1. Count the lines in a file in which specific words occur

==========================================================

This is the implementation for the first white-board problem!

The task:

Part A:

-------

Write a function called "word\_count", which takes a filename and a list

of words as arguments. It then opens the file, reads it line by line and

checks if the specified words occur in that line. If so, the counter for

the appearing word is increased. At the end, it returns a dictionary,

which contains in how many lines each word occurred.

Example:

Input file is: "foo bar\nbaz blah baz baz baz baz\nbar baz\nbaz".

Filename in this example is "my\_example.txt"

You call:

word\_count("my\_example.txt", ["bar", xyz", "baz"])

This should return:

{"bar" : 2, "xyz" : 0, "baz" : 3}

Please document your code, especially anything that you find interesting

or which you believe the reader might need some explanation on.

Part B:

-------

Write a unit test for the word\_count() function.

Using the pyunit framework, write a test-case for the above function.

The test case should include functions to test the expected behavior for:

\* Edge case: File can't be found (should raise exception)

\* Edge case: File is empty (counts stay zero)

\* Edge case: None of the search words occur (counts stay zero)

\* Some normal inputs producing regular results

Where and how do you create the test files for this test? How do you

ensure those test files are cleaned up after the test run?

2. Remove common lines from a set of files

==========================================

Context:

1. You are given a directory full of text files (the 'input files').

2. These files may have certain lines in common (for example, assume

these files are email dumps and many of those emails have disclaimers

or footers or headers).

3. We want to create copies of those files, which have 'common' lines

removed, since those common elements don't add much value for us.

Challenge:

- We may want to use 'identical lines' as basis of comparison for now,

but in the future, we may split up the file differently (maybe by

sentence, or paragraph, or some other unit). The part that cuts up

the input files into units should be flexible. We want this to be

implemented via a base class and specific child-classes.

The task:

Part A:

-------

1. Write a base class, which implements at least (!) the following

functions:

\* 'find\_common'

\* 'strip\_common'

\* 'split\_file'

Common usage for this class would be:

\* Create a class instance.

\* Call the find\_common() function of the object.

\* Then call the strip\_common() function of the object.

2. When the class is instantiated, the following parameters should

be provided:

\* An 'input directory path' (the name of an existing directory).

\* A 'threshold' (the threshold is a float number between 0 and 1).

\* A 'destination directory path' (the name of an existing directory

into which filtered output should be written).

3. The 'find\_common' function:

\* Should read all the files in the source directory.

\* For each file, it should use the 'split\_file' function to return

the smaller 'units' that make up the file (for example lines).

\* It should find all units, which occur in a higher percentage

of input files than indicated via the threshold parameter. For

example, if the threshold value is 0.6 then any unit occurring in

more than 60% of the source files would be detected. Any unit which

occurs that often is deemed 'common'.

\* The function should store a list of all common units in the class

instance.

4. The 'strip\_common' function:

\* This function should write a copy of each of the input files

into the destination file (using the same file name), but those

copies should not contain any units, which were deemed to be

common (as stored in the class).

5. The 'split\_file' function:

\* In the base class, this function should just raise the

'NotImplementedError' exception (which is provided by Python).

Part B:

-------

Write a child-class of the previously written base class, which

implements the 'split\_file' function, simply by treating each line as a

unit (it returns the list of lines).

Part C:

-------

Using the pyunit framework, write unit tests for this class. Think about

where we create the test output directory and how we clean up after the

test. Where does the test input come from?

Notes:

\* Please document your code, especially anything that you find

interesting or which you believe the reader might need some

explanation on.

\* Don't worry too much about performance and optimization for now,

just focus on readable, well documented code.

\* If you do have ideas around performance optimization, feel free

to discuss them in the code comments.