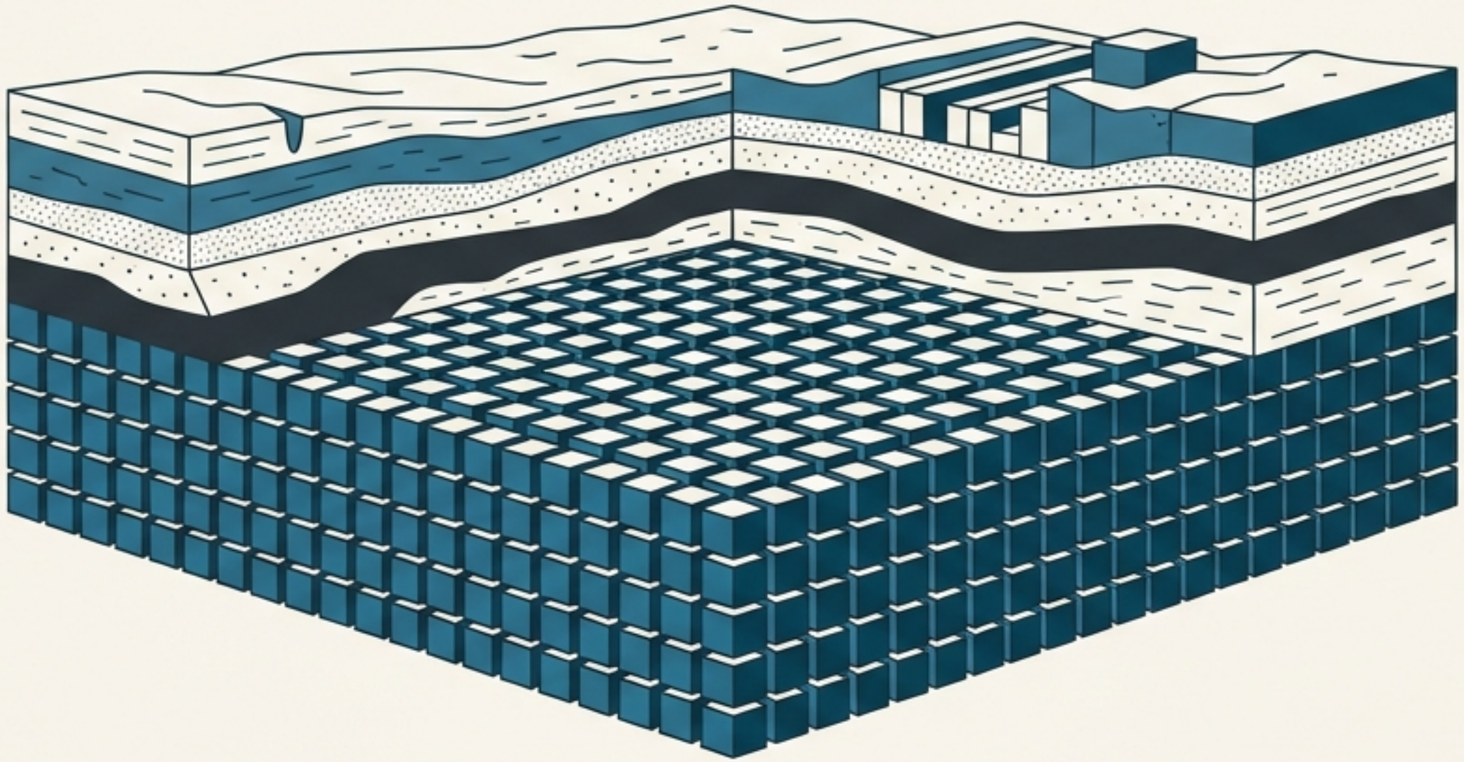
NumPy Fundamentals: The Bedrock of Scientific Computing in Python



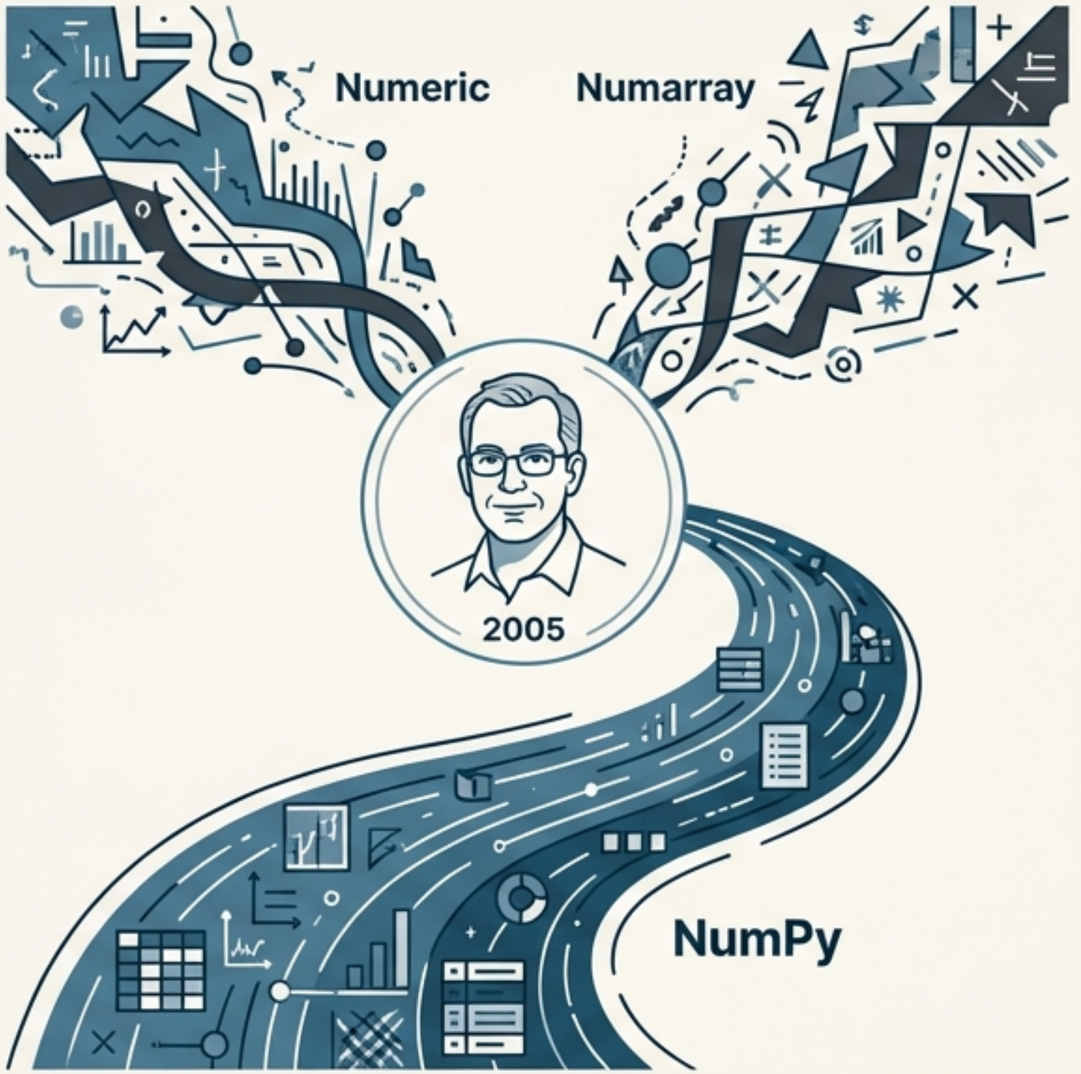
Session 13: Data Science Mentorship Program

Unlocking High-Performance Numerical Processing in Python

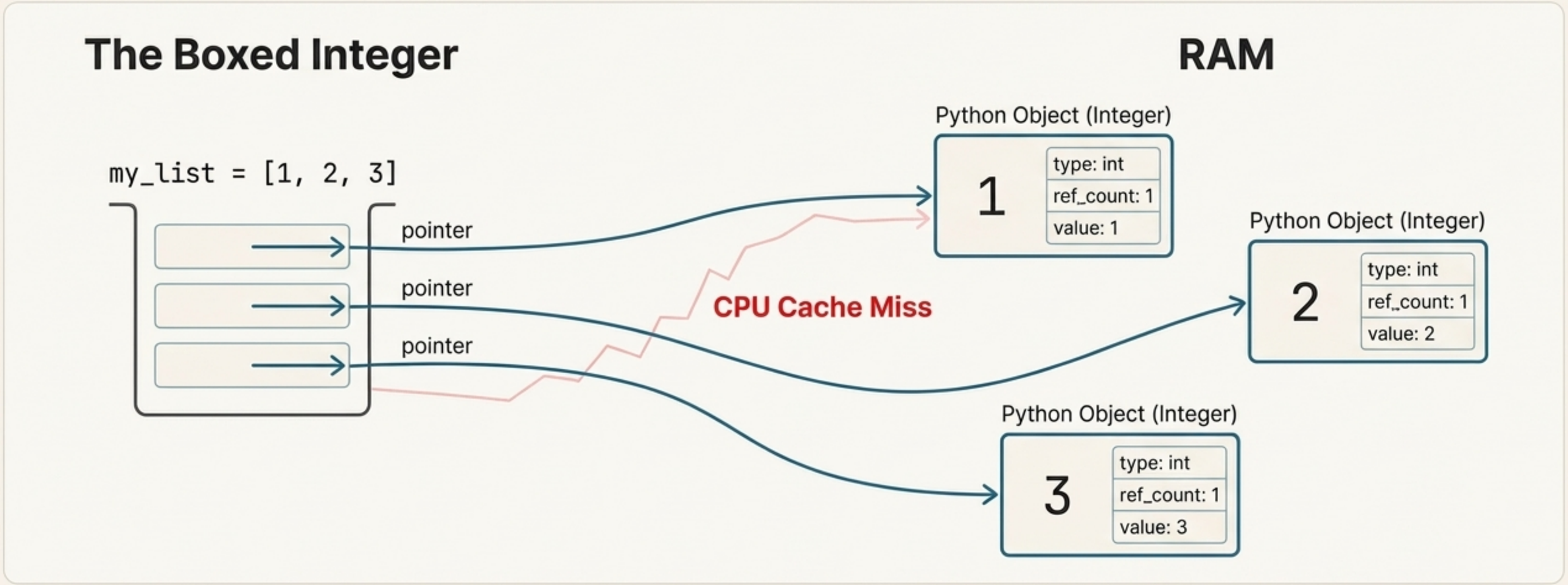
Your Instructor: Dr. Anya Sharma

Why Was NumPy Created? A Unifying Force

- **The Creator:** Travis Oliphant, a data scientist and professor.
- **The Time:** 2005. Python's scientific community was fragmented.
- **The Problem:** Two competing array libraries, **Numeric** and **Numarray**, split the community. Code written for one was incompatible with the other.
- **The Vision:** Oliphant created NumPy to unify these efforts, taking the best features from both to create a single, powerful, and flexible standard for array computing in Python.



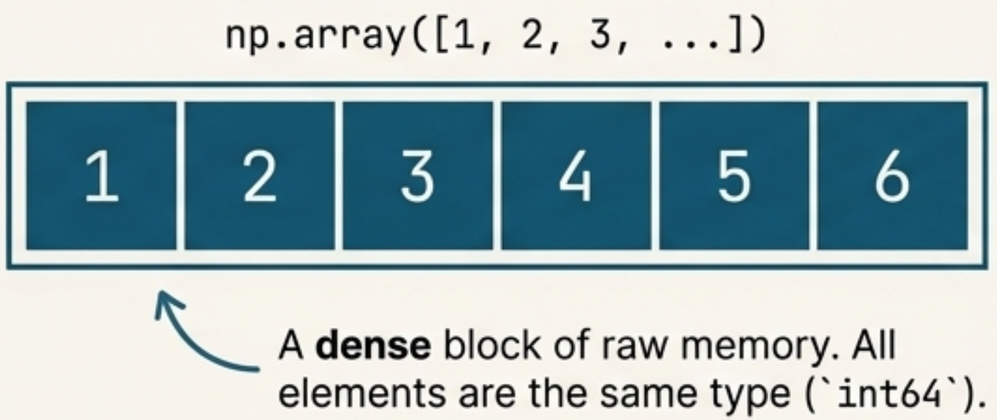
The Bottleneck: Why Python Lists Are Slow for Math



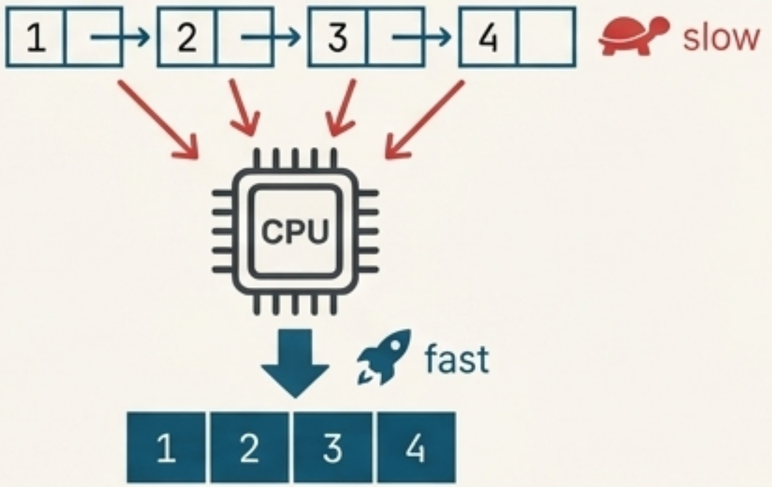
- **Everything is an Object:** A Python list doesn't store data directly.
- **The 'Boxed Integer':** Each number is a full Python object—a 'box' with the value plus metadata.
- **Pointer Chasing:** The list holds pointers (memory addresses) to these scattered 'boxes'.
- **The Result:** Massive memory overhead and slow processing due to inefficient data layout.

The NumPy Solution: Speed Through Simplicity

Contiguous & Homogeneous Memory



Vectorization (SIMD)



Single **I**nstruction, **M**ultiple **D**ata. The CPU processes entire blocks of data in one clock cycle.

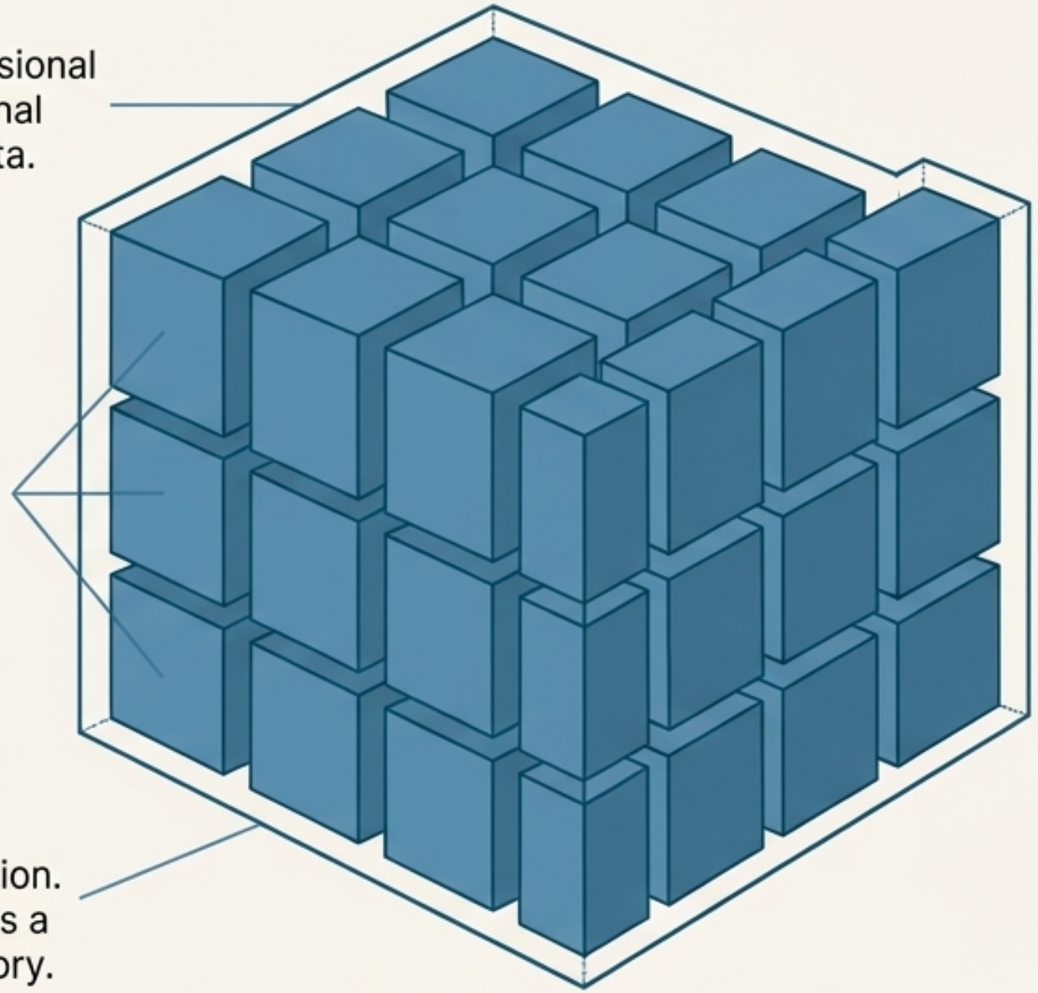
- **Contiguous Memory**: A dense block of raw memory. Values are stored directly. No “boxes,” no pointers.
- **Homogeneity**: All elements must be the same data type, eliminating type-checking during computation.
- **Vectorization (SIMD)**: This allows the CPU to process entire blocks of data at once.
- **The Result**: Operations are 50x-100x faster. C-level speed with Python's simple syntax.

The Core Object: The `ndarray`

The **`ndarray`** (n-dimensional array): A multi-dimensional container for generic data.

Homogeneous: All elements must have the same type (e.g., `float64`).

Fixed Size:
Size is set at creation. Changing it creates a new array in memory.



- The **`ndarray`** (n-dimensional array) is the central data structure in NumPy.
- It's an array of **homogeneous** data (all elements have the same type).
- It has a **fixed size** upon creation; you cannot append elements like a Python list.
- It enables powerful **mathematical operations** on entire blocks of data at once.

Your First `ndarray`: Common Creation Routines

```
import numpy as np
```

From a Python List

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

1	2	3

Uniform Arrays

```
np.zeros((3, 4))
```

0	0	0	0
0	0	0	0
0	0	0	0

```
np.ones((3, 4))
```

1	1	1	1
1	1	1	1
1	1	1	1

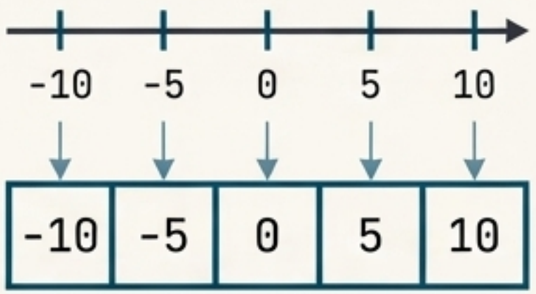
Random Data

```
np.random.random((3, 4))
```

0.12	0.87	0.54	0.39
0.91	0.02	0.75	0.66
0.43	0.28	0.15	0.98

Evenly Spaced Numbers

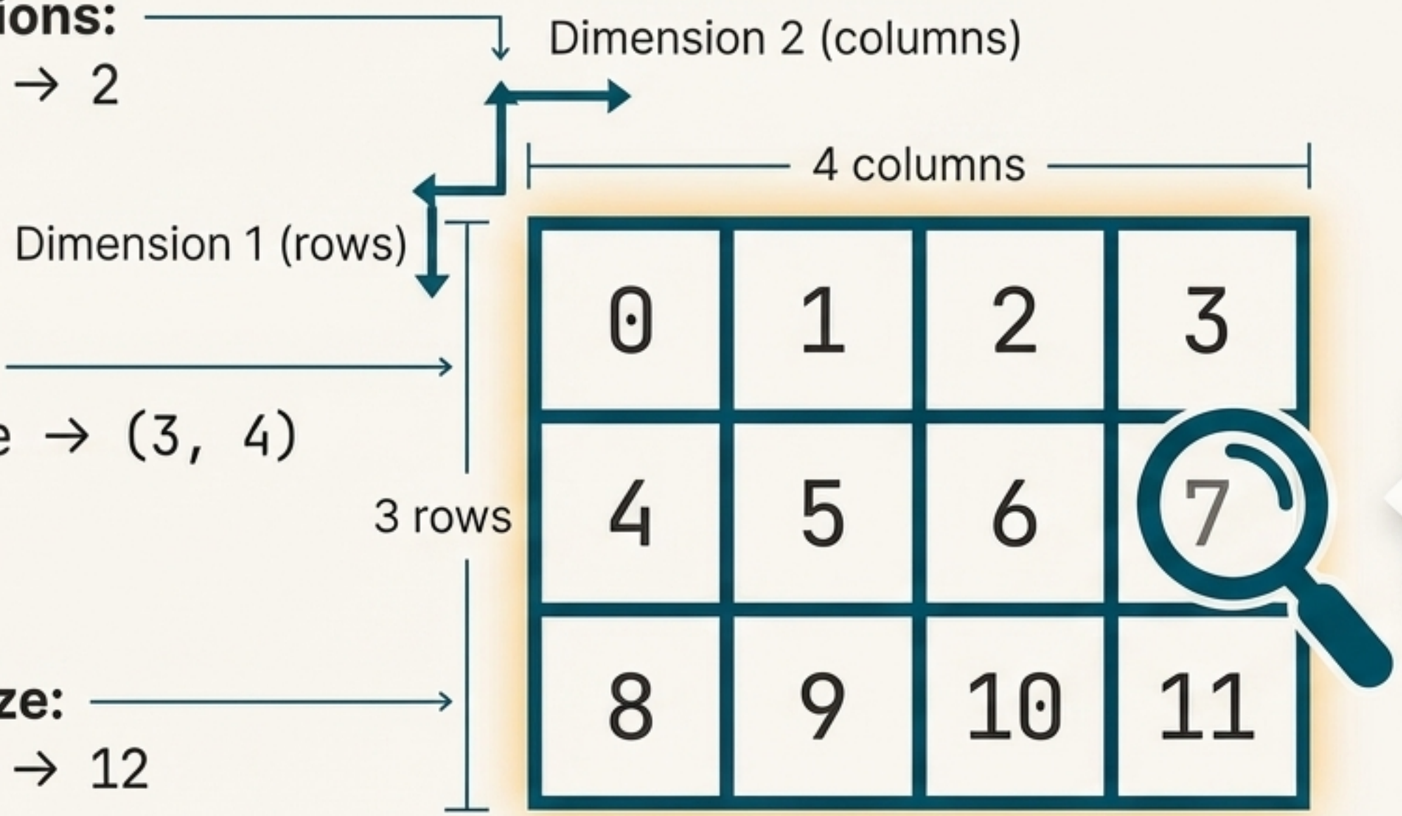
```
np.linspace(-10, 10, 5)
```



Inspecting Your Array: Key Attributes

`a = np.arange(12).reshape(3, 4)`

- **Dimensions:**
  `a.ndim → 2`
- **Shape:**
  `a.shape → (3, 4)`
- **Total Size:**
  `a.size → 12`



- **Data Type:**
  `a.dtype`
  `→ dtype('int64')`

int64

8 bytes

- **Item Size:**
  `a.itemsize → 8`

Accessing Data: Indexing and Slicing

1. 1D & 2D Indexing

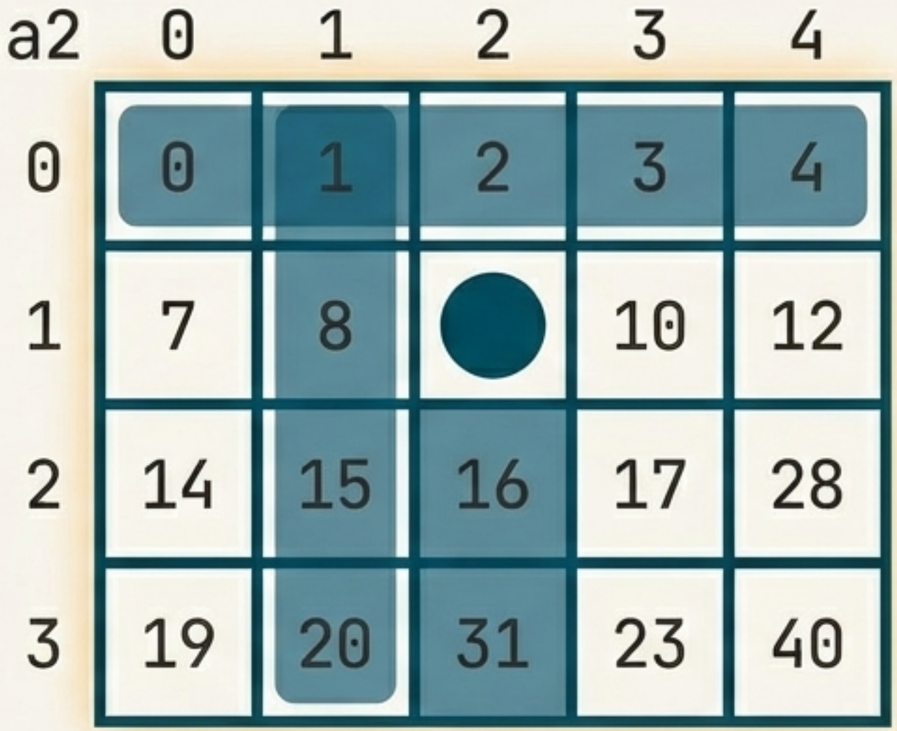
- **1D Indexing:**
  Same as Python lists.
  `a1[5], `a1[-1]`

- **2D Indexing:**
  Use comma-separated indices.
  `arr[row, col]`

  `a2[1, 2]`

2. Slicing Rows & Columns

`a2[0, :]`



`a2[:, 1]`

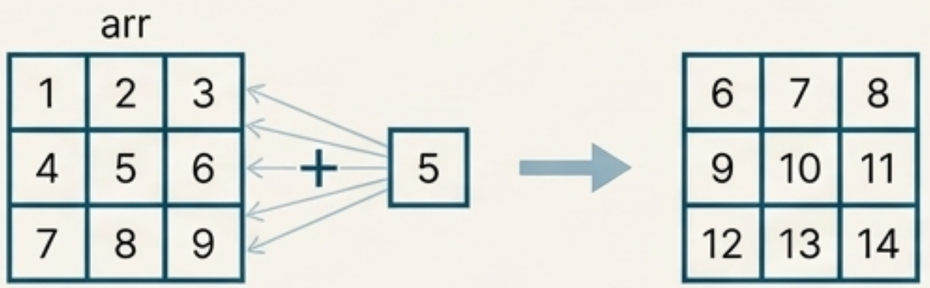
3. Slicing a Sub-matrix

`a2[1:, 1:3]`

0	1	2	3	4
1	1	8	12	14
2	15	16	17	28
3	14	28	32	40

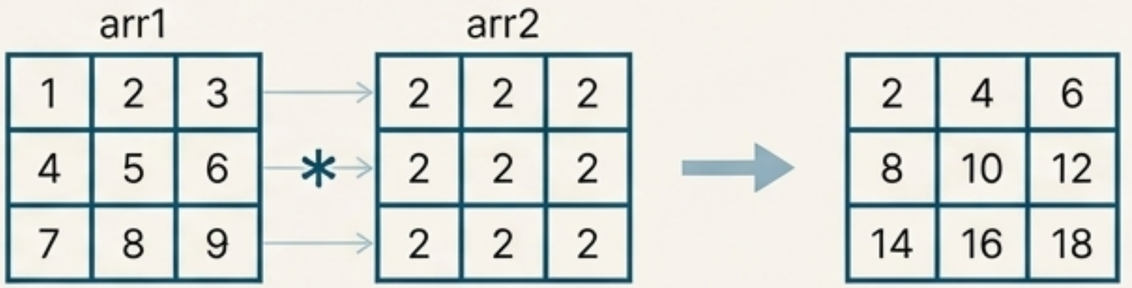
Universal Functions: Effortless Math

1. **Scalar Operations:** Array and a single number.



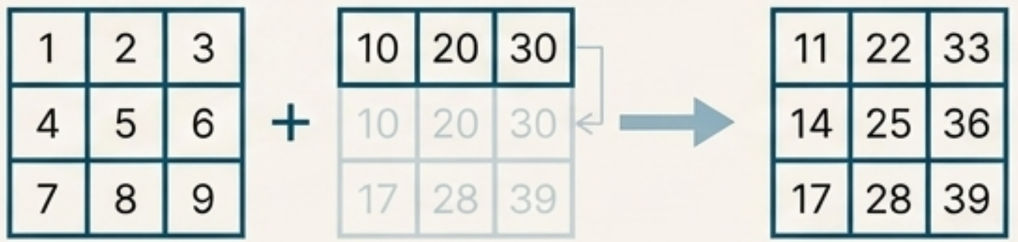

```
arr + 5
```

2. **Vector Operations:** Between two arrays of the same shape.

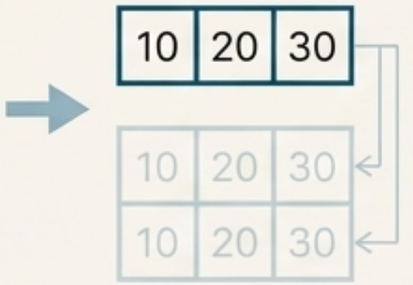



```
arr1 * arr2
```

3. **Broadcasting:** Operations on arrays of different (but compatible) shapes.




```
arr1 + arr2
```

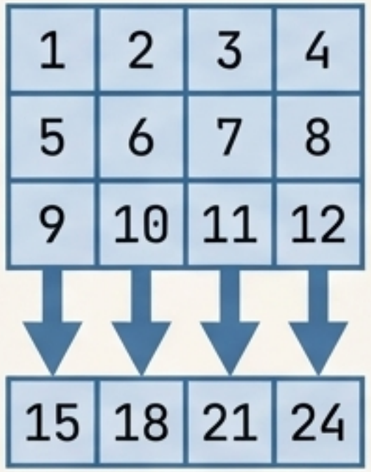


Smaller array is “stretched” to match the shape of the larger one without using more memory.

Aggregations: Summarizing Your Data

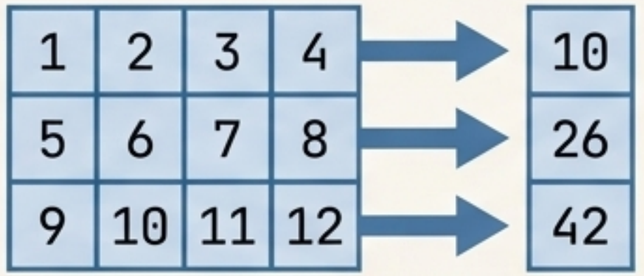
- Functions that perform a calculation on an array and return a single value or a smaller array.
- Common Functions: `np.max()`, `np.min()`, `np.sum()`, `np.mean()`, `np.std()`
- The **`axis`** Parameter: The key to targeted aggregations.

`arr.sum(axis=0)`



Sums down the columns
(collapses the rows)

`arr.sum(axis=1)`

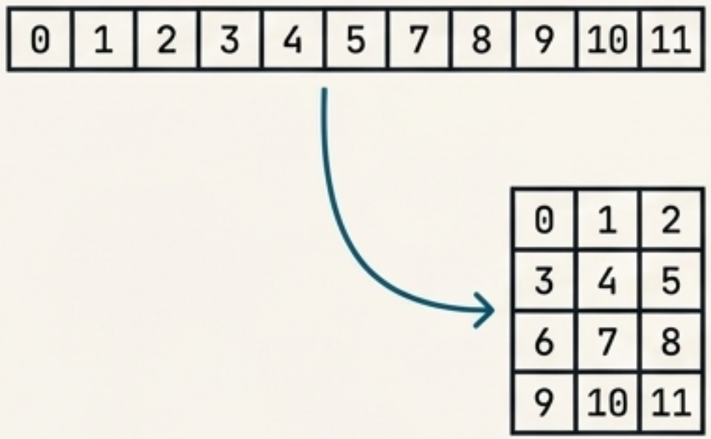


Sums across the rows
(collapses the columns)

Shaping and Combining Arrays

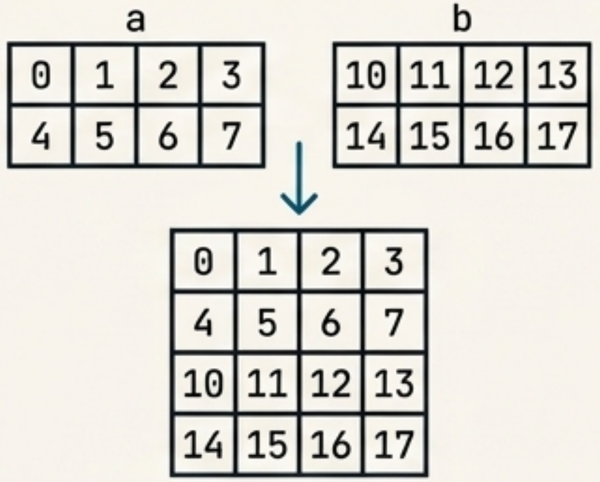
Reshaping: Change the shape without changing the data.

```
a.reshape(4, 3)
```

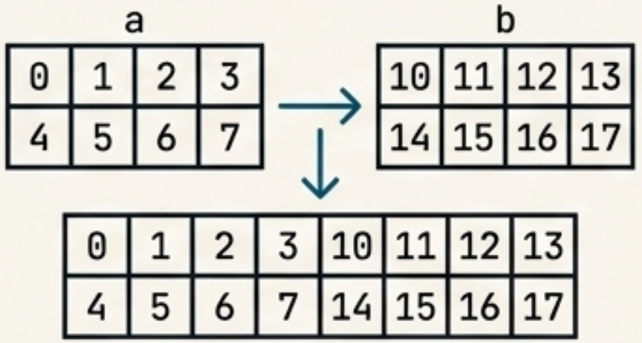


Stacking: Combine multiple arrays into one.

```
np.vstack((a, b))
```

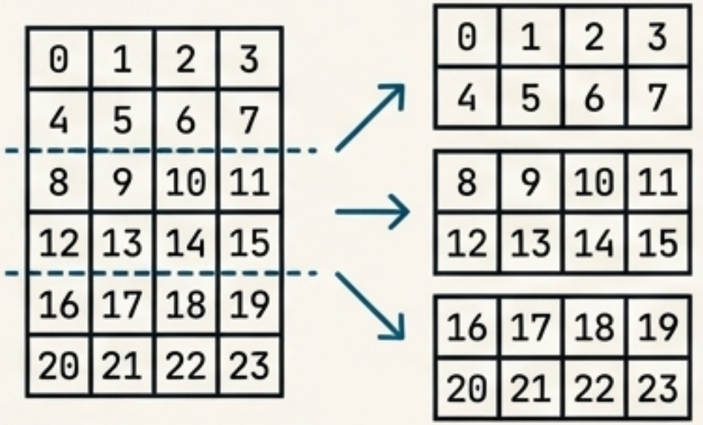



```
np.hstack((a, b))
```

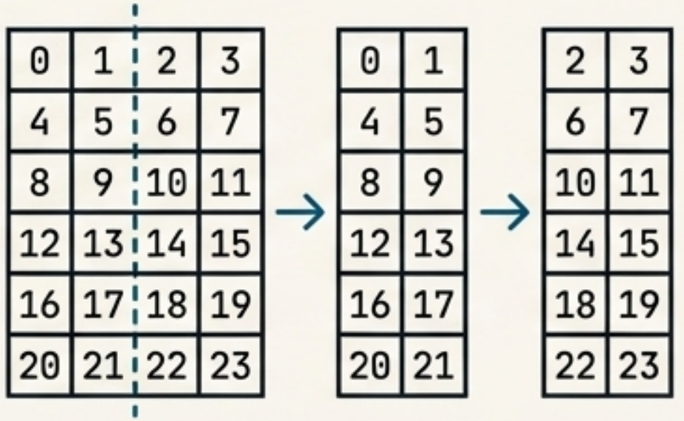


Splitting: Break one array into smaller arrays.

```
np.vsplit(a, 3)
```




```
np.hsplit(a, 2)
```



NumPy: The Foundation You'll Build Upon

- **Why NumPy?** It solves Python's slowness for numerical tasks with contiguous, homogeneous arrays.
- **The Core:** The `ndarray` object enables vectorized, SIMD-powered operations.
- **The Power:** An elegant syntax for complex math, aggregations, and manipulations.
- **The Ecosystem:** Mastering NumPy is the first true step to becoming a professional data scientist.