# MC-102 — Aula 10
# Listas

Eduardo C. Xavier

Instituto de Computação – Unicamp

15 de Outubro de 2020

# 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

## 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:

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

# Exemplos de declaração 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 = []
```

# Exemplos de declaração de listas

## list

Como vimos a função **list** pode ser aplicado sobre alguns objetos específicos para se criar uma lista.

Exemplo de criação de listas a partir de objetos (devem ser *iterable*):

- Uma lista a partir de um objeto do tipo **range**.

```
>>> a = range(1,10)
>>> list(a)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

- Uma lista a partir de um objeto do tipo **string**.

```
>>> a = "ola turma"
>>> list(a)
['o', 'l', 'a', ' ', 't', 'u', 'r', 'm', 'a']
>>>
```

# Usando uma Lista

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

```
>>> notas= [4.5, 8.6, 9, 7.8, 7]
>>> notas[0]
4.5
>>> notas[4]
7
```

# Usando uma Lista

- 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
```

- Podemos alterar o valor de uma posição específica com o comando de atribuição.

```
>>> 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
```

# Usando uma Lista

- 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

- Qual o resultado deste acesso?

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

- Qual o resultado deste acesso?

```
>>> notas= [4.5, 8.6, 9, 7.8, 7]
>>> notas[-100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

# Listas - Slicing

## Slicing

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]`
- 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]
```

# Listas - Slicing

- Obtendo uma lista com todos os itens exceto o da posição 0:

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

- Obtendo uma lista com todos os itens exceto os 2 últimos:

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

# Listas - Slicing

- Qual o resultado da operação abaixo?

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

# Listas - Slicing

- Qual o resultado da operação abaixo?

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

# Funções sobre listas: **len**

## 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** faz a variável de controle assumir todos os valores de uma lista.

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

- O código acima pode ser implementado como:

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

# Funções sobre listas: **append**

## append

Usamos a função **append** para acrescentar um item no final de uma lista.
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 %d:" %i))
    x.append(dado)

print(x)
```

# Funções sobre listas: **concatenação**

## concatenação

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

```
>>> 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: **insert**

**insert**

O método **insert** insere um novo item na lista em uma determinada posição. O item anterior na posição a ser interida é "deslocado" para frente.

- 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]
```

# Funções sobre listas: **remove**

## remove

O método **remove** remove a primeira ocorrência de um dado objeto na lista.

- 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: **in**

**in**

O operador **in** testa se um determinado objeto está ou não na lista.

- Podemos verificar se um objeto está na lista com o operador **in**.

```
>>> x = [4,1,3,2]
>>> 5 in x
False
>>> 4 in x
True
```

# 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**.
- **x in l** True ou False indicando se objeto **x** está ou não 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 = []

#Lendo dados de v1
for i in range(5):
  aux = float(input("v1[%d]:" %i))
  v1.append(aux)

print()
#Lendo dados de v2
for i in range(5):
  aux = float(input("v2[%d]:" %i))
  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: %.2f" %result)
```

# Exemplo 1: Produto Interno de dois vetores

- Agora o código completo.

```
v1 = []
v2 = []

#Lendo dados de v1
for i in range(5):
  aux = float(input("v1[%d]:" %i))
  v1.append(aux)

print()
#Lendo dados de v2
for i in range(5):
  aux = float(input("v2[%d]:" %i))
  v2.append(aux)

print(v1)
print(v2)

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

print("Produto interno: %.2f" %result)
```

# Exercício

- Escreva um programa que cria uma lista com todos os números ímpares de 0 até 100.

# 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.