WORKSHEET 1

4 TO - DO - Task

Please complete all the problem listed below.

4.1 Warming Up Exercise: Basic Vector and Matrix Operation with Numpy.

Problem - 1: Array Creation:

Complete the following Tasks:

1. Initialize an empty array with size 2X2.

=>

Problem - 1: Array Creation:

```
#Initialize an empty array with size 2X2.
import numpy as np
empty_array = np.empty((2,2))
print(empty_array)
```

```
[[4.73430977e-310 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000]]
```

2. Initialize an all one array with size 4X2.

=>

```
#Initialize an all one array with size 4X2.
import numpy as np
one_array = np.ones((4,2))
print(one_array)
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
```

3. Return a new array of given shape and type, filled with fill value.{Hint: np.full}

=>

```
#Return a new array of given shape and type, filled with fill value.{Hint: np.full}
import numpy as np
fill_value = np.full((5,4),6)
print(fill_value)
```

```
[[6 6 6 6]
 [6 6 6 6]
 [6 6 6 6]
 [6 6 6 6]
 [6 6 6 6]]
```

4. Return a new array of zeros with same shape and type as a given array.{Hint: np.zeros like}

=>

```
#Return a new array of zeros with same shape and type as a given array.{Hint: np.zeros like}
import numpy as np
zero_array = np.zeros_like(fill_value)
print(zero_array)
```

```
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

5. Return a new array of ones with same shape and type as a given array.{Hint: np.ones like}

=>

```
#Return a new array of ones with same shape and type as a given array.{Hint: np.ones like}
import numpy as np
one_array = np.ones_like(zero_array)
print(one_array)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

6. For an existing list new_list = [1,2,3,4] convert to an numpy array.{Hint: np.array()}

=>

```
#For an existing list new_list = [1,2,3,4] convert to an numpy array.{Hint: np.array()}
import numpy as np
new_list = [1,2,3,4]
new_array = np.array(new_list)
print(new_array)
```

```
[1 2 3 4]
```

4.1.1 Problem - 2: Array Manipulation: Numerical Ranges and Array indexing:

Complete the following tasks:

1. Create an array with values ranging from 10 to 49. {Hint:np.arrange()}.

=>

Problem - 2: Array Manipulation: Numerical Ranges and Array indexing:

```
#Create an array with values ranging from 10 to 49. {Hint:np.arrange()}.
import numpy as np
array_arrange = np.arange(10,49)
print(array_arrange)
```

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48]
```

2. Create a 3X3 matrix with values ranging from 0 to 8.

{Hint:look for np.reshape()}

=>

```
#Create a 3X3 matrix with values ranging from 0 to 8.{Hint:look for np.reshape()}
import numpy as np
array_shape = np.arange(0,9).reshape(3,3)
print(array_shape)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

3. Create a 3X3 identity matrix.{Hint:np.eye()}

```
#Create a 3X3 identity matrix.{Hint:np.eye()}
import numpy as np
array_eye = np.eye(3,3)
print(array_eye)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

=>

4. Create a random array of size 30 and find the mean of the array.

{Hint:check for np.random.random() and array.mean() function}

=>

```
#Create a random array of size 30 and find the mean of the array. {Hint:check for np.random.random() and array.mean() function}
import numpy as np
random_array = np.random.random(30)
mean_array = random_array.mean()
print(mean_array)
```

```
0.5507252675633549
```

5. Create a 10X10 array with random values and find the minimum and maximum values.

=>

```
#Create a 10X10 array with random values and find the minimum and maximum values.
import numpy as np
random_array = np.random.random((10,10))
min = random_array.min()
max = random_array.max()
print(min)
print(max)
```

```
0.007780011025022682
0.9939738330578829
```

6. Create a zero array of size 10 and replace 5th element with 1.

=>

```
#Create a zero array of size 10 and replace 5th element with 1.
import numpy as np
zero_array = np.zeros(10)
zero_array[4] = 1
print(zero_array)
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

7. Reverse an array arr = [1,2,0,0,4,0].

```
#Reverse an array arr = [1,2,0,0,4,0].
import numpy as np
arr = np.array([1,2,0,0,4,0])
arr = arr[::-1]
print(arr)
```

```
[0 4 0 0 2 1]
```

=>

8. Create a 2d array with 1 on border and 0 inside.

=>

```
#Create a 2d array with 1 on border and 0 inside.
import numpy as np
rows,cols = 5,6
arr = np.ones((rows,cols))
arr[1:-1,1:-1] = 0
print(arr)
```

```
[[1. 1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1. 1.]]
```

9. Create a 8X8 matrix and fill it with a checkerboard pattern.

=>

```
#Create a 8X8 matrix and fill it with a checkerboard pattern.
import numpy as np
n = 8
arr = np.zeros((n,n))
arr[::2,::2] = 1
arr[1::2,1::2] = 1
print(arr)
```

```
[[1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]]
```

Problem - 3: Array Operations:

For the following arrays:

x = np.array([[1,2],[3,5]]) and y = np.array([[5,6],[7,8]]);

v = np.array([9,10]) and w = np.array([11,12]);

Complete all the task using numpy:

1. Add the two array.

2. Subtract the two array.

3. Multiply the array with any integers of your choice.

4. Find the square of each element of the array.

5. Find the dot product between: v(and)w ; x(and)v ; x(and)y.

=>

```
#Problem - 3: Array Operations:
import numpy as np
x = np.array([[1,2],[3,5]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([9,10])

#ADD
add = x + y
print("Addition between x and y",add)

#Subtract
sub = v - w
print("Difference of v and w",sub)

#multiply
multiply = 9 * x
print("x multiplied by 9",multiply)

#Square
sq = np.square(x)
print("Square of x",sq)

#Dot product
vwpro = np.dot(v,w)
xvpro = np.dot(x,v)
xypro = np.dot(x,y)

print("Dot product of v and w",vwpro)
print("Dot product of x and v",xvpro)
print("Dot product of x and y",xypro)
```

```
Addition between x and y [[ 6  8]
 [10 13]]
Difference of v and w [0 0]
x multiplied by 9 [[ 9 18]
 [27 45]]
Square of x [[ 1  4]
 [ 9 25]]
Dot product of v and w 181
Dot product of x and v [29 77]
Dot product of x and y [[19 22]
 [50 58]]
```

6. Concatenate x(and)y along row and Concatenate v(and)w along column.

{Hint:try np.concatenate() or np.vstack() functions.

7. Concatenate x(and)v; if you get an error, observe and explain why did you get the error?

=>

6 & 7 ANS

```
#Concatenate along row
row_concat = np.concatenate((x,y))
print("Concatenate along row",row_concat)

#Concatenate along column
col_concat = np.vstack((v,w))
print("Concatenate along col",col_concat)

#Concatenate x and v
xv_con = np.concatenate((x,v))
print("Concatenate x and v",xv_con)
#x and v cannot be concatenated because they are of different diensions. Only array with same dimension can be concatenated.
```

```
Concatenate along row [[1 2]
 [3 5]
 [5 6]
 [7 8]]
Concatenate along col [[ 9 10]
 [ 9 10]]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-08a957561dac> in <cell line: 10>()
      8
      9 #Concatenate x and v
---> 10 xv_con = np.concatenate((x,v))
     11 print("Concatenate x and v",xv_con)
     12 #x and v cannot be concatenated because they are of different diensions. Only array with same dimension can be concatenated.

ValueError: all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimension(s) and the array at index
```

Problem - 4: Matrix Operations:

• For the following arrays:

A = np.array([[3,4],[7,8]]) and B = np.array([[5,3],[2,1]]);

Prove following with Numpy:

1. Prove A.A−1 = I.

```
[21] #Problem - 4: Matrix Operations:
     import numpy as np
     a = np.array([[3,4],[7,8]])
     b = np.array([[5,3],[2,1]])

     a_inv = np.linalg.inv(a)
     identity = np.dot(a,a_inv)
     print(identity)
```

```
[[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]
```

=>

2. Prove AB≠ BA.

```
24] #ab != ba
    import numpy as np
    ab_dot = np.dot(a,b)

    ba_dot = np.dot(b,a)
    if not np.array_equal(ab_dot, ba_dot):
      print("True")
```

```
True
```

=>

3. Prove (AB)

T = BTAT

```
#(AB)tran == (b)tran.(a)tran
import numpy as np
abdot_tran = np.transpose(ab_dot)
b_tran = np.transpose(b)
a_tran = np.transpose(a)
abtran_dot = np.dot(b_tran, a_tran)
if np.array_equal(abdot_tran, abtran_dot):
  print("Equal")
```

```
Equal
```

.=>

• Solve the following system of Linear equation using Inverse Methods.

2x − 3y + z = −1

x − y + 2z = −3

3x + y − z = 9

{Hint: First use Numpy array to represent the equation in Matrix form. Then Solve for: AX = B}

• Now: solve the above equation using np.linalg.inv function.{Explore more about ”linalg” function

of Numpy}

=>

```
#equation
a = np.array([[2,-3,1],[1,-1,2],[3,1,-1]])
b = np.array([-1,-3,9])
a_inv = np.linalg.inv(a)
x = np.dot(a,a_inv)
print("Solution is:",x)

#Using linalg
x = np.linalg.solve(a,b)
print("Solution using linalg:",x)
```

```
Solution is: [[ 1.00000000e+00 -1.11022302e-16 -8.32667268e-17]
 [ 0.00000000e+00  1.00000000e+00 -2.77555756e-17]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
Solution using linalg: [ 2.  1. -2.]
```