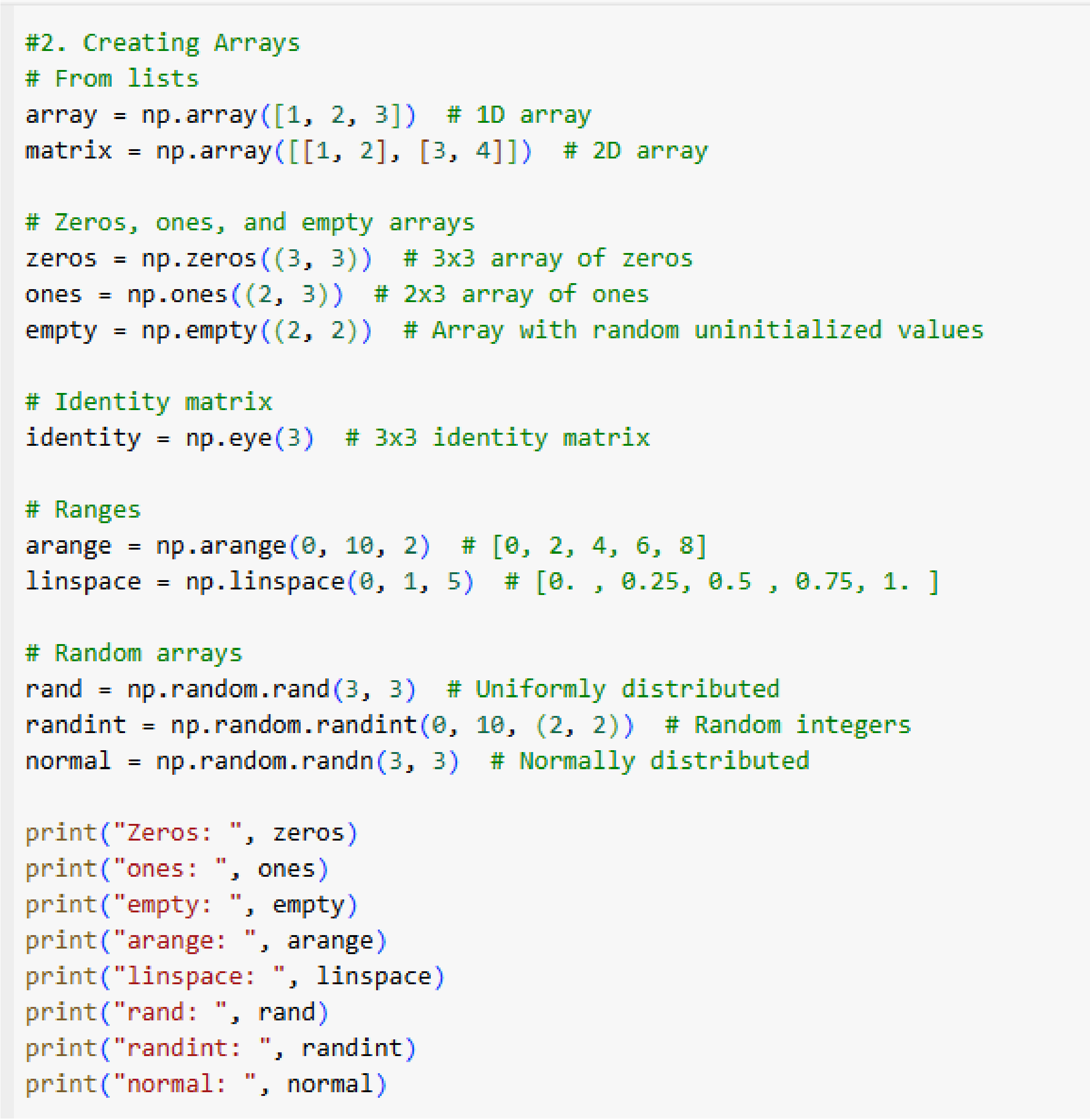
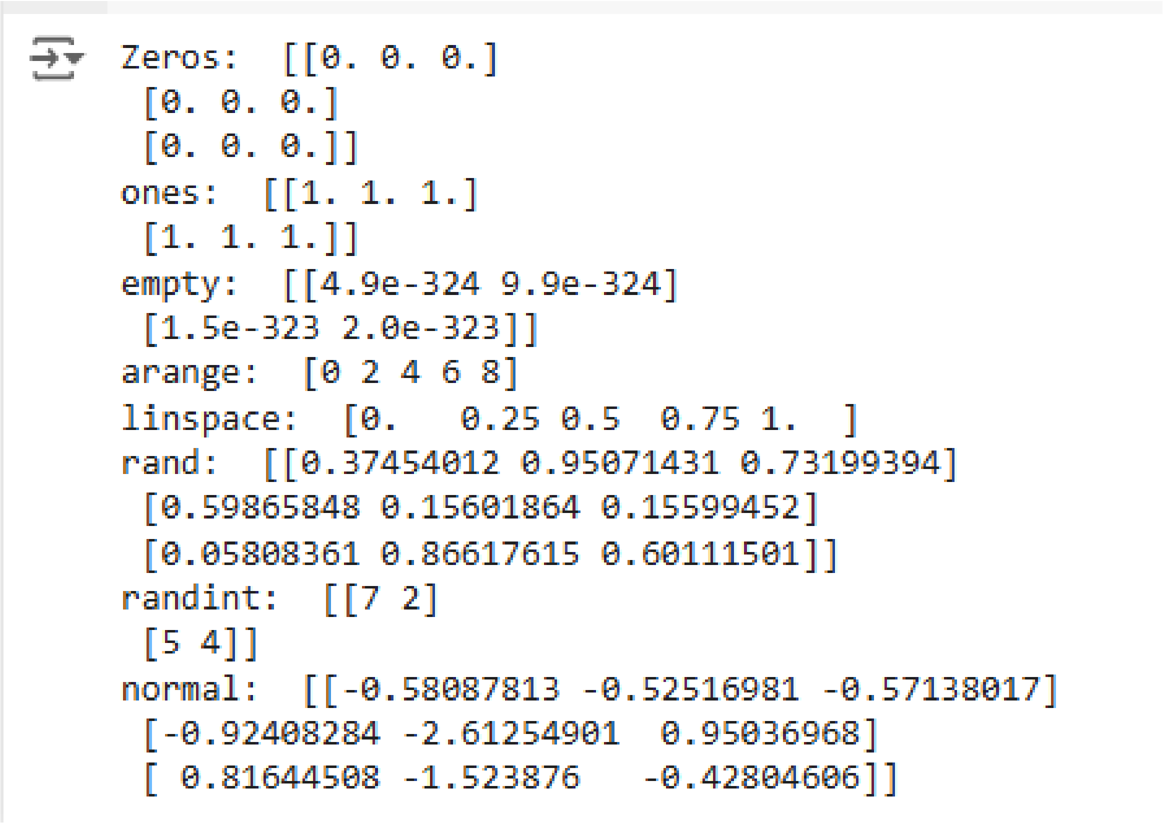
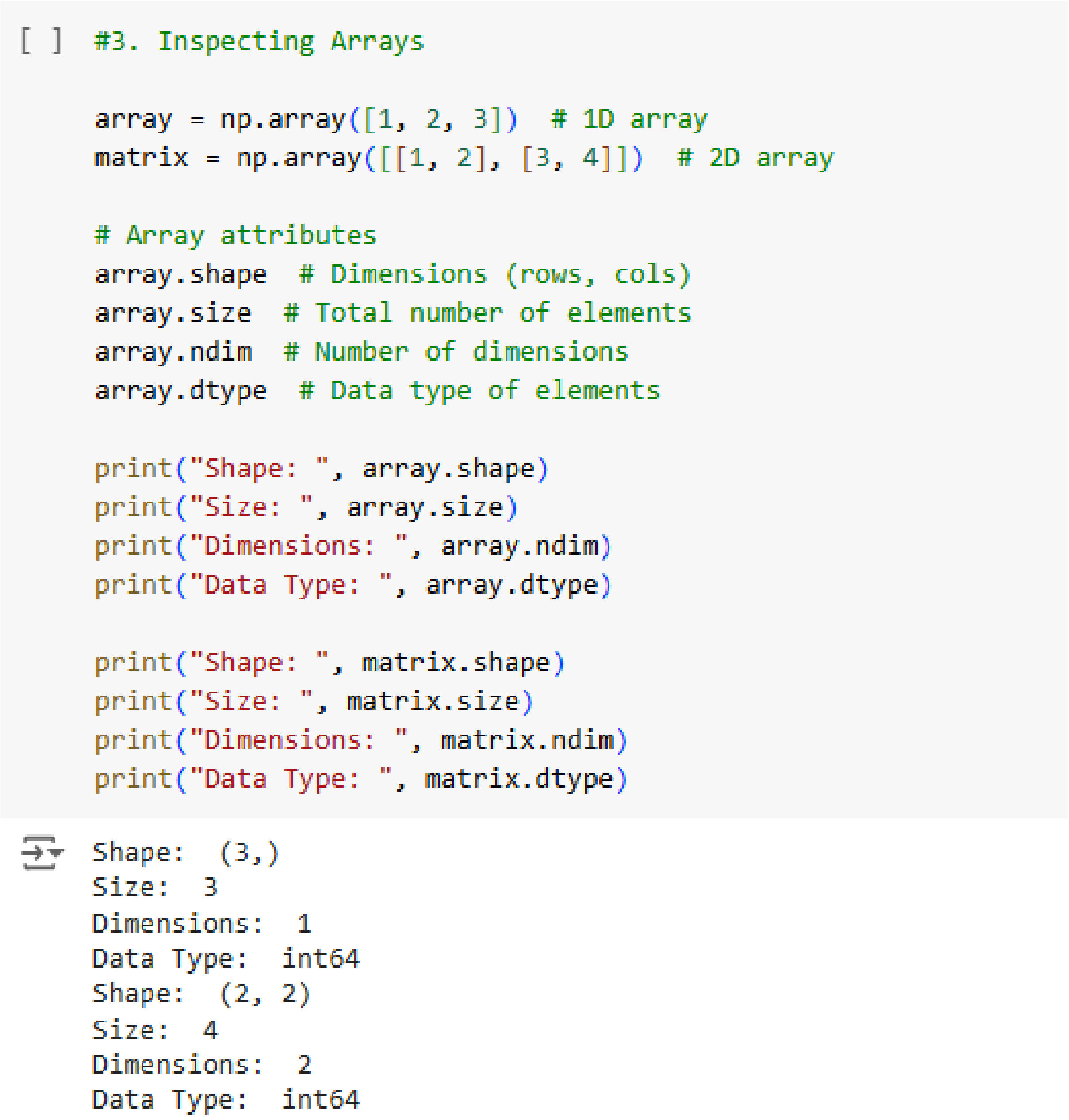


```
#2. Creating Arrays
# From lists
array = np.array([1, 2, 3])  # 1D array
matrix = np.array([[1, 2], [3, 4]])  # 2D array

# Zeros, ones, and empty arrays
zeros = np.zeros((3, 3))  # 3x3 array of zeros
ones = np.ones((2, 3))  # 2x3 array of ones
empty = np.empty((2, 2))  # Array with random uninitialized values

# Identity matrix
identity = np.eye(3)  # 3x3 identity matrix

# Ranges
arange = np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
linspace = np.linspace(0, 1, 5)  # [0.  , 0.25, 0.5 , 0.75, 1. ]

# Random arrays
rand = np.random.rand(3, 3)  # Uniformly distributed
randint = np.random.randint(0, 10, (2, 2))  # Random integers
normal = np.random.randn(3, 3)  # Normally distributed

print("Zeros: ", zeros)
print("ones: ", ones)
print("empty: ", empty)
print("arange: ", arange)
print("linspace: ", linspace)
print("rand: ", rand)
print("randint: ", randint)
print("normal: ", normal)
```

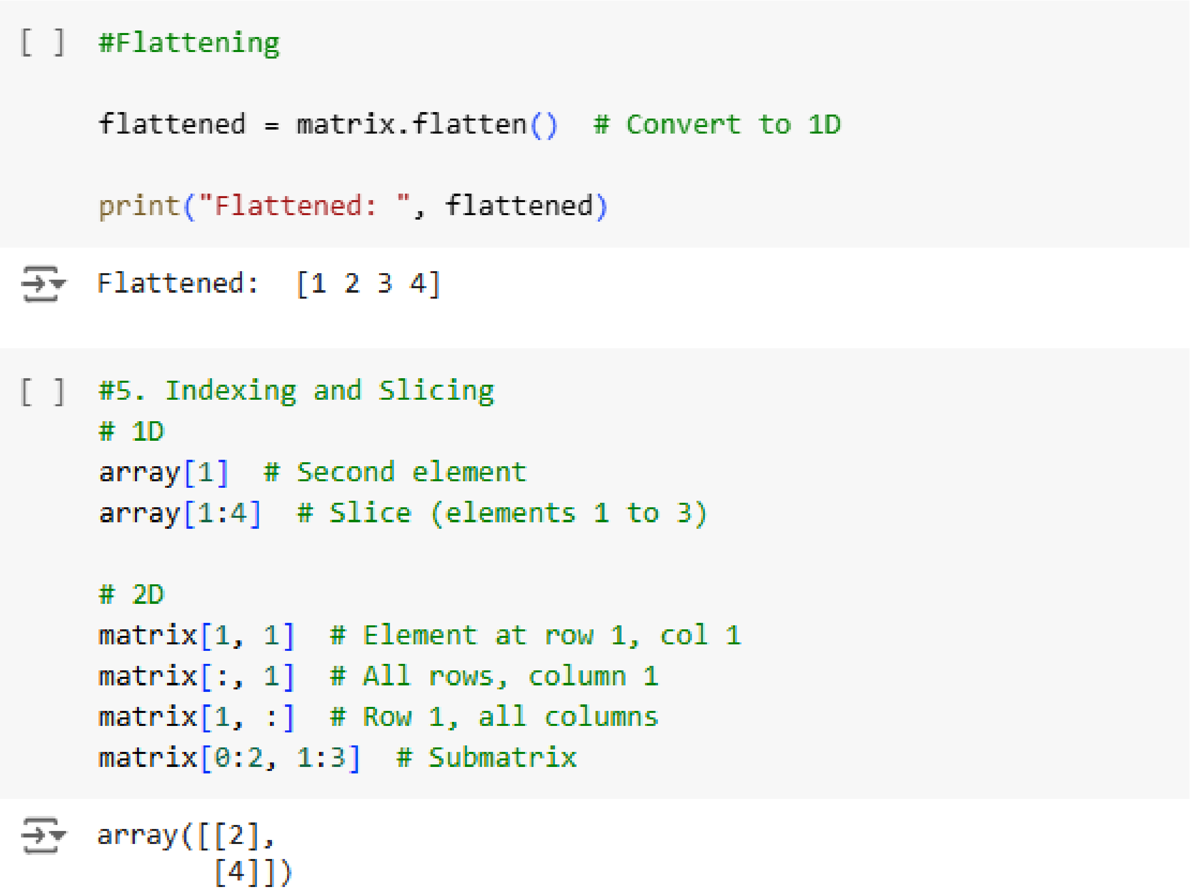


```
Zeros:  [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
ones:  [[1. 1. 1.]
 [1. 1. 1.]]
empty:  [[4.9e-324 9.9e-324]
 [1.5e-323 2.0e-323]]
arange:  [0 2 4 6 8]
linspace:  [0.   0.25 0.5  0.75 1.  ]
rand:  [[0.37454012 0.95071431 0.73199394]
 [0.59865848 0.15601864 0.15599452]
 [0.05808361 0.86617615 0.60111501]]
randint:  [[7 2]
 [5 4]]
normal:  [[-0.58087813 -0.52516981 -0.57138017]
 [-0.92408284 -2.61254901  0.95036968]
 [ 0.81644508 -1.523876   -0.42804606]]
```



```python
#3. Inspecting Arrays

array = np.array([1, 2, 3])  # 1D array
matrix = np.array([[1, 2], [3, 4]])  # 2D array

# Array attributes
array.shape  # Dimensions (rows, cols)
array.size  # Total number of elements
array.ndim  # Number of dimensions
array.dtype  # Data type of elements

print("Shape: ", array.shape)
print("Size: ", array.size)
print("Dimensions: ", array.ndim)
print("Data Type: ", array.dtype)

print("Shape: ", matrix.shape)
print("Size: ", matrix.size)
print("Dimensions: ", matrix.ndim)
print("Data Type: ", matrix.dtype)
```

```
Shape:  (3,)
Size:  3
Dimensions:  1
Data Type:  int64
Shape:  (2, 2)
Size:  4
Dimensions:  2
Data Type:  int64
```

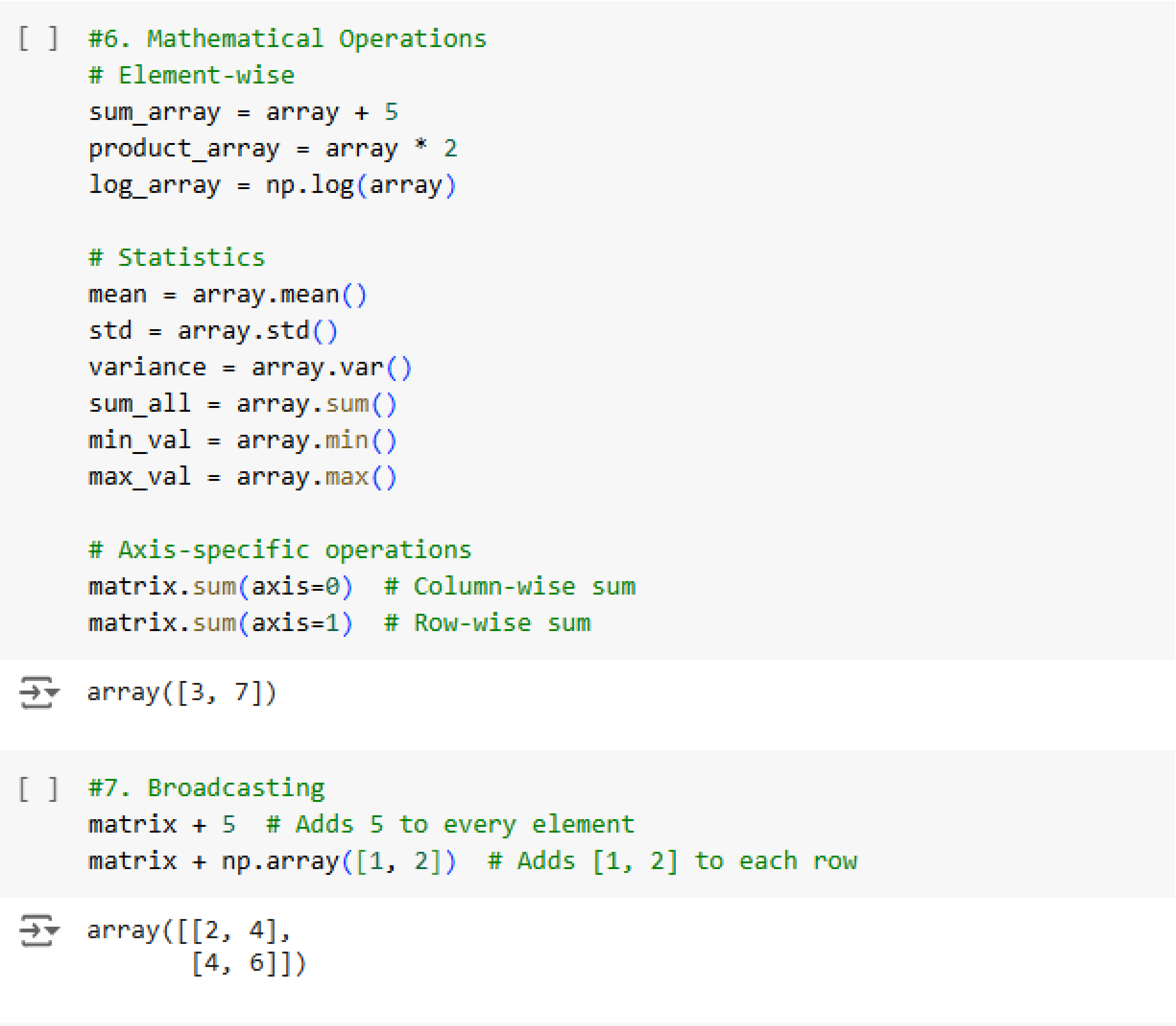


```
#Flattening

flattened = matrix.flatten()  # Convert to 1D

print("Flattened: ", flattened)
```

```
Flattened:  [1 2 3 4]
```

```
#5. Indexing and Slicing
# 1D
array[1]  # Second element
array[1:4]  # Slice (elements 1 to 3)

# 2D
matrix[1, 1]  # Element at row 1, col 1
matrix[:, 1]  # All rows, column 1
matrix[1, :]  # Row 1, all columns
matrix[0:2, 1:3]  # Submatrix
```

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



```
#6. Mathematical Operations
# Element-wise
sum_array = array + 5
product_array = array * 2
log_array = np.log(array)

# Statistics
mean = array.mean()
std = array.std()
variance = array.var()
sum_all = array.sum()
min_val = array.min()
max_val = array.max()

# Axis-specific operations
matrix.sum(axis=0)  # Column-wise sum
matrix.sum(axis=1)  # Row-wise sum
```

```
array([3, 7])
```

```
#7. Broadcasting
matrix + 5  # Adds 5 to every element
matrix + np.array([1, 2])  # Adds [1, 2] to each row
```

```
array([[2, 4],
       [4, 6]])
```

```
#7. Broadcasting
matrix + 5  # Adds 5 to every element
matrix + np.array([1, 2])  # Adds [1, 2] to each row
```

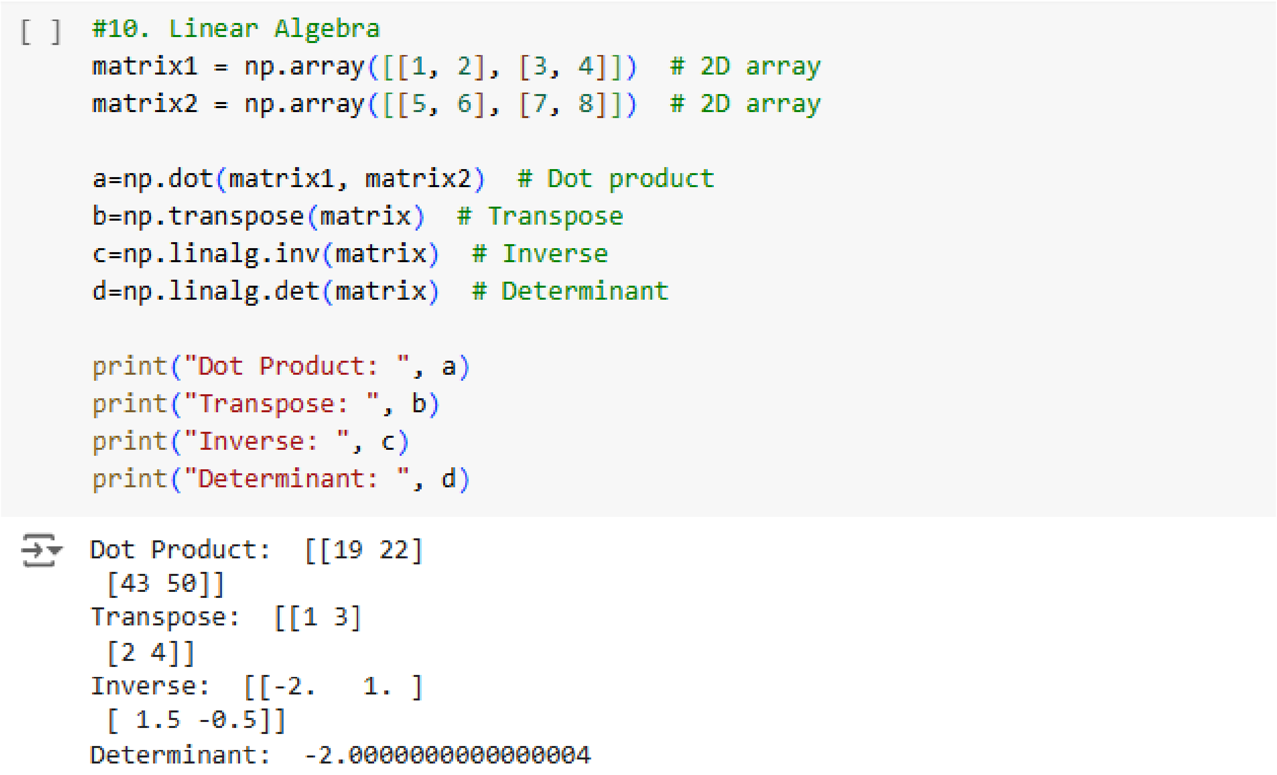
```
array([[2, 4],
       [4, 6]])
```

```
#8. Boolean Indexing
array[array > 5]  # Elements greater than 5
array[(array > 2) & (array < 8)]  # Elements between 2 and 8
```

```
array([3])
```

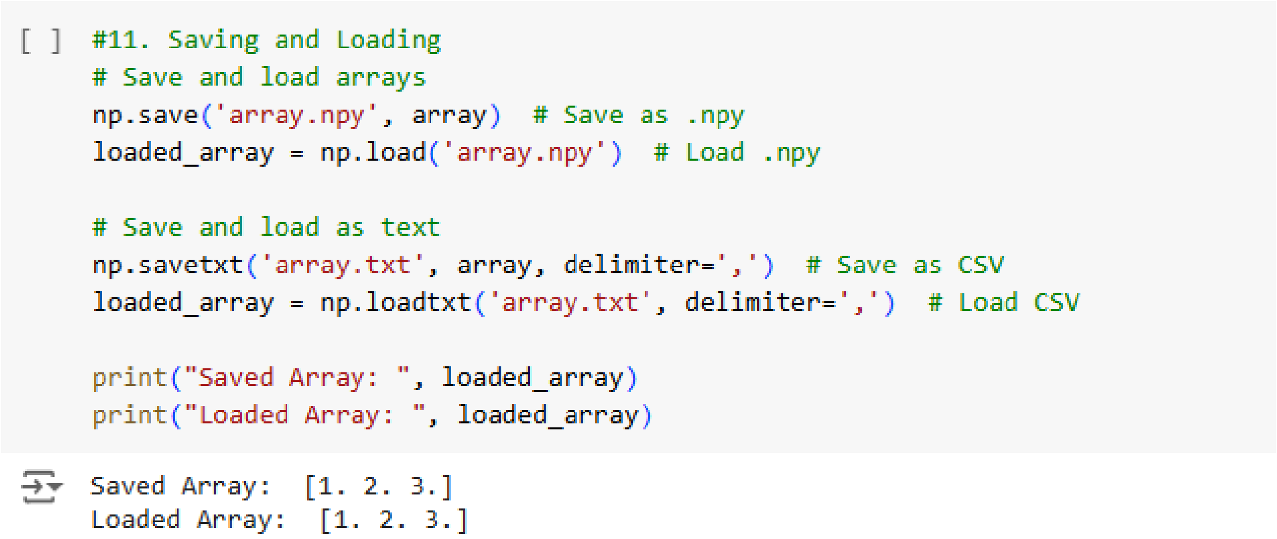
```
#9. Useful Functions
np.unique(array)  # Unique values
np.sort(array)  # Sorted array
np.argsort(array)  # Indices of sorted array
np.where(array > 5, 1, 0)  # If > 5, replace with 1, else 0
```

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



```
[ ] #10. Linear Algebra
    matrix1 = np.array([[1, 2], [3, 4]])  # 2D array
    matrix2 = np.array([[5, 6], [7, 8]])  # 2D array

    a=np.dot(matrix1, matrix2)  # Dot product
    b=np.transpose(matrix)  # Transpose
    c=np.linalg.inv(matrix)  # Inverse
    d=np.linalg.det(matrix)  # Determinant

    print("Dot Product: ", a)
    print("Transpose: ", b)
    print("Inverse: ", c)
    print("Determinant: ", d)
```

```
Dot Product:  [[19 22]
 [43 50]]
Transpose:  [[1 3]
 [2 4]]
Inverse:  [[-2.   1. ]
 [ 1.5 -0.5]]
Determinant:  -2.0000000000000004
```



```
[ ] #11. Saving and Loading
    # Save and load arrays
    np.save('array.npy', array)  # Save as .npy
    loaded_array = np.load('array.npy')  # Load .npy

    # Save and load as text
    np.savetxt('array.txt', array, delimiter=',')  # Save as CSV
    loaded_array = np.loadtxt('array.txt', delimiter=',')  # Load CSV

    print("Saved Array: ", loaded_array)
    print("Loaded Array: ", loaded_array)
```

```
Saved Array:  [1. 2. 3.]
Loaded Array:  [1. 2. 3.]
```