ECE 498 - Python Homework 4 Fall 2020

Goals: More functions, control structures, strings and lists, programming practice.

Do the following in a single Google Colabs file and share your results with me. Keep all your sections in the order below. Write your function prototypes exactly as given so I can easily add code to test them. Do 5 problems for C grade, 6 for B grade and all 7 for A grade.

1) Write a Python function with prototype "`def find1dup(lst):`" that will find the first item in the list that is duplicated. The function will return a tuple containing the index of the first item having a duplicate and the index of the first item after that that it duplicates. Return the tuple (-1,-1) if there are no duplicates. Example `find1dup([1,2,3,4,5,6,3,4,3,5,5])` will return `(2, 6)`. Use "`enumerate()`" in your outer "`for`" loop.

2) Write a Python function with prototype "`def finddups(lst):`" that will find all the items in a list which have a duplicate and will return these items in a list. The returned list will have only one instance of each item, regardless of how many times it appeared. E.g., `finddups([1,2,3,4,5,6,3,4,5,6])` will return `[3,4,5,6]`. Use "`enumerate()`" in your "`for`" loop.

3) Write a Python function with prototype "`def to2d(lst, numcols, pad = None):`" that will turn the list into a two dimensional one with the given number of columns. If the number of elements is not a multiple of `numcols`, then pad it with the "`pad`" value.
E.g., `to2d([1,2,3,4,5,6,7],3)` will return `[[1,2,3], [4,5,6], [7,None,None]]`.

4) Write a Python function with prototype "`def flatten(lst):`" that will "flatten a list of lists." Don't recurse into any sub-lists. E.g., `flatten([[1,2],[3,4,5],[6,[7,8]]])` will return the list [1, 2, 3, 4, 5, 6, [7, 8]].

5) Write a Python function with prototype
"`def getwords(filename = 'mostcommon3000.txt'):`" that will open the given file (which is assumed to contain a list of words, one word per line) and will return a list of lists of the words in lower case. The function will first determine the length of the longest word. The first sub-list of the returned list will be the words of length 0, the second will be words of length 1, the next those of length 2, etc. For example, if the file only contained the words: `'a', 'ab', 'abc', 'bc'` and `'bcde'`, then [[], ['a'], ['ab', 'bc'], ['abc'], ['bcde']] will be returned.

6) Write a Python function with prototype "`def anagramdictionary(wordlist):`" that will return an "anagram dictionary" of the given `wordlist` (output from the previous function). An anagram dictionary has each word with the letters sorted alphabetically creating a "key". E.g., `'data'` sorts to `'aadt'`. These are grouped by the number of letters and each group is sorted according to the sorted-letter version of the word.

So, the function return value will be a list of sub-lists of tuples where each tuple contains the sorted-letter version followed by the word. E.g., `('aadt', 'data')`. As with "`wordlist`", the first sub-list of the return value contains tuples for words of length 0, the next sub-list, tuples for words of length 1, the third, tuples for words of length 2, etc. Note that each of these sub-lists is sorted.

E.g., for the “`mostcommon3000.txt`” file, the slice “`anagramdictionary(words)[5][:4]`” is `[('aadeh','ahead'), ('aadls','salad'), ('aadmr','drama'), ('aadpt','adapt')]`. Note that these are words of length 5, sorted by the keys rather than the words they correspond to.

FYI, with fairly straight-forward code, my code for this was seven fairly short lines.

7) Write a Python function with prototype “`def dictdups(adict):`” that will take the anagram dictionary created in the previous problem and will return lists of pairs of words (as a tuple) that are anagrams of one another. (As before one list for each word length). Note that these will be adjacent to each other in the anagram dictionary. For simplicity you only need to report instances of where one sorted word is the same as the next in the list (although occasionally three-in-row will match). A snippet of output is

```
('host', 'shot'), ('shut', 'thus'), ('sink', 'skin')
```

.

Notes

Use “`enumerate`” when you loop through values and need the index as well as the item. Enumerate iterates though things and returns a tuple containing the index and the value

```
    for i, item in enumerate(list):
        print('Item ', i, ' is ', item)
```

This is the Pythonic way, as opposed to the C/C++ way:

```
    for i in range(len(list)):
        print('Item ', i, ' is ', list[i])
```