Read in and manipulate data
Plotting —line data, histograms, images.
mcmaster for practice
Generating (and plotting) data Planck + density profile

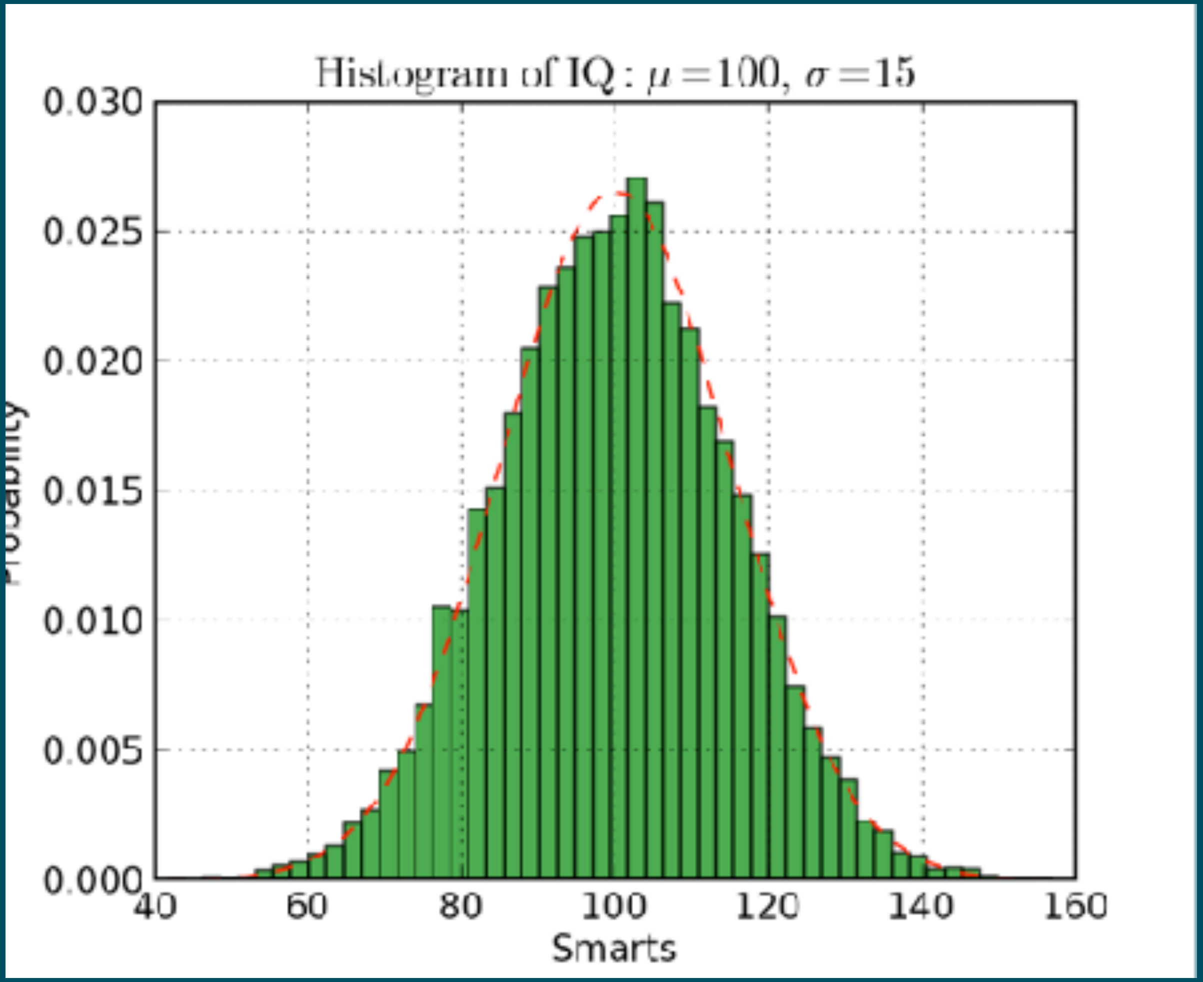
There are lots of modules to read in data from a file — try numpy modules genfromtxt or loadtxt

```
import numpy as np
help(np.genfromtxt)

x1,y1 = np.genfromtxt('dataset1.txt',unpack=True,dtype=np.float)

# that was it!
# we use unpack to tell python to throw out the two columns and we caught them with arrays x and y
# but we could have just captured whatever came out,  then it just would be a merged array:
data = np.genfromtxt('dataset1.txt',dtype=np.float)
print data
print data.shape
print data[:,0]
print x1

# another nice thing, genfromtxt will read data from a URL! What?!
A = np.genfromtxt('https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv',unpack=True, delimiter=',')
```

```
# OK, let's fit our x and y data with a straight line, first define a line function:
def myline(x,m,b):
    return m*x+b
# simple, nothing to it!

# now let's grab a function that performs a least-squares curve fit from the scipy.optimize package:
from scipy.optimize import curve_fit
help(curve_fit)
bestfit, covar_mat = curve_fit(myline, x1, y1, p0 = [1.0,1.0])
print(bestfit)

# and overplot the best-fit:
plt.plot(x1,mylin(x1,bestfit[0],bestfit[1]),'k:')
plt.xlim((-1,21)) # limit the X range of our plot

# make a better sampled plot of our best fit line with a thicker line:
x_fit = np.arange(0, 20, 0.5)
plt.plot(x_fit, myline(x_fit, bestfit[0], bestfit[1]), 'k--', lw=2)

# always add axis labels!
plt.xlabel('Xvalue')
plt.ylabel('Yvalue')
```

Let’s practice slightly more complicated plots

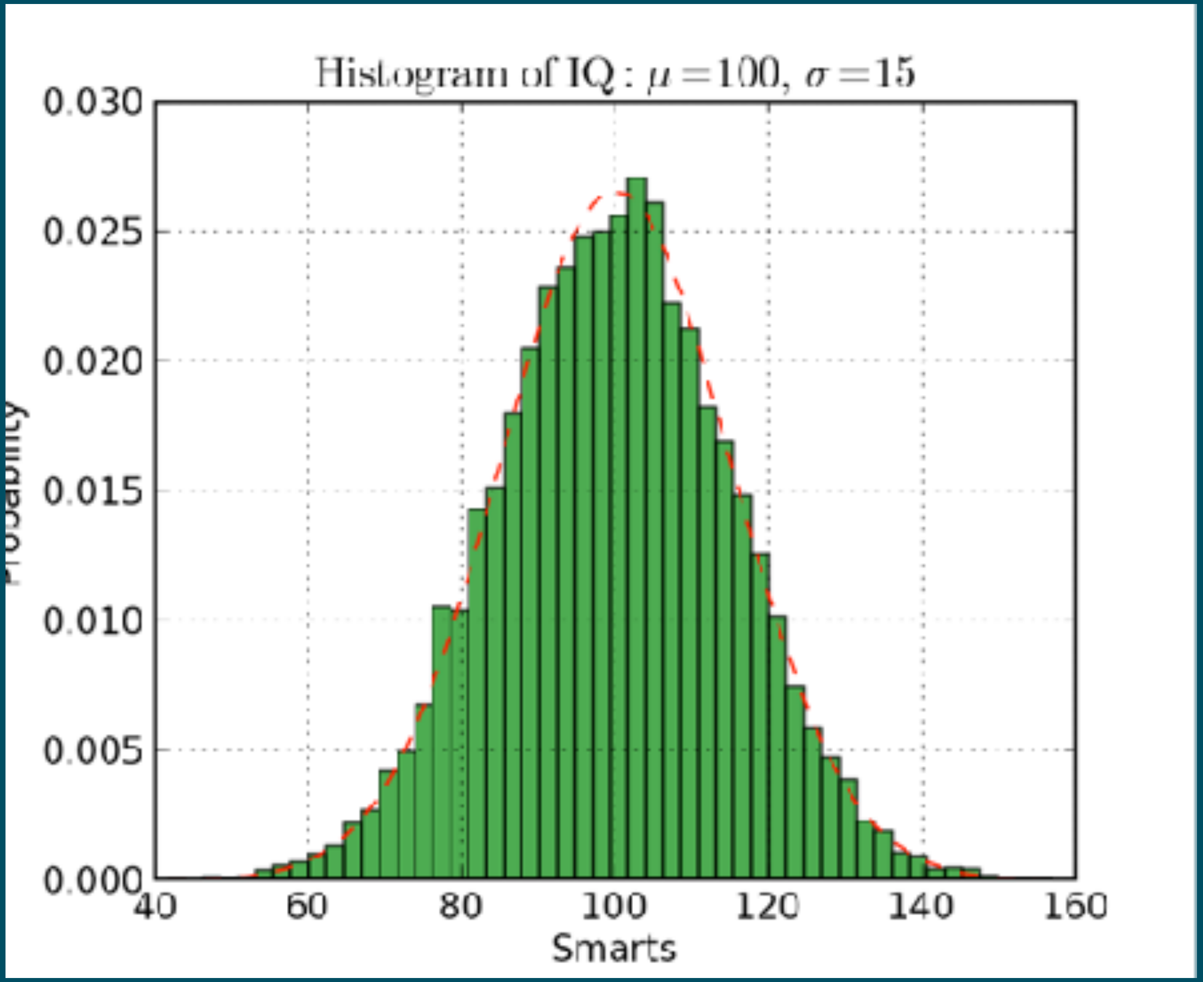
Type **python** and then enter these commands:




```
import numpy as np
import pylab as P
mu,sigma = 100, 15
x=mu + sigma*P.randn(10000)
n,bins,patches = P.hist(x,50,normed=1,histtype='stepfilled',
facecolor=‘green’, alpha=0.75)
y=P.normpdf(bins,mu,sigma)
l=P.plot(bins,y,'k',linewidth=1.5)
P.xlabel(‘smarts’)
P.ylabel(‘Probability’)
P.show()
```

Let's practice slightly more complicated plots

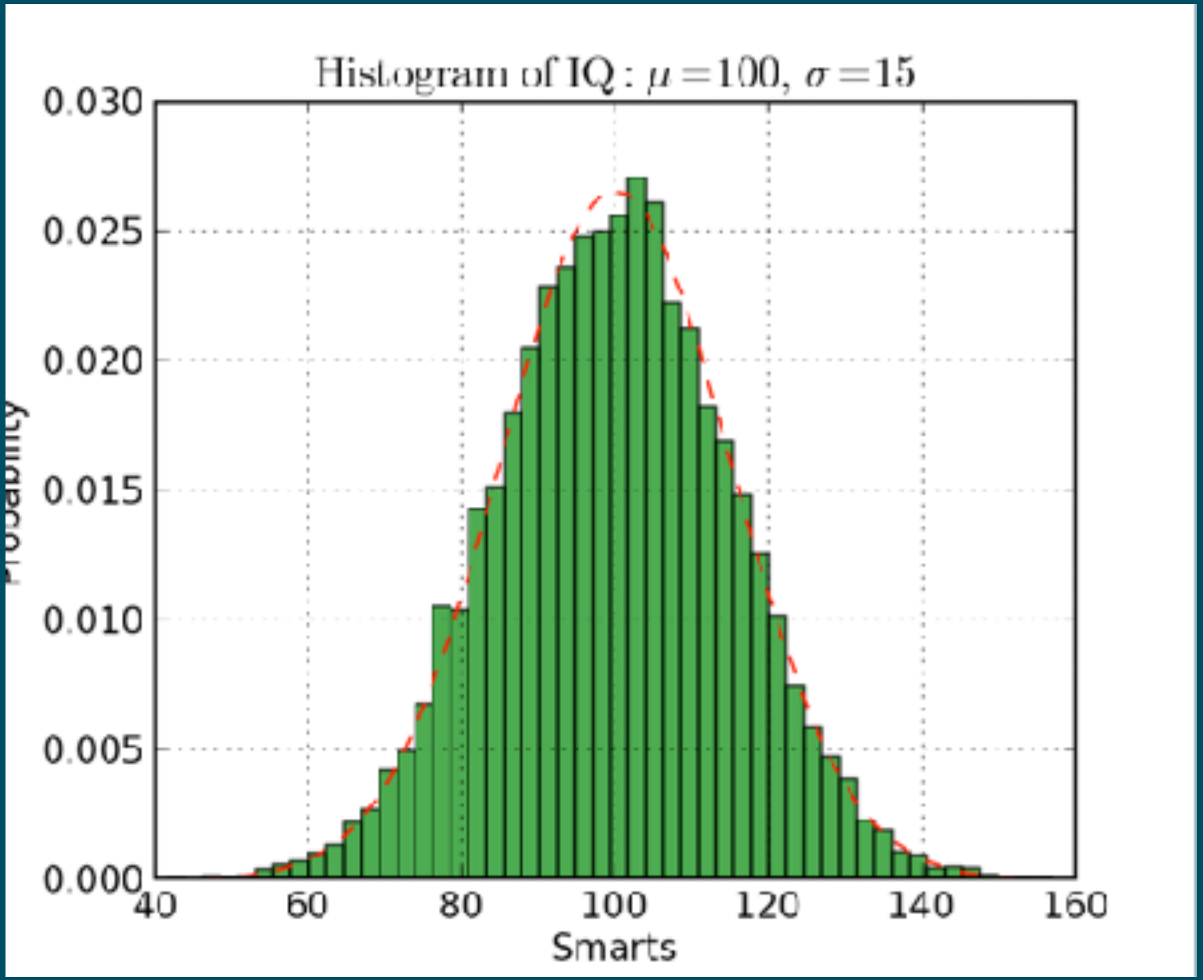
Type **python** and then enter these commands:




```
import numpy as np
import pylab as P
mu,sigma = 100, 15
x=mu + sigma*P.randn(10000)
n,bins,patches = P.hist(x,50,normed=1,histtype='stepfilled',
facecolor='green', alpha=0.75)
y=P.normpdf(bins,mu,sigma)
l=P.plot(bins,y,'k',linewidth=1.5)
P.xlabel('Smarts')
P.ylabel('Probability')
P.show()
```

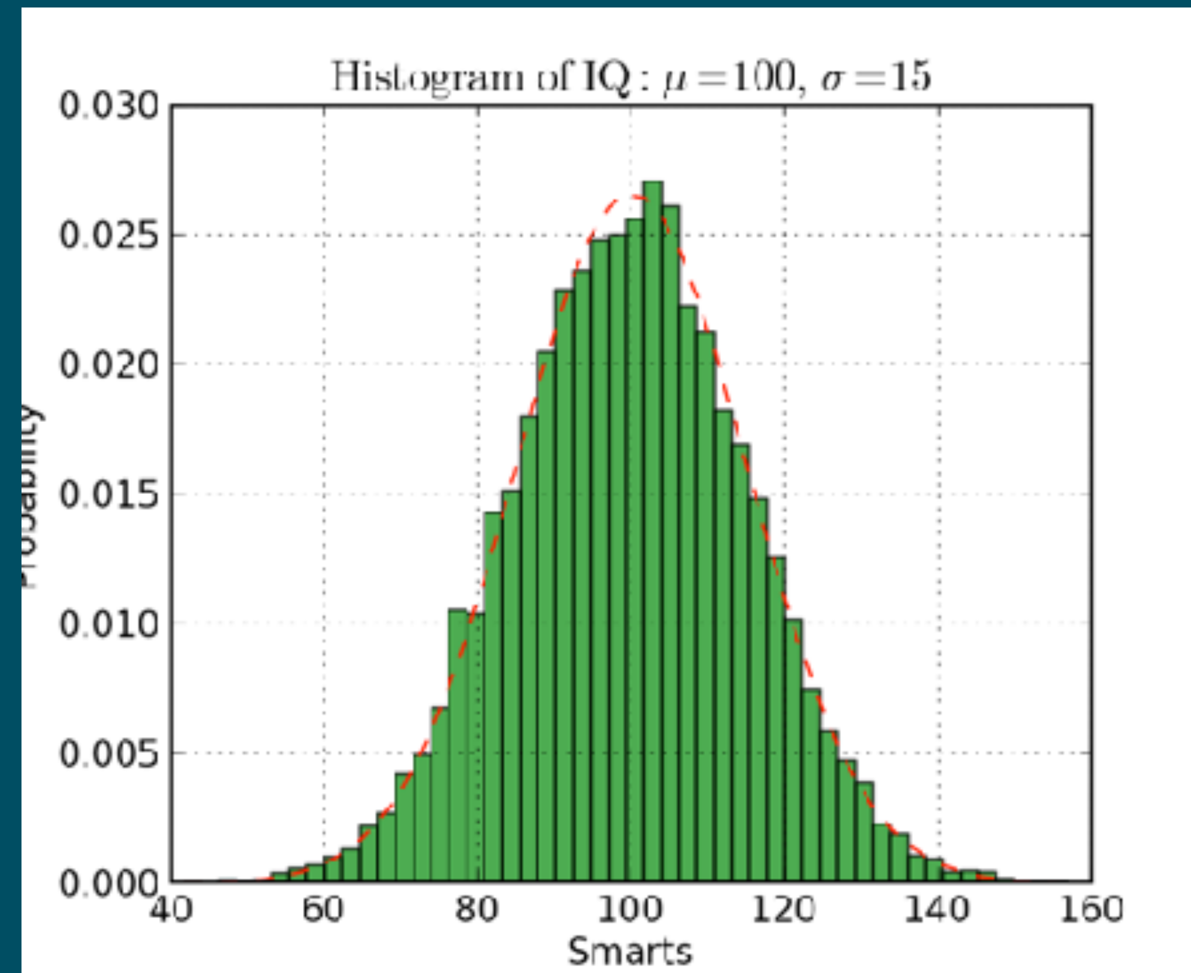
Let’s practice slightly more complicated plots

Type **python** and then enter these commands:




```
import numpy as np
import pylab as P
mu,sigma = 100, 15
x=mu + sigma*P.randn(10000)
n,bins,patches = P.hist(x,50,normed=1,histtype='stepfilled',
facecolor=‘green’,alpha=0.75)
y=P.normpdf(bins,mu,sigma)
l=P.plot(bins,y,'k',linewidth=1.5)
P.xlabel(‘Smarts’)
P.ylabel(‘Probability’)
P.show()
```

Let’s practice slightly more complicated plots

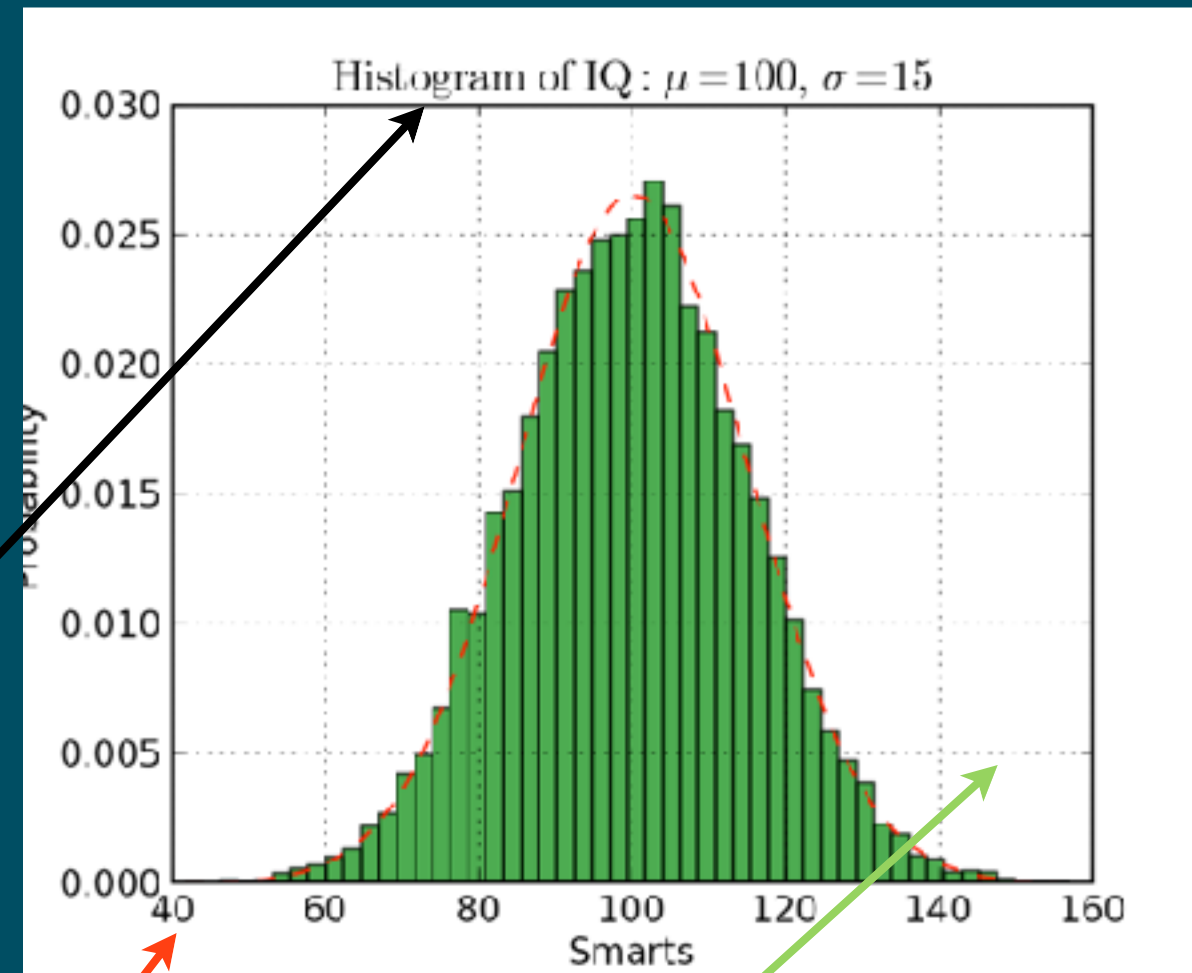


Type **python** and then enter these commands:

```
import numpy as np
import pylab as P
mu,sigma = 100, 15
x=mu + sigma*P.randn(10000)
n,bins,patches = P.hist(x,50,normed=1,histtype='stepfilled',
facecolor=‘green’, alpha=0.75)
y=P.normpdf(bins,mu,sigma)
l=P.plot(bins,y,'k',linewidth=1.5)
P.xlabel(‘Smarts’)
P.ylabel(‘Probability’)
P.show()
```

Let's practice slightly more complicated plots

Type **python** and then enter these commands:



import numpy as np
import pylab as P
mu,sigma = 100, 15
x=mu + sigma*P.randn(10000)
n,bins,patches = P.hist(x,50,normed=1,histtype='stepfilled',
facec
y=P.n
l=P.pl
P.xlab
P.ylabel('Probability')
P.show()

<u>Extra bells and whistles:</u>

P.title(r'$\mathrm{Histogram\ of\ IQ:}\ \mu=100,\ \sigma=15$')

P.axis([40, 160, 0, 0.03])

P.grid(True)

How do we know the possible options for hist (or any python command?)

`matplotlib.pyplot.hist`(*x*, *bins=10*, *range=None*, *normed=False*, *weights=None*, *cumulative=False*, *bottom=None*, *histtype='bar'*, *align='mid'*, *orientation='vertical'*, *rwidth=None*, *log=False*, *color=None*, *label=None*, *hold=None*, ***kwargs*)

Call signature:

```
hist(x, bins=10, range=None, normed=False, weights=None,
        cumulative=False, bottom=None, histtype='bar', align='mid',
        orientation='vertical', rwidth=None, log=False,
        color=None, label=None,
        **kwargs)
```

look for the documentation from the imported library -- here it's

http://matplotlib.sourceforge.net/index.html

Now you practice! Using your pretty version of mcmaster globular cluster data, make a histogram of the number of globs as function of declination

In class practice:

Let's make your own python script!

Write a script that will generate a file containing the Planck spectrum (wavelength and Intensity at that wavelength for many wavelengths)

Create an equation in Python syntax such that for temperature T=300 K, and wavelength lambda = 1cm, it finds the Intensity of a Planck spectrum.

$$I = \frac{2hc^2}{\lambda^5} \frac{1}{e^{hc/\lambda kT} - 1.0}$$

Create an equation in Python syntax such that for temperature T=300 K, and wavelength lambda = 1cm, it finds the Intensity of a Planck spectrum.

$$I = \frac{2hc^2}{\lambda^5} \frac{1}{e^{hc/\lambda kT} - 1.0}$$

```
intensity= ((2*h*c**2)/lambda**5)*
(1.0/ (e**((h*c)/(lambda*k*T) -1.0)
```

Create an equation in Python syntax such that for temperature T=300 K, and wavelength lambda = 1cm, it finds the Intensity of a Planck spectrum.

$$I = \frac{2hc^2}{\lambda^5} \frac{1}{e^{hc/\lambda kT} - 1.0}$$

```
intensity= ((2*h*c**2)/wavelen**5)*
(1.0/ (e**((h*c)/(wavelen*k*T) -1.0)
```

careful, lambda's a reserved word

Create an equation in Python syntax such that for temperature T=300 K, and wavelength lambda = 1cm, it finds the Intensity of a Planck spectrum.

$$I = \frac{2hc^2}{\lambda^5} \frac{1}{e^{hc/\lambda kT} - 1.0}$$

```
intensity= ((2*h*c**2)/wavelen**5)*
(1.0/ (e**((h*c)/(wavelen*k*T) -1.0)
```

have to define these constants ahead of time

In class practice:

Simple version:

planck_simple.py

```
#!/usr/bin/env python

from math import exp

# Set up some parameters
wavelen_min = 1.0e-7     # Minimum wavelength in meters
wavelen_max = 6.0e-5     # Maximum wavelength in meters
samples = 100            # Number of steps to output
T = 300                  # Temperature in Kelvin

# Define some fundamental constants in SI units
c = 3.0e8                # Speed of light
h = 6.626e-34            # Plank's constant
k = 1.38e-23             # Boltzmann's constant


# Open file we want to write to
file = open('spectrum.dat', 'w')

# Calculate plank spectrum and write to file in a loop
for i in range(samples):
  wavelen = wavelen_min + float(i)*(wavelen_max - wavelen_min)/float(samples)
  radiance = (2 * h * c**2 / wavelen**5) * (1.0 / (exp(h * c / (wavelen * k * T)
) - 1.0))
  file.write("%g %g\n" % (wavelen, radiance))

# Close the file now that we're finished
file.close()
```

-:**- planck_simple.py   All (1,0)   (Python)

In class practice:

Fancy version:

planck_fancy.py

```
#!/usr/bin/env python

import sys
import numpy as np

# Set up some parameters
wavelen_min = 1.0e-7    # Minimum wavelength in meters
wavelen_max = 6.0e-5    # Maximum wavelength in meters
samples = 100           # Number of steps to output

# Define some fundamental constants in SI units
c = 3.0e8               # Speed of light
h = 6.626e-34           # Plank's constant
k = 1.38e-23            # Boltzmann's constant


def plank(T, wavelen):
  radiance = (2 * h * c**2 / wavelen**5) * (1.0 / (np.exp(h * c / (wavelen * k * T)) - 1.0))
  return radiance


def main():
  T = float(sys.argv[1])       # Get temperature as a command line parameter
  wavelen = np.arange(samples)
  wavelen = wavelen_min + (wavelen_max - wavelen_min) * wavelen / float(samples)
  radiance = plank(T, wavelen)
  data = np.column_stack( (wavelen, radiance) )
  np.savetxt('spectrum.dat', data)


if __name__ == '__main__':
  main()
```

-:--- planck_fancy.py   Top (1,0)   (Python)

Using the python shell

In class practice:

Now let's plot the data!

show.py

```python
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt

data = np.genfromtxt('spectrum.dat')
fig = plt.figure()
im = plt.plot(data[:,0], data[:,1])
plt.show()
```

-:--- show.py All (1,0) (Python)

Using the python shell

In class practice:

Now let's plot the data (and add axes, titles, all that good stuff)...

lineplotdemo.py

```python
## To run at the command line, in the
## directory containing the data file,
## type:
## python lineplotdemo.py

## Import necessary plotting libraries.
import numpy as np
import matplotlib.pyplot as plt

## Read and unpack data from file
## into arrays x and y.
x,y = np.loadtxt('spectrum.dat',unpack=True)

## Set up figure object.
fig = plt.figure()

## Set up axes object. Anything preceded
## by ax. will be plotted on these axes.
## This becomes more important when you
## wish to plot multiple subplots on the
## same figure.
ax=fig.add_subplot(111)

## Plot the data arrays. Use plot for line
## plots, scatter for point plots.
ax.plot(x,y)

ax.set_title('Planck Spectrum')
ax.set_xlabel('wavelength')
ax.set_ylabel('Intensity')

## Display the figure. This SHOULD ONLY
## be called once at the end of your
## plotting script.
plt.show()
```

-:--- lineplotdemo.py All (18,0) (Python)

Homework for tomorrow:

Take the mcmaster* file again, and plot the positions of the globular clusters in galactic coordinates. Prettiest plot wins a prize. Put it in a keynote slide to display to the class!

Final Python Project!

Write a python script that finds the density profile of the globular cluster model in the file king.dat in Kelly's home/bootcamp/2014 directory.

Plot a two panel figure with the xy projection of the model on the left, and the density profile on the right.

Hardcopy plus source code is due by the end of class.