14.4. Randomly Walking Turtles

Suppose we want to entertain ourselves by watching a turtle wander around randomly inside the screen. When we run the program we want the turtle and program to behave in the following way:

1. The turtle begins in the center of the screen.
2. Flip a coin. If it's heads then turn to the left 90 degrees. If it's tails then turn to the right 90 degrees.
3. Take 50 steps forward.
4. If the turtle has moved outside the screen then stop, otherwise go back to step 2 and repeat.

Notice that we cannot predict how many times the turtle will need to flip the coin before it wanders out of the screen, so we can't use a for loop in this case. In fact, although very unlikely, this program might never end, that is why we call this indefinite iteration.

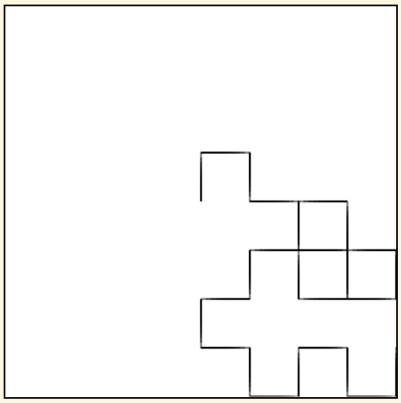
So based on the problem description above, we can outline a program as follows:

```
create a window and a turtle

while the turtle is still in the window:
    generate a random number between 0 and 1
    if the number == 0 (heads):
        turn left
    else:
        turn right
    move the turtle forward 50
```

Now, probably the only thing that seems a bit confusing to you is the part about whether or not the turtle is still in the screen. But this is the nice thing about programming, we can delay the tough stuff and get *something* in our program working right away. The way we are going to do this is to delegate the work of deciding whether the turtle is still in the screen or not to a boolean function. Let's call this boolean function `isInScreen` We can write a very simple version of this boolean function by having it always return `True`, or by having it decide randomly, the point is to have it do something simple so that we can focus on the parts we already know how to do well and get them working. Since having it always return True would not be a good idea we will write our version to decide randomly. Let's say that there is a 90% chance the turtle is still in the window and 10% that the turtle has escaped.

Save & Run Original - 1 of 1

```
import random
import turtle


def isInScreen(w, t):
    if random.random() > 0.1:
        return True
    else:
        return False


t = turtle.Turtle()
wn = turtle.Screen()

t.shape('turtle')
```



Activity: 1 -- ActiveCode (ac14_4_1)

Now we have a working program that draws a random walk of our turtle that has a 90% chance of staying on the screen. We are in a good position, because a large part of our program is working and we can focus on the next bit of work – deciding whether the turtle is inside the screen boundaries or not.

We can find out the width and the height of the screen using the `window_width` and `window_height` methods of the screen object. However, remember that the turtle starts at position 0,0 in the middle of the screen. So we never want the turtle to go farther right than width/2 or farther left than negative width/2. We never want the turtle to go further up than height/2 or further down than negative height/2. Once we know what the boundaries are we can use some conditionals to check the turtle position against the boundaries and return `False` if the turtle is outside or `True` if the turtle is inside.

Once we have computed our boundaries we can get the current position of the turtle and then use conditionals to decide. Here is one implementation:

```
def isInScreen(wn,t):
    leftBound = -(wn.window_width() / 2)
    rightBound = wn.window_width() / 2
    topBound = wn.window_height() / 2
```

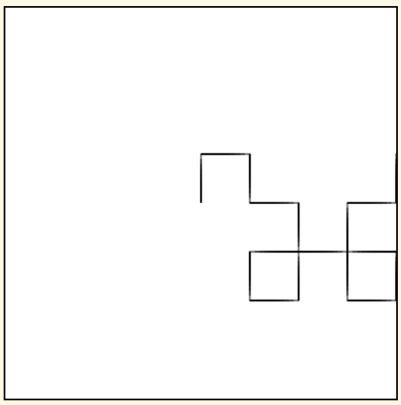
```
    bottomBound = -(wn.window_height() / 2)

    turtleX = t.xcor()
    turtleY = t.ycor()

    stillIn = True
    if turtleX > rightBound or turtleX < leftBound:
        stillIn = False
    if turtleY > topBound or turtleY < bottomBound:
        stillIn = False

    return stillIn
```

There are lots of ways that the conditional could be written. In this case we have given `stillIn` the default value of `True` and use two `if` statements to possibly set the value to `False`. You could rewrite this to use nested conditionals or `elif` statements and set `stillIn` to `True` in an else clause.

Here is the full version of our random walk program.

Save & Run Original - 1 of 1

```
1  import random
2  import turtle
3
4  def isInScreen(w,t):
5      leftBound = - w.window_width() / 2
6      rightBound = w.window_width() / 2
7      topBound = w.window_height() / 2
8      bottomBound = -w.window_height() / 2
9
10     turtleX = t.xcor()
11     turtleY = t.ycor()
12
13     stillIn = True
14     if turtleX > rightBound or turtleX < leftBound:
15         stillIn = False
```



Activity: 2 -- ActiveCode (ac14_4_2)

We could have written this program without using a boolean function. You might want to try to rewrite it using a complex condition on the while statement. However, using a boolean function makes the program much more readable and easier to understand. It also gives us another tool to use if this was a larger program and we needed to have a check for whether the turtle was still in the screen in another part of the program. Another advantage is that if you ever need to write a similar program, you can reuse this function with confidence the next time you need it. Breaking up this program into a couple of parts is another example of functional decomposition.

Check your understanding

moreiter-4-1: In the random walk program in this section, what does the isInScreen function do?

- (•) A. Returns True if the turtle is still on the screen and False if the turtle is no longer on the screen.
- ( ) B. Uses a while loop to move the turtle randomly until it goes off the screen.
- ( ) C. Turns the turtle right or left at random and moves the turtle forward 50.
- ( ) D. Calculates and returns the position of the turtle in the window.

Check me Compare me

✔ The isInScreen function computes the boolean test of whether the turtle is still in the window. It makes the condition of the while loop in the main part of the code simpler.

Activity: 3 -- Multiple Choice (question14_4_1)

You have attempted 4 of 3 activities on this page

14.3. The Listener Loop">
e Listener Loop">
14.5. Break and Continue">
✔ Completed. Well Done!
14.5. Break and Continue">Next Section - 14.5. Break and Continue

| Back to top