```
import numpy as np


#1. Create a NumPy array 'arr' of integers from 0 to 5 and prin
arr=np.array([0,1,2,3,4])
print(arr)
```

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

```
#2.. Given a NumPy array 'arr', check if its data type is float
arr = np.array([1.5, 2.6, 3.7])
if type(arr)==np.float64:
  print("is true")
else:
  print("false")
```

```
false
```

```
#3.Create a NumPy array 'arr' with a data type of complex128 co
complex_list= [3 + 4j, 10 - 3j, 1 + 2j]
arr=np.array(complex_list,dtype=np.complex128)
print(arr)
print(arr.dtype)
```

```
[ 3.+4.j 10.-3.j  1.+2.j]
complex128
```

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

```
dtype('int64')
```

```
arr=np.array(arr,dtype=np.float32)
arr.dtype
```

```
dtype('float32')
```

```
#5Given a NumPy array 'arr' with float64 data type, convert it
arr=np.ones(4,dtype=np.float64)
arr.dtype
```

```
dtype('float64')
```

```
arr=np.float32(arr)
arr.dtype
```

```
dtype('float32')
```

```
#6. Write a function array_attributes that takes a NumPy array
arr=np.ndarray((3,4))
arr
```

```
array([[5.00659675e-310, 0.00000000e+000, 5.00707234e-310,
        1.33360289e+241],
       [5.00707234e-310, 3.19411549e+132, 5.00707234e-310,
        3.67767335e-308],
```

AttributeError

You

Please explain the error:

```
AttributeError: 'NoneType' object ha
```

Gemini

Hypothesis

The error message "AttributeError: 'NoneType' object has no attribute 'determinant'" tells us that you're trying to call the `determinant()` method on an object that is `None`. Looking at the traceback, we see that the error occurs in this line:

```
 += ((-1) ** j) * self.data[0][j] *
```

Use code with caution

The most likely cause of this error is that the `minor()` method of your `Matrix` class is returning `None` instead of a `Matrix` object.

Suggested Changes

```
lf, data):
 data


(self):
self.data), len(self.data[0])
 m, "Matrix must be square"


self.data[0][0]


nge(n):
((-1) ** j) * self.data[0][j] * self


 i, j):
new matrix by excluding row i and co
= [row[:j] + row[j+1:] for row in se
```

```
            [5.02620509e+180, 9.08366791e+223, 5.67633874e-311,
             8.62730520e-174]])
```

```
arr.shape
```

```
(3, 4)
```

```
arr.size
```

```
12
```

```
arr.dtype
```

```
dtype('float64')
```

```
#7.Create a function array_dimension that takes a NumPy array a
arr=np.ndarray((3,4))
arr.ndim
```

```
2
```

```
#8. Design a function item_size_info that takes a NumPy array a
```

```
def item_size_info(arr):
    item_size = arr.itemsize
    total_size = arr.nbytes
    return item_size, total_size

# Example usage:
if __name__ == "__main__":
    my_array = np.array([1, 2, 3], dtype=np.float64)
    item_size, total_size = item_size_info(my_array)
    print(f"Item size: {item_size} bytes")
    print(f"Total size: {total_size} bytes")
```

```
Item size: 8 bytes
Total size: 24 bytes
```

```
#9.Create a function array_strides that takes a NumPy array as
```

```
def array_strides(arr):
    return arr.strides
x = np.array([[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]], dtype=np.int32
print("Strides for x:", x.strides)
```

```
Strides for x: (20, 4)
```

```
ix(minor_data)  # Return a new Matri

[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
al.determinant()
rminant of F: {det_manual:.2f}")
```

Use code with caution

Explanation:

1. **Implement `minor()` Method:** The `minor()` method was previously not implemented. We have added code to create a new matrix by excluding the specified row and column, and then return a new `Matrix` object constructed from this data.

By returning a `Matrix` object from `minor()`, the `determinant()` method can now correctly call the `determinant()` method on the mino matrix.

Rate this answer

```
#10. Design a function shape_stride_relationship that takes a N


import numpy as np

def shape_stride_relationship(arr):
    shape = arr.shape
    strides = arr.strides
    return shape, strides
if __name__ == "__main__":
    array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
    shape, strides = shape_stride_relationship(array)
    print("Shape:", shape)
    print("Strides:", strides)
```

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

```
#11.Create a function `create_zeros_array` that takes an intege


def create_zeros_array(n):
    return np.zeros(n)
n_elements = 5
zeros_array = create_zeros_array(n_elements)
print(zeros_array)
```

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

```
#12.. Write a function `create_ones_matrix` that takes integers


def create_ones_matrix(rows, cols):
    return np.ones((rows, cols))
rows = 3
cols = 4
ones_matrix = create_ones_matrix(rows, cols)
print(ones_matrix)
```

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

```
#13.. Write a function `generate_range_array` that takes three in
#creates a NumPy array with a range starting from `start`, ending
#`step`


def generate_range_array(start, stop, step):
    return np.arange(start, stop, step)
start_value = 11
end_value = 17
step_value = 0.5
result_array = generate_range_array(start_value, end_value, step_
print("Generated array:", result_array)
```

```
Generated array: [11.  11.5 12.  12.5 13.  13.5 14.  14.5 1
```

```
#14.. Design a function `generate_linear_space` that takes two
#arguments and generates a NumPy array with num equally spaced
#(inclusive)


def generate_linear_space(start: float, stop: float, num: int)
    return np.linspace(start, stop, num)
start_value = 0.0
stop_value = 1.0
num_values = 5
result_array = generate_linear_space(start_value, stop_value, n
print(result_array)
```

```
[0.   0.25 0.5  0.75 1.  ]
```

```
#15.Create a function `create_identity_matrix` that takes an in
#identity matrix of size `n x n` using `numpy.eye`.

def create_identity_matrix(n):

    return np.eye(n)

# Example usage:
n = 4
identity_matrix = create_identity_matrix(n)
print(identity_matrix)
```

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

```
#16. Write a function that takes a Python list and converts it
my_list = [1, 7, 0, 6, 2, 5, 6]
my_array = np.array(my_list)

print("List:", my_list)
print("Array:", my_array)
```

```
List: [1, 7, 0, 6, 2, 5, 6]
Array: [1 7 0 6 2 5 6]
```

```
#17.. Create a NumPy array and demonstrate the use of `numpy.vi

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

```
Original array:
[1 2 3 4 5]
```

```
#18.Write a function that takes two NumPy arrays and concatenat


def concatenate_arrays(arr1, arr2, axis=0):

    return np.concatenate((arr1, arr2), axis=axis)
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6]])
result = concatenate_arrays(array1, array2, axis=0)
print(result)
```

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

```
#19. Create two NumPy arrays with different shapes and concaten


array1 = np.array([[1, 2, 3], [4, 5, 6]])
array2 = np.array([[7, 8], [9, 10]])

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

print("array1",array1)

print("array2",array2)

print("concatenated",concatenated_array)
```

```
array1 [[1 2 3]
 [4 5 6]]
array2 [[ 7  8]
 [ 9 10]]
concatenated [[ 1  2  3  7  8]
 [ 4  5  6  9 10]]
```

```
#20.Write a function that vertically stacks multiple NumPy arra


def stack_arrays_vertically(arr_list):
    return np.vstack(arr_list)
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = stack_arrays_vertically([a, b])
print(result)
```

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

```
#21.Write a Python function using NumPy to create an array of int

start = 11
stop = 17
step = 0.5
result_array = np.arange(start, stop + step, step)

print(result_array)
```

```
[11.  11.5 12.  12.5 13.  13.5 14.  14.5 15.  15.5 16.  16.
```

```
#22. Write a Python function using NumPy to generate an array o
sequence = np.linspace(0, 1, num=10)
print(sequence)
```

```
[0.         0.11111111 0.22222222 0.33333333 0.44444444 0.5
 0.66666667 0.77777778 0.88888889 1.        ]
```

```
#23.. Write a Python function using NumPy to create an array of

def create_logspace_array():
    logspace_values = np.logspace(0, 3, num=5)
    return logspace_values
log_values = create_logspace_array()
print(log_values)

import numpy as np

def create_logspace_array():
    logspace_values = np.logspace(0, 3, num=5)
    return logspace_values
log_values = create_logspace_array()
print(log_values)


def create_logspace_array():
    logspace_values = np.logspace(0, 3, num=5)
    return logspace_values
log_values = create_logspace_array()
print(log_values)
```

```
[   1.            5.62341325   31.6227766   177.827941   10
[   1.            5.62341325   31.6227766   177.827941   10
[   1.            5.62341325   31.6227766   177.827941   10
```

```
#24. Create a Pandas DataFrame using a NumPy array that contain
import pandas as pd
arr=np.random.randint(0,101,size=(5,3))
df=pd.DataFrame(arr)
df
```

	0	1	2
0	93	88	7
1	79	48	99
2	72	98	39
3	86	58	7
4	67	96	10

```
#25.Write a function that takes a Pandas DataFrame and replaces

def replace_negatives_with_zeros(df, column_name):

    df[column_name] = np.where(df[column_name] < 0, 0, df[colum
    return df
df = pd.DataFrame({'A': [-1, 2, -3, 4]})
result_df = replace_negatives_with_zeros(df, 'A')
print(result_df)
```

```
   A
0  0
1  2
2  0
3  4
```

```
#26.Access the 3rd element from the given NumPy array.arr = np.
arr = np.array([10, 20, 30, 40, 50])
arr1=arr[2]
print(arr1)
```

```
30
```

```
#27.Retrieve the element at index (1, 2) from the 2D NumPy array.
               #    [4, 5, 6],
                #   [7, 8, 9]])

arr_2d=np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])
arr=arr_2d[1][2]
print(arr)
```

```
6
```

```
#28.Using boolean indexing, extract elements greater than 5 from
arr = np.array([3, 8, 2, 10, 5, 7])
arr>5
```

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

```
#29. Perform basic slicing to extract elements from index 2 to 5
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
arr[2:5]
```

```
array([3, 4, 5])
```

```
#30. Slice the 2D NumPy array to extract the sub-array `[[2, 3]
"""arr_2d = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])"""
arr_2d[0:2,1:3]
```

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

```
#31Write a NumPy function to extract elements in specific order

A = np.array([[14, 7, 30], [44, 76, 65], [42, 87, 11]])
indices = np.array([0, 1, 2])
result = A[indices, range(len(A))]

print(result)
```

```
[14 76 11]
```

```
#32. Create a NumPy function that filters elements greater than

def filter_elements(arr, threshold):
    return arr[arr > threshold]

a = np.array([1, 2, 3, 4, 5])
threshold = 3
filtered_array = filter_elements(a, threshold)
print(filtered_array)
```

```
[4 5]
```

```
#33Develop a NumPy function that extracts specific elements fro

def extract_elements_3d_array(my_array, indices_x, indices_y, i
    assert indices_x.shape == indices_y.shape == indices_z.shap
    extracted_elements = my_array[indices_x, indices_y, indices

    return extracted_elements
m, n, p = 65, 65, 300
vals = np.random.rand(m, n, p)
indices_x = np.random.randint(0, m, (m, n))
indices_y = np.random.randint(0, n, (m, n))
indices_z = np.random.randint(0, p, (m, n))

result = extract_elements_3d_array(vals, indices_x, indices_y,
print("Extracted elements shape:", result.shape)
```

```
Extracted elements shape: (65, 65)
```

```
#34.Write a NumPy function that returns elements from an array

def select_elements(dists, r, dr):
    result = np.vectorize(lambda d: d >= r and d <= (r + dr))(d
    indices = np.where(result)

    return indices
dists = np.array([1.5, 3.0, 4.2, 6.8, 7.5])
r = 3.0
dr = 2.0
selected_indices = select_elements(dists, r, dr)
print("Selected indices:", selected_indices)
```

```
Selected indices: (array([1, 2]),)
```

```
#35.. Create a NumPy function that extracts elements from a 2D
a = np.array([[1, 2, 3, 4], [4, 5, 6, 7]])
row_indices = np.array([0, 1])
col_indices = np.array([2, 3])
result = a[row_indices[:, None], col_indices]
print(result)
```

```
[[3 4]
 [6 7]]
```

```
#36. Given an array arr of shape (3, 3), add a scalar value of

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


scalar = 5
result = arr + scalar
```

```
#37. Given arrays arr1 of shape (2, 3) and arr2 of shape (2, 2)


arr1 = np.array([[1, 2, 3], [4, 5, 6]])
arr2 = np.array([[10, 20], [14, 25]])
arr2_reshaped = arr2.reshape(2, 1, 2)
result = arr1 * arr2_reshaped
print(result)
```

```
---------------------------------------------------
--------------------------
ValueError
Traceback (most recent call last)
<ipython-input-14-6de41df8d75a> in <cell line: 7>
()
      5 arr2 = np.array([[10, 20], [14, 25]])
      6 arr2_reshaped = arr2.reshape(2, 1, 2)
----> 7 result = arr1 * arr2_reshaped
      8 print(result)
```

```
#38.Given a 1D array arr1 of shape (1, 4) and a 2D array arr2 o

arr1 = np.array([[1, 2, 3, 4]])
arr2 = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90], [100
arr1_reshaped = arr1.reshape(4, 1)
result = arr1_reshaped + arr2
print(result)
```

```
[[ 11  21  31]
 [ 42  52  62]
 [ 73  83  93]
 [104 114 124]]
```

```
#39Consider two arrays arr1 of shape (3, 1) and arr2 of shape (
arr1 = np.array([[1.0], [2.0], [3.0]])
arr2 = np.array([[2.0, 2.0, 2.0]])

result = arr1 + arr2
print(result)
```

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

```
#40.Given arrays arr1 of shape (2, 3) and arr2 of shape (2, 2),
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
arr2 = np.array([[2, 3], [1, 2]])
arr2_reshaped = arr2.reshape(2, 1, 2)
arr1_transposed = arr1.T
result = arr1_transposed * arr2_reshaped
print(result)
```

```
[[[ 2 12]
  [ 4 15]
  [ 6 18]]

 [[ 1  8]
  [ 2 10]
  [ 3 12]]]
```

```
#41.Calculate column wise mean for the given array arr = np.arr


arr = np.array([[1, 2, 3], [4, 5, 6]])
column_means = np.mean(arr, axis=0)

print("Column-wise means:")
print(column_means)
```

```
Column-wise means:
[2.5 3.5 4.5]
```

```
#42.Find maximum value in each row of the given array arr = np.
import numpy as np

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

print(max_values)
```

```
[3 6]
```

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

```
[4 5 6]
```

```
#44. For the given array, apply custom function to calculate mo

def moving_sum(arr):
    return np.cumsum(arr, axis=1)

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

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

```
#45. In the given array, check if all elements in each column a
arr = np.array([[2, 4, 6], [3, 5, 7]])
transposed_arr = arr.T
result = (transposed_arr == transposed_arr[0]).all()

print(f"All elements in each column are even: {result}")
```

```
All elements in each column are even: False
```

```
#46.Given a NumPy array arr, reshape it into a matrix of dimens

original_array = np.array([1, 2, 3, 4, 5, 6])
m, n = 2, 3

reshaped_matrix = np.reshape(original_array, (m, n))
print(reshaped_matrix)
```

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

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

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

```
#48. Write a function that concatenates two given arrays along

array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6], [7, 8]])
result = np.concatenate((array1, array2), axis=0)

print(result)
```

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

```
#49.Create a function that splits an array into multiple sub-ar

def split_array(original_array, axis):
    return np.split(original_array, original_array.shape[axis],

original_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
sub_arrays_rows = split_array(original_array, axis=0)
sub_arrays_columns = split_array(original_array, axis=1)

print("Sub-arrays along rows:")
for arr in sub_arrays_rows:
    print(arr)

print("\nSub-arrays along columns:")
for arr in sub_arrays_columns:
    print(arr)
```

```
Sub-arrays along rows:
[[1 2 3]]
[[4 5 6]]
[[7 8 9]]

Sub-arrays along columns:
[[1]
 [4]
 [7]]
[[2]
 [5]
 [8]]
[[3]
 [6]
 [9]]
```

```
#50. Write a function that inserts and then deletes elements fr
import numpy as np

def modify_array(original_array, indices_to_insert, values_to_i

    modified_array = np.copy(original_array)
    for idx, value in zip(indices_to_insert, values_to_insert):
        modified_array = np.insert(modified_array, idx, value)
    modified_array = np.delete(modified_array, indices_to_delet

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

result_array = modify_array(original_array, indices_to_insert,
print("Modified array:", result_array)
```

```
Modified array: [ 1 10 11  4  5]
```

```
#51.. Create a NumPy array `arr1` with random integers and anot

arr1 = np.random.randint(0, 100, size=(5, 5))
arr2 = np.arange(1, 26).reshape(5,5)
result = arr1 + arr2

print("arr1:")
print(arr1)
print("\narr2:")
print(arr2)
print("\nResult (element-wise addition):")
print(result)
```

```
arr1:
[[66  4 27  2 72]
 [49 78 88 58 94]
 [17 47 74 42 74]
 [73  0 52 95 99]
 [65 61 78 25 45]]

arr2:
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]

Result (element-wise addition):
[[ 67   6  30   6  77]
 [ 55  85  96  67 104]
 [ 28  59  87  56  89]
 [ 89  17  70 114 119]
 [ 86  83 101  49  70]]
```

```
#52.generate a NumPy array `arr1` with sequential integers from
#from 1 to 10. Subtract `arr2` from `arr1` element-wise.

arr1 = np.arange(10, 0, -1)
arr2 = np.arange(1, 11)
result = arr1 - arr2
result
```

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

```
#53.create a NumPy array `arr1` with random integers and anothe
#Perform element-wise multiplication between `arr1` and `arr2`.

arr1 = np.random.randint(0, 100, size=(3, 3))

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


result = arr1 * arr2[:, np.newaxis]

print("arr1:")
print(arr1)
print("\narr2:")

print(arr2)
print("\nResult (element-wise multiplication):")
print(result)
```

```
arr1:
[[57 62 49]
 [42 70 98]
 [59 97 27]]

arr2:
[[1]
 [2]
 [3]
 [4]
 [5]]

Result (element-wise multiplication):
[[[ 57  62  49]
  [ 42  70  98]
  [ 59  97  27]]

 [[114 124  98]
  [ 84 140 196]
  [118 194  54]]

 [[171 186 147]
  [126 210 294]
  [177 291  81]]

 [[228 248 196]
  [168 280 392]
  [236 388 108]]

 [[285 310 245]
  [210 350 490]
  [295 485 135]]]
```

```
#54. Generate a NumPy array `arr1` with even integers from 2 to
arr1=np.random.randint(2,10,size=(3,3))
arr2=np.random.randint(1,5,size=(3,3))
arr=arr1/arr2
print(arr)
```

```
[[1.   2.5  2.  ]
 [8.   6.   2.25]
 [1.   8.   0.5 ]]
```

```
#55 Create a NumPy array `arr1` with integers from 1 to 5 and a
import numpy as np
arr1 = np.arange(1, 6)
arr2 = arr1[::-1]
result = np.power(arr1, arr2)
print(result)
```

```
[ 1 16 27 16  5]
```

```
#56.. Write a function that counts the occurrences of a specifi

def count_substring(arr, sub):
    arr_lower = np.char.lower(arr)
    occurrences = np.char.count(arr_lower, sub)

    return occurrences
arr = np.array(['hello', 'world', 'hello', 'numpy', 'hello'])
substring = 'hello'
result = count_substring(arr, substring)
print(result)
```

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

```
#57.Write a function that extracts uppercase characters from a

def extract_uppercase(arr):
    return np.char.upper(arr)

arr = np.array(['Hello', 'World', 'OpenAI', 'GPT'])
result = extract_uppercase(arr)
print(result)
```

```
['HELLO' 'WORLD' 'OPENAI' 'GPT']
```

```
#58. Write a function that replaces occurrences of a substring

def replace_substring(arr, old_substring, new_substring):
    return np.char.replace(arr, old_substring, new_substring)

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

result = replace_substring(arr, old_substring, new_substring)
print(result)
```

```
['fruit' 'banana' 'grape' 'pinefruit']
```

```
#59.Write a function that concatenates strings in a NumPy array
#arr1 = np.array(['Hello', 'World'])
#arr2 = np.array(['Open', 'AI'])


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

result = np.char.add(arr1, arr2)
print(result)
```

```
['HelloOpen' 'WorldAI']
```

```
#60.Write a function that finds the length of the longest strin
#arr = np.array(['apple', 'banana', 'grape', 'pineapple'])
def find_longest_string_length(arr):
    return len(max(arr, key=len))

# Example usage:
arr = np.array(['apple', 'banana', 'grape', 'pineapple'])
longest_length = find_longest_string_length(arr)
print(f"The length of the longest string in the array is {longe
```

```
The length of the longest string in the array is 9.
```

```
#61.Create a dataset of 100 random integers between 1 and 1000.
random_integers = np.random.randint(1, 1001, size=100)
mean_value = np.mean(random_integers)
median_value = np.median(random_integers)
variance_value = np.var(random_integers)
std_deviation_value = np.std(random_integers)

print(f"Mean: {mean_value:.2f}")
print(f"Median: {median_value:.2f}")
print(f"Variance: {variance_value:.2f}")
print(f"Standard deviation: {std_deviation_value:.2f}")
```

```
Mean: 470.98
Median: 426.50
Variance: 84267.22
Standard deviation: 290.29
```

```
#62. Generate an array of 50 random numbers between 1 and 100.
```

```
#63.Create two arrays representing two sets of variables. Compu

xarr = np.random.random((3, 3))
yarr = np.random.random((3, 3))
R = np.corrcoef(xarr, yarr)

print("Correlation coefficient matrix:")
print(R)
```

```
Correlation coefficient matrix:
[[ 1.          0.93577278 -0.49699423  0.38674212  0.950466
 [ 0.93577278  1.         -0.15910055  0.68706952  0.779822
 [-0.49699423 -0.15910055  1.          0.6080235  -0.742096
 [ 0.38674212  0.68706952  0.6080235   1.          0.080946
 [ 0.95046699  0.77982294 -0.74209665  0.08094608  1.
 [-0.99962164 -0.92572004  0.52067455 -0.36123017 -0.958656
```

```
#64.Create two matrices and perform matrix multiplication using


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

result = np.dot(A, B)
print(result)
```

```
[[19 22]
 [43 50]]
```

```
#65.Create an array of 50 integers between 10 and 1000. Calcula



#66. Create a NumPy array of integers and find the index of a s

my_array = np.array([1, 3, 5, 7, 9])
index = np.where(my_array == 7)
print("Index of 7:", index[0][0])
```

```
Index of 7: 3
```

```
#67.. Generate a random NumPy array and sort it in ascending or

random_array = np.random.rand(10)
sorted_array = np.sort(random_array)

print("Random array:", random_array)
print("Sorted array:", sorted_array)
```

```
Random array: [0.95355614 0.75826461 0.00245851 0.40487277
 0.42741316 0.06588132 0.76474933 0.14142848]
Sorted array: [0.00245851 0.06588132 0.14142848 0.34308935
 0.75826461 0.76474933 0.95355614 0.97981857]
```

```
#68.. Filter elements >20  in the given NumPy array ,arr = np.a

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

print(filtered_elements)
```

```
[25 42 30]
```

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


arr = np.array([1, 5, 8, 12, 15])
mask = (arr % 3 == 0)  # Create a boolean mask for elements div
filtered_arr = arr[mask]  # Apply the mask to the original arra

print(filtered_arr)  # Output: [ 3 12 15]
```

```
[12 15]
```

```
#70. Filter elements which are ≥ 20 and ≤ 40 from a given NumPy

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

print(filtered_arr)
```

```
[20 30 40]
```

```
#71..order using the `dty For the given NumPy array, check its
arr = np.array([1, 2, 3])
V = [4, 6, 2]
column = arr.tolist()
order = [column.index(item) for item in V]

reordered_arr = arr[order]

print(reordered_arr)
```

```
---------------------------------------------------
--------------------------
ValueError
Traceback (most recent call last)
<ipython-input-20-a736f0cfe389> in <cell line: 5>
()
      3 V = [4, 6, 2]
      4 column = arr.tolist()
----> 5 order = [column.index(item) for item in
V]
      6
      7 reordered_arr = arr[order]

<ipython-input-20-a736f0cfe389> in <listcomp>(.0)
      3 V = [4, 6, 2]
      4 column = arr.tolist()
----> 5 order = [column.index(item) for item in
V]
      6
      7 reordered_arr = arr[order]
```

Next steps: Explain error

#72. For the given NumPy array, perform byte swapping in place

```
arr = np.array([1, 2, 3], dtype=np.int32)
arr.byteswap(True)
print(arr)
```

```
[16777216 33554432 50331648]
```

#73. For the given NumPy array, swap its byte order without mod

```
original_array = np.array([1, 2, 3], dtype=np.int16)
swapped_array = original_array.newbyteorder('>')
print("Original array:", original_array)
print("Swapped array:", swapped_array)
```

```
Original array: [1 2 3]
Swapped array: [256 512 768]
```

```
#74.. For the given NumPy array and swap its byte order conditi
```

```
#74.. For the given NumPy array and swap its byte order conditi
import sys # Import the sys module

arr = np.array([1, 2, 3], dtype=np.int16)

sys_is_le = sys.byteorder == 'little'
native_code = '<' if sys_is_le else '>'
swapped_code = '>' if sys_is_le else '<'
native_dt = np.dtype(native_code + 'i2')
swapped_dt = np.dtype(swapped_code + 'i2')


swapped_arr = arr.astype(native_dt).newbyteorder('S')

print("Original array:", arr)
print("Swapped array:", swapped_arr)
```

```
Original array: [1 2 3]
Swapped array: [256 512 768]
```

```
#75. For the given NumPy array, check if byte swapping is neces

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

if byte_order == "=":
    print("No byte swapping needed (native byte order)")
elif byte_order == "<":
    print("Byte swapping needed (little-endian system)")
elif byte_order == ">":
    print("Byte swapping needed (big-endian system)")
else:
    print("Unknown byte order")
```

```
No byte swapping needed (native byte order)
```

```
#76. Create a NumPy array `arr1` with values from 1 to 10. Crea
arr1 = np.arange(1, 11)
copy_arr = arr1.copy()
copy_arr[3] = 99
print("Original arr1:", arr1)
print("Modified copy_arr:", copy_arr)
```

```
Original arr1: [ 1  2  3  4  5  6  7  8  9 10]
Modified copy_arr: [ 1  2  3 99  5  6  7  8  9 10]
```

```
#77. Create a 2D NumPy array `matrix` of shape (3, 3) with rand
matrix = np.random.randint(10, size=(3, 3))
view_slice = matrix[:2, :2]
view_slice[1, 1] = 99
print("Original matrix:")
print(matrix)

print("\nModified view_slice:")
print(view_slice)
```

```
Original matrix:
[[ 1  2  3]
 [ 0 99  9]
 [ 7  4  7]]

Modified view_slice:
[[ 1  2]
 [ 0 99]]
```

```
#78. Create a NumPy array `array_a` of shape (4, 3) with sequen
array_a = np.arange(1, 13).reshape(4, 3)
view_b = array_a[1:3, 1:]
view_b += 5
print("Original array_a:")
print(array_a)
```

```
Original array_a:
[[ 1  2  3]
 [ 4 10 11]
 [ 7 13 14]
 [10 11 12]]
```

```
#79.Create a NumPy array `orig_array` of shape (2, 4) with valu
"""`reshaped_view` of shape (4, 2) from orig_array. Modify an e
reflects changes in the original `orig_array`"""
orig_array = np.arange(1, 9).reshape(2, 4)
reshaped_view = orig_array.reshape(4, 2)
reshaped_view[1, 1] = 99
print("Original array:")
print(orig_array)

print("\nReshaped view:")
print(reshaped_view)
```

```
Original array:
[[ 1  2  3 99]
 [ 5  6  7  8]]
```

```
Reshaped view:
[[ 1  2]
 [ 3 99]
 [ 5  6]
 [ 7  8]]
```

```
#80.Create a NumPy array `data` of shape (3, 4) with random int
```

```
data = np.random.randint(low=0, high=10, size=(3, 4))

data_copy = data[data > 5]
data_copy[0, ] = 10
print("Original 'data':")
print(data)
print("\nModified 'data_copy':")
print(data_copy)
```

```
Original 'data':
[[5 9 2 2]
 [3 9 5 3]
 [7 0 6 7]]

Modified 'data_copy':
[10  9  7  6  7]
```

```
#81. Create two matrices A and B of identical shape containing
```

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[4, 5], [6, 7]])
addition_result = np.add(A, B)
print("Addition of two matrices:")
print(addition_result)
```

```
Addition of two matrices:
[[ 5  7]
 [ 9 11]]
```

```
#82.Generate two matrices `C` (3x2) and `D` (2x4) and perform m
```

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

D = np.array([[7, 8, 9, 10],
              [11, 12, 13, 14]])


result = np.matmul(C, D)

print("Resultant matrix C:")
print(result)
```

```
Resultant matrix C:
[[ 29  32  35  38]
 [ 65  72  79  86]
 [101 112 123 134]]
```

Responses may display inaccurate or offensive information