**CSC 4992**

**Python Programming**

**Winter Term 2015**

**Project 01**

**80 points**

**Due 03/24/2013 (11:45 P.M.)**

**The goal of this project is:**

Give you some experience with dictionaries

Practice with file I/O.

Practice with functions.

Using turtle graphics

**Problem 01 (50 points)**

**Background**

Document retrieval is the task of finding documents that meet the search criteria input by a user. The most well-known example is web search, where a user types in a set of key words and the search engine finds web pages that are relevant to their search query. True document retrieval can be quite difficult, as it needs to take into account many different factors. In this project you will implement a very simple document retrieval engine.

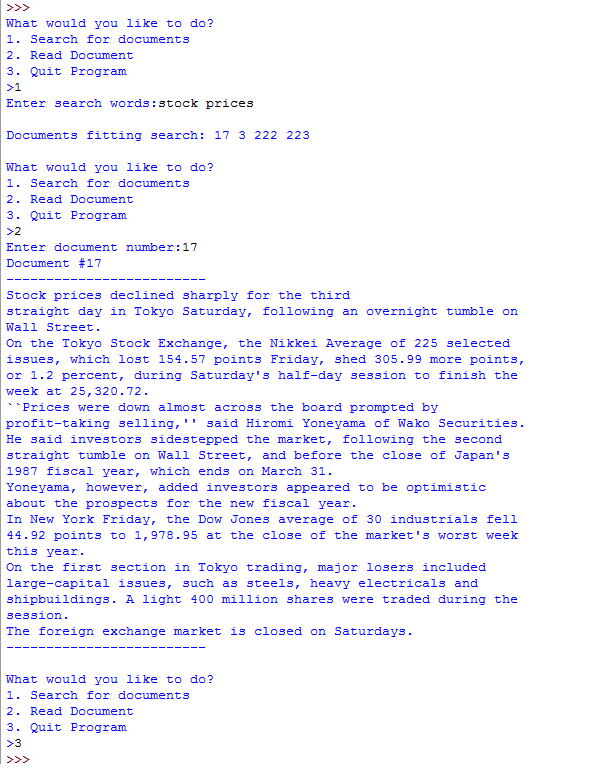
**Project Specifications**

The file “ap\_docs.txt” contains several old newswire articles. We will use this as our document collection; when a user gives us a set of keywords we will find the documents in this collection that match their search terms. Each article in the collection is separated by a line that contains only the “<NEW DOCUMENT>” token.

Your program will read in the documents from the file and number each document starting with 1 (the first document in the file is document 1, the second is document 2, etc.). In order to look up search terms, we will need to know which words appear in each document. We will use a dictionary for this purpose. *Each entry in your dictionary should have a word as the key and the word’s value as the set of documents that this word appears in.* This arrangement allows you to look up a keyword in the dictionary and immediately get all the documents that it appears in, making it easy to figure out documents that might meet a search query.

Once your program has read the file, it will prompt the user to do one of three things: 1) search for documents that match the search words input by the user, 2) display a document, or 3) quit the program. If the user choses to search, your program should prompt for a string of search words and find documents that contain *all* of those keywords. It will then print out the document number of all of the relevant documents. If no documents in the collection contain every keyword input by the user, your program should print a message that says that no relevant documents were found. If the user chooses to display a document, your program should prompt for a document number and print out the entire document that corresponds to that number. Your program should continue to prompt until the user chooses to quit.

**Example:**



**Problem 01 Hints**

1. You will likely need to store two types of data in two different data structures. In one, you will have a dictionary that stores single words as the key with the value as the set of documents (numbers) that the word appears in. In the other data structure, you will store the actual text of the documents, so that you can display it for the user when they ask. You can use a list or a dictionary for the second data structure.
2. Search queries should not be case sensitive, i.e. searching for “Stocks” should give all documents that contain ‘stocks’, ‘STOCKS’, etc.
3. You should remove punctuation from the start and end of a word as well. If the string “stock,” appears in the document, this should be counted as an instance of the word “stock” (without the comma). You might find the string module’s string.punctuation useful.
4. You may find sets to be useful. Sets are unordered collections that cannot contain duplicate values. Like lists, you can go through the values in a set with a for loop. You can add elements to a set with the ‘add’ method. You can also make a set out of an existing list. If you want to find the values that are common between two sets, you can use set intersection:

>>> set1 = set() # initialize to an empty set

>>> set1.add(2)

>>> set1.add(3)

>>> set1

{2, 3}

>>> my\_lst = [3,3,1,3,5]

>>> set2 = set(my\_lst)

>>> set2

{1, 3, 5}

>>> set3 = set1 & set2 # intersection of set1 and set2

>>> set3

{3}

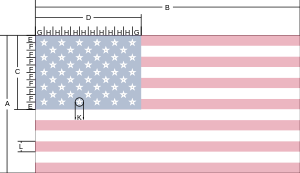
1. Begin small. We have provided a tiny test file for development: “ap\_docs2.txt”. This file has three two-line documents. A sample output using this file is appended below.
2. Develop in pieces.
   1. You may want to modify the ap\_docs2.txt file to make it all lower case and remove punctuation for now—adding them back in for step (j) below.
   2. Read and print the whole file—just to do something to get started.
   3. Read the file, divide the file into documents, and put the documents in a list. I simply made a long string out of each document. The main idea here is to collect lines of a file into a document string until you encounter the token that indicates the start of the next document. At that point, append your document string to your list and start the next document with an empty string.
   4. Make step (b) into a function that returns the list.
   5. Starting with an empty dictionary, go through each document in your list one word at a time. If a word is in your dictionary, add the current document number to the word’s set. If a word isn’t yet in the dictionary, add it to the dictionary with its value as a set with the current document number as its only element.
   6. Make step (d) into a function that returns the dictionary.
   7. Prompt for one search word and use your dictionary to find the documents (numbers) that contain that word.
   8. Next enter two words at the prompt, use the dictionary to find each word’s documents, and then use the set intersection operation to find the common documents. Make this step general enough to handle any number of search words.
   9. Now work on the main part of the program that is a loop that prompts for user input.
   10. Put upper case and punctuation back into the test file ap\_docs2.txt
   11. Thoroughly test on ap\_docs2.txt, and then test on the full ap\_docs.txt file

**Problem 02 (30 points)**

**Background**

This part is conceptually simple: draw an American flag using turtle graphics. The exercise is to use functions. There is a Wikipedia page (<http://en.wikipedia.org/wiki/American_flag>) that has specifications for dimensions that are very useful—all dimensions are derived from a specified height (A).

**Flag Specifications**

[](http://en.wikipedia.org/wiki/File:Flag_of_the_United_States_specification.svg)

The [specification](http://en.wikipedia.org/wiki/File:Flag_of_the_United_States_specification.jpg) gives the following values:

* Hoist (height) of the flag: ***A* = 1.0**
* Fly (width) of the flag: ***B* = 1.9 × A**
* Hoist (height) of the canton ("union"): ***C* =** ***A* × 7/13** (spanning seven stripes)
* Fly (width) of the canton: ***D* = *B* × 2/5** (two-fifths of the flag width)
* ***E* = *F* = *C*/10** (One-tenth of the height of the canton)
* ***G* = *H* = *D*/12** (One twelfth of the width of the canton)
* Diameter of star: ***K* = *L* × 4/5** (four-fifths of the stripe width)
* Width of stripe: ***L* = *A*/13** (One thirteenth of the flag height)

**Project Specification**

You must use at least the four functions specified below. You are encouraged to use more functions of your own design. For example, I found that it was useful to have a function that drew a row of stars.

Important specification: *global variables are not allowed*. That is, any variable names used within any function must be parameters or be created in the function (appear on the lhs of an assignment).

1. A function to draw a rectangle name draw\_rectangle and fills it with the color.  
   draw\_rectangle(length,height,color)
   * Parameters: length (float), height (float), color (string)
   * Returns nothing
2. A function to draw a star named draw\_star and fills it with the color.  
   draw\_star(size,color)
   * Parameters: size (float), color (string)
   * Returns nothing
3. A function to determine rgb colors named get\_color from a parameter color string.  
   get\_color(color)
   * Parameter: color (string)  
     Use color strings such as “red”, “blue”, and “white”.
   * Returns: r, g, b (floats (or ints if you changed the turtle colormode setting))  
     Note that a return can have multiple comma-separated parts so you can use  
     return r, g, b  
     if your function call is of the form  
     r, g, b = get\_color(“red”)  
     Also, you may have more than one return statement in one function.

(Python note: you really aren’t returning three things, but instead it is one collection (a tuple) of three things.)

1. A function draw\_flag to draw a flag of a certain height (“A” from Wikipedia dimensions) according to the official specifications as noted in the Wikipedia article. The exception is in the size and spacing of stars—following the exact specification of stars and their spacing is not worth the effort so simply making stars look good is satisfactory.  
   draw\_flag(height)
   * Parameter: height (float)
   * Returns nothing

**Problem 02 Hints**

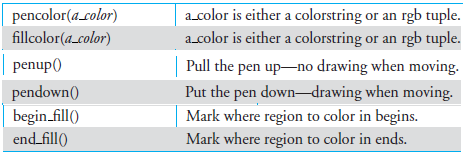
1. Colors. You may use simple colors, e.g. “red” is the rgb (1,0,0), or you can use the official colors as described on the Wikipedia page. You may also be creative with colors as long as there are the correct stripes and stars, and at least three colors.
2. Stars. For full credit you need all 50 stars arranged in 5 rows of 6 and 4 rows of 5.
3. Using turtle.speed(100) is helpful to draw fast, but I suggest that you begin with slow drawing so you can better observe what is happening. Make your Grader happy by including turtle.speed(100) in your final version of your draw\_flag function.

**Example Output:**



**Drawing with Color**

You can specify two different drawing colors: the color of the pen and the color used to fill a region drawn on the screen. The first is set by the pencolor command and the second by the fillcolor command. The below Table lists the commands.



Filling an area is done by bracketing the drawing with the two commands, begin

fill() and end fill(). The area filled is whatever occurs between the two calls.

Example:

import turtle

A = 100

length = A \* 1.9

height = A/13

r,g,b = 1,0,0 #red

turtle.pencolor(r,g,b)

turtle.fillcolor(r,g,b)

turtle.begin\_fill()

turtle.pendown()

turtle.forward(length)

turtle.left(90)

turtle.forward(height)

turtle.left(90)

turtle.forward(length)

turtle.left(90)

turtle.forward(height)

turtle.left(90)

turtle.end\_fill()