NUMPY III:
THE SEARCH FOR MIN

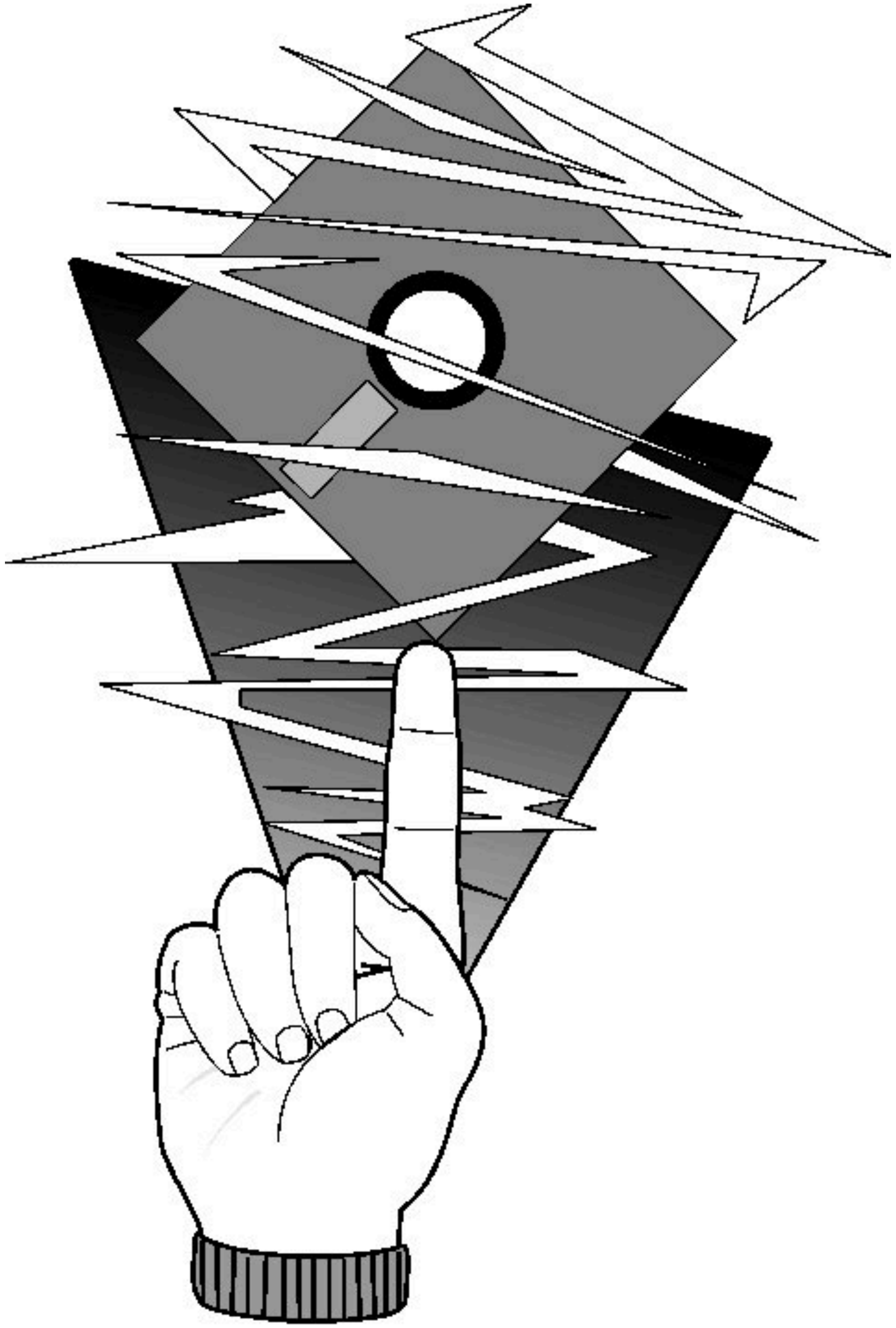
9.18.2020

PROBLEM SET 1

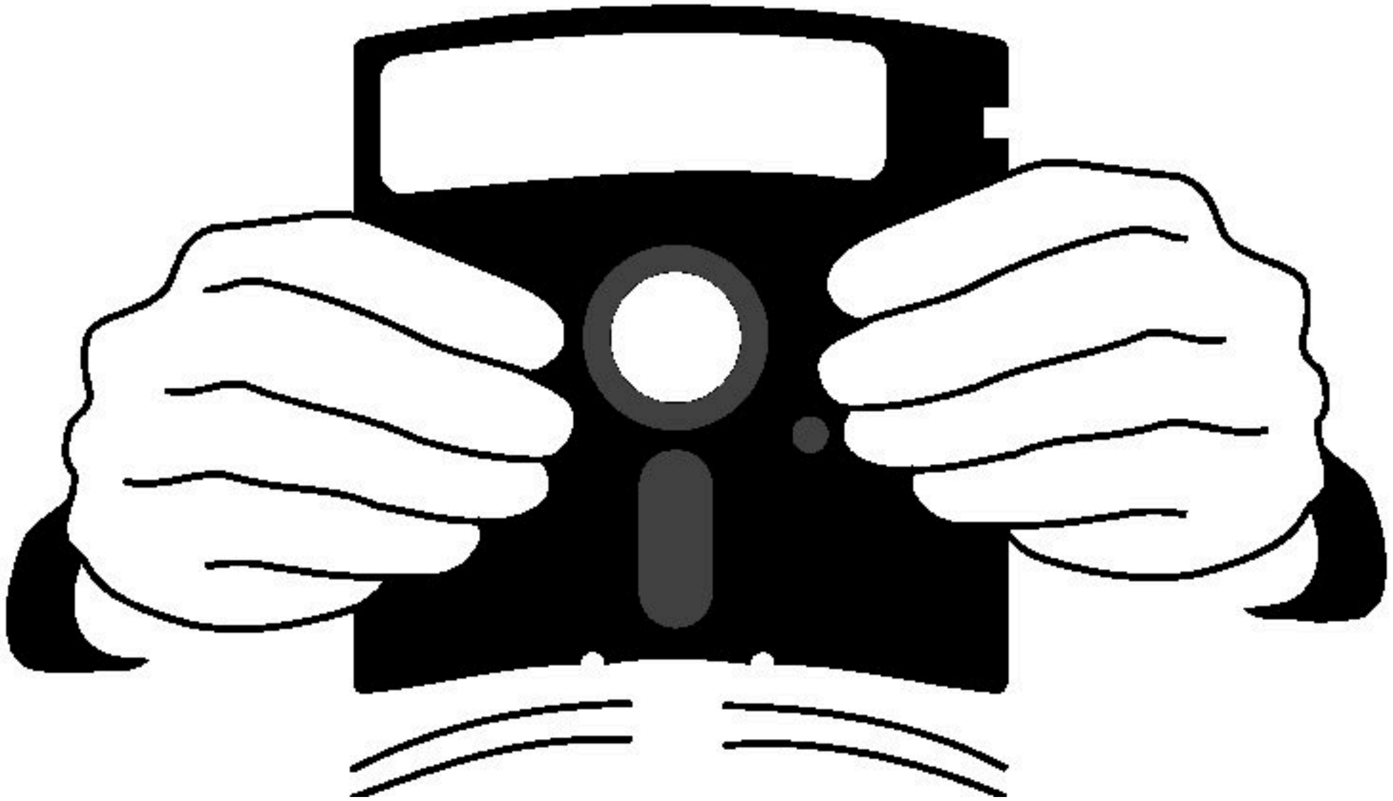
* Was due before the beginning of class!

* So you’re probably done with it!

* Yay!



PROBLEM SET 2



* Will be assigned
  today :)

REMINDER

* Please read chapter 2 of the Python Data Science Handbook: https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html

* (This will be extremely useful for homework 2!)

BINARY INDEXING REVISITED

- To recap: ndarrays can be indexed *by ndarrays (or lists) of booleans*

- this creates a new array containing all the elements where the index array was True

- e.g.

```
>>> arr = np.arange(4)
>>> inds = [True, False, True, False]
>>> arr[inds]
array([0, 2])
```

BOOLEAN COMPARISONS REVISITED

* Using boolean operators (>, <, ==, etc.) on arrays performs a check on every element of the array separately

* e.g.

  ```
  >>> arr = np.array([0, 1, -1])
  >>> arr >= 1
  array([False, True, False])
  ```

BINARY INDEXING REVISITED

* Combining binary indexing with boolean comparisons enables really nifty things

* e.g.

```
>>> arr1[arr2 >= 1]
```

BINARY INDEXING REVISITED

- **CAUTIONARY NOTE**

- Although binary (True and False) can sometimes be treated as 1 and 0, they are NOT the same for indexing purposes

- e.g.

```
>>> arr[[True, False, True]]
```

is not the same as

```
>>> arr[[1, 0, 1]]
```

AGGREGATION

* Suppose you want to find the smallest value (aka the minimum) across all the elements in an array

* Numpy provides a way to do this (and many other things!) through **aggregation** functions

* e.g.

  ```
  >>> arr = np.array([0.3, 1.7, 0.2, 0.1])
  >>> arr.min()
  0.1
  ```

AGGREGATION

* Somewhat confusingly, there are two versions of each aggregation function

* e.g.

```
>>> arr.min()
>>> np.min(arr)
```

* ^ these both do the same thing

AGGREGATION

* There are many aggregation functions:

* sum, prod, mean, std, var, min, max, argmin, argmax, median, any, all

* Each of these functions **reduces a set of numbers to a single number**

AGGREGATION

- By default, any aggregation operation runs over the ***entire*** array, giving you a single number regardless of the number of dimensions, etc.

AGGREGATION

* But you can control this using the “axis” parameter to the aggregation function

* e.g.

```
>>> arr = np.array([[0,3,5],[1,2,-1]])

>>> arr.min(axis=0)
array([0,2,-1])

>>> arr.min(axis=1)
array([0,-1])
```

AGGREGATION

- When you select an “axis” for aggregation, that is the axis that ends up being reduced to a single number

- Thus, aggregation effectively removes a dimension from the array, much like indexing!

AGGREGATION

```
* >>> arr.shape
  (24, 2985)

  >>> arr.min().shape # ?

  >>> arr.min(axis=0).shape # ?

  >>> arr.min(axis=1).shape # ?

  >>> arr.min(axis=2).shape # ?
```

END