Listas

Patrícia de Siqueira Ramos

UNIFAL-MG, *campus* Varginha

∗ Aula inspirada no curso 'Introdução à Ciência da Computação com Python', de Fabio Kon, disponível na plataforma Coursera

7 de Outubro de 2020

Listas

- Até o momento, apenas trabalhamos com variáveis simples, capazes de armazenar um valor por vez (do tipo `inteiro`, `real`, `string`, `lógico`)
- A partir de agora veremos como armazenar vários valores em um tipo de estrutura sequencial indexada muito utilizada: listas
- Uma lista é uma sequência ou coleção ordenada de valores de qualquer tipo ou classe, inclusive listas (é uma estrutura heterogênea)

Listas em Python

- É uma das principais estruturas em Python
- O jeito mais simples de criar listas é com o uso de colchetes
- Os colchetes marcam o início e o fim da lista e os objetos são separados por vírgulas
- Trabalharemos as listas apenas em Python, não usaremos pseudocódigo e fluxograma

Exemplo 1 - Criação de listas em Python

```
primos = [2, 3, 5, 7, 11]
lista1 = [1, 'oi', 4.5, True]
lista2 = ['João', 'masculino', 15, 1.78]
lista3 = [2, 5 * 3, [10, 20]]
```

Para visualizar o conteúdo de uma lista basta escrever seu nome

```
lista1
```

```
[1, 'oi', 4.5, True]
```

Índices

- Cada valor na lista é identificado por um índice
- Dizemos que uma lista é uma estrutura sequencial indexada pois os seus elementos podem ser acessados sequencialmente utilizando índices
- No `Python`, o primeiro elemento da lista tem índice 0, o segundo tem índice 1, e assim por diante
- Para acessar um elemento de uma lista usamos o operador de indexação [ ]
- A expressão dentro dos colchetes especifica o índice

Índices

```
lista1 = [1, 'oi', 4.5, True]
lista1[0]
```

Saída: 1

```
lista1[1]
```

Saída: 'oi'

```
# índices negativos indicam a ordem da direita
# para a esquerda
lista1[-1]
```

Saída:

Índices

```
lista1 = [1, 'oi', 4.5, True]
lista1[0]
```

Saída: 1

```
lista1[1]
```

Saída: 'oi'

```
# índices negativos indicam a ordem da direita
# para a esquerda
lista1[-1]
```

Saída: **True**

Índices negativos

O `Python` possibilita o uso de posições negativas na lista, indicando a ordem da direita para esquerda:

```
primos = [2, 3, 5, 7, 11, 13]
```

Por exemplo:

```
lista1[-1]
```

retornará True

```
lista3[-1]
```

retornará [10, 20], que é uma lista

```
lista3[2][0]
```

retornará 10

Comprimento de listas e como adicionar elementos a ela

- No `Python` há a função `len`, que retorna o comprimento de uma lista, ou seja, quantos elementos ou objetos ela possui

```
len(primos)
```

que retornará o valor 5

Comprimento de listas e como adicionar elementos a ela

- No `Python` há a função `len`, que retorna o comprimento de uma lista, ou seja, quantos elementos ou objetos ela possui

```
len(primos)
```

que retornará o valor 5

- A partir de qualquer lista, podemos inserir um novo elemento utilizando o método `append`:

```
primos.append(13)
```

A `primos` após a inserção ficará:

```
[1, 3, 5, 7, 11, 13]
```

Criação de listas

- Para criar listas a partir da inserção de dados pelo usuário adotaremos o seguinte:
  - criar uma lista vazia
  - inserir um novo elemento nessa lista com o método `append`

Exemplo 2 - Criação e uso de uma lista com 10 notas em Python

```
notas = [ ]
print('Insira as notas de 10 alunos')
for i in range(10):
    nota = float(input())
    notas.append(nota)
notas
```

Exemplo 2 - Criação e uso de uma lista com 10 notas em Python

```
notas = [ ]
print('Insira as notas de 10 alunos')
for i in range(10):
    nota = float(input())
    notas.append(nota)
notas
```

Após a inserção de 10 notas o que aparece na tela é uma lista:

```
[4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 0.0, 2.0, 3.0, 4.0]
```

Exemplo 3 - Leitura e inserção das notas de 10 alunos e retorno da média

```
notas = [ ]
soma = 0
print('Insira as notas de 10 alunos')
for i in range(10):
    notas.append(float(input()))
    soma = soma + notas[i]
media = soma / 10
print(media)
```

Exemplo 3 - Leitura e inserção das notas de 10 alunos e retorno da média

```
notas = [ ]
soma = 0
print('Insira as notas de 10 alunos')
for i in range(10):
    notas.append(float(input()))
    soma = soma + notas[i]
media = soma / 10
print(media)
```

Após a inserção de 10 notas (por exemplo: 6, 7, 8, 5, 6, 7, 8, 5, 6, 7), o que aparece na tela é:

Média da turma: 6.5

Exercício 1

Como ficaria o exemplo 3 se o usuário definisse o número de notas que ele deseja inserir?

Mais tópicos importantes sobre listas no Python

- outras formas de controlar o `for`
- fatias de listas
- cuidado com atribuição de listas
- clone (= cópia) de listas

Outras formas de controlar os índices dentro do for

O `Python` permite percorrer uma lista da seguinte forma (não precisamos sempre da função `range()`):

```
primos = [2, 3, 5, 7, 11, 13]
for elem in primos:
    print(elem, end=‘ ’)
```

E o resultado será:

```
2 3 5 7 11 13
```

Teremos a mesma saída se usarmos:

```
primos = [2, 3, 5, 7, 11, 13]
for elem in range(len(primos)):
    print(primos[elem])
```

Fatias de listas

Fatias de listas são usadas quando precisamos apenas de alguns valores, em intervalos:

```
primos = [2, 3, 5, 7, 11]
primos[1:2]
```

Qual será a saída na tela?

Fatias de listas

Fatias de listas são usadas quando precisamos apenas de alguns valores, em intervalos:

```
primos = [2, 3, 5, 7, 11]
primos[1:2]
```

Qual será a saída na tela?

```
[3]
```

Fatias de listas

Fatias de listas são usadas quando precisamos apenas de alguns valores, em intervalos:

```
primos = [2, 3, 5, 7, 11]
primos[1:2]
```

Qual será a saída na tela?

```
[3]
```

- `primos`$[2:4]$
  Saída:

Fatias de listas

Fatias de listas são usadas quando precisamos apenas de alguns valores, em intervalos:

```
primos = [2, 3, 5, 7, 11]
primos[1:2]
```

Qual será a saída na tela?

```
[3]
```

- `primos[2 : 4]`
  Saída: $[5, 7]$
- `primos[: 3] # observe que o início não precisa ser definido`
  Saída:

Fatias de listas

Fatias de listas são usadas quando precisamos apenas de alguns valores, em intervalos:

```
primos = [2, 3, 5, 7, 11]
primos[1:2]
```

Qual será a saída na tela?

```
[3]
```

- `primos[2 : 4]`
  Saída: $[5, 7]$
- `primos[: 3] # observe que o início não precisa ser definido`
  Saída: $[2, 3, 5]$
- `primos[3 :] # observe que o fim não precisa ser definido`
  Saída:

Fatias de listas

Fatias de listas são usadas quando precisamos apenas de alguns valores, em intervalos:

```
primos = [2, 3, 5, 7, 11]
primos[1:2]
```

Qual será a saída na tela?

```
[3]
```

- `primos[2 : 4]`
  Saída: $[5, 7]$
- `primos[: 3] # observe que o início não precisa ser definido`
  Saída: $[2, 3, 5]$
- `primos[3 :] # observe que o fim não precisa ser definido`
  Saída: $[7, 11]$
- `primos[:] # o mesmo que escrever somente primos`
  Saída:

Fatias de listas

Fatias de listas são usadas quando precisamos apenas de alguns valores, em intervalos:

```
primos = [2, 3, 5, 7, 11]
primos[1:2]
```

Qual será a saída na tela?

```
[3]
```

- `primos[2 : 4]`
  Saída: $[5, 7]$
- `primos[: 3] # observe que o início não precisa ser definido`
  Saída: $[2, 3, 5]$
- `primos[3 :] # observe que o fim não precisa ser definido`
  Saída: $[7, 11]$
- `primos[:] # o mesmo que escrever somente primos`
  Saída: $[2, 3, 5, 7, 11]$

Cuidados com atribuição de listas

- Atribuição de listas criam uma nova referência para lista, não criam um clone (= cópia) da lista. Exemplo:

```
a = [2, 3, 4]
b = a
b[2] = 100
a
```

O que aparecerá na tela (qual o conteúdo de a)?

Cuidados com atribuição de listas

- Atribuição de listas criam uma nova referência para lista, não criam um clone (= cópia) da lista. Exemplo:

```
a = [2, 3, 4]
b = a
b[2] = 100
a
```

O que aparecerá na tela (qual o conteúdo de a)?

```
[2, 3, 100]
```

Clone = cópia de listas

- Para que esse problema não ocorra, devemos criar um clone da lista a com `[:]`

```
a = [2, 3, 4]
b = a[:]
b[2] = 100
a
```

E o resultado será

```
[2, 3, 4]
```

Função soma com lista

Uma função pode receber parâmetros. Até agora, os parâmetros que usamos eram números ou textos, mas listas também podem ser passadas como parâmetros para funções:

```
def soma_elementos(lista):
    soma = 0
    for i in range(len(lista)):
        soma = soma + lista[i]
    return soma

num = [4, 2, 6, -3, -2]
soma_elementos(num)
```

E o resultado será 7

Testar comandos de listas do Python

```
lista1 = [‘vermelho’, ‘verde’, ‘azul’]
lista2 = lista1[:]
lista2[0] = ‘rosa’
lista2
lista1

# pertinência a uma lista
‘rosa’ in lista1
‘rosa’ in lista2
‘preto’ not in lista1

# concatenação de listas
[4,2,4,5] + [10,33,55]

a = [1, 2, 3]
b = [4, 5, 6]
a + b
b + a

# inserir elementos em posições específicas da lista
lista4 = [10, 20, 40, 50, 60]
lista4.insert(2, 30)
lista4
```

Testar comandos de listas do Python

```
# repetição de listas
a3 = a * 3
a3

# apagar elementos de uma lista por meio do índice
del a3[2]
a3
del a3[2:3]
a3

# apagar elementos de uma lista por meio do valor
a3.remove(2)
a3
```

Testar comandos de listas do Python

```
carnes = [‘picanha’, ‘alcatra’, ‘filé mignon’, ‘cupim’]
carnes2 = []
for c in carnes:
    carnes2.append(c)
carnes2.append(‘ponta de alcatra’)

carnes
carnes2

carnes = [‘picanha’, ‘alcatra’, ‘filé mignon’, ‘cupim’]
x = carnes
del (x[-1])

carnes

x
```