Question number 1

```
import numpy as np
a = np.array([1,2,3,4,5])
a.shape
```

```
(5,)
```

```
a.ndim
```

```
1
```

```
a
```

```
array([1, 2, 3, 4, 5])
```

```
type(a)
```

```
numpy.ndarray
```

```
a = np.array([[1,2,3,4,5],[4,8,9,0,11]])
a
```

```
array([[ 1,  2,  3,  4,  5],
       [ 4,  8,  9,  0, 11]])
```

```
b = np.array([1,2,3,4,5])
b
```

```
array([1, 2, 3, 4, 5])
```

```
c = np.arange(16).reshape(4,4)
c
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
f = np.eye(4)
f
```

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

```
h =np.array([[1,2,3,4,5],[4,8,9,0,11]])
h
```

```
array([[ 1,  2,  3,  4,  5],
       [ 4,  8,  9,  0, 11]])
```

```
j = np.array([[1,2,3,4,5],[4,3,9,10,19]])
j
```

```
array([[ 1,  2,  3,  4,  5],
       [ 4,  3,  9, 10, 19]])
```

```
k = np.concatenate([h,j])
k
```

```
array([[ 1,  2,  3,  4,  5],
       [ 4,  8,  9,  0, 11],
       [ 1,  2,  3,  4,  5],
       [ 4,  3,  9, 10, 19]])
```

```
l = np.concatenate([h,j], axis = 1)
l
```

```
array([[ 1,  2,  3,  4,  5,  1,  2,  3,  4,  5],
       [ 4,  8,  9,  0, 11,  4,  3,  9, 10, 19]])
```

```
m = np . add(h,j)
m
```

```
array([[ 2,  4,  6,  8, 10],
       [ 8, 11, 18, 10, 30]])
```

```
n = np.hstack([h,j])
n
```

```
array([[ 1,  2,  3,  4,  5,  1,  2,  3,  4,  5],
       [ 4,  8,  9,  0, 11,  4,  3,  9, 10, 19]])
```

```
p = np.vstack([c,])
p
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
arr = np.array([[1,2,3],[4,5,6]])
arr
```

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

```
new_array = arr * 2
new_array
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12]])
```

```
copied_array = arr.copy()
copied_array
```

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

```
sliced_array = arr[1,2]
sliced_array
```

```
np.int64(6)
```

Initialize an empty array with size 2X2.

```
arr = np.empty((1, 3))
print(arr)
```

```
[[1.35264683e-315 5.31410685e-317 6.71093279e-310]]
```

Initialize an all one array with size 4X2.

```
arr = np.array([
    [2., 3.],
    [2., 5.],
    [3., 7.],
    [4., 9.]
])

print(arr)
```

```
[[2. 3.]
 [2. 5.]
 [3. 7.]
 [4. 9.]]
```

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

```
print(np.full((2, 5), 9))

print(np.array([
    [3, 5, 7],
    [1, 9, 7]
]))
```

```
    [1, 9, 7]
]))
```

```
[[9 9 9 9 9]
 [9 9 9 9 9]]
[[3 5 7]
 [1 9 7]]
```

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

```
a = np.array([[1, 3, 5],
              [7, 9, 1]])

print(a)

print(np.zeros_like(a))
```

```
[[1 3 5]
 [7 9 1]]
[[0 0 0]
 [0 0 0]]
```

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

```
a = np.array([[1, 3, 5],
              [7, 9, 1]])

print(np.ones_like(a))
```

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

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

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

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

2: Array Manipulation: Numerical Ranges and Array indexing: Complete the following tasks:

```
np.arange(13,53)
```

```
array([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, 50, 51, 52])
```

Create a 3X3 matrix with values ranging from 0 to 8. {Hint:look for np.reshape()}

```
arr = np.array([0,1,3,5,7,9,11,13,15])
arr.reshape(3,3)
```

```
array([[ 0,  1,  3],
       [ 5,  7,  9],
       [11, 13, 15]])
```

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

```
np.identity(3)
```

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

Create a random array of size 30 and find the mean of the array. {Hint:check for np.random.random() and array.mean() function}

```
arr = np.random.randint(0,1, size = 30)
print(arr)
arr.mean()
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
np.float64(0.0)
```

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())
```

```
[[67 54 42 13  4 72 83 27 75 33]
 [46  5  9 90 44 68 48 27 32 49]
 [81 32 60 33  7 37 73 54 95 46]
 [22 69 71 75  2 75 66 11 80 89]
 [95 25 44 19 27 94 43  4 97 70]
 [97  6 77 63 37 14 97 58 25 18]
 [60 62 33 38 19 11 35 78 60 71]
 [76 97 20  2 88 42 86 39  2 52]
 [10 45 17 80 52 68 84 75 16 81]
 [48 47 56 53  4 50  8 46 13 70]]
2
97
```

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

```
arr = np.zeros(11)
arr[4] = 1
arr
```

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

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

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

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

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

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

```
[[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.]]
```

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

```
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.]]
```

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:

Add the two array.

```
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]]
```

Subtract the two array.

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

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

Multiply the array with any integers of your choice.

```
print(3*x)
```

```
[[ 3  6]
 [ 9 15]]
```

Find the square of each element of the array.

```
print(np.square (y))
```

```
[[25 36]
 [49 64]]
```

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

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

```
219


[29 77]
```

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

```
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]]
```

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

```
np.concatenate((x, v))
```

```
---------------------------------------------------------------------------
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 has 1 dimension(s)
```

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

```
A = np.random.randint(1,1000,100000)
B = np.random.randint(1,1000,100000)

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

```
np.int64(100022640)
```

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.

```
import time

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("Time taken:", end - start)
```

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

```
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: 0.0035696029663085938 seconds
```

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.

```
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()

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

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

```
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: 0.00180816650390625 seconds
```

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):
```

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: 0.0654752254486084 seconds
```