```
In [1]: import numpy as np
        import pandas as pd

In [ ]: #1.Create a Numpy 'arr' of integers from 0 to 5 and print its data type.

In [4]: arr = np.arange(6)
        print(arr)
        print(arr.dtype)

[0 1 2 3 4 5]
int64

In [ ]: #2. Given a Numpy array 'arr' , check if its data type is float64.

In [5]: arr = np.array([1.5 , 2.6 , 3.7])
        print(arr.dtype == np.float64)

True

In [ ]: #3. Create a Numpy array 'arr' with a data type of complex 128 containing three complex numbers.

In [8]: array = np.array([1+2j , 2+3j , 3+4j] , dtype = np.complex128)
        print(array)
        print(array.dtype)

[1.+2.j 2.+3.j 3.+4.j]
complex128

In [ ]: #4.Convert an existing Numpy array 'arr' of integers to float32 data type

In [9]: array = np.array([1,2,3,4,5])
        array = array.astype(np.float32)
        print(array)
        print(array.dtype)

[1. 2. 3. 4. 5.]
float32

In [ ]: #5.Given a Numpy array 'arr' with float64 data type , convert it to float32 to reduce decimal precision.

In [10]: arr = np.array([1.123456789 , 2.23456789 , 3.3456789])
         arr = arr.astype(np.float32)
         print(arr)
         print(arr.dtype)

[1.1234568 2.2345679 3.3456788]
float32

In [ ]: #6. Write a function array_attributes that takes a Numpy array as input and returns its shape , size , and data type.

In [11]: array = np.array([[1 , 2 , 3], [4,5,6]])
         print(array.shape,array.size,array.dtype)

(2, 3) 6 int64

In [ ]: #7. Create a function array_dimension that takes a Numpy array as input and returns its dimensionality.

In [12]: arrya = np.array([[1,2,3,],[4,5,6]])
         print(array.ndim)

2

In [ ]: #8. Design a function item_size_info that takes a Numpy array as input and returns the item size and the total size in bytes.

In [14]: array =np.array([[1,2,3],[4,5,6]])
         print("Item size :", array.itemsize)
         print("Total size: ", array.nbytes)

Item size : 8
Total size:  48

In [ ]: #9. Create a function array_strides that takes a Numpy array as input and returns strides of the array.

In [15]: array = np.array([[1,2,3], [4,5,6]])
         print("Strides: ", array.strides)

Strides:  (24, 8)

In [ ]: #10. Design a function shape_stride_relationship that takes a Numpy array as input and returns the shape and strides of the array.

In [16]: array = np.array([[1,2,3], [4,5,6]])
         print("Shape: " , array.shape)
         print("Strides: ", array.strides)

Shape:  (2, 3)
Strides:  (24, 8)

In [ ]: #11.Create a function `create_zeros_array` that takes an intefer`n` as input and returns a Numpy array of zeros with`n` elements.

In [17]: n = 5
         array = np.zeros(n)
         print(array)

[0. 0. 0. 0. 0.]

In [ ]: #12.Write a function`create_ones_matrix` that takes integers `rows` and `cols` as inputs and generate a 2D NUmpy array filled with ones of size `rows x cols`.

In [19]: rows = 3
         cols = 4
         matrix = np.ones((rows, cols))
         print(matrix)

[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

In [ ]:  #13.Write a function `generate_range_array` that takes three integers start, stop, and step as arguments and creates a NumPy array with a range starting from `start`, ending at stop (exclusive), and with

In [21]: start =1
         stop = 10
         step = 2
         array = np.arange(start , stop , step)
         print(array)

[1 3 5 7 9]

In [ ]: #14. Design a function `generate_linear_space` that takes two floats `start`, `stop`, and an integer `num` as arguments and generates a NumPy array with num equally spaced values between `start` and `stop

In [22]: start =1.0
         stop = 10.0
         num = 5
         array = np.linspace(start , stop , num)
         print(array)

[ 1.    3.25  5.5   7.75 10.  ]

In [ ]: #15. Create a function `create_identity_matrix` that takes an integer `n` as input and generates a square identity matrix of size `n x n` using `numpy.eye`.

In [23]: n = 4
         print(np.eye(n))

[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

In [ ]: #16.Write a function that takes a Python list and converts it into a NumPy array.

In [26]: input_list = [1,2,3,4,5]
         print(np.array(input_list))

[1 2 3 4 5]

In [ ]: #17.Create a NumPy array and demonstrate the use of `numpy.view` to create a new array object with the same data

In [27]: original_array = np.array([4,2,5,1,3])
         print(np.sort(original_array))

[1 2 3 4 5]

In [ ]: #18. Write a function that takes two NumPy arrays and concatenates them along a specified axis.

In [28]: a , b = np.array([[1,2], [3,4]]), np.array([[5,6], [7,8]])
         print(np.concatenate((a,b),0))
         print(np.concatenate((a,b),1))

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

In [ ]: #19.Create two NumPy arrays with different shapes and concatenate them horizontally using `numpy.concatenate`.

In [33]: array1 = np.array([[1,2],[3,4]])
         array2 = np.array([[5,6,7],[8,9,10]])
         print(np.concatenate((array1,array2),axis =1))

[[ 1  2  5  6  7]
 [ 3  4  8  9 10]]

In [ ]: #20.Write a function that vertically stacks multiple NumPy arrays given as a list.

In [35]: array1 = np.array([1,2])
         array2 = np.array([3,4])
         array3 = np.array([5,6])

         print(np.vstack([np.array([1,2]),np.array([3,4]),np.array([5,6])]))

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

In [ ]: #21. Write a Python function using NumPy to create an array of integers within a specified range (inclusive) with a given step size

In [38]: array = np.arange(1,21 ,2)
         print(array)

[ 1  3  5  7  9 11 13 15 17 19]

In [ ]: #22. Write a Python function using NumPy to generate an array of 10 equally spaced values between 0 and 1 (inclusive).

In [39]: array = np.linspace(0,1,10)
         print(array)

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

In [ ]: #23. Write a Python function using NumPy to create an array of 5 logarithmically spaced values between 1 and 1000 (inclusive).

In [40]: array = np.logspace(0,3,5)
         print(array)

[   1.            5.62341325   31.6227766   177.827941   1000.        ]

In [ ]: #24.Create a Pandas DataFrame using a NumPy array that contains 5 rows and 3 columns, where the values are random integers between 1 and 100.

In [43]: np_array = np.random.randint(1,101, size = (5,3))
         df = pd.DataFrame(np_array,columns =['column A','column B', 'column C'])
         print(df)

   column A  column B  column C
0        43        34        22
1        71       100        28
2        29        37        58
3        48        21        69
4         8        66        52

In [ ]: #25. Write a function that takes a Pandas DataFrame and replaces all negative values in a specific column with zeros. Use NumPy operations within the Pandas DataFrame.

In [44]: df = pd.DataFrame({'A':[1,-2,3,-4,5],
                           'B': [10,20,30,40,50]})
         df['A'] = np.where(df['A'] < 0,0,df['A'])
         print(df)

   A   B
0  1  10
1  0  20
2  3  30
3  0  40
4  5  50

In [ ]: #26.Access the 3rd element from the given NumPy array.

In [47]: arr = np.array([10, 20, 30, 40, 50])
         third_element = arr[2]
         print(third_element)

30

In [ ]: #27.Retrieve the element at index (1, 2) from the 2D NumPy array.

In [49]: arr_2d = np.array([[1, 2, 3],
                            [4, 5, 6],
                            [7, 8, 9]])
         index = (1,2)
         element = arr_2d[1,2]
         print(element)

6

In [ ]: #28.Using boolean indexing, extract elements greater than 5 from the given NumPy array.

In [50]: arr = np.array([3, 8, 2, 10, 5, 7])
         elements_greater_than_5 = arr[arr > 5]
         print(elements_greater_than_5)

[ 8 10  7]

In [ ]: #29.Perform basic slicing to extract elements from index 2 to 5 (inclusive) from the given NumPy array.

In [51]: arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
         sliced_arr = arr[2:6]
         print(sliced_arr)

[3 4 5 6]

In [ ]: #30.Slice the 2D NumPy array to extract the sub-array `[[2, 3], [5, 6]]` from the given array.

In [52]: arr_2d = np.array([[1, 2, 3],
                            [4, 5, 6],
                            [7, 8, 9]])
         sliced_arr = arr_2d[1:3,1:3]
         print(sliced_arr)

[[5 6]
 [8 9]]

In [ ]: #31.Write a NumPy function to extract elements in specific order from a given 2D array based on indices provided in another array.

In [56]: arr_2d = np.array([[1,2,3],[4,5,6],[7,8,9]])
         indices = np.array([[0,1],[1,2]])

         extracted_elements = arr_2d[indices[:,0],indices[:,1]]
         print(extracted_elements)

[2 6]

In [ ]: #32.Create a NumPy function that filters elements greater than a threshold from a given 1D array using boolean indexing.

In [59]: arr = np.array([1,2,3,4,5,6])
         threshold = 3

         filtered_elements = arr[arr >threshold]
         print(filtered_elements)

[4 5 6]

In [ ]: #33.Develop a NumPy function that extracts specific elements from a 3D array using indices provided in three separate arrays for each dimension.

In [60]: arr_3d = np.array([[[10,20], [30,40]], [[50,60],[70,80]]])
         row_indices = np.array([1,0])
         col_indices = np.array([1,1])
         depth_indices = np.array([0,1])
         print(arr_3d[row_indices,col_indices , depth_indices])

[70 40]

In [ ]: #34. Write a NumPy function that returns elements from an array where both two conditions are satisfied using boolean indexing.

In [62]: arr = np.array([1,2,3,4,5,6,7,8,9])
         print(arr[(arr>3) & (arr<7)])

[4 5 6]

In [ ]: #35.Create a NumPy function that extracts elements from a 2D array using row and column indices provided in separate arrays.

In [63]: arr_2d = np.array([[1,2,3],[4,5,6],[7,8,9]])
         row_indices = np.array([0,1,2])
         col_indices = np.array([1,2,0])
         print(arr_2d[row_indices,col_indices])

[2 6 7]

In [ ]: #36.Given an array arr of shape (3, 3), add a scalar value of 5 to each element using NumPy broadcasting.

In [64]: arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
         scaler = 5
         result = arr + scaler
         print(result)

[[ 6  7  8]
 [ 9 10 11]
 [12 13 14]]

In [ ]: #37.Consider two arrays arr1 of shape (1, 3) and arr2 of shape (3, 4). Multiply each row of arr2 by the corresponding element in arr1 using NumPy broadcasting.

In [68]: arr1 = np.array([[2,3,4]])
         arr2 = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
         arr1 = arr1.T
         result = arr1 * arr2
         print(result)

[[ 2  4  6  8]
 [15 18 21 24]
 [36 40 44 48]]

In [ ]: #38.Given a 1D array arr1 of shape (1, 4) and a 2D array arr2 of shape (4, 3), add arr1 to each row of arr2 using NumPy broadcasting.

In [70]: arr1 = np.array([[2,3,4]])
         arr2 = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
         arr1 = arr1.T
         result = arr1 + arr2
         print(result)

[[ 3  4  5  6]
 [ 8  9 10 11]
 [13 14 15 16]]

In [ ]: #39. Consider two arrays arr1 of shape (3, 1) and arr2 of shape (1, 3). Add these arrays using NumPy broadcasting.

In [72]: arr1 = np.array([[1],[2], [3]])
         arr2 = np.array([[4,5,6]])
         result = arr1 + arr2
         print(result)

[[5 6 7]
 [6 7 8]
 [7 8 9]]

In [ ]: #40.Given arrays arr1 of shape (2, 3) and arr2 of shape (2, 2), perform multiplication using NumPy broadcasting. Handle the shape incompatibility.

In [73]: arr1 = np.array([[1,2,3],[4,5,6]])
         arr2 = np.array([[7,8],[9,10]])
         result = arr1[:,:2] * arr2
         print(result)

[[ 7 16]
 [36 50]]

In [ ]: #41.Calculate column wise mean for the given array:

In [76]: arr = np.array([[1, 2, 3], [4, 5, 6]])
         column_mean = np.mean(arr,axis =0)
         print(column_mean)

[2.5 3.5 4.5]

In [ ]: #42.Find maximum value in each row of the given array:

In [77]: arr = np.array([[1, 2, 3], [4, 5, 6]])
         row_max = np.max(arr,axis=1)
         print(row_max)

[3 6]

In [ ]: #43.For the given array, find indices of maximum value in each column.

In [78]: arr = np.array([[1, 2, 3], [4, 5, 6]])
         column_max_indices = np.argmax(arr,axis =0)
         print(column_max_indices)

[1 1 1]

In [ ]: #44.For the given array, apply custom function to calculate moving sum along rows.

In [80]: arr = np.array([[1, 2, 3], [4, 5, 6]])
         result = np.array([[sum(row[:i+1])
         for i in range(len(row))]
         for row in arr])
         print(result)

[[ 1  3  6]
 [ 4  9 15]]

In [ ]: #45.In the given array, check if all elements in each column are even.

In [81]: arr = np.array([[2, 4, 6], [3, 5, 7]])
         result = np.all(arr % 2 ==0,axis =0)
         print(result)

[False False False]

In [ ]: #46.Given a NumPy array arr, reshape it into a matrix of dimensions `m` rows and `n` columns. Return the reshaped matrix.

In [83]: original_array = np.array([1, 2, 3, 4, 5, 6])
         m , n=2,3
         reshaped_matrix = original_array.reshape(m,n)
         print(reshaped_matrix)

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

In [ ]: #47.Create a function that takes a matrix as input and returns the flattened array.

In [84]: input_matrix = np.array([[1, 2, 3], [4, 5, 6]])
         flattened_array = [item for sublist in input_matrix for item in sublist]
         print(flattened_array)

[np.int64(1), np.int64(2), np.int64(3), np.int64(4), np.int64(5), np.int64(6)]

In [ ]: #48.Write a function that concatenates two given arrays along a specified axis.

In [88]: array1 = np.array([[1, 2], [3, 4]])
         array2 = np.array([[5, 6], [7, 8]])
         axis = 1
         result = np.concatenate((array1,array2),axis =axis)
         print(result)

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

In [61]: #49.Create a function that splits an array into multiple sub-arrays along a specified axis.

In [90]: original_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
         axis = 0
         num_subarrays = 3
         result = np.split(original_array, num_subarrays, axis = axis)
         print(result)

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

In [ ]: #50.Write a function that inserts and then deletes elements from a given array at specified indices.

In [92]: original_array = np.array([1, 2, 3, 4, 5])
         indices_to_insert = [2, 4]
         values_to_insert = [10, 11]
         indices_to_delete = [1, 3]

         original_array = np.insert(original_array, indices_to_insert , values_to_insert)
         original_array = np.delete(original_array, indices_to_delete)

         print(original_array)

[ 1 10  4 11  5]

In [ ]: #51.Create a NumPy array `arr1` with random integers and another array `arr2` with integers from 1 to 10. Perform element-wise addition between `arr1` and `arr2

In [93]: arr1 = np.random.randint(1,10, size = 10)
         print("Array 1 :" , arr1)

         arr2 = np.arange(1,11)
         print("Array 2 : ", arr2)


         result = arr1 + arr2
         print("Result: " , result)

Array 1 : [1 4 2 1 4 7 7 1 8 1]
Array 2 :  [ 1  2  3  4  5  6  7  8  9 10]
Result:  [ 2  6  5  5  9 13 14  9 17 11]

In [ ]: #52.Generate a NumPy array `arr1` with sequential integers from 10 to 1 and another array `arr2` with integers from 1 to 10. Subtract `arr2` from `arr1` element-wise.

In [95]: arr1 = np.arange(10 , 0 , -1)
         print("Array 1 :" , arr1)

         arr2 = np.arange(1,11)
         print("Array 2 : ", arr2)


         result = arr1 - arr2
         print("Result: " , result)

Array 1 : [10  9  8  7  6  5  4  3  2  1]
Array 2 :  [ 1  2  3  4  5  6  7  8  9 10]
Result:  [ 9  7  5  3  1 -1 -3 -5 -7 -9]

In [ ]: #53. Create a NumPy array `arr1` with random integers and another array `arr2` with integers from 1 to 5. Perform element-wise multiplication between `arr1` and `arr2`.

In [96]: arr1 = np.random.randint(1,10, size = 5)
         print("Array 1 :" , arr1)

         arr2 = np.arange(1,6)
         print("Array 2 : ", arr2)


         result = arr1 * arr2
         print("Result: " , result)

Array 1 : [6 6 6 2 2]
Array 2 :  [1 2 3 4 5]
Result:  [ 6 12 18  8 10]

In [ ]: #54. Generate a NumPy array `arr1` with even integers from 2 to 10 and another array `arr2` with integers from 1 to 5. Perform element-wise division of `arr1` by `arr2`.

In [97]: arr1 = np.arange(2,11,2)
         print("Array 1 :" , arr1)

         arr2 = np.arange(1,6)
         print("Array 2 : ", arr2)


         result = arr1 / arr2
         print("Result: " , result)

Array 1 : [ 2  4  6  8 10]
Array 2 :  [1 2 3 4 5]
Result:  [2. 2. 2. 2. 2.]

In [ ]: #55.Create a NumPy array `arr1` with integers from 1 to 5 and another array `arr2` with the same numbers reversed. Calculate the exponentiation of `arr1` raised to the power of `arr2` element-wise.

In [98]: arr1 = np.arange(1,6)
         print("Array 1 :" , arr1)

         arr2 = np.arange(5,0,-1)
         print("Array 2 : ", arr2)


         result = np.power(arr1 , arr2)
         print("Result: " , result)

Array 1 : [1 2 3 4 5]
Array 2 :  [5 4 3 2 1]
Result:  [ 1 16 27 16  5]

In [ ]: #56.Write a function that counts the occurrences of a specific substring within a NumPy array of strings.

In [99]: arr = np.array(['hello', 'world', 'hello', 'numpy', 'hello'])

         print((arr =='hello').sum())

3

In [ ]: #57.Write a function that extracts uppercase characters from a NumPy array of strings. arr = np.array(['Hello', 'World', 'OpenAI', 'GPT'])

In [101… arr = np.array(['Hello', 'World', 'OpenAI', 'GPT'])
         uppercase_chars = np.array([char for string in arr for char in string if char.isupper()])
         print(uppercase_chars)

['H' 'W' 'O' 'A' 'I' 'G' 'P' 'T']

In [ ]: #58.Write a function that replaces occurrences of a substring in a NumPy array of strings with a new string.

In [103… arr = np.array(['apple', 'banana', 'grape', 'pineapple'])
         old_string = 'apple'
         new_string = 'mango'
         result  = [string.replace(old_string, new_string)for string in arr]
         print(result)

['mango', 'banana', 'grape', 'pinemango']

In [ ]: #59.Write a function that concatenates strings in a NumPy array element-wise.

In [104… arr1 = np.array(['Hello', 'World'])
         arr2 = np.array(['Open', 'AI'])

         result = np.array([a + ' ' + b for a, b in zip(arr1, arr2)])
         print(result)

['Hello Open' 'World AI']

In [ ]: #60.Write a function that finds the length of the longest string in a NumPy array. arr = np.array(['apple', 'banana', 'grape', 'pineapple'])

In [105… arr = np.array(['apple', 'banana', 'grape', 'pineapple'])

         result = max(len(string)for string in arr)
         print(result)

9

In [ ]: #61.Create a dataset of 100 random integers between 1 and 1000. Compute the mean, median, variance, and standard deviation of the dataset using NumPy's functions.

In [4]: data = np.random.randint(1,1001 , 100)
        mean = np.mean(data)
        print("Mean : ", mean)

        median = np.median(data)
        print("Median: ", median)

        variance = np.var(data)
        print("Variance :" , variance)

        std_dev = np.std(data)
        print("Standard Deviation : " , std_dev)

Mean :  455.15
Median:  422.5
Variance : 78353.7075
Standard Deviation :  279.91732261508935

In [ ]: #62.Generate an array of 50 random numbers between 1 and 100. Find the 25th and 75th percentiles of the dataset.

In [9]: data = np.random.randint(1,101,50)
        print(np.percentile(data , [25 , 75]))

[28.5  77.75]

In [ ]: #63. Create two arrays representing two sets of variables. Compute the correlation coefficient between these arrays using NumPy's `corrcoef` function

In [13]: x = np.array([1,2,3,4,5])
         y = np.array([2,4,6,8,10])

         print(np.corrcoef(x, y) [0,1])

0.9999999999999999

In [ ]: #64.Create two matrices and perform matrix multiplication using NumPy's `dot` function.

In [14]: A = np.array([[1,2] ,[3,4]])
         B = np.array([[5,6], [7,8]])

         print(np.dot(A,B))

[[19 22]
 [43 50]]

In [ ]: #65.Create an array of 50 integers between 10 and 1000. Calculate the 10th, 50th (median), and 90th percentiles along with the first and third quartiles.

In [15]: np.random.randint(10,101,50)

         print(np.percentile(data,[10,25,50,75,90]))

[13.9  28.5  57.   77.75 87.6 ]

In [ ]: #66.Create a NumPy array of integers and find the index of a specific element.

In [16]: arr = np.array([10,20,30,40,50])

         print(np.where(arr == 30 )[0][0])

2

In [ ]: #67. Generate a random NumPy array and sort it in ascending order.

In [17]: arr = np.random.randint(1,100,10)

         print(np.sort(arr))

[13 15 20 31 45 62 65 83 88 89]

In [ ]: #68.Filter elements >20  in the given NumPy array.

In [19]: arr = np.array([12, 25, 6, 42, 8, 30])
         filtered_arr = arr[arr > 20]

         print(filtered_arr)

[25 42 30]

In [ ]: #69. Filter elements which are divisible by 3 from a given NumPy array.

In [20]: arr = np.array([1, 5, 8, 12, 15])
         filtered_arr = arr[arr % 3 ==0]

         print(filtered_arr)

[12 15]

In [ ]: #70.Filter elements which are ≥ 20 and ≤ 40 from a given NumPy array.

In [21]: arr = np.array([10, 20, 30, 40, 50])
         filtered_arr = arr[(arr >= 20) & (arr <= 40)]

         print(filtered_arr)

[20 30 40]

In [ ]: #71.For the given NumPy array, check its byte order using the `dtype` attribute byteorder.

In [22]: arr = np.array([1, 2, 3])
         print(arr.dtype.byteorder)

=

In [ ]: #72.For the given NumPy array, perform byte swapping in place using `byteswap()`.

In [23]: arr = np.array([1, 2, 3], dtype=np.int32)

         arr = arr.byteswap()
         print(arr)

[16777216 33554432 50331648]

In [ ]: #73.For the given NumPy array, swap its byte order without modifying the original array using `newbyteorder()`.

In [29]: arr = np.array([1, 2, 3], dtype=np.int32)
         arr_swapped = arr.view()
         print(arr_swapped)

[1 2 3]

In [ ]: #74.For the given NumPy array and swap its byte order conditionally based on system endianness using `newbyteorder()`.

In [42]: arr = np.array([1, 2, 3], dtype=np.int32)
         arr_swaped = arr.astype('>i4'if np.little_endian else 'i4')
         print(arr_swaped)

[1 2 3]

In [ ]: #75.For the given NumPy array, check if byte swapping is necessary for the current system using `dtype` attribute `byteorder`.

In [45]: arr = np.array([1, 2, 3], dtype=np.int32)

         if arr.dtype.byteorder == '=':
             print("No swap needed")
         else:
             print("Swap needed")

No swap needed

In [ ]: #76.Create a NumPy array `arr1` with values from 1 to 10. Create a copy of `arr1` named `copy_arr` and modify an element in `copy_arr`. Check if modifying `copy_arr` affects `arr1`.

In [46]: arr1 = np.arange (1, 11)
         copy_arr = arr1.copy()
         copy_arr[0] = 100
         print( arr1 , copy_arr , sep='\n')

[ 1  2  3  4  5  6  7  8  9 10]
[100   2   3   4   5   6   7   8   9  10]

In [ ]: #77.Create a 2D NumPy array `matrix` of shape (3, 3) with random integers. Extract a slice `view_slice` from the matrix. Modify an element in `view_slice` and observe if it changes the original `matrix`.

In [47]: matrix = np.random.randint(0,10,(3,3))
         view_slice = matrix[:2, :2]

         view_slice[0 , 0] = 100

         print(matrix , view_slice, sep='\n')

[[100   8   1]
 [  0   0   4]
 [  4   0   0]]
[[100   8]
 [  0   0]]

In [ ]: #78.  Create a NumPy array `array_a` of shape (4, 3) with sequential integers from 1 to 12. Extract a slice `view_b` from `array_a` and broadcast the addition of 5 to view_b. Check if it alters the origin

In [50]: array_a = np.arange(1,13).reshape(4,3)
         view_b = array_a [:2 , :]
         view_b +=5
         print(array_a , view_b , sep='\n')

[[ 6  7  8]
 [ 9 10 11]
 [ 7  8  9]
 [10 11 12]]
[[ 6  7  8]
 [ 9 10 11]]

In [ ]: #79.Create a NumPy array `orig_array` of shape (2, 4) with values from 1 to 8. Create a reshaped view `reshaped_view` of shape (4, 2) from orig_array. Modify an element in `reshaped_view` and check if it

In [52]: original_array = np.arange(1,9).reshape(2,4)
         reshaped_view = original_array.reshape(4,2)

         reshaped_view[0, 0 ] = 100

         print(original_array , reshaped_view , sep='\n')

[[100   2   3   4]
 [  5   6   7   8]]
[[100   2]
 [  3   4]
 [  5   6]
 [  7   8]]

In [ ]: #80.Create a NumPy array `data` of shape (3, 4) with random integers. Extract a copy `data_copy` of elements greater than 5. Modify an element in `data_copy` and verify if it affects the original `data`.

In [53]: data = np.random.randint(0,10, (3,4))
         data_copy = data[data > 5].copy()

         if len(data_copy) > 0:
             data_copy[0] = 100

         print(data , data_copy , sep='\n')

[[7 8 6 7]
 [1 2 1 2]
 [6 2 5 1]]
[100   8   6   7   6]

In [ ]: #81.Create two matrices A and B of identical shape containing integers and perform addition and subtraction operations between them.

In [54]: A , B = np.array([[1,2,3], [4,5,6]]) , np.array([[7,8, 9] , [10 , 11, 12]])


         print(A , "\n" , B , "\n" , A+B , "\n" , A-B)

[[1 2 3]
 [4 5 6]]
 [[ 7  8  9]
 [10 11 12]]
 [[ 8 10 12]
 [14 16 18]]
 [[-6 -6 -6]
 [-6 -6 -6]]

In [ ]: #82.Generate two matrices `C` (3x2) and `D` (2x4) and perform matrix multiplication.

In [56]: A , B = np.array([[1,2] , [3,4] , [5,6]]),np.array([[7,8 , 9,10] , [11,12,13,14]])

         print(A @ B)

[[ 29  32  35  38]
 [ 65  72  79  86]
 [101 112 123 134]]

In [ ]: #83.Create a matrix `E` and find its transpose.

In [57]: Z = np.array([[1,2,3] , [4,5,6]])

         print("Matrix Z : \n" , Z )

         print("Transporse of Matrix Z :\n" , Z.T)

Matrix Z :
 [[1 2 3]
 [4 5 6]]
Transporse of Matrix Z :
 [[1 4]
 [2 5]
 [3 6]]

In [ ]: #84.Generate a square matrix `F` and compute its determinant.

In [58]: F = np.array([[1,2] , [3,4]])
         print("Matrix F : \n" , F)
         print("Determinant of Matrix F:\n" , np.linalg.det(F))

Matrix F :
 [[1 2]
 [3 4]]
Determinant of Matrix F:
 -2.0000000000000004

In [ ]: #85. Create a square matrix `G` and find its inverse.

In [59]: A = np.array([[1,2], [3,4]])

         print("Matrix A:\n" , A)
         print("Inverse of Matrix A:\n" , np.linalg.inv(A))

Matrix A:
 [[1 2]
 [3 4]]
Inverse of Matrix A:
```