1. Why is NumPy Faster than Python Lists?

NumPy is significantly faster than Python lists due to the following reasons:

Homogeneous Data Storage

NumPy arrays store elements of the same data type, whereas Python lists can store mixed data types. This uniformity allows NumPy to perform operations more efficiently.

Contiguous Memory Allocation

NumPy arrays store elements in contiguous memory blocks. Python lists store references to objects scattered in memory. Continuous memory improves computational speed and cache performance.

Vectorized Operations

NumPy performs operations on entire arrays at once without using explicit loops. Python lists require iteration using loops, which is slower.

Implementation in C

NumPy is implemented using C under the hood. Therefore, mathematical operations are executed much faster compared to pure Python code.

2. What is Broadcasting?

Broadcasting is a mechanism in NumPy that allows arithmetic operations between arrays of different shapes. NumPy automatically expands the smaller array to match the shape of the larger array without copying data.

Example:

```
import numpy as np

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

print(a + b)
```

Output:

```
[6 7 8]
```

3. What is Vectorization? Why is it Important?

Definition

Vectorization refers to performing operations on entire arrays instead of using loops.

Example:

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

print(a + b)
```

Importance

- Faster computation
- Cleaner and shorter code
- Efficient memory usage
- Essential for machine learning and data science tasks

4. How NumPy Integrates with Machine Learning?

NumPy plays a foundational role in Machine Learning.

Matrix and Vector Operations

Machine learning models rely heavily on vectors and matrices. NumPy efficiently handles matrix multiplication, dot products, and transformations.

Linear Algebra Support

NumPy provides functions for: - Matrix multiplication - Transpose - Inverse - Eigenvalues

Data Preprocessing

NumPy is used for reshaping data, normalization, scaling, and numerical transformations.

Integration with ML Libraries

Libraries such as Scikit-learn, TensorFlow, and PyTorch internally use NumPy arrays for data handling.

5. Advantages of NumPy in Industrial Scenarios

- High performance numerical computing
- Memory efficiency

- Supports multi-dimensional arrays
- Efficient large dataset handling
- Widely used in AI, Data Science, Finance, Image Processing, and Scientific Research

6. Creating Single and Multi-Dimensional Arrays

One-Dimensional Array

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

Two-Dimensional Array

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

Using ndmin

```
c = np.array([1,2,3], ndmin=2)
```

7. Indexing and Slicing

Indexing in 1D Array

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

print(a[1])
print(a[1:4])
```

Indexing in 2D Array

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

print(b[0,1])
print(b[:,1])
print(b[0:2,1:3])
```

8. Properties of NumPy Arrays

shape

Returns the dimensions of the array.

```
a.shape
```

size

Returns the total number of elements.

```
a.size
```

dtype

Returns the data type of array elements.

```
a.dtype
```

ndmin

Specifies the minimum number of dimensions.

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

9. Statistical Operations and Aggregate Functions

Mean

```
np.mean(a)
```

Median

```
np.median(a)
```

Sum

```
np.sum(a)
```

Standard Deviation

```
np.std(a)
```

Minimum and Maximum

```
np.min(a)
np.max(a)
```