MC-102 — Aula 10
Listas

Prof. Luiz F. Bittencourt

Turmas QR

Instituto de Computação – Unicamp

2019

Conteúdo adaptado de slides fornecidos pelo Prof. Eduardo Xavier.

Roteiro

Listas

- Listas são construções de linguagens de programação que servem para armazenar vários dados de forma simplificada.

Listas

- Suponha que desejamos guardar notas de alunos.
- Com o que sabemos, como armazenaríamos 3 notas?

```
print("Entre com a nota 1")
nota1=float(input())
print("Entre com a nota 2")
nota2=float(input())
print("Entre com a nota 3")
nota3=float(input())
```

Listas

- Com o que sabemos, como armazenaríamos 100 notas?

```
print("Entre com a nota 1")
nota1=float(input())
print("Entre com a nota 2")
nota2=float(input())
print("Entre com a nota 3")
nota3=float(input())
...
print("Entre com a nota 100")
nota100=float(input())
```

- Criar 100 variáveis distintas não é uma solução elegante para este problema.

Listas — Definição

- Coleção de valores referenciados por um **identificador único**.
- Características:
  - Acesso por meio de um índice inteiro.
  - Listas podem ser modificadas.
  - Pode-se incluir e remover itens de listas.

Declaração de uma lista

Declara-se uma lista colocando entre colchetes uma sequência de dados separados por vírgula:

$$\text{identificador} = [\text{dado}_1, \text{dado}_2, \ldots, \text{dado}_n]$$

Exemplo de listas

Exemplo de listas:

- Uma lista de inteiros.

```
x = [2, 45, 12, 9.78, -2]
```

- Listas podem conter dados de tipos diferentes.

```
x = [2, "qwerty" ,45.99087, 0, "a"]
```

- Listas podem conter outras listas.

```
x = [2, [4,5], [9]]
```

- Ou podem não conter nada. Neste caso **[]** indica a lista vazia.

```
x = []
```

Usando uma lista

- Pode-se acessar uma determinada posição da lista utilizando-se um índice de valor inteiro.
- Sendo n o tamanho da lista, os índices válidos para ela vão de 0 até $n - 1$.
  - A primeira posição da lista tem índice 0.
  - A última posição da lista tem índice $n - 1$.
- A sintaxe para acesso de uma determinada posição é:
  - identificador[**posição**];

Um elemento de uma lista em uma posição específica tem o mesmo comportamento que uma variável simples.

```
>>> notas= [4.5, 8.6, 9, 7.8, 7]
>>> notas[1]+2
10.6
>>> notas[3]=0.4
>>> notas
[4.5, 8.6, 9, 0.4, 7]
```

Usando uma lista

- Você deve usar valores inteiros como índice para acessar uma posição da lista.
- O valor pode ser inclusive uma variável inteira.

```
>>> for i in range(5):
...   print(notas[i])
4.5
8.6
9
7.8
7
```

- Quais valores estarão armazenados em cada posição da lista após a execução deste código abaixo?

```
l = [0,0,0,0,0,0,0,0,0,0]
for i in range(10):
   l[i] = 5*i
print(l)
```

Listas - índices

- Índices negativos se referem a lista da direita para a esquerda:

```
>>> notas = [4.5, 8.6, 9, 7.8, 7]
>>> notas[-1]
7
```

- Ocorre um erro se tentarmos acessar uma posição da lista que não existe.

```
>>> notas[100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range

>>>> notas[-6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Listas - índices

- Listas em Python suportam uma operação conhecida como **slicing**, que consiste em obter uma sub-lista contendo os elementos de uma posição inicial até uma posição final de uma lista.
- O **slicing** em Python é obtido assim:
  - identificador`[ind1:ind2]`

  e o resultado é uma sub-lista com os elementos de $ind1$ até $ind2 - 1$.

```
>>> notas= [4.5, 8.6, 9, 7.8, 7]
>>> notas[1:4]
[8.6, 9, 7.8]
>>>
```

Listas - funções em listas

- A função **len(lista)** retorna o número de itens na lista.

```
>>> x=[16,5,4,4,7,9]
>>> len(x)
6
```

- É muito comum usar a função **len** junto com o laço **for** para percorrer todas as posições de uma lista:

```
x=[6,5,6,4,7,9]
for i in range(len(x)):
   print(x[i])
```

Listas - **for**

- Lembre-se que o **for** na verdade faz a variável de controle assumir todos os valores de uma lista. Assim:

```
for i in range(len(x)):
    print(x[i])
```

pode ser implementado como:

```
for a in x:
    print(x)
```

Listas - **append**

- Uma operação importante é acrescentar um item no final de uma lista. Isto é feito pela função **append**.

```
lista.append(item)
```

```
>>> x=[6,5]
>>> x.append(98)
>>> x
[6, 5, 98]
```

- Note o formato diferente da função **append**. A lista que será modificada aparece antes, seguida de um ponto, seguida do **append** com o item a ser incluído como argumento. Formalmente, este tipo de função é chamada de `método`.

Preenchendo uma lista

- A combinação de uma lista vazia que vai sofrendo "appends" permite ler dados e preencher uma lista com esses dados:

```
x=[]
n=int(input("Entre com o numero de notas:"))
for i in range(n):
    dado = float(input("Entre com a nota " + str(i) +":"))
    x.append(dado)

print(x)
```

Listas - funções em listas

- A operação de soma em listas gera uma nova lista que é o resultado de "grudar" **lista2** ao final da **lista1**. Isto é conhecido como **concatenação** de listas.

lista1 + lista2

```
>>> lista1=[1,2,4]
>>> lista2=[27,28,29,30,33]
>>> x=lista1+lista2
>>> x
[1, 2, 4, 27, 28, 29, 30, 33]
>>> lista1
[1, 2, 4]
>>> lista2
[27, 28, 29, 30, 33]
```

- Veja que a operação de concatenação não modifica as listas originais.

Listas - funções em listas

- O operador "*" faz repetições da concatenação:

```
>>> x=[1,2]
>>> y=4*x
>>> y
[1, 2, 1, 2, 1, 2, 1, 2]
```

- O resultado da operação do exemplo, é o mesmo que somar (concatenar) 4 vezes a lista **x**.

Listas - outros métodos em listas

- Além do **append**, há outros métodos que modificam listas.
- **lista.insert(índice,dado)** insere na lista o `dado` **antes** do elemento da posição `índice` na lista atual.

```
>>> x=[40,30,10,40]
>>> x.insert(1,99)
>>> x
[40, 99, 30, 10, 40]
```

- O comando **del lista[posição]** remove da lista o item da posição especificada.

```
>>> del x[2]
>>> x
[40, 99, 10, 40]
```

- Também podemos remover um item da lista utilizando o método **remove**.

```
>>> x.remove(10)
>>> x
[40, 99, 40]
```

Exemplo: Produto Interno de dois vetores

- Ler dois vetores de dimensão 5 e computar o produto interno destes.

Exemplo: Produto Interno de dois vetores

- Abaixo temos o código para ler dois vetores de dimensão 5.
- Inicializamos os dois vetores como listas vazias e fazemos **appends** com os dados lidos.

```
x=[]
y=[]

for i in range(5):
    x.append(float(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 1:")
print()
for i in range(5):
    y.append(float(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 2:")

#calculando o produto interno
...
```

Exemplo: Produto Interno de dois vetores

- Abaixo temos a parte do código para computar o produto interno dos vetores.

```
#calculando o produto interno
resultado = 0.0;
for i in range(5):
    resultado = resultado +  x[i]*y[i]

print("O produto interno é:",resultado);
```

Exemplo: Produto Interno de dois vetores - versão 1

- Agora o código completo.

```
x=[]
y=[]

for i in range(5):
    x.append(float(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 1:")))
print()
for i in range(5):
    y.append(float(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 2:")))

#calculando o produto interno
resultado = 0.0;
for i in range(5):
    resultado = resultado +  x[i]*y[i]

print("O produto interno é:",resultado);
```

Exemplo: Produto Interno de dois vetores - versão 2

- Como sabemos os tamanhos dos vetores, podemos criá-los com zeros na inicialização:

```
x=5*[0]
y=5*[0]

for i in range(5):
    x[i]=float(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 1:"))
print()
for i in range(5):
    y[i]=float(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 2:"))

#calculando o produto interno
resultado = 0.0;
for i in range(5):
    resultado = resultado +  x[i]*y[i]

print("O produto interno é:",resultado);
```

Exemplo: Elementos Iguais

- Ler dois vetores com 5 inteiros cada.
- Checar quais elementos do segundo vetor são iguais a algum elemento do primeiro vetor.
- Se não houver elementos em comum, o programa deve informar isso.

Exemplo: Elementos Iguais

- Abaixo está o código que faz a leitura de dois vetores.

```
x=[]
y=[]

for i in range(5):
    x.append(int(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 1:")))
print()
for i in range(5):
    y.append(int(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 2:")))

...
```

Exemplo: Elementos Iguais

- Para cada elemento de **x** testamos todos os outros elementos de **y** para saber se são iguais.
- Usamos uma variável indicadora para decidir, ao final dos laços encaixados, se os vetores possuem ou não um elemento em comum.

```
...
umEmComum = False   #Supondo que não há elementos comuns
for i in range(len(x)):
    for j in range(len(y)):
        if(x[i]==y[j]):
            umEmComum=True  # Descobrimos que há elemento comum
            print("Elemento da pos. "+str(i)+" igual ao elemento da pos. "+str(j))

if not umEmComum:
    print("Nenhum elemento em comum")
```

Exemplo: Elementos Iguais - versão 1

- Código completo abaixo.

```
x=[]
y=[]

for i in range(5):
    x.append(int(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 1:")))
print()
for i in range(5):
    y.append(int(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 2:")))

umEmComum = False  #Assumimos que não hajam elementos comuns

for i in range(len(x)):
    for j in range(len(y)):
        if(x[i]==y[j]):
            umEmComum=True  # Descobrimos que há elemento comum
            print("Elemento da pos. "+str(i)+" igual ao elemento da pos. "+str(j))

if not umEmComum:
    print("Nenhum elemento em comum")
```

Exemplo: Elementos Iguais - versão 2

- Temos o mesmo programa anterior, mas agora nos laços percorremos as listas diretamente com seus valores, ao invés de se utilizar índices para as listas.

```
x=[]
y=[]

for i in range(5):
    x.append(int(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 1:")))
print()
for i in range(5):
    y.append(int(input("Entre com o "+str(i+1)+"-ésimo valor para vetor 2:")))

umEmComum = False

for a in x:  #a e b são os valores das listas
    for b in y:
        if(a==b):
            umEmComum=True

if not umEmComum:
    print("Nenhum elemento em comum")
```

Informações Extras: Inicialização de uma lista

- Em algumas situações é necessário declarar e já atribuir um conjunto de valores constantes para uma lista.
- Em Python podemos usar o que é conhecido como **compreensão de listas**.
- Dentro da lista incluímos uma construção com um laço que gerará valores iniciais para a lista.
- Exemplo: inicializar uma lista com 5 zeros:

```
>>> x = [ 0 for i in range(5)]
>>> x
[0, 0, 0, 0, 0]
```

- Exemplo: inicializar uma lista com os 5 primeiros números pares:

```
>>> x = [ 2*i for i in range(5)]
>>> x
[0, 2, 4, 6, 8]
```

Exercício

- Escreva um programa que lê 10 números inteiros e os salva em uma lista. Em seguida o programa deve encontrar a posição do maior elemento da lista e imprimir esta posição.

Exercício

- Escreva um programa que lê 10 números ponto flutuante e os salva em uma lista. Em seguida o programa deve calcular a média dos valores armazenados na lista e imprimir este valor.

Exercício

- Escreva um programa que lê 10 números inteiros e os salva em uma lista. Em seguida o programa deve ler um outro número inteiro C. O programa deve então encontrar dois números de posições distintas da lista cuja multiplicação seja C e imprimí-los. Caso não existam tais números, o programa deve informar isto.
- Exemplo: Se $l = [2, 4, 5, -10, 7]$ e $C = 35$, então o programa deve imprimir "5 e 7". Se $C = -1$ então o programa deve imprimir "Não existem tais números".