




Funções em Python

Prof. Gustavo N. Friedrich

Funções em Python

- Uma **função** é um bloco de código que executa uma tarefa específica e pode ser reutilizado sempre que necessário. Elas ajudam a **organizar o código, evitar repetições e facilitar a manutenção.**

Estrutura de uma função em Python



```
def nome_da_funcao(parametros):  
    """Comentário opcional explicando o que a função faz"""  
    # código a ser executado  
    return resultado
```

- def → palavra-chave para definir uma função.
- nome_da_funcao → nome que identifica a função.
- parametros → valores que a função pode receber (opcional).
- return → valor que a função devolve (opcional).

Escopo de variáveis

- Escopo local: Variáveis criadas dentro da função só existem nela.
- Escopo global: Variáveis criadas fora da função podem ser acessadas por todo o programa.

```
def minha_funcao():  
    x = 10 # variável local  
    print("Dentro da função:", x)  
  
x = 5 # variável global  
minha_funcao()  
print("Fora da função:", x)
```

Função simples (sem parâmetros e sem retorno)



```
def saudacao():  
    print("Olá! Seja bem-vindo ao curso de Python.")  
  
# Chamando a função  
saudacao()
```

Função com parâmetros



```
def saudacao_personalizada(nome):  
    print(f"Olá, {nome}! Seja bem-vindo.")  
  
saudacao_personalizada("Maria")  
saudacao_personalizada("João")
```

Função com retorno



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

```
resultado = soma(5, 3)  
print("O resultado da soma é:", resultado)
```

Parâmetros com valor padrão



```
def saudacao_padrao(nome="Visitante"):  
    print(f"Olá, {nome}!")
```

```
saudacao_padrao( )           # Usa o valor padrão  
saudacao_padrao("Carla")    # Substitui o valor padrão
```


Funções com vários parâmetros



```
def exibir_dados(nome, idade, cidade):  
    print(f"{nome} tem {idade} anos e mora em {cidade}.")  
  
exibir_dados("Lucas", 25, "Porto Alegre")
```

Funções com múltiplos retornos



```
def calcular(val1, val2):  
    soma = val1 + val2  
    produto = val1 * val2  
    return soma, produto
```


```
s, p = calcular(4, 5)  
print("Soma:", s)  
print("Produto:", p)
```

Funções com n° variável de argumentos (*args)



```
def somar_todos(*numeros):  
    return sum(numeros)  
  
print(somar_todos(1, 2, 3))  
print(somar_todos(5, 10, 15, 20))
```

Funções com parâmetros nomeados dinâmicos (**kwargs)



```
def exibir_info(**dados):  
    for chave, valor in dados.items():  
        print(f"{chave}: {valor}")  
  
exibir_info(nome="Ana", idade=25, cidade="Curitiba")
```

Funções com parâmetros nomeados dinâmicos (**kwargs) – recebendo mais de uma lista

```
def exibir_info(**kwargs):  
    for chave, valor in kwargs.items():  
        print(f"{chave}: {valor}")
```

```
exibir_info(lista1=[1, 2, 3], lista2=["a", "b"], outra_lista=[True, False])
```

```
lista1: [1, 2, 3]
```

```
lista2: ['a', 'b']
```

```
outra_lista: [True, False]
```

Funções anônimas (lambda)




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



```
dobro = lambda x: x * 2  
  
print(dobro(5))      # 10  
print(dobro(7))      # 14  
print(dobro(10))     # 20
```

Funções anônimas (lambda)

- Aqui, a lambda funciona como um “atalho” para chamar a função saudacao.



```
def saudacao(nome):  
    return f"Olá, {nome}!"  
  
# Lambda chamando uma função def  
cumprimentar = lambda nome: saudacao(nome)  
  
print(cumprimentar("Maria"))
```

Funções anônimas (lambda)

- Lambdas devem ter apenas uma expressão (não podem conter múltiplas linhas ou comandos complexos como if/for tradicionais, só expressões inline).
- Se a lógica começar a ficar grande, é melhor usar def direto para manter a legibilidade.