MC-102 — Aula 20
Tuplas e Dicionários

Prof. Luiz F. Bittencourt

Turmas QR

Instituto de Computação – Unicamp

2019

Conteúdo adaptado de slides fornecidos pelo Prof. Eduardo Xavier.

Roteiro

Tuplas

- Tuplas são similares a listas, isto é, uma sequência de dados de qualquer tipo.
- Porém, ao contrário de listas, as tuplas são imutáveis.
- Tuplas são representadas por uma sequência de valores separados por vírgula, e entre um abre e fecha parênteses.
  - `(1,2,5,''aaa'')` é uma tupla de 4 elementos.
- As operações para acessar os elementos ou sub-sequências de uma lista e de uma string, também funcionam em tuplas.

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

Tuplas

- Como strings, tuplas são imutáveis.

```
>>> a=(1,2,5,"aa")
>>> a[2]=9
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 na aula de hoje, na seção de dicionários.

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).

```
>>> 67,90
(67, 90)
```

Dicionários

- Dicionários são estruturas de dados que associam uma chave com um valor.
- Os valores podem ser dados de qualquer tipo, mas as chaves só podem ser dados de tipos imutáveis.
- As chaves precisam ser únicas.
- Veja um exemplo de criação de dicionário:

```
>>> dd={"Jose":12345678, "maria": 78765432}
>>> type(dd)
<class 'dict'>
```

- O dicionário acima pode representar uma agenda de telefones, com o nome (uma string, que é imutável) como chave e o valor associado a cada chave é o telefone (um inteiro).
- Acessar o valor associado à uma chave é feito como no exemplo:

```
>>> dd["maria"]
78765432
```

Dicionários

- O valor associado à uma chave pode ser modificado, ou uma nova chave (e seu valor) podem ser incluídos no dicionário.

```
>>> dd={"Jose":12345678, "maria": 78765432}
>>> dd
{'maria': 78765432, 'Jose': 12345678}
>>> dd['maria'] = 777777
>>> dd['carlos'] = 888888
>>> dd
{'carlos': 888888, 'maria': 777777, 'Jose': 12345678}
>>>
```

- A ordem dos pares chave/valor no dicionário é arbitrária

```
>>> dd['ana']=999999
>>> dd
{'carlos': 888888, 'maria': 777777, 'Jose': 12345678, 'ana': 9999
>>>
```

Operações em Dicionários

- O laço **for** aplicado a um dicionário faz a variável do laço passar por todas as chaves do dicionário (na ordem interna).

```
>>> for x in dd:
...     print(x)
...
carlos
maria
Jose
ana
```

- O operador **in** verifica se uma chave está no dicionário.

```
>>> 'ana' in dd
True
>>> 'Ana' in dd
False
```

Dicionários

- Acessar uma chave que não existe causa erro de execução.

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

- O método **items** retorna tuplas dos pares chave/valor.

```
>>> dd.items()
dict_items([('ana', 333444555), ('maria', 77777777), ('Jose', 12345678),
                                ('carlos', 1112223334)])
```

Exemplo: contando as letras de uma string

- Faça uma função que dada uma string, retorna a letra mais comum nessa string (em caso de empate retorne qualquer uma das mais frequentes).
- Idéia: usar um dicionário para contar cada letra.
- A letra é a chave do dicionário, e o valor será quantas vezes a letra foi encontrada.

Exemplo: contando as letras de uma string

```
def contaletra(s):
  conta={} # dicionario vazio
  for car in s:
    if car in conta:
      conta[car]=conta[car]+1
    else:
      conta[car]=1
...
```

Ao final deste segmento de função temos um dicionário com pares letras/contador.

Exemplo: contando as letras de uma string

Agora vamos determinar a letra mais comum:

```
def contaletra(s):
    ...
    letramais=''
    for x in conta:
        if letramais=='': # nenhuma mais comum ainda
            letramais=x
        elif conta[x] > conta[letramais]:
            letramais=x

    return letramais
```

Exemplo: contando as letras de uma string

A função completa é:

```
def contaletra(s):
  conta={} # dicionario vazio
  for car in s:
    if car in conta:
       conta[car]=conta[car]+1
    else:
       conta[car]=1
  letramais=''
  for x in conta:
    if letramais=='': # nenhuma mais comum ainda
       letramais=x
    elif conta[x] > conta[letramais]:
       letramais=x
  return letramais
```

Exemplo: contando as letras de uma string

Testando:

```
>>> contaletra("ouviram do ipiranga")
'i'
>>> contaletra("ouviram do ipirangaaa")
'a'
>>> contaletra("ouviram do ipiranga    ")
' '
>>> contaletra("ouviram do Ipiranga")
'a'
```

Exercício 1

Modifique a função **contaletra** para que ela

- não conte brancos e pontuação como letras.
- conte as letras maiúsculas e minúsculas como as mesmas letras (o caso do "I" no exemplo.

Dê uma olhada na função **lower**

`https://docs.python.org/3.4/library/stdtypes.html#str.lower`

e na constante `punctuation`

`https://docs.python.org/3.4/library/string.html#string.punctuation`

da biblioteca `string`

Exercício 2

Escreva uma função que retorna a palavra mais comum de uma string:

- 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 (mais difícil).