Project 4

Due: Monday, August 16th at 11:59 p.m. PDT

Please read the **entire document** before writing any code.

Contents

1 Introduction

Please read the **entire document** before writing any code.

1.1 Objectives

For this project, you will write a pogram that deals with manipulating strings. This assignment also tests your understanding of conditional logic, loops, lists, and functions.

1.2 Collaboration

You **may not collaborate** in any way on your project. See the syllabus for more details.

2 Problem Description

For this program, you will be writing a solver for 10 × 10 word search puzzles. (If you are unfamiliar with the idea of a word search, you may read more about them here: Word Search.) For our puzzles, words can appear running **up**, **down**, **forward**, **backward**, or **diagonally down/right**. You *do not* need to check other diagonal directions (i.e., up/right, up/left, down/left).

Here is a sample puzzle:

```
WAQHGTTWZE
CBNSZQQELS
APXWKWIIML
LDELFXXSAV
PONDTMVUXN
OEDSDYQPOB
LGQCKGMMIT
YCSLOACAZM
XVDMGSXCYZ
UUIUNIXFNU
```

The above puzzle contains the words (shown in **red bold**): `UNIX`, `CALPOLY`, `CPE`, `GCC`, `SLO`, and `CAMPUS`.

2.1 Input

Your program will begin by reading in the puzzle and words for which to search. You **should not** be prompting for this input; you should just assume that it will be there.

The description for each puzzle will be input in the following format:

```
<100 character long string representing 10x10 puzzle>
<list of words>
```

The first line will be the puzzle grid. It will be given to you as one long string of length 100. The first 10 characters will represent the first row, the second 10 characters will represent the second row, et cetera.

The second line will be a list of the words for which you will be searching, separated by spaces.

For example, a sample puzzle could look like:

```
WAQHGTTWEECBMIVQQELSAPXWKWIIILLDELFXPIPVPONDTMVAMNOEDSOYQGOBLGQCKGMMCTYCSLOACUZMXVDMGSXCYZUUIUNIXFNU
UNIX CALPOLY GCC SLO CPE COMPILE VIM TEST
```

2.2 Running your program

Remember that your program **should not** be prompting the user for input.

Your program should be run as:

```
python3 word_finder.py < test_files/puzzle0.txt
```

Where `puzzle0.txt` contains the puzzle data as specified above.

2.3 Output

As for the output, your program will print what the puzzle looks like, followed by all of the words for which you were searching, in the order they were given to you, along with the direction you found it in and the row and column numbers, indexed starting at 0, where that word occurs. If the word is not in the puzzle, you must state this. For the sample puzzle above, the output would look like:

```
Puzzle:

WAQHGTTWEE
CBMIVQQELS
APXWKWIIIL
LDELFXPIPV
PONDTMVAMN
OEDSOYQGOB
LGQCKGMMCT
YCSLOACUZM
XVDMGSXCYZ
UUIUNIXFNU

UNIX: (FORWARD) row: 9 column: 3
CALPOLY: (DOWN) row: 1 column: 0
GCC: (DIAGONAL) row: 6 column: 5
SLO: (FORWARD) row: 7 column: 2
CPE: (DIAGONAL) row: 1 column: 0
COMPILE: (UP) row: 6 column: 8
VIM: (BACKWARD) row: 1 column: 4
TEST: word not found
```

3 Specification

You must have a total of three files in your submission:

- `finder_funcs.py`: This file contains your function definitions/implementation.
- `funcs_tests.py`: This file contains unit tests for the functions in `finder_funcs.py`. You must have 100% code coverage.
- `word_finder.py`: This file contains (and calls) the `main` function, which calls the functions that you developed in `finder_funcs.py`.

Note that I'm not specifying a design for your program. You must decide what functions you will need and then implement and test them. Part of your grade will be based on how well you decompose the problem into functions.

3.1 Suggestions

- I store the puzzle as a list of strings. Each string represents one row of the puzzle.
- I store the words as a list of strings. Each string represents one word for which to search.
- I use the existing Python string functions `find`, `split`, and `join`.

- Remember that a function can return multiple values using a tuple.
- You can break up a string or list using slicing.

4 Grading

Your program grade will be based on:

- thoroughness of your `funcs_tests.py`,
- adherence to the specification,
- the number of `diff` test cases passed,
- your program design, and
- program style.

Your program will be tested with test cases that you have not seen. Be sure to test your program thoroughly!!! To pass a test case, your output must match mine EXACTLY. Use `diff` to make sure your output matches mine in the test cases given and in test cases that you make up. If there are any differences whatsoever, you will not pass the test case.

5 GitHub Submission

Push your finished code back to GitHub. Refer to Lab 1, as needed, to remember how to push your local code.