universidade de aveiro
escola superior de tecnologia e
gestão de águeda

Introdução à Programação

Listas

cbt@ua.pt - RSI
pop@ua.pt - CS

Listas

universidade de aveiro
escola superior de tecnologia e gestão de águeda

Tal como uma string, uma lista é uma sequência de valores.

Os valores de uma lista podem ser de qualquer tipo e são chamados de elementos ou itens.

Existem várias formas de criar uma nova lista.

O mais simples é colocar os elementos entre parêntesis retos [ ]

[10, 20, 30, 40]

['crunchy frog', 'ram bladder', 'lark vomit’]

Os elementos de uma lista não precisam ser do mesmo tipo.

['spam', 2.0, 5, [10, 20]]

Uma lista que não contém elementos é chamada de lista vazia e é representada por parêntesis retos vazios [].

Listas são mutáveis

A sintaxe para aceder os elementos de uma lista é a mesma para aceder a caracteres de uma string.

A expressão dentro dos parêntesis especifica o índice. Não esquecer que os índices começam em 0.

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda’]
>>> cheeses[0]
'Cheddar’
```

Ao contrário das strings, as listas são mutáveis. O que permite alterar os seu valores.

```
>>> numbers = [42, 123]
>>> numbers[1] = 5
>>> numbers
[42, 5]
```

Listas são mutáveis

Os índices de uma lista funcionam da mesma maneira que os índices de string:

- Qualquer expressão inteira pode ser usada como índice.
- Ao tentar ler ou escrever um elemento que não existe, recebemos um IndexError.
- Se um índice tiver um valor negativo, ele conta para trás a partir do final da lista.

O operador in também funcionada em listas.

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

Percorrendo uma lista

A maneira mais comum de percorrer os elementos de uma lista é com um loop *for*.

A sintaxe é a mesma das strings:

```
for cheese in cheeses:
    print(cheese)
```

Para escrever ou atualizar os elementos, precisamos dos índices. Uma maneira comum de fazer isso é combinar as funções internas range e len:

```
numbers = [42, 123]
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```

len retorna o número de elementos na lista.

range retorna uma lista de índices de 0 a n - 1, onde n é o comprimento da lista.

Percorrendo uma lista

A iteração sobre uma lista vazia, nunca executa o corpo do código

```
for x in []:
    print('This never happens.')
```

Embora uma lista possa conter outra lista, a lista contida conta como um único elemento.

O comprimento desta lista é quatro:

```
['spam', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]
```

Operações em listas

O operador + concatena listas:

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

O operador * repete a lista um determinado número de vezes:

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

Sublistas

Assim como nas strings também podemos criar sublistas a partir de uma dada lista

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f’]
```

Ao omitir o primeiro índice, a sublista começará no início. Se você omitir o segundo, a sublista irá até ao fim. Ao omitir ambos, a sublista será uma cópia de toda a lista.

```
>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

Métodos das listas

O Python fornece métodos que operam em listas.

append adiciona um novo elemento ao final de uma lista:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> t
['a', 'b', 'c', 'd’]
```

extend recebe uma lista como argumento e acrescenta todos os elementos:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> t1
['a', 'b', 'c', 'd', 'e']
```

Métodos das listas

universidade de aveiro
escola superior de tecnologia e gestão de águeda

sort organiza os elementos da lista de baixo para cima:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> t
['a', 'b', 'c', 'd', 'e']
```

Apagar elementos da listas

Existem várias maneiras de excluir elementos de uma lista.

Sabendo o índice do elemento que deseja apagar pode usar o método **pop**:

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> t
['a', 'c']
>>> x
'b'
```

Se não for dado um índice o elemento removido será o último elemento da lista.

Se não for necessário guardar o valor removido podemos usar o operador **del**:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> t
['a', 'c']
```

Apagar elementos da listas

Se conhecermos o elemento que se quer eliminar mas não o seu índice podemos usar o **remove**:

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> t
['a', 'c’]
```

Para remover mais do que um element podemos usar o operador sublista:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> t
['a', 'f']
```

Listas e String

Uma string é uma sequência de caracteres e uma lista é uma sequência de valores, mas uma lista de caracteres não é o mesmo que uma string.

Para converter de uma string para uma lista de caracteres, podemos usar a função **list**:

```
>>> s = 'spam'
>>> t = list(s)
>>> t
['s', 'p', 'a', 'm’]
```

A função list quebra uma string em letras individuais. Para quebrar uma string em palavras, usamos o método **split**:

```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> t
['pining', 'for', 'the', 'fjords']
```

Cópia de Listas

Não é possível copiar uma lista simplesmente fazendo

```
lista2 = lista1
```

A lista2 será apenas uma referência à lista1 e as alterações feitas na lista1 serão automaticamente feitas também na lista2.

Para copiar uma lista, podemos utilizar:

- O método copy()

```
lista2 = lista1.copy()
```

- O método list()

```
lista2 = list(lista1)
```

- O operador :

```
lista2 = lista1[:]
```