Hashing and Plotting

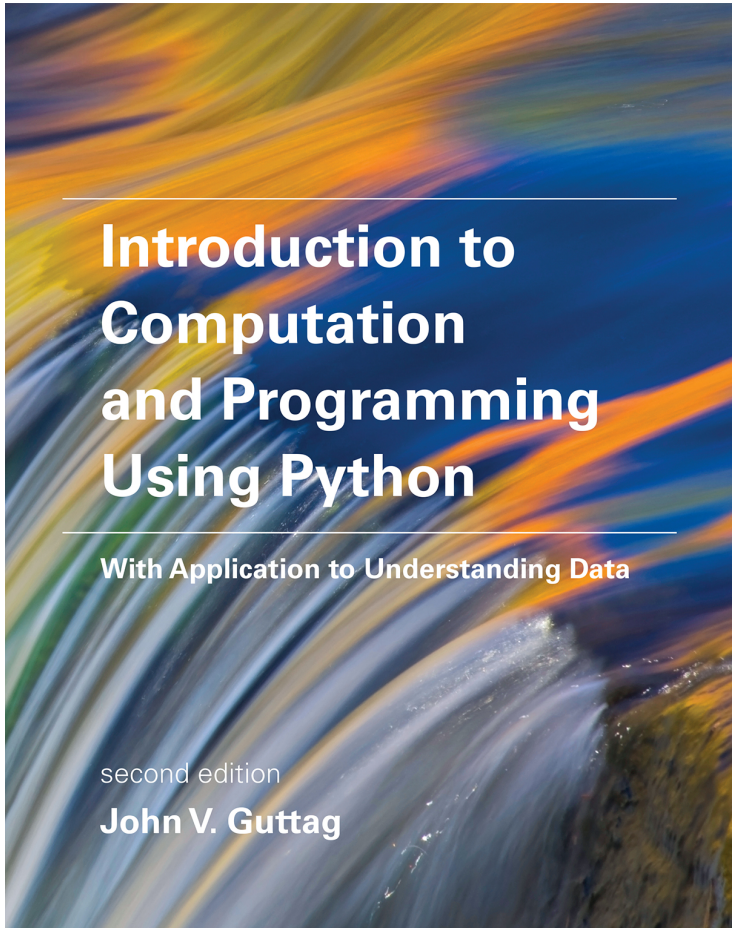
John Guttag

(download slides and .py files to follow along!)

6.0001 LECTURE 11

Assigned Reading

- Today
  - 10.3
  - 11
- Reminder: 6.0001 final a week from Wednesday



HALF-TERM EVALUATIONS

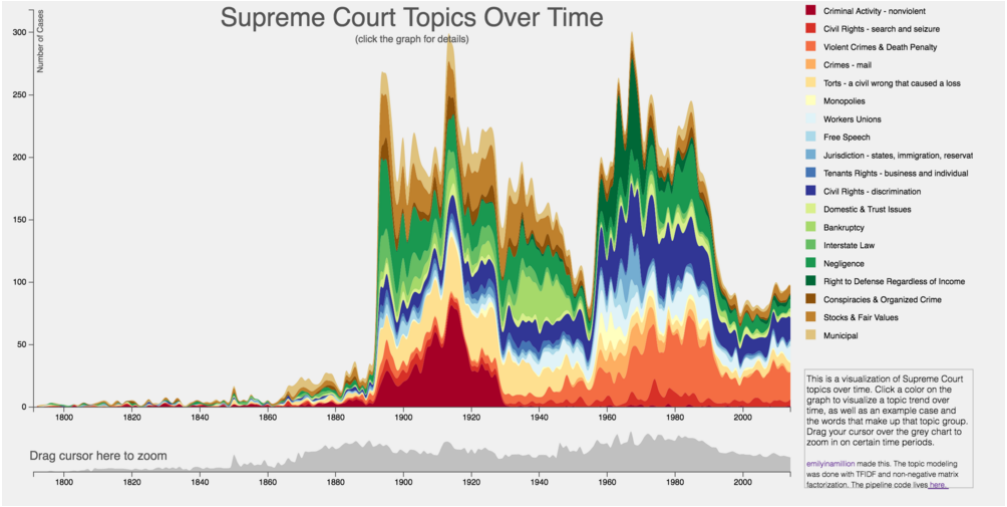
- You have until 9AM on March 16 to evaluate 6.0001
  - http://web.mit.edu/subjectevaluation/evaluate.html

TODAY

- Hashing

= 37984321

- Plotting



HASHING

COMPLEXITY OF SOME PYTHON OPERATIONS (RECAP)

- Lists: `n` is `len(L)`
  - index O(1)
  - store O(1)
  - length O(1)
  - append O(1)
  - == O(n)
  - remove O(n)
  - copy O(n)
  - reverse O(n)
  - iteration O(n)
  - in list O(n)

- Dictionaries: `n` is `len(d)`
- worst case (very rare)
  - index O(n)
  - store O(n)
  - length O(n)
  - delete O(n)
  - iteration O(n)
- average case
  - index O(1)
  - store O(1)
  - delete O(1)
  - iteration O(n)

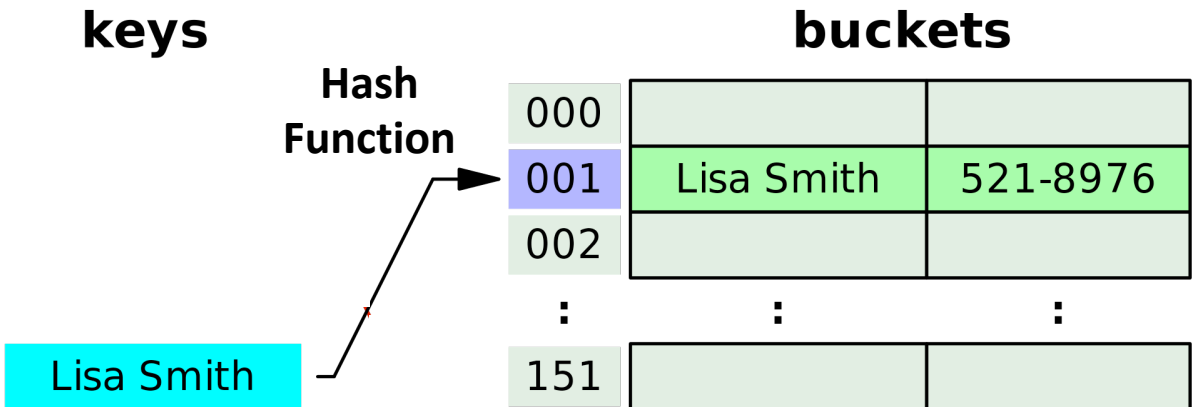
Why?

Dictionaries Implemented Using Hash Tables

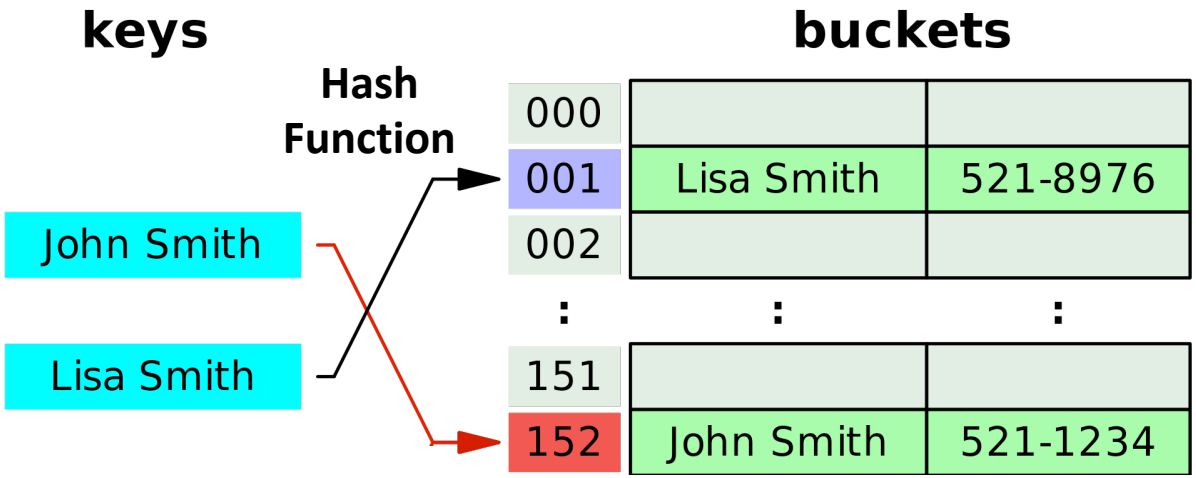
DICTIONARIES IMPLEMENTED USING HASH TABLES

- Convert value of an object to an integer
- Use that integer to index into a list

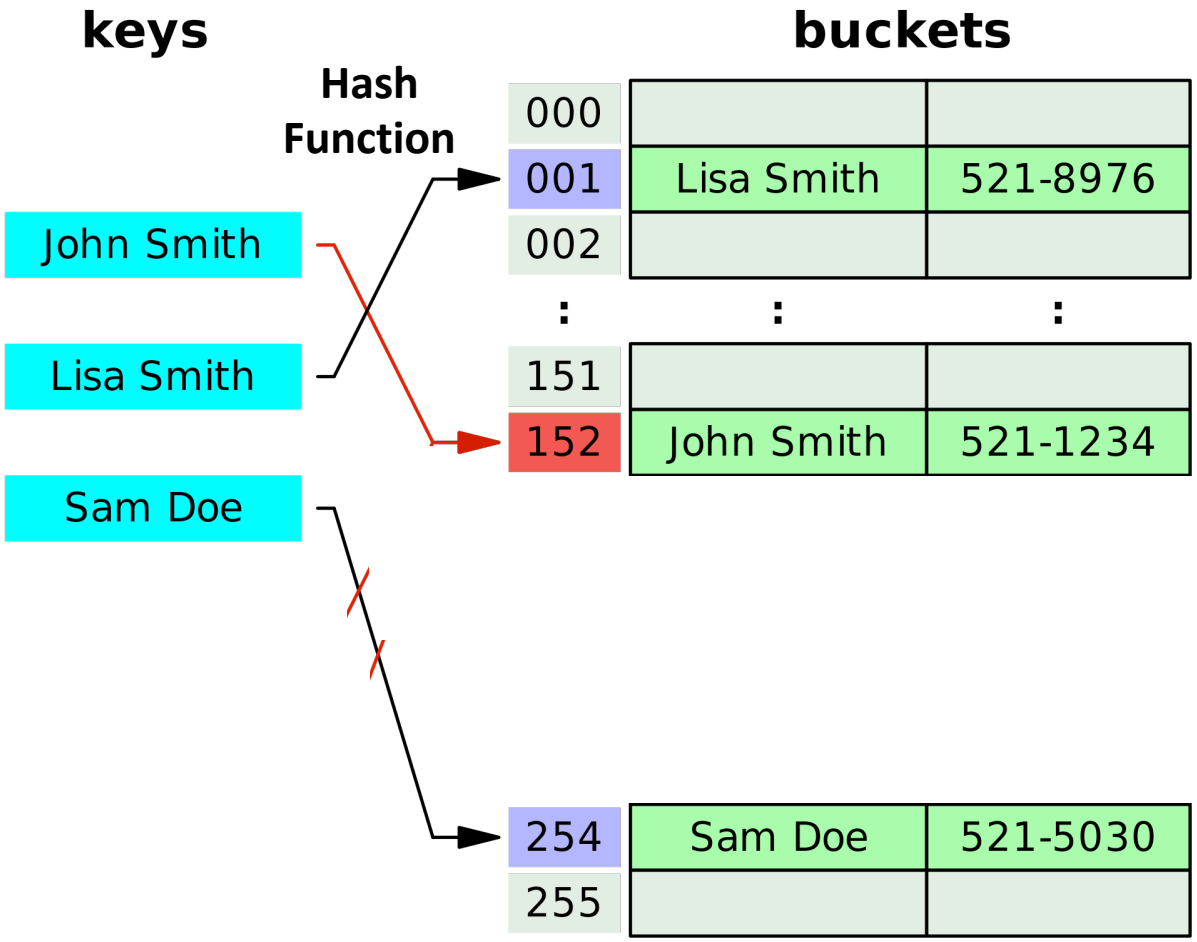
HASH TABLES


keys
buckets
Hash
Function
000
001
002
151
Lisa Smith
Lisa Smith
521-8976

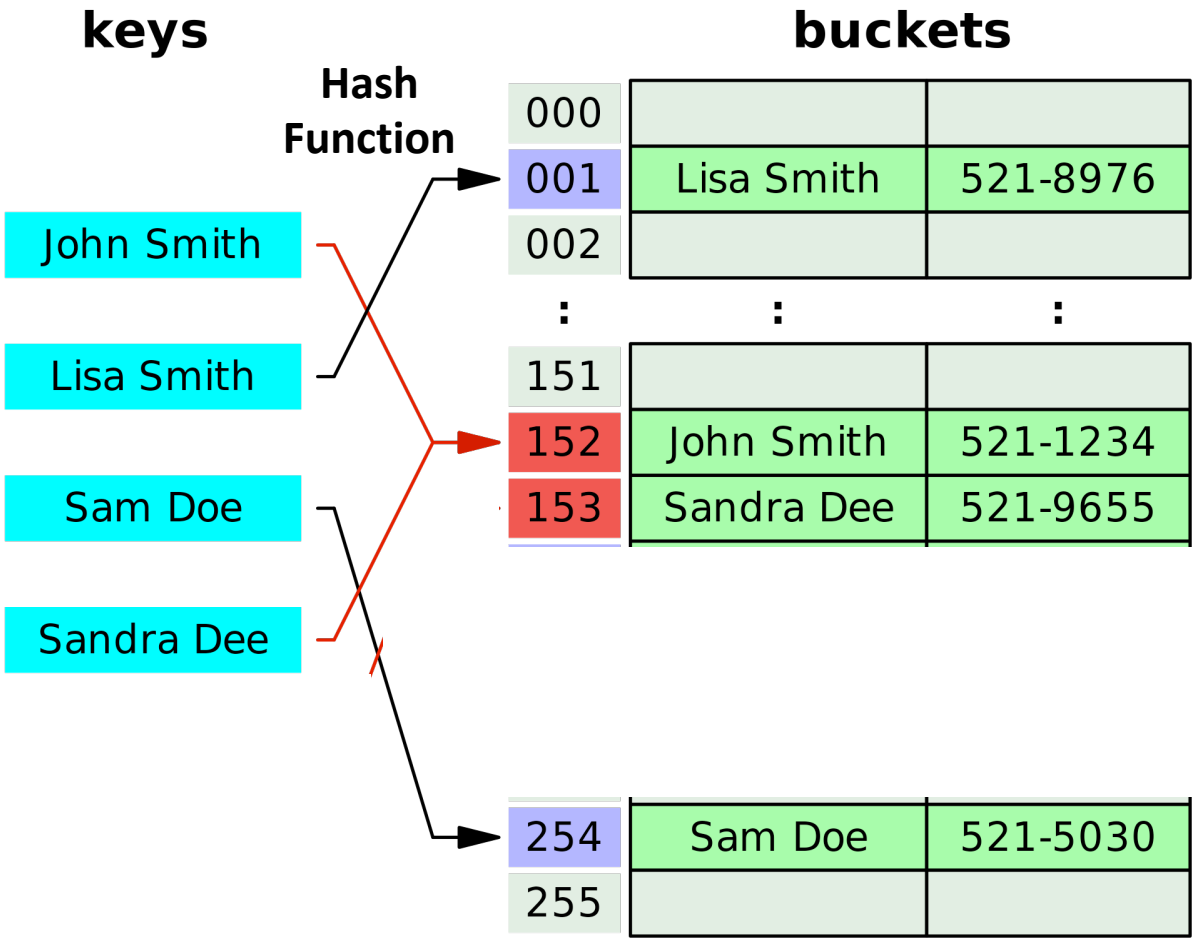
HASH TABLES


keys
buckets
Hash
Function
John Smith
Lisa Smith
000
001
002
151
152
Lisa Smith
521-8976
John Smith
521-1234

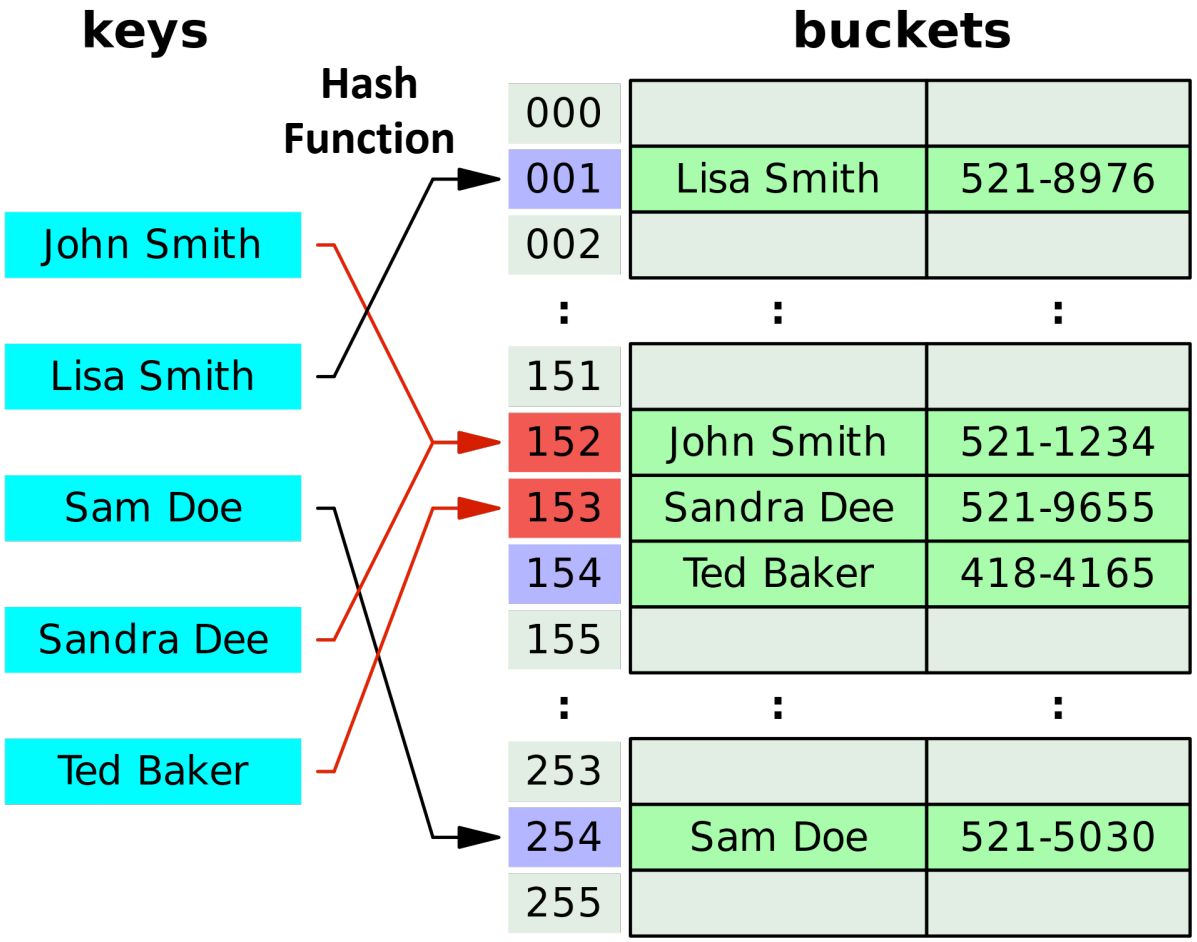
HASH TABLES


keys
Hash Function
buckets
John Smith
Lisa Smith
Sam Doe
000
001
002
151
152
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sam Doe
521-5030

HASH TABLES


keys
Hash Function
buckets
John Smith
Lisa Smith
Sam Doe
Sandra Dee
000
001
002
151
152
153
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Sam Doe
521-5030

HASH TABLES


keys
Hash Function
buckets
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
155
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030

NAMES TO INDICES

- E.g., ‘Ana Bell’

  = 01000001 01101110 01100001 00100000
  01000010 01100101 01101100 01101100

  = 4,714,812,651,084,278,892

- **Advantage**: unique names mapped to **unique indices**
- **Disadvantage**: VERY **space inefficient**
- Consider a table containing MIT’s ~4,000 undergraduates
  - Assume longest name is 20 characters
  - Each character 8 bits, so 160 bits per name
  - How many entries will table have? 2^{160}

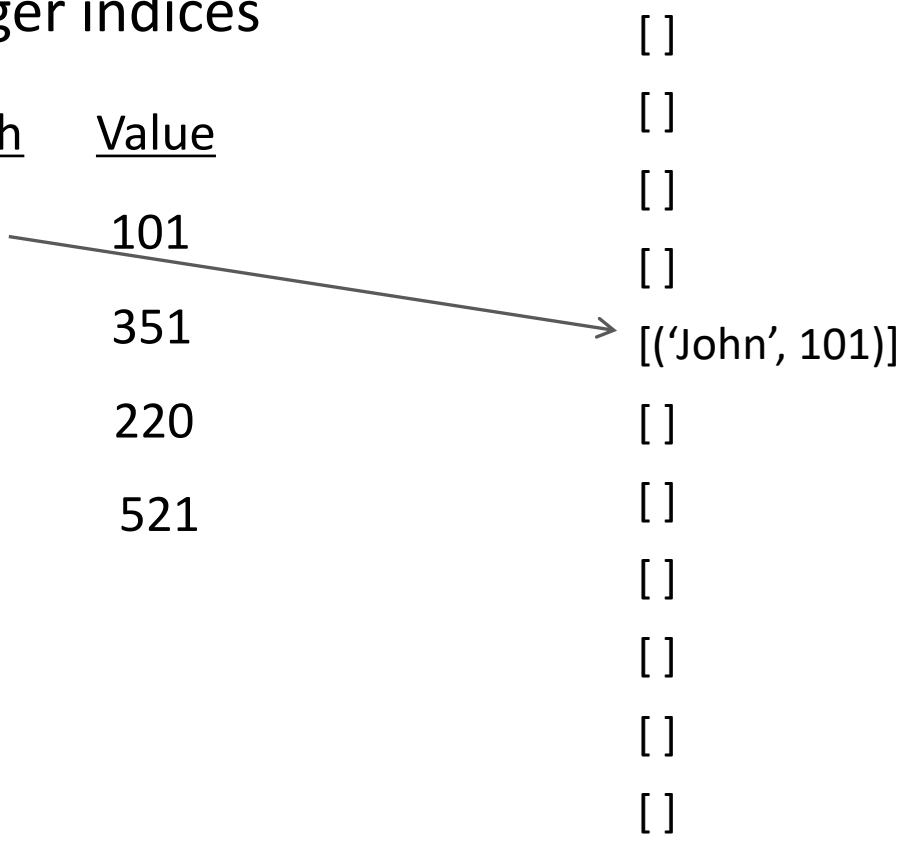
How Big is 2^{160}?

A BETTER IDEA: ALLOW COLLISIONS

A BETTER IDEA

- Use a **hash function** that allows **collisions**
- Hash function maps a large space of keys to smaller space of integer indices

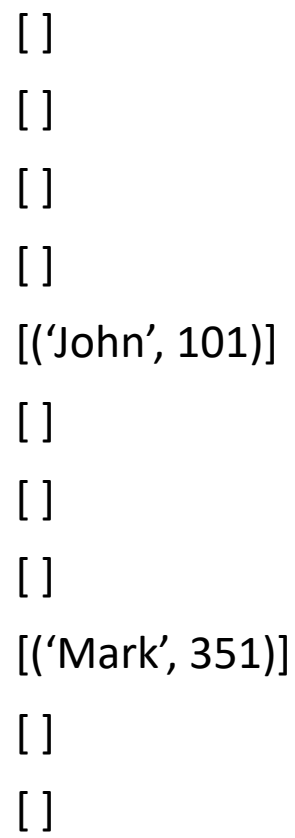
Key	Hash	Value
John	4	101
Mark		351
Andrea		220
Mike		521



A BETTER IDEA

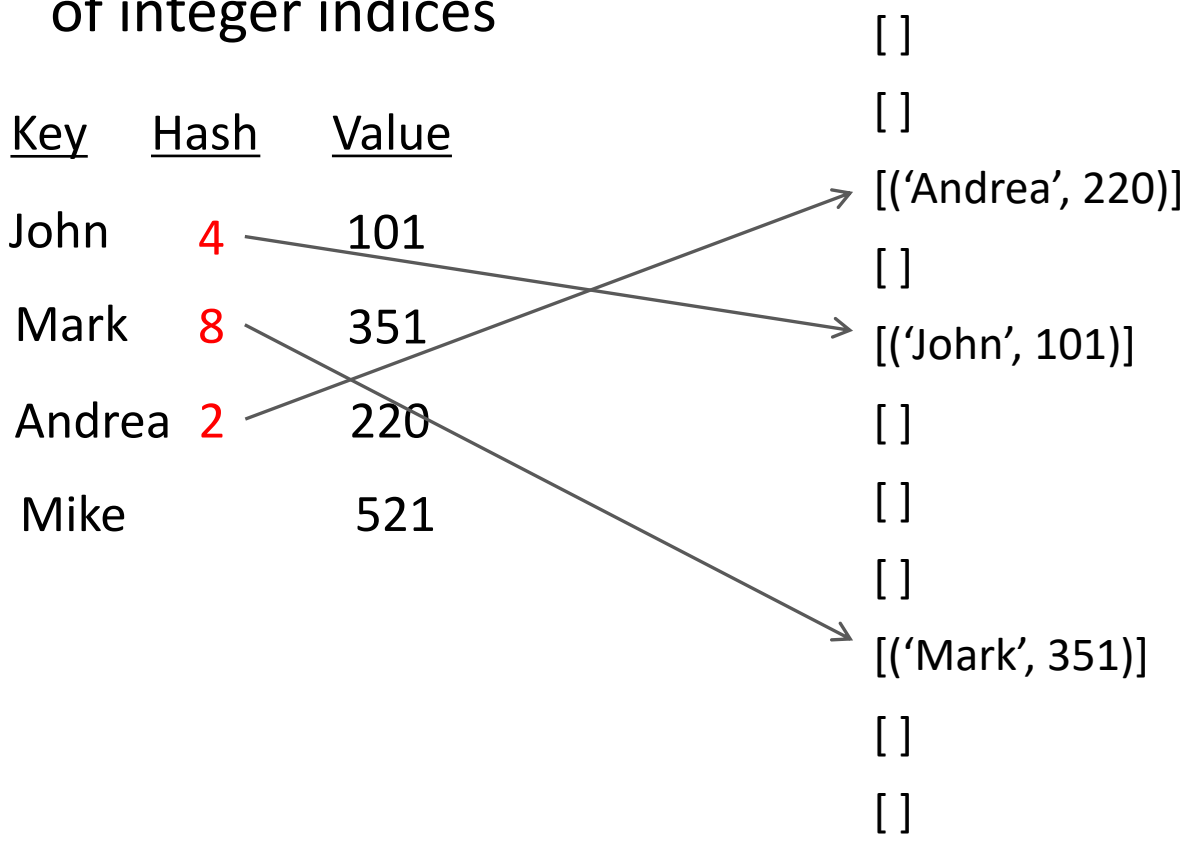
- Use a **hash function** that allows **collisions**
- Hash function maps a large space of keys to smaller space of integer indices

Key	Hash	Value
John	4	101
Mark	8	351
Andrea		220
Mike		521



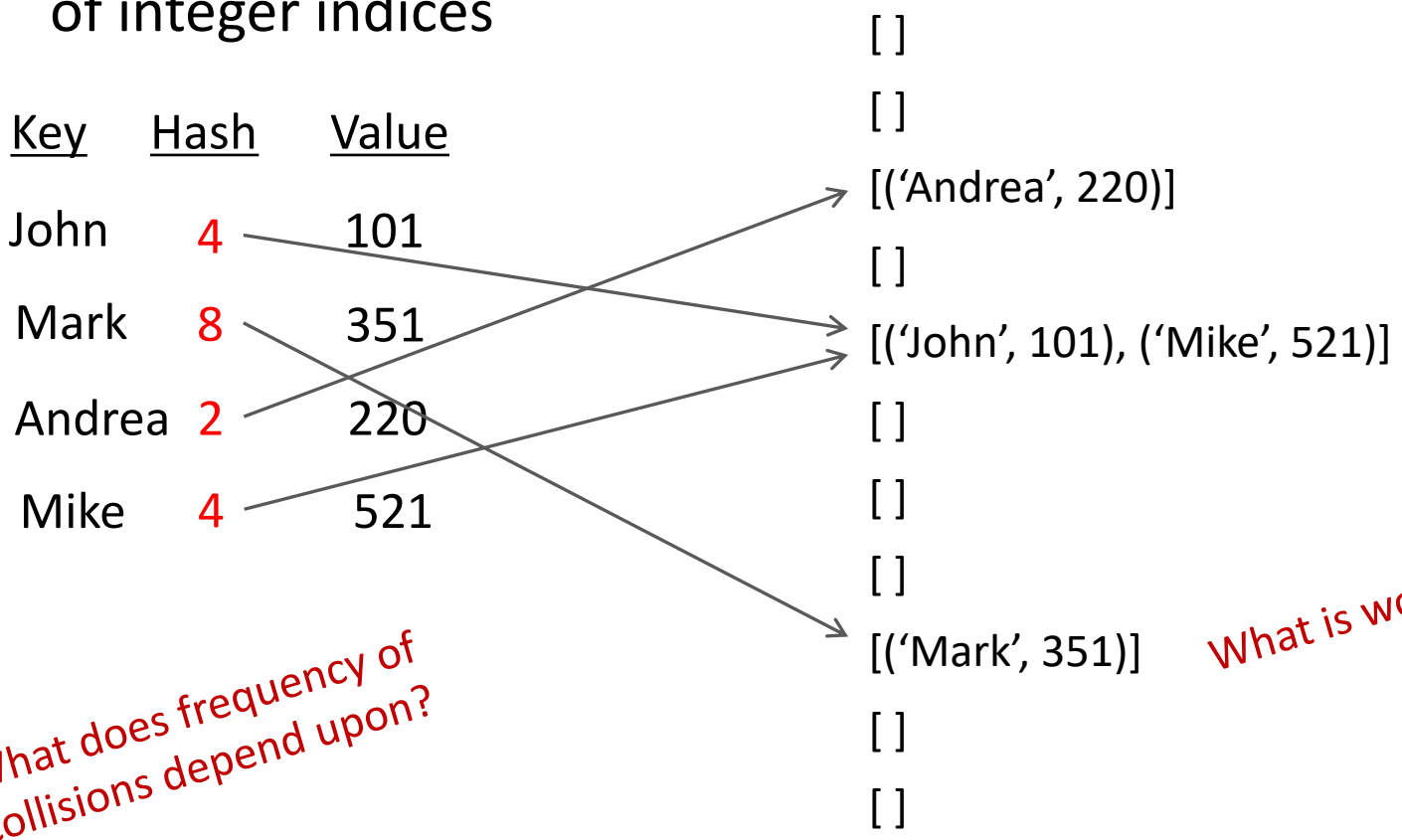
A BETTER IDEA

- Use a **hash function** that allows **collisions**
- Hash function maps a large space of keys to smaller space of integer indices



A BETTER IDEA

- Use a **hash function** that allows **collisions**
- Hash function maps a large space of keys to smaller space of integer indices



PROPERTIES OF A GOOD HASH FUNCTION

- Maps domain of interest to integers between 0 and size of hash table
- The hash value is **fully determined by value** being hashed (nothing random)
- The hash function uses the entire input to be hashed
- **Distribution of values is uniform**, i.e., equally likely to land on any entry in hash table
- Side Reminder: keys in a dictionary must be hashable, e.g., immutable, so that they always hash to the same value

COMPLEXITY OF SOME PYTHON OPERATIONS (RECAP)

- Dictionaries: `n` is `len(d)`
- worst case (very rare)
  - index O(n)
  - store O(n)
  - length O(n)
  - delete O(n)
  - iteration O(n)
- average case
  - index O(1)
  - store O(1)
  - delete O(1)
  - iteration O(n)

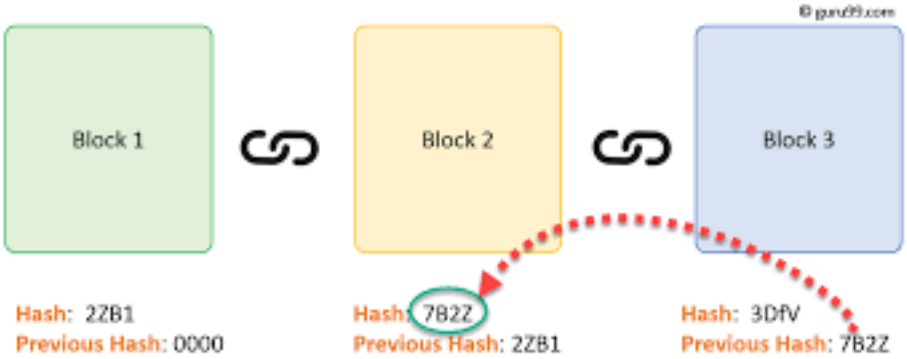
If all keys hash to the same index

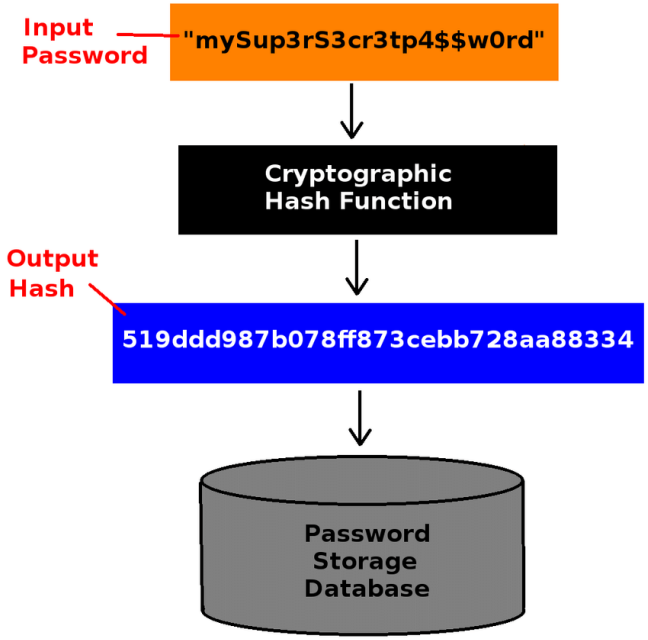
Hash table is large relative to number of keys
Hash function good enough

Hashing Not Just for dicts

- Used to convert any digitized value to an integer

= 3798432





Onto Plotting

An Early Break

WHY PLOTTING?

- Sooner or later, everyone needs to produce plots
  - As we look at data in 6.0002 half of term, we will make extensive use of them
  - For those of you leaving us after next week, this is a nice way to visualize data
- Example of **leveraging an existing library**, rather than writing procedures from scratch
  - See problem set 5
- Python provides libraries for (among other topics):
  - Plotting
  - Numerical computation
  - Stochastic computation

Very similar to Matlab

6.0002

Matplotlib

- Can **import library** into computing environment

```
import matplotlib.pyplot as plt
```

  - Allows **code to reference library** procedures as `plt.<procName>`

- Provides access to existing set of graphing/plotting procedures

- Here will just show some simple examples; lots of additional information available in documentation associated with `pyplot`

- Will see many other examples and details of these ideas if you opt to take 6.0002

A SIMPLE EXAMPLE

```
nVals = []
linear = []
quadratic = []
cubic = []
exponential = []

for n in range(0, 30):
    nVals.append(n)
    linear.append(n)
    quadratic.append(n**2)
    cubic.append(n**3)
    exponential.append(1.5**n)
```

Used 1.5 to keep displays visible, more common value for order of growth example would be 2

PLOTTING THE DATA

- To generate a plot:

Typically N

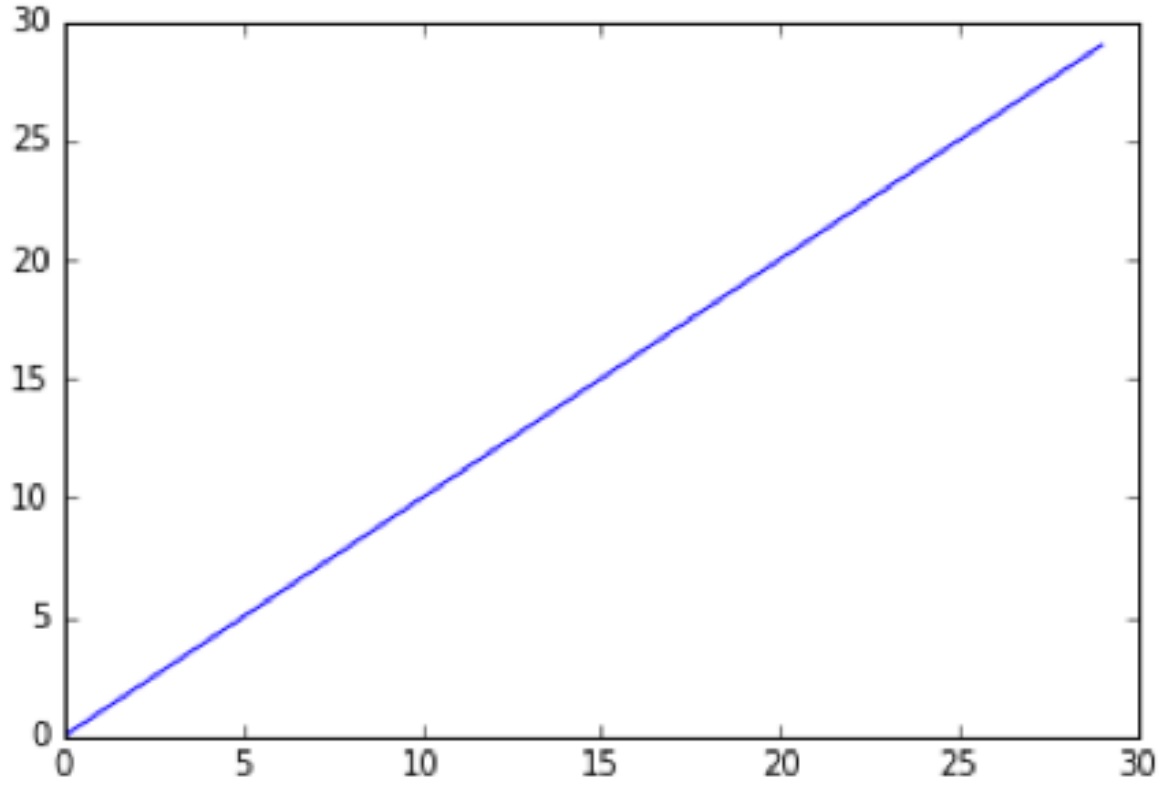
Typically a function of N

```
plt.plot(x values, y values)
```

- Arguments are lists of numbers (for now)
  - lists must be of the same length
  - Generates a sequence of <x, y> values
  - Plotted in order, then connected with lines
- Calling function in an iPython console will generate plots **in different places depending upon preferences setting**

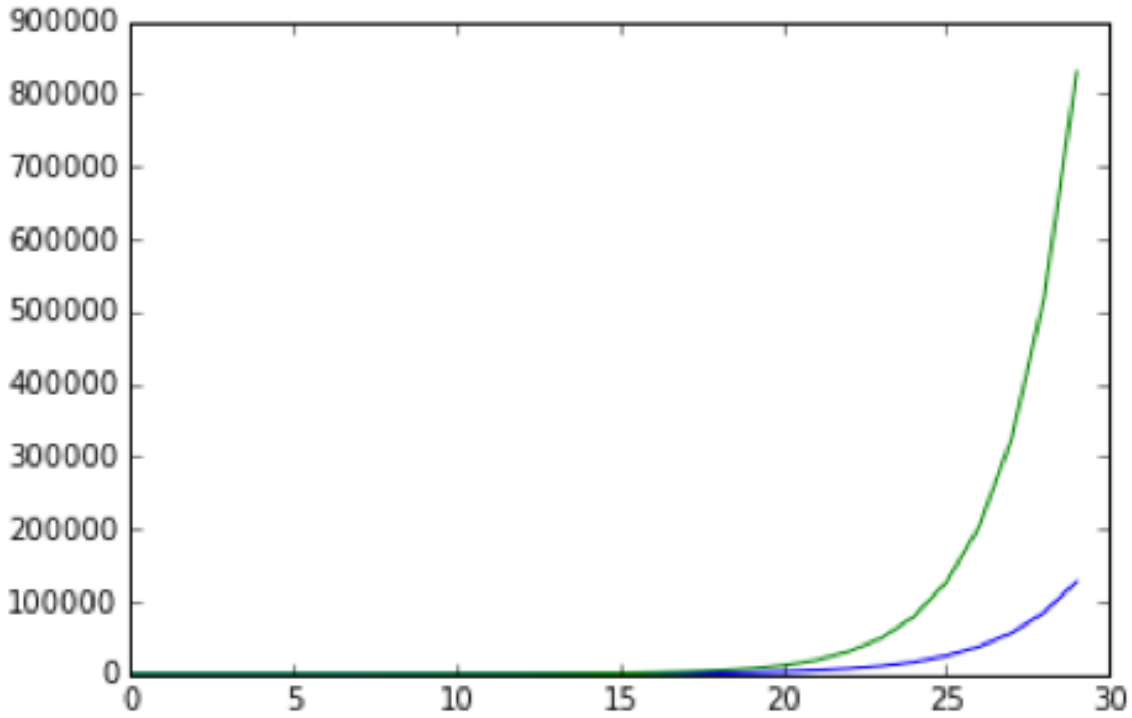
EXAMPLE

```
plt.plot(nVals, linear)
```



SHOWING ALL DATA ON ONE PLOT

```
plt.plot(nVals, linear)
plt.plot(nVals, quadratic)
plt.plot(nVals, cubic)
plt.plot(nVals, exponential)
```



Impossible to see linear graph, or even quadratic graph

Problem is that scale is so different

PRODUCING MULTIPLE PLOTS

- Let's graph each one separately
- Call

```
plt.figure(<arg>)
```

gives a name to this figure; allows us to reference for future use

  - Creates a new display with that name if one does not already exist
  - If a display with that name exists, reopens it for processing

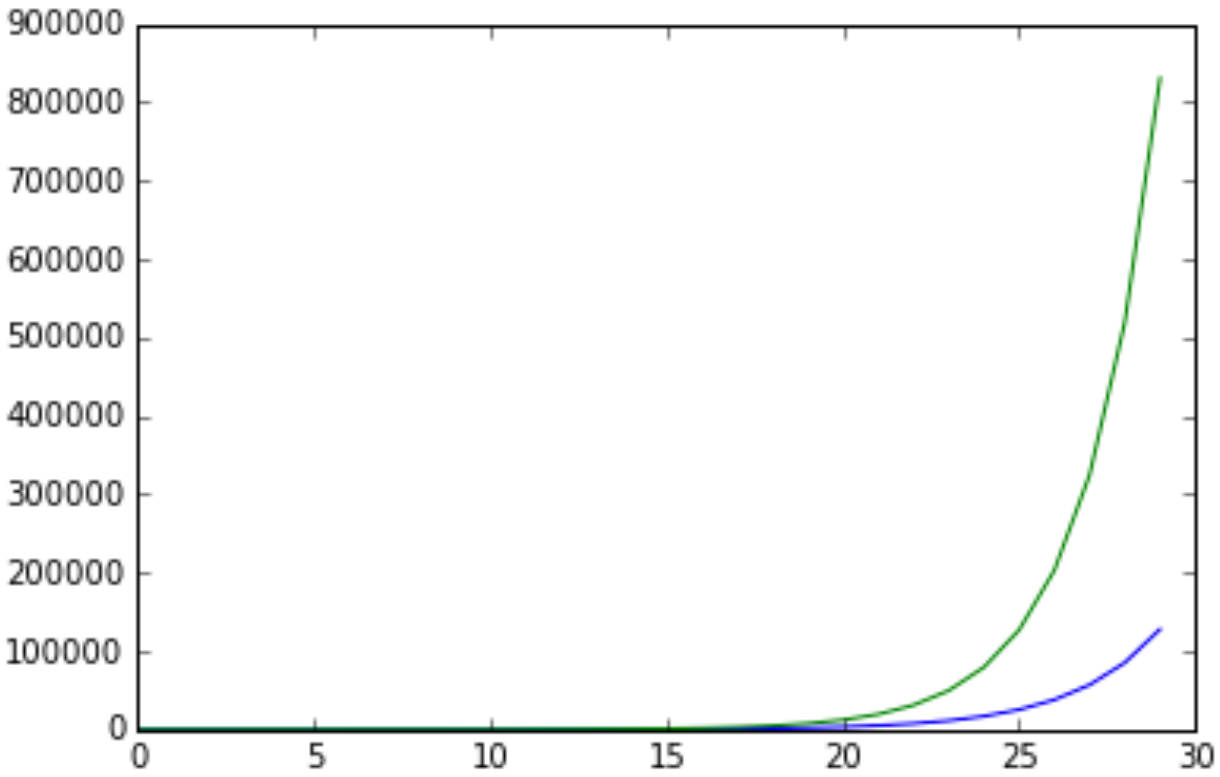
EXAMPLE CODE

```
plt.figure('expo')
plt.plot(nVals, exponential)
plt.figure('lin')
plt.plot(nVals, linear)
plt.figure('quad')
plt.plot(nVals, quadratic)
plt.figure('cube')
plt.plot(nVals, cubic)
newExpo = []
for i in range(30):
    newExpo.append(1.6**i)
plt.figure('expo')
plt.plot(nVals, newExpo)
```

Go back to expo

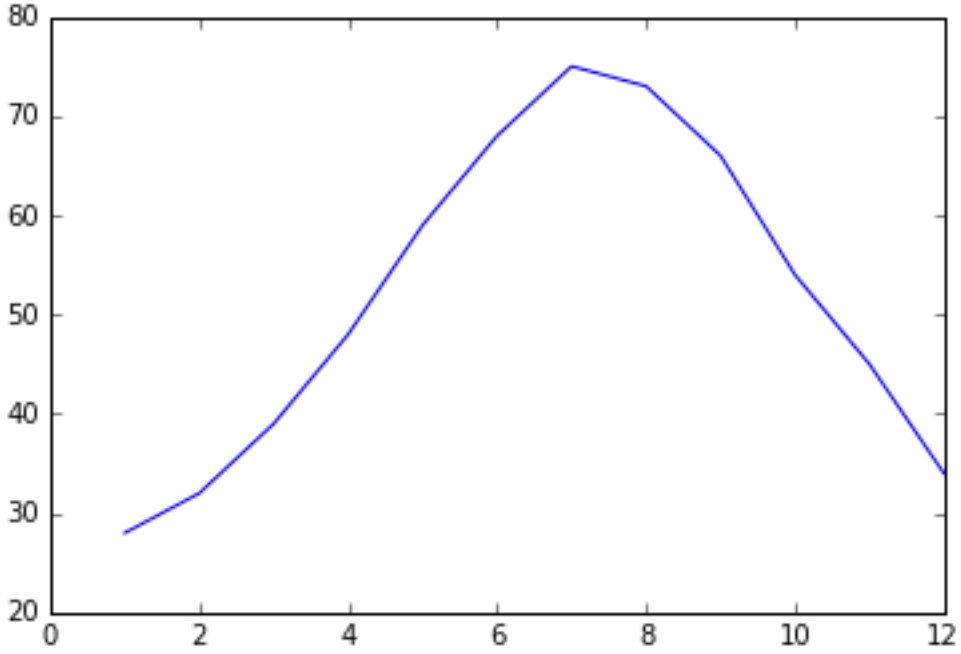
Add on to that figure

PLOT WITH EXPONENTIALS



A "REAL" EXAMPLE

```
months = range(1, 13, 1)
temps = [28,32,39,48,59,68,75,73,66,54,45,34]
plt.plot(months, temps)
```



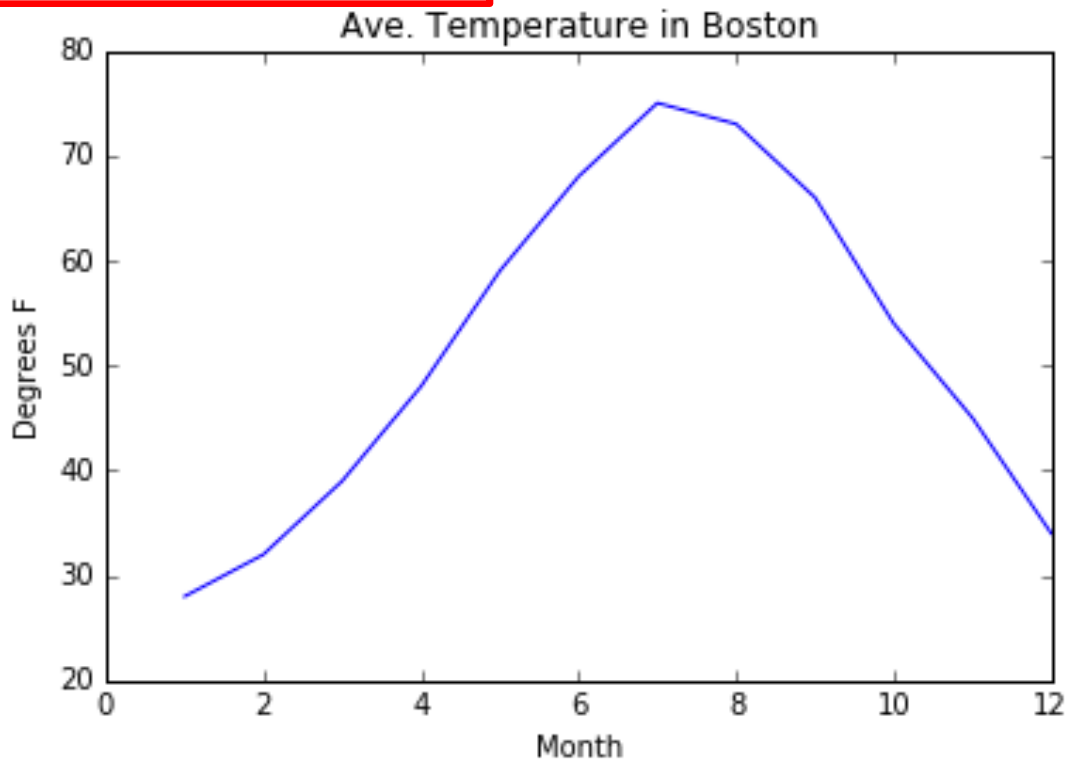
But what is this trying to tell us?

A "REAL" EXAMPLE

```
months = range(1, 13, 1)
temps = [28,32,39,48,59,68,75,73,66,54,45,34]
plt.plot(months, temps)

plt.title('Ave. Temperature in Boston')
plt.xlabel('Month')
plt.ylabel(('Degrees F'))
```

Still a bit weird looking

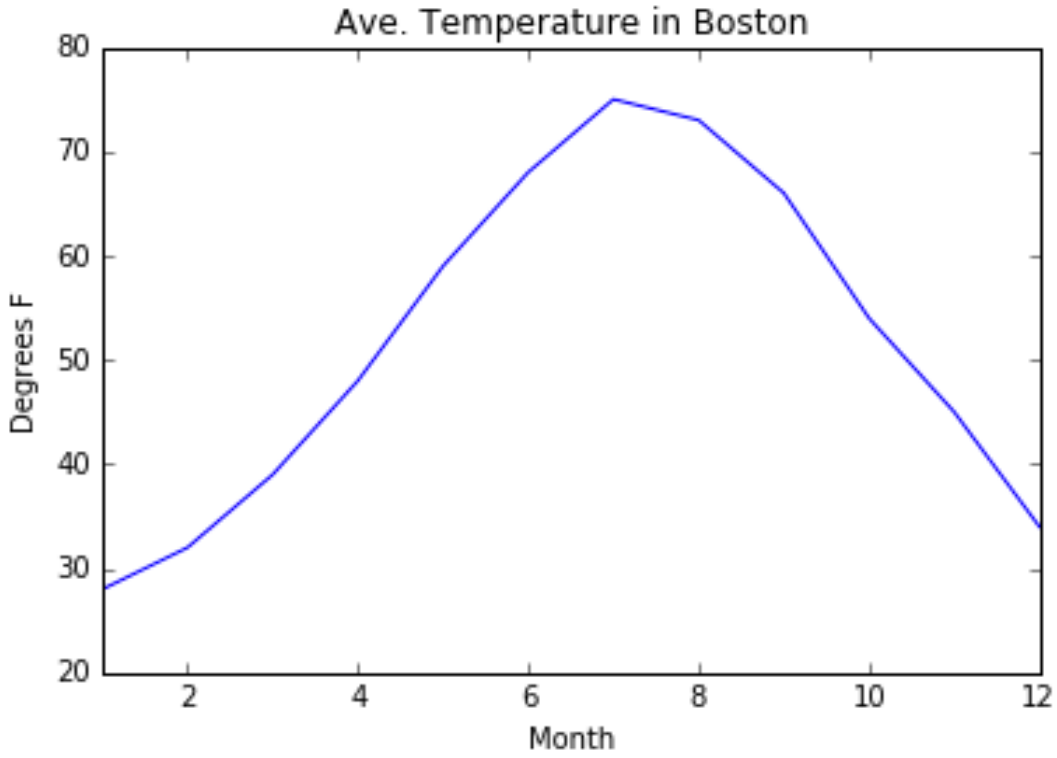


A "REAL" EXAMPLE

```
months = range(1, 13, 1)
temps = [28,32,39,48,59,68,75,73,66,54,45,34]
plt.plot(months, temps)

plt.title('Ave. Temperature in Boston')
plt.xlabel('Month')
plt.ylabel(('Degrees F'))
plt.xlim(1, 12)
```

Suppose I want to see each month on x-axis?



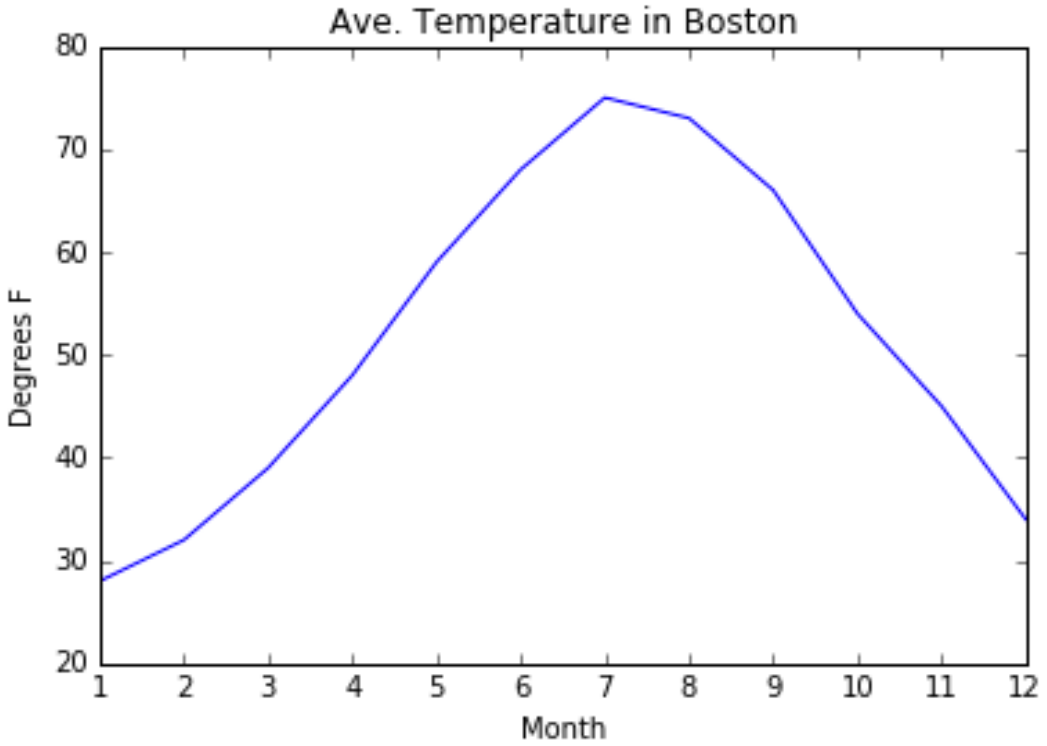
A "REAL" EXAMPLE

```
months = range(1, 13, 1)
temps = [28,32,39,48,59,68,75,73,66,54,45,34]
plt.plot(months, temps)

plt.title('Ave. Temperature in Boston')
plt.xlabel('Month')
plt.ylabel(('Degrees F'))
```

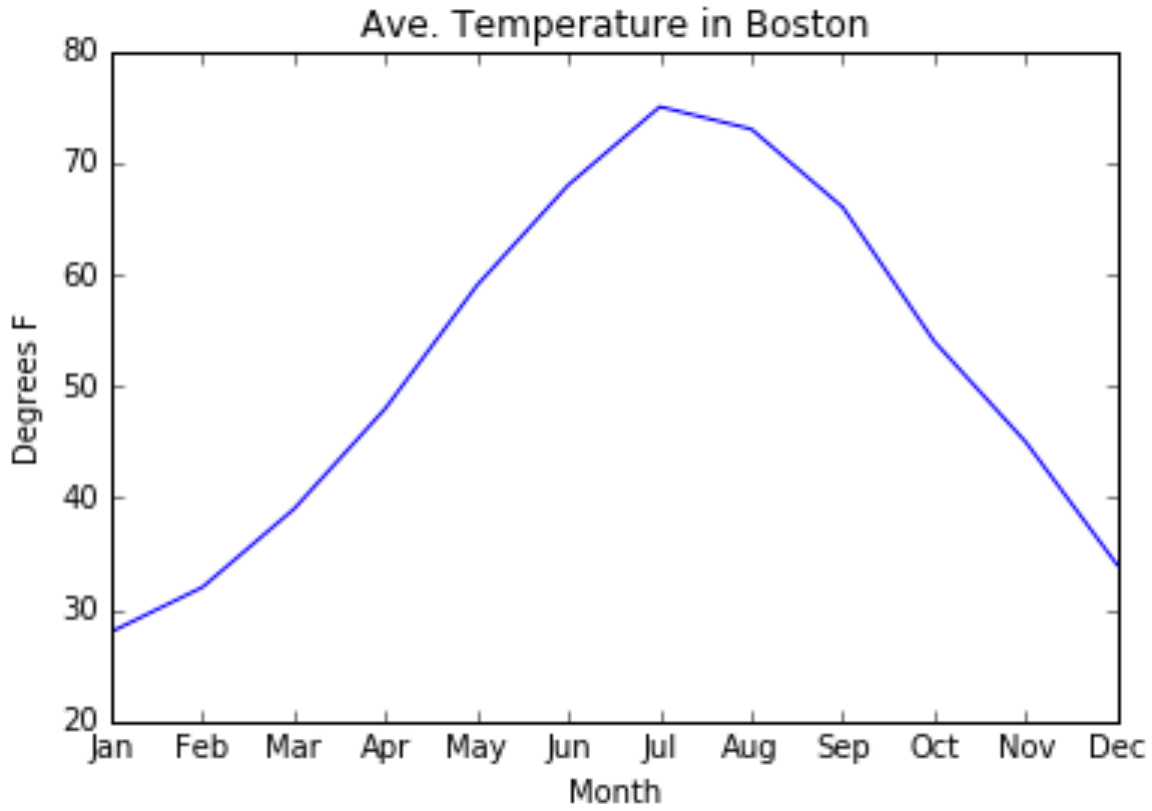
```
plt.xticks((1,2,3,4,5,6,7,8,9,10,11,12))
```

But what about those who can't map numbers to months?



A "REAL" EXAMPLE

```
plt.xticks((1,2,3,4,5,6,7,8,9,10,11,12),
           ('Jan','Feb','Mar','Apr','May','Jun',\
           'Jul','Aug','Sep','Oct','Nov','Dec'))
```

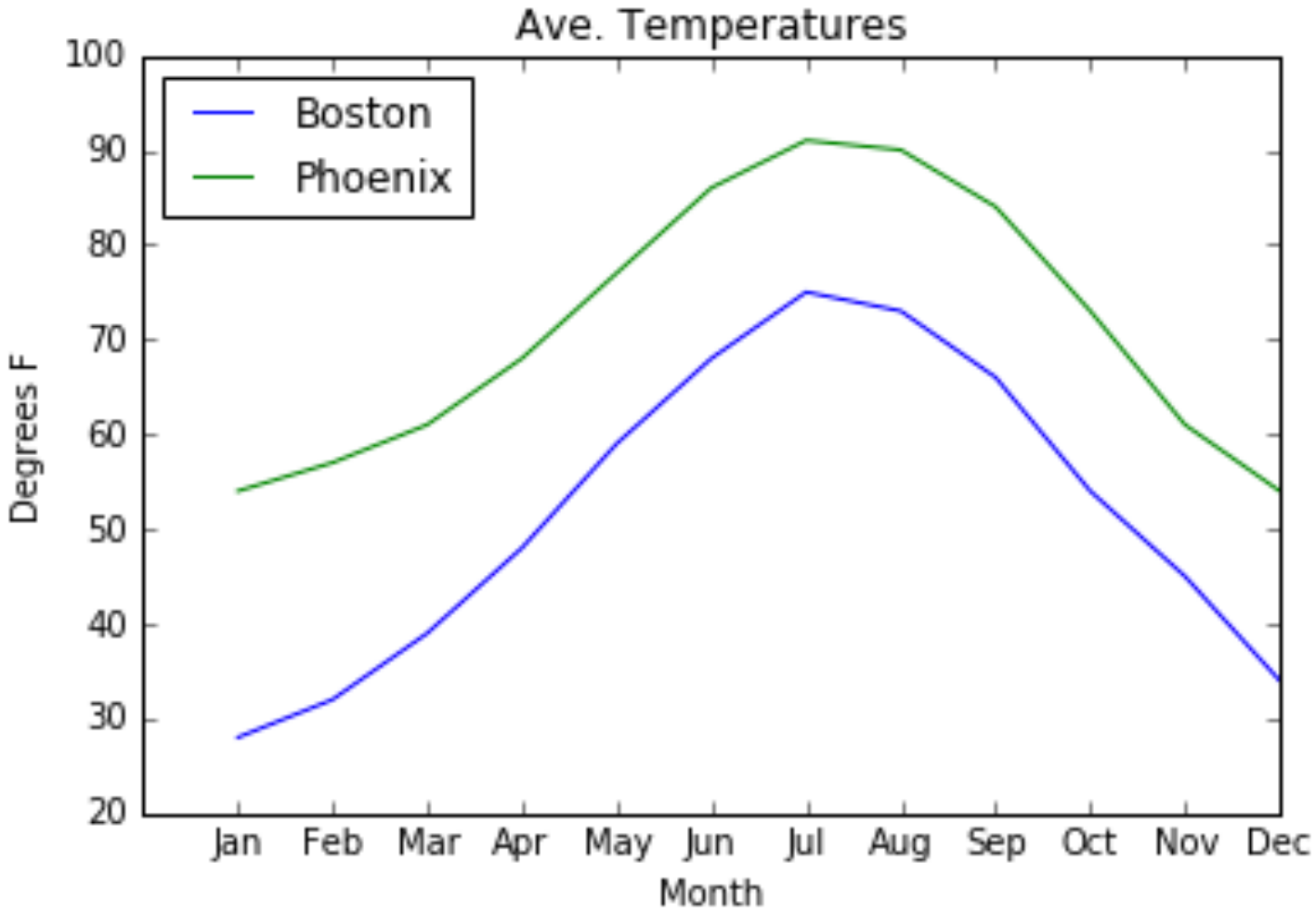


LET'S ADD ANOTHER CITY

```
months = range(1, 13, 1)
boston = [28,32,39,48,59,68,75,73,66,54,45,34]
plt.plot(months, boston, label = 'Boston')
phoenix = [54,57,61,68,77,86,91,90,84,73,61,54]
plt.plot(months, phoenix, label = 'Phoenix')
plt.legend(loc = 'best')

plt.title('Ave. Temperatures')
plt.xlabel('Month')
plt.ylabel(('Degrees F'))
```

PLOT WITH TWO CURVES



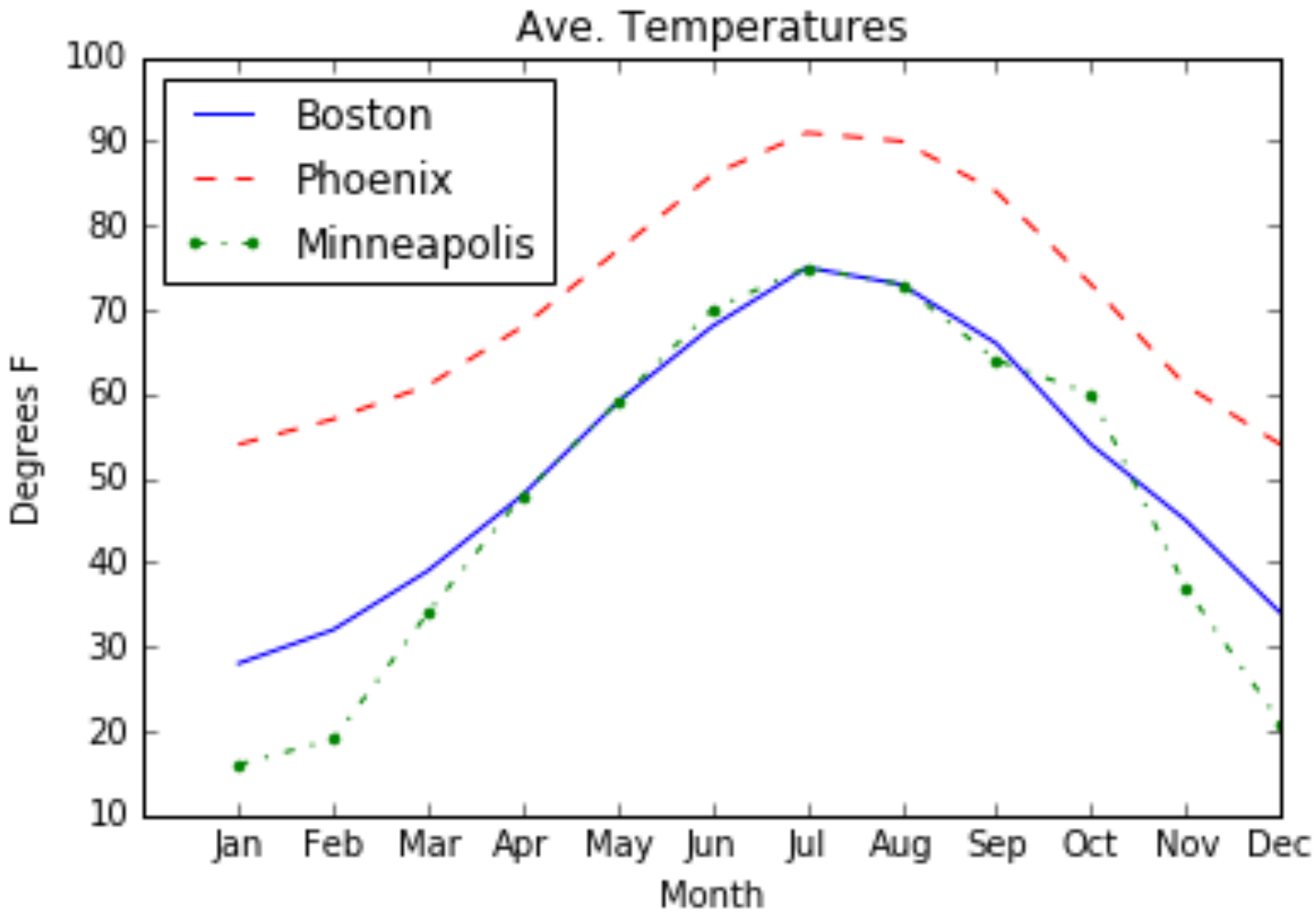
CONTROLLING DISPLAY PARAMETERS

- Suppose we want to control details of the displays themselves
- Examples:
  - changing color or style of data sets
  - changing width of lines or displays
  - using subplots

CONTROLLING COLOR AND STYLE

```
months = range(1, 13, 1)
boston = [28,32,39,48,59,68,75,73,66,54,45,34]
plt.plot(months, boston, 'b-', label = 'Boston')
phoenix = [54,57,61,68,77,86,91,90,84,73,61,54]
plt.plot(months, phoenix, 'r--', label = 'Phoenix')
msp = [16,19,34,48,59,70,75,73,64,60,37,21]
plt.plot(months, msp, 'g.-.', label = 'Minneapolis')
plt.legend(loc = 'best')
```

CONTROLLING COLOR AND STYLE



CONTROLLING SCALE

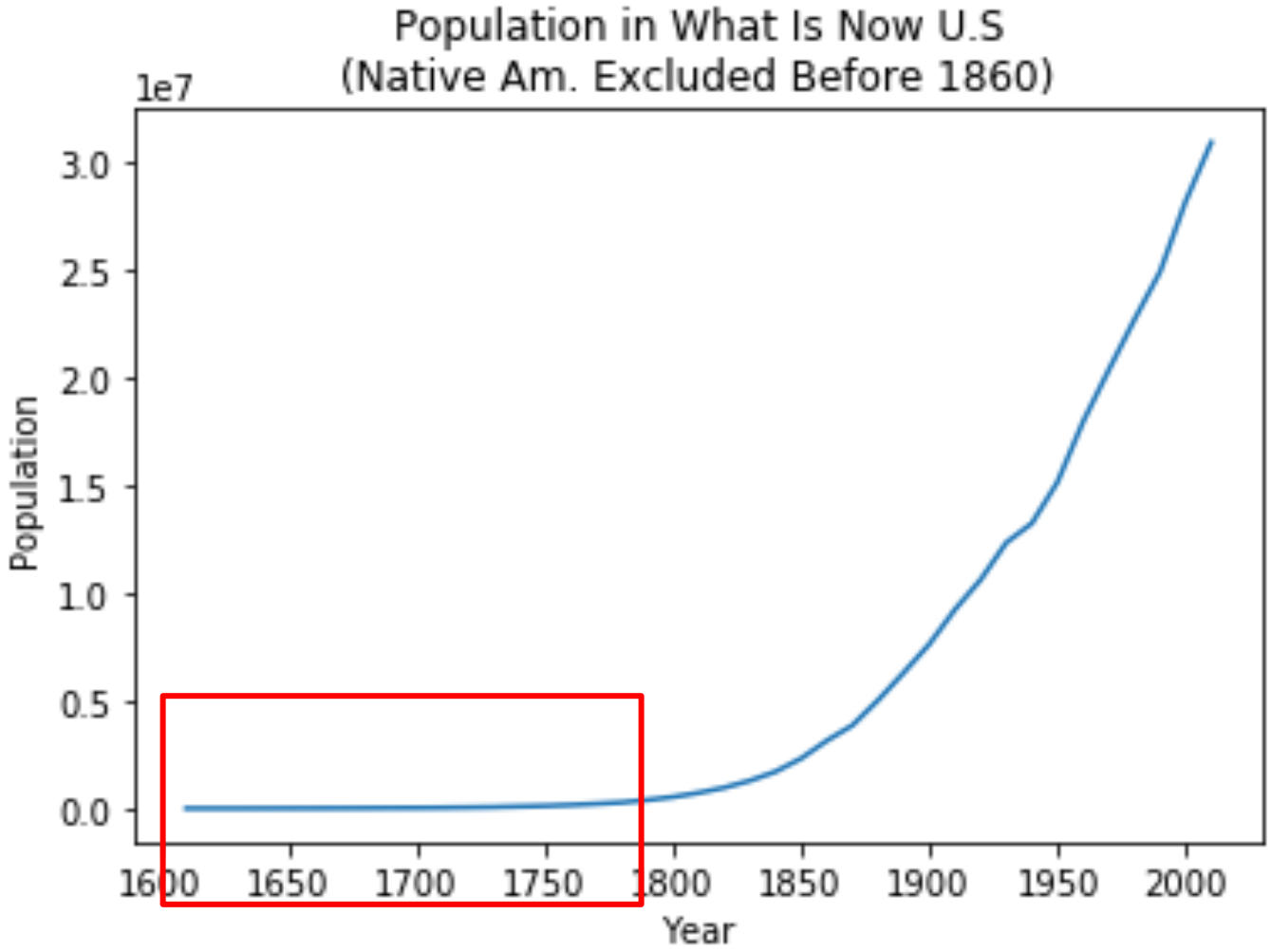
```
def getPop(fileName):
    inFile = open(fileName, 'r')
    dates, pops = [], []
    for l in inFile:
        line = ''
        for c in l:
            if c in '0123456789 ':
                line += c
        line = line.split(' ')
        dates.append(int(line[0]))
        pops.append(int(line[1][:-1]))
    return dates, pops

dates, pops = getPop('USPopulation.txt')
plt.plot(dates, pops)
plt.title('Population in What Is Now U.S\n' +\
          '(Native Am. Excluded Before 1860)')
plt.xlabel('Year')
plt.ylabel('Population')
```

USPopulation.txt

1610 350
1620 2,302
1630 4,646
1640 26,634
1650 50,368
1660 75,058
1670 111,935
.
.
.
2010 308,745,538

POPULATION GROWTH

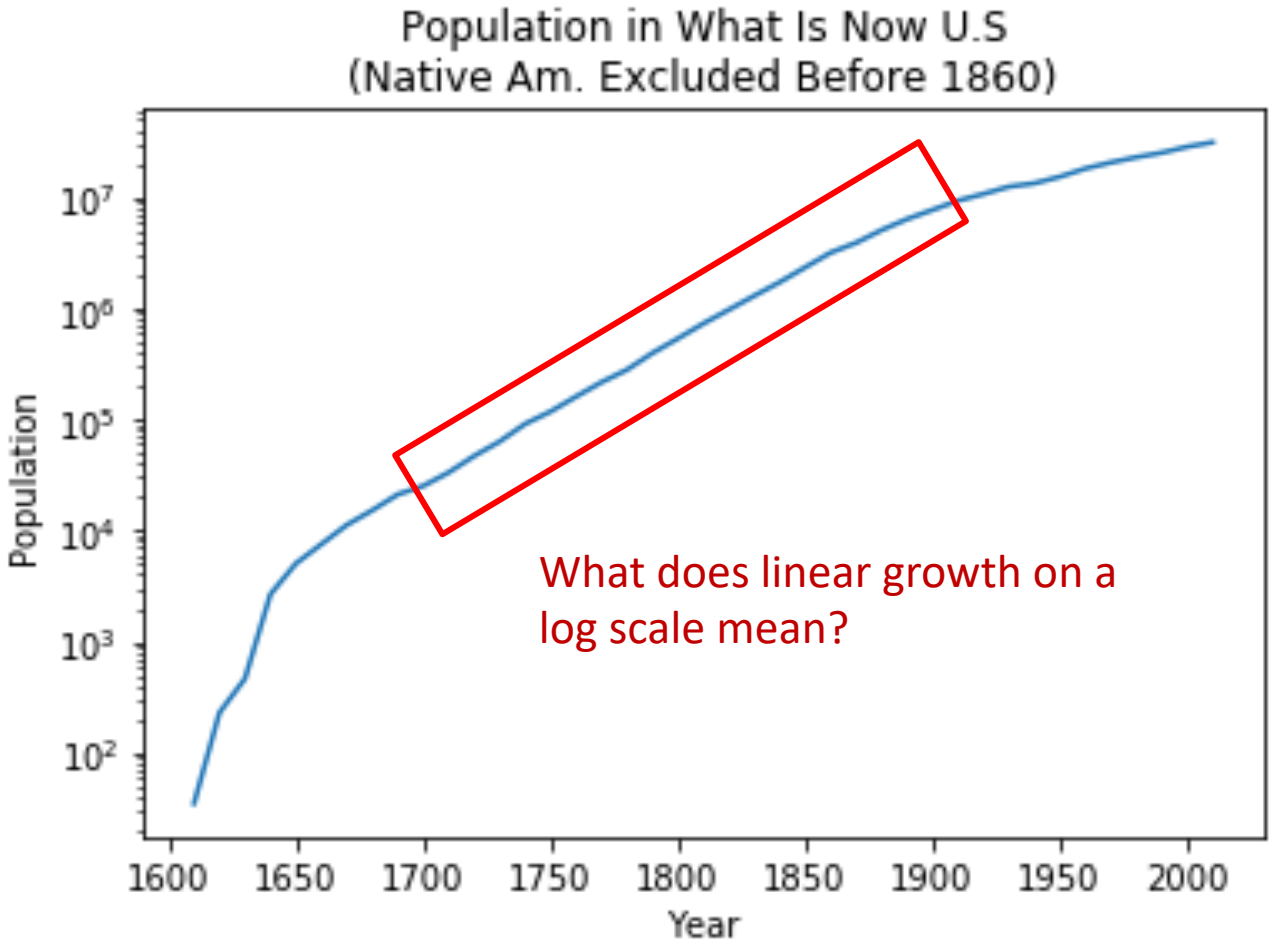


What's going on in the early years?

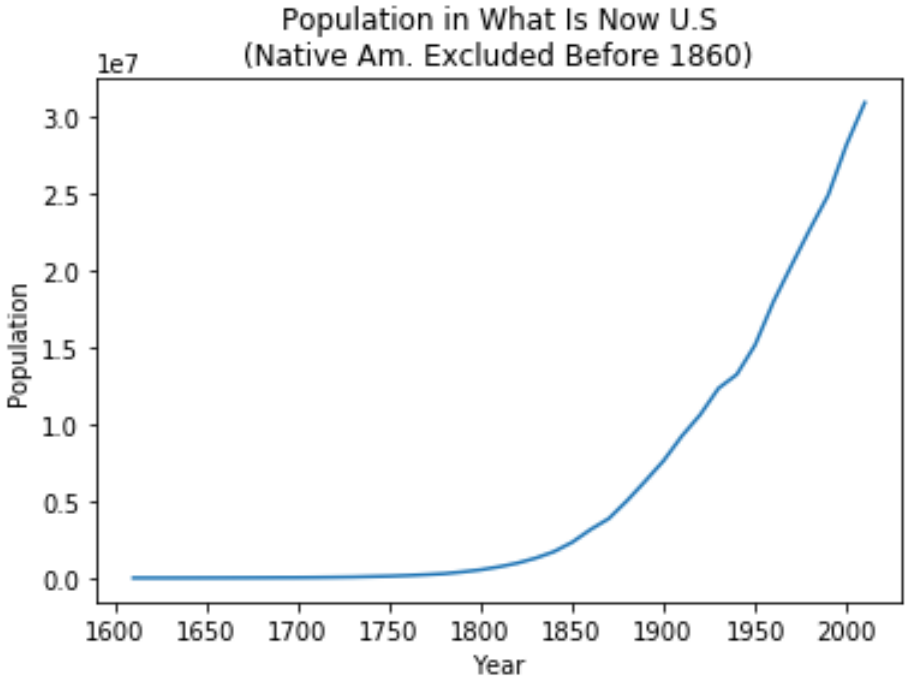
CHANGING THE SCALE

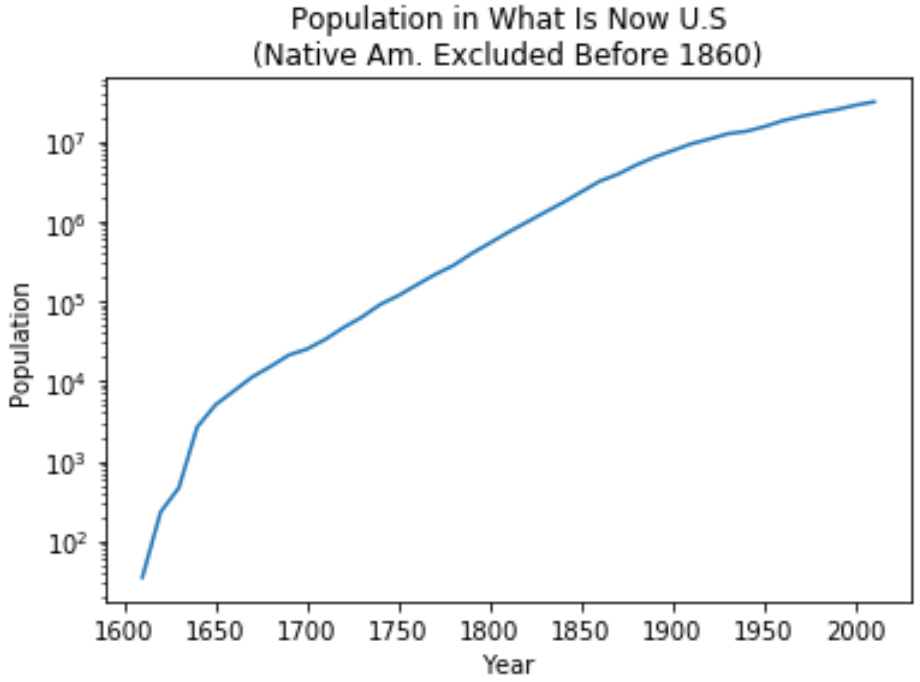
```
dates, pops = getUSPop('USPopulation.txt')
plt.plot(dates, pops)
plt.title('Population in What Is Now U.S\n' +\
          '(Native Am. Excluded Before 1860)')
plt.xlabel('Year')
plt.ylabel('Population')
plt.semilogy()
```

POPULATION GROWTH



WHICH DO YOU FIND MORE INFORMATIVE?





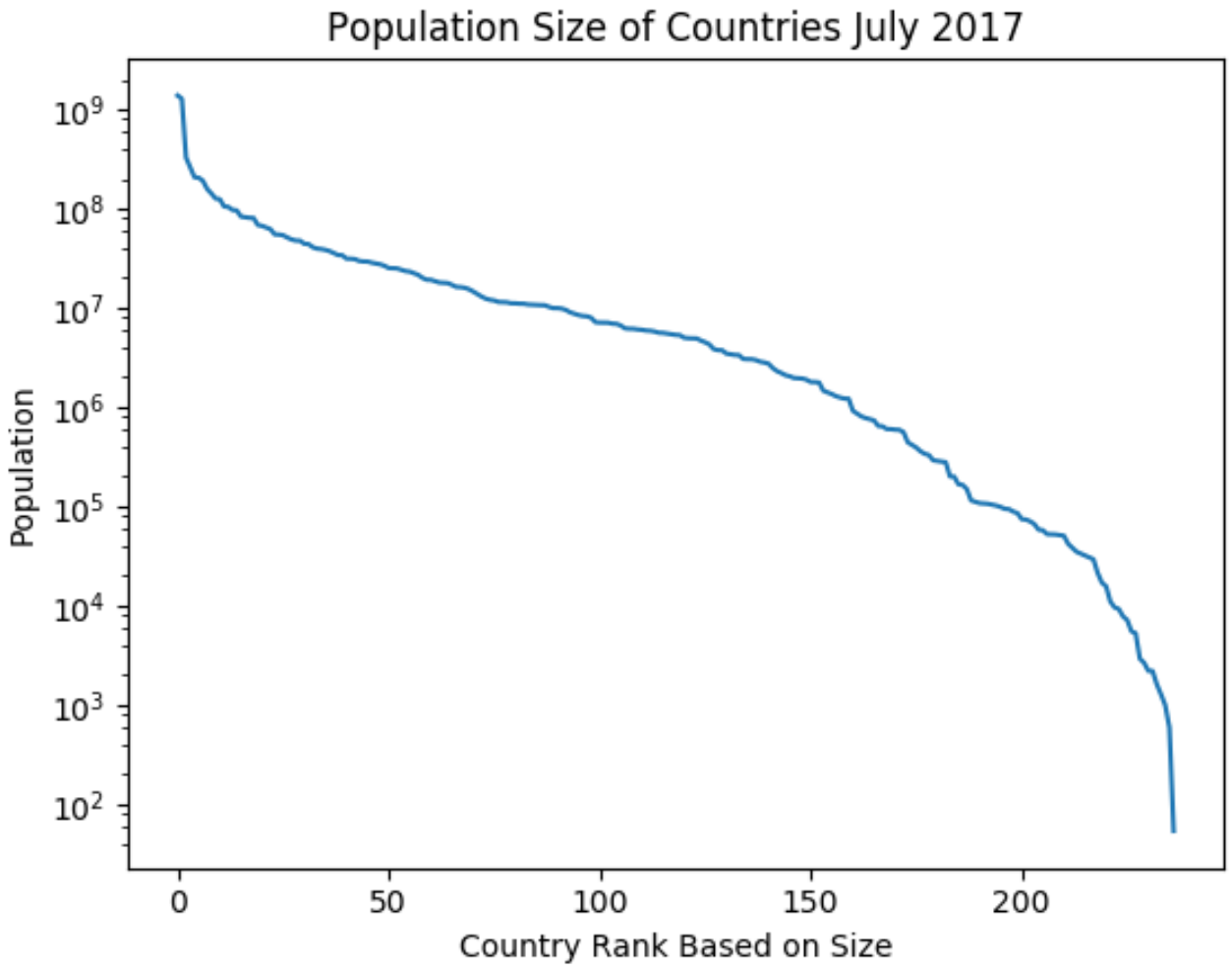
ANOTHER EXAMPLE

```
def getCountryPops(fileName):
    inFile = open(fileName, 'r')
    pops = []
    for l in inFile:
        line = l.split('\t')
        l = line[2]
        pop = ''
        for c in l:
            if c in '0123456789':
                pop += c
        pops.append(int(pop))
    return pops
```

1	China	1,379,302,771	July 2017 est.
2	India	1,281,935,911	July 2017 est.
3	United States	326,625,791	July 2017 est.
4	Indonesia	260,580,739	July 2017 est.
5	Brazil	207,353,391	July 2017 est.
6	Pakistan	204,924,861	July 2017 est.
7	Nigeria	190,632,261	July 2017 est.
8	Bangladesh	157,826,578	July 2017 est.
9	Russia	142,257,519	July 2017 est.
10	Japan	126,451,398	July 2017 est.
.			
.			
.			
230	Svalbard	2,667	July 2016 est.
231	Norfolk Island	2,210	July 2014 est.
232	Christmas Island	2,205	July 2016 est.
233	Niue	1,626	June 2015 est.
234	Tokelau	1,285	2016 est.
235	Holy See (Vatican City)	1,000	2015 est.
236	Cocos (Keeling) Islands	596	July 2014 est.
237	Pitcairn Islands	54	July 2016 est.

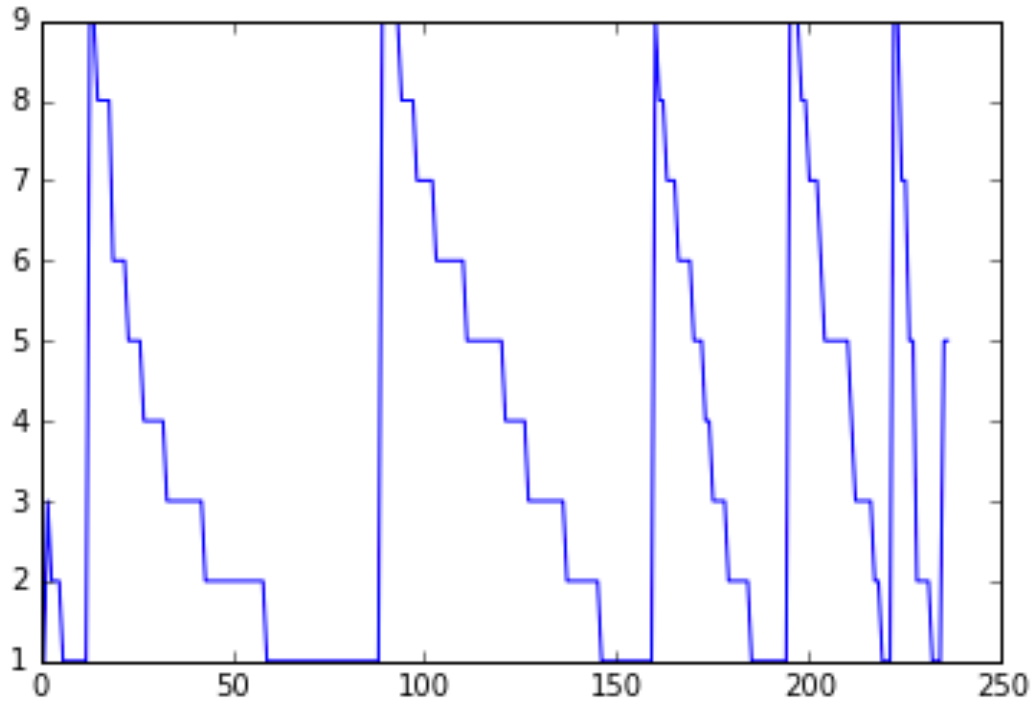
```
pops = getCountryPops('lect11_countryPops.txt')
plt.plot(pops)
plt.title('Population Size of Countries July 2017')
plt.ylabel('Population')
plt.xlabel('Country Rank Based on Size')
plt.semilogy()
```

POPULATION SIZES



STRANGE INVESTIGATION: FIRST DIGITS

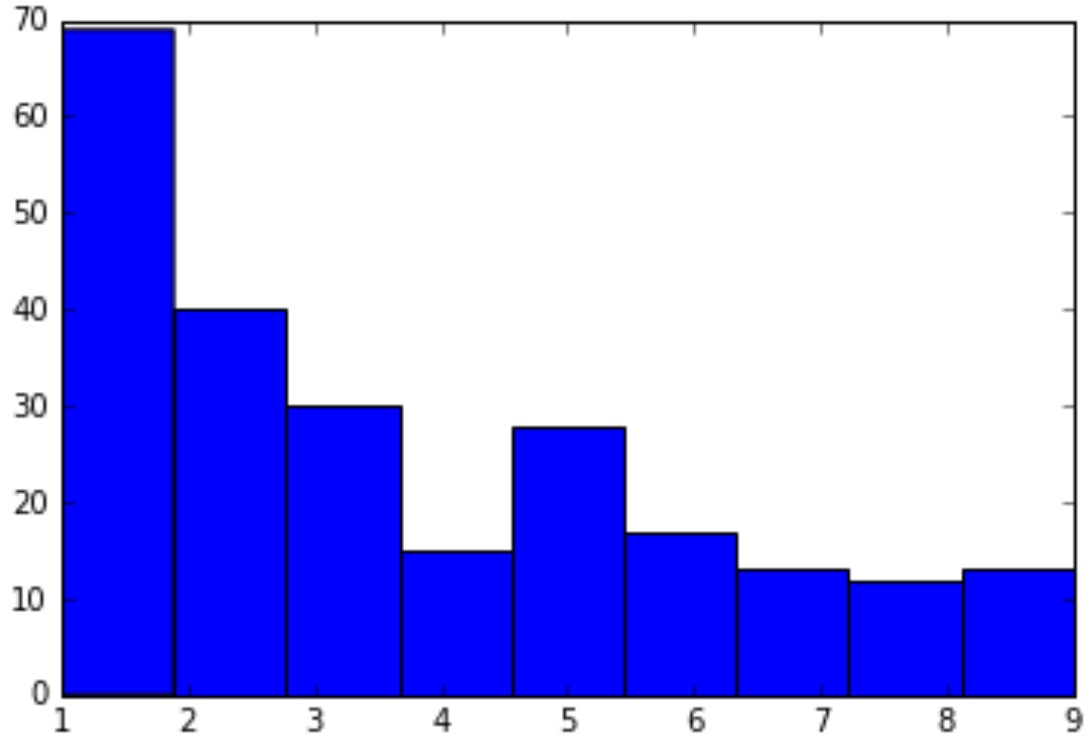
```
pops = getCountryPops('countryPops.txt')
firstDigits = []
for p in pops:
    firstDigits.append(int(str(p)[0]))
plt.plot(firstDigits)
```



Why the saw tooth pattern?

FREQUENCY OF EACH DIGIT

```
plt.hist(firstDigits, bins = 9)
```



Surprising?

28% 1's

Benford's Law

$$P(d) = log_{10}(1+\frac{1}{d})$$

CAN CONTROL LOTS OF OTHER THINGS

- Size of
  - Markers
  - Lines
  - Title
  - Labels
  - x and y ticks
- Scales of both axes
- Subplots
- Text boxes
- Kind of plot
  - Scatter plots
  - Bar plots
  - …

Scratch the surface today

Show other things as useful during our excursion into understanding data later in term

But still only a small sample of what's possible

rcParams

```
#change defaults for plotting
#set line width
plt.rcParams['lines.linewidth'] = 4
#set font size for titles
plt.rcParams['axes.titlesize'] = 20
#set font size for labels on axes
plt.rcParams['axes.labelsize'] = 20
#set size of numbers on x-axis
plt.rcParams['xtick.labelsize'] = 16
#set size of numbers on y-axis
plt.rcParams['ytick.labelsize'] = 16
#set size of ticks on x-axis
plt.rcParams['xtick.major.size'] = 7
#set size of ticks on y-axis
plt.rcParams['ytick.major.size'] = 7
#set size of markers, e.g., circles representing points
#set numpoints for legend
plt.rcParams['legend.numpoints'] = 1
#set parameters for saving figures
plt.rcParams['savefig.dpi'] = 1000
plt.rcParams['savefig.bbox'] = 'tight'
plt.rcParams['savefig.pad_inches'] = 0
```

HOME STRETCH OF 6.0001

- Wednesday
  - Things your mother didn't tell you about Python
  - Wrap up
- Monday
  - TAs will conduct a review session for test
- Wednesday
  - Final exam
- Monday after spring break
  - Start 6.0002