Unnat Jung Thapa

Problem 1:

```
import numpy as np

# 1. Initialize an empty array with size 2X2
empty_array = np.empty((2, 2))
print("1. Empty Array (2x2):")
print(empty_array)
```

```
1. Empty Array (2x2):
[[2.45355258e-315 0.00000000e+000]
 [6.65347456e-310 5.77904633e-317]]
```

```
# 2. Initialize an all-one array with size 4X2
ones_array = np.ones((4, 2))
print("\n2. All-Ones Array (4x2):")
print(ones_array)
```

```

2. All-Ones Array (4x2):
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
```

```
# 3. New array filled with a given value (example: shape 3x3, fill value = 7)
full_array = np.full((3, 3), 7)
print("\n3. Full Array (3x3 filled with 7):")
print(full_array)
```

```
3. Full Array (3x3 filled with 7):
[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

```
given_array = np.array([[5, 6, 7], [8, 9, 10]])
zeros_like_array = np.zeros_like(given_array)
print("\n4. Zeros Like Given Array:")
print(zeros_like_array)

# 5. Ones array with same shape and type as a given array
ones_like_array = np.ones_like(given_array)
print("\n5. Ones Like Given Array:")
print(ones_like_array)
```

```
...
4. Zeros Like Given Array:
[[0 0 0]
 [0 0 0]]

5. Ones Like Given Array:
[[1 1 1]
 [1 1 1]]
```

```
new_list = [1, 2, 3, 4]
numpy_array = np.array(new_list)
print("\n6. Converted List to Numpy Array:")
print(numpy_array)
```

```
...
6. Converted List to Numpy Array:
[1 2 3 4]
```

\

Problem 2:

```
#problem 2

import numpy as np

# 1. Create an array with values from 10 to 49
arr1 = np.arange(10, 50)
print("1. Array from 10 to 49:\n", arr1)
```

```
1. Array from 10 to 49:
 [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 49]
```

```
# 2. Create a 3x3 matrix with values 0-8
arr2 = np.arange(9).reshape(3, 3)
print("\n2. 3x3 Matrix 0-8:\n", arr2)
```

```
2. 3x3 Matrix 0-8:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
```

+ Code + Text

```
# 3. Create a 3x3 identity matrix
identity = np.eye(3)
print("\n3. 3x3 Identity Matrix:\n", identity)
```

```
3. 3x3 Identity Matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
# 4. Random array of size 30 and find the mean
rand_arr = np.random.random(30)
print("\n4. Random Array Mean:\n", rand_arr.mean())
```

```
4. Random Array Mean:
 0.512974328556917
```

```python
# 5. 10x10 array with random values → min & max
arr5 = np.random.random((10, 10))
print("\n5. Min:", arr5.min(), " | Max:", arr5.max())
```

```
5. Min: 0.015032182682892081  | Max: 0.9954049980145323
```

```python
# 6. Zero array (size 10) → replace 5th element with 1
arr6 = np.zeros(10)
arr6[4] = 1   # index starts at 0
print("\n6. Replace 5th element with 1:\n", arr6)
```

```
...
6. Replace 5th element with 1:
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

```python
# 7. Reverse an array
arr7 = np.array([1, 2, 0, 0, 4, 0])
reversed_arr = arr7[::-1]
print("\n7. Reversed Array:\n", reversed_arr)
```

```
7. Reversed Array:
 [0 4 0 0 2 1]
```

```python
# 8. 2D array with 1 on border and 0 inside
arr8 = np.ones((5, 5))
arr8[1:-1, 1:-1] = 0
print("\n8. Border 1, Inside 0:\n", arr8)
```

```
...
8. Border 1, Inside 0:
 [[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

```python
# 9. 8x8 checkerboard pattern
checkerboard = np.zeros((8, 8), dtype=int)
checkerboard[1::2, ::2] = 1
checkerboard[::2, 1::2] = 1
print("\n9. Checkerboard Pattern:\n", checkerboard)
```

```
9. Checkerboard Pattern:
 [[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]
 [1 0 1 0 1 0 1 0]]
```

Problem 3:

```python
# Given arrays
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("Array x:\n", x)
print("\nArray y:\n", y)
print("\nVector v:", v)
print("\nVector w:", w)

# 1. Add two arrays
add_xy = x + y
print("\n1. x + y:\n", add_xy)
```

```
Array x:
 [[1 2]
 [3 5]]

Array y:
 [[5 6]
 [7 8]]

Vector v: [ 9 10]

Vector w: [11 12]

1. x + y:
 [[ 6  8]
 [10 13]]
```

```python
# 2. Subtract two arrays
sub_xy = x - y
print("\n2. x - y:\n", sub_xy)
```

```
2. x - y:
 [[-4 -4]
 [-4 -3]]
```

```python
# 3. Multiply array x by integer (example: 3)
mul_x = x * 3
print("\n3. x * 3:\n", mul_x)
```

```
3. x * 3:
 [[ 3  6]
 [ 9 15]]
```

```python
# 4. Square of each element of x
square_x = x ** 2
print("\n4. Square of elements of x:\n", square_x)
```

```
4. Square of elements of x:
 [[ 1  4]
 [ 9 25]]
```

```python
# 5. Dot products
dot_vw = np.dot(v, w)
dot_xv = np.dot(x, v)
dot_xy = np.dot(x, y)
print("\n5. Dot Products:")
print("v · w =", dot_vw)
print("x · v =\n", dot_xv)
print("x · y =\n", dot_xy)
```

```
5. Dot Products:
v · w = 219
x · v =
 [29 77]
x · y =
 [[19 22]
 [50 58]]
```

```python
# 6. Concatenate x & y along rows, v & w along columns
concat_xy = np.concatenate((x, y), axis=0)
concat_vw = np.vstack((v, w))
print("\n6. Concatenate x & y (rows):\n", concat_xy)
print("\nConcatenate v & w (columns):\n", concat_vw)
```

```
6. Concatenate x & y (rows):
 [[1 2]
 [3 5]
 [5 6]
 [7 8]]

Concatenate v & w (columns):
 [[ 9 10]
 [11 12]]
```

```python
# 7. Concatenate x and v (Explain error)
try:
    concat_xv = np.concatenate((x, v))
    print("\n7. Concatenate x & v:\n", concat_xv)
except Exception as e:
    print("\n7. ERROR while concatenating x & v:")
    print(str(e))
    print("\n📌 Reason: x is 2D (2x2 matrix) but v is 1D (2 elements).")
    print("Their shapes don't match → Cannot concatenate directly without reshaping v.")
```

```
7. ERROR while concatenating x & v:
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 has 1 dimension(s)

📌 Reason: x is 2D (2x2 matrix) but v is 1D (2 elements).
Their shapes don't match → Cannot concatenate directly without reshaping v.
```

Problem 4:

```python
#problem 4

import numpy as np

# Given matrices
A = np.array([[3, 4], [7, 8]])
B = np.array([[5, 3], [2, 1]])

print("Matrix A:\n", A)
print("\nMatrix B:\n", B)

# 1. Prove A * A⁻¹ = I
A_inv = np.linalg.inv(A)
I = np.dot(A, A_inv)
print("\n1. A * A⁻¹ = I:\n", I)
```

```
Matrix A:
 [[3 4]
 [7 8]]

Matrix B:
 [[5 3]
 [2 1]]

1. A * A⁻¹ = I:
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]
```

```python
# 2. Prove AB ≠ BA
AB = np.dot(A, B)
BA = np.dot(B, A)
print("\n2. AB:\n", AB)
print("\nBA:\n", BA)
print("\nSince AB ≠ BA → Matrix multiplication is NOT commutative.")
```

```
2. AB:
 [[23 13]
 [51 29]]

BA:
 [[36 44]
 [13 16]]

Since AB ≠ BA → Matrix multiplication is NOT commutative.
```

```python
# 3. Prove (AB)ᵀ = BᵀAᵀ
AB = np.dot(A, B)
BA = np.dot(B, A)
LHS = (AB).T        # Transpose of AB
RHS = np.dot(B.T, A.T)  # BᵀAᵀ
print("\n3. (AB)ᵀ:\n", LHS)
print("\nBᵀAᵀ:\n", RHS)
print("\nTherefore, (AB)ᵀ = BᵀAᵀ ✓")
```

```
3. (AB)ᵀ:
 [[23 51]
 [13 29]]

BᵀAᵀ:
 [[23 51]
 [13 29]]

Therefore, (AB)ᵀ = BᵀAᵀ ✓
```

Solve Using Inverse:

```python
# SOlve using inverse
# Coefficient matrix A and constants matrix B
A2 = np.array([[2, -3, 1],
               [1, -1, 2],
               [3, 1, -1]])

B2 = np.array([-1, -3, 9])

# Using inverse method → X = A⁻¹B
A2_inv = np.linalg.inv(A2)
solution = np.dot(A2_inv, B2)

print("\nSolution using Inverse Method:")
print("x, y, z =", solution)
```

```
Solution using Inverse Method:
x, y, z = [ 2.  1. -2.]
```

SOlve Using DIrect Method:

```python
#Solve using np.linalg.solve() (Direct Method)

solution2 = np.linalg.solve(A2, B2)
print("\nSolution using numpy solve():")
print("x, y, z =", solution2)
```

```
Solution using numpy solve():
x, y, z = [ 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.
• Using Numpy Arrays, Repeat the calculation and measure and print the time taken for this Operation.

```python
import numpy as np
import time

 #Task 1: Element-wise Addition

# Python List
list1 = list(range(1_000_000))
list2 = list(range(1_000_000))
start = time.time()
result_list_add = [list1[i] + list2[i] for i in range(1_000_000)]
end = time.time()
print("1. Python List Addition Time:", end - start, "seconds")

# NumPy Array
arr1 = np.arange(1_000_000)#
arr2 = np.arange(1_000_000)
start = time.time()
result_np_add = arr1 + arr2
end = time.time()
print("1. NumPy Array Addition Time:", end - start, "seconds")
```

```
1. Python List Addition Time: 0.203108549118042 seconds
1. NumPy Array Addition Time: 0.00829625129699707 seconds
```

2. Element-wise Multiplication

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

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

Operation.

```
#  Task 2: Element-wise Multiplication

# Python List
start = time.time()
result_list_mul = [list1[i] * list2[i] for i in range(1_000_000)]
end = time.time()
print("\n2. Python List Multiplication Time:", end - start, "seconds")

# NumPy Array
start = time.time()
result_np_mul = arr1 * arr2
end = time.time()
print("2. NumPy Array Multiplication Time:", end - start, "seconds")
```

```
2. Python List Multiplication Time: 0.3114616870880127 seconds
2. NumPy Array Multiplication Time: 0.005810976028442383 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.

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

```
# --------------------- Task 3: Dot Product ---------------------

# Python List
start = time.time()
dot_list = sum(list1[i] * list2[i] for i in range(1_000_000))
end = time.time()
print("\n3. Python List Dot Product Time:", end - start, "seconds")

# NumPy Array
start = time.time()
dot_np = np.dot(arr1, arr2)
end = time.time()
print("3. NumPy Array Dot Product Time:", end - start, "seconds")
```

```
3. Python List Dot Product Time: 0.11272192001342773 seconds
3. NumPy Array Dot Product Time: 0.001867532730102539 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.

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

```python
#--------------------- Task 4: Matrix Multiplication ---------------------

# Python List Matrix
mat_size = 300  # 🚨 Set to 1000x1000 only if your PC is powerful → 300 avoids freezing in class laptops.

A_list = [[1]*mat_size for _ in range(mat_size)]
B_list = [[1]*mat_size for _ in range(mat_size)]

start = time.time()
result_matrix_list = [[sum(A_list[i][k] * B_list[k][j] for k in range(mat_size))
                       for j in range(mat_size)] for i in range(mat_size)]
end = time.time()
print("\n4. Python List Matrix Multiplication Time:", end - start, "seconds")

# NumPy Array Matrix
A_np = np.ones((mat_size, mat_size))
B_np = np.ones((mat_size, mat_size))

start = time.time()
result_matrix_np = np.dot(A_np, B_np)
end = time.time()
print("4. NumPy Array Matrix Multiplication Time:", end - start, "seconds")
```

```
4. Python List Matrix Multiplication Time: 2.8233327865600586 seconds
4. NumPy Array Matrix Multiplication Time: 0.002578258514404297 seconds
```