NLP Lab: Week 1

Introduction to Python

Computing Ngrams

Jason S. Chang
National Tsing Hua University

Room 323, T7T8T9

based on slides by Paul Prescod
Vancouver Python/Zope Users' Group
http://www.vanpyz.org

Outline

- Python 簡介
- 內建資料形態
- 流程控制
- 函式與模組
- 今天的實作題

What is Python?

- Python is an easy to learn, powerful programming language.
  - Efficient high-level data structures
    - lists, hash table (dictionary), sets
  - Elegant syntax and dynamic typing
  - Up-and-coming language in the open source world

Running python code

- Interactive interpreters
  - Type your code after the prompt ">>>"

```
>>> 5 + 3 * 4
17
```

- Python scripts
  - Make a .py file with your code in it
  - One step interpretation (no compile, link steps)

```
$ python foo.py
```

Python is dynamically typed

```
>>> width = 20
>>> print(width)
20
>>> height = 5 * 9
>>> print(height)
45
>>> print(width * height)
900
>>> width = "really wide"
>>> print(width)
really wide
```

Coding notes

- Continuing Statements
    - Continue one line with a backslash \

```
>>> a_long_variable_name = \
    a_long_function_name( param1, param2)
```

- Indentation (4 spaces, not tabs)

```
if this_function(that_variable):
    do_something()
else:
    do_something_else()
```

- Comment
    - Start comment with a number sign #

Outline

- Python 簡介
- 內建資料形態
- 流程控制
- 函式與模組
- 今天的實作題

Numeric Types

- **Int** : integral number , e.g. "x=5"
- **Float** : accuracy depends on platform , e.g. "x=3.14"
- **Bool** : True, False
- Convert from one type to another

```
>>> x=  int(“5”)
>>> y=  int(5.9)
```

Math Operations

- Basic operations

```
>>> 5+3
>>> 5*2+3
>>> 3+5*2
```

- Integer Division

```
>>> 5/2
2
>>> 5.0/2
2.5
>>> float(5) / 2
2.5
>>> 5.0//2
2
>>> 5.0 % 2
1.0
```

Strings

- Can be enclosed in single or double quotes:

```
>>>  print("spam eggs”)
spam eggs
>>>  #backslash escapes quotes
>>>  print(' "Isn\'t," she said.’)
"Isn't," she said.
```

- Basic string operations

```
>>>  myStr = "abc"                # assignment
>>>  myStr = myStr + "def"  # = "abcdef“
>>>  myStr = "abc"*3            # = "abcabcabc“
>>>  for char in myStr:         # iterate
             print(char)
>>>  myStr = str(5)               # = ‘5’
>>>  myStr = str([1,2])          # = ‘[1, 2]’
```

String methods

```
>>> s = "hello there"
>>> print(s.replace("h", “j"))        # jello tjere
>>> print(s.capitalize())             # Hello there
>>> print(s.title())                  #Hello There
>>> print(s.upper())                  #HELLO THERE
>>>  string = "positively python powered"
>>>  print(string.count("po"))         #2
>>>  print(string.startswith("abc")) #False
>>>  print(string.endswith("ed"))    #True
>>>  print(string.find("pow"))       #18
```

String Formatting

```
>>> print("(%s, %d, %4.2f)" % (str1, int1, real1))
(orange, 12, 3.5)
```

Unicode

- Unicode data type:

  normal string: “hello”
  unicode string: u“hello”

- Encode/ decode a string

```
>>> string = 'hello'
>>> string.decode('cp950')   # u'hello'
```

- Coding in different system

  Windows : cp950
  Linux/Unix : utf-8

Lists

```
>>> List1 = ["a", 5, 3.25]
>>> List2 = ["a",  ["3", "2"]]
>>> List3 = List1 + List2
>>> print List3
["a", 5, 3.25, "a",  ["3", "2"]]
>>> range(5)
[0, 1, 2, 3, 4]
>>> [ i*i for i in range(5)]
[0, 1, 4, 9, 16]
```

Splitting and Joining

```
>>> names = ["Peter", "Paul","Mary"]
>>> joined =  ' and '.join(names)
>>> joined   'Peter and Paul and Mary'
>>> joined.split(' and ')
['Peter', 'Paul', 'Mary']
>>> ' and '. joined.split()
['Peter',‘and',  'Paul',‘and',  'Mary']
```

Sequence Types

- Sequence types can be looped over, indexed, sliced ...
  - **Strings** “1a3”
  - **Lists** [1,"a",3]
  - **Tuples** (1,2,"b")
- Sequence operation
  - **Iteration**

    ```
    >>> for i in myList:
    >>>     print(i)
    ```

  - **Numeric indexing** `myList[3]`
  - **Slicing** `mylist[2:5]`

Sequences Operations

- Iterating over sequences

```
>>> strlist = ["abc", "def", "ghi"]
>>> for item in strlist:
        for char in item:
            print(char)
```

- Sequence Concatenation

```
>>> word = 'Help' + 'Me'              # == HelpMe
>>> list = ["Hello"] + ["World"]  # == ['Hello', 'World']
```

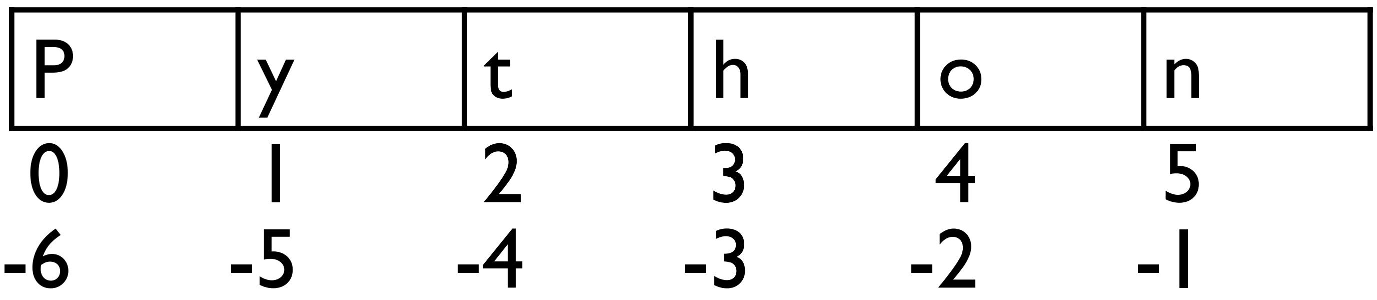
- Getting sequence length

```
>>> len( "abc" )                      # == 3
>>> len( ["abc","def"] )              # == 2
```

Sequence Indexing

```
>>> str="abc"
>>> print(str[0])     # a
>>> print(str[1])     # b
>>> print(str[-1])    # c
```

- Negative indices
  - Counting forward: 0,   1, 2, ..., n-1
  - Counting backward: -1, -2, -3, ...,   -1



P	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

Sequence Slicing

- Basic slice form sequence[x:y]

```
>>> word = "Python"
>>> word[1]          # y
>>> word[0:2]        # Py
>>> word[:2]         # Py
>>> word[2:4]        # th
>>> word[-1]         # n
>>> word[0:-1]       # Pytho
>>> word[0:-2]       # Pyth
>>> word[2:-2]       # th
>>> word[–2:]        # on
```

List Comprehension

```
>>> word = 'generataed'
>>> splits = [(word[:i], word[i:]) for i in range(len(word)+1)]
>>> splits
[('', 'generataed'),('g, 'enerataed'),
   ('ge', 'nerataed'),('gen', 'erataed'),
   ('gene', 'rataed'),('gener', 'ataed'),
   ('genera', 'taed'),('generat', 'aed'),
   ('generata', 'ed'),('generatae', 'd'),
   ('generataed', '')]
>>> deletes = [a + b[1:] for a, b in splits if b]
>>> deletes
['enerataed','gnerataed','genrataed',
   'geneataed','genertaed','generaaed',
   'generated','generatad','generatae']
```

List Comprehension

```
>>> word = 'generataed'
>>> splits = [(word[:i], word[i:]) for i in range(len(word)+1)]
>>> splits
[ ('', 'generataed'),('g, 'enerataed'),
   ('ge', 'nerataed'),('gen', 'erataed'),
   ('gene', 'rataed'),('gener', 'ataed'),
   ('genera', 'taed'),('generat', 'aed'),
   ('generata', 'ed'),('generatae', 'd'),
   ('generataed', '')]
>>> deletes = [a + b[1:] for a, b in splits if b]
>>> deletes
[ 'enerataed','gnerataed','genrataed',
   'geneataed','genertaed','generaaed',
   'generated','generatad','generatae']
```

Sorting List

Sorting -- generating new value data vs. sorting *in place*

```
>>> sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]

>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
```

sorting *in place*

```
>>> students = [
        ('john', 'A', 15),
        ('dave', 'B', 10),
        ('jane', 'B', 12) ]
>>> students.sort(key=lambda x: x[2], reverse=True)# sort by age
>>> students
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

Bindings

- When we assign a variable, we establish another reference or “binding” to the original value.

  ```
  >>>  a=b    # the same object
  ```

  If you change a, you change b!

-
  ```
  >>>  a= (b)    # the same object
  ```

  - No binding

Mutability

Lists Are Mutable

- Lists can be changed "in-place"

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> b = a
>>>  print(b)
['spam', 'eggs', 100, 1234]
>>>  a[2] = 5.5
>>>  print(a)
['spam', 'eggs', 5.5, 1234]
>>>  print(b)
['spam', 'eggs', 5.5, 1234]
```

- Strings are not mutable

```
>>> a = "abcdefgh"; a[3]="k"
** TypeError
```

Tuples Are Not Mutable

- Immutable list-like objects are called **tuples**

```
>>> tup = (“a”, 1, 5.3, 4)
>>> a[3]="k"
Traceback (innermost last):
File "<stdin>", line 1, in ?
TypeError: object doesn't have assignment
```

- Use tuples not lists as key for **dictionary**

Working With Tuples (1)

- Tuples can be returned from functions.
- Easy to return multiple values without defining object and class.

```
>>> import time

>>> (year, month, day, hr, min, sec, wday, ydays, saving) = time.localtime()
>>> print y year, month, day, hr, min, sec, wday, ydays, saving
2012 9 17 13 25 25 0 261 0
```

Working With Tuples (2)

- Tuples can be used to pass parameters

```
>>> def foo(a, b, c):
        return a+b+c
>>> x = (1, 2, 3)
>>> foo(x)
TypeError: foo() takes exactly 3 arguments (1 given)
>>> x = (1, 2, 3)
>>> foo(*x)
6
```

Tuple Shortcut

- We can usually leave out the parens:

```
>>> j=1,2
>>> j=(1,2)       # same as above
>>> a,b = 1,2
>>> a,b = b,a
>>> j=[1,2]
>>> a,b=j
>>> x,y = getXYCoords()
```

Dictionaries

- Serve as a lookup table
- Maps "keys" to "values".
- Keys can be of any **immutable** type
- Assignment adds or changes members
- keys() method returns keys

Dictionaries

```
>>> mydict={"a":"alpha", "b":"bravo","c":"charlie"}
>>> mydict["abc"]=10
>>> mydict[5]="def"
>>> mydict[2.52]=6.71
>>> print(mydict)
{2.52: 6.71, 5: 'def', 'abc': 10, 'b': 'bravo', 'c': 'charlie', 'a': 'alpha'}
```

Constructing Dictionaries

- Dictionaries can be constructed directly or by using “dict”

```
>>> list_of_tuples = [("a", "alpha"), ("b", "bravo"), ("c", "charlie")]

    or

    >>> mydic = {}

    >>> mydic[ "a" ] =  "alpha"

    >>> mydic[ "b" ],  mydic[ "c" ] =  "bravo", "charlie"

>>> mydict = dict(list_of_tuples)

>>> print(mydict)

{'a': 'alpha', 'c': 'charlie', 'b': 'bravo'}
```

Dictionary Methods

```
>>> mydict={"a":"alpha", "b":"bravo", "c":"charlie"}
>>> mydict.keys()
['a', 'c', 'b']
>>> mydict.values()
['alpha', 'charlie', 'bravo']
>>> mydict.items()
[('a', 'alpha'), ('c', 'charlie'), ('b', 'bravo')]
>>> mydict.clear()
>>> print(dict)
{}
```

Dictionary for word counts

```
>>> model = {}
>>> if 'generataed'  in  model:
. . .    model [ 'generataed'] += 1
. . . else:
. . .    model [ 'generataed'] = 1
```

Put the code in function ‘count’

```
>>> def count(word):
. . .    if word  in  model:
. . .       model [ word ] += 1
. . .   else:
. . .        model [ word ] = 1
```

Calling count

```
>>> count('generataed')
```

File Objects

- File objects represent opened files:

```
>>> infile = open( "catalog.txt", "r" )
>>> data = infile.read()
>>> infile.close()
>>> outfile = open( "catalog2.txt", "w" )
>>> data = data+ "more data"
>>> outfile.write( data )
>>> outfile.close()
```

- You may sometimes see the name “open” used to create files. That is an older name.

System Enviroment

```
import sys
print sys.argv
```

sys_ex.py

```
>>> python sys_ex.py 1 2 3
['sysex.py', '1', '2', '3']
```

Working With Lines of Files

- Process your file line by line

```
>>> infile = open( "catalog2.txt", "r")
>>> for line in infile:
        print line

>>> infile.close()
>>> infile = open( "catalog2.txt", "r")
>>> lines = infile.readlines()
>>> lines = list(infile)
```

Working With Lines of Files

- Process really big files
    - read and process one line at a time

```
fh = open( "catalog2.txt", "r")
while True:
    line = fh.readline()
    if line = ‘’:
        break
    <handle line>
fh.close()
```

Outline

- Python 簡介
- 內建資料形態
- 流程控制
- 函式與模組
- 今天的實作題

Basic Flow Control

- if / elif / else (test condition)
- while (loop until condition changes)
- for (iterate over iteraterable object)

if Statement

```
if j=="Hello":
    doSomething()
elif j=="World":
    doSomethingElse()
else:
    doTheRightThing()
```

while Statement

```
str= ""
while str!="quit":
        str=raw_input()
        print str
print("Done")
```

Breaking Out

- break statement allows you to jump out of the middle of a loop

```
while str!="quit":
    str=raw_input()
    if str=="exit":
        break
    print(str)
print("Done")
```

Continue Statement

- The continue statement is a short-cut way of jumping to the top of the loop.

```
string = ""
while string!="quit":
    string = raw_input()
    if string=="wait":
        continue
    print("String was not wait", string)
```

for Statement

-
  ```
  myList = ["a", "b", "c", "d", "e"]
  for item in myList:
      print(item)
  for i in range( 10 ):
      print(i)
  for i in range( len( myList ) ):
      if myList[i]=="c":
          myList[i]=None
  for i, item in enumerate( myList ):
      print(i, item)
  ```

Function Definitions

- Encapsulate bits of code.
- Can take a fixed or variable number of arguments.
- Arguments can have default values.

Function Definition

```
def double( a ):
  return a*2

def quadruple( a ):
  return double( double( a ) )

print quadruple( 8 )
```

Return Values

- The return statement returns a value.
- If no return statement is executed the function returns the value None.
- These are the same:

```
def display( a ):
    print(a)
    return None
```

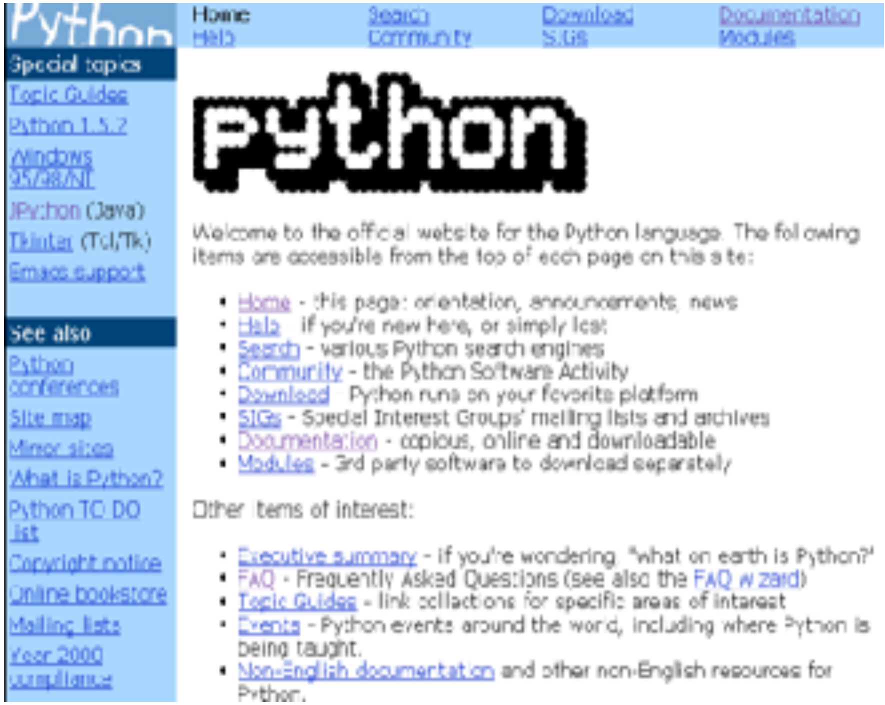
```
def display( a ):
    print(a)
```

Python Resources

The Python website

- Run by the PSF
- FAQ (Frequently Asked Questions)
- Downloads
- Standard Documentation Set
- Search engines
- PyPi

www.python.org



Home Search Download Documentation
Help Community SIGs Modules

Special topics

Topic Guides
Python 1.5.2
Windows 95/98/NT
JPython (Java)
Tkinter (Tcl/Tk)
Emacs support

See also

Python conferences
Site map
Mirror sites
What is Python?
Python TO DO list
Copyright notice
Online bookstore
Mailing lists
Year 2000 compliance

Welcome to the official website for the Python language. The following items are accessible from the top of each page on this site:

- Home - this page: orientation, announcements, news
- Help - if you're new here, or simply lost
- Search - various Python search engines
- Community - the Python Software Activity
- Download - Python runs on your favorite platform
- SIGs - Special Interest Groups' mailing lists and archives
- Documentation - copious, online and downloadable
- Modules - 3rd party software to download separately

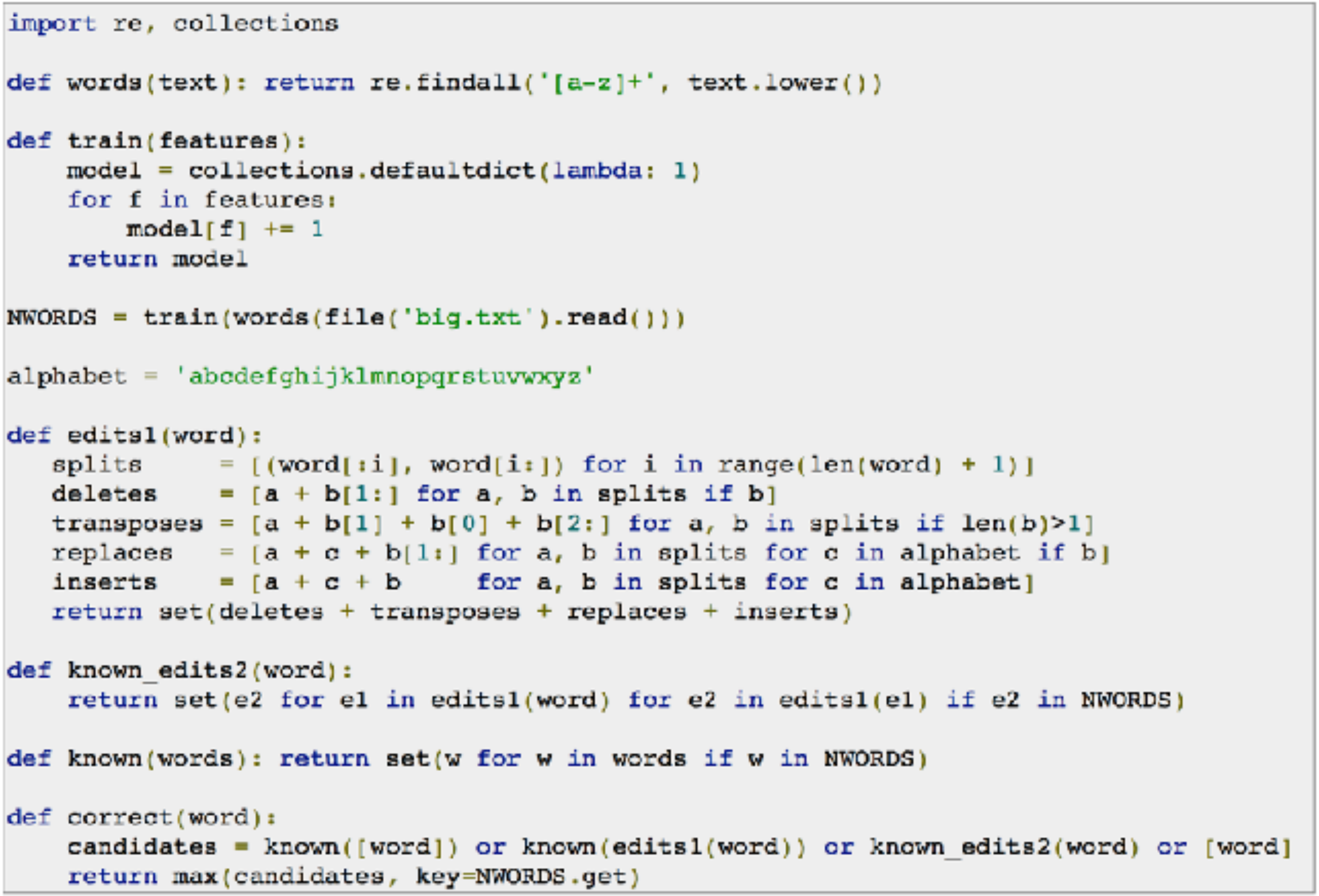
Other items of interest:

- Executive summary - if you're wondering "what on earth is Python?"
- FAQ - Frequently Asked Questions (see also the FAQ wizard)
- Topic Guides - link collections for specific areas of interest
- Events - Python events around the world, including where Python is being taught.
- Non-English documentation and other non-English resources for Python.

Assignment

Expand the program in http://norvig.com/spell-correct.html to deal with the follow types of spelling error:

- Fusion errors (e.g. “taketo” → “take to”)
- Multi-token errors (e.g. “mor efun” → “more fun”)
- Fusion errors (e.g. “with out” → “without”)



```python
import re, collections

def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(file('big.txt').read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word)) or known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)
```

Downloadable Python Books

- *How to think like a Computer Scientist -* introductory programming book that comes in Python and Java version. by Downey, Elkner, and Meyers
- *Dive Into Python* - free Python book for experienced programmers. By Mark Pilgrim
- *Thinking In Python -* for intermediate Python programmers. By Bruce Eckel
- *Python Text Processing with NLTK 2.0 Cookbook*. By Jacob Perkins