Algoritmos e Programação de Computadores

Tuplas

Prof. Edson Borin
Instituto de Computação (IC/Unicamp)

Agenda

- Tuplas
- Exemplo

Tuplas

- Tuplas são uma sequência de elementos separados por vírgulas, representados ou não entre parênteses, isto é, os parênteses não são obrigatórios.
- Pode-se ainda misturar elementos de tipos diferentes.
- Porém, ao contrário de listas, as **tuplas são imutáveis**.
- Exemplo: `(18, "abril", 9.5, 1)` é uma tupla de 4 elementos.

Tuplas: Criando Tuplas

- Mais exemplos de tuplas.

```
tupla1 = ('abril', 18, 4, 2018)
tupla2 = (1, 2, 3, 4, 5, 6, 7)
tupla3 = "a", "b", "c", "d"
tupla4 = ("MC102", )
tupla5 = ()
```

`tupla4` representa uma tupla com um único elemento. A vírgula após o elemento é necessária para diferenciar de uma expressão entre parênteses.

Tuplas: Criando Tuplas

- Tuplas podem ser criadas a partir de listas com a função tuple()

```
tupla1 = ('abril', 18, 4, 2018)
lista1 = ['abril', 18, 4, 2018]
tupla2 = tuple(lista1)
print("tupla1:", tupla1)
print("lista1:", lista1)
print("tupla2:", tupla2)
```

```
tupla1: ('abril', 18, 4, 2018)
lista1: ['abril', 18, 4, 2018]
tupla2: ('abril', 18, 4, 2018)
```

Tuplas

- O que será impresso?

```
t1 = 'A',
t2 = ('A')
print(type(t1))
print(type(t2))
```

```
<class 'tuple'>
<class 'str'>
```

Tuplas

- Como strings, tuplas são **imutáveis**.

```
a = (18, "abril", 9.5, 1)
a[2] = 9.0

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- A utilidade de uma lista imutável ficará mais clara quando discutirmos dicionários.

Tuplas: Acessando Valores

- As operações para acessar os elementos ou sub-sequências de uma lista também funcionam em tuplas.

```
a = (18, "abril", 9.5, 1)
a[2]
9.5

a[1:3]
("abril", 9.5)
```

Tuplas: Acessando Valores

- Quase todas operações que são válidas para listas são válidas para tuplas, com exceção das operações que modificam o objeto (como *insert*, *sort* e *remove*)

```
a = (18, "abril", 9.5, 1)
print(a[1])
print(a[-1])
```

```
abril
1
```

Tuplas: Criando tuplas com slicing

- O operador de slicing também funciona em tuplas!

```
a = (18, "abril", 9.5, 1)
b = a[2:]
print("a:",a)
print("b:",b)
```

```
a: (18, 'abril', 9.5, 1)
b: (9.5, 1)
```

Tuplas: operador in

- O operador **in** também funciona em tuplas!

```
a = (18, "abril", 9.5, 1)
if 9.5 in a:
  print("a contém o valor 9.5")
else:
  print("a não contém o valor 9.5")
```

```
a contém o valor 9.5
```

Tuplas: operador +

- O operador **+** pode ser usado para concatenar tuplas.

```
a = (18, "abril")
b = (9.5, 1)
c = a + b
print("a:",a)
print("b:",b)
print("c:",c)
```

```
a: (18, 'abril')
b: (9.5, 1)
c: (18, 'abril', 9.5, 1)
```

Tuplas: acessando elementos com for

- O laço for pode ser usado para iterar sobre os valores de um tupla

```
t = (18, "abril", 9.5, (2.0, 5.0))
for i in t:
  print(i)
```

```
18
abril
9.5
(2.0, 5.0)
```

Tuplas: Empacotamento e Desempacotamento

- Os elementos de uma tupla podem ser acessados de uma forma implícita na atribuição (conhecido como desempacotamento).

```
x, y = (18, 20)
print("x=",x)
print("y=",y)
```

```
x= 18
y= 20
```

Tuplas: Empacotamento e Desempacotamento

- A tupla também pode ser implicitamente criada apenas separando os elementos por vírgula (conhecido como empacotamento).

```
x = 18, 20
print("x=",x)
```

```
x= (18, 20)
```