

Listas

Prof. Alberto Costa Neto
Programação em Python

Uma Lista é um tipo de Coleção



- Uma coleção permite colocar vários valores em um única “variável”
- Coleções são práticas porque permitem carregar muitos valores empacotados de forma conveniente pelo programa

```
amigos = [ 'Joseph', 'Glenn', 'Sally' ]
```

```
bagagem = [ 'meia', 'camisa', 'perfume' ]
```

O que **não** é uma “Coleção”?

A maioria de nossas **variáveis** tem um valor apenas – quando um novo valor é colocado nela, o antigo é **sobrescrito**

```
$ python  
>>> x = 2  
>>> x = 4  
>>> print x  
4
```

Constantes List

- **Listas** são circundadas por colchetes e seus elementos são separados entre si por vírgula
- Um elemento de uma **lista** pode ser qualquer objeto de Python – até mesmo **outra lista**
- Uma **lista** pode estar vazia

```
>>> print [1, 24, 76]
[1, 24, 76]
>>> print ['red', 'yellow', 'blue']
['red', 'yellow', 'blue']
>>> print ['red', 24, 98.6]
['red', 24, 98.599999999999994]
>>> print [ 1, [5, 6], 7]
[1, [5, 6], 7]
>>> print []
[]
```

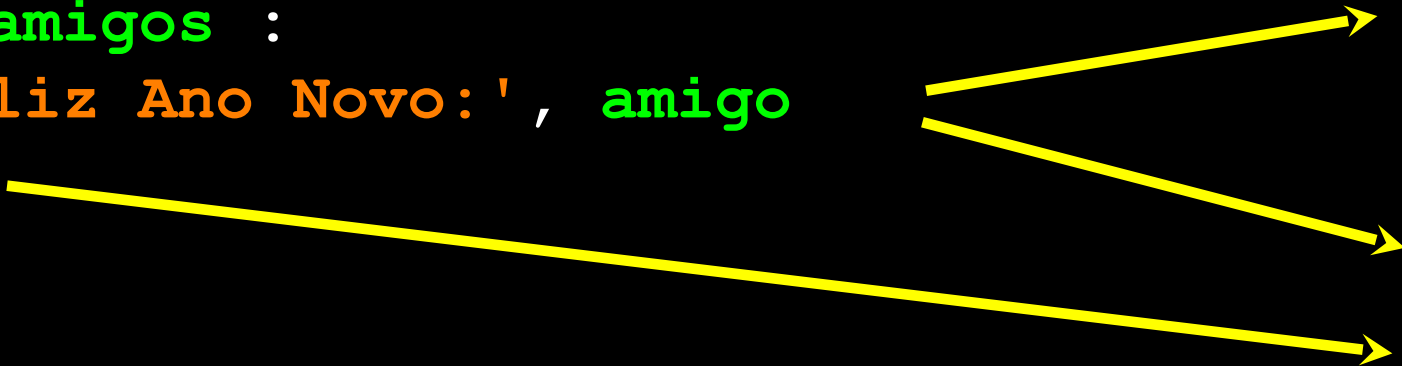
Nós já usamos listas!

```
for i in [5, 4, 3, 2, 1] :  
    print i  
print 'Fim!'
```

5
4
3
2
1
Fim!

Listas e laços definidos (grandes parceiros)

```
amigos = ['Joseph', 'Glenn', 'Sally']  
for amigo in amigos :  
    print 'Feliz Ano Novo:', amigo  
print 'Fim!'
```



Feliz Ano Novo: Joseph
Feliz Ano Novo: Glenn
Feliz Ano Novo: Sally
Fim!



Detalhes internos das Listas

Da mesma forma que as strings, nós podemos acessar qualquer elemento na lista usando seu índice entre **colchetes**

| | | |
|--------|-------|-------|
| Joseph | Glenn | Sally |
| 0 | 1 | 2 |

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

Listas são Mutáveis

- Strings são “imutáveis” - não podemos mudar o conteúdo de uma string, mas podemos criar uma nova string com as mudanças desejadas
- Listas são “mutáveis” - podemos modificar qualquer elemento de uma lista. Para isso basta utilizar seu índice

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


Qual é o Comprimento da Lista?

- A função `len()` pode receber uma *lista* como parâmetro e retorna o número de *elementos* na *lista*.
- Na verdade `len()` informa o número de elementos de qualquer conjunto ou seqüência (já vimos que funciona para string...)

```
>>> cump = 'Ola Bob'
>>> print len(cump)
7
>>> x = [1, 2, 'joe', 99]
>>> print len(x)
4
>>>
```

Usando a função range

- A função `range` retorna uma lista de números que variam de zero até um a menos que o parâmetro
- Assim fica fácil construir um laço `for` e dispor de índices

```
>>> print range(4)
[0, 1, 2, 3]
>>> amigos = ['Joseph', 'Glenn', 'Sally']
>>> print len(amigos)
3
>>> print range(len(amigos))
[0, 1, 2]
>>>
```

Laços para todos os gostos...

```
amigos = ['Joseph', 'Glenn', 'Sally']
```

```
for amigo in amigos :  
    print 'Feliz Ano Novo:', amigo
```

```
for i in range(len(amigos)) :  
    amigo = amigos[i]  
    print 'Feliz Ano Novo:', amigo
```

```
>>> amigos = ['Joseph', 'Glenn', 'Sally']
```

```
>>> print len(amigos)
```

```
3
```

```
>>> print range(len(amigos))
```

```
[0, 1, 2]
```

```
>>>
```

Feliz Ano Novo: Joseph

Feliz Ano Novo: Glenn

Feliz Ano Novo: Sally

Concatenando listas usando +

- Podemos criar uma nova lista a partir da junção dos elementos de 2 listas existentes

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

Listas podem ser **partidas** (**sliced**) usando :

```
>>> 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]
```

Lembrete: *Da mesma forma que em strings, o segundo número é “até, mas não o incluindo”*

Métodos (funções) de Listas

```
>>> x = list()
>>> type(x)<type 'list'>
>>> dir(x) ['append', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

Criando uma lista do início

- Podemos criar uma lista vazia e então adicionar os elementos usando o método `append`
- A lista permanece na ordem de inserção. Novos elementos são adicionados no final da lista

```
>>> lista = list()
>>> lista.append('livro')
>>> lista.append(99)
>>> print lista
['livro', 99]
>>> lista.append('biscoito')
>>> print lista
['livro', 99, 'biscoito']
```

Há algo na Lista?

- Python provê dois operadores (`in` e `not in`) que permitem checar, respectivamente, se um item está ou não em uma lista
- Eles são operadores lógicos (retornam `True` ou `False` e não modificam a lista)

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


Uma **Lista** é uma seqüência **Ordenada**

- Uma **lista** guarda os elementos e mantém na mesma ordem em que foram inseridos ou atribuídos
- Uma **lista** pode ser **ordenada** (mudando sua ordem)
- O método **sort** (diferente das strings) significa “**ordene a si mesmo**”

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

Funções Built-in e Listas

- Existem várias funções predefinidas (built-in) em Python que aceitam listas como parâmetro
- Lembram dos idiomas para os laços? Estas funções são muito mais simples!

```
>>> 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
```

```
total = 0
cont = 0
while True :
    valor = raw_input('Numero: ')
    if valor == 'fim' : break
    num = float(valor)
    total = total + num
    cont = cont + 1
```

```
media = total / cont
print 'Media:', media
```

Numero: 3

Numero: 9

Numero: 5

Numero: fim

Media: 5.6666666666667

```
numlist = list()
```

```
while True :
    valor = raw_input('Numero: ')
    if valor == 'fim' : break
    num = float(valor)
    numlist.append(num)
```

```
media = sum(numlist) / len(numlist)
print 'Media:', media
```

Melhores Amigos: Strings e Listas

```
>>> abc = 'Com tres palavras'
>>> palavras = abc.split()
>>> print palavras
['Com', 'tres', 'palavras']
>>> print len(palavras)
3
>>> print palavras[0]
Com
```

```
>>> print palavras
['Com', 'tres', 'palavras']
>>> for p in palavras :
...     print p
...
Com
tres
palavras
>>>
```

`split` quebra uma string em partes e produz uma lista de strings. Podemos pensar em algo análogo a quebrar uma frase em uma lista de palavras. Podemos acessar uma palavra específica ou `iterar` sobre todas as palavras

```
>>> linha = 'Um monte                de espaços'
>>> palavras = linha.split()
>>> print palavras
['Um', 'monte', 'de', 'espaços']
>>>
>>> linha = 'primeiro;segundo;terceiro'
>>> algo = linha.split()
>>> print algo
['primeiro;segundo;terceiro']
>>> print len(algo)
1
>>> algo = linha.split(';')
>>> print algo
['primeiro', 'segundo', 'terceiro']
>>> print len(algo)
3
>>>
```

- Quando não é especificado um **delimitador**, múltiplos espaços são tratados como um delimitador
- Podemos especificar qual **caractere delimitador** desejamos usar passando-o como parâmetro na chamada a **split**

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
arq = open('mbox-short.txt')
for linha in arq:
    linha = linha.rstrip()
    if not linha.startswith('From ') : continue
    palavras = linha.split()
    print palavras[2]
```

Sat
Fri
Fri
Fri
...

```
>>> linha = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> palavras = linha.split()
>>> print palavras
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>
```

Padrão Particionamento Duplo

“Double Split”

- Às vezes particionamos uma linha (ou string) uma vez, e depois pegamos os pedaços e fazemos uma nova partição

From `stephen.marquard@uct.ac.za` Sat Jan 5 09:14:16 2008

```
palavras = linha.split()  
email = palavras[1]
```

Padrão Particionamento Duplo

“Double Split”

- Às vezes particionamos uma linha (ou string) uma vez, e depois pegamos os pedaços e fazemos uma nova partição

From `stephen.marquard@uct.ac.za` Sat Jan 5 09:14:16 2008

```
palavras = linha.split()  
email = palavras[1]
```

`stephen.marquard@uct.ac.za`

Padrão Particionamento Duplo

“Double Split”

- Às vezes particionamos uma linha (ou string) uma vez, e depois pegamos os pedaços e fazemos uma nova partição

From `stephen.marquard@uct.ac.za` Sat Jan 5 09:14:16 2008

```
palavras = linha.split()  
email = palavras[1]  
pedacos = email.split('@')
```

```
stephen.marquard@uct.ac.za  
['stephen.marquard', 'uct.ac.za']
```

Padrão Particionamento Duplo

“Double Split”

- Às vezes particionamos uma linha (ou string) uma vez, e depois pegamos os pedaços e fazemos uma nova partição

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
palavras = linha.split()
email = palavras[1]
pedaces = email.split('@')
print pedacos[1]
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
'uct.ac.za'
```



Acknowledgements / Contributions

Agradecimentos / Contribuições



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information



These slides were translated and adapted by Alberto Costa Neto (albertocn.sytes.net) of the Federal University of Sergipe