4 TO - DO - Task

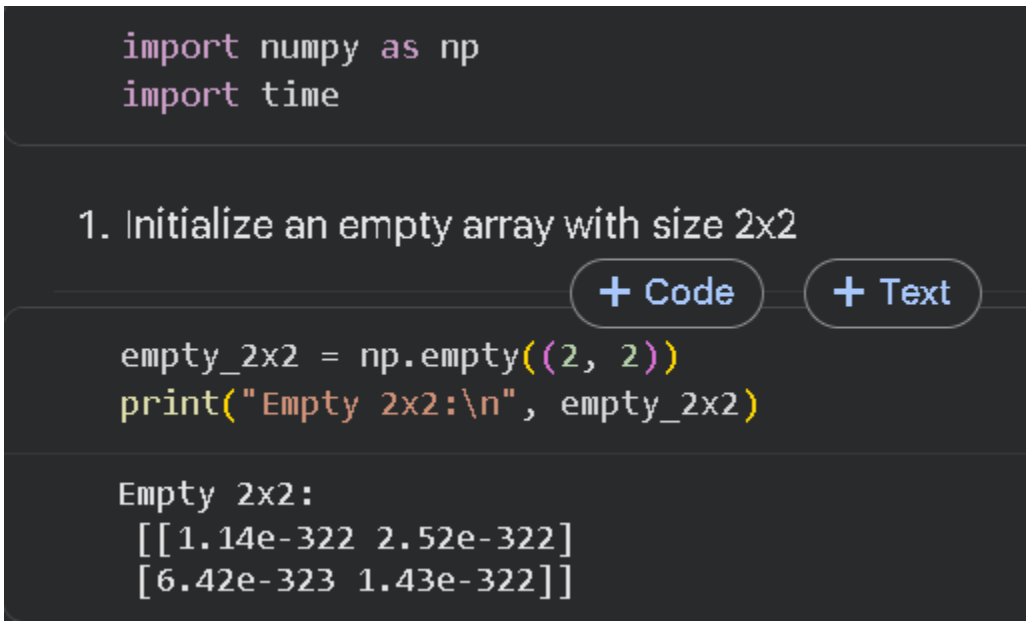
Please complete all the problem listed below.

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

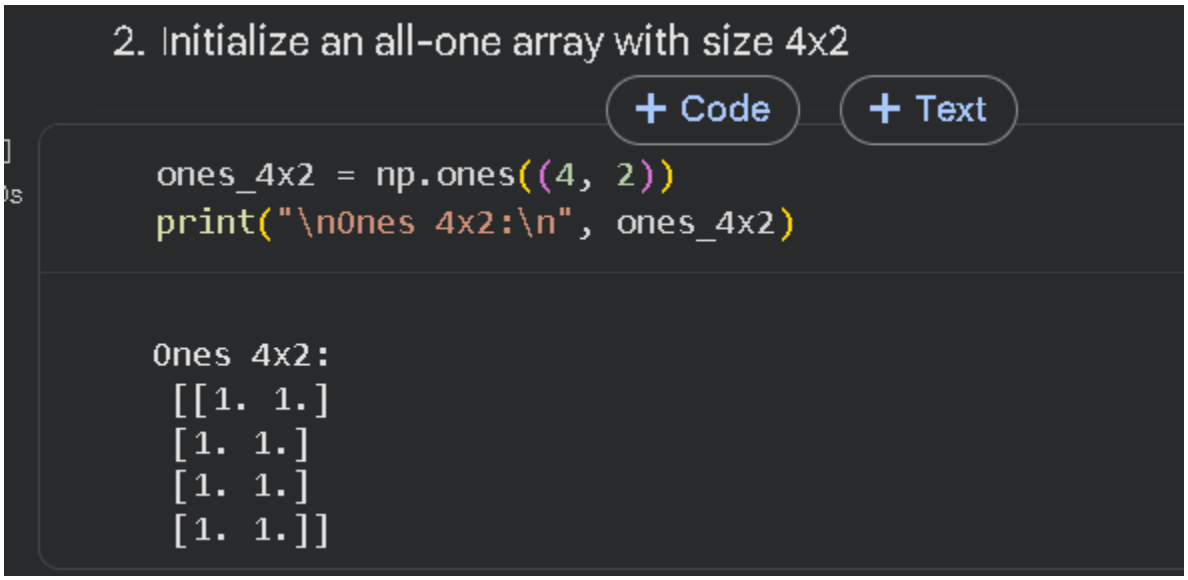
Problem - 1: Array Creation:

Complete the following Tasks:

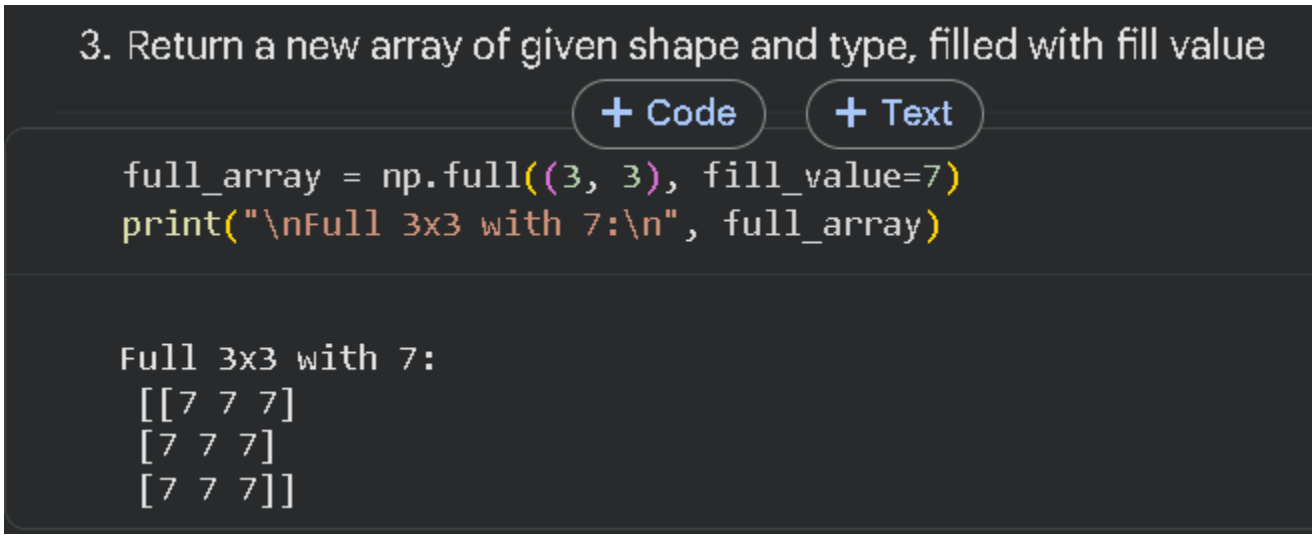
1. Initialize an empty array with size 2*X*2.



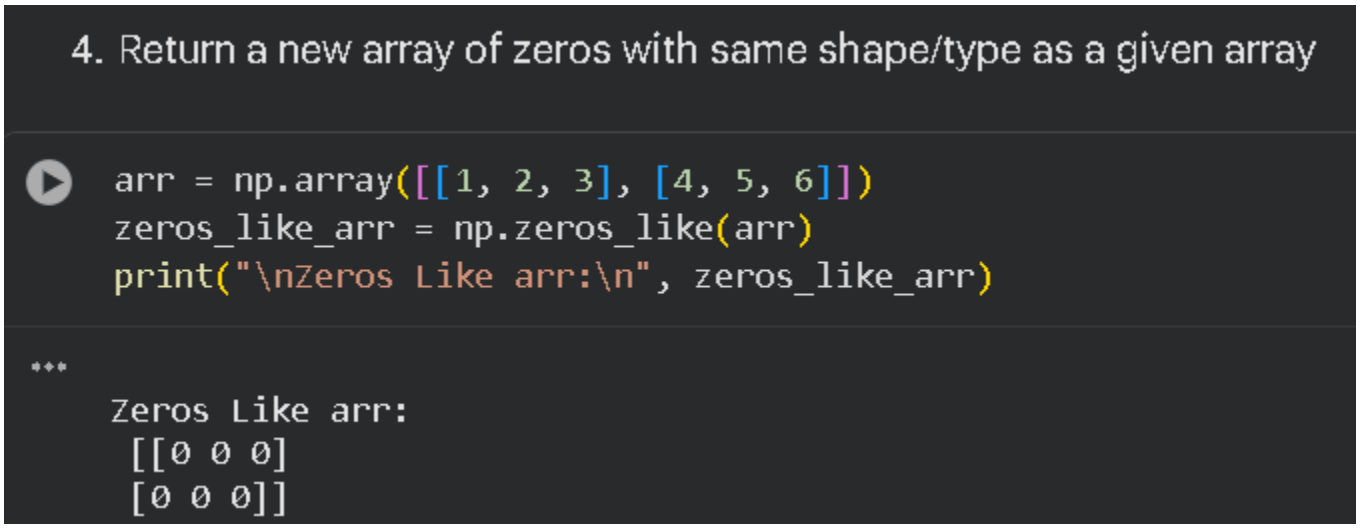
2. Initialize an all one array with size 4*X*2.



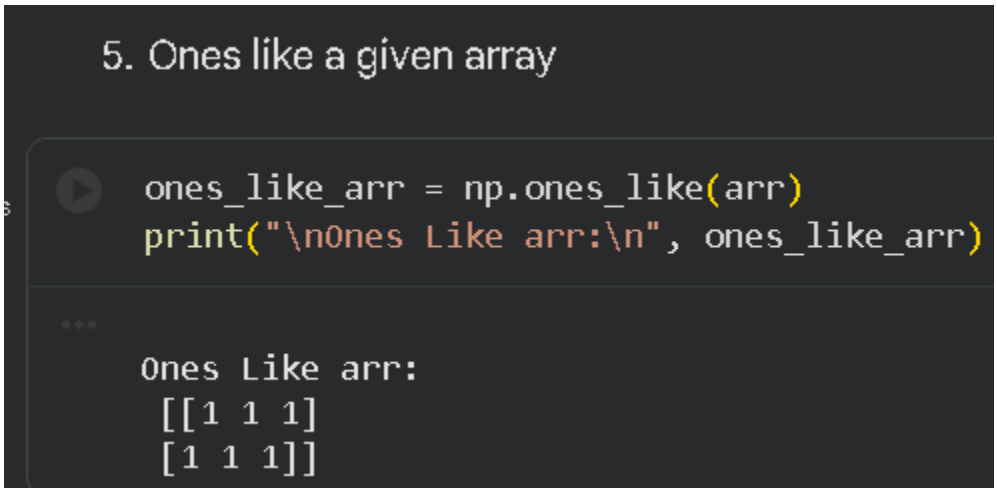
3. Return a new array of given shape and type, filled with fill value.{Hint: np.full}



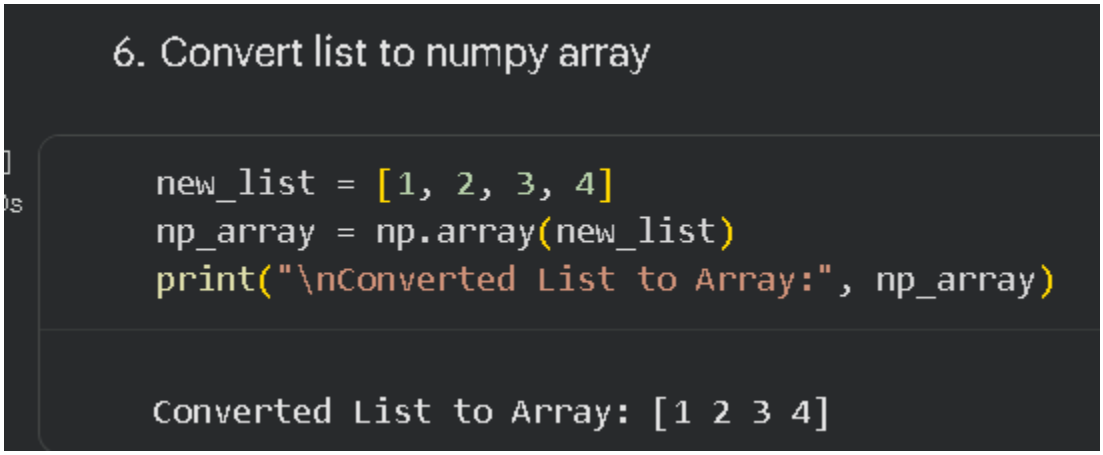
4. Return a new array of zeros with same shape and type as a given array.{Hint: np.zeros like}



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



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



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()}.

```
# 1. Array from 10 to 49
arr_10_49 = np.arange(10, 50)
print("10 to 49:\n", arr_10_49)
```

```
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
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]
```

2. Create a 3*X*3 matrix with values ranging from 0 to 8.
   {Hint:look for np.reshape()}

```
# 2. 3x3 matrix with values 0-8
matrix_0_8 = np.arange(9).reshape(3, 3)
print("\n3x3 0-8:\n", matrix_0_8)
```

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

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

```
# 3. 3x3 identity matrix
identity_3 = np.eye(3)
print("\nIdentity 3x3:\n", identity_3)
```

```
Identity 3x3:
 [[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}

```
# 4. Random array of size 30 and mean
rand_30 = np.random.random(30)
print("\nMean of random array:", rand_30.mean())
```

```
Mean of random array: 0.5363943296673173
```

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

```
# 5. 10x10 random array
rand_10x10 = np.random.random((10, 10))
print("\nMin:", rand_10x10.min(), " Max:", rand_10x10.max())
```

```
Min: 0.012972034360483886  Max: 0.9948552746390181
```

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

```
# 6. Zero array size 10, replace 5th element with 1
zero_arr = np.zeros(10)
zero_arr[4] = 1
print("\nZero array with 5th replaced:", zero_arr)
```

```
Zero array with 5th replaced: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

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

```
# 7. Reverse an array
arr = np.array([1, 2, 0, 0, 4, 0])
reversed_arr = arr[::-1]
print("\nReversed array:", reversed_arr)
```

```
Reversed array: [0 4 0 0 2 1]
```

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

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

```
...
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.]]
```

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

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

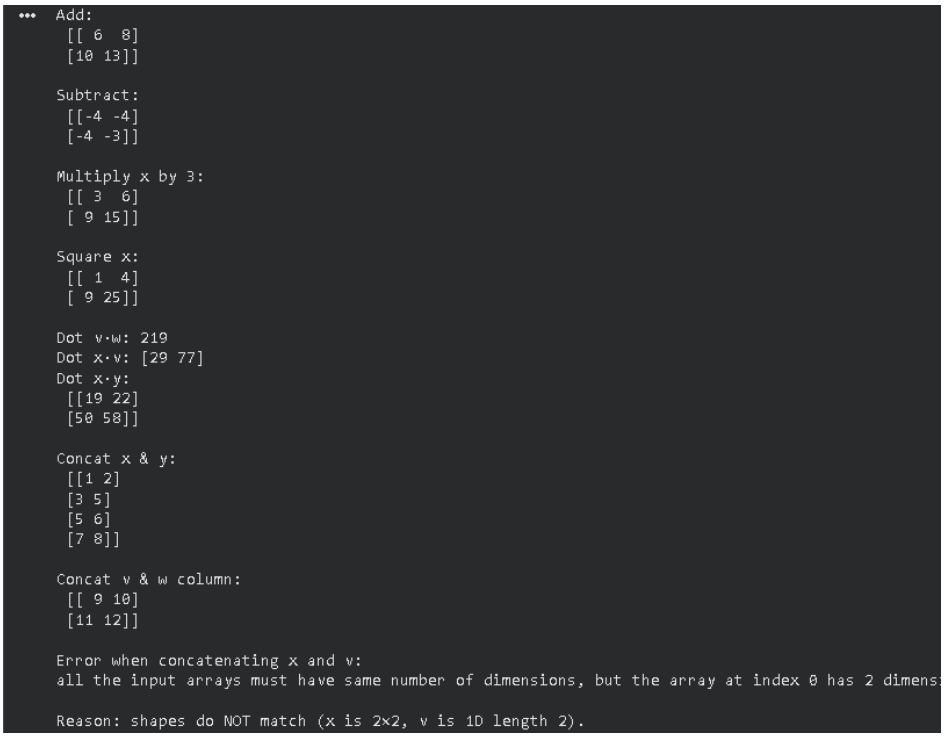
```
...
Checkerboard 8x8:
 [[0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]]
```

Problem - 3: 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.
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?

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

# 1. Add
print("Add:\n", x + y)

# 2. Subtract
print("\nSubtract:\n", x - y)

# 3. Multiply array with integer
print("\nMultiply x by 3:\n", x * 3)

# 4. Square each element
print("\nSquare x:\n", x**2)

# 5. Dot products
print("\nDot v·w:", np.dot(v, w))
print("Dot x·v:", np.dot(x, v))
print("Dot x·y:\n", np.dot(x, y))

# 6. Concatenate
concat_xy = np.concatenate((x, y), axis=0)
concat_vw = np.vstack((v, w))
print("\nConcat x & y:\n", concat_xy)
print("\nConcat v & w column:\n", concat_vw)

# 7. Concatenate
try:
    print(np.concatenate((x, v)))
except Exception as e:
    print("\nError when concatenating x and v:")
    print(e)
    print("\nReason: shapes do NOT match (x is 2×2, v is 1D length 2).")
```



```
Add:
 [[ 6  8]
 [10 13]]

Subtract:
 [[-4 -4]
 [-4 -3]]

Multiply x by 3:
 [[ 3  6]
 [ 9 15]]

Square x:
 [[ 1  4]
 [ 9 25]]

Dot v·w: 219
Dot x·v: [29 77]
Dot x·y:
 [[19 22]
 [50 58]]

Concat x & y:
 [[1 2]
 [3 5]
 [5 6]
 [7 8]]

Concat v & w column:
 [[ 9 10]
 [11 12]]

Error when concatenating x and v:
all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimens

Reason: shapes do NOT match (x is 2×2, v is 1D length 2).
```

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 $A.A^{-1} = I$.
2. Prove $AB \neq BA$.
3. Prove $(AB)^T = B^T A^T$.

- 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}

```python
import numpy as np
A = np.array([[3, 4],
              [7, 8]])
B = np.array([[5, 3],
              [2, 1]])

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

# 2. Prove AB ≠ BA
print("\nAB:\n", A @ B)
print("\nBA:\n", B @ A)

# 3. Prove (AB)^T = B^T A^T
lhs = (A @ B).T
rhs = B.T @ A.T
print("\n(AB)^T:\n", lhs)
print("\nB^T A^T:\n", rhs)

#Equation:
#2x - 3y + z = -1
#x - y + 2z = -3
#3x + y - z = 9

C = np.array([
    [2, -3,  1],
    [1, -1,  2],
    [3,  1, -1]
])

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

# Solve using inverse
C_inv = np.linalg.inv(C)
solution = C_inv @ d
print("\nSolution using inverse:\n", solution)

# Solve using np.linalg.solve
solution2 = np.linalg.solve(C, d)
print("\nSolution using np.linalg.solve:\n", solution2)
```

```
A * A_inv:
 [[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]

AB:
 [[23 13]
 [51 29]]

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

(AB)^T:
 [[23 51]
 [13 29]]

B^T A^T:
 [[23 51]
 [13 29]]

Solution using inverse:
 [ 2.  1. -2.]

Solution using np.linalg.solve:
 [ 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.

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.

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.

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
N = 1_000_000
# Python lists
L1 = list(range(N))
L2 = list(range(N))

start = time.time()
L_add = [L1[i] + L2[i] for i in range(N)]
end = time.time()
print("Python list addition time:", end - start)

# NumPy arrays
A1 = np.arange(N)
A2 = np.arange(N)

start = time.time()
A_add = A1 + A2
end = time.time()
print("NumPy addition time:", end - start)

# 2. Element-wise multiplication

# Python lists
start = time.time()
L_mul = [L1[i] * L2[i] for i in range(N)]
end = time.time()
print("\nPython list multiplication time:", end - start)

# NumPy arrays
start = time.time()
A_mul = A1 * A2
end = time.time()
print("NumPy multiplication time:", end - start)
```

```
# 3. Dot product

# Python lists
start = time.time()
dot_list = sum([L1[i] * L2[i] for i in range(N)])
end = time.time()
print("\nPython list dot product time:", end - start)

# NumPy arrays
start = time.time()
dot_np = np.dot(A1, A2)
end = time.time()
print("NumPy dot product time:", end - start)

# 4. Matrix multiplication 1000×1000

# Python list matrices
M1 = [[i for i in range(1000)] for _ in range(1000)]
M2 = [[i for i in range(1000)] for _ in range(1000)]

start = time.time()
M3 = [[sum(a*b for a, b in zip(row, col))
       for col in zip(*M2)]
      for row in M1]
end = time.time()
print("\nPython 1000x1000 matrix multiplication time:", end - start)

# NumPy matrices
N1 = np.arange(1_000_000).reshape(1000, 1000)
N2 = np.arange(1_000_000).reshape(1000, 1000)

start = time.time()
N3 = N1 @ N2
end = time.time()
print("NumPy 1000x1000 matrix multiplication time:", end - start)
```

```
Python list addition time: 0.12966704368591309
NumPy addition time: 0.004094839096069336

Python list multiplication time: 0.11907625198364258
NumPy multiplication time: 0.004414796829223633

Python list dot product time: 0.131072998046875
NumPy dot product time: 0.0020017623901367188

Python 1000x1000 matrix multiplication time: 172.02519488334656
NumPy 1000x1000 matrix multiplication time: 2.101590156555176
```

——————————– The - End ——————————-