ECE 498 - Python Homework 3 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.

1) Write a Python function with prototype "`def print_type_id(listname)`:" that, for each item in the list, will print the item, its "type" and its "id". Your print should separate the three items in a line with two "tab" characters: "`\t\t`". Your main code should create the following list and pass it to the function. Note there are a few types we haven't covered yet.

```
mylist = [True, 42, 3.14159, 3+4j, 'Hi everybody!', ('a', 'b'), [7, 11], {"key
0":'banana', "key 1":'orange'}, lambda:None]
```

2) Write a Python function with prototype "`def triangletype(a,b,c):`" that will return the type of a triangle as a string, given the lengths of the three sides (a, b and c). The function will return "equilateral" (if all sides equal), "isosceles" (if two sides equal), "scalene" (if all sides different). If it is also a right triangle (Pythagorean Theorem holds), then append " right" to the string. (Technically an equilateral triangle is also isosceles, but only report "equilateral" if that is the case).

3) Write a Python function with prototype "`def season(month, day)`:" that will return the season as a string given a month and day as integers (January = 1, month starts on 1). Assume season changes occur on the 21st of the month (that is the first day of the next season). Seasons are in the list

```
      seasons = ['winter', 'spring', 'summer', 'fall']
```

4) Write a Python function "`printMtable()`" that **computes** a multiplication table and prints it. Output should look exactly as follows (including spaces):

```
     1  2  3  4  5  6  7  8  9
     -------------------------
 1 | 1  2  3  4  5  6  7  8  9
 2 | 2  4  6  8 10 12 14 16 18
 3 | 3  6  9 12 15 18 21 24 27
 4 | 4  8 12 16 20 24 28 32 36
 5 | 5 10 15 20 25 30 35 40 45
 6 | 6 12 18 24 30 36 42 48 54
 7 | 7 14 21 28 35 42 49 56 63
 8 | 8 16 24 32 40 48 56 64 72
 9 | 9 18 27 36 45 54 63 72 81
```

For the following, you might consult the notes given below:

5) Write a function with prototype "`def checkwords(filename)`" that will open the given file, then print

> a) words that are palindromes (same backwards as forwards)
> b) words that contain the letter 'q'
> c) words that contain the letter 'q' but not 'qu'
> d) words whose last letter is 'w'
> e) words of at least three letters whose third letter is 'w'

Print all words that satisfy a), then all that satisfy b), etc. Ignore the case of the letters (convert everything to lower case.

Download the file "mostcommon3000.txt" from the web page. I've also shared it with you on Google Drive. Your main code should call "`checkwords()`" with this file. I've shared a brief Google Colabs document with you (Colab-import.ipynb) that shows how to import files and open files on your Google Drive.

6) Write a function with prototype "`def wordlengths(filename)`" that will open the given file, then print

> a) the number of times the word "the" appears (ignoring case – convert to lower case)
> b) create a "histogram" of word lengths (how many have each length), as follows:
>> create a list of 20 counters, initially all zero and for each word, increment the counter whose index corresponds to the length of the word.
>
> For example, if the length of the word is 4, increment counter 4. Print the list of counters when finished

Download the file "DOIwords.txt" from the web page. I've also shared it with you on Google Drive. Your main code should call "`wordslengths()`" with this file. I've shared a brief Google Colabs document with you (Colab-import.ipynb) that shows how to import files and open files on your Google Drive.

Notes

You can open a file and read the contents (presumably ASCII text) using:

```
    with open(filename, 'r') as f:
        filelines = f.readlines()
```

This will open the file in "read" mode and read the lines into a list called "`filelines`". The items of "`filelines`" are each one line of the file as a string. Each item includes the trailing newline character (represented as '`\n`').

"`str.rstrip()`" will return a string with trailing white space characters removed (without changing the original string. Given a string argument, it will strip any of the characters in the string argument from the right side of the string, stopping upon encountering the first character that is not in string argument. e.g., "`str.rstrip("*#! \n3`") strips any combination of the given characters.

"`str.split()`" will split a string on white space, returning a list of the strings between the whitespace. For example:

```
     "  abc def \n ghi \n".split()     # returns ['abc', 'def', 'ghi']
```

If you give "`split()`" an argument (as a string), it will split on occurrences of that string

```
   "abcdA Bcd 1\n3cdcd".split("cd")  # returns ['ab', 'A B', ' 1\n3', '', '']
```

Did you see the "NULL" strings at the end of the list?

"`join()`" is the opposite of "`split()`", as it joins items in a list of strings into a single string. The syntax may look strange to you:

```
     " ".join(['abc', 'def', 'ghi'])     # returns  'abc def ghi'
     "###".join(['abc', 'def', 'ghi'])   # returns  'abc###def###ghi'
     "".join(['abc', 'def', 'ghi'])      # returns  'abcdefghi'
```

The functions "`lower()`" and "`upper()`" will return the string in lower and upper case, respectively (the string is not changed). For example

```
      "Hello".lower()       # returns  "hello"
      "Hello".upper()       # returns  "HELLO"
```

The following will reverse a string:

```
      "Hello"[::-1]        # returns  "olleH"
```

You can tell if an item is in a list or string using the "`in`" keyword:

```
      "ghi" in ['abc', 'def', 'ghi']     # returns  True
      "ghi" in ['abc', 'def', 'ghijk']   # returns  False
      "ghi" in 'abcdefghijk'             # returns  True
```