Algoritmos e Programação de Computadores

Listas

Prof. Edson Borin
Instituto de Computação (IC/Unicamp)

Agenda

- Introdução
- Listas
- Exemplos e Exercícios

Listas

Listas (Breve Introdução)

- Uma lista em Python é uma estrutura que armazena vários dados, que **podem ser de um mesmo tipo ou não**.
- Uma lista é criada como a construção: $[dado_1, dado_2, ..., dado_n]$

```
lista1 = [10, 20, 30, 40]
lista2 = ["programação", "mc102", "python"]
lista3 = ["oi", 2.0, 5, [10, 20]]
```

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?

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

Listas

- Com o que sabemos, como armazenaríamos 130 notas?

```
nota1 = float(input("Entre com a nota 1: "))
nota2 = float(input("Entre com a nota 2: "))
nota3 = float(input("Entre com a nota 3: "))
nota4 = float(input("Entre com a nota 4: "))
nota5 = float(input("Entre com a nota 5: "))
…
nota130 = float(input("Entre com a nota 130: "))
```

- Criar 130 variáveis distintas **não** é uma solução elegante.

Listas: Definição

- Coleção de valores referenciados por um **identificador único**.

```
identificador = [dado_1,dado_2,...,dado_n]
```

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
```

- Características:
  - Acesso por meio de um índice inteiro que inicia em 0 (zero).
  - Listas podem ser modificadas.
  - Pode-se incluir e remover itens de listas.

Exemplos de Listas

- Lista de inteiros:

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

Exemplos de Listas

- Também é possível criar uma lista de objetos inteiros combinando a função list() com a função range()

```
x = list(range(2,7))
print(x)
```

```
[2, 3, 4, 5, 6]
```

Exemplos de Listas

- A função list() também pode ser usada para converter uma string em uma lista de caracteres:

```
x = list("MC102 é legal")
print(x)
```

```
['M', 'C', '1', '0', '2', ' ', 'é', ' ', 'l', 'e', 'g', 'a', 'l']
```

Listas: Como Usar

- Pode-se acessar uma determinada posição da lista utilizando-se um índice de valor inteiro.
- A sintaxe para acesso de uma determinada posição é:
  - `identificador[`**`posição`**`]`
- 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.

Listas: Como Usar

Lista `notas` : tamanho *n* = 5, ou seja, os índices válidos são de 0 até 4 (5 – 1).

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
print(notas[0])
print(notas[1])
print(notas[2])
print(notas[3])
print(notas[4])
```

```
8.0
5.5
9.3
7.6
3.1
```

Listas: Como Usar

Lista `notas` : tamanho $n = 5$, ou seja, os índices válidos são de 0 até 4 (5 – 1).

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
print(notas[0])
print(notas[1])
print(notas[2])
print(notas[3])
print(notas[4])
```

```
8.0
5.5
9.3
7.6
3.1
```

A primeira posição da lista tem índice 0

A última posição da lista tem índice $n - 1$

Listas: Como Usar

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

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]

print(notas[0]+2)
10.0

notas[3] = 0.5

print(notas)
[8.0, 5.5, 9.3, 0.5, 3.1]
```

Listas: Como Usar

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

```
notas = [8.0, 5.5, 9.3, 7.6,
print(notas[0]+2)
10.0

notas[3] = 0.5

print(notas)
[8.0, 5.5, 9.3, 0.5, 3.1]
```

Esta operação de atribuição modifica a lista substituindo o quarto objeto (índice 3) da lista pelo objeto produzido pela expressão à direita do =. Neste caso um objeto do tipo float com valor 0.5.

Listas: Como Usar

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

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]

for i in range(5)
    print(notas[i])
```

```
8.0
5.5
9.3
0.5
3.1
```

Listas: Como Usar

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

```
lista = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

for i in range(10):
    lista[i] = 5*i
print(lista)
```

Listas: Como Usar

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

```
lista = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

for i in range(10):
    lista[i] = 5*i
print(lista)
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

Listas: Índices

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

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
print(notas[-1])
print(notas[-2])
print(notas[-3])
print(notas[-4])
print(notas[-5])
print(notas[-6])
```

```
3.1
0.5
9.3
5.5
8.0
IndexError: list
index out of range
```

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

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 como

  ```
  identificador[ind1:ind2]
  ```

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

Listas: Índices

- O **slicing** em Python é obtido como

  `identificador[ind1:ind2]`

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

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
print(notas[1:4])
```

```
[5.5, 9.3, 0.5]
```

Listas: Índices

- O **slicing** pode incluir um "passo"

```
identificador[ind1:ind2:passo]
```

e o resultado é uma sub-lista com os elementos de `ind1` até `ind2-1` saltando de passo em passo

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1, 2.2]
print(notas[1:6:2])
```

```
[5.5, 0.5, 2.2]
```

Listas: Índices

- O índice inicial e/ou o final podem ser omitidos. Nestes casos, o **slicing** supõe que o índice inicial/final é o primeiro/último da lista.

```
Identificador[ind1:] ou identificador[:ind2]
```

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1, 2.2]
print(notas[:2])
```

```
[8.0, 5.5]
```

```
print(notas[3:])
```

```
[0.5, 3.1, 2.2]
```

Listas: Função `len`

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

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
len(notas)
```

```
5
```

- É possível comum usar a função **`len`** junto com a função **range** para gerar uma lista de índices válidos e percorrer uma lista com um laço **`for`**:

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
for i in range(len(notas)):
    print(notas[i])
```

Listas: `for`

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

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
for i in range(len(notas)):
    print(notas[i])
```

- E também pode ser implementado como:

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
for i in notas:
    print(i)
```

Listas: método `append`

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

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

O identificador da lista que será modificada (p.ex: nome da variável que aponta para a lista) 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.

Listas: método `append`

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

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

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
notas.append(9.5)
print(notas)
```

```
[8.0, 5.5, 9.3, 0.5, 3.1, 9.5]
```

Listas: método `append`

- A combinação de uma lista vazia que vai sofrendo “appends” permite ler dados e preencher uma lista com estes dados:

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

Listas: operador +

- Quando aplicado em listas, o operador + produz uma nova lista que corresponde à união das duas listas. Isto é conhecido como **concatenação de listas**.

```
lista1 + lista2
```

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

Listas: operador +

- Quando aplicado em listas, o operador + produz uma nova lista que corresponde à união das duas listas. Isto é conhecido como **concatenação de listas**.

```
lista1 + lista2
```

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

```
[1, 2, 4, 27, 28, 29, 30, 33]
```

Listas: operador *

- O operador “*” faz repetições da concatenação:

```
x = [1, 2, 3]
y = 4*x
print(y)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

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

Outras operações em Listas

- Podemos contar o número de elementos na lista com um certo valor usando o método `count`.

```
x = [40, 99, 10, 20, 13, 20, 14]
N = x.count(20)
print(N)
```

```
2
```

Outras operações em Listas

- Podemos verificar se um elemento pertence a uma lista usando o operador **`in`**:

```
x = [40, 99, 10, 20, 13, 20, 14]
if 99 in x:
  print("99 está em x")
```

```
99 está em x
```

Outras operações em Listas

- `lista.insert(índice,dado)` insere na lista o dado antes da posição índice.

```
x = [40, 30, 10, 20]
x.insert(1,99)
print(x)
```

Outras operações em Listas

- `lista.insert(índice,dado)` insere na lista o dado antes da posição índice.

```
x = [40, 30, 10, 20]
x.insert(1,99)
print(x)
```

```
[40, 99, 30, 10, 20]
```

Outras operações em Listas

- `del lista[posição]` remove da lista o item da posição especificada.

```
x = [40, 99, 30, 10, 20]
del x[2]
print(x)
```

```
[40, 99, 10, 20]
```

Outras operações em Listas

- Também podemos remover um item da lista pelo valor utilizando o método `remove`.

```
x = [40, 99, 10, 20]
x.remove(10)
print(x)
```

```
[40, 99, 20]
```

- Nota: Caso haja múltiplos elementos com o valor especificado, este método remove a primeira ocorrência do valor.

Outras operações em Listas

- Podemos remover todos os elementos com um dado valor repedindo a operação de remover até que não haja elementos com o valor

```
x = [40, 99, 10, 99, 20, 99]
while 99 in x:
  x.remove(99)
print(x)
```

```
[40, 10, 20]
```

Outras operações em Listas

- Outra forma de remover todos os elementos com um dado valor:

```
x = [40, 99, 10, 99, 20, 99]
for i in range(x.count(99)):
  x.remove(99)
print(x)
```

```
[40, 10, 20]
```

Outras operações em Listas

- O método pop(i) pode ser usado para extrair (remover e retornar) o elemento de índice i.

```
x = [40, 99, 10, 99, 20, 99]
y = x.pop(2)
print("x=",x)
print("y=",y)
```

```
x= [40, 99, 99, 20, 99]
y= 10
```

Outras operações em Listas

- O método sort() pode ser usado para ordenar os elementos de uma lista.

```
x = [80, 10, 20, 50, 30]
x.sort()
print(x)
```

```
[10, 20, 30, 50, 80]
```

Outras operações em Listas

- O método sort() pode ser usado para ordenar os elementos de uma lista. **OBS: Os elementos devem possuir tipos comparáveis!!!**

```
x = [80, 10, "oi", 50, 30]
x.sort()
print(x)
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    x.sort()
TypeError: '<' not supported between instances of 'str' and 'int'
```

Outras operações em Listas

- O método sort() pode ser usado para ordenar os elementos de uma lista. **OBS: Os elementos devem possuir tipos comparáveis!!!**
    - Float é comparável com int!

```
x = [80, 10, 5.0, 50, 30]
x.sort()
print(x)
```

```
[5.0, 10, 30, 50, 80]
```

Outras operações em Listas

- O método sort() pode ser usado para ordenar os elementos de uma lista. **Cuidado ao ordenar strings: Caracteres maiúsculos precedem os minúsculos nesta linguagem.**

```
cores = ["azul","Azul","branco","Branco","Cinza"]
cores.sort()
print(cores)
```

```
['Azul', 'Branco', 'Cinza', 'azul', 'branco']
```

Outras operações em Listas

- O método reverse() pode ser usado para inverter a ordem dos elementos.

```
x = [1, 20, 30, 100]
x.reverse()
print(x)
```

```
[100, 30, 20, 1]
```

Outras operações em Listas

- O método index(v) pode ser usado para obter o índice do primeiro elemento que tenha o valor v.

```
disciplinas = ["MC102", "MC202", "MC404", "MC722"]
i = disciplinas.index("MC404")
print("MC404 está na posição",i)
```

```
MC404 está na posição 2
```

Outras operações em Listas

- A função max(v) compara todos os elementos e retorna o de maior valor.

```
l = [10, 42, 1001, 5.0]
print("Maior de l:", max(l))
d = ["MC102", "MC202", "MC404", "MC722"]
print("Maior de d:", max(d))
```

```
Maior de l: 1001
Maior de d: MC722
```

Outras operações em Listas

- A função min(v) compara todos os elementos e retorna o de menor valor.

```
l = [10, 42, 1001, 5.0]
print("Menor de l:", min(l))
d = ["MC102", "MC202", "MC404", "MC722"]
print("Menor de d:", min(d))
```

```
Menor de l: 5.0
Menor de d: MC102
```

Outras operações em Listas

- A função sum(v) soma todos os elementos da lista e retorna o resultado. **OBS: A lista deve conter apenas valores numéricos (int ou float)**

```
l1 = [10, 42, 1001]
print("sum(l1):", sum(l1))
l2 = [10, 42, 1001.0]
print("sum(l2):", sum(l2))
```

```
sum(l1): 1053
sum(l2): 1053.0
```

Informações Extras: Inicialização de uma Lista

- Em algumas situações é necessário declarar e já atribuir um conjunto de valores constantes para uma lista.
- Dentro da lista incluímos uma construção com um laço que gerará valores iniciais para a lista. (Compreensão de listas)

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

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

Cópia de listas

- O que o seguinte trecho de código imprime?

```
a = [40, 99, 10, 20]
b = a
b.append(1969)
print("a=",a)
print("b=",b)
```

Cópia de listas

- O que o seguinte trecho de código imprime?

```
a = [40, 99, 10, 20]
b = a
b.append(1969)
print("a=",a)
print("b=",b)
```

```
a= [40, 99, 10, 20, 1969]
b= [40, 99, 10, 20, 1969]
```

Cópia de listas

- No caso anterior, existe apenas uma cópia do objeto lista na memória e ambas as variáveis (a e b) apontam para a mesma lista. Logo, chamar o método append() para a lista apontada pela variável a é equivalente a chamar o método append() para a lista apontada pela variável b!

```
a = [40, 99, 10, 20]
b = a
b.append(1969) # equivale a a.append(1969)
print("a=",a)
print("b=",b)
```

Cópia de listas

- Para realizar uma cópia da lista, podemos usar o método copy().

```
a = [40, 99, 10, 20]
b = a.copy()
b.append(1969)
print("a=",a)
print("b=",b)
```

Realiza uma cópia da lista apontada pela variável a e faz a variável b apontar para a nova lista.

```
a= [40, 99, 10, 20]
b= [40, 99, 10, 20, 1969]
```

Cópia de listas

- Também é possível copiar listas com a função list() ou com o operador de slicing.

```
a = [40, 99, 10, 20]
b = list(a)
b.append(1969)
c = b[:]
c.append(2020)
print("a=",a)
print("b=",b)
print("c=",c)
```

```
a= [40, 99, 10, 20]
b= [40, 99, 10, 20, 1969]
c= [40, 99, 10, 20, 1969, 2020]
```

Exemplos & Exercícios

Exercício

- Faça um programa que leia um número *n* e imprima *n* linhas na tela com o seguinte formato (exemplo se *n* = 5):

Entrada	Saída
5	1 1 2 1 2 3 1 2 3 4 1 2 3 4 5

Exemplo de Solução (com Laços Encaixados)

- Faça um programa que leia um número *n* e imprima *n* linhas na tela com o seguinte formato (exemplo se *n* = 5):

```
n = int(input())

for i in range(1,n+1):
    for j in range(1,i+1):
        print(j, end=" ")
    print("")
```

Exemplo de Solução

- Faça um programa que leia um número *n* e imprima *n* linhas na tela com o seguinte formato (exemplo se *n* = 5):

```
n = int(input("Digite um número: "))
l = []
for i in range(1,n+1):
  l.append(i)
  print(l)
```

Exemplo de Solução (Alun* de MC102)

- Faça um programa que leia um número *n* e imprima *n* linhas na tela com o seguinte formato (exemplo se *n* = 5):

```
n = int(input("Digite um número: "))
l = []
for i in range(1,n+1):
  l.append(i)
  for j in l :
    print(j,end=" ")
  print("")
```

Exercício

- Faça um programa que leia *n* notas, mostre as notas e a média.

Entrada	Saída
5 8.0 5.5 9.3 0.5 3.1	[8.0, 5.5, 9.3, 0.5, 3.1] 5.3

Faça um programa que leia *n* notas, mostre as notas e a média.

```
# Mostra as n notas
notas = []
n = int(input())
for i in range(n):
    dado = float(input())
    notas.append(dado)
print(notas)
```

Essa parte lê as *n* notas e mostra na tela.

```
# Calcula a média
soma = 0
for i in range(len(notas)):
    soma = soma + notas[i]
media = soma/n
print(format(media, ".1f"))
```

Essa parte calcula a média e mostra na tela.

Faça um programa que leia *n* notas, mostre as notas e a média.

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

# Calcula a média
soma = 0
for i in range(len(notas)):
    soma = soma + notas[i]
media = soma/n
print(format(media, ".1f"))
```

Faça um programa que leia *n* notas, mostre as notas e a média.

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

# Calcula a média
soma = 0
for i in notas:
    soma = soma + i
media = soma/n
print(format(media, ".1f"))
```

Faça um programa que leia *n* notas, mostre as notas e a média.

```
# Mostra as n notas e Calcula a média
notas = []
soma = 0

n = int(input("Entre com o número de notas: "))
for i in range(n):
    dado = float(input("Entre com a nota " + str(i) + ": "))
    notas.append(dado)
    soma = soma + dado
print(notas)

# Calcular a média
media = soma/n
print(format(media, ".1f"))
```

Faça um programa que leia *n* notas, mostre as notas e a média.

```
# Mostra as n notas e Calcula a média
notas = []

n = int(input("Entre com o número de notas: "))
for i in range(n):
  dado = float(input("Entre com a nota " + str(i) + ": "))
  notas.append(dado)
print(notas)

# Calcular a média
media = sum(notas)/n
print("%.1f"%(media))
```

Exercício

- Faça um programa que:
  - Lê duas listas com 5 inteiros cada.
  - Checa quais elementos da segunda lista são iguais a algum elemento da primeira lista.
  - Se não houver elementos em comum, o programa deve informar isso.

Entrada	Saída
`[1, 2, 3, 4, 5]` `[0, 7, 6, 10, 3]`	`3`

Entrada	Saída
`[1, 2, 3, 4, 5]` `[0, 7, 6, 10, 8]`	`Não tem.`

```
x = []
y = []
for i in range(5):
    x.append(int(input()))
print()

for i in range(5):
    y.append(int(input()))
print()

#Supomos que não temos elementos comuns
um_elemento_comum  = False

for i in range(len(x)):
    for j in range(len(y)):
        if (x[i] == y[j]):
            um_elemento_comum  = True # há elemento comum
            print(str(x[i]))

if not um_elemento_comum:
    print("Não tem elemento comum.")
```

```
x = []
y = []
for i in range(5):
    x.append(int(input()))
print()

for i in range(5):
    y.append(int(input()))
print()

#Supomos que não temos elementos comuns
um_elemento_comum  = False

for a in x:
    for b in y:
        if (a == b):
            um_elemento_comum  = True # há elemento comum
            print(str(a))

if not um_elemento_comum:
    print("Não tem elemento comum.")
```

```
x = []
y = []
for i in range(5):
  x.append(int(input()))
print(x)

for i in range(5):
  y.append(int(input()))
print(y)

#Supomos que não temos elementos comuns
um_elemento_comum  = False

for a in y:
  if a in x:
    um_elemento_comum  = True # há elemento comum
    print(a)

if not um_elemento_comum:
  print("Não tem elemento comum.")
```

```
#Lê listas
x = [ int(input("Entre com x["+str(i)+"]:")) for i in range(5) ]
y = [ int(input("Entre com y["+str(i)+"]:")) for i in range(5) ]

#Supomos que não temos elementos comuns
um_elemento_comum = False

#Para cada elemento em y, verifica se está em x também
for a in y:
  if a in x:
    um_elemento_comum = True # há elemento comum
    print(a)

if not um_elemento_comum:
  print("Não tem elemento comum.")
```

Exercício

- Faça um programa que:
  - Lê duas listas com 5 inteiros cada.
  - Imprima todos os elementos exclusivos, ou seja, que não aparecem nas duas listas.
  - Os números devem ser impressos de forma ordenada.

Entrada	Saída
`[1, 2, 3, 4, 5]` `[0, 2, 4, 10, 3]`	`0, 1, 5, 10`

Entrada	Saída
`[1, 2, 3, 4, 5]` `[2, 3, 1, 5, 4]`	`Não tem.`

Referências & Exercícios

- https://wiki.python.org.br/ExerciciosListas: 24 exercícios =)

Créditos

Os *slides* deste curso foram baseados nos slides produzidos e cedidos gentilmente pela Professora Sandra Ávila, do Instituto de Computação da Unicamp.