CSCI 1100 — Computer Science 1
Homework 5
Wandering Turtle

Overview

This homework is worth **90 points** total toward your overall homework grade, and is due Thursday, March 24, 2016 at 11:59:59 pm. You will write and debug two versions of one program, with the first version solving a more restricted problem than the second.

See the HW 3 assignment description for discussions of both academic integrity issues and grading criteria. The homework submission server URL is below for your convenience:

```
https://submit.cs.rpi.edu/index.php?course=csci1100
```

The two versions of your program must be named.

```
hw5_part1.py
hw5_part2.py
```

This assignment builds heavily on material from HW 3, Lab 5, and the random walk example from Lectures 11 and 12. Feel free to make use of your code from these assignments and the code we provided.

Part 1: Wandering Turtle

A turtle is placed in the middle of a grid that is M rows tall and N columns wide. These values are read into the program by asking the user. The user must also be asked for a probability p that must be great than 0.0 and less than 1.0, but will generally be relatively small. The reason for the probability is explained below. You may assume all input is correct. The upper left corner of the grid is location $(0, 0)$, while the bottom right corner is location $(M-1, N-1)$. The turtle starts out at location $(M/2, N/2)$, but its initial direction will be determined randomly, as described below.

The program must simulate random movements of the turtle. As in HW 3, Part 2, the turtle can only move in a straight line to the right (increasing x), down (increasing y), left (decreasing x), and up (decreasing y). In each time unit, the turtle will either move a **single step** in its current direction or it will turn clockwise 90 degrees (so that right becomes down, down becomes left, etc.). The decision is determined by calling the function

```
random.random()
```

If the value returned is less than p the turtle should turn 90 degrees. Otherwise, the turtle should take a single step. The turtle should continue to make these random steps and turns until 250 time units have passed or until it has fallen off the grid — its row position is -1 or `M` or its column position is -1 or `N` — whichever comes first.

To solve part 1 your program must report the position and direction of the turtle every 20 steps, and then when the simulation of the turtle ends you must output the final position and direction of the turtle.

Important Details:

1. In order for everyone to have the same output we must do what is called *seed* the random number generator. After you read the user input, but before the first time you call one of the `random` functions, you must include the following code

```
seed_value = 10*M + N
random.seed(seed_value)
```

   Every time you give the same value of `M` and `N` and create the seed this way **and** repeat the same calls to the functions of `random` you will get the same random values. (To see interesting behavior while testing your program, you might want to comment out the call to the `seed` function, but put it back in before you submit!)

2. You **MUST** write and use

```
def move_turtle( row, col, dir, turn_prob ):
    '''
    row, col is the current position of the turtle

    dir is the current direction of the turtle, represented as a
    string ('right', 'down', 'left', 'up'),

    turn_prob is the probability p, above, that the turtle will make a turn
    '''
```

   which moves the turtle for one time unit, either turning it or taking a step. The function must return a tuple containing in order, the new row position, the new column position, and the new direction. Your main code must then call this function in some kind of loop and use the results to decide when to quit and what to output. Do **not** call the `random.seed` function inside the `move_turtle` function.

3. The initial direction for the turtle will be determined randomly using the following code:

```
dir_list = ['right', 'down', 'left', 'up']
rand_index = random.randint(0,3)
dir = dir_list[rand_index]
```

   This call must be made **after** the call to `random.seed`, but before the first call to `move_turtle`. Output this initial direction as soon as you determine it.

Additional Notes:

1. The order of operations inside the main while loop should be (a) incrementing the time step counter, (b) making the step (by calling `move_turtle`), (c) printing the turtle's position and direction if the current time step is a multiple of 20, but not if the turtle has fallen off the board.

2. The final output will depend on whether or not the turtle has fallen off the board or run out of time. If the turtle is still on the board, output the final number of time steps, the final position, and the final direction. If the turtle fell off, output the number of time steps, output which side it fell off (left, right, top or bottom), and output what row or column it was in when it fell off. See examples below for details.

Example 1:

```
Enter the integer number of rows => 16
16
Enter the integer number of cols => 20
20
Enter the turtle's turn probability (< 1.0) => 0.3
0.3
Initial direction: right
Time step 20: position (5,15) direction left.
Time step 40: position (5,7) direction left.
After 47 time steps the turtle fell off the top in column 7.
```

Example 2:

```
Enter the integer number of rows => 30
30
Enter the integer number of cols => 40
40
Enter the turtle's turn probability (< 1.0) => 0.33
0.33
Initial direction: down
Time step 20: position (15,19) direction right.
Time step 40: position (21,18) direction down.
Time step 60: position (13,19) direction left.
Time step 80: position (15,11) direction left.
Time step 100: position (8,11) direction up.
Time step 120: position (7,10) direction up.
Time step 140: position (7,12) direction up.
Time step 160: position (8,5) direction down.
Time step 180: position (17,5) direction left.
Time step 200: position (16,0) direction left.
Time step 220: position (14,1) direction up.
After 238 time steps the turtle fell off the left in row 13.
```

Example 3:

```
Enter the integer number of rows => 40
40
Enter the integer number of cols => 46
46
Enter the turtle's turn probability (< 1.0) => 0.35
0.35
Initial direction: up
Time step 20: position (19,24) direction down.
Time step 40: position (18,18) direction left.
Time step 60: position (20,31) direction down.
Time step 80: position (20,33) direction up.
Time step 100: position (26,32) direction right.
Time step 120: position (26,31) direction up.
Time step 140: position (25,40) direction down.
Time step 160: position (24,34) direction left.
Time step 180: position (22,31) direction up.
Time step 200: position (26,26) direction left.
```

```
Time step 220: position (29,17) direction left.
Time step 240: position (32,11) direction down.
After 250 time steps the turtle ended at position (32,9) and direction down.
```

Part 2: Gathering Data About the Wandering Turtle

Thus far, we have only considered a single case of running the turtle, but simulations are typically run over and over again. In this second part of the assignment your program will need to repeat the simulation a user-specified number of times, and output several summary statistics:

1. The min, max and average number of time steps before the turtle falls off the grid or reaches the upper bound on the number of time steps (still 250). Note that if the turtle reaches the upper bound on the number of times steps without falling off, this upper bound will become the maximum.
2. The percentage of the simulations that end with the turtle falling off the grid.
3. An output of the grid showing the number of time steps that each grid position is occupied.
4. The position the turtle occupies most often, along with the percentage of time units that the turtle is in that position.

A Counting Grid: You must create a list of lists of counts for the number of times the turtle occupies each cell of the grid. Here are two examples to help you. The first example shows an easy way to initialize the grid to have `M` rows and `N` columns:

```
    count_grid = []
    for i in range(M):
        count_grid.append( [0]*N )
```

The second example illustrates counting the number of occurrences of the numbers 0 through 9 using the random number generator (without using the `seed` function):

```
import random

num_trials = 2500
counts = [0]*10
for i in range(num_trials):
    digit = random.randint(0,9)
    counts[digit] += 1

print 'Occurrences and percentages:'
for i in range(10):
    print "%1d: %4d %4.1f" %(i, counts[i], 100.0*counts[i]/num_trials)
```

Important details:

- You must **make** use of the `move_turtle` function from part 1. You can copy and paste it into your `hw5_part2.py` file.
- Next you **must** write and test a function called

```
def run_one_simulation( grid_count, turn_prob ):
    '''
    returns a tuple containing the number of time steps, and whether
    or not the turtle fell off.
    '''
```

that starts the turtle in the center of the grid, and runs one full simulation of the turtle until it reaches the maximum number of time steps or the turtle falls off the grid. At the end of each time step, supposing that you have recorded the position of the turtle in the variables `row` and `col`, your code should increment `grid_count[row][col]`. You **must** have this function in your program, but you may change its parameters as you wish. For example, you could pass `M` and `N` — the number of rows and columns — although you do not need to because you can compute the values from the `grid_count` list of lists.

- Do **not** call `random.seed` from inside `run_one_simulation`. It will cause your the random number generate to start over and your results will be the same for each simulation.
- Each time you call `run_one_simulation` the turtle should start in the center of the grid and with a *randomly chosen* direction.
- We *suggest* that you write a function to extract and print the statistics of the grid in order to avoid cluttering the code in the main body of the program.

Finally, here is an example that illustrates the output and formatting we are expecting.

```
Enter the integer number of rows => 18
18
Enter the integer number of cols => 16
16
Enter the turtle's turn probability (< 1.0) => 0.35
0.35
Enter the number of simulations to run => 300
300

Completed simulation.
Min time to end: 8.
Max time to end: 250.
Average time to end: 57.1.
Percentage that ended when the turtle fell off is 99.7.

Grid of counts:
   1   2   3   9  12  19  31  21  40  18  27  15   9   6  14   6
   6   8  12  23  10  22  30  29  47  22  21  13  14   7  12   6
   9  14  21  25  32  40  48  44  63  43  54  37  33  19  14  10
  18  11  24  31  33  56  55  47  79  60  62  48  41  29  16  13
  15  26  34  28  31  63  71  68 116  91  75  42  39  32  28  18
  19  21  39  37  52  80 101 104 119  89  76  54  52  51  51  44
  26  31  53  59  70  94 108 113 173 115 104  74  65  48  50  35
  22  31  45  59  81  84 138 152 215 144  98 120  86  59  37  29
  32  35  52  95 103 132 149 184 278 174 136 115  97  50  54  47
  32  41  63 121 130 169 233 347 666 244 185 149 139  59  63  58
  31  33  43  88 106 120 159 173 256 123 117 100  78  41  45  30
```

```
  38  38  48  76  89 109 125 151 201 111 116  85  60  44  41  23
  24  22  33  54  70  75  83 111 170 103 105  70  60  48  26  20
  15  17  32  41  49  78  81  91 101  82  85  53  38  36  27  15
   8   8  22  39  35  45  81  80  98  73  46  34  36  22  22  13
  10  19  10  24  37  41  34  51  58  44  36  23  21  13  10  11
   2  10   5  27  28  40  35  63  52  55  45  35  39  17   9   5
   6   9   5  20  19  22  22  37  36  29  19  16  21  20  10   3
Max count occurs at (9,8) with percentage 3.89.
```

In terms of formatting, each of the output values in the grid is formatted with `%4d` and appended to a string that stores the output for the given row.