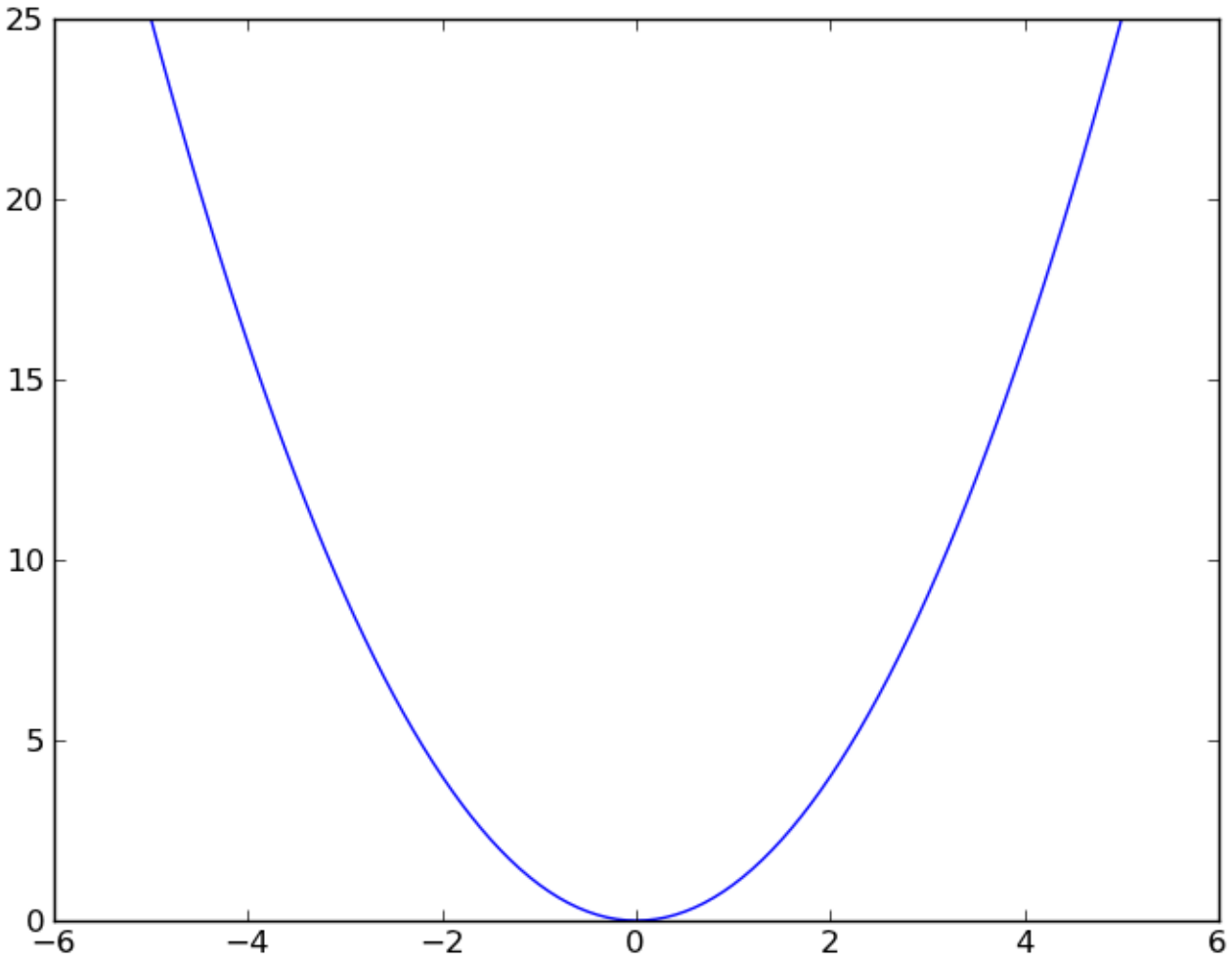
Curve Plotting

- To plot curves import the module ***matplotlib.pyplot***, use:

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

- Plotting requires that the data be stored in arrays or lists.

```
# plotX2.py
import numpy as np
import matplotlib.pyplot as plt

def plotCurve(xCoords,yCoords):
    plt.figure()     # create a figure to hold the
plot
    plt.plot(xCoords, yCoords)  # create the plot
    plt.show()       # display the plot

def main():
    xCoords = np.arange(-5., 5.1, .1)
    yCoords = xCoords**2
    plotCurve(xCoords, yCoords)

main()
```


25
20
15
10
5
0
−6
−4
−2
0
2
4
6

•To ensure that one plot does not overwrite another plot use the *figure* function.

```
figure()    # ensure plots are not overwritten
```

•To create a plot you must use the *plot* function passing it the arrays that contain the data points.

```
plot(x, y) # create the plot
```

•To display the plot you must use the *show* function.

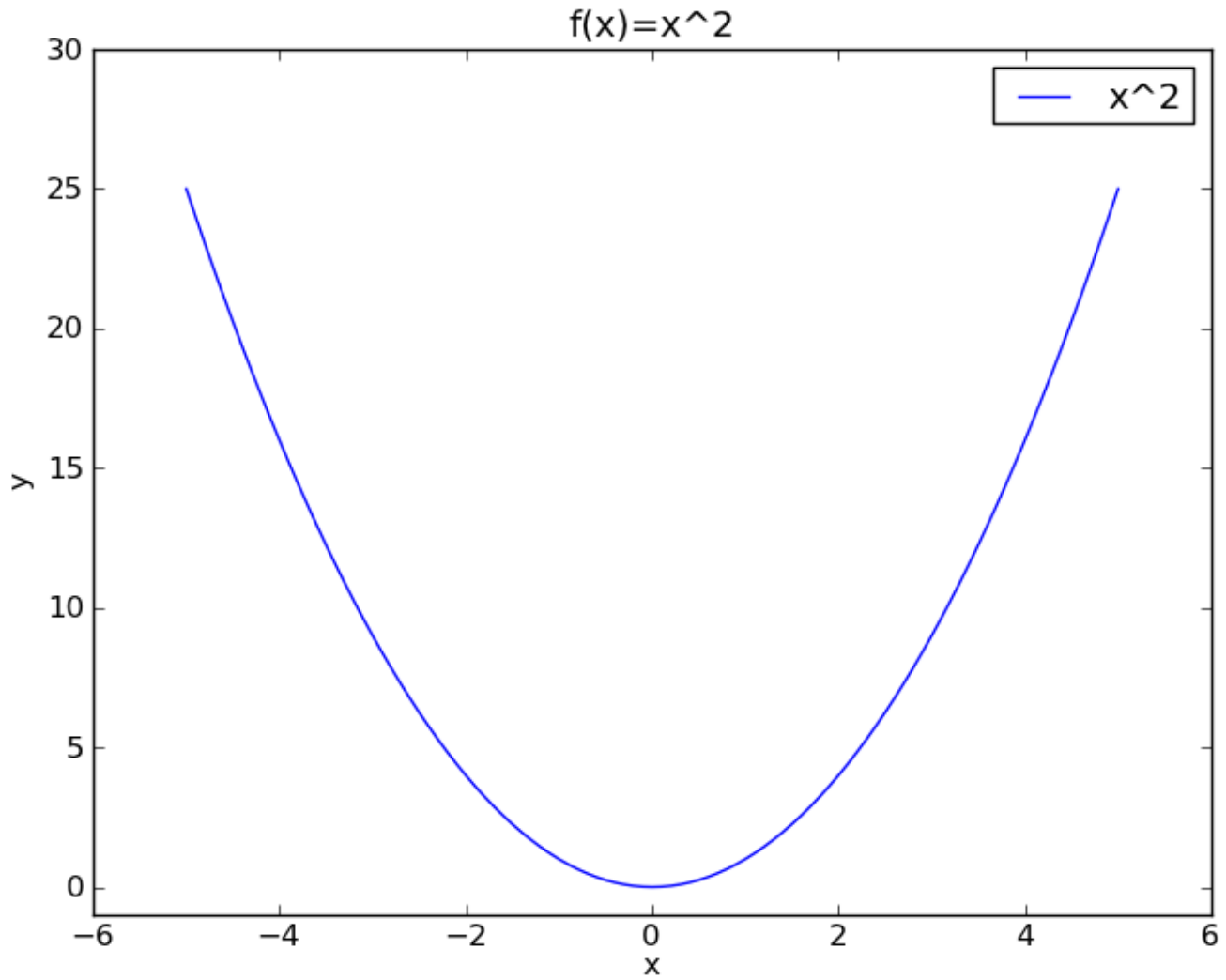
```
show()      # display the plot
```

•You can save the plots in various electronic formats in the same folder as the program using the *savefig* function.

```
# plotX2Fig.py
import numpy as np
import matplotlib.pyplot as plt

def plotCurve(x, y):
    plt.figure()
    plt.plot(x, y)  # create the plot
    plt.savefig('PlotX2.eps') # postscript
    plt.savefig('PlotX2.png') # png file
    plt.show()      # display the plot

def main():
    xCoords = np.arange(-5., 5.1, .1)
    yCoords = xCoords**2
    plotCurve(xCoords, yCoords)

main()
```

- It would be nice to decorate our plot with labels on the axes, a legend and a title.
- xlabel('x') puts the label 'x' on the x axis.
- ylabel('y') puts the label 'y' on the y axis.
- axis([-6., 6., -1, 25]) specifies the minimum and maximum values to display for the x and y axes as [xmin, xmax, ymin, ymax]. (rarely used)
- legend(['x^2']) puts the text 'x^2' in the legend
- title('f(x)=x^2') puts the text 'f(x)=x^2' in the title
- These statements must follow the ***plot*** statement.

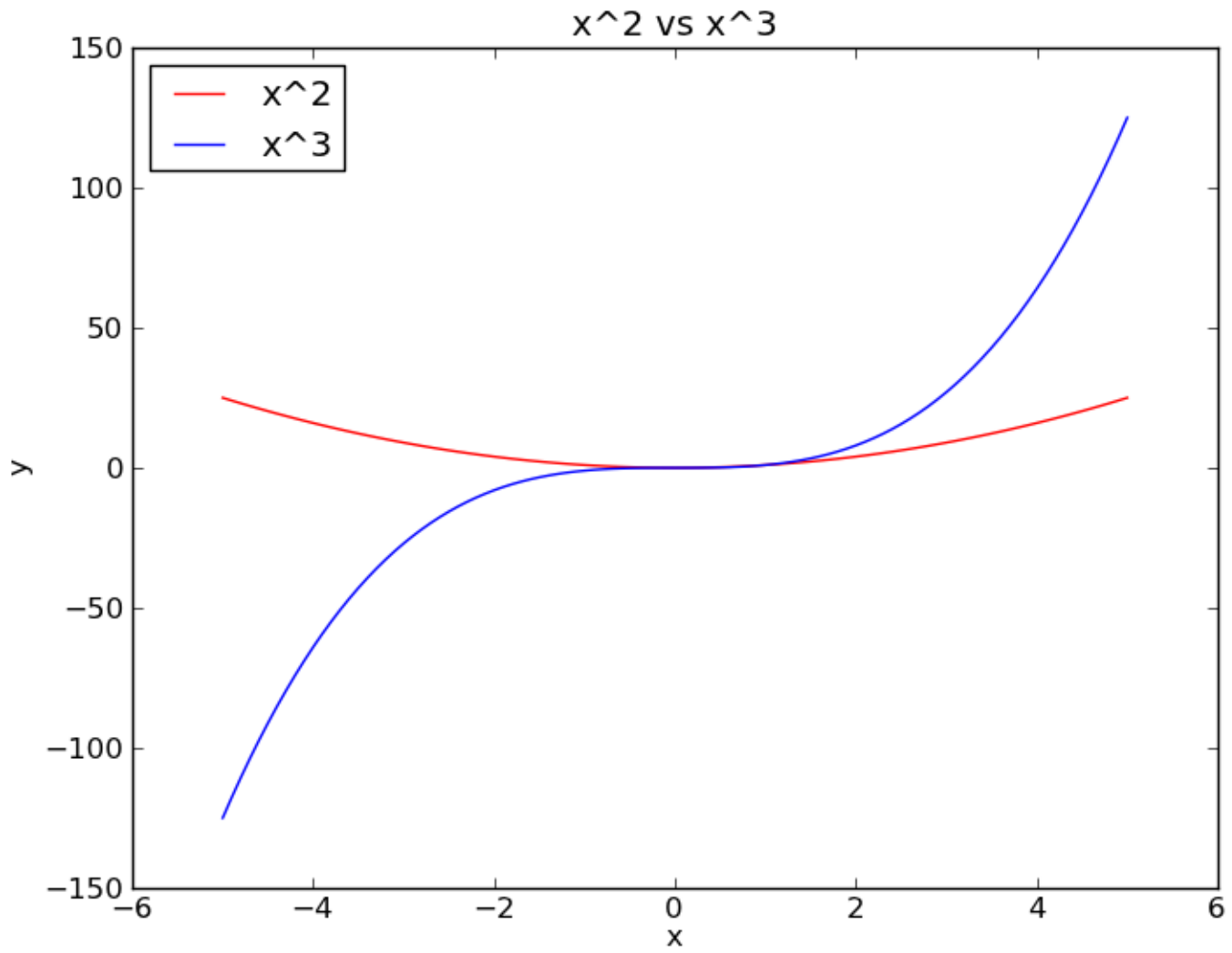
```
# plotX2Decorate.py
import numpy as np
import matplotlib.pyplot as plt

def plotCurve(x,y):
    plt.figure()
    plt.plot(x, y)  # create the plot
    plt.xlabel('x')
    plt.ylabel('y')
    plt.axis([-6., 6., -1, 30]) # rarely used
    plt.legend(['x^2'])  # legend is a list of strings
    plt.title('f(x)=x^2')
    plt.savefig('PlotX2Dec.png') # png file
    plt.show()       # display the plot

x = np.arange(-5., 5.1, .1)
y = x**2
plotCurve(x,y)
```


f(x)=x^2
x^2
30
25
20
15
10
5
0
y
−6
−4
−2
0
2
4
6
x

•Often it is necessary to draw more than one curve on the same plot.

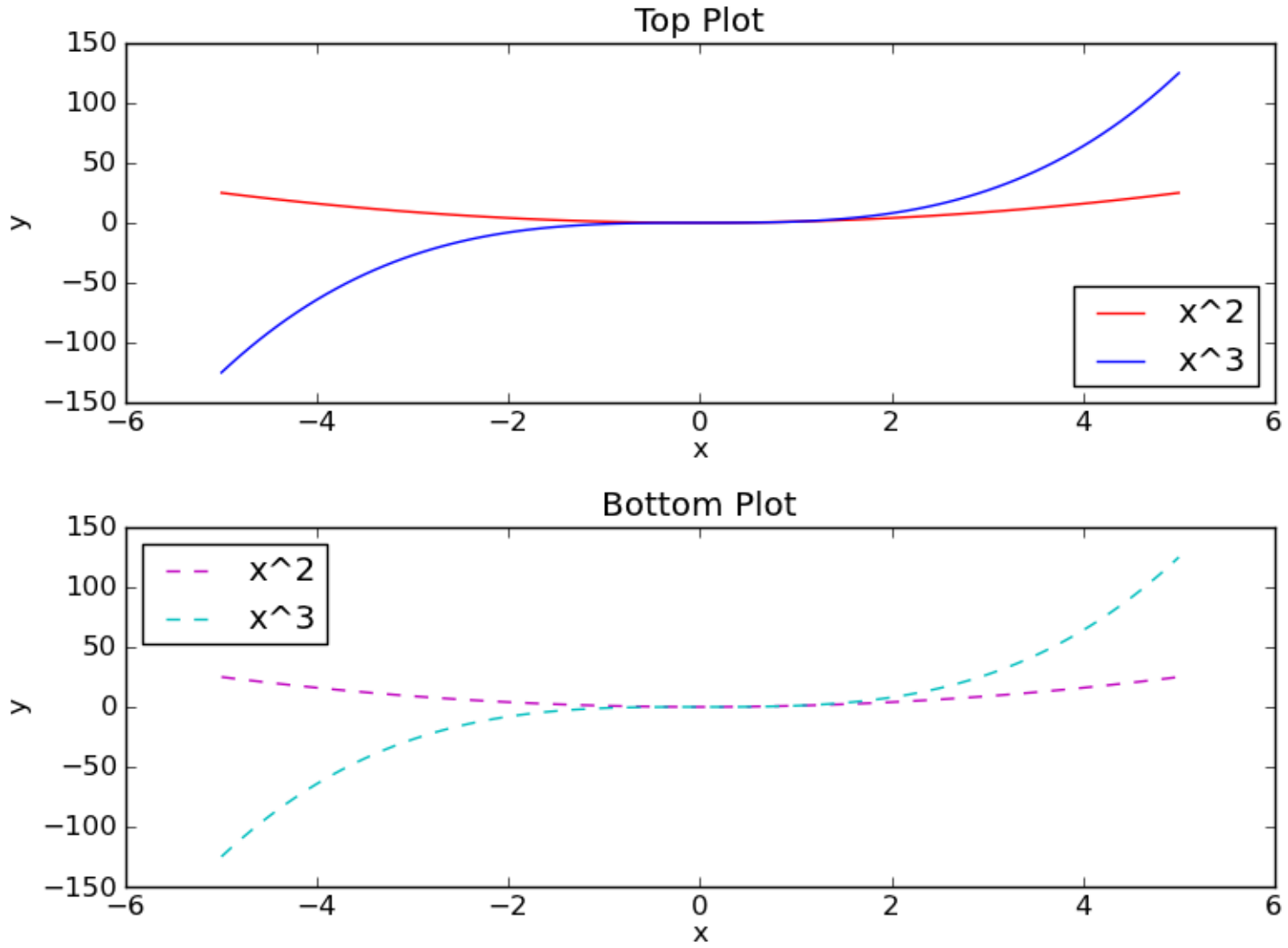
•Plot the first curve, then plot the second curve.

```
# plotX2X3.py
import numpy as np
import matplotlib.pyplot as plt
def plotCurves(x,y2,y3):
    plt.figure()
    plt.plot(x, y2, 'r-')  # red solid line
    plt.plot(x, y3, 'b-') # blue solid line
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(['x^2', 'x^3'], loc='upper left')
    plt.title('x^2 vs x^3')
    plt.savefig('PlotX2X3.png') # png file
    plt.show()       # display the plot

x = np.arange(-5., 5.1, .1)
y2 = x**2
y3 = x**3
plotCurves(x,y2,y3)
```


x^2 vs x^3
x^2
x^3
150
100
50
0
−50
−100
−150
y
−6
−4
−2
0
2
4
6
x

- Valid values for the *loc* attribute of the *legend* command are:
  - ‘best’
  - ‘upper right’
  - ‘upper left’
  - ‘lower left’
  - ‘lower right’
  - ‘center left’
  - ‘center right’
  - ‘lower center’
  - ‘upper center’
  - ‘center’

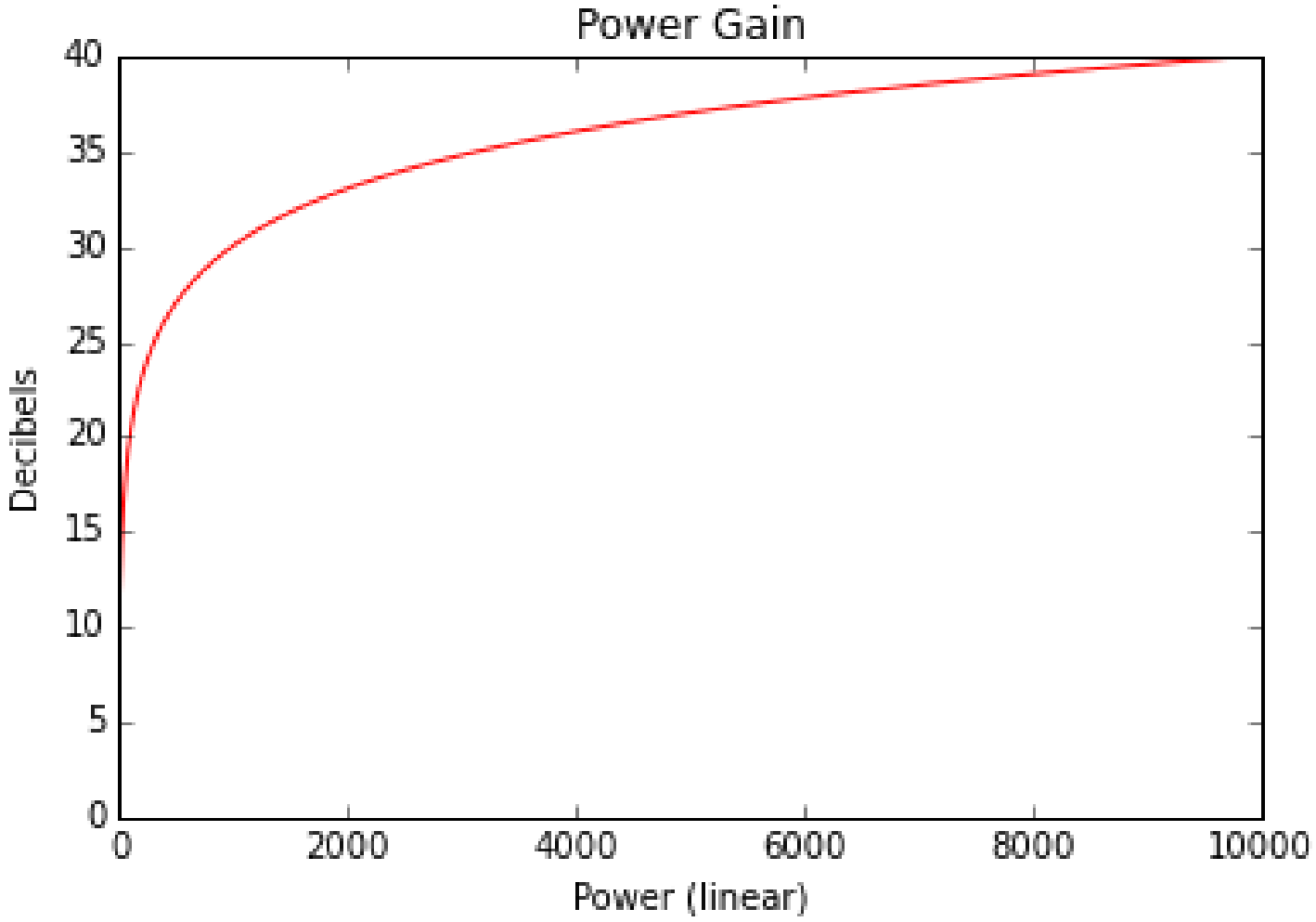
```
# plotX2X3sub.py
import numpy as np
import matplotlib.pyplot as plt
def plotSubPlots(x, y2, y3):
    plt.figure()
    plt.subplot(2, 1, 1)
    plt.plot(x, y2, 'r-', x, y3, 'b-')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(['x^2', 'x^3'], loc='lower right')
    plt.title('Top Plot')

    plt.subplot(2, 1, 2)
    plt.plot(x, y2, 'm--', x, y3, 'c--')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(['x^2', 'x^3'], loc='best')
    plt.title('Bottom Plot')
    plt.tight_layout()
    plt.savefig('PlotX2X3.png') # png file
    plt.show()      # display the plot

x = np.arange(-5., 5.1, .1)
y2 = x**2
y3 = x**3
plotSubPlots(x, y2, y3)
```

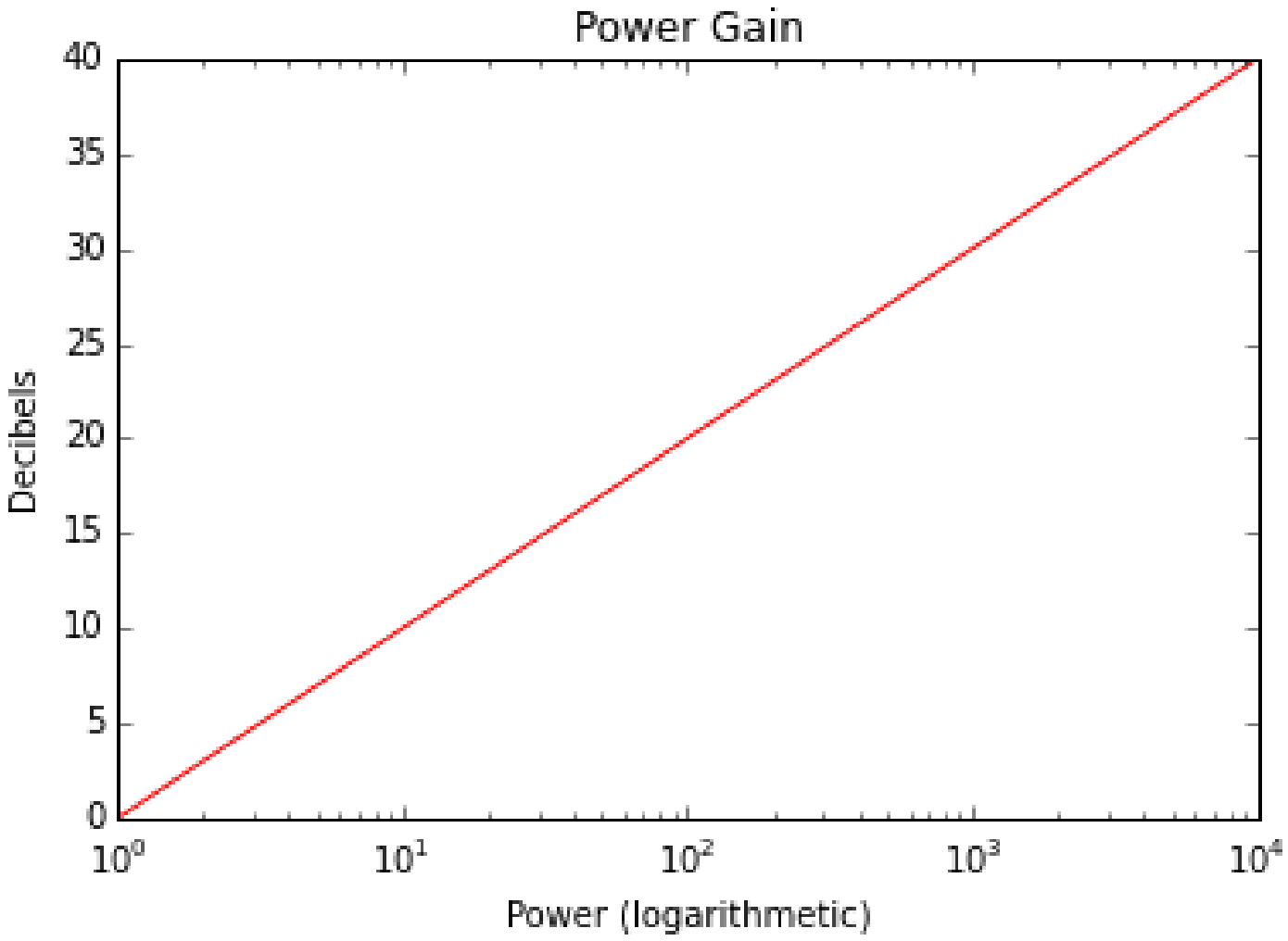

Top Plot
150
100
50
0
−50
−100
−150
y
−6
−4
−2
0
2
4
6
x
x^2
x^3
Bottom Plot
150
100
50
0
−50
−100
−150
y
−6
−4
−2
0
2
4
6
x
x^2
x^3

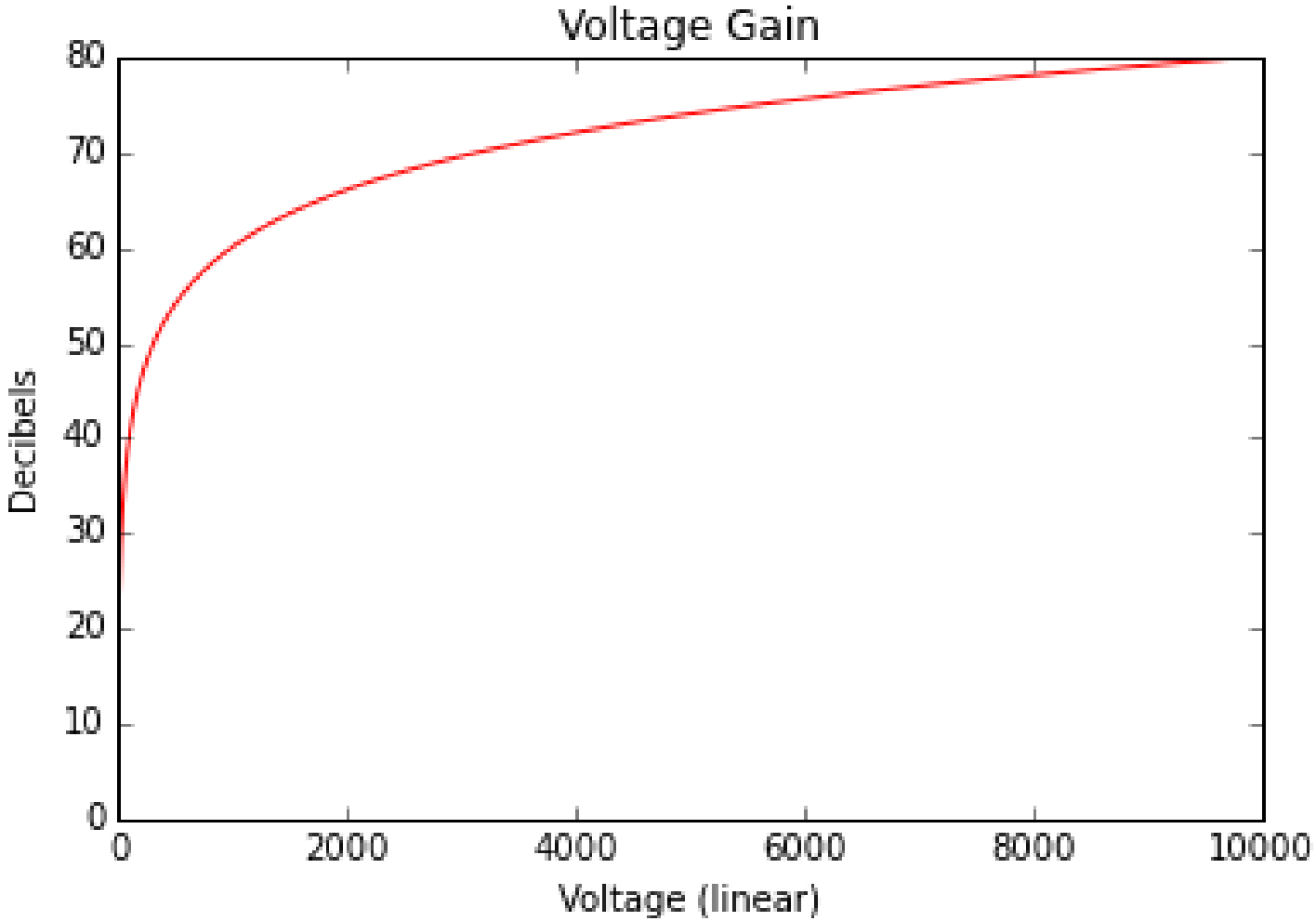
```
# decibels.py
from time import ctime
import numpy as np
import matplotlib.pyplot as plt

def displayTerminationMessage():
    print('''
Programmed by Stew Dent.
Date: %s
End of Processing.''' % ctime())

def powerGain(maxPower, intervals):
    powers = np.linspace(1, maxPower, intervals+1)
    decibels = 20 * np.log10(powers)
    return powers, decibels

def voltageGain(maxVoltage, intervals):
    voltages = np.linspace(1, maxVoltage, intervals+1)
    decibels = 20 * np.log10(voltages)
    return voltages, decibels
```

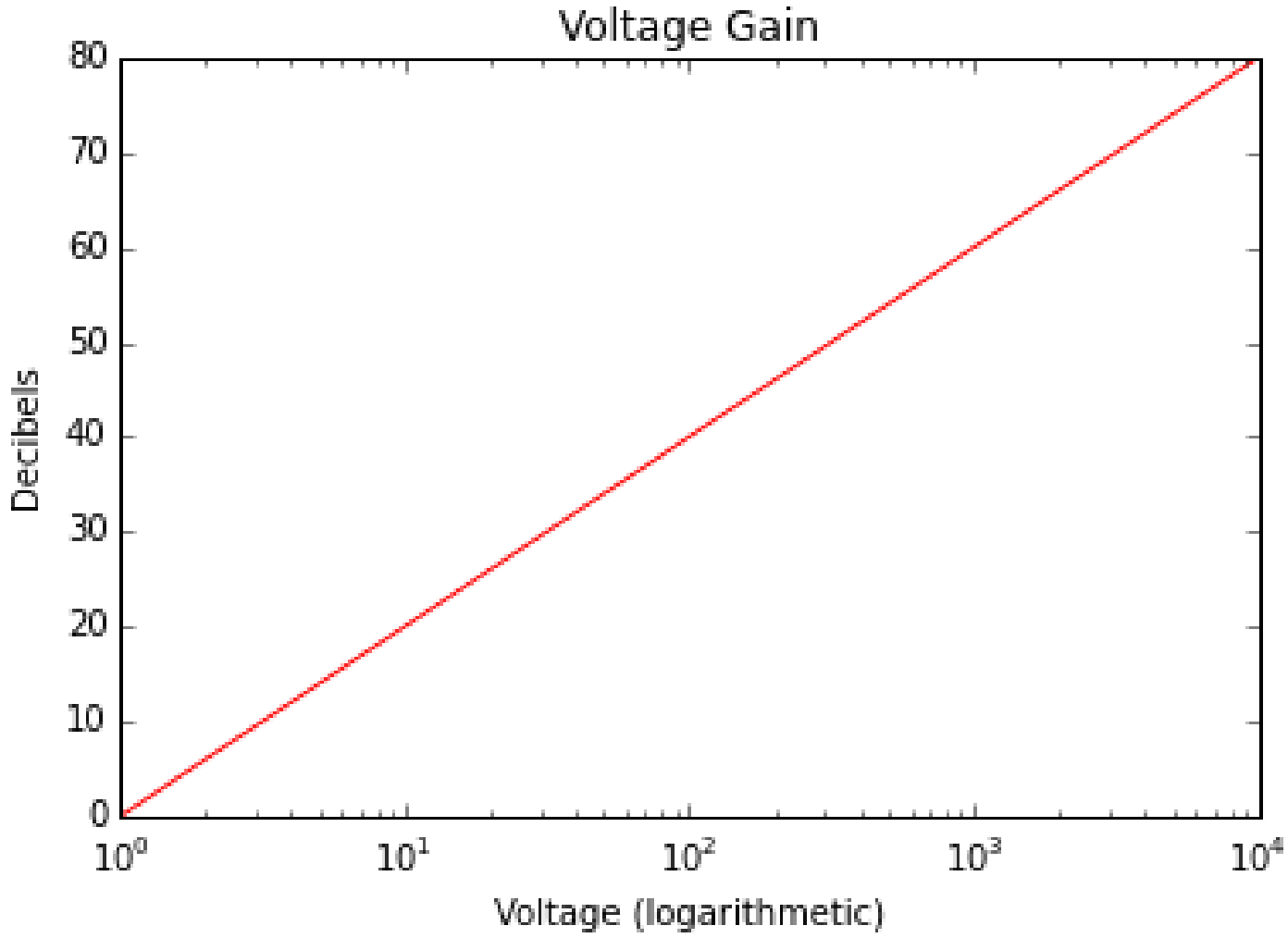
```
def plotGain(units, gain, heading):
    plt.figure()
    plt.plot(units,gain,'r-')
    plt.xlabel(heading + ' (linear)')
    plt.ylabel('Decibels')
    plt.title(heading + ' Gain')
    plt.savefig(heading + 'Gain1.png')
    plt.show()

    plt.figure()
    plt.semilogx(units,gain,'r-')
    plt.xlabel(heading + ' (logarithmetic)')
    plt.ylabel('Decibels')
    plt.title(heading + ' Gain')
    plt.savefig(heading + 'Gain2.png')
    plt.show()
```

```
def main():
    maxPower = 10000
    maxVoltage = 10000
    intervals = 100000
    powers, powerGainDecibels =
            powerGain(maxPower, intervals)
    plotGain(powers, powerGainDecibels, 'Power')
    voltages, voltageGainDecibels =
              voltageGain(maxVoltage, intervals)
    plotGain(voltages, voltageGainDecibels,
            'Voltage')
    displayTerminationMessage()

main()
```


Power Gain
Decibels
40
35
30
25
20
15
10
5
0
0
2000
4000
6000
8000
10000
Power (linear)

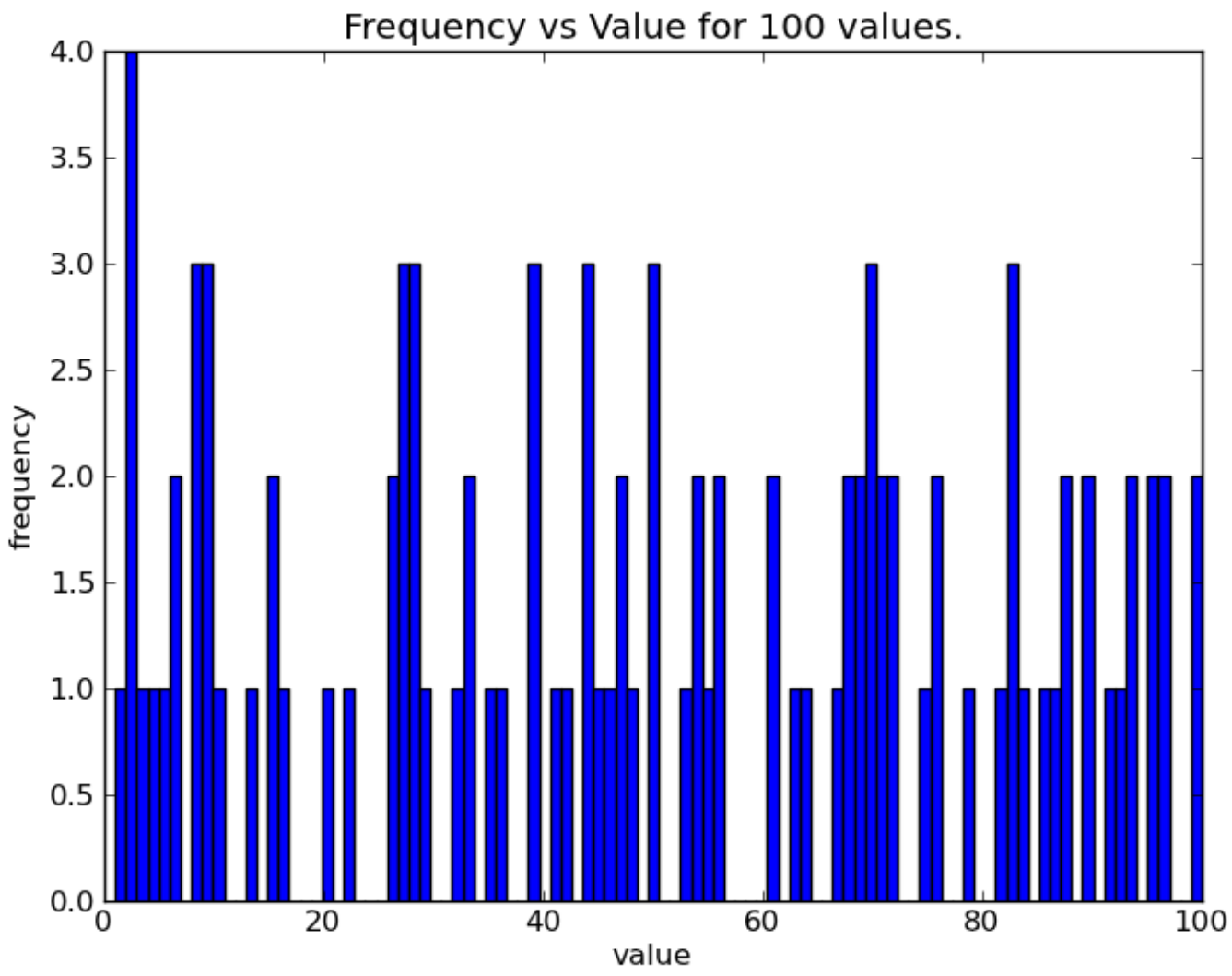

Power Gain
Decibels
40
35
30
25
20
15
10
5
0
10^0
10^1
10^2
10^3
10^4
Power (logarithmetic)

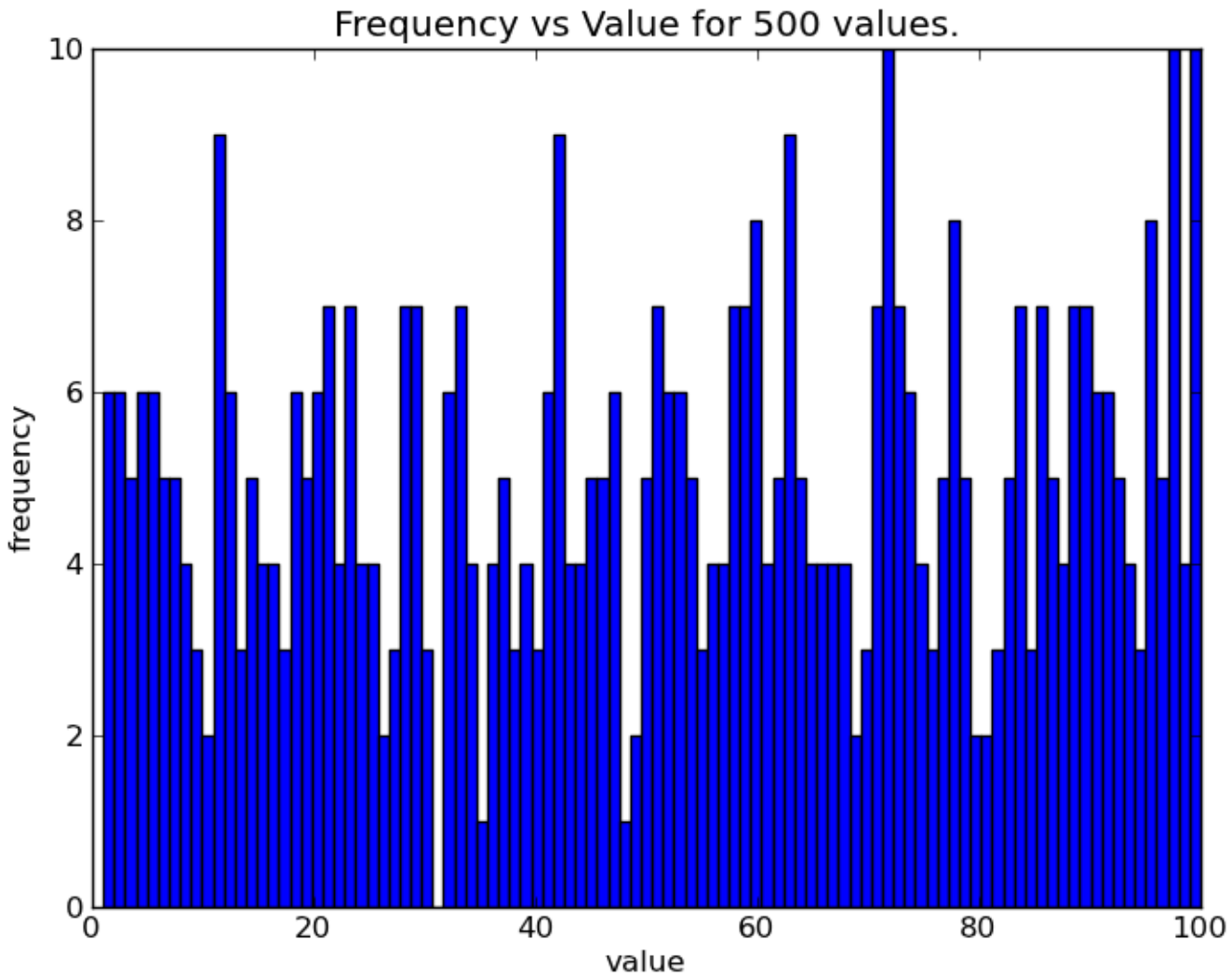

Voltage Gain
Decibels
Voltage (linear)
0
10
20
30
40
50
60
70
80
0
2000
4000
6000
8000
10000


Voltage Gain
Decibels
80
70
60
50
40
30
20
10
0
10^0
10^1
10^2
10^3
10^4
Voltage (logarithmetic)

•Plot a histogram of random numbers. The x value is the value of the random number and the y value is the number of times a random number occurs.

```
# plotHistogram.py
import numpy as np
import matplotlib.pyplot as plt

count = 100
while True:
    rands = np.random.randint(1, 101, count)
    print('%d random numbers in histogram.' % count)
    myTitle = 'Frequency vs Value for ' + str(count) + ' values.'
    plt.hist(rands, bins=100)
    plt.xlabel('value')
    plt.ylabel('frequency')
    plt.title(myTitle)
    plt.savefig('PlotHistogram.png')
    plt.show()

    data = input(
        "Press return to continue (any key to quit): ").strip()
    if data != '':
        break
    count += 100
```


Frequency vs Value for 100 values.
frequency
value
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100


Frequency vs Value for 500 values.
frequency
value
0
2
4
6
8
10
0
20
40
60
80
100

•Modify the trajectory program to plot a trajectory.

```
# trajectoryPlot.py
from time import ctime
from math import pi, cos, tan
from numpy.lib.scimath import sqrt
import numpy as np
import matplotlib.pyplot as plt

def trajectory(v0, y0, theta, intervals):
    theta = theta * pi / 180  # convert to radians
    G = 9.81                  # m/s**2
    a = -G / (2 * v0**2 * cos(theta)**2)
    b = tan(theta)
    c = y0
    distance = (-b - sqrt(b*b - 4*a*c)) / (2 * a)
    #root2 = (-b + sqrt(b*b - 4*a*c)) / (2 * a)
    xValues = np.linspace(0., distance, intervals+1)
    yValues = (a * xValues + b) * xValues + c
    return (xValues, yValues)
```

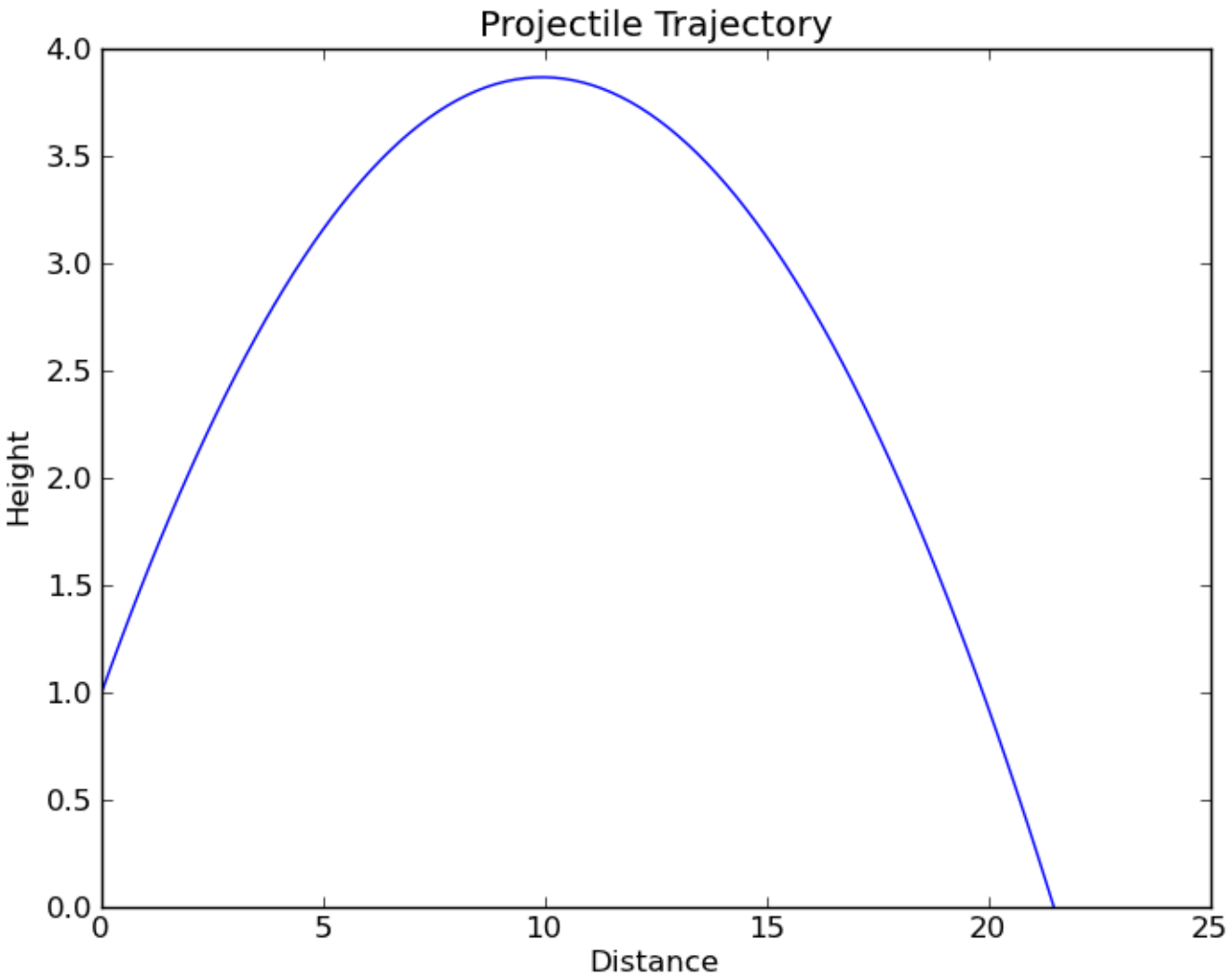
```
def plotTrajectory(xValues, yValues):
    plt.figure()
    plt.plot(xValues, yValues, 'b-')
    plt.xlabel('Distance')
    plt.ylabel('Height')
    plt.title('Projectile Trajectory')
    plt.savefig('trajectory.png')
    plt.show()

print('\n' + '-'* 80)
intervals = 100

while True:
  try:
    v0 = float(input("Enter the initial velocity: "))
    y0 = float(input('Enter the initial height: '))
    theta = float(input('Enter the launch angle: '))
    break
  except:
    print('Invalid data, try again!')
```

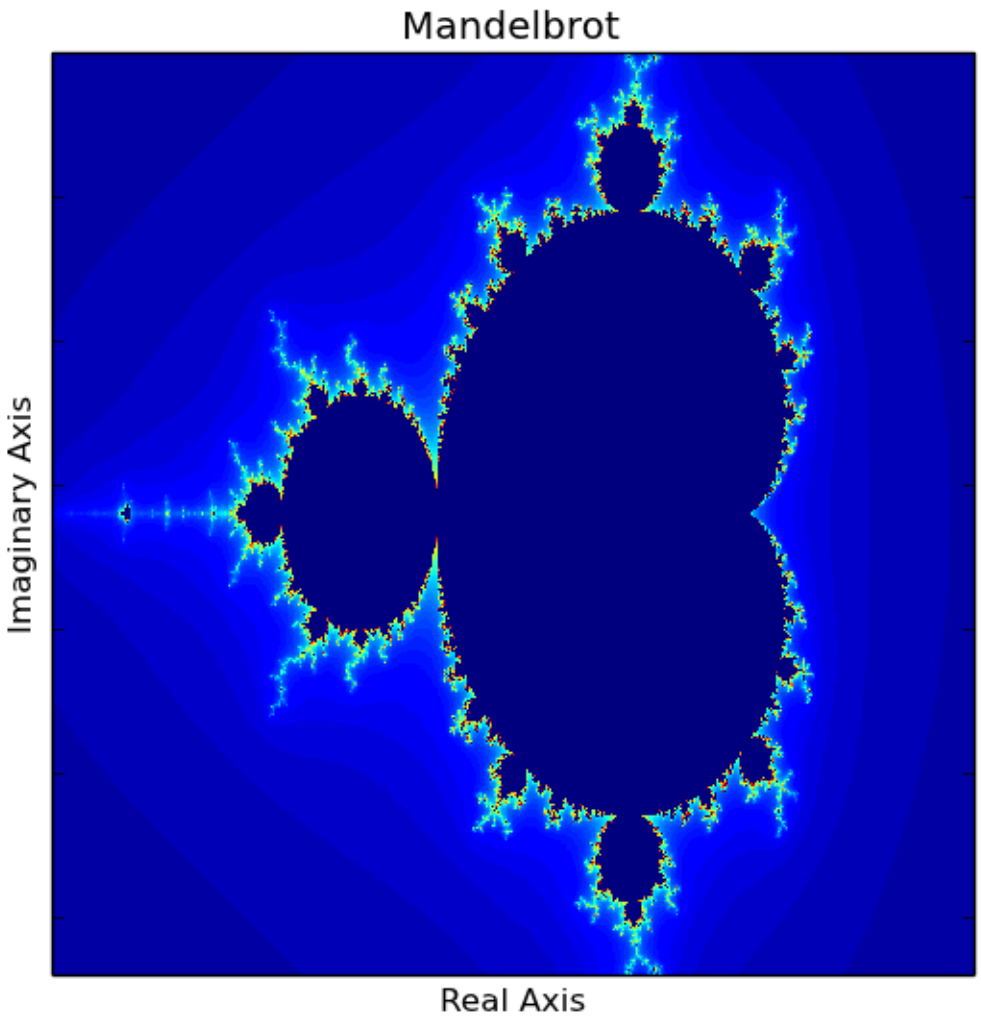
```
# display the values of the variables
print("""
v0        = %.1f m/s
y0        = %.1f m
theta     = %d degrees
intervals = %d
""" % (v0, y0, theta, intervals))

xValues, yValues = trajectory(v0, y0, theta, \
                              intervals)
plotTrajectory(xValues, yValues)

print("""
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())
```

•The output from the program is:

```
-----------------------------------------

Enter the initial velocity: 15
Enter the initial height: 1
Enter the launch angle: 30

v0        = 15.0 m/s
y0        = 1.0 m
theta     = 30 degrees
intervals = 100

Programmed by Stew Dent.
Date: Thu Jun 28 10:22:43 2018.
End of processing.
```

•The plot is shown on the next slide.


Projectile Trajectory
Height
Distance
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25


Mandelbrot
Imaginary Axis
Real Axis

```
def mandelbrot(realPart, imagPart, imgSize, iterations, xOffset):
    img = np.zeros((imgSize,imgSize),int)
    deltaX = realPart / img.shape[1] * 2.
    deltaY = imagPart / img.shape[0] * 2.

    INFINITY = 16.0

    y = -imagPart
    # row number is along the y (imaginary) axis
    for row in range(img.shape[0]):
        x = -realPart - xOffset
        # column number is along the x (real) axis
        for column in range(img.shape[1]):
            # do the work
            a, b = x, y
            n = 0
            while n < iterations:
                aa, bb = a*a, b*b
                a, b = aa - bb + x, 2.0 * a * b + y
                if aa + bb > INFINITY:
                    break
                n += 1
            if n < iterations:
                img[row, column] = n
            x += deltaX
        y += deltaY
    drawMandelbrot(img)
```

```
def drawMandelbrot(img):
    # img - the 2-D array holding the image to
    #       display
    plt.figure()
    plt.imshow(img, interpolation='nearest')
    plt.title('Mandelbrot')
    plt.xlabel('Real Axis')
    plt.ylabel('Imaginary Axis')
    plt.tick_params(axis='both', which='both',
        bottom='off', top='off',
        labelbottom='off', labelleft='off')
    plt.savefig('mandelbrot.png')
    plt.show()
```

```
def main():
    # realPart - real part of the intial complex number
    # imagPart - imaginary part of the initial complex number
    # imgSize - the size of the square grid for the image
    # iterations - the number of iterations before giving up
    imgSize = int(input("Enter the size of the grid: "))
    realPart = float(input('Enter the maximum real part:'))
    imagPart = float(input('Enter the maximum imaginary
               part: '))
    xOffset = float(input('Enter the x offet: '))
    iterations = int(input('Enter the number of
                 iterations: '))
    mandelbrot(realPart, imagPart, imgSize, iterations,
               xOffset)
    displayTerminationMessage()
```

More Information

- More information is available at http://matplotlib.sourceforge.net/contents.html
- You can view the manual online or download the PDF version.

Sets in Python

In Python a sequence of values / variables separated by commas enclosed in **braces** forms a set.

```
In [1]: a = {'a', 'b'}
In [2]: type(a)
Out[3]: set
In [4]: b = {1, 2, 3}
In [5]: type(b)
Out[6]: set
In [7]: w = 1.25; x = -3.5; y = 99.9
In [8]: z = {w, x, y}
In [9]: type(z)
Out[10]: set
In [11]: z
Out[11]: {-3.5, 1.25, 99.9}
```

An empty set can be created as follows.

```
In [1]: s5 = set()  # s5 = {} creates a dictionary
In [2]: s5
Out[2]: set()
```

Use the ***add*** function to add an element to a set.

```
In [3]: s5.add(1.27)
In [4]: s5
Out[4]: {1.27}
In [5]: s5.add(-3.23)
In [6]: s5
Out[6]: {-3.23, 1.27}
```

Use the ***discard*** function to remove/discard an element from a set.

```
In [7]: s5.discard(1.27)
In [8]: s5
Out[8]: {-3.23}
```

Python supports standard set operations such as ***union*** and ***intersection***.

```
In [1]: s1 = {1, 2, 3}
In [2]: s2 = {2, 'a', 'b', 'c', 'd'}
In [3]: s3 = {1, 3, 5, 7}
In [4]: s4 = {2, 4, 6, 8}
In [5]: u1 = s1.union(s2)   # u1 = s1 | s2
In [6]: u1    # no duplicates in a set
Out[6]: {1, 2, 3, 'a', 'b', 'c', 'd'}
In [7]: i1 = u1.intersection(s3) # i1 = u1 & s3
In [8]: i1
Out[8]: {1, 3}
In [9]: i2 = u1.intersection(s4)
In [10]: i2
Out[10]: {2}
```

```
In [1]: s3 = {1, 3, 5, 7}
In [2]: s4 = {2, 4, 6, 8}

In [3]: i3 = s3 & s4   # & does intersection
In [4]: i3
Out[4]: set()
```

As s3 and s4 have no elements in common the intersection is the empty set. The sets s3 and s4 are disjoint.

```
In [5]: s3.isdisjoint(s4)
Out[5]: True
In [6]: u1.isdisjoint(s2)
Out[6]: False
```

To determine if a particular value is in a set use the ***in*** operator.

```
In [1]: u1
Out[1]: {1, 2, 3, 'a', 'b', 'c', 'd'}
In [2]: 'a' in u1
Out[2]: True
In [3]: 10 in u1
Out[3]: False
```

Use the ***issubset*** function or the **<=** operator to determine if one set is a subset of another set.

```
In [4]: {1, 3, 5, 'b', 'd'}.issubset(u1)
Out[4]: False
In [5]: s1.issubset(u1)
Out[5]: True
In [6]: {1, 3 , 'b', 'd'} <= u1
Out[6]: True
```

Use the ***issuperset*** function or the **>=** operator to determine if one set is a superset of another set.

```
In [1]: u1
Out[1]: {1, 2, 3, 'a', 'b', 'c', 'd'}
In [2]: u1 >= {2, 'a', 'c'}
Out[2]: True
In [3]: u1.issuperset({2, 4, 'a', 'c'})
Out[3]: False
```

To determine the number of elements in a set use the ***len*** function.

```
In [4]: len(u1)
Out[4]: 7
```

•Elements in a set are unique! (uniqueNumbers.py)

```
def getInteger(prompt):
    while True:
        number = input(prompt).strip()
        if number != '':
            try:
                number = eval(number, {}, {})
            except:
                print(number, 'is not valid!')
            else:
                if type(number) is int:
                    break
                else:
                    print(number, 'is not an integer!')
        else:
            break
    return number
```

```
def displaySequence(heading, theSequence):
    print('\n%s' % heading)
    for element in theSequence:
        print(element)
    print('There are %d elements in the sequence.' %
          len(theSequence))

def displayTerminationMessage():
    print('''
Programmed by Stew Dent.
Date: %s
End of processing.''' % ctime())
```

```
def main():
    listOfNumbers = []
    setOfNumbers = set()
    number = getInteger("Enter an integer: ")
    while number != '':
        listOfNumbers.append(number)
        setOfNumbers.add(number)
        number = getInteger("Enter an integer: ")
    displaySequence(
         'The elements in the list of numbers are:',
          listOfNumbers)
    displaySequence(
         'The elements in the set of numbers are:',
         setOfNumbers)
    displayTerminationMessage()

main()
```

Suppose we wish to convert the words in a sentence into a set of words.

```
In [1]: sentence = 'Hello there my good friend'
In [2]: wordList = sentence.strip().split()
In [3]: wordList
Out[4]: ['Hello', 'there', 'my', 'good', 'friend']
In [5]: wordSet = set(wordList) # convert to set
In [6]: wordSet
Out[6]: {'Hello', 'friend', 'good', 'my', 'there'}
```

Notice that testing for a word in the set of words is case sensitive.

```
In [7]: 'hello' in wordSet
Out[7]: False
In [8]: 'Hello' in wordSet
Out[8]: True
```

Suppose we wish to know the number of words in two different sentences that are the same. (sentences.py)

```
def sentenceToSet(sentence):
return set(sentence.strip().split())

def displaySet(heading, theSet):
print(heading)
for word in theSet:
            print(word)
print('The number of common words is %d' % (len(theSet)))

def main():
print('Each sentence must consist of words in lowercase')
print(' only, no punctuation!')
sentence1 = input("Enter the first sentence:\n")
wordSet1 = sentenceToSet(sentence1)
sentence2 = input("Enter the second sentence:\n")
wordSet2 = sentenceToSet(sentence2)
commonWords = wordSet1.intersection(wordSet2)
displaySet('\nThe set of words common to the sentences is:',
               commonWords)
```

Dictionaries in Python

Each element in a dictionary consists of a **key** and an associated data value of any type.

For example consider a dictionary that contains a city name as the **key** and its current temperature as the data.

```
In [1]: temps = {'Winnipeg' : -30.7,
         'Victoria' : 3.1, 'Halifax' : -6.5}
In [2]: temps
Out[2]: {'Halifax': -6.5, 'Victoria': 3.1,
          'Winnipeg': -30.7}
In [3]: temps['Victoria']
Out[3]: 3.1
```

A for loop can be used to get the key values from the dictionary.

The key value can then be used to get the data value associated with the key.

```
In [1]: for temp in temps:
   ...:     print(temp, temps[temp])
   ...:
Halifax -6.5
Winnipeg -30.7
Victoria 3.1
```

Use the ***len*** function to determine the number of key-value pairs in the dictionary.

```
In [2]: len(temps)
Out[2]: 3
```

Create an empty dictionary and add values to it.

```
In [1]: dd = {}
In [2]: dd['hammer'] = 15.95
In [3]: dd['saw'] = 29.99
In [4]: dd['tape'] = 3.50
In [5]: dd
Out[5]: {'hammer': 15.95, 'saw': 29.99,
         'tape': 3.5}
In [6]: dd['saw'] = 39.99 # change price of saw
In [7]: dd
Out[7]: {'hammer': 15.95, 'saw': 39.99,
         'tape': 3.5}
```

Use the ***del*** function to delete a key-value pair from a dictionary.

```
In [8]: del dd['tape']
In [9]: dd
Out[9]: {'hammer': 15.95, 'saw': 39.99}
```

```
In [1]: capitals = {'Manitoba' : 'Winnipeg',
'Saskatchewan' : 'Regina', 'Alberta' : 'Edmonton',
'British Columbia' : 'Victoria'}

In [2]: capitals
Out[2]:
{'Alberta': 'Edmonton',
 'British Columbia': 'Victoria',
 'Manitoba': 'Winnipeg',
 'Saskatchewan': 'Regina'}

In [3]: list(capitals.keys())
Out[3]: ['Alberta', 'Manitoba', 'British Columbia',
'Saskatchewan']

In [4]: list(capitals.values())
Out[4]: ['Edmonton', 'Winnipeg', 'Victoria',
'Regina']
```

```
In [1]: capitals = {'Manitoba' : 'Winnipeg',
'Saskatchewan' : 'Regina', 'Alberta' : 'Edmonton',
'British Columbia' : 'Victoria'}

In [2]: list(capitals.items())
Out[7]:
[('Manitoba', 'Winnipeg'),
 ('Saskatchewan', 'Regina'),
 ('Alberta', 'Edmonton'),
 ('British Columbia', 'Victoria')]

In [3]: for key, value in capitals.items():
    print(key, value)

Manitoba Winnipeg
Saskatchewan Regina
Alberta Edmonton
British Columbia Victoria
```

•Use a dictionary to keep a count of each integer value entered into a program by the user. (frequencies.py)

```
from time import ctime

def displayTerminationMessage():
print('''
Programmed by Stew Dent.
Date: %s
End of processing.''' % ctime())

def displayFrequencies(frequencies):
print('\n%10s %10s' % ('Number', 'Frequency'))
for number in frequencies:
      print('%10d %10d' % (number, frequencies[number]))

def displaySortedFrequencies(frequencies):
sortedKeys = sorted(frequencies.keys())
print('\n%10s %10s' % ('Number', 'Frequency'))
for key in sortedKeys :
      print('%10d %10d' % (key, frequencies[key]))
```

```
def main():
    frequencies = {}

    data=input("Enter an integer: ").strip()
    while data:
        number = int(data)
        if number in frequencies:
            frequencies[number] += 1
        else:
            frequencies[number] = 1
        data=input("Enter an integer: ").strip()
    displayFrequencies(frequencies)
    displaySortedFrequencies(frequencies)
    displayTerminationMessage()

main()
```

- How would this be done using parallel lists? (index.py)

```
# index.py
from time import ctime

def displayTerminationMessage():
    print('''
Programmed by Stew Dent.
Date: %s
End of processing.''' % ctime())

def displayLists(numbers, counts):
    print('\n%10s %10s' % ('Number', 'Frequency'))
    for number, count in zip(numbers, counts):
        print('%10d %10d' % (number, count))

def displaySortedLists(numbers, counts):
    sortedNumbers = sorted(numbers)
    print('\n%10s %10s' % ('Number', 'Frequency'))
    for number in sortedNumbers :
        position = numbers.index(number)
        print('%10d %10d' % (number, counts[position]))
```

```
def main():
    numbers = []
    counts = []

    data=input("Enter an integer: ").strip()
    while data:
        number = int(data)
        if number not in numbers:
            numbers.append(number)
            counts.append(1)
        else:
            position = numbers.index(number)
            counts[position] += 1
        data=input("Enter an integer: ").strip()
    displayLists(numbers, counts)
    displaySortedLists(numbers, counts)
    displayTerminationMessage()

main()
```