Funções Lambda em Python

Funções anônimas, expressões e introdução à programação funcional

Prof. Guilherme Soares

λ Roteiro da aula

01 Fundamentos

O que é? · sintaxe · lambda × def · regra da expressão única · argumentos · operador ternário

02 Funções de ordem superior

map() · filter() · reduce() · sorted/min/max com key= · lambda × list comprehension

03 Boas práticas & armadilhas

Quando usar (e evitar) · PEP 8 · closures e late binding · usos no mundo real

04 Prática no Google Colab

Bateria de exercícios com verificação automática (assert) — do básico ao mini-projeto

01

Fundamentos

O que é uma função lambda, sua sintaxe e como ela se compara a uma função def.

λ O que é uma função lambda?

Uma função lambda é uma função **anônima** (sem nome) definida em uma **única expressão**.

- λ **É uma expressão:** produz um objeto-função, que pode ser passado adiante como qualquer valor.
- λ **Retorno implícito:** o valor da expressão é devolvido. Ou seja, não precisa da palavra return.
- λ **Curta e descartável:** ideal para funções simples usadas em um único lugar.

```
# uma função que devolve o dobro
lambda x: x * 2
```

λ Sintaxe: anatomia de uma lambda

Forma geral: `lambda argumentos: expressão`

```
# chamando uma lambda na hora (IIFE)
(lambda x, y: x + y)(3, 4)     # -> 7
```

λ lambda × def: a mesma ideia

def — função nomeada

```
def dobro(x):
    return x * 2

dobro(5)      # -> 10
```

lambda — função anônima

```
dobro = lambda x: x * 2


dobro(5)      # -> 10
```

Aspecto	def	lambda
Nome	tem nome próprio	anônima (<lambda>)
Corpo	vários comandos	uma única expressão
Retorno	explícito (return)	implícito
Melhor para	lógica reutilizável	funções curtas e pontuais

λ A regra é: uma só expressão

O corpo de uma lambda aceita apenas UMA expressão, nunca comandos (statements).

✓ Expressões — permitido

- `lambda x: x * 2 + 1`
- `lambda s: s.upper()`
- `lambda x: "par" if x%2==0 else "ímpar"`
- `lambda n: sum(range(n))`
- `lambda x: [i*i for i in range(x)]`

✗ Comandos — proibido

- ✗ `return` (o retorno já é implícito)
- ✗ `x = 10` (atribuição com =)
- ✗ `for / while` (laços)
- ✗ `if x: …` (if em bloco)
- ✗ `try / except` · várias linhas

Precisa de mais de uma expressão ou de comandos? Use def.

λ Argumentos da lambda

λ Vários argumentos

```
soma = lambda a, b, c: a + b + c
soma(1, 2, 3)          # -> 6
```

λ Sem argumentos

```
agora = lambda: 42
agora()                # -> 42
```

λ Valor padrão

```
inc = lambda x, p=1: x + p
inc(10)                # -> 11
```

λ *args / **kwargs

```
total = lambda *xs: sum(xs)
total(1, 2, 3, 4)      # -> 10
```

λ Decisões: o operador ternário

Como não há if em bloco, usamos a expressão condicional dentro da lambda.

valor_se_verdadeiro **if** *condição* **else** *valor_se_falso*

```
classificar = lambda x: "par" if x % 2 == 0 else "ímpar"
classificar(4)        # -> "par"

maioridade = lambda i: "adulto" if i >= 18 else "menor"

# ternário aninhado (use com moderação)
sinal = lambda n: "+" if n > 0 else ("-" if n < 0 else "0")
```

02

Funções de ordem superior

Onde a lambda brilha: map(), filter(), reduce(), sorted() e amigos.

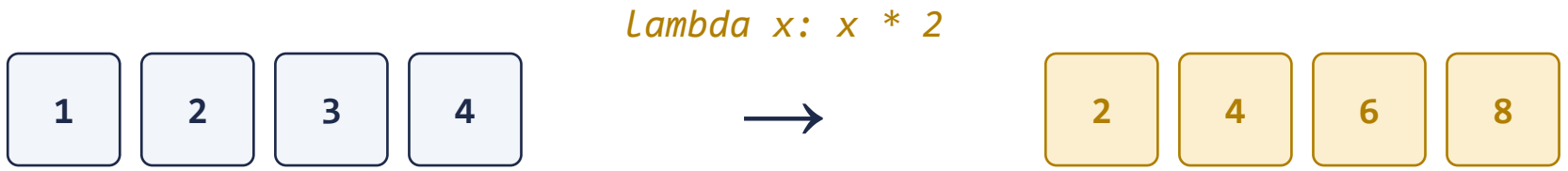
λ O que são funções de ordem superior?

Função de ordem superior é aquela que **recebe uma função como argumento** e/ou **devolve uma função**. A lambda é perfeita para fornecer essa “função descartável”.

- `map()` — transforma cada item
- `filter()` — seleciona itens
- `reduce()` — combina em um valor
- `sorted() / max() / min()` — ordena/escolhe por um critério (key=)

λ map(): transformar cada item

`map(função, iterável)` — aplica a função a cada elemento e devolve um iterador.




```
nums = [1, 2, 3, 4]
dobrados = list(map(lambda x: x * 2, nums))
# dobrados -> [2, 4, 6, 8]
```

⚠️ **Atenção:** map() é preguiçoso — devolve um iterador, não uma lista. Envolva em `list()` (ou itere) para ver os resultados.

λ filter(): selecionar itens

`filter(predicado, iterável)` — mantém apenas os itens em que a função retorna True.


```
nums = [-2, -1, 0, 1, 2]
positivos = list(filter(lambda x: x > 0, nums))
# positivos -> [1, 2]
```

Predicado = função que devolve True/False. `filter()` também é preguiçoso (use `list()`).

λ reduce(): combinar em um valor

`reduce(função, iterável[, inicial])` — acumula os itens, dois a dois, num único resultado.

```python
from functools import reduce

nums = [1, 2, 3, 4]
soma = reduce(lambda a, b: a + b, nums)
# soma -> 10
```

Como acumula:

1. `1 + 2 = 3`
2. `3 + 3 = 6`
3. `6 + 4 = 10`

Lembre-se: reduce não é embutida — é preciso `from functools import reduce`. Para somar, prefira a embutida `sum()`.

λ sorted(), max(), min() com key=

O uso mais comum de lambda no dia a dia: definir o critério de ordenação/escolha.

```
palavras = ["pera", "uva", "banana"]
sorted(palavras, key=lambda s: len(s))
# -> ['uva', 'pera', 'banana']
```

```
alunos = [("Ana", 9.0), ("Bruno", 7.5)]
max(alunos, key=lambda a: a[1])
# -> ('Ana', 9.0)
```

```
# ordem decrescente por nota
sorted(alunos, key=lambda a: a[1], reverse=True)
# -> [('Ana', 9.0), ('Bruno', 7.5)]
```

key recebe uma função aplicada a cada item; a ordenação usa o valor devolvido. `reverse=True` inverte a ordem.

λ map/filter × list comprehension

Em Python, comprehensions costumam ser mais legíveis. Conheça as duas formas.

Com map/filter + lambda

```
list(map(lambda x: x*2, xs))

list(filter(lambda x: x > 0, xs))
```

Com list comprehension

```
[x*2 for x in xs]

[x for x in xs if x > 0]
```

Quando usar cada uma?

- **Comprehension:** transformações/filtros simples — mais idiomática e legível.
- **map/filter:** quando já existe uma função pronta (ex.: `map(str.upper, xs)`) ou em pipelines preguiçosos com grandes volumes.

03

Boas práticas & armadilhas

Quando usar, o que diz a PEP 8 e a clássica pegadinha do late binding.

λ Quando usar (e quando evitar)

✓ Boas situações

- Função curta e descartável
- Passada como argumento (key=, callbacks)
- Quando nomear não agrega clareza

✗ Evite

- Atribuir a um nome (use def — PEP 8)
- Lógica complexa ou de várias linhas
- Quando prejudica a legibilidade

```
# ✗ Evite: atribuir lambda a um nome
f = lambda x: x * 2

# ✓ Prefira: def (melhor para depuração/traceback)
def f(x):
    return x * 2
```

PEP 8

"Sempre use um def em vez de atribuir uma expressão lambda diretamente a um identificador."

λ Armadilha: closures e late binding

⚠️ **Avançado —** variáveis são lidas quando a lambda é chamada, não quando é criada.

✗ O problema

```
funcs = [lambda: i for i in range(3)]
[f() for f in funcs]
# -> [2, 2, 2]   (e não [0,1,2]!)
```

✓ A correção

```
# fixe o valor com um argumento padrão
funcs = [lambda i=i: i for i in range(3)]
[f() for f in funcs]
# -> [0, 1, 2]
```

Por quê? Todas as lambdas compartilham a mesma variável `i`, que ao final do laço vale 2. O argumento padrão `i=i` captura o valor atual de `i` no momento da criação de cada função.

λ Lambda no mundo real

λ Análise de dados (pandas)

```
df["preco"].apply(lambda p: p * 1.1)
```

λ Ordenar dados

```
sorted(produtos, key=lambda p: p["preco"])
```

λ Interfaces gráficas

```
Button(janela, command=lambda: salvar(id))
```

λ Escolher por critério

```
max(palavras, key=len)   # aqui nem precisa de lambda!
```

λ Resumo: o essencial

- **Definição** — função anônima de uma só expressão; retorno implícito.
- **Sintaxe** — lambda args: expressão.
- **lambda × def** — equivalentes; para nomear, use def (PEP 8).
- **Ordem superior** — map transforma · filter seleciona · reduce combina.
- **key=** — uso campeão: sorted/max/min por um critério.
- **Comprehensions** — muitas vezes mais legíveis que map/filter.
- **Armadilha** — late binding em laços → use arg padrão i=i.

PRÁTICA GUIADA

Hora de praticar no Google Colab

1 **Abra o notebook** — Exercicios_Lambda_Aluno.ipynb em colab.research.google.com

2 **Salve a sua cópia** — Arquivo → Salvar uma cópia no Drive (para editar e guardar).

3 **Resolva os exercícios** — Substitua cada TODO pelo seu código, do básico ao mini-projeto.

4 **Verifique-se** — Rode a célula: os assert confirmam ✅ quando a resposta está correta.

λ Referências — documentação & artigos

- Documentação Python (PT-BR) · Tutorial: funções lambda — docs.python.org/pt-br/3/tutorial/controlflow.html#lambda-expressions
- Referência da linguagem · Expressões lambda — docs.python.org/3/reference/expressions.html#lambda
- Biblioteca padrão · functools.reduce — docs.python.org/pt-br/3/library/functools.html#functools.reduce
- Funções embutidas · map(), filter(), sorted() — docs.python.org/pt-br/3/library/functions.html
- PEP 8 · Guia de estilo (recomendação sobre lambda) — peps.python.org/pep-0008/
- Real Python · How to Use Python Lambda Functions — realpython.com/python-lambda
- W3Schools · Python Lambda — w3schools.com/python/python_lambda.asp
- GeeksforGeeks · Lambda, filter, map e reduce — geeksforgeeks.org/python/python-lambda-anonymous-functions-filter-map-reduce