

DICIONÁRIOS (DICTS) NO PYTHON

Guia completo de Dicionários (dicts) em Python: criação, métodos (get, update, pop), iteração, casos reais (cache, contadores, config), dict vs defaultdict vs Counter, O(1) lookups.

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 

 **Atualizado para Python 3.13** (Dezembro 2025)

Conteúdo enriquecido com casos de uso reais, comparação com defaultdict/Counter e análise de performance O(1).

Opa Dev!

Dicionários são a estrutura chave-valor mais importante do Python! Com **lookups O(1)**, são perfeitos para mapeamentos, cache, contadores e configurações.

Neste guia completo, você vai aprender: -  Criação e operações básicas -  Métodos essenciais (get, update, pop, keys, values) -  **Casos de uso reais** (cache, contadores, config) -  **dict vs defaultdict vs Counter** - quando usar

Garanto que você vai sair daqui um craque em Dicionários, combinado? Então já deixa o café pronto e **BORA!**

Introdução

Os dicionários são coleções de itens e seus elementos são armazenados de forma não ordenada.

Seus elementos contém uma chave e valor, isto é:

- Uma **chave** que vai servir para indexar (posicionar) determinado elemento no dicionário.
- Um **valor** que contém... Bem, um valor 😊 Este valor aceita diversos tipos: listas, outros dicionários, inteiros, strings e etc.

Sua sintaxe básica é: `{'chave': 'valor'}`. Utiliza-se `{}` para delimitar o dicionário e a chave é separada do valor por dois pontos `:`.

Exemplo de sua sintaxe:

```
dicio = {'chave': 'valor'}  
  
print(type(dicio))
```

Quando utilizar `type()` e a saída for essa abaixo, pode ter certeza que é um dicionário!

```
<class 'dict'>
```

Criando dicionários

Agora vamos ver 6 maneiras de criar um mesmo dicionário!

O modo mais simples de criar um dicionário:

```
dicio_2 = {'primeiro': 1, 'segundo': 2, 'terceiro': 3}
```

Também podemos utilizar a função `dict` do próprio Python (*built-in function*), passando as chaves e valores como parâmetros:

```
dicio = dict(primeiro=1, segundo=2, terceiro=3)
```

Utilizando a função `zip` para concatenar a `chave:valor` em um elemento do objeto `dict`:

```
dicio_3 = dict(zip(['primeiro', 'segundo', 'terceiro'], [1, 2, 3]))
```

Utilizando uma lista de tuplas com itens simbolizando `chave` e `valor` em um objeto `dict`:

```
dicio_4 = dict([('primeiro', 1), ('segundo', 2), ('terceiro', 3)])
```

Com *Dict Comprehensions!*

Não sabe o que é? Nós temos um [Post completíssimo sobre Dict Comprehensions!][dict-comprehensions]

Veja como:

```
dicio_6 = {name: idx + 1 for idx, name in enumerate(('primeiro', 'segundo', 'terceiro'))}
```

Por mais estranho que pareça, podemos tentar transformar uma variável do tipo dicionário em dicionário.

```
dicio_5 = dict({'primeiro': 1, 'segundo': 2, 'terceiro': 3})
```

Todos esses exemplos tem o mesmo resultado, que esta sendo mostrado abaixo:

```
{'primeiro': 1, 'segundo': 2, 'terceiro': 3}
```

Acessando itens

Para acessar o valor de um dicionário podemos utilizar sua chave. Exemplo:

```
pessoa = {'nome': 'Pythonico', 'altura': 1.65, 'idade': 21}  
print(pessoa['nome'])
```

Saída com o valor da chave especificada:

```
Pythonico
```

Dicionários também contam com um método chamado `get()` que fornecerá o mesmo resultado:

```
print(pessoa.get('nome'))
```

A saída será a mesma, o resultado: `Pythonico`

Uma coisa interessante sobre esse método é que você pode definir um valor *default*, para o caso da chave não ser encontrada, exemplo:

```
print(pessoa.get('peso', 'Chave não encontrada'))
```

Como não existe nenhuma chave chamada `'peso'`, ele retorna:

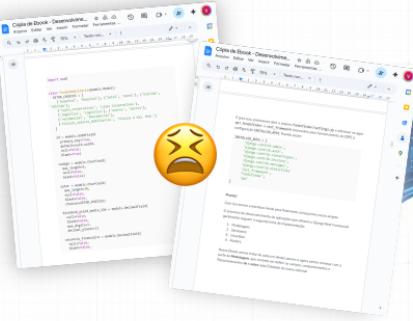
```
Chave não encontrada
```

 Estou desenvolvendo o **DevBook**, uma plataforma que usa IA para gerar ebooks técnicos profissionais. Depois de ler, dá uma passada no site!

 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 

Obtendo chaves e valores

Se você deseja obter todas as chaves de um dicionário, use o método `keys()`:

```
computador = {'CPU': 'Intel', 'RAM': '8gb', 'SSD': '250bg'}  
  
print(computador.keys())
```

A saída é um objeto do tipo `dict_keys` que equivale a uma lista com todas as chaves:

```
dict_keys(['CPU', 'RAM', 'SSD'])
```

Para obter apenas os valores das chaves em seu dicionário:

```
notas = {'Mat': 5, 'Por': 7, 'His': 8}

print(notas.values())
```

A saída será uma lista com os valores de cada chave:

```
dict_values([5, 7, 8])
```

Percorrendo os elementos de um Dicionário

Podemos percorrer os elementos de um Dicionário a partir de suas chaves `dict.keys()` ou a partir de seus valores `dict.values()`.

Percorrendo elementos com a função `dict.keys()`:

```
computador = {'CPU': 'Intel', 'RAM': '8gb', 'SSD': '250bg'}

for chave in computador.keys():
    print(f'Chave = {chave} e Valor = {computador[chave]}')
```

A saída será:

```
Chave = CPU e Valor = Intel
Chave = RAM e Valor = 8gb
Chave = SSD e Valor = 250bg
```

Percorrendo elementos com a função `dict.values()` :

```
notas = {'Mat': 5, 'Por': 7, 'His': 8}

for valor in notas.values():
    print(f'Valor: {valor}')
```

A saída será:

```
Valor: 5
Valor: 7
Valor: 8
```

Percorrendo as chaves e valores de um Dicionário

Uma maneira de obter chaves e valores é utilizando o método `dict.items()`.

Veja como:

```
frutas = {'pera': 10, 'uva': 2, 'maça': 55}

print(frutas.items())
```

A saída equivale a uma lista com elementos organizados em tuplas, sendo:

- O primeiro elemento da tupla a chave do dicionário e
- O segundo elemento, o valor presente naquela chave.

Veja:

```
dict_items([('pera', 10), ('uva', 2), ('maça', 55)])
```

Podemos acessar esses valores utilizando o loop `for`, para percorrer esses dados:

```
for chave, valor in frutas.items():
    print(f"Chave = {chave} - Valor = {valor}")
```

Uma maneira segura de acessar um elemento de um dicionário é verificando primeiro se a chave existe.

Exemplo:

```
dicio_cores = {'amarelo': 10, 'vermelho': 2, 'cinza': 55}

# Verificando se chave existe
if 'amarelo' in dicio_cores:
    print(f"Como a cor desejada existe: {dicio_cores['amarelo']}")

# Verificando se valor existe
if 10 in dicio_cores.values():
    print('O valor desejado existe')
```

A saída será:

```
Como a cor desejada existe: 10
O valor desejado existe
```

Adicionando e atualizando itens

A maneira mais simples de adicionar um item a um dicionário é criar uma nova chave e atribuir um valor. Assim:

```
dicio = {'nome': 'Erick'}  
  
dicio['idade'] = 20  
  
print(dicio)
```

Resultando na adição da chave e valor especificada acima:

```
{'nome': 'Erick', 'idade': 20}
```

O método `update()` tanto pode criar, como atualizar dados:

```
dicio = {'nome': 'Erick'}  
  
# Atualiza o elemento de chave 'nome'  
dicio.update({'nome': 'Matheus'})  
  
# Cria os elementos de chave 'idade' e 'tamanho'  
dicio.update({'idade': 18})  
dicio.update(tamanho=1.60)  
  
print(dicio)
```

Saída:

```
{'nome': 'Matheus', 'idade': 18, 'tamanho': 1.60}
```

Excluindo elementos no dicionário

Você pode excluir um elemento do dicionário com a keyword `del` do Python, especificando sua chave:

```
pessoa = {'nome': 'Matheus', 'idade': 18, 'tamanho': 1.60}

del pessoa['tamanho']

print(pessoa)
```

Resultando na exclusão do dado especificado:

```
{'nome': 'Matheus', 'idade': 18}
```

Excluindo e retornando o elemento excluído

O método `pop()` remove o item cuja chave foi especificada:

```
sacola = {'maça': 2, 'ovos': 6, 'farinha': 2}

ovos = sacola.pop('ovos')

print(ovos)
print(sacola)
```

Saída:

```
6
{'maça': 2, 'farinha': 2}
```

Veja que o dicionário resultante não possui mais a chave ['ovos']

Excluindo último item

O método `popitem()` também remove o último elemento do dicionário.

Contudo, diferente do método `pop()` que apenas retorna o valor daquele elemento, `popitem()` retorna o elemento todo, contendo chave e valor.

Veja usando o mesmo exemplo do método `pop()`:

```
sacola = {'maça': 2, 'ovos': 6, 'farinha': 2}

farinha = sacola.popitem()

print(farinha)
print(sacula)
```

Veja que o retorno foi uma tupla e o último item foi removido:

```
('farinha', 2)
{'maça': 2, 'ovos': 6}
```

Limpando dicionário

Uma maneira de esvaziar um dicionário é utilizando o método `clear()`, dessa forma:

```
dicio = {'nome': 'F9', 'motor': 'v8', 'ano': 2019}
dicio.clear()

print(dicio)
```

O retorno é um dicionário vazio: `[]`.

Copiando Dicionário

Em Python, você não pode simplesmente digitar `dict1 = dict2` para copiar um dicionário.

Dessa forma você está apenas copiando a referência do `dict1` ao `dict2`.

A maneira correta de criar uma cópia do dicionário é utilizando o método `copy()`, assim:

```
dicio = {"operacao": "web scraping", "dados": 250}  
  
copia = dicio.copy()  
  
print(copia)  
print(dicio)
```

Resultando na cópia exata do dicionário

```
{'operacao': 'web scraping', 'dados': 250}  
{'operacao': 'web scraping', 'dados': 250}
```

O método `dict.setdefault()`

O método `dict.setdefault()` retorna o valor da chave especificada, e, caso a chave não exista, a cria com o valor especificado.

Vamos ao exemplo:

```
dicio = {'coleta': 'scrapy', 'dados': 200}

set_dados = dicio.setdefault('dados')

print(set_dados)
print(dicio)
```

Como a chave `dados` existe, ela é retornada:

```
200
{'coleta': 'scrapy', 'dados': 200}
```

Agora vamos adicionar apenas a chave sem valor, e outro com chave e valor:

```
dicio = {'coleta': 'scrapy', 'dados': 200}

set_data = dicio.setdefault('data')
set_teste = dicio.setdefault('teste', 1)

print(set_data)
print(set_teste)
print(dicio)
```

A saída resulta em:

```
None
1
{'coleta': 'scrapy', 'dados': 200, 'data': None, 'teste': 1}
```

O `set_data` que teve apenas atribuída uma chave, teve seu valor criado como `None`, enquanto `set_teste` teve o valor adicionado.

Criando Dicionários a partir de lista de valores

Com o método `dict.fromkeys()` é possível criar um dicionário a partir de uma lista de chaves e um valor, que será usado em todas as chaves.

Veja o exemplo:

```
chaves = ['chave1', 'chave2', 'chave3']
valor = 0

dicio = dict.fromkeys(chaves , valor)

print(dicio)
```

Veja que o valor 0 foi atribuído em todas as chaves:

```
{'chave1': 0, 'chave2': 0, 'chave3': 0}
```

Operador de “desempacotamento”

O Python introduziu em sua linguagem um operador de “*unpacking*”, ou operador de desempacotamento.

Para utilizar esse operador, devemos usar dois asteriscos antes da variável que vai ser desconstruída (`**variavel`).

Pode ser utilizado em tuplas, listas e em nossos queridos dicionários.

No caso do Dicionário, o que ele faz é desconstruir cada elemento do dicionário em chave e valor, tentando casar o nome do argumento da função com a chave do dicionário.

Ficou confuso? 😕 Vamos ao exemplo!

Suponha a seguinte função:

```
def funcao(argumento):  
    print(argumento)
```

Agora vamos chamá-la duas vezes, uma sem o operador de desempacotamento e outra com ele:

```
dicionario = {'argumento': 1}  
  
funcao(dicionario)  
funcao(**dicionario)
```

Agora veja a saída de cada um:

```
{'argumento': 1}  
1
```

Na primeira chamada, sem o operador, a saída foi apenas o `print` daquilo que foi passado. No caso, um dicionário.

Já na segunda, o operador fez com que a função tentasse “encaixar” a chave `argumento` do dicionário (de valor 1) com o argumento da função.

Dessa forma, apenas o valor foi printado!

Dicionários como `args` e `kwargs`

Como vimos aqui em cima, dicionários podem ser passados como valores para funções.

Além disso, podemos usar o operador de desempacotamento para casar as chaves do dicionário com os argumentos da função:

```
regulagem = {'max': 10, 'meio': 5, 'min': 0}

def funcao(max, meio, min):
    print(max)
    print(meio)
    print(min)

funcao(**regulagem)
```

Resultando em:

```
10
5
0
```

Se caso você desconheça as chaves de um dicionário, basta adicionar em sua função o atributo `kwargs` (*Keyword Arguments* ou **Argumentos Nomeados**).

O atributo `kwargs` recebe todos os argumento que tenham sido passados com nome.

Exemplo: `funcao(arg=1)`. Nesse caso, `arg` é um **Argumento Nomeado** e pode ser acessado pela variável `kwargs`.

Percorrendo o `kwargs` temos acesso às chaves, onde podemos, portanto, buscar os valores que desejamos através do método `.get()`.

Veja o exemplo:

```
# Definição da Função recebendo kwargs
def funcao(**kwargs):
    # Percorrendo argumento nomeados
    for chave in kwargs:
        print(f"Acessando Chave '{chave}', valor = {kwargs.get(chave)}")

regulagem = {'max': 10, 'meio': 5, 'min': 0}

funcao(**regulagem)
```

Saída impressa pela função:

```
Acessando Chave 'max', valor = 10
Acessando Chave 'meio', valor = 5
Acessando Chave 'min', valor = 0
```

Já o argumento `args` possui os **Argumentos Não-Nomeados**.

Se necessário pegar apenas as chaves do dicionário, podemos utilizar `args` para isso e chamando a função passando um asterisco em nosso dicionário, da seguinte maneira:

```
def funcao(*args):
    for chave in args:
        print(chave)

funcao(*regulagem)
```

Veja que saída contém apenas as chaves:

```
max  
meio  
min
```

Juntando dicionários com **

Outra coisa que podemos fazer com o operador de desempacotamento é juntar dicionários.

Como esse operador desconstrói um dicionário em chave e valor, podemos utilizá-lo para juntar dicionários, da seguinte forma:

```
regulagem = {'max': 10, 'meio': 5, 'min': 0}  
extra = {'passo': 2}  
  
# Junção de dicionários com **  
juncao_dicio = {**regulagem, **extra}  
  
print(juncao_dicio)
```

Saída:

```
{'max': 10, 'meio': 5, 'min': 0, 'passo': 2}
```

Casos de Uso Reais

1. Cache Simples

```
# Cache de resultados de funções pesadas
cache = {}

def processar_dados(user_id):
    if user_id in cache:
        print(f"Cache hit para {user_id}!")
        return cache[user_id]

    # Simular processamento pesado
    result = f"Dados processados para {user_id}"
    cache[user_id] = result
    return result

processar_dados(123)  # Processa
processar_dados(123)  # Retorna do cache!
```

2. Contador de Frequências

```
# Contar palavras em texto
texto = "python é ótimo python é rápido"
palavras = texto.split()

contador = {}
for palavra in palavras:
    contador[palavra] = contador.get(palavra, 0) + 1

print(contador)  # {'python': 2, 'é': 2, 'ótimo': 1, 'rápido': 1}
```

3. Agrupar Dados por Chave

```
# Agrupar usuários por cidade
users = [
    {'name': 'Alice', 'city': 'SP'},
    {'name': 'Bob', 'city': 'RJ'},
    {'name': 'Charlie', 'city': 'SP'}
]

by_city = {}
for user in users:
    city = user['city']
    if city not in by_city:
        by_city[city] = []
    by_city[city].append(user['name'])

print(by_city) # {'SP': ['Alice', 'Charlie'], 'RJ': ['Bob']}
```

4. Configurações de Aplicação

```
# Configurações com valores padrão
default_config = {
    'debug': False,
    'port': 8000,
    'host': 'localhost'
}

user_config = {'port': 9000}

# Mesclar configs (user sobrescreve default)
config = {**default_config, **user_config}
print(config) # {'debug': False, 'port': 9000, 'host': 'localhost'}
```

dict vs defaultdict vs Counter

Comparação Rápida

| Tipo | Quando usar | Vantagem |
|--------------------|---------------------|-----------------------------------|
| dict | Geral | Padrão, versátil |
| defaultdict | Chaves inexistentes | Auto-inicialização |
| Counter | Contagem | Métodos específicos |
| OrderedDict | Ordem (< 3.7) | Python 3.7+ dicts já mantém ordem |

Exemplos Práticos

```
from collections import defaultdict, Counter

# dict: Manual
contador = {}
for letra in 'mississippi':
    contador[letra] = contador.get(letra, 0) + 1

# defaultdict: Auto-inicializa (mais limpo!)
contador = defaultdict(int)
for letra in 'mississippi':
    contador[letra] += 1 # Não precisa checar se existe

# Counter: Especializado (mais simples!)
contador = Counter('mississippi')
print(contador.most_common(2)) # [('i', 4), ('s', 4)]
```

Quando Usar Cada Um?

✓ **Use dict quando:** - Mapeamento **geral** chave-valor - Precisa controlar inicialização - API pública (mais conhecido)

✓ **Use defaultdict quando:** - **Muitas chaves inexistentes** - Agrupar/agregar dados - Evitar `.get(key, default)`

✓ **Use Counter quando:** - **Contar frequências** - Operações matemáticas (adição, subtração) - `.most_common()` necessário

Performance: O(1) Lookups

```
import time

# Busca em lista: O(n)
lista = list(range(100000))
start = time.time()
99999 in lista # Varre toda a lista
print(f"Lista: {time.time() - start:.6f}s")

# Busca em dict: O(1)
dict_data = {i: i for i in range(100000)}
start = time.time()
99999 in dict_data # Instantâneo!
print(f"Dict: {time.time() - start:.6f}s")
```

Resultado: Dict é **~1000x mais rápido** para buscas!

Conclusão

Neste guia completo sobre **Dicionários**, você aprendeu:

- ✓ **Operações básicas** - Criar, acessar, modificar
- ✓ **Métodos essenciais** - get, update, pop, keys, values, items
- ✓ **Casos de uso reais** - Cache, contadores, agrupamentos, config
- ✓ **dict vs defaultdict vs Counter** - Quando usar cada um
- ✓ **Performance** - Lookups O(1) vs listas O(n)

Principais lições: - Dicts têm **busca O(1)** - instantâneo! - Use `.get(key, default)` para evitar `KeyError` - Python 3.7+ mantém **ordem de inserção** - **defaultdict** simplifica código com muitas chaves novas - **Counter** é especializado para contagem

Próximos passos: - Pratique **dict comprehensions** - Explore `ChainMap` para múltiplos dicts - Aprenda `dict.setdefault()` e `dict.fromkeys()` - Estude serialização JSON de dicts

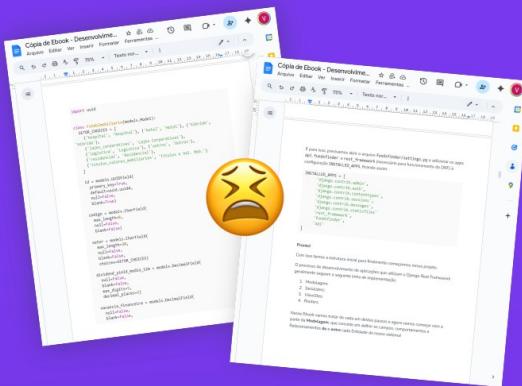
Nesse Post vimos do básico ao avançado sobre Dicionários!

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



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



Arquitetura de Software Moderna

A arquitetura de software alvo é profissional contendo o e-mail e produções de software para arquiteturas modernas. Oferece recursos como interface gráfica com interface de usuário.

```
import python
import python

class Arquitetura_de_Software_Moderna:
    ...
    def share(self):
        pass
    ...
    return "Arquitetura de NeXt", "arquitetura_moderna"
}

def __init__(self):
    if user_authenticated():
        self.user_authenticated = user_authenticated()
        self.user_email = self.user_authenticated['email']
        self.user_name = self.user_authenticated['name']
    ...
    # Envie AI para gerar o código
    return type
}
resource_available
```

AI-generated system

A arquitetura com propósito é a arquitetura moderna. Seus componentes incluem interface de usuário, banco de dados e outros sistemas externos. Chama-se de sistema gerado por IA.

Clean layout

O layout é limpo e organizado, facilitando a leitura e compreensão do código gerado.



</> 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