universidade de aveiro
escola superior de tecnologia e gestão de águeda

Introdução à Programação

Tuplos

cbt@ua.pt – RSI
pop@ua.pt - CS

2025-2026

Tuplos

estga
universidade de aveiro
escola superior de tecnologia e gestão de águeda

Um tuplo é uma sequência de valores de qualquer tipo e são indexados por inteiros.

Nesse aspecto os tuplos são parecidos com as listas. A diferença importante é que os **tuplos são imutáveis**.

Sintaticamente, um tuplo é **uma lista de valores separados por vírgulas**.

```
>>>  t = 'a', 'b', 'c', 'd', 'e'
```

imutáveis. Os seus elementos não podem ser alterados

Embora não seja necessário, é comum colocar tuplos entre parênteses

```
>>>  t = ('a', 'b', 'c', 'd', 'e')
```

Para criar um tuplo com um único elemento, devemos incluir uma vírgula final:

```
>>>  t1 = 'a'
>>>  type(t1)
<class 'tuple'>
```

Tuplos

Um valor dentro de parênteses não é um tuplo

```
>>>  t2 = (‘a’)
>>>  type(t2)
<class ‘str’>
```

Outra forma de criar um tuplo é usar a função **tuple**. Se não forem passados argumentos será criado um tuplo vazio

```
>>> t = tuple
>>>  t
()
```

Se o argumento for uma sequência (string, lista ou tuplo), o resultado será um tuplo com os elementos da sequência:

```
>>> t = tuple('lupins')
>>> t
('l', 'u', 'p', 'i', 'n', 's')
```

Tuplos

A maioria dos operadores de lista também funciona em tuplos.

O operador parenteses retos indexa um elemento

```
>>> t = 'a', 'b', 'c', 'd', 'e’
>>> t[0]
‘a’
```

Assim como o operado de subsecção

```
>>> t[1:3]
('b', 'c’)
```

Uma vez que os tuplos são imutáveis se tentamos modificar um elemento iremos obter um erro

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

No entanto podemos substituir um tuplo por outro

```
>>> t = ('A',) + t[1:]
>>> t
('A', 'b', 'c', 'd', 'e')
```

Atribuição

Muitas vezes é útil trocar os valores de duas variáveis.

Com atribuições convencionais, teremos de usar uma variável temporária.

```
>>> temp = a
>>> a = b
>>> b = temp
```

A atribuição de tuplos é mais elegante

```
>>> a, b = b, a
```

O **número de variáveis à esquerda e o número de valores à direita devem ser os mesmos**

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

Atribuição

O lado direito pode ser **qualquer tipo de sequência** (string, lista ou tupla).

Para dividir um endereço de email em nome de utilizador e domínio, podemos escrever

```
>>> addr = 'monty@python.org'
>>> uname, domain =
addr.split('@')
```

O valor de retorno de **split** é uma lista com dois elementos; o primeiro elemento é atribuído ao *uname*, o segundo ao *domain*.

```
>>> uname
'monty'
>>> domain
'python.org'
```

Listas e Tuplos

zip é uma função que recebe duas ou mais sequências e as intercala.

```
>>> s = 'abc'
>>> t = [0, 1, 2]
>>> zip(s, t)
<zip object at 0x7f7d0a9e7c48>
```

O resultado é um objeto zip que sabe como iterar pelos pares. O uso mais comum do zip é num loop for:

```
>>> for pair in zip(s, t):
        print(pair)
('a', 0)
('b', 1)
('c', 2)
```

Listas e Tuplos

Um objeto zip é um tipo de iterador, que itera uma sequência.

Os iteradores são semelhantes às listas em alguns aspetos, mas ao contrário das listas, não podemos usar um índice para selecionar um elemento do iterador.

Para usar operadores e métodos de lista podemos usar um objeto zip para criar uma lista

```
>>> s = 'abc'
>>> t = [0, 1, 2]
>>> list(zip(s, t))
[('a', 0), ('b', 1), ('c', 2)]
```

O resultado é uma lista de tuplos; neste exemplo, cada tuplo contém um caractere da string e o elemento correspondente da lista.

Listas e Tuplos

Se as sequências não tiverem o mesmo comprimento, o resultado terá o comprimento da mais curta

```
>>> list(zip('Anne', 'Elk'))
[('A', 'E'), ('n', 'l'), ('n', 'k')]
```

Podemos usar a atribuição de tuplos num **loop *for*** para percorrer uma lista de tuplos

```
>>> t = [('a', 0), ('b', 1), ('c', 2)]
>>> for letter, number in t:
        print(number, letter)

0 a
1 b
2 c
```

Dicionários e Tuplos

Os dicionários têm um método chamado **items** que retorna uma sequência de tuplos, onde cada tuplo é um par chave-valor.

```
>>> d = {'a':0, 'b':1, 'c':2}
>>> t = d.items()
>>> t
dict_items([('c', 2), ('a', 0), ('b', 1)])
```

O resultado é um objeto dict_items, que é um iterador que itera os pares chave-valor. Podemos usá-lo num loop *for* como este:

```
>>> for key, value in d.items():
        print(key, value)
c 2
a 0
b 1
```

Dicionários e Tuplos

Podemos usar uma lista de tuplos para inicializar um novo dicionário:

```
>>> t = [('a', 0), ('c', 2), ('b', 1)]
>>> d = dict(t)
>>> d
{'a': 0, 'c': 2, 'b': 1}
```

Combinar **dict** com **zip** produz uma maneira concisa de criar um dicionário:

```
>>> d = dict(zip('abc', range(3)))
>>> d
{'a': 0, 'c': 2, 'b': 1}
```

O método **update** dos dicionários também pega uma lista de tuplos e adiciona-os, como pares chave-valor.