Linguagem de Programação Python

Programação Básica / Listas

Prof André Carvalho

Listas

- Listas são variáveis capazes de armazenar diversos valores ao mesmo tempo
- Listas podem ser vazias (sem nenhum elemento)
- Os valores armazenados em uma lista são chamados de elementos
- Listas homogêneas são listas, cujos elementos, têm todos o mesmo tipo
- Listas heterogêneas são listas cujos elementos podem ter tipos variados
- Listas podem ter elementos que também são listas
- Os elementos de uma lista são numerados da esquerda para a direita a partir de 0 (zero)
- Os elementos de uma lista são numerados da direita para a esquerda a partir de -1 (menos um)
- Listas podem ser modificadas, ou seja, podemos incluir, atualizar e remover elementos de uma lista

Exemplos de lista

Com base na quantidade de elementos:

- zero = [ ]
- um = [“Python”]
- dois = [“C”, “Python”]
- tres = [“C”, “Java”, “Python”]
- Etc

Com base na homogeneidade:

- homogenea = [2,3,5,7,11,13,17]
- heterogenea = [1,-7,1.3,True,“Python”]

Com elementos que são listas:

- homogenea = [[1,3],[2],[ ],[7,3,5]]
- heterogenea = [[2,4],1,[[6,3],8]]

Listas com dados de algo do mundo real

```
# dados de uma classe do ensino médio
#           N,Nome,              por,mat,qui,fis,bio,his,geo,soc,fil,ing,art
classe = [[ 1,"Joao da Silva", [9.0,8.5,7.5,4.5,8.5,9.0,6.5,8.5,9.0,6.0,9.5]],
          [ 2,"José de Souza", [7.5,4.5,9.0,6.0,8.5,9.0,6.5,8.5,9.5,9.0,8.5]],
          [ 3,"Luana Alves",   [9.5,9.0,8.5,7.5,8.5,9.0,6.5,8.5,4.5,9.0,6.0]],
          ...
          [40,"Maria Silva",   [8.5,9.0,7.5,4.5,9.0,6.0,9.0,8.5,6.5,8.5,9.5]]]
```

Algumas perguntas cabíveis e respondíveis via programação:

- Qual a média de um certo aluno?
- Qual a média de cada matéria?
- Quais as matérias que deram as menores notas?

Fatiamento ou slicing

- Listas suportam uma operação que chamamos de fatiamento ou slicing, em inglês
- Consiste de obter uma sub-lista de uma lista
- lista[pos1: pos2] resulta numa sub-lista da lista que contém os elementos da posição pos1 até a posição pos2-1
- Exemplos:

```
lista = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]

fatia = lista [4:8]
print (fatia) # [11,13,17,19]

fatia = [-10:-8]
print (fatia) # [53,59]

fatia[19:]
print (fatia) # [71,73,79,83,89,97]

fatia[:4]
print (fatia) # [2,3,5,7]
```

Fatiamento ou slicing

- O fatiamento ou slicing, em inglês, pode, facultativamente incluir um passo, ou step, em inglês
- No caso do passo não ser fornecido, como ocorreu no slide anterior, assume-se que ele vale 1
- lista[pos1: pos2: passo] resulta numa sub-lista da lista que contém os elementos da posição pos1 até a posição pos2-1, de "passo" em "passo"
- Acima, se o passo fornecido vale 2, tomaremos valores de 2 em 2; se o passo fornecido vale 3, tomaremos valores de 3 em 3, e assim por diante
- Exemplos:

```
lista = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]

fatia = lista [4:11:2]
print (fatia) # [11,17,23,31]

fatia = [-10:-3:3]
print (fatia) # [53,67,79]
```

Fatiamento ou slicing

- lista[:] ou lista[: :] resulta em uma lista com TODOS os elementos da lista original
- IMPORTANTE: a lista resultante será uma nova lista, ou seja, numa lista com os mesmos valores, mas com um novo id
- Exemplos:

```
lista = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]

fatia = lista [ : ]
print (fatia) # [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]

fatia = lista [ : : ]
print (fatia) # [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]
```

Manipulação de listas

Sendo **lista**, uma lista qualquer:

- len(**lista**) é uma função que resulta a quantidade de elementos da **lista**
- **lista**.append(**elemento**) é um método que acrescenta o **elemento** fornecido no final da **lista**
- **lista**.insert(**posicao**,**elemento**) é um método que insere o **elemento** fornecido NA da **posição** indicada, empurrando uma posição adiante o elemento que ALI se encontrava, bem como os elementos seguintes
- **lista**.remove(**elemento**) é um método que remove o elemento **indicado** da **lista**
- **Lista**.pop(**posicao**) é um método que resulta o elemento que se encontra na **posição** indicada da **lista**, removendo-o da **lista**; **posicao** pode ser omitida, caso em que o método resulta o elemento que se encontra na última posição da **lista**, removendo-o da **lista**
- del **lista**[**posicao**] é um comando que remove o elemento que se encontra na **posicao** indicada da **lista**
- **lista**.clear() é um método que remove todos os elementos da **lista**

Manipulação de listas

Sendo **lista**, **lista1** e **lista2**, listas quaisquer, sendo **n**, um número natural e sendo **coisa**, algo formado, eventualmente por vários elementos, como por exemplo uma lista, uma tupla, um dicionário ou um texto (string):

- **lista** = **lista1** + **lista2** é uma operação que gera uma **lista** com todos os elementos da **lista1**, seguidos por todos os elementos da **lista2** (chamamos esta operação de concatenação)

- **lista.**extend(**coisa**) é um método que acrescenta à **lista** todos os elementos da **coisa** (trata-se de uma espécie de concatenação)

- **lista** = **n** * **lista1** é uma operação que gera uma **lista** com os elementos da **lista1** concatenados **n** vezes

Manipulação de listas

Sendo **lista**, uma lista qualquer:

- **lista**.index(**elemento**,**inicio**,**final**) é um método que procura o elemento na **lista** a partir da posição **inicio** até a posição **final**, resultando a posição onde encontrar sua primeira aparição; **inicio** pode ser omitido, caso em que assume-se que **inicio** seja 0; **final** pode ser omitido, caso em que, assume-se que **final** seja a última posição da **lista**; lança a exceção **ValueError**, caso **elemento** não esteja presenta na **lista**

- **lista.**count(**elemento**) é um método que resulta quantas vezes **elemento** ocorre na **lista**
- **lista.**reverse() é um étodo que inverte ordem dos elementos da **lista**
- **lista.**copy() é um método que retorna uma cópia da **lista**, ou seja, uma lista nova, porém com os mesmos elementos da **lista**

Manipulação de listas

Sendo **lista**, uma lista qualquer:

- **lista**.sort() é um método que ordena os elemento na **lista** em ordem crescente; lança a exceção **TypeError**, caso a **lista** seja heterogênea; o fornecimento do parâmetro opcional **reverse=True**, torna a ordenação decrescente; o fornecimento do parâmetro opcional **key=subprog**, permite estabelecer um critério de ordenação, por exemplo:

```
lings=['JAVA','PYTHON','COBOL']

lings.sort()
print(lings) # ['COBOL',JAVA',PYTHON']

lings.sort(key=tamanho)
print(lings) # ['JAVA',COBOL',PYTHON']

lings.sort(key=tamanho,reverse=True)
print(lings) # ['PYTHON',COBOL',JAVA']
```

```
def tamanho (x):
    return len(x)
```

Compreensão de listas

- Muitas vezes precisamos iniciar uma lista com um conjunto de valores inicial que seguem uma regra de formação
- Em uma situação como esta podemos usar o que se conhece em Python por "compreensão de listas"
- Para tanto iniciamos a lista com uma expressão envolvendo uma variável que varia em um laço que especificamos diante da referida expressão (tudo entre colchetes, naturalmente)
- Exemplos:

```
lista = [0 for i in range(10)]
print (lista) # [0,0,0,0,0,0,0,0,0,0]

lista = [3*i for i in range(1,11,1)]
print (lista) # [3,6,9,12,15,18,21,24,27,30]
```

Compreensão de listas

- Exemplos usando if e, eventualmente, else:

```
lista = [3*i for i in range(1,11,1) if (3*i)%2==0]
print (lista) # [6,12,18,24,30]

lista = [3*i for i in range(1,11,1) if (3*i)%2==0 if 3*i!=18]
print (lista) # [6,12,24,30]

lista = [True if 3*i%2==0 else False for i in range(1,11,1)]
print (lista) # [False,True,False,True,False,True,False,True,False,True]
```

Compreensão de listas

- Exemplos com aninhamento:

```
matriz = [[1,2,3],\
          [4,5,6],\
          [7,8,9],\
          [0,1,2]]

# transformando linhas em colunas
matriz=[[linha[i] for linha in matriz] for i in range(len(matriz)+1]

'''
a matriz tornou-se:
[[1,4,7,0],\
 [2,5,8,1],\
 [3,6,9,2]]

'''
```