SEPTEMBER 6,
2024

i am having issues in google colab so i'm doing it in jupyter

1 NUMPY ASSIGNMENT

Q1

[4]:
```
import numpy as np

arr = np.array([0, 1, 2, 3, 4, 5])
print(arr.dtype)
```

```
int32
```

Q2

[9]:
```
def check_dtype(arr):
    return arr.dtype == np.float64


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

```
True
```

Q3

[12]:
```
arr = np.array([1+2j, 3+4j, 5+6j], dtype=np.complex128)
print(arr)
```

```
[1.+2.j 3.+4.j 5.+6.j]
```

Q4

[15]:
```
arr = np.array([1, 2, 3, 4, 5])
arr_float32 = arr.astype(np.float32)
print(arr_float32)
```

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

Q5

[18]:

```python
def convert_to_float32(arr):
    return  arr.astype(np.float32)

# EX
arr = np.array([1.2345678901234567, 2.3456789012345678], dtype=np.float64)
arr_float32  = convert_to_float32(arr)
print(arr_float32)
```

```
[1.2345679 2.3456788]
```

Q6

[21]:

```python
def  array_attributes(arr):
    return arr.shape, arr.size, arr.dtype

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

```
((2, 3), 6, dtype('int32'))
```

Q7

[26]:

```python
def array_dimension(arr):
    return arr.ndim

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

```
2
```

Q8

[29]:

```python
def item_size_info(arr):
    return arr.itemsize, arr.nbytes

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

```
(4, 24)
```

Q9

[43]:

```python
def array_strides(arr):
    return arr.strides

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

```
(12, 4)
```

Q10

[35]:

```python
def shape_stride_relationship(arr):
    return arr.shape, arr.strides

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

```
((2, 3), (12, 4))
```

Q11

[38]:

```python
def  create_zeros_array(n):
    return  np.zeros(n)

# EX
print(create_zeros_array(5))
```

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

Q12

[40]:

```python
def create_ones_matrix(rows, cols):
    return np.ones((rows, cols))

# EX
print(create_ones_matrix(3, 4))
```

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

Q13

[46]:

```python
def generate_range_array(start, stop, step):
    return np.arange(start, stop, step)

# EX:
print(generate_range_array(0, 10, 2))
```

```
[0 2 4 6 8]
```

Q14

[49]:

```python
def generate_linear_space(start, stop, num):
    return np.linspace(start, stop, num)

# EX:
print(generate_linear_space(0.0, 1.0, 5))
```

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

Q15

[54]:
```
def create_identity_matrix(n):
    return np.eye(n)

# EX:
print(create_identity_matrix(3))
```

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

Q16

[57]:
```
def list_to_numpy_array(lst):
    return np.array(lst)

# EX:
lst = [1, 2, 3, 4]
print(list_to_numpy_array(lst))
```

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

Q17

[60]:
```
arr = np.array([1, 2, 3, 4])
view_arr = arr.view()
print("Original array:", arr)
print("View of the array:", view_arr)
```

```
Original array: [1 2 3 4]
View of the array: [1 2 3 4]
```

Q18

[63]:
```
def concatenate_arrays(arr1, arr2, axis=0):
    return np.concatenate((arr1, arr2), axis=axis)

# EX
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
print(concatenate_arrays(arr1, arr2, axis=0))
```

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

Q19

[66]:

```python
def concatenate_horizontally(arr1, arr2):
    return np.concatenate((arr1, arr2), axis=1)

# EX
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5], [6]])
print(concatenate_horizontally(arr1, arr2))
```

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

Q20

[69]:

```python
def  vertical_stack(arrays):
    return np.vstack(arrays)

# EX:
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])
print(vertical_stack([arr1, arr2]))
```

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

Q21

[72]:

```python
def create_integer_range(start, stop, step):
    return np.arange(start, stop + 1, step)

# EX
print(create_integer_range(0, 10, 2))
```

```
[ 0  2  4  6  8 10]
```

Q22

[75]:

```python
def generate_equal_spacing():
    return np.linspace(0, 1, 10)

# EX
print(generate_equal_spacing())
```

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

Q23

[78]:

```python
def generate_log_spacing():
    return np.logspace(0, 3, 5, base=10.0)

# EX
```

```
print(generate_log_spacing())
```

```
[   1.           5.62341325   31.6227766   177.827941  1000.        ]
```

Q24

[81]:

```
import pandas as pd

def create_random_dataframe():
    arr = np.random.randint(1, 101, size=(5, 3))
    df = pd.DataFrame(arr, columns=["Column1", "Column2", "Column3"])
    return df

# EX
print(create_random_dataframe())
```

```
   Column1  Column2  Column3
0        2       76       89
1       26       30       85
2       43       42       74
3       88       59       44
4       98       68       10
```

Q25

[84]:

```
def replace_negatives_with_zero(df, column):
    df[column] = np.where(df[column] < 0, 0, df[column])
    return df

# EX
df = pd.DataFrame({
    "A": [1, -2, 3, -4],
    "B": [-1, 2, -3, 4]
})
print(replace_negatives_with_zero(df, "A"))
```

```
   A  B
0  1 -1
1  0  2
2  3 -3
3  0  4
```

Q26

[89]:

```
def access_third_element(arr):
    return arr[2]

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

```
30
```

Q27

[92]:
```python
def access_element_2d(arr):
    return arr[1, 2]

# EX:
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(access_element_2d(arr))
```

```
6
```

Q28

[95]:
```python
def extract_elements_gt_5(arr):
    return arr[arr > 5]

# EX:
arr = np.array([3, 8, 2, 10, 5, 7])
print(extract_elements_gt_5(arr))
```

```
[ 8 10  7]
```

Q29

[107]:
```python
def slice_array(arr):
    return arr[2:5]

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

```
[3 4 5]
```

Q30

[101]:
```python
def  slice_2d_array(arr):
    return  arr[0:2,  1:3]

# Ex
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(slice_2d_array(arr))
```

```
[[2 3]
 [5 6]]
```

Q31

[110]:
```python
def extract_specific_order(arr,  indices):
    return arr[indices]
```

```
# EX
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
indices = np.array([0, 2])
print(extract_specific_order(arr, indices))
```

```
[[1 2 3]
 [7 8 9]]
```

Q32

[113]:

```
def filter_greater_than(arr, threshold):
    return arr[arr > threshold]

# EX:
arr = np.array([1, 3, 7, 2, 5])
print(filter_greater_than(arr, 3))
```

```
[7 5]
```

Q33

[116]:

```
def extract_elements_3d(arr, x_indices, y_indices, z_indices):
    return arr[x_indices, y_indices, z_indices]

# EX
arr = np.random.randint(1, 10, size=(3, 3, 3))
x_indices = np.array([0, 1])
y_indices = np.array([1, 2])
z_indices = np.array([2, 0])
print(extract_elements_3d(arr, x_indices, y_indices, z_indices))
```

```
[1 8]
```

Q34

[119]:

```
def filter_two_conditions(arr, cond1, cond2):
    return arr[(arr > cond1) & (arr < cond2)]

# EX:
arr = np.array([1, 3, 7, 2, 5])
print(filter_two_conditions(arr, 2, 6))
```

```
[3 5]
```

Q35

[122]:

```
def extract_using_indices(arr, row_indices, col_indices):
    return arr[row_indices, col_indices]

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

```
row_indices = np.array([0, 2])
col_indices = np.array([1, 0])
print(extract_using_indices(arr, row_indices, col_indices))
```

```
[2 7]
```

Q36

[125]:

```
def add_scalar(arr, scalar):
    return arr + scalar

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

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

Q37

[128]:

```
def multiply_broadcast(arr1, arr2):
    return arr2 * arr1.T

# Ex
arr1 = np.array([[1, 2, 3]])
arr2 = np.array([[4, 5, 6, 7], [8, 9, 10, 11], [12, 13, 14, 15]])
print(multiply_broadcast(arr1, arr2))
```

```
[[ 4  5  6  7]
 [16 18 20 22]
 [36 39 42 45]]
```

Q38

[133]:

```
def add_broadcast(arr1, arr2):
    return arr2 + arr1.T

# Ex
arr1 = np.array([[1, 2, 3]])
arr2 = np.array([[4, 5, 6, 7], [8, 9, 10, 11], [12, 13, 14, 15]])
print(add_broadcast(arr1, arr2))
```

```
[[ 5  6  7  8]
 [10 11 12 13]
 [15 16 17 18]]
```

Q39

[136]:

```
def add_arrays_broadcast(arr1, arr2):
    return arr1 + arr2
```

```
# Ex
arr1 = np.array([[1], [2], [3]])
arr2 = np.array([[4, 5, 6]])
print(add_arrays_broadcast(arr1, arr2))
```

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

Q40

[139]:

```
def multiply_with_broadcasting(arr1, arr2):
    arr2_broadcasted = np.broadcast_to(arr2[:, :1], arr1.shape)
    return arr1 * arr2_broadcasted

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

```
[[ 7 14 21]
 [36 45 54]]
```

Q41

[144]:

```
def column_mean(arr):
    return np.mean(arr, axis=0)

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

```
[2.5 3.5 4.5]
```

Q42

[147]:

```
def row_max(arr):
    return np.max(arr, axis=1)

#  Ex:
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(row_max(arr))
```

```
[3 6]
```

Q43

[150]:

```
def max_value_indices(arr):
    return np.argmax(arr, axis=0)
```

```
# Ex:
arr = np.array([[1, 5, 3], [4, 2, 6]])
print(max_value_indices(arr))
```

```
[1 0 1]
```

Q44

[153]:

```
def moving_sum(arr, window):
    return np.apply_along_axis(lambda m: np.convolve(m, np.ones(window,␣
 ↪dtype=int), 'valid'), axis=1, arr=arr)

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

```
[[ 3  5]
 [ 9 11]]
```

Q45

[156]:

```
def check_even_columns(arr):
    return np.all(arr % 2 == 0, axis=0)

# Ex:
arr = np.array([[2, 4, 6], [3, 5, 7]])
print(check_even_columns(arr))
```

```
[False False False]
```

Q46

[159]:

```
def reshape_array(arr, m, n):
    return arr.reshape(m, n)

# Ex:
arr = np.array([1, 2, 3, 4, 5, 6])
print(reshape_array(arr, 2, 3))
```

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

Q47

[168]:

```
def  flatten_matrix(arr):
    return  arr.flatten()

# Ex:
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(flatten_matrix(arr))
```

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

Q48

[165]:
```python
def concatenate_arrays(arr1, arr2, axis=0):
    return np.concatenate((arr1, arr2), axis=axis)

# Ex:
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
print(concatenate_arrays(arr1, arr2, axis=1))
```

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

Q49

[171]:
```python
def split_array(arr, indices, axis=0):
    return np.split(arr, indices, axis=axis)

# Example usage:
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(split_array(arr, [2], axis=1))
```

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

Q50

[174]:
```python
def insert_and_delete(arr, insert_indices, values_to_insert, delete_indices):
    # Inserting elements at the specified indices
    arr_inserted = np.insert(arr, insert_indices, values_to_insert)
    # Deleting elements at the specified indices
    arr_deleted = np.delete(arr_inserted, delete_indices)
    return arr_deleted

# Ex:
arr = np.array([1, 2, 3, 4, 5])
insert_indices = [1, 3]
values_to_insert = [10, 11]
delete_indices = [2, 4]
print(insert_and_delete(arr, insert_indices, values_to_insert, delete_indices))
```

```
[ 1 10  3  4  5]
```

Q51

[177]:
```python
def elementwise_addition(arr1, arr2):
    return arr1 + arr2
```

```
# Ex:
arr1 = np.random.randint(1, 10, size=10)
arr2 = np.arange(1, 11)
print(elementwise_addition(arr1, arr2))
```

```
[ 5 11 11 12  8 10  8 16 11 13]
```

Q52

[180]:

```
def elementwise_subtraction(arr1, arr2):
    return arr1 - arr2

# Ex:
arr1 = np.arange(10, 0, -1)
arr2 = np.arange(1, 11)
print(elementwise_subtraction(arr1, arr2))
```

```
[ 9  7  5  3  1 -1 -3 -5 -7 -9]
```

Q53

[183]:

```
def elementwise_multiplication(arr1, arr2):
    return arr1 * arr2

# Ex:
arr1 = np.random.randint(1, 10, size=5)
arr2 = np.arange(1, 6)
print(elementwise_multiplication(arr1, arr2))
```

```
[ 6 14  6  8 25]
```

Q54

[186]:

```
def elementwise_division(arr1, arr2):
    return arr1 / arr2

# Ex:
arr1 = np.arange(2, 11, 2)
arr2 = np.arange(1, 6)
print(elementwise_division(arr1, arr2))
```

```
[2. 2. 2. 2. 2.]
```

Q55

[189]:

```
def elementwise_exponentiation(arr1, arr2):
    return arr1 ** arr2

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

```
print(elementwise_exponentiation(arr1, arr2))
```

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

Q56

[192]:

```
def count_substring(arr, substring):
    return np.char.count(arr, substring)

# Ex:
arr = np.array(["apple", "banana", "applepie", "pineapple"])
print(count_substring(arr, "apple"))
```

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

Q57

[195]:

```
def extract_uppercase(arr):
    return np.char.join("", np.char.upper(arr))

# Ex:
arr = np.array(["Hello", "World", "OpenAI", "GPT"])
print(extract_uppercase(arr))
```

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

Q58

[204]:

```
def replace_substring(arr, old_substr, new_substr):
    return np.char.replace(arr, old_substr, new_substr)

# Ex:
arr = np.array(["apple", "banana"])
print(replace_substring(arr, "grape", "pineapple"))
```

```
['apple' 'banana']
```

Q59

[206]:

```
def concatenate_strings(arr1, arr2):
    return np.char.add(arr1, arr2)

# Ex:
arr1 = np.array(["hello ", "open"])
arr2 = np.array(["world", "AI"])
print(concatenate_strings(arr1, arr2))
```

```
['hello world' 'openAI']
```

Q60

[208]:

```
def longest_string_length(arr):
    return np.max(np.char.str_len(arr))

# Ex:
arr = np.array(['apple', 'banana', 'grape', 'pineapple'])
print(longest_string_length(arr))
```

```
9
```

Q61

[211]:

```
def compute_statistics(arr):
    mean = np.mean(arr)
    median = np.median(arr)
    variance = np.var(arr)
    std_dev = np.std(arr)
    return mean, median, variance, std_dev

# Ex:
arr = np.random.randint(1, 1001, size=100)
print(compute_statistics(arr))
```

```
(495.11, 477.5, 86787.89790000001, 294.59785793518597)
```

Q62

[216]:

```
def find_percentiles(arr):
    percentile_25 = np.percentile(arr, 25)
    percentile_75 = np.percentile(arr, 75)
    return percentile_25, percentile_75

# Ex:
arr = np.random.randint(1, 101, size=50)
print(find_percentiles(arr))
```

```
(19.25, 71.75)
```

Q63

[219]:

```
def compute_correlation(arr1, arr2):
    return np.corrcoef(arr1, arr2)[0, 1]

# Ex:
arr1 = np.random.randint(1, 100, size=50)
arr2 = np.random.randint(1, 100, size=50)
print(compute_correlation(arr1, arr2))
```

```
-0.004035605614860668
```

Q64

[222]:

```python
def matrix_multiplication(mat1, mat2):
    return np.dot(mat1, mat2)

# Ex:
mat1 = np.random.randint(1, 10, size=(3, 2))
mat2 = np.random.randint(1, 10, size=(2, 3))
print(matrix_multiplication(mat1, mat2))
```

```
[[96 48 48]
 [45 18 27]
 [93 42 51]]
```

Q65

[225]:

```python
def calculate_percentiles(arr):
    percentiles = {
        '10th': np.percentile(arr, 10),
        '50th (median)': np.percentile(arr, 50),
        '90th': np.percentile(arr, 90),
        '1st Quartile': np.percentile(arr, 25),
        '3rd Quartile': np.percentile(arr, 75)
    }
    return percentiles

# Ex:
arr = np.random.randint(10, 1001, size=50)
print(calculate_percentiles(arr))
```

```
{'10th': 145.4, '50th (median)': 597.0, '90th': 868.6, '1st Quartile': 333.5,
'3rd Quartile': 778.0}
```

Q66

[228]:

```python
def find_index(arr, element):
    return np.where(arr == element)[0]

# Ex:
arr = np.array([10, 20, 30, 40, 50])
print(find_index(arr, 30))
```

```
[2]
```

Q67

[233]:

```python
def sort_array(arr):
    return np.sort(arr)

# Ex:
arr = np.random.randint(1, 100, size=10)
print(sort_array(arr))
```

```
[26 37 51 52 56 58 59 65 79 89]
```

Q68

[236]:

```python
def filter_greater_than_20(arr):
    return arr[arr > 20]

# Ex:
arr = np.array([12, 25, 6, 42, 8, 30])
print(filter_greater_than_20(arr))
```

```
[25 42 30]
```

Q69

[239]:

```python
def  filter_divisible_by_3(arr):
    return arr[arr % 3 == 0]

# Ex:
arr = np.array([1, 5, 8, 12, 15])
print(filter_divisible_by_3(arr))
```

```
[12 15]
```

Q70

[242]:

```python
def filter_between_20_and_40(arr):
    return arr[(arr >= 20) & (arr <= 40)]

# Ex:
arr = np.array([10, 20, 30, 40, 50])
print(filter_between_20_and_40(arr))
```

```
[20 30 40]
```

Q71

[245]:

```python
def check_byte_order(arr):
    return arr.dtype.byteorder

# Ex:
arr = np.array([1, 2, 3], dtype=np.int32)
print(check_byte_order(arr))
```

```
=
```

Q72

[248]:

```python
def byte_swap_in_place(arr):
    arr.byteswap(True)
    return arr
```

```
# Ex:
arr = np.array([1, 256, 65536], dtype=np.int32)
print(byte_swap_in_place(arr))
```

```
[16777216    65536      256]
```

Q73

[251]:

```
def byte_swap_new(arr):
    return arr.newbyteorder()

# Ex:
arr = np.array([1, 256, 65536], dtype=np.int32)
print(byte_swap_new(arr))
```

```
[16777216    65536      256]
```

Q74

[254]:

```
def conditional_byte_swap(arr):
    if arr.dtype.byteorder == '=':  # Native byte order
        return arr.newbyteorder()
    else:
        return arr

# Ex:
arr = np.array([1, 256, 65536], dtype=np.int32)
print(conditional_byte_swap(arr))
```

```
[16777216    65536      256]
```

Q75

[257]:

```
def is_byte_swap_necessary(arr):
    return arr.dtype.byteorder not in ('=', '|')  # '=' for native byte order,␣
 ↪'|' for not applicable

# Ex:
arr = np.array([1, 256, 65536], dtype=np.int32)
print(is_byte_swap_necessary(arr))
```

```
False
```

Q76

[260]:

```
def check_copy_behavior():
    arr1 = np.arange(1, 11)
    copy_arr = arr1.copy()
    copy_arr[0] = 99
    return arr1, copy_arr
```

```python
# Ex:
arr1, copy_arr = check_copy_behavior()
print("Original array:", arr1)
print("Modified copy:", copy_arr)
```

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

Q77

[263]:
```python
def check_view_behavior():
    matrix = np.random.randint(1, 10, size=(3, 3))
    view_slice = matrix[:2, :2]
    view_slice[0, 0] = 99
    return matrix, view_slice

# EX:
matrix, view_slice = check_view_behavior()
print("Original matrix after modification:", matrix)
print("View slice:", view_slice)
```

```
Original matrix after modification: [[99  4  6]
 [ 7  5  5]
 [ 6  5  6]]
View slice: [[99  4]
 [ 7  5]]
```

Q78

[266]:
```python
def broadcast_and_modify():
    array_a = np.arange(1, 13).reshape(4, 3)
    view_b = array_a[1:3, 1:3]
    view_b += 5
    return array_a, view_b

# Ex:
array_a, view_b = broadcast_and_modify()
print("Original array after broadcast modification:", array_a)
print("View slice:", view_b)
```

```
Original array after broadcast modification: [[ 1  2  3]
 [ 4 10 11]
 [ 7 13 14]
 [10 11 12]]
View slice: [[10 11]
 [13 14]]
```

Q79

[269]:

```
def reshape_and_modify():
    orig_array = np.arange(1, 9).reshape(2, 4)
    reshaped_view = orig_array.reshape(4, 2)
    reshaped_view[0, 0] = 99
    return orig_array, reshaped_view

# Ex:
orig_array, reshaped_view = reshape_and_modify()
print("Original array after reshaped view modification:", orig_array)
print("Reshaped view:", reshaped_view)
```

```
Original array after reshaped view modification: [[99  2  3  4]
 [ 5  6  7  8]]
Reshaped view: [[99  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]]
```

Q80

[272]:

```
def modify_copy():
    data = np.random.randint(1, 10, size=(3, 4))
    data_copy = data[data > 5].copy()
    data_copy[0] = 99
    return data, data_copy

# Ex:
data, data_copy = modify_copy()
print("Original data after copy modification:", data)
print("Modified copy:", data_copy)
```

```
Original data after copy modification: [[5 5 4 6]
 [1 7 5 4]
 [8 1 3 6]]
Modified copy: [99  7  8  6]
```

Q81

[275]:

```
def add_subtract_matrices(A, B):
    addition_result = A + B
    subtraction_result = A - B
    return addition_result, subtraction_result

# EX:
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
addition, subtraction = add_subtract_matrices(A, B)
print("Addition Result:\n", addition)
print("Subtraction Result:\n", subtraction)
```

```
Addition Result:
 [[ 6  8]
 [10 12]]
Subtraction Result:
 [[-4 -4]
 [-4 -4]]
```

Q82

[278]:

```python
def matrix_multiply(C, D):
    return np.dot(C, D)

# Ex:
C = np.random.randint(1, 10, size=(3, 2))
D = np.random.randint(1, 10, size=(2, 4))
result = matrix_multiply(C, D)
print("Matrix Multiplication Result:\n", result)
```

```
Matrix Multiplication Result:
 [[34 36 14 14]
 [60 52 35 35]
 [50 48 25 25]]
```

Q83

[281]:

```python
def transpose_matrix(E):
    return E.T

# Ex:
E = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
transpose = transpose_matrix(E)
print("Transpose of E:\n", transpose)
```

```
Transpose of E:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Q84

[284]:

```python
def compute_determinant(F):
    return np.linalg.det(F)

# Ex:
F = np.array([[1, 2], [3, 4]])
determinant = compute_determinant(F)
print("Determinant of F:", determinant)
```

```
Determinant of F: -2.0000000000000004
```

Q85

[287]:

```
def find_inverse(G):
    return np.linalg.inv(G)

# Ex:
G = np.array([[1, 2], [3, 4]])
inverse = find_inverse(G)
print("Inverse of G:\n", inverse)
```

```
Inverse of G:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

[ ]: