Project 3

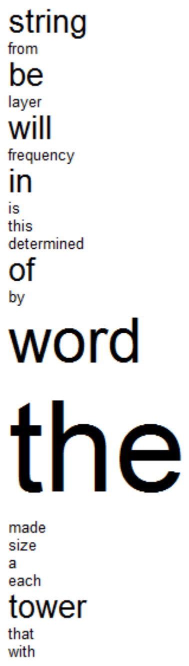
Due Wednesday, October 26 by 4:00pm

Programming is often used to store, analyze and visualize data in new or interesting ways. Take, for example, the "word cloud':

BIG
DATA
INFORMATION
STORAGE
LARGE
PETABYTES
SEARCH
ANALYSIS
RESEARCH
SETS
MILLION
MANAGEMENT
TECHNOLOGIES
SOFTWARE
VOLUME
DEVELOPMENT
TECHNOLOGY
INTERNET
SCIENCE
FUTURE
NEW
BILLION
PROCESS
GOVERNMENT
SAN
NAS
EXABYTES
LEARNING
WORLD
BASED
SIZE
PROCESSING
DATABASES
ANALYTICS
APPLICATIONS
AMOUNT
PARALLEL
INITIATIVE
ECONOMIC
TERABYTES
TIME
USED
NETWORKS
HYPOTHESIS
PAST
SOCIAL
YEARS
HIGH
REAL
TERMS
YEAR
SUCCESSFUL
NEEDED
USES
SHARED
ARCHITECTURE
TIMES
EVERY
PRIVACY
INTERNATIONAL
IMPLEMENTATION
BECOMES
FRAMEWORK
BUSINESS
BETWEEN
MASSIVE
APPROACHES
CRITIQUES
MAPREDUCE

A word cloud is a fancy way of analyzing the contents of a document (or speech or blog post). All the words in the document are jumbled together, with each word being sized according to the number of times the word appears in the document. Words that appear many times are printed larger.

In this project you will make a new technique for word-frequency visualization. Destined to become the newest coolest way to visualize word frequencies, behold the "word tower":



Ok, so here's the project specifications. I want you to make a **single** python file called `Project_3` that can be used as a *module*, with everything wrapped up in functions that can be used later. At the top of your file (below your name) you must import the `string` and `turtle` modules. The functions you need to define are as follows. Don't be too intimidated, I am writing a lot of words here mainly so its clear what I'm asking you to do. Read the instructions carefully and let me know if you have any questions.

1) You must have a function named `save_data_string` which takes as input a **string**, and saves the string to a file called `data_file.txt`. Nothing is returned. Make sure this function works even if the file hasn't been created yet, and if the file does already exist, make sure this function replaces whatever is on the file with the new string. You may assume the user gives a string as intended, and not an integer or anything.

2) You must have a function named `print_data` which simply **prints** the contents of the text file `data_file.txt` as one big string. There is no input (even though it accesses the file), and nothing is returned. You can assume that the text file has been created before calling this function. This function is included so the user can check that the text file has the correct text in it. This function and the previous function should both be pretty short, maybe 4 lines.

3) I'm giving you this one for free. You must include the following function, exactly as it appears here:

```
def remove_punc(in_string):
    exclude = set(string.punctuation)
    out_string = ''.join(ch for ch in in_string if ch not in exclude)
    return out_string.lower()
```

   I should probably tell you what this function does. It takes an input string and returns the same string with all of its punctuation removed and with all letters made lower case (spaces are left alone, however). This function is needed so that punctuation, like an exclamation point, does not count as a "word" in our word tower, and so that 'This' and 'this' are counted as the same "word".

4) This is the hard one. You must make a function named `word_freq_dict` which takes no input, but accesses the text file and returns a *dictionary* with each key being a different "word" in the text and the corresponding value being the number of times that word appears in the text. You must take the text file contents, put those contents into the function `remove_punc` above (to remove punctuation and make every letter lower case), then probably use the .split() method to make a list of "words", and somehow count how many times each word appears in the list, storing that data in the dictionary.

5) This one is longer but I'll give you the first part. You must define a function named `word_tower` which starts with:

```
def word_tower():
    t = turtle.Turtle()
    t.hideturtle()
    t.up()
    t.goto(-200,-200)
    (more code goes here...)
```

   This function shouldn't return anything but it should, starting at coordinates (-200,-200), start writing a word, then move the turtle up, then write another word, then move the turtle up, etc., for all the words in the text (use `word_freq_dict` above to get the dictionary of words). Notice I use `t.hideturtle()` to hide the turtle cursor so it doesn't get in the way of seeing the words.

Two important things. First, to get Python to type the word `word` with size `size` in the turtle window, use the method `write` as in

```
t.write(word, font = ('Ariel', size, 'normal'))
```

Don't change anything but the value of `word` (a string) and `size` (a number) in the line above. You must set the value of `size` to be 10 times the word count of `word`. The second important thing: after writing a word you must move the turtle *upwards* exactly 1.5 times the `size` of the word. Actually, I'm flexible. If you want to get creative here and make a different shape with your words (and maybe play around with colors), I'd be interested to see what you come up with. Just make sure your words don't overlap, and words with a higher frequency are given a bigger size.

Once you've written all those functions (and checked that they work by running them and testing), you don't need to do anything else. To help you double check your work, the commands below show my module in action, with the last command giving the word tower seen on the first page. NOTE: dictionaries do not have a specific order, so if your dictionary or your word tower appears in a different order, that's fine.

```
>>> from Project_3 import *
>>> save_data_string("This is the string that the word tower will be made from.  Each
word in the string will be a layer in the tower, with the size of the word determined
by the frequency of the word.")
>>> word_freq_dict()
{'with': 1, 'that': 1, 'tower': 2, 'each': 1, 'a': 1, 'size': 1, 'made': 1, 'the': 8,
'word': 4, 'by': 1, 'of': 2, 'determined': 1, 'this': 1, 'is': 1, 'in': 2,
'frequency': 1, 'will': 2, 'layer': 1, 'be': 2, 'from': 1, 'string': 2}
>>> word_tower()
```

Lastly, a few rules that apply to all projects:

1) First and foremost, the Python interpreter must be able to execute your script. It is better to have a running but partial solution than a program that attempts to do all computations, but that fails to execute. Handing in an incomplete but working program is better than handing in a program that crashes or does not run at all.

2) The first line of your Python program must be

   `# MCS 260 Project One by Author`

   where you replace `Author` by your name.

2) Add comments to clarify your choice of variables and to indicate what is happening at different steps of the program.

3) Programs will be graded partly on the basis of good hygiene. Programs should be readable with a logical, clear structure and they should avoid unnecessary or overly convoluted calculations. The "PEP 8" style guide for Python helps to improve your grade in this regard.

4) Your .py file must be sent to me via email as an attachment and **in addition** must be printed out and submitted in class (so I can comment on your work easily). My email address is jbergen@uic.edu.