

Processamento Estatístico da Linguagem Natural

Aula 7

Professora Bianca
(Sala 302 – Bloco E)
bianca@ic.uff.br

<http://www.ic.uff.br/~bianca/peln/>

Aula 7 - 18/09/2008

1

Linguagem Python

- Introdução
 - Características de Python
 - Rodando programas
 - Módulos
- Tipos básicos
 - Números e variáveis
 - Strings
 - Listas e tuplas
 - Dicionários
- Fluxo de controle
 - Execução condicional
 - Repetições

Aula 7 - 18/09/2008

2

Strings

- Manipular strings em Python é fácil.
 - Comparado com linguagens como C, Java e C++.
- Strings podem ser escritas usando aspas simples ou aspas duplas.
 - Torna mais fácil incluir aspas dentro da string.
 - Exemplos:
 - 'Isso é uma string em Python'
 - "Isso também é uma string em Python"
 - 'Ele disse "Alô?" e desligou o telefone'
 - "Gota d'água"

Aula 7 - 18/09/2008

3

Barra invertida em strings

- A barra invertida pode ser usada como caracter de escape para caracteres especiais.
- Exemplo: \n é newline, \t é tab

```
>>> s = 'Name\tAge\nJohn\t21\nBob\t44'
>>> print s
Name      Age
John      21
Bob       44
>>> t = "Mary's"
>>> print t
Mary's"
```

Aula 7 - 18/09/2008

4

Aspas triplas

- Aspas triplas (""" ou ''') são usadas para strings que ocupam várias linhas.

```
>>> s = """this is
... a string
... over 3 lines"""
>>> t = '''so
... is
... this'''
>>> print s
this is
a string
over 3 lines
>>> print t
so
is
this
```

Aula 7 - 18/09/2008

5

Operações com Strings

- Concatenação (+)
- Comprimento (len)
- Repetição (*)
- Indexação e "Slicing" ([])

```
s = 'computational'
t = 'linguistics'
cl = s + ' ' + t      # 'computational linguistics'
l = len(cl)           # 25
u = '-' * 6           # -----
c = s[3]              # p
x = cl[11:16]         # 'al li'
y = cl[20:]           # 'stics'
z = cl[: -1]          # 'computational linguistic'
```

Aula 7 - 18/09/2008

6

Indexação e Quebra

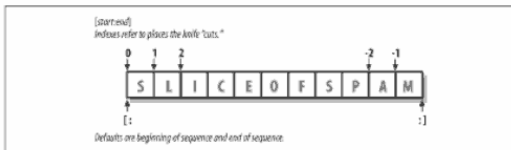


Figure 7-1. Offsets and slices: positive offsets start from the left end (offset 0 is the first item), and negatives count back from the right end (offset -1 is the last item). Either kind of offset can be used to give positions in indexing and slicing.

Aula 7 - 18/09/2008

7

Métodos de strings

- Alguns exemplos:

```
s = 'example'
s = s.capitalize()      # 'Example'
t = s.lower()           # 'example'
flag = s.isalpha()      # True
s = s.replace('amp', 'M') # 'exMle'
i = t.find('xa')        # 1
n = t.count('e')        # 2
```

Aula 7 - 18/09/2008

8

Listas em Python

- Listas são coleções ordenadas de objetos arbitrários.
- Objetos são acessados usando índices que indicam a posição na lista a partir do início.
- Têm tamanho variável (crescem automaticamente à medida que os objetos são inseridos).
- São heterogêneas, isto é, podem conter qualquer tipo de objeto, incluindo outras listas.
- São mutáveis, ao contrário de strings.

```
>>> s = ['a', 'b', 'c']
>>> t = [1, 2, 3]
>>> u = s + t          # ['a', 'b', 'c', 1, 2, 3]
>>> n = len(u)         # 6
```

Aula 7 - 18/09/2008

9

Indexação e “Slicing” de Listas

- Funciona da mesma forma que com strings.
- A indexação retorna o objeto em uma dada posição.
- O “slicing” retorna uma lista.
- Podemos usar indexação e “slicing” para mudar o conteúdo da lista.

```
l = ['a', 'b', 'c', 'd']
x = l[2]                # 'c'
m = l[1:]               # ['b', 'c', 'd']
l[2] = 'z'              # ['a', 'b', 'z', 'd']
l[0:2] = ['x', 'y']     # ['x', 'y', 'z', 'd']
```

Aula 7 - 18/09/2008

10

Métodos de listas

- Alguns exemplos:

```
l = [7, 8, 9, 3]
l.sort()                # [3, 7, 8, 9]
l.append(6)             # [3, 7, 8, 9, 6]
l.append([1, 2])        # [3, 7, 8, 9, [1, 2]]
l.extend(['r', 's'])    # [3, 7, 8, 9, [1, 2], 'r', 's']
```

Aula 7 - 18/09/2008

11

Dicionários

- Dicionários são endereçados por *chave*, não por posição.
 - Podem ser vistos como uma coleção de pares chave:valor.
- São coleções *não-ordenadas* de objetos arbitrários.
- Tem tamanho *variável* e podem conter objetos de qualquer tipo, inclusive outros dicionários.
- São *mutáveis* como as listas.

Aula 7 - 18/09/2008

12

Exemplo com dicionários

```
level = {'low':1, 'medium':5}
x = level['medium']      # 5
n = len(level)           # 2

flag = level.has_key('low') # True
l = level.keys()         # ['low', 'medium']

level['low'] = 2          # {'low':2, 'medium':5}
level['high'] = 10        # {'low':2, 'high':10, 'medium':5}

level.items()
[('low',2), ('high',10), ('medium',5)]

level.values()
[2, 10, 5]
```

Aula 7 - 18/09/2008

13

Sobre dicionários

- Operações sobre seqüências não funcionam (ex.: "slice") já que dicionários são mapeamentos e não seqüências.
- Dicionários têm um conjunto de chaves.
 - Só pode haver um valor por chave.
- Atribuir um valor a uma nova chave adiciona uma nova entrada ao dicionário.
- As chaves podem ser qualquer tipo de objeto.
- Dicionários podem ser usados como "records".
- Dicionários podem ser usados para armazenar matrizes esparsas.

Aula 7 - 18/09/2008

14

Outros Objetos

- Tuplas: como listas, porém imutáveis.

```
emptyT = ()
t1 = (1, 2, 3)
x = t1[1]      # 2
n = len(t1)    # 3
y = t1[1:]     # (2, 3)
```

- Arquivos: objetos com métodos para ler e escrever arquivos.

```
file = open('myfile', 'w')
file.write('hello file\n')
file.close()

f2 = open('myfile', 'r')
s = f2.readline()      # 'hello file\n'
t = f2.readline()      # ''
all = open('myfile').read() #entire file as a string
```

Aula 7 - 18/09/2008

15

Execução Condicional

```
course = 'Syntax'
if course == 'Syntax':
    print 'Bhatt'
    print 'or Potts'
elif course == 'Computational Linguistics':
    print 'McCallum'
else:
    print 'Someone else'
```

- A indentação determina a estrutura do bloco.
 - É o único lugar onde o espaço em branco importa.
- A indentação ajuda na legibilidade do código.
- Expressões depois do if e elif podem ser de quase qualquer tipo.
 - False, 0, [], (), "", funcionam como falso, o resto é verdadeiro.

Aula 7 - 18/09/2008

16

Laços "While"

- Um laço do tipo "while" continua enquanto a expressão no topo for verdadeira.

```
a = 0
b = 10
while a < b:
    print a
    a = a + 1
```

```
s = 'abcdefg'
while len(s) > 0:
    print s
    s = s[1:]
```

Aula 7 - 18/09/2008

17

Laços "For"

- "For" é usado pra percorrer uma seqüência qualquer de objetos.

```
l = ['a', 'b', 'c']
for i in l:
    print i
```

```
sum = 0
for x in [1, 2, 3, 4, 5, 6]:
    sum = sum + x
print sum
```

- O uso de "range" pode ser útil.

```
range(5)      # [0, 1, 2, 3, 4]
range(2,5)    # [2, 3, 4]
range(0,6,2)  # [0, 2, 4]
```

Aula 7 - 18/09/2008

18

Laços “For”

- Fazer alguma coisa com cada item de uma lista.

```
l = [1, 2, 3, 4, 5, 6] # or l = range(1,7)

# one way to print the square
for x in l:
    print x*x

# another way to do it
n = len(l)
for i in range(n):
    print l[i]*l[i]
```

Aula 7 - 18/09/2008

19

Exemplo: Interseção

```
l1 = ['a', 'd', 'f', 'g']
l2 = ['a', 'b', 'c', 'd']
# one way
result = []
for x in l1:
    for y in l2:
        if x == y:
            result.append(x)
# or, alternatively
result = []
for x in l1:
    if x in l2:
        result.append(x) # result == ['a', 'd']
```

Aula 7 - 18/09/2008

20

Funções “built-in”, importadas e definidas pelo usuário

- “Built-in”

```
l = len(['a', 'b', 'c'])
```

- Importadas

```
import math
from os import getcwd
print getcwd() # which directory am I in?
x = math.sqrt(9) # 3
```

- Definidas pelo usuário

```
def multiply(a, b):
    return a * b
print multiply(4,5)
print multiply('-',5)
```

Aula 7 - 18/09/2008

21

Definição de funções

- Def** cria um objeto do tipo função e dá um nome a ele.
- Return** retorna um objeto a quem chamou a função.

```
def square(x): # create and assign
    return x*x
y = square(5) # y gets 25
```

Aula 7 - 18/09/2008

22

Exemplo

```
def intersect(seq1, seq2):
    result = []
    for x in seq1:
        if x in seq2:
            result.append(x)
    return result
```

Aula 7 - 18/09/2008

23

Variáveis locais

- Variáveis dentro de uma função são locais àquela função.

```
>>> intersect(s1, s2):
... result = []
... for x in s1:
...     if x in s2:
...         result.append(x)
... return result
...
>>> intersect([1,2,3,4], [1,5,6,4])
[1, 4]
>>> result
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'result' is not defined
```

Aula 7 - 18/09/2008

24

Passagem de Parâmetros

- Objetos imutáveis são passados “por valor”.

```
>>> def plusone(x):
...     x = x + 1
...     return x
...
>>> plusone(3)
4
>>> x = 6
>>> plusone(x)
7
>>> x
6
```

Aula 7 - 18/09/2008

25

Passando parâmetros mutáveis

- Números, strings e tuplas são imutáveis enquanto listas e dicionários são mutáveis.
- Objetos mutáveis são passados “por referência”.

```
>>> def appendone(s):
...     s.append('one')
...     return s
...
>>> appendone(['a', 'b'])
['a', 'b', 'one']
>>> l = ['x', 'y']
>>> appendone(l)
['x', 'y', 'one']
>>> l
['x', 'y', 'one']
```

Aula 7 - 18/09/2008

26

map

```
>>> counters = range(1,6)
>>> updated = []
>>> for x in counters:
...     updated.append(x+3)
...
>>> updated
[4, 5, 6, 7, 8]

# Another way...
>>> def addthree(x):
...     return x+3
...
# map() applies a function to all elements of a list
>>> map(addthree, counters)
[4, 5, 6, 7, 8]
```

Aula 7 - 18/09/2008

27

Funções anônimas

```
# lambda is a way to define a function with no name
>>> map((lambda x: x+3), counters)
[4, 5, 6, 7, 8]

# a list comprehension does something similar,
# but can offer more flexibility
>>> result = [addthree(x) for x in counters]
>>> result
[4, 5, 6, 7, 8]
>>> [addthree(x) for x in counters if x < 4]
[4, 5, 6]
```

Aula 7 - 18/09/2008

28

Número variável de parâmetros

```
def max (*a):
    maximum = 9999999
    for x in a:
        if x > maximum:
            maximum = x
    return maximum
```

Aula 7 - 18/09/2008

29

Parâmetros opcionais

```
def exp (x, exponent=2.718):
    return exponent ** x
```

```
>>> exp(1)
2.718
>>> exp(1, 2.0)
2.0
>>> exp(3, 2.0)
8.0
>>> exp(3, exponent=2.0)
8.0
```

Aula 7 - 18/09/2008

30

Múltiplos parâmetros opcionais

```
def exp_plus (x, exponent=2.718, addend=0):  
    return (exponent ** x) + addend
```

```
>>> exp(1)  
2.718  
>>> exp(1, 2.0)  
2.0  
>>> exp(1, exponent=2.0)  
2.0  
>>> exp(1, addend=2.0)  
4.718
```

Aula 7 - 18/09/2008

31

Número arbitrário de parâmetros opcionais

- A notação ** recebe todos os parâmetros extra em um dicionário.

```
def showargs (separator, **d):  
    for key in d.keys():  
        print str(key)+":"+str(d[key])+separator,  
        print
```

```
>>> showargs(";", bi=2, tri=3, quad=4)  
tri:3;bi:2;quad:4;
```

```
def showargs (separator, **d):  
    for (key,val) in d.items():  
        print str(key)+":"+str(val)+separator,  
        print
```

Aula 7 - 18/09/2008

32