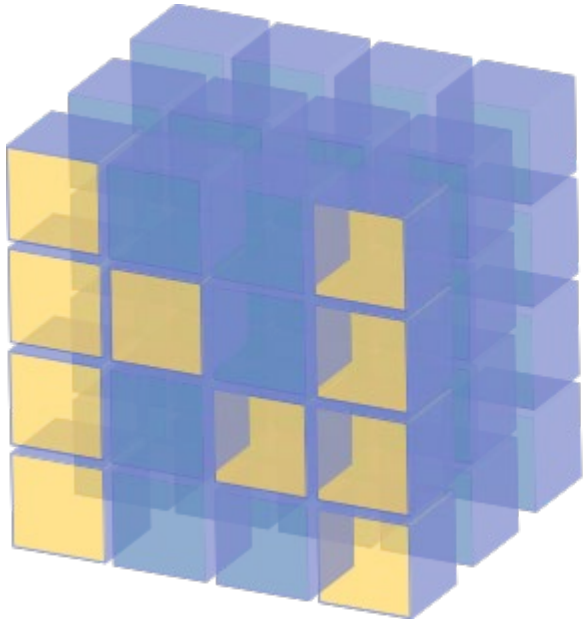
Information, Computation, Communication

Learning Python



NumPy

Agenda

- Why NumPy? Lists vs. arrays
- Creating arrays
  - Shape, size, and data type
- Indexing, slicing, views, and copies
- Axes, computation, and reshaping
- Broadcasting and practical examples



Why NumPy?

Think in arrays rather than individual elements.

Lists and arrays behave differently

- Can you predict the two outputs?

```
import numpy as np
a = [1, 2, 3]
b = np.array([1, 2, 3])
print(a * 2)
print(b * 2)
```

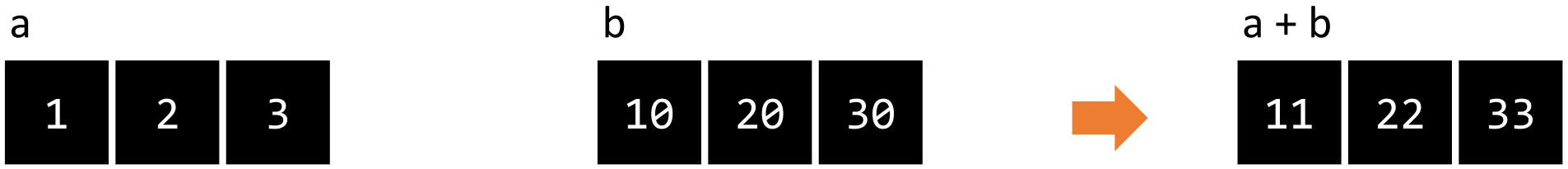
Lists and arrays behave differently

- Can you predict the two outputs?

```
import numpy as np
a = [1, 2, 3]
b = np.array([1, 2, 3])
print(a * 2) # [1, 2, 3, 1, 2, 3]
print(b * 2) # [2, 4, 6]
```

Element-wise Computation

```
a = np.array([1, 2, 3])
b = np.array([10, 20, 30])
print(a + b)  # [11 22 33]
print(a * b)  # [10 40 90]
```



What is NumPy?

- NumPy provides efficient N-dimensional arrays (**ndarray**)
- Arithmetic is performed element by element unless specified otherwise
- Arrays support indexing, slicing, statistics, and linear algebra
- The standard import is `import numpy as np`

One Array, One Data Type

- NumPy chooses a common **dtype** for compatible values

```
a = np.array([1, 2, 3])
print(a.dtype)  # integer dtype (platform dependent)
b = np.array([1, 2.5, 3])
print(b)  # [1.  2.5 3. ]
```



Creating and Describing Arrays

What does the shape of an array tell us?

Creating 1D and 2D Arrays

```
a = np.array([2, 2, 4, 4])
print(a)  # [2 2 4 4]
b = np.array([[3, 3, 5, 5],
              [2, 2, 4, 4],
              [0, 1, 0, 1]])
```

b: 3 rows, 4 columns

3	3	5	5
2	2	4	4
0	1	0	1

Size, Dimensions and Shape

- **size** = number of elements
- **ndim** = number of axes

```
print(b.size)   # 12
print(b.ndim)   # 2
print(b.shape)  # (3, 4)
```

b

3	3	5	5
2	2	4	4
0	1	0	1

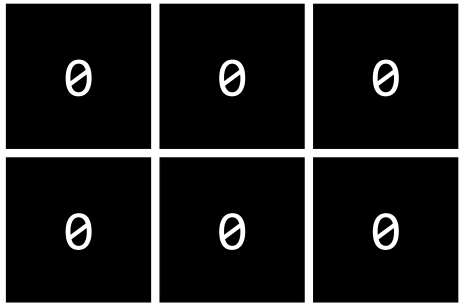
Predict the Shape

- How many rows, columns, and elements?

```
a = np.zeros((2, 3))
print(a.shape)
print(a.size)

# Answers
# (2, 3)
# 6
```

Two rows and three columns



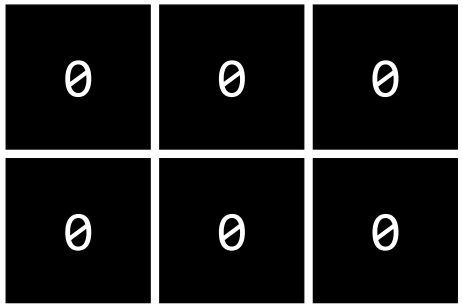
EXAMPLES

[Answer] Predict the Shape

- How many rows, columns, and elements?

```
a = np.zeros((2, 3))
print(a.shape) # (2, 3)
print(a.size)  # 6
```

Two rows and three columns



Create Arrays Quickly

- **zeros()** and **ones()** use floating-point values by default.

```
np.zeros((2, 3))     # 2-by-3 array of zeros
np.ones(4)           # [1. 1. 1. 1.]
np.full((2, 3), 9)   # 2-by-3 array of nines
np.arange(2, 10, 2) # [2 4 6 8]
```

arange() vs. linspace()

- **arange(start, stop, step)**: stop excluded
- **linspace(start, stop, num)**: endpoints included by default

```
np.arange(2, 10, 2)
# [2 4 6 8]
np.linspace(2, 10, 5)
# [ 2.  4.  6.  8. 10.]
```

Generating Random Numbers

- The values are pseudo-random floats in [0, 1).
- A fixed seed makes this sequence reproducible.
- The output shape is determined by (2, 3).

```
rng = np.random.default_rng(42)
a = rng.random((2, 3))
print(a.shape)  # (2, 3)
```



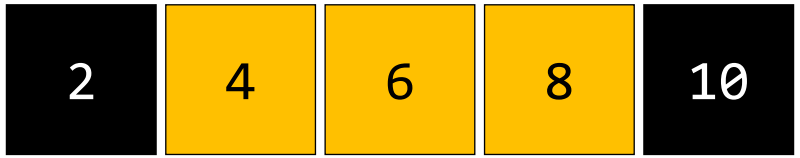
Indexing, Slicing, and Memory

Find data and know whether you copied it.

Indexing a 1D Array

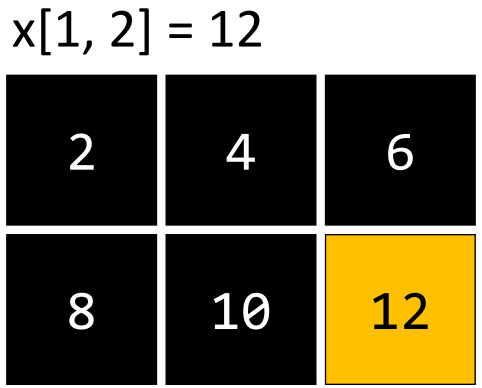
```
x = np.array([2, 4, 6, 8, 10])
print(x[0])     # 2
print(x[-1])    # 10
print(x[1:4])   # [4 6 8]
```

x[1:4] selects indices 1, 2, 3



Indexing a 2D Array

```
x = np.array([[2, 4, 6], [8, 10, 12]])
print(x[1, 2])  # 12
print(x[0, :])  # [2 4 6]
print(x[:, 1])  # [ 4 10]
```



Predict: View or Copy?

Does changing b also change a?

```
a = np.array([1, 2, 3])
b = a[1:]
b[0] = 9
print(a)
```

[Answer] Predict: View or Copy?

Does changing b also change a?

```
a = np.array([1, 2, 3])
b = a[1:]
b[0] = 9
print(a) # [1 9 3]
# Basic slicing returns a view: b shares data with a.
```

A View Shares the Original Data

```
x = np.array([[2, 4, 6], [8, 10, 12]])
print(x[1, 2])  # 12
print(x[0, :])  # [2 4 6]
print(x[:, 1])  # [ 4 10]
```

x: the view changes shared elements



Make an Independent Copy

```
x = np.array([2, 4, 6, 8, 10])
temp = x[2:4].copy()
temp[1] = 7
print(x)      # [ 2  4  6  8 10]
print(temp)   # [6 7]
```

x unchanged



independent temp



Boolean Indexing

```
a = np.array([3, 8, 2, 9, 5])
print(a > 5)
# [False  True False  True False]
print(a[a > 5])  # [8 9]
```



Basic Versus Advanced Indexing

```
x = np.arange(6)
a = x[1:4]      # basic slice: view
b = x[[1, 3]]   # integer indexing: copy
c = x[x > 3]    # Boolean indexing: copy
print(b)        # [1 3]
print(c)        # [4 5]
```

Shapes, Axes and Computation

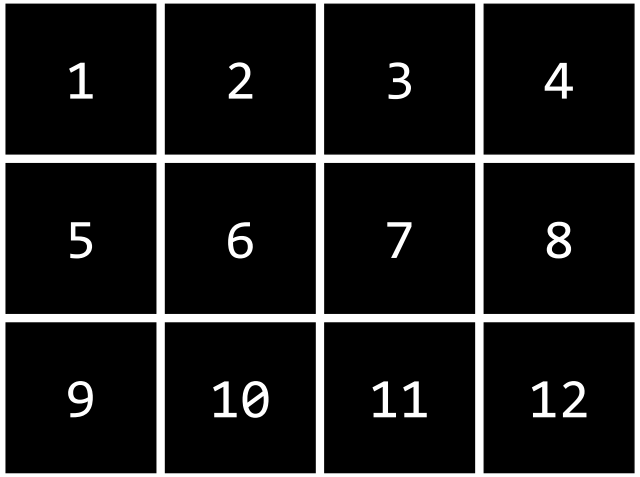
Use the array structure to express work concisely.

What is an Axis?

shape = (3, 4)

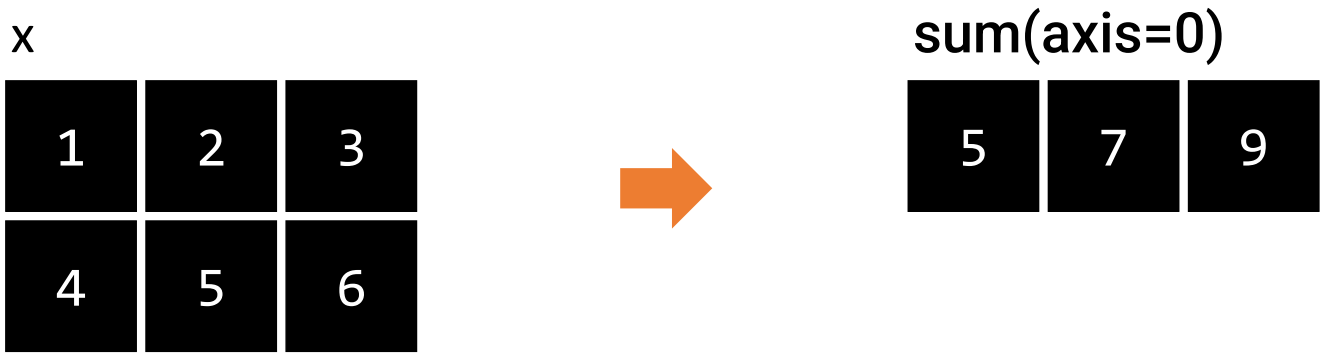
axis=0: down rows ↓

axis=1: across columns →



Sum Along an Axis

```
x = np.array([[1, 2, 3], [4, 5, 6]])
print(x.sum(axis=0))  # [5 7 9]
print(x.sum(axis=1))  # [ 6 15]
```



Predict a Reduction Shape

Which dimension disappears when summing?

```
x = np.ones((3, 4))
print(x.sum(axis=0).shape)
print(x.sum(axis=1).shape)
```

[Answer] Predict a Reduction Shape

Which dimension disappears when summing?

```
x = np.ones((3, 4))
print(x.sum(axis=0).shape) # (4,)
print(x.sum(axis=1).shape) # (3,)
```

- axis=0 sums down rows, leaving four columns.
- axis=1 sums across columns, leaving three rows.

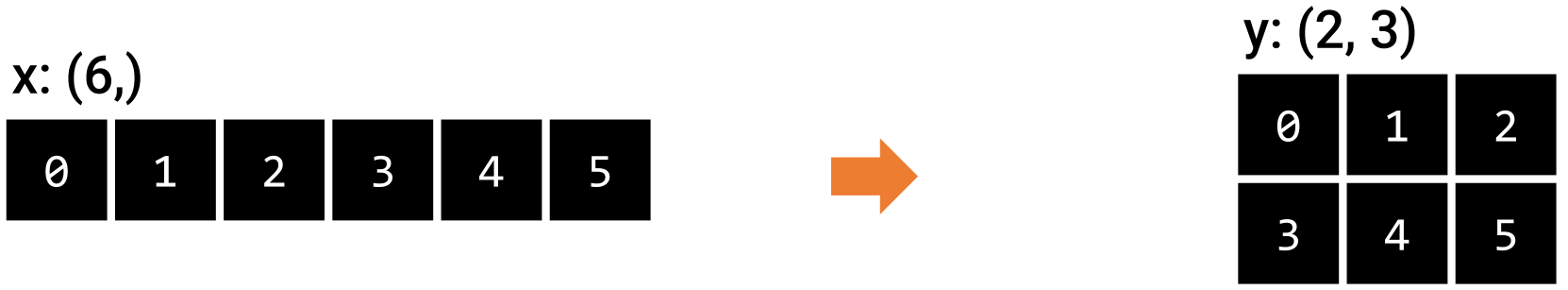
Some Useful Reductions

```
x = np.array([[1, 7, 4], [3, 1, 0]])
print(x.max(axis=1))  # [7 3]
print(x.min(axis=1))  # [1 0]
print(x.mean(axis=1))
# [4.         1.33333333]
```

Other functions include **std()**, **var()**, and **sum()**.

Reshape: Same Elements, New Arrangement

```
x = np.arange(6)
y = x.reshape(2, 3)
print(y)  # [[0 1 2]
          #  [3 4 5]]
print(y.shape)  # (2, 3)
```



Reshape with -1

- -1 tells NumPy to infer the missing dimension.
- The total number of elements must stay the same.
- Only one dimension may be specified as -1.

```
x = np.arange(12)
a = x.reshape(3, 4)
b = x.reshape(3, -1)
print(b.shape)  # (3, 4)
```

Transpose: Swap Rows and Columns

```
x = np.array([[1, 2, 3], [4, 5, 6]])
y = x.T
print(y.shape)  # (3, 2)
```

x: (2, 3)

1	2	3
4	5	6



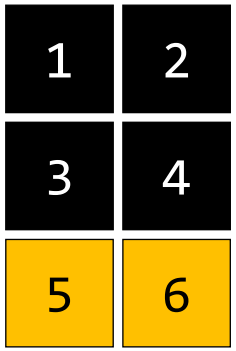
x.T: (3, 2)

1	4
2	5
3	6

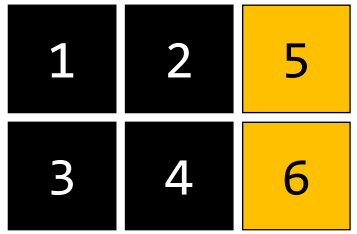
Joining Arrays by Rows or Columns

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
rows = np.concatenate((a, b), axis=0)
cols = np.concatenate((a, b.T), axis=1)
```

rows: (3, 2)



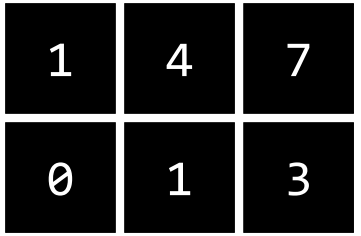
cols: (2, 3)



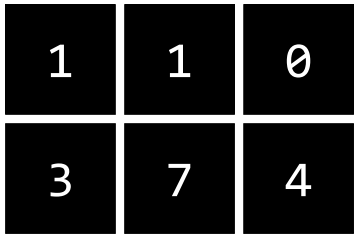
Sorting: Which Direction?

```
x = np.array([[1, 7, 4], [3, 1, 0]])
print(np.sort(x, axis=1))
# [[1 4 7]
#  [0 1 3]]
print(np.sort(x, axis=0))
# [[1 1 0]
#  [3 7 4]]
```

axis=1: sort rows



axis=0: sort columns

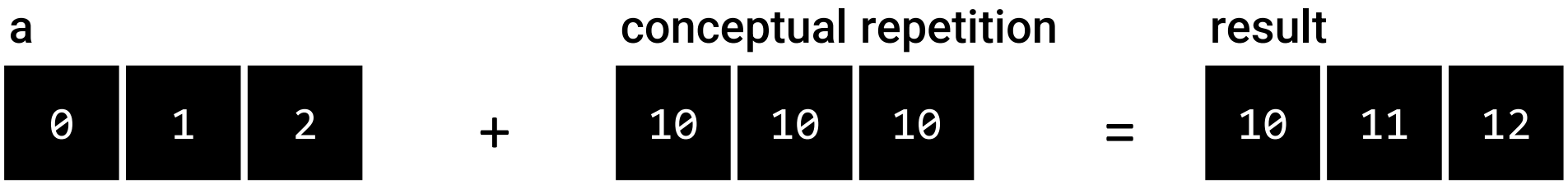


Broadcasting

Working with arrays of compatible shapes.

Broadcasting a Scalar

```
a = np.array([0, 1, 2])
print(a + 10)  # [10 11 12]
```



Broadcasting: Compare Shapes from the Right

Dimensions are compatible when equal or when one is 1.

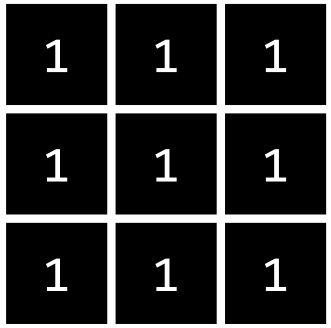
```
a = np.ones((3, 4))
b = np.arange(4)
print((a + b).shape)  # (3, 4)
```

```
(3, 4)
   (4,)
------
(3, 4)
```

Broadcasting a Row Vector

```
a = np.ones((3, 3), dtype=int)
b = np.arange(3)
print(a + b)
# [[1 2 3]
#  [1 2 3]
#  [1 2 3]]
```

a: (3, 3)

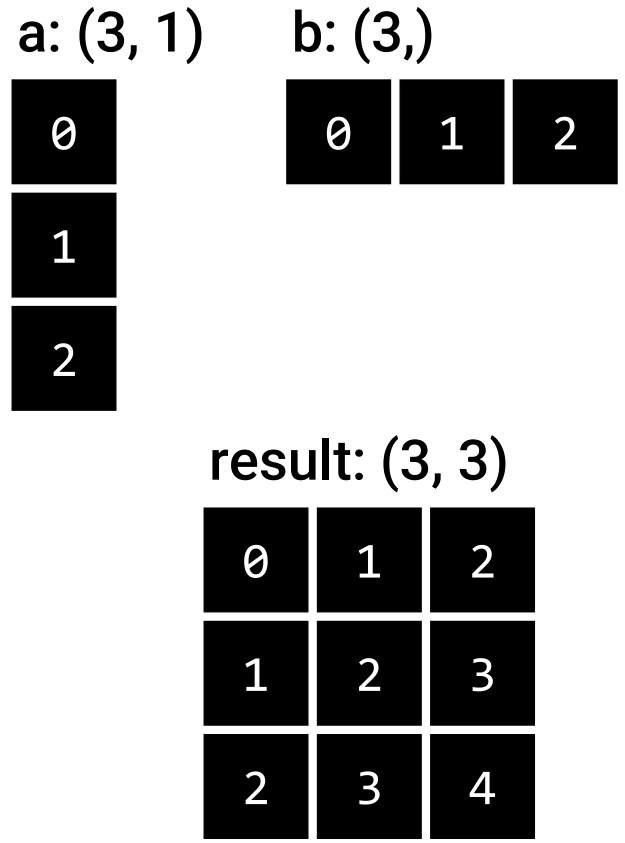


b: (3,) repeated per row



Broadcasting a Column and a Row

```
a = np.arange(3).reshape(3, 1)
b = np.arange(3)
print(a + b)
# [[0 1 2]
#  [1 2 3]
#  [2 3 4]]
```



When Broadcasting Fails

Will this addition work? Compare shapes from the right.

```
a = np.ones((3, 4))
b = np.arange(3)
print(a + b)
```

[Answer] When Broadcasting Fails

Will this addition work? Compare shapes from the right.

```
a = np.ones((3, 4))
b = np.arange(3)
print(a + b) # raises ValueError
```

The trailing dimensions 4 and 3 are incompatible.

(3, 4) + (3,) → cannot broadcast

Summary

- A **ndarray** has a shape, size, dimensionality, and dtype.
- Array arithmetic is normally element-wise.
- Basic slices create views
  - Use copy() for independent data.
- An axis tells NumPy which dimension to operate along.
- Broadcasting combines compatible shapes.

Importing NumPy

- NumPy shares the names of its functions with functions in other modules (e.g., math module in Python standard library). Therefore, it is not recommended to use imports like the following

```
from numpy import *
```

- Instead, it is recommended to follow the convention of using np as alias:

```
import numpy as np
```