Guia Completo de Funções em Python para Leigos

Introdução: Por que Funções são Essenciais?

Imagine que você está construindo uma casa e a tarefa de "instalar uma janela" aparece 10 vezes no seu plano. Em vez de escrever a lista de passos (medir, cortar, parafusar, vedar) em cada um dos 10 lugares, você cria um manual chamado instalar_janela(). Sempre que precisar, você simplesmente consulta esse manual.

Em programação, uma **função** é exatamente isso: um bloco de código que agrupa instruções para uma tarefa específica. Você a define uma vez e a usa sempre que precisar.

Você já usou funções sem perceber, como **print()** (para exibir algo na tela) e **input()** (para pegar dados do usuário), que já vêm prontas no Python.

---

1. Criando e Chamando Funções

Para criar sua própria função, usamos a palavra-chave **def**.

Sintaxe Básica:

Python

```
def nome_da_funcao():
    # Bloco de código da função
    print("Olá, eu sou uma função!")
```

Importante: O código dentro da função deve ter um recuo, chamado de **indentação**. Para "ativar" a função e fazê-la rodar, você deve **chamá-la** pelo nome, seguido de parênteses.

Exemplo:

Python

```
def boas_vindas():
    print("Seja bem-vindo(a)!")
    print("Estamos felizes em ter você aqui.")

# Chamando a função para executá-la
boas_vindas()
```

---

2. Parâmetros e Retorno

Uma função pode ser mais útil se você puder passar informações para ela. Essas informações são os **parâmetros**.

Exemplo Prático (sem return):

Python

```
def saudacao_personalizada(nome: str):
    print(f"Olá, {nome}!")

saudacao_personalizada("Maria")
```

A palavra-chave **return** permite que uma função devolva um valor para ser usado em outras partes do seu código.

Exemplo Prático (com return):

Python

```python
def somar_numeros(a: int, b: int) -> int:
    return a + b

total: int = somar_numeros(5, 3)
print(f"A soma é: {total}")
```

Você pode retornar mais de um valor, e o Python os agrupa em uma **tupla** automaticamente, que pode ser desempacotada para facilitar o uso.

Python

```python
def calcular_area_perimetro(base: float, altura: float) -> tuple[float, float]:
    area: float = base * altura
    perimetro: float = 2 * (base + altura)
    return area, perimetro

area, perimetro = calcular_area_perimetro(5.0, 10.0)
print(f"Área: {area}, Perímetro: {perimetro}")
```

---

3. O return em Condições e Loops

O return tem um superpoder: ele encerra a execução da função **imediatamente**, não importa onde ele esteja.

return dentro de um if: Escolhendo um Caminho

Ele é perfeito para decidir qual valor devolver com base em uma condição.

Python

```python
def verificar_idade(idade: int) -> str:
    if idade >= 18:
        return "Maior de idade"

    return "Menor de idade"

print(verificar_idade(25))
print(verificar_idade(16))
```

return dentro de um loop: Encontrando o que Precisamos

Ele pode ser usado para parar um loop assim que a condição desejada for encontrada.

Python

```python
def encontrar_primeiro_negativo(lista_de_numeros: list[int]) -> int | None:
    for numero in lista_de_numeros:
        if numero < 0:
            return numero

    return None

print(encontrar_primeiro_negativo([5, 8, -3, 10, -1]))
```

---

4. Argumentos Variáveis (*args e **kwargs)

E se você não souber quantos argumentos uma função vai receber?

- ***args**: Aceita um número variável de argumentos e os empacota em uma **tupla**.
- ****kwargs**: Aceita um número variável de argumentos nomeados e os empacota em um **dicionário**.

Exemplo:

Python

```
def exibir_pedido(item: str, *extras: str, **observacoes: str) -> None:
    print(f"Item Principal: {item}")
    print(f"Extras: {extras}")
    print(f"Observações: {observacoes}")

exibir_pedido("Pizza", "Queijo extra", "Bacon", borda="recheada")
```

---

5. Funções Anônimas (lambda)

Uma **função lambda** é uma pequena função anônima, de uma única linha. Elas são úteis para tarefas rápidas e simples.

Sintaxe: lambda argumentos: expressao

Exemplo:

Python

```
dobrar = lambda x: x * 2
print(dobrar(5))
```

Por que funções são essenciais? (Onde se aplica o DRY e o SRP)

Com o que você aprendeu até agora, é fácil ver o poder das funções. Elas seguem dois princípios fundamentais da programação:

- **Não se Repetir (Don't Repeat Yourself - DRY)**: Você escreve o código uma vez (na função) e o reutiliza em vários lugares. Se precisar corrigir um erro ou fazer uma melhoria, você só precisa alterar um único local.
- **Responsabilidade Única (Single Responsibility Principle - SRP)**: Cada função tem uma única responsabilidade. Por exemplo, a função calcular_area apenas calcula a área; ela não imprime o resultado ou solicita dados do usuário. Isso torna o código mais fácil de testar, entender e manter.

---

6. Tipagem: Deixando o Código mais Claro

Embora não seja obrigatório em Python, a **tipagem** ajuda a entender o que a função espera receber e o que ela vai devolver, prevenindo erros.

Python

```
# A função espera um número inteiro e uma string
def saudar_com_idade(nome: str, idade: int) -> str:
    return f"Olá, {nome}. Você tem {idade} anos."
```

---

7. Decoradores: Estendendo Funções

Um **decorador** é uma função que "embrulha" outra função para adicionar ou modificar seu comportamento. Isso promove a **separação de responsabilidades**: sua função principal faz uma coisa, e o decorador cuida de tarefas auxiliares como logs ou validações.

A sintaxe de um decorador usa o símbolo **@**.

Exemplo:

Python

```python
def log_chamada(func):
    def wrapper(*args, **kwargs):
        print(f"Chamando '{func.__name__}' com args: {args}, kwargs: {kwargs}")
        resultado = func(*args, **kwargs)
        print(f"'{func.__name__}' retornou: {resultado}")
        return resultado
    return wrapper

@log_chamada
def multiplicar(a: int, b: int) -> int:
    return a * b

multiplicar(5, 10)
```

Desafios Práticos

Desafio 1: Calculadora de Área e Perímetro

Crie uma função calcular_area_perimetro que recebe a base e altura e retorna ambos os valores.

Python

```python
def calcular_area_perimetro(base: float, altura: float) -> tuple[float, float]:
    area: float = base * altura
    perimetro: float = 2 * (base + altura)
    return area, perimetro

area, perimetro = calcular_area_perimetro(5.0, 10.0)

print(f"A área é: {area}")
print(f"O perímetro é: {perimetro}")
```

Desafio 2: Gerenciador de Clientes

Crie um menu interativo que usa funções para adicionar, remover e mostrar clientes de uma lista global.

Python

```python
clientes: list[str] = []

def adicionar_cliente() -> None:
    nome: str = input("Digite o nome do cliente: ")
    clientes.append(nome)
    print(f"Cliente '{nome}' adicionado com sucesso!")

def remover_cliente() -> None:
    try:
        if not clientes:
            print("A lista de clientes está vazia.")
            return
        indice: int = int(input("Digite o número do cliente para remover: "))
        if 0 <= indice < len(clientes):
            clientes.pop(indice)
            print("Cliente removido com sucesso!")
        else:
            print("Índice inválido.")
    except ValueError:
        print("Entrada inválida. Digite um número.")

def mostrar_clientes() -> None:
    if clientes:
        print("Lista de Clientes:")
        for cliente in clientes:
            print(f"- {cliente}")
    else:
        print("A lista de clientes está vazia.")

def menu() -> None:
```

```python
    while True:
        print("\n--- Menu ---")
        print("1. Adicionar cliente")
        print("2. Remover cliente")
        print("3. Mostrar clientes")
        print("4. Sair")
        escolha: str = input("Escolha uma opção: ")

        if escolha == "1":
            adicionar_cliente()
        elif escolha == "2":
            remover_cliente()
        elif escolha == "3":
            mostrar_clientes()
        elif escolha == "4":
            print("Saindo. Até mais!")
            break
        else:
            print("Opção inválida.")

menu()
```

Desafio 3: Gerenciador de Notas

Crie um programa que gerencia as notas de um aluno com funções para adicionar, calcular a média e verificar se o aluno foi aprovado (média >= 7).

Python

```python
notas: list[float] = []

def adicionar_nota() -> None:
    try:
        nota: float = float(input("Digite a nota: "))
        if 0 <= nota <= 10:
            notas.append(nota)
            print("Nota adicionada!")
```

```
        else:
            print("Nota deve ser entre 0 e 10.")
    except ValueError:
        print("Entrada inválida.")

def calcular_media() -> float:
    if not notas:
        return 0.0
    return sum(notas) / len(notas)

def verificar_aprovacao(media: float) -> str:
    if media >= 7:
        return "Aprovado"
    else:
        return "Reprovado"

def menu_notas() -> None:
    while True:
        print("\n--- Gerenciador de Notas ---")
        print("1. Adicionar nota")
        print("2. Calcular média")
        print("3. Verificar aprovação")
        print("4. Sair")
        escolha: str = input("Escolha uma opção: ")

        if escolha == "1":
            adicionar_nota()
        elif escolha == "2":
            media: float = calcular_media()
            print(f"Média: {media:.2f}")
        elif escolha == "3":
            media = calcular_media()
            status: str = verificar_aprovacao(media)
            print(f"Status: {status}")
        elif escolha == "4":
            break

menu_notas()
```

Desafio 4: Autenticador de Usuário com Decorador

Crie um decorador que valida um login e senha. Se a validação passar, o decorador deve executar a função principal. A função principal só será chamada se o usuário for autêntico.

Python

```python
USUARIOS_VALIDOS: dict[str, str] = {
    "admin": "senha123",
    "joao": "12345"
}

def autenticar(func):
    def wrapper(usuario: str, senha: str) -> str | None:
        if USUARIOS_VALIDOS.get(usuario) == senha:
            print(f"Usuário '{usuario}' autenticado com sucesso.")
            return func(usuario)
        else:
            print("Falha na autenticação. Usuário ou senha inválidos.")
            return None
    return wrapper

@autenticar
def exibir_painel(usuario: str) -> str:
    return f"Bem-vindo ao painel de controle, {usuario}!"

print(exibir_painel("admin", "senha123"))
print("---")
print(exibir_painel("joao", "senha_errada"))
```