

Departamento de Engenharia de Sistemas e Computação

Processamento de Listas

Fatiamento, compreensão de listas e métodos

João Araujo

Algoritmos Computacionais I

Aula 17

Roteiro

- Listas por compreensão (Cap. 5.4)
- Somando e multiplicando listas (5.5)
- Extraindo sublistas: fatiamento (5.6)
- Prática com exercícios do site

Dica

Esta é uma aula de **prática**: tente cada exemplo no interpretador antes de avançar.

Listas por compreensão

```
lista = [1, 2, 3, 5, 7]
lista_nova = [x * x for x in lista]
lista_quadrados = [x * x for x in range(5)]
print(lista_nova)         # [1, 4, 9, 25, 49]
print(lista_quadrados)    # [0, 1, 4, 9, 16]
```

Receita

`[` *expressão* `for` *item* `in` *sequência* `]`: constrói uma lista nova aplicando a expressão a cada item.

Equivale ao laço com append, mas em uma linha:

```
lista_nova = []
for x in lista:
    lista_nova.append(x * x)
```

Fonte: code/Cap5/prog.5.15_lista_compreensao.py

Compreensão com filtro

```
lista_nova = [x * x for x in range(10) if x % 2 == 0]
print(lista_nova)          # [0, 4, 16, 36, 64]

nomes = ['José', 'João', 'Joaquim']
lista = ['Bom dia, ' + x for x in nomes]
for saudacao in lista:
    print(saudacao)
```

- O `if` no final escolhe *quais* itens entram
- Exercícios 5.7 a 5.10: quadrados dos pares, quadrados dos ímpares, cubos, múltiplos de 7

Fonte: `code/Cap5/prog.5.16_lista_compreensaocomfiltro.py`, `5.17_bomdialistas.py`

Diferença entre listas com compreensão

```
a = [2, 3, 4, 5, 5, 4, 7]
b = [7, 4, 5]
lista_nova = [x for x in a if x not in b]
print(lista_nova)          # [2, 3]
```

- "Elementos de a que não estão em b": compreensão + filtro com `not in`
- Muito usada para limpar dados: remover valores inválidos, selecionar registros...

Exemplo de engenharia

```
validas = [t for t in medidas if -50 <= t <= 150]
```

Descarta leituras impossíveis de um sensor de temperatura.

Fonte: `code/Cap5/prog.5.19_lista_diferenca.py`

Somando e multiplicando listas

```
lista1 = [1, 2, 3]
print(lista1 * 3)              # [1, 2, 3, 1, 2, 3, 1, 2, 3]
print(lista1 + [4, 5])         # [1, 2, 3, 4, 5]
zeros = [0] * 10               # lista com 10 zeros
print(zeros)
```

- + concatena listas; * repete a lista
- Não é soma vetorial! `[1,2] + [3,4]` é `[1,2,3,4]`, e não `[4,6]` (para isso existe NumPy, Aula 21)
- Exercícios 5.11 e 5.12: rotação de lista; separar pares e ímpares

Fonte: `code/Cap5/prog.5.18_multlista.py`

Fatiamento (*slicing*)

```
planetas = ['Mercúrio', 'Vênus', 'Terra', 'Marte', 'Saturno',
            'Júpiter', 'Urano', 'Netuno']
print(planetas[2:5])      # ['Terra', 'Marte', 'Saturno']
print(planetas[:3])       # ['Mercúrio', 'Vênus', 'Terra']
print(planetas[3:])       # de Marte até o fim
print(planetas[4:-1])     # de Saturno até Urano
print(planetas[:])        # cópia completa
```

Regra: `lista[inicio:fim:passo]`

Inclui o `inicio`, exclui o `fim`; omitidos valem: começo, fim da lista, passo 1.

Fonte: `code/Cap5/prog.5.20_planetas.py` (Exercício 5.16)

Fatiamento com passo

```
primos = [2, 3, 13, 17, 19]
print(primos[::2])       # de 2 em 2: [2, 13, 19]
print(primos[::3])       # de 3 em 3: [2, 17]
primos[2:2] = [5, 7]     # insere sem apagar nada
primos[4:4] = [11]
print(primos)            # [2, 3, 5, 7, 11, 13, 17, 19]
primos[2:3] = []         # remove um elemento
print(primos[3::2])      # de 2 em 2 a partir do índice 3
```

- Fatias à esquerda do = *substituem*, *inserem* ou *removem* trechos

Fonte: `code/Cap5/prog.5.21_listas_fatiamento.py`

Passo negativo: invertendo

```
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(lista[::-1])      # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
print(lista[4::-1])     # [4, 3, 2, 1, 0]
print(lista[:5:-1])     # [9, 8, 7, 6]
print(lista[5:2:-1])    # [5, 4, 3]
```

- `[::-1]` é o jeito mais curto de inverter uma lista (ou uma string!)
- Com passo negativo, o "começo" e o "fim" são trocados
- Exercícios 5.13 a 5.16: fatiamento e fatiamento negativo

Fonte: `code/Cap5/prog.5.22_passo_negativo.py`

Revisando as ideias em um exemplo

Dadas as **leituras horárias** de temperatura de um dia:

```
temps = [18.5, 18.0, 17.8, 17.5, 18.2, 19.5, 21.0, 23.4,
         25.1, 26.8, 28.0, 29.1, 29.5, 29.2, 28.4, 27.0,
         25.3, 23.9, 22.5, 21.4, 20.5, 19.8, 19.2, 18.7]
manha = temps[6:12]                    # 6h às 11h
tarde = temps[12:18]                   # 12h às 17h
quentes = [t for t in temps if t > 28]
em_f = [round(9 / 5 * t + 32, 1) for t in temps]
alternadas = temps[::2]                # a cada 2 horas
```

Uma lista, várias visões: fatias para recortar, compreensões para transformar e filtrar.

Atividade: mãos à obra

Com a lista `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`, escreva (em uma linha cada) o código que gera:

1. `[5, 6, 7, 8, 9, 0, 1, 2, 3, 4]` (rotação, Exercício 5.11)
2. `[1, 3, 5, 7, 9, 0, 2, 4, 6, 8]` (Exercício 5.12)
3. `[9, 7, 5, 3, 1]`
4. `[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`
5. os múltiplos de 7 entre 100 e 1000 que **não** são múltiplos de 3 (Exercício 5.25)

Depois confira com as soluções do site.

Para praticar: exercícios do livro

Site do livro – Capítulo 5

- Compreensão: **5.7, 5.8, 5.9, 5.10**
- Somar e multiplicar listas: **5.11, 5.12**
- Fatiamento: **5.13, 5.14, 5.15, 5.16**

Enunciados e soluções comentadas em
`https://www.araujo.eng.uerj.br/livro/exercicios/cap5/`

Exercícios adicionais (sem solução): `https://www.araujo.eng.uerj.br/livro/extras/`

Resumo da aula

- Compreensão: `[expr for x in seq if cond]`
- `+` concatena; `*` repete
- Fatias: `lista[ini:fim:passo]` (fim exclusivo); `[::-1]` inverte
- Fatias à esquerda do `=` modificam trechos da lista
- Próxima aula: operações avançadas, cópias e listas como parâmetros