Tópicos avançados em Informática I
Programação Python

Estruturas de dados compostas
- Parte 1 -

Profa. Ana Paula Canal
Universidade Franciscana

Sumário

- Introdução
- Listas
- Listas: *slicing* ou fatiamento
- Percorrendo uma lista

Estruturas compostas

- A funcionalidade destas estruturas é manter os dados organizados e permitir um fácil acesso a estes dados

- Vetores
- Matrizes
- Registros

- No Python as estruturas que vamos estudar são: listas, tuplas, conjuntos e dicionários

Listas

- Uma lista armazena uma coleção de dados sobre os quais é é possível executar uma série de operações.
- Pode ser formada por *0*, *n* ou infinitos itens.
- Estes itens podem ser inteiros, números reais, caracteres ou qualquer outro tipo, inclusive outra lista
- Ao atribuir a uma variável uma sequência de elementos, separados por vírgula e limitados por colchetes faremos a definição de uma lista

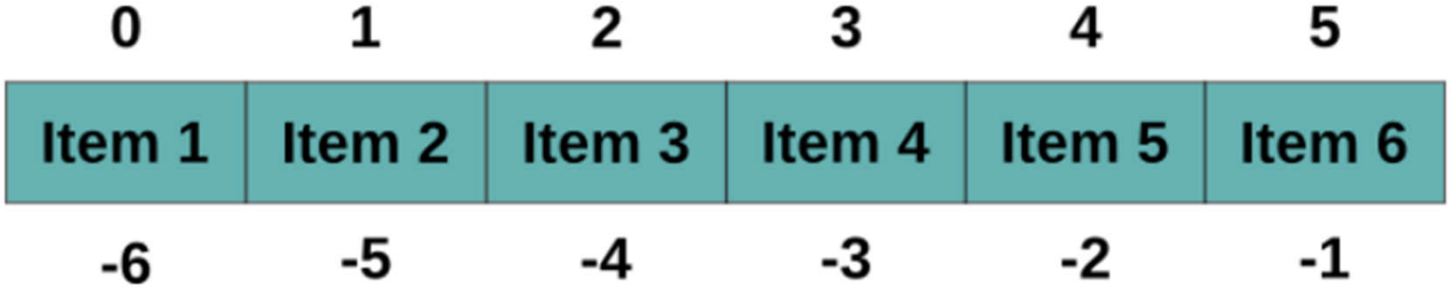
Listas

```
>>> lista1 = []
>>> lista2 = [1,2,9.5,'Linguagem','Python',[100,200,300]]
>>> lista1
[]
>>> lista2
[1, 2, 9.5, 'Linguagem', 'Python', [100, 200, 300]]
>>>
```

- O acesso aos elementos de uma lista é realizado pela associação do nome da lista com o índice
- Um número inteiro indica a posição do item na lista

Listas

- O acesso aos elementos de uma lista é realizado pela associação do nome da lista com o índice
- Um número inteiro indica a posição do item na lista
- O índice 0 corresponde a primeira posição, o índice 1 o segundo item e assim sucessivamente.
- Os índices negativos, permitem a localização dos itens na ordem contrária, -1 o último, -2 o penúltimo
- O primeiro elemento sempre terá o índice igual a 0
- O último elemento terá índice igual a -1



Listas

```
>>> lista = [1,2,9.5,'Linguagem', 'Python', [100,200,300]]
>>> lista[0]
1
>>> lista[3]
'Linguagem'
>>> lista[-1]
[100, 200, 300]
>>> lista[-2]
'Python'
>>> lista[6]
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    lista[6]
IndexError: list index out of range
>>> lista[-1][1]
200
>>>
```

Listas: *slicing* ou fatiamento

- Enumera um espaço entre os elementos da lista (uma fatia da lista)
- [*n* : *m* : *x*]
  - n e m correspondem ao intervalo de índices
  - O início do intervalo é fechado
  - O final é aberto, pois representa o elemento da posição informada menos um
  - O valor de x, quando informado, indica o *step*, o tamanho do passo que será aplicado no intervalo
- Se o valor de *n* for omitido o *slicing* iniciará com o primeiro item da lista
- Se o valor de *m* for omitido a lista será considerada até o seu final

Listas: *slicing* ou fatiamento

```
>>> lista = [1,2,9.5,'Linguagem', 'Python', [100,200,300]]
>>> lista[0:3]
[1, 2, 9.5]
>>> lista[1:-1]
[2, 9.5, 'Linguagem', 'Python']
>>> lista[1:]
[2, 9.5, 'Linguagem', 'Python', [100, 200, 300]]
>>> lista[:4]
[1, 2, 9.5, 'Linguagem']
>>> lista[:]
[1, 2, 9.5, 'Linguagem', 'Python', [100, 200, 300]]
>>> lista[1:5:2]
[2, 'Linguagem']
>>> lista[:-1:2]
[1, 9.5, 'Python']
>>> lista[::2]
[1, 9.5, 'Python']
>>> lista[::-1]
[[100, 200, 300], 'Python', 'Linguagem', 9.5, 2, 1]
```

Percorrendo uma lista

- Percorremos as listas em busca de elementos e/ou para a realização de operações sobre os elementos.

```
>>> lista = [1,2,9.5,'Linguagem','Python',[100,200,300]]
>>> i = 0
>>> while (i < 6):
        print(lista[i])
        i += 1


1
2
9.5
Linguagem
Python
[100, 200, 300]
```

Percorrendo uma lista

- A função **len** retorna o tamanho desta lista

```
>>> lista = [1,2,9.5,'Linguagem','Python',[100,200,300]]
>>> i = 0
>>> while (i < len(lista)):
        print(lista[i])
        i += 1


1
2
9.5
Linguagem
Python
[100, 200, 300]
```

Operador **in**

- O operador lógico **in** verifica se um determinado item está contido na lista

```
>>> lista = [1,2,9.5,'Linguagem','Python',[100,200,300]]
>>> 1 in lista
True
>>> 'Linguagem' in lista
True
>>> 'Python ' in lista
False
>>> 100 in lista
False
>>> 100 in lista[5]
True
```

Percorrendo uma lista

- Com a combinação do **for** e do **in** elimina-se a variável que faz o controle do laço

- Para cada item pertencente a lista aplica-se a operação desejada

```
>>> lista = [1,2,9.5,'Linguagem','Python',[100,200,300]]
>>> for item in lista:
        print(item)


1
2
9.5
Linguagem
Python
[100, 200, 300]
```

Listas

- Listas são mutáveis, isto é, os seus itens podem ser alterados
- Para atualizar um item deve-se informar a posição correspondente entre colchetes e fazer uma atribuição
- O recurso de fatiamento também pode ser usado na atualização de listas. Informa-se o nome da lista e entre colchetes a fatia do intervalo que será substituído
- Para inserir novos elementos na lista, define-se a posição com a utilização de uma fatia e insere o elemento na posição determinada pelos índices. O valor atribuído deve ser uma sequência (lista ou *string*)
- Para a remoção de elementos atribui-se uma lista vazia no intervalo definido. O aconselhável é utilizar o comando **del**, que remove o item da lista, de acordo com o índice informado. Se for informado um intervalo de índices, a remoção será de todos os itens inclusos no intervalo

Listas

- Uma concatenação pode ser realizada ao ser utilizado o operador +
- O operador * caracteriza a operação de repetição nos elementos da lista
- É aconselhável fazer a inicialização de listas antes de utilizá-las, pois o Python não permite a atribuição de um valor a uma posição que não existe na lista
- Se você sabe a quantidade inicial de itens de uma lista, mas não sabe qual o tipo de dados que será armazenado, uma boa possibilidade de escolha é inicializar a lista usando o valor *None*

Listas

```
>>> lista = [1,2,9.5,'Linguagem','Python',[100,200,300]]
>>> lista[2] = 4.2
>>> lista
[1, 2, 4.2, 'Linguagem', 'Python', [100, 200, 300]]
>>> lista[0:2] = [10, 20]
>>> lista
[10, 20, 4.2, 'Linguagem', 'Python', [100, 200, 300]]
>>> lista[3:3] = [3.14159]
>>> lista
[10, 20, 4.2, 3.14159, 'Linguagem', 'Python', [100, 200, 300]]
>>> lista[2:4] = []
>>> lista
[10, 20, 'Linguagem', 'Python', [100, 200, 300]]
>>> del lista[2]
>>> lista
[10, 20, 'Python', [100, 200, 300]]
>>> lista2 = [5]*5
>>> lista2
[5, 5, 5, 5, 5]
>>> lista = lista + lista2
>>> lista
[10, 20, 'Python', [100, 200, 300], 5, 5, 5, 5, 5]
>>> lista += lista + [4]*4
>>> lista
[10, 20, 'Python', [100, 200, 300], 5, 5, 5, 5, 5, 10, 20, 'Python',
[100, 200, 300], 5, 5, 5, 5, 5, 4, 4, 4, 4]
>>> del lista[9:]
>>> lista
[10, 20, 'Python', [100, 200, 300], 5, 5, 5, 5, 5]
>>> lista += [4]*4
>>> lista
[10, 20, 'Python', [100, 200, 300], 5, 5, 5, 5, 5, 4, 4, 4, 4]
>>> lista3 = [0]*10
>>> lista3
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> lista4 = [None]*5
>>> lista4
[None, None, None, None, None]
```

Listas

Função	Descrição
len(lista)	Retorna o tamanho da lista, isto é, o número de elementos que a lista contém
max(lista)	Retorna o maior elemento da lista
min(lista)	Retorna o menor elemento da lista
list(string)	Converte uma *string* em lista

Listas

```
>>> lista = [1,2,9,8,7,5,6,3]
>>> lista
[1, 2, 9, 8, 7, 5, 6, 3]
>>> len(lista)
8
>>> max(lista)
9
>>> min(lista)
1
>>> lista2 = list('Linguagem')
>>> lista2
['L', 'i', 'n', 'g', 'u', 'a', 'g', 'e', 'm']
```

Listas

- Duas ou mais listas somente são consideradas iguais quando possuem o mesmo número de elementos e todos os elementos possuem valores correspondentes em suas posições
- Quando se define uma variável do tipo *list*, está se fazendo uma referência para um valor do tipo lista (*list*)
- Quando se atribui uma variável a outra variável do tipo *list*, está se criando uma nova referência, que aponta para o mesmo endereço, e não uma nova lista
- A classe list possui métodos que facilitam o trabalho com as listas
- Os métodos são chamados com o uso do . (ponto)

Listas

Método	Descrição
append(item)	Adiciona o item no fim da lista
clear()	Remove todos os itens da lista
copy()	Copia os elementos da lista para outra lista
count(item)	Conta o número de vezes que o item aparece na lista
extend(lista_aux)	Insere os itens de lista_aux ao final da lista
index(item)	Retorna o índice do item na primeira ocorrência na lista Se item não existe na lista retorna uma mensagem de erro
insert(índice, item)	Insere o item na posição definida pelo índice
pop(índice)	Remove o item na posição definida pelo índice Se o índice não for informado removerá o último item
remove(item)	Remove o item na primeira ocorrência na lista Se item não existe o retorna uma mensagem de erro
reverse	Inverte a ordem dos elementos na lista
sort(reverse = False)	Ordena a lista, que por default será em ordem crescente reverse = True ordena a lista em ordem decrescente

Listas

```
>>> lista = [1,2,9,8,7,5,6,3]
>>> lista
[1, 2, 9, 8, 7, 5, 6, 3]
>>> lista.append(10)
>>> lista
[1, 2, 9, 8, 7, 5, 6, 3, 10]
>>> lista.count(8)
1
>>> lista_aux = [5,7,9]
>>> lista.extend(lista_aux)
>>> lista
[1, 2, 9, 8, 7, 5, 6, 3, 10, 5, 7, 9]
>>> lista.insert(2,4)
>>> lista
[1, 2, 4, 9, 8, 7, 5, 6, 3, 10, 5, 7, 9]
>>> lista.pop(7)
6
>>> lista
[1, 2, 4, 9, 8, 7, 5, 3, 10, 5, 7, 9]
>>> lista.pop()
9
>>> lista
[1, 2, 4, 9, 8, 7, 5, 3, 10, 5, 7]
>>> lista.reverse()
>>> lista
[7, 5, 10, 3, 5, 7, 8, 9, 4, 2, 1]
>>> lista.sort()
>>> lista
[1, 2, 3, 4, 5, 5, 7, 7, 8, 9, 10]
>>> lista.sort(reverse=True)
>>> lista
[10, 9, 8, 7, 7, 5, 5, 4, 3, 2, 1]
>>> lista2 = lista.copy()
>>> lista2
[10, 9, 8, 7, 7, 5, 5, 4, 3, 2, 1]
>>> lista.clear()
>>> lista
[]
```