# Solutions - Chapter 15

## 15-1: Cubes

A number raised to the third power is a cube. Plot the first five cubic numbers, and then plot the first 5000 cubic numbers.

Plotting 5 cubes:

from matplotlib import pyplot **as** plt

*# Define data.*

x\_values **=** [1, 2, 3, 4, 5]

cubes **=** [1, 8, 27, 64, 125]

*# Make plot.*

plt**.**scatter(x\_values, cubes, edgecolor**=**'none', s**=**40)

*# Customize plot.*

plt**.**title("Cubes", fontsize**=**24)

plt**.**xlabel('Value', fontsize**=**14)

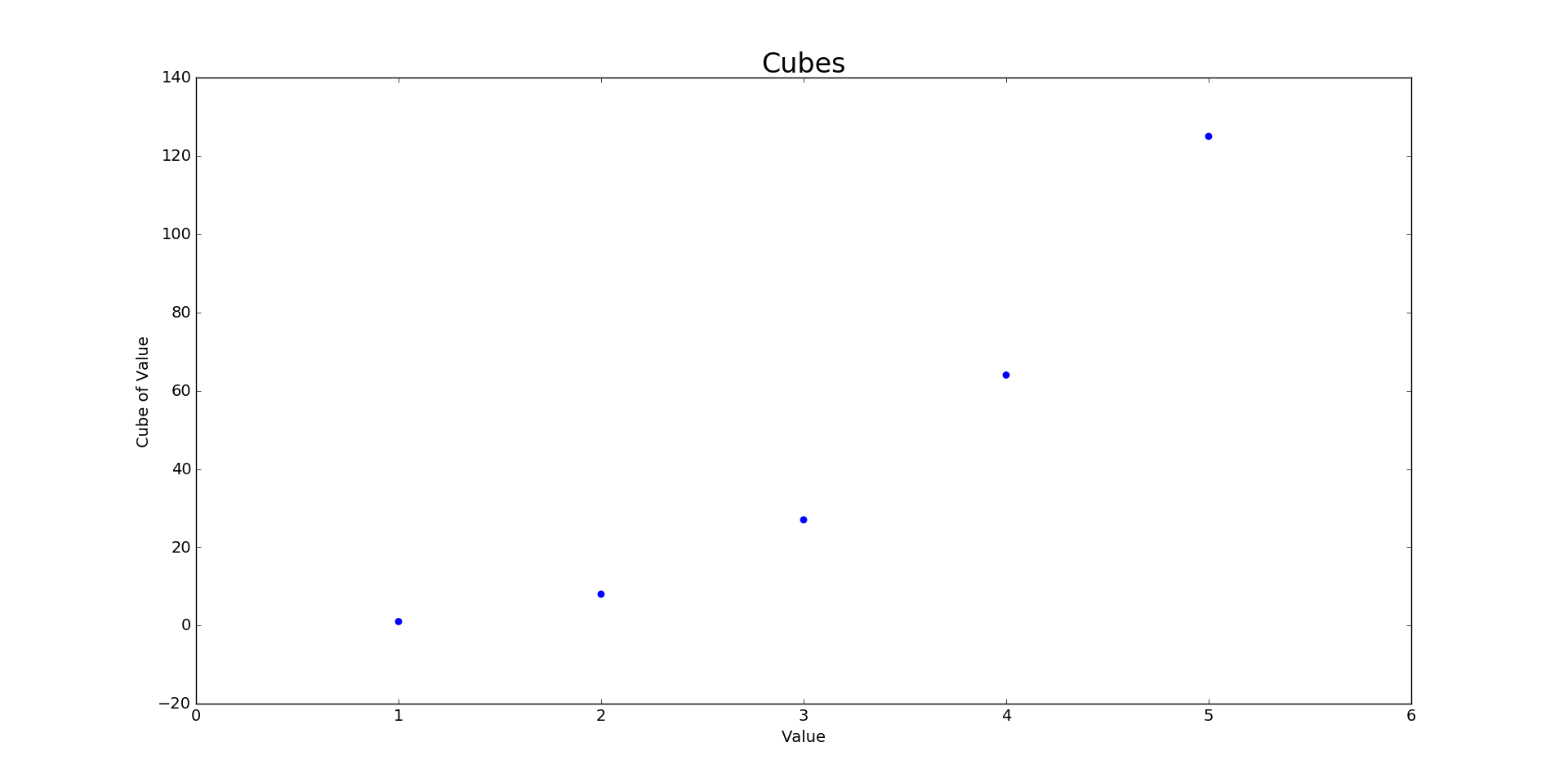
plt**.**ylabel('Cube of Value', fontsize**=**14)

plt**.**tick\_params(axis**=**'both', labelsize**=**14)

*# Show plot.*

plt**.**show()

Output:



Plotting 5000 cubes:

from matplotlib import pyplot **as** plt

*# Define data.*

x\_values **=** list(range(5001))

cubes **=** [x**\*\***3 **for** x **in** x\_values]

*# Make plot.*

plt**.**scatter(x\_values, cubes, edgecolor**=**'none', s**=**40)

*# Customize plot.*

plt**.**title("Cubes", fontsize**=**24)

plt**.**xlabel('Value', fontsize**=**14)

plt**.**ylabel('Cube of Value', fontsize**=**14)

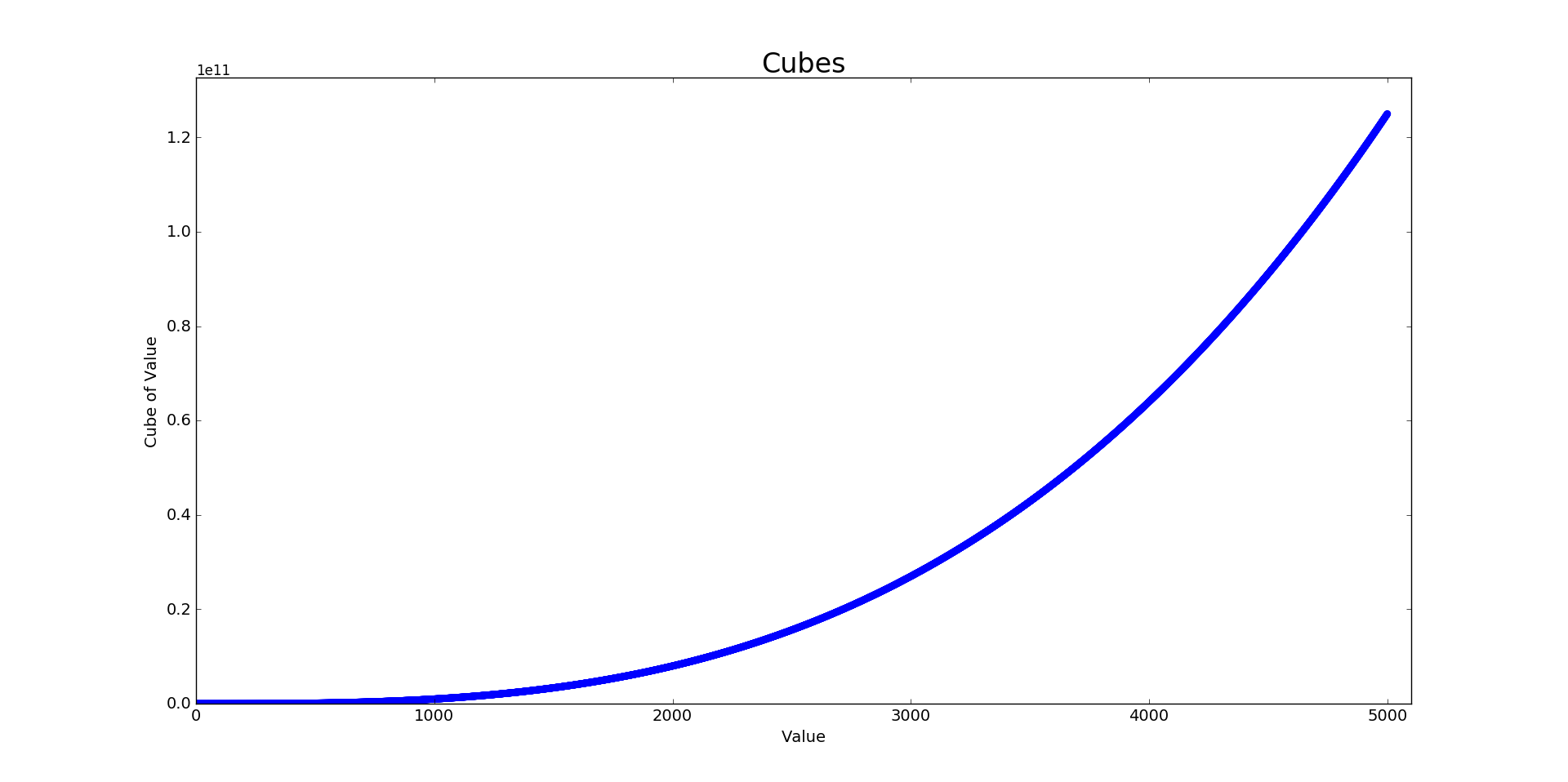
plt**.**tick\_params(axis**=**'both', labelsize**=**14)

plt**.**axis([0, 5100, 0, 5100**\*\***3])

*# Show plot.*

plt**.**show()

Output:



## 15-2: Colored Cubes

Apply a colormap to your cubes plot.

from matplotlib import pyplot **as** plt

*# Define data.*

x\_values **=** list(range(5001))

cubes **=** [x**\*\***3 **for** x **in** x\_values]

*# Make plot.*

plt**.**scatter(x\_values, cubes, edgecolor**=**'none', c**=**cubes, cmap**=**plt**.**cm**.**BuGn, s**=**40)

*# Customize plot.*

plt**.**title("Cubes", fontsize**=**24)

plt**.**xlabel('Value', fontsize**=**14)

plt**.**ylabel('Cube of Value', fontsize**=**14)

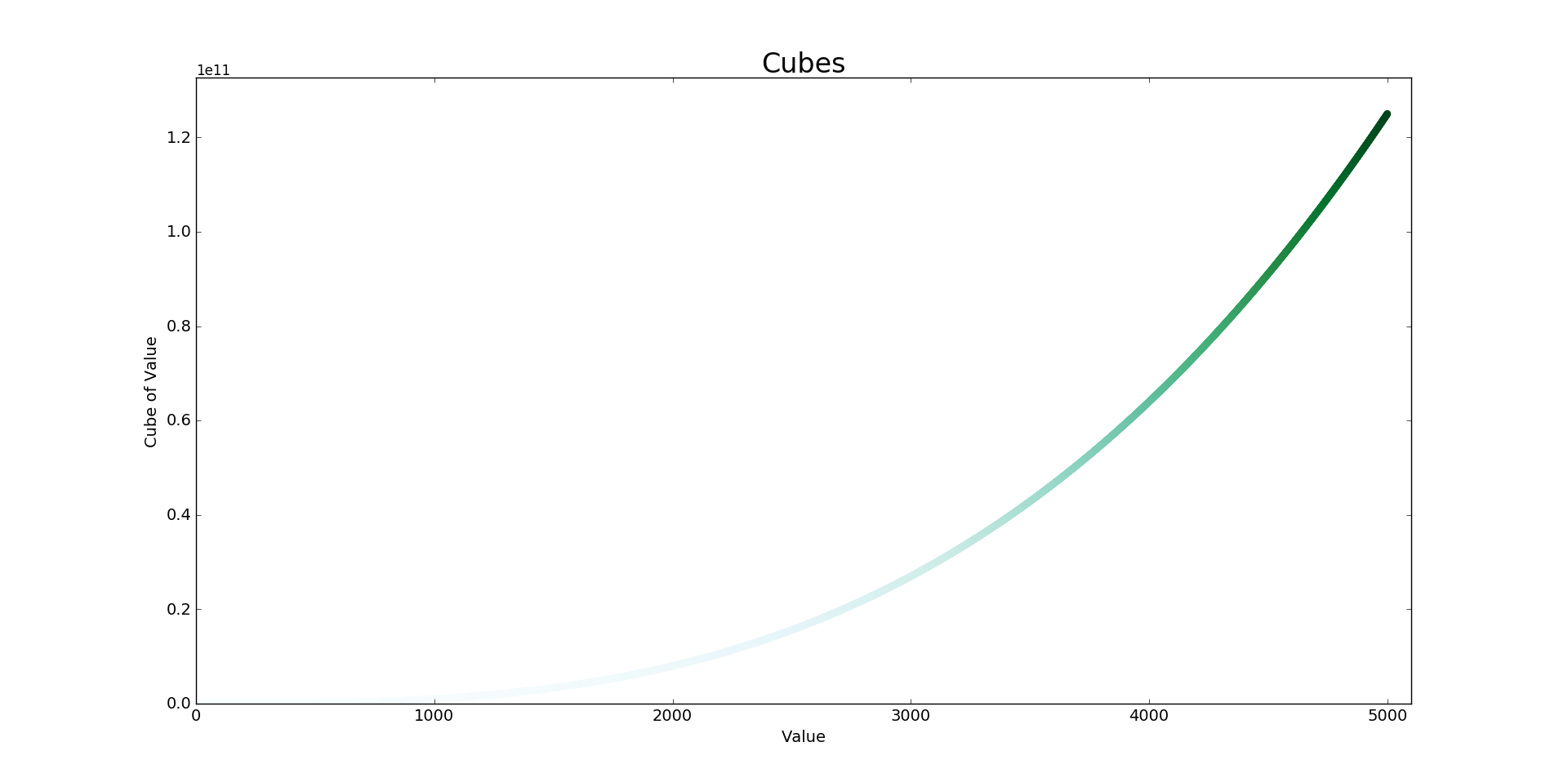
plt**.**tick\_params(axis**=**'both', labelsize**=**14)

plt**.**axis([0, 5100, 0, 5100**\*\***3])

*# Show plot.*

plt**.**show()

Output:



## 15-3: Molecular Motion

Modify rw\_visual.py by replacing plt.scatter() with plt.plot(). to simulate the path of a pollen grain on the surface of a drop of water, pass in the rw.x\_values and rw.y\_values, and include alinewidth argument. Use 5000 instead of 50,000 points.

import matplotlib.pyplot **as** plt

from random\_walk import RandomWalk

*# Keep making new walks, as long as the program is active.*

**while** True:

*# Make a random walk, and plot the points.*

rw **=** RandomWalk(5000)

rw**.**fill\_walk()

*# Set the size of the plotting window.*

plt**.**figure(dpi**=**128, figsize**=**(10, 6))

point\_numbers **=** list(range(rw**.**num\_points))

plt**.**plot(rw**.**x\_values, rw**.**y\_values, linewidth**=**1)

*# Emphasize the first and last points.*

plt**.**scatter(0, 0, c**=**'green', edgecolors**=**'none', s**=**75)

plt**.**scatter(rw**.**x\_values[**-**1], rw**.**y\_values[**-**1], c**=**'red', edgecolors**=**'none',

s**=**75)

*# Remove the axes.*

plt**.**axes()**.**get\_xaxis()**.**set\_visible(False)

plt**.**axes()**.**get\_yaxis()**.**set\_visible(False)

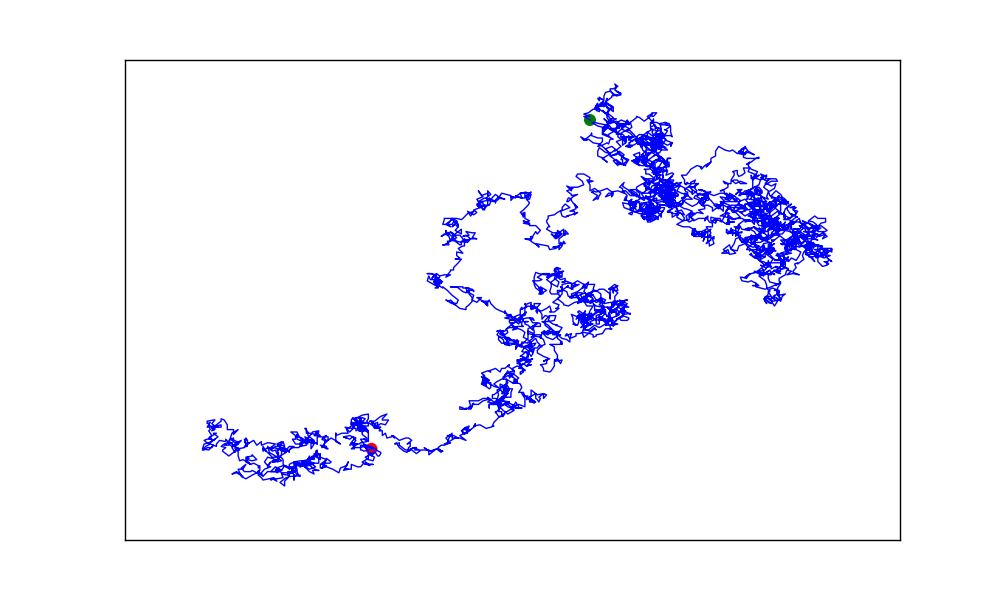
plt**.**show()

keep\_running **=** input("Make another walk? (y/n): ")

**if** keep\_running **==** 'n':

**break**

Output:



The scatter plots appear behind the lines. To place them on top of the lines, we can use the zorderargument. Plot elements with higher zorder values are placed on top of elements with lowerzorder values.

import matplotlib.pyplot **as** plt

from random\_walk import RandomWalk

*# Keep making new walks, as long as the program is active.*

**while** True:

*# Make a random walk, and plot the points.*

rw **=** RandomWalk(5000)

rw**.**fill\_walk()

*# Set the size of the plotting window.*

plt**.**figure(dpi**=**128, figsize**=**(10, 6))

point\_numbers **=** list(range(rw**.**num\_points))

plt**.**plot(rw**.**x\_values, rw**.**y\_values, linewidth**=**1, zorder**=**1)

*# Emphasize the first and last points.*

plt**.**scatter(0, 0, c**=**'green', edgecolors**=**'none', s**=**75, zorder**=**2)

plt**.**scatter(rw**.**x\_values[**-**1], rw**.**y\_values[**-**1], c**=**'red', edgecolors**=**'none',

s**=**75, zorder**=**2)

*# Remove the axes.*

plt**.**axes()**.**get\_xaxis()**.**set\_visible(False)

plt**.**axes()**.**get\_yaxis()**.**set\_visible(False)

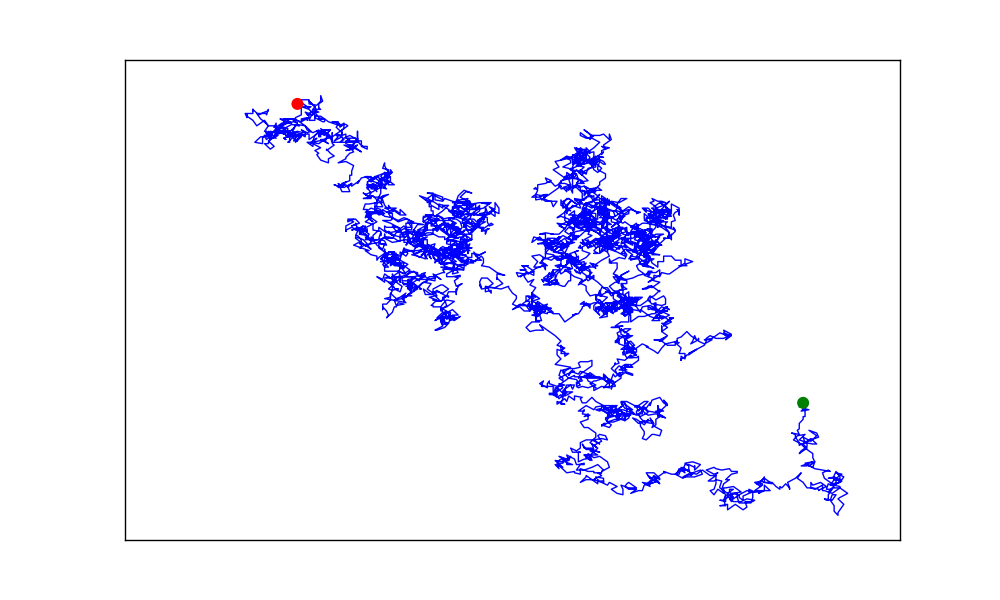
plt**.**show()

keep\_running **=** input("Make another walk? (y/n): ")

**if** keep\_running **==** 'n':

**break**

Output:



## 15-5: Refactoring

The method fill\_walk() is lengthy. Create a new method called get\_step() to determine the direction and distance for each step, and then calculate the step. You should end up with two calls to get\_step() in fill\_walk():

x\_step **=** get\_step()

y\_step **=** get\_step()

This refactoring should reduce the size of fill\_walk() and make the method easier to read and understand.

random\_walk.py:

from random import choice

**class** **RandomWalk**():

"""A class to generate random walks."""

**def** **\_\_init\_\_**(self, num\_points**=**5000):

"""Initialize attributes of a walk."""

self**.**num\_points **=** num\_points

*# All walks start at (0, 0).*

self**.**x\_values **=** [0]

self**.**y\_values **=** [0]

**def** **get\_step**(self):

"""Determine the direction and distance for a step."""

direction **=** choice([1, **-**1])

distance **=** choice([0, 1, 2, 3, 4])

step **=** direction **\*** distance

**return** step

**def** **fill\_walk**(self):

"""Calculate all the points in the walk."""

*# Keep taking steps until the walk reaches the desired length.*

**while** len(self**.**x\_values) **<** self**.**num\_points:

*# Decide which direction to go, and how far to go in that direction.*

x\_step **=** self**.**get\_step()

y\_step **=** self**.**get\_step()

*# Reject moves that go nowhere.*

**if** x\_step **==** 0 **and** y\_step **==** 0:

**continue**

*# Calculate the next x and y values.*

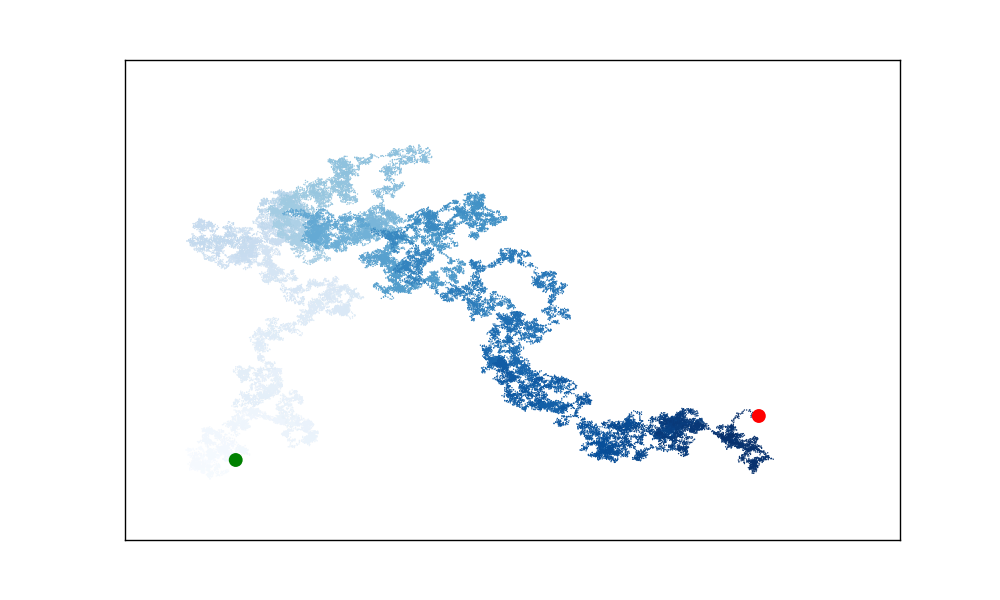
next\_x **=** self**.**x\_values[**-**1] **+** x\_step

next\_y **=** self**.**y\_values[**-**1] **+** y\_step

self**.**x\_values**.**append(next\_x)

self**.**y\_values**.**append(next\_y)

Output:



## 15-6: Automatic Labels

Modify die.py and dice\_visual.py by replacing the list we used to set the value of hist.x\_labels with a loop to generate this list automatically. If you’re comfortable with list comprehensions, try replacing the other for loops in die\_visual.py and dice\_visual.py with comprehensions as well.

***Note:***This should say to modify die\_visual.py, not die.py. This will be corrected in future printings.

die\_visual.py:

import pygal

from die import Die

*# Create a D6.*

die **=** Die()

*# Make some rolls, and store results in a list.*

results **=** [die**.**roll() **for** roll\_num **in** range(1000)]

*# Analyze the results.*

frequencies **=** [results**.**count(value) **for** value **in** range(1, die**.**num\_sides**+**1)]

*# Visualize the results.*

hist **=** pygal**.**Bar()

hist**.**title **=** "Results of rolling one D6 1000 times."

hist**.**x\_labels **=** [str(x) **for** x **in** range(1, die**.**num\_sides**+**1)]

hist**.**x\_title **=** "Result"

hist**.**y\_title **=** "Frequency of Result"

hist**.**add('D6', frequencies)

hist**.**render\_to\_file('die\_visual.svg')

dice\_visual.py:

import pygal

from die import Die

*# Create two D6 dice.*

die\_1 **=** Die()

die\_2 **=** Die()

*# Make some rolls, and store results in a list.*

results **=** [die\_1**.**roll() **+** die\_2**.**roll() **for** roll\_num **in** range(1000)]

*# Analyze the results.*

max\_result **=** die\_1**.**num\_sides **+** die\_2**.**num\_sides

frequencies **=** [results**.**count(value) **for** value **in** range(2, max\_result**+**1)]

*# Visualize the results.*

hist **=** pygal**.**Bar()

hist**.**title **=** "Results of rolling two D6 dice 1000 times."

hist**.**x\_labels **=** [str(x) **for** x **in** range(2, max\_result**+**1)]

hist**.**x\_title **=** "Result"

hist**.**y\_title **=** "Frequency of Result"

hist**.**add('D6 + D6', frequencies)

hist**.**render\_to\_file('dice\_visual.svg')

## 15-7: Two D8s

Create a simulation showing what happens if you roll two eight-sided dice 1000 times. Increase the number of rolls gradually until you start to see the limits of your system’s capabilities.

import pygal

from die import Die

*# Create two D8 dice.*

die\_1 **=** Die(8)

die\_2 **=** Die(8)

*# Make some rolls, and store results in a list.*

results **=** []

**for** roll\_num **in** range(1000000):

result **=** die\_1**.**roll() **+** die\_2**.**roll()

results**.**append(result)

*# Analyze the results.*

frequencies **=** []

max\_result **=** die\_1**.**num\_sides **+** die\_2**.**num\_sides

**for** value **in** range(2, max\_result**+**1):

frequency **=** results**.**count(value)

frequencies**.**append(frequency)

*# Visualize the results.*

hist **=** pygal**.**Bar()

hist**.**title **=** "Results of rolling two D8 dice 1,000,000 times."

hist**.**x\_labels **=** [str(x) **for** x **in** range(2, max\_result**+**1)]

hist**.**x\_title **=** "Result"

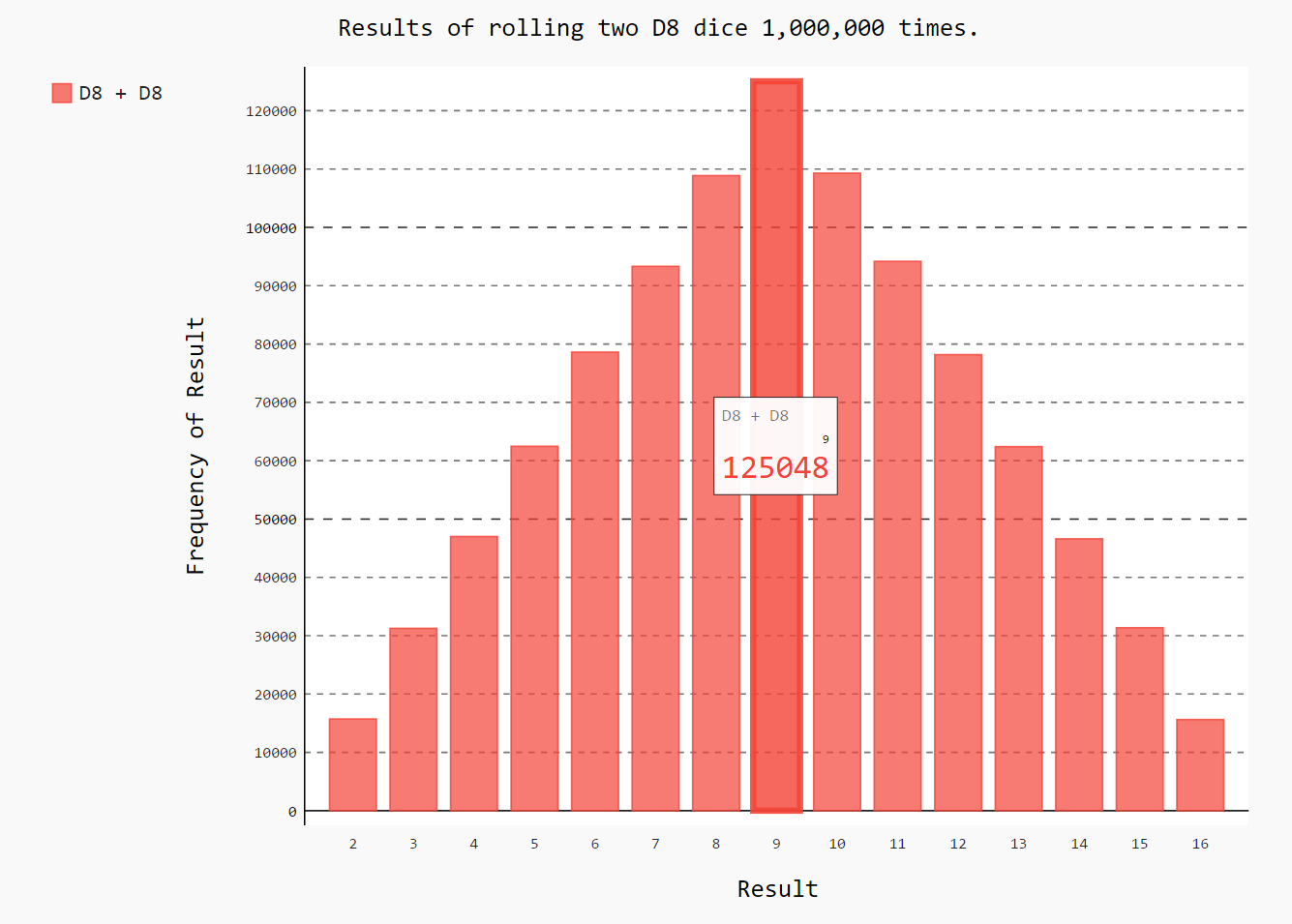
hist**.**y\_title **=** "Frequency of Result"

hist**.**add('D8 + D8', frequencies)

hist**.**render\_to\_file('dice\_visual.svg')

Note: This solution only uses a list comprehension for the hist.x\_labels parameter. You might want to try replacing the other loops with comprehensions as well.

Output:



## 15-8: Three Dice

If you roll three D6 dice, the smallest number you can roll is 3 and the largest number is 18. Create a visualization that shows what happens when you roll three D6 dice.

import pygal

from die import Die

*# Create three D6 dice.*

die\_1 **=** Die()

die\_2 **=** Die()

die\_3 **=** Die()

*# Make some rolls, and store results in a list.*

results **=** []

**for** roll\_num **in** range(1000000):

result **=** die\_1**.**roll() **+** die\_2**.**roll() **+** die\_3**.**roll()

results**.**append(result)

*# Analyze the results.*

frequencies **=** []

max\_result **=** die\_1**.**num\_sides **+** die\_2**.**num\_sides **+** die\_3**.**num\_sides

**for** value **in** range(3, max\_result**+**1):

frequency **=** results**.**count(value)

frequencies**.**append(frequency)

*# Visualize the results.*

hist **=** pygal**.**Bar()

hist**.**title **=** "Results of rolling three D6 dice 1,000,000 times."

hist**.**x\_labels **=** [str(x) **for** x **in** range(3, max\_result**+**1)]

hist**.**x\_title **=** "Result"

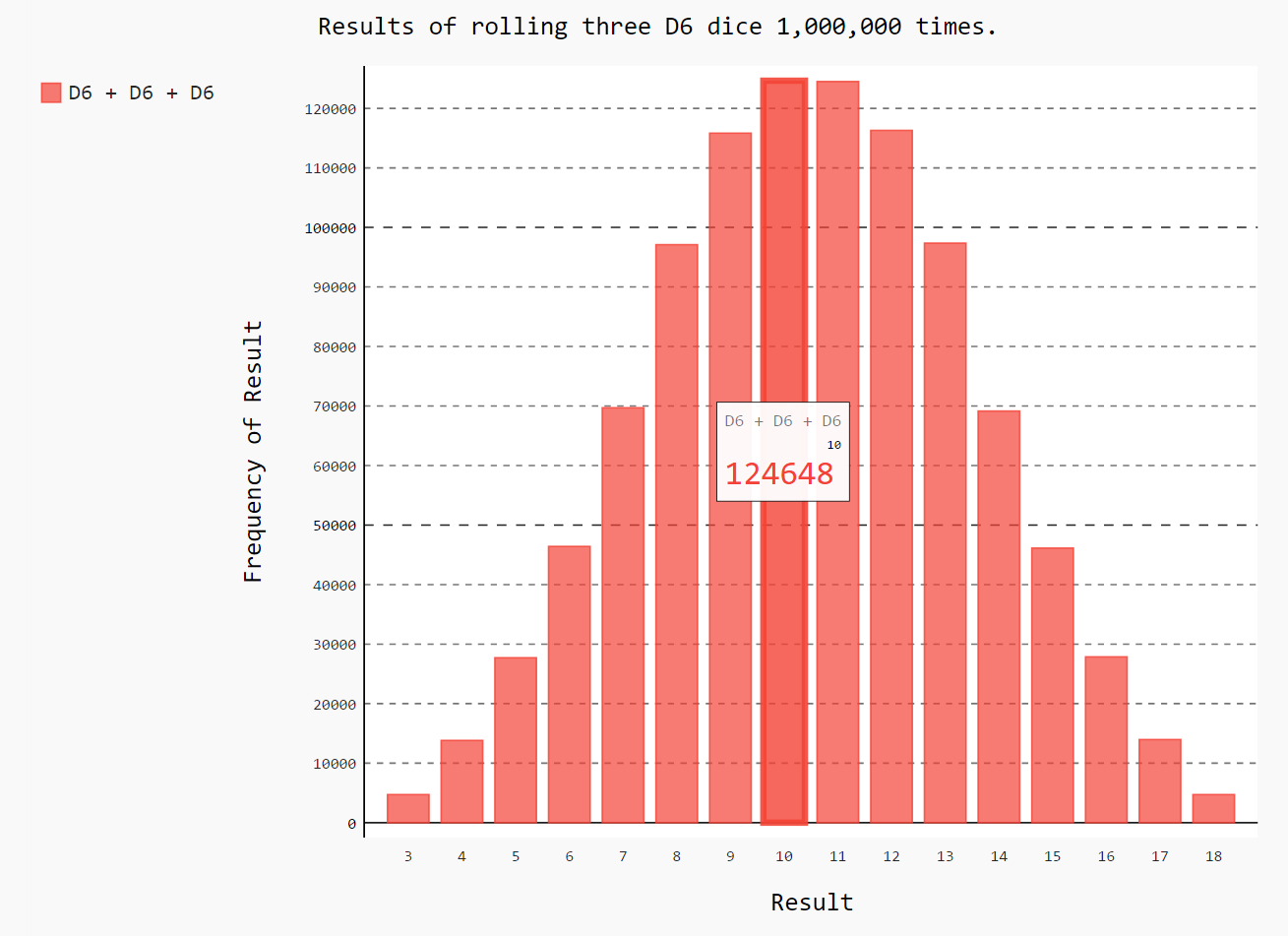
hist**.**y\_title **=** "Frequency of Result"

hist**.**add('D6 + D6 + D6', frequencies)

hist**.**render\_to\_file('dice\_visual.svg')

Note: This solution only uses a list comprehension for the hist.x\_labels parameter. You might want to try replacing the other loops with comprehensions as well.

Output:



## 15-9: Multiplication

When you roll two dice, you usually add the two numbers together to get the result. Create a visualization that shows what happens if you multiply these numbers instead.

import pygal

from die import Die

*# Create two D6 dice.*

die\_1 **=** Die()

die\_2 **=** Die()

*# Make some rolls, and store results in a list.*

results **=** []

**for** roll\_num **in** range(1000000):

result **=** die\_1**.**roll() **\*** die\_2**.**roll()

results**.**append(result)

*# Analyze the results.*

frequencies **=** []

max\_result **=** die\_1**.**num\_sides **\*** die\_2**.**num\_sides

**for** value **in** range(1, max\_result**+**1):

frequency **=** results**.**count(value)

frequencies**.**append(frequency)

*# Visualize the results.*

hist **=** pygal**.**Bar()

hist**.**title **=** "Results of multiplying two D6 dice. (1,000,000 rolls)"

hist**.**x\_labels **=** [str(x) **for** x **in** range(1, max\_result**+**1)]

hist**.**x\_title **=** "Result"

hist**.**y\_title **=** "Frequency of Result"

hist**.**add('D6 \* D6', frequencies)

hist**.**render\_to\_file('dice\_visual.svg')

Note: This solution only uses a list comprehension for the hist.x\_labels parameter. You might want to try replacing the other loops with comprehensions as well.

Output:

