CS618 Lab 6: Understanding Numpy abd pandas libraries

Warmup task A:

This task requires you to fill in the blanks in a function (`in tests.py and warmup.py`) and return the value described in the comment for the function.

a) You need to solve all 20 of the warmup problems in warmups.py. They are all solvable with one line of code.
b) You need to solve all 20 problems in tests.py. Many are not solvable in one line. You may not use a loop to solve any of the problems, with the exception of t10 (but this one can also be solved without loops).

For each problem, the following is provided:
Inputs: `The arguments that are provided to the function`
Returns: `What you are supposed to return from the function`
Par: `How many lines of code it should take. If it takes more than this, there is probably a better way to solve it. Except for t10, you should not use any explicit loops.`
Instructor: `How many lines ideal solution takes`
Hints: `Functions and other tips you might find useful for this problem`

Task 1: Conway's Game of Life

Conway's Game of Life [1] is a classic example of a cellular automaton devised by mathematician John Conway. The game is a classic example of how simple rules can give rise to complex behavior. The game is played on an `m-by-n board`, which we will represent as an `m-by-n matrix`. The game proceeds in steps. At any given time, each cell of the board (i.e., entry of our matrix), is either `alive` (which we will represent by the Boolean `True`) or `dead` (which we will represent by the Boolean `False`). At each step, the board evolves according to a few simple rules:

- A live cell with fewer than two live neighbors becomes a dead cell.
- A live cell with more than three live neighbors becomes a dead cell.
- A live cell with two or three live neighbors remains alive.
- ˆA dead cell with exactly three live neighbors becomes alive.
- ˆAll other dead cells remain dead.

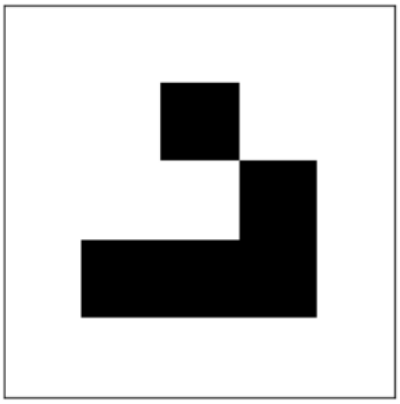
The neighbors of a cell are the 8 cells adjacent to it, i.e., left, right, above, below, upper-left, lower-left, upper-right and lower-right. Here, we follow the convention that the board is toroidal, so that using matrix-like notation (i.e., the cell (0, 0) is in the upper-left of the board and the first coordinate specifies a row), the upper neighbor of the cell (0, 0) is (m − 1, 0), the right neighbor of the cell (m − 1, n − 1) is (m − 1, 0), etc. That is, the board "wraps around".
Note: This definition gets a bit odd when the board is smaller than three-by-three. So you may choose to have board sizes greater than three-by-three.

What needs to be implemented ?

1. Write a function `is_valid_board` that takes a single argument and returns a Python Boolean that is `True` if and only if the argument is a valid representation of a Game of Life board. A valid board is any two-dimensional numpy `ndarray` whose entries are all numpy Booleans (i.e., numpy.bool_), with all entries either 0.0 and 1.0.

2. Write a function called `gol_step` that takes an `m-by-n numpy array` as its argument and returns `another numpy array` of the same size (i.e., `also m-by-n`), corresponding to the board at the next step of the game. Your function should perform error checking to ensure that the provided argument is a valid Game of Life board.

3. Write a function called `draw_gol_board` that takes an m-by-n numpy array `(i.e., an ndarray)` as its only argument and draws the board as an m-by-n set of tiles, colored black or white correspond to whether the corresponding cell is alive or dead, respectively. Your plot should not have any grid lines, nor should it have any axis labels or axis ticks. Hint: see the functions `plt.xticks()` and `plt.yticks()` for changing axis ticks. Hint: you may find the function `plt.get_cmap` to be useful for working with the matplotlib Colormap objects.
4. Create a 20-by-20 numpy array corresponding to a Game of Life board in which all cells are dead, with the exception that the top-left 5-by-5 section of the board looks like this:



Plot this 20-by-20 board using draw_gol_board, save this plot in .png file called glider_board.png, and include it in your submission.

Generate a plot with 5 subplots, arranged in a 5-by-1 grid, showing the first five steps of the Game of Life when started with the board you just created, with the steps ordered from top to bottom. The first plot should be the starting board just described, so that the starting board is the first “step”. The figure in the 5-by-5 sub-board above is called a glider, and it is interesting in that, as you can see from your plot, it seems to move along the board as you run the game. Save the resulting plot in a .png file called glider_5steps.png, and include it in your submission.

Bonus Question: Create a function that takes two arguments, a Game of Life board and a number of steps, and generates an animation of the game as it runs for the given number of steps. Note: this is an optional exercise.

Warmup task B: constructing pandas objects

Create a pandas Series object with indices given by the first 10 letters of the English alphabet and values given by the first 10 primes. Store it in a variable called Alphabet.

Below is a table that gets used in genetics experiments. Reconstruct this as a pandas DataFrame. Store it in a variable called simple_table.

			score1	score2
animal	parent1	parent2		
goat	A	A	1	2
		a	2	4
	a	A	3	4
		a	4	6
bird	A	A	5	6
		a	6	8
	a	A	7	8
		a	8	10
llama	A	A	9	10
		a	10	12
	a	A	11	12
		a	12	14

Task 2:

For this task, you will be working with pandas DataFrames, reading them into and out of memory, changing their contents and performing aggregation operations. For this problem, you'll need to download the `iris.csv` file

1. Read iris.csv into Python as a pandas DataFrame. Note that the CSV file includes column headers. How many data points are there in this data set? What are the data types of the columns? What are the column names? The column names correspond to flower species names, as well as four basic measurements one can make of a flower: the width and length of its petals and the width and length of its sepal (the part of the plant that supports and protects the flower itself). How many species of flower are included in the data?
   Define a function called t2_q1_sol() which prints the solutions to all the questions asked above

2. The iris.csv file that you have downloaded has some errors. Using 1-indexing, these errors are in the 35th and 38th rows. The 35th row should read 4.9,3.1,1.5,0.2,"setosa", where the fourth feature is incorrect as it appears in the file, and the 38th row should read 4.9,3.6,1.4,0.1,"setosa", where the second and third features are incorrect as they appear in the file. Correct these entries of your DataFrame.
   The iris dataset is commonly used in machine learning as a proving ground for clustering and classification algorithms. Some researchers have found it useful to use two additional features, called `Petal ratio` and `Sepal ratio`, defined as the ratio of the petal length to petal width and the ratio of the sepal length to sepal width, respectively. Add two columns to your DataFrame corresponding to these two new features. Name these columns `Petal.Ratio` and `Sepal.Ratio`, respectively. Save your corrected and extended iris DataFrame to a csv file called `iris_corrected.csv`. Please include this file in your submission.

3. Use a pandas aggregate operation to determine the mean, median, minimum, maximum and standard deviation of the petal and sepal ratio for each of the three species in the data set. Store the results of your aggregation operation in a variable called `species_agg`. Note: it is possible that you will be able to get all of these numbers in a single table using a single line of code.

4. Create a scatterplot of the iris specimens, with sepal ratio on the x-axis and petal ratio on the y-axis, and with observations colored according to species. Save your plot in a file called `iris_scatter.pdf`. Please include this file in your submission.