CSC 110 Assignment 8:
Dictionaries, Lists, Tuples, File IO!

Learning Outcomes:

When you have completed this assignment, you should understand:

- How to create a dictionary by reading the contents of an input file
- How to write functions that operate on dictionaries, lists, and tuples

How to hand it in:

Submit your `assignment8.py` file through the Assignment 8 link on the CSC110 conneX page.

Grading:

- Late submissions will be given a **zero grade**.
- You must use the **`assignment8.py`** file provided to write your solution. Changing the filename or any of the code given in the file will result in a **zero grade**.
- Your function names must match exactly as specified in this document or you will be given a **zero grade**.
- Function arguments must be exactly as specified in this document. Specifically, do not change the number of and/or order of the arguments or you will be given a zero grade for the function.
- We will do **spot-check grading** in this course. That is, all assignments are graded BUT only a subset of your code might be graded. You will not know which portions of the code will be graded, so all of your code must be complete and adhere to specifications to receive marks.
- Your code must run without errors on the ECS 258 Lab machines or a **zero grade** will be given.
- It is recommended that you use a plain text editor such as Notepad++ as used in the labs or Atom for Mac computers. We also recommend you run your programs through terminal / command prompt, as shown in the labs.
- It is the responsibility of the student to submit any and all correct files. Only submitted files will be marked. Submitting an incorrect file is not grounds for a regrade.
- If the assignment requires the submission of multiple files, then all files must be submitted.

Marks will be given for…

- your code producing the correct output
- the **tests** for your functions **providing sufficient coverage** (at least 2 tests and enough to cover all possible paths through the code)
- your code following good coding conventions (see lab and lecture code for examples)
    - Proper indentation
    - Documentation of signature and purpose using the format we have followed in lectures and previous assignments.
    - Names of variables should have meaning relevant to what they are storing
    - Use of whitespace to improve readability
    - Proper use of variables to store intermediate computation results

Get started…

- Download `assignment8.py` from the conneX ***Files*** tab and save it to your working directory.
- Write your name and student V# at the top of the file
- For each of the 9 function specifications provided below you must:
    - Uncomment the **calls** to each functions test function.
    - You are free to comment out these test calls in your main function as you progress through the assignment, but you MUST leave all of your tests in place for us to grade
    - Add any tests to the test function that you feel are necessary. The functions have tests provided for you, but you may want to add tests to adequately test each function.
    - **Complete the function definition** according to the specification provided.

Setting the stage (word frequency dictionary):

Text analysis is done in many real-world applications including advertising (tons!), spam filtering, risk management, and sentiment analysis. Text is read and analysed, and the result of the analysis can be used to inform massive business, legal, and technical decisions.

For this assignment, you will be creating a word-frequency dictionary where the key-value pairs include a word and an associated integer representing the number of times the word has been seen in an input file. For example, consider we have an input file containing the following tongue-twister:

how much wood would a woodchuck chuck if a woodchuck could chuck wood
a woodchuck would chuck as much wood as a woodchuck could chuck if a woodchuck could chuck wood

The resulting word-frequency dictionary would be defined as the following:

```
word_freq = {'how': 1, 'much': 2, 'wood': 4, 'would': 2, 'a': 5,
'woodchuck': 5, 'chuck': 5, 'if': 2, 'could': 3, 'as': 2}
```

The first element has the key "how" and associated value 1, because the word "how" is found once in the input file. On the other hand, "woodchuck" occurred 5 times so its associated value is 5.

Application of our word frequency dictionary

There are a number of small input files to text your functions with. The final analysis will be on the lyrics of the top 15 songs from this week's American Top 40. You will compare the word frequencies of the current lyrics against the lyrics found in the top 15 songs of this week 10 and 20 years ago. The songs (shown below) can be found in the **top40.txt** file found in Assignment 8 files section on Connex.

2019.txt (current):

Someone You Loved - Lewis Capaldi
Truth Hurts - LIZZO
Señorita - Shawn Mendes Feat. Camila Cabello
Only Human - Jonas Brothers
Circles - Post Malone
Goodbyes - Post Malone
Beautiful People - Ed Sheeran (feat. Khalid)
I Don't Care - Ed Sheeran & Justin Bieber
Trampoline - SHAED
bad guy - Billie Eilish
How Do You Sleep? - Sam Smith
Good As Hell - LIZZO
Time - NF
Motivation - Normani
Talk - Khalid

2009.txt (10 years ago):

Fireflies - Owl City
Whatcha Say - Jason DeRulo
Down - Jay Sean (feat Lil Wayne)
Party in the U.S.A - Miley Cyrus
Run This Town –Jay-Z, Rihanna & Kanye
Paparazzi - Lady Gaga
Meet Me Halfway - the Black Eyed Peas
3 - Britney Spears
I Gotta Feeling - the Black Eyed Peas
Sweet Dreams - Beyonce
You Belong With Me - Taylor Swift
Replay - Iyaz
Empire State Of Mind - Jay-Z & Alicia Keys
Forever - Drake (Kanye, Lil Wayne, Eminem)
Use Somebody - Kings of Leon

1999.txt (20 years ago):

Smooth - Santana (feat Rob Thomas)
Satisfy You - Puff Daddy (featuring R. Kelly)
Heartbreaker - Mariah Carey (feat Jay-Z)
Mambo No. 5 (A little bit of ...) - Lou Bega
Unpretty - TLC
I Need To Know - Marc Anthony
My Love Is Your Love - Whitney Houston
Back At One - Brian McKnight
We Can't Be Friends - Deborah Cox & R.L.
Steal My Sunshine - Len
Someday - Sugar Ray
Scar Tissue - Red Hot Chili Peppers
Music of My Heart - 'N Sync & Gloria Estefan
(You Drive Me) Crazy - Britney Spears
Genie In A Bottle - Christina Aguilera

Part 1: Dictionary operations we will need for the analysis

Exercise 1 – Obtaining a value associated with a given key from a dictionary

Complete the function design for the `get_value()` function, which takes a {str:int} dictionary and a string representing a key as a parameter, and returns the integer value associated with the given key. If the key does not exist in the dictionary, 0 is returned.

Exercise 2 – Determining the sum of all values in a dictionary

Complete the function design for the `sum_values()` function, which takes a {str:int} dictionary as a parameter, and returns the sum of all values found in the dictionary.

Exercise 3 – Combination!

Complete the function design for the `frequency_percentage()` function, which takes a {str:int} dictionary and a string representing a key as parameters. The function should return the result of dividing the value associated with the given key by the sum all values found in the dictionary. This allows us to see the frequency a word occurred in an input file out of all of the words found.

Part 2: Creating the dictionary

Exercise 4 – Creating a word frequency dictionary

Complete the function design for the `frequency_dictionary()` function, which takes a string as a parameter representing the name of the file to read. A word frequency dictionary will be created with the form {str:int} where the keys are each of the unique words found in the input file, and the associated integer values represent the number of occurrences the word was found in the input file. If the file cannot be read, an empty dictionary should be returned.

HINT: Look at the first tests that work with really small files. Think about what the dictionary should look like after reading through the file and seeing the first word. How should it be updated when it reads a second unique word. What about when it reads a word that it has seen before?

Part 3: Sorting a list of tuples

Exercise 5 – Converting a dictionary to a list of tuples

Complete the function design for the `dict_to_list()` function, which takes a {str:int} dictionary as a parameter and creates and returns a new list of tuples where each tuple is the key-value pair from the dictionary. Each tuple should have the form (str, int).

Exercise 6 – Swapping elements in a list

Complete the function design for the `swap()` function, which takes a list of tuples as a parameter, and two integers representing indexes values. The function should modify the given list by swapping the elements in the list at the two given indexes.

Exercise 7 – Finding the index of the maximum value searching a specific range in the list

Complete the function design for the `find_max_index_from()` function, which takes a list of tuples as a parameter, and an integer representing an index value. The function should return the index of the largest value found in the list, beginning the search at the given index (and searching only elements from that index onward).

Exercise 8 – Implementing the Selection Sort algorithm

Complete the function design for the `selection_sort()` function, which takes a list of tuples as a parameter, and sorts the elements in the list from largest to smallest value. Note that tuples have the form (str, int), and the list should be sorted so that the tuple with the LARGEST integer value is at the front of the list (index 0). The list must be sorted using the Selection Sort algorithm:

1. Find the index of the largest value in the unsorted section of the list
2. Swap the largest element with the element in the first slot of the unsorted section
3. Continue performing steps 1 and 2 until there is no more unsorted section.

Note: You should call the functions designed earlier in Part 3 inside `selection_sort()`

Part 4: Writing the results to a file

Exercise 9 – Writing values from a list into a file

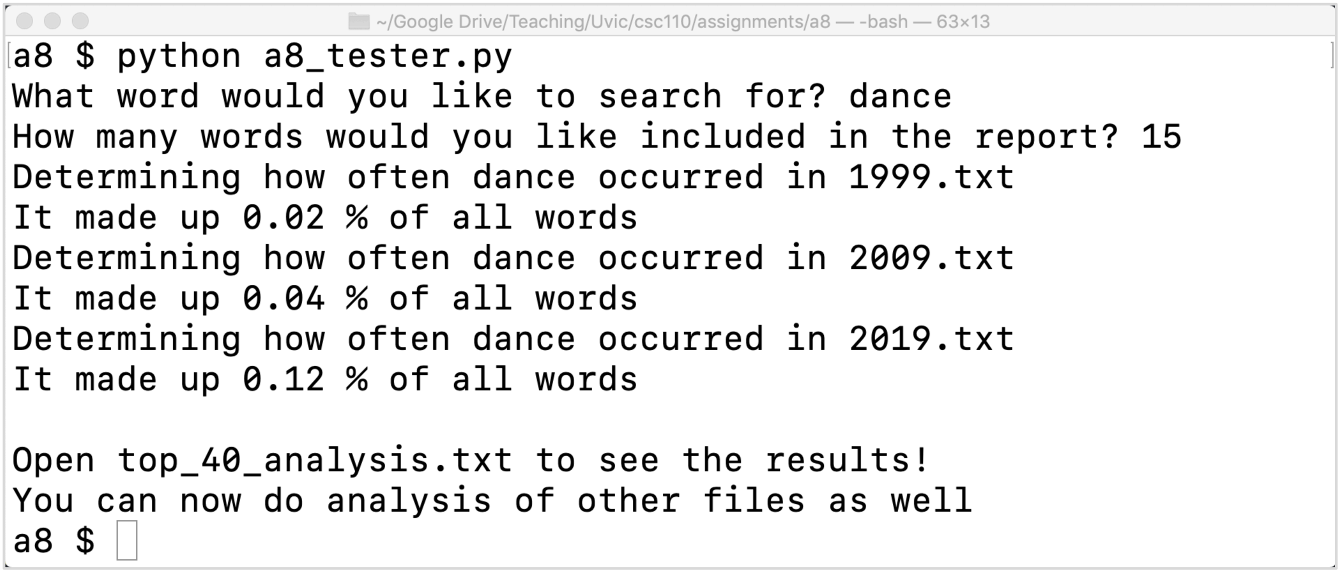
Complete the function design for the `write_most_frequent()` function, which takes 4 parameters, a string, a list of tuples, an integer, and another string:

- The first string represents the name of the file to write to (with the **append** usage mode)
- The list of tuples contains the information to write. Assume the list has already been sorted.
- The integer represents the number of elements to read from the list and write to the file
- The title should be written to the file before writing data from the list.

Exercise 10 – Using the a8_tester.py

Similar to Assignment 7, the **a8_tester.py** can be run after you have completed all of the above functions. It runs through the text files containing Top 40 lyrics and allows you to enter what name to search for, and then writes to a file called **top_40_analysis.txt** the most frequent words.

On the Connex Discussion Board, post your results when you search for other words. You could also use your functions to determine word frequency in other text files. Sample output:


~/Google Drive/Teaching/Uvic/csc110/assignments/a8 — -bash — 63×13

```
a8 $ python a8_tester.py
What word would you like to search for? dance
How many words would you like included in the report? 15
Determining how often dance occurred in 1999.txt
It made up 0.02 % of all words
Determining how often dance occurred in 2009.txt
It made up 0.04 % of all words
Determining how often dance occurred in 2019.txt
It made up 0.12 % of all words

Open top_40_analysis.txt to see the results!
You can now do analysis of other files as well
a8 $ 
```

Inside **top_40_analysis.txt** (there are 15 words because that is the number I specified above):

```
Results for 1999.txt:
you: 209
I: 203
me: 202
the: 171
to: 157
my: 110
of: 93
your: 92
love: 92
a: 75
and: 74
And: 70
in: 67
it: 53
oh: 53
```

```
Results for 2009.txt:
I: 222
the: 208
you: 196
my: 124
a: 123
me: 121
to: 107
down: 84
in: 74
and: 72
like: 72
that: 70
I'm: 69
it: 65
be: 62

Results for 2019.txt:
I: 222
you: 188
the: 118
to: 95
I'm: 92
me: 73
it: 69
my: 69
we: 63
in: 53
that: 53
don't: 49
la: 44
a: 41
your: 41
```