ECE 498 - Python Homework 5 Fall 2020

Goals: importing, list comprehensions, dictionaries, more programming practice.

For this homework, put all your functions in one file and all your test code in a second file. Both of these should be saved as a ".py" files (rather than ".ipynb" files). In your test code, use of each of the following forms of the "import" statement (at least one of each):

```
import hw05
from hw05 import somefct
from hw05 import fct1, fct2
from hw05 import somefct as somename
```

Note that all of these "imports" can coexist in the same file. Your test code should make use of each of the versions at some point. Submit the two ".py" files for grading. Note that some points may be deducted if your code could obviously be written in a "more Pythonic" way. Your test code should also have all tests in the body of this "if" statement:

```
if __name__ == "__main__":
```

1) Write Python functions with the following prototypes (and described below). **All but the last should have a body that is one line long:**

```
def everythird(listr):
def replacews(string, replc = "#"):
def tupletwo(alist):
def wordlengths_list(alist):
def wordlengths_str(string):
def longerwords_list(alist, length = 3):
def longerwords_str(string, length = 3):
def longerwords_listr(listr, length = 3):
```

a) function "everythird()" will take a list (or string) and will return a list (or string) consisting of every third element beginning with the second element (index 1). For example "everythird([1,2,3,4,5,6,7,8,9])" will return "[2,5,8]" and "everythird('abcdefghi')" will return the string "beh"

b) function "replacews()" will replace whitespace in a string with the replc character/string. Use split and join. For example: replacews("Has eighteen letters does") will return "Has#eighteen#letters#does" and replacews("Has eighteen letters does","**") will return "Has**eighteen**letters**does"

c) function "tupletwo()" will take a list of tuples and will return a list containing the second item of each tuple. E.g., "tupletwo([(1,2), (3,4), (5,6), (7,8)])" will return "[2, 4, 6, 8]".

d) function "wordlengths_list()" will take a list of words and will return a list containing the length of each corresponding word. E.g., given ["cat", "banana", "quark", "nimbostratus"] it returns [3, 6, 5, 12]

e) function "`wordlengths_str()`" will take a string and will return a list containing the length of each word (in order). E.g., given "`cat banana quark nimbostratus`" it returns `[3, 6, 5, 12]`.

f) function "`longerwords_list()`" will take a list of words and will return a list of the words that are longer than the given length (in the same order). E.g., `longerwords_list(['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog'], 3)` will return `['quick', 'brown', 'jumps', 'over', 'lazy']`.

g) function "`longerwords_str()`" will take a string of words separated by white-space and will return the string but keeping only words longer than the given length.
E.g., `longerwords_str("The quick brown fox jumps over the lazy dog", 3)` will return the string "`quick brown jumps over lazy`"

h) function "`longerwords_listr()`" will combine the functionality of the previous two functions. If a list of word it passed it will return what "`longerwords_list()`" does, but if a string is passed it will return what "`longerwords_str()`" does. Granted, you can do this in one line, but it looks cleaner if you use a few lines. (So, it is OK to use more than one line for this one.)

2) Write a function with prototype "`def reverse_lookup(d, val):`" that will return a <u>list</u> of keys in dictionary "`d`" corresponding to a value equal to "`val`". The function body should be just <u>one line</u>.
E.g., if `d = {'a':2, 'b':4, 'c':6, 'd':8, 'e':2, 'f':6}`
then `reverse_lookup(d, 6)` will return `['c', 'f']`. Looking up `42` returns "`[]`"

3) Write a function with prototype "`def combinedict(d1, d2):`" that will combine two dictionaries (without changing either original) and will return the result. For any key that is the same in the two dictionaries, assume that the associated values are comparable with the "`>`" operator and use the higher one for the value in the resulting dictionary. For example given:

```
d1 = {'a':5, 'b':"cow", 'c':42, 'd':(1,2,3)}
d2 = {'b':"horse", 'c':7, 'd':(2,0),'e':0}
```

`combinedict(d1,d2)` returns `{'a': 5, 'b': 'horse', 'c': 42, 'd': (2, 0), 'e': 0}`

4) Write a function with prototype "`def subtractdict(d1, d2):`" that will return a dictionary that is the same as d1, but omitting items having a key found in d2's keys. E.g.,

```
d1 = {'a':5, 'b':"cow", 'c':42, 'd':(1,2,3)}
d2 = {'b':"horse", 'c':7, 'd':(2,0),'e':0}
```

then `subtractdict(d1, d2)` returns `{'a': 5}`

5) Write a function with prototype "`def subtractdict2(d1, d2):`" that will remove the items from `d1` whose keys are also keys in `d2`. **This is similar to the above, but `d1` changes and the function returns `None`.**

6) Write a function with prototype "`def wordfreq(filename = "DOIwords.txt"):`" that will read a given file that contains words separated by spaces (perhaps multiple words on a line) and will create a dictionary whose keys are the words and the value is the number of times the word appears. Convert each word to lower case before processing.

7) Write a function with prototype "`def sortedbyvals(d):`" that, for a dictionary having values that can be compared to one another, with return a list of key/value tuples sorted by the values. The function body can be written in one line. For example, if `d = {'a':2, 'b':4, 'c':6, 'd':8, 'e':2, 'f':6}` then the function will return `[('a', 2), ('e', 2), ('b', 4), ('c', 6), ('f', 6), ('d', 8)]`

8) Write a function with prototype "`def mostcommon(d, n)"):`" that takes a dictionary and an integer, where the dictionary represents a histogram (where the values are how many times the key has appeared). The function returns a list of the "`n`" most common keys (i.e., the "`n`" keys having the highest values). You can do this as another one-liner (without calling the function in the previous problem). Test using the dictionary produced by the previous problem. For words in `DOIwords.txt` with `n = 5`, the result should be `['of', 'the', 'to', 'and', 'for'].`

9) Write a function with prototype "`def uniquevalue(d):`" that takes a dictionary and returns a list of the keys that have unique values (no other key has that value). Multiple lines is fine here.
For example, with `d = {'a':2, 'b':4, 'c':6, 'd':8, 'e':2, 'f':6}` the function will return `['b', 'd'].`