FUNDAMENTOS DE PROGRAMAÇÃO

Listas

Prof. Bruno Góis Mateus

Índice

INTRODUÇÃO

Introdução

- Uma lista é como se fosse uma coleção
  - Permite guardar vários valores em uma única variável
  - Os valores/elementos são armazenados de maneira ordenada
  - Cada valor é identificado pelo seu índice
  - Os elementos da lista não precisam ser do mesmo tipo

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]
carryon = [ 'socks', 'shirt', 'perfume' ]
```

CRIANDO UMA LISTA

Criando uma lista

- Existem diversas maneira de criar uma lista

```
lista1 = [10, 20, 30, 40]
lista2 = ["spam", "bungee", "swallow"]
lista3 = [10, "spam", 30]
lista4 = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
lista5 = list()
lista6 = []
```

Adicionando elementos na lista

- Podemos criar uma lista vazia e adicionar elementos
  - Para isso utilizamos o método append
    - Os elementos são adicionados no final da lista
  - A lista mantém-se na ordem de adição

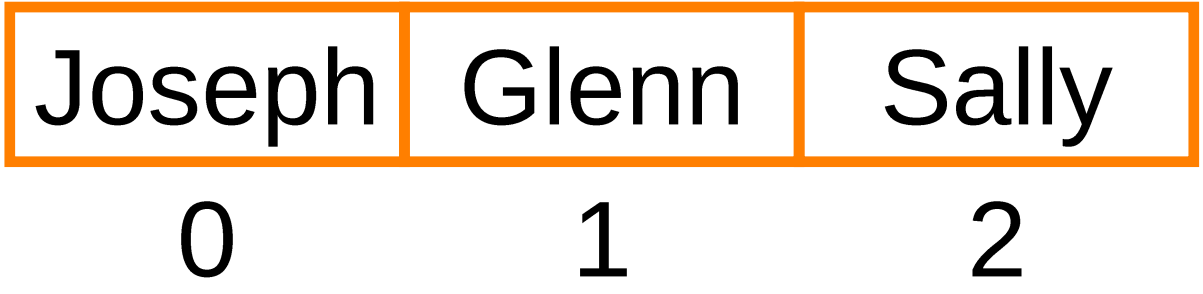
```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print stuff
['book', 99]
>>> stuff.append('cookie')
>>> print stuff
['book', 99, 'cookie']
```

OLHANDO DENTRO DE UMA LISTA

Olhando para dentro de uma lista

- A sintaxe para acessar um elemento é a mesma utilizada em string
  - É utilizado o operador de índice [ ] (não confundir com uma lista vazia)
    - A expressão dentro dos colchetes especificam o índice
    - O índice inicia em zero

Olhando dentro de uma lista




```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print friends[1]
Glenn
```

Olhando para dentro de uma lista

```
alist = [3, 67, "cat", [56, 57, "dog"], [ ], 3.14, False]
print(alist[5])
```

a) [ ]
b) 3.14
c) False

Olhando para dentro de uma lista

```
alist = [3, 67, "cat", [56, 57, "dog"], [ ], 3.14, False]
print(alist[2].upper())
```

a) Erro
b) 2
c) CAT

Olhando para dentro de uma lista

```
alist = [3, 67, "cat", [56, 57, "dog"], [ ], 3.14, False]
print(alist[2][0])
```

a) 56
b) c
c) cat
d) erro

Qual o tamanho da lista ?

- Utilizamos a função len para descobrir o tamanho de uma lista

```
alist = [3, 67, "cat", 3.14, False]
print(len(alist))
>>> 5
alist = [3, 67, "cat", [56, 57, "dog"], [ ], 3.14, False]
print(len(alist))
>>> 7
```

OPERAÇÕES COM LISTAS

Concatenação e repetição

- Concatenando +
  - Podemos criar uma nova lista juntando duas outras existentes
- Repetição *
  - Repete os itens de uma lista um dado número de vezes
- Ambas operações resultam em uma nova lista

```
fruit = ["apple", "orange", "banana", "cherry"]
print([1, 2] + [3, 4])
print(fruit + [6, 7, 8, 9])
print([0] * 4)
print([1, 2, ["hello", "goodbye"]] * 2)
```

Está contido na lista ?

- Python prover dois operadores para checar se um item está contido ou não em uma lista
  - in
  - not in
  - São operadores lógicos
    - Retorna True ou Flase
  - Não alteram a lista

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
```

Slicing

- Funciona da mesma forma que usando String

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

LISTA SÃO MUTÁVEIS

Lista são mutáveis

- Ao contrário das String, listas são mutáveis
  - Podemos alterar diretamente um elemento da lista
    - Utilizamos o operador de índice
  - Em String, precisamos criar uma nova string

Lista são mutáveis

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> x = fruit.lower()
>>> print x
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print lotto
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print lotto
[2, 14, 28, 41, 63]
```

Lista são mutáveis

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
alist[1:3] = ['x', 'y']
print(alist)
alist = ['a', 'b', 'c', 'd', 'e', 'f']
alist[1:3] = []
print(alist)
alist = ['a', 'b', 'c', 'd', 'e', 'f']
alist[1:3] = []
print(alist)
alist = ['a', 'd', 'f']
alist[1:1] = ['b', 'c']
print(alist)
alist[4:4] = ['e']
print(alist)
```

List são mutáveis

- Utilizar o operador de slice para deletar deixa o código pouco legível e mais passivo a errros
- Python prover uma opção mais legível
  - A função del remove um elemento da lista
    - Utiliza a posição do elemento

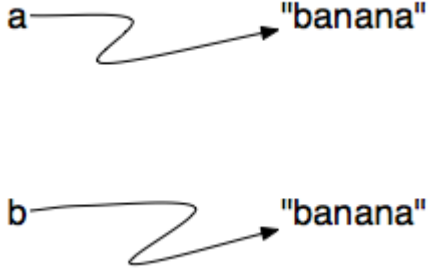
```
a = ['one', 'two', 'three']
del a[1]
print(a)

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

Objetos e referências

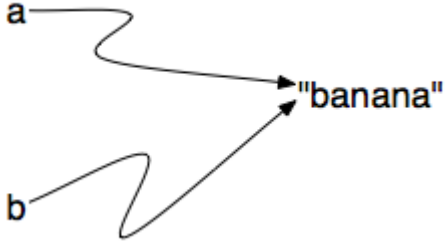
- Considere o código abaixo:

```
a = "banana"
b = "banana"
```

- Existem duas possíveis de organizar esses dados internamente



ou



Objetos e referências

- Podemos conferir da seguinte forma
  - Verificando o id de cada objeto
    - Para isso usamos a função id
  - Utilizar o operador is

```
a = "banana"
b = "banana"
print id(a)
>>> 1396361322142256
print id(b)
>>> 1396361322142256
print b is a
>>> True
```

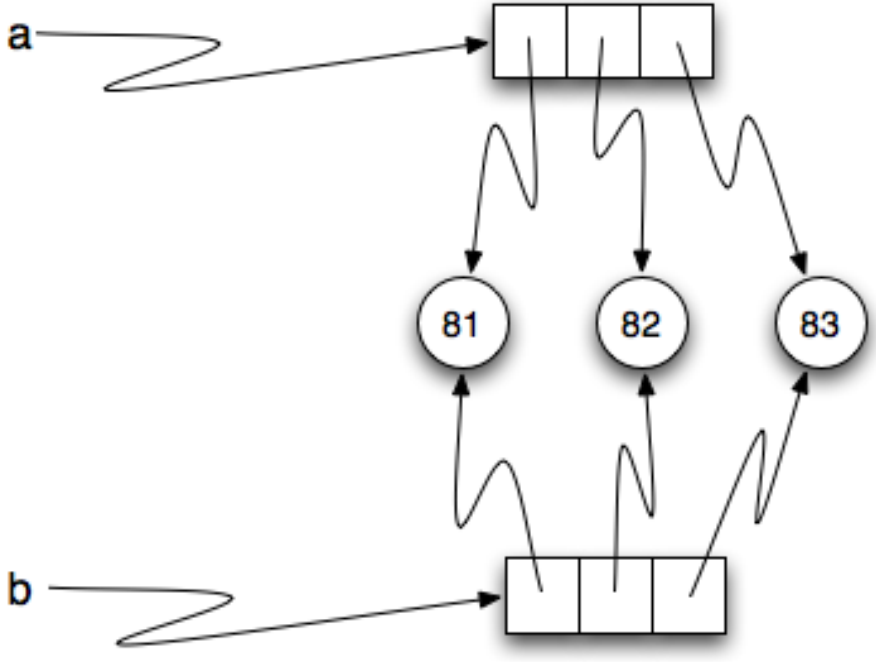
Objetos e referências

- Vejamos agora no caso de listas

```
a = [81, 82, 83]
b = [81, 82, 83]
print(a is b)
>>> False
print(a == b)
>>> True
```

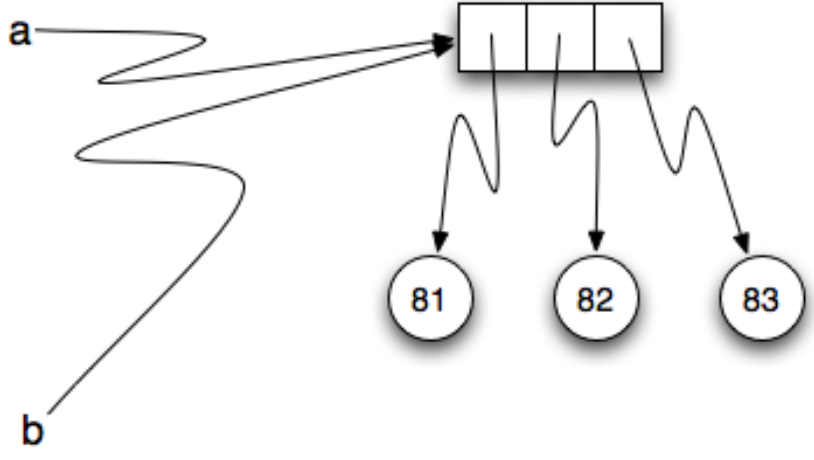
Objetos e referências

- Listas são uma coleção de referências



Aliasing

```
a = [81, 82, 83]
b = a
print(a is b)
>>> True
print(a == b)
>>> True
b[0] = 5
print a
>>> [5, 82, 83]
```



Clonando uma lista

- Quando queremos alterar uma lista e mantar a original é necessário clonar a lista
- A maneira mais de fácil de fazer isso é usando o operador de slice

Clonando uma lista

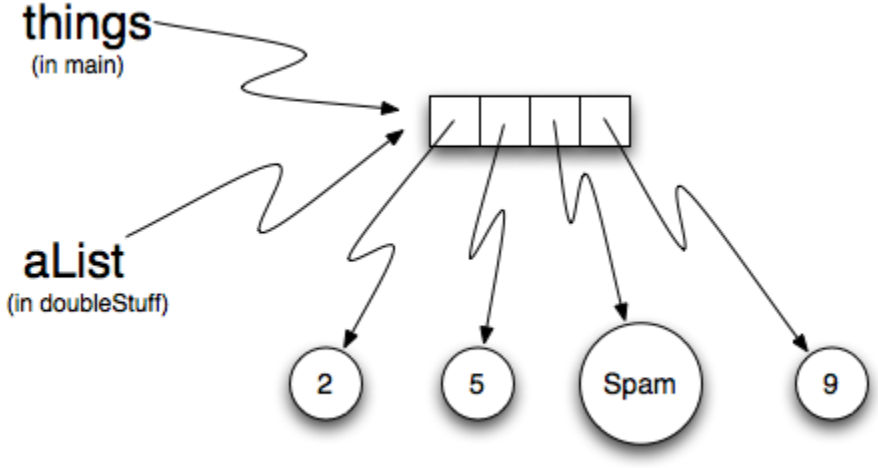
```
a = [81, 82, 83]
b = a[:]
print(a is b)
>>> False
print(a == b)
>>> True
b[0] = 5
print a
>>> [81, 82, 83]
print b
>>> [5, 82, 83]
```

Usando lista como parâmetros

- Funções que recebem uma lista como parâmetro e as modifica são chamadas de modificadoras
  - Possuem efeito colateral
- Ao passar uma lista como parâmetro estamos passando a referência e não uma cópia

Usando lista como parâmetros

```
def doubleStuff(aList):
    """ Overwrite each element in aList with double its value.
"""
    for position in range(len(aList)):
        aList[position] = 2 * aList[position]

things = [2, 5, 9]
print(things)
doubleStuff(things)
print(things)
```



Funções puras

- São aquelas funções que não produzem efeitos colaterais

```
def doubleStuff(a_list):
    """ Return a new list in which contains doubles of the
elements in a_list. """
    new_list = []
    for value in a_list:
        new_elem = 2 * value
        new_list.append(new_elem)
    return new_list

things = [2, 5, 9]
print(things)
things = doubleStuff(things)
print(things)
```

MÉTODOS DE LISTA

Métodos de lista

- O python nos fornece diversas métodos built-in para trabalhar com listas
  - Para utilizá-las, adicionamos um ponto final após o nome da variável e em seguida colocamos o nome da função
    - Similar ao que fazemos com string

Métodos de lista

Método	Parâmetros	Descrição
append	item	Adiciona o novo elemento ao final da lista
insert	Posição, item	Adiciona o novo elemento na posição informada
pop	nada	Remove e retorna o último elemento da lista
pop	Posição	Remove e retorna o item da posição
sort	nada	Ordena a lista
reverse	nada	Inverte a ordenação da lista
index	item	Retorna o índice da primeira ocorrência do item
count	item	Retorna o número de ocorrência do item
remove	item	Remove a primeira ocorrência do item

Métodos da lista

```
alist = [4, 2, 8, 6, 5]
alist.append(True)
alist.append(False)
print(alist)
```

a) [4, 2, 8, 6, 5, False, True]

b) [4, 2, 8, 6, 5, True, False]

c) [True, False, 4, 2, 8, 6, 5]

Métodos da lista

```
alist = [4, 2, 8, 6, 5]
alist.insert(2, True)
alist.insert(0, False)
print(alist)
```

a) [False, 4, 2, True, 8, 6, 5]

b) [4, False, True, 2, 8, 6, 5]

c) [False, 2, True, 6, 5]

Métodos da lista

```
alist = [4, 2, 8, 6, 5]
temp = alist.pop(2)
temp = alist.pop()
print(alist)
```

a) [4, 8, 6]

b) [2, 6, 5]

c) [4, 2, 6]

Ordenando uma lista

- Uma lista pode conter vários itens
  - Estes são mantidos em uma ordem
  - Podemos alterar essa ordem a qualquer momento

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort()
>>> print friends
['Glenn', 'Joseph', 'Sally']
>>> print friends[1]
Joseph
```

Outra funções built-in

- O Python fornece ainda algumas funções built-in úteis para trabalhar com string

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print len(nums)
6
>>> print max(nums)
74
>>> print min(nums)
3
>>> print sum(nums)
154
>>> print sum(nums)/len(nums)
25
```

LISTAS ANINHADAS E MATRIZES

Lista aninhadas

- Uma lista aninhada é uma lista que é elemento de outra lista
- Podemos acessar os elementos de uma lista aninhada de basicamente duas maneiras
  - Buscando a lista interna e depois escolhendo o elemento
  - Acessando diretamente o elemento da lista interna

Lista aninhadas

```
lista = ["hello", 2.0, 5, [10, 20]]
lista_interna = lista[3]

print(lista_interna)
item = lista_interna[1]
print(item)

print(lista[3][1])
```

Listas aninhadas

- Podemos ter quantos níveis quisermos de aninhamento
  - Para cada nível adicionamos um par de colchetes para se quisermos acessar diretamente os elementos das listas internas

Listas aninhadas

```
lista1 = [0, 1]
lista2 = [lista1]
lista3 = [lista1, lista2]
lista4 = [lista1, lista2, lista3]

print lista1[1]
print lista2[0][1]
print lista3[0][1], lista3[1][0][1]
print lista4[0][1], lista4[1][0][1], lista4[2][0][1],
lista4[2][1][0][1]
```

Matrizes

- Matriz é uma tabela composta por linhas e colunas
  - Dizemos que uma matriz é $m \, X \, n$ quando ela possuí:
    - M linhas
    - N colonas
- Esse tipo de estrutura é bastante utilizada em computação gráfica
  - Em geral utilizamos lista de listas para representar matrizes

Matriz

- Matriz quadrada:
  - Número de linhas igual ao número de colunas

```
m1 = [[1, 2], [3, 4]]
m2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

- Matriz identidade
  - Matriz onde todos os elementos da diagonal principal são iguais a 1 e as demais iguais a 0

```
identidade = [[1, 0 , 0], [0, 1, 0], [0, 0, 1]]
```

Matriz

- Percorrendo uma matriz

```
for linha in m:
    for item in linha:
        print item,
    print "\n"
```

Matriz

- Percorrendo uma matriz

```
for i in range(len(m)):
    for j in range(len(m[i])):
        print m[i][j],
    print "\n"
```

LIST COMPREHENSIONS

List comprehensions

- Uma maneira concisa de criar listas

Sintaxe: [expressão for item in sequência if condição]

```
mylist = [1,2,3,4,5]

yourlist = [item ** 2 for item in mylist]

print(yourlist)
```

List comprehensions

```
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
]

m2 = [[row[i] for row in matrix] for i in range(4)]
print m2
```

O que vem por aí

- Arquivos