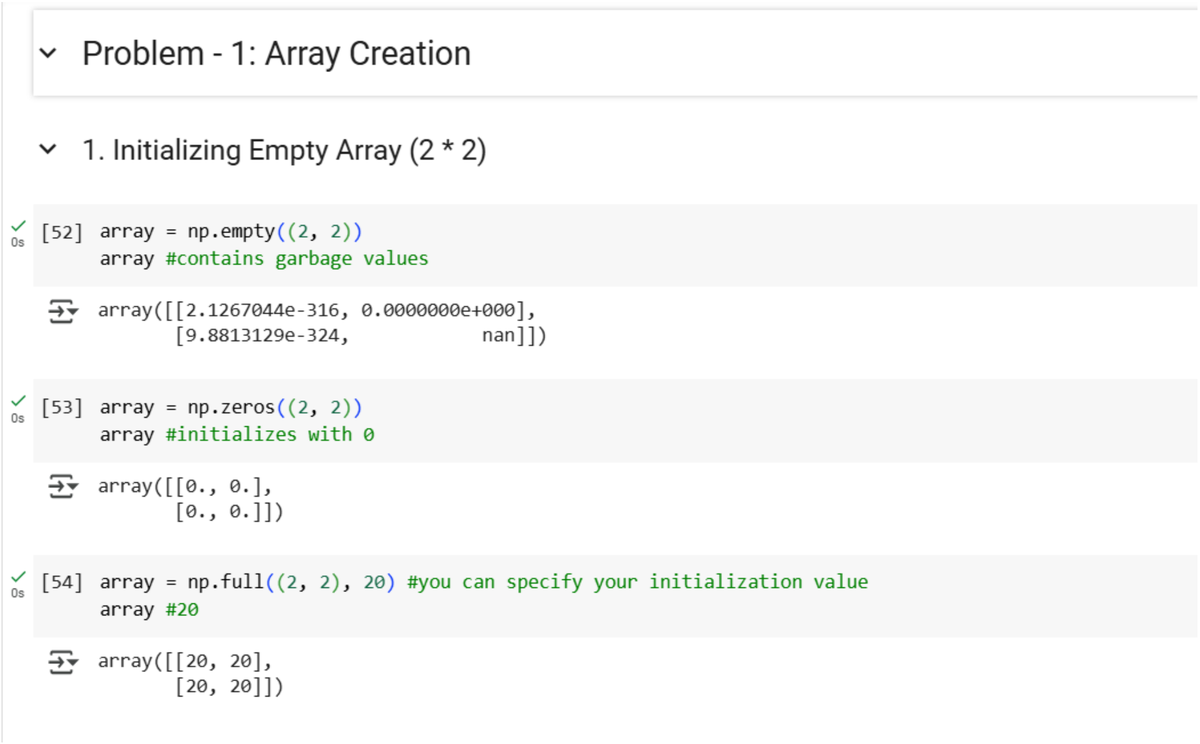
Worksheet-0

Screenshot of Outputs

10. TO-Do-NumPy

10.1 Basic Vector and Matrix Operation with Numpy.



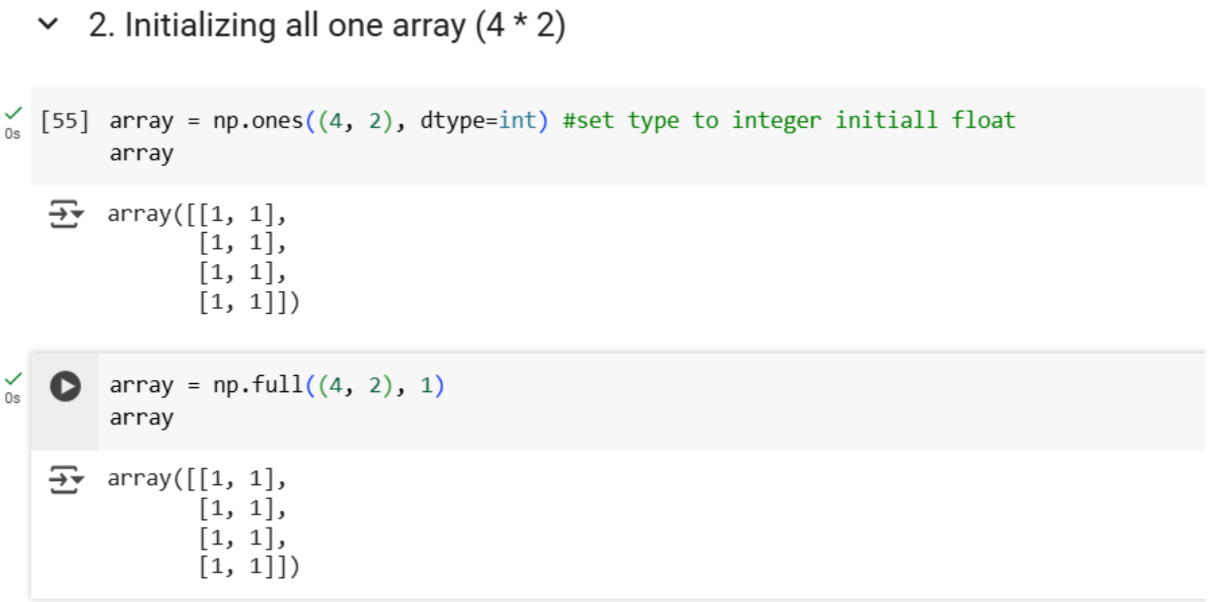
Problem - 1: Array Creation

1. Initializing Empty Array (2 * 2)

```
[52] array = np.empty((2, 2))
     array #contains garbage values

array([[2.1267044e-316, 0.0000000e+000],
       [9.8813129e-324,            nan]])
```

```
[53] array = np.zeros((2, 2))
     array #initializes with 0

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

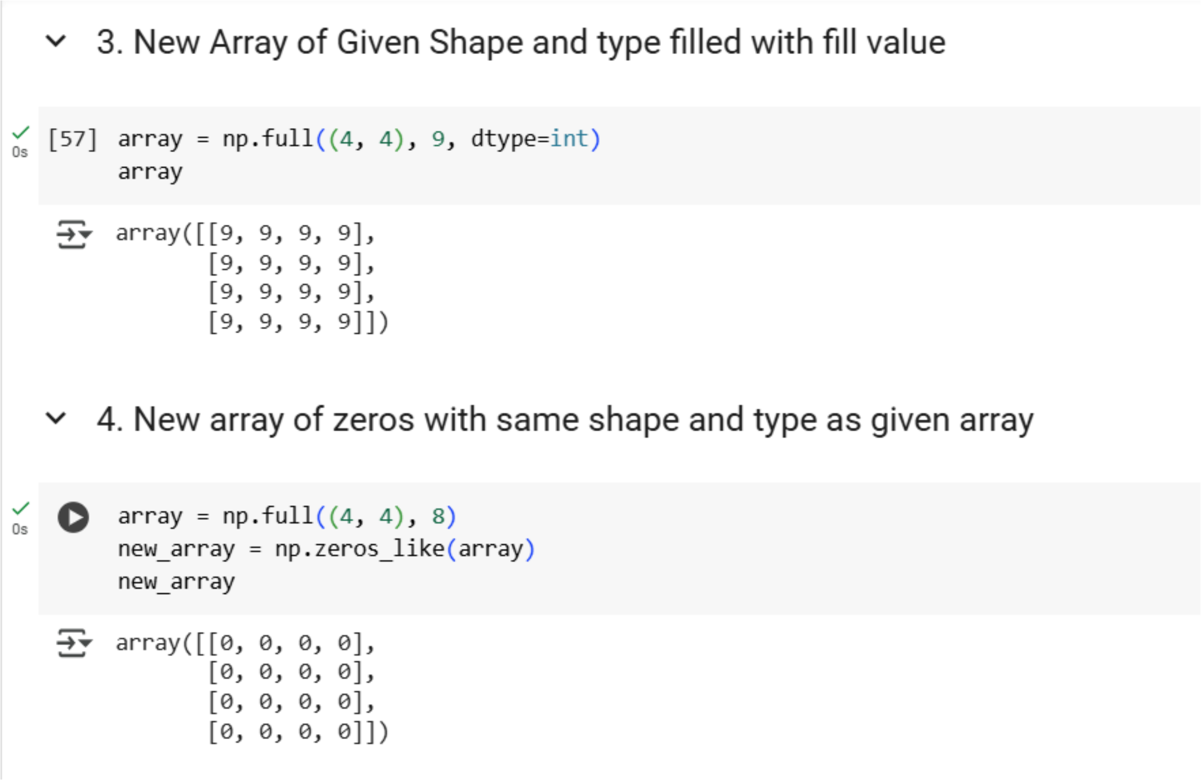
```
[54] array = np.full((2, 2), 20) #you can specify your initialization value
     array #20

array([[20, 20],
       [20, 20]])
```



2. Initializing all one array (4 * 2)

```
[55] array = np.ones((4, 2), dtype=int) #set type to integer initiall float
     array

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

```
array = np.full((4, 2), 1)
array

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



3. New Array of Given Shape and type filled with fill value

✓ 0s [57]

```
array = np.full((4, 4), 9, dtype=int)
array
```

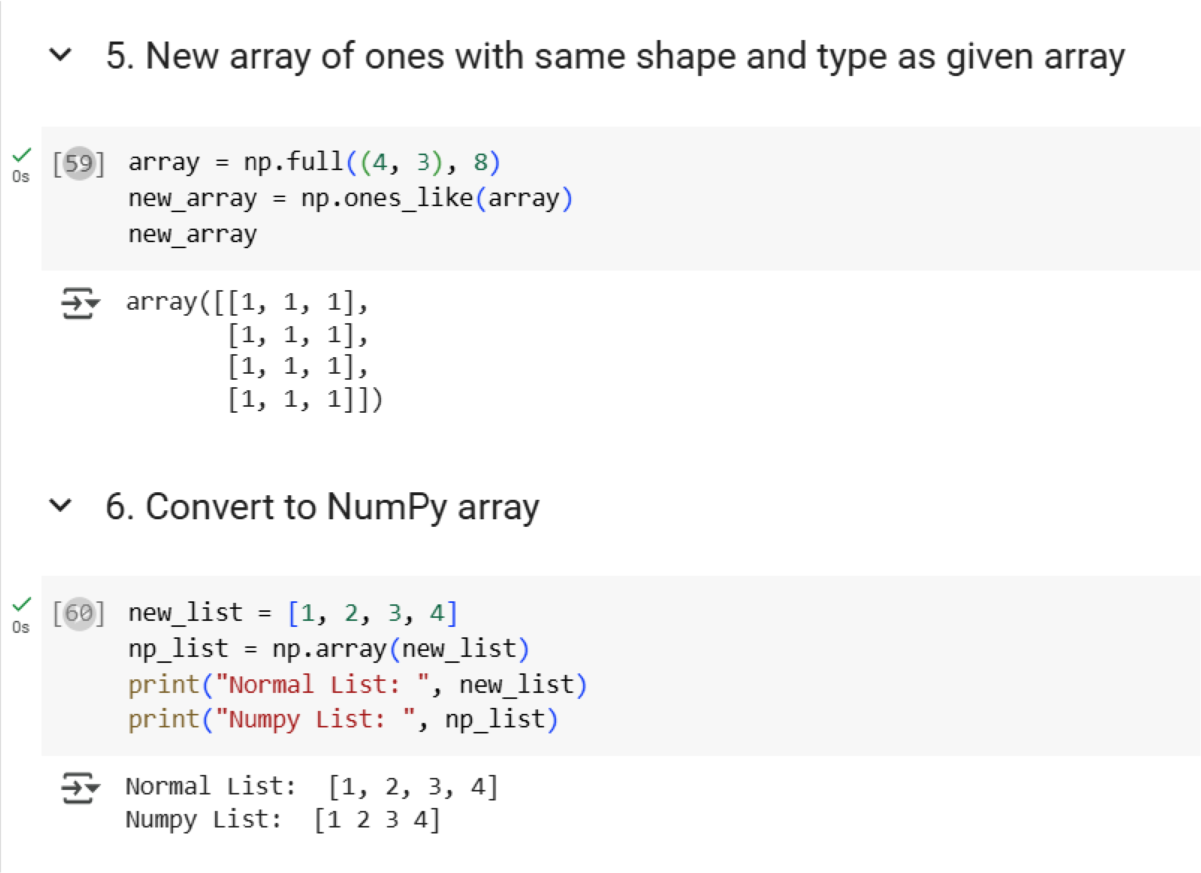
```
array([[9, 9, 9, 9],
       [9, 9, 9, 9],
       [9, 9, 9, 9],
       [9, 9, 9, 9]])
```

4. New array of zeros with same shape and type as given array

✓ 0s

```
array = np.full((4, 4), 8)
new_array = np.zeros_like(array)
new_array
```

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



5. New array of ones with same shape and type as given array

✓ 0s [59]

```
array = np.full((4, 3), 8)
new_array = np.ones_like(array)
new_array
```

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

6. Convert to NumPy array

✓ 0s [60]

```
new_list = [1, 2, 3, 4]
np_list = np.array(new_list)
print("Normal List: ", new_list)
print("Numpy List: ", np_list)
```

```
Normal List:  [1, 2, 3, 4]
Numpy List:  [1 2 3 4]
```

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

1. Create an array with values ranging from 10 to 49

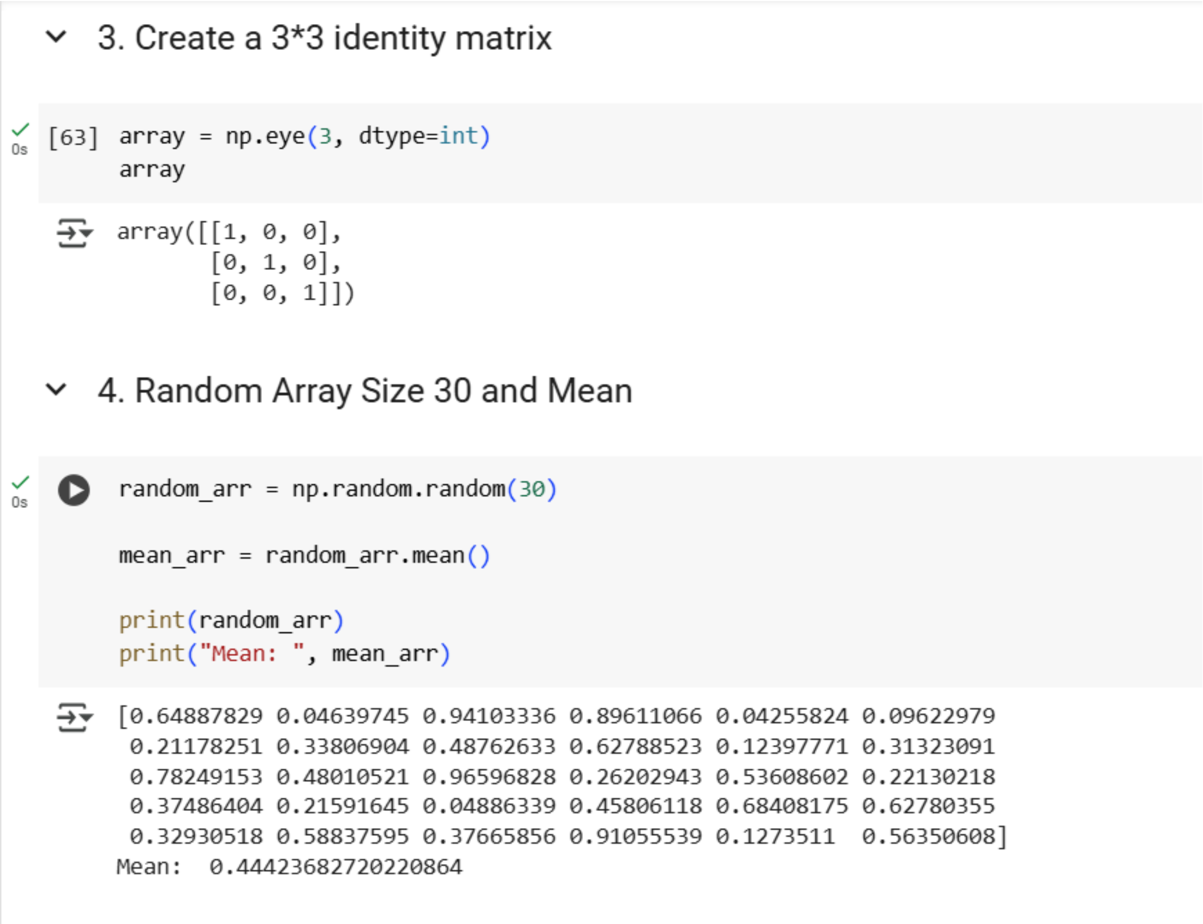
[61]
```
array = np.arange(10, 50)
array
```

```
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, 49])
```

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

[62]
```
array = np.arange(0, 9)
reshaped = np.reshape(array, (3, 3))
print(array)
print(reshaped)
```

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



3. Create a 3*3 identity matrix

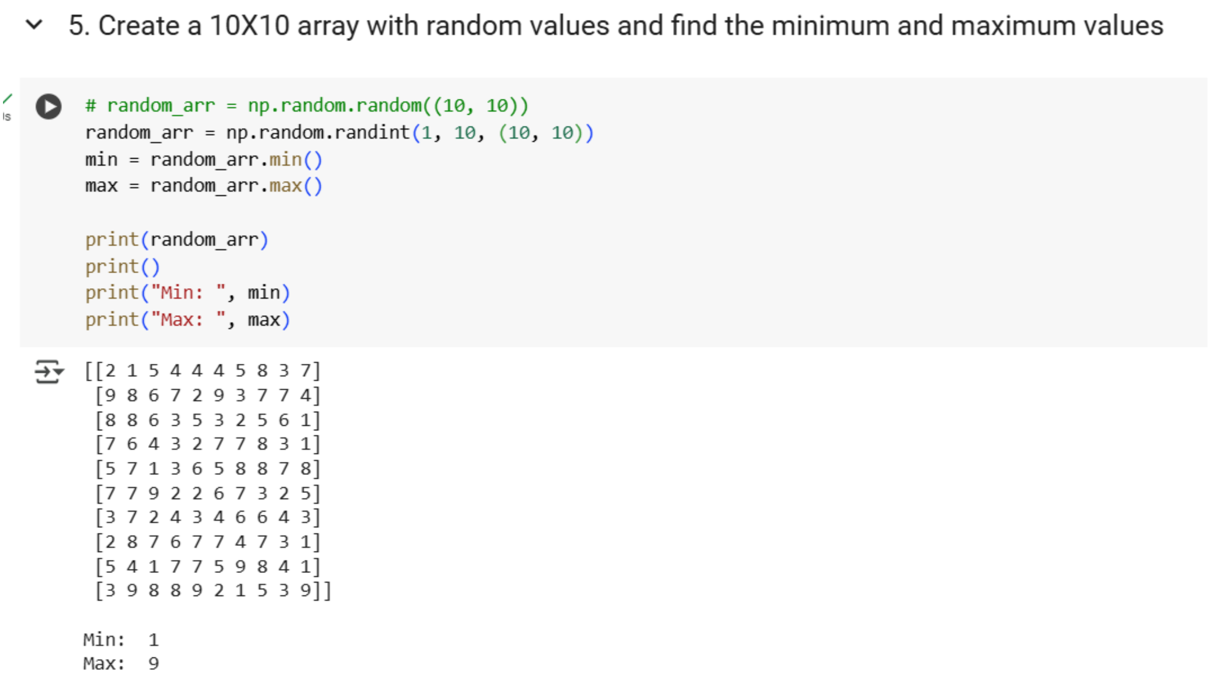
[63]
```
array = np.eye(3, dtype=int)
array
```

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

4. Random Array Size 30 and Mean

```
random_arr = np.random.random(30)

mean_arr = random_arr.mean()

print(random_arr)
print("Mean: ", mean_arr)
```

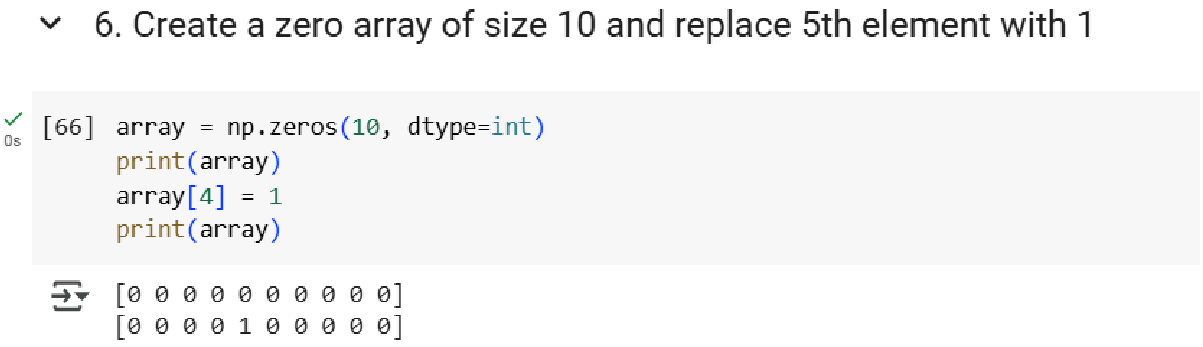
```
[0.64887829 0.04639745 0.94103336 0.89611066 0.04255824 0.09622979
 0.21178251 0.33806904 0.48762633 0.62788523 0.12397771 0.31323091
 0.78249153 0.48010521 0.96596828 0.26202943 0.53608602 0.22130218
 0.37486404 0.21591645 0.04886339 0.45806118 0.68408175 0.62780355
 0.32930518 0.58837595 0.37665856 0.91055539 0.1273511  0.56350608]
Mean:  0.44423682720220864
```



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

```
# random_arr = np.random.random((10, 10))
random_arr = np.random.randint(1, 10, (10, 10))
min = random_arr.min()
max = random_arr.max()

print(random_arr)
print()
print("Min: ", min)
print("Max: ", max)
```

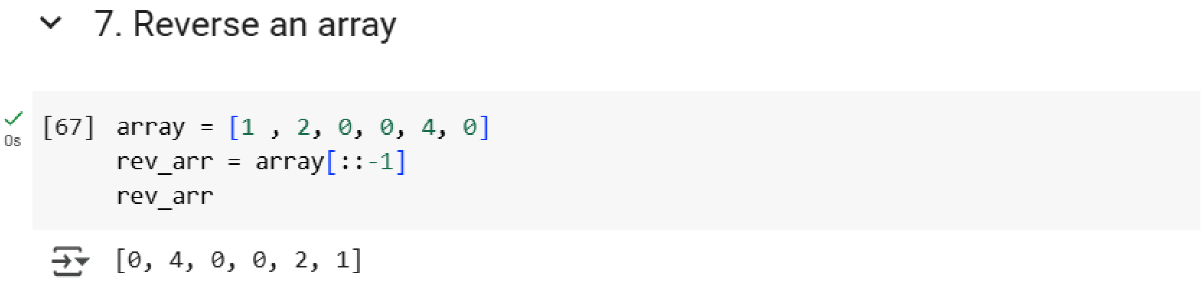
```
[[2 1 5 4 4 4 5 8 3 7]
 [9 8 6 7 2 9 3 7 7 4]
 [8 8 6 3 5 3 2 5 6 1]
 [7 6 4 3 2 7 7 8 3 1]
 [5 7 1 3 6 5 8 8 7 8]
 [7 7 9 2 2 6 7 3 2 5]
 [3 7 2 4 3 4 6 6 4 3]
 [2 8 7 6 7 7 4 7 3 1]
 [5 4 1 7 7 5 9 8 4 1]
 [3 9 8 8 9 2 1 5 3 9]]

Min:  1
Max:  9
```



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

```
array = np.zeros(10, dtype=int)
print(array)
array[4] = 1
print(array)
```

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



7. Reverse an array

```
array = [1 , 2, 0, 0, 4, 0]
rev_arr = array[::-1]
rev_arr
```

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

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

0s

```
arr = np.random.randint(1, 10, (7, 7))

# arr = np.zeros_like(arr)
arr[0, :] = 1
arr[-1, :] = 1
arr[:, 0] = 1
arr[:, -1] = 1

arr[0:-1, 1:-1] = 0

arr
```

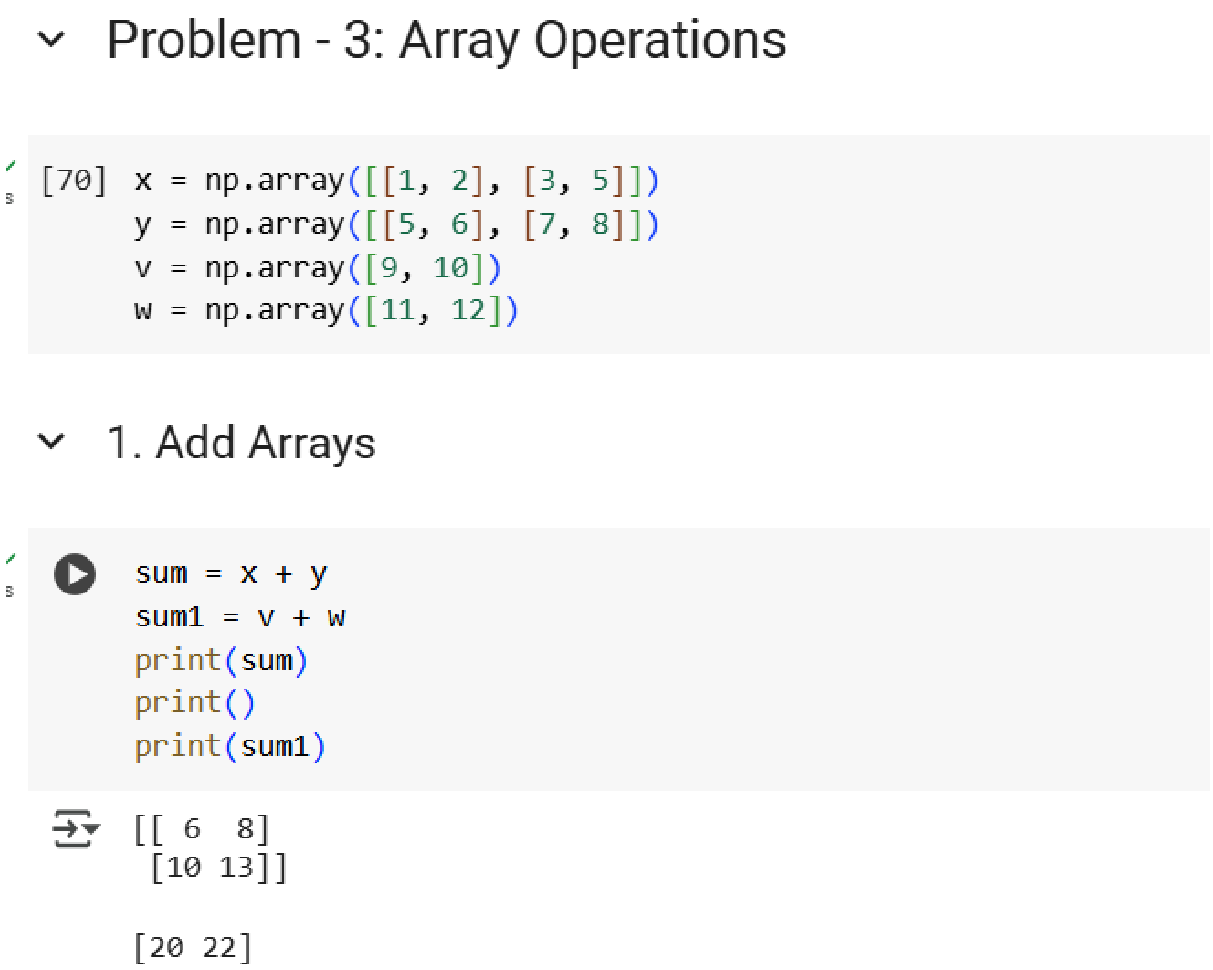
```
array([[1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 1, 1, 1, 1, 1, 1]])
```

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

0s [78]

```
array = np.zeros((8, 8), dtype=int)

array[1::2, 0::2] = 1
array[::2, 1::2] = 1
array
```

```
array([[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]])
```



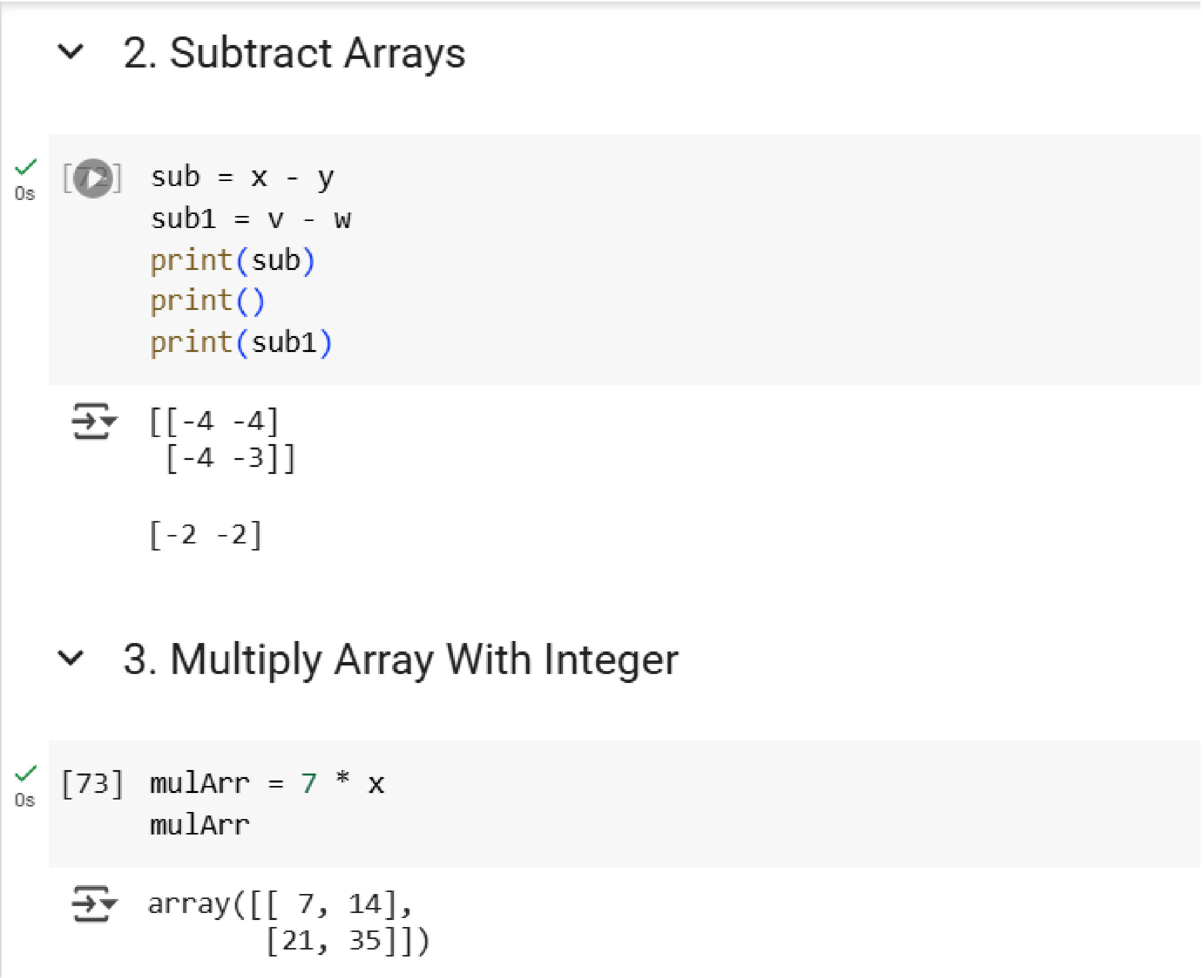
Problem - 3: Array Operations

```
[70] 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 Arrays

```
sum = x + y
sum1 = v + w
print(sum)
print()
print(sum1)
```

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

[20 22]
```



2. Subtract Arrays

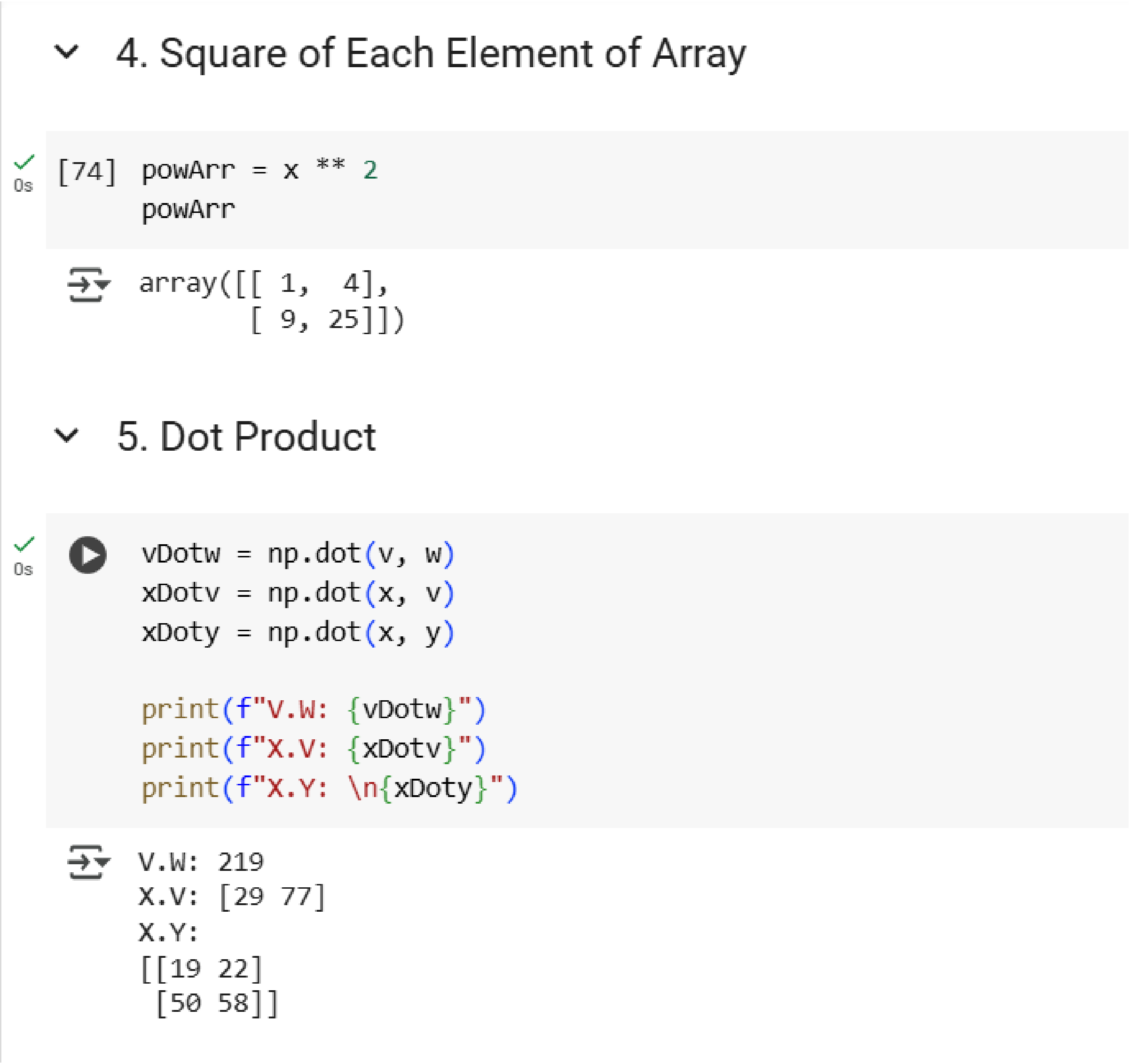
```
sub = x - y
sub1 = v - w
print(sub)
print()
print(sub1)
```

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

[-2 -2]
```

3. Multiply Array With Integer

```
mulArr = 7 * x
mulArr
```

```
array([[ 7, 14],
       [21, 35]])
```



4. Square of Each Element of Array

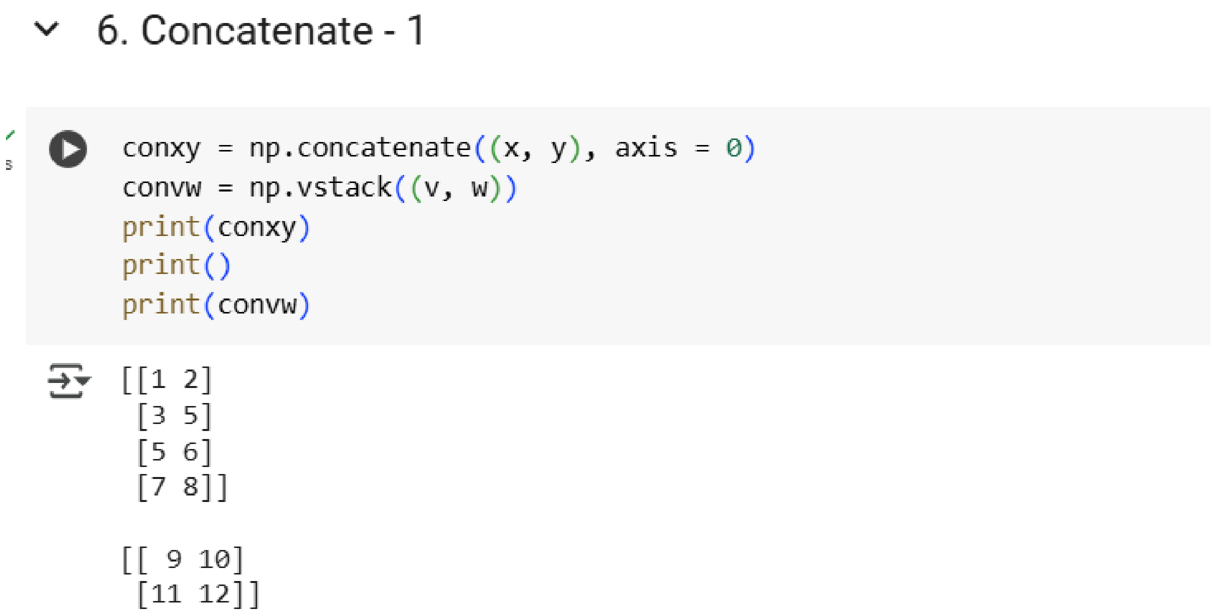
```
powArr = x ** 2
powArr
```

```
array([[ 1,  4],
       [ 9, 25]])
```

5. Dot Product

```
vDotw = np.dot(v, w)
xDotv = np.dot(x, v)
xDoty = np.dot(x, y)

print(f"V.W: {vDotw}")
print(f"X.V: {xDotv}")
print(f"X.Y: \n{xDoty}")
```

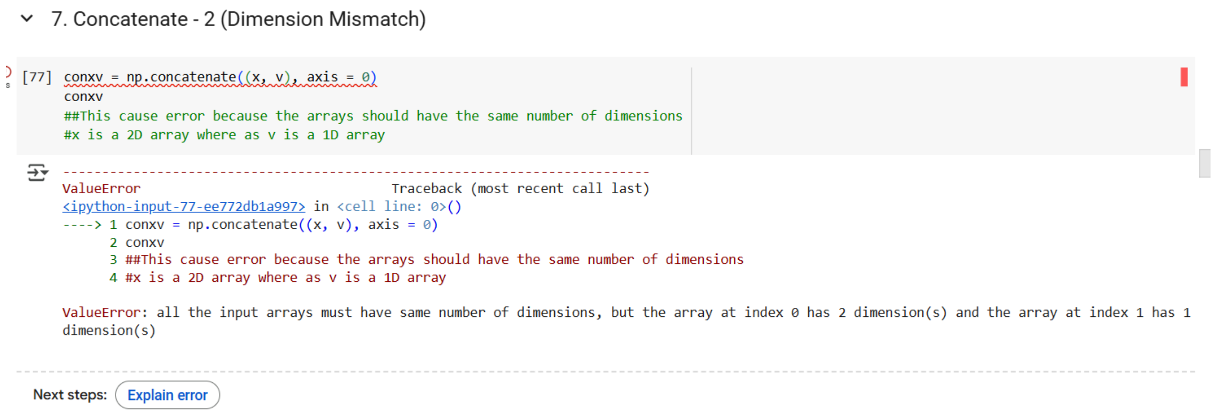
```
V.W: 219
X.V: [29 77]
X.Y: 
[[19 22]
 [50 58]]
```



6. Concatenate - 1

```
conxy = np.concatenate((x, y), axis = 0)
convw = np.vstack((v, w))
print(conxy)
print()
print(convw)
```

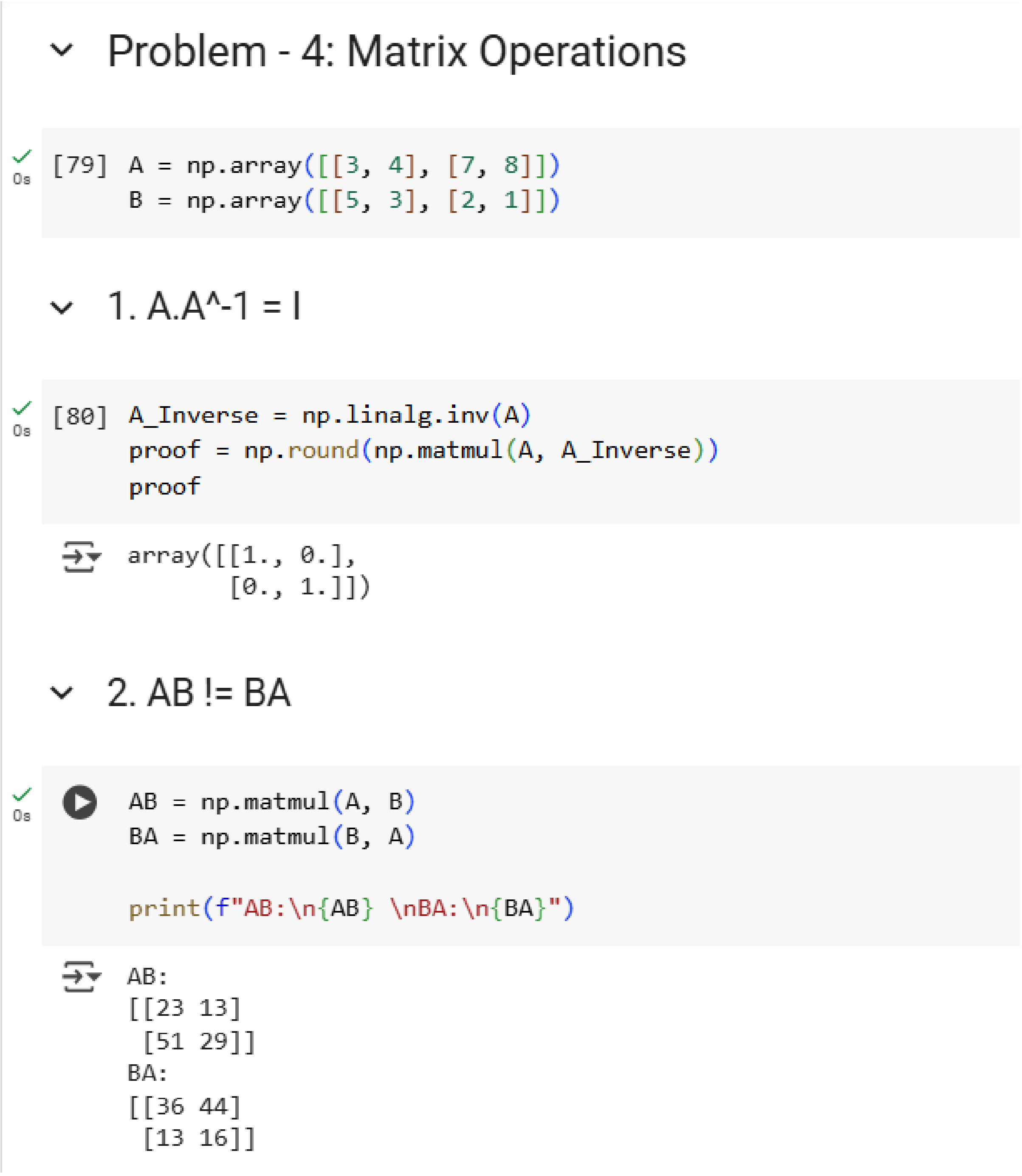
```
[[1 2]
 [3 5]
 [5 6]
 [7 8]]

[[ 9 10]
 [11 12]]
```



7. Concatenate - 2 (Dimension Mismatch)

```
[77] conxv = np.concatenate((x, v), axis = 0)
     conxv
     ##This cause error because the arrays should have the same number of dimensions
     #x is a 2D array where as v is a 1D array
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-77-ee772db1a997> in <cell line: 0>()
----> 1 conxv = np.concatenate((x, v), axis = 0)
      2 conxv
      3 ##This cause error because the arrays should have the same number of dimensions
      4 #x is a 2D array where as v is a 1D array

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)
```



Problem - 4: Matrix Operations

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

1. A.A^-1 = I

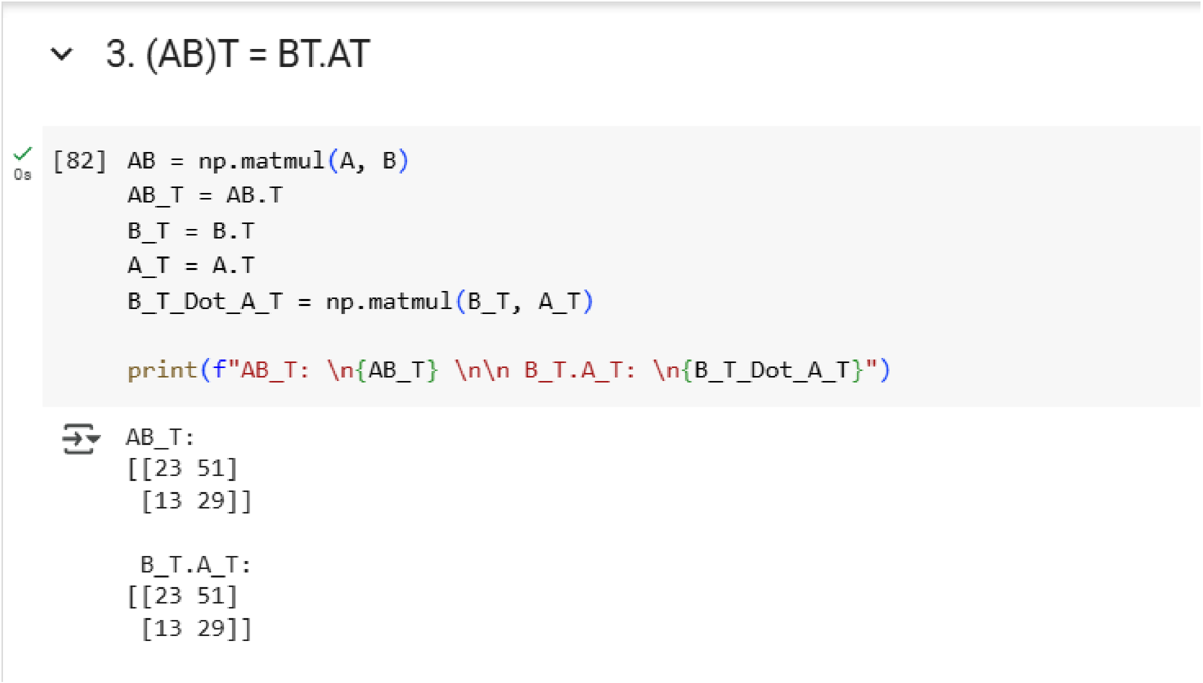
```
A_Inverse = np.linalg.inv(A)
proof = np.round(np.matmul(A, A_Inverse))
proof
```

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

2. AB != BA

```
AB = np.matmul(A, B)
BA = np.matmul(B, A)

print(f"AB:\n{AB} \nBA:\n{BA}")
```

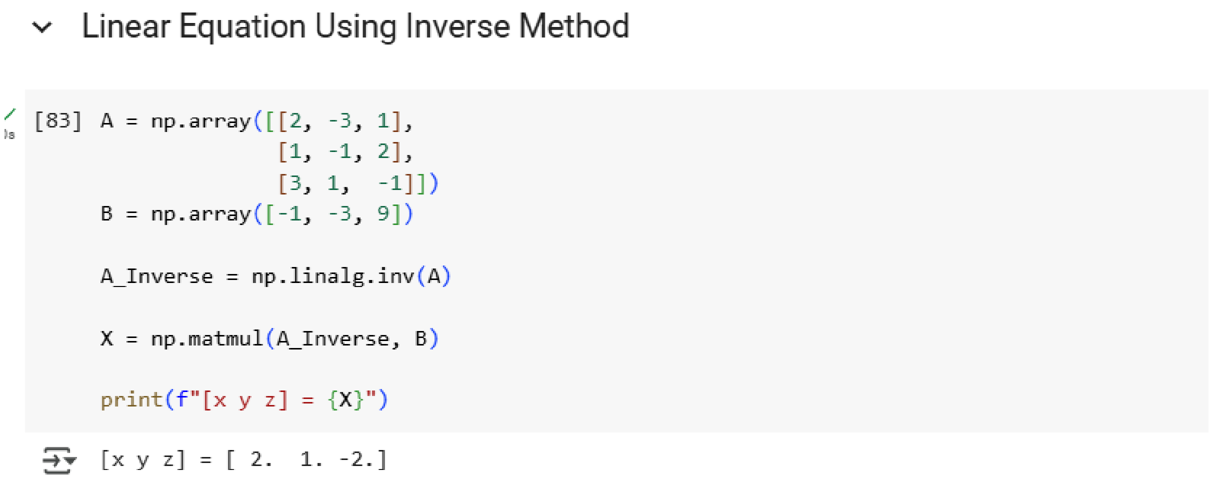
```
AB:
[[23 13]
 [51 29]]
BA:
[[36 44]
 [13 16]]
```



3. (AB)T = BT.AT

```
[82] AB = np.matmul(A, B)
     AB_T = AB.T
     B_T = B.T
     A_T = A.T
     B_T_Dot_A_T = np.matmul(B_T, A_T)

     print(f"AB_T: \n{AB_T} \n\n B_T.A_T: \n{B_T_Dot_A_T}")
```

```
AB_T:
[[23 51]
 [13 29]]

 B_T.A_T:
[[23 51]
 [13 29]]
```



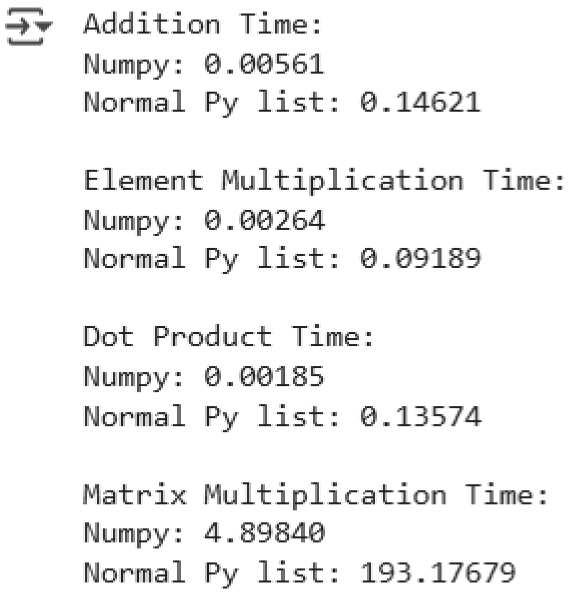
Linear Equation Using Inverse Method

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

     A_Inverse = np.linalg.inv(A)

     X = np.matmul(A_Inverse, B)

     print(f"[x y z] = {X}")
```

```
[x y z] = [ 2.  1. -2.]
```

10.2 Experiment: How Fast is Numpy?

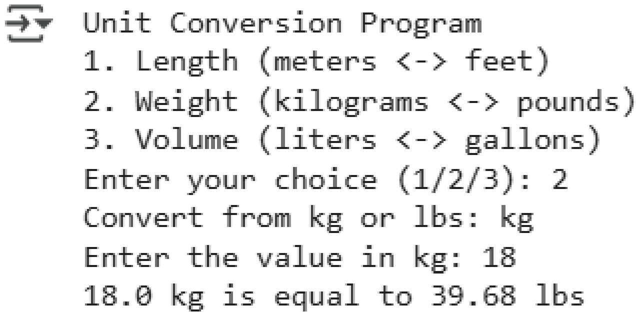
Numpy Speed



```
Addition Time:
Numpy: 0.00561
Normal Py list: 0.14621

Element Multiplication Time:
Numpy: 0.00264
Normal Py list: 0.09189

Dot Product Time:
Numpy: 0.00185
Normal Py list: 0.13574

Matrix Multiplication Time:
Numpy: 4.89840
Normal Py list: 193.17679
```

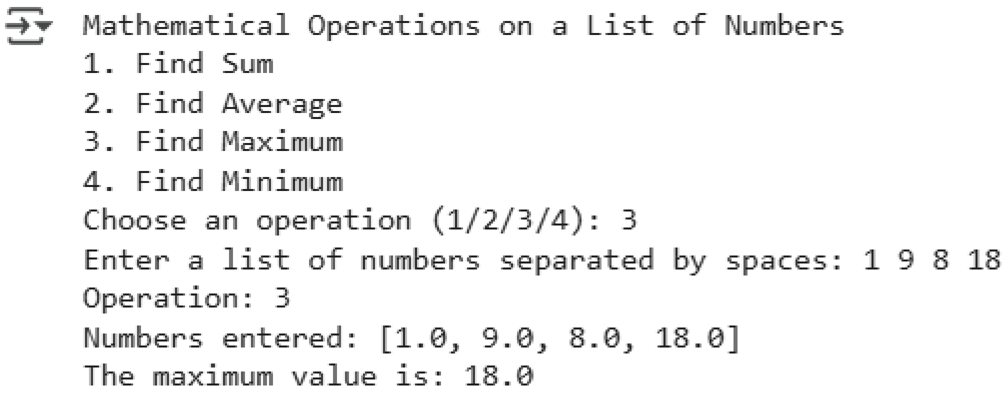
4.1 Exercise on Functions

Task-1



```
Unit Conversion Program
1. Length (meters <-> feet)
2. Weight (kilograms <-> pounds)
3. Volume (liters <-> gallons)
Enter your choice (1/2/3): 2
Convert from kg or lbs: kg
Enter the value in kg: 18
18.0 kg is equal to 39.68 lbs
```

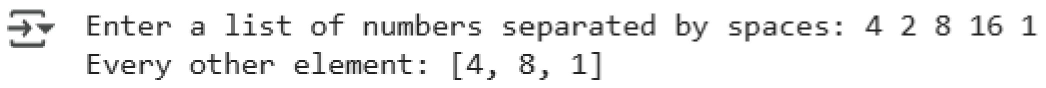
Task-2



```
Mathematical Operations on a List of Numbers
1. Find Sum
2. Find Average
3. Find Maximum
4. Find Minimum
Choose an operation (1/2/3/4): 3
Enter a list of numbers separated by spaces: 1 9 8 18
Operation: 3
Numbers entered: [1.0, 9.0, 8.0, 18.0]
The maximum value is: 18.0
```

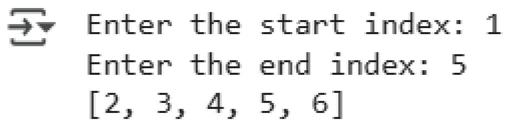
4.2 Exercise on List Manipulation

1. Extract Every Other Element



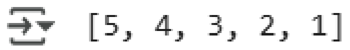
```
Enter a list of numbers separated by spaces: 4 2 8 16 1
Every other element: [4, 8, 1]
```

2. Slice a Sublist



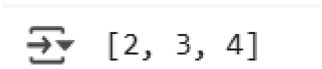
```
Enter the start index: 1
Enter the end index: 5
[2, 3, 4, 5, 6]
```

3. Reverse a List Using Slicing



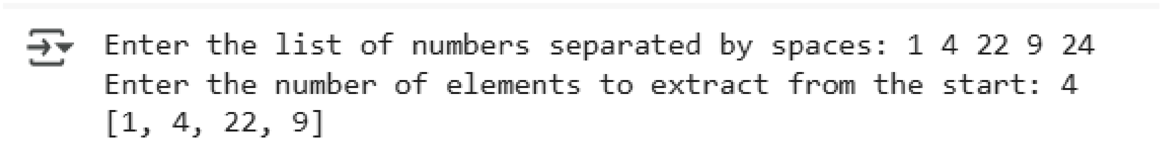
```
[5, 4, 3, 2, 1]
```

4. Remove the First and Last Elements



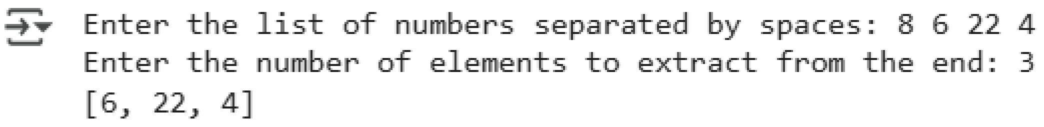
```
[2, 3, 4]
```

5. Get the First n Elements



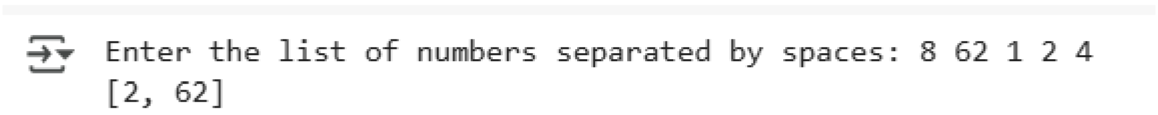
```
Enter the list of numbers separated by spaces: 1 4 22 9 24
Enter the number of elements to extract from the start: 4
[1, 4, 22, 9]
```

6. Extract Elements from the End



```
Enter the list of numbers separated by spaces: 8 6 22 4
Enter the number of elements to extract from the end: 3
[6, 22, 4]
```

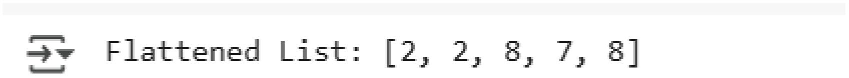
7. Extract Elements in Reverse Order



```
Enter the list of numbers separated by spaces: 8 62 1 2 4
[2, 62]
```

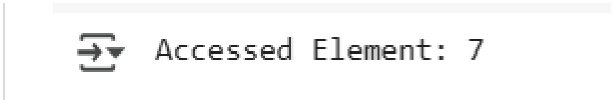
4.3 Exercise on Nested List

1. Flatten a Nested List



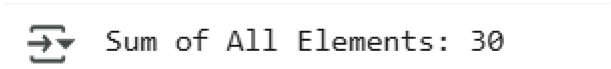
```
Flattened List: [2, 2, 8, 7, 8]
```

2. Accessing Nested List Elements



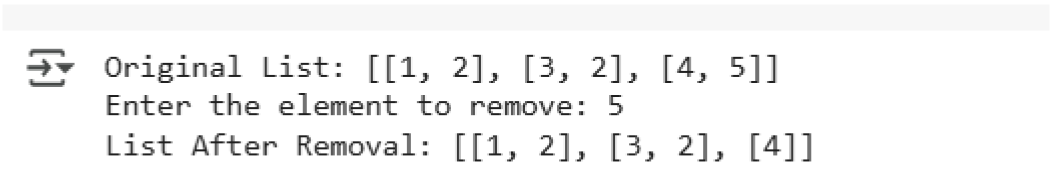
```
Accessed Element: 7
```

3. Sum of All Elements in a Nested List



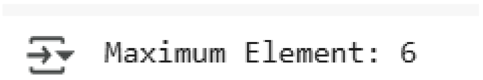
```
Sum of All Elements: 30
```

4. Remove Specific Element from a Nested List



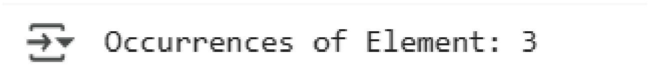
```
Original List: [[1, 2], [3, 2], [4, 5]]
Enter the element to remove: 5
List After Removal: [[1, 2], [3, 2], [4]]
```

5. Find the Maximum Element in a Nested List



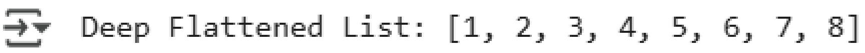
```
Maximum Element: 6
```

6. Count Occurrences of an Element in a Nested List



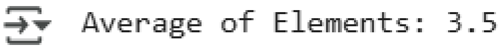
```
Occurrences of Element: 3
```

7. Flatten a List of Lists of Lists



```
Deep Flattened List: [1, 2, 3, 4, 5, 6, 7, 8]
```

8. Nested List Average



```
Average of Elements: 3.5
```