CSC 110: Fundamentals of Programming I
Assignment #6: Arrays

Due date

August 3rd (Friday) at 11:55 via submission to conneX of your solution for uvic_minesweeper.py. Submissions will be accepted until August 7 (Tuesday), 9:00 am, with a penalty of 10%. **The submitted program must work using Anaconda Python 3 as it is installed on the computers in our course's lab room.**

Learning outcomes

When you have completed this assignment, you should understand:

- How to create and work with *two-dimensional (i.e., 2D) NumPy arrays*. Note that in this assignment we use NumPy arrays as a container and for the way NumPy enforces the types of values assigned to NumPy elements (i.e., we will not need to use vectorized operations).

- Write a Python program that is meant to be executed outside of any particular Jupyter Notebook file (i.e., is meant to be executed using the command-line *python* interpreter). Using Python in this way has been covered in the labs.

Minesweeper!

Problem description

Your task in this assignment is to create `uvic_minesweeper.py`. This will be a somewhat simplified version of the original "Minesweeper" game. You can find the rules for the full game at:

- `https://en.wikipedia.org/wiki/Minesweeper_(video_game)`

and you can try out one version of the full game at:

- `http://minesweeperonline.com`

Your version will be much simpler than these and therefore easier to write. (And you need not start off from scratch, and we have started the code for `uvic_minesweeper.py`.). The simplifications in particular are as follows:

1. The playing "grid" will be eight rows by eight columns (8 x 8) and will contain ten mines. *The mines are placed randomly* and code for generating these

random positions is already given to you in the function named `initialize`. Note that the function takes one parameter (and random seed) such that the same seed always results in mines appearing in the same (pseudo-random) grid locations.

2. Your input will be from the computer keyboard instead of via the mouse. You will output all information to the console window. *If the user enters an invalid "grid cell" location (which is just a row/column), then you must indicate the error and ask until you get a valid cell location.*

3. The actions for clearing blank "grid cells" will be much simpler. If the user uncovers a blank cell with no adjacent mines, then the program uncover the eight adjacent cells (and your function does not uncover any further than this). It may be that the cell has adjacent mines, or the cell itself has a mine – see below under `reveal_board_cells` for details on how these two situations are to be handled. After each move your program will display to the user the current state of the board (i.e., blank cells, cells with neighbouring mines, etc.). Again, there is more about this below in the description of the function called `reveal_board_cell`.

4. There will be no score and no timing. *The user wins* if they uncover all cells except for those containing mines. *The user loses* if they directly uncover a cell containing a mine. When the game ends, the location of the mines are to be revealed on the board.

The output of your program must match the format shown in `sample_output_1.pdf` and `sample_output_2.pdf` (i.e., files provided with the assignment). The nature of the interaction with the user shown in the samples should also match. The files show sample games using the default random seed of 200 during initialization. (Note: In these PDFs any text in red boxes is meant to explain how the program output is related to actual game play. You are not meant to produce the text in these red boxes.)

There is some other code also provided for you in `uvic_minesweeper.py`, including that which will read a seed argument from the command line. For example, if you enter the following command in the Anaconda shell (assuming your current directory contains your copy of `uvic_minesweeper.py`):

```
python uvic_minesweeper.py 1234
```

then `1234` will be used as the seed when calling `initialize`. If you enter the following command instead:

```
python uvic_minesweeper.py
```

then the default value of the seed (seen at the top of the Python file) is used instead by `initialize`. (By controlling the randomness this way we can simplifying our

programming. A real game, of course, would not let the user specify the random seed.)

Functions

Your solution **must** implement and use the following two functions, and in addition your solution may use more of your own functions – but they may not be nested functions! You can decide some details of what passed as parameters to these functions and what is to be returned. I have used "???" to refer to one or many NumPy 2D arrays; you are free to choose an appropriate type, but the parameter must be a 2D array (i.e., lists of lists are not permitted). The only global variables you may use are those already given in the `uvic_minesweeper.py` file given to you.

1. Write the following function:

   ```
   reveal_board_cell(???, row, column)
   ```

   where "???" corresponds to parameters you yourself must specify. If the cell at row/column is blank and no mines are adjacent to that cell, then uncover all eight neighbouring cells. Otherwise what appears at that cell is the total number of mines in the eight adjacent cells. Note: You must be careful of edge cases when uncovering neighbours. If the board cell is at an edge, it will have fewer than eight neighbours. And finally, if the cell itself contains a mine, then the game is lost and all mine locations must be shown.

2. Write the following function:

   ```
   print_full_board(???)
   ```

   where "???" corresponds to parameters you must specify. The function will draw the current state of the game board to the console window.

Ensure that your code handles invalid row or column indices throughout the program so that invalid array locations are never accessed (i.e., will not result in the program crashing with an `IndexError`).

Implementation ideas

- Use a 2D array of some type to keep track of the game board (i.e., has cell been revealed? how many mines in neighbouring cells?). This would therefore be a 2D array of strings. The Jupyter Notebook named `a6_notes.ipynb` provided with this assignment contains a few ideas to help you with this.

- You can either pre-compute the number of mines surrounding each game-board cell (i.e., store it in a 2D array of integer), or you can compute this number at run time.

- You can keep the user-input loop within the main function.

Evaluation

The assignment is out of 10 marks.

- 2 marks: Correct interaction with the user
- 2 marks: Game boards are output in the required format.
- 6 marks: Overall correctness of gameplay (i.e., changes to game board are as expected when user enters inputs row/column values).