Importing **numpy**

```
import numpy as np
```

Exercise 1:

Create a 1D array with values ranging from 0 to 9.

```
import numpy as np
arr = np.arange(10)
arr
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Exercise 2:

Convert a 1D array to a 2D array with 2 rows.

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

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

Exercise 3:

Multiply a 5x3 matrix by a 3x2 matrix.

```
arr1 = np.arange(15).reshape((5,3))
arr2 = np.arange(6).reshape((3,2))

prod = np.dot(arr1, arr2)

print("Arr1: ",arr1)
print("Arr2: ",arr2)
print("Prod: ",prod)
```

```
Arr1:  [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
Arr2:  [[0 1]
 [2 3]
 [4 5]]
Prod:  [[ 10  13]
 [ 28  40]
 [ 46  67]
```

```
 [ 64  94]
 [ 82 121]]
```

Exercise 4:

Extract all odd numbers from an array of 1-10.

```
arr = np.arange(10)

odd_arr = arr[arr%2==1]

odd_arr
```

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

Exercise 5:

Replace all odd numbers in an array of 1-10 with -1.

```
arr = np.arange(1,10)
arr[arr%2 == 1] = -1
arr
```

```
array([-1,  2, -1,  4, -1,  6, -1,  8, -1])
```

Exercise 6:

Convert a 1D array to a boolean array where all positive values become True.

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

bool_arr = arr > 0

bool_arr
```

```
array([ True, False,  True, False,  True, False])
```

Exercise 7:

Replace all even numbers in a 1D array with their negative

```
arr = np.arange(1,20)

arr[arr%2==0] *= -1

arr
```

```
array([  1,  -2,   3,  -4,   5,  -6,   7,  -8,   9, -10,  11, -12,
13,
       -14,  15, -16,  17, -18,  19])
```

Exercise 8:

Create a random 3x3 matrix and normalize it.

```
matrix = np.random.random((3,3))

print(matrix)

normalized_matrix = (matrix - np.mean(matrix))/np.std(matrix)

print(normalized_matrix)
```

```
[[0.40664817 0.59056952 0.37785327]
 [0.30422729 0.28030095 0.01010885]
 [0.26534306 0.67981854 0.0118612 ]]
[[ 0.38206516  1.24474033  0.24700391]
 [-0.09833555 -0.21056103 -1.47788541]
 [-0.28072034  1.66335904 -1.46966611]]
```

Exercise 9:

Calculate the sum of the diagonal elements of a 3x3 matrix.

```
arr = np.arange(1,10).reshape(3,3)

print(arr)

diagonal_sum = np.trace(arr)

print(diagonal_sum)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
15
```

Exercise 10:

Find the indices of non-zero elements from [1,2,0,0,4,0].

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

non_zero_indices = np.nonzero(arr)

non_zero_indices
```

```
(array([0, 1, 4]),)
```

Exercise 11:

Reverse a 1D array (first element becomes the last).

```
arr = np.arange(1,10)

arr = np.flip(arr)

arr
```

```
array([9, 8, 7, 6, 5, 4, 3, 2, 1])
```

Exercise 12:

Create a 3x3 identity matrix

```
identity = np.eye(3)

identity
```

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

Exercise 13:

Reshape a 1D array to a 2D array with 5 rows and 2 columns.

```
arr = np.arange(10).reshape(5,2)

arr
```

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

Exercise 14:

Stack two arrays vertically.

```
arr1 = np.array([1,2,3,4,5,6])

arr2 = np.array([3,4,5,7,4,6])

stacked_array = np.vstack((arr1,arr2))

stacked_array
```

```
array([[1, 2, 3, 4, 5, 6],
       [3, 4, 5, 7, 4, 6]])
```

Exercise 15:

Get the common items between two arrays.

```
arr1 = np.array([1, 2, 3, 4, 5])

arr2 = np.array([3, 4, 5, 6, 7])

common_items = np.intersect1d(arr1, arr2)

print(common_items)
```

```
[3 4 5]
```

Exercise 16:

Create a 5x5 matrix with row values ranging from 0 to 4.

```
matrix = np.zeros((5,5))

matrix += np.array([0,1,2,3,4])

matrix
```

```
array([[0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.]])
```

Exercise 17:

Find the index of the maximum value in a 1D array.

```
arr = np.array([1,2,3,4,5,6,67,8,9])

max_index = np.argmax(arr)

max_index
```

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

Exercise 18:

Normalize the values in a 1D array between 0 and 1.

```
arr = np.array([2, 5, 8, 9, 12, 56])

normalized_arr = (arr - np.min(arr)) / (np.max(arr) - np.min(arr))

print(normalized_arr)
```

```
[0.         0.05555556 0.11111111 0.12962963 0.18518519 1.        ]
```

Exercise 19:

Calculate the dot product of two arrays

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

dot_product = np.dot(arr1, arr2)

print(dot_product)
```

```
32
```

Exercise 20:

Count the number of elements in an array within a specific range.

```
arr = np.array([1,2,3,4,5,6,45,7,7,5,4,4,5,3,4,4,4,7,7,6,8,9,0])

count_within_range = np.sum((arr>=5) & (arr<=9))

count_within_range
```

```
np.int64(11)
```

Exercise 21:

Find the mean of each row in a 2D array.

```
arr = np.arange(15).reshape(3,5)

mean = np.mean(arr, axis=1)

mean
```

```
array([ 2.,  7., 12.])
```

Exercise 22:

Create a random 4x4 matrix and extract the diagonal elements.

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

print(arr)

diagonal_elements = np.diag(arr)

print(diagonal_elements)
```

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

Exercise 23:

Count the number of occurrences of a specific value in an array.

```
def countSpecificNum(arr, num):
    return np.sum((arr==num))

arr = np.array([1,2,13,4,5,34,6,7,8,5,44,6,45,4,4,3,22,2,2,2])

print(countSpecificNum(arr, 2))
```

```
4
```

Exercise 24:

Replace all values in a 1D array with the mean of the array.

```
arr = np.array([1,2,13,4,5,34,6,7,8,5,44,6,45,4,4,3,22,2,2,2])

arr[:] = np.mean(arr)

print(arr)
```

```
[10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10]
```

Exercise 25:

Find the indices of the maximum and minimum values in a 1D array.

```
arr =
np.array([1,2,4,3,546,45,634,324,2,234,5,3,35,65,6,7,5,34,34,6,36,36,3
6,3,74,86,85])

min = np.argmin(arr)

max = np.argmax(arr)

print(min, max)
```

```
0 6
```

Exercise 26:

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

```
arr = np.ones((5,5))

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

arr
```

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

Exercise 27:

Find the unique values and their counts in a 1D array.

```
arr = np.array([1,2,1,33,1,3,3,4,3,4,5,6,4,5,6,5,3,3,53,3,2,4,3,2,2])

unique, counts = np.unique(arr, return_counts=True)

print(unique)

print(counts)
```

```
[ 1  2  3  4  5  6 33 53]
[3 4 7 4 3 2 1 1]
```

Exercise 28:

Create a 3x3 matrix with values ranging from 0 to 8.

```
arr = np.arange(9).reshape(3,3)

arr
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Exercise 29:

Calculate the exponential of all elements in a 1D array.

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

exponential_arr = np.exp(arr)

print(exponential_arr)
```

```
[  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]
```

Exercise 30:

Swap two rows in a 2D array.

```
arr = np.array(range(15)).reshape(3,5)

arr[[1,2]] = arr[[2,1]]

arr
```

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

Exercise 31:

Create a random 3x3 matrix and replace all values greater than 0.5 with 1 and all others with 0.

```
arr = np.random.random((3,3))

arr[arr > 0.5] = 1

arr[arr <=0.5] = 0

arr
```

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

Exercise 32:

Find the indices of the top N maximum values in a 1D array.

```
arr = np.array([1,2,3,345,4,6,5,4,3,3,5,3,3,42,41,2,5,2])

top_indices = np.argsort(arr)[-5:]

top_indices
```

```
array([10,  5, 14, 13,  3])
```

Exercise 33:

Calculate the mean of each column in a 2D array.

```
matrix = np.arange(15).reshape(5,3)

print(matrix)

column_means = np.mean(matrix, axis=0)
```

```
print(column_means)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
[6. 7. 8.]
```

Exercise 34:

Normalize the values in each column of a 2D array.

```
matrix = np.arange(15).reshape(5,3)

print(matrix)

normalized_matrix = (matrix - np.mean(matrix,
axis=0))/np.std(matrix,axis=0)

print(normalized_matrix)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
[[-1.41421356 -1.41421356 -1.41421356]
 [-0.70710678 -0.70710678 -0.70710678]
 [ 0.          0.          0.        ]
 [ 0.70710678  0.70710678  0.70710678]
 [ 1.41421356  1.41421356  1.41421356]]
```

Exercise 35:

Concatenate two 1D arrays.

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

concatenated_arr = np.concatenate((arr1, arr2))

print(concatenated_arr)
```

```
[1 2 3 4 5 6]
```

Exercise 36:

Create a 2D array with random values and sort each row.

```
arr = np.array([1,2,4,5,43,3,2,4,4,2,2,1,1,1,4,6,7,4]).reshape(2,-1)

print(arr)

sorted_arr = np.sort(arr, axis=1)

print(sorted_arr)
```

```
[[ 1  2  4  5 43  3  2  4  4]
 [ 2  2  1  1  1  4  6  7  4]]
[[ 1  2  2  3  4  4  4  5 43]
 [ 1  1  1  2  2  4  4  6  7]]
```

Exercise 37:

Compute the mean squared error between two arrays.

```
arr1 = np.array([1, 2, 3, 4])

arr2 = np.array([2, 3, 4, 5])

mse = np.mean((arr1-arr2)**2)

print(mse)
```

```
1.0
```

Exercise 38:

Replace all negative values in an array with 0.

```
arr = np.array([1,2,1,3,2,2,3,-1,-3,-53,43,3,4,3,3,-45,2])

arr[arr<0] = 0

arr
```

```
array([ 1,  2,  1,  3,  2,  2,  3,  0,  0,  0, 43,  3,  4,  3,  3,  0,
2])
```

Exercise 39:

Find the 5th and 95th percentiles of an array.

```
arr = np.array([1,2,2,4,4,3,4,3,2,4,5,6,6,5,7,8,9,9,7,6,5,5,5])

percentile_5th = np.percentile(arr, 5)

percentile_95th = np.percentile(arr, 95)

print("5th Percentile:", percentile_5th)
```

```
print("95th Percentile:", percentile_95th)
```

```
5th Percentile: 2.0
95th Percentile: 8.899999999999999
```

Exercise 40:

Create a random 2x2 matrix and compute its determinant.

```
matrix = np.random.random((3,3))

det = np.linalg.det(matrix)

det
```

```
np.float64(-0.33597499420634197)
```

Exercise 41:

Count the number of elements in an array that are greater than the mean.

```
arr = np.arange(10)

mean = np.mean(arr)

count_above_mean = np.sum(arr > mean)

count_above_mean
```

```
np.int64(5)
```

Exercise 42:

Calculate the square root of each element in a 1D array.

```
arr = np.arange(10)

sqrt_arr = np.sqrt(arr)

sqrt_arr
```

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

Exercise 43:

Create a 3x3 matrix and compute the matrix square root.

```
matrix = np.arange(9).reshape(3,3)
```

```
matrix_sqrt = np.linalg.matrix_power(matrix, 2)

matrix_sqrt
```

```
array([[ 15,  18,  21],
       [ 42,  54,  66],
       [ 69,  90, 111]])
```

Exercise 44:

Convert the data type of an array to float.

```
arr = np.array([1, 2, 3, 4], dtype=int)

float_arr = arr.astype(float)

print(float_arr)
```

```
[1. 2. 3. 4.]
```

Exercise 45:

Calculate the element-wise absolute values of an array.

```
arr = np.random.random(10)

arr = np.abs(arr)

arr
```

```
array([0.31185165, 0.61674077, 0.81222901, 0.28951428, 0.46198019,
       0.09033235, 0.51626975, 0.37368957, 0.51384   , 0.50407205])
```

Exercise 46:

Find the indices where elements of two arrays match.

```
arr1 = np.array([1,2,6,4,5])

arr2 = np.array([1,2,3,4,5])

intersection = np.where(arr1 == arr2)

intersection
```

```
(array([0, 1, 3, 4]),)
```

Exercise 47:

Calculate the cumulative sum of elements in a 1D array.

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

cumulative_sum = np.cumsum(arr)

print(cumulative_sum)
```

```
[ 1  3  6 10 15]
```

Exercise 48:

Compute the inverse of a 2x2 matrix.

```
matrix = np.random.random((2, 2))

inverse_matrix = np.linalg.inv(matrix)

print(inverse_matrix)
```

```
[[-0.19164845  1.50542516]
 [ 1.38809013 -1.52076013]]
```

Exercise 49:

Count the number of non-zero elements in a 2D array.

```
matrix = np.array([[0, 1, 0], [2, 0, 3], [0, 4, 0]])

non_zero_count = np.count_nonzero(matrix)

print(non_zero_count)
```

```
4
```

Exercise 50:

Create a 2D array and replace all nan values with 0.

```
matrix = np.array([[1, np.nan, 3], [4, 5, np.nan], [7, 8, 9]])

matrix[np.isnan(matrix)] = 0

print(matrix)
```

```
[[1. 0. 3.]
 [4. 5. 0.]
 [7. 8. 9.]]
```

Exercise 51:

Find the correlation coefficient between two arrays.

```
arr1 = np.array([1, 2, 3, 4, 5])

arr2 = np.array([3, 4, 5, 6, 7])

correlation_coefficient = np.corrcoef(arr1, arr2)[0, 1]

print(correlation_coefficient)
```

```
0.9999999999999999
```

Exercise 52:

Create a 1D array and remove all duplicate values.

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

unique_arr = np.unique(arr)

print(unique_arr)
```

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

Exercise 53:

Compute the element-wise product of two arrays.

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

elementwise_product = np.multiply(arr1, arr2)

print(elementwise_product)
```

```
[ 4 10 18]
```

Exercise 54:

Calculate the standard deviation of each column in a 2D array.

```
matrix = np.random.random((4, 3))

column_stddev = np.std(matrix, axis=0)

print(column_stddev)
```

```
[0.23174062 0.20571746 0.1133317 ]
```

Exercise 55:

Create a 2D array and set all values above a certain threshold to that threshold.

```
matrix = np.random.random((5,5))

threshold = 0.7

matrix[matrix > threshold] = threshold

matrix
```

```
array([[0.7       , 0.7       , 0.61664415, 0.36153169, 0.7       ],
       [0.34058178, 0.60615516, 0.25606929, 0.48004909, 0.4160477 ],
       [0.21846163, 0.63571649, 0.7       , 0.69989727, 0.7       ],
       [0.7       , 0.28382798, 0.07672507, 0.33055863, 0.7       ],
       [0.13652981, 0.15322908, 0.5089672 , 0.1759915 , 0.69817983]])
```

Exercise 56:

Create a random 5x5 matrix and replace the maximum value by -1.

```
matrix = np.random.random((5,5))

max_val_index = np.unravel_index(np.argmax(matrix), matrix.shape)

matrix[max_val_index] = -1

matrix
```

```
array([[ 0.65844028,  0.06966707,  0.79188262,  0.72509655,
0.54358994],
       [ 0.41006894,  0.7731639 ,  0.55467873, -1.        ,
0.02983383],
       [ 0.76918083,  0.07298825,  0.40447898,  0.83341763,  0.2715571
],
       [ 0.67597477,  0.60860731,  0.05866259,  0.8353305 ,
0.28681344],
       [ 0.01963475,  0.00940313,  0.54897905,  0.51532883,
0.85243046]])
```

Exercise 57:

Convert a 1D array of Fahrenheit temperatures to Celsius.

```
fahrenheit_temps = np.array([32, 68, 100, 212])

celsius_temps = (fahrenheit_temps - 32) * 5/9

print(celsius_temps)
```

```
[  0.          20.          37.77777778 100.        ]
```

Exercise 58:

Compute the outer product of two arrays.

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

outer_product = np.outer(arr1, arr2)

print(outer_product)
```

```
[[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
```

Exercise 59:

Create a 1D array with 10 equidistant values between 0 and 1.

```
equidistant_arr = np.linspace(0, 1, 10)

print(equidistant_arr)
```

```
[0.         0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
 0.66666667 0.77777778 0.88888889 1.        ]
```

Exercise 60:

Compute the cross product of two 3D arrays.

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

cross_product = np.cross(arr1, arr2)

print(cross_product)
```

```
[-3  6 -3]
```

Exercise 61:

Calculate the percentile along a specific axis of a 2D array.

```
matrix = np.random.random((3, 4))

percentiles_axis1 = np.percentile(matrix, 75, axis=1)

print(percentiles_axis1)
```

```
[0.49520653 0.82536149 0.67340904]
```

Exercise 62:

Create a 1D array and add a border of 0s around it.

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

arr_with_border = np.pad(arr, (1, 1), mode='constant',
constant_values=0)

print(arr_with_border)
```

```
[0 1 2 3 4 0]
```

Exercise 63:

Compute the histogram of a 1D array.

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

hist, bins = np.histogram(arr, bins=[1, 2, 3, 4])

print("Histogram:", hist)

print("Bin edges:", bins)
```

```
Histogram: [2 3 4]
Bin edges: [1 2 3 4]
```

Exercise 64:

Create a 2D array with random values and normalize each row.

```
matrix = np.random.random((4, 3))

normalized_rows = matrix / np.linalg.norm(matrix, axis=1,
keepdims=True)

print(normalized_rows)
```

```
[[0.39361705 0.26406977 0.88052983]
 [0.47796161 0.58025384 0.65943777]
 [0.72243792 0.45736546 0.51855597]
 [0.63511903 0.51800677 0.57296842]]
```

Exercise 65:

Create a random 2D array and sort it by the second column.

```
matrix = np.random.random((3, 4))

sorted_matrix_by_column2 = matrix[matrix[:, 1].argsort()]
```

```
print(sorted_matrix_by_column2)
```

```
[[0.5397669  0.42059072 0.59910447 0.36911212]
 [0.69595643 0.75800702 0.57659647 0.99641196]
 [0.70707283 0.94256978 0.12162756 0.72313119]]
```

Exercise 66:

Calculate the determinant of a 3x3 matrix.

```
matrix = np.random.random((3, 3))

determinant = np.linalg.det(matrix)

print(determinant)
```

```
0.11492573230831755
```

Exercise 67:

Calculate the element-wise exponentiation of a 1D array.

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

exponentiated_arr = np.exp(arr)

print(exponentiated_arr)
```

```
[ 7.3890561  20.08553692 54.59815003]
```

Exercise 68:

Calculate the Frobenius norm of a 2D array.

```
matrix = np.random.random((3, 4))

frobenius_norm = np.linalg.norm(matrix)

print(frobenius_norm)
```

```
2.31144069977028
```

Exercise 69:

Create a 2D array with random values and replace the maximum value with the minimum.

```
matrix = np.random.random((3, 4))

max_value_index = np.unravel_index(np.argmax(matrix), matrix.shape)
```

```
min_value = np.min(matrix)

matrix[max_value_index] = min_value

print(matrix)
```

```
[[0.36966875 0.13369273 0.49680474 0.08246654]
 [0.14499955 0.20420326 0.87398942 0.55191384]
 [0.08246654 0.24527059 0.24514376 0.36157058]]
```

Exercise 70:

Compute the matrix multiplication of two 2D arrays.

```
matrix1 = np.random.random((3, 4))

matrix2 = np.random.random((4, 5))

matrix_multiplication = np.dot(matrix1, matrix2)

print(matrix_multiplication)
```

```
[[0.26295296 1.2221307  1.44507405 1.35712679 0.96082222]
 [0.23977892 1.31504053 1.6159018  1.44164087 1.02755   ]
 [0.22443044 1.06504316 1.00676121 1.23954536 1.03785833]]
```

Exercise 71:

Create a 1D array and set the values between 10 and 20 to 0.

```
arr = np.array([5, 15, 12, 18, 25])

arr[(arr >= 10) & (arr <= 20)] = 0

print(arr)
```

```
[ 5  0  0  0 25]
```

Exercise 72:

Compute the inverse hyperbolic sine of each element in a 1D array.

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

inverse_sineh_arr = np.arcsinh(arr)

print(inverse_sineh_arr)
```

```
[0.88137359 1.44363548 1.81844646 2.09471255]
```

Exercise 73:

Compute the Kronecker product of two arrays.

```
arr1 = np.array([1, 2])

arr2 = np.array([3, 4])

kronecker_product = np.kron(arr1, arr2)

print(kronecker_product)
```

```
[3 4 6 8]
```

Exercise 74:

Calculate the mean absolute deviation of a 1D array.

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

mean_absolute_deviation = np.mean(np.abs(arr - np.mean(arr)))

print(mean_absolute_deviation)
```

```
1.2
```

Exercise 75:

Create a 3x3 matrix and set all values above the main diagonal to zero.

```
matrix = np.random.random((3, 3))

matrix[np.triu_indices(3, 1)] = 0

print(matrix)
```

```
[[0.2002779  0.         0.        ]
 [0.9133617  0.68914002 0.        ]
 [0.00822214 0.28599613 0.96752018]]
```

Exercise 76:

Count the number of occurrences of each unique value in a 1D array.

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

unique_values, counts = np.unique(arr, return_counts=True)

print("Unique values:", unique_values)

print("Counts:", counts)
```

```
Unique values: [1 2 3 4]
Counts: [1 2 3 1]
```

Exercise 77:

Compute the cumulative product of elements along a given axis in a 2D array.

```
matrix = np.random.random((3, 4))

cumulative_product_axis0 = np.cumprod(matrix, axis=0)

print(cumulative_product_axis0)
```

```
[[0.06998515 0.15095811 0.85915806 0.64004544]
 [0.00333249 0.05832544 0.6219033  0.2224238 ]
 [0.00177066 0.0386447  0.02331987 0.16742843]]
```

Exercise 78:

Round elements of a 1D array to the nearest integer.

```
arr = np.array([1.2, 2.7, 3.5, 4.9])

rounded_arr = np.round(arr)

print(rounded_arr)
```

```
[1. 3. 4. 5.]
```

Exercise 79:

Create a 1D array and append a new element to the end.

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

new_element = 4

arr = np.append(arr, new_element)

print(arr)
```

```
[1 2 3 4]
```

Exercise 80:

Calculate the element-wise absolute difference between two arrays.

```
arr1 = np.array([3, 7, 1, 10, 4])

arr2 = np.array([2, 5, 8, 1, 7])
```

```
absolute_difference = np.abs(arr1 - arr2)

print(absolute_difference)
```

```
[1 2 7 9 3]
```

Exercise 81:

Create a 2D array with random values and replace the maximum value in each row with -1.

```
matrix = np.random.random((3, 4))

max_values_indices = np.argmax(matrix, axis=1)

matrix[np.arange(matrix.shape[0]), max_values_indices] = -1

print(matrix)
```

```
[[-1.          0.66833848  0.19546939  0.15771911]
 [-1.          0.5865088   0.09942719  0.1740521 ]
 [ 0.01976112 -1.          0.05670424  0.619874  ]]
```

Exercise 82:

Normalize the columns of a 2D array to have a sum of 1.

```
matrix = np.random.random((3, 4))

normalized_columns = matrix / np.sum(matrix, axis=0, keepdims=True)

print(normalized_columns)
```

```
[[0.10755237 0.41785867 0.41635855 0.29955895]
 [0.46510048 0.30574235 0.1930727  0.31329433]
 [0.42734714 0.27639898 0.39056874 0.38714673]]
```

Exercise 83:

Find the indices of the top N minimum values in a 1D array.

```
arr = np.array([10, 5, 8, 1, 7])

top_indices = np.argsort(arr)[:2]

print(top_indices)
```

```
[3 1]
```

Exercise 84:

Convert the elements of a 1D array to strings.

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

string_arr = arr.astype(str)

print(string_arr)
```

```
['1' '2' '3' '4']
```

Exercise 85:

Compute the percentile rank of each element in a 1D array.

```
from scipy.stats import percentileofscore

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

# Calculate percentile rank for each element in arr

percentile_rank = np.array([percentileofscore(arr, value) for value in
arr])

print(percentile_rank)
```

```
[ 20.  40.  60.  80. 100.]
```

Exercise 86:

Create a 1D array and shuffle its elements randomly.

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

np.random.shuffle(arr)

print(arr)
```

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

Exercise 87:

Check if all elements in a 1D array are non-zero.

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

all_nonzero = np.all(arr != 0)

print(all_nonzero)
```

```
True
```

Exercise 88:

Find the indices of the maximum value in each row of a 2D array.

```
matrix = np.random.random((3, 4))

max_indices_per_row = np.argmax(matrix, axis=1)

print(max_indices_per_row)
```

```
[1 3 3]
```

Exercise 89:

Create a 2D array and replace all nan values with the mean of the array.

```
matrix = np.array([[1, np.nan, 3], [4, 5, np.nan], [7, 8, 9]])

nan_mean = np.nanmean(matrix)

matrix[np.isnan(matrix)] = nan_mean

print(matrix)
```

```
[[1.         5.28571429 3.        ]
 [4.         5.         5.28571429]
 [7.         8.         9.        ]]
```

Exercise 90:

Calculate the mean of each row in a 2D array ignoring nan values.

```
matrix = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])

row_means_ignore_nan = np.nanmean(matrix, axis=1)

print(row_means_ignore_nan)
```

```
[1.5 5.  8. ]
```

Exercise 91:

Compute the sum of diagonal elements in a 2D array.

```
matrix = np.random.random((3, 3))

diagonal_sum = np.trace(matrix)

print(diagonal_sum)
```

```
1.3848216665202455
```

Exercise 92:

Convert radians to degrees for each element in a 1D array.

```
arr_in_radians = np.array([np.pi/2, np.pi, 3*np.pi/2])

arr_in_degrees = np.degrees(arr_in_radians)

print(arr_in_degrees)
```

```
[ 90. 180. 270.]
```

Exercise 93:

Calculate the pairwise Euclidean distance between two arrays.

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

euclidean_distance = np.linalg.norm(arr1 - arr2)

print(euclidean_distance)
```

```
5.196152422706632
```

Exercise 94:

Create a 1D array and set the values between the 25th and 75th percentile to 0.

```
arr = np.array([10, 20, 30, 40, 50])

percentile_25th = np.percentile(arr, 25)

percentile_75th = np.percentile(arr, 75)

arr[(arr >= percentile_25th) & (arr <= percentile_75th)] = 0

print(arr)
```

```
[10  0  0  0 50]
```

Exercise 95:

Calculate the element-wise square of the difference between two arrays.

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

squared_difference = (arr1 - arr2)**2
```

```
print(squared_difference)
```

```
[9 9 9]
```

Exercise 96:

Replace all even numbers in a 1D array with the next odd number.

```
arr = np.array([2, 5, 8, 12, 15])

arr[arr % 2 == 0] += 1

print(arr)
```

```
[ 3  5  9 13 15]
```

Exercise 97:

Create a 2D array and normalize each column by its range..

```
matrix = np.random.random((3, 4))

normalized_columns_range = (matrix - np.min(matrix, axis=0)) /
(np.max(matrix, axis=0) - np.min(matrix, axis=0))

print(normalized_columns_range)
```

```
[[0.         0.87502617 0.69747972 1.        ]
 [0.14947338 1.         1.         0.        ]
 [1.         0.         0.         0.82647549]]
```

Exercise 98:

Compute the cumulative sum of elements along a given axis in a 2D array.

```
matrix = np.random.random((3, 4))

cumulative_sum_axis1 = np.cumsum(matrix, axis=1)

print(cumulative_sum_axis1)
```

```
[[0.0950331  0.73823707 1.49742809 1.7696128 ]
 [0.35113253 1.02664904 1.95150933 2.13362176]
 [0.72458396 1.62665096 2.62421113 3.4634039 ]]
```

Exercise 99:

Check if any element in a 1D array is non-zero.

```
arr = np.array([0, 0, 0, 1, 0])

any_nonzero = np.any(arr != 0)

print(any_nonzero)
```

```
True
```

Exercise 100:

Create a 2D array with random integers and replace all values greater than a certain threshold with that threshold.

```
matrix = np.random.randint(0, 100, size=(3, 4))

threshold = 75

matrix[matrix > threshold] = threshold

print(matrix)
```

```
[[10 55  0 69]
 [66 36  0 12]
 [23 23 16 74]]
```