PYTHON ACADEMY

ARGS E KWARGS NO PYTHON

Guia completo de *args e **kwargs: argumentos variáveis, desempacotamento, casos práticos (logger, decorators, wrappers), ordem de parâmetros, *args vs **kwargs.

PYTHONACADEMY.COM.BR

Crie ebooks profissionais incríveis em minutos com IA

Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

E deixe que nossa IA faça o trabalho pesado!

- Capas gerados por IA
- Infográficos feitos por IA
- Edite em Markdown em Tempo Real
- Adicione Banners Promocionais

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS

> ✅ ***Atualizado para Python 3.13*** *(Dezembro 2025) Conteúdo enriquecido com casos práticos (decorators, wrappers) e ordem de parâmetros.*

Salve salve querido Pythonista!

`*args` **e** `**kwargs` permitem número **variável de argumentos** em funções! Essenciais para decorators, wrappers e APIs flexíveis.

Neste guia, você vai aprender:

- ✅ `*args` - Argumentos posicionais variáveis
- ✅ `**kwargs` - Argumentos nomeados variáveis
- ✅ **Ordem de parâmetros** - positional, *args, keyword, **kwargs
- ✅ **Casos práticos** - Decorators, loggers, wrappers

Nesse post veremos dois conceitos muito importantes, o conceito de `*args` (ou argumentos não nomeados) e `**kwargs` (ou argumentos nomeados)!

Ambos são geralmente utilizados no contexto da criação de **Funções** em Python.

Se ainda não domina Funções, clique aqui e veja nosso post completo sobre Funções no Python

Agora vamos desvendar o que essas duas palavrinhas querem dizer em Python! 😉

`*args` e `**kwargs`

*Args e **kwargs permitem que você passe um número não especificado de argumentos para uma função.

Dessa forma, ao escrever uma função, você não precisa definir quantos argumentos serão passados para sua função.

Observação importante: Escrever `*args` e `**kwargs` é apenas uma convenção: `*args` vem do inglês *"arguments"* (argumntos) e `**kwargs` vem do inglês *"keyword arguments"* (argumentos nomeados). Podemos, por exemplo, utilizar `*banana` e `**bananas` se quisermos 😆

Argumentos não nomeados `*args`

Vamos dar uma olhada em `*args` primeiro!

O `*args` é usado para receber uma lista de argumentos de comprimento variável sem a palavra-chave (*keyword*) na função.

Aqui está um exemplo pra facilitar o entendimento:

```
def função_com_argumentos_variaveis(arg, *args):
    print("Primeiro argumento:", arg)

    for agumento in args:
        print("Argumento de *args", argumento)

função_com_argumentos_variaveis('primeiro_arg', 'arg2', 'arg3', 'arg3')
```

Veja a saída:

```
Primeiro argumento: primeiro_arg
Argumento de *args: arg2
Argumento de *args: arg3
Argumento de *args: arg3
```

Fique atento a ordem!

Argumentos não nomeados (`*args`) vem sempre primeiro que os argumentos nomeados (`**kwargs`).

Veja a seguinte chamada de função:

```
funcao_com_argumentos_variaveis(arg='arg', 'arg2', 'arg3', 'arg3')
```

O seguinte erro será lançado:

```
File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
```

Agora vamos a um exemplo mais útil!

Uma função que soma os argumentos passados, independente do número de argumentos que seja passado, veja:

```python
def adicao(*args):
    resultado = 0

    for argumento in args:
        resultado += argumento

    return resultado

print(adicao(1, 2))
print(adição(1, 2, 4, 6))
print(adição(1, 2, 4, 6, 8, 10))
```

Resultando em:

```
3
13
31
```

> 💡 *Estou construindo o **Ebookr.ai**, uma plataforma onde você cria ebooks profissionais com IA sobre qualquer assunto — do zero ao PDF pronto, com capas e infográficos gerados automaticamente. Dá uma olhada!*

Argumentos nomeados `**kwargs`

O `**kwargs` possibilita verificarmos os parâmetros nomeados da função, isto é, aqueles parâmetros que são passados com um nome!

Eles estarão disponíveis como um dicionário (`{'chave': 'valor'}`) dentro da função.

Vamos ao exemplo:

```python
def concatena (**kwargs):
    print(f'Valores recebidos: {kwargs}')
    resultado = ""

    for valor in kwargs.values():
        resultado += f'{valor} '
    return resultado

print(concatena(a="Python", b="Academy",  c="Rules!"))
print()
print(concatena(a="Python", b="é", c="na", d="Python", e="Academy!"))
```

Veja a saída, perceba que recebemos um `dict` na variável `kwargs`:

```
Valores recebidos: {'a': 'Python', 'b': 'Academy', 'c': 'Rules!'}
Python Academy Rules!

Valores recebidos: {'a': 'Python', 'b': 'é', 'c': 'na', 'd': 'Python',
'e': 'Academy!'}
Python é na Python Academy!
```

Para melhor entender vamos juntar tudo e printar o que vem em cada um dos tipos de parâmetros.

Veja o exemplo:

```python
def funcao(arg_1, arg_2, *args, arg_kw, **kwargs):
    print(f'Parâmetro 1: {arg_1}')
    print(f'Parâmetro 2: {arg_2}')
    print(f'*args: {args}')
    print(f'Argumento nomeado: {arg_kw}')
    print(f'**kwargs: {kwargs}')

# Chamada da Função
funcao(
    1,          # Parâmetro 1
    'a',        # Parâmetro 2
    'arg1',     # *args
    'arg2',     # *args
    2,          # *args
    arg_kw='KW',# arg_kw
    banana='B', # **kwargs
    bananas='A' # **kwargs
)
```

A saída deste código será:

```
Parâmetro 1: 1
Parâmetro 2: a
*args: ('arg1', 'arg2', 2)
Argumento nomeado: KW
**kwargs: {'banana': 'B', 'bananas': 'A'}
```

Casos Práticos

1. Logger Flexível

```python
def log(level, message, **metadata):
    print(f"[{level}] {message}")
    if metadata:
        print(f"Metadata: {metadata}")

log("INFO", "Usuário logou")
log("ERROR", "Falha no banco", user_id=123, ip="192.168.1.1")
log("DEBUG", "Query executada", query="SELECT *", duration=0.5)
```

2. Decorator Genérico

```python
import time

def timer(func):
    def wrapper(*args, **kwargs):  # Aceita qualquer argumento!
        start = time.time()
        result = func(*args, **kwargs)  # Passa adiante
        duration = time.time() - start
        print(f"{func.__name__} levou {duration:.2f}s")
        return result
    return wrapper

@timer
def processar(n, verbose=False):
    time.sleep(n)
    if verbose:
        print(f"Processado {n}s")

processar(1)
processar(2, verbose=True)
```

3. Criar Objeto Dinâmico

```python
class Config:
    def __init__(self, **settings):
        for key, value in settings.items():
            setattr(self, key, value)

    def __repr__(self):
        attrs = ', '.join(f"{k}={v}" for k, v in self.__dict__.items())
        return f"Config({attrs})"

config = Config(host="localhost", port=8000, debug=True)
print(config.host)  # localhost
print(config)  # Config(host=localhost, port=8000, debug=True)
```

4. Combinar Dicionários

```python
def merge_configs(*configs, override=True):
    result = {}
    for config in configs:
        if override:
            result.update(config)  # Último sobrescreve
        else:
            for k, v in config.items():
                result.setdefault(k, v)  # Primeiro vence
    return result

default = {'host': 'localhost', 'port': 8000}
user = {'port': 9000, 'debug': True}

final = merge_configs(default, user)
print(final)  # {'host': 'localhost', 'port': 9000, 'debug': True}
```

Ordem de Parâmetros

```python
def funcao(a, b, *args, key1, key2="default", **kwargs):
    print(f"Posicionais obrigatórios: {a}, {b}")
    print(f"Args extras: {args}")
    print(f"Keyword obrigatório: {key1}")
    print(f"Keyword com padrão: {key2}")
    print(f"Kwargs extras: {kwargs}")

funcao(1, 2, 3, 4, key1="val1", key3="val3", key4="val4")
# Posicionais obrigatórios: 1, 2
# Args extras: (3, 4)
# Keyword obrigatório: val1
# Keyword com padrão: default
# Kwargs extras: {'key3': 'val3', 'key4': 'val4'}
```

Ordem SEMPRE:

1. Posicionais obrigatórios: `a, b`
2. `*args` (variáveis posicionais)
3. Keywords obrigatórios: `key1`
4. Keywords com padrão: `key2="default"`
5. `**kwargs` (variáveis nomeados)

Conclusão

Neste guia de **`*args`** **e** **`**kwargs`**, você aprendeu:

✅ `*args` - Tupla de argumentos posicionais variáveis ✅ `**kwargs` - Dict de argumentos nomeados variáveis ✅ **Desempacotamento** - `func(*lista)`, `func(**dict)` ✅ **Ordem de parâmetros** - Posicionais → *args → Keywords → **kwargs** ✅ Casos práticos** - Decorators, loggers, wrappers

Principais lições:

- `*args` captura **argumentos posicionais extras**
- `**kwargs` captura **argumentos nomeados extras**
- Essenciais para **decorators genéricos**
- Use para **APIs flexíveis**
- Respeite a **ordem de parâmetros**!

Próximos passos:

- Pratique Decorators
- Aprenda Funções avançadas
- Explore type hints com `*args: int` e `**kwargs: str`
- Estude `*` (keyword-only) e `/` (positional-only)

Agora você entende de onde vem esse tal de `*args` e `**kwargs`!

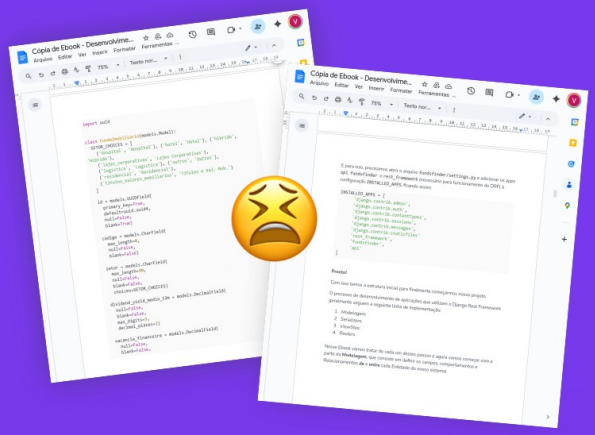
Se tiver qualquer dúvida, manda aqui embaixo no box de comentários 😉

Até a próxima!

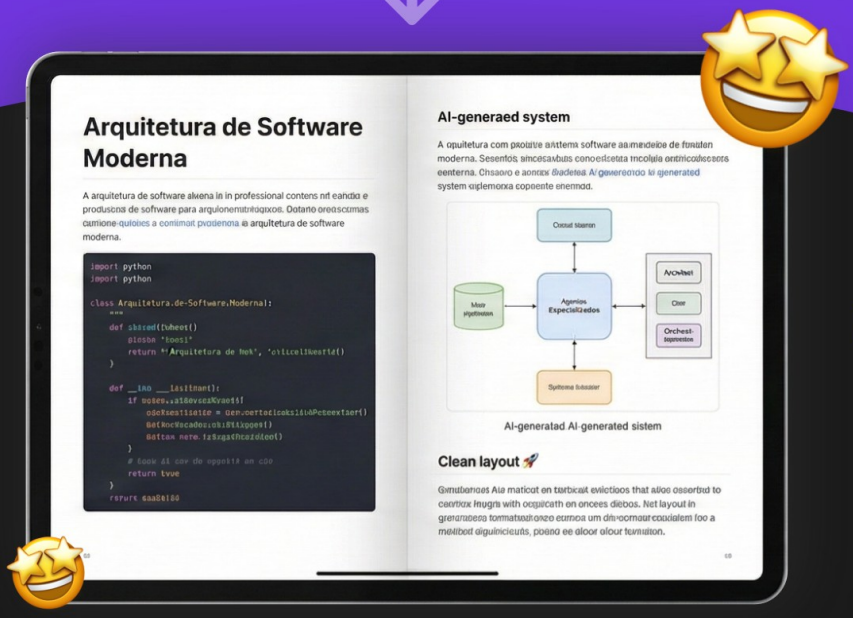
Não se esqueça de conferir!

Ebookr

Crie Ebooks profissionais em minutos com IA



Chega de formatar código no Google Docs ou Word



Capas gerados por IA

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