NumPy Basics: Numbers in Arrays

A NumPy array holds numbers in one fast block, so maths on a whole column takes one line. Arrays versus lists, mean, sum and max, and operations on every value at once.

Introduction

A Python list can hold anything — numbers, words, mixed types — and that freedom has a cost. When you want to add 5 to every mark in a list, you must loop through it yourself, one value at a time. For ten marks that is fine. For ten thousand, it is slow and clumsy.

NumPy fixes this with one idea: the **array**. An array holds numbers of one type, packed together in a single block. Because the numbers sit together and share a type, NumPy can do maths on the whole array at once, in fast code underneath Python. One line adds 5 to every mark. One line finds the average. That is the whole reason data scientists use NumPy: not new maths, but the same maths, much faster and much shorter.

This lesson uses only small arrays you can check by hand, so you can trust every result before you trust the tool.

So far every array here has been one row of numbers — a single column of marks. Real data is often a table, and NumPy handles that too: a **2D array** has rows and columns, like a small spreadsheet. A class's marks in two subjects could be `[[85, 90], [70, 80], [95, 88]]` — three students, two subjects. Its shape is (3, 2): 3 rows, 2 columns. Almost everything in this lesson works on 2D arrays as well, with one extra choice: do you want the average across each row, or down each column? That choice is the `axis` argument, and Example 4 shows it.

(Roman Urdu: List mein har cheez alag hoti hai, is liye hisaab dheema hota hai. NumPy ka array tamam numbers ko ek saath rakhta hai, to poori column par hisaab ek line mein ho jata hai.)

Key points

- **An array is a list built for numbers.** All values share one type, and they sit together in memory.
- `np.array()` turns a Python list into an array: `np.array([85, 92, 74])`.
- **Arrays are faster for maths** because the loop happens in fast, low-level code instead of in Python — the bigger the data, the bigger the win.
- **One operation hits every value:** `marks + 5` adds 5 to each mark, with no loop written by you.
- `np.mean()`, `np.sum()`, `np.max()`, `np.min()` give average, total, highest and lowest in one call each.
- **A column from pandas is already array-like** — `df["marks"].values` gives you the NumPy array behind a DataFrame column, so the two libraries work together.
- **Arrays have a shape and a type:** `arr.shape` gives the size, `arr.dtype` gives the number type, such as whole numbers or decimals.

- **A 2D array is a table of numbers.** `np.array([[85, 90], [70, 80]])` has shape (2, 2): 2 rows, 2 columns.
- **`axis=0` works down each column, `axis=1` works across each row.** `np.mean(table, axis=0)` gives one average per subject; `axis=1` gives one average per student.
- **`arr.reshape()`** changes the shape without changing the numbers — for example, 6 numbers as (2, 3) instead of a flat row of 6.

Examples

Example 1 — A list versus an array

Plain Python first — a loop for a simple job:

```
marks = [85, 92, 74, 88, 61]
bonus = []
for m in marks:
    bonus.append(m + 5)
print(bonus)  # [90, 97, 79, 93, 66]
```

Now the same job with NumPy. This example needs `pip install numpy` first:

```
import numpy as np

marks = np.array([85, 92, 74, 88, 61])
print(marks + 5)   # [90 97 79 93 66]
print(marks * 2)   # [170 184 148 176 122]
```

What this means: no loop, no new list built by hand — the `+ 5` applies to every value at once. Both versions give the same answers, but the array version stays one line no matter how many marks you add.

Example 2 — Mean, sum, max and min in four calls

This example needs `pip install numpy` first:

```
import numpy as np

marks = np.array([85, 92, 74, 88, 61])

print("Average:", np.mean(marks))  # 80.0
print("Total:", np.sum(marks))     # 400
print("Highest:", np.max(marks))   # 92
print("Lowest:", np.min(marks))    # 61
print("How many:", marks.size)     # 5
```

What this means: four questions about a column, four short calls, and every answer matches what you would get by hand. `size` tells you how many values there are — useful when you later divide a total yourself.

Example 3 — Work on a whole column at once, with a condition

This example needs `pip install numpy` first:

```
import numpy as np

marks = np.array([85, 92, 74, 88, 61])

passed = marks >= 70
print(passed)                  # [ True  True  True  True False]
print("Passed:", np.sum(passed))  # 4 (True counts as 1)
print(marks[passed])           # [85 92 74 88] — only passing marks
print(np.mean(marks[passed]))  # 84.75 — average of passing marks only
```

What this means: `marks >= 70` does not give one answer — it gives one true/false answer per value. That true/false array then picks out just the passing marks, and the average of those is 84.75. This pattern — test the whole array, then use the result to pick values — is the core trick of NumPy, and it is the same idea as filtering rows in pandas.

Example 4 — A 2D table: averages per subject and per student

This example needs `pip install numpy` first:

```
import numpy as np

# rows = students, columns = Maths, English
table = np.array([[85, 90],
                  [70, 80],
                  [95, 88]])

print("Shape:", table.shape)                    # (3, 2)
print("Per subject:", np.mean(table, axis=0))   # [83.33 86.  ]
print("Per student:", np.mean(table, axis=1))   # [87.5 75.  91.5]
print("Whole table:", np.mean(table))           # 84.67
```

What this means: `axis=0` collapses the rows, leaving one number per column — English (86.0) beats Maths (83.33) overall. `axis=1` collapses the columns, leaving one number per row — the third student's average, 91.5, is the highest. With no axis at all, you get one average for the whole table, 84.67. Check one by hand: the Maths column is (85 + 70 + 95) / 3 = 83.33. The axis rule to remember: axis=0 goes down, axis=1 goes across.

Example 5 — Turning raw scores into percentages

This example needs `pip install numpy` first:

```
import numpy as np

out_of_50 = np.array([42, 45, 38, 47, 30])
percent = out_of_50 / 50 * 100
print(percent)                    # [84. 90. 76. 94. 60.]
print("Above 80%:", np.sum(percent >= 80))  # 3
print("Rounded:", np.round(percent))       # same here, no decimals
```

What this means: the same whole-array maths converts every score at once — divide, multiply, compare, count. Three students cleared 80%. Notice the order: divide first, then multiply, exactly as in ordinary maths, and NumPy applies each step to every value before moving on.

Example 6 — Real-world case: a weather station's thousand readings

A weather station records the temperature every minute — over a thousand readings a day. Nobody loops through those by hand. The station's software stores them in an array, then one line gives the day's average, one line the highest, one line how many readings crossed 40 degrees. Your phone does the same with its sensors: thousands of step and movement readings per day, summarised by array maths into the single numbers you see on screen. NumPy's speed matters exactly here — when the data is far too big to check value by value, the one-line operation is the only practical way.

(Roman Urdu: Mausam station ki hazaron readings bhi array mein hi sambhali jati hain — ek line average, ek line sab se zyada, bas.)

Quick recap

- An array holds numbers of one type, packed together.
- `np.array()` builds one from a list.
- Maths on an array hits every value at once — no loop needed.
- `mean`, `sum`, `max`, `min` summarise a column in one call each.
- A true/false test on an array gives a true/false answer per value, which can pick values out.

Common mistakes

1. **Mixing text into a number array** — `np.array([85, "Sara"])` turns everything into text, and then maths on it fails. Fix: keep arrays to one type; put names in a separate array or a pandas column.
2. **Forgetting that `marks + 5` makes a new array** — the original `marks` is unchanged. Fix: save the result if you need it: `bonus = marks + 5`.
3. **Using a Python loop where an array operation would do** — it works, but it is slower and longer. Fix: try the array version first; keep the loop only when each value truly needs a different rule.
4. **Mixing shapes that do not match** — adding an array of 5 numbers to an array of 3 fails, because NumPy cannot pair the values up. Fix: check both shapes first; if they differ, find out why before forcing them together.

5. **Taking the mean of a 2D array and expecting per-column answers** — `np.mean(table)` with no axis gives one number for the whole table, not one per subject. Fix: say which axis you mean: `axis=0` for per-column, `axis=1` for per-row.

Practice

1. Make an array of 6 marks of your choice. Print its mean, sum, highest and lowest values, and check the mean by hand.
2. Add 10 bonus marks to the whole array in one line, and print both the old and the new array.
3. Use a true/false test to pick out only the marks above 80, then find their average.
4. Make a plain-Python list with the same marks and compute the average with a loop. Compare the two answers — they should match.

1. Make a 2D array of 3 students and 2 subjects of your choice. Print the per-subject averages (axis=0) and the per-student averages (axis=1), and check one of each by hand.
2. Take marks out of 40 for 5 students, convert them all to percentages in one line, and count how many are above 75%.
3. Make an array of daily temperatures for a week (7 numbers). Find the average, the hottest day's value, and how many days were above 35 degrees — all with array operations, no loops.

Practice quiz

1. What is a NumPy array?

- A list of words
- A grid of same-type numbers that does maths on the whole column at once
- A chart type
- A pandas table

Answer: A grid of same-type numbers that does maths on the whole column at once

2. What does np.mean() calculate?

- The biggest value
- The average of the numbers
- The count of numbers
- The middle value

Answer: The average of the numbers

3. Why are arrays handy for maths on data?

- They use less typing
- One operation works on every value at once, instead of looping by hand
- They never have errors

- They only hold integers

Answer: One operation works on every value at once, instead of looping by hand

---

Watch the video lesson: https://youtu.be/T2KpPBIqOxs

Read online: https://scopetreasure.com/notes/ds-numpy-basics · More free tools and notes at https://scopetreasure.com