Lab1.md

Lab 1

Try to complete as many of the problems as you can. Hand in your code in Canvas before midnight tomorrow (24 August). A single file, `FullName_lab1.py` containing the code in the same order as the given problems. You can use File->New File in IDLE to create the file.

If you can't manage to complete a particular problem please hand in your incomplete code -- comment it out if it produces an error.

Chapters 1 in the NLTK book is relevant to this lab. The Python Tutorial or some other Python reference might come in handy.

The objective of the lab is to make sure we have Python and NLTK up and running and then take a look at Regular Expressions in Python and Text Corpora in NLTK.

1. Some useful Python functions

```
#Use dir() to see the names that exist in the current scope.

dir()

#You can use help() to see what dir does.

help(dir)

#Now define three variables, a list, and a dictionary.
#Feel free to change their names and values:

my_str = "This is an ordinary string"
my_int = 5
my_float = 4.6
my_list = ['A','B','C','D']
my_dict = { 'key': 74143, 'str': 'blah' }

#Use dir() again. What has changed?

#Use type() to see the type of each variable.

type(my_str)
...

#Now use dir on the five types you defined.

dir(my_str)
...

#Many of the names are methods that can be applied to the types.
#For example dir(my_str) lists 'upper' so its possible to do the following:

my_str.upper()

#Some of the names have double low dashes on both sides, e.g. __len__
#in the dir(my_str) list. Those are functions that can be applied to the type.

len(my_str)

#You can use help to see what each function does:

help(my_str.upper)
```

```
help(len)

#Use dir and help to select one function to apply to each of the variables
#and the list.
```

TODO: Return the your code for applying the five functions you selected in the last part.

2. Testing out NLTK

Before you can get started with the NLTK corpora you have download it with nltk.download(). You only have to do this once, but you can go back later to add or remove resources.

```
import nltk
```

TODO: Now apply NLTK functions to do the following:

- **Import text6 from nltk**
- **Show the concordance of the word *knight***
- **Find words occuring in similar contexts to *knight***
- **Find the top 25 collocations in text6**

3. Tokenization with regular expressions

The Gutenberg corpus contains text we can work with. Lets select Moby Dick for today's lab.

```
>>> print(nltk.corpus.gutenberg.fileids())
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt',
'bryant-stories.txt', 'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt',
'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.txt', 'shakespeare-hamlet.txt',
'shakespeare-macbeth.txt', 'whitman-leaves.txt']
>>> text = nltk.corpus.gutenberg.raw('melville-moby_dick.txt')
```

NLTK includes a couple of tokenization functions which we will look at in next week.

```
tokens = nltk.wordpunct_tokenize(text)
```

TODO: Use the function `findall` from the Python Regular Expression package and write a regular expression that tries to copy the functionality of the `wordpunkt_tokenize()` function.

```
import re
my_tokens = re.findall(r"###ADD RE###", text)
```

If you match the functionality perfectly you the two token list should be identical.

```
>>> len(tokens) - len(my_tokens)
0
>>> if my_tokens == tokens:
        print("Perfect solution!")

Perfect solution!
```

You can use the following code to find the first mismatch between the lists while you are honing your regular expression skills. It returns the index where the mismatch occurs and corresponding words from both lists. Note that it will through an error when the lists are identical.

```
next((idx, my_item, item) for idx, (my_item, item) #no line break here!
```

```
in enumerate(zip(my_tokens, tokens)) if my_item != item)
```

You could wrap this code up in a helper function that also prints out the surrounding words of the mismatch in both lists (eg. `print(my_tokens[idx-5:idx+5])`)

Is there anything you thought should have been handled differently in the tokenization process?

##4. Matching specific tokens with Regular Expressions

```
m = re.match(r'((\d+)\.(\d+))', '123.4567')
```

The function `re.match()` returns a Match Object.

It is possible to check the truth value of the object:

```
if m:
    print('Found a match!')
```

The method `.group(0)` can be used to print the (last) match.

```
>>> m.group(0)
'123.4567'
```

If you use parenthesis to match more than one groups or subgroups they can be found in `.group(1)` and so on. You can use `.groups()` to see all the matches. It's even possible to name the groups and use the names instead of numbers.

```
>>> m.groups()
('123.4567', '123', '4567')
```

Now lets use `re.match` to examine the words in the list of tokens in part 1.

We can for example use it to try find all mentions of years in the text.

```
year_words = [w for w in tokens if re.match('[12]\d{3}', w)]
```

It's better to compile the regular expression if it's going to be used repeatedly, e.g. in a for loop.

```
year_re = re.compile('[12]\d{3}')
year_words = [w for w in tokens if year_re.match(w)]
```

TODO: Examine the tokens list and try to find a couple of meaningful word groups to write a regular expression for. Then write them.

5. Frequency Distributions

Chapter one introduces FreqDist(). We can use it to create a frequency distribution for the tokens in Moby Dick.

```
fd = nltk.FreqDist(tokens)
```

TODO: find the frequency of *starboard*, find the token that appears most frequently, and show the 15 most common tokens Also find the most frequent token that is not a stopword and how many times it appears.