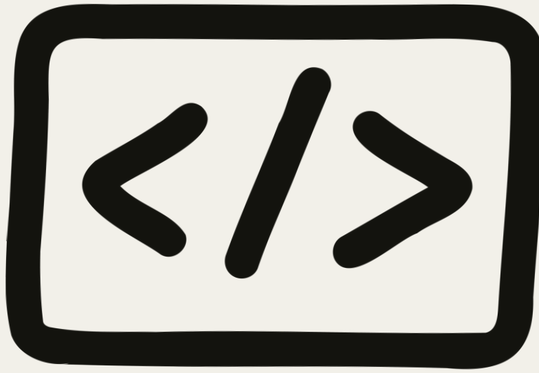


NumPy Review

لاتنسونا من دعواتكم

الله يوفقنا ويوفقكم وبإذن الله نتقابل في الصيفية 🤲🏻

لو عندكم اي سؤال او اقتراح
هذي حسابتنا في X

Muath Sara Ice

Why NumPy? Beyond Standard Python Lists

Slow & Verbose

```
# Adding elements of two lists
list_a = [1, 2, 3]
list_b = [4, 5, 6]
result = []
for i in range(len(list_a)):
    result.append(list_a[i] + list_b[i])
```

- Requires explicit, slow loops for element-wise operations.
- Inefficient memory usage for numerical data.
- Lacks a rich library of mathematical functions.

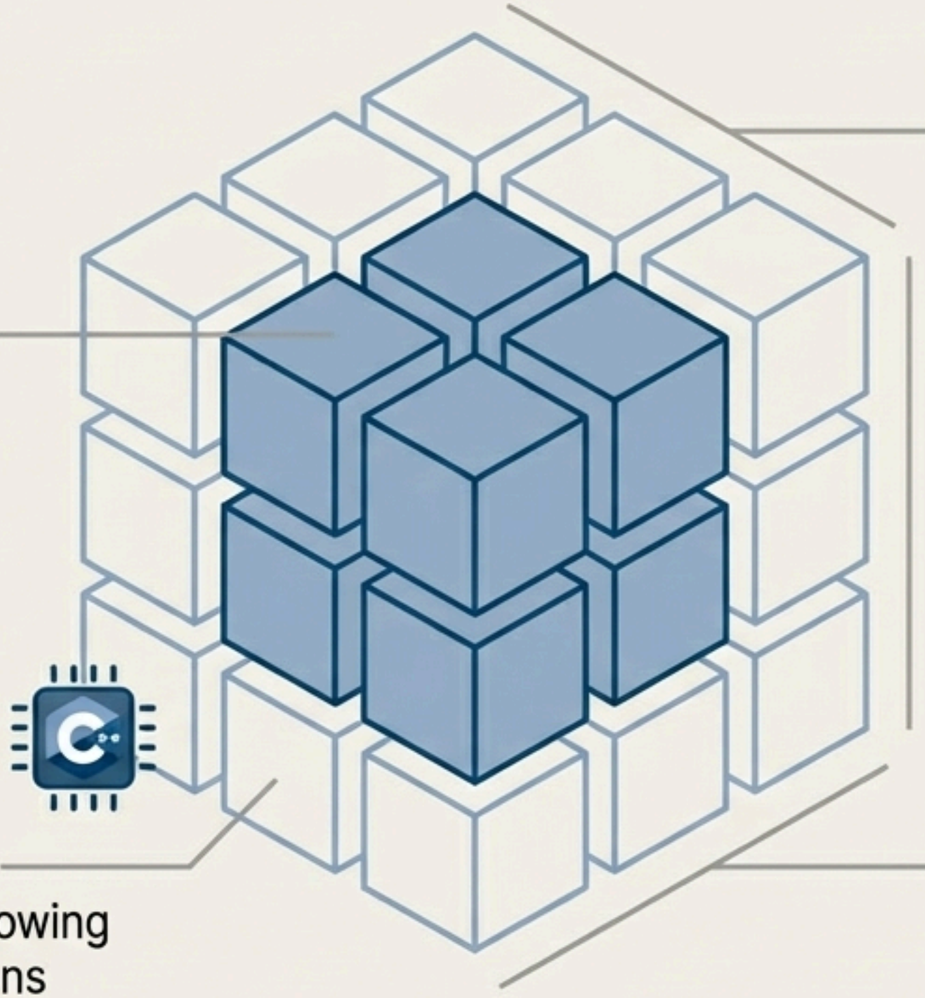
Fast & Concise

```
# Adding elements of two NumPy arrays
import numpy as np
array_a = np.array([1, 2, 3])
array_b = np.array([4, 5, 6])
result = array_a + array_b
```

- Performs vectorized operations using highly optimized C code.
- Stores data in a contiguous block of memory for performance.
- The foundation for Pandas, Scikit-learn, and the entire data science ecosystem.

The Core Object: The N-Dimensional Array (`ndarray`)

Homogeneous: All elements must be the same data type (e.g., `int64`, `float64`), ensuring memory efficiency. This is a key difference from Python lists.



N-Dimensional: A grid of values that can have any number of dimensions.

High-Performance: Points to a contiguous block of memory, allowing for optimized C/Fortran operations without Python's overhead.

Fixed-Size: The size of an array is determined at creation.

Array Creation I: From Existing Data & Placeholders

From a List

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

1	2	3	4

Zeros

```
np.zeros((2, 3))
```

shape: 2 rows, 3 columns

0.	0.	0.
0.	0.	0.

Ones

```
np.ones((2, 3))
```

1.	1.	1.
1.	1.	1.

Identity Matrix

```
np.eye(3)
```

1.	0.	0.
0.	1.	0.
0.	0.	1.

A cornerstone of linear algebra (مصفوفه الوحده)

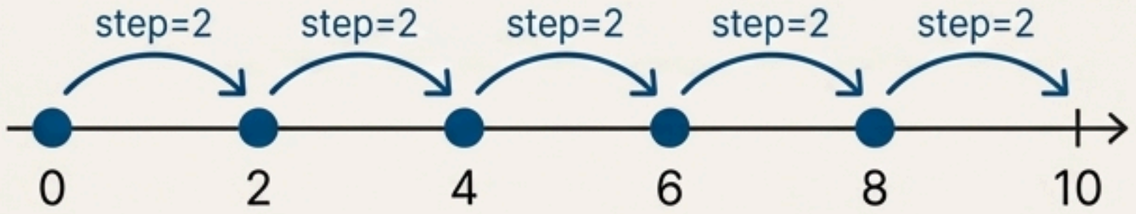
Array Creation II: Generating Numerical Sequences

np.arange()

Create values with a defined *step size*.

```
np.arange(start, stop, step)
```

```
np.arange(0, 10, 2)
```




```
[0 2 4 6 8]
```

np.linspace()

Create a specific *number* of evenly spaced values.

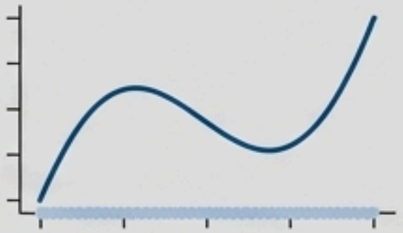
```
np.linspace(start, stop, num_points)
```

```
np.linspace(0, 1, 5)
```


```
[0. 0.25 0.5 0.75 1.]
```

Pro-Tip: Plotting

Essential for generating X-coordinates for plotting functions. To draw a smooth curve, you might generate 1000 points between your min and max X-values using `linspace`.



A Critical Best Practice: Ensuring Reproducibility with `np.random.seed()`

Pseudo-random number generators are deterministic. A 'seed' initializes the generator. Using the same seed guarantees the same sequence of 'random' numbers, which is essential for debugging, sharing results, and scientific validity.

Without a Seed

```
# Run 1
np.random.rand(3)
```

→ `0.37` `0.95` `0.73`

```
# Run 2
np.random.rand(3)
```

→ `0.59` `0.15` `0.05`

≠ Results are different each time.

With `np.random.seed(42)`

```
# Run 1
np.random.seed(42)
np.random.rand(3)
```

→ `0.37` `0.95` `0.73`

```
# Run 2
np.random.seed(42)
np.random.rand(3)
```

→ `0.37` `0.95` `0.73`

= Results are identical every time.

Knowing and Shaping Your Array

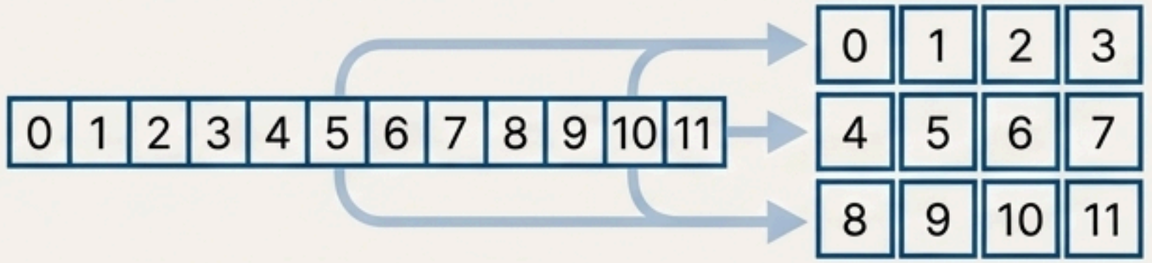
```
arr = np.arange(12).reshape(3, 4)
```

Inspection

`arr.shape` →	`(3, 4)`	The dimensions
`arr.ndim` →	`2`	The number of dimensions
`arr.size` →	`12`	The total number of elements
`arr.dtype` →	`int64`	The data type of the elements

Reshaping

`arr.reshape(new_shape)` changes the shape without changing the data.




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

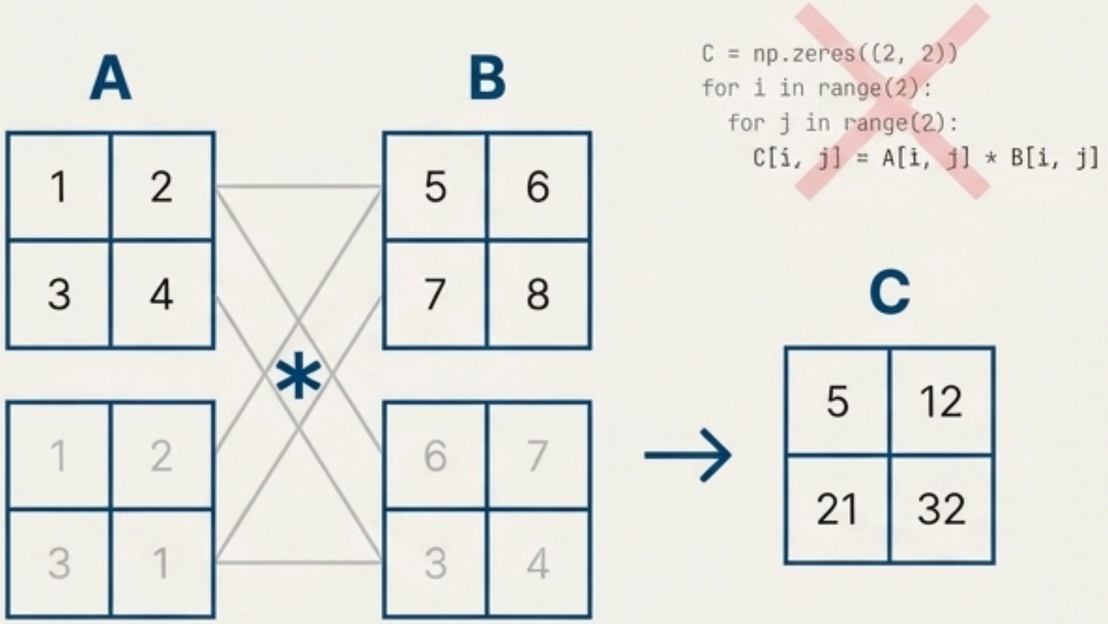
Pro-Tip: Infer Dimension

Use -1 to have NumPy automatically infer a dimension. `a.reshape(6, -1)` will create a 6x2 array.

NumPy’s Superpowers: Vectorization and Broadcasting

Vectorization

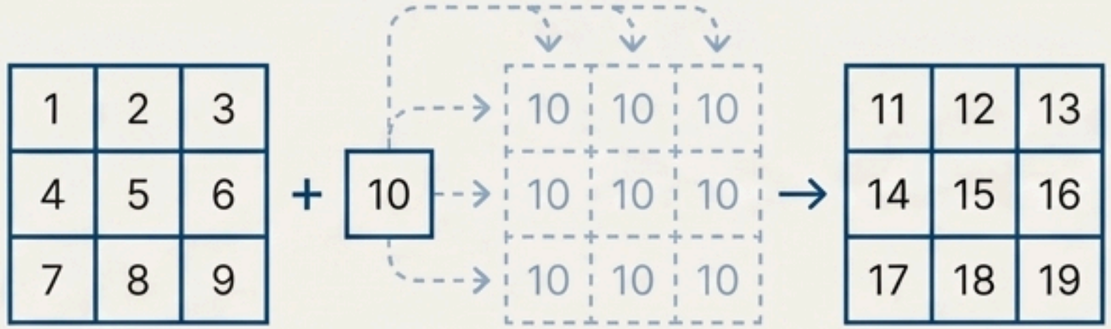
Operations are applied element-by-element without writing explicit Python loops.




```
C = np.zeres((2, 2))
for i in range(2):
  for j in range(2):
    C[i, j] = A[i, j] * B[i, j]
```

Broadcasting

NumPy’s ability to stretch smaller arrays to match the shape of larger arrays for element-wise operations.



Broadcasting works when dimensions are compatible (either equal, or one of them is 1).

Accessing Data with Precision: Multi-Dimensional Slicing

`array[row_slice, column_slice]`

Select a single element

`arr[0, 1]`

Select a single row

`arr[2, :]`

Select a single column

`arr[:, 3]`

Select a sub-grid

`arr[1:4, 0:2]`

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19
4	20	21	22	23	24

Aggregating Data Across Dimensions with `axis`

*"Imagine an array where rows are **students** and columns are **exam scores**."*

grades

Students:	Exams: 0	1	2	3
0	85	92	78	90
1	76	88	80	85
2	90	82	89	93

Aggregate over the entire array

`grades.sum()`

1018

85	92	78	90
76	88	80	85
90	82	89	93

Aggregate down the columns (`axis=0`)

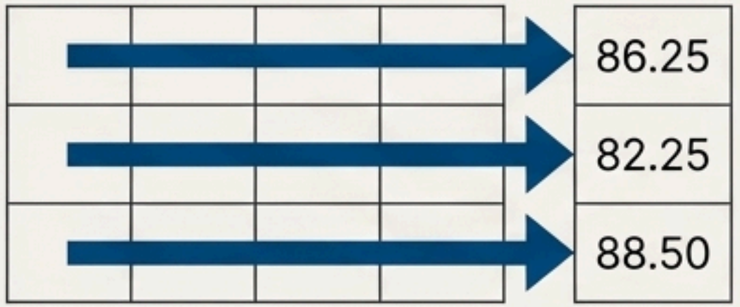
`grades.mean(axis=0)`

Calculates the average score for *each exam* (column).

Aggregate across the rows (`axis=1`)

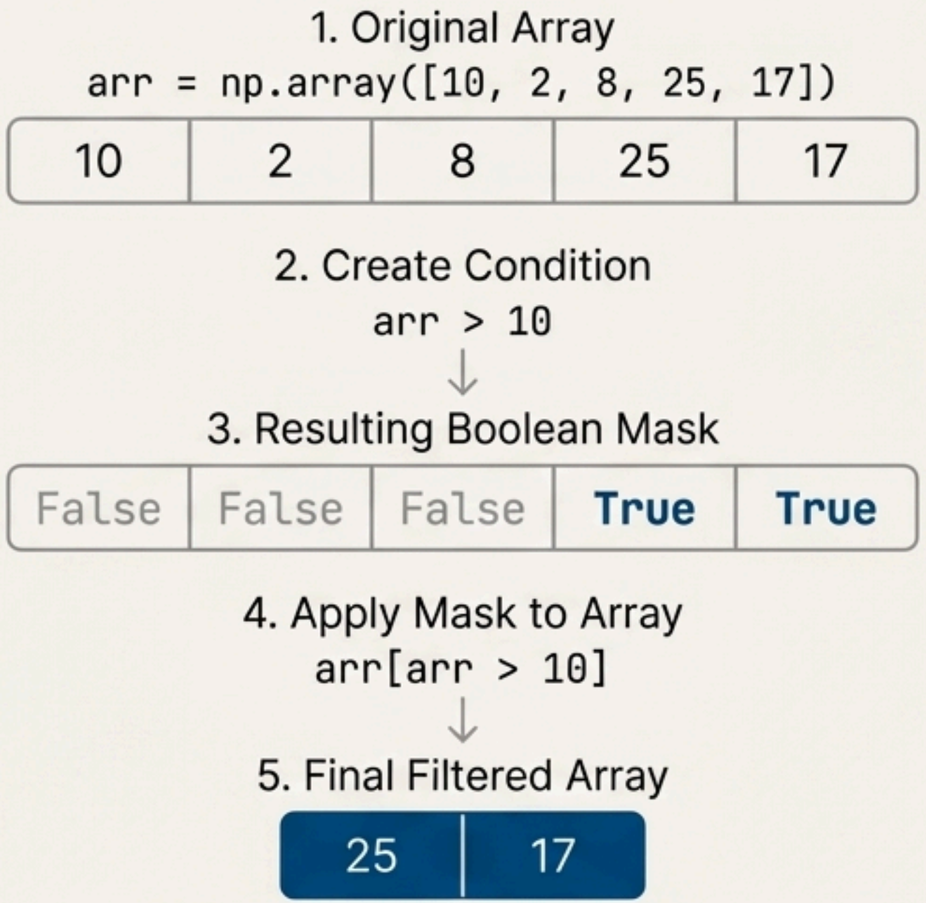
`grades.mean(axis=1)`

Calculates the average score for *each student* (row).



Advanced Selection: Filtering with Boolean Masking

Use a conditional expression inside indexing brackets to create a temporary boolean array (a "mask"). Only elements corresponding to `True` are returned.

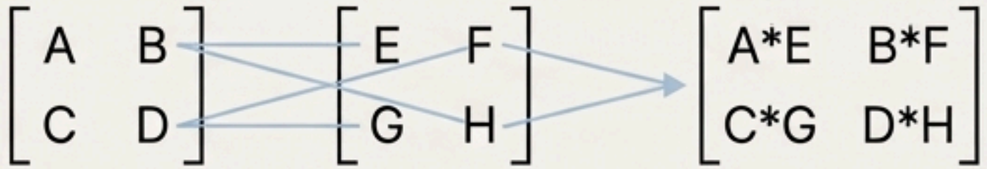


This is a highly efficient way to select data based on complex conditions without writing loops.

A Tale of Two Multiplications: Element-wise vs. Matrix

Element-wise Product

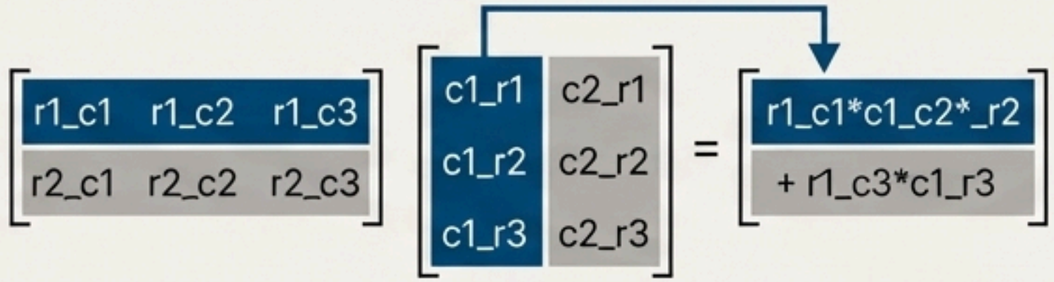
Multiplies corresponding elements in two arrays of the same shape. This is the default behavior.



Arrays must have the **same shape** (or be broadcastable).

Matrix Product

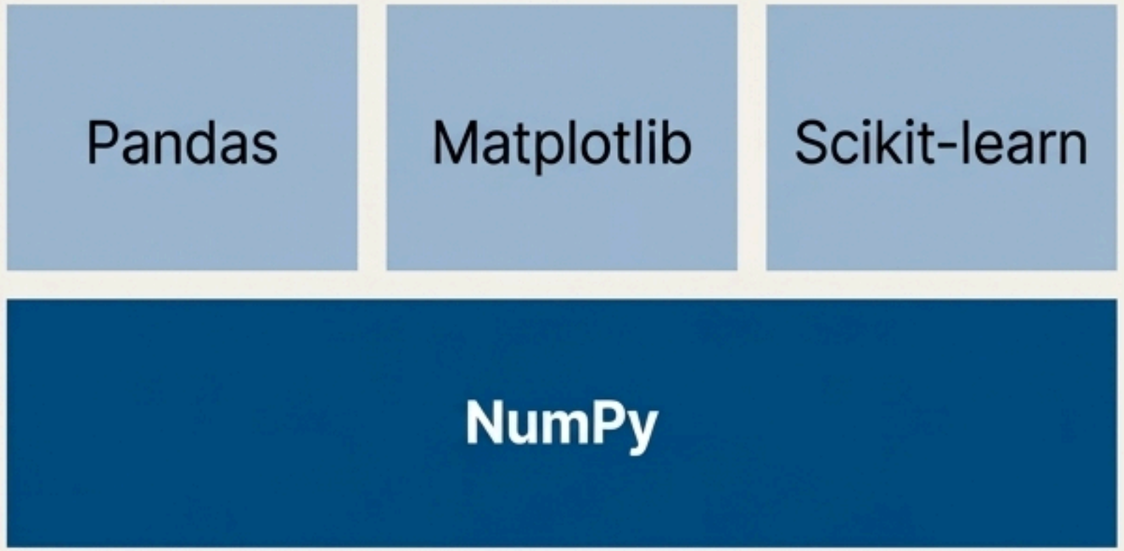
Performs the dot product (true linear algebra matrix multiplication).



The inner dimensions must match (e.g., **(M, K) @ (K, N)`**). The matching **K** dimensions are highlighted.

NumPy: The Bedrock of the Data Science Stack

- The `ndarray` is the high-performance heart of numerical computing in Python.
- Vectorized operations and broadcasting provide unmatched speed and code elegance, eliminating slow Python loops.
- Mastery of multi-dimensional slicing and `axis`-based aggregations is key to effective data manipulation.



Understanding NumPy is the first and most critical step in mastering data science in Python. It is the language that all other data libraries speak.