2D-ARRAYS & LOOPS & FUNCTIONS. The Tigercito

Document Check List (for the Ex#2 Portfolio):

	EX-1	EX-2	EX-5	EX-8	EX-9	EX-10
Code						
Output						

In these exercises you will practice input, output, and processing of two-dimensional arrays. Under the umbrella of processing you will learn to implement arrays in a programming solution (otherwise scalar) manipulate 2D array indexing, slicing, iteration (using loops, if), array creation (np.array, np.linspace, np.arange, np.ones, np.zeros, arrayname.fill, etc.), plot, and library functions accepting arrays (e.g., np.min,np.max,mp.mean, np.average, etc.),

For the following problems (1-6) in (A) use programming structures, e.g., loops and if statements. Afterwards in (B) use library functions.

Given two arrays, A and B (for Ex-1 through 6**)**

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$

Ex-1. Practicing with library functions: Write code to compute the sum of all elements on the 2D A-array.

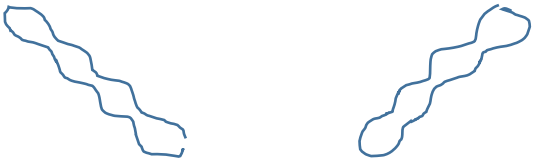
(A) Use loops
(B) Use the numpy.sum function.

Hint: Need to read a reference for the numpy.sum function. This function use several arguments and can sum (1) all the elements, (2) only rows, (3) only columns, etc. Combined with slicing you can sum portions of the array, also. Depending on the syntax.

Ex-2. Transfer the diagonal elements of A to the diagonal elements of B. Hint elements in the main diagonal have indices [i,j], such as, the condition i==j is true. Nested loops are appropriate for this problem

EX-3. Transfer the diagonal elements of A to the antidiagonal elements of B

EX-4. Multiply diagonal elements of A times antidiagonal elements of B, top down position-wise and store them as the new diagonal elements of B, e.g.,



$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$

Procedure: (1*13), (6*10), (11*7), (16*4)

$$B = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 60 & 14 \\ 3 & 77 & 11 & 15 \\ 64 & 8 & 12 & 16 \end{bmatrix}$$

(A) Use loops

(B) Explore if there is a numpy function to reference diagonal elements and use it.

EX-5. Multiply element-by-element the 3rd column of A times the 3rd row of B. As (3)(3), (7)(7), etc. Store the result in the column vector C (for python this is a 2D array).

(A) Use loops, can't use slicing

(B) Slicing is allowed

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix} \quad =>=> \quad C = \begin{bmatrix} 9 \\ 49 \\ \dots \\ \dots \end{bmatrix}$$

Ex-6. Triple the elements in the first column of B.

(A) Use loops, referencing element by element.

(B) Use slicing and element-wise operations

Ex-7. Initialize a 4x4 array named A, write code to verify which elements of A are divisible by A[0,0]. Report factors with a boolean array

$$A = \begin{bmatrix} 5 & 6 & 7 & 8 \\ 13 & 15 & 17 & 20 \\ 25 & 19 & 30 & 33 \\ 125 & 10 & 43 & 51 \end{bmatrix} ===> \begin{bmatrix} True & False & False & False \\ False & True & False & True \\ True & False & True & False \\ True & True & False & False \end{bmatrix}$$

Ex-8. Write code to delete the first and third column of the following 2D array:

$$A = \begin{bmatrix} 5 & 6 & 7 & 8 \\ 13 & 15 & 17 & 20 \\ 25 & 19 & 30 & 33 \\ 125 & 10 & 43 & 51 \end{bmatrix} ==> A = \begin{bmatrix} 6 & 8 \\ 15 & 20 \\ 19 & 33 \\ 10 & 51 \end{bmatrix}$$

Ex-9. Inspired by the piece of code below, which produces a 2D array named X with 5 rows whose elements are [1,2,3,4,5] in each row:

Code:	Output
# Initialize 2D array using loops: **Z=np.zeros((5,5),dtype=int)** **for i in range(0,5,1):** ** for j in range(0,5,1):** ** Z[i,j]=j+1** **print('Z=\n',Z)**	Z= [[1 2 3 4 5] [1 2 3 4 5] [1 2 3 4 5] [1 2 3 4 5] [1 2 3 4 5]]

Create a new 2D matrix named ZZ with the elements shown below. Use nested loops (as above). **Hint:** modified the code parameters above.

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

Ex-10. Inspired by the piece of code below, which produces a 2D array named X with 5 rows whose elements are [1,2,3,4,5] in each row:

Code	Output
# for-i parameters: **starti=1; stopi=6; stepi=1** **# for-j parameter** **startj=1; stopj=6; stepj=1** **# List Comprehension** **X=[[i for i in range(starti,stopi,stepi)] for j in range(startj,stopj,stepj)]** **print(X)** **# This construction is called Nested List Comprehension as one** **# loop is inside the other**	[[1, 2, 3, 4, 5], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]

Create a new 2D matrix named XX with the elements shown below. Use nested loops within List Comprehension as above. **Hint:** modified the code parameters above.

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

SOLUTION

Code	Output
starti=1; stopi=5; stepi=1 startj=1; stopj=31; stepj=5 k=0 MM=[[i for i in range(j,stopi+j,stepi)] for j in range(startj,stopj,stepj)] print(MM)	[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20], [21, 22, 23, 24, 25], [26, 27, 28, 29, 30]]