CS180/280A Discussion #1

Konpat

Credits:
Justin, Chung Min

Welcome!!

GSIs

Justin Kerr

Konpat Preechakul

Chung Min Kim

Brent Yi

Tutors

Jameson Crate

Jorge Diaz Chao

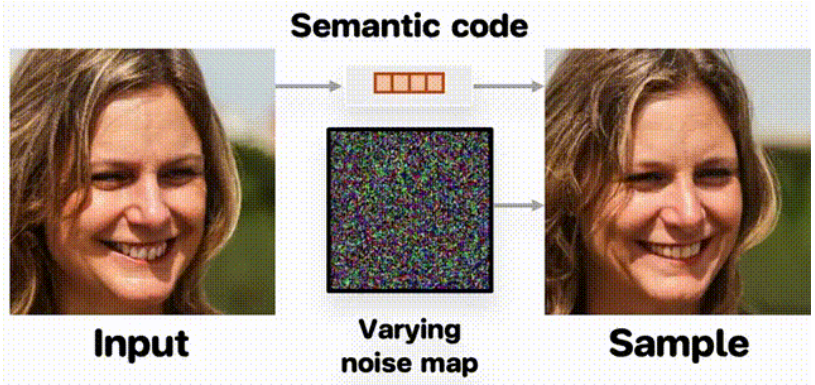
Natalie Wei

Jingfeng Yang

Me: **Konpat** Preechakul 👋

"Learning abstractions from pixels"

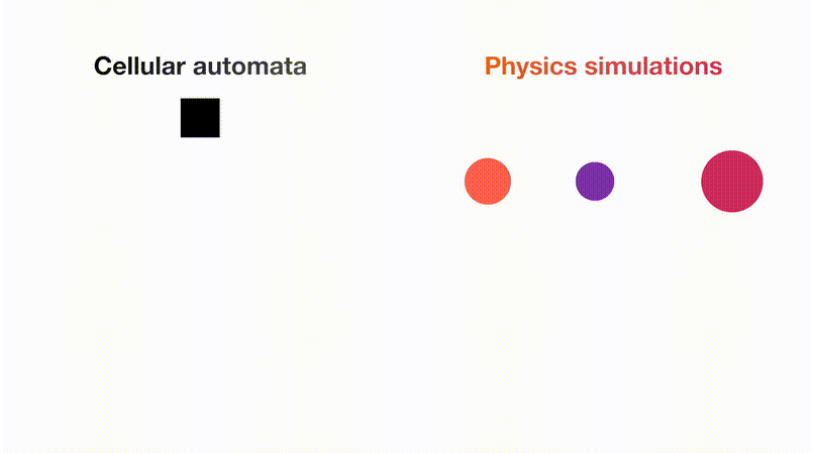
Diffusion models & Representation learning



Scene understanding

Visual Jenga

Machine learning



Reminders

Proj1 due Fri **9/12** 11:59pm

OH dates are released!

Worksheets online:

Topics

Discussion	Topic	Materials
Week 1	Python & NumPy Fundamentals for Computer Vision	Main sheet | Challenge | Solutions

Slides

Discussions this year!

- **Practical practice** (for Projs) + **Conceptual understanding** (for exams)
- **Collaborative**! Move to be near someone! :)
- **Minimal laptop.** We want you to go through with your hand!
- **Worksheet problems are in scope** for exams. Bonus questions are intended to be *hard.*
- **Note:** these are new this year! (Rough □)

What are your questions?

Agenda

- Short lectures (15 mins)
- Problems 1 (7 mins) + Answer (3 mins)
- Problems 2 (7 mins) + Answer (3 mins)
- A bit more lectures (10 mins)
- A bit more problems (10 mins)

This discussion: Visual world 🤝 Computers

We need to learn how to **1) input, 2) store, and 3) manipulate 4) output images!**

Storing data?

Visualizing?

What data structure are images?

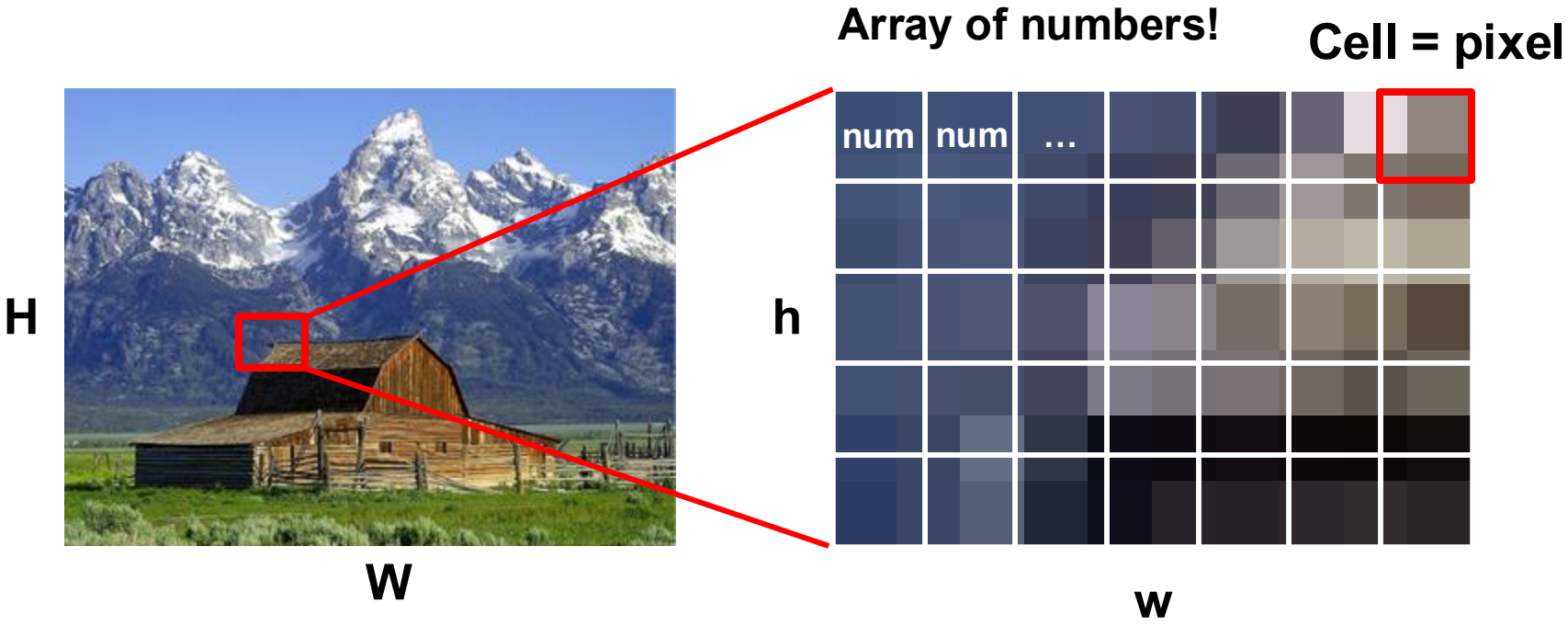
An image is an **array** of pixels!

H

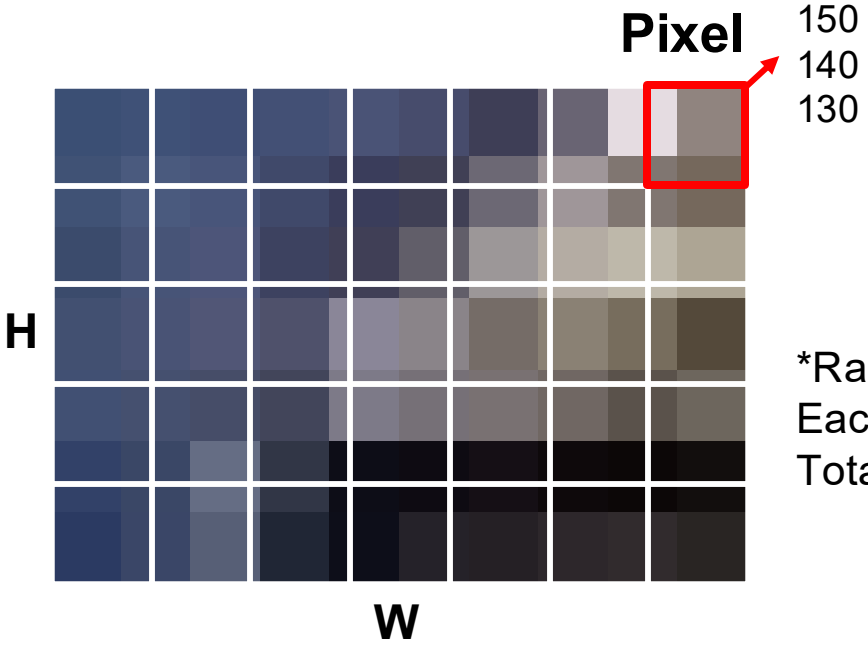
W

What data structure are images?

An image is an **array** of pixels!



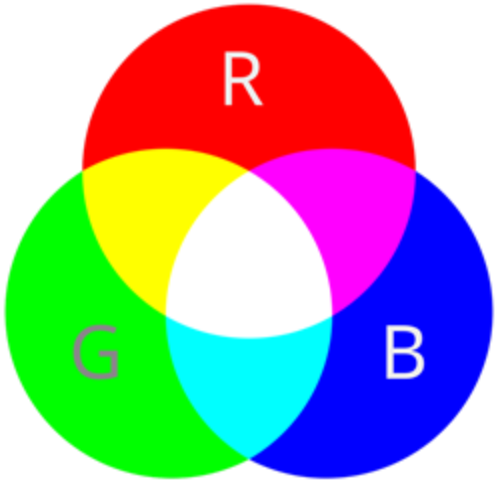
What is a pixel?



*Range (0 – 255)
Each color 8 bits
Total **24-bit color**

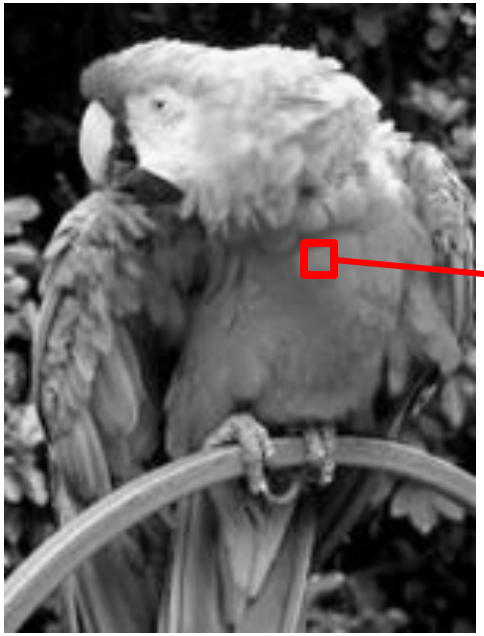
COLOR!

3 channels: red, green, blue



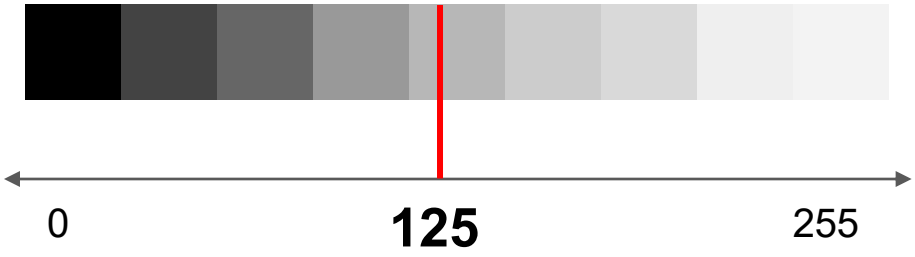
*RGB vs. BGR conventions

Pixel is not always 3 numbers!



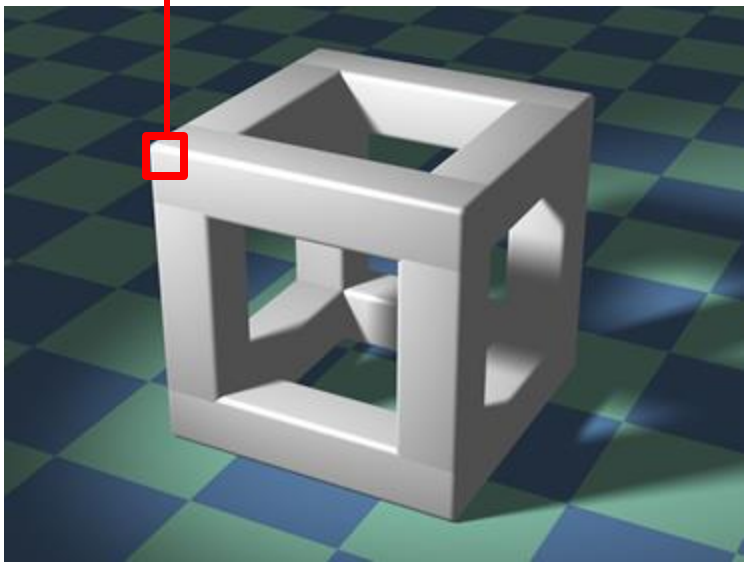
Grayscale!

1 channels: intensity



Pixel can be something else!

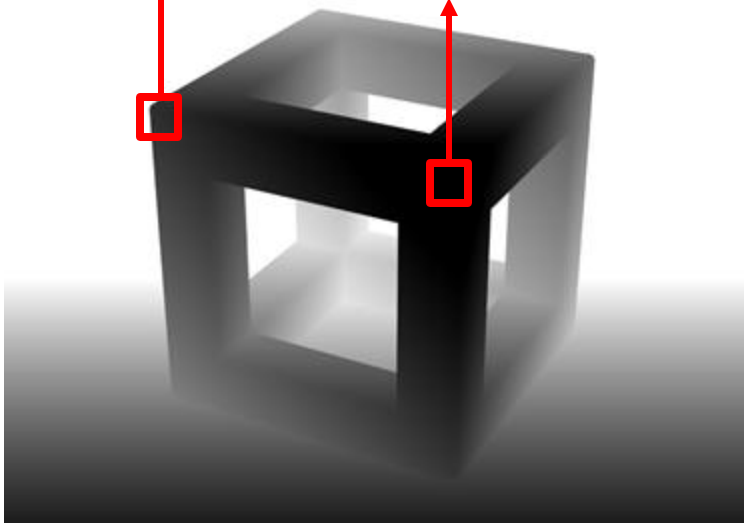
180 190 170



Depth!

3 (feet)

2.5 (feet)



Inspecting images

```
>>> img = cv2.imread("img.jpg")

>>> print(img.shape) # shape (1080, 1920, 3)

>>> print(img.dtype) # np.uint8

>>> print(img.min(), img.max()) # 0 255

>>> plt.imshow(img)
```

2 primary formats:

- uint8, 0->255 scaling
- float, 0->1.0 scaling

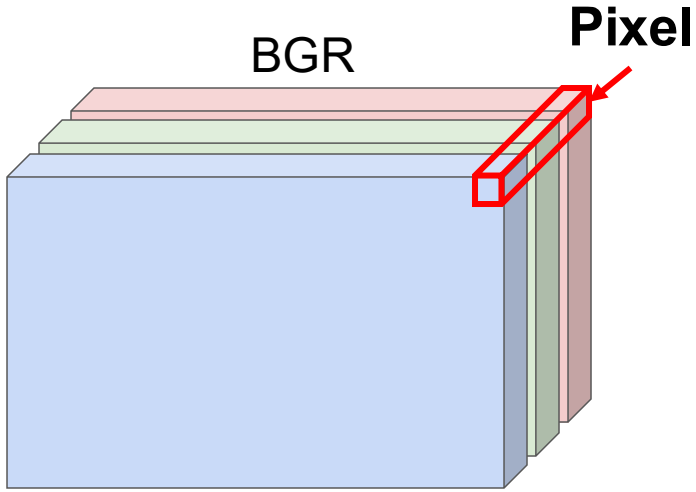
Be careful converting!

***Let me show you…**

Pixel layout in an array

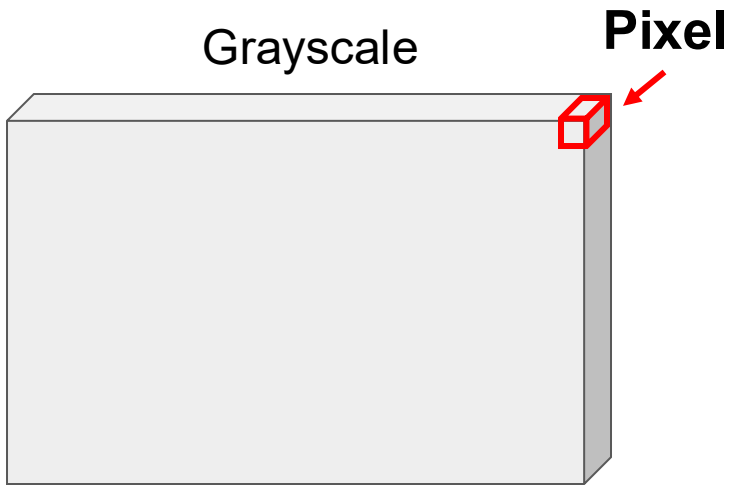
```
>>> img = cv2.imread("img.jpg") # shape (1080, 1920, 3)
```

Pixels stored along ***channels***.



`(H,W,C)` `C=3` "channel-last"

… `or` `(C,H,W)` "channel-first"



`(H,W)` or `(H,W,1)`

Singleton dim.

*BGR is a bit hard to visualize actually…

Good news: many image operations are just array operations!

So fast, so easy 😉

Let's start: Color channel manipulation (1.3 Slicing)

```
>>> img
```

Original Image

150
101
105

```
>>> img[...,0]
```

Blue Channel

105
Blue

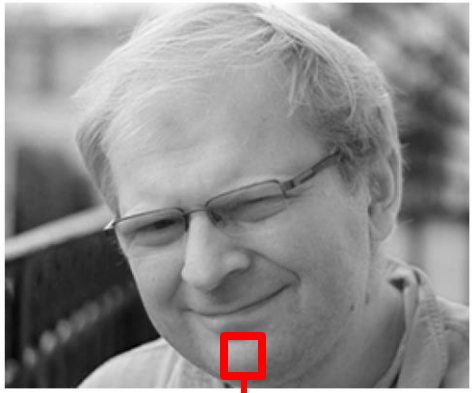
```
>>> img[...,1]
```

Green Channel

101
Green

```
>>> img[...,2]
```

Red Channel

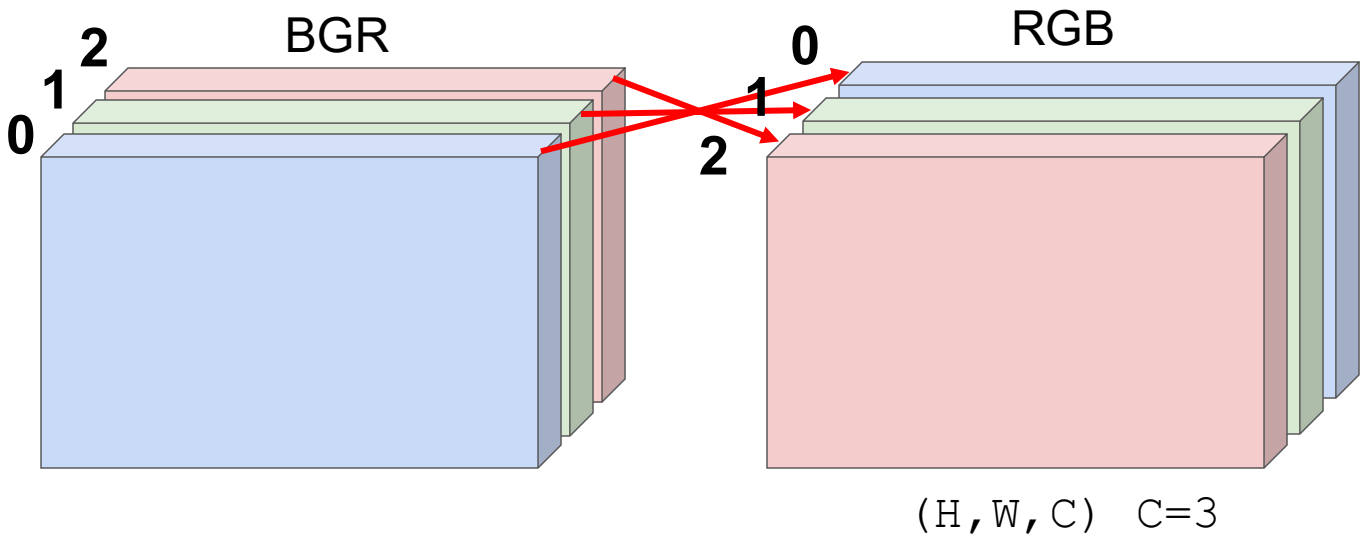


150
Red

*Reminds of Proj 1!

BGR => RGB (1.3 Slicing)

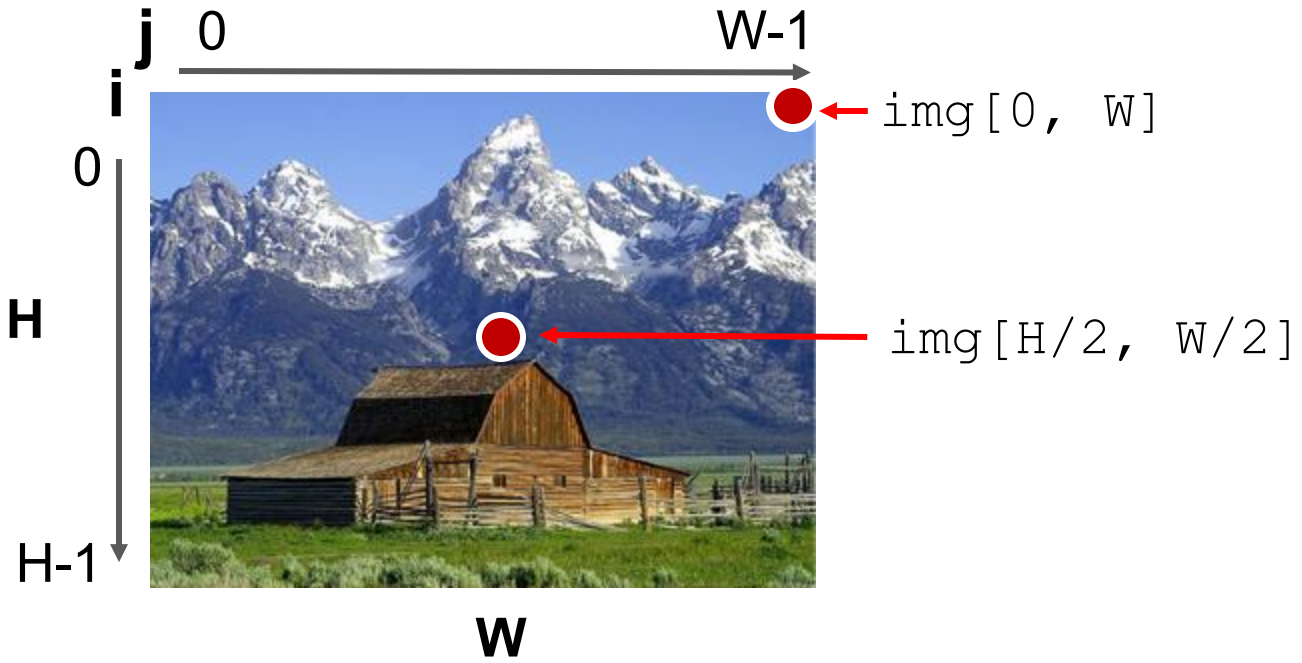
```
>>> img = img[:, :, [2,1,0]]
```



*Easier to plot. Show on notebook

Indexing conventions (1.3 Slicing)

Index into arrays like i,j in a matrix, **not like** x,y in a coordinate plane!



: (colon)

Cropping images (1.3 Slicing)

```
>>> left_half = img[:, :100, :]

>>> bottom_half = img[100:]
```

:100

100:

*Let me show you guys

Joining images (1.2 Stack & Concat)

```
>>> vertical = np.concatenate([angjoo,alyosha],axis=0)
```

angjoo

alyosha

Joining images (1.2 Stack & Concat)


angjoo


alyosha

```
np.concatenate([angjoo,alyosha],axis=0)
```


```
np.concatenate([angjoo,alyosha],axis=1)
```

What are videos? (1.2 Stack & Concat)

```
>>> video = np.stack([ frame1, frame2 , ...], axis=0)
```

Videos are just arrays of *batches* of images!

video
(T,H,W,C)

video[0] **video[5]** **video[11]** **video[15]**

(H,W,C)

What are videos? (1.2 Stack & Concat)

```
>>> video = np.stack([ frame1, frame2 , ...], axis=0)
```

Videos are just arrays of *batches* of images!

Video being played

(T,H,W,C)

Actual video "Cube"

(T,H,W,C)

NumPy basics: do **Problems 1.1-1.3 (5 mins)** with the people around you!

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

np.full(shape , value)

array.astype(type)

type: np.uint8, np.float32, np.float64

array[i, j], array[i:j]

np.concatenate([a ,b], axis=?)

np.stack([a, b], axis=?)

(Then we will go over quickly)

Pixel operations: Do **Problems 2.1-2.5 & 2.9 (5 mins)**

np.flip(img, axis=?)

np.transpose(img, axes=[…]).

img.transpose(…)

img.astype(…)

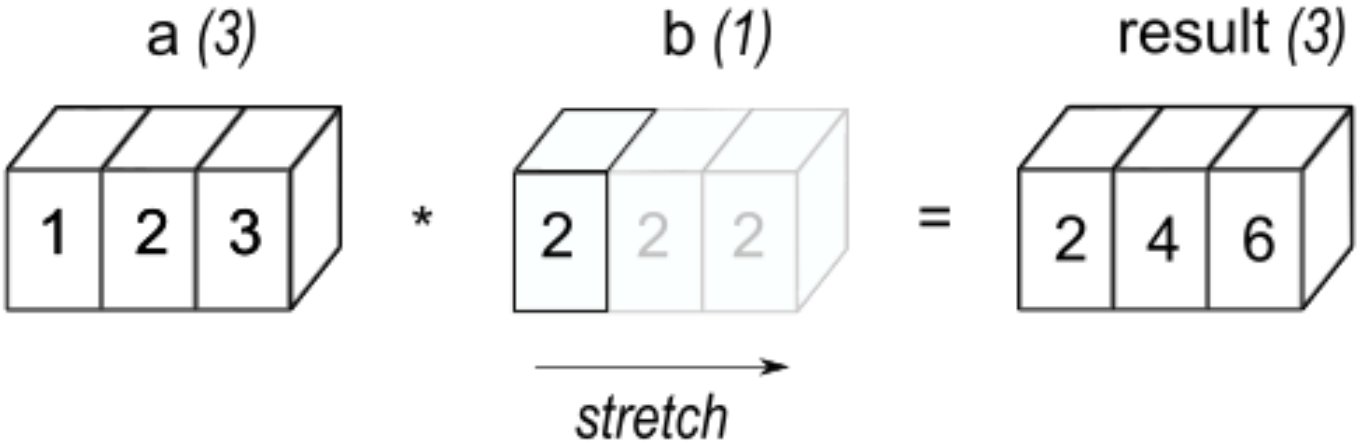
np.mean(img, axis=?)

img.mean(?)

(Then we will go over quickly).

Broadcasting: automatically *repeat* elements to match!

```
>>> a = np.array([1, 2, 3])        # shape (3,)
>>> b = np.array(2)                # shape ()!
>>> print( a * b )                 # [2.0,4.0,6.0] shape (3,)
```



Broadcasting

Rule for figuring out behavior:

1. **Line up** array shapes starting from the **right**
2. **For each axis:**
    a. If shapes match, continue to the left
    b. If shapes don't match and one is 1, stretch its values to fit the larger
    c. If shapes don't match and *neither* are 1, throw an error

A = (2, 3)
B = (2, 1, 1) → A = (2, 3)
B = (2, 1, 3) → A = (2, 3)
B = (2, 2, 3) → A = (1, 2, 3)
B = (2, 2, 3) → A = (2, 2, 3)
B = (2, 2, 3)

Broadcast: Do **Problems 3.1-3.3 (5 mins)**

Rule for figuring out behavior:

1. **Line up** array shapes starting from the **right**
2. **For each axis:**
    a. If shapes match, continue to the left
    b. If shapes don't match and one is 1, stretch its values to fit the larger
    c. If shapes don't match and *neither* are 1, throw an error

Vectorization: No Loops!

Avg.

"Vectorization" means doing things with native NumPy >> Python loops

Much faster when possible!

Native C (low overhead) vs Python (high overhead)

Example (averaging):

SLOW

```
for i in range(H):
    for j in range(W):
        out[i,j,:] = (ang[i,j] + aly[i,j])/2.0
```

Fast `out=(ang + aly)/2.0`

Vectorization: do **Problems 4.1, 4.3!**

“Vectorization” means writing things with native NumPy operations rather than for loops

np.**mean**(…)

np.**sum**(…)

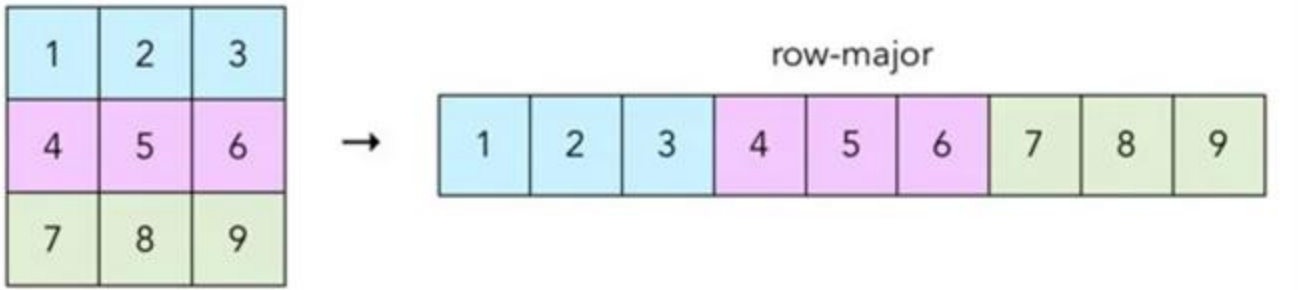
Thanks for coming!

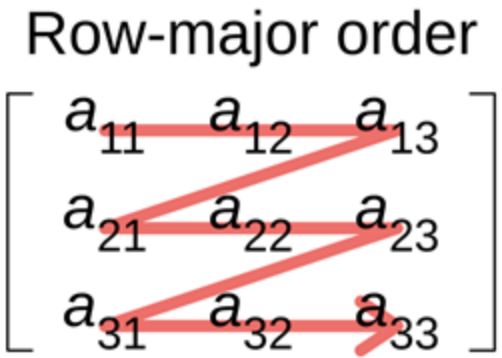
Explore (the full sheet): Bonus & Einsum & Finish the rest.

Manipulating shapes (5)

Many times we want to shuffle the order of axes or combine them.

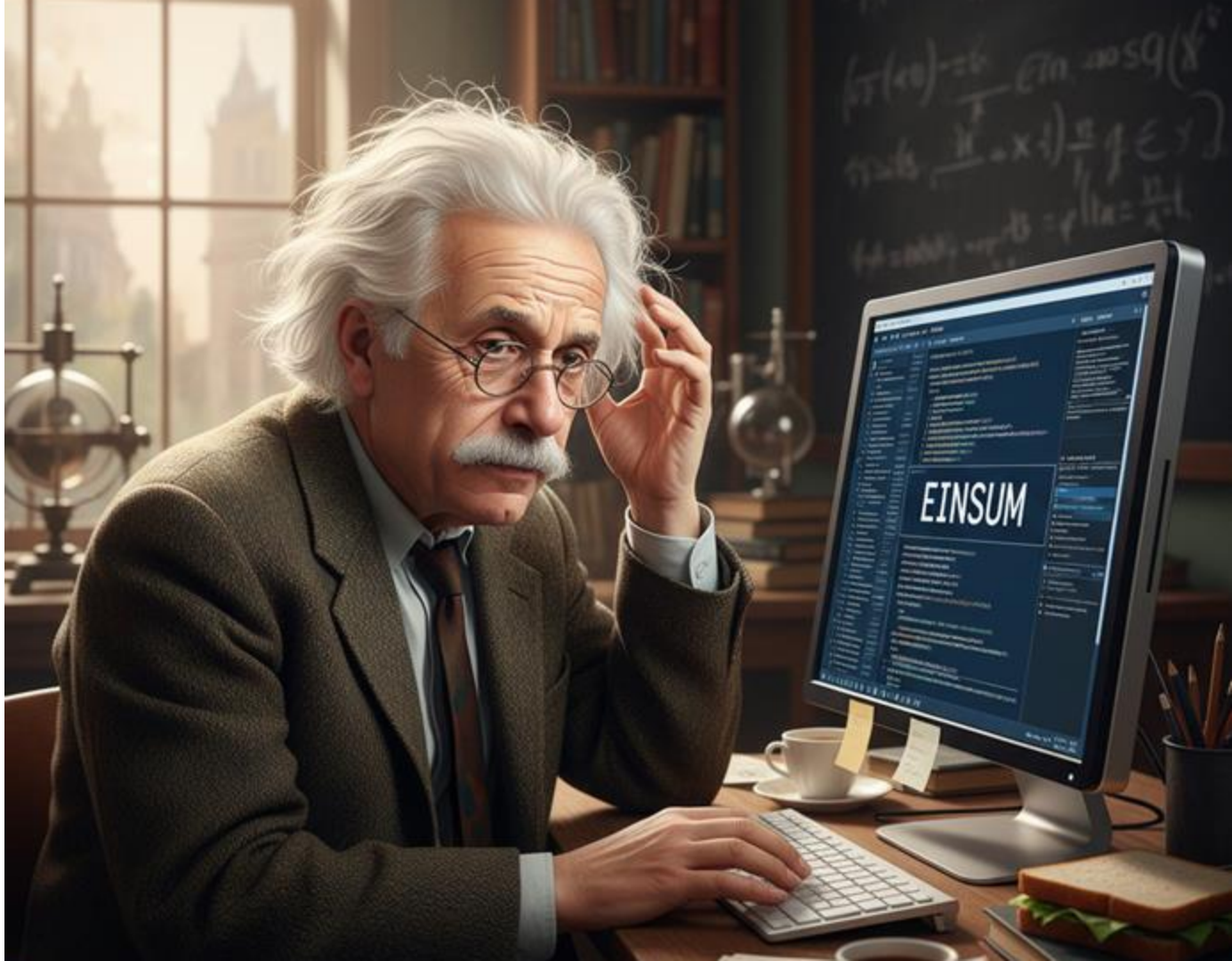
Remember arrays are *row-major!*





Do problems 5.1 => 5.3

Einsum



`einsum` examples!

```
>>> a = np.arange(4)  # (4,)
>>> b = np.arange(4)  # (4,)
```

```
>>> np.einsum('i,i->i', a, b)
>>> np.einsum('i,j->ij', a, b)
>>> np.einsum('...i,...i->...', a, b)
```

```
array([0, 1, 4, 9])     # (4,)
```

```
array([[0, 0, 0, 0],
       [0, 1, 2, 3],
       [0, 2, 4, 6],
       [0, 3, 6, 9]])  # (4, 4)
```

```
np.int64(14)            # (,)
```