicações, o “motor” que processa dados, acessa bancos de dados, faz cálculos e lógica de negócio.

Como Python não é usado para criar interfaces visuais como botões ou páginas web diretamente (isso é trabalho do front-end), ele é executado normalmente no terminal (ou linha de comando). É ali que vamos digitar e rodar nossos códigos.

A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. Below the tabs, the terminal shows a command prompt 'PS C:\Users\maria\Desktop\Aula\_Python>' followed by a cursor.

# COISAS BÁSICAS SOBRE O PYTHON

## 3 - PEP8 (Python Enhancement Proposal 8) :

É um documento oficial que define as boas práticas de formatação de código em Python. Ele foi criado para tornar o código mais legível e uniforme entre diferentes projetos e programadores.

Versão original em inglês da PEP 8

<https://www.python.org/dev/peps/pep-0008/>

Versão em português da PEP 8

<https://wiki.python.org.br/GuiaDeEstilo>

# PREPARANDO O AMBIENTE

## 1 - Instalar o interpretador Python

Faça o download e a instalação do interpretador Python, que é o componente responsável por executar os códigos escritos em linguagem Python, convertendo-os em instruções que o computador possa entender e processar.



<https://www.python.org/downloads/>

```
C:\Users\maria>python
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:0
3:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for m
ore information.
>>>
```

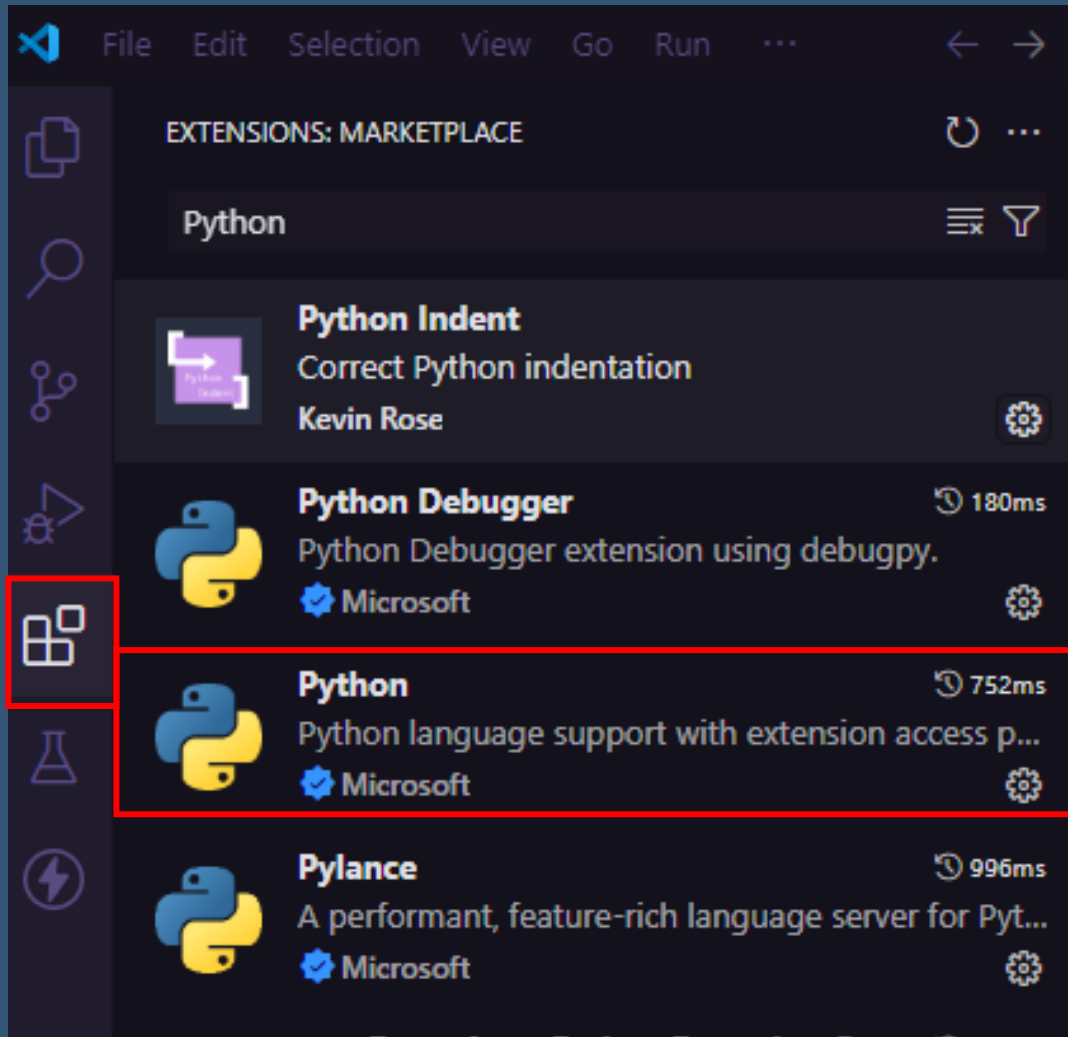
# PREPARANDO O AMBIENTE

## 2 - Instalar o ambiente de desenvolvimento (IDE)

Um ambiente de desenvolvimento (IDE) é o conjunto de ferramentas e configurações que você usa para escrever, testar e executar seus códigos de forma prática e produtiva.

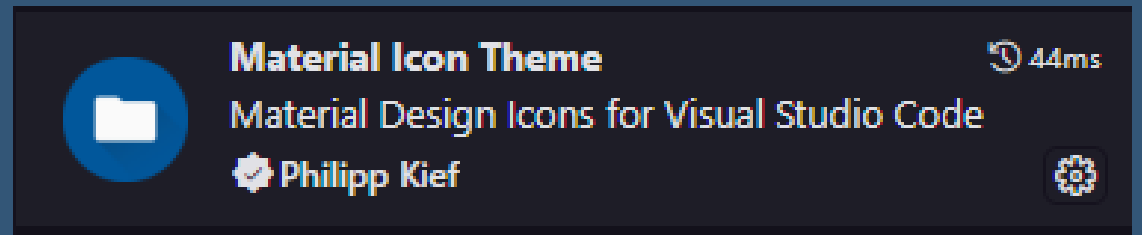
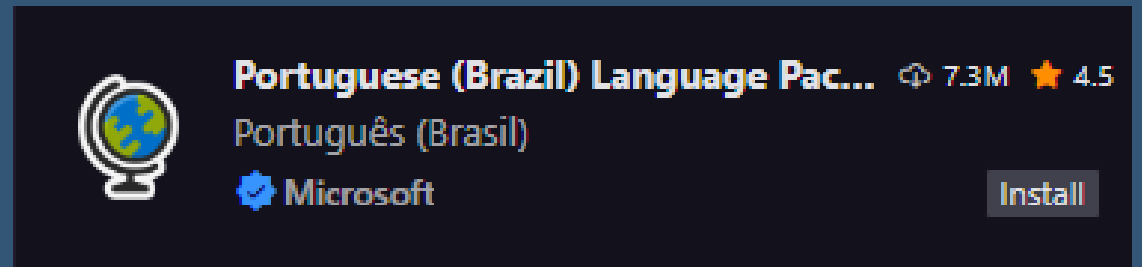


# PREPARANDO O AMBIENTE



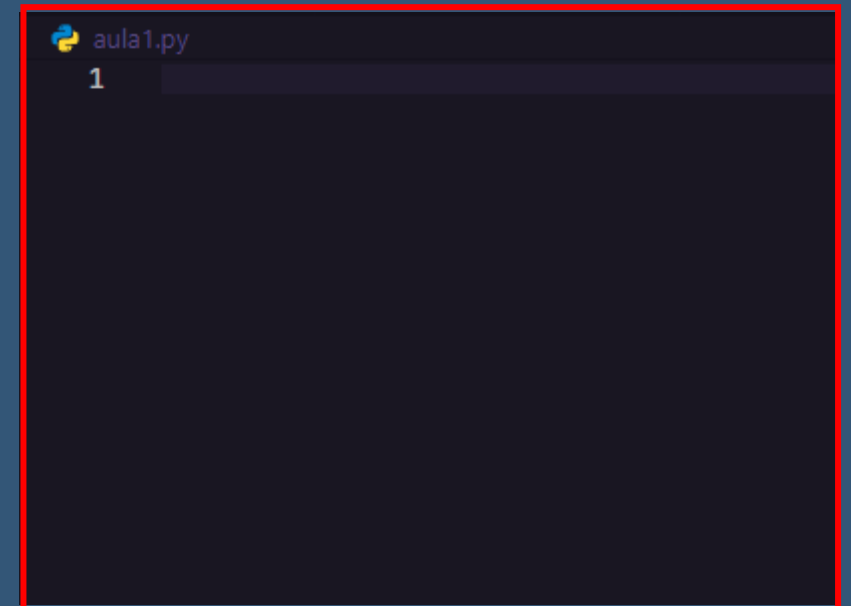
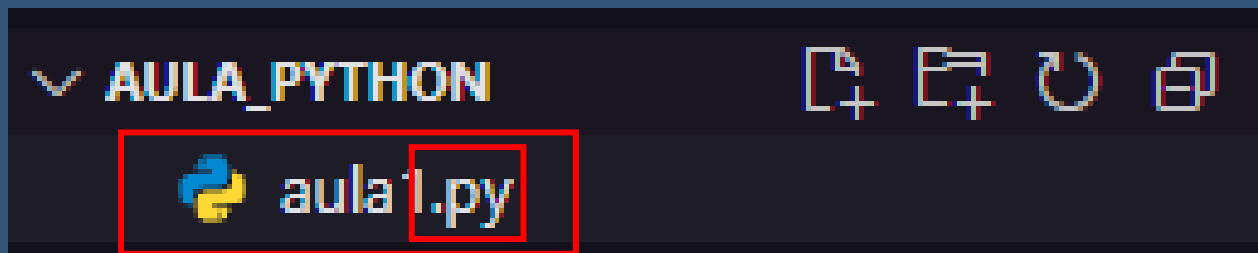
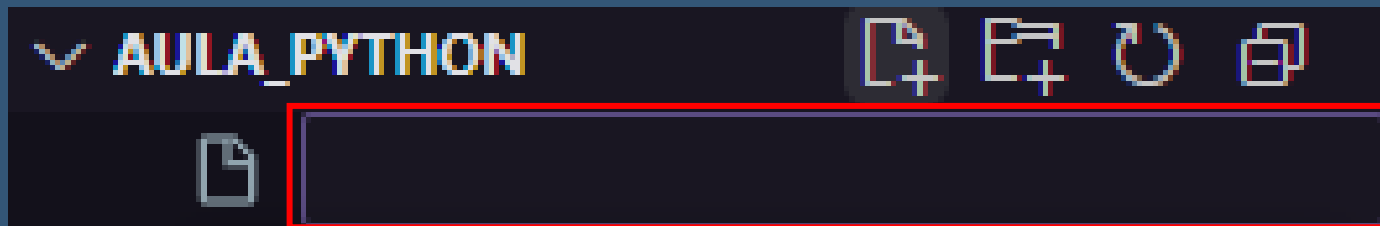
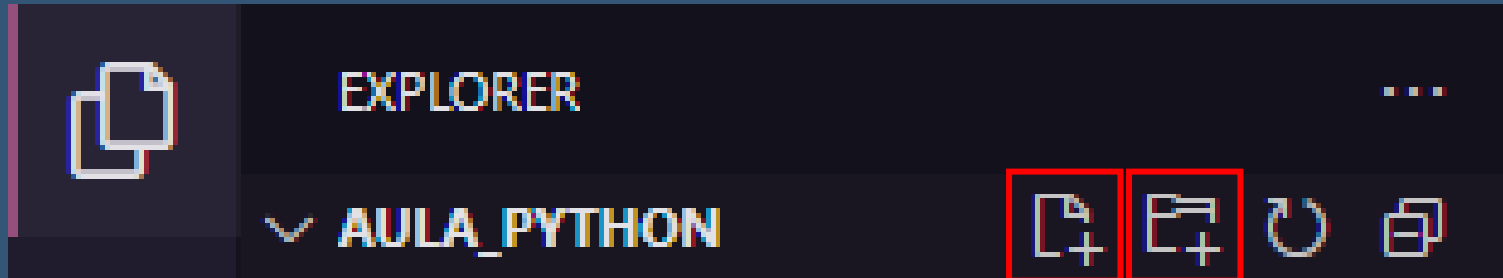
## 2 - Instalar extensões

Extensões são complementos (plugins) que você instala no VS Code para adicionar novas funcionalidades ou melhorar a experiência de programação.





# CRIANDO UM ARQUIVO .pyc



# OLÁ MUNDO!



```
aula1.py x
aula1.py
1 print('Olá mundo')
```

PRINT: Ele imprime no terminal (ou console) o que você quiser mostrar: textos, números, resultados de contas, valores de variáveis, etc.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + v [icon] [icon] ... ^ x

PS E:\PythonFundamentos1\AulaPythonSábado> & C:/Users/maria/AppData/Local/Programs/Python/Python312/python.exe e:/PythonFundame
ntos1/AulaPythonSábado/aula1.py
● Olá mundo
○ PS E:\PythonFundamentos1\AulaPythonSábado>
```

# TIPOS DE DADOS E VARIÁVEIS

## INT (INTEIRO):

- Números sem casas decimais.
- Exemplo: 10, -3, 0

## FLOAT (NUMEROS DECIMAIS / PONTO FLUTUANTE):

- Números com casas decimais.
- Exemplo: 3.14, -0.5, 10.0

## STR (STRINGS / TEXTO):

- Cadeia de caracteres (entre aspas).
- Exemplo: "Olá", 'Python'

## BOOL (BOOLEAN):

- Verdadeiro ou falso.
- Exemplo: True, False

# TIPOS DE DADOS E VARIÁVEIS

Variáveis são nomes simbólicos que você usa para armazenar dados na memória do computador. Elas permitem que você guarde valores (como números, textos, listas) e trabalhe com eles ao longo do seu programa.

```
aula1.py X
aula1.py > ...
1 nome = 'Mariana'
2 sobrenome = 'Badu'
3 idade = 21
4 altura = 1.63
```

```
aula1.py X
aula1.py > ...
1 nome = 'Mariana'
2 sobrenome = 'Badu'
3 idade = 21
4 altura = 1.63
5
6 print(nome)
7
8
9 |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
• PS E:\PythonFundamentos1\AulaPythonSábado> & C:/Us
:/PythonFundamentos1/AulaPythonSábado/aula1.py
Mariana
○ PS E:\PythonFundamentos1\AulaPythonSábado>
```

# OPERADORA ARITMÉTICOS

+	ADIÇÃO
-	SUBTRAÇÃO
*	MULTIPLICAÇÃO
/	DIVISÃO
//	DIVISÃO INTEIRA (SEM PONTO FLUTUANTE)
%	MODULO (RESTO DA DIVISÃO)
**	EXPONENCIAÇÃO

# EXEMPLOS PRÁTICOS

```
# Exemplo 1: Declarando variáveis simples
nome = "João"
idade = 25
altura = 1.75

print("Nome:", nome)
print("Idade:", idade)
print("Altura:", altura)
```

```
# Exemplo 2: Entrada de dados do usuário
nome = input("Digite seu nome: ")
idade = input("Digite sua idade: ")

print("Olá,", nome + "! Você tem", idade, "anos.")
# OU
print(f'Olá {nome}! Você tem {idade} anos.')
```



**Interpolação** é quando você insere variáveis dentro de uma string, ou seja, mistura texto fixo com valores dinâmicos (variáveis, expressões, etc).

```
# Exemplo 3: Entrada de número e cálculo
num1 = float(input("Digite o primeiro número: "))
num2 = float(input("Digite o segundo número: "))

soma = num1 + num2

print("A soma é:", soma)

# OBS: input() sempre retorna uma string (cadeia de caracteres),
# então se quiser converter para número, use int() ou float().
```

```
# Exemplo 4: Usando variáveis para fazer uma média
nota1 = float(input("Digite a primeira nota: "))
nota2 = float(input("Digite a segunda nota: "))

media = (nota1 + nota2) / 2

print("A média é:", media)
```

# ATIVIDADE

1 - Crie um programa que solicite ao usuário a largura e a altura de um retângulo (em metros), calcule a área e exiba o resultado na tela.

Formula:  $\text{largura} * \text{altura}$

2 - Escreva um programa que leia três números inteiros digitados pelo usuário, some esses valores e exiba o total da soma.

3 - Desenvolva um programa que receba o peso (em kg) e a altura (em metros) de uma pessoa, calcule o IMC (Índice de Massa Corporal) e mostre o resultado.

Formula:  $\text{IMC} = \text{peso} / (\text{altura}^2)$

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 1:
largura = float(input("Digite a largura do retângulo (em metros): "))
altura = float(input("Digite a altura do retângulo (em metros): "))

area = largura * altura

print(f"A área do retângulo é {area} metros quadrados.")
```

```
# ATIVIDADE 2:
num1 = int(input("Digite o primeiro número inteiro: "))
num2 = int(input("Digite o segundo número inteiro: "))
num3 = int(input("Digite o terceiro número inteiro: "))

soma = num1 + num2 + num3

print(f"A soma dos três números é: {soma}")
```



# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 3:
peso = float(input("Digite seu peso (em kg): "))
altura = float(input("Digite sua altura (em metros): "))

imc = peso / (altura ** 2)

print(f"Seu IMC é: {imc:.2f}")

# OBS: 0 :.2f no print é usado para mostrar o número com 2 casas decimais.
```

# ESTRUTURAS CONDICIONAIS

Condicionais permitem que o programa tome decisões diferentes dependendo de uma condição ser verdadeira (True) ou falsa (False).

Podemos pensar como uma pergunta:

Se isso for verdade, faça isso.  
Senão, talvez faça outra coisa.

ESTRUTURA BÁSICA DA CONDICIONAL:

```
if condição1:
    # executa se condição1 for True
elif condição2:
    # executa se condição2 for True
else:
    # executa se nenhuma das anteriores for True
    # o 'ELSE' não tem condição a ser analisada
```

**Importante:** A indentação é obrigatória em Python. Normalmente usamos 4 espaços.

# ESTRUTURAS CONDICIONAIS

COMO ESSAS CONDIÇÕES SÃO AVALIDADA:

É necessária utilizar operadores de comparação e lógicos:

OPERADORES DE COMPARAÇÃO:

==	IGUAL A
!=	DIFERENTE DE
>	MAIOR QUE
<	MENOR QUE
>=	MAIOR OU IGUAL A
<=	MENOR OU IGUAL A

OPERADORES LÓGICOS:

and	RETORNA TRUE SE AMBAS AS CONDIÇÕES FOREM TRUE
or	RETORNA TRUE SE PELO MENOS UMA CONDIÇÃO FOR TRUE
not	INVERTE O VALOR LÓGICO (SE TRUE VIRA FALSE E VICE-VERSA)

# ESTRUTURAS CONDICIONAIS

EXEMPLOS COM CADA OPERADOR DE COMPARAÇÃO:

1- Igual a (==):

```
# IGUAL A (==):  
cor = "vermelho"  
  
if cor == "vermelho":  
    print("A cor é vermelho.") # A cor é vermelha
```

2 - Diferente de (!=):

```
# DIFERENTE DE (!=):  
usuario = "admin"  
  
if usuario != "admin":  
    print("Acesso negado.")  
else:  
    print("Bem-vindo, admin.")
```

3 - Menor que (<):

```
# MENOR QUE (<):  
idade = 15  
  
if idade < 18:  
    print("Menor de idade.")
```

4 - Menor ou igual a (<=):

```
# MENOR OU IGUAL A (<=):  
nota = 5  
  
if nota <= 5:  
    print("Precisa melhorar.")
```

# ESTRUTURAS CONDICIONAIS

EXEMPLOS COM CADA OPERADOR DE COMPARAÇÃO:

5 – Maior ou igual a ( $\geq$ ):

```
# MAIOR OU IGUAL A ( $\geq$ ):  
salario = 3000  
  
if salario  $\geq$  2500:  
    print("Salário acima do mínimo.")
```

6 – Maior que ( $>$ ):

```
# MAIOR QUE ( $>$ ):  
temperatura = 38  
  
if temperatura > 37:  
    print("Febre detectada.")
```

EXEMPLOS COM CADA OPERADOR LÓGICO:

AND (e):

```
# AND (e):  
idade = 20  
tem_carteira = True # tipo booleano (True ou False)  
  
if idade  $\geq$  18 and tem_carteira:  
    print("Pode dirigir.")
```

OR (ou):

```
# OR (ou):  
feriado = False  
final_de_semana = True  
  
if feriado or final_de_semana:  
    print("Pode descansar!")
```

# ESTRUTURAS CONDICIONAIS

EXEMPLOS COM CADA OPERADOR LÓGICO:

NOT (não / negação):

```
# NOT (não / negação)
chovendo = False

if not chovendo:
    print("Pode sair sem guarda-chuva.")
```

EXEMPLOS COMBINANDO TUDO:  
OPERADORES DE COMPARAÇÃO E LÓGICOS

```
# COMBINANDO OPERADORES LÓGICOS E DE COMPARAÇÃO
idade = 17
autorizado = True

if idade >= 18 or (idade >= 16 and autorizado):
    print("Pode votar.")
else:
    print("Não pode votar.")
```



Em Python, os **operadores lógicos** seguem uma ordem de precedência, ou seja, quem é avaliado primeiro quando aparecem juntos em uma expressão.

Operador	Nome	Avaliação
<code>not</code>	negação	primeiro
<code>and</code>	e lógico	segundo
<code>or</code>	ou lógico	por último

# ESTRUTURAS CONDICIONAIS: EXEMPLOS PRÁTICOS

```
# EXEMPLO 1: CALCULO DE FRETE GRÁTIS
compra = float(input("Qual o valor da sua compra? "))

if compra ≥ 100:
    print("Você ganhou frete grátis!")
else:
    print("Frete: R$20,00")
```

```
# EXEMPLO 2: CLASSIFICAR FAIXA ETÁRIA
idade = int(input("Qual sua idade? "))

if idade ≤ 12:
    print("Criança")
elif idade < 18:
    print("Adolescente")
elif idade ≤ 60:
    print("Adulto")
else:
    print("Idoso")
```

```
# EXEMPLO 3: VERIFICAR HORARIO DE FUNCIONAMENTO
hora = int(input("Digite a hora atual (0 a 23): "))

if hora ≥ 9 and hora ≤ 18:
    print("Loja aberta.")
else:
    print("Loja fechada.")
```

```
# EXEMPLO 4: TEMPERATURA E CLIMA
temperatura = float(input("Qual a temperatura agora? "))

if temperatura ≥ 30:
    print("Está quente! Use roupas leves.")
elif temperatura ≥ 20:
    print("Clima agradável.")
else:
    print("Está frio! Vista um casaco.")
```

```
# EXEMPLO 5: VERIFICANDO SE UM NUMERO É PAR / DIVISIVEL POR 2
numero = int(input("Digite um número: "))

if numero % 2 == 0:
    print("O número é par.")
else:
    print("O número é ímpar.")
```

# ATIVIDADE

1. Peça ao usuário para digitar um número inteiro. Diga se o número é par ou ímpar.
2. Peça o nome de usuário e a senha. Só permita o acesso se o usuário for "admin" e a senha for "1234".
3. Peça três números ao usuário e diga qual é o maior deles.
4. Peça ao usuário uma nota de 0 a 10.

Mostre se ele está:

- Aprovado (nota  $\geq 7$ )
- Recuperação (nota entre 5 e 6.9)
- Reprovado (nota  $< 5$ )

5. Peça ao usuário um número inteiro. Informe se ele é múltiplo de 3, 5, ambos, ou nenhum.



# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 1:
numero = int(input("Digite um número inteiro: "))

if numero % 2 == 0:
    print("O número é par.")
else:
    print("O número é ímpar.")
```

```
# ATIVIDADE 2:
usuario = input("Digite o nome de usuário: ")
senha = input("Digite a senha: ")

if usuario == "admin" and senha == "1234":
    print("Acesso permitido.")
else:
    print("Usuário ou senha incorretos.")
```

```
# ATIVIDADE 3:
a = float(input("Digite o primeiro número: "))
b = float(input("Digite o segundo número: "))
c = float(input("Digite o terceiro número: "))

✓ if a >= b and a >= c:
    print(f"O maior número é {a}")
✓ elif b >= a and b >= c:
    print(f"O maior número é {b}")
✓ else:
    print(f"O maior número é {c}")
```

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 4:
nota = float(input("Digite sua nota (0 a 10): "))

if nota >= 7:
    print("Aprovado!")
elif nota >= 5:
    print("Recuperação.")
else:
    print("Reprovado.")

# OU

nota = float(input("Digite sua nota (0 a 10): "))

if nota < 0 or nota > 10:
    print("Erro: o valor digitado está fora do intervalo permitido (0 a 10).")
elif nota >= 7:
    print("Aprovado!")
elif nota >= 5:
    print("Recuperação.")
else:
    print("Reprovado.")
```

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 5:
num = int(input("Digite um número: "))

if num % 3 == 0 and num % 5 == 0:
    print("É múltiplo de 3 e 5.")
elif num % 3 == 0:
    print("É múltiplo de 3.")
elif num % 5 == 0:
    print("É múltiplo de 5.")
else:
    print("Não é múltiplo de 3 nem de 5.")
```

# ESTRUTURAS DE REPETIÇÃO

As estruturas de repetição em Python são usadas para executar um bloco de código várias vezes. Python oferece duas principais estruturas para isso:

1 - WHILE (Enquanto): Executará um bloco de código **ENQUANTO** a condição proposta for verdadeira.

Utilizamos o WHILE quando não sabemos quantas vezes um bloco de código irá se repetir.

SINTAXE:

```
while condição:  
    # BLOCO DE CÓDIGO QUE SERÁ EXECUTADO ENQUANTO  
    # A CONDIÇÃO FOR VERDADEIRA
```

EXEMPLO:

```
1  contador = 0  
2  
3  # ENQUANTO o contador for menor ou igual 5  
4  while contador ≤ 5:  
5      # Exibira no console: Contador: 0  
6      print(f"Contador: {contador}")  
7      contador += 1 # contador = contador + 1  
8
```

```
● PS E:\PythonFundamentos1\Aula_Python> python estrutura_repeticao.py  
Contador: 0  
Contador: 1  
Contador: 2  
Contador: 3  
Contador: 4  
Contador: 5
```

# ESTRUTURAS DE REPETIÇÃO



## CUIDADO COM WHILE INFINITO

Se a condição **nunca parar de ser verdadeira**, o programa **nunca para!**

### EXEMPLO 1:

```
1 while True:
2     print("Isso nunca vai parar!")
3
```

### EXEMPLO 2:

```
1 contador = 1
2
3 while contador > 0:
4     print("Contando...", contador)
5     # contador nunca diminui!
```

A condição `contador > 0` sempre será verdadeira, porque nunca mudamos o valor do contador para parar o loop.

## COMO PODEMOS EVITAR UM WHILE INFINITO

1 - Sempre defina bem o que você quer fazer naquele bloco de código:

```
contador = 5

while contador > 0:
    print("Contando...", contador)
    contador = contador - 1 # agora vai parar!
```

2 - Pode ser utilizado uma condição de parada (exemplo: input do usuário):

```
resposta = ""

while resposta != "sair":
    resposta = input("Digite 'sair' para encerrar: ")
```

3 - Podemos usar o **break** para forçar uma parada:

```
while True:
    comando = input("Digite um comando: ")
    if comando == "parar":
        break

    print("Você digitou:", comando)
```

# ESTRUTURAS DE REPETIÇÃO

## EXEMPLO PRÁTICOS:

EXEMPLO 1: Perguntar a senha até aceitar

```
senha_correta = "python123"
tentativa = ""

while tentativa != senha_correta:
    tentativa = input("Digite a senha: ")

print("Acesso liberado!")
```

EXEMPLO 2: Somar vários números

```
soma = 0
numero = int(input("Digite um número (0 para sair): "))

while numero != 0:
    soma += numero
    numero = int(input("Digite outro número (0 para sair): "))

print("A soma total é:", soma)
```

EXEMPLO 3: Contar quantas palavras o usuário digitou

```
frase = input("Digite uma frase (ou 'sair' para encerrar): ")
quantidade = 0

while frase != "sair":
    quantidade += len(frase.split())
    frase = input("Digite outra frase (ou 'sair' para encerrar): ")

print("Você digitou", quantidade, "palavras.")
```

`split()`: quebra uma frase em palavras onde tem espaço.

`len()`: é uma função que conta quantos itens tem dentro de uma lista, texto ou outro grupo de coisas.



# ESTRUTURAS DE REPETIÇÃO

2 – FOR (para): O **for** é uma estrutura de repetição usada para repetir uma ação para cada item de uma lista ou sequência (strings, por exemplo).

Utilizamos o **for** quando sabemos quantas vezes determinado bloco de código deve repetir, ou quando quisermos percorrer/iterar todos os itens de algo (como letras, números ou nomes).

SINTAX **for** variável **in** sequência:  
    bloco de código

- **variável**: é o nome que damos para cada item durante a repetição.
- **sequência**: é o que vamos percorrer (pode ser uma lista, texto, intervalo de números com `range()`...).

EXEMPLO:

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

Para cada número na sequência [1, 2, 3, 4, 5], faça o que está dentro do laço.

```
PythonFundamentos1/AulaPythonSábado/teste.py  
1  
2  
3  
4  
5
```

RESUMINDO A LÓGICA DO FOR:

1. `range(1, 6)` cria os números 1 a 5.
2. O **for** pega cada número, um por um.
3. Para cada número, executa o código dentro (no caso, o `print()`).
4. Quando os números acabam, o laço termina automaticamente.

# ESTRUTURAS DE REPETIÇÃO

## RANGE():

É uma função embutida do Python que retorna um objeto iterável, que gera uma sequência de números inteiros, de forma eficiente (sem ocupar muita memória). É muito usado em loops for.

Sintaxe geral:

`range(inicio, fim, passo)`

**início (opcional):** número inicial da sequência.  
Padrão: 0

**fim (obrigatório):** até onde vai a sequência (não é incluído)

**passo (opcional):** intervalo entre os números.  
Padrão: 1

EXEMPLOS:

1 - FIM:

```
for i in range(5):  
    print(i)
```

2 - INICIO e FIM:

```
for i in range(2, 6):  
    print(i)
```

3 - INICIO, FIM e PASSO:

```
for i in range(1, 10, 2):  
    print(i)
```



# ESTRUTURAS DE REPETIÇÃO

4 - PASSO negativo (Contagem Regressiva)

```
for i in range(10, 0, -1):  
    print(i)
```

Observações sobre o RANGE:

- `range()` não inclui o valor final
- Funciona apenas com números inteiros

# ESTRUTURAS DE REPETIÇÃO

## EXEMPLOS PRÁTICOS:

EXEMPLO 1: Contador de 1 a 10

```
for numero in range(1, 11):  
    print(numero)
```

```
PS E:\PythonFundamentos1\Aul  
os1/AulaPythonSábado/Aula 3
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

EXEMPLO 2: Mostrar só números pares de 2 a 10

```
for numero in range(2, 11, 2):  
    print(numero)
```

```
PS E:\PythonFundamentos1\Aul  
os1/AulaPythonSábado/Aula 3  
2  
4  
6  
8  
10
```

EXEMPLO 3: Contagem regressiva

```
for numero in range(5, 0, -1):  
    print(numero)  
  
print("Vai!")
```

# OPERADORES DE ATRIBUIÇÃO

1. = ATRIBUIÇÃO SIMPLES (utilizamos ao declarar variáveis)
2. += ATRIBUIÇÃO DE SOMA (EX:  $A += 5 \rightarrow A = A + 5$ )
3. -= ATRIBUIÇÃO DE SUBTRAÇÃO
4. \*= ATRIBUIÇÃO DE MULTIPLICAÇÃO
5. /= ATRIBUIÇÃO DE DIVISÃO

# OPERADORES DE ATRIBUIÇÃO

## EXEMPLOS:

```
# Atribuição simples  
x = 10  
print(x) # 10
```

```
# Divisão e atribuição  
x /= 4 # x = x / 4  
print(x) # 6.0
```

```
# Soma e atribuição  
x += 5 # x = x + 5  
print(x) # 15
```

```
# Multiplicação e atribuição  
x *= 2 # x = x * 2  
print(x) # 24
```

```
# Subtração e atribuição  
x -= 3 # x = x - 3  
print(x) # 12
```

# ATIVIDADE

## WHILE

1. Crie um programa que escolha um número secreto de 1 a 10 (você define esse número no código). O usuário deverá tentar adivinhar o número. O programa deve continuar pedindo tentativas até o número correto ser digitado. Ao final, mostre quantas tentativas foram feitas até o acerto.
2. O usuário digita vários números (um por vez). O programa só para quando ele digitar 0. Ao final, mostre quantos números pares foram digitados.

## FOR

1. Peça um número ao usuário e imprima a tabuada dele de 10 até 1 (ordem decrescente).
2. Peça ao usuário para digitar uma frase e mostre uma palavra por linha.
3. Crie um programa que calcule e mostre a soma de todos os números ímpares entre 1 e 100.

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 1:
numero_secreto = 4
tentativas = 0
palpite = 0

while palpite != numero_secreto:
    palpite = int(input("Tente adivinhar o número de 1 a 10: "))
    tentativas += 1

print("Você acertou em", tentativas, "tentativa(s)!")
```

```
# ATIVIDADE 2:
contador_pares = 0
numero = int(input("Digite um número (0 para parar): "))

while numero != 0:
    if numero % 2 == 0:
        contador_pares += 1
    numero = int(input("Digite outro número (0 para parar): "))

print("Você digitou", contador_pares, "números pares.")
```

```
# ATIVIDADE 3:
numero = int(input("Digite um número: "))

for i in range(10, 0, -1):
    print(f"{numero} x {i} = {numero * i}")
```

```
# ATIVIDADE 4:
frase = input("Digite uma frase: ")
palavras = frase.split()

for palavra in palavras:
    print(palavra)
```

```
# ATIVIDADE 5:
soma = 0

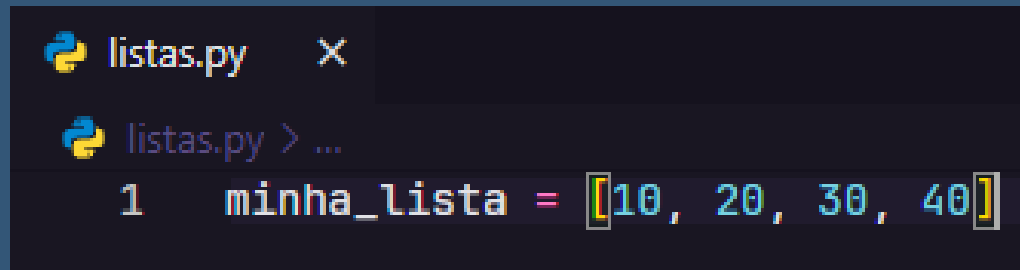
for i in range(1, 101):
    if i % 2 != 0:
        soma += i

print("A soma dos ímpares de 1 a 100 é:", soma)
```

# LISTAS

Uma lista em Python é uma coleção ordenada e mutável de elementos. Você pode armazenar vários valores em uma única variável e acessar, modificar, adicionar ou remover elementos com facilidade.

SINTAXE:

A screenshot of a Python IDE window titled 'listas.py'. The editor shows a single line of code: '1 minha\_lista = [10, 20, 30, 40]'. The list elements are highlighted in yellow, and the closing bracket is highlighted in blue. The background is dark with light-colored text.

```
listas.py X  
listas.py > ...  
1  minha_lista = [10, 20, 30, 40]
```

As listas são definidas por colchetes `[]`, e os elementos são separados por vírgulas.

# LISTAS

## PRINCIPAIS OPERAÇÕES USANDO LISTAS:

1 - Acessar elementos: Especificamos a posição/índice do elemento.

```
listas.py X
listas.py > ...
1  nomes = ["Ana", "Bruno", "Carlos"]
2  print(nomes[0]) # Ana
3  print(nomes[1]) # Bruno
4
```

2 - Alterar elementos:

```
listas.py X
listas.py > ...
1  nomes = ["Ana", "Bruno", "Carlos"]
2  nomes[1] = "Beatriz"
3  print(nomes) # ['Ana', 'Beatriz', 'Carlos']
4
```



# LISTAS

3 - Adicionando um elemento ao final da lista: Utilizamos a função `append()`.

```
listas.py  X
listas.py > ...
1  nomes = ["Ana", "Bruno", "Carlos"]
2  nomes.append("Daniela") # Adiciona no final
3  print(nomes)
```

4 - Remover elementos:

```
listas.py  X
listas.py > ...
1  nomes = ["Ana", "Bruno", "Carlos"]
2  nomes.remove("Carlos") # Remove o valor
3  print(nomes)
4
5  nomes.pop(0) # Remove pelo índice (nesse caso, "Ana")
6  print(nomes)
```

# LISTAS

5 - Verificar tamanho da lista / quantidade de elementos da lista:

```
listas.py X
listas.py > ...
1 nomes = ["Ana", "Bruno", "Carlos"]
2 print(len(nomes)) # Número de elementos
```

6 - Percorrendo uma lista utilizando o loop FOR:

```
listas.py X
listas.py > ...
1 nomes = ["Ana", "Bruno", "Carlos"]
2
3 for nome in nomes:
4     print(f"Olá, {nome}!")
5
```

7 - Verificar se há um item na lista com a estrutura condicional IF:

```
nomes = ['Ana', 'Bruno', 'Carlos']

if 'Ana' in nomes:
    print('Ana está na lista!')
else:
    print('Não há esse elemento na lista :(')
```

# LISTAS

## EXEMPLOS PRÁTICOS:

### EXEMPLO 1: Simulando um to do list

```
tarefas = ["Estudar Python", "Lavar a louça", "Fazer exercícios"]

# Mostrar todas as tarefas
for tarefa in tarefas:
    print("Tarefa:", tarefa)

# Adicionar uma nova tarefa
tarefas.append("Revisar listas em Python")

# Marcar a primeira tarefa como feita (remoção)
tarefas.remove("Estudar Python")

print("Tarefas restantes:", tarefas)
```

### EXEMPLO 2: Cálculo de média

```
notas = [8.5, 7.0, 9.2, 6.8, 5.0]

media = sum(notas) / len(notas)

print("Notas:", notas)
print("Média final:", round(media, 2))
```

**sum():** função de soma, vai somar todos os valores que estão armazenados na variável notas.

**len():** conta quantos elementos existem dentro da lista: 5 elementos.

**round():** função que vai arredondar o valor (media) e deixar apenas com duas casas decimais.



# LISTAS

## LISTAS ANINHADAS:

Como foi dito anteriormente, podemos adicionar qualquer tipo de dado dentro de uma lista, incluindo uma outra lista. Chamamos isso de lista aninhada.

## EXEMPLO:

```
lista_aninhada = [[1, 2], [3, 4], [5, 6]]
print(lista_aninhada)

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

```
[[1, 2], [3, 4], [5, 6]]
[1, 2]
[3, 4]
[5, 6]
```

## COMO ACESSAMOS ELEMENTOS DE LISTAS ANINHADAS:

```
print(lista_aninhada[0])      [1, 2]
print(lista_aninhada[0][1])   2
print(lista_aninhada[2][0])   5
```

```
# Mostra: [1, 2]
# Mostra: 2 (segunda posição da primeira lista)
# Mostra: 5 (primeira posição da terceira lista)
```

# LISTAS

## EXEMPLOS PRÁTICOS:

EXEMPLO 1: Listando alunos e apresentando a média final de acordo com as notas dentro da lista

```
alunos = [  
    ["Ana", [7.5, 8.0, 9.0]],  
    ["Lucas", [6.0, 6.5, 7.0]],  
    ["Bia", [9.5, 9.0, 8.5]]  
]  
  
for aluno in alunos:  
    nome = aluno[0]  
    notas = aluno[1]  
    media = sum(notas) / len(notas)  
    print(f"{nome} tem média {round(media, 1)}")
```

EXEMPLO 2: Agenda com horários. Listando e fazendo o desempacotamento.

```
agenda = [  
    ["08:00", "Reunião com o time"],  
    ["10:30", "Estudar Python"],  
    ["14:00", "Ir ao médico"]  
]  
  
for evento in agenda:  
    horario, descricao = evento  
    print(f"Às {horario} - {descricao}")
```

## DESEMPACOTAMENTO DE LISTA:

```
horario = evento[0]  
descricao = evento[1]
```



# TUPLAS

Tuplas em Python é semelhante a listas, a diferença é que listas são mutáveis, tuplas não.

SINTAXE:

```
tupla = ('valor1', 'valor2', 'valor3')
```

As tuplas são definidas por parênteses `()`, e os elementos são separados por vírgulas.

ACESSANDO VALORES/ELEMENTOS DE UMA TUPLA (IGUAL A LISTAS):

```
print(tupla[0])
print(tupla[1])
print(tupla[2])
print(f'Há {len(tupla)} elementos dentro da tupla')

for i in tupla:
    print(f'Valores: {i}')
```

```
valor1
valor2
valor3
Há 3 elementos dentro da tupla
Valores: valor1
Valores: valor2
Valores: valor3
```

# TUPLAS

## EXEMPLOS PRÁTICOS:

EXEMPLO 1: Armazenando dados fixos (como CPF, nome...)

```
cliente = ("João", "123.456.789-00")  
  
nome, cpf = cliente  
print(f"{nome} - CPF: {cpf}")
```

EXEMPLO 2: Dias da semana

```
dias_da_semana = ("Seg", "Ter", "Qua", "Qui", "Sex", "Sáb", "Dom")  
print(dias_da_semana[0])
```

# LISTAS / TUPLAS

## QUAIS OS CENÁRIOS EM QUE USAMOS LISTAS OU TUPLAS:

### TUPLAS:

- Coordenadas de um ponto no mapa (ex: latitude e longitude).
- Informações de identidade de uma pessoa (nome e CPF). Dias da semana ou meses do ano (não mudam nunca).
- Resultado de uma operação que devolve 2 ou mais valores (ex: resultado e status).
- Dados lidos de um banco de dados que você só vai exibir, não alterar.

### LISTAS:

- Lista de tarefas, onde você pode adicionar ou remover atividades.
- Carrinho de compras, com produtos que o usuário vai adicionando.
- Lista de alunos que pode mudar conforme pessoas entram ou saem.
- Notas de uma prova, onde você calcula a média e pode atualizar.



# LISTAS / TUPLAS

## EXEMPLO PRÁTICO LISTAS E TUPLAS:

### CADASTRO DE ALUNOS E NOTAS

Crie um programa em Python que:

Cadastre 3 alunos.

Para cada aluno, armazene:

- Seu nome (como string).
- Sua matrícula (como tupla, pois não muda).
- Uma lista com 3 notas.

Ao final, mostre:

- O nome, matrícula, todas as notas e a média de cada aluno.
- A quantidade total de alunos aprovados (média  $\geq 7.0$ )

# LISTAS / TUPLAS

```
# Lista para armazenar os dados de todos os alunos
alunos = []

# Loop para cadastrar 3 alunos
for i in range(3):
    print(f"\nAluno {i+1}:") # Exibe "Aluno 1", "Aluno 2", etc.

    nome = input("Nome: ") # Lê o nome do aluno
    matricula = input("Matrícula: ") # Lê a matrícula do aluno (como texto)

    # Lista que armazenará as 3 notas do aluno
    notas = []
    for j in range(3): # Pede 3 notas
        nota = float(input(f"Nota {j+1}: ")) # Converte para float
        notas.append(nota) # Adiciona à lista de notas

    # Cria uma tupla com os dados do aluno:
    # (nome, (matrícula, [notas]))
    # A matrícula está dentro de outra tupla para ser imutável
    dados_aluno = (nome, matricula, notas)

    # Adiciona os dados do aluno à lista geral de alunos
    alunos.append(dados_aluno)
```

# LISTAS / TUPLAS

```
# Relatório final
print("\nRELATÓRIO FINAL:")
aprovados = 0 # Contador de alunos aprovados

# Percorre todos os alunos cadastrados
for aluno in alunos:
    nome, matricula, notas = aluno # Desempacota os dados
    media = sum(notas) / len(notas) # Calcula a média das notas

    # Exibe os dados do aluno
    print(f"\nNome: {nome}")
    print(f"Matrícula: {matricula}") # A matrícula está dentro de uma tupla, por isso usamos [0]
    print(f"Notas: {notas}")
    print(f"Média: {round(media, 2)}") # Arredonda a média para 2 casas decimais

    # Verifica se o aluno está aprovado
    if media >= 7.0:
        aprovados += 1 # Incrementa o contador de aprovados

# Mostra o total de alunos aprovados
print(f"\nTotal de alunos aprovados: {aprovados}")
```

# ATIVIDADE

1. Crie um programa que: Crie uma lista vazia chamada `compras`. Peça ao usuário para digitar 3 itens de mercado. Mostre todos os itens ao final.
2. Crie uma tupla com 3 informações sobre você: nome, idade e cidade. Depois, mostre uma frase com esses dados.
3. Peça para o usuário digitar 3 notas. Armazene em uma lista, calcule e mostre a média com 1 casa decimal.

4. Crie um programa que:

Permita ao usuário cadastrar 3 filmes que ele assistiu recentemente. Para cada filme, peça:

- O título do filme
- A nota que o usuário dá de 0 a 10

Guarde essas informações em uma lista aninhada (cada filme é uma sublista com título e nota). Ao final, exiba todos os filmes com sua respectiva nota, e destaque quais receberam nota maior ou igual a 8 como "Recomendado!".

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 1:
compras = []

for i in range(3):
    item = input(f"Digite o item {i+1}: ")
    compras.append(item)

print("\nLista de compras:")
for item in compras:
    print("-", item)
```

```
# ATIVIDADE 3:
notas = []

for i in range(3):
    nota = float(input(f"Digite a nota {i+1}: "))
    notas.append(nota)

media = sum(notas) / len(notas)

print("Notas:", notas)
print("Média final:", round(media, 1))
```

```
# ATIVIDADE 2:
pessoa = ("Lucas", 25, "Belo Horizonte")

print(f"Olá! Meu nome é {pessoa[0]}, tenho {pessoa[1]} anos e moro em {pessoa[2]}.")
```

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 4:
filmes = []

for i in range(3):
    titulo = input(f"Digite o título do filme {i+1}: ")
    nota = float(input(f"Digite sua nota para '{titulo}': "))
    filmes.append([titulo, nota])

print("\nFilmes assistidos:")
for filme in filmes:
    titulo, nota = filme
    if nota >= 8:
        print(f"Filme: {titulo} | Nota: {nota} - Recomendado!")
    else:
        print(f"Filme: {titulo} | Nota: {nota}")
```

# ATIVIDADE 2: RECAPTULANDO

1. Crie um programa que simule um menu de opções com as seguintes ações:

- Dizer "Olá"
- Dizer a hora (use um print fixo, ex: "São 14:30")
- Sair

O programa deve repetir até o usuário escolher sair.

2. Peça ao usuário para cadastrar 3 produtos. Para cada produto, informe: nome e preço.

Guarde essas informações em uma lista. Depois, mostre todos os produtos e seus preços e a média dos preços

3. Cadastre 4 visitantes de um museu. Para cada um, registre nome e idade. Guarde os dados em uma lista de tuplas. Depois mostre quem é o visitante mais velho e a média de idade.

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 1:
```

```
while True:
```

```
    print("\nMenu:")
```

```
    print("1 - Dizer Olá")
```

```
    print("2 - Mostrar Hora")
```

```
    print("3 - Sair")
```

```
    opcao = input("Escolha uma opção: ")
```

```
    if opcao == "1":
```

```
        print("Olá!")
```

```
    elif opcao == "2":
```

```
        print("São 14:30 (simulado)")
```

```
    elif opcao == "3":
```

```
        print("Saindo...")
```

```
        break
```

```
    else:
```

```
        print("Opção inválida!")
```

```
# ATIVIDADE 2:
```

```
produtos = []
```

```
for i in range(3):
```

```
    nome = input(f"Nome do {i+1}º produto: ")
```

```
    preco = float(input("Preço: "))
```

```
    produtos.append([nome, preco])
```

```
total = 0
```

```
print("\nProdutos cadastrados:")
```

```
for p in produtos:
```

```
    print(f"{p[0]} - R$ {p[1]:.2f}")
```

```
    total += p[1]
```

```
media = total / 3
```

```
print(f"\nMédia dos preços: R$ {media:.2f}")
```



# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 3
visitantes = []

for i in range(4):
    nome = input(f"Nome do {i+1}º visitante: ") # Solicita o nome
    idade = int(input("Idade: ")) # Solicita a idade e converte para inteiro
    visitantes.append((nome, idade)) # Armazena como tupla na lista

# Inicializa variáveis
mais_velho = visitantes[0] # Assume que o primeiro visitante é o mais velho (por enquanto)
soma_idades = 0 # Inicializa a soma das idades

# Percorre todos os visitantes para calcular a soma das idades e encontrar o mais velho
for v in visitantes:
    # soma_idades = soma_idades + v[1]
    soma_idades += v[1] # Soma a idade do visitante atual
    if v[1] > mais_velho[1]: # Se a idade atual for maior que a do mais velho registrado
        mais_velho = v # Atualiza o mais velho


# Calcula a média das idades
media = soma_idades / 4

# Exibe os resultados
print(f"\nVisitante mais velho: {mais_velho[0]} ({mais_velho[1]} anos)")
print(f"Média de idade: {media:.1f}")
```

# FUNÇÕES

Funções são blocos de código que realizam determinadas tarefas que normalmente precisam ser executadas diversas vezes dentro de uma aplicação. Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

SINTAXE:



```
1 def funcao():  
2     print("Bloco de código")
```

Observando essa função, podemos extrair algumas informações, iniciando com a palavra reservada para **funções** **def** o nome atribuído à função **funcao** e os **parênteses ()** utilizado para definição dos dados de entrada da função, também chamados de **parâmetros**.

# FUNÇÕES

Em seguida usa-se dois pontos `:` e abaixo o bloco de código a ser executado, que neste caso é apenas imprimir de uma string.

Para “chamar” uma função, utilizamos o nome que foi definido, dessa forma:

```
def funcao():  
    print("Bloco de código")  
  
funcao()
```

```
Bloco de código
```

# FUNÇÕES

- `def`: diz ao Python que você está definindo uma função.
- `nome_da_funcao`: pode ser qualquer nome, tipo `soma`, `mostrar_nome`, etc.
- `(parâmetros)`: são valores que você envia pra função usar. Pode ser vazio também.
- `return`: é o que a função devolve. (Não é obrigatório.)

```
def mostrar_nome(nome):  
    print(f"Nome: {nome}")
```

```
mostrar_nome("Mariana")  
mostrar_nome("Cleide")  
mostrar_nome("Geovanna")
```

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

```
resultado = soma(3, 4)  
print(resultado)
```

# FUNÇÕES

## 1 – Função sem parâmetro:

```
# EXEMPLO 2: Sem parametro
def saudacao():
    print("Olá! Seja bem-vindo.")

saudacao() # Chamando a função
```

## 2 – Função com parâmetro:

```
# EXEMPLO 3: Com parametro
def saudacao(nome):
    print(f"Olá, {nome}!")

saudacao("Maria")
```

## 3 – Função para verificar se o numero é par:

```
# EXEMPLO 4: Verificar se o numero é par
def eh_par(numero):
    return numero % 2 == 0

print(eh_par(4)) # True
print(eh_par(5)) # False
```

# FUNÇÕES

## EXEMPLO PRÁTICO:

```
# FUNÇÃO DE CALCULADORA:
def calculadora(a, b, operacao):
    if operacao == '+':
        return a + b
    elif operacao == '-':
        return a - b
    elif operacao == '*':
        return a * b
    elif operacao == '/':
        if b == 0:
            return "Erro: divisão por zero"
        return a / b
    else:
        return "Operação inválida"

# Exemplo de uso
a = float(input("Digite o primeiro numero da operação: "))
b = float(input("Digite o segundo numero da operação: "))
operacao = input("Digite qual a operação (soma, subtração, multiplicação ou divisão): ")
print(calculadora(a, b, operacao))
```

```
# CONTANDO CARACTERES DE UMA STRING
def contar_caracteres(texto):
    return len(texto)

# Exemplo de uso
texto = input('Digite uma palavra: ')
print(f'A palavra que voce digitou tem {contar_caracteres(texto)} letras')
```

# ATIVIDADE

1 - Crie uma função chamada `calcular_imc` que receba o peso (em kg) e a altura (em metros) de uma pessoa e retorne o IMC ( $\text{peso} / \text{altura}^2$ )

A partir da função anterior, crie uma função `classificar_imc` que retorne a categoria do IMC:

- Abaixo de 18.5: Abaixo do peso
- Entre 18.5 e 24.9: Peso normal
- Entre 25.0 e 29.9: Sobrepeso
- 30.0 ou mais: Obesidade

OBS: Peça para o usuário informar os valores

2 - Crie uma função `area_circulo` que receba o raio de um círculo e retorne sua área. Use 3.14 como valor aproximado de pi.

3 - Crie uma função `celsius_para_fahrenheit` que receba uma temperatura em Celsius e retorne o equivalente em Fahrenheit. Peça para o usuário informar um valor.

4 - Crie uma função `mostrar_tabuada(numero)` que receba um número inteiro e imprima a tabuada dele de 1 a 10. O programa deve permitir que o usuário digite vários números, até digitar 0 para encerrar.

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 1:
def calcular_imc(peso, altura):
    return peso / (altura ** 2)

def classificar_imc(imc):
    if imc < 18.5:
        return "Abaixo do peso"
    elif imc < 25:
        return "Peso normal"
    elif imc < 30:
        return "Sobrepeso"
    else:
        return "Obesidade"

peso = float(input("Informe o seu peso: "))
altura = float(input("Informe a sua altura: "))
imc = calcular_imc(peso, altura)
print(classificar_imc(imc))
```

```
# ATIVIDADE 2:
def area_circulo(raio):
    pi = 3.14
    return pi * raio ** 2

raio = float(input('Informe um valor para calcular a area do circulo: '))
print(area_circulo(raio))
```



# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 3:
def celsius_para_fahrenheit(celsius):
    return (celsius * 9 / 5) + 32

celsius = int(input("Informe o valor que você gostaria de converter para fahrenheit: "))
print(celsius_para_fahrenheit(celsius))
```

```
# ATIVIDADE 4:
import os

def mostrar_tabuada(numero):
    print(f"\nTabuada do {numero}")
    for i in range(1, 11):
        print(f"{numero} x {i} = {numero * i}")

while True:
    os.system('cls')
    num = int(input("Digite um número para ver a tabuada (0 para sair): "))
    if num == 0:
        break
    mostrar_tabuada(num)
    input("\nPressione Enter para continuar...")
```

# TRATAMENTO DE ERROS

O tratamento de erros em Python é feito com o bloco `try-except`. Ele serve para lidar com exceções de forma controlada, sem que o programa quebre inesperadamente.

O QUE SÃO EXCEÇÕES:

Exceções são eventos que ocorrem quando algo dá errado durante a execução do programa. Elas interrompem o fluxo normal se não forem tratadas.

```
x = 10 / 0 # ZeroDivisionError
```

```
PS G:\PythonFundamentos1\AulaPythonSábado> & C:/Python312/python.exe "g:/PythonFundamentos1/AulaPythonSábado
Traceback (most recent call last):
  File "g:\PythonFundamentos1\AulaPythonSábado\Aula 7 - Try Except\exemplo_pratico.py", line 3, in <module>
    x = 10 / 0 # ZeroDivisionError
        ~~~~^
ZeroDivisionError: division by zero
PS G:\PythonFundamentos1\AulaPythonSábado> █
```

Sem tratamento, isso faz o programa quebrar. Com tratamento, ele continua funcionando.

# TRATAMENTO DE ERROS

## SINTAXE:

```
# SINTAXE:
try:
    n = int(input("Digite um número: "))
    resultado = 10 / n
except ValueError:
    print("Você não digitou um número.")
except ZeroDivisionError:
    print("Divisão por zero não é permitida.")
else:
    print("Resultado:", resultado)
finally:
    print("Fim da operação.")
```

- **try:** tenta o bloco.
- **except:** executado se erro ocorrer.
- **else:** só executa se não houver erro.
- **finally:** sempre executa (com erro ou não).

# TRATAMENTO DE ERROS

## TIPOS COMUNS DE EXCEÇÕES EM PYTHON:

Exceção	Quando ocorre
<code>ValueError</code>	Conversão inválida (ex: <code>int("abc")</code> )
<code>ZeroDivisionError</code>	Divisão por zero
<code>IndexError</code>	Índice inválido em listas ou tuplas
<code>KeyError</code>	Chave inexistente em dicionários
<code>FileNotFoundError</code>	Arquivo não encontrado
<code>TypeError</code>	Operações com tipos incompatíveis (ex: <code>3 + "a"</code> )

# TRATAMENTO DE ERROS

## EXEMPLOS:

### 1 - ValueError:

```
# EXEMPLO 1:
def solicitar_idade():
    try:
        idade = int(input("Digite sua idade: "))
        print(f"Idade registrada: {idade} anos")
    except ValueError:
        print("Erro: você deve digitar um número inteiro.")

solicitar_idade()
```

### 2 - ZeroDivisionError e ValueError:

```
# EXEMPLO 2:
def dividir():
    try:
        numerador = float(input("Digite o numerador: "))
        denominador = float(input("Digite o denominador: "))
        resultado = numerador / denominador
        print(f"Resultado: {resultado}")
    except ZeroDivisionError:
        print("Erro: divisão por zero não é permitida.")
    except ValueError:
        print("Erro: entrada inválida. Digite números válidos.")

dividir()
```

### 3 - Exception:

```
# EXEMPLO 3:
def operacao_risco():
    try:
        x = int("dez") # Isso vai gerar ValueError
    except Exception as e:
        print(f"Ocorreu um erro inesperado: {e}")

operacao_risco()
```

### EXCEPTION:

Captura qualquer exceção/erro genérica (porque toda exceção em Python herda de Exception).  
O as e é usado para guardar a exceção em uma variável chamada e.



# TRATAMENTO DE ERROS

## 4 - IndexError:

```
# EXEMPLO 4:
lista = [10, 20, 30]

try:
    print(lista[5]) # índice fora do alcance
except IndexError:
    print("Erro: índice fora da lista!")
```

## 5 - Validação de senha:

```
# EXEMPLO: Validação de senha:
def verificar_senha(senha):
    if len(senha) < 6:
        raise ValueError("Senha muito curta!")
    if " " in senha:
        raise ValueError("Senha não pode conter espaços.")
    print("Senha válida.")

try:
    verificar_senha("abc")
except ValueError as erro:
    print("Erro:", erro)
```

### RAISE:

raise é uma palavra-chave usada para disparar (lançar) uma exceção manualmente em Python



# ATIVIDADE

- 1 - Peça nome, idade e email e apresente no terminal. Use Exception para capturar qualquer erro inesperado no processo.
- 2- Crie um programa que peça ao usuário uma temperatura em Celsius e converta para Fahrenheit. Trate o erro se ele digitar texto ou valor inválido.
- 3- Crie uma função chamada main que, dentro de um loop, peça ao usuário para escolher uma opção. Use try e except para tratar entradas inválidas (não numéricas). Se o usuário digitar 1, chame capturar\_dados(), se digitar 2, chame converter\_celsius\_para\_fahrenheit(), e se digitar 0, encerre o programa. Para números fora do intervalo, exiba uma mensagem de opção inválida.

# ATIVIDADE: RESOLUÇÃO

```
#1
def capturar_dados():
    try:
        nome = input("Digite seu nome: ")
        idade = int(input("Digite sua idade: "))
        email = input("Digite seu email: ")
        print(f"\nNome: {nome}\nIdade: {idade}\nEmail: {email}")
    except Exception as erro:
        print("Erro inesperado:", erro)
```

```
#2
def converter_celsius_para_fahrenheit():
    try:
        celsius = float(input("Temperatura em Celsius: "))
        fahrenheit = (celsius * 9/5) + 32
        print(f"{celsius}°C = {fahrenheit:.2f}°F")
    except ValueError:
        print("Erro: Digite um valor numérico válido.")
```

```
#3
def main():
    while True:
        print("\n=== Menu Principal ===")
        print("1 - Capturar dados")
        print("2 - Converter Celsius para Fahrenheit")
        print("0 - Sair")

        try:
            opcao = int(input("Escolha: "))
            if opcao == 1:
                capturar_dados()
            elif opcao == 2:
                converter_celsius_para_fahrenheit()
            elif opcao == 0:
                print("Encerrando...")
                break
            else:
                print("Opção inválida.")
        except ValueError:
            print("Erro: Digite um número válido.")

main()
```



# MÓDULOS

Módulos são arquivos que contêm código Python – como funções, variáveis e classes – que podem ser reutilizados em diferentes programas. Eles ajudam a manter o código mais organizado, evitam repetições desnecessárias e facilitam a manutenção.

O uso de módulos é importante porque contribui para a economia de tempo e o aumento da produtividade. Ao reutilizar código já escrito em outros projetos, você evita retrabalho, acelera o processo de desenvolvimento e garante mais consistência entre diferentes aplicações.

## TIPOS DE MÓDULOS:

Módulos internos:

- math
- random
- os

Módulos externos (bibliotecas):

- pandas
- requests
- flask

Módulos personalizados:

- Criados pelo próprio programador

# MÓDULOS

## MÓDULOS INTERNOS:

**MATH:** Utilizamos para operações matemáticas mais complexas, como raiz quadrada, seno, logaritmos etc.

```
# IMPORTANDO O MÓDULO
import math

print(math.sqrt(25))          # Raiz quadrada → 5.0
print(math.pow(3, 2))         # Potência → 9.0
print(math.factorial(4))      # Fatorial → 24
print(math.pi)                # Valor de  $\pi$ 
print(math.sin(math.radians(30))) # Seno de 30°
```

```
# COMPARANDO: POTENCIAÇÃO
print(2 ** 3)                 # 8
print(math.pow(2, 3))         # 8.0 (sempre float)
```

# MÓDULOS

## MÓDULOS INTERNOS:

**RANDOM:** Utilizamos para gerar números aleatórios ou sortear itens de uma coleção.

```
# IMPORTANDO O MÓDULO
import random

print(random.randint(1, 100))      # Número inteiro entre 1 e 100
print(random.random())             # Float entre 0 e 1
print(random.uniform(10, 20))     # Float entre 10 e 20

itens = ["Python", "Java", "C++"]
print(random.choice(itens))        # Escolhe um item aleatório

random.shuffle(itens)
print(itens)                       # Embaralha a lista
```

# MÓDULOS

## MÓDULOS INTERNOS:

**OS:** Utilizamos para interagir com o sistema de arquivos, diretórios, renomear e excluir arquivos, etc.

```
import os

print(os.getcwd())           # Mostra o diretório atual
os.mkdir("teste")           # Cria a pasta 'teste'
os.rename("teste", "novo_teste") # Renomeia
os.remove("arquivo.txt")     # Remove arquivo (se existir)
```

```
from os import system
```

```
system('cls')
|
from os import getcwd

print("Diretório atual:", getcwd())
```

```
from os import system, getcwd
```

```
system('cls')
print('Diretório atual:', getcwd())
#OU
print(f'Diretório atual: {getcwd()}')
```

# MÓDULOS

EXEMPLO **OS**: Criando uma pasta e um arquivo .txt

```
import os

# Mostra o diretório atual
print("Diretório atual:", os.getcwd())

# Cria a pasta 'teste' se ela não existir
if not os.path.exists("teste"):
    os.mkdir("teste")
    print("Pasta 'teste' criada.")

# Caminho completo do novo arquivo dentro da pasta
# os.path.join(...) junta corretamente os nomes de caminho, independente
# do sistema operacional (Windows usa \, Linux usa /).
caminho_arquivo = os.path.join("teste", "arquivo.txt")

# Cria um arquivo e escreve algo nele
with open(caminho_arquivo, "w") as arquivo:
    arquivo.write("Este é um arquivo criado dentro da pasta 'teste'.\n")

# with: Garante abertura e fechamento seguros
# open(...): Abre o arquivo para leitura, escrita, etc.
# as arquivo   Dá um nome temporário para o objeto do arquivo

print(f"Arquivo criado em: {caminho_arquivo}")
```

# MÓDULOS

## EXEMPLO **RANDOM**: Sorteando alunos

```
# EXEMPLO RANDOM: Sorteando alunos
import random

alunos = ["Ana", "Bruno", "Carla", "Daniel", "Eduarda"]

# Sorteia um nome aleatório da lista
escolhido = random.choice(alunos)

print(f"O aluno sorteado foi: {escolhido}")
```

# ATIVIDADE

1 - Crie um jogo em que o computador sorteia um número aleatório entre 1 e 10, e o jogador tenta adivinhar qual é esse número. O programa deve dizer se o palpite está certo ou errado. Após cada rodada, pergunte ao jogador se ele quer jogar novamente. O jogo continua até o jogador digitar "não".

2 - Crie um jogo de Pedra, Papel ou Tesoura, onde o jogador escolhe uma das três opções e o computador escolhe aleatoriamente. O jogo deve mostrar quem ganhou a rodada. Depois de cada partida, o programa deve perguntar se o jogador quer continuar. O jogo só termina quando o jogador digitar "não".

# ATIVIDADE: RESOLUÇÃO

```
# ATIVIDADE 1:
import random

print("Bem-vindo ao jogo: Adivinhe o Número!\n")

while True:
    numero_secreto = random.randint(1, 10)
    palpite = int(input("Adivinhe o número que estou pensando (entre 1 e 10): "))

    if palpite == numero_secreto:
        print("Parabéns! Você acertou o número!\n")
    else:
        print(f"Você errou. O número correto era {numero_secreto}.\n")

    continuar = input("Quer jogar novamente? (sim/não): ").lower()
    if continuar != "sim":
        print("Obrigado por jogar! Até a próxima.")
        break
```

```
# ATIVIDADE 2:
import random

opcoes = ["pedra", "papel", "tesoura"]

print("Bem-vindo ao jogo: Pedra, Papel ou Tesoura!\n")

while True:
    jogador = input("Escolha pedra, papel ou tesoura: ").lower()
    if jogador not in opcoes:
        print("Escolha inválida. Tente novamente.\n")
        continue

    computador = random.choice(opcoes)

    print(f"\nVocê escolheu: {jogador}")
    print(f"Computador escolheu: {computador}")

    if jogador == computador:
        print("Empate!\n")
    elif (jogador == "pedra" and computador == "tesoura") or \
         (jogador == "tesoura" and computador == "papel") or \
         (jogador == "papel" and computador == "pedra"):
        print("Você venceu!\n")
    else:
        print("Você perdeu!\n")

    continuar = input("Quer jogar novamente? (sim/não): ").lower()
    if continuar != "sim":
        print("Obrigado por jogar! Até logo.")
        break
```



# CLASS

Uma classe é como um molde ou modelo para criar objetos (instâncias). Ela define quais atributos (características) e métodos (comportamentos) um objeto terá.

## POR QUE USAMOS CLASSES:

Imagine que você quer criar um programa que modele carros. Poderia ser escrito da seguinte forma:

```
class.py > ...  
1  carro1_marca = "Toyota"  
2  carro1_modelo = "Corolla"  
3  carro1_ano = 2020  
4  
5  carro2_marca = "Honda"  
6  carro2_modelo = "Civic"  
7  carro2_ano = 2022
```

Essa é uma maneira que daria certo, mas não é eficiente. Com classes podemos criar um molde que representa um carro, e depois criar quantos carros quisermos a partir desse molde.

# CLASS

## SINTAXE:

```
# SINTAXE:
class NomeDaClasse:
    def __init__(self, parametros):
        # atributos do objeto
        self.atributo = valor

    def metodo(self):
        # comportamento do objeto
        pass
```

### class NomeDaClasse:

- Define a classe
- O nome de classes geralmente começa com letra maiúscula.

### def \_\_init\_\_(self, ...):

- Método especial chamado CONSTRUTOR
- Ele é automaticamente executado quando você cria um objeto da classe
- Serve para inicializar os atributos do objeto

## self:

- Sempre aparece como primeiro parâmetro de qualquer método dentro da classe
- É uma referencia ao próprio objeto que esta sendo criado ou modificado
- Com o self conseguimos acessar os atributos e métodos do objeto

# CLASS

EXEMPLO:

1 - CRIANDO UMA CLASSE COM ATRIBUTOS E METODOS:

```
# EXEMPLO 1: Criando uma classe PESSOA
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome # Atributo nome
        self.idade = idade # Atributo idade

    def falar(self):
        print(f"{self.nome} está falando.")

    def aniversario(self):
        self.idade += 1
        print(f"{self.nome} fez aniversário e agora tem {self.idade} anos!")
```

2 - INSTANCIANDO / CRIANDO UM OBJETO PARA A CLASSE:

Para dar valores aos atributos nome e idade, usamos a instanciação da classe:

```
nome = input("Digite o seu nome: ")
idade = int(input("Digite a sua idade: "))
p1 = Pessoa(nome, idade)
p2 = Pessoa('Geovanna', '12')
```

3 - ACESSANDO E ALTERANDO ESSES ATRIBUTOS COM O OBJETO CRIADO:

```
# ACESSANDO E ALTERANDO OS ATRIBUTOS
print(p1.nome)
print(p1.idade)

p2.nome = "Fulano"
p2.idade = 31
```

4 - CHAMANDO OS METODOS / FUNÇÕES DAS CLASSES:

```
p1.falar()
p1.aniversario()
```

# CLASS

EXEMPLO PRÁTICO:

SIMULANDO CONTA BANCÁRIA:

```
# EXEMPLO PRÁTICO: SIMULANDO CONTA BANCARIA
class ContaBancaria:
    def __init__(self, titular, saldo=0):
        self.titular = titular
        self.saldo = saldo

    def depositar(self, valor):
        self.saldo += valor
        print(f"Depósito de R$ {valor:.2f} realizado.")

    def sacar(self, valor):
        if valor <= self.saldo:
            self.saldo -= valor
            print(f"Saque de R$ {valor:.2f} realizado.")
        else:
            print("Saldo insuficiente.")

    def consultar_saldo(self):
        print(f"Saldo atual de {self.titular}: R$ {self.saldo:.2f}")
```

```
conta1 = ContaBancaria("João")

conta1.depositar(200)
conta1.sacar(50)
conta1.consultar_saldo()
```

Dentro de uma classe, métodos são funções que representam comportamentos dos objetos.

Assim como funções normais em Python, esses métodos podem receber parâmetros adicionais além de self, para permitir entrada de dados no momento da execução.

# CLASS

EXEMPLO PRÁTICO:

SIMULANDO CONTA BANCÁRIA:

```
# EXEMPLO PRATICO: SIMULANDO PRODUTO
class Produto:
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco

    def aplicar_desconto(self, porcentagem):
        desconto = self.preco * (porcentagem / 100)
        self.preco -= desconto
        print(f"Novo preço de {self.nome}: R$ {self.preco:.2f}")

p1 = Produto("Notebook", 3000)

p1.aplicar_desconto(10)
```

# CLASS

## EXEMPLO PRÁTICO:

Crie um sistema simples de cadastro de tarefas usando classes em Python. O programa deve permitir ao usuário:

- Adicionar uma nova tarefa (descrição).
- Listar todas as tarefas.
- Remover uma tarefa pelo número.

Use uma classe chamada `ListaDeTarefas`, com métodos para adicionar, listar e remover tarefas. O sistema deve rodar em um loop while, com um menu de opções até o usuário digitar “sair”.

```
# EXEMPLO PRATICO: CADASTRO DE TAREFAS
class ListaDeTarefas:
    def __init__(self):
        self.tarefas = []

    def adicionar_tarefa(self, descricao):
        self.tarefas.append(descricao)
        print("Tarefa adicionada com sucesso.")

    def listar_tarefas(self):
        if not self.tarefas:
            print("Nenhuma tarefa cadastrada.")
        else:
            print("\nLista de Tarefas:")
            for i, tarefa in enumerate(self.tarefas, start=1):
                print(f"{i}. {tarefa}")
            print()

    def remover_tarefa(self, numero):
        if 1 <= numero <= len(self.tarefas):
            removida = self.tarefas.pop(numero - 1)
            print(f"Tarefa '{removida}' removida.")
        else:
            print("Número inválido.")
```

# CLASS

## EXEMPLO PRÁTICO:

Crie um sistema simples de cadastro de tarefas usando classes em Python. O programa deve permitir ao usuário:

- Adicionar uma nova tarefa (descrição).
- Listar todas as tarefas.
- Remover uma tarefa pelo número.

Use uma classe chamada `ListaDeTarefas`, com métodos para adicionar, listar e remover tarefas. O sistema deve rodar em um loop `while`, com um menu de opções até o usuário digitar "sair".

```
# CRIANDO O OBJETO LISTA E ATRIBUINDO A CLASSE ListaDeTarefas
lista = ListaDeTarefas()

while True:
    print("\n---- MENU ----")
    print("1 - Adicionar tarefa")
    print("2 - Listar tarefas")
    print("3 - Remover tarefa")
    print("4 - Sair")

    opcao = input("Escolha uma opção: ")

    if opcao == "1":
        desc = input("Digite a descrição da tarefa: ")
        lista.adicionar_tarefa(desc)

    elif opcao == "2":
        lista.listar_tarefas()

    elif opcao == "3":
        lista.listar_tarefas()
        num = int(input("Digite o número da tarefa a remover: "))
        lista.remover_tarefa(num)

    elif opcao == "4":
        print("Saindo do programa. 🍷")
        break

    else:
        print("Opção inválida. Tente novamente.")
```

# CLASS

## EXEMPLO PRÁTICO:

Criar um objeto aluno que guarda um nome e uma lista de notas.

```
# EXEMPLO: ALUNOS COM NOTAS
class Aluno:
    def __init__(self, nome, notas):
        self.nome = nome
        self.notas = notas # Aqui está o array (lista) dentro do objeto

    def calcular_media(self):
        if self.notas:
            return sum(self.notas) / len(self.notas)
        else:
            return 0

    def exibir_informacoes(self):
        print(f"Aluno: {self.nome}")
        print(f"Notas: {self.notas}")
        print(f"Média: {self.calcular_media():.2f}")

notas_mariana = [8.5, 7.0, 9.0]
aluno1 = Aluno("Mariana", notas_mariana)
aluno1.exibir_informacoes()
```



# ATIVIDADE

1 - Crie um sistema de compras com a classe Produto. Cada produto tem nome e preço. Crie um carrinho (lista) para armazenar os produtos comprados. Mostre um menu em loop para:

- Adicionar produto
- Listar produtos no carrinho
- Calcular total
- Sair

2 - Crie um jogo de perguntas e respostas (quiz)

Crie uma classe Pergunta com enunciado, lista de alternativas e a resposta correta. Guarde várias perguntas em uma lista. Em cada rodada:

- Sorteie uma pergunta aleatória.
- Mostre as alternativas.
- Peça a resposta do usuário.
- Mostre se ele acertou ou errou e continue até o jogador escolher sair.