CS2311-Algorithms and Data Structures with Python
Homework 2 Programming
Due Date: See Blackboard

Submission guidelines

- First design, develop and test your code in a Jupyter notebook
- Then copy your final code and markdown cells into the Jupyter Notebook file (.ipynb) provided for the assignment and submit to Blackboard
- Note that the Jupyter notebook for final submission may contain testing code to help you check that your output and expected match. Follow the instructions in the notebook for copying your code and running the testing code.
- If asked, also provide any supporting files or images requested in the assignment

Grading Criteria:

- Good documentation/comments and program readability using both markdown cells and code comments
- Algorithm/pseudo-code is explained in a markdown cell and is efficiently written
- Program runs correctly for test cases with no syntax errors or logical errors

The instructor should be able to reproduce your work from your notebook.

1. [15 points] File compare function

Write a function called **file_compare(myfile, correctfile)** that compares two text files line by line (you can assume that the files have the same number of lines). If the lines are different, the function prints out a **list** of the characters in the line for the first file and the corresponding line in the second file so that the user can inspect the difference. Make sure your comparison shows the "blank" characters i.e. space, \n, \t that print out as blanks on the screen if you use print(). You can then use this function to compare the output of your homework assignments to the outputs provided in the homework assignments. (Hint: you may want to use file.readlines())

Test your function against the two files provided called **"myfile.txt"** and **"correctfile.txt"**.

Your output should look exactly as follows:

```
Line>    1  - same
Line>    2  in myfile is different

        CORRECT>>
['20           7.00        38,696.84\n']
        MYFILE>>
['30           7.00        76,122.55\n']
```

```
Line>    3  - same
Line>    4  in myfile is different

        CORRECT>>
['Question 1.B: Present Value Calculation:\n']
        MYFILE>>
['Question 1.B: Present Value Calculation: \n']

Line>    5  - same
Line>    6  - same
Line>    7  - same
Line>    8  - same
Line>    9  - same
```

2. [20 points] Counting Duplicate Words

Write a script that uses a dictionary to determine and print the number of ***duplicate*** words in sorted order in a sentence. Put your script into a function called **p1_word_counts(text)** that takes a string of words. Test your function. Treat uppercase and lowercase letters the same and assume there is no punctuation in the sentence. Words with counts larger than 1 have duplicates. Print the words out in sorted order.

Use the following sentence to test your code:

```
text = ('this is sample Text with several words '
        'this is more sample text with some different Words')
```

Your output for this sentence should be exactly as follows (WORD is in a field width of 11):

```
WORD        COUNT
is          2
sample      2
text        2
this        2
with        2
words       2
```

3. [20 points] Duplicate Word Removal

Write a script and put it into a function called **p2_unique_words(text*)*** that receives a string of words and then determines and displays in alphabetical order only the unique words. Remove any periods in the sentence and treat uppercase and lowercase letters the same. **The function should use a set to get the unique words** in the string. Test your function.

Use the following string to test your code:

text = ('This. is. sample. text. with. several. words. This is more sample text with some different Words')

Hints: You may want to use text.split(), text.replace(), word.lower(), sorted()

Your output for this string should be exactly as follows:

```
UNIQUE SORTED WORDS:
different
is
more
sample
several
some
text
this
with
words
```

4. [20 points] Dictionary Manipulations

Write a script that performs the tasks below and put it into a function **p3_dictionary_manipulations(d,params)** that takes the dictionary **d** shown, and a list of parameters, **params = ['Canada', 'France','sw','se'].** The dictionary maps country names to Internet top-level domains (TLDs). The list of parameters is used to perform tasks a., b., d., and e..

```
d = {'Canada': 'ca', 'United States': 'us', 'Mexico': 'mx'}
```

The function should perform the tasks shown:

a. Check whether the dictionary contains the key `'Canada'`.
b. Check whether the dictionary contains the key `'France'`.
c. Iterate through the key–value pairs and display them in two-column format.
d. Add the key–value pair `'Sweden'` and `'sw'` (which is incorrect).
e. Update the value for the key `'Sweden'` to `'se'`.
f. Use a dictionary comprehension to reverse the keys and values.
g. With the result of part (f), use a dictionary comprehension to convert the country names to all uppercase letters.

Your output for this input should be exactly as follows (there is a blank line after the answer to g.):

```
Dictionary is:
{'Canada': 'ca', 'United States': 'us', 'Mexico': 'mx'}

a. Canada in dict:
True

b. France in dict:
False

c. Keys and values in dict (key in field width 20):
Canada               ca
United States        us
Mexico               mx

d. Dict with Sweden set to 'sw':
{'Canada': 'ca', 'United States': 'us', 'Mexico': 'mx', 'Sweden': 's
w'}

e. Dict with Sweden corrected to 'se':
{'Canada': 'ca', 'United States': 'us', 'Mexico': 'mx', 'Sweden': 's
e'}

f. Reverse mapping using dictionary comprehension:
{'ca': 'Canada', 'us': 'United States', 'mx': 'Mexico', 'se': 'Swede
n'}

g. Reverse mapping uppercase using dictionary comprehension:
{'ca': 'CANADA', 'us': 'UNITED STATES', 'mx': 'MEXICO', 'se': 'SWEDE
N'}
```