python™

Rafael Vieira Coelho

rafaelvc2@gmail.com

PARTE 1

8 - Listas

O que é uma lista?

- Uma lista é um conjunto ordenado de valores, onde cada valor é identificado por um índice.

- Os valores que compõem uma lista são chamados elementos.

- Listas são similares a *strings*, que são conjuntos ordenados de caracteres, com a diferença que os elementos de uma lista podem possuir qualquer tipo.

```
>>> bicycles = ['trek', 'cannondale', 'redline', 'specialized']
>>> print(bicycles)
['trek', 'cannondale', 'redline', 'specialized']
```

python™

Acessando elementos de uma lista

Listas são coleções ordenadas, portanto você pode acessar qualquer elemento de uma lista informando a posição – ou índice – do item desejado ao interpretador Python.

Para acessar um elemento de uma lista, escreva o nome da lista seguido do índice do item entre colchetes.

```
>>> bicycles = ['trek', 'cannondale', 'redline', ‘specialized’]
>>> print(bicycles[0])
trek
```

A posição dos índices começa em 0, e não em 1

Python considera que o primeiro item de uma lista está na posição 0, e não na posição 1.

As instruções a seguir acessam as bicicletas nos índices 1 e 3:

```
>>> bicycles = ['trek', 'cannondale', 'redline', 'specialized']
>>> print(bicycles[1])
cannondale
>>> print(bicycles[3])
specialized
```

Função range()

Python fornece uma maneira simples de criar inteiros consecutivos:

```
>>> numeros = list(range(1,5))
>>> print(numeros)
[1, 2, 3, 4]
```

Com um argumento simples, ela cria uma lista que inicia em 0:

```
>>> list(range(10))
[0,1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Se existe um terceiro argumento, ele especifica o espaço entre os valores sucessivos, que é chamado de tamanho do passo.

```
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
```

Obs: a função range não inclui o segundo argumento na lista!

python™

8.2 Acessado elementos

A sintaxe para acessar os elementos de uma lista é a mesma que a sintaxe para acessar os caracteres de uma string, o operator colchete ([ ]).

A expressão dentro dos colchetes especifica o índice. Lembre-se que os índices iniciam em 0:

```
>>> print (numeros[0])
>>> numeros[1]= 5
```

É comum utilizar uma variável de laço como um índice da lista:

```
cavaleiros = ['guerra', 'fome', 'peste', ‘morte']
i = 0
while i < 4:
        print (cavaleiros[i])
        i = i + 1
```

python™

8.3 Comprimento da lista

A função **len** devolve o comprimento de uma lista.

É uma boa idéia utilizar este valor como o limite superior de um laço ao invés de uma constante.

Desta forma, se o tamanho da lista mudar, você não precisará ir através de todo o programa modificando todos os laços; eles funcionarão corretamente para qualquer tamanho de lista:

```
cavaleiros = ['guerra', 'fome', 'peste', 'morte']
i = 0
while i < len(cavaleiros):
        print (cavaleiros[i])
        i = i + 1
```

8.4 Membros de uma lista

in é um operador lógico que testa se um elemento é membro de uma seqüência.

Já o usamos com strings, mas ele também funciona com listas e outras seqüências:

```
>>> cavaleiros = [’guerra’, ’fome’, ’peste’, ’morte’]

>>> ’peste’ in cavaleiros
True

>>> ’depravação’ in cavaleiros
False
```

8.5 Listas e laços for

O laço for é mais conciso que o while porque podemos eliminar a variável do laço, i.

Aqui está o laço anterior escrito com um laço for:

```
for cavaleiro in cavaleiros:
        print (cavaleiro)
```

Quase se lê como Português: “For (para cada) cavaleiro in (na lista de) cavaleiros, print (imprima o nome do) cavaleiro.”

8.5 Listas e laços for

Todas as opções de for

```
for cavaleiro in cavaleiros: # percorre toda lista sem se preocupar com o índice
        print (cavaleiro)

for i in len(cavaleiros):  # a função len() retorna o tamanho da lista
        print (cavaleiros[i])

for i in range(len(cavaleiros)):  # de 0 até o tamanho da lista
        print (cavaleiros[i])

for i in range(0, len(cavaleiros)): # de 0 até o tamanho da lista
        print (cavaleiros[i])
```

8.6 Operações em listas

O operador + concatena listas:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print (c)
[1, 2, 3, 4, 5, 6]
```

Similarmente, o operador * repete uma lista um número dado de vezes:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

O primeiro exemplo repete [0] quatro vezes.
O segundo exemplo repete a lista [1, 2, 3] três vezes.

8.7 Fatiamento de listas

A operação de fatiamento de strings também funciona sobre listas:

```
>>> lista = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’]
>>> lista[1:3]
['b', 'c']

>>> lista[:4]
['a', 'b', 'c', 'd']

>>> lista[3:]
['d', 'e', ‘f']

>>> copia_lista = lista[:]
>>> print(copia_lista)
['a', 'b', 'c', 'd', 'e', 'f']
```

python™

8.8 Listas são mutáveis

Diferentemente das strings, as listas são mutáveis, o que significa que podemos modificar seus elementos.

Utilizando o operador colchete no lado esquerdo de uma atribuição, podemos atualizar um de seus elementos:

```
>>> fruta = ["banana", "abacaxi", "laranja"]
>>> fruta[0] = "abacate"
>>> fruta[-1] = "tangerina"
>>> print (fruta)
['abacate', 'abacaxi', 'tangerina']
```

python™

8.8 Listas são mutáveis

Com o operador de fatiamento podemos atualizar vários elementos de uma vez:

```
>>> lista = ['a', 'b', 'c', 'd', 'e', 'f']
>>> lista[1:3] = [’x’, ’y’]
>>> print (lista)
['a', 'x', 'y', 'd', 'e', 'f']
```

Também podemos remover elementos de uma lista atribuindo a lista vazia a eles:

```
>>> lista = ['a', 'b', 'c', 'd', 'e', 'f']
>>> lista[1:3] = []
>>> print (lista)
[‘a', 'd', 'e', 'f']
```

8.9 Remoção em lista

Utilizando fatias para remover elementos pode ser complicado, e desta forma propenso a erro.

del remove um elemento de uma lista:

```
>>> a = ['um', 'dois', 'tres']
>>> del a[1]
>>> a
['um', 'tres']
```

Você também pode utilizar uma faixa como um índice para del:

```
>>> lista = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del lista[1:5]
>>> print (lista)
['a', 'f']
```

python™

8.9 Remoção em lista

- Removendo de uma determinada posição:

```
>>> motorcycles = ['honda', 'yamaha', 'suzuki']
>>> first_owned = motorcycles.pop(0)
>>> print('The first motorcycle I owned was a ' + first_owned.title() + '.')
>>> print(motorcycles)
['yamaha', 'suzuki']
```

- Removendo um determinado elemento da lista:

```
>>> motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
>>> motorcycles.remove('ducati')
>>> print(motorcycles)
['honda', 'yamaha', 'suzuki']
```

python™

8.10 Adição em lista

A partir de qualquer lista, podemos inserir um novo elemento utilizando o método append. Por exemplo, para inserir o 13 em nossa lista de primos podemos escrever:

```
>>> numeros = [1, 3, 7, 9]
>>> numeros.append(13)
>>> números
[1, 3, 7, 9, 13]

>>> motorcycles = ['honda', 'yamaha', 'suzuki']
>>> print(motorcycles)
['honda', 'yamaha', ‘suzuki']

>>> motorcycles.append('ducati')
>>> print(motorcycles)
['honda', 'yamaha', ‘suzuki’, 'ducati']
```

python™

8.10 Adição em lista

Podemos inserir em uma posição determinada.

```
>>> motorcycles = ['honda', 'yamaha', 'suzuki']
>>> print(motorcycles)
```

[‘honda', 'yamaha', ‘suzuki']

```
motorcycles.insert(0, 'ducati')
>>> print(motorcycles)
```

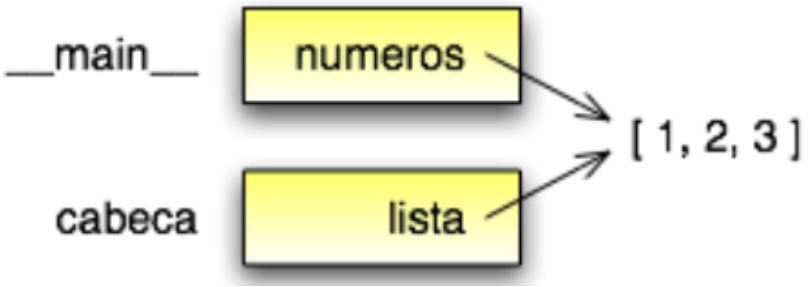
[‘ducati','honda', 'yamaha', ‘suzuki’]

8.11 Lista como parâmetro

Passar uma lista como um argumento passa realmente uma referência à lista, não uma cópia da lista.

Por exemplo, a função cabeca pega uma lista como parâmetro e devolve o seu primeiro elemento:

```
>>> def cabeca(lista):
        return lista[0]
```



Eis como ela é utilizada:

```
>>> numeros = [1, 2, 3]
>>> cabeca(numeros)
1
```

O parâmetro lista e a variável numeros são apelidos para o mesmo objeto.

python™

8.12 Matrizes

Listas aninhadas são freqüentemente utilizadas para representar matrizes. Por exemplo, a matriz:

poderia ser representada como:

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

matriz é uma lista com três elementos, onde cada elemento é uma linha da matriz. Podemos selecionar uma linha inteira da matriz:

```
>>> matriz[1]
[4, 5, 6]
```

Ou podemos extrair um único elemento da matriz utilizado a forma de duplo índice:

```
>>> matriz[1][1] #matriz[linha][coluna]
5
```

python™

8.13 Strings e listas

Duas das mais úteis funções no módulo string envolvem listas de strings. A função split (separar) quebra uma string em uma lista de palavras.

Por padrão, qualquer número de caracteres espaço em branco é considerado um limite de uma palavra:

```
import string

poesia = "O orvalho no carvalho..."
lista = poesia.split(" ")
for palavra in lista:
        print(palavra)
```

['O', 'orvalho', 'no', 'carvalho...']

8.14 Ordenação de Listas: sort()

O método sort() de Python faz com que seja relativamente fácil ordenar uma lista.

```
>>> cars = ['bmw', 'audi', 'toyota', ‘subaru']
>>> cars.sort()
>>> print(cars)
['audi', 'bmw', 'subaru', ‘toyota']
```

Também podemos ordenar essa lista em ordem alfabética inversa:

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> cars.sort(reverse=True)
>>> print(cars)
['toyota', 'subaru', 'bmw', 'audi']
```

python™

8.14 Ordenação de Listas: sorted()

Podemos ordenar, mas não alterar a lista:

```
>>> cars = ['bmw', 'audi', 'toyota', ‘subaru']

>>> print("Here is the original list:")
>>> print(cars)

>>> print("\nHere is the sorted list:")
>>> print(sorted(cars))

>>> print("\nHere is the original list again:")
>>> print(cars)
```

```
Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']

Here is the sorted list:
['audi', 'bmw', 'subaru', ‘toyota']

Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

python™

8.14 Ordenação de Listas: reverse()

Para inverter a ordem original de uma lista, podemos usar o método reverse().

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> print(cars)
```

['bmw', 'audi', 'toyota', 'subaru']

```
>>> cars.reverse()
>>> print(cars)
```

['subaru', 'toyota', 'audi', 'bmw']

8.15 Estatísticas simples com uma lista de números

Encontrando o menor (min), o maior (max) e a soma (sum) dos elementos da lista.

```
>>> digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> min(digits)
0
>>> max(digits)
9
>>> sum(digits)
45
```

Ordenação de Listas

```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
```

```
>>> sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]
```

```
sorted([3,2,1,4,5,0],reverse=True)
[5, 4, 3, 2, 1, 0]
>>> sorted([3,2,1,4,5,0])
[0, 1, 2, 3, 4, 5]
```

python™

Erros Comuns com Listas

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[3])
```

Esse exemplo resulta em um *erro de índice*:

```
Traceback (most recent call last):
 File "motorcycles.py", line 3, in <module>
  print(motorcycles[3])
IndexError: list index out of range
```

```
motorcycles = []
print(motorcycles[-1])
```

Não há nenhum item em `motorcycles`, portanto Python devolve outro erro de índice:

```
Traceback (most recent call last):
 File "motorcyles.py", line 3, in <module>
  print(motorcycles[-1])
IndexError: list index out of range
```

```
File "magicians.py", line 3
 print(magician)
    ^
IndentationError: expected an indented block
```

python™

Passando Lista como Parâmetro

```
def greet_user(names):
    for name in names:
        msg = "Olá, " + name.title() + "!"
        print(msg)

def main():
  names = [‘hanna’, ‘jane’, 'margot']
  greet_user(names)

main()
```

```
Olá, Hannah!
Olá, Ty!
Olá, Margot!
```

E se eu modificar a lista dentro da função?

Se precisarmos alterar, mas não queremos modificar a lista original, devemos usar greet_user(names[:])

python™

Exercícios

0) Armazene os nomes de alguns de seus amigos em uma lista chamada names. Exiba o nome de cada pessoa acessando cada elemento da lista, um de cada vez. E juntamente uma mensagem. O texto de cada mensagem deve ser o mesmo, porém cada mensagem deve estar personalizada com o nome da pessoa.

Exercícios

1) Faça um Programa que leia 4 notas, mostre as notas e a média na tela.

2) Faça um Programa que leia 10 caracteres separadamente (um por vez), e diga quantas consoantes foram lidas. Imprima as consoantes.

3) Utilizando listas faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são:
"Telefonou para a vítima?"
"Esteve no local do crime?"
"Mora perto da vítima?"
"Devia para a vítima?"
"Já trabalhou com a vítima?"

O programa deve no final emitir uma classificação sobre a participação da pessoa no crime. Se a pessoa responder positivamente a 2 questões ela deve ser classificada como "Suspeita", entre 3 e 4 como "Cúmplice" e 5 como "Assassino". Caso contrário, ele será classificado como "Inocente".

Exercícios

4) Uma empresa de pesquisas precisa tabular os resultados da seguinte enquete feita a um grande quantidade de organizações: "Qual o melhor Sistema Operacional para uso em servidores?"

As possíveis respostas são:
1- Windows Server
2- Unix
3- Linux
4- Netware
5- Mac OS
6- Outro

Você foi contratado para desenvolver um programa que leia o resultado da enquete e informe ao final o resultado da mesma.

Exercícios: continuação da questão 4

O programa deverá ler os valores até ser informado o valor 0, que encerra a entrada dos dados. Não deverão ser aceitos valores além dos válidos para o programa (0 a 6). Os valores referentes a cada uma das opções devem ser armazenados num vetor. Após os dados terem sido completamente informados, o programa deverá calcular a percentual de cada um dos concorrentes e informar o vencedor da enquete. O formato da saída foi dado pela empresa, e é o seguinte:

SO	Votos	%
Windows Server	1500	17%
Unix	3500	40%
Linux	3000	34%
Netware	500	5%
Mac OS	150	2%
Outro	150	2%
Total	8800	

O SO mais votado foi o Unix, com 3500 votos, correspondendo a 40% dos votos.