9. Axis Concept

1. Topic Overview

What this topic is

The **axis** in NumPy tells **along which direction an operation runs** on an array.

Why it exists

Arrays can have many dimensions.
NumPy needs a clear way to say:

- operate **row-wise**
- operate **column-wise**
- or operate **across higher dimensions**

Axis gives that control.

One real-world analogy

Think of an Excel sheet:

- Rows = students
- Columns = subjects
  Axis tells NumPy whether to work **per student** or **per subject**.

2. Core Theory (Deep but Clear)

2.1 What “axis” actually means

Axis is **a dimension index**.

- Axis `0` → first dimension

- Axis `1` → second dimension
- Axis `2` → third dimension
- And so on

NumPy always counts axes from **left to right**.

2.2 Axis vs Shape

Consider this array:

```python
import numpy as np

a = np.array([
    [10, 20, 30],
    [40, 50, 60]
])
```

```
Shape = (2, 3)
```

Meaning:

- Axis 0 has size 2
- Axis 1 has size 3

Visual view:

```
Axis 1 →
[10  20  30]
[40  50  60]
 ↑
Axis 0
```

2.3 How NumPy internally thinks

Internally NumPy sees:

- A **block of memory**

- Plus **shape**
- Plus **axis numbers**

When you pass `axis=n`, NumPy:

1. Fixes that axis
2. Collapses values along it
3. Reduces array dimensions (unless told not to)

2.4 Important rule (MEMORIZE)

> **Axis = direction you REMOVE values along**

Not the direction you keep.

This is the #1 confusion point.

2.5 Axis in higher dimensions

3D example:

```
arr.shape = (2, 3, 4)
```

Axes:

- Axis 0 → depth (2 blocks)
- Axis 1 → rows (3)
- Axis 2 → columns (4)

Operations collapse **one axis at a time**.

3. Syntax & Examples

3.1 Axis = 0

Meaning

Operate **down the rows**.
Collapse rows.
Result keeps columns.

Syntax

```
np.sum(array, axis=0)
```

Example 1

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

print(np.sum(a, axis=0))
```

Output

```
[5 7 9]
```

Explanation

- Column 1: 1 + 4 = 5
- Column 2: 2 + 5 = 7
- Column 3: 3 + 6 = 9

Example 2

```
print(np.mean(a, axis=0))
```

Output

```
[2.5 3.5 4.5]
```

Each column averaged.

3.2 Axis = 1

Meaning

Operate **across columns**.
Collapse columns.
Result keeps rows.

Syntax

```
np.sum(array, axis=1)
```

Example 1

```
print(np.sum(a, axis=1))
```

Output

```
[ 6 15]
```

Explanation

- Row 1: 1 + 2 + 3 = 6
- Row 2: 4 + 5 + 6 = 15

Example 2

```
print(np.max(a, axis=1))
```

Output

```
[3 6]
```

Max per row.

3.3 Axis in 3D arrays

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

```
Shape = (2, 2, 2)
```

Axis = 0

```
print(np.sum(b, axis=0))
```

Output

```
[[ 6  8]
 [10 12]]
```

Collapses first dimension.

Axis = 1

```
print(np.sum(b, axis=1))
```

Output

```
[[ 4  6]
 [12 14]]
```

Axis = 2

```
print(np.sum(b, axis=2))
```

Output

```
[[ 3  7]
 [11 15]]
```

3.4 keepdims (important)

```
np.sum(a, axis=1, keepdims=True)
```

Keeps dimension size as `1`.

Used for broadcasting later.

4. Why This Matters in Data Science

Data cleaning

- Column-wise missing value handling → `axis=0`
- Row-wise anomaly detection → `axis=1`

Feature engineering

- Feature scaling per column
- Row normalization

Model input preparation

- ML models expect exact shapes
- Wrong axis = wrong tensor shape

ML / DL pipelines

- Batch operations depend on axis
- Loss functions reduce specific axes

What breaks if you don’t understand axis

- Wrong statistics
- Shape mismatch errors
- Silent logical bugs (most dangerous)

5. Common Mistakes (VERY IMPORTANT)

1. **Thinking axis means direction kept**
    - Axis means direction removed
    - Always visualize shape
2. **Using axis=1 on 1D arrays**
    - 1D arrays only have axis=0
3. **Confusing rows and columns**
    - Always check `.shape`
4. **Forgetting keepdims**
    - Breaks broadcasting later
5. **Hardcoding axis without understanding**
    - Code breaks when data shape changes

6. Performance & Best Practices

When it is fast

- Contiguous memory

- Simple reductions (sum, mean)

When it is slow

- Large high-dim arrays
- Multiple axis operations chained

Memory warnings

- Reduction creates new arrays
- keepdims saves shape but not memory

Best practices

- Always print shape before axis ops
- Use comments for axis meaning
- Prefer vectorized axis ops over loops

7. 20 Practice Problems (MANDATORY)

Easy (5)

1. Sum each column of a 2D array.
2. Find max of each row.
3. Compute mean of a 1D array using axis.
4. Use `keepdims=True` and print shape.
5. Count non-zero values per column.

Medium (7)

6. Normalize each row so sum = 1.
7. Replace column means for missing values.
8. Compute variance per feature.
9. Find row with highest sum.
10. Standardize features using axis.

11. Reduce 3D tensor along batch axis.
12. Compare output shapes for different axes.

Hard (5)

13. Implement row-wise softmax.
14. Compute per-class accuracy matrix.
15. Collapse last axis of a 4D tensor.
16. Debug a wrong axis causing shape mismatch.
17. Apply multiple reductions correctly.

Industry-Level Tasks (3)

18. Prepare image batch for CNN input.
19. Aggregate user behavior logs by feature.
20. Reduce time-series data across windows.

8. Mini Checklist

- Axis is a dimension index
- Axis tells what gets reduced
- Axis starts from 0
- Shape changes after reduction
- keepdims controls dimensionality
- Wrong axis = wrong logic

9B. Advanced Axis Concepts (MANDATORY FOR DATA SCIENCE)

1. Topic Overview

What this topic is

Advanced axis concepts control **how dimensions are referenced, reduced, aligned, and expanded** in NumPy.

Why it exists

Real datasets are:

- High dimensional
- Dynamic in shape
- Used in pipelines

Hardcoding `axis=0` or `1` is fragile.

One real-world analogy

GPS directions:

- “Turn left” is vague
- “Turn west” is precise
  Negative and named axis logic makes code robust.

2. Core Theory (Deep but Clear)

2.1 Negative Axis Indexing (CRITICAL)

What it is

Negative axis counts **from the last dimension**.

Axis	Meaning
`axis=-1`	Last axis

Axis	Meaning
`axis=-2`	Second last
`axis=-3`	Third last

Why NumPy supports this

- Shapes change
- Last axis is often “features”
- Negative axis avoids hardcoding

Internal NumPy view

For shape `(A, B, C)` :

Axis	Same as
`0`	`-3`
`1`	`-2`
`2`	`-1`

2.2 Tuple of Axes (Multi-Axis Reduction)

What it is

Reduce **multiple axes at once**.

Syntax

```
np.sum(arr, axis=(axis1, axis2))
```

Why it exists

ML tensors are often:

- 3D
- 4D
- 5D

Reducing one axis at a time is slow and messy.

Internal logic

NumPy collapses **all listed axes together**.

2.3 keepdims (Broadcast Safety)

You saw it earlier.
Here is why it is **mandatory**.

What it does

Preserves reduced axes with size `1`.

Why NumPy needs this

Broadcasting requires compatible shapes.

Without `keepdims`, pipelines break.

2.4 Axis and Broadcasting Alignment

Key rule

Broadcasting works **from the last axis backward**.

Axis mistakes silently break math.

Internal rule

NumPy aligns arrays **right to left**.

Axis placement controls this alignment.

2.5 np.newaxis and axis expansion

What it is

Adds a new axis of size `1`.

Why it matters

- Model inputs
- Broadcasting
- Batch dimension creation

2.6 Axis Order Conventions (INDUSTRY STANDARD)

You MUST know this.

Common conventions

Data	Shape
Tabular	(samples, features)
Images (NumPy)	(batch, height, width, channels)
Images (PyTorch)	(batch, channels, height, width)
Time series	(samples, timesteps, features)

Axis meaning depends on **convention**, not preference.

3. Syntax & Examples

3.1 Negative Axis

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

```
print(np.sum(a, axis=-1))
```

Output

```
[ 6 15]
```

Explanation

- `axis=-1` means last axis
- Same as `axis=1`

```
print(np.sum(a, axis=-2))
```

Output

```
[5 7 9]
```

Same as `axis=0`.

3.2 Tuple of Axes

```
b = np.ones((2, 3, 4))
```

```
print(np.sum(b, axis=(1, 2)))
```

Output

```
[12. 12.]
```

Explanation

- Reduce axes 1 and 2
- Only axis 0 remains

3.3 keepdims

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


print(np.sum(x, axis=1))
print(np.sum(x, axis=1, keepdims=True))
```

Output

```
[6]
[[6]]
```

Shape difference:

- `(1,)` vs `(1,1)`

3.4 Broadcasting with Axis

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

mean = np.mean(data, axis=0, keepdims=True)
print(data - mean)
```

Explanation

- keepdims keeps column shape
- subtraction works safely

3.5 np.newaxis

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


print(v[:, np.newaxis].shape)
```

Output

```
(3, 1)
```

Creates column vector.

4. Why This Matters in Data Science

Data cleaning

- Column stats with dynamic shapes
- Safe broadcasting

Feature engineering

- Normalization across last axis
- Batch feature aggregation

Model input preparation

- Adding batch axis
- Matching framework conventions

ML / DL pipelines

- Loss reduction axes
- Attention mechanisms
- CNN tensor reductions

What breaks if you don’t know this

- Silent math bugs
- Wrong gradients
- Model training instability

5. Common Mistakes (VERY IMPORTANT)

1. Hardcoding `axis=1`
   - Breaks when shape changes
2. Ignoring negative axis
   - Makes code brittle
3. Forgetting keepdims
   - Broadcasting crashes later
4. Reducing wrong axes in tensors
   - Model logic becomes invalid
5. Mixing framework axis conventions
   - PyTorch vs NumPy mismatch

6. Performance & Best Practices

Fast

- Multi-axis reduction
- Using negative axis

Slow

- Sequential axis reductions
- Repeated reshape calls

Memory warnings

- Each reduction creates new array
- keepdims saves shape, not memory

Best practices

- Use `axis=-1` for features
- Comment axis meaning
- Print shapes during debugging

7. 20 Practice Problems (MANDATORY)

Easy (5)

1. Sum last axis of a 2D array.
2. Compute mean using negative axis.
3. Reduce two axes at once.
4. Show shape difference with keepdims.
5. Convert vector to column using newaxis.

Medium (7)

6. Normalize features using axis=-1.
7. Aggregate 3D tensor over spatial axes.
8. Add batch axis to input data.
9. Reduce time-series over timesteps.
10. Use tuple axis for image channels.
11. Debug broadcasting failure.
12. Convert between row and column vectors.

Hard (5)

13. Implement batch-wise normalization.
14. Reduce CNN feature maps correctly.
15. Handle unknown feature dimension safely.
16. Write axis-agnostic reduction code.
17. Fix wrong axis in loss calculation.

Industry-Level Tasks (3)

18. Prepare image tensor for CNN training.
19. Aggregate multi-sensor time-series data.
20. Build reusable normalization utility.

8. Mini Checklist

- Axis can be negative
- Axis can be a tuple
- keepdims is pipeline safety
- Broadcasting depends on axis
- newaxis creates dimensions
- Axis conventions matter

Final honest note (important)

If you **fully master axis**,
NumPy becomes easy.
If you don’t, **everything breaks quietly**.