




OCTOBER 29, 2023

DATA SCIENCE REPORT

ASSIGNMENT 2

ABDULLAH MEHMOOD MINHAS
20F-0204
B(SE)-7A



Question 1:

#Question 1

```
array1 = np.full((10, 10), 4)
array2 = np.arange(1,101,dtype=int).reshape((10,10))
array_final = array1 * array2
array_final
```

```
Out[66]: array([[ 4,  8, 12, 16, 20, 24, 28, 32, 36, 40],
 [ 44, 48, 52, 56, 60, 64, 68, 72, 76, 80],
 [ 84, 88, 92, 96, 100, 104, 108, 112, 116, 120],
 [124, 128, 132, 136, 140, 144, 148, 152, 156, 160],
 [164, 168, 172, 176, 180, 184, 188, 192, 196, 200],
 [204, 208, 212, 216, 220, 224, 228, 232, 236, 240],
 [244, 248, 252, 256, 260, 264, 268, 272, 276, 280],
 [284, 288, 292, 296, 300, 304, 308, 312, 316, 320],
 [324, 328, 332, 336, 340, 344, 348, 352, 356, 360],
 [364, 368, 372, 376, 380, 384, 388, 392, 396, 400]])
```

Question 2:

#Question 2

```
list1 = [[x for x in range(10,0,-1) if x % 2 == 0],[x for x in range(0,10) if x % 2 == 0]]
```

```
array_org = np.array(list1)
print(array_org)
```

```
[[10  8  6  4  2]
 [ 0  2  4  6  8]]
```

Question 3:

#Question 3

```
fltn_array = array_org.flatten()
fltn_array[1] = 10
print('original array = ', array_org)
print('Flatten Array = ', fltn_array)
```

```
ravel_array = array_org.ravel()
ravel_array[1] = 10
print('original array = ', array_org)
print('Ravel Array = ', ravel_array)
```

#After applying ravel function, original array elements changes

```
original array = [[10 10  6  4  2]
 [ 0  2  4  6  8]]
Flatten Array = [10 10  6  4  2  0  2  4  6  8]
original array = [[10 10  6  4  2]
 [ 0  2  4  6  8]]
Ravel Array = [10 10  6  4  2  0  2  4  6  8]
```

Question 4:

#Question 4

```
ls_array = np.linspace(1.1, 6.6, num = 6)
```

```
df = pd.DataFrame(data=ls_array)
```

```
print(df.astype('int'))
```

```
print(df.astype('float'))
```

```
print(df.astype('str'))
```

```

      0
0  1
1  2
2  3
3  4
4  5
5  6
      0
0  1.1
1  2.2
2  3.3
3  4.4
4  5.5
5  6.6
      0
0      1.1
1      2.2
2  3.3000000000000003
3      4.4
4      5.5
5      6.6

```

Syntax:

```
DataFrame.astype(dtype, copy=None, errors='raise')
```

Effect of astype:

Use a str, numpy.dtype, pandas.ExtensionDtype or Python type to cast entire pandas object to the same type. Alternatively, use a mapping, e.g. {col: dtype, ...}, where col is a column label and dtype is a numpy.dtype or Python type to cast one or more of the DataFrame's columns to column-specific types.

Conclusion:

Based on the findings from the given problem, we started with the array of floats. Then, when we converted it 'int.' It dropped the decimal part from the array. Then we converted it to 'float' the decimal part was intact in this case, and in case of 'string' it converts the array elements to strings. However, in case of string it behaves a little differently which is kind of funny but useful in certain scenarios. To learn more about that you can visit stack overflow link given below.

References:

<https://github.com/pandas-dev/pandas/blob/v2.1.2/pandas/core/generic.py#L6368-L6551>

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.astype.html>

<https://stackoverflow.com/questions/26592723/what-rules-dictate-how-python-floats-are-rounded>

Question 5

#Question 5

```
val = [[x*y for x in range(1, 10)] for y in range(1, 10)]
```

```
largest_num = max([y for x in val for y in x])
```

```
largest_len = len(str(largest_num))
```

```
newlist = [[] for _ in range(9)];
```

```
for idx,row in enumerate(val):
```

```
    for char in row:
```

```
        if(len(str(char)) == largest_len):
```

```
            value = str(char)
```

```
        else:
```

```
            spaces = largest_len - len(str(char))
```

```
            value = (' '*spaces) + str(char)
```

```
            newlist[idx].append(value)
```

```
newlist
```

```

In [177]: #Question 5
val = [[x*y for x in range(1, 10)] for y in range(1, 10)]

largest_num = max([y for x in val for y in x])
largest_len = len(str(largest_num))

newlist = [[] for _ in range(9)];

for idx,row in enumerate(val):
    for char in row:
        if(len(str(char)) == largest_len):|
            value = str(char)
        else:
            spaces = largest_len - len(str(char))
            value = (' '*spaces) + str(char)
        newlist[idx].append(value)
newlist

```

```

Out[177]: [[' 1', ' 2', ' 3', ' 4', ' 5', ' 6', ' 7', ' 8', ' 9'],
[' 2', ' 4', ' 6', ' 8', '10', '12', '14', '16', '18'],
[' 3', ' 6', ' 9', '12', '15', '18', '21', '24', '27'],
[' 4', ' 8', '12', '16', '20', '24', '28', '32', '36'],
[' 5', '10', '15', '20', '25', '30', '35', '40', '45'],
[' 6', '12', '18', '24', '30', '36', '42', '48', '54'],
[' 7', '14', '21', '28', '35', '42', '49', '56', '63'],
[' 8', '16', '24', '32', '40', '48', '56', '64', '72'],
[' 9', '18', '27', '36', '45', '54', '63', '72', '81']]

```

Explanation:

First, I generated data in the form of a 2d list. For questions 5 and 6 I am going to work on this data. This approach will work fine on any data. Then I flattened the list and found the largest number in the whole matrix of 2d lists. And finally, I find the length of the largest number on basis of which we are going to assign spaces to our output. This approach is common in both questions.

After that, I made my own sort of array which is almost identical to np array. The only issue with this is It is containing only string type of data else if we convert it to integer, it drops the spaces. The other way to do this is to store data in form of a string, in that case it will not be like np array, it'd more of a matrix made up of strings.

Question 6

#Question 6

```
val = [[x*y for x in range(1, 10)] for y in range(1, 10)]
```

```
largest_num = max([y for x in val for y in x])
largest_len = len(str(largest_num))
```

```
for idx,row in enumerate(val):
    if(idx==0):
        print(' ', end='')
        for x in range(len(max(val))):
            spaces = 0
            if(len(str(x)) != largest_len):
                spaces = largest_len - len(str(x))
            print(' '*spaces, end='')
            print(x, end=' ')
        print(end='\n\n')
    print(idx, end=' ')
    for char in row:
        if(len(str(char)) == largest_len):
            value = char
        else:
            spaces = largest_len - len(str(char))
            value = (' '*spaces) + str(char)
        print(value, end=' ')
    print(end='\n')
```

```

In [205]: #Question 6
val = [[x*y for x in range(1, 10)] for y in range(1, 10)]

largest_num = max([y for x in val for y in x])
largest_len = len(str(largest_num))

for idx,row in enumerate(val):
    if(idx==0):
        print(' ', end='')
        for x in range(len(max(val))):
            spaces = 0
            if(len(str(x)) != largest_len):
                spaces = largest_len - len(str(x))
            print(' '*spaces, end='')
            print(x, end=' ')
        print(end='\n\n')
    print(idx, end=' ')
    for char in row:
        if(len(str(char)) == largest_len):
            value = char
        else:
            spaces = largest_len - len(str(char))
            value = (' '*spaces) + str(char)
        print(value, end=' ')
    print(end='\n')

```

	0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8	9
1	2	4	6	8	10	12	14	16	18
2	3	6	9	12	15	18	21	24	27
3	4	8	12	16	20	24	28	32	36
4	5	10	15	20	25	30	35	40	45
5	6	12	18	24	30	36	42	48	54
6	7	14	21	28	35	42	49	56	63
7	8	16	24	32	40	48	56	64	72
8	9	18	27	36	45	54	63	72	81

Explanation

Same as the previous question, I generated data in the form of a 2d list. For questions 5 and 6 I am going to work on this data. This approach will work fine on any data. Then I flattened the list and found the largest number in the whole matrix of 2d lists. And finally, I find the length of the largest number on basis of which we are going to assign spaces to our output. This approach is common in both questions.

Then I tried to print it like the DataFrame of pandas. Well I think this looks a lot like a data frame :)

Question 7:

#Question 7

```
input_array = ['01','00','02','10','11','12','20','21','22']
grid = np.full((3, 3), ' ')
print("Let's Start the Game..!!!", end='\n\n')
print(grid, end='\n\n')
```

```
player = 1
draw = True
def check_winner(grid):
    # Check rows
    for row in grid:
        if row[0] == row[1] == row[2] != ' ':
            return row[0]

    # Check columns
    for col in range(3):
        if grid[0][col] == grid[1][col] == grid[2][col] != ' ':
            return grid[0][col]

    # Check diagonals
    if grid[0][0] == grid[1][1] == grid[2][2] != ' ':
        return grid[0][0]
    if grid[0][2] == grid[1][1] == grid[2][0] != ' ':
        return grid[0][2]

    return None
```

```
for x in range(9):
    if x % 2 == 0:
        print('Player 1\'s Turn')
    else:
        print('Player 2\'s Turn')

    while True:
        val = input('Enter your move = ')
        if val in input_array:
            input_array.remove(val)
            row = int(val[0])
            col = int(val[1])
```



```
    if grid[row][col] == ' ':
        if player == 1:
            grid[row][col] = 'X'
        else:
            grid[row][col] = 'O'
        break
    else:
        print('This position is already taken. Try again.')
else:
    print('Oops, wrong input format. Enter again.')

print(grid, end='\n\n')
winner = check_winner(grid)
if x > 3 and winner:
    print(f'Player {winner} wins!')
    draw = False
    break
player = 3 - player
if(draw):
    print("It's a Draw....!!!!")
```

Let's Start the Game..!!!

```
[[ ' ' ' ' ' ' ' ' ]  
[ ' ' ' ' ' ' ' ' ]  
[ ' ' ' ' ' ' ' ' ]
```

Player 1's Turn

Enter your move = 12

```
[[ ' ' ' ' ' ' ' ' ]  
[ ' ' ' ' 'X' ' ' ]  
[ ' ' ' ' ' ' ' ' ]
```

Player 2's Turn

Enter your move = 12

Oops, wrong input format. Enter again.

Enter your move = 11

```
[[ ' ' ' ' ' ' ' ' ]  
[ ' ' 'O' 'X' ' ' ]  
[ ' ' ' ' ' ' ' ' ]
```

Player 1's Turn

Enter your move = 21

```
[[ ' ' ' ' ' ' ' ' ]  
[ ' ' 'O' 'X' ' ' ]  
[ ' ' 'X' ' ' ' ' ]
```

Player 2's Turn

Enter your move = 22

```
[[ ' ' ' ' ' ' ' ' ]  
[ ' ' 'O' 'X' ' ' ]  
[ ' ' 'X' 'O' ' ' ]
```

Player 1's Turn

Enter your move = 10

```
[[ ' ' ' ' ' ' ' ' ]  
[ 'X' 'O' ' 'X' ' ' ]  
[ ' ' 'X' 'O' ' ' ]
```

Player 2's Turn

Enter your move = 00

```
[[ 'O' ' ' ' ' ' ' ]  
[ 'X' 'O' ' 'X' ' ' ]  
[ ' ' 'X' 'O' ' ' ]
```

Player 0 wins!

Question 8:

#Question 8

```
import random
```

```
input_array = ['01','00','02','10','11','12','20','21','22']
grid = np.full((3, 3), ' ')
print("Let's Start the Game..!!!", end='\n\n')
print(grid, end='\n\n')
```

```
player = 1
draw = True;
while True:
    turn = input('You want to play as player 1 or 2: ')
    if(turn == '1' or turn == '2'):
        break
    print('Worng input, Enter 1 or 2...!!!')
```

```
num = 4
if(turn == 1):
    num=5
```

```
def check_winner(grid):
    for row in grid:
        if row[0] == row[1] == row[2] != ' ':
            return row[0]

    for col in range(3):
        if grid[0][col] == grid[1][col] == grid[2][col] != ' ':
            return grid[0][col]

    if grid[0][0] == grid[1][1] == grid[2][2] != ' ':
        return grid[0][0]
    if grid[0][2] == grid[1][1] == grid[2][0] != ' ':
        return grid[0][2]

    return None
```

```
for x in range(num+1):
```

```
    #Player's Turn
    if((turn == '2' and x!=0) or (turn == '1')):
        print('Player\'s Turn')
        while True:
            val = input('Enter your move (row and column, e.g., 01) = ')
```

```

if val in input_array:
    input_array.remove(val)
    row = int(val[0])
    col = int(val[1])
    if grid[row][col] == ' ':
        grid[row][col] = 'X'
        print(grid, end='\n\n')
        winner = check_winner(grid)
        break
    else:
        print('This position is already taken. Try again.')
else:
    print('Oops, wrong input format. Enter again.')
if winner:
    print(f'You win!')
    draw = False
    break

if((turn=='1' and x!=4)or(turn == '2')):
#Computer's turn
print('Computer\'s Turn')
while True:
    row = random.randint(0, 2)
    col = random.randint(0, 2)
    if grid[row][col] == ' ':
        temp = str(row)+str(col)
        input_array.remove(temp)
        grid[row][col] = 'O'
        print(grid, end='\n\n')
        winner = check_winner(grid)
        break
if winner:
    print(f'Computer wins!')
    draw = False
    break
if(draw):
    print("It's a Draw....!!!!")

```

Let's Start the Game..!!!

```
[[ ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ]
```

You want to play as player 1 or 2: 1

Player's Turn

Enter your move (row and column, e.g., 01) = 12

```
[[ ' ' ' ' ' ' ' ' ]
[ ' ' ' ' 'X' ' ' ]
[ ' ' ' ' ' ' ' ' ]
```

Computer's Turn

```
[[ ' ' ' ' ' ' ' ' ]
[ ' ' ' ' 'X' ' ' ]
[ ' ' 'O' ' ' ' ' ]
```

Player's Turn

Enter your move (row and column, e.g., 01) = 02

```
[[ ' ' ' ' 'X' ' ' ]
[ ' ' ' ' 'X' ' ' ]
[ ' ' 'O' ' ' ' ' ]
```

Computer's Turn

```
[[ ' ' ' ' 'X' ' ' ]
[ ' ' ' ' 'X' ' ' ]
[ ' ' 'O' 'O' ' ' ' ]
```

Player's Turn

Enter your move (row and column, e.g., 01) = 20

```
[[ ' ' ' ' 'X' ' ' ]
[ ' ' ' ' 'X' ' ' ]
[ 'X' 'O' 'O' ' ' ' ]
```

Computer's Turn

```
[[ ' ' ' ' 'X' ' ' ]
[ 'O' ' ' 'X' ' ' ]
[ 'X' 'O' 'O' ' ' ' ]
```

```

Player's Turn
Enter your move (row and column, e.g., 01) = 00
[['X' ' ' 'X']
 ['O' ' ' 'X']
 ['X' 'O' 'O']]

Computer's Turn
[['X' 'O' 'X']
 ['O' ' ' 'X']
 ['X' 'O' 'O']]

Player's Turn
Enter your move (row and column, e.g., 01) = 11
[['X' 'O' 'X']
 ['O' 'X' 'X']
 ['X' 'O' 'O']]

You win!

```

Question 9:

3D Tic-Tac-Toe

#Question 9

```

import re
grid = np.full((4, 4, 4), ' ')
print("Let's Start the 3D Tic-Tac-Toe Game..!!!", end='\n\n')
print(grid, end='\n\n')

player = 1
draw = True

while True:
    turn = input('You want to play as player 1 or 2: ')
    if turn in ('1', '2'):
        break
    print('Wrong input, Enter 1 or 2...!!!')

def check_winner(grid):
    for layer in grid:
        for row in layer:
            if row[0] == row[1] == row[2] == row[3] != ' ':
                return row[0]

        for col in range(3):
            if layer[0][col] == layer[1][col] == layer[2][col] == layer[3][col] != ' ':

```

```

        return layer[0][col]

    if layer[0][0] == layer[1][1] == layer[2][2] == layer[3][3] != ' ':
        return layer[0][0]
    if layer[0][3] == layer[1][2] == layer[2][1] == layer[3][0] != ' ':
        return layer[0][3]

    if grid[0][0][0] == grid[1][1][1] == grid[2][2][2] == grid[3][3][3] != ' ':
        return grid[0][0][0]
    if grid[0][0][3] == grid[1][1][2] == grid[2][2][1] == grid[3][3][0] != ' ':
        return grid[0][0][3]
    if grid[0][3][0] == grid[1][2][1] == grid[2][1][2] == grid[3][0][3] != ' ':
        return grid[0][3][0]
    if grid[0][3][3] == grid[1][2][2] == grid[2][1][1] == grid[3][0][0] != ' ':
        return grid[0][3][3]

    return None

for x in range(32):

    if (turn == '2' and x != 0) or (turn == '1'):
        print('Player\'s Turn')
        while True:
            val = input('Enter your move (layer, row, and column, e.g., 012) = ')
            if re.search('[0-3]{3}', val):
                layer = int(val[0])
                row = int(val[1])
                col = int(val[2])
                if grid[layer][row][col] == ' ':
                    grid[layer][row][col] = 'X'
                    print(grid)
                    winner = check_winner(grid)
                    break
            else:
                print('This position is already taken. Try again.')
        else:
            print('Oops, wrong input format. Enter again.')
    if winner:
        print(f'Player wins!')
        draw = False
        break

    if (turn == '2' and x != 31) or (turn == '1'):
        print('Computer\'s Turn')

```

```
if draw:
    print("It's a Draw....!!!!")
```

$$\begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \\ \begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix} \\ \begin{bmatrix} 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix} \\ \begin{bmatrix} 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{bmatrix} \end{bmatrix}$$
$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$
$$\begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \\ \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \\ \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \\ \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \end{bmatrix}$$
$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

```

Array 3 with horizontal stacking [[0 2 1 3]
 [4 6 5 7]]
Array 4 with vertical splitting for top values [[0 2 1 3]]
Array 5 with vertical splitting for bottom values [[4 6 5 7]]
[[0 1 2 3 4 5 6 7]]
[[ 0  1  2  3  4  5  6  7]
 [ 0 10 20 30 40 50 60 70]]

```

References:

[https://sparkbyexamples.com/numpy/numpy-hstack-function/#:~:text=NumPy%20hstack\(\)%20function%20in,into%20a%20single%20NumPy%20array.](https://sparkbyexamples.com/numpy/numpy-hstack-function/#:~:text=NumPy%20hstack()%20function%20in,into%20a%20single%20NumPy%20array.)

<https://numpy.org/doc/stable/reference/generated/numpy.vsplit.html>

Question 11:

#Question 11

```
votes = np.random.randint(2, size=99)
```

```
print(votes)
```

```
print(np.bincount(votes))
```

```
candidate1,candidate2 = np.bincount(votes)
```

```
print(candidate1)
```

```
print(candidate2)
```

```

In [29]: #Question 11
votes = np.random.randint(2, size=99)
print(votes)

print(np.bincount(votes))
candidate1,candidate2 = np.bincount(votes)

print(candidate1)
print(candidate2)

```

```

[0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 1
 1 1 1 0 1 1 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 1
 0 1 0 0 0 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 0 1 0]
[50 49]
50
49

```

References:

<https://numpy.org/doc/stable/reference/generated/numpy.bincount.html>

Question 12:

#Question 12

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

```
array2 = np.array([[1, 2, 2], [3, 3, 4]])
```

```
array3 = np.array([1, 2, 2, 3, 4, 4, 4])
```

```
def median_func(arr):
```

```
    flattened_arr = arr.flatten()
```

```
    sorted_arr = np.sort(flattened_arr)
```

```
    n = len(sorted_arr)
```

```
    if n % 2 == 0:
```

```
        mid1 = sorted_arr[n // 2 - 1]
```

```
        mid2 = sorted_arr[n // 2]
```

```
        median_value = (mid1 + mid2) / 2
```

```
    else:
```

```
        median_value = sorted_arr[n // 2]
```

```
    return median_value
```

```
def mode_func(arr):
```

```
    return np.argmax(np.bincount(arr.flatten()))
```

```
print("Array 1:")
```

```
print("Median:", median_func(array1))
```

```
print("Mode:", mode_func(array1))
```

```
print("Array 2:")
```

```
print("Median:", median_func(array2))
```

```
print("Mode:", mode_func(array2))
```

```
print("Array 3:")
```

```
print("Median:", median_func(array3))
```

```
print("Mode:", mode_func(array3))
```

```

In [44]: #Question 12
array1 = np.array([[1, 2, 3], [4, 5, 6]])
array2 = np.array([[1, 2, 2], [3, 3, 4]])
array3 = np.array([1, 2, 2, 3, 4, 4, 4])

def median_func(arr):
    flattened_arr = arr.flatten()
    sorted_arr = np.sort(flattened_arr)
    n = len(sorted_arr)
    if n % 2 == 0:
        mid1 = sorted_arr[n // 2 - 1]
        mid2 = sorted_arr[n // 2]
        median_value = (mid1 + mid2) / 2
    else:
        median_value = sorted_arr[n // 2]
    return median_value

def mode_func(arr):
    return np.argmax(np.bincount(arr.flatten()))

print("Array 1:")
print("Median:", median_func(array1))
print("Mode:", mode_func(array1))

print("Array 2:")
print("Median:", median_func(array2))
print("Mode:", mode_func(array2))

print("Array 3:")
print("Median:", median_func(array3))
print("Mode:", mode_func(array3))

Array 1:
Median: 3.5
Mode: 1
Array 2:
Median: 2.5
Mode: 2
Array 3:
Median: 3
Mode: 4

```

Question 13:

```

#Question 13
def median_func(arr, axis=None):
    sorted_arr = np.sort(arr, axis=axis)

```

```

n = len(sorted_arr)
if n % 2 == 0:
    mid1 = sorted_arr[..., n // 2 - 1]
    mid2 = sorted_arr[..., n // 2]
    median_value = (mid1 + mid2) / 2
else:
    median_value = sorted_arr[..., n // 2]
return median_value

def mode_func(arr, axis=None):
    flattened_arr = arr.flatten()
    return np.argmax(np.bincount(flattened_arr))

array1 = np.array([[1, 2, 3], [4, 5, 6]])
array2 = np.array([[1, 2, 2], [3, 3, 4]])
array3 = np.array([1, 2, 2, 3, 4, 4, 4])

print("Array 1:")
print("Median (row-wise):", median_func(array1, axis=1))
print("Median (column-wise):", median_func(array1, axis=0))
print("Mode (row-wise):", mode_func(array1, axis=0))
print("Mode (column-wise):", mode_func(array1, axis=1))

print("Array 2:")
print("Median (row-wise):", median_func(array2, axis=1))
print("Median (column-wise):", median_func(array2, axis=0))
print("Mode (row-wise):", mode_func(array2, axis=0))
print("Mode (column-wise):", mode_func(array2, axis=1))

print("Array 3:")
print("Median (row-wise):", median_func(array3, axis=0))
print("Median (column-wise):", median_func(array2, axis=1))
print("Mode (row-wise):", mode_func(array3, axis=0))
print("Mode (column-wise):", mode_func(array3, axis=1))

```

```

Array 1:
Median (row-wise): [1.5 4.5]
Median (column-wise): [1.5 4.5]
Mode (row-wise): 1
Mode (column-wise): 1
Array 2:
Median (row-wise): [1.5 3. ]
Median (column-wise): [1.5 3. ]
Mode (row-wise): 2
Mode (column-wise): 2
Array 3:
Median (row-wise): 3
Median (column-wise): [1.5 3. ]
Mode (row-wise): 4
Mode (column-wise): 4

```

Explanation:

There are no sorting happening in mode so that's why mode is simple flattened list. On the other hand, being different collections of lists, the median varies.

Question 14

(A)

	0	1	2	3	4	5	6	7
0	48	22	35	12	46	24	37	2
1	34	11	47	23	36	1	44	25
2	21	49		55	13	57	3	38
3	10	33		58	k	45	26	43
4	80	20	54		56	14	39	4
5	32	9	59		53	27	42	15
6	19	51	7	30	17	40	5	28
7	8	31	18	52	6	29	16	41

(B)

#Question 14

#(A)

#Done on Paper, Added to the report

#(B)

squares = [x+1 for x in range(64)]

board = np.zeros((8,8), dtype=int)

current_row = 3

current_column = 4

x=1;

while True:

temp = str(current_column)+str(current_row)

if(int(temp) in squares):

squares.remove(int(temp))

board[current_row][current_column] = x

if current_column + 2 < 8 and current_row + 1 < 8 and (int(str(current_column + 2) +
str(current_row + 1)) in squares):

current_column +=2

current_row +=1

elif current_column +2 < 8 and current_row - 1 >= 0 and (int(str(current_column + 2) +
str(current_row - 1)) in squares):

current_column +=2

current_row -=1

elif current_column - 2 >= 0 and current_row - 1 >= 0 and (int(str(current_column - 2) +
str(current_row - 1)) in squares):

current_column -=2

current_row -=1

elif current_column -2 >= 0 and current_row + 1 < 8 and (int(str(current_column - 2) +
str(current_row + 1)) in squares):

current_column -=2

current_row +=1

elif current_column -1 >= 0 and current_row - 2 >= 0 and (int(str(current_column -1) +
str(current_row - 2)) in squares):

current_column -=1

current_row -=2

elif current_column -1 >= 0 and current_row + 2 < 8 and (int(str(current_column - 1) +
str(current_row + 2)) in squares):

current_column -=1

current_row +=2

```

    elif current_column + 1 < 8 and current_row + 2 < 8 and (int(str(current_column + 1) +
str(current_row + 2)) in squares):
        current_column += 1
        current_row += 2

    elif current_column + 1 < 8 and current_row - 2 >= 0 and (int(str(current_column + 1) +
str(current_row - 2)) in squares):
        current_column += 1
        current_row -= 2
    else:
        break

print('Move ',x)
print(board)
x+=1

```

```

Move 1
[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]]
Move 2
[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 2 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]]
Move 3

```

```
[ 0 0 21 16 0 0 0 0]]
```

```
Move 34
```

```
[[ 0 0 0 0 30 9 0 0]
```

```
[ 0 0 31 10 7 0 29 0]
```

```
[24 0 6 0 28 11 8 0]
```

```
[ 5 32 25 12 1 0 27 0]
```

```
[18 23 4 33 26 13 2 0]
```

```
[ 0 0 19 14 3 34 0 0]
```

```
[22 17 0 0 20 15 0 0]
```

```
[ 0 0 21 16 0 0 0 0]]
```

```
Move 35
```

```
[[ 0 0 0 0 30 9 0 0]
```

```
[ 0 0 31 10 7 0 29 0]
```

```
[24 0 6 0 28 11 8 0]
```

```
[ 5 32 25 12 1 0 27 0]
```

```
[18 23 4 33 26 13 2 0]
```

```
[ 0 0 19 14 3 34 0 0]
```

```
[22 17 0 35 20 15 0 0]
```

```
[ 0 0 21 16 0 0 0 0]]
```

#(C) NA