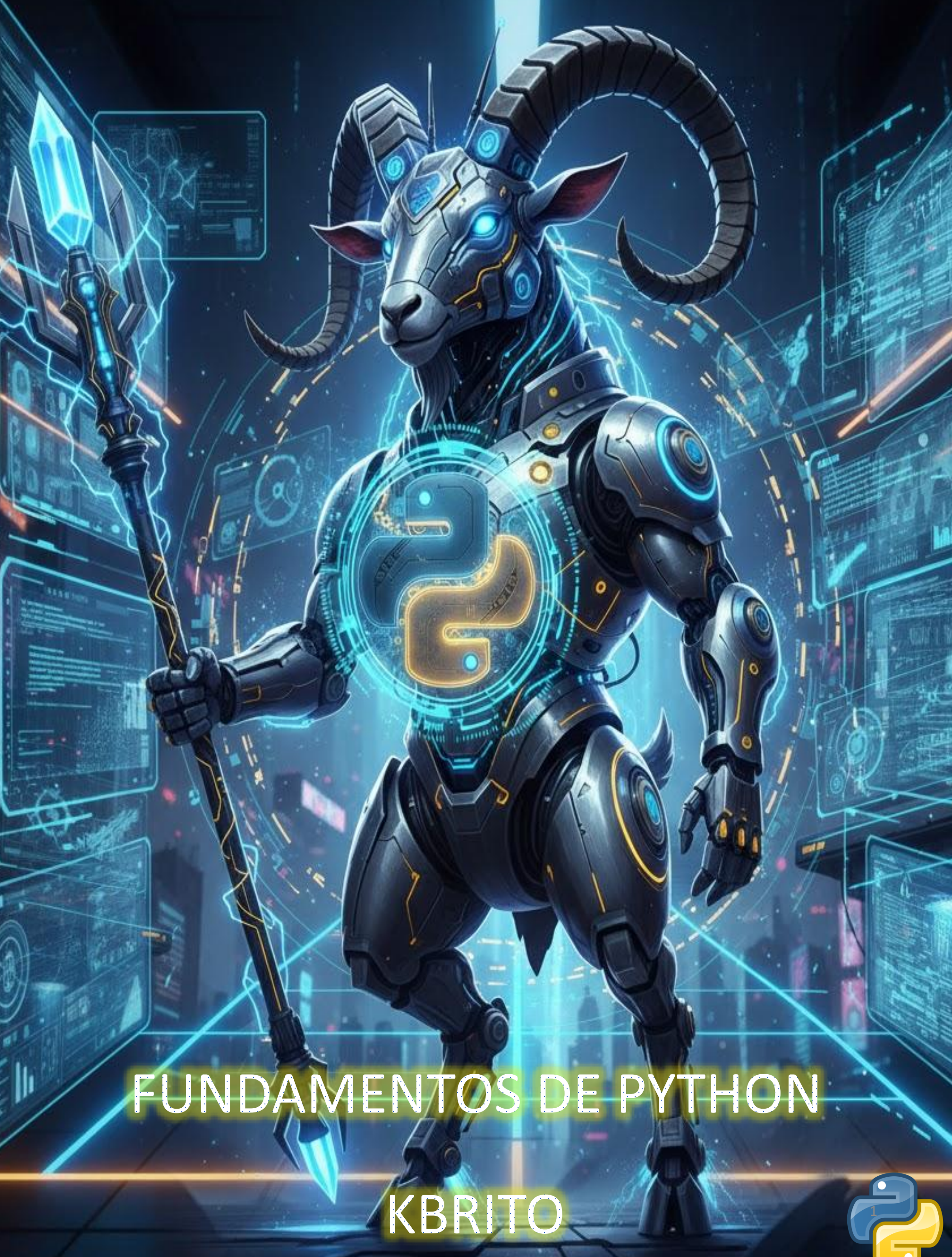

FUNDAMENTOS DE PYTHON
KBRITO

O ESSENCIAL DO PYTHON

BOAS PRÁTICAS PARA INICIAR UMA ANÁLISE DE DADOS COM PYTHON

Antes de abrir o editor e digitar as primeiras linhas de código, é importante entender que analisar dados é um processo lógico, não apenas técnico. O Python é a ferramenta, mas o raciocínio vem antes. Uma boa análise começa com clareza de propósito. Pergunte-se:

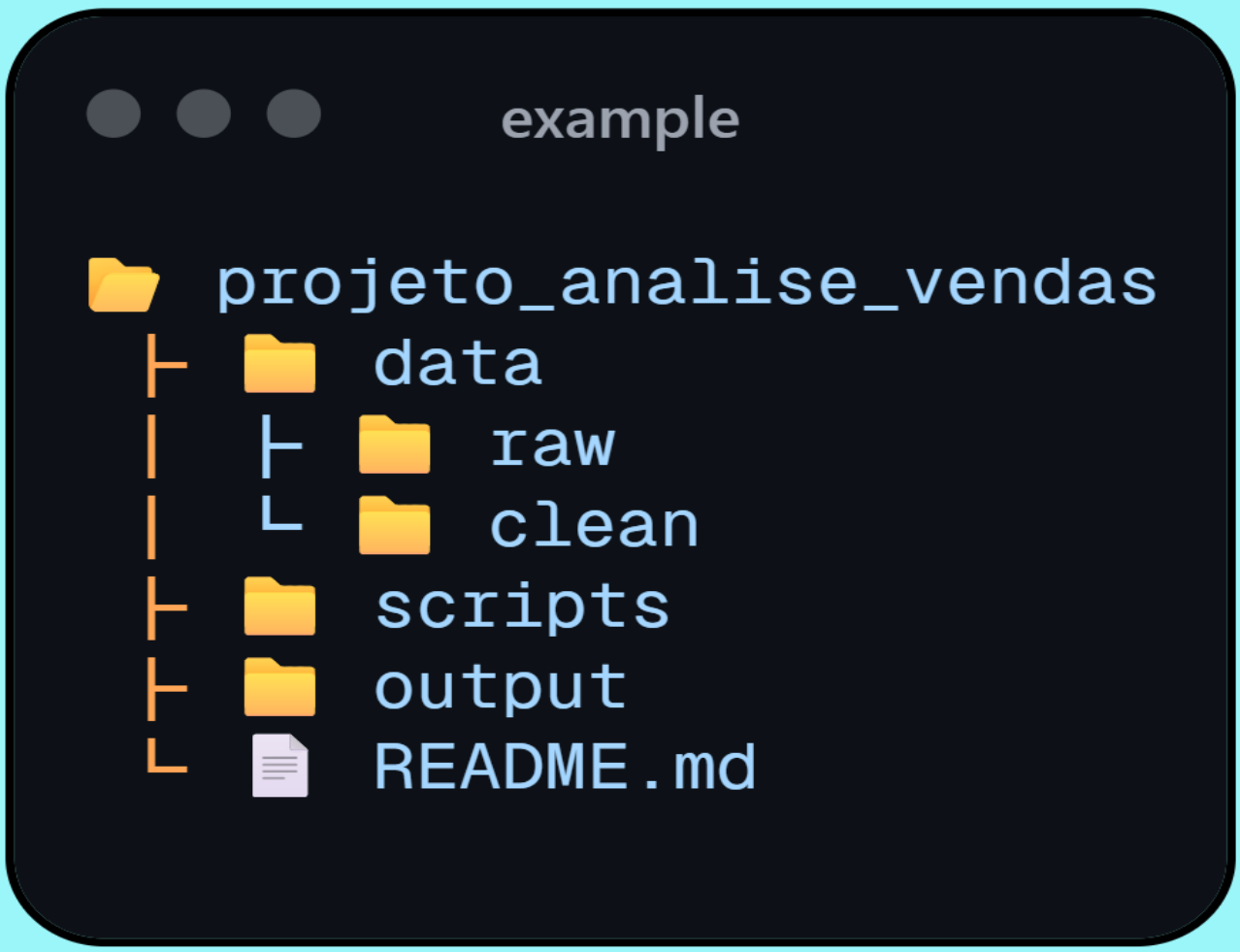
- O que quero descobrir?
- Quais dados preciso para isso?
- O que realmente é relevante para o negócio?.

Ter essas respostas evita desperdício de tempo limpando, filtrando e tratando informações que não têm impacto real e/ou não necessite. Sair fazendo com Python e d+ linguagens não é recomendável. Estruture seu raciocínio sempre!!!

DICAS DO AUTOR:

1. ESTRUTURE SEU AMBIENTE:

Organize-se desde o início. Crie uma pasta dedicada ao projeto, com subpastas para dados brutos **(/data/raw),** dados tratados **(/data/clean)**, gráficos **(/output)**, e scripts **(/scripts)**.Isso mantém o trabalho limpo e permite repetir as análises futuramente sem confusão.



💡 *Boas análises são reprodutíveis. Organização é o primeiro passo da confiabilidade.*

DICAS DO AUTOR:

2. NOMEIE E DOCUMENTE TUDO:

Evite nomes genéricos como **dados1.csv** ou **analise_final.py**. Não é raro e quase sempre acontece 😍

Prefira nomes descritivos e datas quando necessário:

example

```
vendas_2025_raw.csv
limpeza_vendas.py
relatorio_resumo_2025.xlsx
```

E dentro do código, use comentários curtos e objetivos:

example.py

```python
# Calcula o total de vendas por região
vendas_por_regiao = df.groupby("Região")["Valor"].sum()
```

01

FUNDAMENTOS RÁPIDOS DE PYTHON

Para darmos passos longos e consistentes em quaisquer que sejam a direção, a base de conhecimento precisa ter total solidez e entendimento. Tendo isso, a lógica flui de forma simples e eficaz.

RESULTADOS NA TELA

O FAMOSINHO AMIGO DELATOR NÃO PREMIADO

PRINT()

COMO FUNCIONA:
A sintaxe é básica e simples, seguida dos valores que se deseja imprimir, separados por vírgulas. Os argumentos podem ser textos (strings), variáveis, números ou a combinação deles.

example.py

```
nome = "Mundo"
print("Olá", nome)
# Saída: Olá Mundo
```

example.py

```
print("Olá", end=" ")
print("Mundo!")
# Saída: Olá Mundo!
```

example.py

```
print("Análise de Vendas Concluída com Sucesso!")
```

VARIÁVEIS

O SIMPLES BEM FEITO É MELHOR QUE PERFEITO.

Princípios de variáveis:
Seja descritivo e significativo: O nome da variável deve expressar seu propósito ou o que ela armazena, de forma concisa.
Evite abreviações obscuras: Nomes encurtados ou enigmáticos como d, x, nmd, podem gerar confusão, exigindo que o leitor adivinhe seu significado.
Seja consistente: Adote um padrão de nomenclatura e mantenha-o em todo o código. Em Python, a convenção padrão é o snake_case (palavras separadas por sublinhado) para nomes com várias palavras.

example.py

```python
#Código ruim, não colaborativo

x = 1000
d = 10
res = x / d
# O que 'x', 'd' e 'res' representam?
# É difícil entender a lógica sem comentários.
```

example.py

```python
#Código bom e colaborativo

total_vendas = 1000
numero_de_dias = 10
media_diaria_vendas = total_vendas / numero_de_dias

# O nome da variável já explica seu conteúdo e objetivo,
# tornando o código autodescritivo.
```

TIPOS DE CLASSIFICAÇÃO

O BOM JOGADOR CONHECE AS REGRAS NA PALMA DA MÃO.

Texto (str)

Usado para armazenar sequências de caracteres, como nomes, frases ou parágrafos. É definido usando aspas simples ('...'), aspas duplas ("...") ou aspas triplas ("""...""").

example.py

```
nome = "João Silva"
mensagem = 'Bem-vindo!'
```

Número (int)

Usado para armazenar números inteiros, ou seja, sem casas decimais. Pode ser positivo ou negativo.

example.py

```
idade = 25
ano_nascimento = 1999
```

Decimal (float)

Usado para armazenar números com casas decimais. A vírgula é representada por um ponto (.).

example.py

```
altura = 1.75
preco = 9.99
```

Booleano (bool)
O tipo mais simples, pois armazena apenas dois valores: **True** (verdadeiro) ou **False** (falso).
É usado para operações lógicas e condicionais, como verificar se algo é verdadeiro ou falso.
Note que a primeira letra deve ser sempre maiúscula.

example.py

```
logado = True
tem_permissao = False
```

DECISÃO — IF / ELIF / ELSE

COMO FUNCIONA:
if: A primeira e principal condição a ser verificada. O bloco de código abaixo do if só é executado se a condição for verdadeira (True).
elif: É a contração de **"else if" ("senão, se")**. Permite testar múltiplas condições. O programa só verifica um **elif,** se a condição do **if** anterior for falsa. É possível ter vários **elifs** em sequência.
else: É o "plano B". O bloco de código do **else** é executado somente se todas as condições anteriores **(if e elifs)** forem falsas. A instrução **else** não precisa de uma condição, pois cobre todos os outros casos.

EXEMPLO - IF / ELIF / ELSE:

example.py

```
temperatura = 20

if temperatura > 25:
    print("Dia quente!")
elif temperatura > 15:
    print("Dia agradável!")
else:
    print("Dia frio!")
```

EXPLICAÇÃO DO EXEMPLO:

- A condição do **if** (temperatura > 25) é testada. Como 20 não é maior que 25, ela é **falsa**.
- O programa passa para o próximo teste, o **elif** (temperatura > 15). Como 20 é maior que 15, essa condição é **verdadeira**.
- O código dentro do bloco **elif** é executado **(print("Dia agradável!"))** e a estrutura de decisão é finalizada. O **else** é ignorado.

Saída: Dia agradável!

REPETIÇÃO SIMPLES — FOR

A repetição, ou loop, for em Python é usada para iterar sobre uma sequência de elementos, como uma lista, uma string ou um intervalo de números. É a maneira mais comum e "pitônica" de repetir uma ação para cada item em um conjunto.

COMO FUNCIONA:

O loop for percorre cada item da sequência, um por um, e executa o bloco de código indentado para cada iteração.

Sintaxe básica:

example.py

```
for variavel_temporaria in sequencia:
    # bloco de código a ser repetido
```

example.py

```
produtos = ["Monitor", "Mouse", "Teclado"]
for item in produtos:
    print(f"Processando: {item}")
```

ESTRUTURA DE DADOS

As estruturas de dados são "recipientes" usados para organizar e armazenar dados de forma eficiente na memória do computador. Elas definem como os dados se relacionam entre si e quais operações podem ser feitas neles. Em Python, as estruturas de dados nativas mais comuns são as listas, tuplas, dicionários e conjuntos.

Lista (list)

O que é: Uma coleção ordenada e mutável (editável) de itens. É a estrutura mais versátil e usada em Python.

Características: Permite itens duplicados e de diferentes tipos. Os itens podem ser acessados por um índice numérico, começando do zero.

Exemplo:

example.py

```
frutas = ["maçã", "banana", "cereja"]
frutas.append("laranja") # Adiciona um item
print(frutas[0])         # Saída: maçã
```

Tupla (tuple)

O que é: Uma coleção ordenada, mas imutável (não pode ser alterada após a criação) de itens.

Características: Semelhante a uma lista, mas é usada para dados que não devem mudar, como as coordenadas de um ponto. É mais rápida que a lista para iteração.

Exemplo:

example.py

```
coordenadas = (10.0, 20.0)
# coordenadas.append(5.0) # Isso causaria um erro
print(coordenadas[1])     # Saída: 20.0
```

Dicionário (dict)

O que é: Uma coleção não ordenada e mutável de pares chave-valor.
Características: Ao invés de usar índices numéricos, os dicionários usam chaves únicas (geralmente strings) para acessar os valores. É ideal para representar dados com relações claras, como um cadastro de pessoa.

Exemplo:

```python
pessoa = {"nome": "Maria", "idade": 30, "cidade": "São Paulo"}
print(pessoa["nome"]) # Saída: Maria
pessoa["idade"] = 31  # Altera a idade
```

Conjunto (set)

O que é: Uma coleção não ordenada de itens únicos.

Características: Automaticamente remove duplicatas. É muito eficiente para testar se um item existe em uma coleção e para operações matemáticas de conjuntos (união, intersecção).

Exemplo:

```python
numeros = {1, 2, 3, 2, 1}
print(numeros) # Saída: {1, 2, 3} (duplicatas removidas)

if 2 in numeros:
    print("O número 2 está no conjunto.")
```

RESUMO

A escolha da estrutura de dados certa depende do que você precisa fazer:

Lista: Para coleções que precisam mudar de tamanho ou ordem**.**

Tupla: Para dados fixos e imutáveis**.**

Dicionário: Para dados que precisam ser acessados por um nome ou identificador específico (chave).
Conjunto: Para coleções de itens únicos onde a ordem não importa.

FUNÇÕES

As funções em Python são blocos de código reutilizáveis projetados para executar uma tarefa específica. Elas são a base para organizar o código de forma modular, facilitando a leitura, a depuração e a manutenção.

O princípio central das funções é: "Uma função deve fazer uma coisa, e fazer bem feito" (Princípio da Responsabilidade Única).

COMO FUNCIONA:

Uma função é definida usando a palavra-chave **def**, seguida por um nome descritivo e parênteses. O código que pertence à função deve ser indentado**.**

Definição: Cria a função.

Chamada: Executa o código dentro da função.

example.py

```
def saudacao():
    """Esta função imprime uma mensagem de saudação."""
    print("Olá, mundo!")

# Chamando a função para executar o bloco de código:
saudacao()
# Saída: Olá, mundo!
```

PARÂMETROS E RETORNO

As funções se tornam mais poderosas quando aceitam parâmetros (dados de entrada) e retornam um valor (resultado da operação).

1. Parâmetros (Entrada)

Permitem que você passe informações para a função trabalhar.

example.py

```python
def saudacao_personalizada(nome): # 'nome' é o parâmetro
    print(f"Olá, {nome}!")

saudacao_personalizada("Alice")
saudacao_personalizada("Bob")
# Saída:
# Olá, Alice!
# Olá, Bob!
```

2. Retorno (Saída)

A palavra-chave return permite que a função envie um valor de volta para o local onde foi chamada, para ser usado em outras partes do programa

example.py

```python
def calcular_desconto(valor, taxa):
    return valor - (valor * taxa)

print("Preço:", calcular_desconto(1000, 0.1))
```

O PRINCÍPIO DA "UMA RESPONSABILIDADE“

Manter as funções focadas em uma única tarefa evita que o código se torne complexo e difícil de testar.

Ruim: Uma função que carrega dados, processa e imprime um relatório.

Bom: Três funções separadas: uma para carregar dados, outra para processar e outra para formatar/imprimir o relatório.

Isso torna o código mais fácil de entender, depurar e reutilizar

“Uma mente que se abre para uma nova ideia jamais voltará ao seu tamanho original.”

Albert Einstein

DÊ O PRIMEIRO PASSO