MC-102 — Aula 10
Listas

Eduardo C. Xavier

Instituto de Computação – Unicamp

3 de Abril de 2018

Roteiro

Listas

- Listas são construções de linguagens de programação que servem para armazenar vários dados de forma simplificada.
- No caso de Python, listas podem armazenar dados de um mesmo tipo (lista uniforme) ou dados de tipos diferentes.

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.

Definição de Listas

- Coleção ordenada de objetos referenciados por um **identificador único**.
- Características de Listas em Python:
  - Acesso de um objeto por meio de um índice inteiro.
  - Uma lista pode ser modificada incluindo-se ou removendo-se objetos dela.
  - Não possui tamanho pré-determinado, podendo aumentar ou diminuir de tamanho dependendo do número de objetos armazenados.

Declaração de uma lista

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

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

Exemplo de listas

Exemplo de listas:

- Uma lista de inteiros.

```
x = [2, 45, 12, 9, -2]
```

- Listas podem conter objetos de tipos diferentes.

```
x = [2, “qwerty” ,45.99087, 0, “a”]
```

- Uma lista é um objeto (tudo é um objeto em Python), portanto listas podem conter outras listas.

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

- Listas podem ser vazias, indicadas por **[]**.

```
x = []
```

Usando uma Lista

- Pode-se acessar um objeto numa 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

- O índice usado para acessar um elemento da lista pode ser uma variável inteira.

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

Listas - índices

- Qual o resultado deste acesso?

```
>>> notas= [4.5, 8.6, 9, 7.8, 7]
>>> notas[-3]
???
```

- Qual o resultado deste acesso?

```
>>> notas= [4.5, 8.6, 9, 7.8, 7]
>>> notas[-100]
???
```

Listas - índices

- A operação de **slicing** 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 outra **lista** com os elementos de $ind1$ até $ind2 - 1$.

```
>>> l = [10, 20, 30, 40, 50]
>>> l[1:4]
[20, 30, 40]
>>> l[1:]
[20, 30, 40, 50]
>>> l[:-2]
[10, 20, 30]
```

Listas - índices

- Qual o resultado da operação abaixo?

```
>>> l = [10, 20, 30, 40, 50]
>>> l[-3:-1]
???
```

Funções sobre listas: **len**

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

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

- É 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])
```

Iteração sobre listas: **for**

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

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

pode ser implementado como:

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

- Na verdade este último **for** é a maneira mais comum para se percorrer todos elementos de uma lista em ordem.

Funções sobre 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 estes 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)
```

Funções sobre listas: **concatenação**

- A operação de soma em listas gera uma nova lista que é o resultado da **concatenação** das 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.

Funções sobre listas: **concatenação**

- 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**.

Funções sobre listas: **insert** e **del**

- O método **lista.insert(índice,dado)** insere na lista o dado na posição `índice`.

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

- A função **del(lista[posição])** remove da lista o item da posição especificada.

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

Funções sobre listas: **remove**

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

```
>>> x = [4,1,3,2,1]
>>> x.remove(1)
>>> x
[4, 3, 2, 1]
```

- Note que apenas a primeira ocorrência do objeto especificado é removida.

Funções sobre listas: resumo

Resumo de alguns métodos em listas:

- **l.append(e)** adiciona **e** no fim da lista **l**.
- **l.count(e)** devolve o número de ocorrências de **e** em **l**.
- **l.insert(i, e)** insere **e** na posição **i** em **l**.
- **l.extend(l2)** adiciona elementos de **l2** no fim de **l**.
- **l.remove(e)** remove a primeira ocorrência de **e**, e gera exceção se **e** não ocorre em **l**.
- **l.index(e)** devolve o índice da primeira ocorrência de **e** em **l** e gera exceção se **e** não ocorre em **l**.

Exemplo 1: Produto Interno de dois vetores

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

Exemplo 1: 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.

```
v1 = []
v2 = []

for i in range(5):
    aux = float(input('Dado '+str(i)+' de v1:'))
    v1.append(aux)

print('----------------------')

for i in range(5):
    aux = float(input('Dado '+str(i)+' de v2:'))
    v2.append(aux)

#calculando o produto interno
...
```

Exemplo 1: Produto Interno de dois vetores

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

```
#calculando o produto interno
result = 0
for i in range(5):
    result = result + v1[i]*v2[i]

print('Produto Interno: ',result)
```

Exemplo 1: Produto Interno de dois vetores

- Agora o código completo.

```
v1 = []
v2 = []

for i in range(5):
    aux = float(input('Dado '+str(i)+' de v1:'))
    v1.append(aux)

print('----------')

for i in range(5):
    aux = float(input('Dado '+str(i)+' de v2:'))
    v2.append(aux)

#calculando o produto interno
result = 0
for i in range(5):
    result = result + v1[i]*v2[i]

print('Produto Interno: ',result)
```

Exemplo 2: Elementos Iguais

- Ler duas listas com 5 inteiros cada.
- Checar quais elementos da segunda lista são iguais a algum elemento da primeira lista.
- Se não houver elementos em comum, o programa deve informar isso.

Exemplo 2: Elementos Iguais

- Abaixo está o código que faz a leitura das listas.

```
v1 = []
v2 = []

for i in range(5):
    aux = float(input('Dado '+str(i)+' da lista 1:'))
    v1.append(aux)

print('----------------------')

for i in range(5):
    aux = float(input('Dado '+str(i)+' da lista 2:'))
    v2.append(aux)
  ...
```

Exemplo 2: Elementos Iguais

- Para cada elemento de **v1** testamos todos os outros elementos de **v2** para saber se são iguais.
- Usamos uma variável indicadora para decidir ao final dos laços encaixados, se as listas possuem ou não um elemento em comum.

```
comum = False   #Assumimos que não hajam elementos comuns
for i in range(len(v1)):
    for j in range(len(v2)):
        if(v1[i] == v2[j]):
            comum=True  # Descobrimos que há elemento comum
            print("lista1["+str(i)+"] igual a lista2["+str(j)+"]")

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

Exemplo 2: Elementos Iguais

- Código completo abaixo.

```
v1 = []
v2 = []

for i in range(5):
    aux = float(input('Dado '+str(i)+' da lista 1:'))
    v1.append(aux)

print('--------------------')

for i in range(5):
    aux = float(input('Dado '+str(i)+' da lista 2:'))
    v2.append(aux)

comum = False   #Assumimos que não hajam elementos comuns
for i in range(len(v1)):
    for j in range(len(v2)):
        if(v1[i] == v2[j]):
            comum=True  # Descobrimos que há elemento comum
            print("lista1["+str(i)+"] igual a lista2["+str(j)+"]")

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

Inicialização de Listas

- Em algumas situações é necessário declarar e já atribuir um conjunto de objetos para uma lista.
- 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]
```

Objetos Mutáveis e Imutáveis

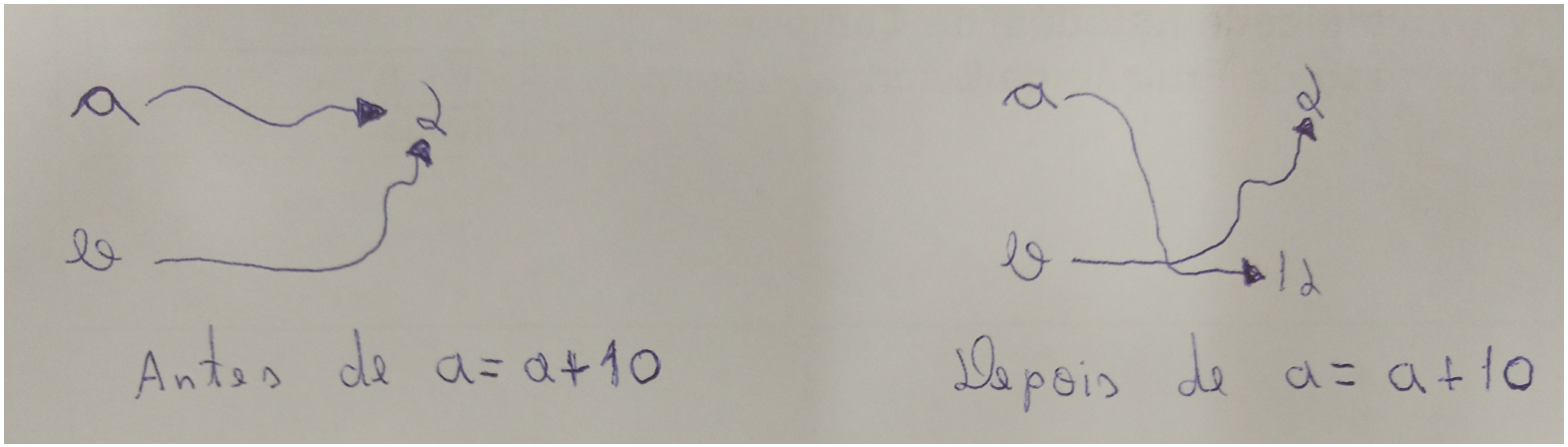
- Tudo em Python é um objeto, e seu tipo especifica que operações podem ser realizadas sobre o objeto.
- O tipo de um objeto também indica se ele é **mutável** ou **imutável**.
- Como o nome sugere, um objeto imutável não pode ser alterado.
  - Os tipos vistos anteriormente, **int, float, string, bool** são todos imutáveis.
- Já as listas são mutáveis, o que significa que operações podem ser realizadas sobre um objeto lista alterando-o (não é criado um novo objeto após a alteração).

Objetos Mutáveis e Imutáveis

- Considere o exemplo:

```
>>> a = 2
>>> b = a
>>> a = a + 10
>>> a
12
>>> b
2
```

- O tipo **int** é imutável, logo quando realizamos a soma **a+10** o objeto **2** não pode ser alterado, e o que ocorre é a criação de um novo objeto **12** e uma nova associação de **a** com este.

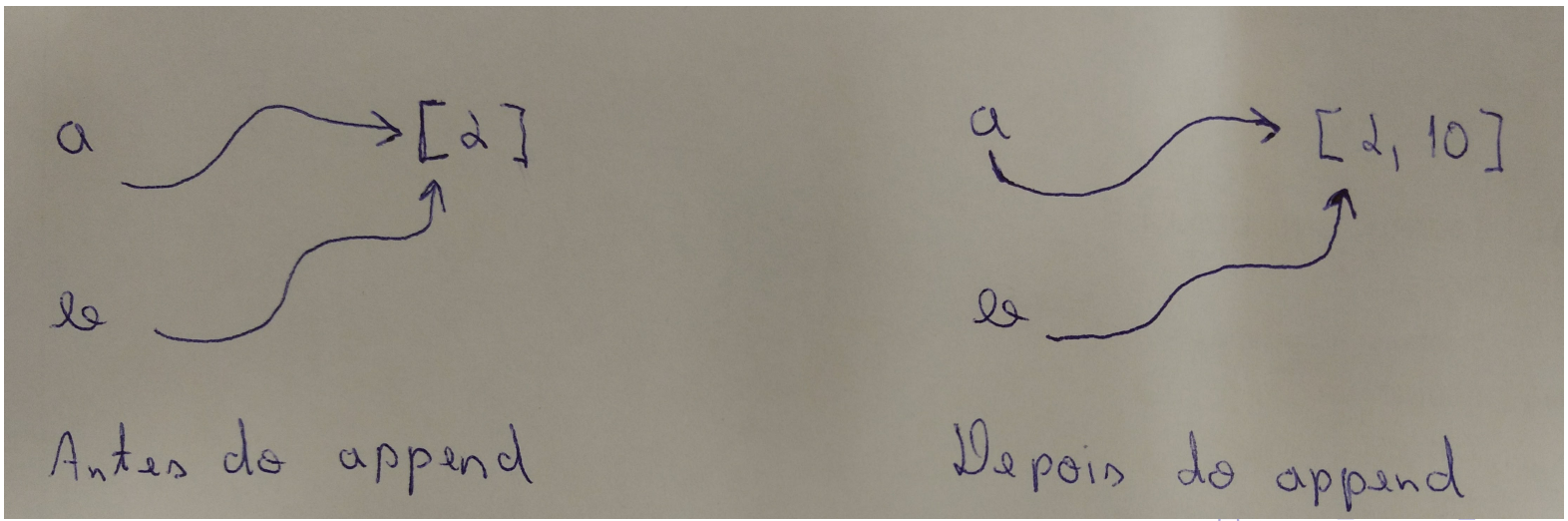


Objetos Mutáveis e Imutáveis

- Considere o exemplo com listas:

```
>>> a=[2]
>>> b=a
>>> a.append(10)
>>> a
[2, 10]
>>> b
[2, 10]
```

- O tipo **list** é mutável, logo quando executamos o método **a.append(10)** a lista é alterada e como **b** estava associada com a mesma lista, o resultado é percebido em **b** também:



Objetos Mutáveis e Imutáveis

- Alguns tipos são mutáveis pois a criação de objetos deste tipo é caro computacionalmente.
- A operação de concatenação (+) sobre listas cria sempre uma nova lista, o que é caro:

```
>>> a=[1,2,3,4]
>>> b=[5,6,7,8,9]
>>> c = a+b
>>> c
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> id(a)
4302774664
>>> id(b)
4302778312
>>> id(c)
4302774408
```

Objetos Mutáveis e Imutáveis

- Se não é problema alterar uma das listas (**a** ou **b**) o método **extend** é preferível à concatenação:

```
>>> a=[1,2,3,4]
>>> b=[5,6,7,8,9]
>>> id(a)
4302774664
>>> a.extend(b)
>>> id(a)
4302774664
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Note que **a** continuou associado com a mesma lista, e esta teve o conteúdo alterado.

Objetos Mutáveis e Imutáveis

- Por outro lado, em algumas situações não desejamos alterar um objeto que sofrerá alterações.
- Neste caso pode-se usar o método **copy** que cria uma copia do objeto especificado.

```
>>> a=[1,2,3,4]
>>> b = a.copy()
>>> id(a)
4302758856
>>> id(b)
4302758088
>>> a.append(10)
>>> a
[1, 2, 3, 4, 10]
>>> b
[1, 2, 3, 4]
```

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 imprimi-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".

Exercício

- Escreva um programa que lê duas listas de 5 inteiros e salva em uma terceira lista os elementos de qualquer uma das duas listas que não está na outra lista (a ordem não é importante).
- Exemplo: Se $l_1 = [2, 4, 3]$ e $l_2 = [1, 2, -10, 4, 5]$ então a a terceira lista deve conter $l_3 = [3, 1, -10, 5]$ ou uma lista com os mesmos elementos em outra ordem.