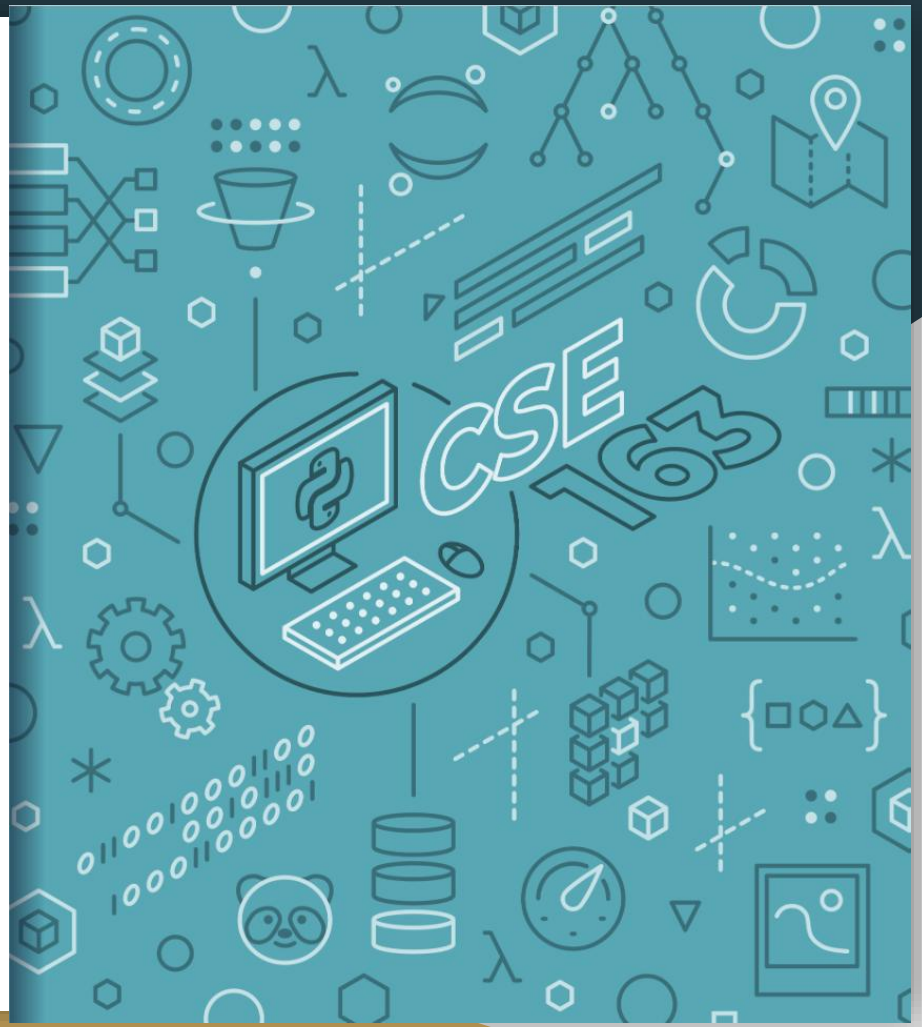
CSE 163
Numpy and Images

Adrian Salguero
Spring 2026

💭 Icebreaker (discuss with neighbors):
What is your favorite quote?
Add to our Slido!

slido.com
#cse163



Announcements

- **Take Home Assessment 5: Mapping** due Tuesday May 26th at 11:59pm
- **Reading Assignment 4** is due on Gradescope by Friday, May 15th at 11:59pm!
- **Checkpoint 4** due Monday, May 18th at 11:59pm!
- **Project Part 2** due tomorrow at 11:59pm!
- **Lesson 19 Canvas Quiz** due tonight at 11:59pm!

Final Reflection

- This course does not have a traditional final exam, but we will still be meeting during Finals Week
- **In-person: Monday, June 8 at 8:30am-10:20am in KNE 220 (this room!)**
- Show up with yourself, your ID, and a writing utensil
- Written reflection (think like time “essays”)
- No notes or reference sheets allowed (we want your in-the-moment thoughts)
- Answer 3 of 5 writing prompts
  - 2 required
  - 1 of 3 choice prompts
- Prompts and writing tips can be found on the course website
- Not meant to be a stressful “exam”

Numpy Arrays

The primary data structure that we work with is called a `numpy.array`

```
a = np.array([12, 6])      # [12, 6]
b = np.arange(5)           # [0, 1, 2, 3, 4]
c = np.ones(3)             # [1, 1, 1]
```

Reshaping

We looked at how numpy arrays have **shape**

```
x = np.ones((2, 3))
```

```
x: array([[1, 1, 1],
          [1, 1, 1]])
```

Numpy provides a function called **reshape** that allows us to change the shape of an array without change the data inside.

```
y = np.arange(9)
y = y.reshape((3, 3))
```

```
y: array([[0, 1, 2],
          [3, 4, 5],
          [6, 7, 8]])
```

Reshaping

We also saw how that numpy arrays have **shape**

```
x = np.ones((2, 3))
```

```
x: array([[1, 1, 1],
          [1, 1, 1]])
```

Numpy provides a function called **reshape** that allows us to change the shape of an array without change the data inside.

```
y = np.arange(9)
y = y.reshape((3, 3))
```

```
y: array([[0, 1, 2],
          [3, 4, 5],
          [6, 7, 8]])
```

Broadcasting

Numpy lets us perform element-wise operations on arrays that have the same shape (and sometimes with different shape!)

```
m = np.ones((2, 3))
v = np.arange(3)
m + v
```

```
m.shape = (2, 3)
v.shape = (3,)
```

```
m:  [[1, 1, 1],
     [1, 1, 1]]
```

+

```
v:  [0, 1, 2]
```

Broadcasting

Numpy lets us perform element-wise operations on arrays that have the same shape (and sometimes with different shape!)

```
m = np.ones((2, 3))
v = np.arange(3)
m + v
```

```
m.shape = (2, 3)
v.shape = (3,)
```

```
m:  [[1, 1, 1],
     [1, 1, 1]]
```

+

```
v:  [0, 1, 2]
```

Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
m.shape = (2, 3)
v.shape = (3,)
```

```
m:  [[1, 1, 1],
     [1, 1, 1]]
```

+

```
v:  [0, 1, 2]
```

Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
m.shape = (2, 3)
v.shape = (1, 3)
```

```
m:  [[1, 1, 1],
     [1, 1, 1]]
```

+

```
v:  [0, 1, 2]
```

Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
m.shape = (2, 3)
v.shape = (1, 3)
```

```
m:  [[1, 1, 1],
     [1, 1, 1]]
```

+

```
v: [[0, 1, 2]]
```

Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
m.shape = (2, 3)
v.shape = (2, 3)
```

```
m:  [[1, 1, 1],
     [1, 1, 1]]
```

+

```
v: [[0, 1, 2]]
```

Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
m.shape = (2, 3)
v.shape = (2, 3)
```

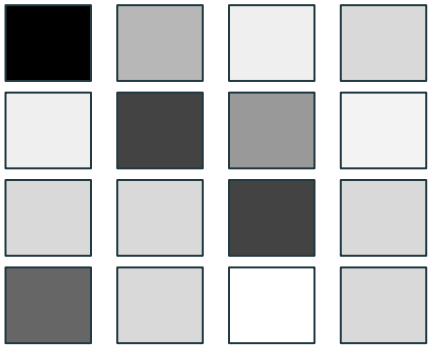
```
m:  [[1, 1, 1],
     [1, 1, 1]]
```

+

```
v: [[0, 1, 2],
    [0, 1, 2]]
```

Images as Matrices

Grey-scale images can be represented as matrices.



Grey-scale: 255

Grey-scale: 0

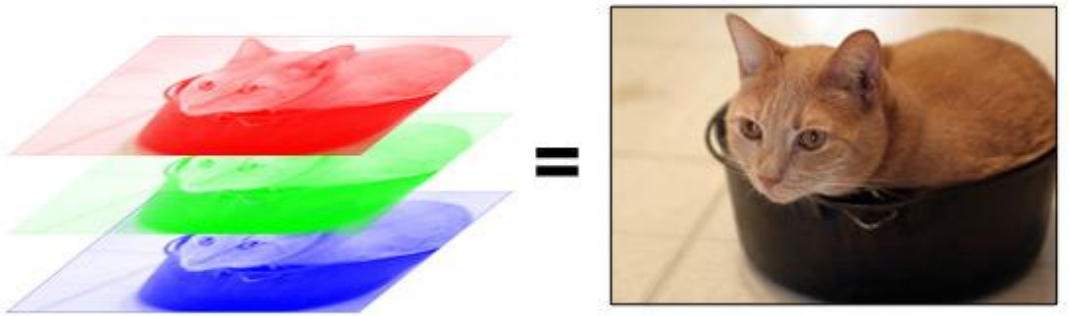
```
data = iio.imread('...')

data[rows, columns] = #
```

Color Images

When you overlap each color channel, it creates a picture we are used to seeing.

- Pixels on your monitor let out specified R/G/B light



```
data[rows, columns, channels] = #
```

Practice: Broadcasting

We started off with the following instructions

```
x = np.arange(3).reshape((3, 1))
y = np.arange(3)
x + y
```

```
x.shape = (3, 1)
y.shape = (3,)
```

```
x:  [[0],
     [1],
     [2]]
```

+

```
y:  [0, 1, 2]
```

Practice: Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
x.shape = (3, 1)
y.shape = (3,)
```

```
x:  [[0],
     [1],
     [2]]
```

+

```
y:  [0, 1, 2]
```

Practice: Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
x.shape = (3, 1)
y.shape = (1, 3)
```

```
x:  [[0],
     [1],
     [2]]
```

+

```
y:  [0, 1, 2]
```

Practice: Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
x.shape = (3, 1)
y.shape = (1, 3)
```

```
x:  [[0],
     [1],
     [2]]
```

+

```
y: [[0, 1, 2]]
```

Practice: Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
x.shape = (3, 3)
y.shape = (1, 3)
```

```
x:  [[0],
     [1],
     [2]]
```

+

```
y: [[0, 1, 2]]
```

Practice: Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
x.shape = (3, 3)
y.shape = (1, 3)
```

```
x:  [[0, 0, 0],
     [1, 1, 1],
     [2, 2, 2]]
```

+

```
y: [[0, 1, 2]]
```

Practice: Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
x.shape = (3, 3)
y.shape = (3, 3)
```

```
x:  [[0, 0, 0],
     [1, 1, 1],
     [2, 2, 2]]
```

+

```
y: [[0, 1, 2]]
```

Practice: Broadcasting

The Rules of Broadcasting

1. If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded on it's left side.
2. If the shape of two arrays does not match on any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If the sizes of any dimension disagree and neither is equal to 1, an error is raised.

```
x.shape = (3, 3)
y.shape = (3, 3)
```

```
x:  [[0, 0, 0],
     [1, 1, 1],
     [2, 2, 2]]
```

+

```
y: [[0, 1, 2],
    [0, 1, 2],
    [0, 1, 2]]
```