Laboratório de Programação 1

Aula 11

Mário Hozano
professor@hozano.com

Ciência da Computação
UFAL - Arapiraca

Relembrando a aula anterior...

- O que são tuplas?
- Como acessar os items de uma tupla?
- Para que serve a função *len()* com tuplas*?*
- Quais as vantagens das atribuições de tuplas?
- Tuplas podem ser usadas como retorno de funções ?
- Tuplas podem ser usadas como parâmetros de funções ?

Roteiro da aula

- Dicionários
- Acessando os itens de uma dicionário
- A função *len()*
- Removendo itens de um dicionário
- Métodos dos dicionários
- Resumo dos tipos complexos estudados

Dicionários

- Vimos anteirormente três tipos compostos: *strings,* listas e tuplas
- Os tipos compostos estudados utilizam números inteiros como índices
- Dicionários podem usar valores de qualquer tipo imutável como índices
- *Strings*, valores booleanos, valores numéricos são exemplos de valores que podem ser índices para os items armazenados em um dicionário
- Dicionários são **mutáveis**

Dicionários

- Um dicionário é caracterizado por um conjunto de itens separados por vírgulas e contornado por chaves *{}*
- Cada item de um dicionário possui um **índice** (chave) e seu respectivo **valor**, separados pelo caractere (*:*)
- O código a seguir cria um dicionário com 3 itens onde as chaves são i1, i2 e i3.

```
>>> dic = {'i1': 'valor1', 'i2': 'valor2', 'i3': 'valor3'}
```

Dicionários

- Uma outra forma de criar um dicionário é iniciando um dicionário vazio (*{}*) e depois adicionando elementos nele

```
>>> dic = {}
>>> dic['um'] = 'one'
>>> dic['dois'] = 'two'
>>> dic['tres'] = 'three'
```

- Dicionários não mantêm ordem de inserção, mas preservam o mapeamento entre chave e valor

```
>>> print(dic)
{'um': 'one', 'tres': 'three', 'dois': 'two'}
```

Dicionários – Acessando os valores

- Assim como *strings*, listas e tuplas, os valores de um dicionário podem ser acessados através do índice (chave)

```
>>> dic = {'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> dic['um']
'one'
>>> dic['tres']
'three'
```

- Da mesma forma podemos alterar um valor armazenado

```
>>> dic['um'] = 'uno'
>>> dic
{'um': 'uno', 'tres': 'three', 'dois': 'two'}
```

Dicionários – A função *len()*

- Assim como em outros tipos compostos a função *len()* pode ser aplicada em dicionários

- Neste caso, a função retorna o número de pares chave-valor do dicionário

```
>>> dic = {'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> len(dic)
3
```

Dicionários – Removendo Elementos

- Existem várias formas de remover elementos de um dicionáro
- Nos casos em que se sabe qual o índice do elemento que deve ser removido pode-se usar o método *pop()*
- O método *pop()* remove o elemento que possui o índice dado como argumento e o retorna na execução

```
>>> dic = {'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> dic.pop('dois')
'two'
>>> dic
{'um': 'one', 'tres': 'three'}
```

Dicionários – Removendo Elementos

- Uma outra forma de remover elementos de um dicionário é usando a instrução *del*
- Ela deve ser usada também nos casos em que se sabe o índice do elemento a ser removido
- Diferentemente do método *pop()* a instrução não retorna nenhum valor

```
>>> dic = {'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> del dic['dois']
>>> dic
{'um': 'one', 'tres': 'three'}
```

Dicionários – Métodos

- Ao tentar remover um item através de uma chave inexistente, o interpetador gera um erro

```
>>> dic = {'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> dic.pop('zero')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'zero'
```

- Para evitar este erro, podemos utilizar antecipadamente o método *has_key()* que retorna um valor booleano indicando se a chave dada existe ou não

```
>>> dic.has_key('zero')
False
>>> dic.has_key('dois')
True
```

Dicionários – Métodos

- Os dicionários possuem métodos que permitem manipular suas estruturas
- O método *keys()* deve ser utilizado para resgatar todas as chaves dos itens armazenados no dicionário
- Da mesma forma, o método *values()* retorna uma lista com todos os valores armazenados

```
>>> dic = {'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> dic.keys()
['um', 'tres', 'dois']
>>> dic.values()
['one', 'three', 'two']
```

Dicionários – Métodos

- Já o método *items()* retorna os pares chave-valor em forma de uma lista de tuplas

```
>>> dic = {'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> dic.items()
[('um', 'one'), ('tres', 'three'), ('dois', 'two')]
```

- Assim como os outros tipos compostos, dicionários são sequências iteráveis em um *for* a partir de suas chaves

```
dic = {'um': 'one', 'dois': 'two', 'tres': 'three'}
for chave in dic:
    print(chave)
```

Dicionários - Resumo dos tipos complexos

- Vimos que os tipos complexos estudados possuem diferentes características

- A tabela a seguir apresenta um resumo do que foi estudado

Tipo	Forma	Mutável	Índices
Strings	''	Não	Inteiros
Listas	[]	Sim	Inteiros
Tuplas	()	Não	Inteiros
Dicionários	{}	Sim	Tipos imutáveis

Exercícios

1. Crie uma função que receba duas listas de tamanho 10 com valores inteiros e retorne um mapa onde as chaves são os valores de uma lista e os valores são os da outra

2. Escreva uma função que receba como argumento um dicionário e um valor. Esta função deve retornar a chave do valor dado caso ele esteja no dicionário.

3. Escreva uma função que crie um dicionário com 10 itens. Os itens devem possuir chaves de 1 a 10 e seus valores devem ser o fatorial de sua chave.

4. Faça um programa que o usuário possa cadastrar em um dicionário a matrícula (chave) e nome dos alunos (valor).