

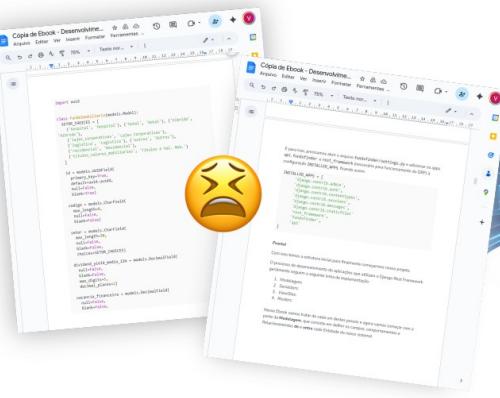


DOMINE SETS NO PYTHON

Guia completo de Sets (conjuntos) em Python: criação, operações (união, interseção, diferença), casos reais (remover duplicatas, validação, comparações), set vs frozenset, 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 práticos, operações de conjuntos e comparação set vs frozenset.

Olá Pythonista!

Sets são coleções **não-ordenadas** e **sem duplicatas** com **lookups O(1)**. Perfeitos para remover duplicatas, testes de membresía e operações matemáticas de conjuntos!

Neste guia, você vai aprender: -  Criação e operações básicas -  **Operações de conjuntos** (união, interseção, diferença) -  **Casos de uso reais** (remove duplicatas, validação) -  **set vs frozenset** - quando usar cada um

Sem mais delongas, vamos para o conteúdo!

Introdução

O `Set` é um tipo de dado bastante peculiar do Python que possui as seguintes características:

- Sets são desordenados
- Não possuem elementos duplicados, ou seja, cada elemento é único.
- Um *set* em si pode ser modificado, contudo os elementos contidos dentro dele precisam ser de tipos imutáveis.

A sintaxe para utilizar *sets* é bem simples: eles são definidos utilizando-se chaves `{}`. Veja um exemplo:

```
{1, 2, 3, 4, 5, 6}
```

Para confirmar seu tipo:

```
meu_set = {'python', 'academy'}  
  
print(type(meu_set))
```

A saída do código acima será:

```
<class 'set'>
```

Portanto, caso em algum momento você se depare com esse objeto, saiba que ele é um **Set** !

Dados duplicados

Como dito na introdução os dados no **Sets** não podem ser duplicados.

Vamos ver um exemplo para confirmar:

```
lista = [1, 1, 2, 2, 3, 3]  
  
sem_duplicados = set(lista)  
  
print(sem_duplicados)
```

Observe como os valores duplicados são ignorados:

```
{1, 2, 3}
```

Adicionando itens

Após a criação de um `Set`, você não pode alterar seus itens.

Contudo você pode adicionar novos itens e para isso podemos utilizar o método `add()`.

Vamos ver como:

```
convidados = {'João', 'Maria', 'Eduarda'}  
  
convidados.add('Marcela')  
  
print(convidados)
```

A chamada ao método `add()` adiciona o elemento ‘Marcela’ ao set:

```
{'Eduarda', 'Maria', 'Marcela', 'João'}
```

Para adicionar itens de outro conjunto ao set especificado, podemos utilizar o método `update()`.

Você pode utilizar esse método com qualquer tipo de objeto iterável (tuplas, listas, dicionários etc.)

```
ids = {10, 12, 13, 14}  
  
novos_ids = {11, 13, 15}  
  
ids.update(novos_ids)  
  
print(ids)
```

Veja como o set `ids` ficou após a chamada ao método `update()`:

```
{10, 11, 12, 13, 14, 15}
```

Acessando itens

Sets não possibilitam acessar seus elementos através de índices (assim como Listas) ou chaves (como os Dicionários).

Veja o que acontece caso tentemos realizar essa operação:

```
set_1 = {1, 2, 3}  
  
print(set_1[0])
```

Saída:

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'set' object does not support indexing
```

Assim, podemos acessá-los de duas maneiras “brutas”: percorrendo o conjunto ou verificando se o elemento desejado se encontra no *set*.

Podemos percorrer seus elementos com `for`, por exemplo:

```
este_set = {'João', 'Maria', 'Eduarda'}  
  
for item in este_set:  
    print(item)
```

Sendo impresso seu conteúdo:

```
João  
Maria  
Eduarda
```

Ou verificando se um elemento existe dentro dele:

```
print('João' in este_set)
```

Imprimindo `True` ou `False`, de acordo com o resultado da condição estabelecida.

Poxa, poucas maneiras de acessar os elementos, Hein? 😞

Calma, não entre em pânico! Podemos transformar o `set` em `list` para ganharmos as facilidades de manipulação das `Listas`:

```
lista = list(este_set)  
print(lista[0])  
print(lista[1])  
print(lista[2])
```

Saída:

```
João  
Maria  
Eduarda
```

Caso você queria saber tudo sobre `Listas`, vale dar uma olhada no nosso [post completo sobre listas!](#) 😊

Removendo itens

Para remover itens de um `set`, você pode, inicialmente, utilizar dois métodos: o `remove()` e `discard()`.

Veja como é simples remover elementos com `remove()`:

```
sacola = {'queijo', 'pão', 'leite'}  
  
sacola.remove('queijo')
```

Para remover com `discard()`, faça:

```
sacola = {'queijo', 'pão', 'leite'}  
  
sacola.discard('queijo')
```

Ambos terão o mesmo resultado:

```
{'pão', 'leite'}
```

Caso o item não exista, será gerado um erro do tipo `TypeError`!

Você também pode usar o método `pop()`, porém no caso do `set` ele não removerá o último item (pois o conjunto é desordenado):

```
compra = {'queijo', 'pão', 'leite'}  
  
item = compra.pop()  
  
print(f"Item removido: {item}")  
print(compra)
```

Ou seja, a saída pode ser um item completamente aleatório:

```
Item removido: pão  
{'queijo', 'leite'}
```

Se você deseja esvaziar completamente o `set`, utilize o `clear()`:

```
compra = {'queijo', 'pão', 'leite'}  
  
compra.clear()
```

O que fará com que o `set` `compra` fique vazio (`{}`).

Podemos deletá-lo completamente, utilizando a keyword `del` do Python, assim:

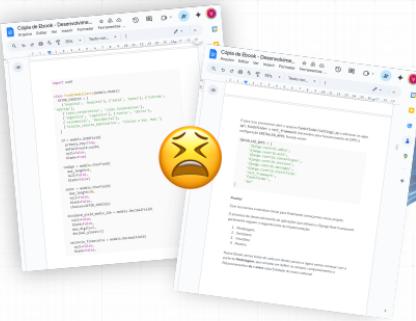
```
compra = {'queijo', 'pão', 'leite'}  
  
del compra
```



*Estou construindo o **DevBook**, uma plataforma que usa IA para criar e-books técnicos — com código formatado e exportação em PDF. Te convido a conhecêr!*

Crie Ebooks técnicos incríveis em minutos com IA

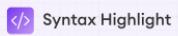
Conheça a 1ª IA Especializada na criação de Ebooks **com código!**



Chega de formatar código no Google Docs



Deixe que nossa IA faça o trabalho pesado



Syntax Highlight



Adicione Banners Promocionais



Edita em Markdown em Tempo Real



Infográficos feitos por IA

TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS 

Operações matemáticas com sets

Agora é hora de relembrar suas aulas de **Matemática**!

Muitas vezes você pode pensar que os sets são bem restritos, porém sua utilização para armazenar elementos distintos é incrivelmente útil.

Os Sets em Python nada mais são que Conjuntos Matemáticos. Neles, você também pode aplicar os conceitos de **Interseção**, **União**, **Diferença** e etc.

Interseção

O método `intersection()` retorna um novo conjunto contendo apenas os itens presentes em ambos:

```
sacola1 = {'Banana', 'Maça', 'Abacate'}
sacola2 = {'Laranja', 'Pera', 'Maça'}

sacola = sacola1.intersection(sacola2)
print(sacola)
# ou
print(sacola1 & sacola2)
```

Resultando em:

```
{'Maça'}
{'Maça'}
```

Se você deseja já **atualizar** um dos *sets* com a interseção entre eles, use o método `intersection_update()`:

```
sacola1 = {'Banana', 'Maça', 'Abacate'}
sacola2 = {'Laranja', 'Pera', 'Maça'}

sacola1.intersection_update(sacola2)

print(sacola1)
```

Veja que `sacola1` agora possui apenas a interseção entre os *sets*:

```
{'Maça'}
```

União

Você pode utilizar o método `union()` para retornar um conjunto de elementos contendo elemento de ambos *sets*:

```
set1 = {1, 2, 3}
set2 = {'z', 'x', 'a'}
```



```
print(set1.union(set2))
# ou
print(set1 | set2)
```

Resultando na união dos sets em um novo conjunto:

```
{'x', 1, 2, 3, 'z', 'a'}
{'x', 1, 2, 3, 'z', 'a'}
```

Diferença

o método `difference()` retorna a diferença, ou seja os valores que existem no `set sacola1`, e não no `set sacola2`:

```
sacola1 = {'Banana', 'Maça'}
sacola2 = {'Laranja', 'Pera', 'Maça'}
```



```
print(sacola1.difference(sacola2))
# ou
print(sacola1 - sacola2)
```

Resultando apenas nos itens que estão contidos no `set sacola1`:

```
{'Banana'}
{'Banana'}
```

Diferença simétrica

O método `symmetric_difference_update()` manterá os elementos que não estão presentes em ambos conjuntos:

```
sacola1 = {'Banana', 'Maça'}
sacola2 = {'Laranja', 'Pera', 'Maça'}

print(sacola1.symmetric_difference(sacola2))
# ou
print(sacola1 ^ sacola2)
```

Apenas os dados que não estão presentes em ambos os *sets* estarão:

```
{'Laranja', 'Banana', 'Pera'}
{'Laranja', 'Banana', 'Pera'}
```

Set Comprehensions ou Compreensão de Sets

Existe uma forma muito Pythonica de se criar *sets*, através da técnica chamada **Set Comprehensions**.

Ele é uma maneira muito concisa de se criar e manipular *sets* que a Linguagem nos oferece!

Se você quiser saber tudo sobre esse assunto, acesse agora [nossa post completo sobre Set Comprehensions AGORA!](#)

Casos de Uso Reais

1. Remover Duplicatas de Lista

```
# Manter elementos únicos
emails = ['user@example.com', 'admin@test.com', 'user@example.com']
unique_emails = list(set(emails))
print(unique_emails) # ['admin@test.com', 'user@example.com']

# Preservar ordem (Python 3.7+)
unique_ordered = list(dict.fromkeys(emails))
```

2. Validação de Permissões

```
user_permissions = {'read', 'write', 'delete'}
required_permissions = {'read', 'write'}

# Verificar se usuário tem todas as permissões
has_access = required_permissions.issubset(user_permissions)
print(has_access) # True

# Permissões faltando
missing = required_permissions - user_permissions
if missing:
    print(f"Faltam: {missing}")
```

3. Comparar Listas

```
old_users = {'alice', 'bob', 'charlie'}
new_users = {'bob', 'charlie', 'dave'}
```

Novos usuários

```
added = new_users - old_users
print(f"Adicionados: {added}") # {'dave'}
```

Usuários removidos

```
removed = old_users - new_users
print(f"Removidos: {removed}") # {'alice'}
```

Usuários que permaneceram

```
stayed = old_users & new_users
print(f"Permaneceram: {stayed}") # {'bob', 'charlie'}
```

4. Filtrar Elementos Comuns

```
# Tags de 2 posts de blog
post1_tags = {'python', 'tutorial', 'web'}
post2_tags = {'python', 'data', 'tutorial'}
```

Tags em comum

```
common_tags = post1_tags & post2_tags
print(common_tags) # {'python', 'tutorial'}
```

Todas as tags

```
all_tags = post1_tags | post2_tags
print(all_tags) # {'python', 'tutorial', 'web', 'data'}
```

set vs frozenset

Tipo	Mutável	Hasheável	Quando usar
set	✓ Sim	✗ Não	Geral, modificações
frozenset	✗ Não	✓ Sim	Chaves dict, elementos de set

Exemplo Prático

```
# set: Mutável
tags = {'python', 'web'}
tags.add('tutorial') # ✓ Pode modificar

# frozenset: Imutável
frozen_tags = frozenset(['python', 'web'])
# frozen_tags.add('tutorial') # ✗ Erro!

# Use frozenset como chave de dict
cache = {}
key = frozenset(['param1', 'param2'])
cache[key] = "resultado"

# Use frozenset como elemento de set
set_of_sets = {frozenset([1, 2]), frozenset([3, 4])}
```

Conclusão

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

- ✓ **Operações básicas** - Criar, adicionar, remover
- ✓ **Operações de conjuntos** - União, interseção, diferença
- ✓ **Casos práticos** - Remover duplicatas, validação, comparações
- ✓ **set vs frozenset** - Mutável vs imutável

Principais lições: - Sets **não têm duplicatas** (automático!) - **Lookups O(1)** - tão rápido quanto dict - Use `|` (união), `&` (interseção), `-` (diferença) - **frozenset** é imutável e hashável - Sets **não são ordenados** (ordem imprevisível)

Próximos passos: - Pratique [set comprehensions](#) - Explore métodos `.issubset()`, `.issuperset()` - Aprenda `.symmetric_difference()` (XOR) - Estude frozenset para dados imutáveis

Nesse Post vimos do básico ao avançado sobre os Sets do Python!

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

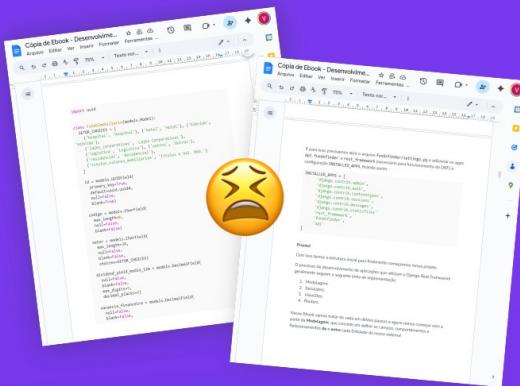
Não se esqueça de conferir!



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



Adicione Banners Promocionais



• Infográficos feitos por...

Deixe que nossa IA faça o trabalho pesado



 Edite em Markdown em Tempo Real

TESTE AGORA



 PRIMEIRO CAPÍTULO 100% GRÁTIS