CSC 120
Lab 2 problems

Problem 1 (files)

In lecture we learned about `open()`, which returns a file object. The method `readlines()` returns a list of the contents of a file, where each element of the list is a single line of the file. This method is invoked on a file object as follows:

```
f = open("sample.txt")
lines = f.readlines()
```

a) Write a function `compare(file1.txt, file2.txt)` that takes two string arguments, `file1.txt` and `file2.txt` and compares the lines of the files in sequence, that is, it compares the first line of `file1.txt` again the first line of `file2.txt` and so on. If the lines are the same, do nothing. If the lines are different, print a message saying the lines are different. Assume the files have the same length.

Solution:

This solution uses `readlines()` to get a list of the lines in a file. Once we have the two lists, we can compare the strings of contained in each, one by one.

```
def compare(file1.txt, file2.txt):
    f1 = open(file1.txt).readlines()
    f2 = open(file2.txt).readlines()
    #Assumes the files have the same length.
    #If not, you should check for the smallest file
    # and use its length to control the iteration.
    for i in range(len(f1)):
        if (f1[i] != f2[i]):
            print("The lines are different.")
```

b) How would you change your code if the files were of different lengths?

Solution:

Compare the lengths of `f1` and `f2`. Use the size of the smaller one to control the loop. It should also report that the files are not of the same length (and so are not equivalent).

Problem 2 (dict)

In the English language, some combinations of adjacent letters are more common than others. For example, 'h' often follows 't' (as in 'th'), but rarely does 'x' follow 't' (as in 'tx'). Knowing how often a given letter follows other letters is useful in many contexts. (See the **Note** below!)

For this problem, you will write a function called `pair_frequencies(word_list)` where `word_list` is a list of strings representing valid English words. Your function will examine `word_list` and print all 2-character sequences of letters along with a count of how many times each pairing occurs. (The function does not return anything.) For example, suppose `word_list` has the following contents:

```
["banana", "bends", "i", "mend", "sandy"]
```

This list of words contains the following 2-character pairs: "ba", "an", "na", "an", "na" from banana; "be", "en", "nd", "ds" from bends; "me", "en", "nd" from mend; and "sa", "an", "nd", "dy" from sandy. (Note that "i" is only one character long, so it contains no pairs.) The call of `pair_frequencies(words_list)` would print the following output:

```
ba : 1
an : 3
na : 2
be : 1
en : 2
nd : 3
ds : 1
me : 1
sa : 1
dy : 1
```

Notice that any pairings that occur more than once in the same word should be counted as separate occurrences. For example, "an" and "na" each occur twice in "banana".

Solution:

```
def pair_frequencies(word_list):
    freq_dict = {}
    for word in word_list:
        if len(word) < 2:   #skip words of length 1
            continue
        for i in range(len(word)-1):
            pair = word[i:i+2]
            print(pair)
            if not pair in freq_dict:
                freq_dict[pair] = 0
            freq_dict[pair] += 1
            print(freq_dict)
    for pair in freq_dict:
        print(pair, ":", freq_dict[pair])
```

Note: In cryptography, we use this data to crack substitution ciphers (codes where each letter has been replaced by a different letter, for example: A -> M, B->T, etc.) by identifying which possible decoding substitutions produce plausible letter combinations and which produce nonsense.

Problem 3 (dict)

Write a Python function `merge_dicts(d1, d2)` that takes as arguments two dictionaries, `d1` and `d2`, and returns a new dictionary consisting of the two dictionaries merged together.

The two argument dictionaries consist of key/value pairs that map strings to integers. For example, consider the two dictionaries below:

```
d1 = {'t':4, 'x': 2, 'z' : 5}

d2 = {'b':3, 'x': 1, 't' : 2, 'c': 9}
```

If a key exists in both dictionaries, then the function sums the corresponding values from the two dictionaries.

The call `merge_dicts(d1, d2)` should return the dictionary

```
{'t': 6, 'x': 3, 'z': 5, 'b': 3, 'c': 9}
```

Solution:

```
def merge_dict(dict1, dict2):
    new_dict = {}

    for key1 in dict1.keys():
        if key1 in dict2:
            new_dict[key1] = dict1[key1] + dict2[key1]
        else:
            new_dict[key1] = dict1[key1]

    for key2 in dict2.keys():
        if key2 not in new_dict:
            new_dict[key2] = dict2[key2]


    return new_dict
```