```
1
```

Dimension and Shape

5 .Write a code snippet to generate a 3-D NumPy array.

```
import numpy as np

# Define the dimensions
dim1 = 3
dim2 = 4
dim3 = 5

# Generate the 3-D NumPy array
array_3d = np.random.rand(dim1, dim2, dim3)

print(array_3d)

```

```
[[[0.28897002 0.70273145 0.42750229 0.45028973 0.14715277]
  [0.53299195 0.30994933 0.74696399 0.83977195 0.75727399]
  [0.17012249 0.70807614 0.05450343 0.78519517 0.6000791 ]
  [0.01531846 0.23887096 0.38675369 0.99022716 0.84912939]]

 [[0.11622418 0.54791287 0.60457364 0.76361887 0.83606912]
  [0.03022956 0.74713951 0.49142214 0.30568219 0.89889268]
  [0.36447238 0.55445455 0.8595426  0.65013951 0.87464767]
  [0.52755788 0.81801987 0.10760364 0.95698037 0.18050241]]

 [[0.00887125 0.72732915 0.98913172 0.44597653 0.48831517]
  [0.70497112 0.61043375 0.69174224 0.65269147 0.77704558]
  [0.48308862 0.22275181 0.99410177 0.47698008 0.22989744]
  [0.97391464 0.727081   0.95992233 0.84670669 0.03623786]]]
```

```
1
```

6. How do you create a NumPy array with a specified number of dimensions using the ndmin parameter?

```
import numpy as np

# Create a 1D array with ndmin=2
arr_2d = np.array([1, 2, 3, 4], ndmin=2)
print("2D Array:")
print(arr_2d)
print("Shape:", arr_2d.shape)

# Create a 2D array with ndmin=3
arr_3d = np.array([[1, 2], [3, 4]], ndmin=3)
print("\n3D Array:")
print(arr_3d)
print("Shape:", arr_3d.shape)

```

```
2D Array:
[[1 2 3 4]]
Shape: (1, 4)

3D Array:
[[[1 2]
  [3 4]]]
Shape: (1, 2, 2)
```

8. How can you create a NumPy array of zeros with a specified shape?

```
import numpy as np

# Create a 1D array of zeros with shape (5,)
arr_1d = np.zeros(5)
print("1D Array of Zeros:")
print(arr_1d)
print("Shape:", arr_1d.shape)

# Create a 2D array of zeros with shape (3, 4)
arr_2d = np.zeros((3, 4))
print("\n2D Array of Zeros:")
print(arr_2d)
print("Shape:", arr_2d.shape)

# Create a 3D array of zeros with shape (2, 3, 2)
arr_3d = np.zeros((2, 3, 2))
print("\n3D Array of Zeros:")
print(arr_3d)
print("Shape:", arr_3d.shape)

```

```
1D Array of Zeros:
[0. 0. 0. 0. 0.]
Shape: (5,)

2D Array of Zeros:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Shape: (3, 4)

3D Array of Zeros:
[[[0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]]]
Shape: (2, 3, 2)
```

8. How can you create a NumPy array of zeros with a specified shape?

```
import numpy as np

# Create a 1D array of ones with shape (5,)
arr_1d = np.ones(5)
print("1D Array of Ones:")
print(arr_1d)
print("Shape:", arr_1d.shape)

# Create a 2D array of ones with shape (3, 4)
arr_2d = np.ones((3, 4))
print("\n2D Array of Ones:")
print(arr_2d)
print("Shape:", arr_2d.shape)

# Create a 3D array of ones with shape (2, 3, 2)
arr_3d = np.ones((2, 3, 2))
print("\n3D Array of Ones:")
print(arr_3d)
print("Shape:", arr_3d.shape)

```

```
1D Array of Ones:
[1. 1. 1. 1. 1.]
Shape: (5,)

2D Array of Ones:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
Shape: (3, 4)

3D Array of Ones:
[[[1. 1.]
  [1. 1.]
  [1. 1.]]
```

```
 [[1. 1.]
  [1. 1.]
  [1. 1.]]]
Shape: (2, 3, 2)
```

10. What is the method to create an empty NumPy array?

```
import numpy as np

# Create an empty 1D array
empty_arr_1d = np.empty(5)
print("Empty 1D Array:")
print(empty_arr_1d)
print("Shape:", empty_arr_1d.shape)

# Create an empty 2D array
empty_arr_2d = np.empty((3, 4))
print("\nEmpty 2D Array:")
print(empty_arr_2d)
print("Shape:", empty_arr_2d.shape)

# Create an empty 3D array
empty_arr_3d = np.empty((2, 3, 2))
print("\nEmpty 3D Array:")
print(empty_arr_3d)
print("Shape:", empty_arr_3d.shape)

```

```
Empty 1D Array:
[0. 0. 0. 0. 0.]
Shape: (5,)

Empty 2D Array:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Shape: (3, 4)

Empty 3D Array:
[[[0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]]]
Shape: (2, 3, 2)
```

11. Show how to create a NumPy array with a range of elements using np.arange.

```
import numpy as np

# Create an array of integers from 0 to 9 (exclusive)
arr1 = np.arange(10)
print("Array 1:")
print(arr1)

# Create an array of integers from 3 to 9 (exclusive)
arr2 = np.arange(3, 10)
print("\nArray 2:")
print(arr2)

# Create an array of even numbers from 2 to 10 (exclusive)
arr3 = np.arange(2, 11, 2)
print("\nArray 3:")
print(arr3)

```

```
Array 1:
[0 1 2 3 4 5 6 7 8 9]

Array 2:
[3 4 5 6 7 8 9]
```

```
Array 3:
[ 2  4  6  8 10]
```

12. How do you create a NumPy array with evenly spaced intervals using np.linspace?

```
import numpy as np

# Create an array with 10 evenly spaced values from 0 to 1 (inclusive)
arr1 = np.linspace(0, 1, 10)
print("Array 1:")
print(arr1)

# Create an array with 5 evenly spaced values from 1 to 10 (inclusive)
arr2 = np.linspace(1, 10, 5)
print("\nArray 2:")
print(arr2)

```

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

Array 2:
[ 1.    3.25  5.5   7.75 10.  ]
```

13. What is the output of np.array([[1, 2, 3], [4, 5, 6]]).shape?

```
'''
The output of np.array([[1, 2, 3], [4, 5, 6]]).shape would be (2, 3). This indicates that the array has 2 rows and 3 columns.'''

```

```
1
```

14. How can you find the total number of elements in a NumPy array?

```
import numpy as np

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

# Find the total number of elements
total_elements = arr.size

print("Total number of elements in the array:", total_elements)

```

```
Total number of elements in the array: 6
```

15. Explain how to get the number of dimensions of a NumPy array.

```
import numpy as np

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

# Get the number of dimensions
num_dimensions = arr.ndim

print("Number of dimensions:", num_dimensions)

```

```
Number of dimensions: 2
```

```
1
```

16. Write a code snippet to change the data type of a NumPy array to float64.

```
import numpy as np

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

# Change the data type of the array to float64
arr_float64 = arr.astype(np.float64)

print("Original array:")
print(arr)
print("\nArray with data type float64:")
print(arr_float64)

```

```
Original array:
[[1 2 3]
 [4 5 6]]

Array with data type float64:
[[1. 2. 3.]
 [4. 5. 6.]]
```

17. convert the data type of a NumPy array to an integer?

```
import numpy as np

# Create a NumPy array with floating-point numbers
arr_float = np.array([1.1, 2.2, 3.3])

# Convert the data type to integer
arr_int = arr_float.astype(int)

print("Original array (float):", arr_float)
print("Array converted to integer:", arr_int)

```

```
Original array (float): [1.1 2.2 3.3]
Array converted to integer: [1 2 3]
```

18. Find the minimum and maximum values in a NumPy array [3, 7, 1, 8, 4, 6, 0, 2,5]

```
import numpy as np

# Create a NumPy array
arr = np.array([3, 7, 1, 8, 4, 6, 0, 2, 5])

# Find the minimum and maximum values
minimum_value = np.min(arr)
maximum_value = np.max(arr)

print("Minimum value:", minimum_value)
print("Maximum value:", maximum_value)

```

```
Minimum value: 0
Maximum value: 8
```

19. Demonstrate how to reshape a NumPy array.

```
import numpy as np

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

# Reshape the array to a 2x3 matrix
reshaped_arr = arr.reshape(2, 3)

print("Original array:")
print(arr)
print("\nReshaped array:")
print(reshaped_arr)

```

```
Original array:
[1 2 3 4 5 6]

Reshaped array:
[[1 2 3]
 [4 5 6]]
```

20. How can you split a NumPy array into three equal parts?

```
import numpy as np

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

# Check if the size of the array is divisible by 3
if len(arr) % 3 != 0:
    raise ValueError("Array size is not divisible by 3")

# Split the array into three equal parts
split_arr = np.split(arr, 3)

print("Original array:", arr)
print("Split array into three equal parts:", split_arr)

```

```
Original array: [1 2 3 4 5 6 7 8 9]
Split array into three equal parts: [array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]
```

21. Explain horizontal and vertical splitting of NumPy arrays.

```
import numpy as np

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

# Split the array into two smaller arrays horizontally
smaller_arrays = np.hsplit(arr, 3)

print(smaller_arrays)

```

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

```
import numpy as np

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

# Split the array into two smaller arrays vertically
smaller_arrays = np.vsplit(arr, 3)

print(smaller_arrays)

```

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

```
import numpy as np

# Create a 1-D NumPy array
arr = np.array([1, 2, 3, 4, 5])

# Add a new axis using np.newaxis
new_arr = arr[:, np.newaxis]

print("Original array (1-D):", arr)
print("New array (2-D):")
print(new_arr)

```

```
Original array (1-D): [1 2 3 4 5]
New array (2-D):
[[1]
 [2]
 [3]
 [4]
 [5]]
```

22. How do you add a new dimension to a 1-D NumPy array?

```
import numpy as np

# Create a 1-D NumPy array
arr = np.array([1, 2, 3, 4, 5])

# Add a new axis using np.expand_dims()
new_arr = np.expand_dims(arr, axis=1)

print("Original array (1-D):", arr)
print("New array (2-D):")
print(new_arr)

```

```
Original array (1-D): [1 2 3 4 5]
New array (2-D):
[[1]
 [2]
 [3]
 [4]
 [5]]
```

23. Write a code to flatten a 2-D NumPy array

```
import numpy as np

# Create a 2-D NumPy array
arr_2d = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Flatten the 2-D array using the flatten() method
flattened_arr = arr_2d.flatten()

print("Original 2-D array:")
print(arr_2d)
print("\nFlattened array:")
print(flattened_arr)

```

```
Original 2-D array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Flattened array:
[1 2 3 4 5 6 7 8 9]
```

24. Demonstrate how to sort a NumPy array.

```
import numpy as np

# Create a NumPy array
arr = np.array([3, 1, 4, 1, 5, 9, 2, 6])

# Sort the array
sorted_arr = np.sort(arr)

print("Original array:", arr)
print("Sorted array:", sorted_arr)


import numpy as np

# Create a 2D NumPy array
arr_2d = np.array([[3, 1, 4],
                   [1, 5, 9],
                   [2, 6, 5]])

# Sort the array along axis 1 (sort each row)
sorted_arr_2d = np.sort(arr_2d, axis=1)

print("Original 2D array:")
print(arr_2d)
print("\nSorted 2D array along axis 1:")
print(sorted_arr_2d)

import numpy as np

# Create a NumPy array
arr = np.array([3, 1, 4, 1, 5, 9, 2, 6])

# Sort the array in descending order
sorted_arr_desc = np.sort(arr)[::-1]

print("Original array:", arr)
print("Sorted array in descending order:", sorted_arr_desc)


```

```
Original array: [3 1 4 1 5 9 2 6]
Sorted array: [1 1 2 3 4 5 6 9]
Original 2D array:
[[3 1 4]
 [1 5 9]
 [2 6 5]]

Sorted 2D array along axis 1:
[[1 3 4]
 [1 5 9]
```

```
 [2 5 6]]
Original array: [3 1 4 1 5 9 2 6]
Sorted array in descending order: [9 6 5 4 3 2 1 1]
```

25. Show how to concatenate two NumPy arrays.

```
import numpy as np

# Create two NumPy arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Concatenate the arrays along axis 0 (default)
concatenated_arr = np.concatenate((arr1, arr2))

print("Array 1:", arr1)
print("Array 2:", arr2)
print("Concatenated array along axis 0:", concatenated_arr)

```

```
Array 1: [1 2 3]
Array 2: [4 5 6]
Concatenated array along axis 0: [1 2 3 4 5 6]
```

26. Write a code snippet for element-wise addition of two NumPy arrays

```
import numpy as np

# Create two NumPy arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Perform element-wise addition using numpy.add()
result = np.add(arr1, arr2)

print("Array 1:", arr1)
print("Array 2:", arr2)
print("Element-wise addition using numpy.add():", result)

```

```
Array 1: [1 2 3]
Array 2: [4 5 6]
Element-wise addition using numpy.add(): [5 7 9]
```

27. How can you perform element-wise exponentiation of a NumPy array?

```
import numpy as np

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

# Perform element-wise exponentiation using numpy.power()
exponentiated_arr = np.power(arr, 2)  # Raise each element to the power of 2

print("Original array:", arr)
print("Element-wise exponentiation using numpy.power():", exponentiated_arr)

```

```
Original array: [1 2 3 4 5]
Element-wise exponentiation using numpy.power(): [ 1  4  9 16 25]
```

28. Calculate the sum, mean, and median of a NumPy array [1, 2, 3].

```
import numpy as np

# Create a NumPy array
arr = np.array([1, 2, 3])

# Calculate the sum of the array
arr_sum = np.sum(arr)

# Calculate the mean of the array
arr_mean = np.mean(arr)

# Calculate the median of the array
arr_median = np.median(arr)

print("Array:", arr)
print("Sum:", arr_sum)
print("Mean:", arr_mean)
print("Median:", arr_median)

```

```
Array: [1 2 3]
Sum: 6
Mean: 2.0
Median: 2.0
```

29. Demonstrate how matrix multiplication works in NumPy.

```
import numpy as np

# Create two NumPy arrays for matrix multiplication
matrix1 = np.array([[1, 2, 3],
                    [4, 5, 6]])

matrix2 = np.array([[7, 8],
                    [9, 10],
                    [11, 12]])

# Perform matrix multiplication using @ operator
result = matrix1 @ matrix2

print("Matrix 1:")
print(matrix1)
print("\nMatrix 2:")
print(matrix2)
print("\nResult of matrix multiplication using @ operator:")
print(result)

```

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

Matrix 2:
[[ 7  8]
 [ 9 10]
 [11 12]]

Result of matrix multiplication using @ operator:
[[ 58  64]
 [139 154]]
```

30. Demonstrate the transposition of a matrix in NumPy.

```
import numpy as np

# Create a NumPy array representing a matrix
matrix = np.array([[1, 2, 3],
                   [4, 5, 6]])

# Transpose the matrix using numpy.transpose()
transposed_matrix = np.transpose(matrix)

print("Original matrix:")
print(matrix)
print("\nTransposed matrix using numpy.transpose():")
print(transposed_matrix)

```

```
Original matrix:
[[1 2 3]
 [4 5 6]]

Transposed matrix using numpy.transpose():
[[1 4]
 [2 5]
 [3 6]]
```

31. What is the purpose of the np.dot function in NumPy?

```
import numpy as np

# Two 2D arrays (matrices)
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Matrix multiplication
result = np.dot(A, B)

result
```

```
array([[19, 22],
       [43, 50]])
```

32. How can you invert a matrix using NumPy?

```
import numpy as np

# Create a square matrix
matrix = np.array([[1, 2],
                   [3, 4]])

# Calculate the inverse of the matrix
inverse_matrix = np.linalg.inv(matrix)

print("Original matrix:")
print(matrix)
print("\nInverse matrix:")
print(inverse_matrix)

```

```
Original matrix:
[[1 2]
 [3 4]]

Inverse matrix:
[[-2.   1. ]
 [ 1.5 -0.5]]
```

33. Demonstrate how to perform linear algebra operations, like matrix multiplication, using NumPy.

```
import numpy as np

# Create two matrices
matrix1 = np.array([[1, 2, 3],
                    [4, 5, 6]])

matrix2 = np.array([[7, 8],
                    [9, 10],
                    [11, 12]])

# Perform matrix multiplication using numpy.matmul() or np.dot()
result = np.matmul(matrix1, matrix2)  # or np.dot(matrix1, matrix2)

print("Matrix 1:")
print(matrix1)
print("\nMatrix 2:")
print(matrix2)
print("\nResult of matrix multiplication:")
print(result)

```

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

Matrix 2:
[[ 7  8]
 [ 9 10]
 [11 12]]

Result of matrix multiplication:
[[ 58  64]
 [139 154]]
```

34. How do you handle missing values in a NumPy array?

```
import numpy as np

# Replace missing values with 0
arr = np.array([1, 2, np.nan, 4, 5])
arr_no_nan = np.nan_to_num(arr, nan=0)

# Impute missing values with mean
arr_imputed = np.nanmean(arr)

```

35. Explain the use of the np.where function in NumPy.

```
import numpy as np

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

# Find indices where elements are greater than 2
indices = np.where(arr > 2)
print(indices)  # Output: (array([2, 3, 4]),)


```

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

```
import numpy as np

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

# Find indices where elements are greater than 2
indices = np.where(arr > 2)
print(indices)  # Output: (array([2, 3, 4]),)


import numpy as np

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

# Replace values greater than 2 with 0
new_arr = np.where(arr > 2, 0, arr)
print(new_arr)  # Output: [1 2 0 0 0]



import numpy as np

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

# Filter elements greater than 2
filtered_arr = arr[np.where(arr > 2)]
print(filtered_arr)  # Output: [3 4 5]


import numpy as np

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

# Find indices where elements are between 2 and 4
indices = np.where((arr > 2) & (arr < 4))
print(indices)  # Output: (array([2]),)


```

```
(array([2, 3, 4]),)
[1 2 0 0 0]
[3 4 5]
(array([2]),)
```

36. Show how to perform element-wise multiplication of two NumPy arrays.

```
import numpy as np

# Create two NumPy arrays for element-wise multiplication
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Perform element-wise multiplication using the * operator
result = arr1 * arr2

print("Array 1:", arr1)
print("Array 2:", arr2)
print("Element-wise multiplication using * operator:", result)

```

```
Array 1: [1 2 3]
Array 2: [4 5 6]
Element-wise multiplication using * operator: [ 4 10 18]
```

37. How can you calculate both the mean and median of each row in a 2-D NumPy array?

```
import numpy as np

# Create a 2D NumPy array
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6],
                     [7, 8, 9]])

# Calculate the mean of each row
row_means = np.mean(array_2d, axis=1)

# Calculate the median of each row
row_medians = np.median(array_2d, axis=1)

# Print the results
print("Array:")
print(array_2d)
print("\nMean of each row:")
print(row_means)
print("\nMedian of each row:")
print(row_medians)

```

```
Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Mean of each row:
[2. 5. 8.]

Median of each row:
[2. 5. 8.]
```

38. How can you calculate the standard deviation of a NumPy array?

```
import numpy as np

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

# Calculate the standard deviation of the array
std_dev = np.std(arr)

print("Array:", arr)
print("Standard deviation:", std_dev)

```

```
Array: [1 2 3 4 5]
Standard deviation: 1.4142135623730951
```

39. Explain the difference between shallow copy (view) and deep copy (copy) in NumPy.

Shallow Copy (View):
A shallow copy, often referred to as a view, creates a new array object that looks at the same data as the original array. Changes made to th
Shallow copy is achieved by slicing, using the view() method, or indexing with a Boolean or integer array.
It's a memory-efficient operation because it doesn't duplicate the underlying data.
Example:
python
Copy code
import numpy as np

# Original array
original_arr = np.array([1, 2, 3, 4, 5])

# Create a shallow copy (view)
view_arr = original_arr.view()

# Modify the view
view_arr[0] = 100

```
print("Original array:", original_arr)
print("View array:", view_arr)
Deep Copy (Copy):
A deep copy creates a new array object with its own unique data. Changes made to the deep copy do not affect the original array, and vice ver
Deep copy is achieved using the copy() method.
It creates a complete, independent copy of the original array, including its data and shape.
Example:
python
Copy code
import numpy as np

# Original array
original_arr = np.array([1, 2, 3, 4, 5])

# Create a deep copy
copy_arr = original_arr.copy()

# Modify the copy
copy_arr[0] = 100

print("Original array:", original_arr)
print("Copy array:", copy_arr)
```

40. Demonstrate how to use slicing to extract a sub-array from a NumPy array.

```
import numpy as np

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

# Extract a sub-array using slicing
sub_arr = arr[0:2, 1:3]  # Rows: 0 to 1, Columns: 1 to 2

print("Original array:")
print(arr)
print("\nSub-array:")
print(sub_arr)

```

```
Original array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Sub-array:
[[2 3]
 [5 6]]
```

41. How do you find the index of a specific value in a NumPy array?

```
import numpy as np

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

# Find the index of a specific value, for example, 3
index = np.where(arr == 3)

print("Index of value 3:", index)

```

```
Index of value 3: (array([2]),)
```

42. Show how to slice elements from the start to a specific index in a NumPy array.

```
import numpy as np

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

# Slice elements from the start to a specific index (exclusive)
specific_index = 5
sliced_array = arr[:specific_index]

print("Sliced array:", sliced_array)

```

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

43. How to select every nth element in a NumPy array.

```
import numpy as np

# Create a NumPy array
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Select every nth element
n = 3
selected_elements = arr[::n]

print("Selected elements:", selected_elements)

```

```
Selected elements: [ 1  4  7 10]
```

44. Demonstrate how to extract a specific column from a 2-D NumPy array.

```
import numpy as np

# Create a 2-D NumPy array
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6],
                     [7, 8, 9]])

# Extract the second column (index 1)
specific_column = array_2d[:, 1]

print("Specific column:", specific_column)

```

```
Specific column: [2 5 8]
```

45. How can you reverse the rows of a 2-D NumPy array?

```
import numpy as np

# Create a 2-D NumPy array
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6],
                     [7, 8, 9]])

# Reverse the rows
reversed_array = array_2d[::-1]

print("Reversed array:")
print(reversed_array)

```

```
Reversed array:
[[7 8 9]
 [4 5 6]
 [1 2 3]]
```

46. Show how to extract a diagonal from a 2-D NumPy array.

```
import numpy as np

# Create a 2-D NumPy array
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6],
                     [7, 8, 9]])

# Extract the diagonal
diagonal_elements = np.diagonal(array_2d)

print("Diagonal elements:", diagonal_elements)

```

```
Diagonal elements: [1 5 9]
```

47. Explain slicing with boolean arrays in NumPy

```
import numpy as np

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

# Create a boolean array based on a condition
condition = arr > 2  # Elements greater than 2

# Apply the boolean array to slice the original array
result = arr[condition]

print("Original array:", arr)
print("Boolean array:", condition)
print("Result:", result)

```

```
Original array: [1 2 3 4 5]
Boolean array: [False False  True  True  True]
Result: [3 4 5]
```

48. How do you select elements based on condition (e.g., all elements greater than a value)?

```
import numpy as np

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

# Select elements greater than 3
selected_elements = arr[arr > 3]

print("Selected elements greater than 3:", selected_elements)

```

```
Selected elements greater than 3: [4 5]
```

49. Demonstrate how to slice a 3-D NumPy array.

```
import numpy as np

# Create a 3-D NumPy array
arr_3d = np.array([[[1, 2, 3],
                    [4, 5, 6]],

                   [[7, 8, 9],
                    [10, 11, 12]]])

# Slicing along each dimension
slice_1 = arr_3d[0, :, :]     # Slicing along the first dimension
slice_2 = arr_3d[:, 1, :]     # Slicing along the second dimension
slice_3 = arr_3d[:, :, 0]     # Slicing along the third dimension

print("Original 3-D array:")
print(arr_3d)
print("\nSlice along the first dimension (arr_3d[0, :, :]):")
print(slice_1)
print("\nSlice along the second dimension (arr_3d[:, 1, :]):")
print(slice_2)
print("\nSlice along the third dimension (arr_3d[:, :, 0]):")
print(slice_3)

```

```
Original 3-D array:
[[[ 1  2  3]
  [ 4  5  6]]

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

Slice along the first dimension (arr_3d[0, :, :]):
[[1 2 3]
 [4 5 6]]

Slice along the second dimension (arr_3d[:, 1, :]):
[[ 4  5  6]
 [10 11 12]]

Slice along the third dimension (arr_3d[:, :, 0]):
[[ 1  4]
 [ 7 10]]
```

50. Show how to extract specific rows from a 2-D array based on a condition.

```
import numpy as np

# Create a 2-D NumPy array
arr_2d = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Define a condition (e.g., sum of elements greater than 10)
condition = np.sum(arr_2d, axis=1) > 10

# Use boolean indexing to extract rows based on the condition
selected_rows = arr_2d[condition]

print("Original 2-D array:")
print(arr_2d)
print("\nSelected rows based on the condition (sum of elements > 10):")
print(selected_rows)

```

```
Original 2-D array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Selected rows based on the condition (sum of elements > 10):
[[4 5 6]
 [7 8 9]]
```

51. Explain the common mistakes to avoid when broadcasting arrays in NumPy

1. Understand broadcasting rules thoroughly.
2. Ensure arrays have compatible shapes for broadcasting.
3. Reshape arrays when necessary for compatibility.
4. Handle single-dimensional arrays correctly in broadcasting.
5. Differentiate broadcasting from element-wise operations.
6. Use broadcasting judiciously to maintain code clarity.
7. Test edge cases rigorously to validate broadcasting behavior.

52. Demonstrate the use of logical operations (like and, or, not) on NumPy arrays.

```
import numpy as np

# Create two NumPy arrays
arr1 = np.array([True, True, False, False])
arr2 = np.array([True, False, True, False])

# Bitwise logical operations
result_and = arr1 & arr2  # Logical AND
result_or = arr1 | arr2   # Logical OR
result_not = ~arr1        # Logical NOT (inverts the values)

print("Bitwise logical operations:")
print("Logical AND:", result_and)
print("Logical OR:", result_or)
print("Logical NOT:", result_not)

# Using NumPy logical functions
result_logical_and = np.logical_and(arr1, arr2)  # Logical AND
result_logical_or = np.logical_or(arr1, arr2)    # Logical OR
result_logical_not = np.logical_not(arr1)        # Logical NOT

print("\nUsing NumPy logical functions:")
print("Logical AND:", result_logical_and)
print("Logical OR:", result_logical_or)
print("Logical NOT:", result_logical_not)

```

```
Bitwise logical operations:
Logical AND: [ True False False False]
Logical OR: [ True  True  True False]
Logical NOT: [False False  True  True]

Using NumPy logical functions:
Logical AND: [ True False False False]
Logical OR: [ True  True  True False]
Logical NOT: [False False  True  True]
```

53. How do you perform element-wise division between two NumPy arrays?

```
import numpy as np

# Create two NumPy arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Perform element-wise division using the / operator
result = arr1 / arr2

print("Result of element-wise division using the / operator:", result)

```

```
Result of element-wise division using the / operator: [0.25 0.4  0.5 ]
```

```
import numpy as np

# Create two NumPy arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Perform element-wise division using the np.divide() function
result = np.divide(arr1, arr2)

print("Result of element-wise division using np.divide() function:", result)

```

```
Result of element-wise division using np.divide() function: [0.25 0.4  0.5 ]
```

54. Explain the functionality of the np.unique function

```
import numpy as np

# Create an array with duplicate elements
arr = np.array([1, 2, 2, 3, 3, 3, 4, 4, 5])

# Find unique elements
unique_elements = np.unique(arr)

print("Unique elements:", unique_elements)

```

```
Unique elements: [1 2 3 4 5]
```

55. Show how to compute the correlation coefficient of two NumPy arrays.

```
import numpy as np

# Create two NumPy arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([5, 4, 3, 2, 1])

# Compute the correlation coefficient
correlation_coefficient = np.corrcoef(arr1, arr2)[0, 1]

print("Correlation coefficient between arr1 and arr2:", correlation_coefficient)

```

```
Correlation coefficient between arr1 and arr2: -0.9999999999999999
```

56. How do you iterate over each element in a NumPy array?

```
import numpy as np

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

# Iterate over each element in the array
for row in arr:
    for element in row:
        print(element)

```

```
1
2
3
4
5
6
7
8
9
```

57. Array Linear Spacing: Generate an array of 20 elements linearly spaced between 5 and 15.

```
import numpy as np

# Generate an array of 20 elements linearly spaced between 5 and 15
linear_array = np.linspace(5, 15, 20)

print("Linearly spaced array:", linear_array)

```

```
Linearly spaced array: [ 5.          5.52631579  6.05263158  6.57894737  7.10526316  7.63157895
  8.15789474  8.68421053  9.21052632  9.73684211 10.26315789 10.78947368
 11.31578947 11.84210526 12.36842105 12.89473684 13.42105263 13.94736842
 14.47368421 15.        ]
```

58. Manhattan Distance: Write a function to compute the Manhattan distance between two 1D arrays, array1 and array2, of equal length.

```
import numpy as np

def manhattan_distance(array1, array2):
    """
    Compute the Manhattan distance between two 1D arrays.

    Parameters:
    array1 (numpy.ndarray): First 1D array.
    array2 (numpy.ndarray): Second 1D array.

    Returns:
    float: Manhattan distance between the two arrays.
    """
    # Check if arrays have equal length
    if len(array1) != len(array2):
        raise ValueError("Arrays must have equal length")

    # Compute Manhattan distance
    distance = np.sum(np.abs(array1 - array2))

    return distance

# Example usage
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([5, 4, 3, 2, 1])
distance = manhattan_distance(array1, array2)
print("Manhattan distance:", distance)

```

```
Manhattan distance: 12
```

59. Flatten Array: Flatten a given 3x3 2D array into a 1D array.

```
import numpy as np

# Create a 3x3 2D array
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6],
                     [7, 8, 9]])

# Flatten the 2D array into a 1D array using np.flatten()
flattened_array1 = array_2d.flatten()

# Flatten the 2D array into a 1D array using np.ravel()
flattened_array2 = np.ravel(array_2d)

print("Original 2D array:")
print(array_2d)

print("\nFlattened array using np.flatten():")
print(flattened_array1)

print("\nFlattened array using np.ravel():")
print(flattened_array2)

```

```
Original 2D array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Flattened array using np.flatten():
[1 2 3 4 5 6 7 8 9]

Flattened array using np.ravel():
[1 2 3 4 5 6 7 8 9]
```

60. NaN Check: Write a script to create an array and then check if it contains any NaN values.

```
import numpy as np

# Create an array with some NaN values
arr = np.array([1, 2, np.nan, 4, np.nan])

# Check if the array contains any NaN values
if np.isnan(arr).any():
    print("Array contains NaN values.")
else:
    print("Array does not contain any NaN values.")

```

```
Array contains NaN values.
```

61. Array Rank Determination: Determine the rank of a given 4x4 matrix.

```
import numpy as np

# Define your 4x4 matrix
matrix = np.array([
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
    [13, 14, 15, 16]
])

# Determine the rank of the matrix
rank = np.linalg.matrix_rank(matrix)

print("Rank of the matrix:", rank)

```

```
Rank of the matrix: 2
```

62. Descending Sort: Sort the elements of a given 1D array in descending order.

```
import numpy as np

# Define your 1D array
arr = np.array([3, 1, 4, 1, 5, 9, 2, 6])

# Sort the array in descending order
arr_sorted = np.sort(arr)[::-1]

print("Original array:", arr)
print("Array sorted in descending order:", arr_sorted)

```

```
Original array: [3 1 4 1 5 9 2 6]
Array sorted in descending order: [9 6 5 4 3 2 1 1]
```

63. 3D Array Slicing: Slice a 3x3x3 array to extract a 2x2x2 sub-array from its corner.

```
1 import numpy as np
2
```