Module 4:
Working with numbers

Shortcomings of native lists for math

1) inhomogeneous → slow
2) no concept of a matrix (as in linear algebra).
    - Every row could have different length.
    - Transposition is inefficient.

```
alist = [12,"a string",X7]
amatrix = [[12,3,1],[9,4,2]]
```

NumPy

https://numpy.org

- Provides a single class: **`ndarray`**
  - homogeneous
  - multi-dimensional
  - intended for working with *numbers*.
- Implements (almost) all of the `math` module (sometimes with slightly different names), and more.

np.array

```
import numpy as np
anarray = np.array([12,3,1])
amatrix = np.array([ [12,3,1] , [4,9,2] ])
```

anarray:

12
3
1

amatrix:

12	3	1
4	9	2

```
anarray.shape → (3,)
amatrix.shape → (2,3)
```

Indexing numpy arrays

- Same rules as with native lists.

`anarray:`

12
3
1

```
anarray[0]      → 12
anarray[-1]     → 1
anarray[-2:-1]  → array([3])
anarray[::2]    → array([12,1])
```

`amatrix:`

12	3	1
4	9	2

```
amatrix[0,0]    → 12
amatrix[-1,-1]  → 2
amatrix[0,:]    → array([12,3,1])
amatrix[1,::2]  → array([9,2])
```

Boolean indexing

- Pass a boolean array (or list) as the index.

`anarray:`

12
3
1

```
anarray[anarray<5]    → array([3,1])
```

`amatrix:`

12	3	1
4	9	2

```
amatrix[:,amatrix[0,:]<5]    → array([[3, 1],
                                      [4, 2]])
```

```
amatrix[(amatrix>5) & (amatrix<10)]    → array([9])
```

Creating arrays

numpy method	Description
`np.empty(shape)`	Empty array with a given shape.
`np.zeros(shape)`	All zeros
`np.ones(shape)`	All ones
`np.full(shape,value)`	== value*ones(shape)
`np.linspace(start,stop,num=50)`	Uniform partition of [start,stop] with N entries.
`np.arange(start,stop,step)`	Uniform partition of [start,stop) with given step size.
`np.meshspace(xcoord,ycoord)`	2D point grid over given x and y coordinates.
`np.hstack((A1,A2,…,An))`	Horizontal concatenation of A1… An
`np.vstack((A1,A2,…,An))`	Vertical concatenation of A1… An

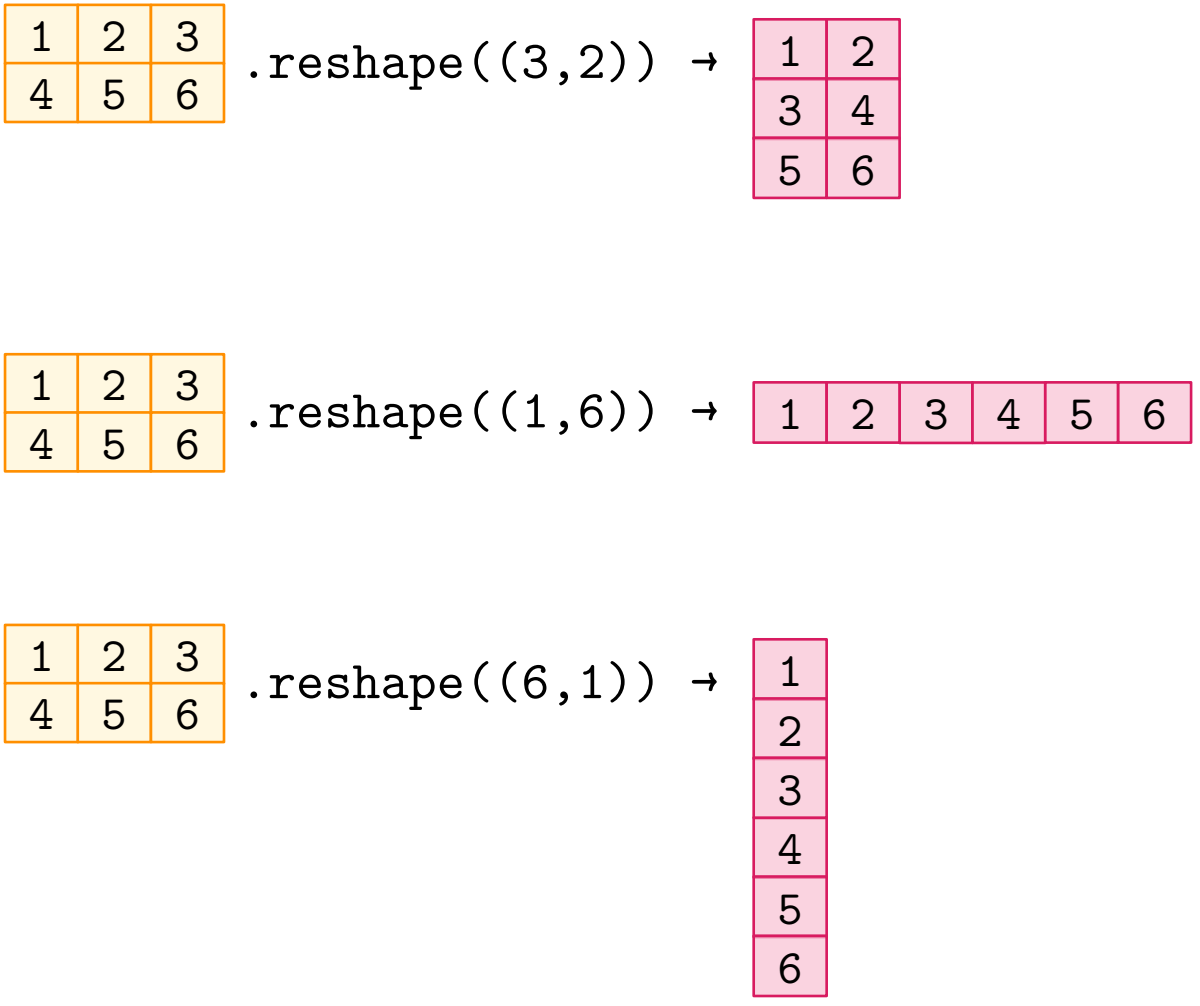
Array methods

Sorting

- `A.sort(axis=-1)` … Sort this array (in-place).
- `A.argsort(axis=-1)` … Indices that sort this array.

Reshaping

- `A.reshape(shape)` … New array, same data, different shape.


1 2 3
4 5 6
.reshape((3,2)) →
1 2
3 4
5 6
1 2 3
4 5 6
.reshape((1,6)) →
1 2 3 4 5 6
1 2 3
4 5 6
.reshape((6,1)) →
1
2
3
4
5
6

Array methods (cntd.)

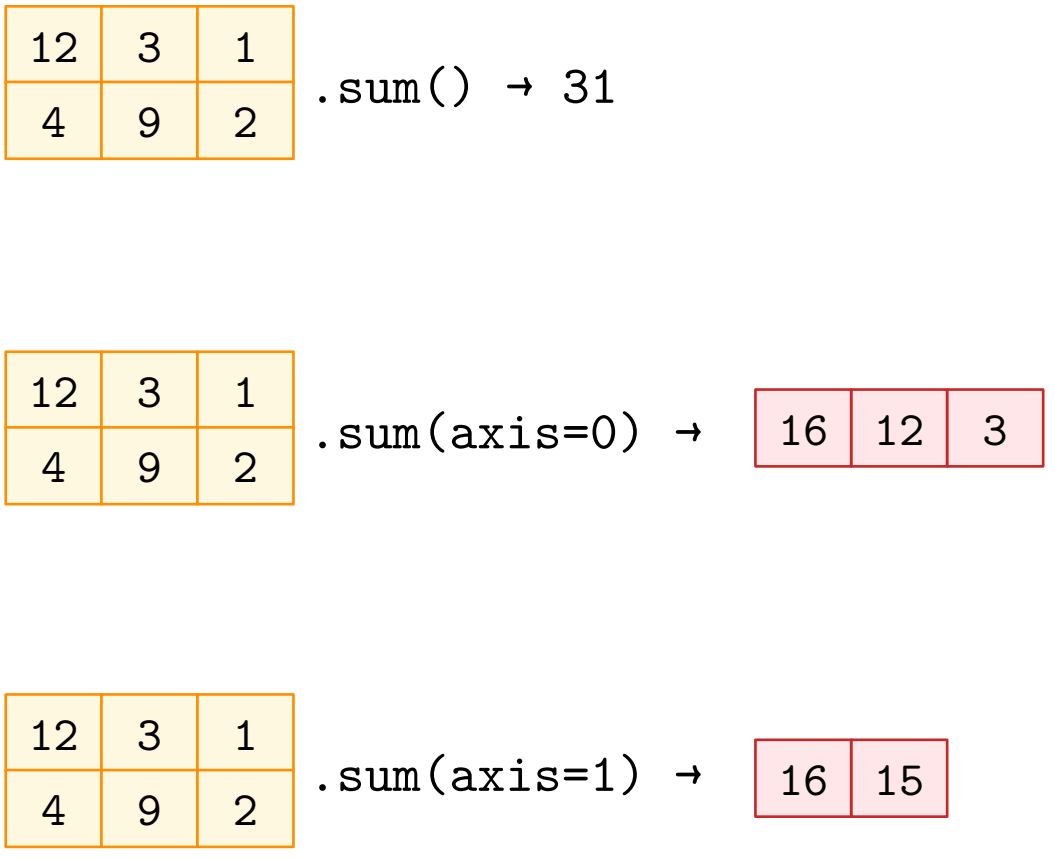
Numerical operations

```
A.sum(axis=None)
A.prod(axis=None)
A.mean(axis=None)
A.max(axis=None)
A.min(axis=None)
A.round(decimals=0)
```

```
A.cumsum(axis=None)
A.cumprod(axis=None)
A.var(axis=None)
A.argmax(axis=None)
A.argmin(axis=None)
```

Boolean operations

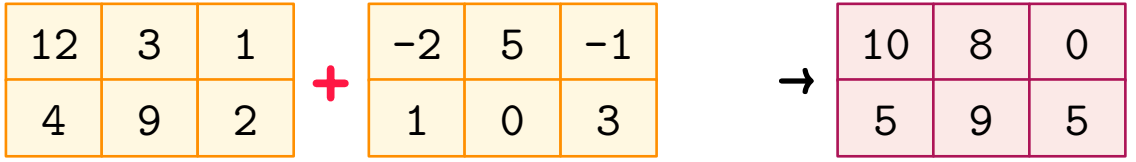
```
A.all(axis=None)
A.any(axis=None)
```


12 3 1
4 9 2
.sum() → 31
12 3 1
4 9 2
.sum(axis=0) →
16 12 3
12 3 1
4 9 2
.sum(axis=1) →
16 15

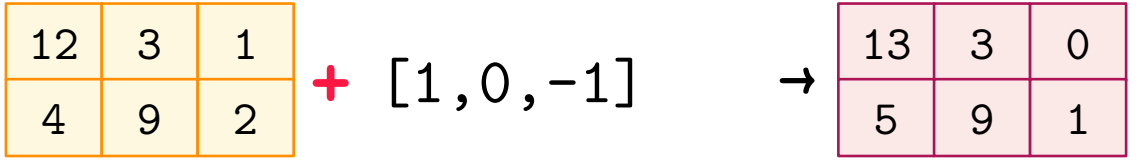
Operators { + - * / ** }

array + scalar

array + array
(same shape)



array + array
(different shape)
broadcasting



Broadcasting

Check:

1) Right align the shapes of the two arrays.
2) Left fill with ones.
3) All columns must either be equal or contain a 1.
4) The result will have the largest of the two numbers in each dimension.

Execute:

1) Copy values over the dimensions with size 1.
2) Do the operation element-wise.

Example

```
      A.shape  → (5,2,1,2)
      B.shape  →   (1,2,2)
(A+B).shape  → (5,2,2,2)
```

Linear algebra

`a.shape →(n,)` `A.shape →(n,m)`
`b.shape →(n,)` `B.shape →(m,r)`

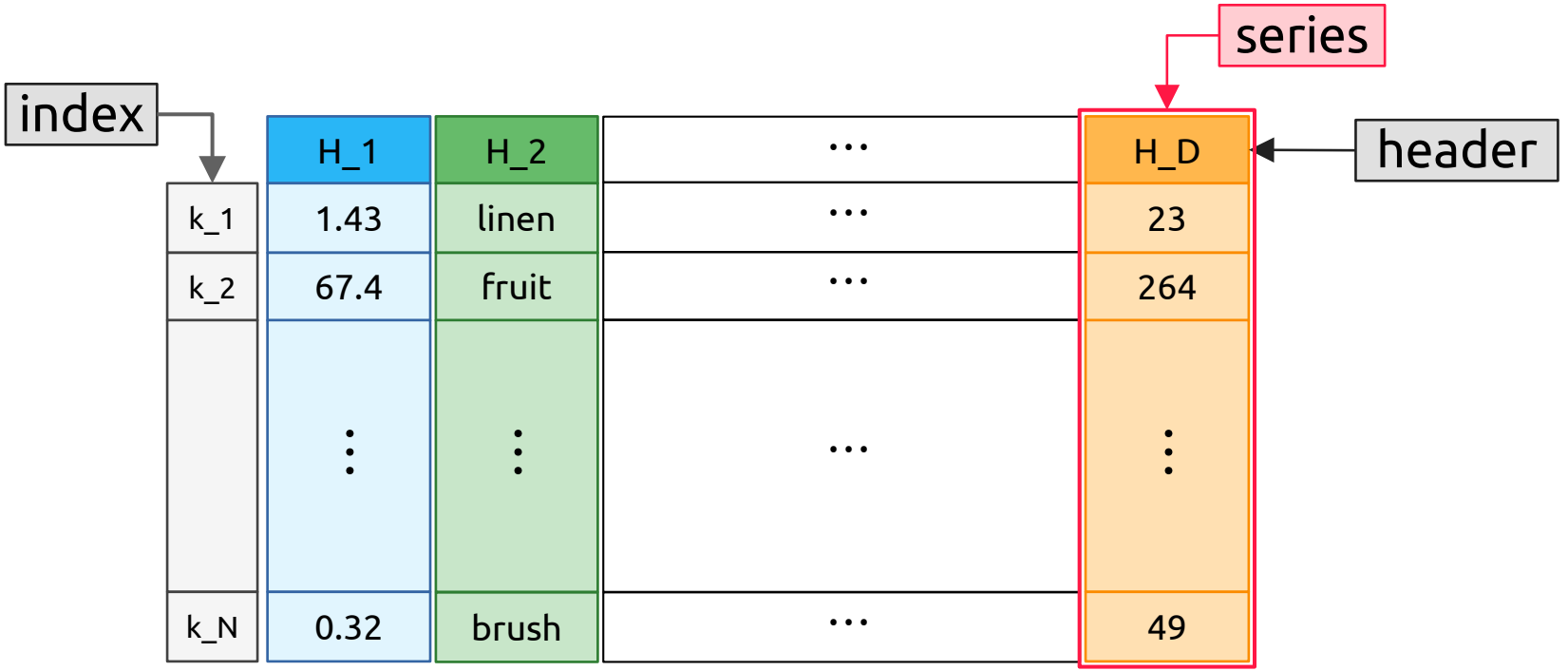
```
import np.linalg as la
```

$a, b \in \mathbb{R}^n$
$A \in \mathbb{R}^{n \times m}$
$B \in \mathbb{R}^{m \times r}$

	Operation	Mathematical notation
`la.dot(a,b)`	Dot product	$a \cdot b$
`A@B`	Matrix multiplication	AB
`A.T`	Matrix transpose	A^T
`la.eig(A)`	Spectral factorization	$Av = \lambda v$
`la.det(A)`	Determinant	$\|A\|$
`la.trace(A)`	Trace	$tr(A)$
`A.inv(A)`	Inverse of a square matrix	A^{-1}
`A.pinv(A)`	Pseudo-inverse	$(A^T A)^{-1} A^T$

pandas: A package for tabular data

- **DataFrame:** (index,{header:series})



Querying a DataFrame

- **Column selectors:** `[]`
  - `X["H1"]` … single column
  - `X[["H1","H2"]]` … multiple columns

- **Selecting rows by index:** `.loc[]`
  - `X.loc[k1]` … single row
  - `X.loc[[k1,k2]]` … multiple rows

- **`loc[]` also accepts a column selector**
  - `X.loc[k1,"H1"]`
  - `X.loc[k1,["H1","H2"]]`
  - `X.loc[[k1,k2],"H1"]`
  - `X.loc[[k1,k2],["H1","H2"]]`

Querying a DataFrame (cntd.)

- **Selecting rows with a conditional**
  - `X.loc[boolean_mask]`
  - `X.loc[boolean_mask,column_selector]`

… syntactic sugar: `X[<boolean mask>]`

- **Ordered rows (integer index)**
  - `X[slice]`
  - `X.loc[slice,column_selector]`

- **Ordered rows and columns:** `.iloc[]`
  - `X.iloc[row_slice]`
  - `X.iloc[row_slice,col_slice]`

Loading and saving data

Single object files

	text	pickle
numpy	`np.savetxt(filename,A)` `A = np.loadtxt(filename)`	`np.save(filename,A)` `A = np.load(filename)`
pandas	`DF.to_csv(filename)` `DF = pd.read_csv(filename)`	`DF.to_pickle(filename)` `DF = pd.read_pickle(filename)`

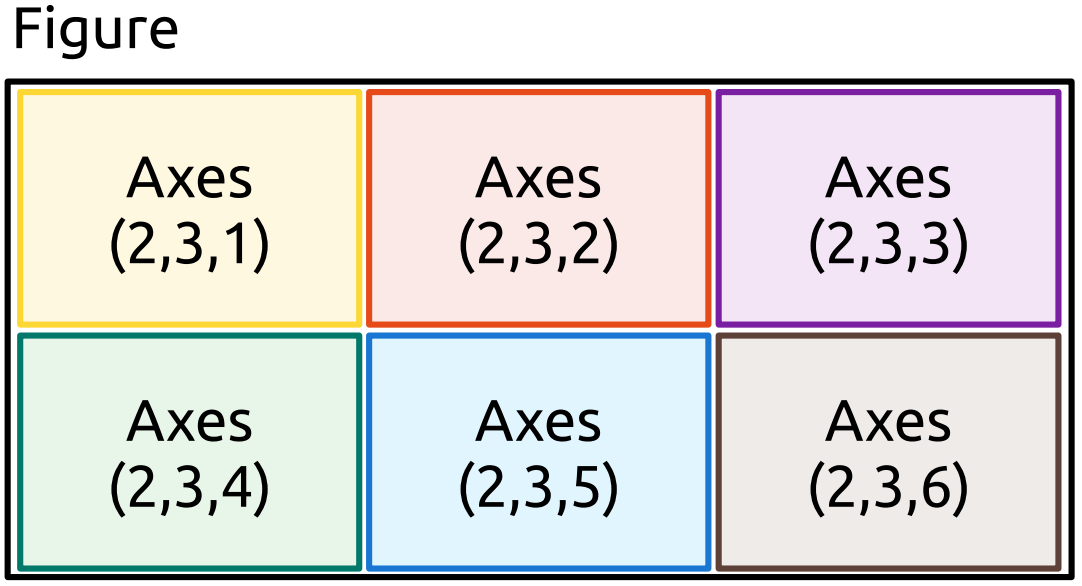
Multiple object files

```
import pickle
```

```
with open("mypickle.pkl","wb") as f:
    pickle.dump((A,D),f)
```

```
with open("mypickle.pkl", "rb") as f:
    Anew, Dnew = pickle.load(f)
```

Plotting with matplotlib



Steps:

1. Create the figure: set dimensions and axes configuration.
2. Add plotting elements to each axes
3. Display the plot