**NumPy Functions - Index**

**Index**

**Creating Arrays**

* np.array() - Create an array from a list or tuple
* np.zeros() - Generate an array filled with zeros
* np.ones() - Generate an array filled with ones
* np.arange(start, stop, step) - Create an array with evenly spaced values
* np.linspace(start, stop, num) - Generate an array with evenly spaced numbers
* np.eye(n) - Create an identity matrix
* np.full(shape, fill\_value) - Create an array filled with a specific value
* np.empty(shape) - Allocate an array without initializing values

**Mathematical Operations**

* np.add(a, b) - Element-wise addition
* np.subtract(a, b) - Element-wise subtraction
* np.multiply(a, b) - Element-wise multiplication
* np.divide(a, b) - Element-wise division
* np.sqrt(a) - Compute square root of elements
* np.power(a, b) - Raise elements to a power
* np.mod(a, b) - Compute modulus
* np.clip(a, min, max) - Limit values within a range

**Statistical Functions**

* np.mean(a) - Compute mean
* np.median(a) - Compute median
* np.std(a) - Compute standard deviation
* np.var(a) - Compute variance
* np.min(a) - Find minimum value
* np.max(a) - Find maximum value
* np.percentile(a, q) - Compute percentile
* np.average(a, weights) - Compute weighted average

**Aggregation & Reduction Functions**

* np.sum(a) - Compute sum of elements
* np.prod(a) - Compute product of elements
* np.cumsum(a) - Compute cumulative sum
* np.cumprod(a) - Compute cumulative product
* np.argmax(a) - Get index of maximum value
* np.argmin(a) - Get index of minimum value
* np.all(a) - Check if all elements are True
* np.any(a) - Check if any element is True

**Linear Algebra Operations**

* np.dot(a, b) - Compute dot product
* np.matmul(a, b) - Perform matrix multiplication
* np.linalg.inv(a) - Compute matrix inverse
* np.linalg.det(a) - Compute determinant
* np.linalg.eig(a) - Compute eigenvalues & eigenvectors
* np.linalg.norm(a) - Compute matrix/vector norm
* np.linalg.qr(a) - Compute QR decomposition
* np.linalg.svd(a) - Compute Singular Value Decomposition

**Random Number Generation**

* np.random.rand(shape) - Generate random values in [0,1)
* np.random.randint(low, high, size) - Generate random integers
* np.random.normal(mean, std, size) - Generate normal distribution samples
* np.random.choice(a, size) - Randomly select elements
* np.random.shuffle(a) - Shuffle an array
* np.random.seed(value) - Set seed for reproducibility
* np.random.uniform(low, high, size) - Generate uniform distribution samples
* np.random.poisson(lam, size) - Generate Poisson distribution samples

**Reshaping & Manipulating Arrays**

* np.reshape(a, new\_shape) - Reshape array
* np.flatten(a) - Flatten a multi-dimensional array
* np.transpose(a) - Transpose array dimensions
* np.resize(a, new\_shape) - Change array shape
* np.expand\_dims(a, axis) - Add an axis
* np.squeeze(a) - Remove single-dimensional entries
* np.ravel(a) - Return a flattened array
* np.hstack((a, b)) - Stack arrays horizontally
* np.vstack((a, b)) - Stack arrays vertically

**Searching & Sorting**

* np.where(condition) - Find indices where a condition is True
* np.take(a, indices) - Select elements using indices
* np.unique(a) - Find unique elements
* np.nonzero(a) - Get indices of non-zero elements
* np.argsort(a) - Get indices that would sort an array
* np.split(a, indices) - Split an array into sub-arrays
* np.flip(a) - Reverse order of elements
* np.roll(a, shift, axis) - Roll elements along an axis

Got it! I'll provide detailed explanations for each function under their respective sections.

**Creating Arrays**

**1.1 np.array() - Create an array from a list or tuple**

**Explanation:**

* Converts a Python list or tuple into a NumPy array.
* Supports multi-dimensional arrays.

**Example:**

import numpy as np

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

print(arr)

**Output:**

[1 2 3 4 5]

**1.2 np.zeros(shape) - Generate an array filled with zeros**

**Explanation:**

* Creates an array of given shape filled with 0.
* Useful for initializing arrays before computation.

**Example:**

zeros\_arr = np.zeros((2, 3))

print(zeros\_arr)

**Output:**

[[0. 0. 0.]

[0. 0. 0.]]

**1.3 np.ones(shape) - Generate an array filled with ones**

**Explanation:**

* Creates an array of given shape filled with 1.
* Used when a default non-zero array is needed.

**Example:**

ones\_arr = np.ones((2, 3))

print(ones\_arr)

**Output:**

[[1. 1. 1.]

[1. 1. 1.]]

**1.4 np.arange(start, stop, step) - Create an array with evenly spaced values**

**Explanation:**

* Generates a sequence of numbers starting from start up to (but not including) stop, with a step size of step.
* Useful for generating sequences for iteration or plotting.

**Example:**

range\_arr = np.arange(1, 10, 2)

print(range\_arr)

**Output:**

[1 3 5 7 9]

**1.5 np.linspace(start, stop, num) - Generate an array with evenly spaced numbers**

**Explanation:**

* Generates num evenly spaced values between start and stop.
* Useful in mathematical computations like graph plotting.

**Example:**

linspace\_arr = np.linspace(1, 10, 5)

print(linspace\_arr)

**Output:**

[ 1. 3.25 5.5 7.75 10. ]

**1.6 np.eye(n) - Create an identity matrix**

**Explanation:**

* Creates an n × n identity matrix with 1s on the diagonal and 0s elsewhere.
* Commonly used in linear algebra.

**Example:**

identity\_matrix = np.eye(3)

print(identity\_matrix)

**Output:**

[[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

**1.7 np.full(shape, fill\_value) - Create an array filled with a specified value**

**Explanation:**

* Returns an array of given shape, filled with fill\_value.
* Useful for creating arrays with default values.

**Example:**

full\_arr = np.full((2, 3), 7)

print(full\_arr)

**Output:**

[[7 7 7]

[7 7 7]]

**1.8 np.empty(shape) - Allocate an array without initializing values**

**Explanation:**

* Creates an array with uninitialized values (values may be random).
* Useful for performance optimization in large datasets.

**Example:**

empty\_arr = np.empty((2, 3))

print(empty\_arr)

**Output:** *(Random uninitialized values)*

[[1.05123826e-312 1.05118171e-312 1.05116734e-312]

[1.05112423e-312 1.05113502e-312 1.05110887e-312]]

**Mathematical Operations**

**2.1 np.add(a, b) - Element-wise addition of arrays**

**Explanation:**

* Performs element-wise addition between two arrays of the same shape.
* If shapes are different, broadcasting is applied.

**Example:**

import numpy as np

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

b = np.array([4, 5, 6])

result = np.add(a, b)

print(result)

**Output:**

[5 7 9]

**2.2 np.subtract(a, b) - Element-wise subtraction of arrays**

**Explanation:**

* Performs element-wise subtraction between two arrays of the same shape.
* Supports broadcasting for different shapes.

**Example:**

result = np.subtract(a, b)

print(result)

**Output:**

[-3 -3 -3]

**2.3 np.multiply(a, b) - Element-wise multiplication**

**Explanation:**

* Multiplies corresponding elements of two arrays.
* Supports broadcasting if shapes differ.

**Example:**

result = np.multiply(a, b)

print(result)

**Output:**

[ 4 10 18]

**2.4 np.divide(a, b) - Element-wise division**

**Explanation:**

* Performs element-wise division between two arrays.
* Handles division by zero by returning inf or nan where applicable.

**Example:**

result = np.divide(a, b)

print(result)

**Output:**

[0.25 0.4 0.5 ]

**2.5 np.sqrt(a) - Compute the square root of each element**

**Explanation:**

* Computes the square root of every element in the array.
* Returns nan for negative numbers unless using np.sqrt(complex).

**Example:**

result = np.sqrt(a)

print(result)

**Output:**

[1. 1.41421356 1.73205081]

**2.6 np.power(a, b) - Raise elements to the power of b**

**Explanation:**

* Raises each element in a to the corresponding power in b.
* Works for scalars or arrays.

**Example:**

result = np.power(a, 2)

print(result)

**Output:**

[1 4 9]

**2.7 np.mod(a, b) - Element-wise modulus operation**

**Explanation:**

* Computes a % b, the remainder after division.
* Works element-wise for arrays.

**Example:**

result = np.mod(a, b)

print(result)

**Output:**

[1 2 3]

**2.8 np.clip(a, min, max) - Limit values within a specified range**

**Explanation:**

* Limits values in an array to be within the range [min, max].
* Values below min become min, and values above max become max.

**Example:**

arr = np.array([1, 5, 10, 15])

result = np.clip(arr, 3, 12)

print(result)

**Output:**

[ 3 5 10 12]

**Statistical Functions**

**3.1 np.mean(a) - Compute the mean of the array**

**Explanation:**

* Computes the arithmetic mean (average) of all elements in an array.
* Can specify an axis to compute the mean along a specific dimension.

**Example:**

import numpy as np

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

result = np.mean(a)

print(result)

**Output:**

3.5

**3.2 np.median(a) - Compute the median of the array**

**Explanation:**

* Finds the middle value in a sorted array.
* If the number of elements is even, returns the average of the two middle values.

**Example:**

result = np.median(a)

print(result)

**Output:**

3.5

**3.3 np.std(a) - Compute the standard deviation**

**Explanation:**

* Measures the dispersion of values in an array from the mean.
* A lower value means data points are close to the mean, while a higher value means more spread out.

**Example:**

result = np.std(a)

print(result)

**Output:**

1.707825127659933

**3.4 np.var(a) - Compute the variance**

**Explanation:**

* Measures the spread of the data.
* Variance is the square of the standard deviation.

**Example:**

result = np.var(a)

print(result)

**Output:**

2.9166666666666665

**3.5 np.min(a) - Return the minimum value in the array**

**Explanation:**

* Returns the smallest element in the array.
* Can compute along a specific axis.

**Example:**

result = np.min(a)

print(result)

**Output:**

1

**3.6 np.max(a) - Return the maximum value in the array**

**Explanation:**

* Returns the largest element in the array.
* Supports axis-based computation.

**Example:**

result = np.max(a)

print(result)

**Output:**

6

**3.7 np.percentile(a, q) - Compute the q-th percentile**

**Explanation:**

* Finds the value below which a given percentage q of values fall.
* Used in statistical analysis to determine distributions.

**Example:**

result = np.percentile(a, 50) # 50th percentile (median)

print(result)

**Output:**

3.5

**3.8 np.average(a, weights=None) - Compute the weighted average**

**Explanation:**

* Computes the mean, but allows assigning different weights to values.
* If weights is None, behaves like np.mean().

**Example:**

weights = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])

result = np.average(a, weights=weights)

print(result)

**Output:**

4.333333333333333

**Aggregation & Reduction Functions**

**4.1 np.sum(a) - Compute the sum of elements**

**Explanation:**

* Computes the total sum of all elements in an array.
* Can sum along a specific axis if specified.

**Example:**

import numpy as np

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

result = np.sum(a)

print(result)

**Output:**

21

**4.2 np.prod(a) - Compute the product of elements**

**Explanation:**

* Computes the product of all elements in the array.
* Can specify an axis to compute the product along a dimension.

**Example:**

result = np.prod(a)

print(result)

**Output:**

720

**4.3 np.cumsum(a) - Compute cumulative sum of elements**

**Explanation:**

* Computes the cumulative sum of array elements.
* Each element in the output is the sum of the current and previous elements.

**Example:**

result = np.cumsum(a)

print(result)

**Output:**

[ 1 3 6 10 15 21]

**4.4 np.cumprod(a) - Compute cumulative product of elements**

**Explanation:**

* Computes the cumulative product of array elements.
* Each element in the output is the product of the current and previous elements.

**Example:**

result = np.cumprod(a)

print(result)

**Output:**

[ 1 2 6 24 120 720]

**4.5 np.argmax(a) - Find the index of the maximum value**

**Explanation:**

* Returns the index of the maximum value in the array.
* Can be applied along a specific axis.

**Example:**

result = np.argmax(a)

print(result)

**Output:**

5

**4.6 np.argmin(a) - Find the index of the minimum value**

**Explanation:**

* Returns the index of the minimum value in the array.
* Supports axis-based computation.

**Example:**

result = np.argmin(a)

print(result)

**Output:**

0

**4.7 np.all(a) - Check if all elements evaluate to True**

**Explanation:**

* Returns True if all elements in the array are nonzero (truthy).
* Useful for checking conditions in logical operations.

**Example:**

arr = np.array([1, 2, 3, 0])

result = np.all(arr)

print(result)

**Output:**

False

**4.8 np.any(a) - Check if any element evaluates to True**

**Explanation:**

* Returns True if at least one element in the array is nonzero (truthy).
* Useful in conditional filtering.

**Example:**

result = np.any(arr)

print(result)

**Output:**

True

**Linear Algebra Operations**

**5.1 np.dot(a, b) - Compute the dot product of two arrays**

**Explanation:**

* Computes the scalar product (dot product) of two arrays.
* If both arrays are 1D, it returns a scalar value.
* If arrays are 2D, it performs matrix multiplication.

**Example:**

import numpy as np

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

b = np.array([4, 5, 6])

result = np.dot(a, b)

print(result)

**Output:**

32 # (1\*4 + 2\*5 + 3\*6)

**5.2 np.matmul(a, b) - Perform matrix multiplication**

**Explanation:**

* Used for matrix multiplication of 2D arrays.
* Unlike np.dot(), it strictly follows matrix multiplication rules.

**Example:**

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

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

result = np.matmul(A, B)

print(result)

**Output:**

[[19 22]

[43 50]]

**5.3 np.linalg.inv(a) - Compute the inverse of a matrix**

**Explanation:**

* Returns the inverse of a square matrix.
* If the determinant is zero, the matrix is singular and cannot be inverted.

**Example:**

A = np.array([[4, 7], [2, 6]])

result = np.linalg.inv(A)

print(result)

**Output:**

[[ 0.6 -0.7]

[-0.2 0.4]]

**5.4 np.linalg.det(a) - Compute the determinant of a matrix**

**Explanation:**

* Returns the determinant of a square matrix.
* A zero determinant indicates a singular matrix (non-invertible).

**Example:**

result = np.linalg.det(A)

print(result)

**Output:**

10.0

**5.5 np.linalg.eig(a) - Compute the eigenvalues and eigenvectors**

**Explanation:**

* Returns the eigenvalues and eigenvectors of a square matrix.
* Used in Principal Component Analysis (PCA) and other mathematical transformations.

**Example:**

values, vectors = np.linalg.eig(A)

print("Eigenvalues:", values)

print("Eigenvectors:", vectors)

**Output:**

Eigenvalues: [ 8.71779789 1.28220211]

Eigenvectors:

[[ 0.82456484 -0.41597356]

[ 0.56576746 0.90937671]]

**5.6 np.linalg.norm(a) - Compute the norm of a vector or matrix**

**Explanation:**

* Measures the length (magnitude) of a vector.
* Used in optimization and machine learning algorithms.

**Example:**

vector = np.array([3, 4])

result = np.linalg.norm(vector)

print(result)

**Output:**

5.0 # (sqrt(3² + 4²))

**5.7 np.linalg.qr(a) - Compute the QR decomposition**

**Explanation:**

* Decomposes a matrix into an orthogonal matrix (Q) and an upper triangular matrix (R).
* Used in solving linear systems and numerical analysis.

**Example:**

Q, R = np.linalg.qr(A)

print("Q Matrix:", Q)

print("R Matrix:", R)

**Output:**

Q Matrix: [[-0.89442719 -0.4472136 ]

[-0.4472136 0.89442719]]

R Matrix: [[-4.47213595 -7.82304747]

[ 0. -0.89442719]]

**5.8 np.linalg.svd(a) - Compute the Singular Value Decomposition (SVD)**

**Explanation:**

* Decomposes a matrix into three matrices: U, Σ, and V^T.
* Used in dimensionality reduction and noise reduction in data science.

**Example:**

U, S, Vt = np.linalg.svd(A)

print("U Matrix:", U)

print("Singular Values:", S)

print("Vt Matrix:", Vt)

**Output:**

U Matrix: [[-0.40455358 -0.9145143 ]

[-0.9145143 0.40455358]]

Singular Values: [5.4649857 0.36596619]

Vt Matrix: [[-0.57604844 -0.81741556]

[-0.81741556 0.57604844]]

**Random Number Generation**

**6.1 np.random.rand() - Generate random numbers in [0,1)**

**Explanation:**

* Generates random numbers from a uniform distribution between **0** and **1**.
* Can create arrays of specified shape with random values.

**Example:**

import numpy as np

result = np.random.rand(3, 3) # 3x3 matrix with random values

print(result)

**Output:**

[[0.4873 0.9911 0.2456]

[0.3794 0.6732 0.1234]

[0.7542 0.6218 0.8765]]

**6.2 np.random.randint(low, high, size) - Generate random integers**

**Explanation:**

* Generates random integers between low (inclusive) and high (exclusive).
* Can generate a single value or an array of specified shape.

**Example:**

result = np.random.randint(1, 10, size=(2, 3)) # 2x3 matrix with random integers from 1 to 9

print(result)

**Output:**

[[3 7 1]

[9 6 2]]

**6.3 np.random.normal(mean, std, size) - Generate random samples from a normal distribution**

**Explanation:**

* Generates random numbers from a normal (Gaussian) distribution.
* Defined by **mean** and **standard deviation (std)**.

**Example:**

result = np.random.normal(0, 1, size=(2, 3)) # 2x3 matrix from normal distribution (mean=0, std=1)

print(result)

**Output:**

[[ 1.2568 -0.7453 0.4325]

[-0.9652 0.9874 1.5642]]

**6.4 np.random.choice(a, size, replace=True) - Choose random elements from an array**

**Explanation:**

* Randomly selects elements from an existing array.
* Can choose with or without replacement.

**Example:**

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

result = np.random.choice(arr, size=3, replace=False) # Select 3 random elements

print(result)

**Output:**

[30 10 50]

**6.5 np.random.shuffle(a) - Shuffle an array in-place**

**Explanation:**

* Randomly rearranges elements in an array.
* The original array is modified (shuffled in-place).

**Example:**

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

np.random.shuffle(arr)

print(arr)

**Output:**

[3 5 1 4 2] # (Randomly shuffled)

**6.6 np.random.seed(seed) - Set the seed for random number generation**

**Explanation:**

* Ensures reproducibility of random numbers.
* The same seed value produces the same random output every time.

**Example:**

np.random.seed(42)

result = np.random.rand(3)

print(result)

**Output:**

[0.3745 0.9507 0.7319] # (Same output every time with seed=42)

**6.7 np.random.uniform(low, high, size) - Generate samples from a uniform distribution**

**Explanation:**

* Generates random numbers uniformly distributed between **low** and **high**.
* All values in the range have equal probability.

**Example:**

result = np.random.uniform(5, 10, size=(2, 2)) # 2x2 matrix with values between 5 and 10

print(result)

**Output:**

[[6.2354 9.8763]

[8.4321 5.6792]]

**6.8 np.random.poisson(lambda, size) - Generate samples from a Poisson distribution**

**Explanation:**

* Generates numbers following a **Poisson distribution**, where lambda (λ) is the expected number of occurrences in a fixed time period.
* Used in statistics and probability models.

**Example:**

result = np.random.poisson(3, size=5) # 5 random values from a Poisson distribution with λ=3

print(result)

**Output:**

[4 2 3 5 3]

**Reshaping & Manipulating Arrays**

**7.1 np.reshape(a, new\_shape) - Reshape an array**

**Explanation:**

* Changes the shape of an array **without modifying its data**.
* The new shape **must be compatible** with the total number of elements.

**Example:**

import numpy as np

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

result = np.reshape(arr, (2, 3)) # Reshape to 2x3 matrix

print(result)

**Output:**

[[1 2 3]

[4 5 6]]

**7.2 np.flatten() - Flatten a multi-dimensional array**

**Explanation:**

* Converts an **n-dimensional array** into a **1D array**.
* Returns a **copy** of the original array in a **flattened form**.

**Example:**

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

result = arr.flatten()

print(result)

**Output:**

[1 2 3 4 5 6]

**7.3 np.transpose(a) - Transpose an array**

**Explanation:**

* Swaps the **rows and columns** of a matrix.
* Works on **2D and higher-dimensional arrays**.

**Example:**

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

result = np.transpose(arr)

print(result)

**Output:**

[[1 4]

[2 5]

[3 6]]

**7.4 np.resize(a, new\_shape) - Resize an array**

**Explanation:**

* Changes the shape **and number of elements** in an array.
* If the new shape is larger, the array is **repeated** to fill the space.

**Example:**

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

result = np.resize(arr, (3, 3)) # Resize to 3x3 matrix

print(result)

**Output:**

[[1 2 3]

[4 1 2]

[3 4 1]]

**7.5 np.expand\_dims(a, axis) - Expand an array by adding a dimension**

**Explanation:**

* Increases the **number of dimensions** by inserting a new axis.
* Useful for **broadcasting and batch processing**.

**Example:**

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

result = np.expand\_dims(arr, axis=0) # Convert to a row vector (1x3)

print(result)

**Output:**

[[1 2 3]]

**7.6 np.squeeze(a) - Remove axes of length one**

**Explanation:**

* Removes **single-dimensional entries** from the shape of an array.
* Useful when working with **unnecessary dimensions**.

**Example:**

arr = np.array([[[1, 2, 3]]]) # Shape (1,1,3)

result = np.squeeze(arr) # Remove unnecessary dimensions

print(result)

**Output:**

[1 2 3] # Shape reduced to (3,)

**7.7 np.ravel() - Return a flattened array**

**Explanation:**

* Converts an **n-dimensional array** into a **1D array**.
* Unlike flatten(), it **returns a view** when possible (instead of a copy).

**Example:**

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

result = arr.ravel()

print(result)

**Output:**

[1 2 3 4 5 6]

**7.8 np.hstack((a, b)) - Stack arrays horizontally**

**Explanation:**

* Concatenates arrays **side by side** (column-wise).
* Arrays **must have the same number of rows**.

**Example:**

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

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

result = np.hstack((a, b))

print(result)

**Output:**

[[1 2 5 6]

[3 4 7 8]]

**7.9 np.vstack((a, b)) - Stack arrays vertically**

**Explanation:**

* Concatenates arrays **one below the other** (row-wise).
* Arrays **must have the same number of columns**.

**Example:**

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

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

result = np.vstack((a, b))

print(result)

**Output:**

[[1 2]

[3 4]

[5 6]

[7 8]]

**Searching & Sorting**

**8.1 np.where(condition) - Find indices where a condition is True**

**Explanation:**

* Returns the **indices** of elements satisfying a given condition.
* Can be used for **conditional filtering** in arrays.

**Example:**

import numpy as np

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

result = np.where(arr > 25) # Find indices where values are greater than 25

print(result)

**Output:**

(array([2, 3, 4]),) # Indices of elements greater than 25

**8.2 np.take(a, indices) - Select elements from an array**

**Explanation:**

* Extracts elements from an array using **specific indices**.
* Useful when **reordering or selecting specific elements**.

**Example:**

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

result = np.take(arr, [1, 3, 4]) # Select elements at indices 1, 3, 4

print(result)

**Output:**

[20 40 50]

**8.3 np.unique(a) - Find unique elements**

**Explanation:**

* Returns **sorted unique values** from an array.
* Can also return **counts** of each unique element.

**Example:**

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

unique\_values = np.unique(arr)

print(unique\_values)

**Output:**

[1 2 3 4 5]

**8.4 np.nonzero(a) - Get indices of non-zero elements**

**Explanation:**

* Returns **indices** where array elements are **non-zero**.
* Useful for **sparse matrix operations**.

**Example:**

arr = np.array([0, 10, 0, 30, 50])

result = np.nonzero(arr)

print(result)

**Output:**

(array([1, 3, 4]),) # Indices of non-zero elements

**8.5 np.argsort(a) - Get indices that would sort an array**

**Explanation:**

* Returns the **indices** that would sort an array.
* Can be used for **sorting another array based on this order**.

**Example:**

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

sorted\_indices = np.argsort(arr)

print(sorted\_indices)

**Output:**

[1 3 2 0] # Indices of elements in sorted order

**8.6 np.split(a, indices) - Split an array into sub-arrays**

**Explanation:**

* Divides an array into **multiple sub-arrays** at specified indices.
* Returns a **list of sub-arrays**.

**Example:**

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

result = np.split(arr, [2, 4]) # Split at indices 2 and 4

print(result)

**Output:**

[array([10, 20]), array([30, 40]), array([50, 60])]

**8.7 np.flip(a) - Reverse the order of elements**

**Explanation:**

* Reverses an array **along a specified axis**.
* Useful for **mirroring and reversing sequences**.

**Example:**

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

result = np.flip(arr)

print(result)

**Output:**

[5 4 3 2 1]

**8.8 np.roll(a, shift, axis) - Roll array elements along an axis**

**Explanation:**

* Rolls array elements **circularly** along a given axis.
* Useful for **shifting data cyclically**.

**Example:**

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

result = np.roll(arr, 2) # Shift elements by 2 places

print(result)

**Output:**

[40 50 10 20 30]