Welcome to Big Data!!!

Today we are going to analyze Rainfall data from central Park to get you started

We are going to do this all in a Jupyter notebook which was formerly called iPython. This will take some getting used to. You actually do your programming in a window in a web browser or in a program. But you are not online. You are running off of your own computer. But we will teach you all you need to do. Make sure to type everything you see correctly. All syntax matters including capital letters and punctuation.

Ok, Lets get you started. But make sure you read the syllabus and get a binder to keep the handouts. You need to do the code.org also!

We are going to use Anaconda python. We have used Enthought Canopy but anaconda seems to work better.

Look on the course site for directions to open Anaconda and then you can move around directories. Remember nothing gets saved on the school computers. so whatever you save each day you need to email to yourself or save some home

Open Jupyter notebook like in the video

Now we are in iPython!!!!

You see that I am using Markdown to add these notes. You make markdown by choosing Markdown above you can change from code to Markdown. Notes are essential to understanding everything. Please make a new cell and add some notes! We will do more of this later. If you download the notebook from Courseworks under syllabus you can double click on the notes and see how I added links! I will show you how to add equations and images later!

But we wanted to analyze precipitation data from Central Park. What questions could we ask and answer? I have downloaded the data from 1876 onward! I want to know the precipiation in the month I was born? I want to know the annual average precipiation in Central Park and I also want to know what monthly Precipitation looks like. Lets see if we can figure that out today. We are going to dive in really deep the first class and show you the power of programming. Then over the course of the semester I will teach you how to do this on your own. So enjoy the first analysis, see what we can do and then work on learning it yourself during the semester.

The first thing we will do is import the python libraries that we will use. When programming you don't use the whole Python programming language. You have a basic part of Python that is always turned-on but also you import or turn on the parts of the language that you will use for your analysis. You will learn about different libraries as the semester progresses.

Type in what you see below. Then to execute the code hit shift-enter. The little circle will fill in on the top right as the computer computes code. It needs to be exact, uppercase, lowercase etc. TYPE EXACTLY!

In [1]:

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib # for plot data
import matplotlib.pylab as plt
```

Now we need to get our data. I will use github a lot this semester. It is a website that allows you to update and share code and to keep track of it. It is like Dropbox on steroids. I got the data from NOAA https://www.ncdc.noaa.gov/cdo-web/datasets/GHCND/stations/GHCND:USW00094728/detail.

The information for this class is at this website https://github.com/bmaillou/BigDataPython

This is the cool part. We don't need to get it. We can just link it below. But you can look at the data and download if you want. follow the link to central park csv file. Then click on it and look at the raw data. csv files are a great way of getting data. csv means commas seperated variables. It is just a text file with data seperated by commas. now copy the link so we can add it below. The link goes to the url. Then we are using the pandas reader to get the data. We are then indexing by Date to organize. In pandas data is usually called "df" for dataframes. It is almost like a monster excel sheet.

In [63]:

```
url='https://raw.githubusercontent.com/bmaillou/BigDataPython/master/CentralPark2022.csv
df = pd.read_csv(url,index_col='DATE',parse_dates=['DATE']) #This is central park data
```

Now the data is in the computer as df. You could just type df to see it. I typed df.head() to see the first 5 rows. Type df.tail() to see the last 5.

In [64]:

```
df.tail()
```

Out[64]:

	STATION	NAME	DASF	MDSF	PRCP	SNOW	SNWD
DATE							
2022-08-24	USW00094728	NY CITY CENTRAL PARK, NY US	NaN	NaN	0.00	0.0	0.0
2022-08-25	USW00094728	NY CITY CENTRAL PARK, NY US	NaN	NaN	0.00	0.0	0.0
2022-08-26	USW00094728	NY CITY CENTRAL PARK, NY US	NaN	NaN	0.24	0.0	0.0
2022-08-27	USW00094728	NY CITY CENTRAL PARK, NY US	NaN	NaN	0.00	0.0	0.0
2022-08-28	USW00094728	NY CITY CENTRAL PARK, NY US	NaN	NaN	0.00	0.0	0.0

You can do a lot with the data. For some quick ideas type df. with the period and hit tab. The list of functions comes up.

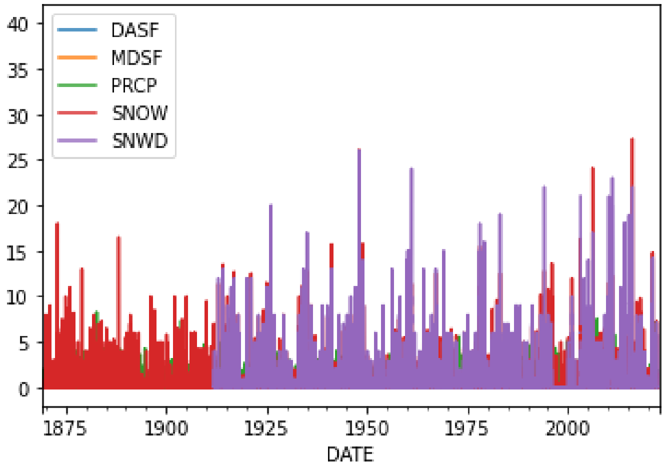
In [ ]:

```

```

But lets make a quick and dirty plot. To plot all the data you type df.plot() and then to show the plot you type plt.show()

In [34]:

```
df.plot()
plt.show()
```



I want to know what type of data I have. So i need to check the data type, show how to type data1. Hit tab and then see what we can do.

In [28]:
```
df.dtypes
```

Out[28]:
```
STATION     object
NAME        object
DASF       float64
MDSF       float64
PRCP       float64
SNOW       float64
SNWD       float64
dtype: object
```

But pandas has better built in functions. try df.info() to learn more about your data

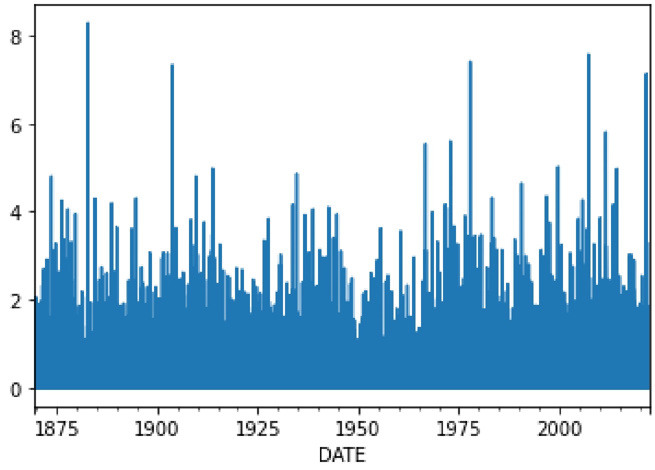
In [29]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 56064 entries, 1869-02-28 to 2022-08-28
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   STATION  56064 non-null  object 
 1   NAME     56064 non-null  object 
 2   DASF     56 non-null     float64
 3   MDSF     56 non-null     float64
 4   PRCP     56064 non-null  float64
 5   SNOW     55901 non-null  float64
 6   SNWD     39565 non-null  float64
dtypes: float64(5), object(2)
memory usage: 3.4+ MB
```

We are only interested in precipitation today. So lets just plot it. We can subchoose just that column.

In [30]:
```
df['PRCP'].plot()
```

Out[30]:
```
<AxesSubplot:xlabel='DATE'>
```



We can use describe to look at the description of the data.

In [31]:
```
df.describe()
```

Out[31]:

	DASF	MDSF	PRCP	SNOW	SNWD
count	56.0	56.000000	56064.000000	55901.000000	39565.000000
mean	2.0	5.172679	0.124627	0.074042	0.279201
std	0.0	5.786808	0.352954	0.625393	1.451665
min	2.0	0.310000	0.000000	0.000000	0.000000
25%	2.0	2.010000	0.000000	0.000000	0.000000
50%	2.0	4.020000	0.000000	0.000000	0.000000
75%	2.0	6.627500	0.050000	0.000000	0.000000
max	2.0	40.000000	8.280000	27.300000	26.000000

Stop and think for a second. What do the results mean above? They say we have over 50,000 precipitation values! We know something about them and we have done this really quickly!

Can you just show the describe for Precitation? Use what you learned above

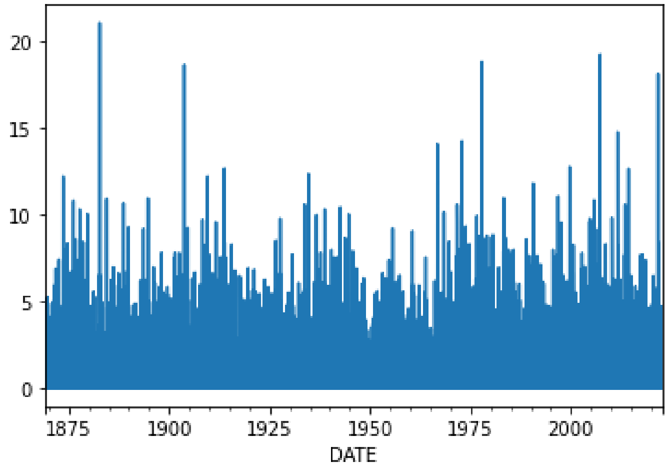
In [35]:
```

```

Out[35]:
```
count    56064.000000
mean         0.124627
std          0.352954
min          0.000000
25%          0.000000
50%          0.000000
75%          0.050000
max          8.280000
Name: PRCP, dtype: float64
```

We are lucky this year. The data is now in inches! They used to give you the data in 10's of mm. That is cm's but I am npt sure why they called it that. So lets convert to cm as practice.
1 in = 2.54 cm.

In [36]:
```
(df.PRCP*2.54/1).plot()  #we are doing the math and plotting at the same time!
```

Out[36]:
```
<AxesSubplot:xlabel='DATE'>
```



Again, stop and think about the data? What are the biggest rainstorms we have gotten? How does this compare to some hurricanes you hear about on the news?? or some other big storm near you? What about Ida?

lets convert to cm so we can save it forever!

```
In [37]: df.PRCP=df.PRCP*2.54/1
```

Check and make sure it worked using describe. Does the data make sense?

Make sure not to convert twice above. Everytime you convert is saves it. Your numbers will keep getting smaller and smaller.....

If you convert more than once you have to go up and re-read in the data

```
In [38]:
```

```
Out[38]: count    56064.000000
         mean         0.316553
         std          0.896502
         min          0.000000
         25%          0.000000
         50%          0.000000
         75%          0.127000
         max         21.031200
         Name: PRCP, dtype: float64
```

I like inches! So lets convert back and then describe again.

```
In [39]: df.PRCP=df.PRCP/2.54
         df.PRCP.describe()
```

```
Out[39]: count    56064.000000
         mean         0.124627
         std          0.352954
         min          0.000000
         25%          0.000000
         50%          0.000000
         75%          0.050000
         max          8.280000
         Name: PRCP, dtype: float64
```

can we just get yearly rainfall? We can resample the data and sum all the rainfall for each year! Plus look and I am adding a comment to the code below!

```
In [40]: df['PRCP'].resample('A').sum()  #This just gave us yearly rainfall!
```

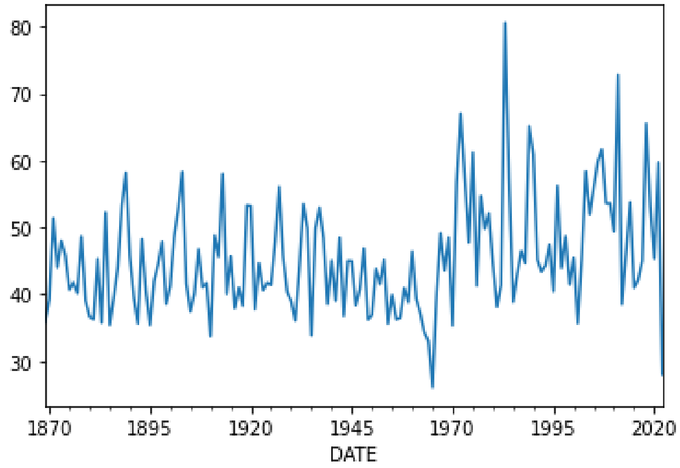
```
Out[40]: DATE
```

```
1869-12-31    35.85
1870-12-31    39.25
1871-12-31    51.38
1872-12-31    43.99
1873-12-31    47.99
              ...  
2018-12-31    65.55
2019-12-31    53.03
2020-12-31    45.35
2021-12-31    59.73
2022-12-31    28.02
Freq: A-DEC, Name: PRCP, Length: 154, dtype: float64
```

Now we could plot the yearly data!

```
In [41]: df['PRCP'].resample('A').sum().plot()
```

```
Out[41]: <AxesSubplot:xlabel='DATE'>
```

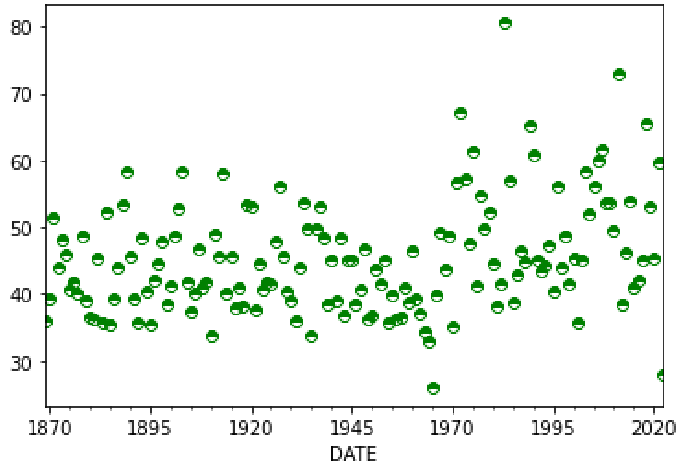


We could also make it a scatter plot with points and no line. This is the power of plotting with matplotlib. You can change anything!

go to http://matplotlib.org/1.3.1/examples/pylab_examples/filledmarker_demo.html and choose your own color and style. if you want to turn the line off do linestyle='none'

```
In [42]: df['PRCP'].resample('A').sum().plot(marker='8',color='g',fillstyle='top',linestyle='none
```

```
Out[42]: <AxesSubplot:xlabel='DATE'>
```



Can you use describe to give statistics on the annual data? How much rain fell on the wettest year?

```
In [44]: 
```

Out[44]:

```
count    154.000000
mean      45.370714
std        8.485716
min       26.090000
25%       39.335000
50%       44.415000
75%       49.307500
max       80.560000
Name: PRCP, dtype: float64
```

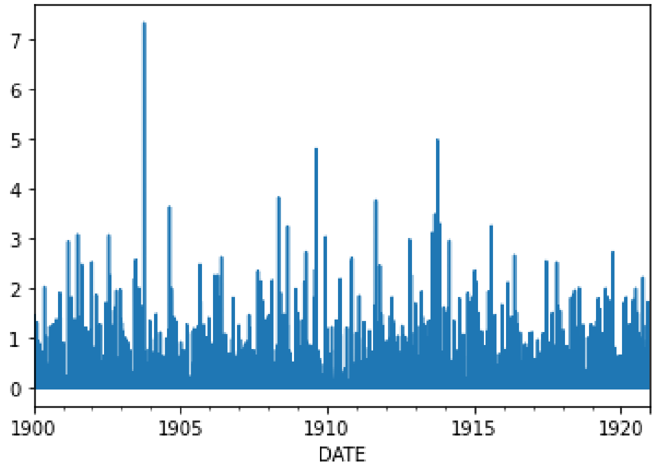
What if we only want to plot some of the data? Lets plot from 1900 to 1920...

In [45]:

```
df.PRCP['1900':'1920'].plot()
```

Out[45]:

```
<AxesSubplot:xlabel='DATE'>
```



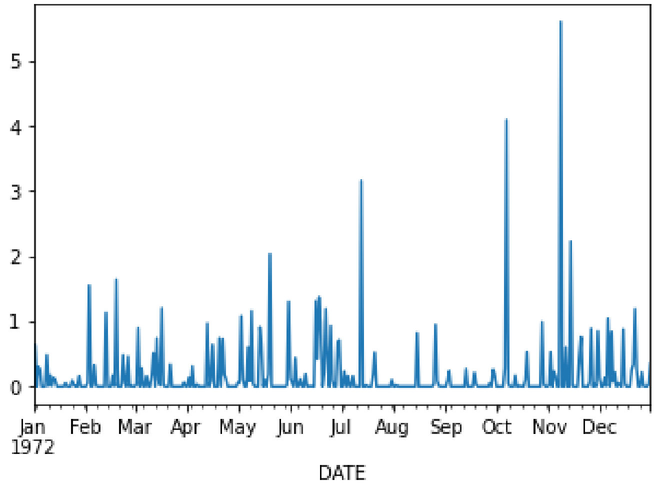
or for just one year

In [46]:

```
df.PRCP['1972'].plot()
```

Out[46]:

```
<AxesSubplot:xlabel='DATE'>
```



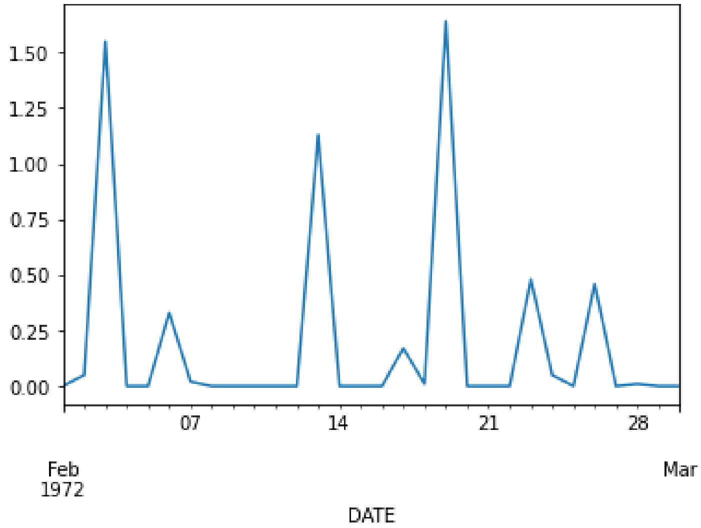
or one month!

In [47]:

```
df.PRCP['02/01/1972':'03/01/1972'].plot()
```

Out[47]:

```
<AxesSubplot:xlabel='DATE'>
```



Can you plot for the month and or year you were born?

In [ ]:

```

```

Now we are going to do some big changes. (I am also showing you here that you can do bullets in the Markdown)

My end point is to look at the monthly data in boxplots.

- But to get there we need to resample to monthly.
- Then we need to make a new column that lists the months.
- Then we can make a boxplot!

So lets start by making a new dataframe which is the data resampled by summing the monthly data.

In [55]:

```
df_month=df.resample('M').sum()
```

Now we can describe the data

In [56]:

```
df_month.describe()
```

Out[56]:

	DASF	MDSF	PRCP	SNOW	SNWD
count	1843.000000	1843.000000	1843.000000	1843.000000	1843.000000
mean	0.060770	0.157173	3.791150	2.245795	5.993814
std	0.437887	1.508668	2.128407	4.847776	24.271023
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	2.310000	0.000000	0.000000
50%	0.000000	0.000000	3.430000	0.000000	0.000000
75%	0.000000	0.000000	4.810000	2.000000	0.000000
max	6.000000	40.000000	18.950000	36.900000	346.000000

So things appear to be working. What does our data look like?

In [57]:

```
print (df_month) # a big difference between python 2 and 3 is you need parentheses on th
```

```
                 DASF   MDSF   PRCP   SNOW   SNWD
DATE
```

```
1869-02-28   0.0    0.0  0.00   0.0    0.0
1869-03-31   0.0    0.0  4.61   0.8    0.0
1869-04-30   0.0    0.0  1.39   0.0    0.0
1869-05-31   0.0    0.0  4.15   0.0    0.0
1869-06-30   0.0    0.0  4.40   0.0    0.0
...          ...    ...   ...   ...    ...
2022-04-30   0.0    0.0  4.53   0.0    0.0
2022-05-31   0.0    0.0  4.52   0.0    0.0
2022-06-30   0.0    0.0  2.92   0.0    0.0
2022-07-31   0.0    0.0  4.55   0.0    0.0
2022-08-31   0.0    0.0  1.59   0.0    0.0

[1843 rows x 5 columns]
```

To make our plotting work we will need a new column that tells us just the month. So we are going to look at the index date and pull out the month and put it into a new column.

In [58]:

```
df_month['month']=df_month.index.month
```

lets just look at 5 rows and see if it worked.

In [59]:

```
df_month.head()  #head just gives you the first 5
```

Out[59]:

	DASF	MDSF	PRCP	SNOW	SNWD	month
DATE						
1869-02-28	0.0	0.0	0.00	0.0	0.0	2
1869-03-31	0.0	0.0	4.61	0.8	0.0	3
1869-04-30	0.0	0.0	1.39	0.0	0.0	4
1869-05-31	0.0	0.0	4.15	0.0	0.0	5
1869-06-30	0.0	0.0	4.40	0.0	0.0	6

Now lets make our boxplot! you can google pandas boxplot to see how. http://pandas.pydata.org/pandas-docs/stable/visualization.html

Foreshadowing. I am going to ask you to use this by notation in a future class!

this is a big hint. remember column and by. I AM GOING TO ASK THIS AGAIN!

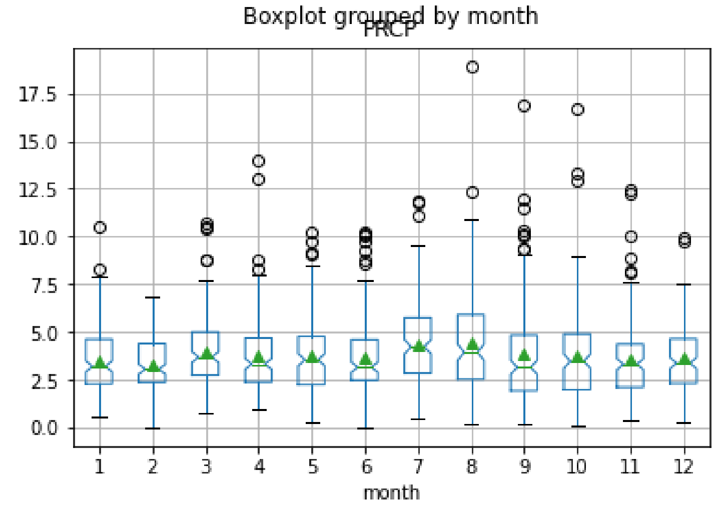
Also, to make it look fancy I am showing the mean values and making it have a notch. You can make both options False or just delete them. They are extra arguments in the function.

In [60]:

```
df_month.boxplot(column=['PRCP'],by='month',showmeans=True,notch=True)
```

Out[60]:

```
<AxesSubplot:title={'center':'PRCP'}, xlabel='month'>
```

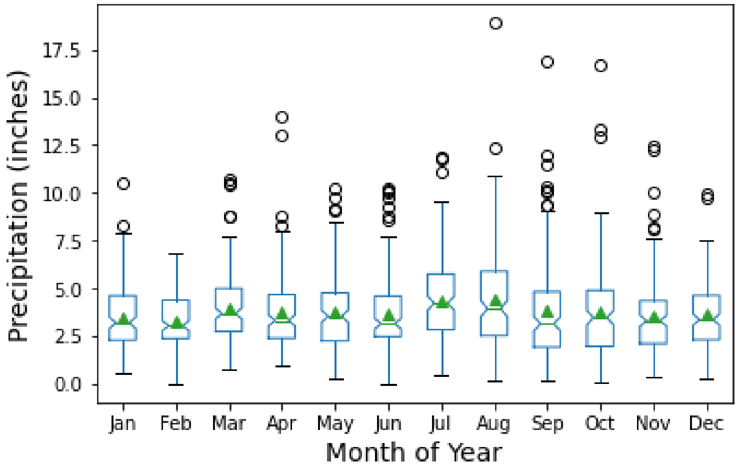


the great thing about python is you can make a professional looking graph and save it and have it ready to use. So lets do it.

- First we turn on fancy graphs. We do fig,ax=plt.subplots() this lets us control all parts of the graph in terms of the axes and the overall figure.
- First we remove the title and suptitle by making them nothing. ax.set_title('') fig.suptitle('') You could add titles if you wanted.
- We label our y-axis. Always label axes. ax.set_ylabel('Precipitation (inches)')
- We make the x-axis nicer.
- We make the label fonts bigger
- We save the file. I added some keywords to make the file nice. the dpi is dots per inch so it has better resolution and the bbox makes sure nothing is cut off.
- You can decide if you want to turn off gridlines. add grid=False
- for a hard one we could add month of year instead of 1-12. I used ax.set_xticklabels(['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'])

I think that is a sharp graph! You should be able to find it on your computer and mail it to yourself.

In [61]:
```
fig,ax=plt.subplots()
df_month.boxplot(column=['PRCP'],by='month',showmeans=True,notch=True,ax=ax,grid=False)
ax.set_title('')
fig.suptitle('')
ax.set_ylabel('Precipitation (inches)',fontsize=14)
ax.set_xlabel('Month of Year',fontsize=14)
ax.set_xticklabels(['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','D
fig.savefig('Central-Park-Monthly-Precip.jpg',dpi=600,bbox_inches='tight')
```



I talk about making figure captions in the homework. So lets make one here. Here is the definition of a boxplot

Figure 1. Boxplot of monthly precipitation Central Park, New York City from 1876 to 2015. The indent is the median, the triangle is the mean, the box goes from Q1 to Q3 with the whiskers showing the range of the data and circles are outliers larger than 1.5 times the interquartile range.

We will also need to learn about how to talk about results. We can see from the figure that precipitation does not systematically vary by month in Central Park and that the mean monthly precipitation is fairly uniform.

to get the monthly data we can groupby month and the describe the data. or just look at the mean. We will also only look at prcp. As you can see rain is pretty uniform by month in NYC!

```
In [62]: df_month.groupby('month').PRCP.mean()
```

```
Out[62]: month
         1     3.490065
         2     3.300649
         3     3.971688
         4     3.720325
         5     3.741818
         6     3.659286
         7     4.388442
         8     4.418961
         9     3.871765
         10    3.731830
         11    3.541765
         12    3.652876
         Name: PRCP, dtype: float64
```

I hope you got some of these to work and are excited about what we can do. You just analyzed 50,000 precipitation data points on your first day! We are ready to begin learning data analysis and Python! But for next class we have to go back to the basics!

```
In [ ]:
```