Python:
Tuplas e Strings

Claudio Esperança

Tuplas

- São estruturas de dados parecidas com listas, mas com a particularidade de serem *imutáveis*
- Tuplas são seqüências e, assim como listas, podem ser indexadas e fatiadas, mas não é possível modificá-las
- Um valor do tipo tupla é uma série de valores separados por vírgulas e entre parênteses

```
>>> x = (1,2,3)
>>> x
(1, 2, 3)
>>> x [0]
1
>>> x [0]=1
...
TypeError: object does not support item
  assignment
```

Tuplas

- Uma tupla vazia se escreve `()`
- Os parênteses são opcionais se não provocarem ambigüidade
- Uma tupla contendo apenas um elemento deve ser escrita com uma vírgula ao final
  - Um valor entre parênteses sem vírgula no final é meramente uma expressão:

```
>>> (10)
10
>>> 10,
(10,)
>>> (10,)
(10,)
>>> 3*(10+3)
39
>>> 3*(10+3,)
(13,13,13)
```

A função *tuple*

- Assim como a função **list** constrói uma lista a partir de uma seqüência qualquer, a função **tuple** constrói uma tupla a partir de uma seqüência qualquer

```
>>> list("abcd”)
['a', 'b', 'c', 'd']
>>> tuple("abcd”)
('a', 'b', 'c', 'd’))
>>> tuple([1,2,3])
(1, 2, 3)
>>> list((1,2,3))
[1, 2, 3]
```

Quando usar tuplas

- Em geral, tuplas podem ser substituídas com vantagem por listas
- Entretanto, algumas construções em Python requerem tuplas ou seqüências imutáveis, por exemplo:
  - Tuplas (ao contrário de listas) podem ser usadas como chaves de dicionários
  - Funções com número variável de argumentos acessam os argumentos por meio de tuplas
  - O operador de formatação aceita tuplas, mas não listas

O operador de formatação

- Strings suportam o operador % que, dada uma string especial (template) e um valor, produz uma string *formatada*
- O formato geral é
  - *template % valor*
- O template é uma string entremeada por códigos de formatação
  - Um código de formatação é em geral composto do caracter % seguido de uma letra descritiva do tipo do valor a formatar (s para string, f para float, d para inteiro, etc)
- Exemplo:

```
>>> '====%d====' % 100
'====100===='
>>> '====%f====' % 1
'====1.000000===='
```

Formatando tuplas

- Um template pode ser aplicado aos diversos valores de uma tupla para construir uma string formatada
- Ex.:

```
>>> template = "%s tem %d anos"
>>> tupla = ('Pedro', 10)
>>> template % tupla
'Pedro tem 10 anos'
```

- Obs: mais tarde veremos que o operador de formatação também pode ser aplicado a dicionários

Anatomia das especificações de formato

- Caracter %
- Flags de conversão (opcionais):
  - - indica alinhamento à esquerda
  - + indica que um sinal deve preceder o valor convertido
  - “ ” (um branco) indica que um espaço deve preceder números positivos
  - 0 indica preenchimento à esquerda com zeros
- Comprimento mínimo do campo (opcional)
  - O valor formatado terá este comprimento no mínimo
  - Se igual a * (asterisco), o comprimento será lido da tupla
- Um “.” (ponto) seguido pela precisão (opcional)
  - Usado para converter as casas decimais de floats
  - Se aplicado para strings, indica o comprimento máximo
  - Se igual a *, o valor será lido da tupla
- Caracter indicador do tipo de formato

Tipos de formato

- d, i Número inteiro escrito em decimal
- o Número inteiro sem sinal escrito em octal
- u Número inteiro sem sinal escrito em decimal
- x Número inteiro sem sinal escrito em hexadecimal (minúsculas)
- X Número inteiro sem sinal escrito em hexadecimal (maiúsculas)
- e Número de ponto flutuante escrito em notação científica ('e' minúsculo)
- E Número de ponto flutuante escrito em notação científica ('E' maiúsculo)
- f, F Número de ponto flutuante escrito em notação convencional
- g Mesmo que e se expoente é maior que -4. Caso contrario, igual a f
- G Mesmo que E se expoente é maior que -4. Caso contrario, igual a F
- C Caractere único (usado com inteiro ou string de tamanho 1)
- r String (entrada é qualquer objeto Python que é convertido usando a função `repr`)

Exemplos

```
>>> "Numero inteiro: %d" % 55
'Numero inteiro: 55'
>>> "Numero inteiro com 3 casas: %3d" % 55
'Numero inteiro com 3 casas:  55'
>>> "Inteiro com 3 casas e zeros a esquerda: %03d" % 55
'Inteiro com 3 casas e zeros a esquerda: 055'
>>> "Inteiro escrito em hexadecimal: %x" % 55
'Inteiro escrito em hexadecimal: 37'
>>> from math import pi
>>> "Ponto flutuante: %f" % pi
'Ponto flutuante: 3.141593'
>>> "Ponto flutuante com 12 decimais: %.12f" % pi
'Ponto flutuante com 12 decimais: 3.141592653590'
>>> "Ponto flutuante com 10 caracteres: %10f" % pi
'Ponto flutuante com 10 caracteres:   3.141593'
>>> "Ponto flutuante em notacao cientifica: %10e" % pi
'Ponto flutuante em notacao cientifica: 3.141593e+00'
>>> "String com tamanho maximo definido: %.3s" % "Pedro"
'String com tamanho maximo definido: Ped'
```

Exemplo: Imprimindo uma tabela

```
Itens    = ["Abacate", "Limão", "Tangerina",
  "Melancia”, "Laranja da China"]
precos = [2.13, 0.19, 1.95, 0.87, 12.00]

len_precos = 10 # Coluna de precos tem 10 caracteres

# Achar a largura da coluna de itens
len_itens = len(itens[0])
for it in itens : len_itens = max(len_itens,len(it))

# Imprimir tabela de precos
print "-"*(len_itens+len_precos)
print "%-*s%*s" % (len_itens, "Item", len_precos,
  "Preço”)
print "-"*(len_itens+len_precos)
for i in range(len(itens)):
   print "%-*s%*.2f" % (len_itens, itens[i],len_precos,
   precos[i])
```

Exemplo: resultados

Item	Preço
Abacate	2.13
Limão	0.19
Tangerina	1.95
Melancia	0.87
Laranja da China	12.00

O Módulo String

- Manipulação de strings é uma atividade freqüente em programas Python
- Existe um módulo chamado `string` que contém uma grande quantidade de funcionalidades para trabalhar com strings
  - Para usá-las:

    ```
    from string import *
    ```
- Entretanto, strings pertencem à classe `str` e a maior parte do que existe no módulo string aparece como métodos da classe `str`

Strings: método *find*

- `find` (*substring*, *inicio*, *fim*)
  - Retorna o índice da primeira ocorrência de *substring*
  - *inicio* e *fim* são opcionais e indicam os intervalos de índices onde a busca será efetuada
    - Os defaults são 0 e o comprimento da string, respectivamente
  - Caso *substring* não apareça na string, é retornado -1
  - Observe que o operador `in` pode ser usado para dizer se uma substring aparece numa string

Strings: método *find* (exemplo)

```
>>> s = "quem parte e reparte, fica com a maior
  parte"
>>> s.find("parte”)
5
>>> s.find("reparte”)
13
>>> s.find("parcela”)
-1
>>> "parte" in s
True
>>> s.find("parte",6)
15
>>> s.find("parte",6,12)
-1
```

Strings: método *join*

- `join`(*seqüência*)
  - Retorna uma string com todos os elementos da *seqüência* concatenados
    - Obs: Os elementos da seqüência têm que ser strings
  - A string objeto é usada como separador entre os elementos
  - Ex.:

```
>>> "/".join(("usr","bin","python"))
'usr/bin/python'
>>> "Q".join((1,2,3,4,5))
...
TypeError: sequence item 0: expected string,
  int found
>>> "Q".join(('1','2','3','4','5'))
'1Q2Q3Q4Q5'
```

Strings: métodos *lower* e *upper*

- `lower()`
  - Retorna a string com todos os caracteres maiúsculos convertidos para minúsculos
- `upper()`
  - Retorna a string com todos os caracteres minúsculos convertidos para maiúsculos
  - Ex.:

```
>>> print "Esperança".upper()
ESPERANÇA
>>> print "Pé de Laranja Lima".lower()
pé de laranja lima
```

Strings: método *replace*

- `replace`(*velho*,*novo*,*n*)
  - Substitui as instâncias da substring *velho* por *novo*
  - Se *n* for especificado, apenas *n* instâncias são trocadas
  - Caso contrário, todas as instâncias são trocadas
  - Ex.:

```
>>> s = "quem parte e reparte, fica com a maior
  parte"
>>> s.replace("parte","parcela”)
'quem parcela e reparcela, fica com a maior
  parcela'
>>> s.replace("parte","parcela",2)
'quem parcela e reparcela, fica com a maior
  parte'
```

Strings: método *split*

- `split`(*separador*)
  - Retorna uma lista com as substrings presentes entre cópias da string *separador*
  - Faz o contrário do método `join`
  - Se *separador* não for especificado, é assumido seqüências de caracteres em branco, tabs ou newlines
  - Ex.:

```
>>> s = "xxx yyy zzz  xxx  yyy zzz"
>>> s.split()
['xxx', 'yyy', 'zzz', 'xxx', 'yyy', 'zzz']
>>> s.split('xxx')
['', ' yyy zzz  ', '  yyy zzz']
```

Strings: método *strip*

- `strip`(*ch*)
  - Retorna a string sem caracteres iniciais ou finais que estejam na string *ch*
  - Se *ch* não for especificada, retira caracteres em branco
  - Pode-se também usar `rstrip`() para retirar caracteres à direita (final) ou `lstrip`() para retirar caracteres à esquerda (início)
  - Ex.:

```
>>> "   xxx afdsfa  ".strip()
'xxx afdsfa'
>>> "xxx yyy zzz xxx".strip("xy ”)
'zzz'
>>> "  xxx   ".rstrip()
'  xxx'
```

Strings: método *translate*

- `translate`(*trans*)
  - Retorna uma cópia da string onde os caracteres são substituídos de acordo com a tabela de tradução *trans*
  - *trans* é uma string com 256 caracteres, um para cada possível código de oito bits
    - Ex.: se *trans* tem 'X' na posição 65 (correspondente ao caractere ASCII 'A'), então, na string retornada, todos os caracteres 'A' terão sido substituídos por 'X'
  - Na verdade, as tabelas de tradução são normalmente construídas com a função `maketrans` do módulo `string`

Função string.maketrans

- `maketrans` (*velho*, *novo*)
  - retorna uma tabela de tradução onde os caracteres em *velho* são substituídos pelos caracteres em *novo*
  - Ex.:

```
>>> from string import maketrans
>>> trans = maketrans('qs', 'kz’)
>>> s = "que surpresa: quebrei a cara"
>>> s.translate(trans)
'kue zurpreza: kuebrei a cara'
```