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.

```
[ ]
arr = np.empty((2, 2))
print(arr)
```

```
[[9.74109714e-316 0.00000000e+000]
 [1.48219694e-323             nan]]
```

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

```
[ ]
np.ones((4,2))
```

```
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}

```
25]
0s
np.full((2, 3), 7)
```

```
array([[7, 7, 7],
       [7, 7, 7]])
```

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

```
26]
0s
a = np.array([[1, 2, 3], [4, 5, 6]])
np.zeros_like(a)
```

```
array([[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}

```
27]
0s
np.ones_like(a)
```

```
array([[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()}

+ Code + Text

```
[28]
0s
a = np.array([1,2,3,4])
a

array([1, 2, 3, 4])
```

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()}.

```
[29]
0s
np.arange(10,49)

array([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()}

```
[30]
0s
arr = np.array([0,1,2,3,4,5,6,7,8])
arr.reshape(3,3)

array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

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

```
[31]
✓ 0s
np.identity(3)

array([[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}

```
[32]
✓ 0s
arr = np.random.randint(0,100, size = 30)
print(arr)
arr.mean()

[57 78 85 40 38  2 41 37 32 91 66 76 87 48 52  2  1  9 67 12 28 52 46 29
 42  7 67 27 92 84]
np.float64(46.5)
```

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

```
arr = np.random.randint(0,100, size = (10,10))
print(arr)
print(arr.min())
print(arr.max())
```

```
[[13 45 71 76 40 79 83 17 35 26]
 [26 28 94 87 15 75 30 45 44 19]
 [11 16 85 22 44 87 49 88 16 61]
 [35 45 19 36 41 96 43 13 38 12]
 [17 11 99 39 31 58 96 40 19  4]
 [19 35 41 73 88 12  4 92 31 20]
 [52 28 34 27 51 34 91 73 46 10]
 [69 46 31 13 42 83 71 69 82 45]
 [75  3 72 23 37 19  7 48 42 49]
 [63 11 45 49 73 77 26 38  4 15]]
3
99
```

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

[34] ✓ 0s

```
arr = np.zeros(10)
arr[4] = 1
arr
```

```
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

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

[35] ✓ 0s

```
arr = [1,2,0,0,4,0]
arr.reverse()
arr
```

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

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

[36] ✓ 0s

```
arr = np.zeros((5,5))
arr[0, :] = 1
arr[:, 0] = 1
arr[-1, :] = 1
arr[:, -1] = 1
print(arr)
```

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

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

[37] ✓ 0s

```
arr = np.zeros((8,8))
for x in range(8):
    for y in range(8):
      if (x+y) % 2 == 0:
        arr[x,y] = 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.

[59] ✓ 0s

```
x = np.array([[1,2],[3,5]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11,12])

print(x+y)
```

```
[[ 6  8]
 [10 13]]
```

2. Subtract the two array.

[39] ✓ 0s

```
print(x-y)
```

```
[[-4 -4]
 [-4 -3]]
```

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

[40] ✓ 0s

```
print(2*x)
```

```
[[ 2  4]
 [ 6 10]]
```

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

[41] ✓ 0s

```
print(np.square(x))
```

```
[[ 1  4]
 [ 9 25]]
```

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

[42] ✓ 0s

```
print(np.dot(v,w))
print("\n")
print(np.dot(x,v))
print("\n")
print(np.dot(x,y))
```

```
219
```

6. Concatenate x(and)y along row and Concatenate v(and)w along column. {Hint:try np.concatenate() or np.vstack() functions.

[43] ✓ 0s

```
print(np.concatenate((x,y), axis = 1))
print("\n")
print(np.vstack((x,y)))
```

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


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

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

[60]

```
np.concatenate((x, v))
# all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimension(s)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-2205799076.py in <cell line: 0>()
----> 1 np.concatenate((x, v))

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 1 ha
dimension(s)
```

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.

[45] ✓ 0s

```
A = np.array([[3,4],[7,8]])
B = np.array([[5,3],[2,1]])
np.dot(A, np.linalg.inv(A))
```

[45] ✓ 0s

```
B = np.array([[5,3],[2,1]])
np.dot(A, np.linalg.inv(A))
```

```
array([[1.00000000e+00, 0.00000000e+00],
       [1.77635684e-15, 1.00000000e+00]])
```

2. Prove AB ≠ BA.

[46] ✓ 0s

```
A = np.array([[3,4],[7,8]])
B = np.array([[5,3],[2,1]])
if np.array_equal(np.dot(A, B), np.dot(B, A)):
    print("False")
else:
    print("True")
```

```
True
```

3. Prove (AB)^T = B^T A^T

[47] ✓ 0s

```
np.array_equal(np.transpose(np.dot(A,B)), np.dot(np.transpose(B), np.transpose(A)))
```

```
True
```

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

$2x - 3y + z = -1$ $x - y + 2z = -3$ $3x + y - z = 9$

• Now: solve the above equation using np.linalg.inv function.{Explore more about "linalg" function of Numpy}

[48] ✓ 0s

```
A = np.array([[2,-3,1],[1,-1,2],[3,1,-1]])
B = np.array([-1,-3,9])

inv_A = np.linalg.inv(A)
print(np.dot(inv_A, B))
```

```
[ 2.  1. -2.]
```

4.2 Experiment: How Fast is Numpy? In this exercise, you will compare the performance and implementation of operations using plain Python lists (arrays) and NumPy arrays. Follow the instructions:

1. Element-wise Addition: • Using Python Lists, perform element-wise addition of two lists of size 1, 000, 000. Measure and Print the time taken for this operation.

[49]

```
A = np.random.randint(1,1000,1000000)
B = np.random.randint(1,1000,1000000)
Sum = 0

for x in A:
  for y in B:
    Sum += x+y
print(Sum)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-347559158.py in <cell line: 0>()
      5 for x in A:
      6   for y in B:
----> 7     Sum += x+y
      8 print(Sum)
      9

KeyboardInterrupt:
```

• Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.

```
A = np.random.randint(1,1000,1000000)
B = np.random.randint(1,1000,1000000)
```

```
[50] ✓ 0s
A = np.random.randint(1,1000,1000000)
B = np.random.randint(1,1000,1000000)

np.sum(A)+np.sum(B)

np.int64(999916163)
```

2. Element-wise Multiplication • Using Python Lists, perform element-wise multiplication of two lists of size 1, 000, 000. Mea- sure and Print the time taken for this operation.

```
[51] ✓ 0s
size = 1_000_000
list1 = [i for i in range(size)]
list2 = [i for i in range(size)]

start = time.time()
list_result = [list1[i] * list2[i] for i in range(size)]
end = time.time()


print("Python List Time:", end - start, "seconds")

Python List Time: 0.11239123344421387 seconds
```

• Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.

```
[52] ✓ 0s
arr1 = np.arange(size)
arr2 = np.arange(size)

start = time.time()
arr_result = arr1 * arr2
end = time.time()

print("NumPy Array Time:", end - start, "seconds")

NumPy Array Time: 0.0026235580444335938 seconds
```

3. Dot Product • Using Python Lists, compute the dot product of two lists of size 1, 000, 000. Measure and Print the time taken for this operation.

```
[53] ✓ 0s
size = 1_000_000
list1 = [i for i in range(size)]
list2 = [i for i in range(size)]

start = time.time()
dot_list = sum(list1[i] * list2[i] for i in range(size))
end = time.time()

✓ 0s
print("Python List Dot Product Time:", end - start, "seconds")

••• Python List Dot Product Time: 0.11751842498779297 seconds
```

• Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this operation.

```
[54] ✓ 0s
arr1 = np.arange(size)
arr2 = np.arange(size)

start = time.time()
dot_numpy = np.dot(arr1, arr2)
end = time.time()

print("NumPy Dot Product Time:", end - start, "seconds")

NumPy Dot Product Time: 0.002943754196166992 seconds
```

4. Matrix Multiplication • Using Python lists, perform matrix multiplication of two matrices of size 1000x1000. Measure and print the time taken for this operation.

```
size = 1000

A = [[1 for _ in range(size)] for _ in range(size)]
B = [[1 for _ in range(size)] for _ in range(size)]

start = time.time()

C = [[0] * size for _ in range(size)]
for i in range(size):
    for j in range(size):
        total = 0
        for k in range(size):
            total += A[i][k] * B[k][j]
        C[i][j] = total

end = time.time()

print("Python List Matrix Multiplication Time:", end - start, "seconds")
```

• Using NumPy arrays, perform matrix multiplication of two matrices of size 1000x1000. Mea- sure and print the time taken for this operation.

```
A_np = np.ones((size, size))
B_np = np.ones((size, size))

start = time.time()
C_np = np.dot(A_np, B_np)   # or A_np @ B_np
end = time.time()

print("NumPy Matrix Multiplication Time:", end - start, "seconds")
```

```
NumPy Matrix Multiplication Time: 0.05902910232543945 seconds
```