MC-102 — Aula 13
Listas

Instituto de Computação – Unicamp

15 de Setembro de 2015

Roteiro

Listas

Como armazenar 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

Como armazenar 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())
```

Listas — Definição

- Coleção de valores referenciados por um nome em comum.
- 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 lista de dados separados por vírgula:

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

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. [] indica a lista vazia.

```
x = []
```

Usando uma lista

Pode-se acessar uma determinada posição da lista utilizando-se um valor inteiro.

identificador [<posição>]

- A primeira posição de uma lista tem índice 0.
- A última posição de uma lista tem índice *<tamanho atual da lista>* - 1.

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

identificador [<posição>]

- Você pode usar valores inteiros para acessar uma posição do vetor.
- O valor pode ser inclusive uma variável.

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

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 ao tentar acessar uma posição da lista que não existe.

```
>>> notas[7]
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 - 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 da 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)
```

- Note que a variável **a** tem uma referência para o elemento da lista. Alterações nessa variável não implicam em alterações nos elemento correspondente da lista se estes forem imutáveis.

```
>>> x=[3,4]
>>> for a in x:
...     a=99
>>> x
[3, 4]
```

Listas - **append**

- Uma operação importante em listas é acrescentar um item ao final. Isto é feito pela função

```
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 estes dados:

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

print(x)
```

Listas - funções em listas

- A operação de soma em listas

lista1 + lista2

gera uma nova lista que é o resultado de grudar `lista2` ao final da `lista1`. Isto é conhecido como **concatenação** de listas.

```
>>> lista1=[1,2,4]
>>> lista2=[27,28,29,30,33]
>>> x=l1+l2
>>> 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 os argumentos 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** da posição `índice`.

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

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

```
>>> 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 - versão 1

Inicializa os dois vetores como listas vazias e faz um "append" dos 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
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 - versao 2

Já que sabemos o tamanho, podemos criar os vetores 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 - versão 1

```
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 i in range(len(x)):
    for j in range(len(y)):
        if(x[i]==y[j]):
            umEmComum=True
            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

```
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:
    for b in y:
        if(a==b):
            umEmComum=True

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

Listas em funções

- Listas também podem ser passados como parâmetros em funções.
- Lembre-se que quando uma variável é passada como parâmetro, na verdade está sendo passado o seu identificador como parâmetro.
- Como uma lista é mutável, isto significa uma lista **pode ser alterada** dentro de uma função!

```
def removeZeros(l):
  i=0
  while(i<len(l)):
    if(l[i]==0):
      del l[i]
    else:
      i = i+1

x = [0,0,0,0,1,0,2,0,3,0]
removeZeros(x)
print(x)
```

Saída: [1, 2, 3]

Exemplo: Elementos Iguais - função que retorna uma lista

- Escreva uma função **elementosIguais(lista1,lista2)** que recebe 2 listas como parâmetro.
- A função deve retornar uma lista dos elementos da primeira lista que são também elementos da segunda lista, e retornar a lista vazia se não houver elementos em comum.

Exemplo: Elementos Iguais - função que retorna uma lista

```
def elementosIguais(x,y):
  iguais=[]
  for a in x:
    for b in y:
      if a==b:
        iguais.append(a)

  return iguais
```

Compreensão de Listas

- Compreensão de lista é uma construção para criação de listas com base na aplicação de alguma operação/função sobre elementos de uma lista já existente.
- Por exemplo, podemos criar uma lista com o quadrado de cada elemento de uma lista dada:

```
>> l1 = [1, 2, 3, 4]
>> l2 = [i*i for i in l1]
>> l2
[1, 4, 9, 16]
```

Compreensão de Listas

- Compreensão de lista pode ser usado com o uso de um *filtro* onde a função só é aplicada caso o elemento da lista satisfaça alguma condição.
- Por exemplo, podemos criar uma lista representando o conjunto:

$$\{x^2 \mid 2 \leq x \leq 1000 \text{ e } x \text{ é primo } \}$$

```
def ePrimo(n):
    div=2
    while(div <= n/2):
        if(n%div == 0):
            return False
        div = div+1
    return True

l = [i*i for i in range(2,1001) if ePrimo(i)]
print(l)
```

Saída: [4, 9, 25, 49 .... 982081, 994009]

Exercício

- Crie uma função
  maiorValor(lista)
  que recebe como parâmetro uma lista e devolve o maior valor armazenado na lista.

Exercício

- Crie uma função
  media(lista)
  que recebe uma lista e devolve a média dos valores armazenados na lista.

Exercício

- Crie uma função
  paresMult(lista, C)
  que recebe como parâmetros uma lista e um inteiro C. A função deve retornar uma lista de pares (lista com dois elementos) onde os dois elementos de cada par são elementos da lista que se multiplicados resultam no valor C.
- Exemplo: Se $lista = [2, 4, 3, 6, -10, 7]$ e $C = 12$ então a função deve devolver [[2,6], [4,3]]
- se $C = 14$ então a função deve devolver [[2,7]]
- e se $C = 15$ a função deve retornar []

Exercício

- Crie uma função
  elementosDiferentes(lista1, lista2)
  que recebe como parâmetros duas listas e retorna uma lista que contém os elementos de qualquer uma das listas que não esta na outra lista (a ordem não é importante).
- Exemplo: Se $lista1 = [2, 4, 3]$ e $lista2 = [1, 2, -10, 4, 5]$ então a função deve devolver [3,1,-10,5] ou uma lista com os mesmos elementos em outra ordem.