Random Walks!

CSC160, Prof. Phil Lombardo

(*A special thanks to Dr. Hank Feild for inspiring much this assignment. It has been adapted from his original work which made a bar chart work with random walks, which we will build on in future labs.*)

In this lab you'll have to plan and implement a program with your partner from scratch. You'll engage in the four steps of software development: understanding the problem, problem/functional decomposition, implementation, and reflection. The design, implementation, and use of functions is the primary focus of the lab.

Please do not allow your in-lab partner (or anyone else) to use your solutions if they were not actively participating in the work to find those solutions. The purpose of the labs in this course is to give you practice *doing* the work, which will help you learn the material.

# Team members

Please list the names of everyone you worked with during lab.

|  |  |
| --- | --- |
| **In lab team** |  |

# Learning Objectives

By the end of this activity, you should feel comfortable:

* designing, implementing, and using functions
* passing data between functions through parameters and return statements
* using global constants in place of magic numbers
* planning and implementing a program from scratch

# 

# Grading

The following rubric will be used to grade your work on this lab.

|  |  |  |
| --- | --- | --- |
|  | **Yes** | **No** |
| You worked diligently with your partner during the lab period, sharing the work and communicating throughout. | 3.3 pts | 0 pts |
| You and your teammates shared the programming and scribe responsibilities throughout the lab. | 3.3 pts | 0 pts |
| You made significant progress on the program and it runs without issues. | 3.3 pts | 0 pts |

# Part 1: Understanding the problem

For the purposes of this lab, we will treat a *random walk* as a series of steps by a turtle where at any given moment, the turtle can move forward, left, or right. (*Note:* in this lab, we are not allowing the turtle to move backwards!) What makes this a **random** walk is that the direction the turtle will move *will be randomly chosen*!

Some of you may think of this as a pointless programming exercise, but there’s actually some really interesting applications of random walks. They’re used for:

* [making stock market predictions](https://www.jstor.org/stable/4469865),
* modeling movement of particles in physics (both [quantum](https://quantumfrontiers.com/2021/01/17/random-walks/) and [other](https://www.britannica.com/science/Brownian-motion)),
* exploring some [mathematical curiosities](https://en.wikipedia.org/wiki/Random_walk#Higher_dimensions),
* and other [applications](https://en.wikipedia.org/wiki/Random_walk#Applications).

Your task today is to plan and implement a program that uses the Turtle library to allow a user to explore a type of random walk: each time the user hits enter, the turtle randomly chooses a direction (right/left/forward) and then moves one step. We will be simulating a random search, where the turtle starts at (0, 0) and the user hits enter to randomly move around with the goal of reaching a target position.

I will demo the program at the start of the lab period so you know what I’m hoping you produce. Please take notes during that time!

To generate the *random* component of the program, please import that random python module. In this module, you can call the function random.random() and it will return a completely random float value between 0 and 1. It will be up to you and your group to figure out how to use this to choose a random direction: right, left, or forward. As a hint, I recommend using an if-elif-else statement when defining the function that randomly chooses your direction.

Please do not use any other functions from the random module; I’m specifically using this approach to give you practice with branching.

1. Fill out the green boxes in this diagram indicating the input and output of the problem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **→** | **Program** | **→** | **Output** |
|  |  |

# Part 2: Planning and functional decomposition

In this part, you are going to break the problem down into subproblems. In the purple box below, create a hierarchical, bulleted list of problems and subproblems. Sub-subproblems (etc.) should be indented.

|  |
| --- |
|  |

# Part 3: Making a draft

1. To start your implementation, create a new Python file. Create a program docstring at the top using the format described in the CSC160 Style Guidelines (linked to from the top of the Canvas page). Include your and your partner's names as authors.

2. Next, let’s set up a **program skeleton**. This is like a sketch of your program, and consists of a docstring describing the program, and **function stubs** for each of the subproblems you created in the previous part. (You saw function stubs in the last lab, and probably during class.)

Recall that a function stub is a function definition in which the body contains no code, except for a return statement returning a dummy value for any functions that need to return something. Use the Python keyword **pass** for any function bodies that don't contain a dummy return statement. Be sure to include a main function at the bottom, as well as the if \_\_name\_\_ == "\_\_main\_\_" guard we've seen in the homeworks and some class examples.

Here's an example of a function stub for one of the functions I'm going to give you since it contains operations we haven't talked about yet:

|  |
| --- |
| def getMaxValue(val1, val2):  '''Finds the maximum value between two values.    Parameters:  val1 (float): the first value to consider.  val2 (float): the second value to consider.    Returns (Numeric): The maximum value found in the list.  '''  return 0 |

Use the functional decomposition guide to set up your skeleton, and call me over when you think you’re ready!

3. *Consider potential global variables.*To avoid magic numbers, think about what values you want to be constants. Constants (variables whose values are initialized to a hardcoded value and are not changed during a run of the program) can be defined in the global namespace. They should be all caps and use underscores to separate words. Define these now. No other variables should be global.

4. *Write some pseudocode for each of your function stubs.*Before implementing anything, go through each function and write some pseudo code. You don't need to be as formal as we saw in Ch. 1 of the book. Instead, write a sequence of comments, trying to keep each comment equivalent to one or two Python statements or involve a call to one of the functions you defined. Here's the function stub example from above with pseudocode comments.

|  |
| --- |
| def getMaxValue(val1, val2):  '''Finds the maximum value between two values.    Parameters:  val1 (float): the first value to consider.  val2 (float): the second value to consider.    Returns (Numeric): The maximum value found in the list.  '''  # if val1 < val2  # set max\_val equal to val2  # else  # set max\_val equal val1  # return max\_val  return 0 |

# Part 4. Implementing your plan

You are now ready to go through and start incrementally implementing your pseudo code. The two ways to approach this is *top down* (start with the higher level subproblems and work your way down to the sub-subproblems) or *bottom up* (start with the smallest sub-subproblems and work your way up to the higher level subproblems). I encourage you to do this top down, starting with main. This will allow you to test frequently and watch the program come to life as you implement.

After implementing a few lines of pseudo code, run your program and see if what is displayed matches what you expect given what you've implemented thus far. Because you've stubbed out your supporting functions, you can make calls to those functions where necessary, but keep in mind that until you implement them, they won't fulfill their intended purpose (in some cases, it may through an error, but you can trouble shoot that).

(See next page for code example.)

|  |
| --- |
| def getMaxValue(val1, val2):  '''Finds the maximum value between two values.  Parameters:  val1 (float): the first value to consider.  val2 (float): the second value to consider.    Returns (Numeric): The maximum value found in the list.  '''  if val1 < val2:  max\_val = val2  else:   max\_val = val1  return max\_val |

**IMPORTANT CONSIDERATIONS:**Here are some things that might help as you implement your code:

* Each of your three directions should be equally likely (or at least very very nearly so).
* I recommend fixing how far the turtle moves with each step. This will allow you to smartly choose a target position that the turtle has a chance of reaching. For example, if my turtle moves in steps of size 20, the coordinates of a target should be multiples of 20.
* For testing reasons later, I recommend putting your target relatively close to your initial turtle position (making it easier to find the target by random steps).
* You may want to consider using the turtle.write() and turtle.dot() functions.
* The turtle position, turtle.pos(), is a ***tuple*** of floats.
  + You will need to access the *x* and *y* coordinates by indexing the tuple. For example, if we have the tuple position = (20.0, 40.0), then we can use position[0] to access the *x-*coordinate of 20.0, and position[1] to access the *y-*coordinate of 40.0.  
    (Feel free to play around in an interactive python shell to get used to this.)
  + Since it contains floats, when checking if you’ve reached the target, you will want to use something like the [isclose()](https://docs.python.org/3/library/math.html#math.isclose) function from the math module on each of the entries in the tuple.
* the Turtle method [teleport](https://docs.python.org/3/library/turtle.html#turtle.teleport) might be handy
* you don't need to show the animation of the Turtle module drawing out the movements of the turtle; instead make use of the [tracer](https://docs.python.org/3/library/turtle.html#turtle.tracer) and [update](https://docs.python.org/3/library/turtle.html#turtle.update) methods to turn off drawing (myScreen.tracer(0)) before drawing begins and show what was drawn (myScreen.update()) after drawing has completed.

# Part 5: Reflecting back

In this step, you want to review your code and think about whether it is accurate, if it meets the requirements of the problem, how it's organized and documented, and its efficiency. We won't worry too much about efficiency right now; you will do a deeper dive if you take CSC 161 next semester.

## Accuracy and completeness

We can often write unit tests in order to determine the accuracy of one or more functions. This can involve passing different potential arguments to functions and checking they respond as desired, or running a program and testing different types of interactions and entries to make sure the program does what we want.

This program is largely drawing to a screen and observing a few different interactions. Run your program several times and verify the behaviors listed in the table below. If you find that something doesn't work, go back and fix it, then update your response.  
(You may need to run the program several times to see what happens when the turtle hits the target. I recommend putting the target close to the starting point to make this more likely; perhaps two or three steps away.)

|  |  |
| --- | --- |
| ***Behavior*** | ***Yes, verified!*** |
| When starting the program, does it draw dot for the starting position as well as a dot for the “target”? |  |
| On hitting enter, does the turtle move at least one step and leave a small line as a trail? |  |
| On hitting enter several times, is it clear that the turtle is moving along a grid (i.e., along little squares). |  |
| On hitting enter, does the movement of the turtle appear random? Is the movement without any discernible pattern? |  |
| On hitting enter, is it true that the turtle *never* moves backward? |  |
| Does hitting ‘SHIFT + C’ restart the game? |  |
| If the turtle hits the target, does hitting ‘Enter’ no longer move the turtle? |  |
| If the turtle hits the target, does the text “I made it!” appear on the screen? |  |

Just as a curiosity, record one bit of information for your program. Start a fresh game, and try letting the turtle randomly search for the target. If you have gone a few hundred steps without hitting the target, try restart the game and try again. In the box below, record how many restarts you require to hit the target!

|  |
| --- |
|  |

## Organization

Here are some things to reflect on; if your answer is "no" to any of these, go back and update your code.

|  |  |
| --- | --- |
|  | **Answer** |
| Are your functions of a reasonable size? This is subjective, but if you have a function that is significantly longer than your other functions, it may be a good candidate to break into two or more smaller functions. |  |
| Is main defined at the bottom of your program? |  |
| Are constants the only global variables you defined? |  |
| Is the only code outside of a function the constants you defined at the top and the if \_\_name\_\_ == "\_\_main\_\_"... guard at the bottom of the file? |  |
| It's annoying when a Turtle window pops up, but the program is waiting for input on the console. Do you prompt the user for the input file before you create a Turtle instance? |  |

## Documentation

Here are some things to reflect on; if your answer is "no" to any of these, go back and update your code.

|  |  |
| --- | --- |
|  | **Answer** |
| Does your program include a docstring at the top formatted as described in the CSC160 Style Guidelines? |  |
| Does every function have a docstring formatted as described in the CSC160 Style Guidelines? |  |
| In each docstring, do the parameters listed match the parameters in the function signature? |  |
| Do you have clear comments around chunks of code that "hang together"? These shouldn't be pseudo code and they shouldn't literally describe the code, but rather be short descriptions addressing the purpose of the chunk of code. |  |
| Are your variable and function names self documenting? |  |
| Are your function names actions? |  |
| Are your variable names nouns? |  |

# Reflection

Please answer the following questions:

1. What confused you most about this activity?

|  |
| --- |
|  |

2. What questions do you have either about the activity directly or about related topics that came to mind while you did the activity?

|  |
| --- |
|  |

3. Do you feel like you achieved all of the learning objectives listed at the top of each part that you completed? If you feel uncertain about any of them, please list them here.

|  |
| --- |
|  |

# Submission

Make sure that your and your collaborator's names are at the top of your Python file.

Download a copy of this document as a .pdf or .docx. Upload that copy along with your code to the Canvas assignment for this lab. Only one student needs to submit from each group.