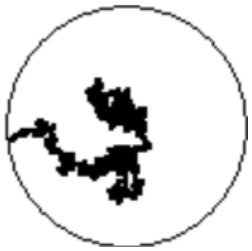
# CS 115, Autumn 2021
# Programming Project #5: Brownian Motion / Random Walk (20 points)
**Due: Tuesday, November 2, 2021, 11:30 PM**

## Program Description:

This assignment focuses on while loops, random numbers, and planning more complex programs. Turn in files named `planning.pdf`, `random_walk_console.py`, and `random_walk.py`.

```
<< your introduction message here >>

Radius? 50
I escaped in 2077 move(s).
```

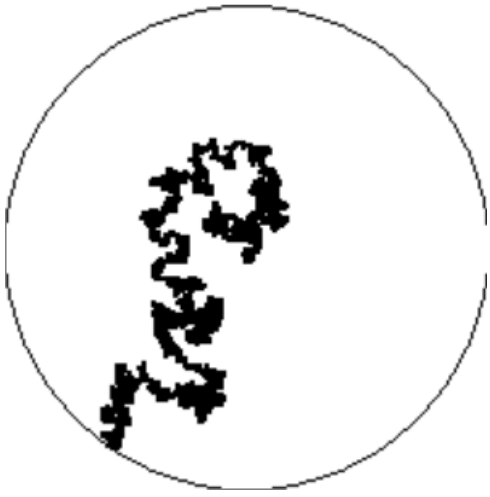


```
<< your introduction message here >>

Radius? 50
I escaped in 34 move(s).
```

```
<< your introduction message here >>

Radius? 100
I escaped in 4976 move(s).
```



be drawn in a `drawing_panel` of size 201.

### Concept:

Brownian Motion describes the motion of particles in a liquid or gas. For example, a piece of pollen in water or a particle of pollution in the air. These particles appear to move randomly as they are constantly being bombarded by other molecules. Since this bombardment is so frequent and constantly changes due to so many different variables, we are unable to create a scientific model based on these variables. Therefore, we use a statistical model based on randomness to allow us to model this phenomenon.

The statistical model we will use in this project is based on the concept of a *random walk.* For the purposes of this assignment, we will assume the particle has an equal probability of moving in each direction.

In this program you will simulate the movement of one particle. The particle's position will be represented as a 1 by 1 black rectangle. Move the particle in random directions and every time the particle moves, mark its new position on the screen until it has moved a certain distance away from its starting location.

Read more Brownian Motion here:

- https://simple.wikipedia.org/wiki/Brownian_motion

Read more about interesting mathematical properties of random walks here:

- http://en.wikipedia.org/wiki/Random_walk

### Program Description:

The program begins with an introduction of your choice. Print anything you like, but keep it to a reasonable number of lines, no offensive remarks, no reading data from the user, ninfinite loops, etc.

Next, your program should perform a random walk. A walk begins by asking the user for the radius (in pixels) of the walk area. If the user enters a number smaller than 1, re-prompt them. Repeat re-prompting until they enter a number greater than 0. After the user inputs an acceptable radius, a `drawing_panel` with a white background and a black-outlined circle with this radius should appear. The panel's size should be exactly enough to contain this circle: one pixel more than twice the radius. For example, a circle of radius 100 should

Next the `drawing_panel` should animate the steps of the random walk. The walker is a single black pixel that begins in the center of the circle. Every 5ms, the walker randomly moves its position *up*, *down*, *left*, or *right* 1 pixel. The walker should choose between these four choices randomly with equal probability. The walk ends when the walker reaches the perimeter of the circle (when it walks so far that its direct distance from the center is greater than or equal to the circle's radius). When the walk ends, the program should print how many moves were made.

The screenshots above demonstrate your program's behavior. Yours will generate different random moves, but the general appearance of the drawn output should be similar

## Implementation Guidelines:

***The repetition and animation in this program must be done using `while` loops***. This program is more complex than the others you have written so far in this class so, to make it easier, we require that you develop this program in stages:

**In `planning.pdf`:**

1. List what pieces you will need (for example, `while` loop, `if`, `if/else`, `if/elif/elif/else`, `global` variable, etc).
2. Write out pseudocode or a flow chart for the program. Make sure that you include all the pieces that you listed.

Make sure that you separate these two pieces in your file so that I can tell which is which.

**In `random_walk_console.py`:**

3. Create a file called `random_walk_console.py`. In this file, work on a text-only version of the program before attempting to draw your walk on the `drawing_panel`.
4. In a console program it will be overwhelming if you use a big radius as you will get a huge amount of output. Instead, use a very small radius, like 3, when you are initially testing your program. This will make it easier to see if something is going wrong.
5. Since you won't be able to draw the random walk, print out the walker's position as shown to the right. This will help you check your work and make sure you are moving correctly and exiting your loop correctly.

**Example output**

```
Radius = 3
x=3, y=2, moves=1
x=2, y=2, moves=2
x=2, y=1, moves=3
x=1, y=1, moves=4
x=1, y=2, moves=5
x=1, y=1, moves=6
x=0, y=1, moves=7
I escaped in 7 move(s).
```

6. Once you have your loop stopping at the right time, add in a count of moves and print it out to check that it is correct.

**In `random_walk.py`:**

7. Create a file called `random_walk_console.py`.
8. Copy your code from `random_walk_console.py` into `random_walk.py`.
9. Alter your code to open a `drawing_panel` and draw the surrounding circle on it between reading from the user and the start of the walk.
10. Alter your `while` loop to color the pixel at the walker's current location black instead of printing out the x, y coordinates of that location. Color a pixel by drawing a `1` by `1` rectangle. Make the walker animate by adding `sleep`.
11. Add the other statistics.

## Hints:

You will need to examine distances between points (x, y locations) to determine whether the walker has exited the circle. The formula to compute the distance between two points is to take the square root of the sum of the squares of the differences in x and y between the two points. This can be written as $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. For example, the distance between the points (11, 4) and (5, 7) is $\sqrt{(11-5)^2 + (4-7)^2}$ or roughly 6.71.

Produce randomness using `randint` as described in class. `randint` produces random integers. These can be mapped to arbitrary random choices. For example, to randomly choose a color between red, yellow, and blue, pick a random integer from 0 through 2, and consider 0 to be red, 1 to be yellow, and 2 to be blue.

## Stylistic Guidelines:

***You must use a `while` loop to produce the walk animation.*** You will receive an extremely low score if you produce animation using `panel.animate(`**`ms`**`)` the way we have so far this quarter.

All of your code must be written in functions. The function that starts your program should be named `main` and be called at the bottom of your file. You are welcome to add any other functions that seem useful to you.

You must include a **constant** representing the size of step the walker takes. In your program this will be `1`. However, we should be able to change this constant's value and have your program adjust accordingly without making any other changes.

For this assignment you are limited to the language features in weeks 1-7 of the quarter. Use whitespace and indentation properly. Limit lines to 80 characters. Give meaningful names to variables and follow Python's naming standards. Localize variables whenever possible.

Include a comment at the beginning of your program with basic description information and a comment at the start of each section of code. Since this program has more complexity than some past programs, also put brief comments on any code that was harder to figure out. Remember, comments should describe what your program (or a portion of it) does not how it does it. Therefore, describe graphical output, animation, console output and user input. Do **not** mention loops, variables, `if/else`, etc.

## Extra Credit (2 points):

Alter your program ask the user if they want to do another walk after every walk they complete.

Assume that the user will give a one-word answer. The program should walk again if the user's response is the word "`yes`" or the word "`Yes`". Otherwise assume that the user does not want to walk again. For example, responses such as "`n`", "`N`", "`no`", "`okay`", "`0`", and "`hello`" would mean that the user doesn't want any more walks.

If the user wants to do another walk, the whole walk process described above repeats, including opening a new `drawing_panel` for the new walk. Each random walk should occur in its own properly sized `drawing_panel`.

If the user chooses not to walk again, the program should print the total number of steps accumulated in all walks. For example, supposing the user simulated three walks the console output to the right might be produced. Note that graphical output, just as described in the program description section of this document, would be produced too.

```
<< your introduction message here >>

Radius? 50
I escaped in 2468 move(s).
Walk again (yes/no)? yes

Radius? 35
I escaped in 987 move(s).
Walk again (yes/no)? Yes

Radius? 100
I escaped in 13713 move(s).
Walk again (yes/no)? n

Total steps = 17168
```