Manipulação de Listas

Instituto de Computação

Leonardo Murta

leomurta@ic.uff.br

Operações sobre listas

Instituto de computação

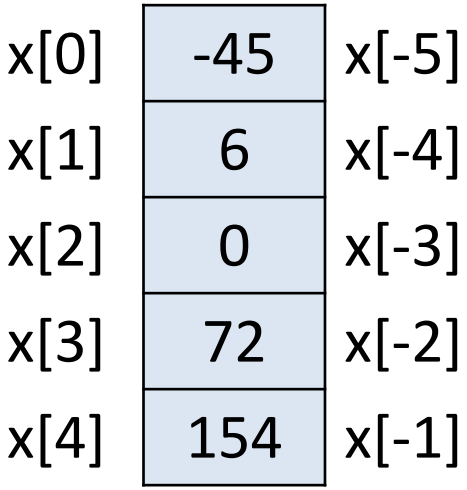
- O Python oferece diversas operações prontas para manipulação de listas
  - Acessar sublistas
  - Saber se um elemento está na lista, concatenar e multiplicar listas
  - Contar o número de ocorrências de um elemento
  - Inserir um elemento em uma determinada posição
  - Encontrar a posição de um elemento
  - Excluir um elemento que está numa determinada posição, ou que tem um determinado valor

Acessar os elementos

Instituto de computação

- Python permite acesso à lista em ordem crescente ou decrescente de posição
  - Última posição é -1
  - Isso vale também para string

```
x = [-45, 6, 0, 72, 154]
print(x[3])                  → 72
print(x[-2])                 → 72
print(x[0] == x[-5])         → True
```

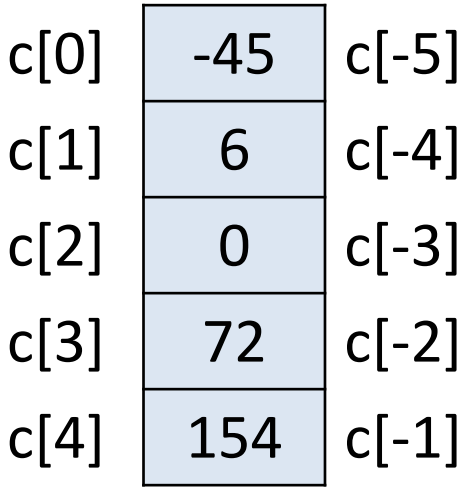


Acessar os elementos

Instituto de computação

- Assim como em string, é possível acessar uma sublista usando [inicio:fim]
- A sublista retornada vai de início (inclusive) até fim – 1
- Se início for omitido, significa zero
- Se fim for omitido, significa len(lista)

```
c = [-45, 6, 0, 72, 154]
print(c[1:3]) → [6, 0]
print(c[:2])  → [-45, 6]
print(c[3:])  → [72, 154]
```



Operadores

Instituto de computação

- Assim como em strings, os seguintes operadores também podem ser usados em listas
  - in
  - +
  - *

in

- elemento **in** lista
  - Retorna True ou False

```
lista = [1, 2, 3, 4]
print(2 in lista) → True
print(0 in lista) → False
print(5 in lista) → False
print(4 in lista) → True
```

+ (Concatenação)

Instituto de computação

- lista1 + lista2
  - Concatena duas listas

```
lista = [1, 2] + [3, 4]
print(lista)  → [1, 2, 3, 4]
lista1 = [1, 2, 3]
lista2 = [5, 6]
lista3 = lista1 + [4] + lista2
print(lista3) → [1, 2, 3, 4, 5, 6]
```

* (Repetição)

Instituto de computação

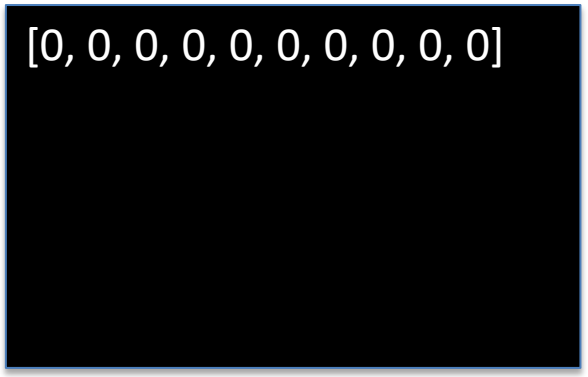
- lista * int
  - Repete os elementos da lista **int** vezes
  - **lista * 5** equivale a **lista + lista + lista + lista + lista**

```
lista1 = [1, 2]
lista2 = lista1 * 2
print(lista2) → [1, 2, 1, 2]
```

* (Repetição)

- Em diversas situações já sabemos de antemão qual será o tamanho da lista
  - Podemos inicializar a lista com o valor zero
  - Isso evita que precisemos usar o *append* depois

```
lista = [0] * 10
print(lista)
```



* (Repetição) Exemplo

Instituto de computação

```
# inicializa vetor de notas com 0
notas = [0] * 3
soma = 0

# preenche as notas, sem usar append
for i in range(len(notas)):
    msg = f'Nota do aluno {i + 1}: '
    notas[i] = float(input(msg))
    soma += notas[i]

print(f'A média da turma é {soma/3}.')
```

Percorrendo uma lista

Instituto de computação

- Os elementos de uma lista podem ser acessados usando uma estrutura de repetição, assim como string

```
lista = [1, 2, 3, 4]
i = 0
while i < len(lista):
    print(lista[i])
    i += 1
```

```
lista = [1, 2, 3, 4]
for i in range(len(lista)):
    print(lista[i])
```

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

Contar Elementos

Instituto de computação

- Contar quantas vezes o elemento de valor 10 aparece na lista

```
lista = [1, 10, 2, 10, 3, 10, 4, 5, 6]
cont = 0
for e in lista:
    if e == 10:
        cont += 1
print(cont)
```

Contar Elementos
Alternativa: *count*

Instituto de computação

```
lista = [1, 10, 2, 10, 3, 10, 4, 5, 6]
cont = lista.count(10)
print(cont)
```

```
3
```

Inserir um elemento

- Inserir o elemento 10 na posição 2 de uma lista

```
lista = [1, 2, 3, 4]
pos, e = 2, 10
temp = []
for i in range(len(lista)):
  if i == pos:
    temp.append(e)
  temp.append(lista[i])
lista = temp
print(lista)
```

```
[1, 2, 10, 3, 4]
```

Inserir um elemento
Alternativa: *insert*

Instituto de computação

```
lista = [1, 2, 3, 4]
pos, e = 2, 10
lista.insert(pos, e)
print(lista)
```

```
[1, 2, 10, 3, 4]
```

Encontrar um elemento

- Encontrar a primeira ocorrência do elemento 10 numa lista e retornar a posição em que ele foi encontrado ou -1 se não estiver na lista

```
lista = [1, 2, 10, 5, 20]
i, e = 0, 10
while i < len(lista) and lista[i] != e:
    i += 1
if i == len(lista):
    i = -1
print(i)
```

Encontrar um elemento
Alternativa: *index*

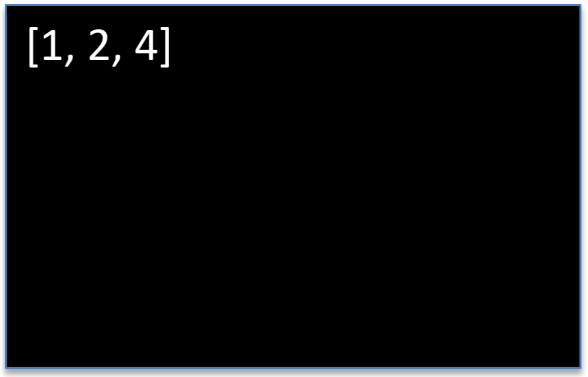
Instituto de computação

```
lista = [1, 2, 10, 5, 20]
e = 10
if e in lista:
    i = lista.index(e)
else:
    i = -1
print(i)
```

Excluir um elemento

- Excluir o elemento da posição 2 de uma lista

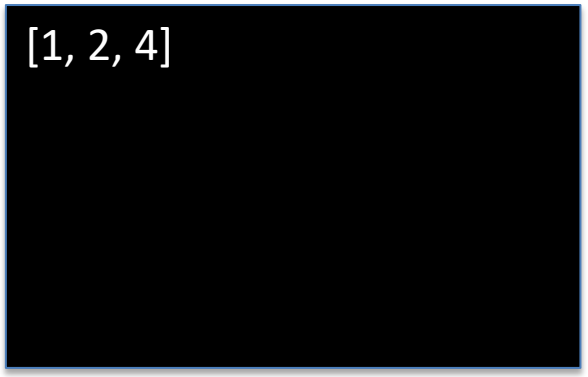
```
lista = [1, 2, 3, 4]
pos = 2
temp = []
for i in range(len(lista)):
  if i != pos:
    temp.append(lista[i])
lista = temp
print(lista)
```



Excluir um elemento
Alternativa: *del*

Instituto de computação

```
lista = [1, 2, 3, 4]
del lista[2]
print(lista)
```



Excluir um elemento
Alternativa: *pop*

Instituto de computação

```
lista = [1, 2, 3, 4]
e = lista.pop(2)
print(e)
print(lista)
```

```
3
[1, 2, 4]
```

Excluir um elemento

- Excluir a primeira ocorrência do elemento de valor 4 da lista

```
lista = [1, 4, 5, 6, 4, 7]
e = 4
if e in lista:
    i = lista.index(e)
    del lista[i]
print(lista)
```

Excluir um elemento
Alternativa: *remove*

Instituto de computação

```
lista = [1, 4, 5, 6, 4, 7]
e = 4
if e in lista:
    lista.remove(e)
print(lista)
```

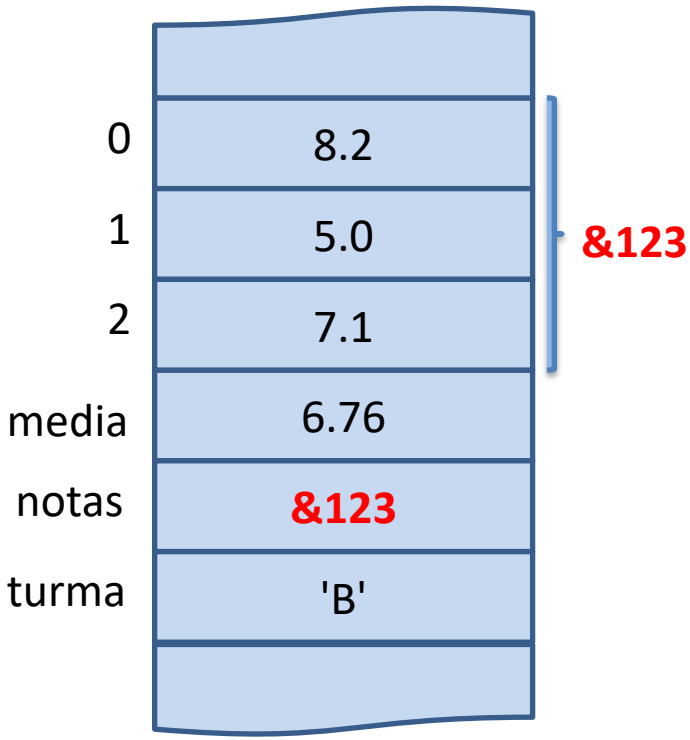
Representação de Listas em Memória

Instituto de computação

Em Phyton

```
notas = [8.2, 5.0, 7.1]
turma = 'B'
media = 0
for i in range(len(notas)):
    media = media + notas[i]
media = media/len(notas)
```

Na Memória



Cópia de listas

Instituto de computação

- O valor de uma variável de lista na verdade é um endereço de memória
- Ao copiar uma lista para outra, o que é feito é copiar o valor do endereço de memória
- Ambas passam a apontar para o mesmo endereço, portanto o que for modificado em uma lista também será modificado na outra

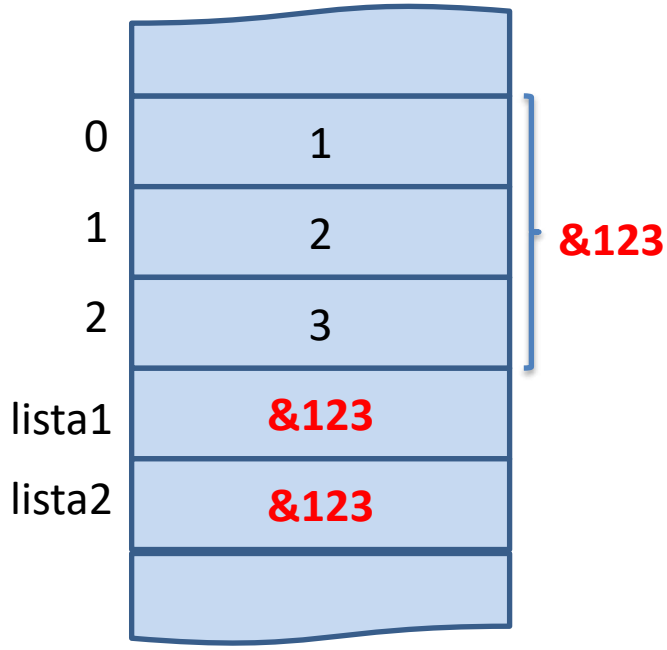
Cópia de Listas

Instituto de computação

Em Phyton

```
lista1 = [1, 2, 3]
lista2 = lista1
```

Na Memória



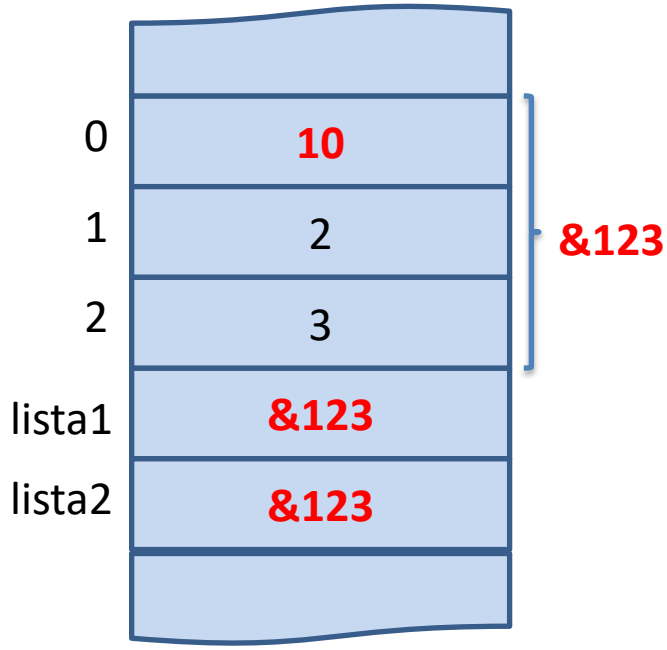
Cópia de Listas

Instituto de computação

Em Phyton

```
lista1 = [1, 2, 3]
lista2 = lista1
lista1[0] = 10
print(lista1) → [10, 2, 3]
print(lista2) → [10, 2, 3]
```

Na Memória



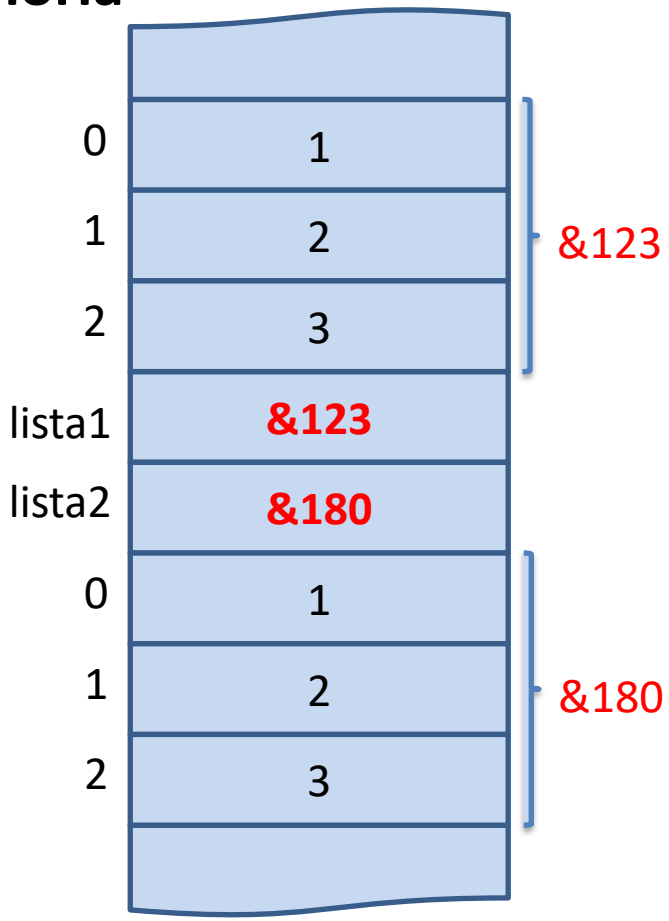
Como evitar isso?

Instituto de computação

Em Phyton

```
lista1 = [1, 2, 3]
lista2 = []
for e in lista1:
    lista2.append(e)
```

Na Memória



Como evitar isso?

Instituto de computação

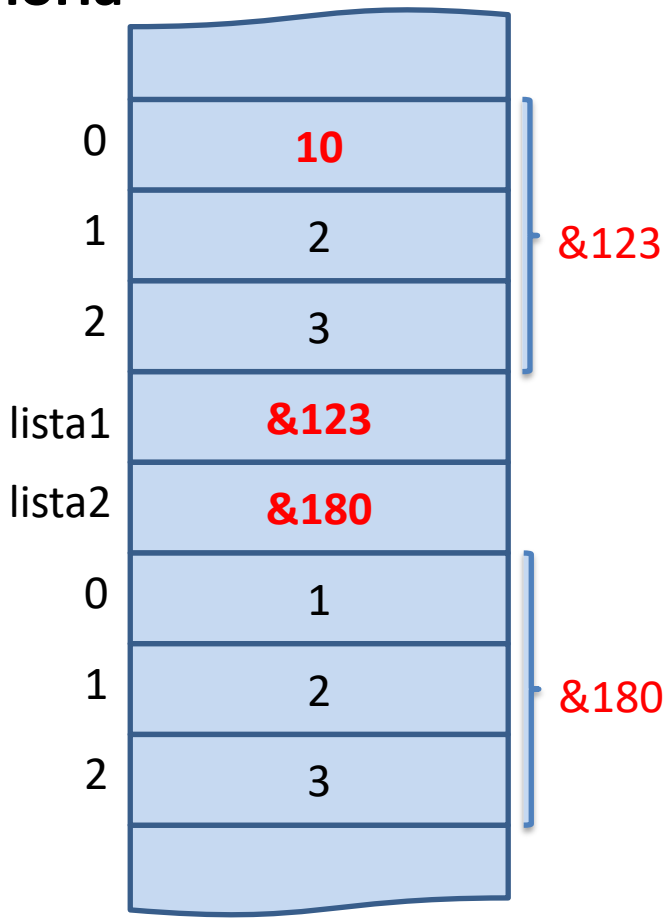
Em Phyton

```
lista1 = [1, 2, 3]
lista2 = []
for e in lista1:
    lista2.append(e)
lista1[0] = 10
```

print(lista1) → [10, 2, 3]

print(lista2) → [1, 2, 3]

Na Memória



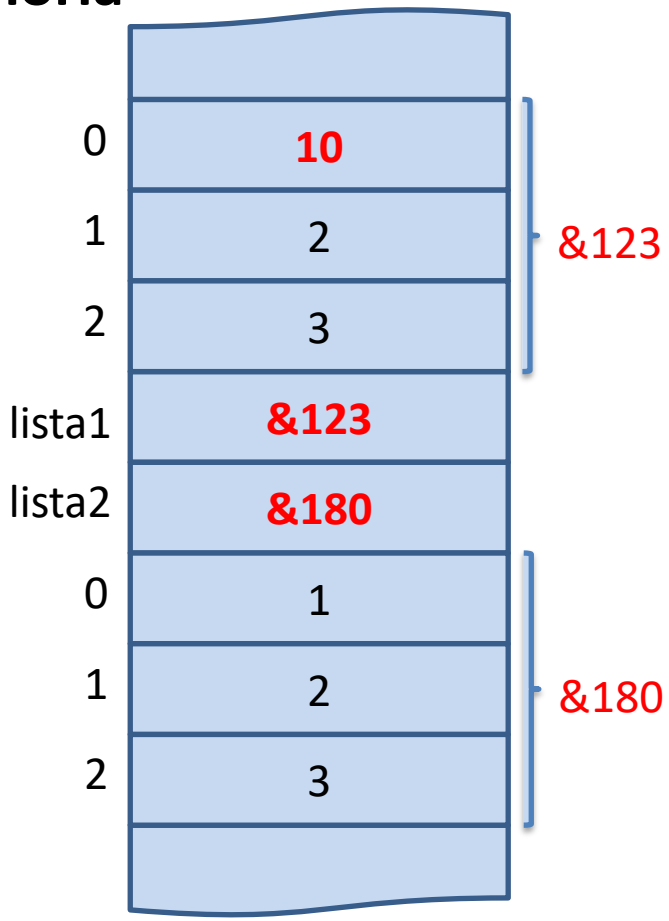
Como evitar isso?
Alternativa: *sublista*

Instituto de computação

Em Phyton

```
lista1 = [1, 2, 3]
lista2 = lista1[0:len(lista1)]
lista1[0] = 10
print(lista1) → [10, 2, 3]
print(lista2) → [1, 2, 3]
```

Na Memória



Exercícios

Instituto de computação

1. Refaça os exercícios da aula passada usando os conhecimentos aprendidos nesta aula.

Exercícios

Instituto de computação

2. Faça um programa que simule um lançamento de dados. Lance o dado 100 vezes e armazene os resultados em um vetor. Depois, monte um outro vetor contendo quantas vezes cada valor foi obtido. Imprima os dois vetores. Use a função random.randint(1,6) para gerar números aleatórios, simulando os lançamentos dos dados.

Exemplo de uma possível saída:
[3, 1, 5, 3, 5, 4, 5, 5, 3, 6]
[1, 0, 3, 1, 4, 1]

Exercícios

Instituto de computação

3. Faça um programa que percorre um vetor e imprime na tela a média dos valores do vetor e o valor mais próximo da média. Exemplo:

Vetor: [2.5, 7.5, 10.0, 4.0]

Média: 6.0

Valor mais próximo da média: 7.5

Exercícios

Instituto de computação

4. Faça um programa que percorre duas listas e intercala os elementos de ambas, formando uma terceira lista. A terceira lista deve começar pelo primeiro elemento da lista menor.

Exemplo:

lista1 = [1, 2, 3, 4]

lista2 = [10, 20, 30, 40, 50, 60]

lista_intercalada = [1, 10, 2, 20, 3, 30, 4, 40, 50, 60]

Referências

- https://docs.python.org/3.8/tutorial/datastructures.html
- Slides feitos em conjunto com Aline Paes e Vanessa Braganholo

Manipulação de Listas

Instituto de Computação

Leonardo Murta

leomurta@ic.uff.br