NumPy

Contents

NumPy (Numerical Python) is a fundamental library for **scientific and numerical computing** in Python.

Brief History

- NumPy was created in **2005** by **Travis Oliphant** as a successor to Numeric and Numarray.

- It quickly became the **de facto standard** for numerical arrays in Python.
- Today, it is maintained by the open-source community under the **NumPy Foundation**.

🔗 Official Website: https://numpy.org

Why NumPy?

- 🚀 **Performance**: Much faster than Python lists for numerical operations.
- 📊 **Flexibility**: Supports **multi-dimensional arrays** (1D, 2D, 3D… nD).
- ⚡ **Vectorization**: Perform operations **without explicit loops**.
- 🧩 **Ecosystem**: Foundation for libraries like **Pandas, SciPy, Matplotlib, TensorFlow, PyTorch**.

NumPy in Artificial Intelligence

- Modern **AI and Machine Learning frameworks** (TensorFlow, PyTorch, JAX) are built on top of NumPy-like tensors.
- It provides the **matrix operations** (dot products, linear algebra) that power neural networks.
- Efficient handling of large datasets is **critical for training AI models**, and NumPy makes it possible.

Installation & Import

```
pip install numpy
```

```
import numpy as np

# Show installed version
print("NumPy version:", np.__version__)
```

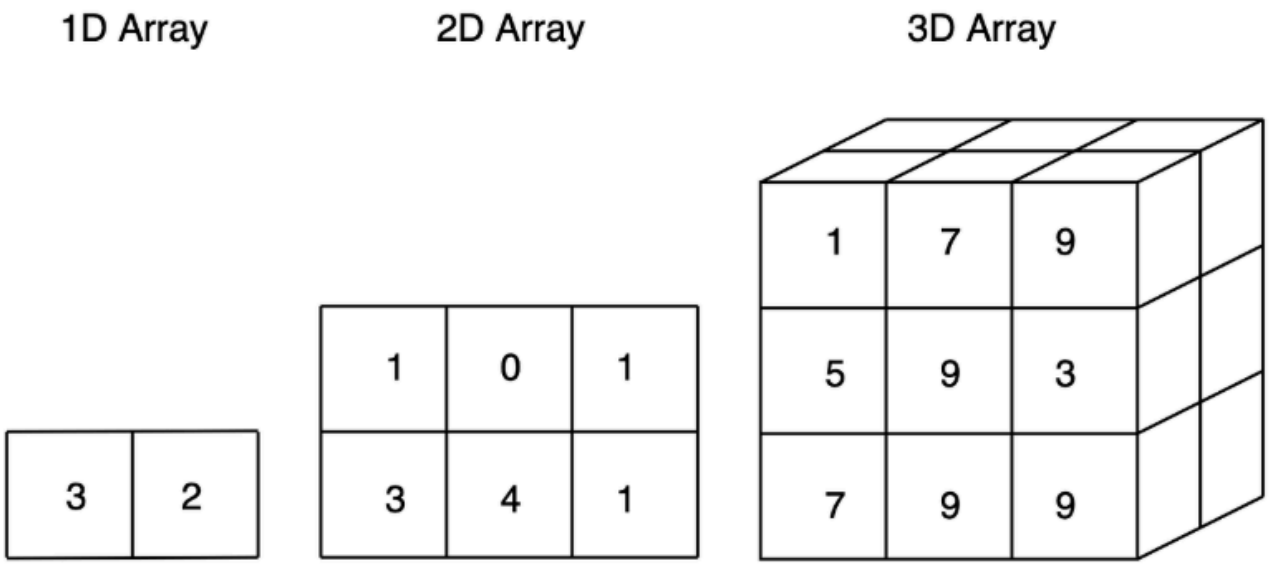
```
NumPy version: 1.26.4
```

Arrays: 1D, 2D, 3D

NumPy arrays can have **different dimensions**:

- **1D Array** → Vector (a simple list).
- **2D Array** → Matrix with rows and columns.
- **3D Array** → A cube (stack of matrices).

Visual Example:



```
# 1D array (vector)
arr1D = np.array([3, 2])
print("1D Array:")
print(arr1D)
print("Shape:", arr1D.shape, "| Dimensions:", arr1D.ndim, "| Data type:", arr1D.dtype)

# 2D array (matrix)
arr2D = np.array([[1, 0, 1], [3, 4, 1]])
print("\n2D Array:")
print(arr2D)
print("Shape:", arr2D.shape, "| Dimensions:", arr2D.ndim, "| Data type:", arr2D.dtype)

# 3D array (cube)
arr3D = np.array([
    [[1, 7, 9], [5, 9, 3], [7, 9, 9]]
])
print("\n3D Array:")
print(arr3D)
print("Shape:", arr3D.shape, "| Dimensions:", arr3D.ndim, "| Data type:", arr3D.dtype)
```

```
1D Array:
[3 2]
Shape: (2,) | Dimensions: 1 | Data type: int32

2D Array:
[[1 0 1]
 [3 4 1]]
Shape: (2, 3) | Dimensions: 2 | Data type: int32

3D Array:
[[[1 7 9]
  [5 9 3]
  [7 9 9]]]
Shape: (1, 3, 3) | Dimensions: 3 | Data type: int32
```

shape, ndim, dtype

- `shape`: a tuple telling you *how many elements along each axis*.
  - `arr1D.shape == (2,)` → one axis of length 2.
  - `arr2D.shape == (2, 3)` → 2 rows × 3 columns.
  - `arr3D.shape == (1, 3, 3)` → 1 matrix (depth) each of shape 3×3.
- `ndim`: number of axes. The examples are 1, 2, and 3 respectively.
- `dtype`: storage type (e.g., `int64`, `float64`, `bool`). NumPy may **upcast** to a common type when mixing ints/floats.

Reading shapes correctly

- `(R, C)` always means **Rows × Columns** for 2D.
- `(D, R, C)` for 3D means **Depth (or planes) × Rows × Columns**.

Typical pitfalls

- Ragged lists (lists of different-length rows) lead to `dtype=object` and lose vectorized speed.
- If you need a specific precision, **set** `dtype` **explicitly** (e.g., `dtype=np.float32`).

Creating Arrays Quickly

NumPy provides many convenient array generators:

- `zeros` → Array of all zeros.
- `ones` → Array of all ones.
- `arange` → Range of numbers.
- `linspace` → Evenly spaced numbers between start & end.
- `eye` → Identity matrix.
- `random.rand` → Random numbers between 0 and 1.

```
print("Zeros (2x3):\n", np.zeros((2, 3)))
print("Ones (3x3):\n", np.ones((3, 3)))
print("Arange 0-10 step 2:", np.arange(0, 10, 2))
print("Linspace 0-1 with 5 values:", np.linspace(0, 1, 5))
print("Identity matrix (4x4):\n", np.eye(4))
print("Random matrix (2x2):\n", np.random.rand(2, 2))
```

```
Zeros (2x3):
 [[0. 0. 0.]
 [0. 0. 0.]]
Ones (3x3):
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
Arange 0-10 step 2: [0 2 4 6 8]
Linspace 0-1 with 5 values: [0.   0.25 0.5  0.75 1.  ]
Identity matrix (4x4):
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Random matrix (2x2):
 [[0.34040653 0.53937439]
 [0.4655407  0.08848227]]
```

Choosing the right constructor (and when not to)

- `np.zeros` / `np.ones(shape, dtype=...)`
  - ✅ Great for initialized buffers, masks, or when you need a known start state.
  - ⚠️ For huge arrays, initialization cost matters—`np.empty` is faster but **leaves garbage values** (use only if you will overwrite *every* element).
- `np.arange(start, stop, step)`
  - ✅ Perfect for integer grids and index ranges.
  - ⚠️ With floats, step accumulation can cause rounding surprises (you may miss the expected stop). For float grids prefer `linspace`.
- `np.linspace(start, stop, num)`
  - ✅ Best for evenly spaced **float** samples including endpoints (e.g., plotting, sampling).
  - ⚠️ If you need an exclusive end or exact integer steps, use `arange`.
- `np.eye(n)`
  - ✅ Identity matrix used in linear algebra (diagonal of ones).
  - 🔁 Variants: `np.diag` to place custom diagonals.
- `np.random.rand(m, n)`
  - ✅ Quick random uniform samples in [0, 1).
  - 🎲 For reproducibility, set a seed via `np.random.seed(123)` or use the Generator API `np.random.default_rng(123)`.

Output reading guide

- The code above will print literal arrays so you can **see shapes and default dtypes** (often float for generators like `zeros`/`ones` when you pass floats, and float for random).

NumPy Functions Overview

1. Array Creation Functions

Function	Description
`np.array([...])`	Create array from Python list/tuple
`np.zeros((m,n))`	Create array of zeros
`np.ones((m,n))`	Create array of ones
`np.full((m,n), val)`	Fill with constant value
`np.arange(start, stop, step)`	Range of numbers
`np.linspace(start, stop, num)`	Evenly spaced numbers
`np.eye(n)`	Identity matrix
`np.random.rand(m,n)`	Random floats in [0,1)
`np.random.randint(low, high, size)`	Random integers

2. Array Inspection Functions

Function	Description
`arr.shape`	Dimensions (rows, cols, ...)
`arr.ndim`	Number of dimensions
`arr.size`	Total number of elements
`arr.dtype`	Data type of elements
`arr.astype(type)`	Convert to another type

3. Mathematical Functions

Function	Description
`+ - * / **`	Element-wise operations
`np.add(a,b)`, `np.subtract(a,b)`	Addition / subtraction
`np.multiply(a,b)`, `np.divide(a,b)`	Multiplication / division
`np.sqrt(a)`, `np.exp(a)`, `np.log(a)`	Square root, exponential, logarithm
`np.sin(a)`, `np.cos(a)`, `np.tan(a)`	Trigonometric functions

4. Aggregation Functions

Function	Description
`np.sum(arr, axis=...)`	Sum (total or by axis)
`np.mean(arr)`	Average value
`np.median(arr)`	Median
`np.std(arr)`	Standard deviation
`np.min(arr)`, `np.max(arr)`	Minimum & Maximum
`np.argmin(arr)`, `np.argmax(arr)`	Indices of min & max

5. Linear Algebra (np.linalg)

Function	Description
`np.dot(a,b)` or `a @ b`	Dot product
`np.linalg.inv(M)`	Inverse of matrix
`np.linalg.det(M)`	Determinant
`np.linalg.eig(M)`	Eigenvalues & eigenvectors
`np.linalg.solve(A,b)`	Solve linear equations

6. Reshaping & Combining

Function	Description
`arr.reshape(new_shape)`	Change shape
`arr.ravel()` / `arr.flatten()`	Flatten to 1D
`np.hstack([a,b])`, `np.vstack([a,b])`	Horizontal / Vertical stack
`np.concatenate([...], axis=...)`	Join arrays along axis

👉 These cover **most common daily NumPy usage**.

1. Array Creation

```
import numpy as np

print("Zeros (2x2):\n", np.zeros((2,2)))
print("Arange 0-9:", np.arange(10))
print("Linspace 0 to 1 (5 values):", np.linspace(0,1,5))
print("Random integers (1-10):\n", np.random.randint(1,10,(2,3)))
```

```
Zeros (2x2):
 [[0. 0.]
 [0. 0.]]
Arange 0-9: [0 1 2 3 4 5 6 7 8 9]
Linspace 0 to 1 (5 values): [0.   0.25 0.5  0.75 1.  ]
Random integers (1-10):
 [[1 4 9]
 [5 1 8]]
```

2. Array Inspection

```

```

```
arr = np.array([[1,2,3],[4,5,6]])
print("Array shape:", arr.shape)
print("Dimensions:", arr.ndim)
print("Size:", arr.size)
print("Data type:", arr.dtype)
```

```
Array shape: (2, 3)
Dimensions: 2
Size: 6
Data type: int32
```

3. Mathematical Operations

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

print("Addition:", a+b)
print("Square root:", np.sqrt(a))
print("Sine:", np.sin(a))
```

```
Addition: [5 7 9]
Square root: [1.         1.41421356 1.73205081]
Sine: [0.84147098 0.90929743 0.14112001]
```

4. Aggregation Functions

```
print("Sum:", arr.sum())
print("Column sums:", arr.sum(axis=0))
print("Row sums:", arr.sum(axis=1))
print("Mean:", arr.mean())
print("Argmax index:", arr.argmax())
```

```
Sum: 21
Column sums: [5 7 9]
Row sums: [ 6 15]
Mean: 3.5
Argmax index: 5
```

5. Linear Algebra

```
M = np.array([[1,2],[3,4]])
N = np.array([[2,0],[1,2]])

print("Matrix product:\n", M @ N)
print("Determinant of M:", np.linalg.det(M))
print("Inverse of M:\n", np.linalg.inv(M))
```

```
Matrix product:
 [[ 4  4]
 [10  8]]
Determinant of M: -2.0000000000000004
Inverse of M:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

6. Reshaping & Combining

```
print("Reshape 2x3 to 3x2:\n", arr.reshape(3,2))
print("Flatten:", arr.flatten())
print("Horizontal stack:", np.hstack([a,b]))
print("Vertical stack:\n", np.vstack([a,b]))
```

```
Reshape 2x3 to 3x2:
 [[1 2]
 [3 4]
 [5 6]]
Flatten: [1 2 3 4 5 6]
Horizontal stack: [1 2 3 4 5 6]
Vertical stack:
 [[1 2 3]
 [4 5 6]]
```