Computer Science Activity—Random Walker

ASSIGNMENT OVERVIEW

In this assignment you'll be creating a creating a graphical simulation of a "random walker" who moves around in random directions in a 2-dimensional world. The random walker begins at the origin (or center of the screen) and each "turn" moves to a randomly selected adjacent square. This process continues until the random walker returns to the origin.

This assignment is worth 50 points and is due on the *crashwhite.polytechnic.org* server at 23:59:59 on the date given in class.

BACKGROUND

Simulating a random walker on the computer is a relatively straightforward process, although there are multiple ways to do it. Your simulation of this process will probably follow one of these strategies:

- **Non-graphical Walker**
  Set the walker's original x- and y-coordinates to 0, 0, and then randomly change one or both of those values by 1 until they return to the values 0, 0. This is the essence of what all versions of the program will be doing, although it's not very interesting to look at.
- **Graphical Text-based Walker (no path)**
  Keep track of the walker's coordinates as above, but every turn, print out the "world" on-screen. Each space in the world is represented by a ".", and the walker itself is represented by an "@". If you have the ability to clear the screen or terminal window, a crude animation effect can be seen as the walker attempts to return to the origin.
- **Graphical Text-based Walker (with path)**
  It might be interesting to see a record of the path followed by the walker. In this strategy, a two-dimensional array (a "list of lists" in Python) can be used to keep a record of where the walker is ("@") and where the walker has been ("+"). (A "sparse array" might also be used to track the walker's progress.)
- **Graphical pixel-based Walker (with path)**
  Using Python's built-in Turtle graphics, the Processing platform, or some other graphics strategy, a larger world can be display on the screen (800 pixels across by 600 pixels high, for example), with the walker represented by a colored pixel or shape, and its path represented by a different colored pixel or shape.

PROGRAM SPECIFICATION

Create a Python program, with associated functions, that:

a. creates a 2-dimensional list of lists on which the simulation will run (optional)
b. initializes variables that indicate the location of the random walker
c. updates the walker's location according to random values
d. keeps track of the number of steps taken
e. displays the walker's position and steps taken (using text, text-based graphical display, pixel-based graphics)
f. continues until the walker returns to its starting position

DELIVERABLES
`random_walker.py`

To submit your assignment for grading, copy your file to your directory in `/home/studentID/forInstructor/` at *crashwhite.polytechnic.org* before the deadline.

ASSIGNMENT NOTES

- This program will probably consist of a main program and 3-4 functions. You'll definitely have a `main()` function, and possibly a `create_grid()` function, a `print_grid()` function, and a `move()` function.
- The two-dimensional grid used in this assignment can be created in Python as a "list of lists." For information on how to create this data structure, you might consult the "Getting Started" section below.
- The simulation will probably have one of two different strategies for changing the walker's location.
    - In a simple "North-South-East-West" strategy, a random direction is selected and the corresponding `x`- or `y`-value changed accordingly.

```
import random
dir = random.randrange(4)
if dir == 0:
    x = x + 1
elif dir == 1:
    y = y + 1
elif dir == 2:
    x = x - 1
else:
    y = y - 1
```

    - A more complex 8-direction strategy can be used by potentially changing both `x`- and `y`-values at the same time, or by using simple trigonometry to determine the next location:

```
import random
import math

angle = random.random() * 2 * math.pi
x = x + math.cos(angle)
y = y + math.sin(angle)
```

- If using the text-based display, you'll want to clear the screen between the display of successive generations. In Apple's macOS you can use this code for that purpose:

```
import os           # at the beginning of the program

os.system("clear")  # any time you want to clear the screen, usually
                    # just before drawing the grid
```

- If you find that the simulation runs faster than your computer display can refresh, producing odd flickering effects., you can insert a brief delay between screen refreshes. You can use this code to introduce a pause for some number of seconds:

```
import time

time.sleep(0.1)     # delays program for 0.1 seconds
```

GETTING STARTED

1. With paper and pencil, and perhaps in collaboration with a partner, sketch out the functions that you'll be using in this simulation, perhaps using pseudocode.

2. In a text editor, begin writing a `random_walker.py` program that you will submit for this assignment.

3. Write a `create_grid()` function that you will use to construct and manipulate the initial board.

4. The 2-dimensional grid that is used for the walker's virtual world is constructed as follows (code and comments on the left, explanation on the right).

```
# set up empty grid as a list which
# will contain the row
grid = []

# identify number of rows, columns
rows = 20
cols = 40

# for each row in our empty grid,
# create a list that will include
# all the columns
for row in range(rows):
    grid.append([])

    # for each row list, append an
    # item that will represent the
    # value at that column
    for col in range(cols):
        grid[row].append('.')
```

Here's what's actually happening in the computer.

```
grid = []
```

```
rows = 20
rows = 40
```

```
row = 0
grid[ [] ]
```

We've appended an empty list to our grid list.

```
col = 0
grid[ ['.'] ]
```

We've appended an item into the list at `grid[0]`.

The next time trough the `col` loop we'll have:

```
col = 1
grid[ ['.', '.'] ]
```

and

```
col = 2
grid[ ['.', '.', '.'] ]
```

and so on. When it's time for the `row` loop to repeat again:

```
row = 1
grid[ ['.', '.', '.'], [] ]
```

Note that we've appended a new column list to our rows. Now the row list begins again...

```
col = 0
grid[ ['.', '.', '.'], ['.'] ]
col = 1
grid[ ['.', '.', '.'], ['.', '.'] ]
```

and so on...

5. Write a `print_grid()` function that will display the world. Once the two-dimensional "list of lists has been created, displaying that list on screen is relatively straightforward.

   ```
   for row in range(len(grid)):
       for col in range(len(grid[0]):
           print(grid[row][col],end='')     # Keep every column on same row
       print()                               # After printing all columns,
                                             # space down to next row
   ```

6. Write a `move()` function that will calculate the next location of the random walker.

7. *Test each bit of code as you go*, making sure that one piece works before you proceed on to the next section. You'll repeatedly run through this edit-compile-test, edit-compile-test process to progressively find bugs and fix them *while* you're writing your program, not afterwards.

8. Save your program from time to time, and once a day or so, save a backup copy of it on another device or machine: a flash drive, your home folder on the *crashwhite.polytechnic.org* server, etc.

9. When your program is completed (but before the deadline), copy a final archived package to the server as indicated above.

REFERENCES

https://en.wikipedia.org/wiki/Random_walk
https://en.wikipedia.org/wiki/Random_walker_algorithm

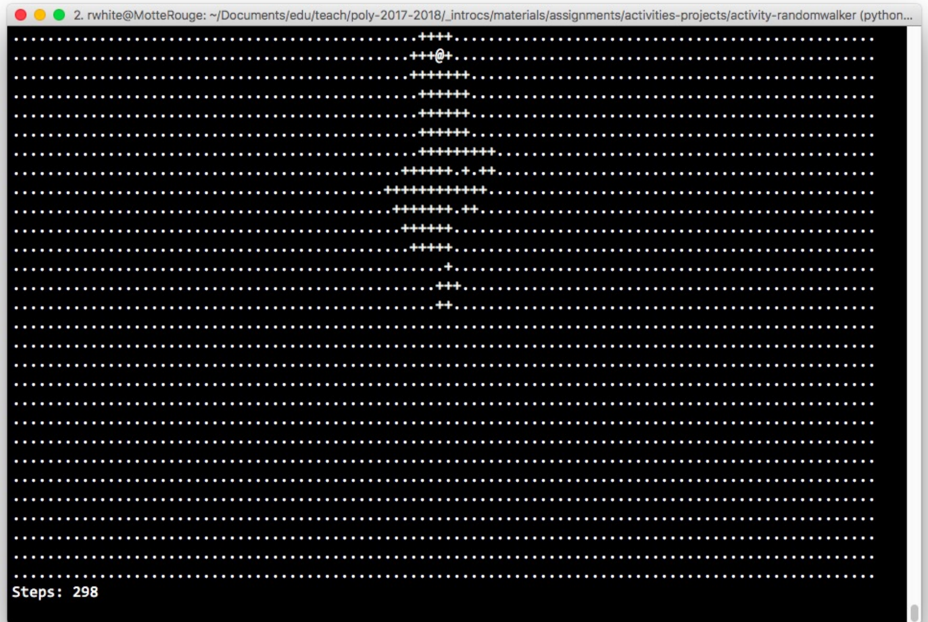
QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. What is the mathematical definition for a "Random Walker?" How does it vary from what we're doing?

2. We've modeled a two-dimensional random walker in this activity. What would a 3-dimensional random walker look like? Or a 1-dimensional random walker? Would those be easier to code than this one, or more difficult to code?

SAMPLE INTERACTIONS

Random Walker who doesn't wrap when wandering



Random Walker who wraps when wandering