MC-102 — Aula 13
Dicionários

Eduardo C. Xavier

Instituto de Computação – Unicamp

29 de Outubro de 2020

Roteiro

Tipo Abstrato de Dados

- 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.

Dicionários

- Dicionários é um TAD que armazena uma coleção de pares de objetos **chave/valor**.
- Há a restrição de que uma chave pode aparecer no máximo uma vez na coleção.
- Algumas operações sobre dicionários são:
  - A inclusão de um par **chave/valor** no dicionário.
  - A exclusão de um par **chave/valor** no dicionário.
  - A modificação de um **valor** de uma **chave** no dicionário.
  - Dada uma **chave**, buscar o seu **valor** associado.
- Implementações de dicionários visam que estas operações sejam realizadas de forma muito eficiente.

Dicionários

- Dicionários em Python são objetos **mutáveis** que armazenam pares **chave/valor** onde:
  - Os valores podem ser de qualquer tipo, mas as **chaves** só podem ser objetos **imutáveis** (hasheable na verdade).
  - As chaves precisam ser únicas, pois são indexadores do dicionário.

Dicionários em Python

Um dicionário é criado usando-se {}, e dentro, uma sequencia de **chave : valor**, separados por vírgula.

Dicionários

- Um dicionário é criado usando-se $\{\}$, e uma sequencia de **chave : valor**, separados por vírgula.

```
>>> d = {'Jan' : 1, 'Fev' : 2, 'Mar' : 3}
>>> d
{'Mar': 3, 'Fev': 2, 'Jan': 1}
>>> type(d)
<class 'dict'>
>>> d['Fev']
2
```

- Você pode 'pensar' em um dicionário como uma lista, mas o acesso a um valor se dá usando a chave.

```
>>> d['Fev']
2
```

Dicionários

- Veja um outro exemplo de criação de dicionário:

```
>>> dd = {'Jon Snow' : 12345678, 'Daenerys Targaryen': 78765432,
  'Eduardo Xavier' : 12341234}
```

- O dicionário acima pode representar uma agenda de telefones:
  - Os nomes (tipo string, que é imutável) como chaves e o valor associado a cada chave é o telefone (um inteiro).
- Acessamos o valor associado à uma chave como abaixo:

```
>>> dd['Jon Snow']
12345678
```

Dicionários

- Dicionários são **mutáveis**. Alteramos o valor de uma chave como no exemplo:

```
>>> dd = {'Jon Snow' : 12345678, 'Daenerys Targaryen': 78765432,
  'Eduardo Xavier' : 12341234}
>>> dd['Jon Snow'] = 1111111
>>> dd['Eduardo Xavier'] = 2222222
>>> dd
{'Jon Snow': 1111111, 'Daenerys Targaryen': 78765432,
        'Eduardo Xavier': 2222222}
>>>
```

Dicionários: métodos e funções

- **d[k] = v** atribui valor **v** para chave **k** caso ela exista em **d**, e se não existir um novo par chave/valor é criado:

```
>>> dd = {'Jon Snow' : 12345678, 'Daenerys Targaryen': 78765432,
          'Eduardo Xavier' : 12341234}
>>>
>>> dd['Eduardo Xavier'] = 1
>>> dd['Cersei Lannister'] = 666666
>>> dd
{'Jon Snow': 12345678, 'Daenerys Targaryen': 78765432,
      'Eduardo Xavier': 1, 'Cersei Lannister': 666666}
```

- **del d[k]** remove o par cuja chave é **k** do dicionário

```
>>> del dd['Jon Snow']
>>> dd
{'Daenerys Targaryen': 78765432, 'Eduardo Xavier': 1,
                'Cersei Lannister': 666666}
```

Dicionários: métodos e funções

- **len(d)** devolve o número de chaves em **d**

```
>>> dd = {'Jon Snow': 'Rei dos 7 reinos!', 'Daenerys Targaryen': 'Rainha dos 7 reinos
    'Eduardo Xavier': 'Zé Nínguem!'}
>>> len(dd)
3
```

Dicionários: métodos e funções

- O operador **in** pode ser usado para verificar se uma determinada chave está no dicionário.

```
>>> dd = {'Jon Snow' : 12345678, 'Daenerys Targaryen': 78765432,
      'Eduardo Xavier' : 12341234}
>>> 'Jon Snow' in dd
True
>>> 'Eduardo' in dd
False
>>> 'Eduardo Xavier' in dd
True
```

- O acesso a uma chave que não existe gera um erro.

```
>>> dd['Cersei']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Cersei'
```

Dicionários: métodos e funções

- **d.keys()** devolve uma view (pode usar no **for**) sobre as chaves.

```
>>> dd = {'Jon Snow': 'Rei dos 7 reinos!', 'Daenerys Targaryen': 'Rainha dos 7 reinos
    'Eduardo Xavier': 'Zé Nínguem!'}
>>> dd.keys()
dict_keys(['Jon Snow', 'Daenerys Targaryen', 'Eduardo Xavier'])
```

- Usando num laço **for**.

```
>>> dd = {'Jon Snow': 'Rei dos 7 reinos!', 'Daenerys Targaryen': 'Rainha dos 7 reinos!
        'Eduardo Xavier':'Zé Nínguem!'}
>>> for k in dd.keys():
...     print(k,'que é', dd[k])
...
Jon Snow que é Rei dos 7 reinos!
Daenerys Targaryen que é Rainha dos 7 reinos!
Eduardo Xavier que é Zé Nínguem!
```

Dicionários: métodos e funções

- **d.values()** devolve uma view (que pode ser usada num laço **for**) sobre os valores.

```
>>> dd = {'Jon Snow' : 'Rei dos 7 reinos!', 'Daenerys Targaryen': 'Rainha dos 7 reinos!
        'Eduardo Xavier':'Zé Nínguem!'}
>>> dd.values()
dict_values(['Rei dos 7 reinos!', 'Rainha dos 7 reinos!', 'Zé Nínguem!'])
```

- Usando num laço:

```
>>> dd = {'Jon Snow' : 'Rei dos 7 reinos!', 'Daenerys Targaryen': 'Rainha dos 7 reinos!
        'Eduardo Xavier':'Zé Nínguem!'}
>>> for v in dd.values():
...     print(v)
...
Rei dos 7 reinos!
Rainha dos 7 reinos!
Zé Nínguem!
```

Dicionários: métodos e funções

- É comum percorrer um dicionário acessando ao mesmo tempo suas chaves e valores. Fazemos isso com uso do método **items** que devolve uma 'lista' de tuplas com os pares (chave, valor).

```
>>> dd = {'Jon Snow' : 'Rei dos 7 reinos!', 'Daenerys Targaryen': 'Rainha dos 7 reinos!
        'Eduardo Xavier':'Zé Nínguem!'}
>>> dd.items()
dict_items([('Jon Snow', 'Rei dos 7 reinos!'),
('Daenerys Targaryen', 'Rainha dos 7 reinos!'), ('Eduardo Xavier', 'Zé Nínguem!')])
>>> list(dd.items())
[('Jon Snow', 'Rei dos 7 reinos!'), ('Daenerys Targaryen', 'Rainha dos 7 reinos!'),
 ('Eduardo Xavier', 'Zé Nínguem!')]
```

Dicionários: métodos e funções

- Usando num laço o resultado de **items** (note que usamos implicitamente o desempacotamento de tuplas):

```
>>> dd = {'Jon Snow' : 'Rei dos 7 reinos!', 'Daenerys Targaryen': 'Rainha dos 7 reinos!
    'Eduardo Xavier':'Zé Nínguem!'}
>>> for k,v in dd.items():
...   print(k,'que é',v)
...
Jon Snow que é Rei dos 7 reinos!
Daenerys Targaryen que é Rainha dos 7 reinos!
Eduardo Xavier que é Zé Nínguem!
```

Dicionários: métodos e funções

- **zip:** A função **zip** recebe como entrada duas sequencias (como listas ou tuplas) e os agrega em tuplas (formando pares).
- Podemos usar a função **zip** juntamente com **dict** para criar um dicionário a parir de duas listas por exemplo.

```
>>> nomes = ['Eduardo', 'Jon', 'Daenerys']
>>> fones = ['123', '222', '333']
>>> z = zip(nomes,fones)
>>> z
<zip object at 0x7fc2a765a6c8>
>>> d = dict(zip(nomes,fones))
>>> d
{'Eduardo': '123', 'Jon': '222', 'Daenerys': '333'}
>>>
```

Dicionários: métodos e funções

- Note que os valores de um dicionário podem ser quaisquer objetos em python, inclusive listas e dicionários.
- Um exemplo com listas.

```
>>> d = {}
>>> d['Eduardo'] = ['Rua 1', 'Campinas', 1390]
>>> d['Jon'] = ['Reino do Norte', 'Sete Reinos', 90]
>>> d
{'Eduardo': ['Rua 1', 'Campinas', 1390], 'Jon': ['Reino do Norte', 'Sete Reinos', 90]}
>>> d['Jon']
['Reino do Norte', 'Sete Reinos', 90]
```

Dicionários: métodos e funções

- Um exemplo com dicionários.

```
>>> d = {'Eduardo': {'Sobrenome': 'Xavier', 'Profissao': 'Professor'}}
>>> d
{'Eduardo': {'Sobrenome': 'Xavier', 'Profissao': 'Professor'}}
>>> d['Jon'] = {'Sobrenome': 'Snow', 'Profissao': 'Rei'}
>>> d
{'Eduardo': {'Sobrenome': 'Xavier', 'Profissao': 'Professor'},
             'Jon': {'Sobrenome': 'Snow', 'Profissao': 'Rei'}}
>>> d['Eduardo']['Sobrenome']
'Xavier'
```

Exemplo: contando as palavras de um texto

- Faça um programa que receba um texto como entrada (sem pontuação) e conte quantas vezes ocorre cada palavra no texto.
- Idéia: usar um dicionário para contar o número de ocorrências de cada palavra.
- A palavra é a chave do dicionário, e o valor será quantas vezes a letra foi encontrada.

Exemplo: contando as letras de uma string

- Abaixo temos a solução (assumindo um texto sem pontuação).

```
s = input() #lemos o texto do teclado
d = {} #criamos um dicionário vazio

#separamos o texto como uma lista de palavras
s = s.split()

for pal in s:
  #se a palavra já estiver no dicionário
  #incrementamos seu número de ocorrências
  if pal in d:
    d[pal] = d[pal] + 1
  #se a palavra não estiver no dicionário
  #iniciamos sua contagem com 1
  else:
    d[pal] = 1

for pal, num in d.items():
  print(pal,'aparece',num,'vezes')
```

Exercício 1

Modifique o programa que conta palavras para que ele

- conte as palavras que só diferem em ter letras maiúsculas e minúsculas como as mesmas palavras.
- conte as palavras mesmo que o texto tenha sinais de pontução.

Dê uma olhada na função **lower**.
`https://docs.python.org/3.4/library/stdtypes.html#str.lower`
da biblioteca `string`

Exercício 2

Escreva um programa que imprime a palavra mais comum de um texto:

- Use o `split()` para quebrar a string em uma lista de palavras.
- Use as palavras como chaves do dicionário.
- Converta cada palavra para minúsculo com a função **lower**.
- Remova os caracteres de pontuação das palavras. Veja a constante `punctuation` `https://docs.python.org/3.4/library/string.html#string.punctuation`