Making a Maze Game!

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 drawing palette to build on our maze lab from last time.*)

In this lab you'll have to understand the code in a partially written program, and then extend it by modifying existing functions and adding new ones. During the lab period, you must perform all of these steps with your partner.

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, and whether you worked alone or not after lab.

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

# Learning Objectives

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

* working with a partner to read and modify code (this is called pair programming)
* reading and making sense of an existing Python program
* modifying an existing Python program to change its behavior
* working with event-driven programs

# 

# 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. | 3.4pts | 0 pts |
| While the scribe, you wrote responses in the document and collaborated with the driver on what to write; you did not write any code. | 3.3 | 0 pts |
| While the driver, you wrote code while communicating with the scribe; you only entered code responses to the document. | 3.3 | 0 pts |

# Part 1: Exploring the code

0. During this part, one of you will act as the **scribe**, which involves having this document open and typing in responses. The non-scribe should be in charge of viewing and running the code (we'll refer to this role as the **driver**). The scribe should not have any code open, and must communicate with the driver through talking. All code written by the driver should be agreed on by both the driver and the scribe. If you are in a group of three, there can be two scribes; have the driver sit in the middle of the two scribes.

Make sure you understand this and that you've taken a look at the grading rubric before continuing.

Note that some of the questions ask you to paste in code that you wrote; the driver is allowed to enter that information since they have easy access to the code at that point in the lab.

Decide which of you will fulfill which role and indicate that below:

|  |  |
| --- | --- |
| Scribe |  |
| Driver |  |

1. The driver should Download lab3.py from this lab's Google Drive folder. Read through the code with your partner, *without actually running the code*. Discuss the code that you see and answer the questions below.

a. What is a “function stub”? Which of the functions in this program would be considered function stubs?

|  |
| --- |
|  |

b. For now, ignoring the function stubs, review the rest of code and try to understand what it does and how it all fits together. *Without running the code*, what do you think the program would do if someone were to run it and then press the right-arrow key on your keyboard?

|  |
| --- |
|  |

c. Now run the program. Once the white Turtle canvas window appears, press the right-arrow on your keyboard. Was your prediction correct? If not, what actually happened? (Keep your program running as you answer the next several questions.)

|  |
| --- |
|  |

d. Now try pressing the up-arrow on your keyboard. What happens, if anything? Why do you think the program behaved that way?

|  |
| --- |
|  |

4. We will not make a modification to the code and try running the program again. In the move\_east() function, change pat.setheading(0) to pat.setheading(90).

a. *Before running the program,* predict what will happen if you run the program and press the right-arrow button on your keyboard

|  |
| --- |
|  |

b. Now run the program and press the right-arrow button on your keyboard. What happened? Was your prediction correct?

|  |
| --- |
|  |

c. Why do you think the program behaved this way? What did the code change do?

|  |
| --- |
|  |

# Part 2: Adding functions

As you address these questions, you may want to look at the [Turtle class documentation](https://docs.python.org/3/library/turtle.html#turtle.dot) for help on finding what methods are available and how to use them.

0. ***Switch the scribe and driver roles*** and note your roles below. The previous driver needs to share code with the new driver; you can choose how you want to do this (email, Discord, pastebin), but a quick and easy way might be to use a Google Doc that you are both editors on (you'll need to exchange the code several times in the next part, as well).

|  |  |
| --- | --- |
| Scribe |  |
| Driver |  |

1. Begin by fixing the code we changed in the previous section so that move\_east() moves the turtle to the right when the right-arrow is pressed.

Now make a plan with your partner how to fill in the code for the remaining movement functions: move\_west(), move\_north(), and move\_south(). These should make the turtle move left, up, and down, respectively.

Briefly describe the changes you plan to implement.

|  |
| --- |
|  |

2. The driver should now change these three functions and save the file. In developing the code, make sure you also fill out the “Doc String” which explains what the function does and its parameters.   
Share the code *for just the* move\_north() *function in the box below.*

|  |
| --- |
|  |

3. Now run your program and try moving the turtle in all four directions.

a. Did it work? Why do you think the turtle behaved this way? (You may want to revisit your summary of the program from the exploration part, and think about how all the functions “fit together.”

|  |
| --- |
|  |

b. If you think you diagnosed the problem, make the appropriate changes and try running your program again to make it work. Keep working on this problem until it works, and feel free to shoot a hand up if you get stuck!

4. You will notice that the move\_east() function has its turtle actions sandwiched between these bits of code: screen.tracer(0) and screen.update(). Using the [Turtle class documentation](https://docs.python.org/3/library/turtle.html#turtle.dot), figure out what these are doing and describe it in the box below.

|  |
| --- |
|  |

# Part 3: Adding new functionality

*For this part, you should* ***alternate who drives after each of the features******below*** *so you each have a similar amount of time driving.*

*There's nothing to answer in this document for these parts. Instead, you will submit the Python file with your modifications from this part. See the* [*Submission*](#_2s8eyo1) *section for instructions.*

1. **Drawing Triangles.**  As the turtle moves through the maze using the keyboard input, it does leave a faint purple trail. However, we want our game to leave more obvious “bread crumbs” along our turtle trail. To implement this feature, please do the following:

* Fill in the code for the draw\_triangle() function so that when this function is called, the turtle draws a small triangle behind its current location and returns to its original position (which it should do automatically if you just trace out a triangle).
  + I recommend the length of the triangle should be about 8 “turtle units”.
  + Fill in the triangle purple; consider the begin\_fill() and end\_fill() functions from the [Turtle class documentation](https://docs.python.org/3/library/turtle.html#turtle.dot).
* Add function calls to draw\_triangle() in the appropriate place or places so that you leave the triangle breadcrumbs as you move around the maze.

2. **Choosing different mazes**. The goal of this modification is to allow the user to chose among our five different mazes: maze1.png, maze2.png, maze3.png, maze4.png, and maze5.png. To implement this feature, please do the following:

* Fill in the code for the create\_maze\_filename() function. When a number is given as an argument to this function, it should return a string with the right filename including the provided number. For example, create\_maze\_filename(3) should return “maze3.png”.
* In an appropriate place of your program, prompt the user to choose from a maze number (1 – 5). Take that input, apply your function to create the appropriate filename, and use this string output to set the background appropriately.

3. **Add a compass to the top of the screen.** For this feature, we want to draw a small, labeled compass above the maze (I have left you space). To implement this feature:

* Explore the turtle actions of write() and teleport() and use them to draw the compass in the appropriate place. (You may need some trial and error to determine where to start your turtle.)
  + Put this code inside a function called draw\_compass(), and make sure you write a Doc String for it!
  + In an appropriate place of your program, call the draw\_compass() function so that appears with the maze before the user begins moving the turtle.

As you complete these features, call me over to show me how you’re progressing!

If you want some new challenges, feel free to ask.

# 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.