An Introduction to Graphing in Python

Based on Software Carpentry Tutorial: https://github.com/swcarpentry/notebooks/blob/master/matplotlib.ipynb (https://github.com/swcarpentry/notebooks/blob/master/matplotlib.ipynb)

The core plotting library in Python is matplotlib (http://matplotlib.org/index.html). The matplotlib gallery (http://matplotlib.org/gallery.html) is a great way to figure out how to make the kind of plots you want. Just look for a plot of the right basic style and click on the image to see the code that generated it.

Two things that may differ a bit from other graphing programs that you are used to:

- Making a plot will return a plot object. If you assign this to something you can use it to modify the figure later. This is why in some of the examples below you will see some output before the figure.
- In some Python interpreters you will need to explicitly tell Python to display the figures. This is done using the `show()` function. Several examples of using `show()` are included below.

Numpy array

The NumPy library can store and manipulate data more efficiently than standard Python arrays. Using NumPy with numerical data is much faster than using Python lists or tuples. It looks similar to Python list, but there is also significant difference between NumPy array and regular Python list.

We will start by importing the library and creating a regular Python list and a numpy array from that list.

In [1]:
```
import numpy

x = [1, 2, 3, 4, 5, 6 ]
np_arr = numpy.array(x)
```

In [2]:
```
x
```

Out[2]: `[1, 2, 3, 4, 5, 6]`

In [3]:
```
np_arr
```

```
Out[3]: array([1, 2, 3, 4, 5, 6])
```

Numpy array can consist only the same type of elements type of the elements must be the same throughout the entire array(normally numbers)

```
In [4]: y=[1,2,3.5,4,5]
        np_y = numpy.array(y)
        np_y
```

```
Out[4]: array([ 1. ,  2. ,  3.5,  4. ,  5. ])
```

broadcasting: single operation is broadcast across the entire array.

```
In [5]: x5 = x * 5
        print x
        print x5
        #x6 = x + 6
        #print x6
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

```
In [6]: np_arr5 = np_arr * 5
        print np_arr
        print np_arr5
        np_arr6 = np_arr + 6
        print np_arr6
```

```
[1 2 3 4 5 6]
[ 5 10 15 20 25 30]
[ 7  8  9 10 11 12]
```

numpy.random.randn(d0) Return a sample (or samples) from the “standard normal” distribution.

```
In [7]: np.random.randn(20)
```

```
Out[7]: array([-0.28762708,  0.57868386, -0.33890299, -1.28807481,  0.61126398,
                1.19204867,  3.07123118, -1.00701038,  0.76814577, -0.36586333,
                0.79498417,  0.97329703, -1.00610433, -0.3104617 ,  0.08913092,
               -0.35189152, -0.80866164,  0.70774416, -0.52588404, -0.58695995])
```

Basic plots of two variables

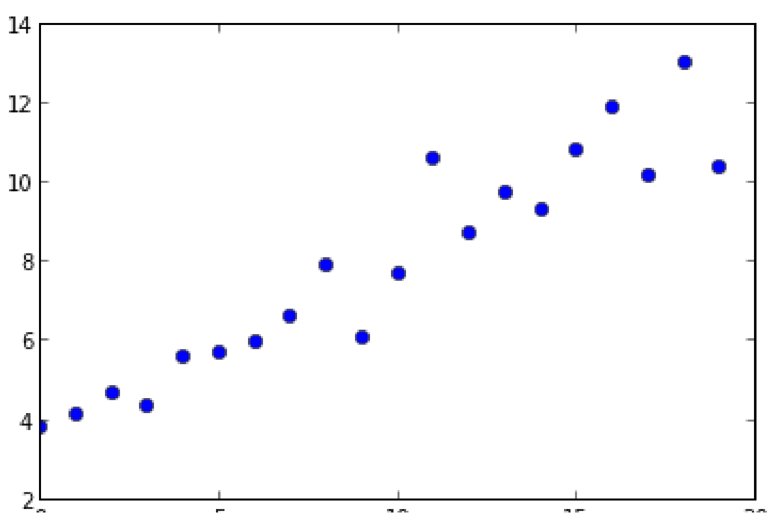
Generating basic bivariate plots is done using the plot() function.

In [8]:
```
range(20)
```

Out[8]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

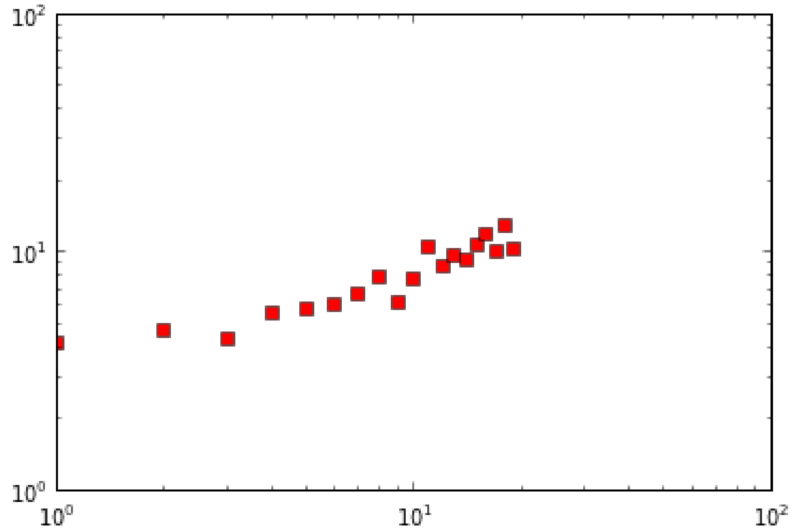
In [9]:
```
import matplotlib.pyplot as plt
import numpy as np

#generate some data
x = np.array(range(20))
y = 3 + 0.5 * x + np.random.randn(20)
print y
print x
#plot the data
plt.plot(x, y, 'bo')
plt.show()
```

```
[  3.80194585   4.16627247   4.67961164   4.35489789   5.57123099
   5.7244663    5.961796     6.61704757   7.89538778   6.09516846
   7.66998241  10.58047538   8.73520115   9.70956201   9.27940851
  10.8108118   11.89702615  10.13895392  12.99246804  10.35515063]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```



We can also scale any of the axes logarithmically.

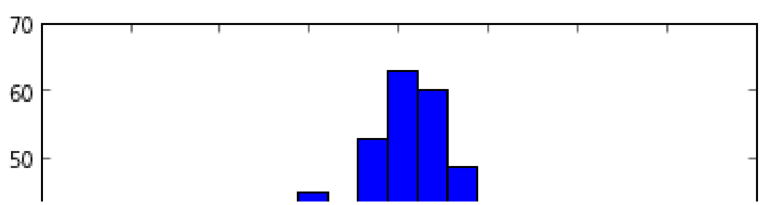
In [10]:
```
fig = plt.loglog(x, y, 'rs')
```

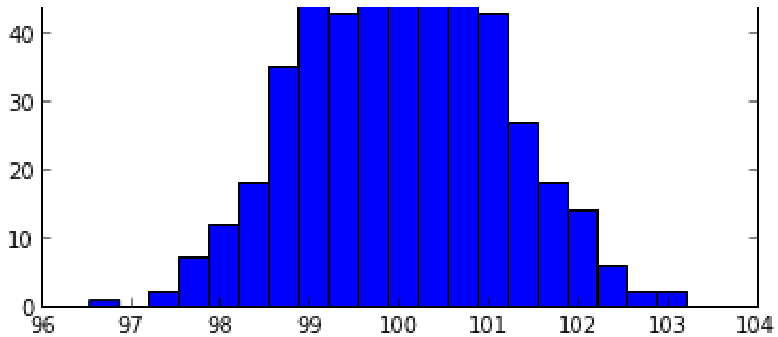


Histograms

Histograms are made using the hist() function.

In [11]:
```
#generate some random numbers from a normal distribution
data = 100 + np.random.randn(500)

#make a histogram with 20 bins
plt.hist(data, 20)
plt.show()
```



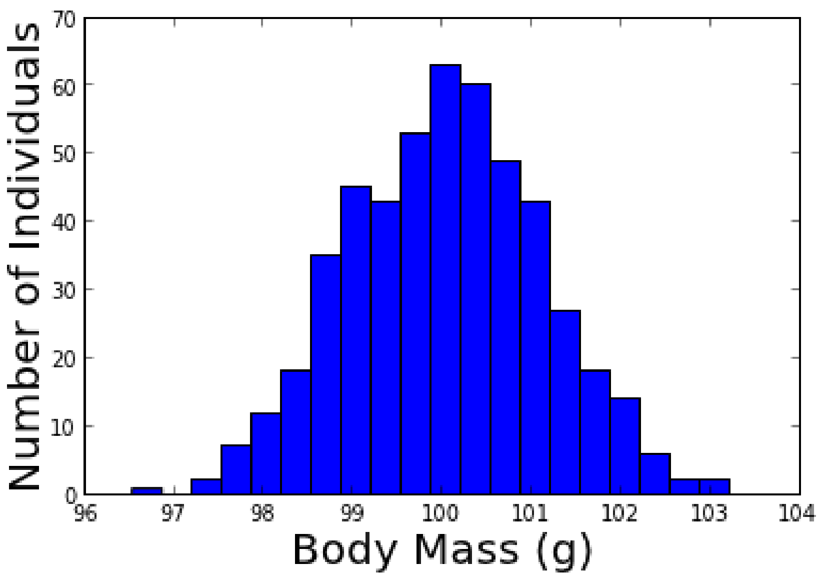


Labels

We can add axis labels to a figure using the xlabel() and ylabel() functions.

In [12]:
```
plt.hist(data, 20)
plt.xlabel('Body Mass (g)', fontsize=20)
plt.ylabel('Number of Individuals', fontsize= 20)
```

Out[12]:
```
<matplotlib.text.Text at 0x10b9c0550>
```



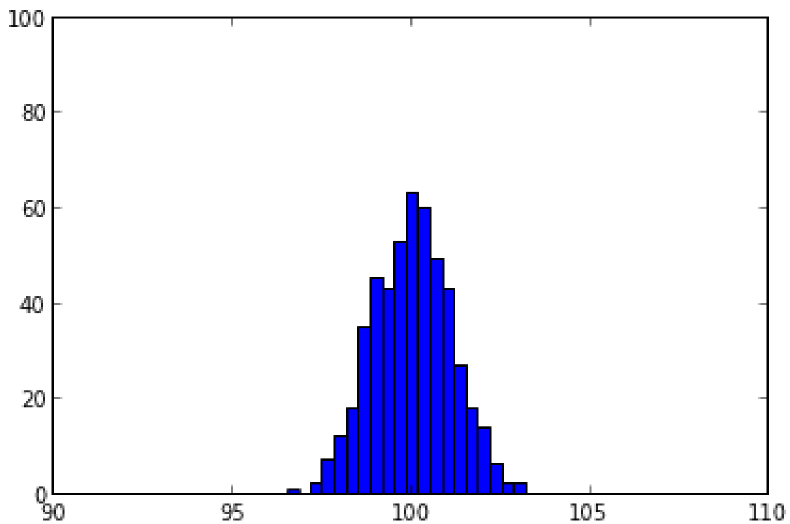
Axis Limits

Axis Limits

Axis limits are changed using the `axis([xmin, xmax, ymin, ymax])` function.

In [13]:
```
plt.hist(data, 20)
plt.axis([90, 110, 0, 100])
```

Out[13]: [90, 110, 0, 100]

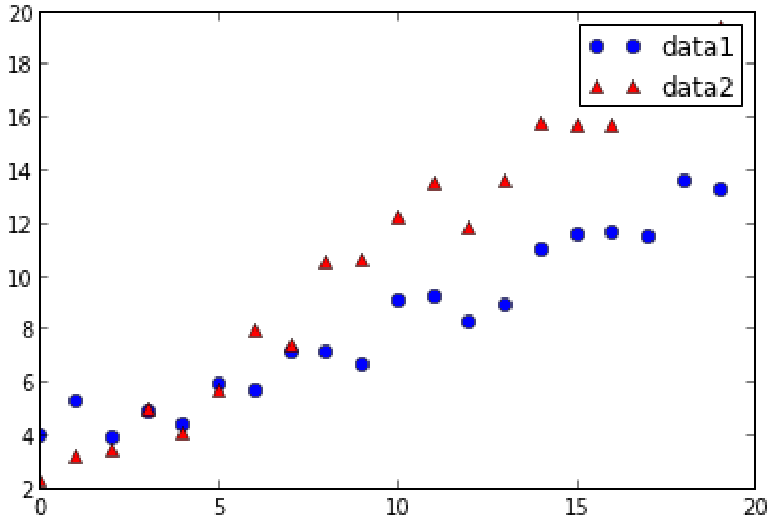


Plotting multiple sets of data together

To plot multiple datasets together we tell Python not to overwrite the previous data using `hold(True)`. Running `hold(False)` will cause Python to start overwriting the figure again.

In [14]:
```
x = np.array(range(20))
y = 3 + 0.5 * x + np.random.randn(20)
z = 2 + 0.9 * x + np.random.randn(20)

#plot the data
plt.plot(x, y, 'bo',label='data1')
#plt.hold(True)
plt.plot(x, z, 'r^',label='data2')
plt.legend()
```

```
plt.show()
```



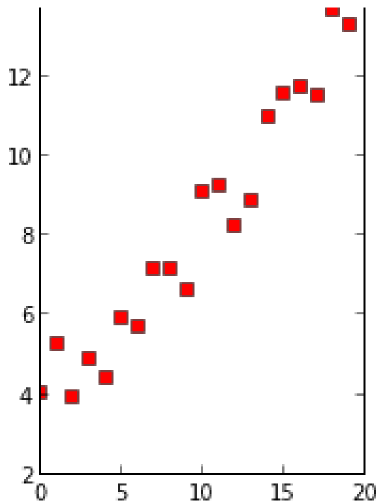
Exercise 1

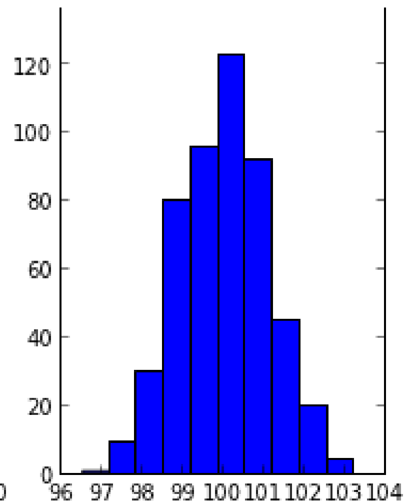
change the lines to dashed line, change the x_limit, etc. set_xlim,set_ylim, Changing axis limits
Changing line colors
Changing lines to dashed (for black and white figures)

Subplots

Subplots are generated using `subplot(#ofRows, #ofCols, Position)`.

In [15]:
```
plt.subplot(1, 2, 1)
plt.plot(x, y, 'rs')
plt.subplot(1, 2, 2)
plt.hist(data, 10)
plt.title('test')
plt.show()
```





Exercise 2

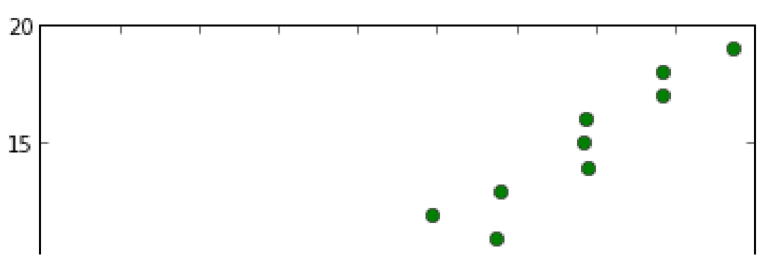
plot a figure with four subplots 2x2, with x-axis y-axis label and legend,etc.

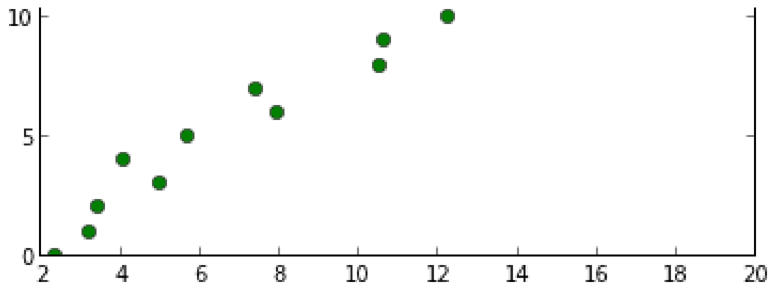
New Figures

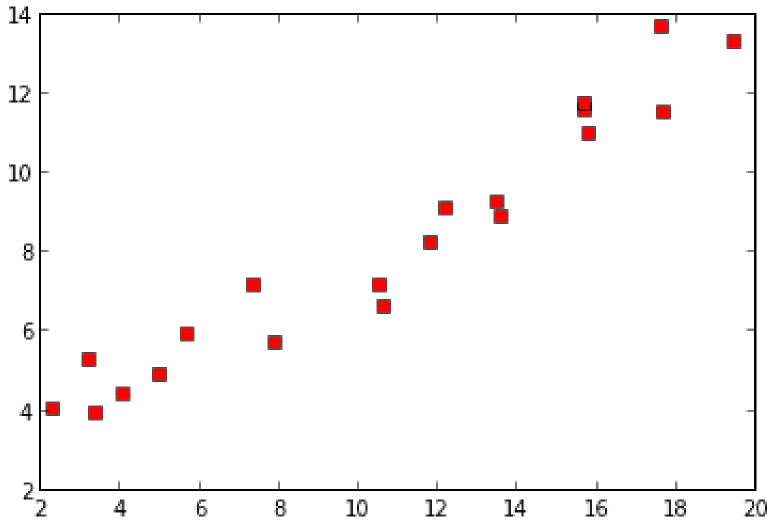
Plotting multiple figures in the same script requires that we create new figures, which is done using `figure()`. In this example the two figures are different figures rather than subplots of a single figure.

In [16]:
```
plt.plot(z, x, 'go')
plt.figure()
plt.plot(z, y, 'rs')
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x10bac48d0>]
```







More complicated figures?

Matplotlib is very powerful and there are lots of functions and arugments to create different kinds of figures and modify them as much as you would like. Check out the website (http://matplotlib.org/) or the gallery (http://matplotlib.org/gallery.html) to learn more.

Exercise 3 Exploring the matplotlib gallery

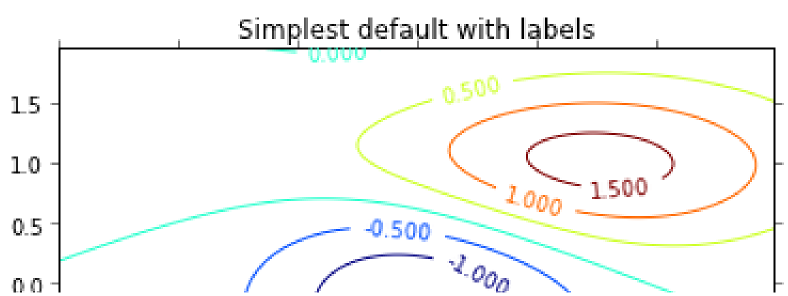
Have a look at the matplotlib gallery, find a cool looking figure, copy the code into the box below, and modify it. Note that some of the examples might require packages that are not installed on your machine (in particular those that make maps) - if this is the case, pick another example for the purposes of this exercise. In IPython, you can use the "load magic". Type %load and then the URL of the py file containing the code, and it will automatically copy it into a cell below. Run the cell with the code to see the figure. In [1]:

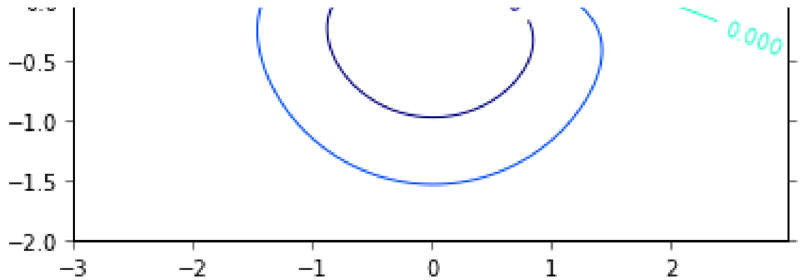
In [17]:
```
# Try it here...
%load http://matplotlib.org/mpl_examples/pylab_examples/contour_demo.py
```

In [18]:
```
#!/usr/bin/env python
"""
Illustrate simple contour plotting, contours on an image with
a colorbar for the contours, and labelled contours.

See also contour_image.py.
"""
import matplotlib
import numpy as np
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

matplotlib.rcParams['xtick.direction'] = 'out'
matplotlib.rcParams['ytick.direction'] = 'out'

delta = 0.025
x = np.arange(-3.0, 3.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = mlab.bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)
Z2 = mlab.bivariate_normal(X, Y, 1.5, 0.5, 1, 1)
# difference of Gaussians
Z = 10.0 * (Z2 - Z1)



# Create a simple contour plot with labels using default colors.  The
# inline argument to clabel will control whether the labels are draw
# over the line segments of the contour, removing the lines beneath
# the label
plt.figure()
CS = plt.contour(X, Y, Z)
plt.clabel(CS, inline=1, fontsize=10)
plt.title('Simplest default with labels')


# contour labels can be placed manually by providing list of positions
```

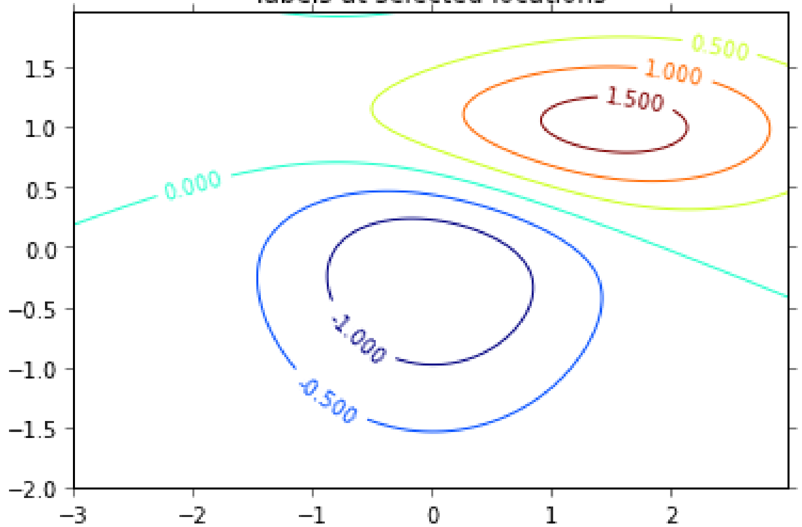
```
# (in data coordinate). See ginput_manual_clabel.py for interactive
# placement.
plt.figure()
CS = plt.contour(X, Y, Z)
manual_locations = [(-1, -1.4), (-0.62, -0.7), (-2, 0.5), (1.7, 1.2), (2.0, 1.4), (2.4, 1.7)]
plt.clabel(CS, inline=1, fontsize=10, manual=manual_locations)
plt.title('labels at selected locations')


# You can force all the contours to be the same color.
plt.figure()
CS = plt.contour(X, Y, Z, 6,
                 colors='k', # negative contours will be dashed by default
                 )
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Single color - negative contours dashed')

# You can set negative contours to be solid instead of dashed:
matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
plt.figure()
CS = plt.contour(X, Y, Z, 6,
                 colors='k', # negative contours will be dashed by default
                 )
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Single color - negative contours solid')


# And you can manually specify the colors of the contour
plt.figure()
CS = plt.contour(X, Y, Z, 6,
                 linewidths=np.arange(.5, 4, .5),
                 colors=('r', 'green', 'blue', (1,1,0), '#afeeee', '0.5')
                 )
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Crazy lines')


# Or you can use a colormap to specify the colors; the default
# colormap will be used for the contour lines
plt.figure()
im = plt.imshow(Z, interpolation='bilinear', origin='lower',
                cmap=cm.gray, extent=(-3,3,-2,2))
levels = np.arange(-1.2, 1.6, 0.2)
```

```
CS = plt.contour(Z, levels,
                 origin='lower',
                 linewidths=2,
                 extent=(-3,3,-2,2))

#Thicken the zero contour.
zc = CS.collections[6]
plt.setp(zc, linewidth=4)

plt.clabel(CS, levels[1::2],  # label every second level
           inline=1,
           fmt='%1.1f',
           fontsize=14)

# make a colorbar for the contour lines
CB = plt.colorbar(CS, shrink=0.8, extend='both')

plt.title('Lines with colorbar')
#plt.hot()  # Now change the colormap for the contour lines and colorbar
plt.flag()

# We can still add a colorbar for the image, too.
CBI = plt.colorbar(im, orientation='horizontal', shrink=0.8)

# This makes the original colorbar look a bit out of place,
# so let's improve its position.

l,b,w,h = plt.gca().get_position().bounds
ll,bb,ww,hh = CB.ax.get_position().bounds
CB.ax.set_position([ll, b+0.1*h, ww, h*0.8])


plt.show()
```

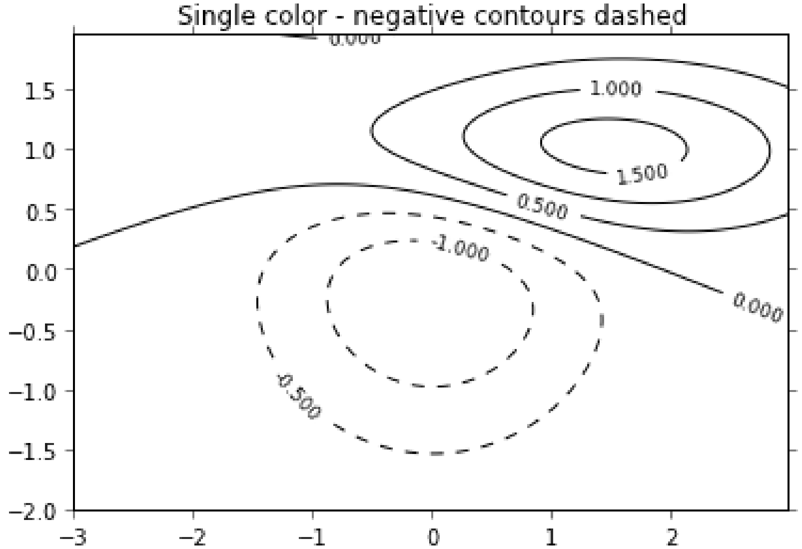




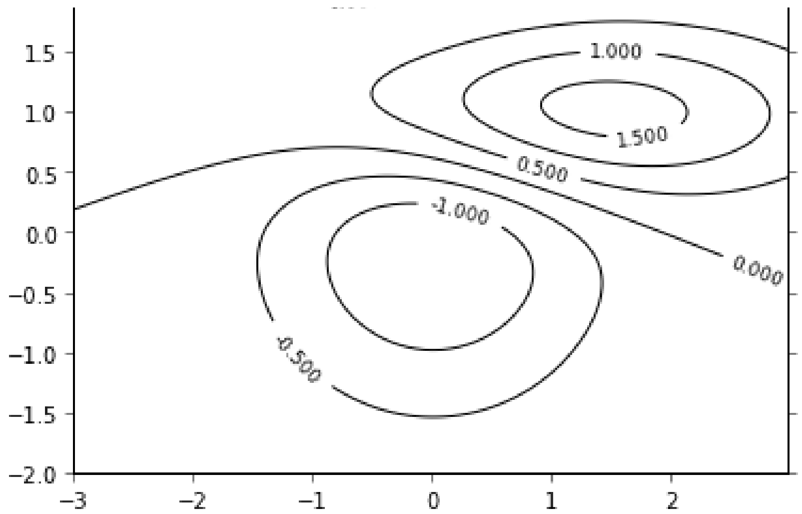
labels at selected locations

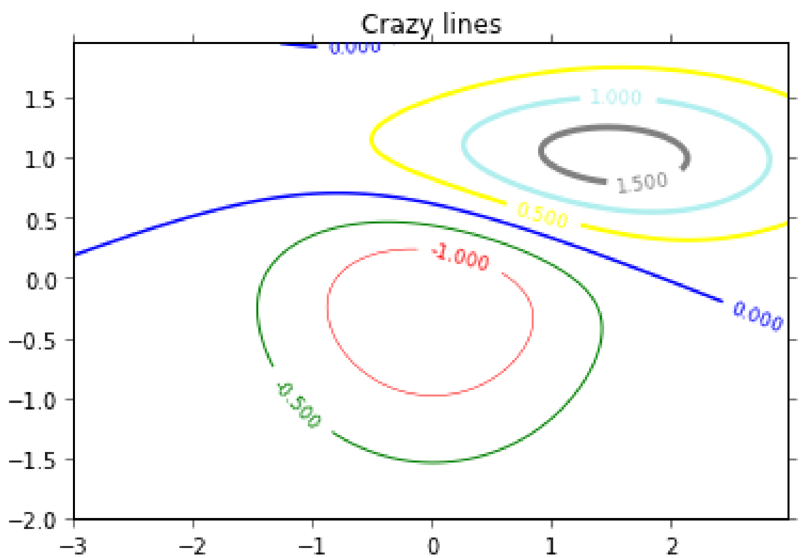


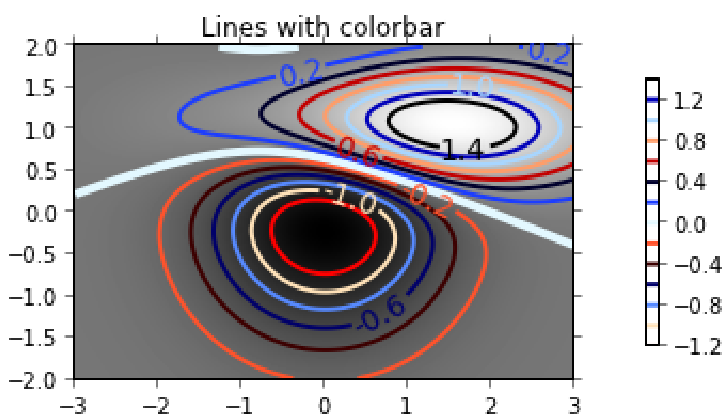
Single color - negative contours dashed

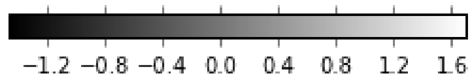


Single color - negative contours solid


1.000
1.500
0.500
-1.000
0.000
-0.500


Crazy lines
0.000
1.000
1.500
0.500
-1.000
0.000
-0.500


Lines with colorbar
-0.2
0.2
1.0
1.4
0.6
-1.0
0.2
-0.6
1.2
0.8
0.4
0.0
−0.4
−0.8
−1.2



A demo - how to plot a more complicated figure

We have data about the average high temperature and low temperature for each month at a small town in the US (Charleston, MO).
temp.txt

Month High Low
1 49.265 32.077
2 51.493 35.162
3 70.765 49.994
4 72.390 51.177
5 84.268 61.458
6 88.177 64.847
7 93.090 73.606
8 89.406 66.506
9 79.417 59.993
10 66.542 47.113
11 56.593 36.120
12 52.010 37.006

We will read this file and extract useful information and plot a bar figure to show the trend.

In [19]:
```
file_obj = open('temp_result.txt','r')
head = file_obj.readline()
month = []
low_temp = []
high_temp = []
for line in file_obj:
    line = line.strip()
    columns = line.split()
    month.append(int(columns[0]))
    high_temp.append(float(columns[1]))
    low_temp.append(float(columns[2]))
```

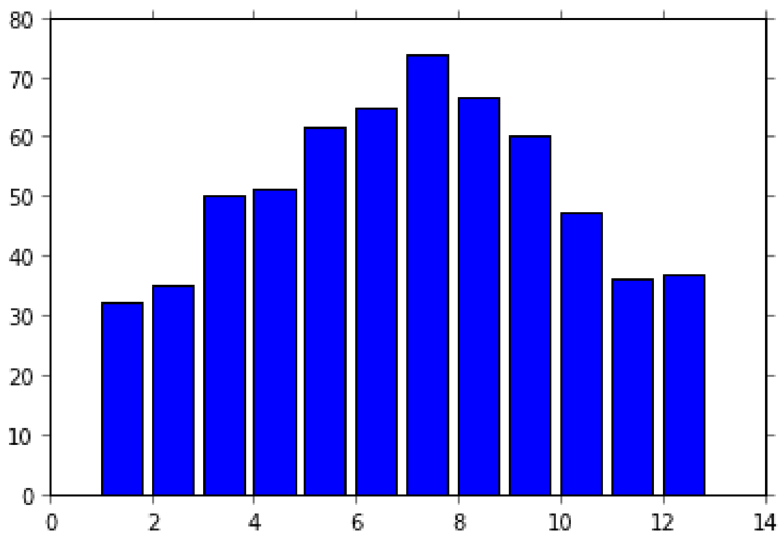
```
print month
print low_temp
print high_temp
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
[32.077, 35.162, 49.994, 51.177, 61.458, 64.847, 73.606, 66.506, 59.993, 47.113, 36.12, 37.006]
[49.265, 51.493, 70.765, 72.39, 84.268, 88.177, 93.09, 89.406, 79.417, 66.542, 56.593, 52.01]
```

Simple bar figure

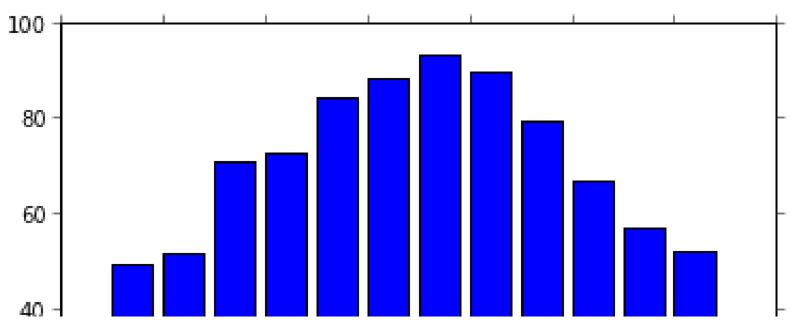
In [20]:
```
plt.bar(month,low_temp)
```

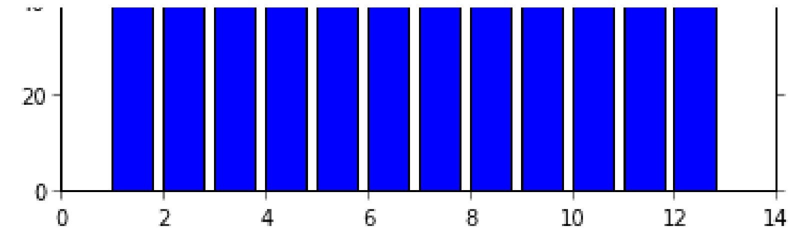
Out[20]: <Container object of 12 artists>



In [21]:
```
plt.bar(month,high_temp)
```

Out[21]: <Container object of 12 artists>





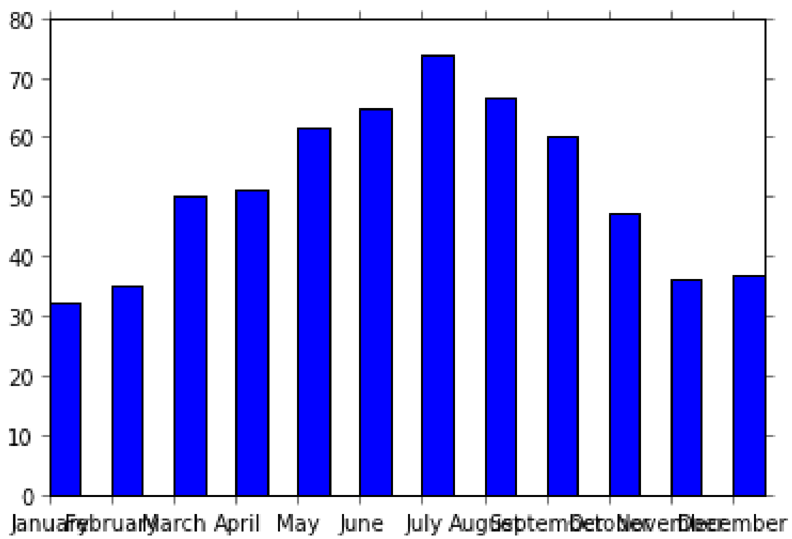
See http://matplotlib.org/examples/pylab_examples/custom_ticker1.html (http://matplotlib.org/examples/pylab_examples/custom_ticker1.html)

In [22]:

```
month_list = ['January','February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'Octobe


bar_width = 0.5
month = numpy.array(month)+2
print month
#plt.xticks(month+bar_width/2.0,month_list,rotation=45)
plt.xticks(month,month_list)

plt.bar(month,low_temp,width=bar_width)
```

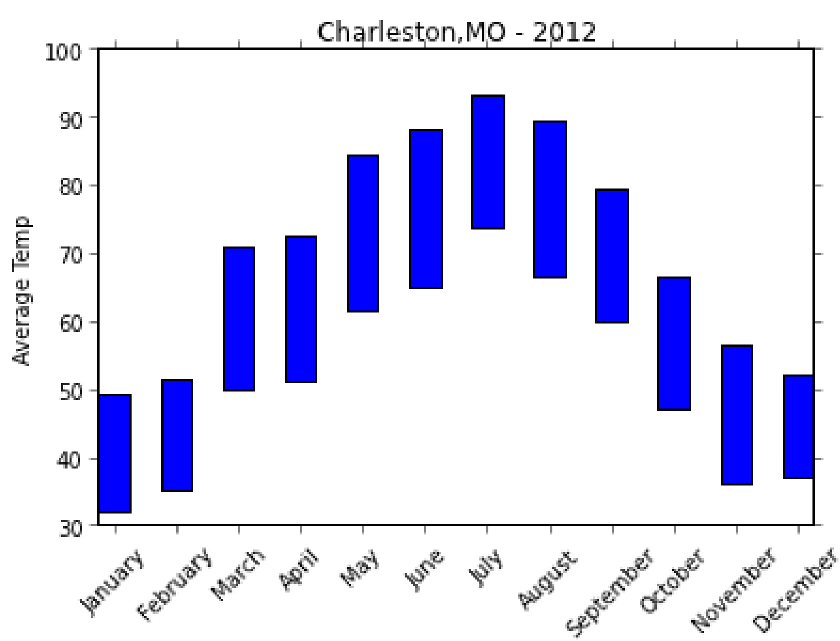
```
[ 3  4  5  6  7  8  9 10 11 12 13 14]
```

Out[22]: <Container object of 12 artists>



Floating bar

In [23]:
```
xList = numpy.arange(12)
month_list = ['January','February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'Octobe


bar_width = 0.5

plt.xticks(xList+bar_width/2.0,month_list,rotation=45)

plt.ylabel("Average Temp")

bar_h=[]
for i in range(len(xList)):
    bar_h.append(high_temp[i]-low_temp[i])


plt.bar(xList,bar_h,width=bar_width,color='b',bottom=low_temp)
plt.title('Charleston,MO - 2012')
plt.show()
plt.savefig('1.png')
plt.close()
```



We also save the figure to the directory. '1.png'

In [24]:
```
!ls
```

```
1.png                   SandBox.ipynb           solar_radiation.txt     temperature.txt
2.png                   matplotlib_intro.ipynb  temp_result.txt
README.md               process_data.py         temp_solar_result.txt
```

Final Exercise

Show the relationship between temperature and solar radiation.
We all know that the solar radiation is the most important resource of the heat on earth. So it is reasonable that the temperature is related to the solar radiation. However, what the relation will be is an interesting question. We all have some idea that the sunlight is the strongest at noon. Does that mean the temperature is the highest at noon either? We will examine that in this project. We will calculate the average solar radiation and average temperature at different time in a day in certain month. Then we can plot a figure to show the relation between them.

see [http://matplotlib.org/examples/api/two_scales.html (http://matplotlib.org/examples/api/two_scales.html)](http://matplotlib.org/examples/api/two_scales.html)

In [26]:
```
!more temp_solar_result.txt
```

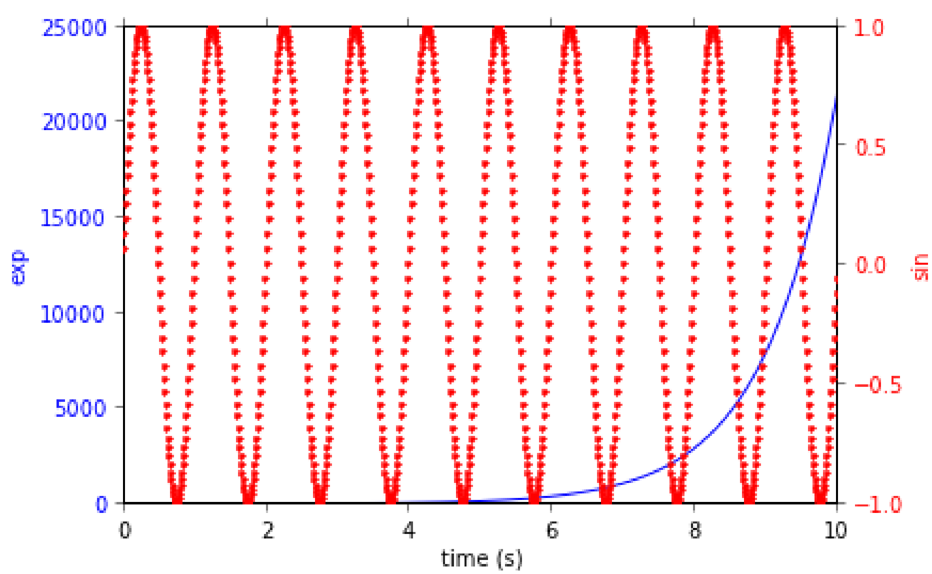
```
[?1h=Hour    Temp    Solar
1       68.820  0.000
2       67.587  0.000
3       66.750  0.000
4       65.737  0.000
5       65.193  4.567
6       66.507  81.300
7       70.430  227.667
8       74.443  396.800
9       78.150  567.167
10      80.987  685.167
11      83.073  722.133
12      84.540  761.033
13      85.880  765.733
14      86.670  715.267
15      87.220  625.433
16      87.410  493.533
17      87.090  356.300
18      85.517  193.367
```

```
19      82.920  66.833
20      78.750  3.200
21      75.453  0.000
22      73.520  0.000
23      72.093  0.000
24      70.843  0.000
[?1l>
```

In [27]:
```
%load http://matplotlib.org/mpl_examples/api/two_scales.py
```

In [28]:
```
#!/usr/bin/env python
"""

Demonstrate how to do two plots on the same axes with different left
right scales.


The trick is to use *2 different axes*.  Turn the axes rectangular
frame off on the 2nd axes to keep it from obscuring the first.
Manually set the tick locs and labels as desired.  You can use
separate matplotlib.ticker formatters and locators as desired since
the two axes are independent.

This is achieved in the following example by calling the Axes.twinx()
method, which performs this work. See the source of twinx() in
axes.py for an example of how to do it for different x scales. (Hint:
use the xaxis instance and call tick_bottom and tick_top in place of
tick_left and tick_right.)

The twinx and twiny methods are also exposed as pyplot functions.

"""

import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)
t = np.arange(0.01, 10.0, 0.01)
s1 = np.exp(t)
ax1.plot(t, s1, 'b-')
ax1.set_xlabel('time (s)')
```

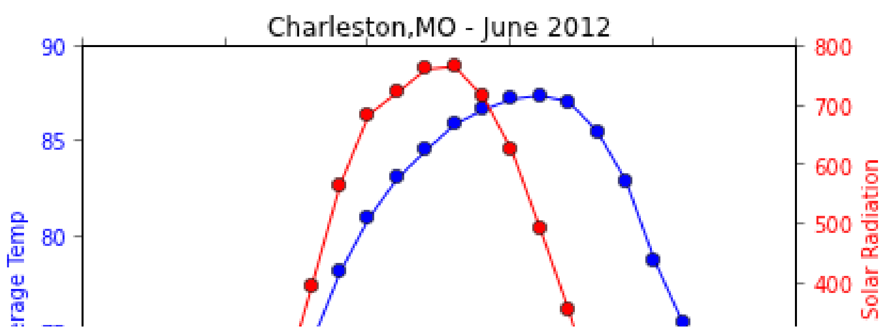
```
# Make the y-axis label and tick labels match the line color.
ax1.set_ylabel('exp', color='b')
for tl in ax1.get_yticklabels():
    tl.set_color('b')


ax2 = ax1.twinx()
s2 = np.sin(2*np.pi*t)
ax2.plot(t, s2, 'r.')
ax2.set_ylabel('sin', color='r')
for tl in ax2.get_yticklabels():
    tl.set_color('r')
plt.show()
```

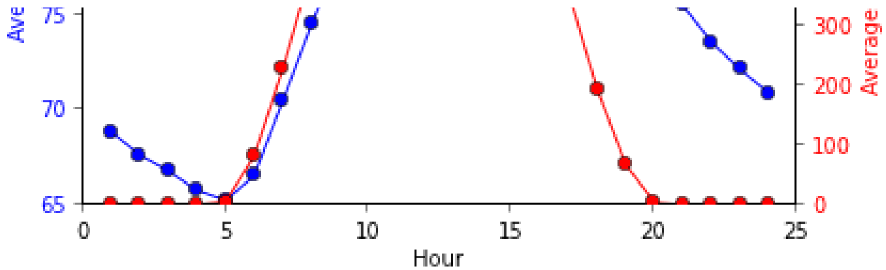


In [30]:
```
file_obj = open('temp_solar_result.txt','r')
head = file_obj.readline()
hour = []
temp = []
solar = []
for line in file_obj:
    line = line.strip()
    columns = line.split()
    hour.append(int(columns[0]))
```

```
    temp.append(float(columns[1]))
    solar.append(float(columns[2]))

print hour
print temp
print solar


fig = plt.figure(1)
ax1 = fig.add_subplot(111)
ax1.plot(hour,temp,'bo-')
ax1.set_xlabel('Hour')
ax1.set_ylabel('Average Temp',color='b')
for tl in ax1.get_yticklabels():
    tl.set_color('b')

ax2 = ax1.twinx()
ax2.plot(hour,solar,'ro-')
ax2.set_ylabel('Average Solar Radiation',color='r')
for tl in ax2.get_yticklabels():
    tl.set_color('r')

plt.title('Charleston,MO - June 2012')
plt.show()
plt.savefig('2.png')
plt.close()
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
[68.82, 67.587, 66.75, 65.737, 65.193, 66.507, 70.43, 74.443, 78.15, 80.987, 83.073, 84.54, 85.88, 86.67,
87.22, 87.41, 87.09, 85.517, 82.92, 78.75, 75.453, 73.52, 72.093, 70.843]
[0.0, 0.0, 0.0, 0.0, 4.567, 81.3, 227.667, 396.8, 567.167, 685.167, 722.133, 761.033, 765.733, 715.267,
625.433, 493.533, 356.3, 193.367, 66.833, 3.2, 0.0, 0.0, 0.0, 0.0]
```





One More Thing...

How does "temp_result.txt" and "temp_solar_result.txt" come from? Write seperate python script to process raw data...

In [31]:
```
%load process_data.py
```

In [ ]:
```
#!/usr/bin/env python


  #MONTH        DAY       YEAR       HOUR  AVG TEMP
  #                                  CST        ?C
  #      1         1       2007       100       -0.4
  #      1         1       2007       200       -0.6
  #      1         1       2007       300       -0.6
  #      1         1       2007       400       -0.3
  #      1         1       2007       500        0.0
  #      1         1       2007       600        0.4
  #      1         1       2007       700        0.4
  #      1         1       2007       800       -0.2
  #      1         1       2007       900       -0.3
  #      1         1       2007      1000        0.6
  #      1         1       2007      1100        1.9
  #      1         1       2007      1200        3.3
  #      1         1       2007      1300        4.7
  #      1         1       2007      1400        5.8


f_temp = open('temperature.txt','r')
```

```python
# average high temp and low temp of every month

e = f_temp.readline()
e = f_temp.readline()
lines = f_temp.readlines()
ave_high_temp_list=[]
ave_low_temp_list = []


for i in range(1,13):
    sum_high_temp=0
    sum_low_temp = 0
    day = 0
    for line in lines:
        line=line.rstrip()
        fields = line.split()
        month = int(fields[0])
        hour = fields[3]
        temp = float(fields[4])
        if  month== i:
            if hour == "100":
                temps=[]
                temps.append(temp)
            elif hour == "2400":
                temps.append(temp)
                high_temp = max(temps)
                #print high_temp
                sum_high_temp = sum_high_temp + high_temp
                low_temp = min(temps)
                sum_low_temp = sum_low_temp + low_temp
                day=day+1

            else:
                temps.append(temp)
    ave_high_temp = sum_high_temp/day
    ave_low_temp = sum_low_temp/day
    ave_high_temp_list.append(ave_high_temp)
    ave_low_temp_list.append(ave_low_temp)


file_out = open('temp_result.txt','w')
file_out.write('Month\tHigh\tLow\n')
for i in range(12):
```

```
    month = i+1

    ave_high_temp = '%.3f' %(ave_high_temp_list[i])
    ave_low_temp = '%.3f' %(ave_low_temp_list[i])
    file_out.write(str(month)+'\t'+str(ave_high_temp)+'\t'+str(ave_low_temp)+'\n')




  #MONTH        DAY       YEAR       HOUR  AVG TEMP
  #                                   CST        ?C
  #     1          1       2007        100      -0.4
  #     1          1       2007        200      -0.6
  #     1          1       2007        300      -0.6
  #     1          1       2007        400      -0.3



f_temp = open('temperature.txt','r')
f_solar = open('solar_radiation.txt','r')

# average high temp and low temp of every month

e = f_temp.readline()
e = f_temp.readline()
temp_lines = f_temp.readlines()

e = f_solar.readline()
e = f_solar.readline()
e = f_solar.readline()
e = f_solar.readline()
solar_lines = f_solar.readlines()


ave_temp_list=[]

for h in range(1,25):
    h = h*100
    sum_temp = 0
    day = 0
    for line in temp_lines:
        line=line.rstrip()
        fields = line.split()
```

```
        month = int(fields[0])
        hour = int(fields[3])
        temp = float(fields[4])
        if  month== 6 and  hour == h:
            sum_temp = sum_temp+temp
            day = day+1
    ave_temp = sum_temp/day
    ave_temp_list.append(ave_temp)

ave_solar_list = []
for h in range(1,25):
    h = h*100
    sum_solar = 0
    day = 0
    for line in solar_lines:
        line=line.rstrip()
        fields = line.split()
        month = int(fields[0])
        hour = int(fields[3])
        solar = float(fields[4])
        if  month== 6 and  hour == h:
            sum_solar = sum_solar+solar
            day = day+1
    ave_solar = sum_solar/day
    ave_solar_list.append(ave_solar)


file_out = open('temp_solar_result.txt','w')
file_out.write('Hour\tTemp\tSolar\n')

for i in range(24):
    hour = i+1

    ave_temp = '%.3f' %(ave_temp_list[i])
    ave_solar = '%.3f' %(ave_solar_list[i])
    file_out.write(str(hour)+'\t'+str(ave_temp)+'\t'+str(ave_solar)+'\n')
```

Better to run the script in Terminal, especiall if it will take too long to process.
$python process_data.py

In [ ]: