



AS FUNÇÕES MAP E FILTER DO PYTHON

Guia completo de map() e filter() em Python: sintaxe, diferenças vs list comprehension, quando usar programação funcional vs Pythônica, casos de uso e exemplos práticos.

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 comparação entre `map/filter` vs `list comprehension` e quando usar cada abordagem.

Olá Pythonista!

`map()` e `filter()` são funções de programação funcional que transformam e filtram dados de forma concisa. Porém, **list comprehensions são geralmente mais Pythônicas** para a maioria dos casos!

Neste guia, você vai aprender: -  Sintaxe de `map()` e `filter()` -  **map/filter vs list comprehension** - Qual usar? -  Quando programação funcional faz sentido -  Casos de uso práticos

Ainda não domina **Laços de Repetição**? Acesse nosso [guia de loops for e while](#)!

Agora sem enrolação, vamos ao conteúdo!

A função `map`

Muitas vezes, enfrentamos situações em que precisamos executar a **mesma operação** em todos os itens de um iterável de entrada (uma lista, um set, uma tupla) para se construir um novo iterável.

A abordagem mais comum para esse tipo de problema é usar loops `for` ou `while` do Python.

Contudo, você também pode resolver esse problema sem um loop de repetição, usando a função `map()` disponível na biblioteca padrão da própria linguagem (sem precisar instalar algo com `pip` ou importar com `import`).

A sintaxe da função `map` é a seguinte:

```
map(funcao, iteravel1, iteravel2, ..., iteravelN)
```

Nela: - `funcao` é a função que iremos aplicar a cada elemento dos iteráveis de entrada. - `iteravel1, ..., iteravelN` são iteráveis nas quais a função `funcao` será aplicada, elemento a elemento

Para entender melhor, vamos aos **exemplos!**

Exemplos da função `map`

Suponha que você tenha uma lista de números e que lhe pedissem para somar 5 a todos os valores da lista.

A forma mais comum seria utilizar um loop `for`, assim:

```
lista = [1, 2, 3, 4, 5]
resultado = []

for item in lista:
    resultado.append(item + 5)
```

É possível conseguir o mesmo resultado utilizando a função `map` !

Veja como o código acima poderia ser reescrito, utilizando esta função:

```
def soma_5(numero):
    return numero + 5

lista = [1, 2, 3, 4, 5]
resultado = list(map(soma_5, lista))
```

Desta forma, estamos aplicando a função `soma_5` a cada elemento da lista `lista` e o resultado será o mesmo da versão com o loop `for`.

A função `map` com múltiplos iteráveis

Como vimos na sintaxe da função, `map` possibilita passarmos **múltiplos** iteráveis de entrada.

Para isso, é necessária uma função que possua dois ou mais argumentos.

Vamos entender no exemplo a seguir!

```
from math import pow

numeros = [1, 2, 3]
expoentes = [4, 5, 6, 7]

resultado = list(map(pow, numeros, expoentes))
```

Agora acompanhe a execução de cada iteração de `map`:

👉 Na primeira iteração, `map` vai aplicar a função `pow`, utilizando `numeros[0]` e `expoentes[0]` como argumentos, assim: `pow(1, 4) = 1`

👉 Em seguida, `map` vai aplicar a função `pow`, utilizando `numeros[1]` e `exponentes[1]`, assim: $\text{pow}(2, 5) = 32$

👉 Por último, `map` vai aplicar a função `pow`, utilizando `números[2]` e `expontes[2]`, assim: $\text{pow}(3, 6) = 729$

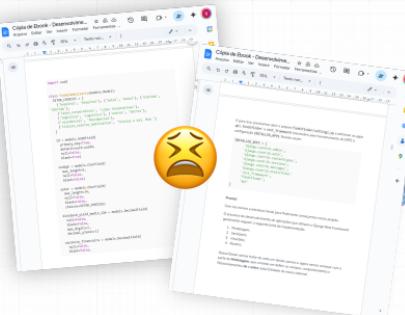
A iteração termina na menor sequência, portanto o número 7, em `expontes[3]`, não será utilizado pois a lista `numeros` não possui o elemento `numeros[3]`.

💡 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 1º 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 

A função filter

A função `filter` do Python serve para **filtrar elementos** de iteráveis que satisfaçam determinadas **condições**.

Essas **condições** são calculadas através de uma função que deve retornar `True` ou `False` para determinado argumento de entrada.

A abordagem mais comum, assim como a função `map`, seria a utilização de **loops** para iterar sobre os elementos verificando elemento a elemento se a condição foi satisfeita ou não.

Contudo, com a função `filter` é possível fazer isso sem ter que se preocupar com as estruturas de repetição `for` ou `while`!

A sintaxe da função `filter` é:

```
filter(funcao, iteravel)
```

Nela:

- `funcao` é a função de filtragem dos elementos, que deve retornar `True` ou `False`; e
- `iteravel` é a estrutura na qual a função `funcao` será aplicada.

Vamos entender melhor com **exemplos**!

Exemplos da função filter

Suponha que você tenha que desenvolver uma função que filtra elementos **maiores que zero** de uma lista.

O código abaixo, utilizando o loop de repetição `for` seria uma possível solução:

```
def numeros_positivos(numeros):
    numeros_positivos = []

    for num in numeros:
        # Condição de filtragem
        if num > 0:
            numeros_positivos.append(num)

    return numeros_positivos

print(numeros_positivos([-2, -1, 0, 1, 2]))
```

E a saída seria:

```
[1, 2]
```

Veja como seria possível chegar ao mesmo resultado utilizando a função `filter`:

```
numeros = [-2, -1, 0, 1, 2]

def verifica_numero_positivo(numero):
    return numero > 0

print(list(filter(verifica_numero_positivo, numeros)))
```

E a saída seria exatamente a mesma: com menos linhas de código e com um código muito mais pythônico!

Dica EXTRA: Funções Lambda

Quer deixar seu código ainda mais **pythônico**?

Utilize **funções lambda** em conjunto com as funções `map` e `filter` e tenha os mesmos resultados dos exemplos anteriores com ainda menos linhas de código!

*Ainda não domina o conceito de **Funções Lambda**?! Não tem problema, acesse nosso [artigo sobre Funções Lambda](#) antes de continuar 😊*

Exemplo “soma + 5” do `map` utilizando **funções lambda**:

```
lista = [1, 2, 3, 4, 5]

resultado = list(map(lambda x: x+5, lista))
```

Exemplo de filtragem de números positivos do `filter` utilizando **funções lambda**:

```
numeros = [-2, -1, 0, 1, 2]

numeros_positivos = list(filter(lambda x: x > 0, numeros))
```

map/filter vs List Comprehension

Comparação Prática

```
numbers = [1, 2, 3, 4, 5]

# Transformar: elevar ao quadrado
# map()
result_map = list(map(lambda x: x**2, numbers))

# List comprehension (⚡ Mais Pythonico!)
result_comp = [x**2 for x in numbers]

# Filtrar: apenas pares
# filter()
evens_filter = list(filter(lambda x: x % 2 == 0, numbers))

# List comprehension (⚡ Mais Pythonico!)
evens_comp = [x for x in numbers if x % 2 == 0]

# Transformar + Filtrar: pares ao quadrado
# map() + filter()
result_functional = list(map(lambda x: x**2, filter(lambda x: x % 2 == 0, numbers)))

# List comprehension (⚡ MUITO mais legível!)
result_pythonic = [x**2 for x in numbers if x % 2 == 0]
```

Quando Usar map/filter?

✓ **Use map() quando:** - Já tem a **função pronta** (não precisa de lambda) - Quer **reutilizar** a transformação - Programação **funcional** é o estilo do projeto

```
# ✅ BOM: Função pronta
import math
numbers = [1, 4, 9, 16]
sqrt_numbers = list(map(math.sqrt, numbers)) # Limpo!

# vs list comprehension (mais verboso)
sqrt_numbers = [math.sqrt(x) for x in numbers]
```

✓ Use **filter()** quando: - **Filtro complexo** já existe como função - Combinar com outros **iteradores**

```
def is_valid_email(email):
    return '@' in email and '.' in email.split('@')[1]

emails = ['user@test.com', 'invalid', 'another@valid.com']
valid = list(filter(is_valid_email, emails)) # ✅ Claro!
```

⚡ Use **list comprehension** quando: - **Maioria dos casos** (mais Pythônico!) - Precisa de **legibilidade** - Lógica é **inline**

Caso de Uso Prático

```
# Processar dados de API
users_raw = [
    {'name': 'Alice', 'age': 30, 'active': True},
    {'name': 'Bob', 'age': 17, 'active': True},
    {'name': 'Charlie', 'age': 25, 'active': False},
]

# Abordagem 1: Programação Funcional
active_adults = list(
    map(lambda u: u['name'],
        filter(lambda u: u['active'] and u['age'] >= 18, users_raw)
    )
)

# Abordagem 2: List Comprehension (⚡ Mais legível!)
active_adults = [
    u['name']
    for u in users_raw
    if u['active'] and u['age'] >= 18
]

print(active_adults) # ['Alice']
```

Use `map`, `filter` em **funções lambda** na sua próxima entrevista de emprego e **impressione** o recrutador! Mas lembre-se: **list comprehensions são mais Pythonicas!**

Conclusão

Neste guia completo sobre **map()** e **filter()**, você aprendeu:

- ✓ **Sintaxe** - map() transforma, filter() filtra
- ✓ **map/filter vs list comprehension** - Comprehensions são mais Pythônicas
- ✓ **Quando usar cada um** - Funções prontas vs inline
- ✓ **Casos práticos** - Processamento de dados

Principais lições: - **List comprehensions** são **mais Pythônicas** na maioria dos casos - Use `map()` quando já tem **função pronta** (sem lambda) - Use `filter()` quando filtro é **complexo** e reutilizável - Evite **lambdas complexas** em map/filter - **Legibilidade > pureza funcional**

Próximos passos: - Pratique **list comprehensions** (mais importante!) - Explore `functools.reduce()` para agregações - Estude `itertools` para operações avançadas - Combine com generators para eficiência

Depois de termos nos aprofundado nas funções **map**, **filter**, além das **funções lambda**, você poderá treinar essas habilidades a fim de se tornar um programador cada vez **melhor!**

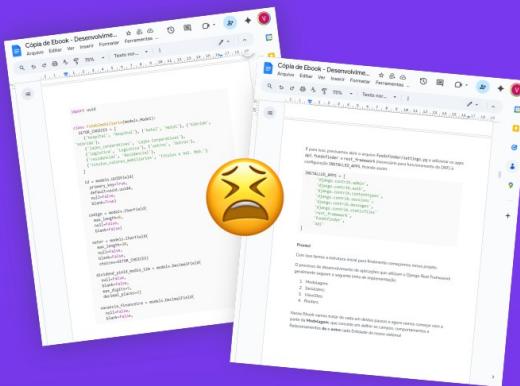
Não se esqueça de conferir!



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 para...

Deixe que nossa IA faça o trabalho pesado



 Edite em Markdown em Tempo Real

TESTE AGORA



 PRIMEIRO CAPÍTULO 100% GRÁTIS