

# **NUMPY: THE FINAL CHAPTER**

9.26.2018

# RECAP: SLICING

```
>>> a[0,3:5]  
array([3,4])
```

```
>>> a[4:,4:]  
array([[44, 45],  
       [54, 55]])
```

```
>>> a[:,2]  
array([2,12,22,32,42,52])
```

```
>>> a[2::2,::2]  
array([[20,22,24],  
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

# RECAP: INDEXING

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]  
array([1, 12, 23, 34, 45])
```

```
>>> a[3:, [0,2,5]]  
array([[30, 32, 35],  
       [40, 42, 45],  
       [50, 52, 55]])
```

```
>>> mask = np.array([1,0,1,0,0,1], dtype=bool)  
>>> a[mask, 2]  
array([2, 22, 52])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

# RECAP: AGGREGATION

\* `np.min(arr)`, `np.sum(arr)`, etc.

# RECAP: AGGREGATION

- \* `np.min(arr, axis=0)`
- \* `np.min(arr, axis=1)`
- \* `etc.`

# ASSIGNMENT

- \* We can change the values inside an ndarray by assigning to them directly

- \* e.g.

```
>>> arr = np.arange(5)
```

```
>>> arr[0] = 17
```

```
>>> arr
```

```
array([17, 1, 2, 3, 4])
```

# ASSIGNMENT

- \* Assignment can also be done to a slice of an array

- \* e.g.

```
>>> arr = np.arange(5)
```

```
>>> arr[:3] = -1
```

```
>>> arr
```

```
array([-1, -1, -1, 3, 4])
```

# ASSIGNMENT

- \* These examples showed assignment of a single value to either a single element of the array or a slice
- \* You can also assign an array of values to a slice!



# ASSIGNMENT

\* e.g.

```
>>> arr = np.arange(5)
```

```
>>> arr[:3] = np.array([-1, 3, 17])
```

```
>>> arr
```

```
array([-1,  3, 17,  3,  4])
```

# COPY VS VIEW

- \* suppose we did this:
- \*

```
>>> arr = np.arange(10)
>>> lil_arr = arr[:5]
>>> lil_arr[0] = 23
>>> arr
???
```

# COPY VS VIEW

\* suppose we did this:

```
* >>> arr = np.arange(10)
>>> lil_arr = arr[:5]
>>> lil_arr[0] = 23
>>> arr
array([23, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

# COPY VS VIEW

- \* this happened because **slicing** always creates a **view** onto the same data
- \* e.g. `arr[0]` and `lil_arr[0]` both point to the exact same piece of memory!
- \* this works for any kind of slicing (with skips, etc.)

# COPY VS VIEW

- \* we can force the slice to point to a separate piece of memory using the **copy** method

- \* e.g.

```
>>> arr = np.arange(10)
```

```
>>> lil_arr = arr[:5].copy()
```

# COPY VS VIEW

\* now suppose we did this:

```
* >>> arr = np.arange(10)
>>> inds = [0,1,2,3,4]
>>> ind_arr = arr[inds]
>>> ind_arr[0] = 23
>>> arr
???
```

# COPY VS VIEW

\* now suppose we did this:

```
* >>> arr = np.arange(10)
>>> inds = [0,1,2,3,4]
>>> ind_arr = arr[inds]
>>> ind_arr[0] = 23
>>> arr
array([0,1,2,3,4,5,6,7,8,9])
```

# COPY VS VIEW

- \* this happened because **indexing** always creates a copy of the data
- \* e.g. `arr[0]` and `ind_arr[0]` point to different places in memory



# PROBLEM SET 2

- \* will be assigned on Friday (at some point)
- \* stay tuned!

**END**