Programação em Python

Listas e tuplos

2023

Departamento de Ciência de Computadores

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Conteúdo

Listas

Listas

- Sequências ordenadas possivelmente com repetições
- Podem conter elementos de quaisquer tipos
- Os elementos são identificados pelos índices

Listas por extensão

- Lista com *n* elementos: `[e1, e2, ..., en]`
- A ordem é significativa
- Podem ocorrer elementos repetidos
- Pode ser a lista vazia: `[]`

Operações básicas

- comprimento

```
>>> len([1,'dois',3])
3
```

- concatenação

```
>>> [1,'dois',3]+[4,5,6]
[1,'dois',3,4,5,6]
```

- repetição

```
>>> 2*[1,'dois',3]
[1,'dois',3,1,'dois',3]
```

- pertença

```
>>> 3 in [1,'dois',3]
True
```

- iteração

```
>>> for x in [1,'dois',3]:
        print(x)
1
dois
3
```

Acesso aos elementos

- Operador de indexação: `lista[i]`
- Índices entre 0 e `len(lista)-1`
- Índices negativos: acesso a partir do fim
- Índices inválidos dão um erro de execução

Exemplo

```
>>> alimentos = ['pão', 'pão', 'queijo', 'queijo']
>>> alimentos[0]
'pão'
>>> alimentos[1]
'pão'
>>> alimentos[2]
'queijo'
>>> len(alimentos)
4
```

Fatias

`lst[i:j]`	elementos entre i e $j-1$ inclusíve
`lst[i:]`	elementos entre i até ao final
`lst[:j]`	elementos do primeiro até $j-1$ inclusíve
`lst[:]`	todos os elementos (cópia da lista)

```
>>> vogais = ['a','e','i','o','u']
>>> vogais[1:4]
['e', 'i', 'o']
>>> vogais[:3]
['a','e','i']
>>> vogais[3:]
['o','u']
>>> vogais[:]
['a', 'e', 'i', 'o', 'u']
```

Fatias (cont.)

Forma geral

`lst[i:j:k]` elementos de i a $j - 1$ com incrementos k

Incrementos negativos: percorrer a lista ao contrário.

```
>>> vogais[::2]        # índices pares
['a','i','u']
>>> vogais[1::2]       # índices ímpares
['e','o']
>>> vogais[::-1]       # inverter a lista
['u', 'o', 'i', 'e', 'a']
```

Percorrer os índices e elementos

```
for i in range(len(lista)):
    print(i, lista[i])
```

- Ciclo sobre índices i de 0 até $len(lista) - 1$
- Elemento $lista[i]$ associado ao índice i

Percorrer todos os elementos

```
for valor in lista:
    print(valor)
```

- Evita manipular explicitamente o índice
- Preferível quando necessitamos dos valores mas não dos índices

Listas são mutáveis

Podemos modificar ou acrescentar elementos:

```
>>> beatles = [1, 2, 3]
>>> beatles[0] = "john"
>>> beatles[2] = "ringo"
>>> beatles
['john', 2, 'ringo']

>>> beatles[1:2] = ['paul', 'george']
>>> beatles
['john', 'paul', 'george', 'ringo']
```

Remover elementos duma lista

```
>>> beatles = ['john', 'paul', 'george', 'ringo']
>>> del beatles[0]
>>> beatles
['paul', 'george', 'ringo']
```

Alternativa:

```
>>> beatles = ['john', 'paul', 'george', 'ringo']
>>> beatles[0:1] = []
>>> beatles
['paul', 'george', 'ringo']
```

Nomes e objectos

É importante distinguir o nome da lista dos valores associados.

Nomes e objectos (1)

Dois nomes, duas listas separadas:

```
>>> a = [1,2,3]
>>> b = [1,2,3]
>>> a[0] = 'oops'
>>> print(a, b)
['oops', 2, 3] [1, 2, 3]
```

Nomes e objectos (2)

Dois nomes, apenas uma lista:

```
>>> a = [1,2,3]
>>> b = a
>>> a[0] = 'oops'
>>> print(a, b)
['oops', 2, 3] ['oops', 2, 3]
```

Nomes e objectos (3)

Dois nomes, duas listas (fazendo uma cópia):

```
>>> a = [1,2,3]
>>> b = a[:]
>>> a[0] = 'oops'
>>> print(a, b)
['oops', 2, 3] [1, 2, 3]
```

Métodos sobre listas

Alguns métodos pré-definidos:

`append` acrescentar um elemento ao final

`insert` acrescentar um elemento numa posição

`remove` remover um elemento

`sort` ordenar os elementos por ordem crescente

- Utilização: `lista.método(argumentos)`
- Modificam a lista

Exemplos

```
>>> beatles = ['john','paul']
>>> beatles.append('george')
>>> beatles.append('ringo')
>>> beatles
['john', 'paul', 'george', 'ringo']
>>> beatles.insert(0, 'paul')
>>> beatles
['paul', 'john', 'paul', 'george', 'ringo']
>>> beatles.sort()
>>> beatles
['george', 'john', 'paul', 'paul', 'ringo']
```

Para obter mais informação:

```
>>> help(list)
```

Listas dentro de listas

- As listas podem conter outras listas
- Podemos assim representar tabelas ou matrizes

```
>>> matriz = [[1,2,-1],
              [3,1,0],
              [0,1,-2]]
>>> matriz[1][0]
3
>>> matriz[1][0] = -3
>>> matriz
[[1, 2, -1], [-3, 1, 0], [0, 1, -2]]
```

Tuplos

Tuplos

- Sequências ordenadas de elementos:

```
(e1, e2, ..., en)
```

- Acesso aos elementos é efetuado através dos índices
- Ao contrário das listas, os tuplos são imutáveis

Operações básicas

- comprimento

```
>>> len(('Pedro',12))
2
```

- concatenação

```
>>> ('Pedro,12)+('João',14)
('Pedro,12,'João',14)
```

- repetição

```
>>> 2*('Pedro',12)
('Pedro',12,'Pedro',12)
```

- pertença

```
>>> 12 in ('Pedro',12)
True
```

- iteração

```
>>> for x in ('Pedro',12):
        print(x)
Pedro
12
```

Acesso aos elementos

```
>>> nota = ('Pedro', 12)
>>> nota[0]
'Pedro'
>>> nota[1]
12
>>> nota[0] = 'Joao'
TypeError: 'tuple' object does not support
item assignment
```

Atribuição a tuplos de variáveis

```
>>> (x,y) = (5,-7)
>>> x
5
>>> y
-7
```

Ou simplesmente:

```
>>> x,y = 5,-7
>>> x
5
>>> y
-7
```

Listas e tuplos combinados

Vamos representar uma agenda como uma lista de pares *nome*/*email*:

```
[('Maria João', 'mj@mail.pt'),
 ('José Manuel', 'jm@mail.pt'),
 ('João Pedro', 'jp@mail.pt')]
```

Operações:

- acrescentar uma entrada (nome e email)
- procurar email pelo nome

Acrescentar uma entrada

```
def acrescentar(agenda, nome, email):
    # Inserir um nome e email na agenda
    agenda.append((nome,email))
```

Procurar um nome (1)

```
def procurar(agenda, txt):
    # Procurar emails utilizando o nome
    emails = []
    for par in agenda:
        if txt in par[0]:          # txt ocorre no nome?
            emails.append(par[1]) # acrescenta email
    return emails
```

Procurar um nome (2)

```
def procurar(agenda, txt):
    # Procurar emails utilizando o nome
    emails = []
    for (nome,email) in agenda:
        if txt in nome:          # txt ocorre no nome?
            emails.append(email) # acrescenta email
    return emails
```

Exemplos

```
>>> agenda = []
>>> acrescentar(agenda, "Maria João",
                "mj@mail.pt")
>>> acrescentar(agenda,"João Pedro",
                "jp@mail.pt")

>>> procurar(agenda, "Maria")
['mj@mail.pt']

>>> procurar(agenda, "João")
['mj@mail.pt', 'jp@mail.pt']
```

Usar listas ou tuplos?

- Utilizamos listas para sequências mutáveis (e.g. uma agenda)
- Utilizamos tuplos para sequências imutáveis (e.g. um par nome, telefone)
- Os tuplos são necessários em casos especiais: (e.g. *chaves de dicionários* — próximas aulas)
- Podemos sempre converter entre os dois tipos de sequência:

  list(...) converter para lista
  tuple(...) converter para tuplo