CS4048 – Data Science
Assignment 2
BSE7A – BCS7C

Submission Guidelines

- Create a PDF report on all the questions given below by strictly following the question order. For all the question with missing solution, you need to mention "NA" comment.
- Turn in the solution to google classroom.
- Keep the naming convention of report and code solutions as f20xxxx_sec_A3.pdf/ipynb.
- Submit all solution codes to just one. ipynb file.
- Naming conventions are required to be followed strictly to avoid any inconvenience.
- **Deadline: Friday, October 27, 2023, 4:00 Sharp.**

1. (Element-Wise array Multiplication) Create a 10-by-10 array containing the number 4 in each position. Create a second 10-by-10 array containing the numbers 1 to 100 in ascending order. Multiply the first array by the second array.

2. (array from List of Lists) Create an array from a list of two lists. The first list consists of the even numbers counting down from 10 to 0 and the second counting up from 0 to 10.

3. (Flattening arrays with flatten vs. ravel) Using the array created in Exercise 2, first, flatten the array with the method flatten. Change the second element of the new array to 10. Compare the new and the original array. Second, flatten the array using the method ravel and perform the same comparison.

4. (Research: array Method astype) Research in the NumPy documentation the array method astype, which converts an array's elements to another type. Use linspace to create a 1-by-6 array with the values 1.1, 2.2, …, 6.6. Then, use astype to convert the array to integers, then convert it to an array with floats and finally, convert it to an array of strings.

5. (Challenge Project: Reimplement NumPy array Output) You saw that NumPy outputs two-dimensional arrays in a nice column-based format that right-aligns every element in a field width. The field width's size is determined by the array element value that requires the most character positions to display. To understand how powerful, it is to have this formatting simply built-in, write a function that reimplements NumPy's array formatting for two-dimensional arrays using loops. Assume the array contains only positive integer values.

6. (Challenge Project: Reimplement DataFrame Output) You saw that pandas display DataFrames in an attractive column-based format with row and column labels. The values within each column are right aligned in the same field width, which is determined by that column's widest value. To understand how powerful, it is to have this formatting built-in, write a function that reimplements DataFrame formatting using loops. Assume the DataFrame contains only positive integer values and that both the row and column labels are each integer values beginning at 0.

7. (Project: Two-Player, Two-Dimensional Tic-Tac-Toe) Write a script to play two-dimensional Tic-Tac-Toe between two human players who alternate entering their moves on the same computer. Use a 3-by-3 two-dimensional array. Each player indicates their moves by entering a pair of numbers representing the row and column indices of the square in which they want to place their mark, either an 'X' or an 'O'. When the first player moves, place an 'X' in the specified square. When the second player moves, place an 'O' in the specified square. Each move must be to an empty square. After each move, determine whether the game has been won and whether it's a draw.

8. (Challenge Project: Tic-Tac-Toe with Player Against the Computer) Modify your script from the previous exercise so that the computer makes the moves for one of the players. Also, allow the player to specify whether he or she wants to go first or second.

9. (Super Challenge Project: 3D Tic-Tac-Toe with Player Against the Computer) Develop a script that plays three-dimensional Tic-Tac-Toe on a 4-by-4-by-4 board.

10. (Research and Use Other Broadcasting Capabilities) Research the NumPy broadcasting rules, then create your own arrays to test the rules. (Horizontal and Vertical Stacking and Splitting) Create the following arrays:

    array1 = np.array([[0, 2], [4, 6]])
    array2 = np.array([[1, 3], [5, 7]])

    a) Use horizontal stacking to create array3 with array2 to the right of array1.
    b) Use vertical splitting to create array4 and array5, with array4 containing the top row of array3 and array5 containing the bottom row.
    c) Sort both array4 and array5 and use horizontal stacking for both sorted arrays to create array6.
    d) Use vertical stacking to create array7 stacking array6 and the product of array6 with 10.

11. (Research: NumPy bincount Function) Use the Numpy randint function to create an array of 99 0's and 1's. A "0" indicates a vote for Candidate 1 and a "1" indicates a vote for Candidate 2. Research and use the NumPy bincount function to count the number of votes per candidate.

12. (Median and Mode of an array) NumPy arrays offer a mean method, but not median or mode. Write functions median and mode that use existing NumPy capabilities to determine the median (middle) and mode (most frequent) of the values in an array. Your functions should determine the median and mode regardless of the array's shape. Test your function on three arrays of different shapes.

13. (Enhanced Median and Mode of an array) Modify your functions from the previous exercise to allow the user to provide an axis keyword argument so the calculations can be performed row-by-row or column-by-column on a two-dimensional array.

14. (AI Project: Introducing Heuristic Programming with the Knight's Tour) An interesting puzzle for chess buffs is the Knight's Tour problem, originally proposed by the mathematician Euler. Can the knight piece move around an empty chessboard and touch each of the 64 squares once and only once? We study this intriguing problem in depth here. The knight makes only L-shaped moves (two spaces in one direction and one space in a perpendicular direction). Thus, as shown in the figure below, from a square near the middle of an empty chessboard, the knight (labeled K) can make eight different moves (numbered 0 through 7).

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0								
1				2		1		
2			3				0	
3					K			
4			4				7	
5				5		6		
6								
7								

Draw an eight-by-eight chessboard on a sheet of paper and attempt a Knight's Tour by hand. Put a 1 in the starting square, a 2 in the second square, a 3 in the third, and so on. Before starting the tour, estimate how far you think you'll get, remembering that a full tour consists of 64 moves. How far did you get? Was this close to your estimate?

b) Now let's develop a script that will move the knight around a chessboard represented by an eight-by-eight two-dimensional array named board. Initialize each square to zero. We describe each of the eight possible moves in terms of its horizontal and vertical components. For example, a move of type 0, as shown in the preceding figure, consists of moving two squares horizontally to the right and one square vertically upward. A move of type 2 consists of moving one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves upward are indicated with negative numbers. The eight moves may be described by two one-dimensional arrays, horizontal and vertical, as follows:

```
horizontal[0] = 2 vertical[0] = -1
horizontal[1] = 1 vertical[1] = -2
horizontal[2] = -1 vertical[2] = -2
horizontal[3] = -2 vertical[3] = -1
horizontal[4] = -2 vertical[4] = 1
horizontal[5] = -1 vertical[5] = 2
horizontal[6] = 1 vertical[6] = 2
horizontal[7] = 2 vertical[7] = 1
```

Let the variables current_row and current_column indicate the row and column, respectively, of the knight's current position. To make a move of type move_number (a value 0–7), your script should use the statements:

```
current_row += vertical[move_number]
current_column += horizontal[move_number]
```

Write a script to move the knight around the chessboard. Keep a counter that varies from 1 to 64. Record the latest count in each square the knight moves to. Test each potential move to see if the knight has already visited that square. Test every potential move to ensure that the knight does not land off the chessboard. Run the application. How many moves did the knight make?

c) After attempting to write and run a Knight's Tour script, you've probably developed some valuable insights. We'll use these insights to develop a heuristic (i.e., a common-sense rule) for moving the knight. Heuristics do not guarantee success, but a carefully developed heuristic greatly improves the chance of success. You may have observed that the outer squares are more troublesome than the squares nearer the center of the board. In fact, the most troublesome or inaccessible squares are the four corners. Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave open those that are easiest to get to so that when the board gets congested near the end of the tour, there will be a greater chance of success. We could develop an "accessibility heuristic" by classifying each of the squares according to how accessible it is and always moving the knight (using the knight's L-shaped moves) to the most inaccessible square. We fill a two-dimensional array accessibility with numbers indicating from how many squares each square is accessible. On a blank chessboard, each of the 16 squares nearest the center is rated as 8, each corner square is rated as 2, and the other squares have accessibility numbers of 3, 4 or 6 as follows:

2	3	4	4	4	4	3	2
3	4	6	6	6	6	4	3
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

Write a new version of the Knight's Tour, using the accessibility heuristic. The knight should always move to the square with the lowest accessibility number. In case of a tie, the knight may move to any of the tied squares. Therefore, the tour may begin in any of the four corners. [Note: As the knight moves around the chessboard, your application should reduce the accessibility numbers as more squares become occupied. In this way, at any given time during the tour, each available square's accessibility number will remain equal to precisely the number of squares from which that square may be reached.] Run this version of your script. Did you get a full tour? Modify the script to run 64 tours, one starting from each square of the chessboard. How many full tours did you get?