Introduction to Text Mining

David Beales, beales@hawaii.edu

We are going to need two tools for our exploration of Python and text mining:

- a text editor
- a command line tool

The Text Editor

We are using the text editor ATOM, an open source text editor distributed by the same team that makes Github.

You can download ATOM here: https://atom.io/

For this workshop we are just going to use the basic ATOM distribution, and make some small changes to it that will help with our text-mining.

Step 1: Go to **File>Open File**… in the menu bar.

Step 2: Navigate to the folder **pythonclass\intro**, which can be found on the desktop, and open the file **firstfile.txt**

The file is a Python script, not a word processing file, but it is saved as a text file and so we are missing out on one of the benefits of using a text editor designed for coding.

Step 3: Go to **File>Save As...**

Step 4: Change the file extension from **.txt** to **.py**

You can see that the text editor now color codes the various parts of the script to make it easier to work with. This is called syntax highlighting.

We need to save all Python scripts with the **.py** extension if we are going to run them from the command line. Speaking of which...

<u>The Command Line</u>

The command prompt gives you complete, precise, transparent and powerful control over what your computer is doing. The price you pay for this power is a more complex and less intuitive interface. There are a large set of commands and options to use, but there are only a few that we need for today's class.

The first thing we need to do is to enter the directory we will be using for our text files and our Python scripts. We can change directories using the **cd** command.

cd	Go to the top of the directory tree.
cd ..	Go up one directory.
cd \\folder1\\folder2\\folder3	Navigate to a specific directory.

We are going to use the **pythonclass** folder that you can see on the desktop.

To navigate to that folder in the command line we are going to use the following command:

```
cd \Desktop\pythonclass
```

You will see the command prompt change to **C:\Users\306\Desktop\pythonclass**

All the Python scripts that we write will be in this folder or in subdirectories.

<u>Using Python on the Command Line</u>

If you type **python** at the command prompt and hit enter, you will see the Python version and distribution you are using.

```
Python 3.6.0 |Anaconda 4.3.0 (64-bit)| (default, Dec 23 2016, 11:57:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

Type **exit()** and hit enter to go back to the command prompt.

To run a specific Python script, you will need to use the Python command and the name of the file.

Enter **python firstfile.py** into the command prompt.

You can see the results of executing the script in the command prompt, but only because the script includes instructions to show certain information in the command line window.

Enter **python copyfile.py** into the command prompt. There will be two prompts that you will need to provide input for:

Enter file to be copied:	**firstfile.txt**
Enter new filename:	**readthemonths.py**

You will notice that there is no feedback regarding the result of the script; you will have to manually look into the directory to see the new file.

You can either use the windows GUI to see the new file, or you can enter **dir** into the command prompt to see a list of files, directories and data in the current directory.

So…that's a quick intro to the command line and Python…

Using Comments

Let's go back to the text editor. Let's look at firstfile.py and copyfile.py in the ATOM text editor. There is one thing missing from these files. Comments.

Comments are added to code so that you or another person may come back to the code later on, or see it for the first time, and understand what the code is supposed to do and what the author was thinking when they wrote it.

In Python, you can add a comment by beginning with the **#** symbol. This symbol instructs the computer to ignore these lines when running the code.

Let's go through these two files, **firstfile.py** and **copyfile.py** and add comments that explain what is happening.

<u>Pride and Prejudice - What could we possibly want to know?</u>

- Open **pride** in ATOM.

We have nice evenly spaced lines, which means that the text was reproduced in digital form using line breaks from a specific text, or a specific line length was decided upon when the digital text was formatted. The answer to that question is now lost...

Lines breaks are controlled in digital text using the line feed and the carriage return. But these characters are invisible, like spaces, so how can we be sure what invisible characters are there?

- Press Ctrl+Shift+P to open to the Command Palette.
- Search for: Applications: Open Your Keymap
- Enter the following code at the bottom of the file.

```
	'body':
		'shift-alt-i': 'window:toggle-invisibles'
```

You have now edited the code of the text-editor itself to create a new hotkey option.

- Press Shift+Alt+i to display the invisible characters.

You can see they are quite difficult to make out, relatively faint… Let’s make another change to the functionality of the text-editor to make them more visible.

- Press Ctrl+Shift+P to open to the Command Palette.
- Search for: Application: Open Your Stylesheet
- Enter the following code at the bottom of the file.

```
.editor {
  .invisible-character {
	color: #c5c8c6;
	opacity: 0.5;
  }
}
```

Now all the invisible characters are very easy to see. They can quickly be toggled on and off using Shift +Alt+i. Very Handy!

So we have a text file, and what we need for our investigations here are a little more structure…

The Solution - Lists, Tuples, and Dictionaries

For these three problems, Python uses three different solutions - Tuples, lists, and dictionaries:

- **Lists** are what they seem - a list of values. Each one of them is numbered, starting from zero - the first one is numbered zero, the second 1, the third 2, etc. You can remove values from the list, and add new values to the end.
- **Tuples** are just like lists, but you can't change their values. The values that you give it first up, are the values that you are stuck with for the rest of the program. Again, each value is numbered starting from zero, for easy reference.
- **Dictionaries** are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing. You can add, remove, and modify the values in dictionaries.

From : http://sthurlow.com/python/lesson06/

Let's count the words…

Create a new file in ATOM and save it to the **pandp** directory.
Save it as **cw.py**

Now enter the following code into that file and be sure to include the comments.

```
text = input("Enter filename:") #asks for user to enter the filename
ptext = open (text)          #opens a file using the name entered by the user
words = []                   #creates an empty list called words

for lines in ptext:          #loop splits up the original text so that each word
    words.extend(lines.split())#is a separate element in a list.

print ("Total length =", len(words), "words")  #displays the total number of words in the
list.

ptext.close()                         #closes the file ptext
```

- Save the file and run the code.

So we have a word count, but we need a little more information.

How can we check that the list of words is accurate?

We can try two things.

First, we can show the first 100 elements in the list by using this line of code.

```
print (words [0:100])
```

Second, we can ask for specific elements from the list.

```
print ("First word:", words[0],"\n", "Second word:", words[1], "\n", "Third word:",
words[2])
```

Replacing the original **print()** statement you wrote and running it again with the new **print()** commands.

Does it make the process more transparent?
What kinds of comments would you add to these lines to explain how they work?

So now we have a small bit of information about Pride and Prejudice. Let's ask another basic literary question…

<u>What is the longest word in the text?</u>

We can start with the same code that turns the text into a list where each word is a element in the list. Instead of entering the same code again, just save the same file under a new name. Since we are looking for the longest word, let's call it **lw.py**

```
text = input("Enter filename:")

ptext = open (text)
words = []
for lines in ptext:
    words.extend(lines.split())
```

Now let's add two new variables that will represent the information that we want to see.

```
longest = ""   #creates an empty variable
len = 0  #creates a variable with the value zero
```

Finally, we can add a loop, This loop will check each item in the list ***words*** and compare it to the previous word. If the next word is longer, then it will be used for comparison, until a longer word is found, and so on, and so on, until the end of the document…

And the print statement at the end will show us both the longest word, and its length.

```
for w in words:
    if len(w) > leng:
        longest=w
        leng = len(w)

print (longest, leng)
```

So we should get this in the command prompt.

```
``Gracechurch-street, 21
```

BUT WAIT!!!!   What are those two accent marks appended to the beginning of our longest word!?

Let’s go back and take a look at our text file.

When we open up **pride** in atom and take a closer look, we can see that our text has not been totally cleaned up and prepared for analysis. The quotes at the beginning of the dialogue are not represented by the same characters as those at the end of the dialogue lines.

We could use find and replace in the ATOM text editor to replace all the `` with “. If we do that and save it as **pride2**, we end up getting a different result from our lw.py script, why is that?

So, we’ve identified a couple of problems…

One, we are only counting the first instance of a longest word. Later words of an identical length are ignored.

Two, we have a large number of punctuation characters included in our list of words.

How can we correct these problems?

Let’s make a small file that will allow us to check the building of the list. We will control the list contents with regular expressions.

```
import re  #tells python to import the regular expressions package for us to use.

text = input("Enter filename:")
ptext = open (text, encoding="utf-8")

words = []

for lines in ptext:
    words.extend(re.findall("[A-z\-\']+", lines.strip()))
#Eliminates numbers but includes hyphenated and contracted words

print (words [0:100])
```

So we see a few things going on in our test list.

- `` are treated as characters just like letters
- ‘ (single quote) are included in the regular expressions we used, and the text uses two single quotes to close dialogue, not one double quote character, so the ‘’ is being treated as a separate word in the list.
- However, the periods, question marks, etc, that were affecting word length have been removed.

So if we go back to the ATOM text editor, open up **pride**, and replace all the `` and ‘’ with a regular double quote character, we should get a much cleaner list of words to explore.

We’ve had enough encoding headaches and cursory explorations...

Let’s take a look at how a certain word is used in Pride and Prejudice. The code below will open the pride file and look for all the lines using the word “dress.”

Enter this code and make sure it runs smoothly.

```
import re

ptext = open (pride, encoding="utf-8")

words=[]  #define a list with nothing in it

for line in ptext:  #read a line from the file
    if re.search(r"\bdress", line):
        print (line)
```

Now how could we change this code to take user input, so we can select the word we want to examine each time, without having to edit the code itself?

In order to build a regular expression using input, we need to build a new variable that represents that regular expression. Then we will use that variable in our loop.

```
import re

text = input("Enter filename:")
ptext = open (text, encoding="utf-8")

words=[]
dress = input("Select a word:")

my_regex = r"\b" + dress

for line in ptext:
    if re.search(my_regex, line):
        print (line)
```

WHEW! Let’s take a breath and ask some questions…

Self-directed Assessment Exercise

Follow the instructions below individually or with a partner.

See if you can use and understand the scripts included in the **lovecraft** directory using the information we’ve discussed so far.

Change directories to **lovecraft** folder.

You’ll see that we have two new texts, **holmes.txt** and **lovecraft.txt**.

We have some scripts that we “found on the web” that we think will do what we want.

Let’s take a look at **hpcount.py**, **countaword.py** and **catwords.py**.

- What do these files do?
- How can we edit them so that we can use them to explore holmes.txt and lovecraft.txt?
- How can we edit them to handle user input instead of editing the script each time we want to ask a new question?
- How can we interpret the regular expressions that are being used?