FACOM3102: Alg e Prog de Computadores

Aula 08 – Listas – Parte 2

Prof. Anilton
Faculdade de Computação
Universidade Federal de Uberlândia

Acessando seus elementos

- As listas suportam acesso a sub-listas:
  - lista[i:j]
    - seleciona a sub-lista dos índices i até j-1
  - lista[i:]
    - seleciona a sub-lista dos índice i até o final
  - lista[:j]
    - seleciona a sub-lista do início até o índice j-1
  - lista[i:j:k]
    - seleciona a sub-lista dos índices i até j-1, indo de k em k
    - i, i+k, i+2k, ..., j-1

Acessando seus elementos

- Selecionando sub-listas de elementos

```
nros = [10, 20, 30, 40, 50]

nros[2:4]
[30, 40]

nros[2:]
[30, 40, 50]

nros[:3]
[10, 20, 30]

nros[:]
[10, 20, 30, 40, 50]

nros[0:5:2]
[10, 30, 50]

nros[::-1]
[50, 40, 30, 20, 10]
```

Acessando seus elementos

- Listas são objetos ***mutáveis***
  - Podemos acessar qualquer elemento ou sequência de elementos e modificá-los;
  - Elementos de uma lista podem possuir qualquer tipo.

```
>>>
>>> lista = [1, 2, 3]
>>> lista[0] = "novo"
>>> lista
['novo', 2, 3]
>>>
>>> lista[2:3] = [10, 20]
>>> lista
['novo', 2, 10, 20]
>>>
```

Acessando seus elementos

- Podemos também associar o valor de cada posição da lista a uma variável sem precisar especificar os índices usando o recurso de ***unpacking*;**
  - Basta definir uma lista de variáveis, entre colchetes, os quais receberão o conteúdo da lista.

```
>>> pontos = [1, 2, 3]
>>> pontos
[1, 2, 3]
>>>
>>> [x, y, z] = pontos
>>> x
1
>>> y
2
>>> z
3
>>>
```

Concatenação e repetição: listas

- Podemos unir ou concatenar duas listas para formar uma nova utilizando o operador de **+**
- Também podemos criar uma lista a partir da repetição de outra utilizando o operador de *****

```
>>>
>>> lista1 = [1, 2, 3]
>>> lista2 = [4, 5, 6]
>>> lista1 + lista2
[1, 2, 3, 4, 5, 6]
>>> lista3 = lista1 + lista2
>>> lista3
[1, 2, 3, 4, 5, 6]
>>>
>>> lista = 3 * lista1
>>> lista
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>>
```

Removendo elementos da lista

- Podemos também remover um ou mais elementos de uma lista de duas maneiras:
  - Usando o operador **del**
  - Atribuindo uma lista vazia àquela posição

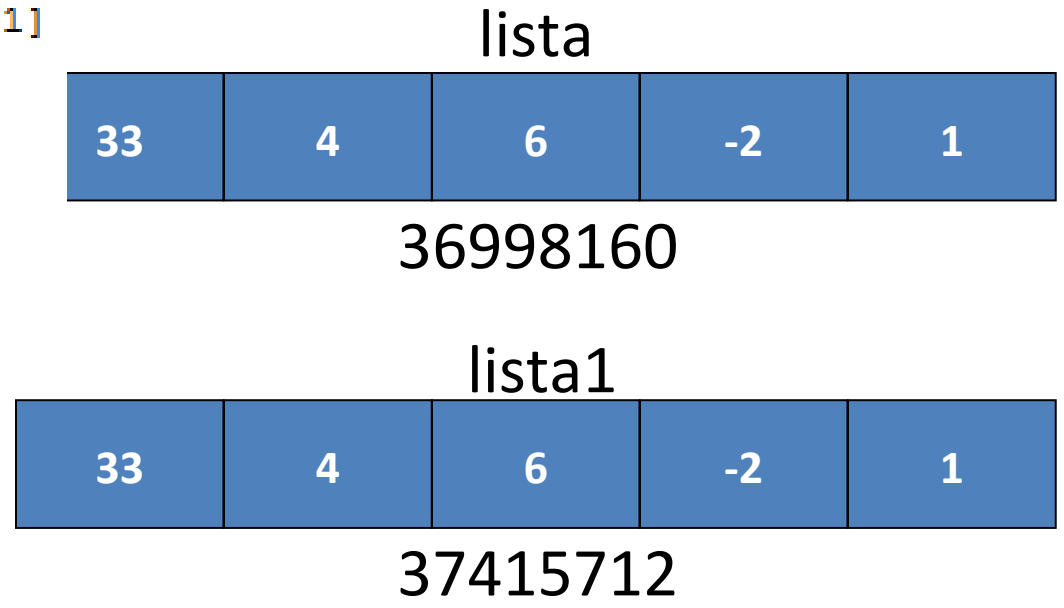
```
>>>
>>> lista = [1, 2, 3, 4, 5, 6]
>>> lista
[1, 2, 3, 4, 5, 6]
>>>
>>> del lista[1]
>>> lista
[1, 3, 4, 5, 6]
>>>
```

```
>>>
>>> lista = [1, 2, 3, 4, 5, 6]
>>> lista
[1, 2, 3, 4, 5, 6]
>>>
>>> lista[1:2] = []
>>> lista
[1, 3, 4, 5, 6]
>>>
```

Copiando uma lista

- Para copiar uma lista, usa-se **[:]** na operação de atribuição com a nova variável a que se quer copiar.

```
>>>
>>> lista = [33, 4, 6, -2, 1]
>>> lista
[33, 4, 6, -2, 1]
>>> lista1 = lista[:]
>>> lista1
[33, 4, 6, -2, 1]
>>>
>>> id(lista)
36998160
>>> id(lista1)
37415712
>>>
```



id – retorna o endereço do objeto na memória.

Procurando um elemento na lista

- O operador **in** permite verificar se um determinado elemento está presente ou não em uma lista.

```
lista = [1,2,3,4,5,6]

if 2 in lista:
    print("Valor existe na lista!")
else:
    print("Valor não existe na lista")
```

Métodos sobre listas

- Existe alguns métodos pré-definidos que podem ser usados em listas, forma geral de uso dos métodos:
  - **lista.nome-método()**
- Alguns métodos
  - sort(): ordena os elementos da lista
  - append(x): insere um elemento **x** no final da lista
  - insert(pos,x): insere um elemento **x** na posição **pos**
  - remove(x): remove o elemento **x** da lista
  - pop(pos): remove e retorna o elemento da posição **pos**

Métodos sobre listas

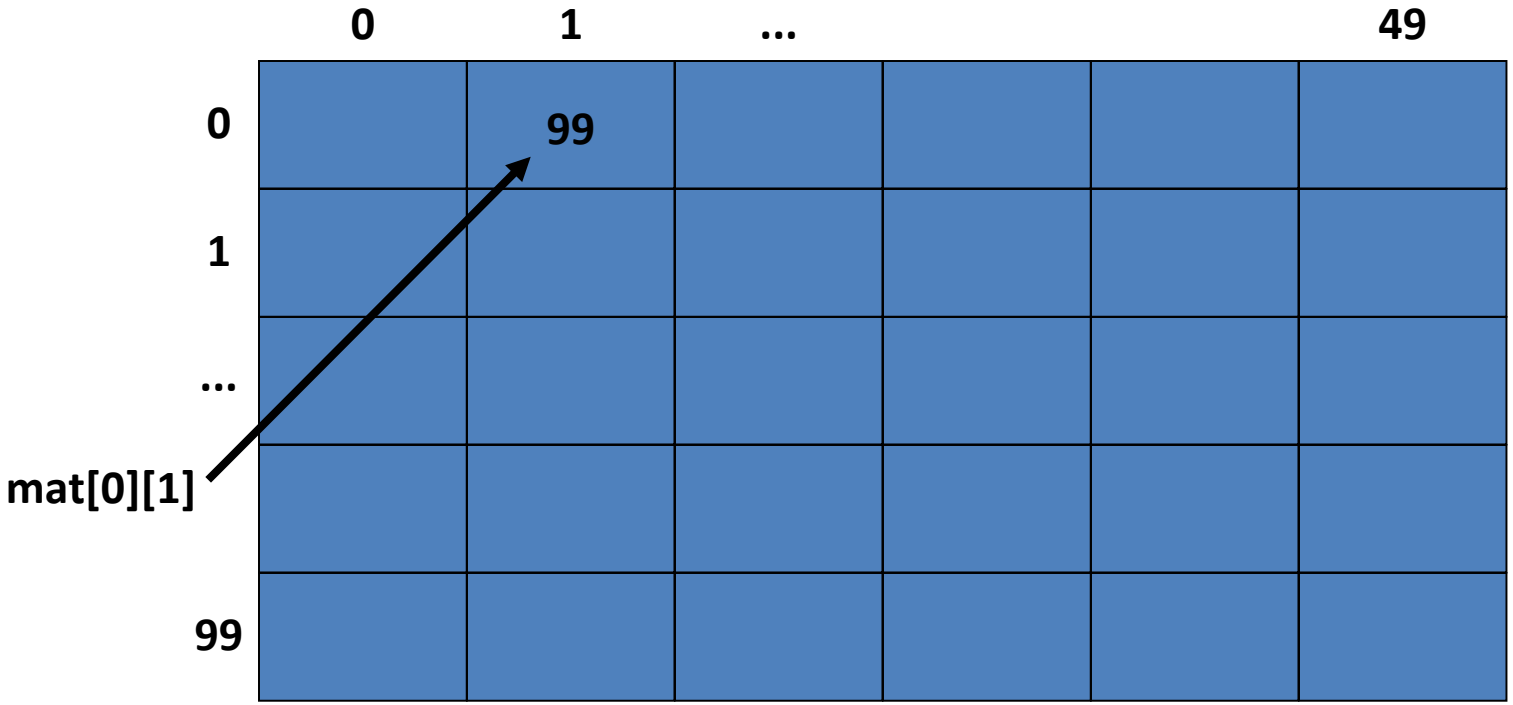
- Exemplos

```
>>>
>>> lista = list(range(5))
>>> lista
[0, 1, 2, 3, 4]
>>>
>>> lista.append(5)
>>> lista
[0, 1, 2, 3, 4, 5]
>>>
>>> lista.insert(0,-1)
>>> lista
[-1, 0, 1, 2, 3, 4, 5]
>>>
>>> lista.insert(2,"meio")
>>> lista
[-1, 0, 'meio', 1, 2, 3, 4, 5]
>>>
>>> lista.remove(1)
>>> lista
[-1, 0, 'meio', 2, 3, 4, 5]
```

```
>>>
>>> lista = [33, 4, 6, -2, 1]
>>> lista
[33, 4, 6, -2, 1]
>>>
>>> lista.sort()
>>> lista
[-2, 1, 4, 6, 33]
>>>
>>> x = lista.pop(2)
>>> x
4
>>> lista
[-2, 1, 6, 33]
>>>
```

Lista aninhadas

- Uma lista pode armazenar qualquer tipo de dado
  - As listas podem inclusive conter outras listas
  - Podemos assim representar tabelas ou **matrizes**



Lista aninhadas

- Para criar uma lista aninhada, basta definir cada elemento como uma nova lista, como?
  - Os elementos são acessados especificando um par de colchetes e índice para cada dimensão da lista;
  - A numeração começa sempre do zero

```
>>>
>>> matriz = [[1,2,3],[4,5,6],[7,8,9]]
>>> matriz
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>>
>>> matriz[0][1]
2
>>> matriz[0][1] = "oi"
>>> matriz
[[1, 'oi', 3], [4, 5, 6], [7, 8, 9]]
>>>
```

	0	1	2
0	1	‘oi’	3
1	4	5	6
2	7	8	9

Lista aninhadas

- Podemos incluir novas linhas e colunas em cada lista:

```
>>>
>>> matriz = [[1,2,3],[4,5,6],[7,8,9]]
>>> matriz.append([10,11,12])
>>> matriz
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
>>>
```

- Podemos remover elementos: a linha inteira é removida ou elemento da linha e coluna é removido:

```
>>> matriz
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
>>>
>>>
>>> del matriz[0][0]
>>> matriz
[[2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
>>>
>>> del matriz[1]
>>> matriz
[[2, 3], [7, 8, 9], [10, 11, 12]]
>>>
```

Lista aninhadas

- Todas as operações em listas aninhadas consideram o fato de que temos agora uma lista dentro de outra:

```
#impressão
matriz = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
for linha in matriz:
    for elem in linha:
        print(elem)

#soma
matriz = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
soma = 0
for linha in matriz:
    for elem in linha:
        soma = soma + elem

print("Soma = ",soma)
```

Alguma dúvida?

Prof. Anilton
Bloco B – Sala 133
anilton@ufu.br