Class 7 Additional Optional Exercises: Working with NumPy and Matplotlib

1. Magic squares. An nXn matrix that is filled with the numbers 1, 2, 3, . . ., n^2 is a magic square if the sum of the elements in each row, in each column, and in the two diagonals is the same value.

e.g.

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Write a program that reads in 16 values as user input and tests whether the particular order forms a magic square when put into a 4x4 array. For example, the above magic square would result if the user inputs numbers in the following order: 16, 3, 2, 13, 5, 10, 11, 8, 9, 6, 7, 12, 4, 15, 14, 1.
Therefore, you need to test two features:
(1) Does each of the numbers 1, 2, ..., 16 occur in the user input?
(2) When the numbers are put into a square, are the sums of the rows, columns, and diagonals equal to each other?

2. Implement the following algorithm in Python to construct magic nxn squares; it works only if n is odd.

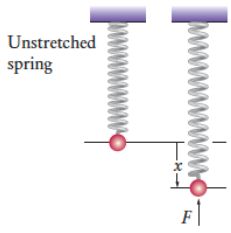
Set row = n - 1, column = n / 2.
For k = 1 ... n^2
  Place k at [row][column].
  Increment row and column.
  If the row or column is n, replace it with 0.
  If the element at [row][column] has already been filled
    Set row and column to their previous values.
    Decrement row.

For example, here is the 5x5 square that you get if you follow this method:

11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Take in the size of the matrix, n, from the user and provide the magic square of order n (only if n is odd.)

3. Write a program that models the movement of an object with mass m that is attached to an oscillating spring. When a spring is displaced from its equilibrium position by an amount x, Hooke's law states that the restoring force is **F = –kx** where k is a constant that depends on the spring. (Use 10 N•m for this simulation.)



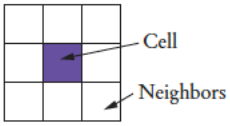
Start with a given displacement x (say, 0.5 meter).
Set the initial velocity v to 0.
Compute the acceleration **a** from Newton's law (F = ma) and Hooke's law, using a mass of 1 kg. Use a small time interval t = 0.01 second. Update the velocity—it changes by **a*t**. Update the displacement—it changes by **v*t**.
Plot displacement by time. (You should start with a NumPy array for time, storing time values from 0 to 10 in increments of 0.01 seconds. Then create the displacement array)

Problems 1-3 are adapted from Horstmann, Big Java

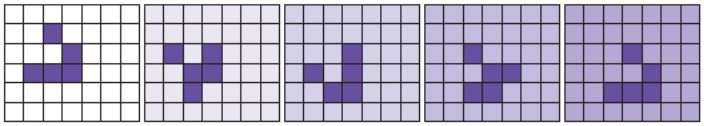
4. The **Game of Life** is a well-known mathematical game called 'cellular automaton' invented by John Conway. Following just a simple rules, you can create amazingly complex and interesting patterns!

(https://bitstorm.org/gameoflife/ Explore this site for a simulation of the game!)

Here are the rules. The game is played on a rectangular board. Each square can be either empty or occupied. At the beginning, you can specify empty and occupied cells in some way; then the game runs automatically. In each generation, the next generation is computed. **A new cell is born on an empty square if it is surrounded by exactly three occupied neighbor cells. A cell dies of overcrowding if it is surrounded by four or more neighbors**, and **it dies of loneliness if it is surrounded by zero or one neighbor**. A neighbor is an occupant of an adjacent square to the left, right, top, or bottom or in a diagonal direction.

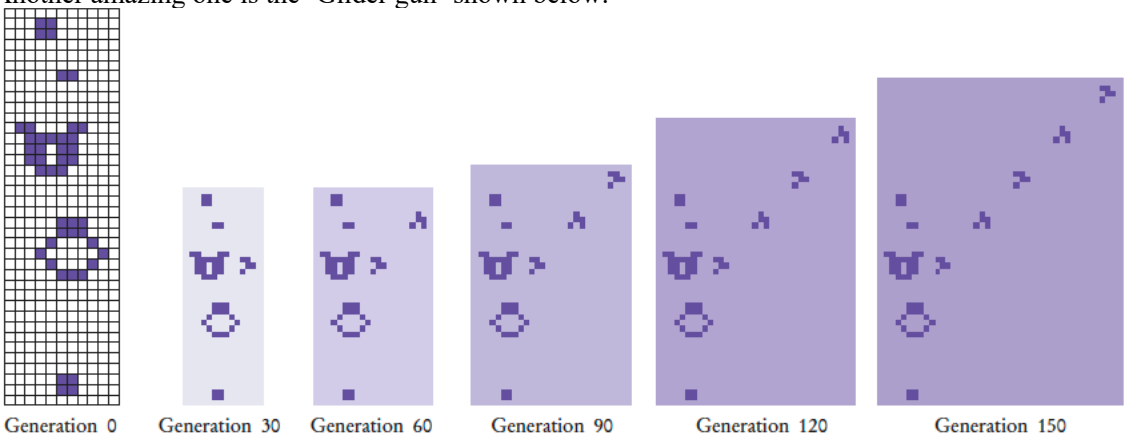


For example, all of these white 8 squares are neighbors of the purple cell in the center.



The most famous example is the "Glider" After four generations, it is transformed into the identical shape, but located one square to the right and below.

Another amazing one is the 'Glider gun' shown below:



The complex pattern in generation 0 turns back into itself with a glider to the right of it after 30 generations.
Your job is to program this game with NumPy matrices.
Ask the user to specify the original configuration (choose whichever way is best for your algorithm: Some suggestions are reading in a text file full of X's and O's for empty and occupied spaces, or giving the user a finite nxn matrix and asking them to specify whether each index is empty or occupied in order.)

Then, show successive generations until the generation specified by the user.