Homework Turnin

Name:	Alexander A Miller
Email:	alexandermiller@email.arizona.edu
Section:	1E
Course:	CS 120 17au
Assignment:	hw1
Receipt ID:	63c23f1e061eccb625ffe410e809f340

Turnin Successful!

The following file(s) were received:

word-grid.py (2023 bytes)

```
"""
File: word-grid
Author: Alexander Miller
Purpose: this program reads 2 integer values from input, grid size and random number seed,
creates an nxn grid of randomly generated lower-case letters, and prints out the grid of
letters one row at a time
"""
import random

def main():
    n = init()
    g = make_grid(n)
    print_grid(g,n)

def init():
    """
    Description: gathers input; sets random seed; returns n
    Parameters: none
    Returns: returns n, the grid size
    Pre-condition: random must have been imported
    Post-Condition: n will be an integer, seed will be established
    """
    n = int(input())
    s = int(input())
    random.seed(s)
    return n


def make_grid(n):
    """
    Description: makes a grid out of n
    Parameters: n, the grid size
    Returns: g, the grid
    Pre-condition: n must be integer; random must be imported; randomseed created
    Post-Condition: grid will be list of lists, nxn in dimension, consisting of randomly-generated lette
    """
    g = []
    alpha = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w',
    accum2 = 0
    for x in range(n): #need number of entries in g to match n
        g = g + ['%']
    while accum2 != n: #number of sublists
        accum = 0
        while accum != n: #number of entries in each sublist
            a_number = random.randint(0,25)
            g[accum2] = g[accum2] +alpha[a_number]
```

```
            accum = accum + 1
        g[accum2] = g[accum2].strip('%') #now i have to strip that '%' from earlier
        accum2 = accum2 + 1
    return g

def print_grid(g,n):
    """
    Description: prints the grid formed in the last function line by line
    Parameters: g, grid; n, gridsize
    Returns: none
    Pre-condition: grid is indeed a grid; n is an integer
    Post-Condition: will have printed out the grid line by line
    """
    accum = 0
    while accum != n:
        a = ','.join(g[accum])
        print(a +'\n')
        accum = accum + 1


main()
```

word-search.py (8043 bytes)

```
"""
File: word-search
Author: Alexander Miller
Purpose:
*nxn grid, acquired via input file
*list of letters for cross reference, acquired via input file
    *G = grid ; L = list of words
    * word-list file contains one word per line
    * grid-of-letters consists of n lines, each consisting of n letters that are separated by whitespace
*searching for words (in L) that are at least 3 letters long, right to left, left to right, upper left to
* search grid for "legal" words (case-insensitive)
* collect and print words to their own lines without extra whitespace
"""


def main():
    G,L = init()
    A = forward_horizontal_search(G)
    A = forward_vertical_search(G,A)
    A = reverse_order_function(A)
    A = forward_diagonal_search(G,A)
    A = reverse_diagonal_search(G,A)
    A = occurs_in(G,A,L)


def init():
    """
    Description: acquires grid and list of words from input file and compiles them into respective lists
    Parameters: none
    Returns: G (grid), L (list of words)
    Pre-condition: G should be grid of letters, separated by white space
        L should be list of words, 1 word per line
        not all file-types tested: file should be txt to read appropriately (as opposed to rtf)
    Post-Condition: G and L will be lists that can be used for rest of program
    """
    word_list = input()
    letter_grid = input()
    #grid input
    openfile = open(letter_grid,'r')
    G = []
    for line in openfile:
        line = line.strip()# should strip \n
        line = line.lower()
        G = G + [line.split()] #splits characters at whitespace and makes list
    # second bracket makes each line its own sublist
    openfile.close()
    #list input
    openfile = open(word_list,'r')
    L = []
```

```
    for line in openfile:
        line = line.strip() #should strip \n
        line = line.lower() #makes all lowercase (case-insensitive)
        L = L + [line] #no need to split this time
    openfile.close()
    return G,L

def forward_horizontal_search(G):
    """
    Description: this finds all possible words in the forward horizontal direction
    Parameters: G, the grid
    Returns: A, the list of possible words
    Pre-condition: G is indeed a list existing in python (first fn satisfies this)
    Post-Condition: A will be a list containing the forward horizontal word candidates
    """
    accum2 = 0
    A = []
    while accum2 != len(G): #moves us through each row
        accum = 0
        c = ''.join(G[accum2])
        d = [c]
        A = A + d
        while accum != len(G): #moves our starting point
            accum3 = len(G)
            while accum3 >= 1: #moves our endpoint
                e = [c[accum:accum3]]
                A = A + e
                accum3 = accum3 - 1
            accum = accum + 1
        accum2 = accum2 + 1
    return A

def forward_vertical_search(G,A):
    """
    Description: this finds all possible words in the forward vertical direction
    Parameters: G, the grid; A, the list of possible words
    Returns: A, the list of possible words
    Pre-condition: G is indeed a list existing in python (first fn satisfies this)
    Post-Condition: A will be a list containing the forward vertical word candidates
    """
    accum2 = 0
    while accum2 != len(G): #accum2 ties us to a fixed letter position
        accum = 0
        b = []
        while accum != len(G): #accum lets us cycle thru the rows
            a = G[accum][accum2] #A[word][letter]
            b = b + [a]
            accum = accum + 1
        b = ''.join(b)
        A = A + [b]
        accum3 = 0
        while accum3 != len(G): #shifts our start point
            accum4 = len(G)
            while accum4 >= 1: #shifts our endpoint
                A = A + [b[accum3:accum4]]
                accum4 = accum4 - 1
            accum3 = accum3 + 1
        accum2 = accum2 + 1
    return A


def reverse_order_function(A): # reverses "words" from vert/horiz and adds them to collection
    """
    Description: this reverses all the horizontal and vertical word candidates, and adds them to the poss
    Parameters: A, the list of possible words
    Returns: A, the list of possible words
    Pre-condition: G is indeed a list existing in python (first fn satisfies this)
    Post-Condition: A will be a list containing the forward vertical and horizontal word candidates, as v
    """
    accum2 = 0
    original_length = len(A)
    while accum2 != original_length: #repeat for every entry of A
        accum = 0
        a = A[accum2]
        length = len(a)
        c = []
```

```
        reverse_index_relationship = -(accum+1)
        while accum!=length:
            b = a[reverse_index_relationship]
            c = c + [b]
            accum = accum + 1
            reverse_index_relationship = -(accum+1)
        c = ''.join(c)
        A = A + [c]
        accum2 = accum2 + 1
    return A


def forward_diagonal_search(G,A):
    """
    Description: this finds all possible words in the forward diagonal direction (upper left to bottom ri
    Parameters: G, the grid; A, the list of possible words
    Returns: A, the list of possible words
    Pre-condition: G is indeed a list existing in python (first fn satisfies this)
    Post-Condition: A will now also contain the forward diagonal word candidates
    """
    accum2 = 0
    accum = 0
    while accum2 != len(G): #number of diagonals = number of rows
        accum3 = 0 # this must reset every time to bring me to first row for each diagonal
        a = [] #need to reset after using last round
        while accum!= len(G): # as accum grows, the length of the diagonal will shrink
            a = a + [G[accum3][accum]]
            accum = accum + 1
            accum3 = accum3 + 1
        a = ''.join(a)
        accum3 = 0
        while accum3 != len(a): #diagonal gets smaller each round!
            accum4 = len(a)
            while accum4 >= 1: #moving endpoint
                b = a[accum3:accum4]
                A = A + [b]
                accum4 = accum4 - 1
            accum3 = accum3 + 1
        accum2 = accum2 + 1
        accum = accum2
    return A

def reverse_diagonal_search(G,A):
    """
    Description: this finds all possible words in the diagonal direction moving left of center (upper le
    Parameters: G, the grid; A, the list of possible words
    Returns: A, the list of possible words
    Pre-condition: G is indeed a list existing in python (first fn satisfies this)
    Post-Condition: A will now also contain the diagonal word candidates left of center
    """
    accum2 = 0
    accum = 1
    a = []
    while accum2 != len(G):
        accum3 = 0
        a = []
        while accum != len(G):
            b = [G[accum][accum3]]#switches reset variable to position, rather than row
            a = a + b
            accum = accum + 1
            accum3 = accum3 + 1
        a = ''.join(a) #now I have word for entire respective diagonal
        accum3 = 0
        while accum3 != len(a): #diagonal gets smaller each round!
            accum4 = len(a)
            while accum4 >= 1: #moving endpoint
                b = a[accum3:accum4]
                A = A + [b]
                accum4 = accum4 - 1
            accum3 = accum3 + 1
        accum2 = accum2 + 1
        accum = accum2 + 1
    return A


def occurs_in(G,A,L):
```

```
    """
    Description: this will remove word candidates fewer than 3 characters,
        cross-reference word candidates with L, and print them to their own lines
    Parameters: G, the grid; A, the word candidate list; L, the "existing words" list
    Returns: none
    Pre-condition: G, is a grid; A contains all relevant directions (satisfied in previous functions); L
    Post-Condition: the existing words in the grid will be printed
    """
    B = []
    for x in A: #removing "words" fewer than 3
        if len(x) >= 3:
            B = B + [x]
    A = B
    for x in A: #prints words that are in both lists
        if x in L:
            print(x)

main()
```