NumPy Arrays

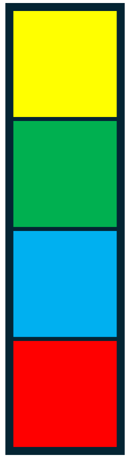
9.25.24

Learning Objectives

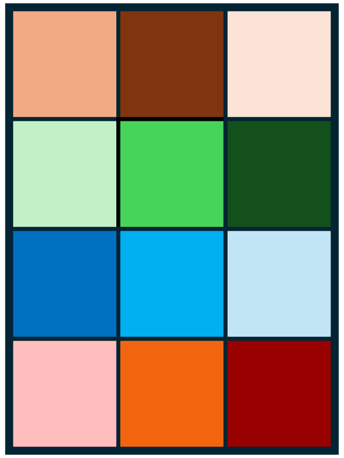
- Array structuring
- Array operations
- Array Indexing

Vectors, Matrices and Arrays

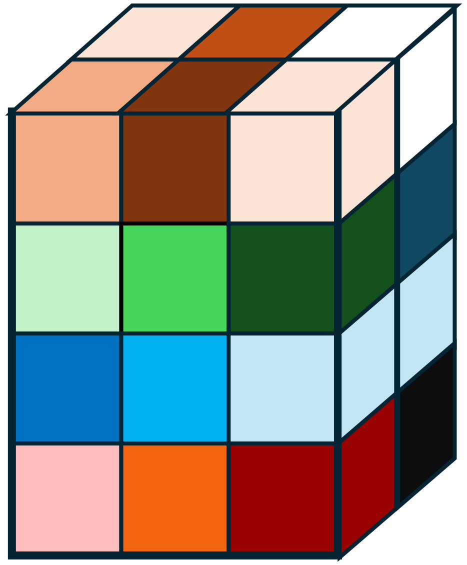
row
vector



column
vector



4x3
matrix



4x3x2
array

numPy displays this
as 4 3x2 matrices

Creating Arrays

- Reshaping a list: **np.reshape**(list, size)

- Converting a structured list: **np.array**([[1, 2] , [3, 4]])

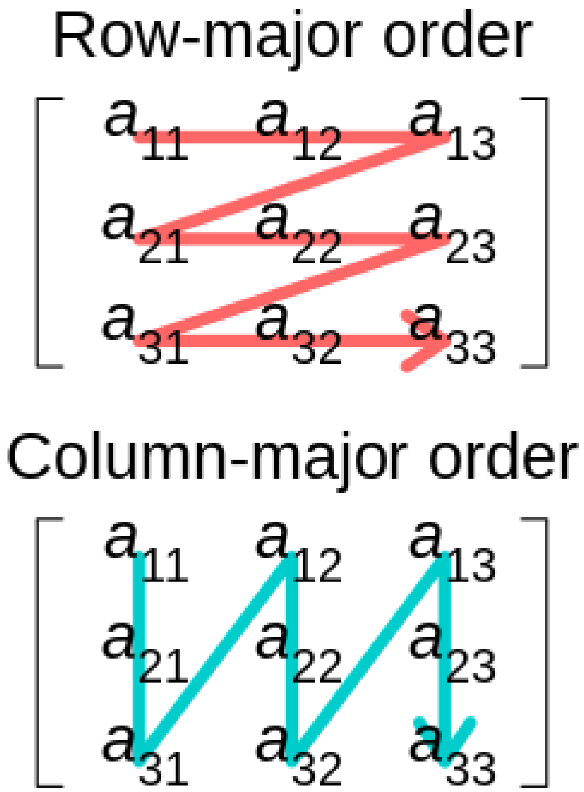
- Save: **np.save**(---.npy)
- Load: **np.load**(---.npy)

NumPy is Row-Major

- Rows are stored as contiguous memory
- Dimensions are still column x row

```
np.reshape([1,2,3,4,5,6],[2,3])
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```



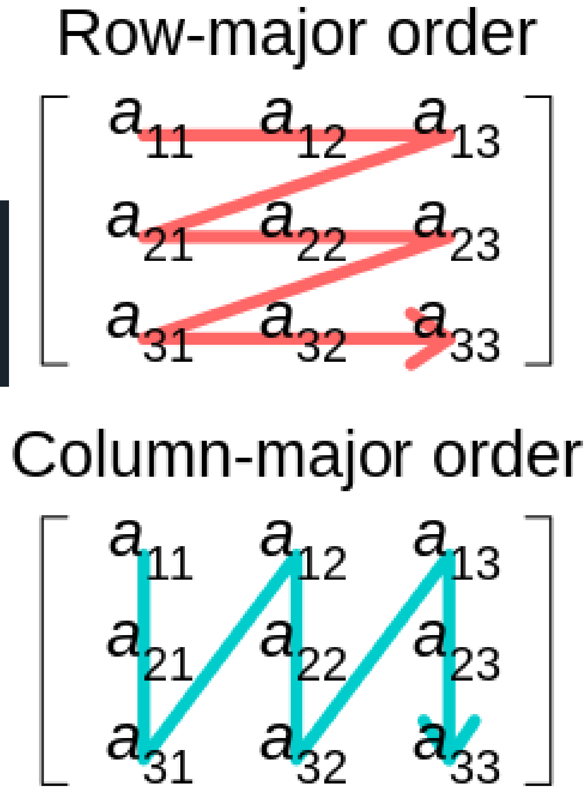
NumPy is Row-Major

- Rows are stored as contiguous memory
- Dimensions are still column x row

```
np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
```

```
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```



Special Arrays

- Homogeneous arrays: **np.full**(size, value)
    - All zeros: **np.zeros**(size)
    - All ones: **np.ones**(size)

- Identity matrix: **np.eye**(n,m)

- “empty” matrix: **np.empty**(size)
    - Preallocates memory. Often filled with garbage

Creating Structured Arrays

- **np.diag**() turns a list/vector into a diagonal matrix and vice-versa

- **np.repeat**([[3,7]],2,axis=1)=$[3, 3, 7, 7]$
- **np.repeat**([[3,7]],2,axis=0)=$\begin{bmatrix} 3 & 7 \\ 3 & 7 \end{bmatrix}$

  Repeats elements

- **np.tile**([3,7],[2,2])=$\left[\begin{array}{cc|cc} 3 & 7 & 3 & 7 \\ \hline 3 & 7 & 3 & 7 \end{array}\right]$

  Repeats a block

Matrix Multiplication

- Matrix-Matrix Multiplication: A@B=C in NumPy

$$\begin{pmatrix} a & b \\ \boxed{c \quad d} \end{pmatrix} \begin{pmatrix} w & \boxed{x} \\ y & \boxed{z} \end{pmatrix} = \begin{pmatrix} aw + by & ax + bz \\ \boxed{cw + dy} & cx + dz \end{pmatrix} \qquad C_{i,j} = \sum_m A_{i,m} B_{m,j}$$

- Elementwise Multiplication: A*B=C in NumPy

$$\begin{pmatrix} a & b \\ \boxed{c} & d \end{pmatrix} \odot \begin{pmatrix} w & x \\ \boxed{y} & z \end{pmatrix} = \begin{pmatrix} aw & bx \\ \boxed{cy} & dz \end{pmatrix} \qquad C_{i,j} = A_{i,j} B_{i,j}$$

Matrix Operations: Transpose/Hermitian

- Transpose: A.T in NumPy

$$\begin{pmatrix} a & b \\ \boxed{c} & d \end{pmatrix}^T = \begin{pmatrix} a & \boxed{c} \\ b & d \end{pmatrix} \qquad C_{i,j} = A_{j,i}$$

- Hermitian Transpose: A.H=A.conj().T in NumPy

$$\begin{pmatrix} a + wi & b + xi \\ \boxed{c + yi} & d + zi \end{pmatrix}^H = \begin{pmatrix} a - wi & \boxed{c - yi} \\ b - xi & d - zi \end{pmatrix} \quad C_{i,j} = \overline{A_{j,i}}$$

Restructuring Arrays

- Check shape: A.**shape** or **np.shape**()
- Reshape: **np.reshape**(list,size)
- Generalized transpose to permute arbitrary dimensions:

  if A is 4 x 3 x 5,   **np.transpose**(A,[2, 0, 1]) is 5 x 4 x 3

  Moves dim. 2 to dim 0 , dim 0 to dim 1, etc.

- If just swapping two dimensions: **np.swapaxis**(A, ax1, ax2)
- Flattening:

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

Indexing

1. **Slice**

2. **“Fancy Indexing” (list-based)**

3. **Logical**

Slice Indexing Arrays

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \text{A[:,0]} = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}$$

$$\text{A[[1,2]]=A[[1,2],:]=} \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

What about A[0:3:2, 0:3:2]?

Slice Indexing Arrays

$$A = \begin{bmatrix} \boxed{1} & 2 & \boxed{3} \\ 4 & 5 & 6 \\ \boxed{7} & 8 & \boxed{9} \end{bmatrix} \quad \text{A[0:3:2, 0:3:2]} = \begin{bmatrix} 1 & 3 \\ 7 & 9 \end{bmatrix}$$

Arbitrary sub-matrices: A[**np.ix_**(list_1,list_2)]

Mutable, slice-based assignment:

```
A[0:3:2,0:3:2]=\
np.full([2,2],12)
```

```
array([[12,  2, 12],
       [ 4,  5,  6],
       [12,  8, 12]])
```

Fancy Indexing

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

A[[0,0,2,1],[1,2,0,1]]=[2 3 7 5]

Logical Indexing

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$\mathrm{A}[\ \underbrace{\mathrm{A}\%2{=}{=}0}_{\text{Boolean matrix}}\ ] = [2\ 4\ 6\ 8]$$

Retrieve logical indices: np.where()

Practice Together

- Given an array B containing taco vs. burger sales find the days that at least 3 more tacos were sold than burgers:

$$\begin{array}{c c} & \begin{array}{ccccccc} \text{Sun} & \text{Mon} & \text{Tue} & \text{Wed} & \text{Thu} & \text{Fri} & \text{Sat} \end{array} \\ \begin{array}{c} \text{Taco} \\ \text{Burger} \end{array} & \begin{bmatrix} 31 & 15 & 79 & 14 & 23 & 30 & 43 \\ 42 & 12 & 16 & 11 & 19 & 35 & 56 \end{bmatrix} \end{array}$$

np.load('…\SalesData_9.25.npy')

Broadcasting over missing dimensions

- Suppose you want to add 1 to row 1 and 2 to row 2?
- Not mathematically legal, but NumPy knows what you mean
  - Checks for match starting from last dimension
  - Stretches dims with size 1

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$B = \begin{bmatrix} 10 & 11 \\ 12 & 13 \end{bmatrix}$$

Valid

A+[1,2]

A+np.reshape([1,2,3],3,1)

Invalid

A+[1,2,3]

$$B+[1,2] = \begin{bmatrix} 11 & 13 \\ 13 & 15 \end{bmatrix}$$

Vectorization Instead of Loops

- (Uncompiled) loops are slow
- Vectorization=Performing elementwise operations all at once, instead of looping.

Elementwise:

np.vectorize(function_obj)

- Returns another function object, can take any number of args

Slice-Based:

np.apply_along_axis(func, axis=.., arr=...)

- Returns array, can only apply to functions with a single arg

Practice: Z-score

- Write a function that takes a m x n array and returns the zscore over each row, using broadcasting

$$zscore = \frac{x - mean(x)}{std(x)}$$

$$std = \sqrt{\frac{\sum\left(x_i - \mu(x)\right)^2}{n-1}}$$

Learning Objectives

- Array structuring
- Array operations
- Array Indexing

Fin