Lesson 10: List Comprehension, Working with Large Texts
Objectives

- Review of Homework #3

- List comprehension

- Practice:
  - working with your custom module in IDLE shell
  - working with large text files
HW#3: Washington vs. Obama

<table>
<thead>
<tr>
<th>Washington (1789)</th>
<th>Obama (2009)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token count: <strong>1539</strong></td>
<td>Token count: <strong>2727</strong></td>
</tr>
<tr>
<td>Type count: <strong>603</strong></td>
<td>Type count: <strong>899</strong></td>
</tr>
<tr>
<td>TTR: <strong>0.391812865497</strong></td>
<td>TTR: <strong>0.329666299963</strong></td>
</tr>
</tbody>
</table>

Washington's TTR is higher. Richer vocabulary in Washington? ➔ NOPE. Washington's speech is MUCH shorter!

- Sentence count: **23**
- Average sent length: **66.91**
- Average word length (symbols excluded): **4.94**

- Sentence count: **110**
- Average sent length: **24.79**
- Average word length (symbols excluded): **4.41**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>116</td>
<td>135</td>
</tr>
<tr>
<td>of</td>
<td>71</td>
<td>0.049504950495</td>
</tr>
<tr>
<td>,</td>
<td>70</td>
<td>0.04767143381</td>
</tr>
<tr>
<td>to</td>
<td>48</td>
<td>0.040704070407</td>
</tr>
<tr>
<td>and</td>
<td>48</td>
<td>0.039970663733</td>
</tr>
<tr>
<td>which</td>
<td>36</td>
<td>0.030069673634</td>
</tr>
<tr>
<td>in</td>
<td>31</td>
<td>0.02566923359</td>
</tr>
<tr>
<td>.</td>
<td>23</td>
<td>0.024569123579</td>
</tr>
<tr>
<td>i</td>
<td>23</td>
<td>0.022735606894</td>
</tr>
<tr>
<td>be</td>
<td>23</td>
<td>0.017968463513</td>
</tr>
<tr>
<td>my</td>
<td>22</td>
<td>0.017235056839</td>
</tr>
<tr>
<td>by</td>
<td>20</td>
<td>0.013201320132</td>
</tr>
<tr>
<td>that</td>
<td>18</td>
<td>0.0129954515919</td>
</tr>
<tr>
<td>with</td>
<td>17</td>
<td>0.0116959064327</td>
</tr>
<tr>
<td>on</td>
<td>15</td>
<td>0.0110461338532</td>
</tr>
<tr>
<td>a</td>
<td>14</td>
<td>0.00974658869396</td>
</tr>
<tr>
<td>as</td>
<td>14</td>
<td>0.00909681611436</td>
</tr>
<tr>
<td>have</td>
<td>12</td>
<td>0.00909681611436</td>
</tr>
<tr>
<td>for</td>
<td>12</td>
<td>0.00779727095517</td>
</tr>
<tr>
<td>it</td>
<td>11</td>
<td>0.00714749837557</td>
</tr>
<tr>
<td>,</td>
<td>130</td>
<td>0.00779727095517</td>
</tr>
<tr>
<td>and</td>
<td>111</td>
<td>0.00779727095517</td>
</tr>
<tr>
<td>.</td>
<td>109</td>
<td>0.00714749837557</td>
</tr>
</tbody>
</table>
Only in one speech

<table>
<thead>
<tr>
<th>Washington (1789)</th>
<th>Obama (2009)</th>
</tr>
</thead>
<tbody>
<tr>
<td>me</td>
<td>'</td>
</tr>
<tr>
<td>present</td>
<td>s</td>
</tr>
<tr>
<td>under</td>
<td>america</td>
</tr>
<tr>
<td>being</td>
<td>because</td>
</tr>
<tr>
<td>duty</td>
<td>what</td>
</tr>
<tr>
<td>myself</td>
<td>do</td>
</tr>
<tr>
<td>ought</td>
<td>let</td>
</tr>
<tr>
<td>since</td>
<td>cannot</td>
</tr>
<tr>
<td>into</td>
<td>common</td>
</tr>
<tr>
<td>measures</td>
<td>today</td>
</tr>
</tbody>
</table>
Top 20 words favored by

<table>
<thead>
<tr>
<th>Washington (1789)</th>
<th>Obama (2009)</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>.</td>
</tr>
<tr>
<td>which</td>
<td>our</td>
</tr>
<tr>
<td>of</td>
<td>we</td>
</tr>
<tr>
<td>i</td>
<td>and</td>
</tr>
<tr>
<td>my</td>
<td>is</td>
</tr>
<tr>
<td>in</td>
<td>a</td>
</tr>
<tr>
<td>be</td>
<td>us</td>
</tr>
<tr>
<td>by</td>
<td>--</td>
</tr>
<tr>
<td>with</td>
<td>that</td>
</tr>
<tr>
<td>to</td>
<td>are</td>
</tr>
<tr>
<td>an</td>
<td>but</td>
</tr>
<tr>
<td>as</td>
<td>they</td>
</tr>
<tr>
<td>your</td>
<td>new</td>
</tr>
<tr>
<td>government</td>
<td>not</td>
</tr>
<tr>
<td>public</td>
<td>who</td>
</tr>
<tr>
<td>on</td>
<td>nation</td>
</tr>
<tr>
<td>more</td>
<td>;</td>
</tr>
<tr>
<td>every</td>
<td>-</td>
</tr>
<tr>
<td>citizens</td>
<td>this</td>
</tr>
<tr>
<td>his</td>
<td>,</td>
</tr>
</tbody>
</table>

3/19/2014
Anything else?

- What discoveries did you make?
- What sort of linguistic inquiry is beyond our current Python capability?
  - Who uses more adjectives/adverbs/function words...?
    ➡️ We need a part-of-speech (POS) tagger.
  - Who uses more relative clauses?
    ➡️ We need a syntactic parser.
  - Who likes "rarer" vocabulary rather than plain words?
    ➡️ We need a large-scale English vocabulary ranking
Filtering a list

```python
>>> mary = 'Mary had a little lamb, whose fleece was white as snow.' .split()
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow. ']
```

How to make a list of words that have 'a'?

```python
>>> alist = []
>>> for w in mary:
...     if 'a' in w:
...         alist.append(w)

>>> alist
['Mary', 'had', 'a', 'lamb,', 'was', 'as']
```

You need to make a new empty list, and then iterate through mary to find items to put in
Filtering a list

How to make a list of words that have 'a'?

```python
>>> mary = 'Mary had a little lamb, whose fleece was white as snow.' .split()
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow. ']
```

The power of LIST COMPREHENSION

Creating a new list where elements meet a certain condition:

```
[x for x in list if ...]
```
Try it out

```python
>>> mary = 'Mary had a little lamb, whose fleece was white as snow.' .split()
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow.]
```

- **Syntax:** `[x for x in list if ... ]`  

---

- **Words that have 'a'**
  ```python
  >>> [w for w in mary if 'a' in w]
  ['Mary', 'had', 'a', 'lamb,', 'was', 'as']
  ```

- **Words that are 5 chars or longer**
  ```python
  >>> [w for w in mary if len(w) >= 5]
  ['little', 'lamb,', 'whose', 'fleece', 'white', 'snow.]
  ```

- **Words that are 5 chars or longer and without symbols**
  ```python
  >>> [w for w in mary if len(w) >= 5 and w.isalnum()]
  ['little', 'whose', 'fleece', 'white']
  ```
Try it out

```python
>>> mary = 'Mary had a little lamb, whose fleece was white as snow.' .split()

>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow. ']
```

- **Syntax:** `[x for x in list if ... ]`

- **Words that have 'a'**
  ```python
  >>> [w for w in mary if 'a' in w]
  ['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'as']
  ```

- **Words that are 5 chars or longer**
  ```python
  >>> [w for w in mary if len(w) >=5]
  ['little', 'lamb,', 'whose', 'fleece', 'white', 'snow. ']
  ```

- **Words that are 5 chars or longer and without symbols**
  ```python
  >>> [w for w in mary if len(w) >=5 and w.isalnum()]
  ['little', 'whose', 'fleece', 'white']
  ```

3/19/2014
Transforming items in list

```python
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow.]
```

How to make a new list with uppercase words?

```python
>>> mary.upper()
... AttributeError: 'list' object has no attribute 'upper'
>>> mup = []
>>> for w in mary:
    mup.append(w.upper())

>>> mup
['MARY', 'HAD', 'A', 'LITTLE', 'LAMB,', 'WHOSE', 'FLEECE', 'WAS', 'WHITE', 'AS', 'SNOW.]
```

Cannot uppercase a list

You have to create an empty new list and then put in uppercased words
Transforming items in list

Uppercased list, using list comprehension

```python
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow.]
```

```python
>>> [w.upper() for w in mary]
['MARY', 'HAD', 'A', 'LITTLE', 'LAMB,', 'WHOSE', 'FLEECE', 'WAS', 'WHITE', 'AS', 'SNOW.]
```

Creating a new list where each element is transformed:

```python
[f(x) for x in list]
```
Try it out

```python
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow. ']
```

- Syntax: `[f(x) for x in list]`

- List of first characters
  ```python
  >>> [w[0] for w in mary]
  ['M', 'h', 'a', 'l', 'l', 'w', 'a', 's']
  ```
- List of word lengths
  ```python
  >>> [len(w) for w in mary]
  [4, 3, 1, 6, 5, 5, 6, 3, 5, 2, 5]
  ```
- List of True/False for having 'a' as substring
  ```python
  >>> ['a' in w for w in mary]
  [True, True, True, False, True, False, False, True, False, True, False]
  ```
Try it out

```python
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow.]
```

Syntax: `[f(x) for x in list]`

- List of first characters
- List of word lengths
- List of True/False for having 'a' as substring

```python
>>> [w.upper() for w in mary]
['MARY', 'HAD', 'A', 'LITTLE', 'LAMB', 'WHOSE', 'FLEECE', 'WAS', 'WHITE', 'AS', 'SNOW.]
```

```python
>>> [w[0] for w in mary]
['M', 'h', 'a', 'l', 'l', 'w', 'w', 'w', 'a', 's']
```

```python
>>> [len(w) for w in mary]
[4, 3, 1, 6, 5, 5, 6, 3, 5, 2, 5]
```

```python
>>> ['a' in w for w in mary]
[True, True, True, False, True, False, False, True, False, True, False]
```
Try it out

```python
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', 'white', 'as', 'snow.]
```

Words that are 6 chars or longer, in upper case

```python
>>> ['LITTLE', 'FLEECE']
```

Calculate the average word length ... in one line!

```python
>>> [len(w) for w in mary]
[4, 3, 1, 6, 5, 5, 6, 3, 5, 2, 5]
```

... in one line!

```python
>>> sum([len(w) for w in mary]) / len(mary)
4.090909090909091
```

2 minutes
Try it out

```python
>>> mary
['Mary', 'had', 'a', 'little', 'lamb,', 'whose', 'fleece', 'was', '
white', 'as', 'snow. ']
```

**Words that are 6 chars or longer, in upper case**

```python
>>> [w.upper() for w in mary if len(w) >= 6]
['LITTLE', 'FLEECE']
```

**Calculate the average word length ... in one line!**

```python
>>> [len(w) for w in mary]
[4, 3, 1, 6, 5, 5, 6, 3, 5, 2, 5]
>>> sum([len(w) for w in mary])
45
>>> sum([len(w) for w in mary]) / len(mary)
4
>>> sum([len(w) for w in mary]) / float(len(mary))
4.090909090909091
```
List comprehension: summary

- **Syntax:** \([f(x) \text{ for } x \text{ in } \text{list if } ...]\)

```python
>>> mary = 'Mary had a little lamb'.split()
>>> mary
['Mary', 'had', 'a', 'little', 'lamb']

>>> [w for w in mary]
['Mary', 'had', 'a', 'little', 'lamb']

>>> [w for w in mary if len(w) > 3]
['Mary', 'little', 'lamb']

>>> [w for w in mary if 'a' in w]
['Mary', 'had', 'a', 'lamb']

>>> [w.upper() for w in mary]
['MARY', 'HAD', 'A', 'LITTLE', 'LAMB']

>>> [len(w) for w in mary]
[4, 3, 1, 6, 4]
```

- Same as `mary`
- Filter out elements that do not meet a certain condition
- Transform each element in list
Building a Python script for a particular task is important.

But before you are able to do that well, you should be comfortable operating in IDLE shell, exploring text files on the fly.

We will try out, in IDLE shell:
- Importing and using our text processing functions
- Working with BIG text files
Using your **own module** in IDLE shell

- Your module file can be imported in shell if it is in your **WD**.

```python
>>> import textproc
Traceback (most recent call last):
  ... 
ImportError: No module named textproc

>>> import os
>>> os.getcwd()
'D:\Lab'
>>> os.chdir('text-processing')
>>> os.getcwd()
'D:\Lab\text-processing'

>>> import textproc
>>> textproc.py cannot be found

Move into the directory where textproc.py is located

Now you can import the module
```
Import and use `textproc` in IDLE shell

```python
>>> dir(textproc)
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'getFreq', 'getRelFreq', 'getToks', 'getTypes', 'main', 'tale']

>>> help(textproc.getToks)
Help on function getToks in module textproc:

getToks(txt)
    Takes a text, returns a tokenized list of words in lowercase

>>> textproc.getToks('Mary had a little lamb.')
['mary', 'had', 'a', 'little', 'lamb', '.']
```

See what's available in the module

Get help() on a function

Use getToks() function
Download the novel from the Project Gutenberg web site: http://www.gutenberg.org/ebooks/11
- Download and save the "Plain Text UTF-8" file.
- Name it "alice.txt"

Clean up the text file
- Open it up using a text editor. You will find:
  - The file begins with a preamble
  - The file ends with many (over 300!) lines of Project Gutenberg Legalese
-> Remove both parts and save.

Our text: Carroll's Alice's Adventures
Read in the text

```python
>>> f = open(r'D:\Lab\text-processing\alice.txt')
>>> alicetxt = f.read()
>>> f.close()

>>> print alicetxt

>>> print alicetxt[:500]

>>> print alicetxt[10000:10500]

>>> print alicetxt[-500:]

would feel with all their simple sorrows, and find a
pleasure in all their simple joys, remembering her own
child-life, and the happy summer days.

THE END

STOP!!!
This sprays the entire text onto screen.

It will likely freeze your IDLE shell.

Slice indexing is your friend.
Look at first 500 chars,
500 chars in the middle,
and last 500 characters.

3/19/2014
Save your steps

- You WILL be crashing your IDLE through this exercise.
  - Ctrl + c might or might not help.

- Open up a text document and copy over your steps, starting from `textproc importing`
  - So you can quickly get back to where you were when you restart your session.
Tokenize the text

```python
>>> alicetoks = textproc.getToks(alicetxt)

>>> alicetoks

>>> alicetoks[100:150]
['well', 'as', 'she', 'could', ',', 'for', 'the', 'hot', 'day', 'made', 'her', 'feel', 'very', 'sleepy', 'and', 'stupid', ')
', ',', 'whether', 'the', 'pleasure', 'of', 'making', 'a', 'daisy', '-', 'chain', 'would', 'be', 'worth', 'the', 'trouble', '
of', 'getting', 'up', 'and', 'picking', 'the', 'daisies', ',', 'when', 'suddenly', 'a', 'white', 'rabbit', 'with', 'pink', 'eyes', 'ran', 'close
```
Explore the tokens

How long is the text?

```python
>>> len(alicetoks)
35399
```

How many times does 'rabbit' occur?

```python
>>> alicetoks.count('rabbit')
51
```

Does 'curiouser' occur in the text?

```python
>>> 'curiouser' in alicetoks
True
```

How many of the word tokens are symbols?

```python
>>> syms = [w for w in alicetoks if not w.isalnum()]
>>> len(syms)
8057
```
Unique word types

```python
>>> alicetypes = textproc.getTypes(alicetoks)

>>> alicetypes[:30]
[',', '', '', '(', ')', '*', ',', '-', '--', '.
'0', '3', ':', ';', '?', '[', ']', '_', 'a', 'abide', 'able', 'about', 'above',
'absence', 'absurd', 'acceptance', 'accident', 'accidentally', 'account', 'accounting']
```
Explore the types

How many types?

```python
>>> len(alicetypes)
2591
```

What's the type token ratio?

```python
>>> len(alicetypes) / float(len(alicetoks))
0.07319415802706292
```

How many are 13+ characters long?

```python
>>> [w for w in alicetypes if len(w) >=13]
['affectionately', 'circumstances', 'contemptuously', 'conversations', 'disappointment', 'extraordinary', 'inquisitively', 'multiplication', 'straightening', 'uncomfortable', 'uncomfortably']
```

```python
>>> [(w, len(w)) for w in alicetypes if len(w) >=13]
[('affectionately', 14), ('circumstances', 13), ('contemptuously', 14), ('conversations', 13), ('disappointment', 14), ('extraordinary', 13), ('inquisitively', 13), ('multiplication', 14), ('straightening', 13), ('uncomfortable', 13), ('uncomfortably', 13)]
```
Careful with list comprehension

How many are 8 chars or longer?

>>> [w for w in alicetypes if len(w) >=8]

This is going to return a long list!

Unless you're reasonably sure your list is short, assign the list to a new variable first...

>>> foo = [w for w in alicetypes if len(w) >=8]

>>> len(foo)
627

>>> foo[:10]
['acceptance', 'accident', 'accidentally', 'accounting', 'accounts', 'accusation', 'accustomed', 'actually', 'addressed', 'addressing']

... and then look at snippets using slice indexing
Explore the types

How many types are vowel-less and 2+ characters?

```python
>>> def hasVowel(w):
    return 'a' in w or 'e' in w or 'i' in w or 'o' in w or 'u' in w
    
>>> [w for w in alicetypes if not hasVowel(w) and len(w) >= 2]
['--', 'by', 'cry', 'dry', 'fly', 'hjckrrh', 'hm', 'll', 'my', 'sh', 'shy', 'shyly', 'sky', 'try', 'why']
```
Word frequencies

```python
>>> alicefreq = textproc.getFreq(alicetoks)
>>> alicefreq['rabbit']
51
>>> alicefreq['the']
1644
>>> alicefreq['curiouser']
2

>>> alicefreq.keys()[:10]
['secondly', 'pardon', 'saves', 'knelt', 'four', 'sleep', 'hanging', 'ringlets', 'oldest', 'hate']

>>> alicefreq.items()[:10]
[('secondly', 2), ('pardon', 6), ('saves', 1), ('knelt', 1), ('four', 8), ('sleep', 6), ('hanging', 3), ('ringlets', 2), ('oldest', 1), ('hate', 2)]
```
Explore the frequencies

```python
>>> for w in sorted(aflicereq, key=alicefreq.get, reverse=True)[:5] :
    print w, alicefreq[w]

' 2871
, 2418
the 1644
. 990
and 872
>>> once = [w for w in alicefreq if alicefreq[w] == 1]
>>> len(once)
1119
>>> once[:20]
['saves', 'knelt', 'oldest', 'blacking', 'inwards', 'sorry', 'rise', 'jack', 'seals', 'fireplace', 'prize', 'wooden', 'favoured', 'leaders', 'feathers', 'elegant', 'louder', 'machines', 'shining', 'hide']
```

What are the top 5 most frequent words?

How many types occur only once?
Wrap-up

- **Next class**
  - Working with a corpus
  - Pickling

- **Exercise #7**
  - List comprehension galore!
  - You will answer such exciting questions as:
    - What's the longest English word?
    - How many 5+ character palindromes are there?
    - What's the most common English letter to begin a word?