ASSIGNMENT (NUMPY)

Q1. Create a NumPy array 'arr' of integers from 0 to 5 and print its data type?

To create a NumPy array arr with integers from 0 to 5 and print its data type, you can use the following code:

```
import numpy as np

# Creating the NumPy array 'arr' with integers from 0 to 5

arr = np.arange(6)

# Printing the data type of the array

print("Data type of the array:", arr.dtype)
```

When you run this code, it will print the data type of the array. The expected data type will be int64 (or int32 depending on your system architecture).

Q2. Given a NumPy array 'arr', check if its data type is float64

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

To check if the data type of the NumPy array arr is float64, you can use the following code:

```
import numpy as np

# Creating the NumPy array 'arr'

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

# Checking if the data type is float64

if arr.dtype == np.float64:
    print("The data type is float64.")
else:
    print("The data type is not float64.")
```

This code will check whether the data type of arr is float64 and print the corresponding message.

Q3. Create a NumPy array 'arr' with a data type of complex128 containing three complex numbers?

you can use the following code:

```
# Creating the NumPy array 'arr' with complex numbers

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

# Printing the array

print(arr)
```

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

Q4. Convert an existing NumPy array 'arr' of integers to float32 data type.

import numpy as np

```
# Example NumPy array 'arr' of integers
arr = np.array([1, 2, 3, 4, 5])
# Converting 'arr' to float32 data type
arr_float32 = arr.astype(np.float32)
# Printing the converted array and its data type
print(arr_float32)
print("Data type after conversion:", arr_float32.dtype)
```

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

Data type after conversion: float32

Q5. Given a NumPy array 'arr' with float64 data type, convert it to float32 to reduce decimal precision.

```
# Example NumPy array 'arr' with float64 data type
arr = np.array([1.123456789, 2.987654321, 3.141592653], dtype=np.float64)

# Converting 'arr' to float32 to reduce decimal precision
arr_float32 = arr.astype(np.float32)

# Printing the converted array and its data type
print(arr_float32)
print("Data type after conversion:", arr_float32.dtype)
```

[1.1234568 2.9876542 3.1415927]

Data type after conversion: float32

Q6. Write a function array_attributes that takes a NumPy array as input and returns its shape, size, and data type?

```
def array_attributes(arr):
    # Getting the shape, size, and data type of the array
    shape = arr.shape
    size = arr.size
    dtype = arr.dtype
```

```
  return shape, size, dtype

# Example usage
arr = np.array([1, 2, 3, 4, 5])
shape, size, dtype = array_attributes(arr)
# Printing the results
print("Shape:", shape)
print("Size:", size)
print("Data type:", dtype)
```

Shape: (5,)

Size: 5

Data type: int32

Q7. Create a function array_dimension that takes a NumPy array as input and returns its dimensionality.?

```
def array_dimension(arr):
    # Getting the dimensionality of the array
    return arr.ndim

# Example usage
arr = np.array([[1, 2, 3], [4, 5, 6]])
dim = array_dimension(arr)

# Printing the result
print("Dimensionality of the array:", dim)
```

Dimensionality of the array: 2

Q8. 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):
    # Getting the item size (in bytes) and total size (in bytes)
    item_size = arr.itemsize
    total_size = arr.nbytes
```

```
    return item_size, total_size

# Example usage
arr = np.array([1, 2, 3, 4, 5])
item_size, total_size = item_size_info(arr)

# Printing the results
print("Item size (in bytes):", item_size)
print("Total size (in bytes):", total_size)
```

Item size (in bytes): 4

Total size (in bytes): 20

Q9. Create a function array_strides that takes a NumPy array as input and returns the strides of the array?

```
def array_strides(arr):
    # Getting the strides of the array
    return arr.strides

# Example usage
arr = np.array([[1, 2, 3], [4, 5, 6]])
strides = array_strides(arr)

# Printing the result
print("Strides of the array:", strides)
```

Strides of the array: (12, 4)

Q10. 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):
    # Getting the shape and strides of the array
    shape = arr.shape
    strides = arr.strides
```

```
    return shape, strides

# Example usage
arr = np.array([[1, 2, 3], [4, 5, 6]])
shape, strides = shape_stride_relationship(arr)

# Printing the results
print("Shape of the array:", shape)
print("Strides of the array:", strides)
```

Shape of the array: (2, 3)

Strides of the array: (12, 4)

Q11. Create a function `create_zeros_array` that takes an integer `n` as input and returns a NumPy array of zeros with `n` elements.?

```
def create_zeros_array(n):
    # Creating a NumPy array of zeros with 'n' elements
    return np.zeros(n)

# Example usage
n = 5
zeros_array = create_zeros_array(n)

# Printing the result
print("Array of zeros:", zeros_array)
```

Array of zeros: [0. 0. 0. 0. 0.]

Q12. 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(rows, cols):
    # Creating a 2D NumPy array of ones with 'rows' and 'cols'
    return np.ones((rows, cols))
```

```
# Example usage
rows = 3
cols = 4
ones_matrix = create_ones_matrix(rows, cols)

# Printing the result
print("Matrix of ones:\n", ones_matrix)
```

Matrix of ones:

[[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]]

Q13. 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(start, stop, step):
    # Creating a NumPy array with the specified range and step
    return np.arange(start, stop, step)

# Example usage
start = 2
stop = 10
step = 2
range_array = generate_range_array(start, stop, step)

# Printing the result
print("Generated range array:", range_array)
```

Generated range array: [2 4 6 8]

Q14. 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(start, stop, num):
```

```
    # Generating a NumPy array with 'num' equally spaced values between 'start' and 'stop'
    return np.linspace(start, stop, num)

# Example usage
start = 1.0
stop = 5.0
num = 5
linear_space_array = generate_linear_space(start, stop, num)

# Printing the result
print("Generated linear space array:", linear_space_array)
```

Generated linear space array: [1. 2. 3. 4. 5.]

Q15Create 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(n):
    # Creating a square identity matrix of size 'n x n'
    return np.eye(n)

# Example usage
n = 4
identity_matrix = create_identity_matrix(n)

# Printing the result
print("Identity matrix:\n", identity_matrix)
```

Identity matrix:

[[1. 0. 0. 0.]

[0. 1. 0. 0.]

[0. 0. 1. 0.]

[0. 0. 0. 1.]]

[ ]:

Q16. Write a function that takes a Python list and converts it into a NumPy array

```
def list_to_numpy_array(py_list):
    # Converting the Python list to a NumPy array
    return np.array(py_list)

# Example usage
py_list = [1, 2, 3, 4, 5]
numpy_array = list_to_numpy_array(py_list)

# Printing the result
print("Converted NumPy array:", numpy_array)
```

Converted NumPy array: [1 2 3 4 5]

Q17. . Create a NumPy array and demonstrate the use of `numpy.view` to create a new array object with the same data.

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

# Using numpy.view to create a new array object with the same data
view_arr = arr.view()

# Printing both arrays to show they have the same data
print("Original array:", arr)
print("New view array:", view_arr)

# Modifying the view array to show that changes affect the original array
view_arr[0] = 10
print("\nAfter modifying the view array:")
print("Original array:", arr)
print("New view array:", view_arr)
```

Original array: [1 2 3 4 5]

New view array: [1 2 3 4 5]

After modifying the view array:

Original array: [10 2 3 4 5]

New view array: [10 2 3 4 5]

Q18. Write a function that takes two NumPy arrays and concatenates them along a specified axis.

```
def concatenate_arrays(arr1, arr2, axis=0):
    # Concatenating the two arrays along the specified axis
    return np.concatenate((arr1, arr2), axis=axis)

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

# Concatenating along axis 0 (vertically)
result_axis_0 = concatenate_arrays(arr1, arr2, axis=0)
print("Concatenated along axis 0:\n", result_axis_0)

# Concatenating along axis 1 (horizontally)
result_axis_1 = concatenate_arrays(arr1, arr2, axis=1)
print("\nConcatenated along axis 1:\n", result_axis_1)
```

Concatenated along axis 0:

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

Concatenated along axis 1:

[[1 2 5 6]

[3 4 7 8]]

Q19. Create two NumPy arrays with different shapes and concatenate them horizontally using `numpy. concatenate`.

```
# Create two NumPy arrays with different shapes
array1 = np.array([[1, 2, 3], [4, 5, 6]])  # Shape (2, 3)
array2 = np.array([[7, 8], [9, 10]])       # Shape (2, 2)

# Concatenate them horizontally
concatenated_array = np.concatenate((array1, array2), axis=1)

print(concatenated_array)
```

[[1 2 3 7 8]

 [4 5 6 9 10]]

20. Write a function that vertically stacks multiple NumPy arrays given as a list.

```
def vertical_stack(arrays):
    # Use np.vstack to stack the arrays vertically
    return np.vstack(arrays)

# Example usage:
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
array3 = np.array([7, 8, 9])

arrays = [array1, array2, array3]
stacked_array = vertical_stack(arrays)

print(stacked_array)
```

[[1 2 3]

 [4 5 6]

[7 8 9]]

Q21. Write a Python function using NumPy to create an array of integers within a specified range (inclusive) with a given step size.

```
def create_array(start, end, step):
    # Create an array using np.arange that includes the 'end' value
    return np.arange(start, end + 1, step)

# Example usage:
start = 1
end = 10
step = 2

array = create_array(start, end, step)
print(array)
```

[1 3 5 7 9]

Q22. Write a Python function using NumPy to generate an array of 10 equally spaced values between 0 and 1 (inclusive).

```
def generate_equally_spaced_values(start, end, num_values):
    # Generate an array of equally spaced values
    return np.linspace(start, end, num_values)

# Example usage:
start = 0
end = 1
num_values = 10

array = generate_equally_spaced_values(start, end, num_values)
print(array)
```

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

Q23. Write a Python function using NumPy to create an array of 5 logarithmically spaced values between 1 and 1000 (inclusive).

```
def generate_logarithmically_spaced_values(start, end, num_values):
    # Generate an array of logarithmically spaced values
    return np.logspace(np.log10(start), np.log10(end), num_values)

# Example usage:
start = 1
end = 1000
num_values = 5

array = generate_logarithmically_spaced_values(start, end, num_values)
print(array)
```

[1. 5.62341325 31.6227766 177.827941 1000.]

Q24. 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.

```
# Generate a NumPy array with random integers between 1 and 100 (5 rows, 3 columns)
random_array = np.random.randint(1, 101, size=(5, 3))

# Create a Pandas DataFrame from the NumPy array
df = pd.DataFrame(random_array, columns=['A', 'B', 'C'])

# Display the DataFrame
print(df)
```

	A	B	C
0	50	39	59
1	74	83	30
2	70	43	91
3	94	97	51
4	42	42	62

Q25. 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.

```
def replace_negatives_with_zeros(df, column_name):
    # Use np.where to replace negative values with zeros in the specified column
    df[column_name] = np.where(df[column_name] < 0, 0, df[column_name])
    return df

# Example usage:
data = {'A': [10, -5, 15, -3, 8], 'B': [-1, 20, -30, 40, 50]}
df = pd.DataFrame(data)

print("Before replacing negatives:")
print(df)

# Replace negative values in column 'A' with zeros
df = replace_negatives_with_zeros(df, 'A')

print("\nAfter replacing negatives in column 'A':")
print(df)
```

Before replacing negatives:

	A	B
0	10	-1
1	-5	20
2	15	-30
3	-3	40
4	8	50

After replacing negatives in column 'A':

	A	B
0	10	-1
1	0	20
2	15	-30

3 0 40

4 8 50

Q26. Access the 3rd element from the given NumPy array.

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

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

# Access the 3rd element (index 2)
third_element = arr[2]
print(third_element)
```

30

Q27. Retrieve the element at index (1, 2) from the 2D NumPy array.

arr_2d = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

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

# Access the element at index (1, 2)
element = arr_2d[1, 2]
print(element)
```

6

Q28. Using boolean indexing, extract elements greater than 5 from the given NumPy array

arr = np.array([3, 8, 2, 10, 5, 7])

arr = np.array([3, 8, 2, 10, 5, 7])

```
# Use boolean indexing to extract elements greater than 5
filtered_arr = arr[arr > 5]
```

```
print(filtered_arr)
```

```
[ 8 10  7]
```

Q29. Perform basic slicing to extract elements from index 2 to 5 (inclusive) from the given NumPy array.

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

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


# Perform basic slicing to extract elements from index 2 to 5 (inclusive)
sliced_arr = arr[2:6]


print(sliced_arr)
```

```
[3 4 5 6]
```

Q30. Slice the 2D NumPy array to extract the sub-array `[[2, 3], [5, 6]]` from the given array.

arr_2d = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

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


# Slice the 2D array to extract the sub-array [[2, 3], [5, 6]]
sub_array = arr_2d[0:2, 1:3]


print(sub_array)
```

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

Q31. Write a NumPy function to extract elements in specific order from a given 2D array based on indices provided in another array.

```
def extract_elements_by_indices(arr_2d, indices):
    # Use advanced indexing to extract elements
```

```
    return arr_2d[indices[:, 0], indices[:, 1]]

# Example usage:
arr_2d = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Indices in the form of a 2D array (row, column)
indices = np.array([[0, 1],  # element at row 0, col 1 -> 2
                    [1, 2],  # element at row 1, col 2 -> 6
                    [2, 0]]) # element at row 2, col 0 -> 7

# Extract elements based on the indices
extracted_elements = extract_elements_by_indices(arr_2d, indices)

print(extracted_elements)
[2 6 7]
```

Q32. Create a NumPy function that filters elements greater than a threshold from a given 1D array using boolean indexing.

```
def filter_elements_greater_than_threshold(arr, threshold):
    # Use boolean indexing to filter elements greater than the threshold
    return arr[arr > threshold]

# Example usage:
arr = np.array([1, 5, 8, 12, 3, 7, 4])
threshold = 6

# Filter elements greater than the threshold
filtered_arr = filter_elements_greater_than_threshold(arr, threshold)

print(filtered_arr)
```

[8 12 7]

Q33. Develop a NumPy function that extracts specific elements from a 3D array using indices provided in three separate arrays for each dimension.

```
def extract_elements_from_3d_array(arr_3d, row_indices, col_indices, depth_indices):
    # Use advanced indexing to extract elements based on the provided indices
    return arr_3d[row_indices, col_indices, depth_indices]


# Example usage:
arr_3d = np.array([[[1, 2], [3, 4], [5, 6]],
                   [[7, 8], [9, 10], [11, 12]],
                   [[13, 14], [15, 16], [17, 18]]])


# Indices for each dimension
row_indices = np.array([0, 1, 2])  # Row indices
col_indices = np.array([1, 0, 2])  # Column indices
depth_indices = np.array([0, 1, 0])  # Depth indices


# Extract elements from the 3D array
```

Q34. Write a NumPy function that returns elements from an array where both two conditions are satisfied using boolean indexing.

```
def filter_elements_by_conditions(arr, condition1, condition2):
    # Use boolean indexing to return elements where both conditions are satisfied
    return arr[condition1 & condition2]


# Example usage:
arr = np.array([1, 5, 8, 12, 3, 7, 4])


# Define two conditions: (1) elements greater than 5, (2) elements are even
condition1 = arr > 5
condition2 = arr % 2 == 0
```

```
# Filter elements that satisfy both conditions
filtered_arr = filter_elements_by_conditions(arr, condition1, condition2)

print(filtered_arr)
```

[8 12]

Q35. Create a NumPy function that extracts elements from a 2D array using row and column indices provided in separate arrays

```
def extract_elements_from_2d_array(arr_2d, row_indices, col_indices):
    # Use advanced indexing to extract elements based on row and column indices
    return arr_2d[row_indices, col_indices]

# Example usage:
arr_2d = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Row and column indices
row_indices = np.array([0, 1, 2])  # Row indices
col_indices = np.array([1, 0, 2])  # Column indices

# Extract elements from the 2D array using the provided indices
extracted_elements = extract_elements_from_2d_array(arr_2d, row_indices, col_indices)

print(extracted_elements)
```

[2 4 9]

Q36. Given an array arr of shape (3, 3), add a scalar value of 5 to each element using NumPy broadcasting.

```
# Create a 3x3 array
arr = np.array([[1, 2, 3],
```

```
                [4, 5, 6],
                [7, 8, 9]])

# Add a scalar value of 5 to each element using broadcasting
arr_plus_5 = arr + 5

print(arr_plus_5)
[[ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
```

Q37. 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.

```
# Define arr1 with shape (1, 3)
arr1 = np.array([[2, 3, 4]])

# Define arr2 with shape (3, 4)
arr2 = np.array([[1, 2, 3, 4],
                 [5, 6, 7, 8],
                 [9, 10, 11, 12]])

# Multiply each row of arr2 by the corresponding element in arr1 using broadcasting
result = arr2 * arr1

print(result)
```

Q38. 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.

```
# Define arr1 with shape (1, 4)
arr1 = np.array([[1, 2, 3, 4]])
```

```
# Define arr2 with shape (4, 3)
arr2 = np.array([[5, 6, 7],
                 [8, 9, 10],
                 [11, 12, 13],
                 [14, 15, 16]])

# Add arr1 to each row of arr2 using broadcasting
result = arr2 + arr1

print(result)
```

Q39. Consider two arrays arr1 of shape (3, 1) and arr2 of shape (1, 3). Add these arrays using NumPy broadcasting.

```
# Define arr1 with shape (1, 3)
arr1 = np.array([[1, 2, 3]])

# Define arr2 with shape (4, 3)
arr2 = np.array([[5, 6, 7],
                 [8, 9, 10],
                 [11, 12, 13],
                 [14, 15, 16]])

# Add arr1 to each row of arr2 using broadcasting
result = arr2 + arr1

print(result)
[[ 6  8 10]
 [ 9 11 13]
 [12 14 16]
 [15 17 19]]
```

Q41. Calculate column wise mean for the given array

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

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

# Calculate column-wise mean
column_mean = np.mean(arr, axis=0)

print(column_mean)
```

[2.5 3.5 4.5]

Q42. Find maximum value in each row of the given array:

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

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

# Find the maximum value in each row
row_max = np.max(arr, axis=1)

print(row_max)
```

[3 6]

Q43. For the given array, find indices of maximum value in each column.

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

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

# Find indices of maximum value in each column
max_indices = np.argmax(arr, axis=0)

print(max_indices)
```

[1 1 1]

Q44. For the given array, apply custom function to calculate moving sum along rows

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

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


# Custom function to calculate the moving sum along rows
def moving_sum(arr, window=2):
    return np.array([np.convolve(row, np.ones(window, dtype=int), mode='valid') for row in arr])


# Apply the custom moving sum function
result = moving_sum(arr)


print(result)
```

[[3 5]

 [9 11]]

Q45. In the given array, check if all elements in each column are even

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

```
# Given array
arr = np.array([[2, 4, 6],
                [3, 5, 7]])


# Check if all elements in each column are even
result = np.all(arr % 2 == 0, axis=0)


print(result)
```

[False False False]

Q.46. 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])

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

# Reshape it into a matrix with 2 rows and 3 columns
reshaped_matrix = original_array.reshape(2, 3)

print(reshaped_matrix)
```

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

Q47.Create a function that takes a matrix as input and returns the flattened array.

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

```
# Function to flatten a matrix
def flatten_matrix(matrix):
    return matrix.flatten()

# Input matrix
input_matrix = np.array([[1, 2, 3], [4, 5, 6]])

# Call the function
flattened_array = flatten_matrix(input_matrix)

print(flattened_array)
```

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

Q48. Write a function that concatenates two given arrays along a specified axis.

array1 = np.array([[1, 2], [3, 4]])

array2 = np.array([[5, 6], [7, 8]])

```
# Function to concatenate two arrays along a specified axis
def concatenate_arrays(array1, array2, axis=0):
    return np.concatenate((array1, array2), axis=axis)

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

# Concatenate along rows (axis=0)
result_axis_0 = concatenate_arrays(array1, array2, axis=0)

# Concatenate along columns (axis=1)
result_axis_1 = concatenate_arrays(array1, array2, axis=1)

print("Concatenated along rows (axis=0):")
print(result_axis_0)

print("\nConcatenated along columns (axis=1):")
print(result_axis_1)
```

```
Concatenated along rows (axis=0):
[[1 2]
 [3 4]
 [5 6]
 [7 8]]

Concatenated along columns (axis=1):
[[1 2 5 6]
 [3 4 7 8]]
```

Q49. Create a function that splits an array into multiple sub-arrays along a specified axis

```
original_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Function to split an array into sub-arrays along a specified axis
def split_array(array, num_splits, axis=0):
    return np.split(array, num_splits, axis=axis)

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

# Split along rows (axis=0)
result_axis_0 = split_array(original_array, 3, axis=0)

# Split along columns (axis=1)
result_axis_1 = split_array(original_array, 3, axis=1)

print("Split along rows (axis=0):")
for sub_array in result_axis_0:
    print(sub_array)

print("\nSplit along columns (axis=1):")
for sub_array in result_axis_1:
    print(sub_array)
Split along rows (axis=0):
[[1 2 3]]
[[4 5 6]]
[[7 8 9]]

Split along columns (axis=1):
[[1]
```

[4]

[7]]

[[2]

[5]

[8]]

[[3]

[6]

[9]]

Q50. 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]

```
# Function to insert and delete elements from an array
def insert_and_delete_elements(original_array, indices_to_insert, values_to_insert, indices_to_delete):
    # Insert values at specified indices
    array_after_insert = np.insert(original_array, indices_to_insert, values_to_insert)

    # Delete elements at specified indices
    array_after_delete = np.delete(array_after_insert, indices_to_delete)

    return array_after_insert, array_after_delete

# Given array and 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]

# Call the function
```

```
array_after_insert, array_after_delete =
insert_and_delete_elements(original_array, indices_to_insert,
values_to_insert, indices_to_delete)


# Print results

print("Array after insertions:")

print(array_after_insert)


print("\nArray after deletions:")

print(array_after_delete)

import numpy as np


# Function to insert and delete elements from an array

def insert_and_delete_elements(original_array, indices_to_insert,
values_to_insert, indices_to_delete):

    # Insert values at specified indices

    array_after_insert = np.insert(original_array, indices_to_insert,
values_to_insert)


    # Delete elements at specified indices

    array_after_delete = np.delete(array_after_insert, indices_to_delete)


    return array_after_insert, array_after_delete


# Given array and 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]


# Call the function
```

```
array_after_insert, array_after_delete =
insert_and_delete_elements(original_array, indices_to_insert,
values_to_insert, indices_to_delete)


# Print results

print("Array after insertions:")

print(array_after_insert)


print("\nArray after deletions:")

print(array_after_delete)
```

```
Array after insertions:

[ 1  2 10  3  4 11  5]


Array after deletions:

[ 1 10  4 11  5]
```

Q51. 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`.

```
import numpy as np


# Create a NumPy array `arr1` with random integers (let's say of size 5)

arr1 = np.random.randint(1, 10, size=5)


# Create a NumPy array `arr2` with integers from 1 to 10

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


# Perform element-wise addition

result = arr1 + arr2


print("Array 1 (Random Integers):", arr1)

print("Array 2 (Integers from 1 to 5):", arr2)
```

```
print("Element-wise Addition Result:", result)
```

```
Array 1 (Random Integers): [5 9 6 6 2]
Array 2 (Integers from 1 to 5): [1 2 3 4 5]
Element-wise Addition Result: [ 6 11  9 10  7]
```

Q52. 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.

```
# Create a NumPy array `arr1` with sequential integers from 10 to 1
arr1 = np.arange(10, 0, -1)

# Create a NumPy array `arr2` with integers from 1 to 10
arr2 = np.arange(1, 11)

# Perform element-wise subtraction (arr1 - arr2)
result = arr1 - arr2

print("Array 1 (Sequential Integers from 10 to 1):", arr1)
print("Array 2 (Integers from 1 to 10):", arr2)
print("Element-wise Subtraction Result:", result)
```

```
Array 1 (Sequential Integers from 10 to 1): [10  9  8  7  6  5  4  3  2  1]
Array 2 (Integers from 1 to 10): [ 1  2  3  4  5  6  7  8  9 10]
Element-wise Subtraction Result: [ 9  7  5  3  1 -1 -3 -5 -7 -9]
```

Q53. 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`.

```
# Create a NumPy array `arr1` with random integers (let's say of size 5)
arr1 = np.random.randint(1, 10, size=5)

# Create a NumPy array `arr2` with integers from 1 to 5
arr2 = np.array([1, 2, 3, 4, 5])

# Perform element-wise multiplication
```

result = arr1 * arr2

print("Array 1 (Random Integers):", arr1)

print("Array 2 (Integers from 1 to 5):", arr2)

print("Element-wise Multiplication Result:", result)

Array 1 (Random Integers): [6 7 9 1 7]

Array 2 (Integers from 1 to 5): [1 2 3 4 5]

Element-wise Multiplication Result: [6 14 27 4 35]

Q54. 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`.

 # Create a NumPy array `arr1` with random integers (let's say of size 5)

arr1 = np.random.randint(1, 10, size=5)

 # Create a NumPy array `arr2` with integers from 1 to 5

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

 # Perform element-wise multiplication

result = arr1 * arr2

print("Array 1 (Random Integers):", arr1)

print("Array 2 (Integers from 1 to 5):", arr2)

print("Element-wise Multiplication Result:", result)

Array 1 (Random Integers): [4 5 3 1 5]

Array 2 (Integers from 1 to 5): [1 2 3 4 5]

Element-wise Multiplication Result: [4 10 9 4 25]

Q55. 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

Create a NumPy array `arr1` with integers from 1 to 5

arr1 = np.arange(1, 6)

```
# Create a NumPy array `arr2` with the same numbers reversed
arr2 = arr1[::-1]

# Perform element-wise exponentiation (arr1 raised to the power of arr2)
result = arr1 ** arr2

print("Array 1 (Integers from 1 to 5):", arr1)
print("Array 2 (Reversed Integers):", arr2)
print("Element-wise Exponentiation Result:", result)
```

```
Array 1 (Integers from 1 to 5): [1 2 3 4 5]
Array 2 (Reversed Integers): [5 4 3 2 1]
Element-wise Exponentiation Result: [ 1 16 27 16  5]
```

Q56. 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'])
# Function to count occurrences of a specific substring
def count_substring(arr, substring):
    # Use np.char.find() to find the substring in each element of the array
    return np.sum(np.char.find(arr, substring) != -1)

# Given NumPy array of strings
arr = np.array(['hello', 'world', 'hello', 'numpy', 'hello'])
```

The substring 'hello' occurs 3 times in the array.

Q57. 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'])
# Function to extract uppercase characters from a NumPy array of strings
def extract_uppercase(arr):
    # Create an empty list to store the uppercase characters
    uppercase_chars = []
```

```
    # Loop through each string in the array
    for string in arr:
        # Extract uppercase characters from each string
        uppercase_chars.extend([char for char in string if char.isupper()])

    return uppercase_chars

# Given NumPy array of strings
arr = np.array(['Hello', 'World', 'OpenAI', 'GPT'])

# Extract uppercase characters
uppercase_characters = extract_uppercase(arr)

print("Uppercase characters extracted:", uppercase_characters)
```

Uppercase characters extracted: ['H', 'W', 'O', 'A', 'I', 'G', 'P', 'T']

Q58. 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'])

# Function to replace occurrences of a substring with a new string in a NumPy array
def replace_substring(arr, old_substring, new_substring):
    # Use np.char.replace() to replace the substring
    return np.char.replace(arr, old_substring, new_substring)

# Given NumPy array of strings
arr = np.array(['apple', 'banana', 'grape', 'pineapple'])

# Define the old substring and the new substring
old_substring = 'apple'
new_substring = 'orange'
```

```
# Replace occurrences of the substring
modified_arr = replace_substring(arr, old_substring, new_substring)


print("Original array:", arr)
print("Modified array:", modified_arr)
```

Original array: ['apple' 'banana' 'grape' 'pineapple']

Modified array: ['orange' 'banana' 'grape' 'pineorange']

Q59. Write a function that concatenates strings in a NumPy array element-wise. arr1 = np.array(['Hello', 'World']) arr2 = np.array(['Open', 'AI'])

```
# Function to concatenate strings element-wise from two NumPy arrays
def concatenate_strings(arr1, arr2):
    # Use np.char.add() to concatenate strings element-wise
    return np.char.add(arr1, arr2)


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


# Concatenate the arrays
concatenated_arr = concatenate_strings(arr1, arr2)
print("Concatenated array:", concatenated_arr)
```

Concatenated array: ['HelloOpen' 'WorldAI']

Q60. Write a function that finds the length of the longest string in a NumPy array. arr = np.array(['apple', 'banana', 'grape', 'pineapple'])

```
arr = np.array(['apple', 'banana', 'grape', 'pineapple'])
# Function to find the length of the longest string in a NumPy array
def longest_string_length(arr):
    # Use np.char.str_len() to get the length of each string in the array
    lengths = np.char.str_len(arr)


    # Find and return the maximum length
```

```
    return np.max(lengths)

# Given NumPy array of strings
arr = np.array(['apple', 'banana', 'grape', 'pineapple'])

# Find the length of the longest string
max_length = longest_string_length(arr)

print("Length of the longest string:", max_length)
```

Length of the longest string: 9

Q61 . 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.

```
# Generate a dataset of 100 random integers between 1 and 1000
dataset = np.random.randint(1, 1001, size=100)

# Compute the mean, median, variance, and standard deviation
mean = np.mean(dataset)
median = np.median(dataset)
variance = np.var(dataset)
std_deviation = np.std(dataset)

# Print the results
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Variance: {variance}")
print(f"Standard Deviation: {std_deviation}")
```

Mean: 509.67

Median: 543.5

Variance: 83440.70109999999

Standard Deviation: 288.8610411599321

Q62. Generate an array of 50 random numbers between 1 and 100. Find the 25th and 75th percentiles of the dataset.

```
# Generate an array of 50 random numbers between 1 and 100
dataset = np.random.randint(1, 101, size=50)

# Compute the 25th and 75th percentiles
percentile_25 = np.percentile(dataset, 25)
percentile_75 = np.percentile(dataset, 75)

# Print the results
print(f"25th Percentile: {percentile_25}")
print(f"75th Percentile: {percentile_75}")
```

25th Percentile: 19.25

75th Percentile: 71.0

Q63. Create two arrays representing two sets of variables. Compute the correlation coefficient between these arrays using NumPy's `corrcoef` function

```
# Create two arrays representing two sets of variables
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([5, 4, 3, 2, 1])

# Compute the correlation coefficient matrix
correlation_matrix = np.corrcoef(arr1, arr2)

# Extract the correlation coefficient between arr1 and arr2
correlation_coefficient = correlation_matrix[0, 1]

# Print the result
print(f"Correlation Coefficient: {correlation_coefficient}")
```

Correlation Coefficient: -0.9999999999999999

Q64. Create two matrices and perform matrix multiplication using NumPy's `dot` function

```
# Create two matrices

matrix1 = np.array([[1, 2], [3, 4]])

matrix2 = np.array([[5, 6], [7, 8]])


# Perform matrix multiplication using np.dot()

result = np.dot(matrix1, matrix2)


# Print the result

print("Matrix multiplication result:\n", result)
```

```
Matrix multiplication result:
 [[19 22]
 [43 50]]
```

Q65. 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.

```
# Create an array of 50 random integers between 10 and 1000

arr = np.random.randint(10, 1001, size=50)


# Calculate the 10th, 50th (median), and 90th percentiles

percentile_10 = np.percentile(arr, 10)

percentile_50 = np.percentile(arr, 50)  # This is also the median

percentile_90 = np.percentile(arr, 90)


# Calculate the first and third quartiles (25th and 75th percentiles)

quartile_25 = np.percentile(arr, 25)

quartile_75 = np.percentile(arr, 75)


# Print the results

print(f"10th Percentile: {percentile_10}")

print(f"50th Percentile (Median): {percentile_50}")
```

print(f"90th Percentile: {percentile_90}")

print(f"First Quartile (25th Percentile): {quartile_25}")

print(f"Third Quartile (75th Percentile): {quartile_75}")

10th Percentile: 95.00000000000001

50th Percentile (Median): 510.0

90th Percentile: 910.2

First Quartile (25th Percentile): 285.5

Third Quartile (75th Percentile): 816.25

Q66. Create a NumPy array of integers and find the index of a specific element.

Create a NumPy array of integers

arr = np.array([10, 20, 30, 40, 50, 60, 70])

Find the index of a specific element, for example, 40

element = 40

index = np.where(arr == element)

Print the index

print(f"The index of {element} is: {index[0][0]}")

The index of 40 is: 3

Q67. Generate a random NumPy array and sort it in ascending order

 # Generate a random NumPy array of 10 integers between 1 and 100

arr = np.random.randint(1, 101, size=10)

 # Sort the array in ascending order

sorted_arr = np.sort(arr)

 # Print the original and sorted arrays

print("Original array:", arr)

print("Sorted array:", sorted_arr)

Original array: [52 7 52 87 4 10 7 29 97 52]

Sorted array: [4 7 7 10 29 52 52 52 87 97]

Q68. Filter elements >20 in the given NumPy array.

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

```
# Given array
arr = np.array([12, 25, 6, 42, 8, 30])

# Filter elements greater than 20
filtered_arr = arr[arr > 20]

# Print the filtered array
print("Elements greater than 20:", filtered_arr)
```

Elements greater than 20: [25 42 30]

Q69.Filter elements which are divisible by 3 from a given NumPy array.

arr = np.array([1, 5, 8, 12, 15])

```
# Given array
arr = np.array([1, 5, 8, 12, 15])

# Filter elements divisible by 3
filtered_arr = arr[arr % 3 == 0]

# Print the filtered array
print("Elements divisible by 3:", filtered_arr)
```

Elements divisible by 3: [12 15]

Q70. Filter elements which are ≥ 20 and ≤ 40 from a given NumPy array

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

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

# Filter elements that are ≥ 20 and ≤ 40
```

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


# Print the filtered array
print("Elements ≥ 20 and ≤ 40:", filtered_arr)
```

```
Elements ≥ 20 and ≤ 40: [20 30 40]
```