CIS 41B
Advanced Python Programming

Plotting Data

De Anza College

Instructor: Clare Nguyen

Introduction

- Data visualization is the clear and efficient communication of information through statistical graphs, plots and charts.
- When numeric data are well represented in dots, bars, lines, shapes, etc, it helps the user see the relationships between data values.
- matplotlib is one of the Python modules that work with numpy (as well as scipy and pandas) to help us plot data.
- A data plot can be 2D, with a horizontal x-axis and a vertical y-axis.
  It can also be 3D, with an x-, y-, and z-axis.
- In a 2D plot, the data represented on the x-axis and the y-axis are related.
  Example: We plot the number of students registering for a class during the registration period.
  - The x-axis is for time, from start of registration to the last day to add.
  - The y-axis is for the number of students in a class.
  - The number of students is dependent on time. The farther along in the registration period, the more students there are in the class.
- matplotlib has a submodule called pyplot, which has a library of functions that we use to plot data.
- Since matplotlib and pyplot are long names, most programmers use:

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

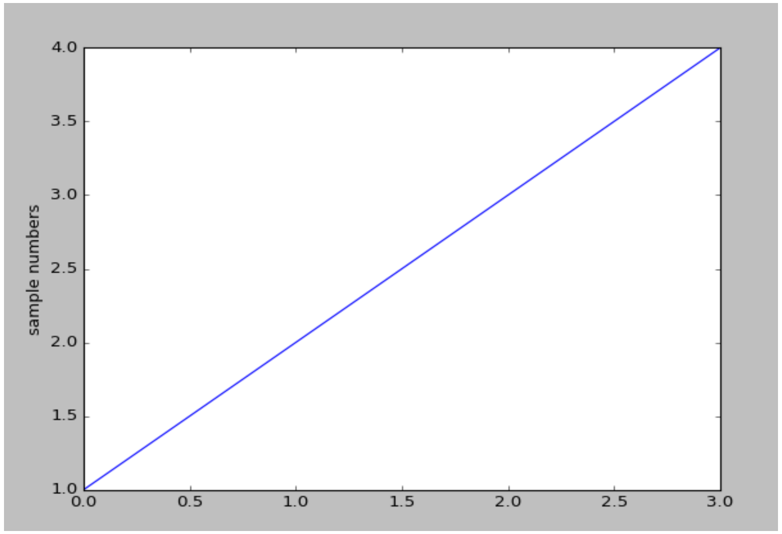
Steps to Plot Data

- The pyplot module has functions to plot data in many different types of plots.
- There are 3 main steps to plot data:
  1. *Choose a plotting function* depending on the type of plot needed
  2. *Format the plot*: add labels, adjust axes, choose color, etc.
  3. *Display the plot*
- Simple example:

```
import  matplotlib.pyplot  as  plt

plt.plot([1,2,3,4])                # step 1: choose a line graph
plt.ylabel(“sample numbers”)   # step 2: label y-axis
plt.show()                         # step 3:display the plot
```

- Resulting plot:



- pyplot uses the data range of 1-4 in the list to determine the range of the y-axis.
- Likewise, since there is no x-axis data, pyplot defaults to start at 0 and count up on the x-axis by the same increment as the y-axis.

Line Graph

- A common type of plot is a line graph. One common use of a line graph is a time series because it's used to visualize a data trend over time.
- Data points are plotted as a sequence of individual markers, or the markers can be connected to form the *line* in a line graph.
- In a line graph the x-axis data is the independent data, and the y-axis data depend on the x-axis data.
- The plot function is used to plot a line graph:

  ```
  plt.plot(xDataList, yDataList)
  ```

- Example:
  We want to create a line graph with the monthly high temperatures of Cupertino for one year.
  - The xDataList is the months 1-12, these are the independent data:

    ```
    np.arange(1,13)
    ```

  - The yDataList is the corresponding high temperatures, these are dependent on the month in a year:

    ```
    [58, 66, 66, 71, 76, 82, 85, 85, 83, 76, 65, 58]
    ```

  - The plot function is:

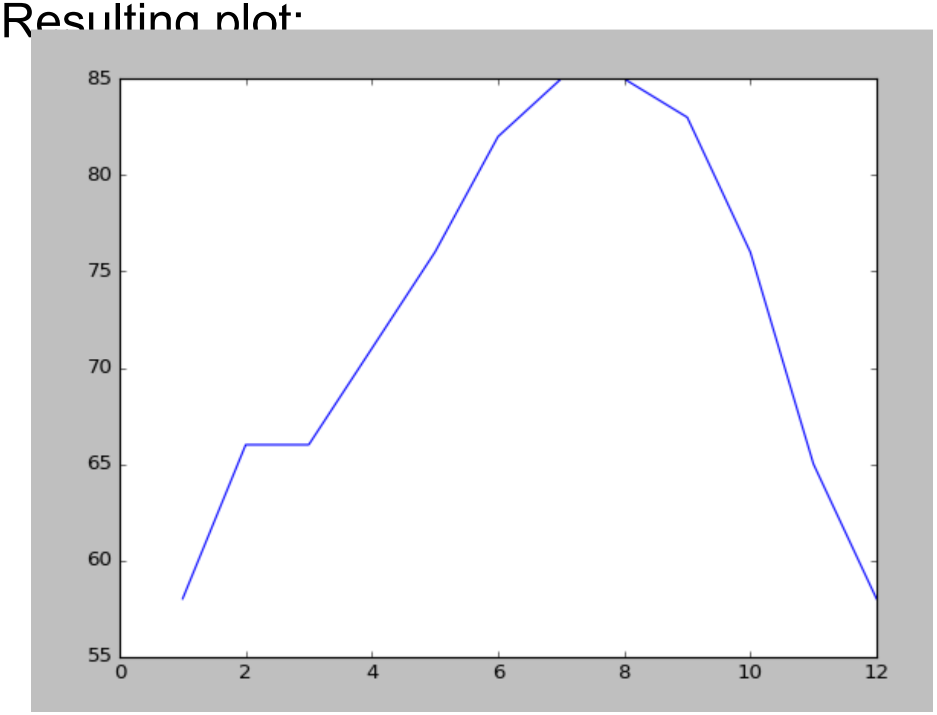
```
plt.plot( np.arange(1,13), [58, 66, 66, 71, 76, 82, 85, 85, 83, 76, 65, 58] )
```

Example: Line Graph

- For the plot of monthly high temperatures of Cupertino for one year, the minimum lines of Python code are:

```
# step 1: choose a line graph
plt.plot(np.arange(1,13), [58, 66, 66, 71,76, 82, 85, 85, 83, 76, 65, 58])
# step 3: display the plot
plt.show()
```

- Resulting plot:



- Note that the range of input data determines the default range of the x- and y-axis.
- But note also that there is no label to explain what data the 2 axes represent.
- And the highest temp (85) reaches the top of the plot.
- This is because the formatting, step 2, is needed.

Formatting: Marker Style and Color

- In the previous plots, the data are connected with a blue line, which is the default style and color for a line graph.
- To change how data points are shown in a plot, add a string of the following characters as input argument after the x and y data lists:

```
plt.plot(xDataList, yDataList, “xg”)
```

The first 1-2 characters is the style

-	solid line (default)
--	dashed line
:	dotted line
.	point
o	circle
v	triangle down
^	triangle up
p	pentagon
*	star
h	hexagon
+	plus
x	x
d	diamond

The last character is the color

b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black

More color choices by using full color names can be found here

Line Graph: Multiple Data Sets

- When there are different but related data sets that span the same time range, it can be useful to plot them in the same graph in order to compare them.
- There are 2 ways to plot multiple datasets.

1. Passing several data sets in one call to plot:

```
plt.plot( xDataList, yData1List, “*g”,
          xDataList, yData2List, “*b”,
          xDataList, yData3List, “*r” )
```

There are 3 datasets, plotted as stars in green, blue, and red.

2. Calling plot multiple times, passing 1 set of data each time:

```
plt.plot( xDataList, yData1List, “-r” )
plt.plot( xDataList, yData2List, “-b” )
```

There are 2 datasets, plotted as lines in red and blue.

- Note that for the purpose of comparison, the xDatalist must be the same across all data sets.
- If we don’t specify the color for the data sets, then pyplot will automatically choose a different color for each set.

Formatting: Add Labels and Title (1)

- When plotting multiple datasets in one plot, it is helpful to have a legend to explain which marker belongs to which dataset.
- We can add an explanation to each dataset by using the label argument:

```
plt.plot( xDataList, yData1List, “-g”, label=“data 1” )
plt.plot( xDataList, yData2List, “-b”, label=“data 2” )
```

pyplot will put the label and the marker style/color in the legend.

- Then we can let pyplot figure out the best place to put the legend:

```
plt.legend(loc=“best”)
```

The “best” place is a place with the minimum data marker so that the legend doesn’t interfere with the data plot.

- It can also be useful to have a label for each axis to explain what data they represent. We use the functions xlabel and ylabel:

```
plt.xlabel( “description for x-axis data”)
```

```
plt.ylabel( “description for y-axis data”)
```

Formatting: Add Labels and Title (2)

- To add a title for the plot:

```
plt.title(“plot title”)
```

- To change the font size of any label or title or the legend, set the keyword argument fontsize with a size (in point).

```
plt.xlabel(“x label text”, fontsize=12)
```

Example: Labels and Title (1)

- Back to our Cupertino temperatures plot, we now add:
  - the temperatures of San Francisco for one year
  - labels, legend, title

```
# plot Cupertino’s temps as a line, in red, with a label
plt.plot( np.arange(1,13), [58, 66, 66, 71, 76, 82, 85, 85, 83, 76, 65, 58],
          "-r", label="Cupertino")
# plot San Francisco’s temps as a line, in blue, with a label
plt.plot( np.arange(1,13), [55, 59, 61, 63, 63, 66, 73, 72, 70, 68, 63, 55],
          "-b", label="San Francisco")
# add a title
plt.title("Monthly Temperature Comparison")
# add a legend
plt.legend(loc=“best”)
# add x-axis label
plt.xlabel("Months")
# add y-axis label
plt.ylabel("Temperature")
# display the plot
plt.show()
```

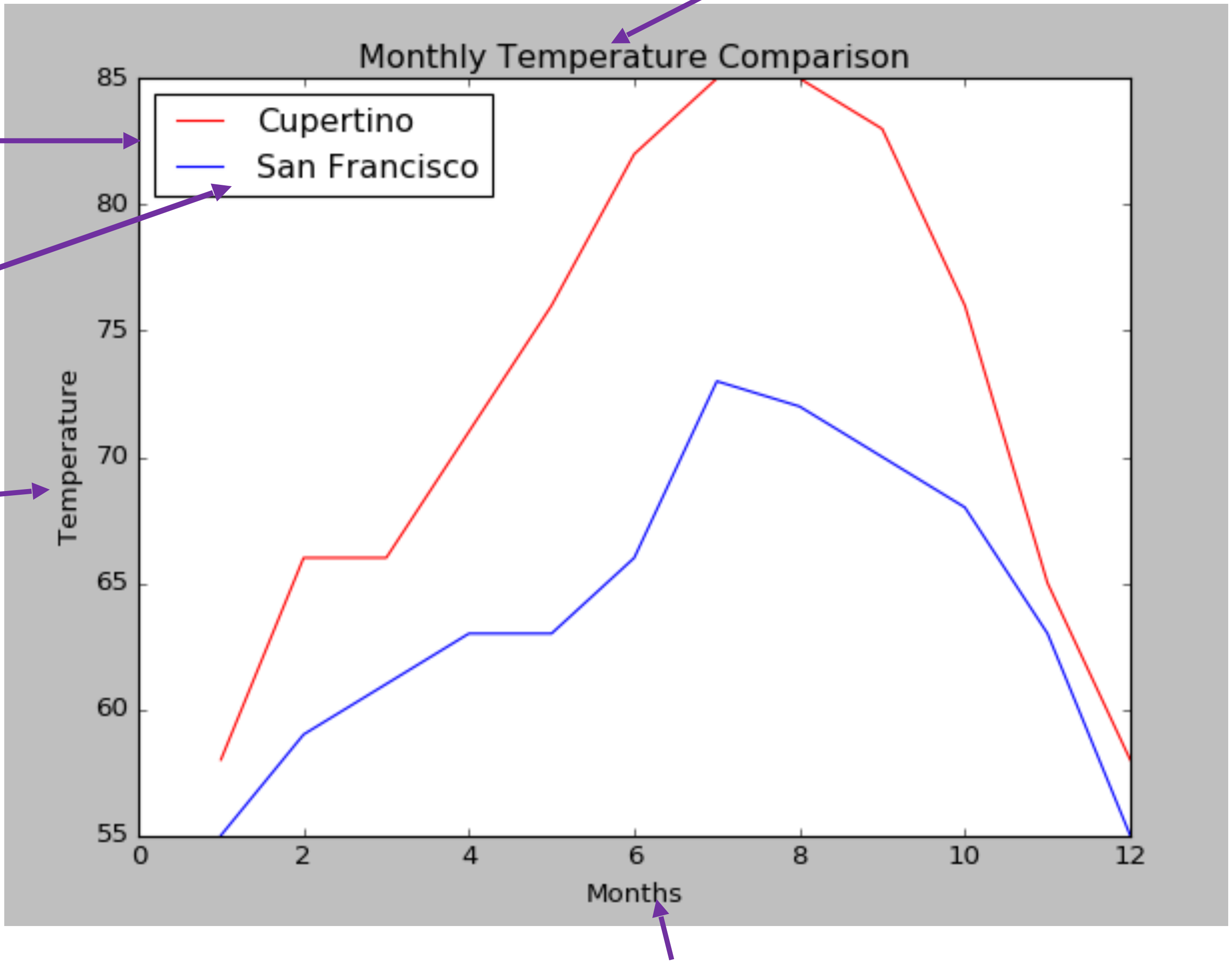
Example: Labels and Title (2)

- The resulting plot is more self-explanatory with labels and a title.

title

legend with marker labels

y-axis label



x-axis label

Formatting: Adjust the Range of Axes

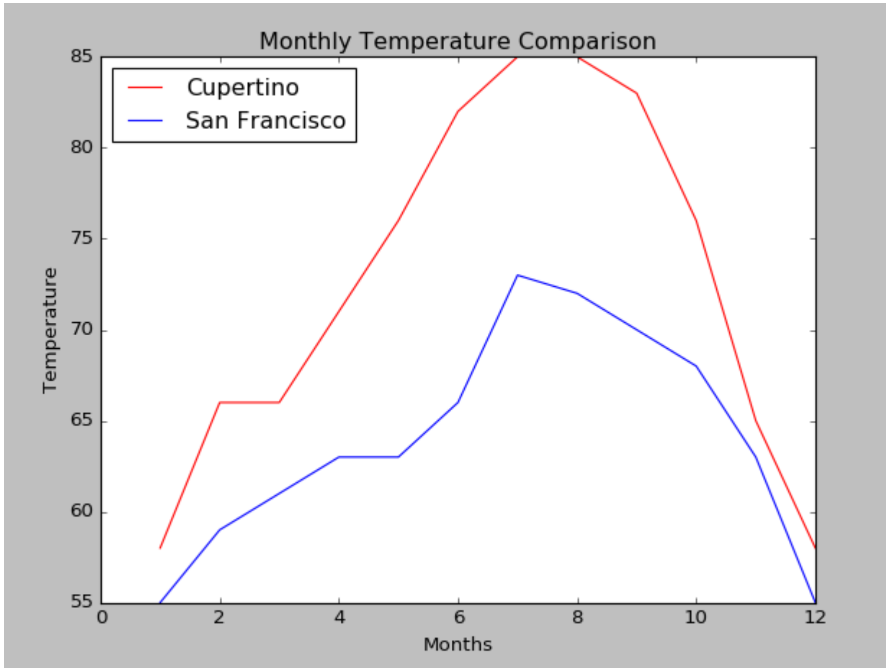
- By default pyplot looks at the range of the data values being plotted in order to set the range of the x-axis and y-axis.
  - The default min value of the axis range is the min value of the data or slightly lower (rounded down from the min data value).
  - The default max value of the axis range is the max value of the data or slightly higher (rounded up from the max data value).
- The min and max default values for the axes might not be the best fit for the range of data being plotted.
  - For example, in the Cupertino temperature plot, the max data markers are shown at the top edge of the plot.
- For these cases we can change the range of the axes to center the data makers better.
- The axis function is used to change the range of the axes by passing in a tuple of 4 values:

```
plt.axis( (xmin, xmax, ymin, ymax) )
```

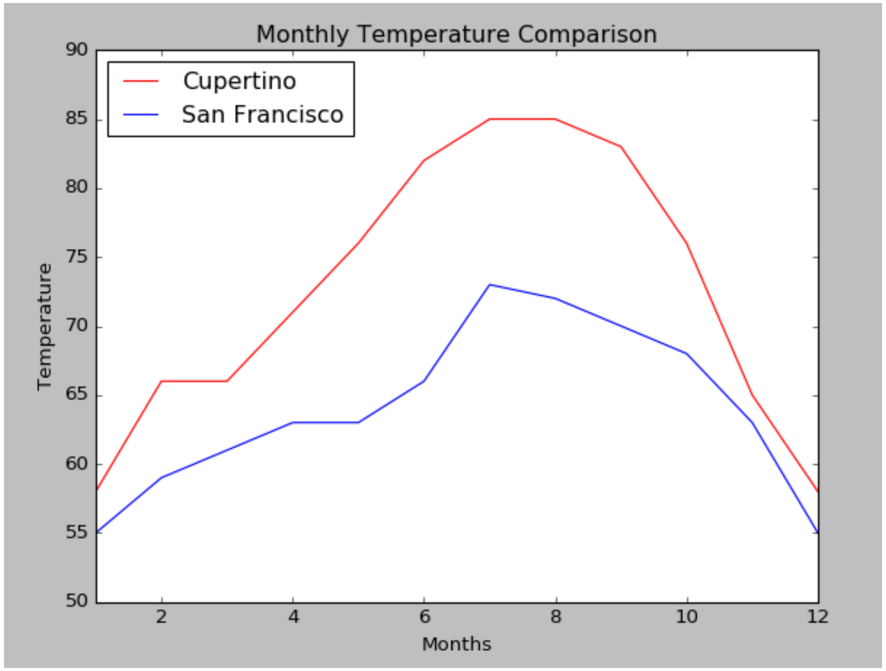
Example: Adjust the Range of Axes

- For the temperature plot we add: `plt.axis( (1, 12, 50, 90) )` so that the data markers are more centered in the plot.

Old version with default axis range



New version with custom axis range



- Be careful of the visualization effect when customizing the axis range. The difference between high and low temps of Cupertino looks larger in the original plot than in the plot with the adjusted range.
- Many dubious data sources manipulate the axis range of their plots to mislead and misrepresent what their data actually say.

Format: Adding Tick Labels

- The tick marks (or ticks) on the axes are defaulted to be equidistant from each other, and this is how we typically want them to be.
- But sometime it is necessary to add text or labels to each tick to explain what they are.
- To add tick labels, use xticks or yticks

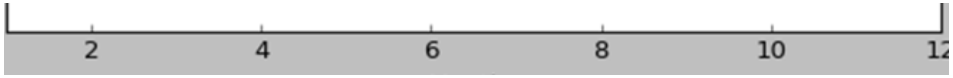
```
plt.xticks( (tuple of locations), (tuple of labels) )
```

```
plt.yticks( (tuple of locations), (tuple of labels) )
```

  - The 2 input argument tuples must be the same size because each tick location must have a corresponding label.
  - The location is a set of numbers that are typically equidistant from each other on the number line, such as (1, 2, 3, 4, 5) or (2, 4, 6)
- By default the tick labels appear in the horizontal direction. To rotate the labels, set the keyword argument rotation with the degree rotation needed, with 90 being the vertical direction.
- If the tick labels are long strings, use plt.tight_layout() to tell matplotlib to expand the plot to fit the labels.

Example: Adjust Ticks

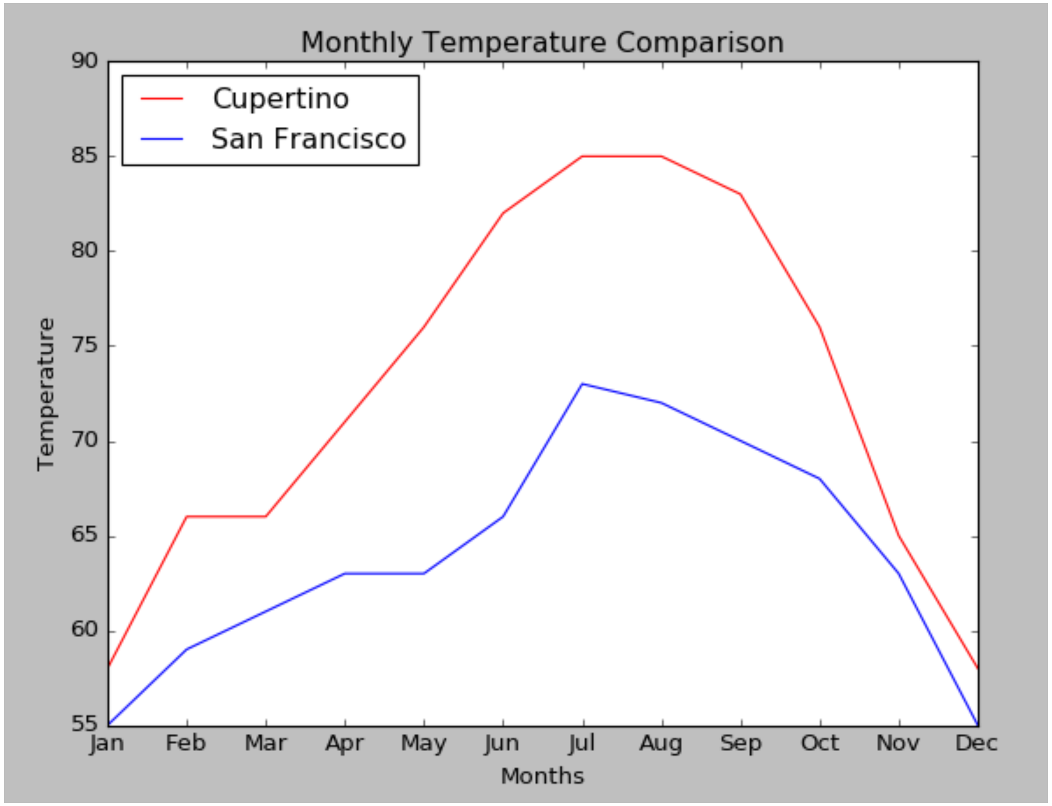
- In the temperature plot the x-axis ticks are the values 1 to 12, with a label at the even numbers:



- It would be more user friendly to change the numbers to month names so that the reader doesn’t have to remember that 8 is for August.
- We add:

```
plt.xticks( np.arange(1,13), "Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec".split())
```

- Resulting plot:



It’s easier to see that July and August are the warmest months.

Histograms

- A histogram shows the *frequency distribution* of data in a dataset, which means it shows the number of times a data value appears in the dataset.
- A histogram or distribution gives us an overview of the data: are data skewed towards larger or smaller values, are the data more concentrated around one value or no particular value, etc.…
- To display data in a histogram, use the hist function of pyplot:

  ```
  plt.hist(dataList)
  ```

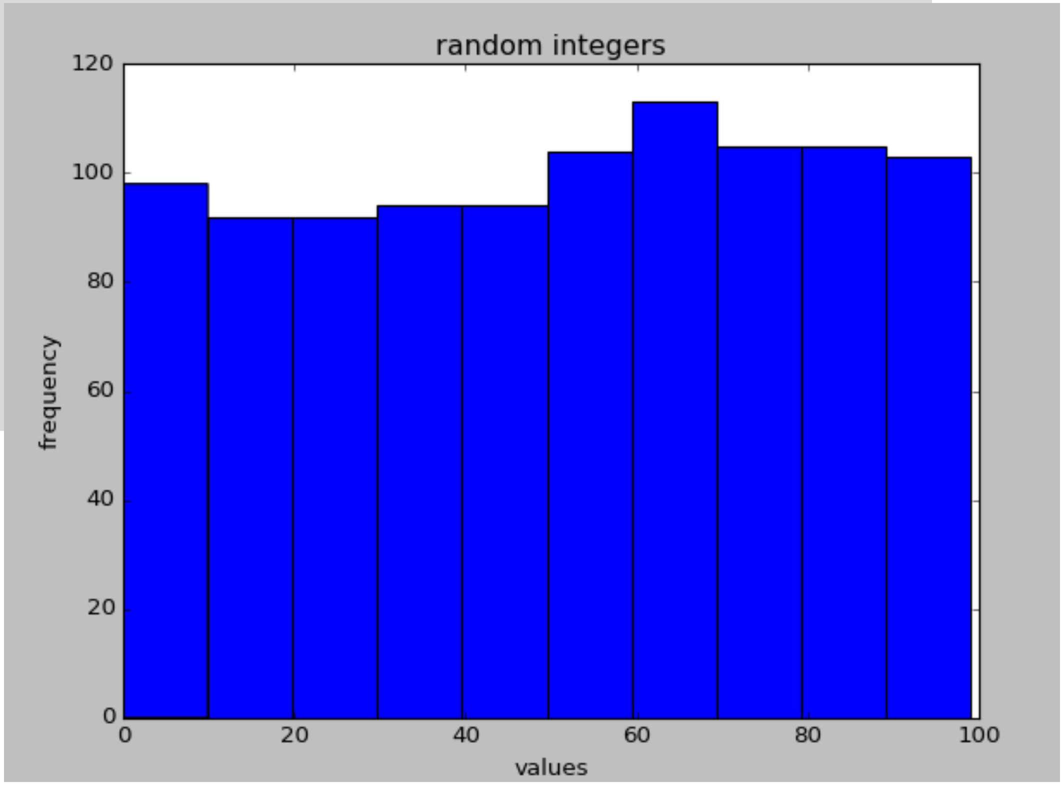
  where dataList is the list of data values to be plotted.
- The range of data values in dataList are separated into intervals or bins, and typically the bins have the same range.

  For example, if the range of data is 0 to 10, then there could be:
  - 10 bins, each bin is the interval from one whole number to another:
    0 to 1   1 to 2   2 to 3 …
  - 5 bins, each bin is the interval from one number to the 2nd next number:
    0 to 2   2 to 4   4 to 6 …
- In a histogram
  - The x-axis represents the bins, and the x-ticks are the bin intervals
  - The y-axis is the frequency or the number of values that are in each bin

Example: Histogram

- We want to check how uniformly distributed are the random data that are generated by numpy, so we use a histogram.

```
# generate random numbers
N = np.random.randint(0, 100, size = 1000)     # 1000 ints between 0 and 100
# create histogram of the numbers
plt.hist(N)
# add title
plt.title("random integers")
# add x and y axis labels
plt.xlabel("values")
plt.ylabel("frequency")
# display plot
plt.show()
```



- As the histogram shows, randint() does a good job.
- There are about 100 ints in each interval of 10 values.

Format: Adjust Bins (1)

- As shown in the histogram of random integer distribution, by default hist divides the range of data into 10 bins.
- We can change the number of bins by setting the bins input argument:

```
plt.hist(dataList, bins=N)
```

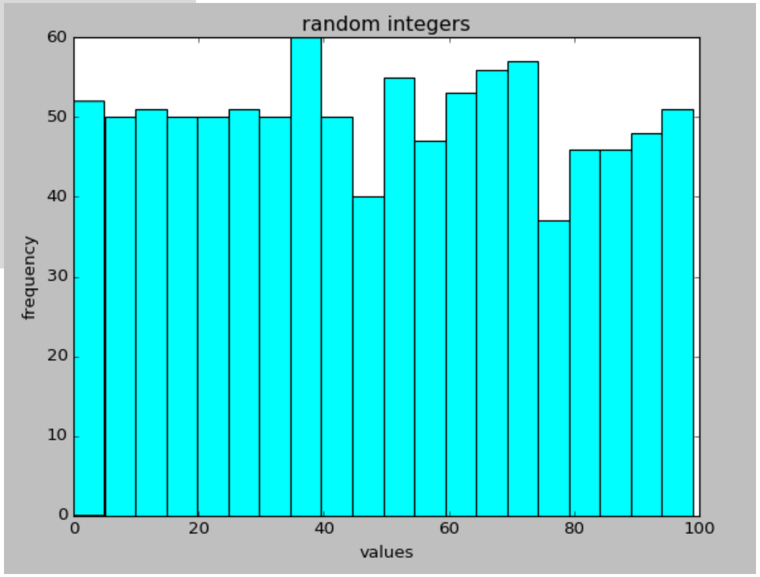
- We can also change the color of the bins by setting the color input argument:

```
plt.hist(dataList, color=“color name”)
```

- Here’s the same code to plot random integer distribution, but with the data being put into more bins and with a color change:

```
numbers = np.random.randint(0,100, size = 1000)
plt.hist(numbers, bins=20, color=“cyan”)
plt.title("random integers")
plt.xlabel("values")
plt.ylabel("frequency")
plt.show()
```

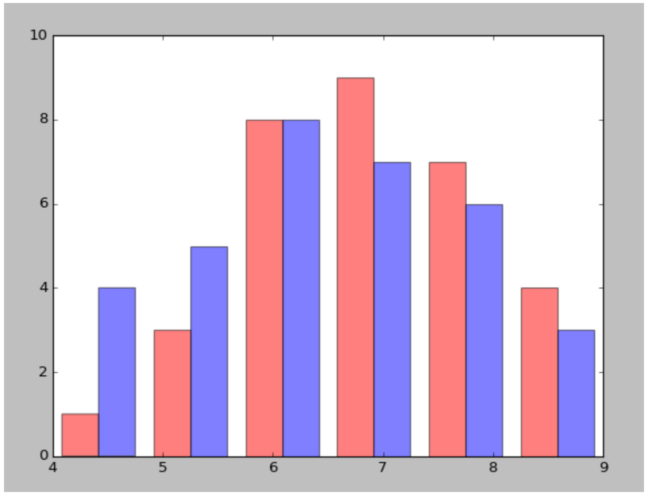
- V
we see that the distribution is not quite as uniform but is still good.



Histogram: Multiple Data Sets

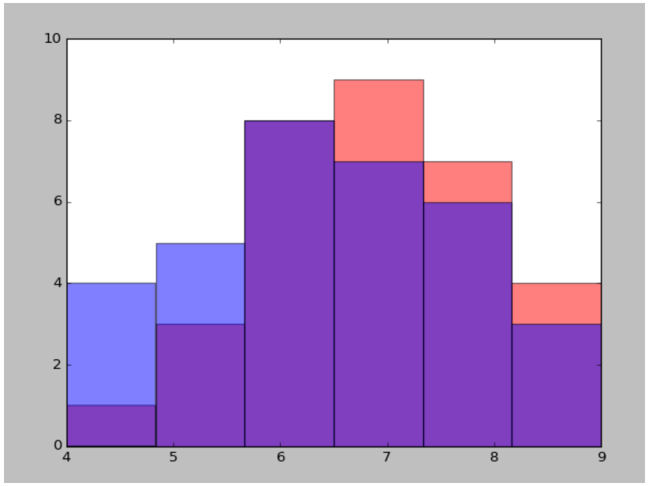
- To plot multiple data sets in the same histogram, the data sets must have the same bin range
- There are 2 ways to plot multiple data sets

1. The data sets are shown side by side: Pass to hist the data sets as a tuple, and optionally the corresponding color and label as tuples.



```
plt.hist( (data1, data2), color=("blue", "red") )
```

2. The data sets overlap each other: Call hist multiple times, each time pass one data set and optionally the color and label. Use the alpha argument to make the color more transparent so that the color in the back can be seen:



```
plt.hist(data1, color='blue', alpha=0.3)
plt.hist(data2, color='red', alpha=0.3)
```

Bar Chart

- A bar chart plots the count of each category of data in a dataset.
- One axis of the bar chart shows the categories of data.
  The other axis shows the measured value (count or values) of those categories.
- To display data in a bar chart, use the bar function of pyplot:

```
plt.bar( xDataList, yDataList, align="center" )
```

where xDatalist is the range of categories
yDataList is the list of data values or counts to be plotted
The align argument shows how the bar is aligned to the ticks

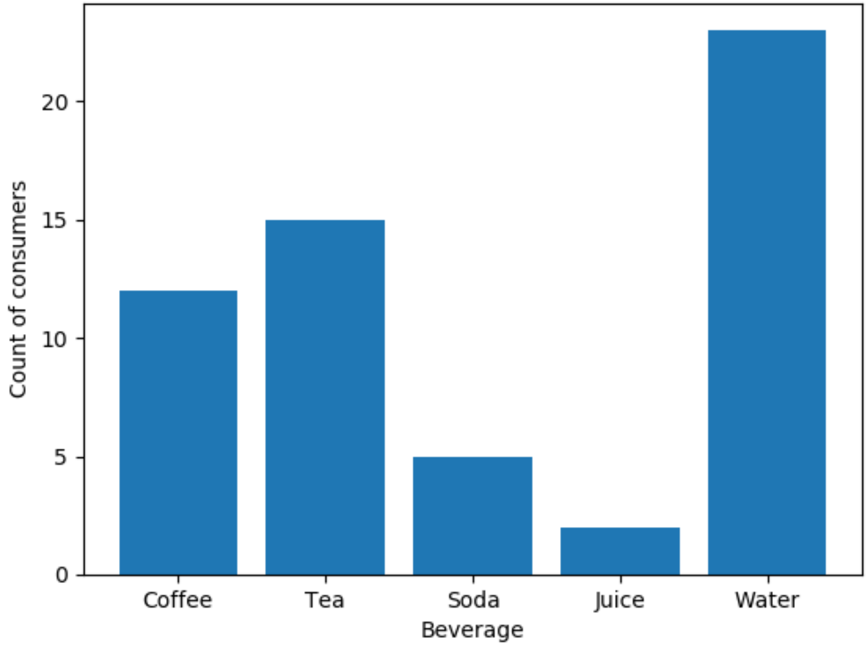
- The default orientation of a bar chart is vertical, with the categories along the x-axis. To plot a horizontal bar chart, we swap the data of the x-axis and y-axis and use the barh function:

```
plt.barh( xDataList, yDataList, align="center" )
```

Example: Bar Chart

- We want to see the popularity of different types of beverage served at a lunch. We use a bar graph to visualize which types are chosen more often.

```
# list of different beverages
label = ["Coffee", "Tea", "Soda", "Juice", "Water"]
# number of people  choosing each type above
bev_count = [12, 15, 5, 2, 23]
# plot vertical bar graph
plt.bar(label, bev_count)
# label x and y axes
plt.xlabel("Beverage")
plt.ylabel("Count of consumers")
# display the plot
plt.show()
```



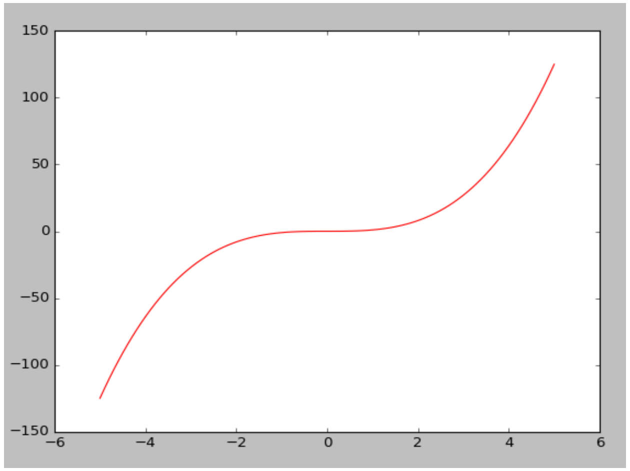
- It's easy to see that water is the most popular at this lunch.

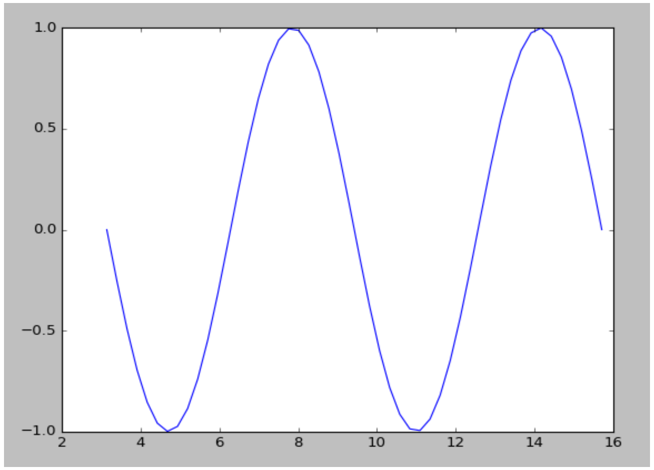
Data for Plotting

- As shown in the previous examples, pyplot can accept input data as a tuple, a list, or a numpy array.
- Internally every sequence of input data is converted into a numpy array for plotting.
- The data for pyplot can also be from math functions or statistical distributions in numpy. Example of plotting some simple math functions:

```
x = np.linspace(-5, 5, 100)
plt.plot(x, x**3, "-r")
plt.show()
```

```
x = np.linspace(np.pi, np.pi * 5, 50)
plt.plot(x, np.sin(x), "-b")
plt.show()
```





Size of Plot

- Sometime we need to change the size of the plot, making it wider or taller, to accommodate the data being plotted.

```
plt.figure(figsize=(10,4))
```

where the tuple is (width, height), in inches.

Going further…

- In this module we learn 3 common types of plots in matplotlib's pyplot module: line graph, histogram, and bar chart. We also learn some basic formatting of the plots to make it easier to understand the data being plotted.
- The pyplot module that we've been working with has many other choices to display data. It also works seamlessly with scipy and pandas for graphing math functions and for data visualization.
- In addition, matplotlib has many other submodules to support cartography (geographical maps), 3D plots, integrating plots with GUI application, etc.
- There is also a list of other graphical plotting modules and packages that work with Python.

Up Next: GUI