CS A109, Python
Assignment 4
Due 7 Mar 2019 by 11:59pm to Blackboard
10 points each (20 points total)

Complete the following programming problems. Be sure to use good style and documentation (worth 10% of the point total; see the ***DocumetationAndStyle.pdf*** file on Blackboard). When complete, please zip your files together and upload to Blackboard.

1. (10 pts) Calculating the Day of the Week

Write a program that takes as input a date (e.g. 25 3 2011 which corresponds to 25 March 2011) and outputs the day of the week that corresponds to that date. There are a lot of parts to this problem but individually each function is relatively short. In the end you will be amazed that it really works! The algorithm is from:

http://en.wikipedia.org/wiki/Calculating_the_day_of_the_week. The implementation will require the following functions:

`is_leap_year(year)`

This function should return True if year is a leap year and False if it is not. Here is pseudocode to determine a leap year:
year is a leap year if ((*year* is divisible by 400) or (*year* is divisible by 4 and *year* is not divisible by 100))

`get_century_value(year)`

This function should take the first two digits of the year (i.e. the century - you can get this using integer division), divide by 4, and save the remainder. Subtract the remainder from 3 and return this value multiplied by 2. For example, the year 2011 becomes: (20/4) = 5 remainder 0. 3 - 0 = 3. Return 3 * 2 = 6.

`get_year_value(year)`

This function computes a value based on the years since the beginning of the century. First, extract the last two digits of the year (use modulus). For example, 11 is extracted for 2011. Divide the resulting value by 4 and discard the remainder. Add the two results together and return this value. For example, from 2011 we extract 11. Then (11/4) = 2 remainder 3. Return 2 + 11 = 13.

`get_month_value(month, year)`

This function should return a value based on the table below and will require invoking the `is_leap_year` function if the month corresponds to January or February:

Month	Return Value
January	0 (6 if year is a leap year)
February	3 (2 if year is a leap year)
March	3
April	6
May	1
June	4
July	6
August	2
September	5
October	0
November	3
December	5

Finally, to compute the day of the week, compute the sum of the date's day plus the values returned by `get_month_value`, `get_year_value`, and `get_century_value`. Divide the sum by 7 and compute the remainder. A remainder of 0 corresponds to Sunday, 1 corresponds to Monday, etc. up to 6 which corresponds to Saturday. For example, the date 25 March 2011 should be computed as (day of month) + (get_month_value) + (get_year_value) + (get_century_value) = 25 + 3 + 13 + 6 = 47. 47/7 = 6 remainder 5. The fifth day of the week corresponds to Friday. Your program should allow the user to enter any date and output the corresponding day of the week.

If you have access to a linux/unix/Mac terminal, you can check the correctness of your program easily by using the cal command. For instance, Alaska officially became a state on 3 Jan 1959, which was a Saturday:

```
> cal 1 1959
    January 1959
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Example program output:

```
Enter the day (as a number): 3
Enter the month (as a number): 1
Enter the year (as a 4-digit number): 1959
That day of the week is a Saturday
```

2. (10 pts) Jumble Solver

Write a program to solve a word jumble puzzle like you would find in the newspaper. Your program will prompt the user for a set of scrambled letters and search the dictionary file, words.txt, for a word with the same set of letters. (No need to submit this file with your solution.) To read the contents of the file, modify the following code (you may also want to read zyBook *Section 12.1*, *Reading files*):

```
f = open('C:\\Users\\XXX\\words.txt', 'r')
contents = f.read()
f.close()
```

Note that in Pyzo, you will probably need to specify the full pathname for the file (or cd to the correct directory in the interpreter). When executing the script from a directory, if you don't specify a path, it will look for the file in the same directory as the .py script.

There are many solutions to this problem. A naïve solution would generate every permutation of the set of input letters and then search the word list each time for a match. A more efficient solution recognizes that if you sort all the letters of every word, then you just have to search the list of words once for the sorted letters in the jumbled letters. This search can be performed very quickly using a dictionary.

Your solution should implement and use the following function to build such a dictionary from a given list of words:

```
def build_dict(word_list):
   '''Build a dictionary from the given word_list where key values of
sorted word letters map to lists of the original word(s)'''
```

For instance, here is a snapshot of the dictionary:

```
{'deeginrs': ['designer', 'resigned'], 'deeegins': ['designee'], 'ddeegins':
['designed'], 'degiinrs': ['desiring', 'residing', 'ringside']}
```

Then, write another function that takes the word jumble and dictionary as arguments and returns the list of original word(s) that match. If no match exists, return the string 'No word found':

```
def unscramble(jumble, sorted_dict):
  '''Return a list of unscrambled words that match the given jumble'''
```

Allow the user to enter a jumble, and then also hardcode the 3 jumbles below to confirm you produce the same output. Example output from running the program:

```
Please enter a jumble to solve: sdf
sdf unscrambles to: No word found
asewes unscrambles to: ['seesaw']
enost unscrambles to: ['notes', 'onset', 'steno', 'stone', 'tones']
aaabcs unscrambles to: ['casaba']
```