

# Introduction to Programming in Python (3)

Steve Renals  
s.renals@ed.ac.uk

ICL — 2 October 2005

## NLTK: Python Natural Language Toolkit

- ▶ NLTK is a set of Python modules which you can import into your programs, eg:

```
from nltk_lite.utilities import re_show
```

## NLTK: Python Natural Language Toolkit

- ▶ NLTK is a set of Python modules which you can `import` into your programs, eg:

```
from nltk_lite.utilities import re_show
```

- ▶ NLTK is distributed with several *corpora* (singular: *corpus*). A corpus is a body of text (or other language data, eg speech).
- ▶ Example corpora with NLTK: `gutenberg` (works of literature from project Gutenberg), `treebank` (parsed text from the (part of) the Penn treebank), `brown` (the first million word, PoS-tagged corpus — 1961!)

## NLTK: Python Natural Language Toolkit

- ▶ NLTK is a set of Python modules which you can import into your programs, eg:

```
from nltk_lite.utilities import re_show
```

- ▶ NLTK is distributed with several *corpora* (singular: *corpus*). A corpus is a body of text (or other language data, eg speech).
- ▶ Example corpora with NLTK: `gutenberg` (works of literature from project Gutenberg), `treebank` (parsed text from the (part of) the Penn treebank), `brown` (the first million word, PoS-tagged corpus — 1961!)
- ▶ Load a corpus (eg `gutenberg`) using:

```
>>> from nltk_lite.corpora import gutenberg  
>>> print gutenberg.items
```

```
['austen-emma', 'austen-persuasion', 'austen-sense', 'b
```

## Simple corpus operations

- ▶ Simple processing of a corpus includes *tokenization* (splitting the text into word tokens), text normalization (eg by case), then many possible operations such as obtaining word statistics, *tagging* and *parsing*

## Simple corpus operations

- ▶ Simple processing of a corpus includes *tokenization* (splitting the text into word tokens), text normalization (eg by case), then many possible operations such as obtaining word statistics, *tagging* and *parsing*

- ▶ Count the number of words in “Macbeth”

```
from nltk_lite.corpora import gutenberg  
nwords = 0
```

```
#iterate over all word tokens in Macbeth  
for word in gutenberg.raw('shakespeare-macbeth'):  
    nwords += 1  
print nwords # 23939
```

## Simple corpus operations

- ▶ Simple processing of a corpus includes *tokenization* (splitting the text into word tokens), text normalization (eg by case), then many possible operations such as obtaining word statistics, *tagging* and *parsing*

- ▶ Count the number of words in “Macbeth”  

```
from nltk_lite.corpora import gutenberg
nwords = 0
```

```
#iterate over all word tokens in Macbeth
for word in gutenberg.raw('shakespeare-macbeth'):
    nwords += 1
print nwords                                     # 23939
```

- ▶ `gutenberg.raw(<textname>)` is an *iterator*, which behaves like a sequence (eg a list) except it returns elements one at a time as requested

## Richer corpora

- ▶ The Gutenberg corpus is tokenized as a sequence of words, with no further structure.



## Richer corpora

- ▶ The Gutenberg corpus is tokenized as a sequence of words, with no further structure.
- ▶ The Brown corpus has sentences marked, and is stored as a list of sentences, where a sentence is a list of word tokens. We can use the `extract` function to obtain individual sentences

```
from nltk_lite.corpora import brown
from nltk_lite.corpora import extract
```

```
firstSentence = extract(0, brown.raw('a'))
```

```
# ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Frid
```

## Richer corpora

- ▶ The Gutenberg corpus is tokenized as a sequence of words, with no further structure.
- ▶ The Brown corpus has sentences marked, and is stored as a list of sentences, where a sentence is a list of word tokens. We can use the `extract` function to obtain individual sentences

```
from nltk_lite.corpora import brown
from nltk_lite.corpora import extract
```

```
firstSentence = extract(0, brown.raw('a'))
```

```
# ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Frid
```

- ▶ Part-of-speech tagged text can also be extracted:

```
taggedFirstSentence = extract(0, brown.tagged('a'))
```

```
# [('The', 'at'), ('Fulton', 'np-tl'), ('County', 'nn-tl'),
```

## Parsed text

Parsed text from the Penn treebank can also be accessed:

```
>>> from nltk_lite.corpora import treebank
>>> parsedSent = extract(0, treebank.parsed())
>>> print parsedSent
(S:
  (NP-SBJ:
    (NP: (NNP: 'Pierre') (NNP: 'Vinken'))
    (,: ',')
    (ADJP: (NP: (CD: '61') (NNS: 'years')) (JJ: 'old'))
    (,: ','))
  (VP:
    (MD: 'will')
    (VP:
      (VB: 'join')
      (NP: (DT: 'the') (NN: 'board'))
      (PP-CLR:
        (IN: 'as')
        (NP: (DT: 'a') (JJ: 'nonexecutive') (NN: 'director'))
```

## Count the frequency of each word in *Macbeth*

```
from nltk_lite.corpora import gutenbergs

count = {} # initialize dictionary

for word in gutenbergs.raw('shakespeare-macbeth'):
    word = word.lower() # normalize case
    if word not in count: # previously unseen word?
        count[word] = 0 # if so set count to 0
    count[word] += 1 # increment word count
```

## Count the frequency of each word in *Macbeth*

```
from nltk_lite.corpora import gutenbergl

count = {} # initialize dictionary

for word in gutenbergl.raw('shakespeare-macbeth'):
    word = word.lower() # normalize case
    if word not in count: # previously unseen word?
        count[word] = 0 # if so set count to 0
    count[word] += 1 # increment word count
```

We can inspect the dictionary:

```
print count['scotland'] # 12
print count['thane'] # 25
print count['blood'] # 24
print count['duncan'] # 10
```

## Sorting by frequency

We would like to sort the dictionary by frequency, but:

- ▶ If we just sort the values (`count.values()`) we lose the link to the keys

## Sorting by frequency

We would like to sort the dictionary by frequency, but:

- ▶ If we just sort the values (`count.values()`) we lose the link to the keys
- ▶ `count.items()` returns a list of the pairs, but naively sorting that list doesn't do what we want:

```
wordfreq = count.items()  
wordfreq.sort()          # WRONG! - sorted by word!
```

## Sorting by frequency

We would like to sort the dictionary by frequency, but:

- ▶ If we just sort the values (`count.values()`) we lose the link to the keys
- ▶ `count.items()` returns a list of the pairs, but naively sorting that list doesn't do what we want:

```
wordfreq = count.items()  
wordfreq.sort()          # WRONG! - sorted by word!
```

We will show five different ways of doing this right in Python...



## Sorting by word frequency (1)

One way to do it:

```
wordfreq = count.items()
res = []
for wf in wordfreq:
    res.append((wf[1], wf[0]))

res.sort()
res.reverse()
print res[:10]
```

## Sorting by word frequency (2)

slightly less clunky

```
wordfreq = count.items()  
res = []
```

```
#for wf in wordfreq:  
#    res.append((wf[1], wf[0]))
```

```
# explicitly assign to elements of a tuple  
for (w, f) in wordfreq:  
    res.append((f, w))
```

```
res.sort()  
res.reverse()
```

## Sorting by word frequency (3)

Could even use a list comprehension:

```
wordfreq = count.items()

#res = []
#for (w, f) in wordfreq:
#    res.append((f, w))

# use a list comprehension instead
res = [(f, w) for (w, f) in wordfreq]

res.sort()
res.reverse()
```

## Sorting by word frequency (4)

- ▶ The `sort` function uses a comparator function `cmp(x,y)` which returns negative if  $x < y$ , zero if  $x == y$ , positive if  $x > y$ .

## Sorting by word frequency (4)

- ▶ The sort function uses a comparator function `cmp(x,y)` which returns negative if  $x < y$ , zero if  $x == y$ , positive if  $x > y$ .
- ▶ You can define your own comparator which sorts on the second element of each item, and sorts with biggest first:

```
def mycmp(s1, s2):  
    return cmp(s2[1], s1[1])
```

## Sorting by word frequency (4)

- ▶ The sort function uses a comparator function `cmp(x,y)` which returns negative if  $x < y$ , zero if  $x == y$ , positive if  $x > y$ .
- ▶ You can define your own comparator which sorts on the second element of each item, and sorts with biggest first:

```
def mycmp(s1, s2):  
    return cmp(s2[1], s1[1])
```

- ▶ Making sorting by word frequency straightforward:

```
wordfreq = count.items()  
wordfreq.sort(mycmp)  
print wordfreq[:10]
```

## Sorting by word frequency (5)

Finally, can even use an anonymous comparator function:

```
wordfreq = count.items()  
wordfreq.sort(lambda s1, s2: cmp(s2[1], s1[1]))  
print wordfreq[:10]
```

## Regular expressions in Python

- ▶ The Python regular expression module `re` can be used for matching, substituting and searching within strings:

```
>>> import re
>>> from nltk_lite.utilities import re_show
>>> s = "Introduction to Computational Linguistics"
>>> re_show('i', s)
Introduct{i}on to Computat{i}onal L{i}ngu{i}st{i}cs
>>> re_show('tion', s)
Introduc{tion} to Computa{tion}al Linguistics
```



## Regular expressions in Python

- ▶ The Python regular expression module `re` can be used for matching, substituting and searching within strings:

```
>>> import re
>>> from nltk_lite.utilities import re_show
>>> s = "Introduction to Computational Linguistics"
>>> re_show('i', s)
Introduct{i}on to Computat{i}onal L{i}ngu{i}st{i}cs
>>> re_show('tion', s)
Introduc{tion} to Computa{tion}al Linguistics
```

- ▶ We can perform a substitution with `re.sub`

```
t = re.sub('tion', 'XX', s)
# t = 'IntroducXX to ComputaXXal Linguistics'
```

Remember strings are immutable; `re.sub` returns a new string

## Disjunction in regular expressions

We can look for one string or another

```
u = re.findall('(ti|c)', s)
# u= ['c', 'ti', 'c', 'ti', 'ti', 'c']
```

## Disjunction in regular expressions

We can look for one string or another

```
u = re.findall('(ti|c)', s)
# u= ['c', 'ti', 'c', 'ti', 'ti', 'c']
```

Or we can disjoin characters, eg `[aeiou]` matches any of a, e, i, o or u (vowels), and `[^aeiou]` matches anything that is not a vowel. So we can match sequences finding non-vowels followed by vowels:

```
v = re.findall('[aeiouAEIOU][^aeiouAEIOU]', s)
# v = ['In', 'od', 'uc', 'on', 'o ', 'om', 'ut', 'at', 'on', 'al']
```

More on regular expressions next week.

## (A brief introduction to) Classes in Python

- ▶ Classes are general data structures containing
  - ▶ *Attributes* — Data: components, parts, properties, etc.
  - ▶ *Methods* — operations that can be performed on the data

## (A brief introduction to) Classes in Python

- ▶ Classes are general data structures containing
  - ▶ *Attributes* — Data: components, parts, properties, etc.
  - ▶ *Methods* — operations that can be performed on the data
- ▶ The Python class system handles this
  - ▶ Define a class using statement `class`
  - ▶ Attributes are defined when first assigned - no need to declare in advanced
  - ▶ All methods have a first parameter that corresponds to the object, named (by convention) `self`
  - ▶ Special methods: `__init__` is called when the object is created; `__str__` is called when a `print` statement is called on the object

## Example Person class

```
class Person:
    def __init__(self, givenName, familyName):
        self.givenName = givenName
        self.familyName = familyName

    def fullName(self):
        return self.givenName + ' ' + self.familyName

    def __str__(self):
        return '<Person: givenName=' + self.givenName +
            ' familyName=' + self.familyName
```

## Creating a Person object

```
>>> from Person import *
>>> p = Person('Steve', 'Renals')
>>> print p
<Person: givenName=Steve familyName=Renals>
>>> p.fullName()
'Steve Renals'
```

## Creating a Person object

```
>>> from Person import *
>>> p = Person('Steve', 'Renals')
>>> print p
<Person: givenName=Steve familyName=Renals>
>>> p.fullName()
'Steve Renals'

>>> p.address=('Buccleuch Place')>
>> print p
<Person: givenName=Steve familyName=Renals>
>>> p.address
'Buccleuch Place'
```



## Creating a Person object

```
>>> from Person import *
>>> p = Person('Steve', 'Renals')
>>> print p
<Person: givenName=Steve familyName=Renals>
>>> p.fullName()
'Steve Renals'

>>> p.address=('Buccleuch Place')
>> print p
<Person: givenName=Steve familyName=Renals>
>>> p.address
'Buccleuch Place'

>>> p.age
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: Person instance has no attribute 'age'
```

# Inheritance

Classes can inherit from a *superclass*:

```
class Student(Person):  
    def __init__(self, givenName, familyName, num):  
        Person.__init__(self, givenName, familyName)  
        self.matricNumber = num  
  
    def __str__(self):  
        return '<Student:' + Person.__str__(self) +  
            ' matricNumber=' + str(self.matricNumber) + '>'
```

## Creating a Student object

```
>>> from Person import *
>>> s = Student('Steve', 'Renals', 123456)
>>> print s
<Student:<Person: givenName=Steve familyName=Renals> matricNumber=123456>
>>> s.matricNumber
123456
```

# Summary

- ▶ The NLTK toolkit
- ▶ Accessing (raw / tagged / parsed) corpora from NLTK
- ▶ Five ways to sort words by corpus frequency
- ▶ Brief introduction to classes in Python