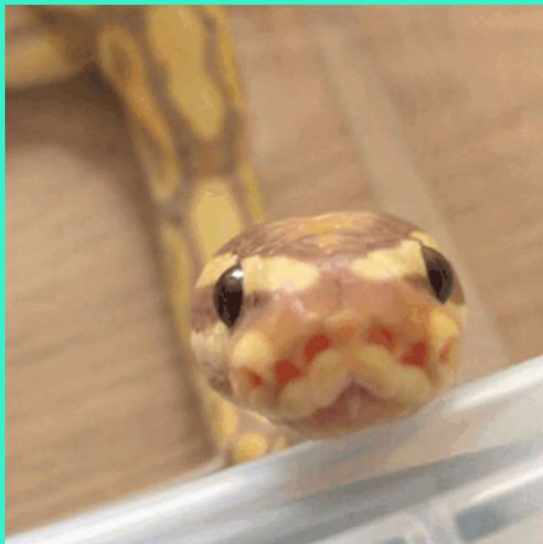
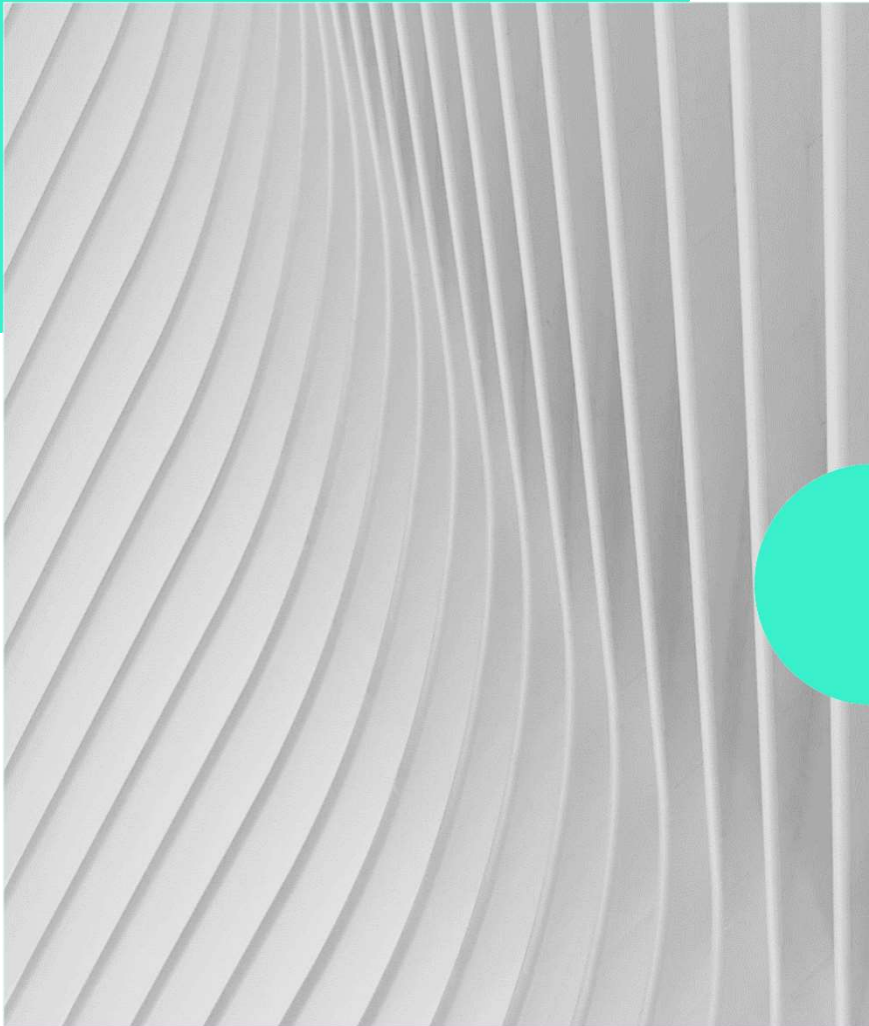


FUNÇÕES EM PYTHON





O que é uma função?

Uma função é um bloco de código reutilizável que realiza uma tarefa específica. Em Python, você pode definir sua própria função ou usar funções embutidas fornecidas pela linguagem.

Por que usar funções?

Reutilização de código: Escreva uma vez, use várias vezes.

Modularidade: Quebra o código em blocos menores e mais manejáveis.

Organização: Facilita a leitura e manutenção do código.

Redução de Erros: Ao evitar a repetição de código, você minimiza a chance de erros.

COMO DEFINIR UMA FUNÇÃO?

Para definir uma função, utilizamos a palavra-chave **def**, seguida do nome da função, parênteses () e dois pontos ":" O corpo da função é indentado.

```
python Copy code  
  
def nome_da_funcao(parametros):  
    # Bloco de código  
    # Geralmente, retorna um valor  
    return resultado
```

```
python Copy code  
  
def saudacao():  
    print("Olá! Seja bem-vindo!")
```

CHAMANDO UMA FUNÇÃO

Para executar o código dentro de uma função, você "chama" a função pelo nome seguido de parênteses ().

python

Copy code

```
saudacao() # Chama a função saudacao
```

python

Copy code

```
def saudacao(nome):  
    print(f"Olá, {nome}! Seja bem-vindo!")  
  
saudacao("Carlos")  
saudacao("Ana")
```

CHAMANDO A FUNÇÃO

Para executar o código dentro de uma função, você "chama" a função pelo nome seguido de parênteses ().

```
python Copy code  
  
def nome_da_funcao(parametros):  
    # Bloco de código  
    # Geralmente, retorna um valor  
    return resultado
```

```
python Copy code  
  
def saudacao():  
    print("Olá! Seja bem-vindo!")
```

FUNÇÕES COM RETORNO


Muitas vezes, uma função realiza um cálculo ou operação e você deseja que ela retorne um resultado. Para isso, usamos a palavra-chave `return`.

```
python Copy code  
  
def soma(a, b):  
    return a + b  
  
resultado = soma(5, 3)  
print(resultado) # Saída: 8
```

FUNÇÕES COM PARÂMETRO PADRÃO

Você pode fornecer valores padrão para parâmetros, o que significa que o usuário não precisa fornecer um valor para esses parâmetros ao chamar a função, a menos que queira alterar o valor padrão.

python

 Copy code

```
def saudacao(nome="Visitante"):
    print(f"Olá, {nome}! Seja bem-vindo!")

saudacao() # Saída: Olá, Visitante! Seja bem-vindo!
saudacao("Carlos") # Saída: Olá, Carlos! Seja bem-vindo!
```

BOAS PRÁTICAS AO USAR FUNÇÕES

Nomes Claros e Descritivos: O nome da função deve indicar claramente o que ela faz.

```
python Copy code  
  
def calc(a, b):  
    return a * b
```

NÃO RECOMENDADO

```
python Copy code  
  
def multiplicar_valores(a, b):  
    return a * b
```


BOAS PRÁTICAS AO USAR FUNÇÕES

Evite Funções Muito Longas: Funções devem ser curtas e realizar uma tarefa específica. Se uma função está muito longa, considere dividi-la em funções menores.

```
def calcular_lucro(preco_compra, preco_venda, taxa, quantidade, tipo_transacao):  
    if tipo_transacao == "compra":  
        resultado = (preco_venda - preco_compra) * quantidade - taxa  
        if resultado > 0:  
            return "Lucro"  
        else:  
            return "Prejuízo"  
    elif tipo_transacao == "venda":  
        resultado = (preco_venda - preco_compra) * quantidade - taxa  
        if resultado > 0:  
            return "Lucro"  
        else:  
            return "Prejuízo"  
    else:  
        return "Tipo de transação inválido"
```

Código Repetitivo: A mesma lógica de cálculo é repetida nos blocos if e elif. Isso não só torna o código mais extenso, mas também mais difícil de manter. Se fosse necessário mudar a fórmula de cálculo, teríamos que mudar em múltiplos lugares.

BOAS PRÁTICAS AO USAR FUNÇÕES

Evite Funções Muito Longas: Funções devem ser curtas e realizar uma tarefa específica. Se uma função está muito longa, considere dividi-la em funções menores.

```
def calcular_lucro(preco_compra, preco_venda, taxa, quantidade, tipo_transacao):  
    if tipo_transacao == "compra":  
        resultado = (preco_venda - preco_compra) * quantidade - taxa  
        if resultado > 0:  
            return "Lucro"  
        else:  
            return "Prejuízo"  
    elif tipo_transacao == "venda":  
        resultado = (preco_venda - preco_compra) * quantidade - taxa  
        if resultado > 0:  
            return "Lucro"  
        else:  
            return "Prejuízo"  
    else:  
        return "Tipo de transação inválido"
```

Verificação Inútil de Tipo de Transação: A lógica do cálculo de lucro/prejuízo não depende do tipo de transação ("compra" ou "venda") na implementação atual, tornando essa verificação desnecessária.

BOAS PRÁTICAS AO USAR FUNÇÕES

Evite Funções Muito Longas: Funções devem ser curtas e realizar uma tarefa específica. Se uma função está muito longa, considere dividi-la em funções menores.

```
def calcular_lucro(preco_compra, preco_venda, taxa, quantidade, tipo_transacao):  
    if tipo_transacao == "compra":  
        resultado = (preco_venda - preco_compra) * quantidade - taxa  
        if resultado > 0:  
            return "Lucro"  
        else:  
            return "Prejuízo"  
    elif tipo_transacao == "venda":  
        resultado = (preco_venda - preco_compra) * quantidade - taxa  
        if resultado > 0:  
            return "Lucro"  
        else:  
            return "Prejuízo"  
    else:  
        return "Tipo de transação inválido"
```

Pouca Clareza e Confusão: O fluxo lógico é confuso e pode causar erros se uma condição adicional for introduzida ou alterada. A lógica do código deveria ser mais direta e clara.

```
def calcular_lucro(preco_compra, preco_venda, taxa, quantidade):  
    resultado = (preco_venda - preco_compra) * quantidade - taxa  
    if resultado > 0:  
        return "Lucro"  
    else:  
        return "Prejuízo"
```

Eliminação de Redundâncias: A lógica do cálculo foi centralizada em um único local, o que elimina a repetição e simplifica o código.

Simplificação do Fluxo Lógico: A função não tenta diferenciar entre "compra" e "venda", o que não é necessário com a lógica atual, tornando o código mais simples e fácil de entender.

Facilidade de Manutenção: Se for necessário alterar a fórmula de cálculo, agora só há um local onde essa mudança precisa ser feita, reduzindo a chance de erros.



THANK YOU

Wagner Coutinho

<https://www.linkedin.com/in/wagner-coutinho-mf/>

wagner.filho@rarolabs.com.br