Word Search

CPE 101: Fundamentals of Computer Science
Winter 2019 - Cal Poly, San Luis Obispo

Purpose

To practice string operations and decomposing a problem into functional units.

Description

Download the project files at:

http://users.csc.calpoly.edu/~dkauffma/101/wordsearch.zip

In this project, you will implement a program which locates words hidden in a string. A sample run of the program is shown below.

```
$ python3 wordsearch.py < test1.in

WAQHGTTWEE
CBMIVQQELS
AZXWKWIIIL
LDWLFXPIPV
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
```

Words can appear in the puzzle forward, backward, upward, and downward. You will not need to check diagonals. You will use the `find` function, which returns the index of the beginning of a given word located in a given string (or -1 if the word is not present). For example, `"UUIUNIXFNU".find("UNIX")` returns 3 because the first character of `UNIX` starts at index 3 of the string. However, `"UUIUNIXFNU".find("SLO")` returns -1 since `"SLO"` is not contained in the string.

Dimensionality Conversion

While the given puzzle is a 100-character string (a one-dimensional sequence), the process of finding the rows and columns of words requires operating in two dimensions. To do so, you must convert a given arbitrary index of the string into two values - one for the row and one for the column - which can be done using simple arithmetic operations.

0	1	2	3	4	5	6	7	8
A	B	C	G	I	T	X	Y	Z

One-Dimensional Character Sequence

	0	1	2
0	A	B	C
1	G	I	T
2	X	Y	Z

Two-Dimensional Character Sequence

Given this 9-character string, calling "ABCGITXYZ".find("GIT") evaluates to 3, the index of the G. However, in order to report the row and column of this character, this 3 must be converted to two values: 1 for the row and 0 for the column.

The resulting output of this example would be:

```
GIT: (FORWARD) row: 1 column: 0
```

Implementation

You may **not** use tuples, lists (including the `split` function), or slicing operations for this assignment.

Input

Text files used as input to the program will have the following properties:

- A 100-character string makes up the first line of the file
- The space-separated words to search for make up the second line of the file

Output

Your program must print the following text:

- The given puzzle as a 10x10 grid of characters
- The result of searching for each word, specifying its direction, row, and column if found or a message indicating that it was not found

Function Definitions

You may define additional functions as necessary. Any function that returns a value must have test cases.

```
main()
```

- Request input for a puzzle and words to find (recall that each call to `input` reads one line from a file)
- Use `strip` to remove trailing newline characters
- Display the puzzle, one row per line
- Iterate through the words, calling function(s) to search for each one
- Ensure the order of the words printed matches the test files

```
parse_word(words: str, index: int) -> str
```

Given a string of space-separated words, return a string containing only the word at the specified index (starting from zero). The index represents the word's position in the string, not an individual character index. For example, in the string `"Computer Science"`, the word `"Science"` would be at index 1. Assume that there are enough words in the string to support the given index.

When calling this function in `main`, you may want to use the `count` function to determine how many spaces are in the string:

```
"ABC DEF GHI".count(" ") -> 2
```

```
find_word(puzzle: str, word: str) -> str
```

Search the puzzle for the given word, attempting all directions if necessary. Return a string containing the search result to be printed in `main`. If the word is found, this string will contain the word, the direction it was found, and its two-dimensional row and column in the puzzle; if the word cannot be found, the string will indicate as much. See the given output files for examples.

In the example above, this function would return:

```
"GIT: (FORWARD) row: 1 column: 0"
```

```
reverse_string(string: str) -> str
```

Return the reverse of the input string.

```
transpose_string(string: str, row_len: int) -> str
```

Return a transposition of the input string, assuming `row_len` characters per row. Transposing a two-dimensional grid means converting its rows to columns and its columns to rows. Since strings are one-dimensional, the result will be a string with its characters shifted around.

For example, `"ABCGITXYZ"` transposes to `"AGXBIYCTZ"`.

Testing

Each puzzle can be found in a separate file:

```
test1.in, test2.in, test3.in
```

Your program should be run using:

```
python3 wordsearch.py < test#.in
```

You should compare your output with the corresponding output files using `diff`:

```
test1.out, test2.out, test3.out
```

For all functions that return a value, write at least 3 tests using `assert` statements in `wstests.py`.

Submission

On a CSL server with `wordsearch.py` and `wstests.py` in your current directory:

```
/home/dkauffma/casey.exe 101 wordsearch
```