Random Walks and Defining Functions
If we take a random walk, will we go places?

**Problem:** Simulate a *random walk* in which a person starts of at point 0 and at each step randomly picks a direction (left or right) and moves 1 step in that direction.

- Take as input a positive integer \( n \) and terminate the simulation when the walk reaches \( n \) or \(-n\).
- Report the *average* number of steps it took for the walk to terminate.
- Do this for various \( n \) and plot the results to get a sense of how rapidly the walk terminates, as a function of \( n \).
The **random** module

- Programs for games and simulation use *randomization* extensively.

- In games, you want to add an element of randomness to the obstacles or adversaries.

- In simulations (e.g., traffic simulation) you want to introduce actors into your simulation according to certain probability distribution.
Some functions in the `random` module

- `random.randint(a, b)`: return a random integer \( N \) such that \( a \leq N \leq b \).

- `random.random()`: Return the next random floating point number in the range \([0.0, 1.0)\).

- `random.uniform(a, b)`: Return a random floating point number \( N \) such that \( a \leq N \leq b \) for \( a \leq b \) and \( b \leq N \leq a \) for \( b < a \).
Problem: Write a program that takes as input a positive integer \( n \) and simulates \( n \) rolls of two six-sided dice. The program should report the number of times 7 appears as the sum of the outcomes of the two dice rolls.
# Programmer: Sriram Pemmaraju
# Feb 8th, 2015

# This program simulates the roll of two six-sided dice
# as many times as specified by the input. Then the program
# outputs the number of times 7 shows up as the sum of the two
# dice rolls

import random
n = int(input("Enter the number of times you want the dice rolled: "))

counter = 0 # keeps track of the number of rolls
numSevens = 0 # keeps track of the number of sevens

# while-loop that simulates the roll of two six-sided dice n times
while counter < n:
    # Roll two six-sided dice and compute the sum of the outcomes
    sumRolls = random.randint(1, 6) + random.randint(1, 6)

    # if sum is seven then update a counter called numSevens
    if sumRolls == 7:
        numSevens = numSevens + 1

    counter = counter + 1

print("The number of sevens is", numSevens)
Taking a single random step

import random

# Version 0. This program starts off a person at 0 and moves
# her one step to the left or right, at random.

location = 0
step = random.randint(0, 1)  # returns 0 or 1, each with prob. 1/2
if step == 0:
    step = -1
location = location + step
print(location)
import random

# Programmer: Sriram Pemmaraju
# Date: Feb 8, 2015

# Version 1: moves the person at random, one step at a time, left or right,
# until the person reaches a barrier n or -n. Outputs the number of steps
# it took to reach the barrier

location = 0 # tracks the person's current location
n = 10 # value of the barrier
length = 0 # tracks the length of the random walk

# This moves the person until she reaches the barrier at n or -n
while abs(location) < n:
    step = random.randint(0, 1) # returns 0 or 1, each with prob. 1/2

    # Adjusts the random number to be either -1 or +1
    if step == 0:
        step = -1
    location = location + step # updates location
    length = length + 1

print(length)
What more is there to do?

There are two more things we need to do to solve our problem:

1. Find the average length of a walk, for a particular value $n$ of the barrier. We have to decide how many runs to take the average over.

2. Repeat this for various values of $n$ and try to understand the trend.

We need a loop around our current code to do (1) and another loop around that code to do (2).
Defining a function

• Things have become complicated enough that we need to reorganize our code using functions.

• The plan is to define a function called randomWalk that takes \( n \) (the barrier distance) as an argument and returns the length of a simulated random walk.

• We can then just call this function from the main part of the program.
The function `randomWalk`  

# This function takes the barrier distance `n` as an argument, simulates  
# the random walk until it hits the barrier (`n` or `-n`), and returns the  
# length of the random walk

```python
def randomWalk( n ):
    location = 0  # tracks the location of the person
    length = 0  # tracks the length of the random walk

    # Loop terminates when the location reaches `n` or `-n`
    while abs(location) != n:
        step = random.randint(0, 1)  # returns 0 or 1, each with prob. 1/2
        if step == 0:
            step = -1
        location = location + step
        length = length + 1

    return length
```
Notes about this function

- The first line of the function:
  ```python
def randomWalk(n)
  ```

- The body of the function is indented.
- It is as though \( n \) is input to the function.
- A function can have one or more arguments.
- The last line of the function is usually a return:
  ```python
  return length
  ```
n = input("Enter a positive integer: ")
print(randomWalk(n))

- `randomWalk(n)` is a call to the function `randomWalk` providing it the number `n` that the user as input as an argument.
- In order to execute the print statement, the function call `randomWalk(n)` needs to be executed first.
- This means that “control” is transferred to the function and we start executing the function starting with its first line.
- The value that the function returns essentially replaces the function call.
Averaging over 100 simulations

```python
n = input("Enter a positive integer: ")
count = 0  # tracks the number of times the walk is repeated
sum = 0   # sum of the lengths of the walk; needed for average
while count < 100:
    sum = sum + randomWalk(n)
    count = count + 1

print sum/100
```

**Function call.** The function is called 100 times.
import random

# Programmer: Sriram Pemmaraju
# Date: Feb 8, 2015
# Version 2: moves the person at random, one step at a time, left or right,
# until the person reaches a barrier n or -n. Outputs the number of steps
# it took to reach the barrier

# This function takes in the value of the barrier, simulates a random
# walk that terminates on reaching the barrier, and returns the length
# of the simulated random walk
def randomWalk(n):
    location = 0 # tracks the person's current location
    length = 0 # tracks the length of the random walk

    # This moves the person until she reaches the barrier at n or -n
    while abs(location) != n:
        step = random.randint(0, 1) # returns 0 or 1, each with prob. 1/2

        # Adjusts the random number to be either -1 or +1
        if step == 0:
            step = -1

        location = location + step # updates location
        length = length + 1

    return length

# This is the main part of the program (i.e., outside the function)
n = input("Enter the value of the barrier (a positive integer): ")
sum = 0 # track the total length of all simulated random walks
counter = 0
# The simulation is repeated 100 times
while counter < 100:
    sum = sum + randomWalk(n)
    counter = counter + 1

print(sum/100)
Notes about programs that contain function definitions

- The first line of the main program is the first line of code that is executed.

- The function is only executed when it is called. In this code, the function is called 100 times and is therefore executed 100 times.

- In general a program will contain many function definitions followed by a main program.

- Functions may call each other.

- Typically the function will return a value. The returned value replaces the function call.
Making another function

# This function repeats a random walk with barrier n as many times
# as specified by the argument numRepitions and returns the length
# of the walk, averaged over all the repititions

def manyRandomWalks(n, numRepititions):
    count = 0  # tracks the number of times the walk is repeated
    sum = 0    # sum of the lengths of the walk; needed for average

    # Repeats the random walk as many times as specified by numRepititions
    while count < numRepititions:
        sum = sum + randomWalk(n)
        count = count + 1

    return sum/numRepititions
n = input("Enter a positive integer: ")
print(manyRandomWalks(n, 100))

- The function call needs to supply arguments in the correct order, i.e., in the order specified in the function definition.

- Names in the function call have nothing to do with names in the function definition. We could have written

  m = input("Enter a positive integer: ")
  print manyRandomWalks(m, 100)

And the value of `m` and the value `100` would be used for `n` and `numRepetitions` in the function.
Trying this out for different barrier values

m = 10  # tracks the value of the barrier
# m travels through 10, 20, ..., 100 in this loop and we compute and print the
# average walk length for each m
while m <= 100:
    print manyRandomWalks(m, 100)
    m = m + 10
Sample output

Length of random walk

112.86
376.4
827.6
1628.04
2570.6
3594.2
4616.14
6035.6
8596.58
10948.58

1 2 3 4 5 6 7 8 9 10