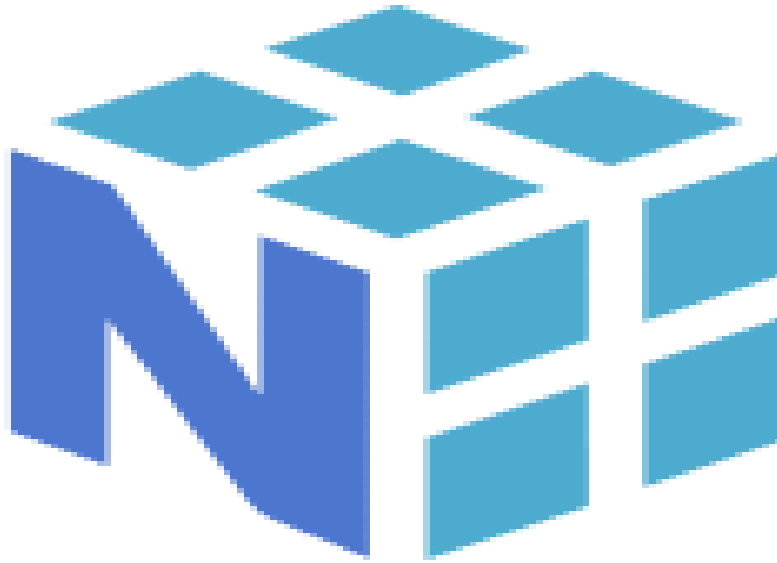


NumPy: The Foundation of Data Science in Python



Introduction to NumPy

NumPy (Numerical Python) is the cornerstone of scientific computing in Python. It provides a powerful N-dimensional array object, tools for integrating C/C++ and Fortran code, and useful linear algebra, Fourier transform, and random number capabilities. Whether you're working with data manipulation, machine learning, or scientific research, NumPy is an essential library to master.

One of its standout features is broadcasting, which simplifies operations on arrays of different shapes and sizes. Additionally, NumPy integrates seamlessly with other popular libraries like Pandas, Scikit-learn, and TensorFlow, enhancing its versatility in the data science ecosystem. With its C-based implementation, NumPy ensures faster execution compared to native Python lists, making it not only efficient but also convenient for handling complex numerical tasks.

Key Features



- **Efficient Array Operations:** NumPy arrays are faster and more memory-efficient than Python lists.
- **Broadcasting:** Perform operations on arrays of different shapes.
- **Mathematical Functions:** Supports a wide range of mathematical operations.
- **Interoperability:** Works seamlessly with other libraries like Pandas, Matplotlib, and TensorFlow.

Installation

- To use NumPy, it must first be installed in your Python environment.
- Install using the **pip** command:
 - **Command:** `pip install numpy`
- Ensure you have an active internet connection while installing.

Importing NumPy Package

- To use NumPy in your program, you need to import it.
- The most common import statement is:

```
python  
  
import numpy as np
```

- **Why use as np?**
 - It shortens the name, making the code more readable and easier to write.

- This is the most commonly accepted alias in the Python community.

ModuleNotFoundError

- If NumPy is not installed, Python will throw this error when you try to import it:

```
vbnet  
  
ModuleNotFoundError: No module named 'numpy'
```

- **Solution:** Install NumPy using `pip install numpy`.

Famous Alias Name for NumPy

- The alias **np** is universally recognized in Python code.
- It's considered a best practice to use np as the alias name:

```
python  
  
np.array([1, 2, 3])
```

NumPy Fundamentals: Key Concepts and Functionalities

I. Creating a NumPy Array

What is a NumPy Array?

- A NumPy array is a multi-dimensional container for storing data. It is similar to a Python list but supports advanced numerical operations.
- Arrays can be 1D, 2D (matrices), or n-dimensional.

Functionalities:

- Create arrays using:
 1. **From a Python list:** Convert a list to a NumPy array.
 2. **Using built-in functions:** Functions like `zeros()`, `ones()`, `arange()`, and `linspace()` can create arrays directly.
- Examples of array types:
 1. **1D array:** A single row of elements.
 2. **2D array:** Rows and columns, similar to a table.
 3. **nD array:** Arrays with more than 2 dimensions, used in advanced computations.



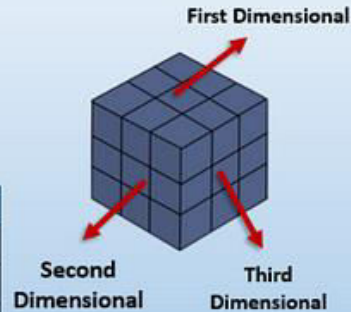
NumPy Narray

10 15 13 8 25

1D-Array

	Column 0	Column 1	Column 2
Row 0	X[0][0]	X[0][1]	X[0][2]
Row 1	X[1][0]	X[1][1]	X[1][2]
Row 2	X[2][0]	X[2][1]	X[2][2]

2D-Array



3D-Array

2. Array Manipulation

- **Reshaping Arrays:**

1. Change the shape of an array without altering its data using the `reshape()` function.
2. Example functionality:
Convert a 1D array to a 2D array.

- **Flattening Arrays:**

Reduce multi-dimensional arrays to a 1D array using the `flatten()` function.

- **Concatenation and Splitting:**

Combine multiple arrays into one or split a single array into parts.

1. **Concatenation:** Merge along an axis (rows or columns).
2. **Splitting:** Divide an array into subarrays.

3. Mathematical Operations

- **Arithmetic Operations:**

1. Perform element-wise operations on arrays such as addition, subtraction, multiplication, and division.

2. Functionalities:

- Add or subtract arrays directly.
- Multiply each element by a scalar (e.g., multiplying the entire array by 2).
-

- **Universal Functions (ufuncs):**

Perform mathematical operations like square root (sqrt), exponentiation (exp), logarithms (log), and trigonometric functions (sin, cos).

- **Broadcasting:**

Enables operations on arrays of different shapes by automatically expanding smaller arrays to match the shape of the larger one.

4. Indexing and Slicing

- **Indexing:**

1. Access specific elements in an array by their index (position).
2. Works like Python lists, but supports multi-dimensional arrays (row and column access).

- **Slicing:**

1. Extract subsets of an array using the slice notation start:stop:step.
2. Functionalities:
 - Access a single row or column.
 - Extract a block of data from a 2D array.
 - Reverse an array using negative indexing.

Key Functionalities Table

Topic	Functionality
Create Array	Use <code>array()</code> , <code>zeros()</code> , <code>ones()</code> , <code>arange()</code> , and <code>linspace()</code>
Reshaping	Use <code>reshape()</code> to change array dimensions
Flattening	Use <code>flatten()</code> to convert a multi-dimensional array into a 1D array
Arithmetic Operations	Perform <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> directly on arrays
Universal Functions	Use <code>np.sqrt()</code> , <code>np.exp()</code> , <code>np.log()</code> , etc., for element-wise mathematical operations
Indexing	Access specific elements using their position (e.g., <code>arr[0]</code> , <code>arr[1, 2]</code>)
Slicing	Extract portions of arrays using slice notation (e.g., <code>arr[:3]</code> , <code>arr[1:5:2]</code>)

NumPy Attributes

NumPy arrays have several attributes that provide important information about the array. These attributes are useful for understanding the structure, size, and data type of the array, which helps in performing operations efficiently.



Key NumPy Attributes:

a) shape

- **Description:** Returns the dimensions of the array as a tuple. For example, (rows, columns) for a 2D array.
- **Use:** Helps determine the size and structure of the array.
- **Example Functionality:**
For an array with 2 rows and 3 columns, the shape will be (2, 3).

b) size

- **Description:** Provides the total number of elements in the array.
- **Use:** Useful for understanding the overall size of the array.
- **Example Functionality:**
For a 3x3 array, the size will be 9.

c) dtype

- **Description:** Specifies the data type of elements in the array (e.g., int32, float64).
- **Use:** Helps confirm the type of data the array holds, which can affect calculations.
- **Example Functionality:**
For an array with integer elements, dtype might return int64.

d) ndim

- **Description:** Indicates the number of dimensions of the array.
- **Use:** Determines whether the array is 1D, 2D, or higher-dimensional.
- **Example Functionality:**
A 1D array has ndim = 1, while a 2D array has ndim = 2.

e) itemsize

- **Description:** Gives the size (in bytes) of each element in the array.
- **Use:** Useful for understanding memory usage of the array.
- **Example Functionality:**
For an array of integers with 4 bytes each, itemsize will return 4.

f) nbytes

- **Description:** Returns the total memory consumed by the array (in bytes).
- **Use:** Useful for checking how much memory the array occupies.
- **Example Functionality:**
For an array with size = 9 and itemsize = 4, nbytes will return 36.

g) T (Transpose)

- **Description:** Provides the transpose of the array. Rows become columns and vice versa.
- **Use:** Commonly used in matrix operations.
- **Example Functionality:**
For a 2x3 array, the transpose will be a 3x2 array.

Summary of Attributes and Their Uses

Attribute	Description	Functionality Example Output
<code>shape</code>	Dimensions of the array	<code>(2, 3)</code>
<code>size</code>	Total number of elements	<code>6</code>
<code>dtype</code>	Data type of array elements	<code>float64</code>
<code>ndim</code>	Number of dimensions	<code>2</code>
<code>itemsize</code>	Size of one element in bytes	<code>4</code>
<code>nbytes</code>	Total memory consumed (in bytes)	<code>24</code>
<code>T</code>	Transpose of the array	Converts 2x3 array to 3x2 array

NumPy: Important Methods

1. `min()` and `max()`

- **Purpose:**
 1. `min()` is used to find the smallest value in an array.
 2. `max()` is used to find the largest value in an array.

- **Functionality:**

Works on 1D or multi-dimensional arrays.

Can find the minimum/maximum along a specific axis in multi-dimensional arrays.

2. `sum()`

- **Purpose:**

Calculates the total sum of elements in an array.

- **Functionality:**

Works on both 1D and multi-dimensional arrays.

Can calculate the sum along a specific axis (row-wise or column-wise).

3. `reshape()`

- **Purpose:**

Changes the shape of an array without altering its data.

- **Functionality:**

Used to create multi-dimensional arrays from 1D arrays or reshape existing arrays into different dimensions.

The total number of elements must remain the same.

4. `count_nonzero()`

- **Purpose:**

Counts the number of non-zero elements in an array.

- **Functionality:**

Useful for sparse data analysis.

Can count non-zero elements across the entire array or along a specific axis.

5. `sort()`

- **Purpose:**

Sorts the elements of an array in ascending order by default.

- **Functionality:**

Can sort along a specific axis in multi-dimensional arrays.

Does not modify the original array but returns a sorted version.

6. `flatten()`

- **Purpose:**

Converts a multi-dimensional array into a 1D array.

- **Functionality:**

Useful when you need to process all elements linearly, regardless of their original structure.

Other Key NumPy Topics

1. Adding Value to an Array of Values

- **What It Is:**
 - Adding a scalar value to each element of an array or performing element-wise addition with another array.
- **Functionality:**
 - NumPy allows you to add a specific value to all elements of an array in a single operation (broadcasting).
 - For example:
 - Adding 5 to [1, 2, 3] results in [6, 7, 8].
- **Key Points:**
 - Broadcasting in NumPy enables arithmetic operations between arrays of different shapes.
 - This method is efficient and avoids writing explicit loops.

2. Diagonal of a Matrix

- **What It Is:**
 - The diagonal of a matrix consists of the elements that run from the top-left corner to the bottom-right corner (main diagonal).
- **Functionality:**
 - The `numpy.diagonal()` function extracts the diagonal elements of a matrix.
- **Applications:**
 - Often used in matrix algebra, such as computing eigenvalues or simplifying operations on square matrices.

3. Trace of a Matrix

- **What It Is:**
 - The trace of a matrix is the sum of its diagonal elements.
- **Functionality:**
 - The `numpy.trace()` function computes the trace of a matrix.
- **Applications:**
 - Used in various mathematical computations, such as determining matrix invariants.

4. Parsing Matrices

- **What It Is:**
 - Parsing matrices refers to analyzing or processing a matrix by extracting or transforming its data.

- In NumPy, matrices are treated as 2D arrays, and parsing typically involves operations like slicing, indexing, or reshaping.
- **Functionality:**
 - **Slicing and Indexing:** Extract specific rows, columns, or elements.
 - Example: Extracting the first row, `array[0, :]`.
 - **Reshaping:** Transform a matrix's shape using `numpy.reshape()`.

5. Adding and Subtracting Matrices

- **What It Is:**
 - Addition and subtraction of matrices involve element-wise operations where corresponding elements are added or subtracted.
- **Functionality:**
 - Use the `+` operator for addition and the `-` operator for subtraction.
 - **Conditions:** Both matrices must have the same shape.

6. Statistical Functions

- These functions are essential for performing basic statistical analysis on arrays.

Function	Description	Example
<code>numpy.mean()</code>	Calculates the average of array elements.	Example: Mean of <code>[1, 2, 3]</code> is <code>2</code> .
<code>numpy.median()</code>	Finds the median value of array elements.	Example: Median of <code>[1, 2, 3]</code> is <code>2</code> .
<code>numpy.std()</code>	Computes the standard deviation.	Example: Std of <code>[1, 2, 3]</code> is <code>0.82</code> .
<code>numpy.sum()</code>	Adds up all elements of the array.	Example: Sum of <code>[1, 2, 3]</code> is <code>6</code> .
<code>numpy.min()</code>	Returns the smallest value in the array.	Example: Min of <code>[1, 2, 3]</code> is <code>1</code> .

7. Filter in NumPy

- **What It Is:**
 - Filtering in NumPy involves extracting elements that satisfy certain conditions.
 - A filter is essentially a Boolean mask that determines which elements to include.
- **Functionality:**
 - Create a condition and apply it to the array.
 - Example:
 - Array: `[1, 2, 3, 4, 5]`
 - Condition: Select elements greater than 2.

- Result: [3, 4, 5]

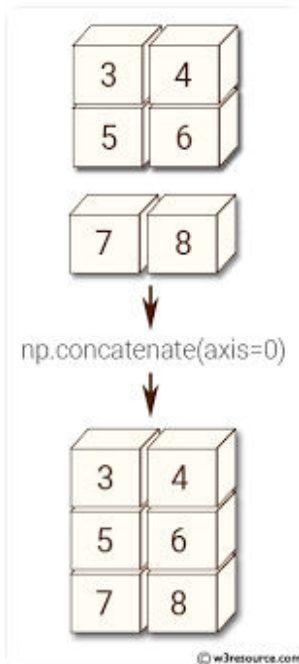
- **Applications:**

- Useful in data pre-processing to remove unwanted data or select specific subsets.

8. Concatenation

Concatenate () function can join the sequence of arrays that we provide, along with the axis specified. If the axis is not explicitly given, it is taken as 0

Concatenating arrays along row:



Q1. Concatenating 1D array

```
arr1=np.array([1,2,3])
arr2=np.array([4,5,6])
arr=np.concatenate((arr1,arr2),axis=0)
print(arr)
```

#output: [1 2 3 4 5 6]

Q2. Concatenating @d array along axis 0

```
arr1=np.array([[1,1,1],[1,1,1]])
arr2=np.array([[9,9,9],[9,9,9]])
arr=np.concatenate((arr1,arr2),axis=0)
print(arr)
```

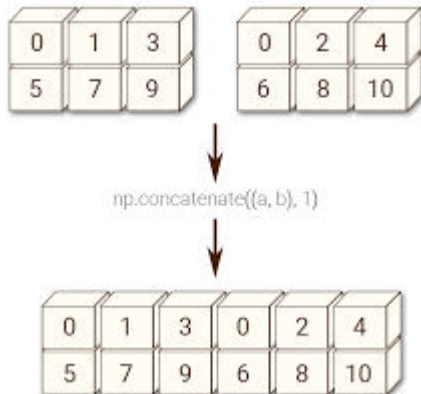
output: `[[1 1 1]`

`[1 1 1]`

`[9 9 9]`

`[9 9 9]]`

Concatenating along the column:



Q1. concatenating along axis 1

```
arr1=np.array([[1,1,1],[1,1,1]])
```

```
arr2=np.array([[9,9,9],[9,9,9]])
```

```
arr=np.concatenate((arr1,arr2),axis=1)
```

```
print(arr)
```

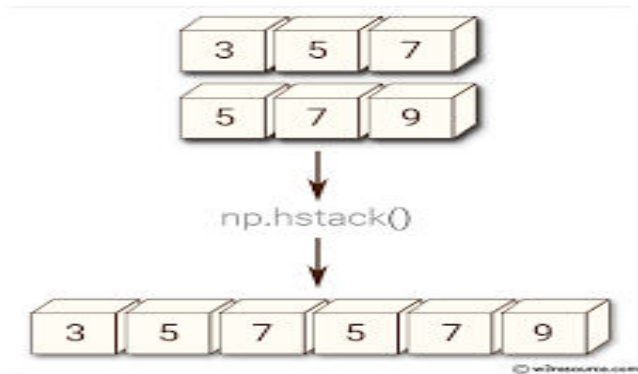
output: `[[1 1 1 9 9 9]`

`[1 1 1 9 9 9]]`

9. Joining array using Stack function

Stacking is the same as concatenation, the only difference is that stacking is done along a new axis.

np.hstack() :



Q1. stacking two 1D arrays horizontally

```
arr1=np.array([0,0])
```

```
arr2=np.array([1,1])
```

```
arr=np.hstack((arr1,arr2))
```

Q2. Stacking two 2D arrays horizontally

```
arr1=np.array([[0,0],[0,0]])
```

```
arr2=np.array([[1,1],[1,1]])
```

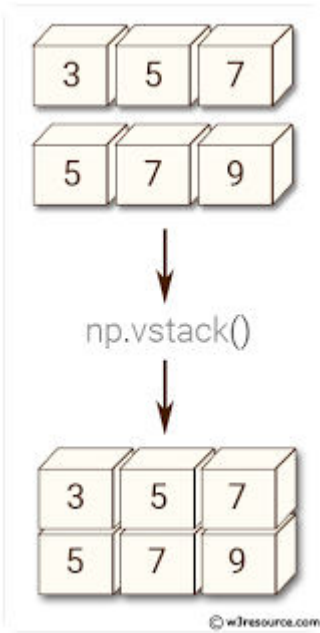
```
arr=np.hstack((arr1,arr2))
```

```
print(arr)
```

output: `[[0 0 1 1]`

`[0 0 1 1]`

np.vstack() :



Q1. Stacking 1D arrays vertically

```
arr1=np.array([0,0])
arr2=np.array([1,1])
arr=np.vstack((arr1,arr2))
print(arr)
```

output: `[[0 0]`
`[1 1]]`

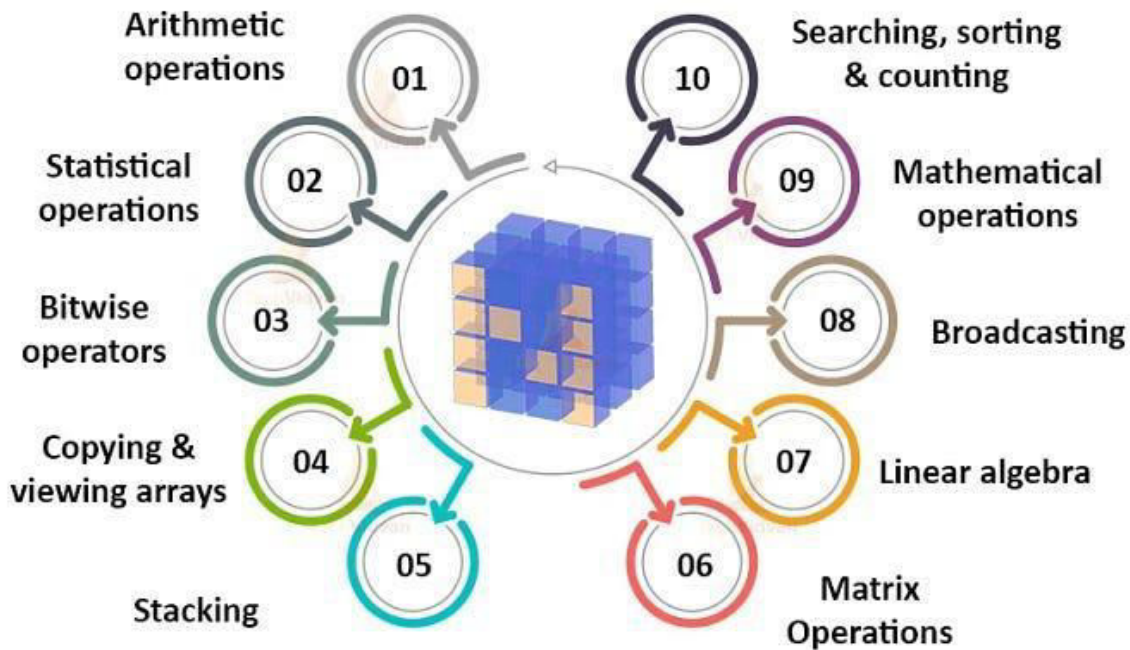
Q2. Stacking 2D arrays vertically

```
arr1=np.array([[0,0],[0,0]])
arr2=np.array([[1,1],[1,1]])
arr=np.vstack((arr1,arr2))
print(arr)
```

output: `[[0 0]`
`[0 0]`
`[1 1]`

[|]]

Uses of NumPy



NumPy Interview Questions and Answers

1. What is NumPy, and why is it used?

- NumPy (Numerical Python) is a Python library used for numerical computations.
- It supports multi-dimensional arrays and provides a wide range of mathematical and statistical operations.

Why use NumPy?

- Faster computations compared to Python lists due to its C-based implementation.
- Memory-efficient as arrays consume less space than lists.
- Extensive functionality for linear algebra, random sampling, and array manipulation.

2. What is the difference between Python lists and NumPy arrays?

Feature	Python List	NumPy Array
Data Type	Can store mixed data types	Stores elements of the same data type
Performance	Slower	Faster due to C-based implementation
Memory Usage	Higher	Lower
Built-in Operations	Limited	Supports mathematical, logical, and statistical operations directly

3. What are broadcasting and its rules in NumPy?

Broadcasting allows NumPy to perform operations on arrays of different shapes.

```
python

a = np.array([1, 2, 3])
b = 2
result = a + b # [3, 4, 5]
```

Rules:

1. If arrays have different dimensions, the smaller array is padded with ones on the left.
2. Arrays must have the same size along each dimension or one of them must be 1.

4. What are some commonly used NumPy functions?

- **Mathematical Operations:** np.add(), np.subtract(), np.multiply(), np.divide().
- **Statistical Functions:** np.mean(), np.median(), np.std(), np.var().
- **Array Manipulation:** np.reshape(), np.flatten(), np.transpose().
- **Sorting and Filtering:** np.sort(), np.where(), np.unique().

5. What is the difference between np.copy() and np.view()?

Feature	<code>np.copy()</code>	<code>np.view()</code>
Data Storage	Creates a new array with copied data.	Shares data with the original array.
Independence	Independent of the original array.	Changes to the view affect the original.
Use Case	When data independence is needed.	When memory efficiency is a priority.

6. How do you filter elements in a NumPy array?

Use conditions to create a Boolean mask, and apply it to the array:

```
python

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

7. Explain the difference between `reshape()` and `resize()` in NumPy.

Function	<code>reshape()</code>	<code>resize()</code>
Behavior	Returns a new array with a different shape.	Modifies the array in-place or creates a new array.
Original Array	Unchanged.	Can change the original array.
Example	<code>np.reshape(arr, (2, 3))</code>	<code>np.resize(arr, (2, 3))</code>

8. What is the difference between `axis=0` and `axis=1` in NumPy?

- **axis=0:** Operations are performed along the rows (column-wise).
- **axis=1:** Operations are performed along the columns (row-wise).
- Example: Summing elements in a 2D array:

```
python

arr = np.array([[1, 2], [3, 4]])
np.sum(arr, axis=0) # [4, 6] (column-wise)
np.sum(arr, axis=1) # [3, 7] (row-wise)
```

9. What is NumPy's `arange()` function?

- `arange()` generates values within a given range at equal intervals.
- Example:

```
python

np.arange(0, 10, 2) # [0, 2, 4, 6, 8]
```

10. How do you handle missing values in NumPy?

- Replace missing values with `np.nan` (Not a Number).
- Use functions that handle nan values:
 - `np.nanmean()` – Computes mean ignoring nan.
 - `np.isnan()` – Checks for nan values.

11. How can you concatenate arrays in NumPy?

Use the `numpy.concatenate()` function:

```
python

a = np.array([1, 2])
b = np.array([3, 4])
result = np.concatenate((a, b)) # [1, 2, 3, 4]
```

12. What is the purpose of NumPy's random module?

The `numpy.random` module is used to generate random numbers:

- Generate random integers: `np.random.randint(1, 10, size=5)`.
- Generate random floats: `np.random.random(size=5)`.
- Shuffle arrays: `np.random.shuffle(array)`.

13. What is the difference between `flatten()` and `ravel()` in NumPy?

Function	<code>flatten()</code>	<code>ravel()</code>
Behavior	Returns a new flattened array.	Returns a view (if possible).
Data Sharing	Creates a copy.	Shares memory with the original.

14. How do you count the number of times a particular value appears in an array of integers?

The `bincount()` function can be used to count the number of times a given value appears. It's worth noting that the `bincount()` function can take either positive integers or boolean expressions as an argument. Negative integers aren't allowed to be utilized.

```
import numpy as np

arr = np.array([0, 5, 4, 0, 4, 4, 3, 0, 0, 5, 2, 1, 1, 9])

print(np.bincount(arr))
```

15. What Is the Distinction Between Numpy & Scipy?

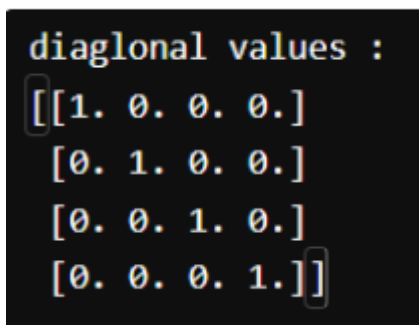
- NumPy would be perfect if it only had the array data type and the most basic operations: indexing, sorting, reshaping, basic elementwise functions, and so on. SciPy would house all numerical codes.
- However, compatibility is one of NumPy's main goals, therefore it tries to keep all features supported by any of its predecessors. As a result, NumPy includes several linear algebra routines that are better appropriately found in SciPy.
- In any case, SciPy has more advanced linear algebra modules, as well as a variety of additional numerical techniques. You should probably install both NumPy and SciPy if you're undertaking scientific computing with Python. The majority of new features belong in SciPy, not NumPy.

16. Is it possible to use `eye()` function to generate diagonal values?

```
import numpy as np

arr = np.eye(4)

print("\ndiagonal values : \n",arr)
```

A terminal window showing the output of the code. The text is as follows:

```
diagonal values :
[[1.  0.  0.  0.]
 [0.  1.  0.  0.]
 [0.  0.  1.  0.]
 [0.  0.  0.  1.]]
```

17. How do you convert a Pandas DataFrame to a NumPy array?

To convert a Pandas DataFrame to a NumPy array, you can use the `.to_numpy()` method. This method returns the underlying data of the DataFrame as a NumPy array.

python

```
import pandas as pd

# Create a DataFrame
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data)

# Convert to NumPy array
numpy_array = df.to_numpy()
print("NumPy Array:\n", numpy_array)
```

18. What do you understand by Vectorization in NumPy?

Vectorization in NumPy refers to performing operations directly on entire arrays rather than looping through individual elements. It is a feature that allows high-performance computation because the operations are implemented in optimized C code under the hood.

Benefits of Vectorization:

- Faster execution compared to loops.
- Cleaner and more concise code.
- Efficient memory usage.

python

```
import numpy as np

# Using Loops
arr = [1, 2, 3, 4, 5]
result = [x**2 for x in arr]
print("Result using loops:", result)

# Using NumPy (Vectorized)
arr_np = np.array([1, 2, 3, 4, 5])
result_vectorized = arr_np**2
print("Result using Vectorization:", result_vectorized)
```

19. Write a program for interchanging two axes of the NumPy array.

You can use the `np.swapaxes()` function or the `.swapaxes()` method of a NumPy array to interchange two axes.

```
python

import numpy as np

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

# Swap axes
swapped = np.swapaxes(arr, 0, 1)
print("Original Array:\n", arr)
print("Swapped Axes Array:\n", swapped)
```

20. What are some of the limitations of NumPy.

NumPy is a powerful library for numerical and scientific computing in Python but it has some limitations depending on some requirements. Here are some of the limitations of NumPy.

- Homogeneous Data Types
- Memory Usage
- Single-threaded
- Limited Support for Missing Data
- Limited Support for Labeling Data
- Limited Support for Advanced Statistics
- Performance Overheads for Small Arrays
- Limited Support for GPU Acceleration
- Complex Installation for Some Platforms
- Limited Support for Distributed Computing