CPE 101
Project 3- Word Puzzle

Like Project 2, you have more freedom in this project. Your code must pass all of the tests to get full credit. You submit the program in two steps:

1) Purpose statement, signature, and pseudocode for each function (**Due: 2/12/2021 @10PM) for 30%**
1) 3 py files (**Due: 2/19/2021 @10PM) for 70%**

Objectives:

- Practice on lists and strings.
- Opportunity to test your understanding of all of the different concepts:
  - Conditional logic, loops,
  - lists, and functions

Required Header:

All students are required to have the following header comment at the top of their source files. Note that the stuff to the right of the colon in red is information for an imaginary example student. (Please put YOUR appropriate information. Also, your header comment is not expected to have red text.) All program files require this header.

```
# Project 4 – Word Puzzle
# Name: Members Name
# Instructor: Dr. S. Einakian
# Section:
```

Resources:

- Your instructor, via office hours, Canvas Inbox, Slack, etc.
- Free Tutoring Center
- Your ISA

Ground Rules:

- You may not collaborate on this project with anyone except your instructor, ISA, and tutoring center.
- Your program must not use any global data (that is, all data must be declared within some function).
- Your program must implement multiple functions to solve the problem.
- Your program must not use recursion.
- Your program must mimic the sample output behavior ***exactly*** and in all cases.

Program Description

In this project, you will implement a program, which locates words in common word search puzzles (all puzzles are 10 by10). A sample puzzle is shown below:

```
WAQHGTTWZE
CNBSZQQELS
AZEKXWIIML
LDWWLXXSAV
PONDTMVUXN
```

OEDSDYQ**P**OB
LGQCKGM**M**IT
YC**SLO**AC**A**ZM
XVDMGSX**C**YZ
UUI**UNIX**FNU

The above puzzle contains the words (shown in bold): UNIX, CALPOLY, SLO, NEW, and CAMPUS. Words can appear in the puzzle running **up, down, forward, backward, and diagonal (left diagonal - rows and columns are equal)**.

Input and Output

Input:

- Your program should read in a **100 character long string** from user via standard keyboard input.
- These strings are available (at the first line) in the input files called: puzzleAndWords0, puzzleAndWords1, puzzleAndWords2
- Using this user input, you should create a 10 by10 puzzle which you can work with, in order to find the words in the puzzle.
- Your program at the same time also read the **words** from user via standard keyboard input.
- These words are available (at the second line) in the input files called: puzzleAndWords0, puzzleAndWords1, puzzleAndWords2).
- These are the words that are to be found in the puzzle by your program.
- You also have expected outputs for each input that help to compare your output.
- Your program access to the files similar to patterns' lab (Lab 04)

Files:

Your program must create 3 files:

1) wordFinderFuncs.py (function definitions)
2) wordFindertTests.py (2 unittest tests for each non I/O functions)
3) wordFinder.py (main function)

Output:

- Your program should be tested as below using terminal (similar to patterns in Lab04):

  python wordFinder.py < puzzleAndWords0 > output0
  - You may review the following files to know what the "expected output" of your program should look like:

  output0, output1, output2

- Remember that the requirement for your project is to print the output on screen, not to a file.
- Output files mentioned above are ONLY provided to you for comparison purposes. So, you can compare your own output (**prints to the screen**) with the "expected output".
- This comparison will help you verify whether or not your program is working as expected.
- You may access the **input** as well as **output** files (that are mentioned in steps above) via Canvas.

Sample output of program is shown as below:

```
Puzzle:

WAQHGTTWEE
CNMIVQQELS
AZEKXWIIIL
LDWWLXXSPV
PONDTMVAMN
OEDSOYQGOB
LGQCKGMMCT
YCSLOACUZM
XVDMGSXCYZ
UUIUNIXFNU

UNIX: (FORWARD) row: 9 column: 3
CALPOLY: (DOWN) row: 1 column: 0
GCC: word not found
SLO: (FORWARD) row: 7 column: 2
COMPILE: (UP) row: 6 column: 8
VIM: (BACKWARD) row: 1 column: 4
TEST: word not found
NEW: (DIAGONAL) row: 1 column: 1
```

Submission

Part 1: (30% of grade)

- Purpose statement, signature, and pseudocode for each function before implementation
  - Due: 2/12@10PM

Part 2: (70% of the grade)

- **Use the same name for your files**
- You must have total of three files in your submission:
  1. **wordFinderFuncs.py →** This file must include the actual functions definitions/implementation.
  2. **wordFinderTests.py →** This file must include at least two unit tests for each function, which contain the assert statements testing the functions developed in funcs.py.
  3. **wordFinder.py →** This file must include the main function, which calls all the functions developed within funcs.py.

Due: 2/19@10PM

NOTE:

Some of the suggested functions are given in a file wordFinderFuncs.py. You can create as many extra functions as you need. You can even ignore the suggested functions. But, you are not allowed to write the whole project in One Function. You need at least 4 functions.