Listas e Mutabilidade

Thiago Leucz Astrizi

Faculdade Municipal de Palhoça

10 de setembro de 2025

Índices e Ordenação

```
1  lista_a = []
2  L = [2, 'a', 4, [1, 2]]
3
4  [1, 2] + [3, 4]  #Avalia para [1, 2, 3, 4]
5  len(L)           # Avalia para 4
6  L[0]             # Avalia para 2
7  L[3]             # Avalia para [1, 2] (outra lista)
8  L[4]             # ERRO
9  max([3, 5, 0])   # Avalia para 5
10 L[1:3]           # Avalia para ['a', 4]
11 for e in L       # Itera sobre a lista
12
13 L[3] = 10        # Modifica a lista para [2, 'a', 4, 10]
```

Mutabilidade

- Listas são mutáveis!
- Atribuir para um elemento de lista **muda** o valor:

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

Mutabilidade

- Compare:
  - Criar L mudando um elemento
  - Criat t criando um novo objeto

```
1 L = [2, 4, 3]
2 L[1] = 5
3
4 t = (2, 4, 3)
5 t = (2, 5, 3)
```

Operações em Listas - append

- Adicione elementos para o final de uma lista com `L.append(element)`
- Modifique a lista!

```
1 L = [2, 1, 3]
2 L.append(5)   # L agora é [2, 1, 3, 5]
```

Operações em Listas - append

- Adicione elementos para o final de uma lista com `L.append(element)`
- Modifique a lista!

```
1 L = [2, 1, 3]
2 L.append(5)      # L agora é [2, 1, 3, 5]
3
4 L = L.append(5) # Erro lógico: não faz o que você espera
```

Operações em Listas - append

- Adicione elementos para o final de uma lista com `L.append(element)`
- Modifique a lista!

```
1 L = [2, 1, 3]
2 L.append(5)   # L agora é [2, 1, 3, 5]
3 L.append(5)   # L agora é [2, 1, 3, 5, 5]
4 print(L)
```

Exercício Rápido

Qual o valor de L1, L2, L3 e L no fim do código abaixo?

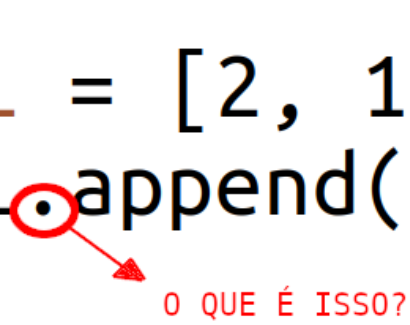
```
1 L1 = ["re"]
2 L2 = ["mi"]
3 L3 = ["do"]
4
5 L4 = L1 + L2
6 L3.append(L4)
7 L = L1.append(L3)
```

Algumas funções modificam a lista e não retornam nada!

Usamos elas só pelo efeito colateral.

Operações em Listas - append

```
1 L = [2, 1, 3]
2 L.append(5)
3
```



- Listas são objetos em Python—tudo é um objeto em Python.
- Objetos possuem **dados**
- Objetos também possuem **operações associadas**
- Acessamos esta informação com `objeto.faz_alguma_coisa()`
- É como uma função `append(L, 5)`

Exercício Rápido

Crie uma função que atenda à seguinte especificação:

```
1 def cria_lista_ordenada(n):
2     """
3     ENTRADA: 'n', um inteiro positivo
4     SAÍDA: Uma lista contendo todos os números inteiros de 0 até 'n' (inclusive) na ordem.
5     """
```

Exercício Rápido

Crie uma função que atenda à seguinte especificação:

```
1  def remove_elemento(L, e):
2      """
3      ENTRADA: 'L', uma lista e 'e', um elemento que pode ou não estar na lsita L.
4      SAÍDA: Uma nova lista com os elementos na mesma ordem que 'L', mas sem
5             nenhum 'e'.
6      """""
7
8  # Exemplo:
9  # L = [1, 2, 2, 2]
10 # print(remove_elem(L, 2)) # Deve imprimir [1]
```

Strings para Listas

- Converta **strings** para **listas** com `list(s)`
  - Todo caractere da string vira elemento da lista
- Use `s.split()`, para separar uma string em um caractere usado como parâmtro. Se quiser separar string nos espaços, não precisa de parâmetro.

```
1 s = "Eu<3 ads &vc?"  # s é uma string
2 L = list(s)          # L é ['E', 'u', '<', '3', ' ', 'a', 'd', 's', ' ', '&', 'v', 'c']
3
4 L1 = s.split(' ')    # L1 é ['Eu<3', 'ads', '&vc?']
5 L2 = s.split('<')    # L2 é ['Eu', '3 ads &vc?']
```

Listas para Strings

- Converta uma lista de strings de volta para uma string
- Use ''.join(L) para tornar uma lista de strings em uma string maior
- Você pode colocar um caractere entre aspas para adicioná-lo entre cada elemento.

```
1 L = ['a', 'b', 'c']           # L é uma lista
2 A = ''.join(L)                # A é "abc"
3 B = '_'.join(L)               # B é "a_b_c"
4 C = ''.join([1, 2, 3])        # ERRO
5 C = ''.join(['1', '2', '3']) # C é "123"
```

Exercício Rápido

Crie uma função que atenda à seguinte especificação:

```
1 def conta_palavras(frase):
2     """
3     ENTRADA: 'frase' é uma string representando uma frase.
4     SAÍDA: Um inteiro não-negativo representando quantas palavras tem a frase.
5            Uma palavra são caracteres delimitados por espaços.
6     """
7
8 print(conta_palavras("Olá, sou eu!")) # Deve retornar 3.
```

Exercício Rápido

Crie uma função que atenda à seguinte especificação:

```
1 def conta_palavras(frase):
2     """
3     ENTRADA: 'frase' é uma string representando uma frase.
4     SAÍDA: Um inteiro não-negativo representando quantas palavras tem a frase.
5            Uma palavra são caracteres delimitados por espaços.
6     """
7
8 print(conta_palavras("Olá, sou eu!")) # Deve retornar 3.
```

OBS: Imagine o quão mais difícil seria fazer isso sem listas.

Algumas Operações Interessantes sobre Listas

- Adicionar elementos no fim da lista com `L.append(elemento)`
  - Modifica a lista
- Ordenar listas com `sort()`:

```
1 L = [4, 2, 7]
2 L.sort()      # L agora é [2, 4, 7]
```

  - Modifica a lista
- Inverter a ordem dos elementos com `reverse()`

```
1 L = [4, 2, 7]
2 L.reverse()      # L agora é [7, 2, 4]
```

  - Modifica a lista
- Gerar nova lista ordenada com `sorted()`

```
1 L = [4, 2, 7]
2
3 novo_L = sorted(L)
```

  - NÃO modifica a lista. Gera nova lista ordenada.

Mutabilidade

```
1 L = [9, 6, 0, 3]
2 L.append(5)
3 a = sorted(L) # Retorna nova lista ordenada, não modifica L
4
5
6 b = L.sort() # Modifica L para [0, 3, 5, 6, 9] e retorna None
7 L.reverse()  # Modifica L para [9, 6, 5, 3, 0] e retorna None
```

Exercício Rápido

Crie uma função que atenda à seguinte especificação:

```
1 def ordena_palavras(frase):
2     """
3     ENTRADA: 'frase' é uma string representando uma frase.
4     SAÍDA: Uma lista contendo todas as palavras de 'frase', mas
5            organizadas em ordem alfabética.
6     """
7
8 print(ordena_palavras("olha esta foto")) # Deve imprimir ["esta", "foto", "olha]
```

Funções com efeitos colaterais modificam entradas.

Você pode escrever as suas próprias!

Listas Suportam Iteração

- Vamos escrever uma função que modifica a entrada.
- Exemplo: Recebe uma lista de números e ele ao quadrado

```
1 def lista_ao_quadrado(L):
2     for elemento in L:
3         # E agora...?
4         # Queremos fazer lista[i] = lista[i] * lista[i]...
5         # Mas não temos este índice 'i'...
```

Listas Suportam Iteração

- Vamos escrever uma função que modifica a entrada.
- Exemplo: Recebe uma lista de números e ele ao quadrado

```
1 def lista_ao_quadrado(L):
2     for elemento in L:
3         # E agora...?
4         # Queremos fazer lista[i] = lista[i] * lista[i]...
5         # Mas não temos este índice 'i'...
```

- Soluções:
  - Cria nova variável `i`, inicializa em 0 e incrementa ela cada iteração do loop.
  - Itere sobre o índice usando `for i in range(len(L))`
  - Usar `enumerate` no `for`: `for i, e in enumerate(L)`

Listas Suportam Iteração

- Vamos escrever uma função que modifica a entrada.
- Exemplo: Recebe uma lista de números e ele ao quadrado

```
1 def lista_ao_quadrado(L):
2     for i in range(len(L)):
3         L[i] = L[i] ** 2
```

- Sem `return`.

Listas Suportam Iteração

- Vamos escrever uma função que modifica a entrada.
- Exemplo: Recebe uma lista de números e ele ao quadrado

```
1 def lista_ao_quadrado(L):
2     for i in range(len(L)):
3         L[i] = L[i] ** 2
4
5 L = [2, 3, 4]
6 print("Antes: ", L)  # Imprime "Antes: [2, 3, 4]"
7 lista_ao_quadrado(L)
8 print("Depois: ", L) # Imprime "Depois: [4, 9, 16]"
```

Funções que modificam a lista de entrada...

Provavelmente iteram sobre `len(L)`, não `L`.
Provavelmente retornam `None`, então o valor de retorno não precisa ser salvo.

Mutação

- Listas são estruturas mutáveis.
- Existem muitas vantagens de ser capaz de mudar uma parte de uma lista.
  - Se tivéssemos uma lista muito grande e quiséssemos modificar só 1 elemento, se não tivéssemos mutações, teríamos que criar uma cópia nova da lista para podermos ter apenas um elemento diferente.
- Mas esta habilidade pode trazer desafios inesperados.

Alguns Exemplos Complicados

1. Estamos iterando sobre os índices de `L`, mas a cada iteração nós modificamos `L` e adicionamos novos elementos.
2. Estamos iterando sobre os elementos de `L`, mas a cada iteração nós modificamos `L` e adicionamos novos elementos.
3. Estamos iterando sobre os elementos de `L`, mas a cada iteração, atribuímos `L` a um novo objeto.
4. Estamos iterando sobre os elementos de `L`, mas a cada iteração nós modificamos `L` e removemos elementos.

Exemplo Complicado 1

Estamos iterando sobre os índices de `L`, mas a cada iteração nós modificamos `L` e adicionamos novos elementos.

- `range` retorna algo que parece uma tupla (mas não é—ele retorna um "iterável")
  - Retorna o primeiro elemento e um método de iteração à partir do qual os elementos seguintes são obtidos

```
1 range(4)          # Algo parecido com a tupla (0, 1, 2, 3)
2 range(2, 9, 2)    # Algo parecido com a tupla (2, 4, 6, 8)
3
4 L = [1, 2, 3, 4]
5
6 for i in range(len(L)):
7     L.append(i)
8     print(L)
```

Exemplo Complicado 2

Estamos iterando sobre os elementos de `L`, mas a cada iteração nós modificamos `L` e adicionamos novos elementos.

```
1 L = [1, 2, 3, 4]
2 i = 0
3
4 for e in L:
5     L.append(i)
6     i += 1
7     print(L)
```

Combinando Listas

- A concatenação (+) cria nova lista com cópias.
- Para modificar a lista, use `L.extend(outra_lista)`

```
1 L1 = [2, 1, 3]
2 L2 = [4, 5, 6]
3 L3 = L1 + L2                  # L3 é agora [2, 1, 3, 4, 5, 6]
4 L1.extend([0, 6])             # L1 é modificada para [2, 1, 3, 0, 6]
5 L2.extend([[1, 2], [3, 4]]) # L2 é modificado para [4, 5, 6, [1, 2], [3, 4]]
```

Exemplo Complicado 3

Estamos iterando sobre os elementos de `L`, mas a cada iteração, atribuímos `L` a um novo objeto.

```
1 L = [1, 2, 3, 4]
2 for e in L:
3     L = L + L
4     print(L)
```

Esvaziando uma lista e checando que ela ainda é o mesmo objeto

- Você pode esvaziar todos os elementos de uma lista.
  - Isso não cria uma nova lista vazia!
- Use `L.clear()`.
- Como checar se é o mesmo objeto na memória?
  - Usefunção `id()`
  - Tente executar:

```
1 L = [4, 5, 6]
2 print(id(L))
3 L.append(8)
4 print(id(L))
5 L.clear()
6 print(id(L))
7 L = [4, 5, 6]
8 print(id(L))
9 L.append(8)
10 print(id(L))
11 L = []
12 print(id(L))
```

Sumário

- Listas e tuplas são uma forma de armazenar dados de modo que seja natural iterar sobre eles.
- Tuplas são imutáveis (como as strings)
  - Tuplas são úteis quando você tem objetos que não precisam mudar: (latitude, longitude), (página 1, linha 2).
- Listas são mutáveis.
  - Podem ser modificadas mudando um elemento em índice existente
  - Podem ser modificadas adicionando elementos no fim
  - Vamos ver mais formas de modificar elas em breve
  - São úteis em situações dinâmicas (lista diária de músicas mais tocadas, ou filmes mais recentemente assistidos)