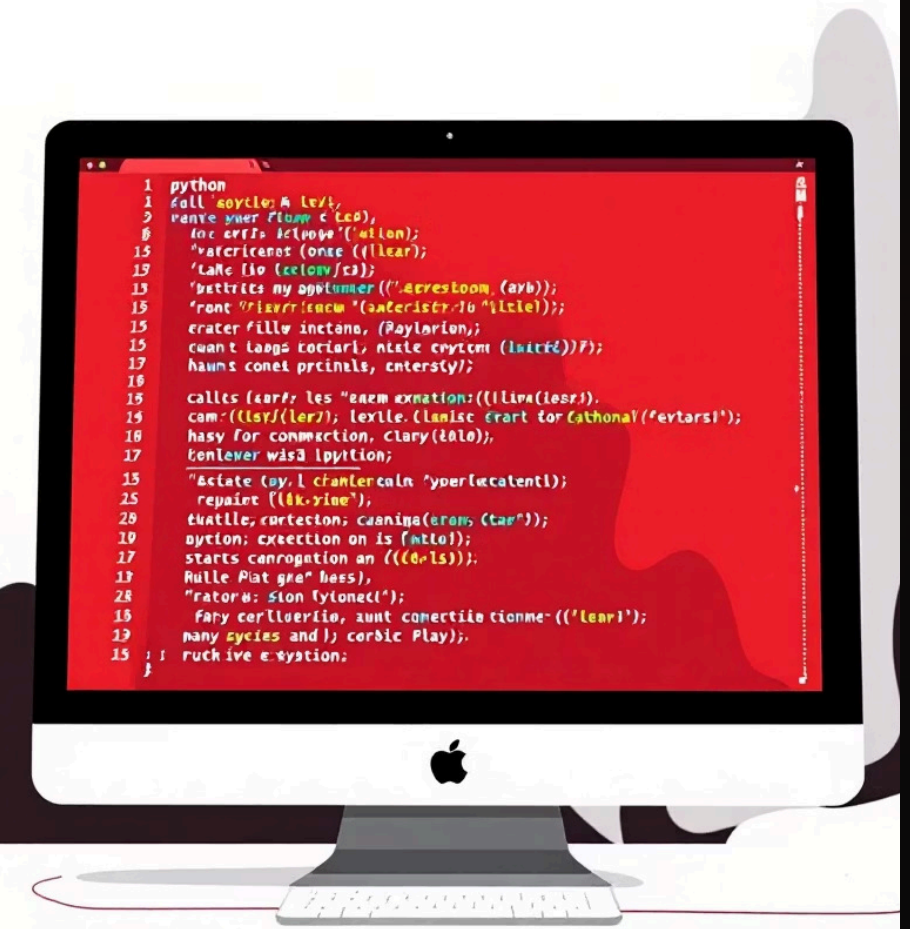


Aula 6 – *args e **kwargs

Dominando argumentos flexíveis em Python para criar funções mais poderosas e reutilizáveis

Objetivos da Aula

Compreender Argumentos Especiais

Entender como funcionam os argumentos especiais em funções Python e sua importância na programação flexível.

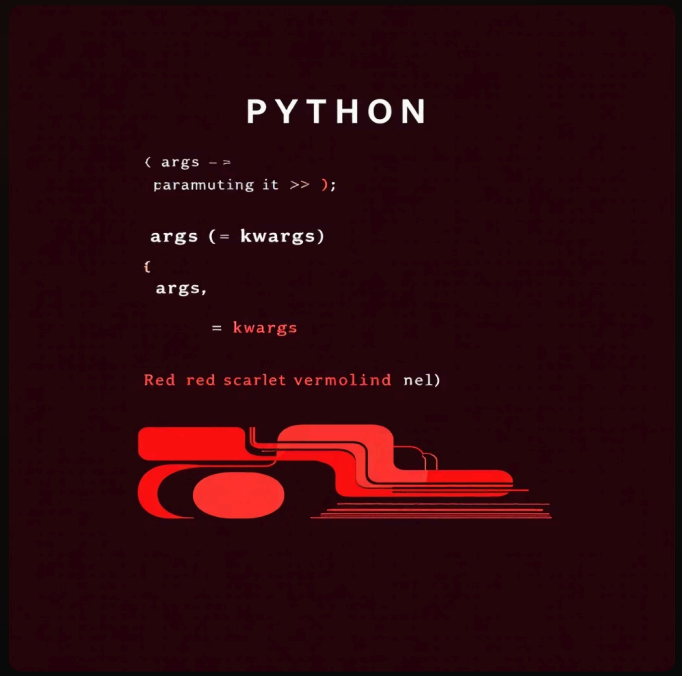
Dominar *args e **kwargs

Aprender a utilizar *args e **kwargs para criar funções mais versáteis e adaptáveis a diferentes cenários.

Desenvolver Funções Flexíveis

Criar funções que aceitam quantidades variáveis de argumentos, tornando seu código mais robusto e reutilizável.

O que são *args e **kwargs?



*args

Recebe **múltiplos argumentos posicionais** em forma de **tupla**, permitindo que sua função aceite qualquer quantidade de valores.

**kwargs

Recebe **múltiplos argumentos nomeados** em forma de **dicionário**, oferecendo flexibilidade com parâmetros opcionais.

Estes recursos permitem criar funções mais **genéricas** e **reutilizáveis**, fundamentais para um código Python eficiente.

Por que e quando usar?

Quantidade Desconhecida

Quando não sabemos quantos valores serão passados para a função, oferecendo máxima flexibilidade na entrada de dados.

Parâmetros Opcionais

Para trabalhar com parâmetros opcionais que podem ou não ser fornecidos pelo usuário da função.

Funções Matemáticas

Ideais para operações matemáticas com múltiplos números, como somas, médias e cálculos estatísticos.

Também utilizados em **configurações de sistemas** e **manipulação de dados variados**.

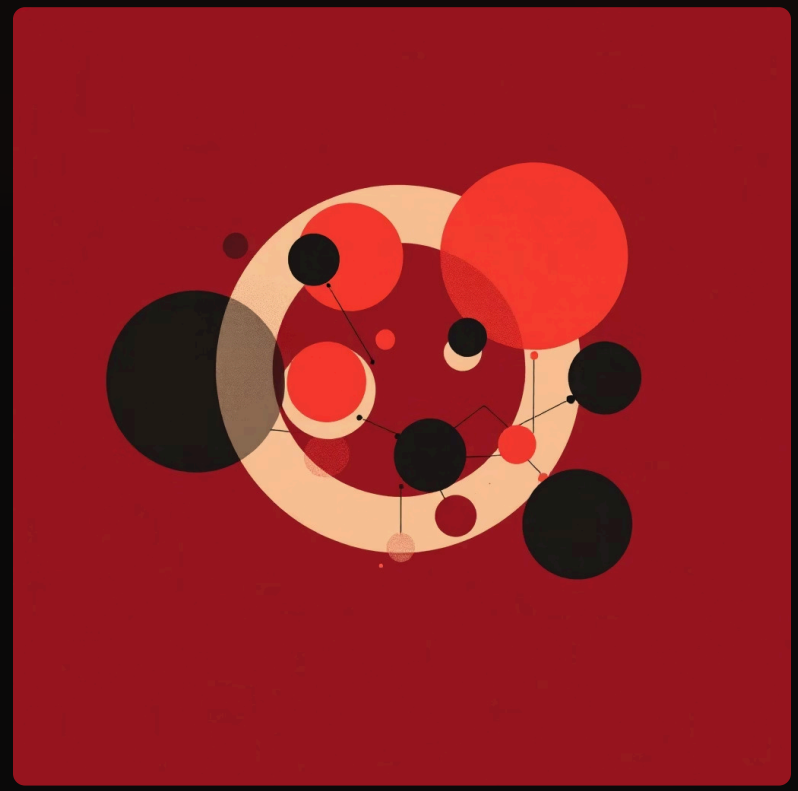
*args em Ação

Exemplo Prático

```
def soma(*numeros):
    return sum(numeros)

print(soma(2, 4, 6))  # 12
print(soma(1, 3, 5, 7, 9))  # 25
print(soma(10))  # 10
```

- Coleta argumentos em **tupla**
- Facilita **iteração**
- Manipulação **livre** dos dados



ⓘ O ***args** coleta todos os argumentos em uma tupla, permitindo operações flexíveis com os valores recebidos.

**kwargs em Ação

Exemplo Prático

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

perfil_usuario(
    nome="Ana",
    idade=25,
    cidade="Curitiba"
)
```

Saída:

```
nome: Ana
idade: 25
cidade: Curitiba
```

**kwargs coleta os argumentos em um dicionário, organizando informações por chaves e valores de forma estruturada.

Combinando *args e **kwargs

01

Argumentos Fixos

Parâmetros obrigatórios que sempre devem ser fornecidos

02

*args

Argumentos posicionais variáveis coletados em tupla

03

**kwargs

Argumentos nomeados variáveis coletados em dicionário

```
def exemplo(arg_fixo, *args, **kwargs):
    print("Fixo:", arg_fixo)
    print("Args:", args)
    print("Kwargs:", kwargs)

exemplo("obrigatório", 1, 2, 3, nome="João", idade=30)
```

⚠ **Ordem correta:** def funcao(fixos, *args, **kwargs)

Boas Práticas

Nomes Descritivos

Use nomes claros e descritivos nos parâmetros fixos para melhor legibilidade do código.

*args para Valores Sem Nome

Utilize *args quando precisar de muitos valores posicionais sem identificadores específicos.

**kwargs para Flexibilidade

Use **kwargs para valores nomeados e opcionais que oferecem configuração flexível.

Documentação Clara

Sempre documente suas funções indicando como devem ser utilizadas e quais parâmetros aceita.

Exercício Prático

Desafio: cadastro_aluno

Crie uma função que demonstre o poder de *args e **kwargs

1

Argumento Fixo

Receba o **nome do aluno** como parâmetro obrigatório

2

*args para Notas

Colete as **notas do aluno** usando argumentos posicionais

3

**kwargs para Extras

Capture **informações adicionais** como curso, idade, etc.

A função deve:

1. Exibir o nome do aluno
2. Calcular e mostrar a média das notas
3. Apresentar as informações extras organizadamente