

A NumPy array is a grid of numbers of one type that you can do maths on all at once, with no loop. Python has no built-in array type, so when people say "array" in Python, they usually mean a NumPy array. For plain Python lists, see the lists cheat sheet. Every section starts with `import numpy as np`.

CREATE

```
import numpy as np
```

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

an array made from a Python list

```
np.zeros(3) → array([0., 0., 0.])
```

floats by default: 0. means 0.0

```
np.ones((2, 2), dtype=int)
```

```
→ array([[1, 1],
```

```
        [1, 1]])
```

the shape (rows, columns) goes in brackets

```
np.arange(0, 10, 3) → array([0, 3, 6, 9])
```

works like range(): 10 is not included

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

```
→ array([0. , 0.25, 0.5 , 0.75, 1.  ])
```

5 evenly spaced values from 0 to 1; 1 is included

SHAPE AND TYPE

```
import numpy as np
```

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

```
m.shape → (2, 3)
```

the shape: 2 rows and 3 columns

```
m.ndim → 2
```

the number of dimensions: 2 (rows and columns)

```
m.size → 6
```

the number of items

```
np.array([0.5, 2]).dtype
```

```
→ dtype('float64')
```

the dtype (data type): the one type that every item has

INDEX AND SLICE

```
import numpy as np
```

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

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

```
m[0, 1] row 0, column 1: the value 2
```

```
m[1] → array([4, 5, 6])
```

row 1, the whole row

```
m[:, 0] → array([1, 4])
```

a column: all rows, column 0

```
m[:, 1:] → array([[2, 3],
```

```
                [5, 6]])
```

all rows; columns 1 to the end

```
v[[0, 2]] → array([10, 30])
```

a list of positions picks items 0 and 2

```
v[v > 15] → array([20, 30, 40])
```

v > 15 is a mask: one True or False per item. Keeps the True items

```
v[v > 25] = 0
```

```
v
```

```
→ array([10, 20, 0, 0])
```

sets every item above 25 to 0

MATHS ON EVERY ITEM

```
import numpy as np
```

```
prices = np.array([2.5, 4.0, 10.0])
```

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

```
celsius = np.array([20, 25, 30])
```

```
prices * qty → array([ 5.,  4., 30.])
```

multiplies item by item: no loop needed

```
prices + 1 → array([ 3.5,  5. , 11. ])
```

```
float((prices * qty).sum()) → 39.0
```

float() turns the NumPy number into a plain Python float

```
celsius * 9 / 5 + 32
```

```
→ array([68., 77., 86.])
```

converts every temperature to Fahrenheit at once

BROADCASTING

Broadcasting: when two shapes differ, NumPy repeats the smaller array to match the bigger one. This works only if the sizes line up from the right.

```
import numpy as np
```

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

```
m * 10
```

```
→ array([[10, 20, 30],
```

```
        [40, 50, 60]])
```

the one number is used with every item

```
m + np.array([100, 200, 300])
```

```
→ array([[101, 202, 303],
```

```
        [104, 205, 306]])
```

a row of 3 is added to each row

```
m + np.array([1, 2]) → raises ValueError
```

shapes (2, 3) and (2,) do not line up

CHANGE THE SHAPE

```
import numpy as np
```

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

```
v = np.arange(6)
```

```
v.reshape(2, 3)
```

```
→ array([[0, 1, 2],
```

```
        [3, 4, 5]])
```

the same 6 items, now in 2 rows of 3

```
v.reshape(3, -1).shape → (3, 2)
```

-1 means: NumPy works out this size

```
m.T.shape → (3, 2)
```

m.T is the transpose: rows become columns

```
m.flatten() → array([1, 2, 3, 4, 5, 6])
```

a copy with all items in one dimension

TOTALS AND AXIS

An axis is one direction of the grid. axis=0 goes down the rows: one answer per column. axis=1 goes across each row: one answer per row.

```
import numpy as np
```

```
scores = np.array([[70, 80, 90], # Ada
```

```
                  [60, 75, 90]]) # Grace
```

```
int(scores.sum()) → 465
```

all the scores added up

```
scores.sum(axis=0)
```

```
→ array([130, 155, 180])
```

a total for each test (each column)

```
scores.mean(axis=1) → array([80., 75.])
```

an average for each student (each row)

```
scores.max(axis=0) → array([70, 80, 90])
```

the best score on each test; min, std and cumsum also take axis

USEFUL FUNCTIONS

```
import numpy as np
```

```
t = np.array([18, 25, 31, 25, 12])
```

```
np.where(t > 24, t, 0)
```

```
→ array([ 0, 25, 31, 25,  0])
```

keeps t where t > 24 is True; puts 0 everywhere else

```
np.unique(t) → array([12, 18, 25, 31])
```

the values sorted, with no repeats

```
np.sort(t) → array([12, 18, 25, 25, 31])
```

a sorted copy; t is not changed

```
int(np.argmax(t)) → 2
```

the position of the biggest value

```
np.round(np.array([3.14159, 2.71828]), 2)
```

```
→ array([3.14, 2.72])
```

rounds every item to 2 decimal places

RANDOM NUMBERS

The seed (42) makes the generator give the same numbers on every run, so other people can repeat your results.

```
import numpy as np
```

```
rng = np.random.default_rng(42)
```

```
names = ["Ada", "Grace", "Linus", "Hedy"]
```

```
rng.integers(1, 7, size=5)
```

```
→ array([1, 5, 4, 3, 3])
```

5 dice rolls from 1 to 6: 7 is not included

```
rng.random(3).round(2)
```

```
→ array([0.7 , 0.09, 0.98])
```

3 floats from 0 up to, but not including, 1

```
rng.choice(names, size=2,
```

```
replace=False).tolist()
```

```
→ ['Linus', 'Hedy']
```

picks 2 different names

```
rng.normal(170, 10, size=3).round(1)
```

```
→ array([166.8, 169.8, 161.5])
```

heights in cm: average 170, typical spread 10

WATCH OUT

```
import numpy as np
```

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

```
s = v[:2]
```

```
s[0] = 99
```

```
v
```

```
→ array([99,  2,  3])
```

a slice is a view: it shares its data with v, so v changes too. Use v[:2].copy()

```
bool(v == 1) → raises ValueError
```

v == 1 gives one True or False per item, so bool() fails. Use (v == 1).any(), .all() or np.array_equal

```
v > 1 and v < 3 → raises ValueError
```

the word and does not work on arrays. Use & with brackets: (v > 1) & (v < 3)

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

```
w[0] = 9.7
```

```
w
```

```
→ array([9, 2])
```

an int array drops the decimals without a warning

```
[1, 2] * 2 → [1, 2, 1, 2]
```

a list repeats its items

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

an array multiplies every item