NumPy Assignment

1) Why is NumPy faster than Python lists? Explain in detail.

NumPy is faster than Python lists because it is implemented using optimized C code and stores data in continuous memory locations. Python lists store references to objects, not actual values, which increases memory overhead and reduces speed.

NumPy arrays contain elements of the same data type. Because of this uniformity, operations can be performed on the whole array at once instead of processing each element separately.

Another important reason is vectorization. NumPy avoids Python loops and performs calculations internally using compiled routines, making execution much faster.

Due to efficient memory usage, better cache performance, and low-level implementation, NumPy can be many times faster than standard Python lists for numerical computations.

2) What is Broadcasting?

Broadcasting is a mechanism that allows NumPy to perform arithmetic operations on arrays of different shapes. Instead of manually resizing arrays, NumPy automatically adjusts the smaller array to match the larger one.

This avoids extra memory usage and simplifies coding.

Example:

```
import numpy as np

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

b = 10

print(a + b)
```

Explanation:

Here, the scalar value 10 is automatically added to each element of array a.

Output:

```
[11 12 13]
```

Broadcasting helps write shorter, cleaner, and faster code.

3) What is Vectorization? Why is it important?

Vectorization means applying operations directly to entire arrays rather than using loops.

Without vectorization:

We manually iterate element by element, which is slower.

With vectorization:

NumPy performs the operation in optimized compiled code.

Example:

```
import numpy as np

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

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

print(a + b)
```

Explanation:

Each element of a is added to the corresponding element of b.

Vectorization is important because it improves speed, reduces code complexity, and is essential for large-scale numerical and machine learning computations.

4) How does NumPy integrate with Machine Learning? Explain in detail.

NumPy acts as the backbone of machine learning. Most ML frameworks like Scikit-learn, TensorFlow, and PyTorch rely heavily on NumPy arrays.

Machine learning algorithms require large amounts of numerical computations such as matrix multiplication, vector operations, and statistical calculations. NumPy provides fast and efficient functions for these tasks.

Before training models, datasets are usually converted into NumPy arrays. This helps in quick manipulation and mathematical processing.

Because NumPy is optimized for performance, it allows ML systems to train faster and handle huge datasets efficiently.

5) What are the advantages of NumPy in industrial scenarios?

NumPy provides high-speed computation and supports handling large volumes of numerical data. It offers powerful multi-dimensional array operations and advanced mathematical functions.

It integrates smoothly with libraries such as pandas for data handling, matplotlib for visualization, and machine learning frameworks.

Industries prefer NumPy because it improves performance, reduces memory consumption, and ensures reliable results.

It is widely used in analytics, finance, artificial intelligence, healthcare, robotics, and scientific research.

6) How can we create single and multi-dimensional arrays? Explain with examples.

NumPy allows creation of arrays with different dimensions.
A dimension represents how data is organized.

1D → line
2D → table
3D → collection of tables

1-D Array

```
import numpy as np
```

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

Explanation:
A single sequence of elements. Only one index is needed.

Structure:

```
[1 2 3 4]
```

2-D Array

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

Explanation:
Contains rows and columns. Two indices are required.

Structure:

```
[[1 2]
 [3 4]]
```

3-D Array

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

Explanation:
A group of 2D arrays stacked together.
Three indices are used → layer, row, column.

7) Give examples of indexing and slicing (1D & 2D).

1D Indexing

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

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

Explanation:
Returns element at position 1 → 20.

1D Slicing

```
print(a[1:3])
```

Explanation:
Returns elements from index 1 to 2 → [20 30].

2D Indexing

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

Explanation:
Row 1, column 2 → 6.

2D Slicing

```
print(b[:,1])
```

Explanation:
Selects all rows from column 1 → [2 5].

8) Explain the properties of NumPy arrays.

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

shape → shows rows and columns
Example output → (2, 3)

size → total elements
Example → 6

dtype → data type of elements
Example → int32 or int64

ndim → number of dimensions
Example → 2

ndmin → forces array to have minimum dimensions during creation.

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

9) Give examples of statistical and aggregate operations.

```
import numpy as np

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

Sum – adds all elements

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

Mean – average value

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

Median – middle value

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

Standard Deviation – spread of data

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

Minimum / Maximum

```
np.min(a)

np.max(a)
```

These functions are widely used in data analysis and machine learning.