```
## Create a NumPy array 'arr' of integers from 0 to 5 and print its data type.
import numpy as np

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

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

```
## Given a NumPy array 'arr', check if its data type is float64.

   # arr = np.array([1.5,2.6,3.7])

arr = np.array([1.5,2.6,3.7])
print(arr.dtype)
```

```
float64
```

```
## Create a NumPy array 'arr' with a data type of complex128 containing three complex numbers.

arr = np.array([1+5j,2+3j ,4+22j])
print(arr.dtype)
```

```
complex128
```

```
## Convert an existing NumPy array 'arr' of integers to float32 data type.
arr = np.array([1,2,3,4,5,6],dtype = np.float32)
arr
```

```
array([1., 2., 3., 4., 5., 6.], dtype=float32)
```

```
## Given a NumPy array 'arr' with float64 data type, convert it to float32 to reduce decimal precision.

arr = np.array([1.5,2.6,3.7,3.6,3.0] ,dtype = np.float32)
print(arr.dtype)
```

```
float32
```

```
# Write a function array_attributes that takes a NumPy array as input and returns its shape, size, and data type.


def  array_attributes():
  arr = eval(input("enter the array"))
  arr = np.array(arr)
  print(arr.shape)
  print(arr.size)
  print(arr.dtype)

array_attributes()
```

```
enter the array([1,2,3,4,5],[9,8,7,6,4])
(2, 5)
10
int64
```

```
## Create a function array_dimension that takes a NumPy array as input and returns its dimensionality.
def  array_dimension():
  arr = eval(input("enter the array"))
  arr = np.array(arr)
  print(arr.ndim)

array_dimension()
```

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

```
## Design a function item_size_info that takes a NumPy array as input and returns the item size and the total size in bytes.

def  item_size_info():
  arr = eval(input("enter the array"))
```

```
  arr = np.array(arr)
  print(arr.itemsize)
  print(arr.nbytes)

item_size_info()
```

```
enter the array([1,2,3,4,5],[9,8,7,6,4])
8
80
```

```
## Create a function array_strides that takes a NumPy array as input and returns the strides of the array.
def array_strides():
  arr = eval(input("enter the array"))
  arr = np.array(arr)
  print(arr.strides)


array_strides()
```

```
enter the array([1,2,3,4,5],[9,8,7,6,4])
(40, 8)
```

```
## Design a function shape_stride_relationship that takes a NumPy array as input and returns the shape and strides of the array.

def shape_stride_relationship():
  arr = eval(input("enter the array"))
  arr = np.array(arr)
  print(arr.strides)
  print(arr.shape)

shape_stride_relationship()
```

```
enter the array([1,2,3,4,5],[9,8,7,6,4])
(40, 8)
(2, 5)
```

```
## Create a function `create_zeros_array` that takes an integer `n` as input and returns a NumPy array of zeros with `n` element

def create_zeros_array():
  arr = eval(input("enter the element of n:"))
  arr = np.zeros(arr,dtype = int)
  print(arr)

create_zeros_array()
```

```
enter the element of n:5
[0 0 0 0 0]
```

```
## Write a function `create_ones_matrix` that takes integers `rows` and `cols` as inputs and generates a 2D
## NumPy array filled with ones of size `rows x cols`.


def create_ones_matrix():
  arr = eval(input("enter rows and cols "))
  arr = np.ones(arr,dtype = int)
  print(arr)

create_ones_matrix()
```

```
enter rows and cols (3,4)
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

```
## 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 the specified
## `step`.

def generate_range_array():
  arr = eval(input("enter the step of array>> start,stop,step : "))
  arr = np.arange(*arr)
  print(arr)
```

```
generate_range_array()
```

```
enter the step of array>> start,stop,step : 2,9,2
[2 4 6 8]
```

```
## 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`
## (inclusive).


def generate_linear_space():
  arr = eval(input("start ,stop , num :"))
  arr = np.linspace(*arr)
  print(arr)
generate_linear_space()
```

```
start ,stop , num :2.3,8.3,5
[2.3 3.8 5.3 6.8 8.3]
```

```
## 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`.

def create_identity_matrix():
  arr = eval(input("enter the integer :"))
  arr = np.eye(arr)
  print(arr)

create_identity_matrix()
```

```
enter the integer :5
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

```
## Write a function that takes a Python list and converts it into a NumPy array.

lis = [1,2,3,4,5,6,7,8]

arr = np.array(lis)
arr
```

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

```
## Create a NumPy array and demonstrate the use of `numpy.view` to create a new array object with the same data.
arr = np.array([1,2,3,4,5,6,7,8])
arr1 = arr.view()
arr1
```

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

```
## Write a function that takes two NumPy arrays and concatenates them along a specified axis.

arr1 = np.array([1,2,3,4,5,6])
arr2 = np.array([9,8,7,5,4,3])
arr = np.concatenate((arr1,arr2))
arr
```

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

```
## Create two NumPy arrays with different shapes and concatenate them horizontally using `numpy concatenate`.

arr1 = np.array([[1,2,3,4,5,6]])
arr2 = np.array([[9,8,7,5,4,3]])
arr = np.concatenate((arr1,arr2),axis = 0)
arr
```

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

```
## Write a function that vertically stacks multiple NumPy arrays given as a list.(20)
import numpy as np



arr1 = np.array([[1,2,3,4,5,6]])
arr2 = np.array([[9,8,7,5,4,3]])
arr3 = np.array([[1,2,3,4,5,6]])
arr = np.concatenate((arr1,arr2,arr3),axis = 0)
arr
```

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

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



a1 = np.arange(1,10,2)
a1
```

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

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

a2 = np.linspace(0,1,10)
a2
```

```
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
```

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

a3 = np.logspace(1,1000,5)
a3
```

```
/usr/local/lib/python3.11/dist-packages/numpy/_core/function_base.py:312: RuntimeWarning: overflow encountered in power
  return _nx.power(base, y)
array([1.00000000e+001, 5.62341325e+250,             inf,             inf,
                    inf])
```

```
## 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.
import pandas as pd


df = pd.DataFrame(np.random.randint(1,100,(5,3)))
df
```

```

```

```
## 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.

df = pd.DataFrame(np.random.randint(-100,100,(5,4)))
df[df<0] = 0
df
```

```

```

```
## Access the 3rd element from the given NumPy array.

## Access the 3rd element from the given NumPy array.

arr = np.array([10, 20, 30, 40, 50])
arr[2]
```

```
np.int64(30)
```

```
## Retrieve the element at index (1, 2) from the 2D NumPy array.
arr = np.random.randint(1,9,(4,3))
arr1 = arr[1,2]
arr1
```

```
np.int64(4)
```

```
## Using boolean indexing, extract elements greater than 5 from the given NumPy array.
## Using boolean indexing, extract elements greater than 5 from the given NumPy array.
arr = np.array([1,2,3,4,5,6,7,8])
arr1 = arr[arr<5]
arr1
```

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

```
## Perform basic slicing to extract elements from index 2 to 5 (inclusive) from the given NumPy array.

arr = np.random.randint(1,50,(6,5))
arr1 = arr[2:5]
arr1
```

```
array([[38, 43, 13, 27, 16],
       [13,  7,  8,  5, 24],
       [16, 45, 45, 24,  1]])
```

```
## Slice the 2D NumPy array to extract the sub-array `[[2, 3], [5, 6]]` from the given array.
arr = np.random.randint(1,50,(8,7))
arr1 =arr[[2,3],[5,6]]
arr1
```

```
array([16, 27])
```

```
## Write a NumPy function to extract elements in specific order from a given 2D array based on indices
## provided in another array.
arr = np.random.randint(1,50,(8,7))
arr1 = arr[[0,4],
           [4,5]]
arr1
```

```
array([12, 25])
```

```
##32. Create a NumPy function that filters elements greater than a threshold from a given 1D array using
##    boolean indexing.
import numpy as np

arr = np.array([1,24,12,13,14,15,16,17,18])
```

```
threshold =14
arr1 = arr[arr>threshold]
arr1
```

```
array([24, 15, 16, 17, 18])
```

```
## Write a NumPy function that returns elements from an array where both two conditions are satisfied
## using boolean indexing.

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

arr1 = arr[(arr>3)&(arr>2)]
arr1
```

```
array([4, 8, 5, 6, 7, 5, 6])
```

```
## Create a NumPy function that extracts elements from a 2D array using row and column indices provided
## in separate arrays.
arr = np.array([[1,2,3,4],
                [8,5,6,7],
                 [3,5,6,2]])
arr1 = arr[[0,1],
           [2,3],
           ]
arr1
```

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

```
##  Given an array arr of shape (3, 3), add a scalar value of 5 to each element using NumPy broadcasting.
arr = np.array([[1,2,4],
                [8,5,6],
                 [3,6,2]])

arr1 = arr+5
arr1
```

```
array([[ 6,  7,  9],
           [13, 10, 11],
           [ 8, 11,  7]])
```

```
## 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.


arr1 = np.array([1,2,3])
arr2 = np.array([[1,2,3,4],
                [8,5,6,7],
                 [3,6,2,4]])
arr = arr1@arr2
arr
```

```
array([26, 30, 21, 30])
```

```
## 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.
arr1 = np.array([1,2,4])
arr2 = np.array([[1,2,3],
                [8,5,6],
                 [3,2,4],
                  [1,5,4]])
arr2 + arr1
```

```
array([[ 2,  4,  7],
           [ 9,  7, 10],
           [ 4,  4,  8],
           [ 2,  7,  8]])
```

```
## Consider two arrays arr1 of shape (3, 1) and arr2 of shape (1, 3). Add these arrays using NumPy
## broadcasting.


arr1 = np.array([[3],
                 [4],
                 [5]])
arr2 = np.array([1,2,3])
arr1+arr2
```

```
array([[4, 5, 6],
       [5, 6, 7],
       [6, 7, 8]])
```

```
## Given arrays arr1 of shape (2, 3) and arr2 of shape (2, 2), perform multiplication using NumPy
 ## broadcasting. Handle the shape incompatibility.

arr1 = np.array([[2,3,4],
                 [6,7,4]])
arr2 = np.array([[2,4],
                   [5,6]])

arr1_reshape = arr1.reshape(2,1,3)
arr2_reshape = arr2.reshape(2,2,1)

arr1_reshape*arr2_reshape
```

```
array([[[ 4,  6,  8],
        [ 8, 12, 16]],

       [[30, 35, 20],
        [36, 42, 24]]])
```

```
## Calculate column wise mean for the given array:
import numpy as np

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


arr.mean(axis = 0)
```

```
array([2.5, 3.5, 4.5])
```

```
## Find maximum value in each row of the given array:
arr = np.array([[1, 2, 3], [4, 5, 6]])

arr.max(axis = 1)
```

```
array([3, 6])
```

```
## For the given array, find indices of maximum value in each column.
arr = np.array([[1, 2, 3], [4, 5, 6]])
np.argmax(arr,axis = 0)



```

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

```
## For the given array, apply custom function to calculate moving sum along rows.
arr = np.array([[1, 2, 3], [4, 5, 6]])
np.cumsum(arr, axis =1)
```

```
array([[ 1,  3,  6],
       [ 4,  9, 15]])
```

```
##45. In the given array, check if all elements in each column are even.
arr = np.array([[2, 4, 6], [3, 5, 7]])
np.all(arr%2==0, axis = 0)
```

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

```
## Given a NumPy array arr, reshape it into a matrix of dimensions `m` rows and `n` columns. Return the
## reshaped matrix.
original_array = np.array([1, 2, 3, 4, 5, 6])
arr = np.matrix(original_array).reshape(2,3)
arr
```

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

```
## Create a function that takes a matrix as input and returns the flattened array.
input_matrix = np.array([[1, 2, 3], [4, 5, 6]])


def flatten_matrix(matrix):                          ### input_matrix = np.array([[1, 2, 3], [4, 5, 6]])
                                                       #  input_matrix.flatten()
  return matrix.flatten()

flatten_matrix = flatten_matrix(input_matrix)
flatten_matrix
```

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

```
## Write a function that concatenates two given arrays along a specified axis.
import numpy as np

array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6], [7, 8]])

array2 = np.concatenate((array1,array2),axis =1)
array2
```

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

```
## Create a function that splits an array into multiple sub-arrays along a specified axis.

array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
def split_array():
  return np.split(array,3,axis = 0)
split_array()
```

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

```
## Write a function that inserts and then deletes elements from a given array at specified indices.

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

def delete_insert():
  inserted_array = np.insert(original_array,indices_to_insert,values_to_insert)

  deleted_array = np.delete(inserted_array,indices_to_delete)

  return deleted_array

delete_insert()
```

```
array([ 1, 10,  4, 11,  5])
```

```
## 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`.

arr1 = np.random.randint(2,9,(1,10))
arr1
arr2 = np.array([1,2,3,4,5,6,7,8,9,10])

arr1 + arr2
```

```
array([[ 9,  5,  8,  8,  9, 10, 10, 11, 13, 13]])
```

```
## 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.
arr1 = np.array([10,9,8,7,6,5,4,3,2,1])
arr2 = np.array([1,2,3,4,5,6,7,8,9,10])

arr1 - arr2
```

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

```
##  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`

arr1 = np.random.randint(2,9,(1,5))
arr1
arr2 = np.array([1,2,3,4,5])
arr =  arr1@arr2
arr
```

```
array([68])
```

```
## 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 `arr

arr1 =  np.array([2,4,6,8,10])
arr2 = np.array([1,2,3,4,5])

arr = arr1/arr2
arr
```

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

```
## 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.

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

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

result = arr2**arr1
```

```
## Write a function that counts the occurrences of a specific substring within a NumPy array of strings.

arr = np.array(['hello', 'world', 'hello', 'numpy', 'hello'])
def count_occurrences(arr,substring ):
  return np.char.count(arr, substring).sum()

result = count_occurrences(arr,'hello')
print(result)
```

```
3
```

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

arr = np.array(['Hello', 'World', 'OpenAI', 'GPT'])

for i in arr:
  for char in i :
    if char.isupper() :
      print(list(char))
```

```
['H']
['W']
['O']
['A']
['I']
```

```
    ['G']
    ['P']
    ['T']
```

```
## Write a function that replaces occurrences of a substring in a NumPy array of strings with a new string.

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

def replace_occ(arr, old_string ,new_string ):
  return np.char.replace(arr,old_string ,new_string)

replace_occ(arr, 'apple', 'mango')
```

```
array(['mango', 'banana', 'grape', 'pinemango'], dtype='<U9')
```

```
## 59. Write a function that concatenates strings in a NumPy array element-wise.

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

arr1 + arr2
```

```
array(['HelloOpen', 'WorldAI'], dtype='<U9')
```

```
## 60. Write a function that finds the length of the longest string in a NumPy array.

arr = np.array(['apple', 'banana', 'grape', 'pineapple'])
def longest():
  max_lenth = 0

  for i in arr :
    len(i) > max_lenth
    max_lenth = len(i)

  return max_lenth
print(longest())
```

```
9
```

```
## 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.
import numpy as np
dataset = np.random.randint(1,1000,100)
dataset
mean_d = dataset.mean()
median_d = np.median(dataset)
variance_d = dataset.var()
standard_d = dataset.std()
print("mean_value:/n ",mean_d)

print("median_value:/n ", median_d)

print("var_value:/n ",variance_d)

print("std:/n ",standard_d)
```

```
mean_value:/n  485.87
    median_value:/n  463.0
    var_value:/n  71509.7931
    std:/n  267.41315057416307
```

```
##Generate an array of 50 random numbers between 1 and 100. Find the 25th and 75th percentiles of the dataset.

arr = np.random.randint(1,101,50)
arr25 = np.percentile(arr,25)
arr75 = np.percentile(arr,75)
print('pr25:',arr25,'pr75:',arr75)
```

```
pr25: 20.75 pr75: 71.0
```

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

import numpy as np


x = np.random.randint(1, 100, 50)
y = np.random.randint(1, 100, 50)

corr_matrix = np.corrcoef(x, y)

corr_matrix
```

```
array([[ 1.        , -0.06728035],
       [-0.06728035,  1.        ]])
```

```
## Create two matrices and perform matrix multiplication using NumPy's `dot` function.

mat1 = np.matrix([[1,2,3],[9,8,7],[4,5,6]])
mat2 = np.matrix([[3,4,5,8],[6,5,4,7],[2,3,4,6]])
mat_multification = mat1.dot(mat2)
mat_multification
```

```
matrix([[ 21,  23,  25,  40],
        [ 89,  97, 105, 170],
        [ 54,  59,  64, 103]])
```

```
## 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.
arr = np.random.randint(10,1001,51)

el_percentile_10th = np.percentile(arr,10)

el_percentile_50th = np.percentile(arr,50)

el_percentile_90th = np.percentile(arr,90)
print('el_percentile_10th :',el_percentile_10th )
print('el_percentile_50th :',el_percentile_50th )
print('el_percentile_90th :',el_percentile_90th)
```

```
el_percentile_10th : 130.0
el_percentile_50th : 467.0
el_percentile_90th : 843.0
```

```
## Create a NumPy array of integers and find the index of a specific element.
arr = np.arange(15)
index = np.where(arr == 4)[0]
print(index )
```

```
[4]
```

```
##Generate a random NumPy array and sort it in ascending order.
arr = np.random.randint(5,100,15)
arr1 = np.sort(arr)
arr1
```

```
array([ 8, 24, 26, 27, 36, 38, 49, 58, 59, 59, 70, 71, 74, 80, 91])
```

```
## Filter elements >20  in the given NumPy array.

arr = np.array([12, 25, 6, 42, 8, 30])
arr1 = arr[arr>20]
arr1
```

```
array([25, 42, 30])
```

```
## Filter elements which are divisible by 3 from a given NumPy array.
```

```
arr = np.array([1, 5, 8, 12, 15])
arr1 = arr[arr%3 == 0]
arr1
```

```
array([12, 15])
```

```
## 70 Filter elements which are ≥ 20 and ≤ 40 from a given NumPy array.
import numpy as np
arr = np.array([10, 20, 30, 40, 50])

arr1 = arr[(arr >= 20) & (arr<=40) ]
arr1
```

```
array([20, 30, 40])
```

```
## For the given NumPy array, check its byte order using the `dtype` attribute byteorder.

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

```
=
```

```
## 72. For the given NumPy array, perform byte swapping in place using `byteswap()`.
arr = np.array([1, 2, 3], dtype=np.int32)

arr.byteswap()
```

```
array([16777216, 33554432, 50331648], dtype=int32)
```

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

arr = np.array([1, 2, 3], dtype=np.int32)
old_byte = arr.dtype.byteorder

print('old byteorder :',old_byte)

print('new byteorder :',arr.dtype.newbyteorder())
```

```
old byteorder : =
    new byteorder : >i4
```

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

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

old_byteorder = arr.dtype.byteorder
print('old_byteorder:',old_byteorder)

if old_byteorder == '=':
  swapped_arr = arr.dtype.newbyteorder()
  print('new_byteorder :', swapped_arr )
```

```
old_byteorder: =
    new_byteorder : >i4
```

```
##75. For the given NumPy array, check if byte swapping is necessary for the current system using `dtype`
## attribute `byteorder`.
arr = np.array([1, 2, 3], dtype=np.int32)
byte_order = arr.dtype.byteorder
if byte_order == '=':
  print('no need of byte swaping ')
else :
  print('byte swaping is necessary ')
```

```
no need of byte swaping
```

```
## 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`.

arr1 = np.array([1,2,3,4,5,6,7,8,9,10])

copy_arr1  = np.array([1,2,3,4,5,6,7,8,9,10])
copy_arr1[4] = 4

if np.array_equal(arr1,copy_arr1) :
  print('copy_arr1 effectes arr1 ')

else:
  print('copy_arr1 does not effect arr1 ')
```

```
copy_arr1 does not effect arr1
```

```
##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`

arr = np.random.randint(2,9,(3,3))
view_slice = arr[0:2,0:3]
view_slice[2] = 8
```

```

```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.