**Intermediate Python for Data Science**

**#4 Random Numbers**

**Random float**

Randomness has many uses in science, art, statistics, cryptography, gaming, gambling, and other fields. You're going to use randomness to simulate a game.

All the functionality you need is contained in the random package, a sub-package of numpy. In this exercise, you'll be using two functions from this package:

* [**seed()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.seed.html): sets the random seed, so that your results are the reproducible between simulations. As an argument, it takes an integer of your choosing. If you call the function, no output will be generated.
* [**rand()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.rand.html): if you don't specify any arguments, it generates a random float between zero and one.

**Instructions**

**100 XP**

* Import numpy as np.
* Use [**seed()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.seed.html) to set the seed; as an argument, pass 123.
* Generate your first random float with [**rand()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.rand.html) and print it out.

Script.py  
01 # Import numpy as np

02 import numpy as np

03

04 # Set the seed

05 np.random.seed(123)

06

07 # Generate and print random float

08 print(np.random.rand())

IPython Shell  
In [1]: # Import numpy as np

import numpy as np

# Set the seed

np.random.seed(123)

# Generate and print random float

print(np.random.rand())

0.6964691855978616

<script.py> output:

0.6964691855978616

In [2]:

**Roll the dice**

In the previous exercise, you used [**rand()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.rand.html), that generates a random float between 0 and 1.

As Filip explained in the video you can just as well use [**randint()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.randint.html), also a function of the random package, to generate integers randomly. The following call generates the integer 4, 5, 6 or 7 randomly. **8 is not included**.

import numpy as np

np.random.randint(4, 8)

Numpy has already been imported as np and a seed has been set. Can you roll some dice?

**Instructions**

**100 XP**

* Use [**randint()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.randint.html) with the appropriate arguments to randomly generate the integer 1, 2, 3, 4, 5 or 6. This simulates a dice. Print it out.
* Repeat the outcome to see if the second throw is different. Again, print out the result.

Script.py  
1 # Import numpy and set seed

2 import numpy as np

3 np.random.seed(123)

4

5 # Use randint() to simulate a dice

6 print(np.random.randint(1, 7))

7

8 # Use randint() again

9 print(np.random.randint(1, 7))

In [2]: # Import numpy and set seed

import numpy as np

np.random.seed(123)

# Use randint() to simulate a dice

print(np.random.randint(1, 7))

# Use randint() again

print(np.random.randint(1, 7))

6

3

In [3]:

**Determine your next move**

In the Empire State Building bet, your next move depends on the number of eyes you throw with the dice. We can perfectly code this with an if-elif-else construct!

The sample code assumes that you're currently at step 50. Can you fill in the missing pieces to finish the script? numpy is already imported as np and the seed has been set to 123, so you don't have to worry about that anymore.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Roll the dice. Use [**randint()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.randint.html) to create the variable dice.
* Finish the if-elif-else construct by replacing \_\_\_:
* If dice is 1 or 2, you go one step down.
* if dice is 3, 4 or 5, you go one step up.
* Else, you throw the dice again. The number of eyes is the number of steps you go up.
* Print out dice and step. Given the value of dice, was step updated correctly?

Script.py  
01 # Numpy is imported, seed is set

02

03 # Starting step

04 step = 50

05

06 # Roll the dice

07 dice = np.random.randint(1, 7)

08

09 # Finish the control construct

10 if dice <= 2 :

11 step = step - 1

12 elif dice <=5 :

13 step = step + 1

14 else :

15 step = step + np.random.randint(1,7)

16

17 # Print out dice and step

18 print(dice)

19 print(step)

IPython Shell  
In [1]: # Numpy is imported, seed is set

# Starting step

step = 50

# Roll the dice

dice = np.random.randint(1, 7)

# Finish the control construct

if dice <= 2 :

step = step - 1

elif dice <=5 :

step = step + 1

else :

step = step + np.random.randint(1,7)

# Print out dice and step

print(dice)

print(step)

6

53

In [2]:

**The next step**

Before, you have already written Python code that determines the next step based on the previous step. Now it's time to put this code inside a for loop so that we can simulate a random walk.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Make a list random\_walk that contains the first step, which is the integer 0.
* Finish the for loop:
* The loop should run 100 times.
* On each iteration, set step equal to the last element in the random\_walk list. You can use the index -1 for this.
* Next, let the if-elif-else construct update step for you.
* The code that appends step to random\_walk is already coded.
* Print out random\_walk.

Script.py  
01 # Numpy is imported, seed is set

02

03 # Initialize random\_walk

04 random\_walk = [0]

05

06 # Complete the \_\_\_

07 for x in range(100) :

08 # Set step: last element in random\_walk

09 step = random\_walk [-1]

10

11 # Roll the dice

12 dice = np.random.randint(1,7)

13

14 # Determine next step

15 if dice <= 2:

16 step = step - 1

17 elif dice <= 5:

18 step = step + 1

19 else:

20 step = step + np.random.randint(1,7)

21

22 # append next\_step to random\_walk

23 random\_walk.append(step)

24

25 # Print random\_walk

26 print(random\_walk)

IPython Shell  
In [1]: # Numpy is imported, seed is set

# Initialize random\_walk

random\_walk = [0]

# Complete the \_\_\_

for x in range(100) :

# Set step: last element in random\_walk

step = random\_walk [-1]

# Roll the dice

dice = np.random.randint(1,7)

# Determine next step

if dice <= 2:

step = step - 1

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

# append next\_step to random\_walk

random\_walk.append(step)

# Print random\_walk

print(random\_walk)

[0, 1, 2, 1, 2, 3, 4, 3, 2, 1, 0, -1, -2, -3, -4, -3, 2, 1, 0, 1, 0, 1, 2, 3, 4, 5, 4, 5, 4, 5, 6, 7, 8, 7, 11, 12, 11, 12, 11, 12, 13, 14, 15, 16, 17, 18, 21, 22, 23, 24, 29, 30, 34, 35, 34, 35, 36, 35, 36, 37, 39, 40, 41, 40, 39, 40, 41, 40, 39, 40, 41, 43, 42, 41, 42, 41, 42, 43, 44, 46, 45, 46, 47, 48, 49, 50, 49, 48, 49, 48, 49, 50, 49, 52, 53, 54, 55, 54, 55, 56, 60]

<script.py> output:

[0, 3, 4, 5, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 0, -1, 0, 5, 4, 3, 4, 3, 4, 5, 6, 7, 8, 7, 8, 7, 8, 9, 10, 11, 10, 14, 15, 14, 15, 14, 15, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 32, 33, 37, 38, 37, 38, 39, 38, 39, 40, 42, 43, 44, 43, 42, 43, 44, 43, 42, 43, 44, 46, 45, 44, 45, 44, 45, 46, 47, 49, 48, 49, 50, 51, 52, 53, 52, 51, 52, 51, 52, 53, 52, 55, 56, 57, 58, 57, 58, 59]

In [2]:

**How low can you go?**

Things are shaping up nicely! You already have code that calculates your location in the Empire State Building after 100 dice throws. However, there's something we haven't thought about - you can't go below 0!

A typical way to solve problems like this is by using [**max()**](https://docs.python.org/3/library/functions.html#max). If you pass [**max()**](https://docs.python.org/3/library/functions.html#max) two arguments, the biggest one gets returned. For example, to make sure that a variable x never goes below 10when you decrease it, you can use:

x = max(10, x - 1)

**Instructions**

**100 XP**

* Use [**max()**](https://docs.python.org/3/library/functions.html#max) in a similar way to make sure that step doesn't go below zero if dice <= 2.
* Hit *Submit Answer* and check the contents of random\_walk.

Script.py  
01 # Numpy is imported, seed is set

02

03 # Initialize random\_walk

04 random\_walk = [0]

05

06 for x in range(100) :

07 step = random\_walk[-1]

08 dice = np.random.randint(1,7)

09

10 if dice <= 2:

11 # Replace below: use max to make sure step can't go below 0

12 step = max(0, step - 1)

13 elif dice <= 5:

14 step = step + 1

15 else:

16 step = step + np.random.randint(1,7)

17

18 random\_walk.append(step)

19

20 print(random\_walk)

IPython Shell  
In [1]: # Numpy is imported, seed is set

# Initialize random\_walk

random\_walk = [0]

for x in range(100) :

step = random\_walk[-1]

dice = np.random.randint(1,7)

if dice <= 2:

# Replace below: use max to make sure step can't go below 0

step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

random\_walk.append(step)

print(random\_walk)

[0, 3, 4, 5, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 0, 0, 1, 6, 5, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 8, 9, 10, 11, 12, 11, 15, 16, 15, 16, 15, 16, 17, 18, 19, 20, 21, 22, 25, 26, 27, 28, 33, 34, 38, 39, 38, 39, 40, 39, 40, 41, 43, 44, 45, 44, 43, 44, 45, 44, 43, 44, 45, 47, 46, 45, 46, 45, 46, 47, 48, 50, 49, 50, 51, 52, 53, 54, 53, 52, 53, 52, 53, 54, 53, 56, 57, 58, 59, 58, 59, 60]

<script.py> output:

[0, 3, 4, 5, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 0, 0, 1, 6, 5, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 8, 9, 10, 11, 12, 11, 15, 16, 15, 16, 15, 16, 17, 18, 19, 20, 21, 22, 25, 26, 27, 28, 33, 34, 38, 39, 38, 39, 40, 39, 40, 41, 43, 44, 45, 44, 43, 44, 45, 44, 43, 44, 45, 47, 46, 45, 46, 45, 46, 47, 48, 50, 49, 50, 51, 52, 53, 54, 53, 52, 53, 52, 53, 54, 53, 56, 57, 58, 59, 58, 59, 60]

In [2]:

**Visualize the walk**

Let's visualize this random walk! Remember how you could use matplotlib to build a line plot?

import matplotlib.pyplot as plt

plt.plot(x, y)

plt.show()

The first list you pass is mapped onto the x axis and the second list is mapped onto the y axis.

If you pass only one argument, Python will know what to do and will use the index of the list to map onto the x axis, and the values in the list onto the y axis.

**Instructions**

**100 XP**

Add some lines of code after the for loop:

* Import matplotlib.pyplot as plt.
* Use [**plt.plot()**](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot) to plot random\_walk.
* Finish off with [**plt.show()**](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show) to actually display the plot.

Script.py  
01 # Numpy is imported, seed is set

02

03 # Initialization

04 random\_walk = [0]

05

06 for x in range(100) :

07 step = random\_walk[-1]

08 dice = np.random.randint(1,7)

09

10 if dice <= 2:

11 step = max(0, step - 1)

12 elif dice <= 5:

13 step = step + 1

14 else:

15 step = step + np.random.randint(1,7)

16

17 random\_walk.append(step)

18

19 # Import matplotlib.pyplot as plt

20 import matplotlib.pyplot as plt

21

22 # Plot random\_walk

23 plt.plot(random\_walk)

24

25 # Show the plot

26 plt.show()

IPython Shell  
In [1]: # Numpy is imported, seed is set

# Initialization

random\_walk = [0]

for x in range(100) :

step = random\_walk[-1]

dice = np.random.randint(1,7)

if dice <= 2:

step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

random\_walk.append(step)

# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

# Plot random\_walk

plt.plot(random\_walk)

# Show the plot

plt.show()

In [2]:

**Simulate multiple walks**

A single random walk is one thing, but that doesn't tell you if you have a good chance at winning the bet.

To get an idea about how big your chances are of reaching 60 steps, you can repeatedly simulate the random walk and collect the results. That's exactly what you'll do in this exercise.

The sample code already sets you off in the right direction. Another for loop is wrapped around the code you already wrote. It's up to you to add some bits and pieces to make sure all of the results are recorded correctly.

**Note: Don't change anything about the initialization of all\_walks that is given. Setting any number inside the list will cause the exercise to crash!**

**Instructions**

**100 XP**

* Fill in the specification of the for loop so that the random walk is simulated 10 times.
* After the random\_walk array is entirely populated, append the array to the all\_walks list.
* Finally, after the top-level for loop, print out all\_walks.

Script.py  
01 # Numpy is imported; seed is set

02

03 # Initialize all\_walks (don't change this line)

04 all\_walks = []

05

06 # Simulate random walk 10 times

07 for i in range(10) :

08

09 # Code from before

10 random\_walk = [0]

11 for x in range(100) :

12 step = random\_walk[-1]

13 dice = np.random.randint(1,7)

14

15 if dice <= 2:

16 step = max(0, step - 1)

17 elif dice <= 5:

18 step = step + 1

19 else:

20 step = step + np.random.randint(1,7)

21 random\_walk.append(step)

22

23 # Append random\_walk to all\_walks

24 all\_walks.append(random\_walk)

25

26 # Print all\_walks

27 print(all\_walks)

IPython Shell  
In [3]: # Numpy is imported; seed is set

# Initialize all\_walks (don't change this line)

all\_walks = []

# Simulate random walk 10 times

for i in range(10) :

# Code from before

random\_walk = [0]

for x in range(100) :

step = random\_walk[-1]

dice = np.random.randint(1,7)

if dice <= 2:

step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

random\_walk.append(step)

# Append random\_walk to all\_walks

all\_walks.append(random\_walk)

# Print all\_walks

print(all\_walks)

[[0, 0, 1, 0, 1, 2, 3, 2, 1, 2, 3, 4, 3, 2, 1, 3, 4, 5, 4, 3, 2, 3, 4, 5, 4, 3, 4, 7, 12, 15, 16, 17, 23, 24, 25, 26, 25, 27, 32, 33, 34, 35, 36, 37, 38, 37, 38, 39, 40, 41, 42, 44, 48, 49, 50, 51, 52, 56, 61, 60, 59, 58, 57, 60, 61, 62, 63, 62, 61, 64, 65, 64, 63, 62, 63, 64, 65, 66, 65, 66, 65, 66, 67, 66, 67, 68, 69, 70, 71, 72, 73, 72, 71, 72, 73, 76, 77, 76, 75, 76, 77], [0, 1, 6, 5, 4, 3, 2, 1, 0, 1, 0, 0, 1, 4, 3, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 21, 22, 23, 24, 25, 26, 25, 24, 23, 24, 25, 26, 27, 29, 30, 31, 32, 34, 38, 37, 36, 35, 34, 35, 36, 37, 36, 35, 34, 33, 32, 31, 32, 36, 40, 41, 42, 41, 40, 41, 42, 43, 49, 50, 49, 48, 49, 48, 49, 48, 49, 50, 49, 50, 49, 48, 49, 50, 49, 50, 49, 50, 53, 54, 55, 56, 57, 56, 57, 58, 63, 62], [0, 1, 2, 3, 7, 6, 5, 6, 7, 9, 8, 7, 8, 11, 10, 11, 10, 16, 17, 18, 19, 20, 19, 18, 19, 22, 21, 22, 23, 24, 25, 26, 30, 29, 30, 29, 30, 31, 32, 33, 34, 35, 36, 35, 36, 37, 38, 39, 40, 39, 42, 43, 44, 43, 44, 43, 44, 43, 44, 43, 42, 43, 44, 45, 46, 47, 46, 45, 44, 45, 46, 45, 46, 45, 46, 47, 46, 47, 46, 45, 46, 47, 52, 51, 52, 53, 52, 53, 54, 60, 61, 60, 59, 60, 64, 63, 62, 61, 60, 59, 58], [0, 1, 2, 7, 9, 13, 12, 17, 18, 19, 18, 17, 18, 19, 20, 19, 18, 19, 20, 21, 22, 21, 20, 21, 22, 23, 24, 30, 32, 33, 38, 39, 38, 37, 38, 39, 44, 43, 44, 45, 47, 48, 47, 48, 49, 50, 49, 50, 49, 48, 47, 48, 47, 46, 47, 46, 47, 52, 53, 54, 55, 56, 55, 56, 55, 57, 58, 57, 58, 57, 56, 57, 56, 62, 63, 62, 63, 64, 65, 64, 66, 67, 71, 72, 73, 72, 71, 72, 73, 74, 75, 78, 77, 76, 77, 78, 77, 78, 79, 78, 83], [0, 1, 2, 8, 9, 10, 9, 8, 9, 10, 11, 12, 13, 14, 15, 20, 19, 18, 17, 18, 19, 20, 19, 23, 29, 30, 35, 36, 35, 36, 42, 41, 42, 41, 42, 41, 42, 41, 42, 47, 48, 47, 48, 49, 50, 51, 50, 51, 52, 53, 52, 53, 56, 55, 56, 57, 58, 59, 62, 61, 60, 66, 67, 69, 72, 73, 76, 77, 78, 79, 80, 81, 82, 81, 82, 83, 82, 83, 82, 81, 82, 85, 84, 83, 84, 85, 84, 85, 91, 90, 91, 90, 91, 92, 91, 90, 89, 91, 90, 95, 96], [0, 3, 4, 5, 6, 11, 17, 20, 23, 22, 23, 24, 23, 26, 25, 26, 27, 28, 29, 28, 29, 30, 36, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 50, 56, 58, 57, 60, 63, 64, 63, 62, 68, 69, 72, 73, 74, 75, 76, 77, 78, 79, 78, 77, 78, 77, 78, 77, 78, 77, 76, 77, 78, 79, 78, 79, 78, 83, 82, 81, 82, 86, 87, 88, 87, 88, 87, 88, 87, 86, 85, 86, 87, 88, 89, 88, 89, 88, 89, 90, 89, 90, 89, 90, 89, 88, 89, 90, 91, 92, 91], [0, 2, 4, 5, 6, 5, 6, 7, 8, 7, 12, 11, 12, 13, 12, 16, 20, 26, 27, 26, 27, 28, 29, 28, 32, 33, 37, 36, 37, 38, 39, 40, 41, 42, 43, 42, 41, 42, 41, 42, 41, 42, 43, 48, 53, 52, 51, 52, 53, 54, 55, 54, 55, 54, 53, 54, 53, 54, 53, 54, 57, 58, 57, 58, 63, 64, 65, 71, 77, 76, 77, 78, 77, 76, 77, 78, 77, 83, 84, 85, 86, 85, 88, 87, 86, 87, 88, 87, 93, 98, 97, 98, 97, 98, 99, 100, 101, 102, 103, 102, 104], [0, 0, 1, 2, 3, 4, 5, 4, 3, 2, 3, 2, 3, 2, 1, 5, 7, 12, 11, 17, 18, 17, 18, 19, 20, 21, 24, 25, 26, 27, 30, 29, 30, 29, 28, 32, 31, 32, 33, 39, 38, 39, 40, 41, 42, 41, 42, 44, 45, 46, 45, 46, 47, 46, 47, 48, 49, 50, 56, 57, 58, 59, 60, 61, 60, 59, 60, 59, 60, 61, 62, 61, 62, 61, 67, 68, 67, 66, 67, 66, 67, 68, 69, 70, 69, 68, 69, 68, 67, 68, 69, 68, 67, 68, 67, 66, 67, 68, 69, 68, 69], [0, 4, 3, 4, 5, 6, 7, 8, 9, 8, 7, 12, 13, 12, 13, 14, 15, 16, 18, 17, 18, 19, 23, 24, 25, 24, 23, 22, 23, 22, 21, 22, 26, 27, 26, 27, 28, 32, 33, 34, 35, 36, 35, 36, 37, 38, 39, 38, 37, 39, 40, 39, 40, 41, 43, 44, 43, 44, 43, 44, 43, 44, 45, 49, 50, 49, 48, 47, 48, 47, 46, 45, 46, 45, 46, 45, 46, 49, 50, 51, 52, 53, 52, 53, 54, 57, 56, 55, 59, 63, 64, 65, 70, 71, 70, 69, 70, 71, 70, 73, 74], [0, 0, 1, 0, 1, 0, 1, 0, 5, 6, 5, 6, 11, 12, 13, 12, 13, 14, 13, 14, 13, 12, 13, 14, 15, 14, 13, 14, 15, 16, 15, 16, 15, 14, 15, 14, 13, 14, 13, 14, 13, 14, 15, 16, 17, 20, 19, 23, 24, 25, 30, 29, 28, 27, 28, 29, 32, 33, 32, 31, 32, 33, 32, 33, 32, 33, 34, 37, 43, 44, 43, 42, 43, 42, 41, 42, 41, 42, 43, 44, 45, 46, 45, 46, 47, 46, 47, 46, 47, 48, 47, 48, 47, 48, 47, 48, 49, 53, 52, 51, 50]]

<script.py> output:

[[0, 3, 4, 5, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 0, 0, 1, 6, 5, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 8, 9, 10, 11, 12, 11, 15, 16, 15, 16, 15, 16, 17, 18, 19, 20, 21, 22, 25, 26, 27, 28, 33, 34, 38, 39, 38, 39, 40, 39, 40, 41, 43, 44, 45, 44, 43, 44, 45, 44, 43, 44, 45, 47, 46, 45, 46, 45, 46, 47, 48, 50, 49, 50, 51, 52, 53, 54, 53, 52, 53, 52, 53, 54, 53, 56, 57, 58, 59, 58, 59, 60], [0, 4, 3, 2, 4, 3, 4, 6, 7, 8, 13, 12, 13, 14, 15, 16, 17, 16, 21, 22, 23, 24, 23, 22, 21, 20, 19, 20, 21, 22, 28, 27, 26, 25, 26, 27, 28, 27, 28, 29, 28, 33, 34, 33, 32, 31, 30, 31, 30, 29, 31, 32, 35, 36, 38, 39, 40, 41, 40, 39, 40, 41, 42, 43, 42, 43, 44, 45, 48, 49, 50, 49, 50, 49, 50, 51, 52, 56, 55, 54, 55, 56, 57, 56, 57, 56, 57, 59, 64, 63, 64, 65, 66, 67, 68, 69, 68, 69, 70, 71, 73], [0, 2, 1, 2, 3, 6, 5, 6, 5, 6, 7, 8, 7, 8, 7, 8, 9, 11, 10, 9, 10, 11, 10, 12, 13, 14, 15, 16, 17, 18, 17, 18, 19, 24, 25, 24, 23, 22, 21, 22, 23, 24, 29, 30, 29, 30, 31, 32, 33, 34, 35, 34, 33, 34, 33, 39, 38, 39, 38, 39, 38, 39, 43, 47, 49, 51, 50, 51, 53, 52, 58, 59, 61, 62, 61, 62, 63, 64, 63, 64, 65, 66, 68, 67, 66, 67, 73, 78, 77, 76, 80, 81, 82, 83, 85, 84, 85, 84, 85, 84, 83], [0, 6, 5, 6, 7, 8, 9, 10, 11, 12, 13, 12, 13, 12, 11, 12, 11, 12, 11, 12, 13, 17, 18, 17, 23, 22, 21, 22, 21, 20, 21, 20, 24, 23, 24, 23, 24, 23, 24, 26, 25, 24, 23, 24, 23, 28, 29, 30, 29, 28, 29, 28, 29, 28, 33, 34, 33, 32, 31, 30, 31, 32, 36, 42, 43, 44, 45, 46, 45, 46, 48, 49, 50, 51, 50, 49, 50, 49, 50, 51, 52, 51, 52, 53, 54, 53, 52, 53, 54, 59, 60, 61, 66, 65, 66, 65, 66, 67, 68, 69, 68], [0, 6, 5, 6, 5, 4, 5, 9, 10, 11, 12, 13, 12, 11, 10, 9, 8, 9, 10, 11, 12, 13, 14, 13, 14, 15, 14, 15, 16, 19, 18, 19, 18, 19, 22, 23, 24, 25, 24, 23, 26, 27, 28, 29, 28, 27, 28, 31, 32, 37, 38, 37, 38, 37, 38, 37, 43, 42, 41, 42, 44, 43, 42, 41, 42, 43, 44, 45, 49, 54, 55, 56, 57, 60, 61, 62, 63, 64, 65, 66, 65, 64, 65, 66, 65, 71, 70, 71, 72, 71, 70, 71, 70, 69, 75, 74, 73, 74, 75, 74, 73], [0, 0, 0, 1, 7, 8, 11, 12, 18, 19, 20, 26, 25, 31, 30, 31, 32, 33, 32, 38, 39, 38, 39, 38, 39, 38, 39, 38, 39, 43, 44, 46, 45, 46, 45, 44, 45, 44, 45, 44, 48, 52, 51, 50, 49, 50, 51, 55, 56, 57, 61, 60, 59, 58, 59, 60, 62, 61, 60, 61, 62, 64, 67, 72, 73, 72, 73, 74, 75, 76, 77, 76, 77, 78, 84, 83, 88, 87, 91, 90, 94, 93, 96, 97, 96, 97, 103, 102, 101, 100, 104, 103, 102, 103, 104, 103, 104, 105, 106, 107, 106], [0, 0, 0, 1, 0, 0, 4, 5, 7, 11, 17, 16, 15, 16, 17, 18, 17, 18, 17, 18, 19, 18, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 33, 32, 35, 36, 35, 34, 35, 36, 37, 36, 35, 34, 33, 34, 35, 36, 37, 38, 39, 40, 39, 40, 41, 43, 42, 43, 44, 47, 49, 50, 49, 48, 47, 46, 45, 46, 45, 46, 48, 49, 50, 49, 50, 49, 48, 49, 48, 47, 46, 47, 46, 45, 46, 47, 48, 50, 51, 52, 51, 50, 51, 57, 56, 57, 58, 63, 62, 63], [0, 0, 1, 2, 1, 2, 3, 9, 10, 11, 12, 11, 13, 14, 15, 16, 15, 16, 17, 18, 19, 18, 19, 18, 19, 20, 19, 20, 24, 25, 28, 29, 33, 34, 33, 34, 35, 34, 33, 38, 39, 40, 39, 38, 39, 40, 41, 40, 44, 43, 44, 45, 46, 47, 48, 49, 50, 49, 48, 47, 48, 49, 53, 54, 53, 54, 55, 54, 60, 61, 62, 63, 62, 63, 64, 67, 66, 67, 66, 65, 64, 65, 66, 68, 69, 70, 74, 75, 74, 73, 74, 75, 74, 73, 74, 75, 76, 75, 74, 75, 76], [0, 1, 0, 1, 2, 1, 0, 0, 1, 2, 3, 4, 5, 10, 14, 13, 14, 13, 12, 11, 12, 11, 12, 13, 12, 16, 17, 16, 17, 16, 15, 16, 15, 19, 20, 21, 22, 23, 24, 23, 24, 25, 26, 27, 28, 27, 32, 33, 34, 33, 34, 33, 34, 35, 34, 35, 40, 41, 42, 41, 42, 43, 44, 43, 44, 43, 44, 45, 44, 43, 42, 43, 44, 43, 42, 41, 42, 46, 47, 48, 49, 50, 51, 50, 51, 52, 51, 52, 57, 58, 57, 56, 57, 56, 55, 54, 58, 59, 60, 61, 60], [0, 1, 2, 3, 4, 5, 4, 3, 6, 5, 4, 3, 2, 3, 9, 10, 9, 10, 11, 10, 9, 10, 11, 12, 11, 15, 16, 15, 17, 18, 17, 18, 19, 20, 21, 22, 23, 22, 21, 22, 23, 22, 23, 24, 23, 22, 21, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 33, 34, 35, 36, 37, 38, 37, 36, 42, 43, 44, 43, 42, 41, 45, 46, 50, 49, 55, 56, 57, 61, 62, 61, 60, 61, 62, 63, 64, 63, 69, 70, 69, 73, 74, 73, 74, 73, 79, 85, 86, 85, 86, 87]]

In [4]:

**Visualize all walks**

all\_walks is a list of lists: every sub-list represents a single random walk. If you convert this list of lists to a Numpy array, you can start making interesting plots! matplotlib.pyplot is already imported as plt.

The nested for loop is already coded for you - don't worry about it. For now, focus on the code that comes after this for loop.

**Instructions**

**100 XP**

* Use [**np.array()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.array.html) to convert all\_walks to a Numpy array, np\_aw.
* Try to use [**plt.plot()**](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot) on np\_aw. Also include [**plt.show()**](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show). Does it work out of the box?
* Transpose np\_aw by calling [**np.transpose()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.transpose.html) on np\_aw. Call the result np\_aw\_t. Now every row in np\_all\_walksrepresents the position after 1 throw for the 10 random walks.
* Use [**plt.plot()**](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot) to plot np\_aw\_t; also include a [**plt.show()**](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show). Does it look better this time?

Script.py  
01 # numpy and matplotlib imported, seed set.

02

03 # initialize and populate all\_walks

04 all\_walks = []

05 for i in range(10) :

06 random\_walk = [0]

07 for x in range(100) :

08 step = random\_walk[-1]

09 dice = np.random.randint(1,7)

10 if dice <= 2:

11 step = max(0, step - 1)

12 elif dice <= 5:

13 step = step + 1

14 else:

15 step = step + np.random.randint(1,7)

16 random\_walk.append(step)

17 all\_walks.append(random\_walk)

18

19 # Convert all\_walks to Numpy array: np\_aw

20 np\_aw = np.array(all\_walks)

21

22 # Plot np\_aw and show

23 plt.plot(np\_aw)

24 plt.show()

25

26 # Clear the figure

27 plt.clf()

28

29 # Transpose np\_aw: np\_aw\_t

30 np\_aw\_t = np.transpose(np\_aw)

31

32 # Plot np\_aw\_t and show

33 plt.plot(np\_aw\_t)

34 plt.show()

IPython Shell  
In [1]: # numpy and matplotlib imported, seed set.

# initialize and populate all\_walks

all\_walks = []

for i in range(10) :

random\_walk = [0]

for x in range(100) :

step = random\_walk[-1]

dice = np.random.randint(1,7)

if dice <= 2:

step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

random\_walk.append(step)

all\_walks.append(random\_walk)

# Convert all\_walks to Numpy array: np\_aw

np\_aw = np.array(all\_walks)

# Plot np\_aw and show

plt.plot(np\_aw)

plt.show()

# Clear the figure

plt.clf()

# Transpose np\_aw: np\_aw\_t

np\_aw\_t = np.transpose(np\_aw)

# Plot np\_aw\_t and show

plt.plot(np\_aw\_t)

plt.show()

In [2]:

**Implement clumsiness**

With this neatly written code of yours, changing the number of times the random walk should be simulated is super-easy. You simply update the [**range()**](https://docs.python.org/3/library/functions.html#func-range) function in the top-level for loop.

There's still something we forgot! You're a bit clumsy and you have a 0.1% chance of falling down. That calls for another random number generation. Basically, you can generate a random float between 0and 1. If this value is less than or equal to 0.001, you should reset step to 0.

**Instructions**

**100 XP**

* Change the [**range()**](https://docs.python.org/3/library/functions.html#func-range) function so that the simulation is performed 250 times.
* Finish the if condition so that step is set to 0 if a random float is less or equal to 0.001. Use [**np.random.rand()**](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.rand.html).

Script.py  
01 # numpy and matplotlib imported, seed set

02

03 # Simulate random walk 250 times

04 all\_walks = []

05 for i in range(250) :

06 random\_walk = [0]

07 for x in range(100) :

08 step = random\_walk[-1]

09 dice = np.random.randint(1,7)

10 if dice <= 2:

11 step = max(0, step - 1)

12 elif dice <= 5:

13 step = step + 1

14 else:

15 step = step + np.random.randint(1,7)

16

17 # Implement clumsiness

18 if np.random.rand() <= 0.001 :

19 step = 0

20

21 random\_walk.append(step)

22 all\_walks.append(random\_walk)

23

24 # Create and plot np\_aw\_t

25 np\_aw\_t = np.transpose(np.array(all\_walks))

26 plt.plot(np\_aw\_t)

27 plt.show()

IPython Shell  
In [1]: # numpy and matplotlib imported, seed set

# Simulate random walk 250 times

all\_walks = []

for i in range(250) :

random\_walk = [0]

for x in range(100) :

step = random\_walk[-1]

dice = np.random.randint(1,7)

if dice <= 2:

step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

# Implement clumsiness

if np.random.rand() <= 0.001 :

step = 0

random\_walk.append(step)

all\_walks.append(random\_walk)

# Create and plot np\_aw\_t

np\_aw\_t = np.transpose(np.array(all\_walks))

plt.plot(np\_aw\_t)

plt.show()

In [2]:

**Plot the distribution**

All these fancy visualizations have put us on a sidetrack. We still have to solve the million-dollar problem: *What are the odds that you'll reach 60 steps high on the Empire State Building?*

Basically, you want to know about the end points of all the random walks you've simulated. These end points have a certain distribution that you can visualize with a histogram.

Note that if your code is taking too long to run, you might be plotting a histogram of the wrong data!

**Instructions**

**100 XP**

* To make sure we've got enough simulations, go crazy. Simulate the random walk 500 times.
* From np\_aw\_t, select the last row. This contains the endpoint of all 500 random walks you've simulated. Store this Numpy array as ends.
* Use [**plt.hist()**](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist) to build a histogram of ends. Don't forget [**plt.show()**](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show) to display the plot.

Script.py  
01 # numpy and matplotlib imported, seed set

02

03 # Simulate random walk 500 times

04 all\_walks = []

05 for i in range(500) :

06 random\_walk = [0]

07 for x in range(100) :

08 step = random\_walk[-1]

09 dice = np.random.randint(1,7)

10 if dice <= 2:

11 step = max(0, step - 1)

12 elif dice <= 5:

13 step = step + 1

14 else:

15 step = step + np.random.randint(1,7)

16 if np.random.rand() <= 0.001 :

17 step = 0

18 random\_walk.append(step)

19 all\_walks.append(random\_walk)

20

21 # Create and plot np\_aw\_t

22 np\_aw\_t = np.transpose(np.array(all\_walks))

23

24 # Select last row from np\_aw\_t: ends

25 ends = np\_aw\_t[-1,:]

26

27 # Plot histogram of ends, display plot

28 plt.hist(ends)

29 plt.show()

IPython Shell  
In [1]: # numpy and matplotlib imported, seed set

# Simulate random walk 500 times

all\_walks = []

for i in range(500) :

random\_walk = [0]

for x in range(100) :

step = random\_walk[-1]

dice = np.random.randint(1,7)

if dice <= 2:

step = max(0, step - 1)

elif dice <= 5:

step = step + 1

else:

step = step + np.random.randint(1,7)

if np.random.rand() <= 0.001 :

step = 0

random\_walk.append(step)

all\_walks.append(random\_walk)

# Create and plot np\_aw\_t

np\_aw\_t = np.transpose(np.array(all\_walks))

# Select last row from np\_aw\_t: ends

ends = np\_aw\_t[-1,:]

# Plot histogram of ends, display plot

plt.hist(ends)

plt.show()

In [2]:

**Calculate the odds**

The histogram of the previous exercise was created from a Numpy array ends, that contains 500 integers. Each integer represents the end point of a random walk. To calculate the chance that this end point is greater than or equal to 60, you can count the number of integers in ends that are greater than or equal to 60 and divide that number by 500, the total number of simulations.

Well then, what's the estimated chance that you'll reach 60 steps high if you play this Empire State Building game? The ends array is everything you need; it's available in your Python session so you can make calculations in the IPython Shell.

**Instructions**

**50 XP**

**Instructions**

**50 XP**

**Possible Answers**

* 

48.8%

* 

73.9%

* 

78.4%

* 

95.9%

Submit Answer