

Introduction to NumPy

What is NumPy?

NumPy stands for **Numerical Python**. It is a core Python library for:

- Scientific computing
- Data analysis
- Machine learning

It Provides:

- Fast multi-dimensional arrays
- Matrix operations
- Mathematical functions

Why Not Use Python Lists for Numerical Work?

1. Performance Issues

Python lists are slow for large-scale numerical operations. The main reason being is that lists are basically *storing references to objects scattered across memory*.

Typically, numerical operations involving lists will also require python loops to access the elements, which are **interpreted & slow**.

2. Memory Inefficiency

The main advantage of lists are that they can hold *data of different data types* (integers, strings etc...). This provides large flexibility to lists.

However Each element requires extra memory for type information. This eventually leads to **slower access due to poor cache locality**.

3. No Built-in Vectorization

As mentioned before, element-wise operations involving lists need explicit loops. Along with this, there is also no *native support for mathematical broadcasting*

How NumPy Solves These Problems

1. Speed: 10–100x Faster

1. Numpy arrays are Written in **C/C++** (compiled, not interpreted)
2. Numpy Uses **vectorized operations, CPU - level optimizations (SIMD) and optimised memory access patterns** to improve efficiency

```
# Python List Approach
list_a = [1, 2, 3, 4, 5]
list_b = [10, 20, 30, 40, 50]
result = []
for i in range(len(list_a)):
    result.append(list_a[i] + list_b[i])

# NumPy Approach
import numpy as np
arr_a = np.array([1, 2, 3, 4, 5])
arr_b = np.array([10, 20, 30, 40, 50])
result = arr_a + arr_b
```

2. Memory Efficiency

NumPy arrays store data in contiguous memory

Numpy has Fixed data type and therefore has very low memory overhead

3. Vectorization

Vectorization of Numpy allows operations to be done on entire arrays **without explicit loops**

```
numbers = np.arange(1000000)
squares = numbers ** 2
```

4. Broadcasting

Numpy arrays can automatically expand dimensions as well as ***broadcast changes to all elements*** of an array easily without the need of explicit loops

Allows operations on different shapes

```
arr = np.array([[1,2,3], [4,5,6]])
result = arr + 10
```

Key Advantages of NumPy

- Rich math functions (Linear Algebra, Statistics)
- Supports 1D, 2D, and 3D arrays
- Backbone of:
 - Pandas
 - Scikit-learn
 - TensorFlow
 - PyTorch
- Universal functions (ufuncs)

Performance Comparison

Operation	Python List	NumPy	Speedup
Sum	85 ms	0.8 ms	106x
Addition	120 ms	1.2 ms	100x
Square	150 ms	2.5 ms	60x
Mean	95 ms	0.5 ms	190x

When to Use NumPy

Use NumPy When:

- Large numerical datasets
- Scientific computing
- Machine learning
- Matrix operations

Avoid NumPy When:

- Mixed data types
- Very small datasets
- Mostly strings or objects

Memory Layout Example

```
# Python List
my_list = [1, 2, 3, 4, 5] # ~140B

# NumPy Array
my_array = np.array([1,2,3,4,5], dtype=np.int32) #
~20B
```

Conclusion

- NumPy turns Python into a scientific powerhouse
- Faster, memory-efficient, and cleaner syntax
- Core foundation for Data Science and AI

Key Takeaway: NumPy makes programs faster, cleaner, and more memory-efficient.