PYTHON ACADEMY

APRENDA A CRIAR FUNÇÕES EM PYTHON

Guia de funções: def, parâmetros, return, *args/**kwargs, funções puras, casos práticos (validação, cálculo, formatadores), DRY principle, funções vs classes.

PYTHONACADEMY.COM.BR

Este ebook foi gerado por

DevBook

Crie Ebooks técnicos incríveis em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!

Chega de formatar código no Google Docs

Deixe que nossa IA faça o trabalho pesado

- Syntax Highlight
- Adicione Banners Promocionais
- Edite em Markdown em Tempo Real
- Infográficos feitos por IA

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS

> ✅ ***Atualizado para Python 3.13*** *(Dezembro 2025)*
> *Funções com type hints, casos práticos e DRY principle.*

Salve salve Pythonista!

Uma **função** nada mais é que um trecho de código que possui uma responsabilidade específica e que damos um nome à ele.

Funções evitam repetição (DRY - Don't Repeat Yourself) e organizam o código!

Neste guia: - ✅ **def** - Definir funções - ✅ **Parâmetros** - Argumentos posicionais e nomeados - ✅ **return** - Retornar valores - ✅ **Casos práticos** - Validação, cálculo, formatadores

Nesse post você vai aprender: - Como criar funções com a palavra reservada `def`, - Como definir seus parâmetros, - O que são `args` e o que são `kwargs`, - Como retornar dados de funções, - Como definir funções com uma linha e **muito mais**!

Então vem com a gente!

Introdução

Funções são blocos de código que executam funcionalidades específicas.

Normalmente são utilizados para evitar que determinada parte do seu código sejá escrito varias vezes.

Em Python sua sintaxe é definida usando `def` e atribuindo um nome a ela, veja um exemplo:

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

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

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()
```

Resultado do código acima:

```
Bloco de código
```

Parâmetros

Além de executar código, funções também podem **receber** e **retornar** dados.

Podemos enviar dados para uma função através de seus parâmetros.

Observe o exemplo:

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

imprime_nome("Erickson")
imprime_nome("Renan")
imprime_nome("Daniel")
```

Resultado do código acima:

```
Nome: Erickson
Nome: Renan
Nome: Daniel
```

> *Não entendeu essa notação do `print(f"Nome: {nome}")` ? Isso se chama F-strings e é uma maneira de formatar código Python e Strings! Quer saber mais sobre esse assunto, então acesse nosso Post completo sobre F-Strings!*

Quando a função é chamada, passamos uma string como dado de entrada - através do parâmetro `nome` - que é concatenada e impressa dentro da função.

Caso nenhum valor seja infomado ao chamar a função, um erro será gerado. Por exemplo, o seguinte código:

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

imprime_nome()
```

Ocasionará o seguinte erro:

```
TypeError: imprime_nome() missing 1 required positional argument:
'nome'
```

Podemos resolver esse erro utilizando os “Valores Padrão” e é exatamente isso que veremos agora!

Valores Padrão (ou Valores *Default*)

A utilização dos valores padrão serve para dar um valor quando quem chamou a função não passar nenhum valor para os parâmetros definidos.

Fazemos isso dessa forma:

```
def flor(flor='Rosa', cor='Vermelha'):
    print("A cor da {flor} é {cor}")

flor()
flor("Orquídea", "Azul")
```

Veja o resultado:

```
A cor da Rosa é Vermelha
A cor da Orquídea é Azul
```

Ou seja, o erro anterior não ocorrou novamente!

Chamada de Função Posicional *versus* Chamada de Função Nomeada

Quando chamamos uma função, podemos utilizar a localização dos parâmetros para fazer o casamento entre o que foi chamado e o que foi definido na função.

Para entender melhor, veja o exemplo a seguir:

```
def monta_computador(cpu='', armazenamento=0, memoria=0):
    print('A configuração é: \n\t- CPU: {cpu}\n\t- Armazenamento: {ar-
        mazenamento}Tb\n\t- Memória: {memoria}Gb')

monta_computador('Intel Core i9', 4, 64)
```

A saída será:

```
A configuração é:
    - CPU: Intel Core i9
    - Armazenamento: 4Tb
    - Memória: 64Gb
```

O programador que escreveu a chamada da função `monta_computador` está respeitando a **posição** dos parâmetros, ou seja:

- O valor `"Intel Core i9"` é referente ao **primeiro** parâmetro (`cpu`)
- O valor `4` é referente ao **segundo** parâmetro (`armazenamento`)
- O valor `64` se refere ao **terceiro** parâmetro (`memoria`)

Essa é uma chamada de função **posicional**, ou seja: que respeita a ordem dos parâmetros.

Outra forma de fazer essa chamada de função é utilizar os **nomes** dos parâmetros!

Dessa forma, não é necessário respeitar a ordem de definição dos parâmetros!

Veja o mesmo exemplo, mas agora utilizando os **nomes** dos parâmetros:

```
monta_computador(memoria=64, armazenamento=4, cpu='Intel Core i9')
```

A saída será a mesma, pois como utilizamos os nomes, o Python saberá qual valor referencia qual parâmetro!

> 💡 *Estou desenvolvendo o **DevBook**, uma plataforma que usa IA para gerar ebooks técnicos profissionais. Te convido a conhecer!*

Parâmetro `*args`

Caso você queira desenvolver uma função que recebe um número variável de parâmetros, você pode utilizar o parâmetro `*args`!

Dessa forma, a função receberá os argumentos em forma de Tupla e você poderá processá-los com um loop `for` por exemplo!

Veja o código abaixo para entender melhor:

```
def maior_30(*args):
    print(args)
    print(type(args))

    for num in args:
        if num > 30:
            print(num)

maior_30(10, 20, 30, 40, 50, 60)
```

A função acima irá receber todos os valores passados para função no parâmetro `*args` e irá iterar sobre eles com um loop `for`.

Veja a saída:

```
(10, 20, 30, 40, 50, 60)
<class 'tuple'>
40
50
60
```

> ***Observação**: O nome `*args` é uma convenção, ou seja uma boa prática entre programadores Python! Contudo, nada te impede de alterar esse nome para `*numeros` por exemplo. Dessa forma, a definição da função seria: `def maior_30(*numeros):`*

Parâmetro `**kwargs`

Agora, se quiser desenvolver uma função com número variado de parâmetros **nomeados**, utilize `**kwargs`.

Dessa forma, todos os dados passados à função serão guardados nessa variável `**kwargs`, em formato de um dicionário.

Oberve como podemos obter a chave a valor deles percorrendo os itens deste dicionário:

```
def dados_pessoa(**kwargs):
    print(type(kwargs))

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

dados_pessoa(nome='João', idade=35, carreira='Desenvolvedor Fullstack')
```

Na saída podemos observar o tipo de dado que a função recebeu e a estrutura construida para percorrer o dicionário:

```
<class 'dict'>
nome: João
idade: 35
carreira: Desenvolvedor Fullstack
```

> ***Observação***: *O nome `**kwargs` é uma convenção, ou seja uma boa prática entre programadores Python! Contudo, nada te impede de alterar esse nome para `**pessoa` por exemplo.*

Funções com retorno de dados

As funções também podem retornar valores através da palavra reservada `return`.

Veja o exemplo:

```
def soma_dois_numeros(valor1, valor2):
    soma = valor1 + valor2
    return soma

valor_soma = soma_dois_numeros(32, 15)
print(valor_soma)
print(soma_dois_numeros(50, 10))
```

Saída com o retorno da soma dos valores introduzidos na função:

```
47
60
```

Funções com retorno múltiplos

Funções também podem retornar múltiplos dados. Veja o exemplo:

```
def soma_dois_numeros_e_calcula_media(valor1, valor2):
    soma = valor1 + valor2
    media = (valor1 + valor2)/2

    return soma, media

valor_soma = soma_dois_numeros_e_calcula_media(32, 15)
print(valor_soma)
print(soma_dois_numeros_e_calcula_media(50, 10))
```

A saída será:

```
(47, 23.5)
(60, 30.0)
```

Palavra reservada `pass`

Caso você deseje definir uma função sem corpo nenhum, ou seja, sem código, saiba que isso irá disparar o erro `IndentationError`, pois funções não podem estar vazias.

Porém se por algum motivo precisar use a palavra reservada `pass`, da seguinte forma:

```
def funcao():
    pass
```

Função de uma linha

Python possibilita a criação de funções com apenas uma linha de código. Veja os exemplo a seguir:

```
# Definição das funções
def soma(valor1, valor2): return valor1 + valor2
def divisao(valor1, valor2): return valor1 / valor2
def multiplicacao(valor1, valor2): return valor1 * valor2

# Chamada das funções
print(soma(1, 5))
print(divisao(8, 2))
print(multiplicacao(8, 2))
```

O resultado do código acima será:

```
6
4.0
16
```

Por hoje é só pessoal!

Espero que tenham curtido este conteúdo 😉

Casos Práticos

1. Validação de Dados

```python
def validar_email(email):
    return '@' in email and '.' in email

def validar_cpf(cpf):
    return len(cpf) == 11 and cpf.isdigit()

print(validar_email("teste@example.com"))  # True
print(validar_cpf("12345678901"))  # True
```

2. Cálculos Reutilizáveis

```python
def calcular_imc(peso, altura):
    imc = peso / (altura ** 2)
    return round(imc, 2)

def calcular_desconto(preco, percentual):
    desconto = preco * (percentual / 100)
    return preco - desconto

print(calcular_imc(70, 1.75))  # 22.86
print(calcular_desconto(100, 10))  # 90.0
```

EBOOK GERADO POR DevBook
ACESSE DEVBOOK.AI ✨ PRIMEIRO CAPÍTULO 100% GRÁTIS

3. Formatadores

```python
def formatar_cpf(cpf):
    return f"{cpf[:3]}.{cpf[3:6]}.{cpf[6:9]}-{cpf[9:]}"

def formatar_telefone(tel):
    return f"({tel[:2]}) {tel[2:7]}-{tel[7:]}"

print(formatar_cpf("12345678901"))  # 123.456.789-01
print(formatar_telefone("11987654321"))  # (11) 98765-4321
```

DRY Principle

DRY = Don't Repeat Yourself (Não se repita)

```python
# ❌ Sem função (repetição!)
preco1 = 100
preco1_com_desconto = preco1 * 0.9

preco2 = 200
preco2_com_desconto = preco2 * 0.9

# ✅ Com função (DRY!)
def aplicar_desconto(preco, percentual=10):
    return preco * (1 - percentual/100)

preco1_com_desconto = aplicar_desconto(100)
preco2_com_desconto = aplicar_desconto(200)
```

Conclusão

Neste guia de **Funções**, você aprendeu:

EBOOK GERADO POR DevBook
ACESSE DEVBOOK.AI ✨ PRIMEIRO CAPÍTULO 100% GRÁTIS

✅ **def** - Criar funções
✅ **Parâmetros** - Argumentos posicionais e nomeados
✅ **return** - Retornar valores
✅ **DRY** - Não repetir código
✅ **Casos práticos** - Validação, cálculo, formatadores

Principais lições: - Funções **evitam repetição** - Use nomes **descritivos** - Funções devem ter **uma responsabilidade** - **return** encerra a função - Parâmetros padrão são **opcionais**

Próximos passos: - Aprenda *args/**kwargs - Explore Decorators - Pratique funções puras - Estude type hints (Python 3.5+)

Nesse post vimos como podemos criar Funções em Python, que nada mais é que "pedaço de código nomeado".

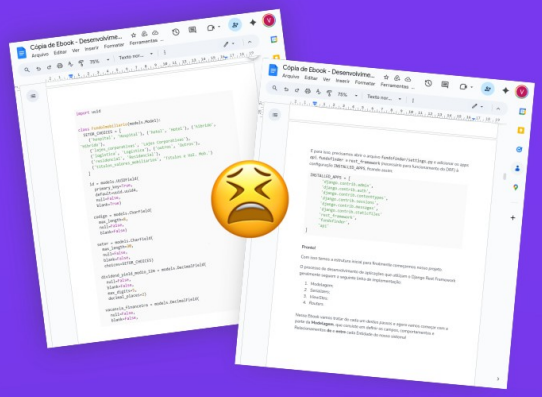
Se ficou com alguma dúvida, fique à vontade para deixar um comentário no box aqui embaixo! Será um prazer te responder! 😉

Não se esqueça de conferir!

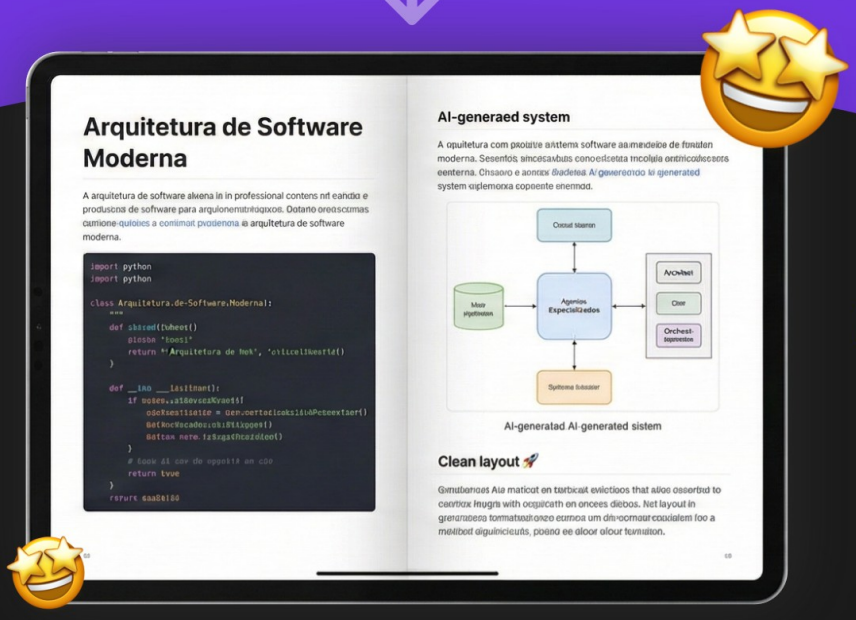
DevBook

Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!



Chega de formatar código no Google Docs



Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS