MC-102 — Aula 19
Tuplas

Eduardo C. Xavier

Instituto de Computação – Unicamp

21 de Outubro de 2020

Roteiro

1. Tipo Abstrato de Dados
2. Tuplas
3. Exercícios

Tipo Abstrato de Dados

- Vimos alguns tipos de dados primitivos (int, float, str, bool.).
- Vimos quais valores são representados por um destes tipos, e que operações podem ser realizadas sobre objetos deste tipo.

Tipo Abstrato de Dados

TAD

Um *Tipo Abstrato de Dados* (TAD) é uma descrição de um tipo de dado, indicando quais valores são representados pelo tipo, e que operações podem ser realizadas sobre objetos deste tipo.

Tipo Abstrato de Dados

- Como exemplo, uma *lista* é um TAD que armazena objetos de qualquer tipo. Os valores que uma lista assume são portanto todas as coleções possíveis de objetos. Um TAD lista possui algumas operações como:
  - Inclusão de um novo objeto na lista.
  - Exclusão de um objeto da lista.
  - Busca por um objeto da lista.
- O tipo **list** de Python implementa o TAD lista descrito acima.
  - Inclusão via métodos **append**, e **insert**.
  - Exclusão via método **remove**.
  - Busca via método **index** e operador **in**.

Tipo Abstrato de Dados

- O uso de um *Tipo Abstrato de Dados* (TAD) serve para criar um nível de abstração onde o usuário do TAD não precisa conhecer os detalhes de implementação do TAD, e apenas faz o uso deste pelos métodos, funções e operações especificadas sobre este.
- Hoje veremos mais um TAD, *tuplas*. Em outros cursos vocês vão aprender a criar TADs como listas e tuplas, e de tal forma que sejam eficientes.
- Por enquanto seremos apenas usuários destes TADs.

Tuplas

- Tuplas são similares a listas, isto é, uma coleção de objetos de qualquer tipo.
- Porém, ao contrário de listas, as tuplas são imutáveis.

Tuplas em Python são representadas por uma sequência de valores separados por vírgula, e entre um abre e fecha parênteses.

- Abaixo temos um exemplo de uma tupla com 4 objetos.

```
(1,2,5,'aaa')
```

Tuplas: criação

- Podemos criar tuplas de forma direta usando parênteses:

```
>>> t = (90,'a', 'ola', 5.65)
>>> type(t)
<class 'tuple'>
```

- Podemos usar a função **tuple** sobre objetos iteráveis para se criar uma tupla.

```
>>> r = range(5,10)
>>> t = tuple(r)
>>> t
(5, 6, 7, 8, 9)
>>>
>>> t = tuple('ola turma')
>>> t
('o', 'l', 'a', ' ', 't', 'u', 'r', 'm', 'a')
```

Tuplas: criação

- Podemos criar tuplas de forma implícita apenas separando objetos por vírgula.

```
>>> t = 90,'a', 'ola', 5.65
>>> t
(90, 'a', 'ola', 5.65)
```

Tuplas: acesso

- As operações para acessar os elementos de um lista, também funcionam em tuplas.
- Abaixo, acessamos uma posição específica da tupla e depois temos um exemplo de slicing.

```
>>> a = (1,2,5,'aaa')
>>> a[2]
5
>>> a[1:3]
(2, 5)
```

Tuplas: in

- Podemos verificar se um objeto está na tupla com o operador **in**.

```
>>> t = tuple("Cubatão")
>>> t
('C', 'u', 'b', 'a', 't', 'ã', 'o')
>>> 'e' in t
False
>>> 'o' in t
True
```

Tuplas: index

- Podemos obter o indice de um objeto que está na tupla com o método **index**.

```
>>> t = tuple("Cubatão")
>>> t
('C', 'u', 'b', 'a', 't', 'ã', 'o')
>>> t.index('a')
3
```

Tuplas: index

- Podemos usar a função **len** para obter o tamanho de uma tupla.

```
>>> t = tuple(range(10,101,10))
>>> t
(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
>>> len(t)
10
```

Tuplas: concatenação

- Podemos concatetar tuplas assim como listas.

```
>>> t1 = 'Verissimo', 'Machado', 'Amado'
>>> t2 = 'Clarisse', 'Cora', 'Cecília'
>>> t3 = t1 + t2
>>> t3
('Verissimo', 'Machado', 'Amado', 'Clarisse', 'Cora', 'Cecília')
```

Tuplas: percorrendo

- Podemos acessar cada elemento da tupla com um laço **for** assim como com listas.

```
>>> t = 'Clarisse', 'Cora', 'Cecília'
>>> for autora in t:
...     print(autora)
...
Clarisse
Cora
Cecília
```

Tuplas

- Tuplas são **imutáveis**, portanto os métodos e atribuições que alteram uma lista não funcionam em tuplas.

```
>>> l = [10,20,30]
>>> l[1] = 200
>>> l
[10, 200, 30]
>>> t = (10,20,30)
>>> t[1] = 200
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tuplas - empacotamento e desempacotamento

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

```
>>> x,y=(9,10)
>>> x
9
>>> y
10
```

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

```
>>> a = 67,5,4
>>> a
(67, 5, 4)
```