Range and Linspace

In this notebook I started putting the answers to the harder questions at the end for your reference.

At the end of last class people were a little confused about the function range and functions in general and there interactions with lists. So we are going to do more.

Functions are defined here https://www.tutorialspoint.com/python/python_functions.htm You can define your own function or use built in functions. Every time we import a library we are giving ourselves the chance to use its functions. In essence we are cheating and using code that was already written. This code is great as we can pass different parameters and it will give us a result. I always forget that all of our plotting is done by functions and depending on what you pass then determines how the plots look. We used some simple functions last time and I want to go through them again as they will keep coming up over and over again.

What was range again? It was a function that returned numbers. lets return numbers 0-1000 by 5's. We are going to use the numpy version which is arange.

Remember you call the function and it has parantheses. This function returns a list.

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
np.arange(0,1001,5)
```

Out[3]:
```
array([   0,    5,   10,   15,   20,   25,   30,   35,   40,   45,   50,
         55,   60,   65,   70,   75,   80,   85,   90,   95,  100,  105,
        110,  115,  120,  125,  130,  135,  140,  145,  150,  155,  160,
        165,  170,  175,  180,  185,  190,  195,  200,  205,  210,  215,
        220,  225,  230,  235,  240,  245,  250,  255,  260,  265,  270,
        275,  280,  285,  290,  295,  300,  305,  310,  315,  320,  325,
        330,  335,  340,  345,  350,  355,  360,  365,  370,  375,  380,
        385,  390,  395,  400,  405,  410,  415,  420,  425,  430,  435,
        440,  445,  450,  455,  460,  465,  470,  475,  480,  485,  490,
        495,  500,  505,  510,  515,  520,  525,  530,  535,  540,  545,
        550,  555,  560,  565,  570,  575,  580,  585,  590,  595,  600,
        605,  610,  615,  620,  625,  630,  635,  640,  645,  650,  655,
        660,  665,  670,  675,  680,  685,  690,  695,  700,  705,  710,
        715,  720,  725,  730,  735,  740,  745,  750,  755,  760,  765,
        770,  775,  780,  785,  790,  795,  800,  805,  810,  815,  820,
        825,  830,  835,  840,  845,  850,  855,  860,  865,  870,  875,
        880,  885,  890,  895,  900,  905,  910,  915,  920,  925,  930,
        935,  940,  945,  950,  955,  960,  965,  970,  975,  980,  985,
        990,  995, 1000])
```

Now rememeber from the last class you made your own list of numbers

In [4]:
```
FirstList=[2,5,10,15,20]
```

Can we link this two ideas?

could we take those numbers from range and make them a list? Yes. Give your list a name (I called mine newList) and then set it equal to the range from above. then print your list. You can use an equal sign to set the arange list to a variable.

In [ ]:
```

```

Now you can access newList the same way we learned last class with square brackets.

In [7]:
```
newList[5]
```

Out[7]:
```
25
```

Aside.

You might hear me use newList=np.array([2,5,10,15,20])

- numpy is short for number python and is good for math. For programming we call numpy as np. so np means we are calling a numpy function.
- using a numpy array is just using a list that does math better
- in the beginning they are very similar but we are going to use numpy and then pandas
- pandas is our ultimate goal is the the best part of python for data analysis

In [9]:
```
newList=np.array([2,5,10,15,20])
```

In [10]:
```
newList
```

Out[10]:
```
array([ 2,  5, 10, 15, 20])
```

In [11]:
```
FirstList=[2,5,10,15,20]
```

In [12]:
```
FirstList
```

Out[12]:
```
[2, 5, 10, 15, 20]
```

In [ ]:
```

```

Now lets look at some more functions and try to learn more and do some math. By the time you finish this packet we will be plotting. I am going to step you through how to plot sine and cosine.

In [3]:
```
?np.sin
```

In [20]:
```
?np.pi
```

np.sin, np.cos, and np.pi are all numpy functions that we are calling and that are giving us results we can use.

Range only gives us integers.... This could be trouble. We can also use linpace. Lets get help.

In [22]:
```
?np.linspace
```

What does linspace do? how will this help us? Lets first start by using np.linspace to give us the number from 1 to 1000

In [4]:
```
np.linspace(0,1000)
```

Out[4]:
```
array([   0.        ,   20.40816327,   40.81632653,   61.2244898 ,
         81.63265306,  102.04081633,  122.44897959,  142.85714286,
        163.26530612,  183.67346939,  204.08163265,  224.48979592,
        244.89795918,  265.30612245,  285.71428571,  306.12244898,
        326.53061224,  346.93877551,  367.34693878,  387.75510204,
        408.16326531,  428.57142857,  448.97959184,  469.3877551 ,
        489.79591837,  510.20408163,  530.6122449 ,  551.02040816,
        571.42857143,  591.83673469,  612.24489796,  632.65306122,
        653.06122449,  673.46938776,  693.87755102,  714.28571429,
        734.69387755,  755.10204082,  775.51020408,  795.91836735,
        816.32653061,  836.73469388,  857.14285714,  877.55102041,
        897.95918367,  918.36734694,  938.7755102 ,  959.18367347,
        979.59183673, 1000.        ])
```

How is this different than arange? does it give us integers? this is better for giving us a set of numbers that are not an integer. Could you give me a set of numbers from 0 to 2*pi? you could type in 3.14 but in python pi can just be added and it will give you the number. As an aside which will become important later linspace is part of numpy so it gives you a numpy array which is like a super list. We will come back to this.

In [8]:
```
np.pi
```

Out[8]:
```
3.141592653589793
```

now can you produce the set of numbers from 0 to 2*pi using linspace?

In [ ]:
```

```

I want to plot the sine from 0 to 2 pi. How can we do this. Lets set our x and y and then plot them. I will show you how to do this for sine and then you can do it for cosine. In python sine takes radians and not degrees

In [6]:
```
x=np.linspace(0,2*np.pi)
y=np.sin(x)
```

In [7]:
```
print (x)
print (y)
```

```
[0.         0.12822827 0.25645654 0.38468481 0.51291309 0.64114136
 0.76936963 0.8975979  1.02582617 1.15405444 1.28228272 1.41051099
 1.53873926 1.66696753 1.7951958  1.92342407 2.05165235 2.17988062
 2.30810889 2.43633716 2.56456543 2.6927937  2.82102197 2.94925025
 3.07747852 3.20570679 3.33393506 3.46216333 3.5903916  3.71861988
 3.84684815 3.97507642 4.10330469 4.23153296 4.35976123 4.48798951
 4.61621778 4.74444605 4.87267432 5.00090259 5.12913086 5.25735913
 5.38558741 5.51381568 5.64204395 5.77027222 5.89850049 6.02672876
 6.15495704 6.28318531]
[ 0.00000000e+00  1.27877162e-01  2.53654584e-01  3.75267005e-01
  4.90717552e-01  5.98110530e-01  6.95682551e-01  7.81831482e-01
  8.55142763e-01  9.14412623e-01  9.58667853e-01  9.87181783e-01
  9.99486216e-01  9.95379113e-01  9.74927912e-01  9.38468422e-01
  8.86599306e-01  8.20172255e-01  7.40277997e-01  6.48228395e-01
  5.45534901e-01  4.33883739e-01  3.15108218e-01  1.91158629e-01
  6.40702200e-02 -6.40702200e-02 -1.91158629e-01 -3.15108218e-01
 -4.33883739e-01 -5.45534901e-01 -6.48228395e-01 -7.40277997e-01
 -8.20172255e-01 -8.86599306e-01 -9.38468422e-01 -9.74927912e-01
 -9.95379113e-01 -9.99486216e-01 -9.87181783e-01 -9.58667853e-01
 -9.14412623e-01 -8.55142763e-01 -7.81831482e-01 -6.95682551e-01
 -5.98110530e-01 -4.90717552e-01 -3.75267005e-01 -2.53654584e-01
 -1.27877162e-01 -2.44929360e-16]
```

Now do it for cosine

In [24]:

```
[ 0.          0.12822827  0.25645654  0.38468481  0.51291309  0.64114136
  0.76936963  0.8975979   1.02582617  1.15405444  1.28228272  1.41051099
  1.53873926  1.66696753  1.7951958   1.92342407  2.05165235  2.17988062
  2.30810889  2.43633716  2.56456543  2.6927937   2.82102197  2.94925025
  3.07747852  3.20570679  3.33393506  3.46216333  3.5903916   3.71861988
  3.84684815  3.97507642  4.10330469  4.23153296  4.35976123  4.48798951
  4.61621778  4.74444605  4.87267432  5.00090259  5.12913086  5.25735913
  5.38558741  5.51381568  5.64204395  5.77027222  5.89850049  6.02672876
  6.15495704  6.28318531]
[ 1.          0.99179001  0.96729486  0.92691676  0.8713187   0.80141362
  0.71834935  0.6234898   0.51839257  0.40478334  0.28452759  0.1595999
  0.03205158 -0.09602303 -0.22252093 -0.34536505 -0.46253829 -0.57211666
 -0.67230089 -0.76144596 -0.8380881  -0.90096887 -0.94905575 -0.98155916
 -0.99794539 -0.99794539 -0.98155916 -0.94905575 -0.90096887 -0.8380881
 -0.76144596 -0.67230089 -0.57211666 -0.46253829 -0.34536505 -0.22252093
 -0.09602303  0.03205158  0.1595999   0.28452759  0.40478334  0.51839257
  0.6234898   0.71834935  0.80141362  0.8713187   0.92691676  0.96729486
  0.99179001  1.        ]
```

What makes these functions really nice is that they do the math on the whole list for you. If you look it went through the list element wise and took the sine of each element. That is really cool and makes our life really easy. Now we can plot the sine of x.

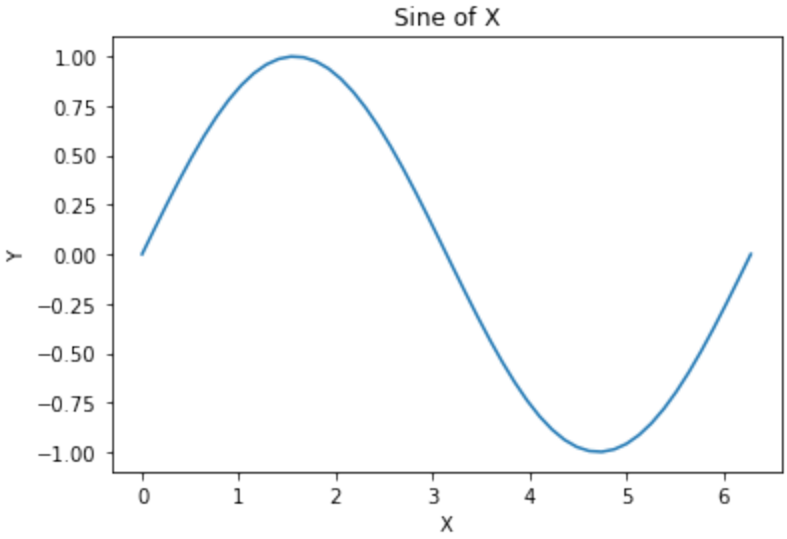
In [8]:

```
x=np.linspace(0,2*np.pi)
y=np.sin(x)

fig,ax=plt.subplots()
ax.plot(x,y)
ax.set_xlabel('X')  #added in
ax.set_ylabel('Y')  #added in
ax.set_title('Sine of X')
```

Out[8]:

```
Text(0.5,1,'Sine of X')
```



can you make the cosine and plot it?

In [49]:

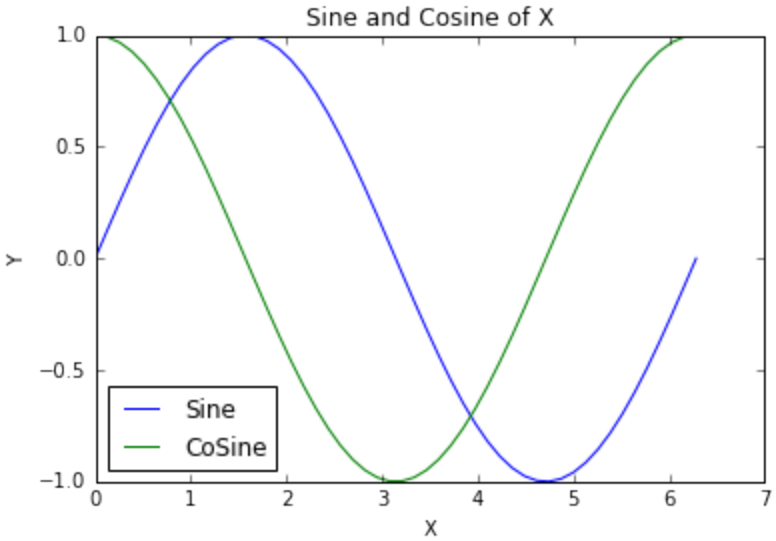
In [49]:

Now to show you some interesting plotting. We can plot both sine and cosine on the same plot. I won't do it for you. But if you call plot twice in a row it will put the plots onto one figure. This is becuase they are both using the same axes which we called ax. So call it twice, once with your x and sin(x) list and a second time with your x and cos(x) data.

I added a legend and label. Again this is foreshadowing. I will show you how soon.

In [13]:

Out[13]: `<matplotlib.legend.Legend at 0x1105780d0>`



In [ ]:

Remember

np.arange is for wanting a list of integers

np.linspace is for wanting evenly spaced numbers.

these are all functions that are easy to call

You can save these functions to a list(or array) you can call anything. We are calling them x and y right now.

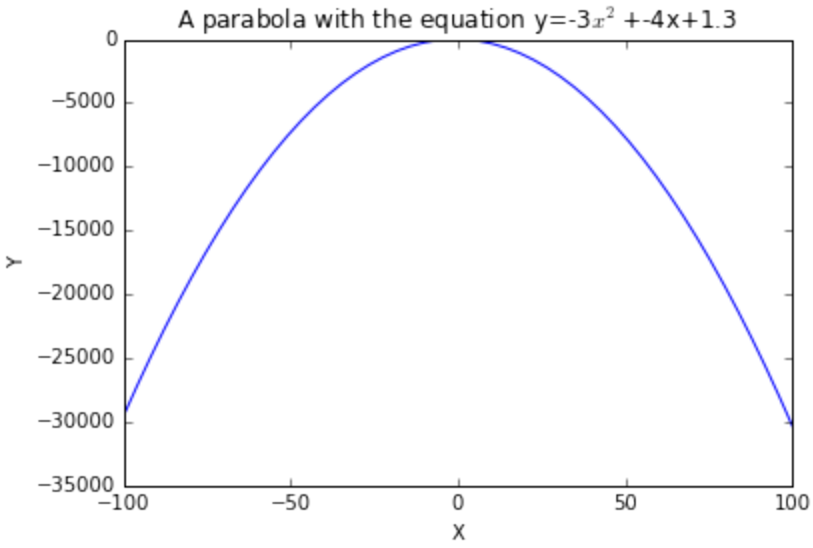
Let's Make a parabola. Remember

$$y = ax^2 + bx + c$$

- set an a,b,and c.
- get an x with np.linspace
- solve for y
- plot it.

In [46]:

Out[46]: <matplotlib.text.Text at 0xbb71d30>



In class I asked if you could set part of a list and this caused confusion. so lets play with this. remember you can access part of a list using brackets. so lets use the brackets to reset a number.

In [48]:
```
#here is our list
x
```

Out[48]:
```
array([-100.        ,  -95.91836735,  -91.83673469,  -87.75510204,
        -83.67346939,  -79.59183673,  -75.51020408,  -71.42857143,
        -67.34693878,  -63.26530612,  -59.18367347,  -55.10204082,
        -51.02040816,  -46.93877551,  -42.85714286,  -38.7755102 ,
        -34.69387755,  -30.6122449 ,  -26.53061224,  -22.44897959,
        -18.36734694,  -14.28571429,  -10.20408163,   -6.12244898,
         -2.04081633,    2.04081633,    6.12244898,   10.20408163,
         14.28571429,   18.36734694,   22.44897959,   26.53061224,
         30.6122449 ,   34.69387755,   38.7755102 ,   42.85714286,
         46.93877551,   51.02040816,   55.10204082,   59.18367347,
         63.26530612,   67.34693878,   71.42857143,   75.51020408,
         79.59183673,   83.67346939,   87.75510204,   91.83673469,
         95.91836735,  100.        ])
```

In [49]:
```
x[0]=0
x
```

Out[49]:
```
array([   0.        ,  -95.91836735,  -91.83673469,  -87.75510204,
        -83.67346939,  -79.59183673,  -75.51020408,  -71.42857143,
        -67.34693878,  -63.26530612,  -59.18367347,  -55.10204082,
        -51.02040816,  -46.93877551,  -42.85714286,  -38.7755102 ,
        -34.69387755,  -30.6122449 ,  -26.53061224,  -22.44897959,
        -18.36734694,  -14.28571429,  -10.20408163,   -6.12244898,
         -2.04081633,    2.04081633,    6.12244898,   10.20408163,
         14.28571429,   18.36734694,   22.44897959,   26.53061224,
         30.6122449 ,   34.69387755,   38.7755102 ,   42.85714286,
         46.93877551,   51.02040816,   55.10204082,   59.18367347,
         63.26530612,   67.34693878,   71.42857143,   75.51020408,
         79.59183673,   83.67346939,   87.75510204,   91.83673469,
         95.91836735,  100.        ])
```

I just changed the zeroth cell to zero. I could have changed it to anything.

In [51]:
```
x[0]=-21
x
```

Out[51]:
```
array([ -21.        ,  -95.91836735,  -91.83673469,  -87.75510204,
        -83.67346939,  -79.59183673,  -75.51020408,  -71.42857143,
        -67.34693878,  -63.26530612,  -59.18367347,  -55.10204082,
        -51.02040816,  -46.93877551,  -42.85714286,  -38.7755102 ,
        -34.69387755,  -30.6122449 ,  -26.53061224,  -22.44897959,
        -18.36734694,  -14.28571429,  -10.20408163,   -6.12244898,
         -2.04081633,    2.04081633,    6.12244898,   10.20408163,
         14.28571429,   18.36734694,   22.44897959,   26.53061224,
         30.6122449 ,   34.69387755,   38.7755102 ,   42.85714286,
         46.93877551,   51.02040816,   55.10204082,   59.18367347,
         63.26530612,   67.34693878,   71.42857143,   75.51020408,
         79.59183673,   83.67346939,   87.75510204,   91.83673469,
         95.91836735,  100.        ])
```

So I set 0 cell/spot/item to -21. You can set and reset how you would like because lists are mutablethis compares to strings which are immutable

In [52]:
```
x[:]=-100
x
```

Out[52]:
```
array([-100., -100., -100., -100., -100., -100., -100., -100., -100.,
       -100., -100., -100., -100., -100., -100., -100., -100., -100.,
       -100., -100., -100., -100., -100., -100., -100., -100., -100.,
       -100., -100., -100., -100., -100., -100., -100., -100., -100.,
       -100., -100., -100., -100., -100., -100., -100., -100., -100.,
       -100., -100., -100., -100., -100.])
```

In [53]:
```
x[::3]=40
x
```

Out[53]:
```
array([  40., -100., -100.,   40., -100., -100.,   40., -100., -100.,
         40., -100., -100.,   40., -100., -100.,   40., -100., -100.,
         40., -100., -100.,   40., -100., -100.,   40., -100., -100.,
         40., -100., -100.,   40., -100., -100.,   40., -100., -100.,
         40., -100., -100.,   40., -100., -100.,   40., -100., -100.,
         40., -100., -100.,   40., -100.])
```

Can you change the array in some fun way?

In [ ]:
```
```

Answers

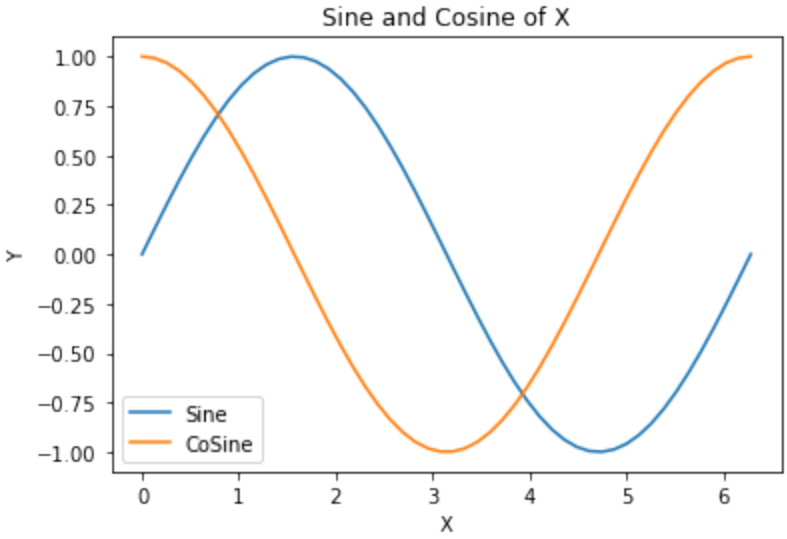
In [6]:
```
newList=np.arange(0,1000,5)
print(newList)
```

```
[  0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85
  90  95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175
 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265
 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355
 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445
 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535
 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625
 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715
 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805
 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895
 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985
 990 995]
```

In [9]:
```
x=np.linspace(0,2*np.pi)

fig,ax=plt.subplots()
ax.plot(x,np.sin(x),label='Sine') #setting the label in plot tells legend the na
ax.plot(x,np.cos(x),label='CoSine')
ax.set_xlabel('X')  #added in
ax.set_ylabel('Y')  #added in
ax.set_title('Sine and Cosine of X')
ax.legend(loc='best')
```

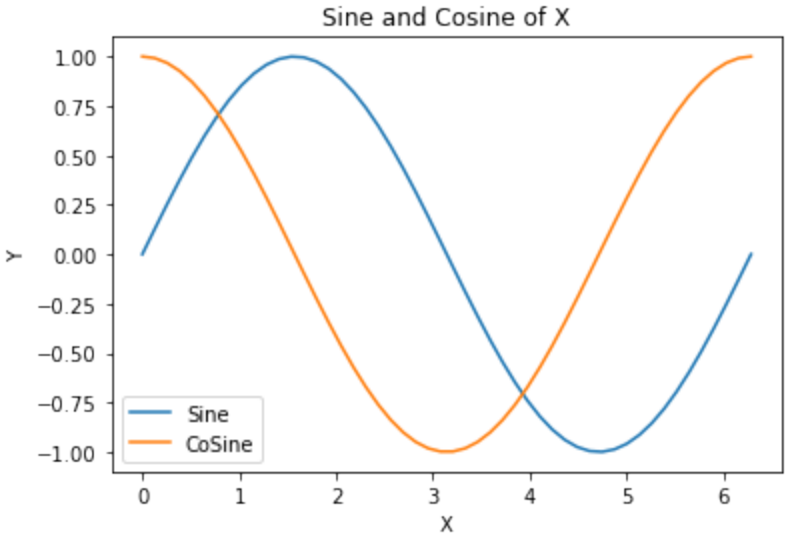
Out[9]:
```
<matplotlib.legend.Legend at 0x1941cb21940>
```



We could also have done it all right in plt.plot I find this harder to read and so it can be nice to set the list parameters. Especially as our code gets longer

In [2]:
```
fig,ax=plt.subplots()
ax.plot(np.linspace(0,2*np.pi),np.sin(np.linspace(0,2*np.pi)),label='Sine')
ax.plot(np.linspace(0,2*np.pi),np.cos(np.linspace(0,2*np.pi)),label='CoSine')
ax.set_xlabel('X')  #added in
ax.set_ylabel('Y')  #added in
ax.set_title('Sine and Cosine of X')
plt.legend(loc='best')
```

Out[2]:
```
<matplotlib.legend.Legend at 0x7fe78b29ac70>
```



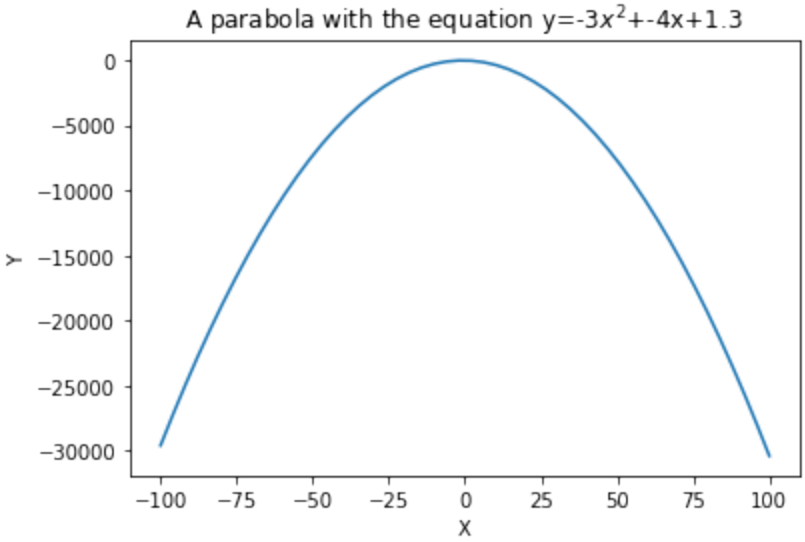
Parabola Answer

In [11]:
```
a=-3
b=-4
c=1.3
x=np.linspace(-100,100)
y=a*x**2+b*x+c

fig,ax=plt.subplots()
```

```
ax.plot(x,y)
ax.set_xlabel('X')  #added in
ax.set_ylabel('Y')  #added in
ax.set_title('A parabola with the equation y={}$x^2$+{}x+{}'.format(a,b,c))
```

Out[11]: Text(0.5,1,'A parabola with the equation y=-3x^2+-4x+1.3')



In [ ]: