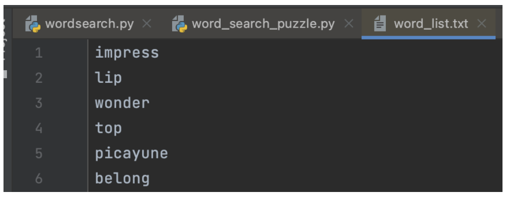
Vicente Maria Feliu
Dr. Hui Keng Lau
Introduction to Programming
Tuesday 25th October 2022

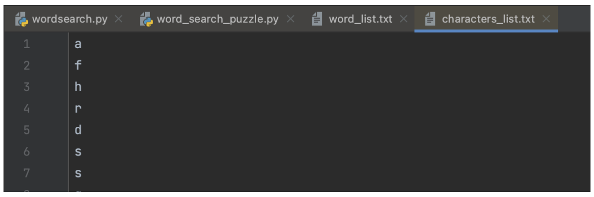
Short Report

This short report aims to generate a valid word search puzzle while also generating a list of all words found in that puzzle. The short report will include snippets of code in order to further demonstrate the solution in a practical way.

Firstly I will create two text files "word_list.txt" including 1000 words and "characters_list.txt" including 100 characters (the files have been submitted in gradescope). I will then begin coding the problem.



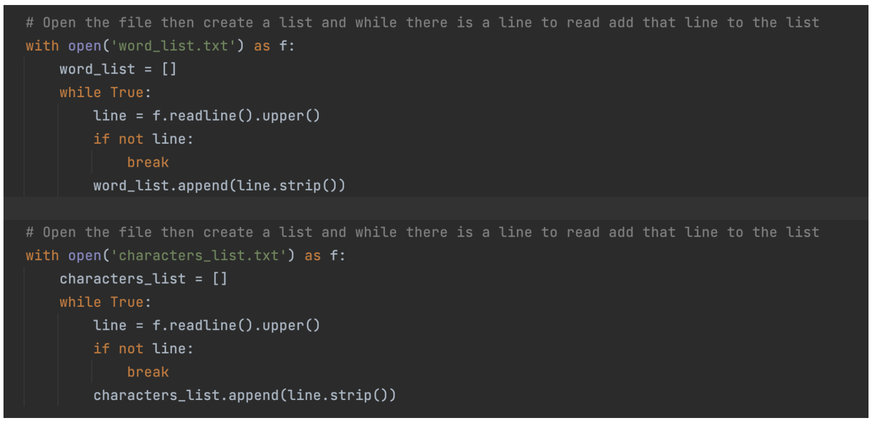
```
impress
lip
wonder
top
picayune
belong
```



```
a
f
h
r
d
s
s
```

(Authors own, 2022)

When coding I first want to open and read these files and pass them to a list, while converting them to upper case. (This can be seen below)

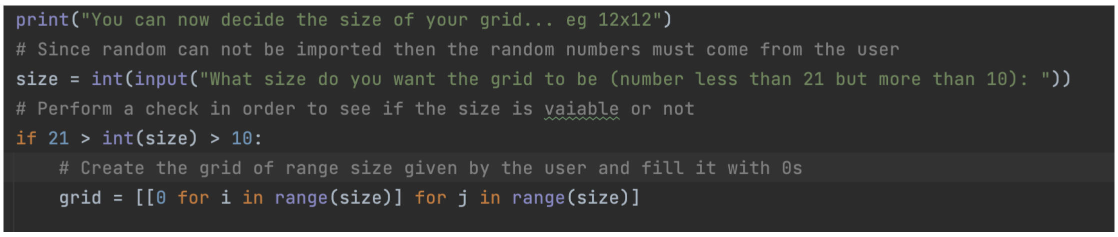


```python
# Open the file then create a list and while there is a line to read add that line to the list
with open('word_list.txt') as f:
    word_list = []
    while True:
        line = f.readline().upper()
        if not line:
            break
        word_list.append(line.strip())

# Open the file then create a list and while there is a line to read add that line to the list
with open('characters_list.txt') as f:
    characters_list = []
    while True:
        line = f.readline().upper()
        if not line:
            break
        characters_list.append(line.strip())
```

(Authors own, 2022)

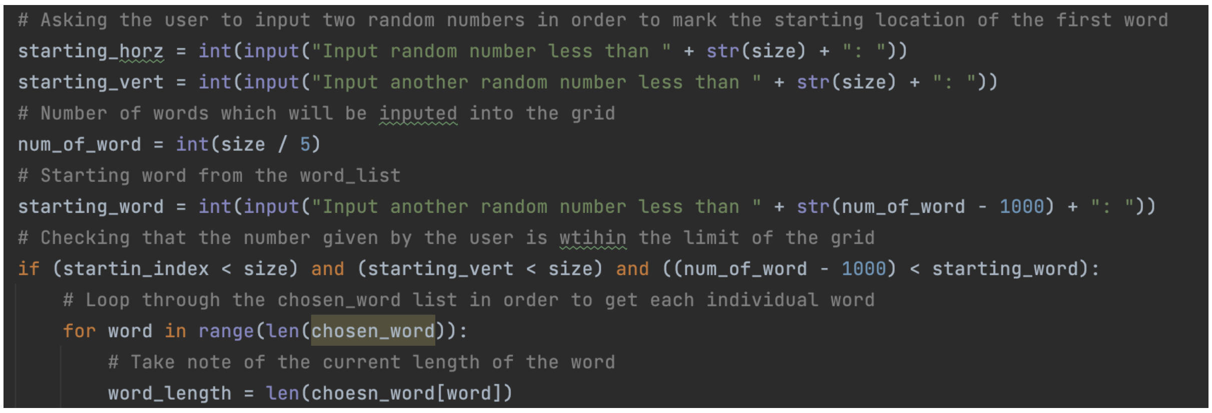
After creating the list for both characters and words we will create the grid/puzzle in random size given by a random number and fill it with 0s. Since no modules can be imported in this coursework then in order to create a random number the user will be asked to input one.



```python
print("You can now decide the size of your grid... eg 12x12")
# Since random can not be imported then the random numbers must come from the user
size = int(input("What size do you want the grid to be (number less than 21 but more than 10): "))
# Perform a check in order to see if the size is vaiable or not
if 21 > int(size) > 10:
    # Create the grid of range size given by the user and fill it with 0s
    grid = [[0 for i in range(size)] for j in range(size)]
```

(Authors own, 2022)

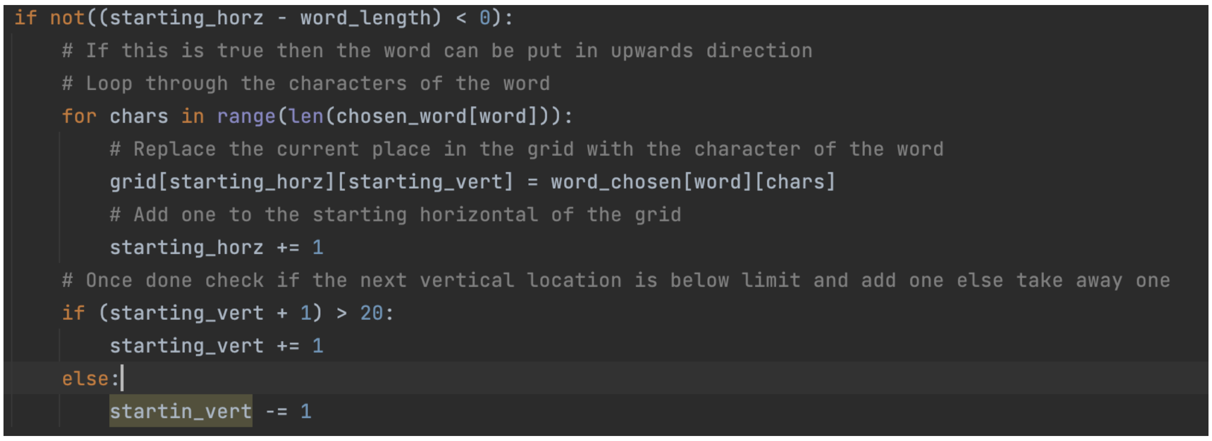
After the grid with all 0s is created with random size, the user will be asked to input two random numbers within the grid's limit in order to create a starting position for the first word. A variable called *'num_of_word'* will be created, which sets the amount of words which will be inputted into the grid. The user will also be asked to input a random number which indicates the starting word from the *'word_list'* which needs to be *'num_word'* minus the amount of words in the *'word_list'*. I will then start looping through the list of words created at the start, and then check the length of that word (shown below).



```
# Asking the user to input two random numbers in order to mark the starting location of the first word
starting_horz = int(input("Input random number less than " + str(size) + ": "))
starting_vert = int(input("Input another random number less than " + str(size) + ": "))
# Number of words which will be inputed into the grid
num_of_word = int(size / 5)
# Starting word from the word_list
starting_word = int(input("Input another random number less than " + str(num_of_word - 1000) + ": "))
# Checking that the number given by the user is wtihin the limit of the grid
if (startin_index < size) and (starting_vert < size) and ((num_of_word - 1000) < starting_word):
    # Loop through the chosen_word list in order to get each individual word
    for word in range(len(chosen_word)):
        # Take note of the current length of the word
        word_length = len(choesn_word[word])
```

(Authors own, 2022)

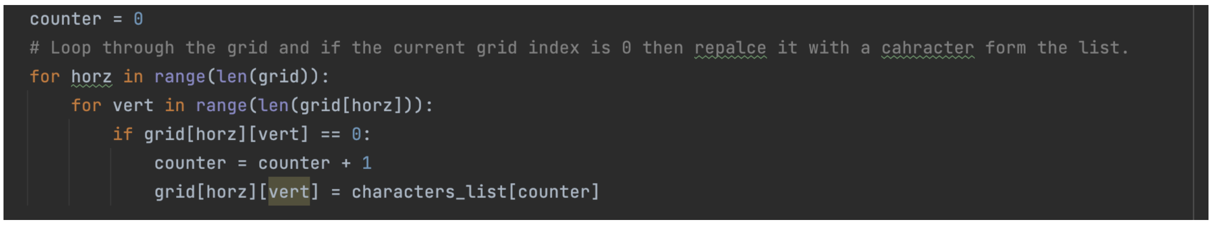
If the location given by the user plus/minus the length of the word in every possible direction is not out of limit, then that direction is viable. Choose which direction to start from and for loop through the single characters of the current word, and go moving the direction chosen while replacing the value of the current location in the grid with the current character of the word and add one to the word length and grid location. Once that word is inputted into the list then change direction by rotating direction 90 degrees and adding one (checking first if that direction is below the limit of course) (eg if moving upwards, when done with inputting word move to the left):



```
if not((starting_horz - word_length) < 0):
    # If this is true then the word can be put in upwards direction
    # Loop through the characters of the word
    for chars in range(len(chosen_word[word])):
        # Replace the current place in the grid with the character of the word
        grid[starting_horz][starting_vert] = word_chosen[word][chars]
        # Add one to the starting horizontal of the grid
        starting_horz += 1
    # Once done check if the next vertical location is below limit and add one else take away one
    if (starting_vert + 1) > 20:
        starting_vert += 1
    else:
        startin_vert -= 1
```

(Authors own, 2022)

After all the words are inputted into the grid, loop through the gird and if the current location of the grid is equal 0 input a character form the character_list using a counter and then add one to that counter.



```
counter = 0
# Loop through the grid and if the current grid index is 0 then repalce it with a cahracter form the list.
for horz in range(len(grid)):
    for vert in range(len(grid[horz])):
        if grid[horz][vert] == 0:
            counter = counter + 1
            grid[horz][vert] = characters_list[counter]
```

(Authors own, 2022)

In summary, this short report has shown that the two problems can be solved in one function. However, it is important to mention that there are little parts of code that are not shown in the snippets due to the restraint of the length of the report.