UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephone: (306) 966-4886, Facimile: (306) 966-4884

Assignment 2

Practicing your Development and Design Skills

Date Due: June 1, 2018, 10pm **Total Marks: 40**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help.
- Programs must be written in Python 3.
- **Assignments must be submitted to Moodle.** There is a link on the course webpage that shows you how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

Question 1 (30 points):

Purpose: To practice the development process for a familiar problem that is complex enough to be interesting. You are to follow a development process, as outlined in the readings and lectures. You are expected to plan the development of this application, starting from the requirements description, given below.

Degree of Difficulty: Moderate

Note:

Before starting Question 1, please read Question 2.

Prominent gangster, Gentleman GoGo, recently made a full confession of all his heinous crimes during a police interrogation. Phoenix Wright was going over the text transcripts of this confession, but found the text transcripts (`confession.txt` on moodle) to be scrambled! With the trial quickly approaching, Phoenix needs to know exactly what crimes this gangster has confessed to. Below is a list of crimes (separated by commas) that this gangster might have confessed to:

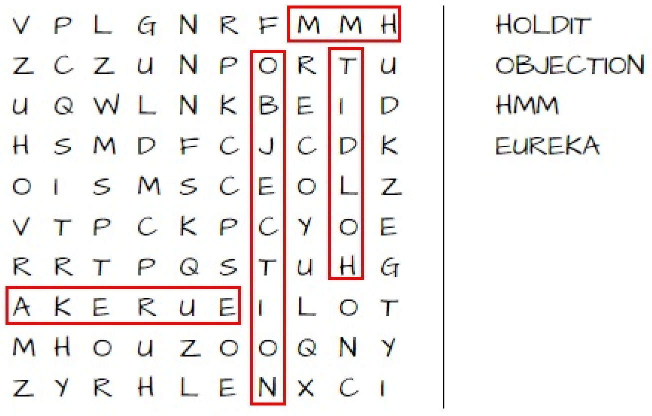
JAYWALKING, BURGLARY, LAUNDERING, BADSINGING, REDRUM, SMOKING, BEINGSMELLY, CONNING, SCAMS, LOITERING

Your job is to write a function to perform a wordsearch, and document which of the above terms were found in the `confession.txt` file (you can include them in your testing program, or show a log if you trying them on the command line).

A *word search* is searching a grid of letters or characters for the presence of a word. We will call this grid of letters a *word square* (`confession.txt` on moodle). If we were looking for the word 'CRIME', we would try to find any directional sequence of letters within the word square to spell this word. Unfortunately, this can get tricky as the word may be present in our word square, but spelled backwards! Or it could be spelled vertically instead! We must search the word square from a multitude of directions or orientations.

- Every *row* must be checked for the presence of the word (going left to right).
- Every reversed *row* must checked for the presence of the word (going right to left).
- Every *column* must checked for the presence of the word (going top to bottom).
- Every reversed *column* must checked for the presence of the word (going bottom to top).

For simplicity's sake, we will NOT be checking diagonals. We will only check in ORTHOGONAL directions (north, south, east, west). Depending on if the word was found using any of the 4 checks above, your program should either output: `Gentleman GoGo IS GUILTY OF <word_to_look_for>` OR `<word_to_look_for> was NOT fou`



```
V P L G N R F M M H
Z C Z U N P O R T U
U Q W L N K B E I D
H S M D F C J C D K
O I S M S C E O L Z
V T P C K P C Y O E
R R T P Q S T U H G
A K E R U E I L O T
M H O U Z O O Q N Y
Z Y R H L E N X C I
```

HOLDIT
OBJECTION
HMM
EUREKA

In the example word search above, several words have been highlighted (note how one of them was spelled RIGHT to LEFT, and another was spelled BOTTOM to TOP). If the user were to search for those highlighted words, your program should return that they were found.

Requirements

You will develop a software application in Python that does the following:

- Runs on the UNIX command line, and accepts two command-line arguments: the name of a text document containing 15 rows of 15 characters, and the word/string to search for.
- Opens and reads the text document.
- Determines whether the desired word is ORTHOGONALLY present in the given word square
- Reports `Gentleman GoGo IS GUILTY OF <word_to_look_for>` OR `<word_to_look_for> was NOT found`

Examples

The following is an example of the application's behaviour:

```
UNIX$ python wordsearch.py confession.txt HUG
HUG was found
UNIX$ python wordsearch.py confession.txt OBJECTION
OBJECTION was NOT found
```

Here is what a wordsquare (`confession.txt`) should look like:

```
UNIX% more confession.txt
M L G Y J U G D T W W I S F P
G Y O H I K O P V F J B J J H
N B I M T M Y W R D J E C A I
I Y M X C U E U Z G U I J Y C
K D P S L R W J I N C N S W T
O P S D I D A B Z I D G D A B
M X D H K E B H U G T S I L G
S G D U Y R V D G N D M L K M
S P X K T W E F P I G E J I T
B U L B C M K I F S I L F N W
Z Q L X H G C J N D I L B G C
M T B W Z L A D A A X Y O K X
A E C Z K F Y V F B U V G A W
Y G O Z E A W J R N S Q J E A
L O I T E R I N G H F I P G R
```

Notice each character is separated by a space.

What to Hand In

- A text document named `a2q1_design.txt` containing your design plan (main functions, your overall approach, main functions have described input/output).
- A Python program named `a2q1.py` containing your implementation.
- A Python script named `a2q1_testing.py` containing your test script (don't forget to test/search for those ten potential crimes listed above. You can simply just paste in your console output as comments in your test file).

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

- 10 marks: Your design plan demonstrated careful planning, including functions, testing, and other important aspects.
  - Your design plan document describes a number of functions in terms of inputs, outputs, and purpose. It describes test cases for each function.
- 10 marks: Your implementation meets the requirements.
  - Your application runs on the command-line, and uses two command line arguments to indicate which file to read, and what word to search for.
  - Your application reads the named file containing 9 rows with 9 columns of integers.
  - Your application determines if a 15×15 wordsquare contains a given word.
  - Your application outputs `yes` or `no` only.
- 10 marks: Your test script demonstrates careful testing.
  - You have testing for each function in your implementation.
  - Your testing is thorough, and could identify errors in your implementation.
  - Your testing checks for the presence of the ten potential crimes listed above.

Advice

It is likely you should read in the word square as a list of lists, with each sublist containing a row of characters: *[['A','B','C'], ['D','E','F'], ...]*.

Searching the word square can be done in a number of different ways. Strings could be constructed going from each direction (left to right, top to bottom, etc.) which could then be checked if the desired word is present in them. A brief summary of this approach would be: construct list of lists to hold individual characters from the word square. Create 1 string per row, and 1 string per column. Check if any of these strings contains the word we're looking for. Note this only checks the directions left to right, and top to bottom; how would we check the directions right to left, and bottom to top? You can check if a string contains a substring by using `in` `(if 'OBJECTION' in some_word: ...)`. You can reverse a string by changing the step parameter of list slicing to go BACKWARDS: `"HOLD IT!"[::-1] = "!TI DLOH"`.

Another approach is to search for the first letter of the word you're looking for, and for each instance found, search the surrounding characters as if they were a 2D array for the subsequent characters. Ex: looking for 'OBJECTION!', find all the 'O's, then check surrounding characters for an 'B', then keep checking in the same direction for subsequent characters. This approach will likely take more time to write than the string checking method in the above.

Question 2 (10 points):

Purpose: To reflect on your experience planning and implementing an application.

Degree of Difficulty: Easy

In this question you will reflect on your experience in Question 1. Answer the following questions about your experience designing and implementing the application. You may use point form, and informal language. Be brief. These are reflection questions, and there is no right answer, and there is no need to go back to Question 1 and change anything as a result of these questions. The intent of these questions is to get you to think about the development process and the way you used them to complete your work.

1. (2 marks) Comment on your **development strategy**. For example, you could address the following issues:
   - In your development, did you follow the waterfall model, or the incremental model?
   - If neither, which was closer to what you did?
2. (2 marks) Comment on your **design plan**. For example, you could address the following issues:
   - How much time did you spend on your design plan?
   - Do you think you planned your application well enough?
   - Did problems arise that you did not plan for? What kinds of problems (if any)?
   - Were there functions you created that were not in your initial plan?
3. (2 marks) Comment on the **implementation stage**. For example, you could address the following issues:
   - Did you estimate the time you'd need to implement the application?
   - Did the implementation take longer or shorter than you planned for, or expected?
   - What took more time than you thought?
   - What took less time than you thought?
4. (2 marks) Comment on your **testing**. For example, you could address the following issues:
   - Did your test script find errors in your functions?
   - Did you discover errors in your functions during your verification stage (when you were running the completed application)?
   - How long did you spend testing and debugging? If you spent more time than you expected, is there any way you can try to reduce the time?
5. (2 marks) Comment on **your use of time**. For example, you could address the following issues:
   - How much time total did you spend, from start to finish (excluding these reflection questions)?
   - Was that more or less than you expected? More or less than you planned?

Remember that the purpose of these questions is to help you learn from your experience.

What to Hand In

Your answers to the above questions in a text file called `a2_reflections.txt` (PDF, rtf, docx or doc are acceptable).

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

Evaluation

Each answer is worth 2 marks. Full marks will be given for any answer that demonstrates thoughtful reflection. Grammar and spelling won't be graded, but practice your professional-level writing skills anyway.