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 off at point 0 and at each step randomly picks a direction (left or right) and moves 1 step in that direction.
- Take 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 \( n \).
Taking a single random step

import random

# Version 1. 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

# Version 2. This program starts off a person at 0 and moves
# her left or right, at random one step at a time until she reaches
# the "barrier" at n or -n.

n = input("Enter a positive integer: ")
location = 0

# 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

print location
import random

# Version 3. This program starts off a person at 0 and moves
# her left or right, at random one step at a time until she reaches
# the "barrier" at n or - n. It outputs the length of the walk.

n = input("Enter a positive integer: ")
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

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

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 float(sum)/100
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
    while count < numRepititions:
        sum = sum + randomWalk(n)
        count = count + 1
    return float(sum)/100
The rest of the program

\[ n = \text{input}(\text{"Enter a positive integer: ")}
\text{print manyRandomWalks}(n, 100) \]

- 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

  \[ m = \text{input}(\text{"Enter a positive integer: ")}
  \text{print manyRandomWalks}(m, 100) \]

And the value of \( m \) and the value 100 would be used for \( n \) and \text{numRepititions} in the function.
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
Length of random walk

112.86
376.4
827.6
1628.04
2570.6
3594.2
4616.14
6035.6
8596.58
10948.58