Name _____

CPADS Assignment #3

1. Your task is to finish a simple game that moves a turtle through a series of obstacles using various headings that the user supplies to avoid collisions with the obstacles and the boundary walls of the game field.

   The user is allowed to have a maximum of four collisions with boundary walls and the obstacles (circles and rectangles). Collisions with the boundary walls occur on the inside of the walls, while collisions with the obstacles occur on the outer edges of the obstacles. The program terminates when the maximum number of allowable collisions has occurred.

   The user is prompted to input a maximum length for each move, within a range of 20 to 200 pixels inclusive. The turtle then moves at the current heading for that maximum distance, unless it hits an obstacle. The program checks for collisions for each pixel that it moves. When the turtle has moved the maximum distance or if a collision occurred, the user is prompted for a new heading. In the case of a collision, the turtle will move the remainder of the distance in the direction of the new heading. In the case of a completed move, the turtle will move in that direction for the maximum distance or until a collision occurs.

   Functions have been provided that draw the boundary lines and the obstacles. Skeleton code has also been supplied that runs the program, and partially implements various functions that are required to detect collisions of the turtle with the boundary walls and the obstacles.

   Your first task is to come up with a strategy to finish the problem. More specifically, you should supply simple strategies for implementing **TODOs 1, 2, 3, 5, 9, and 10**.

   TODOs 1, 2, and 3 are asking you to implement the collision detection for the boundary walls, rectangles, and circles. Each of your strategies should describe, in English, the logic and calculations necessary to detect the collision for each of those TODOs.

   TODOs 5, 9, and 10 are more specific to the logic of playing the game.

   NOTE: When you start to implement your strategies, we recommend that you implement **TODOs 1, 8, 9, and 10 first**, in that order, before moving on to the other TODOs. This way, the program will be able to process moves, detect collisions with the boundary walls, and request new headings.

   More detail is given in the provided code with each of the TODO's.

   Make sure to provide appropriate and sufficient comments for all of the code that you modify or add.

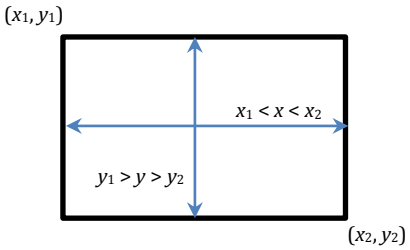
Name _____

Write your strategies for **TODOs 1, 2, 3, 5, 9, 10** here. Feel free to include diagrams, as necessary, to describe your strategy.

SOLUTIONS FOR STRATEGIES:

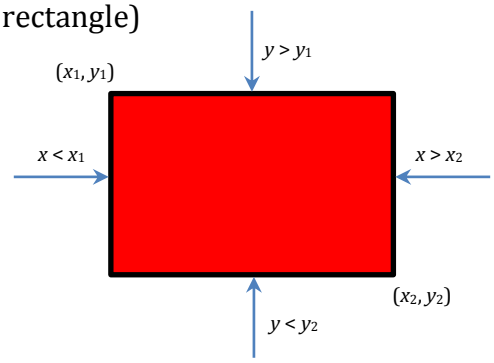
TODO 1: Collision with Boundary

1. Turtle position (x, y) must stay inside 4 boundary "walls":
   a. Check for Turtle between left (x_1) and right (x_2) edges: $x_1 < x < x_2$
   b. Check for Turtle between top (y_1) and bottom (y_2) edges: $y_1 > y > y_2$
2. If either condition fails, return **True** (Turtle collided with a boundary wall)
3. Otherwise, return **False** (Turtle remained within boundary walls)



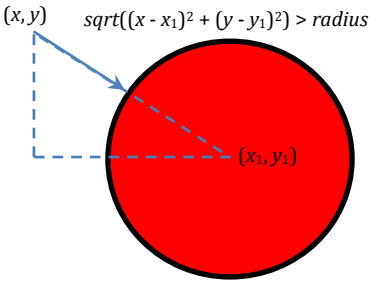
TODO 2: Collision with Rectangle

1. Turtle position (x, y) must remain outside the 4 rectangle "edges"
   a. Turtle outside of both left (x_1) and right (x_2) edges: $x < x_1$ OR $x > x_2$
   b. Turtle outside of both top (y_1) and bottom (y_2) edges: $y > y_1$ OR $y < y_2$
2. If either condition fails, return **True** (Turtle collided with rectangle)
3. Otherwise, return **False** (Turtle remained outside of rectangle)



TODO 3: Collision with Circle

1. Turtle position (x, y) must remain more than *radius* distance away from center (x_1, y_1) of circle: $sqrt((x - x_1)^2 + (y - y_1)^2) > radius$
2. If condition fails, return **True** (Turtle collided with circle
3. Otherwise, return **False** (Turtle remained outside of circle)



Name _____

SOLUTIONS FOR STRATEGIES:

TODO 5: Validate user input for maximum distance for Turtle moves

1. Prompt the user for value (*maxMove*) in the range 20 ≤ *maxMove* ≤ 200
2. If the above condition fails, inform the user and repeat the above prompt
3. Otherwise, validation loop exits and execution proceeds to game loop

TODO 9: Process end of Turtle move

1. Prompt user for next heading (call *getNewHeading()* function)
2. Set Turtle heading to value returned by call to *getNewHeading()*
3. Reset distance moved (*distance*) to 0 for next Turtle move
4. Movement loop resumes for full *distance* with new Turtle heading

TODO 10: Process collision

1. Count collisions
2. If collisions exceed max allowable collisions, inform user, game loop exits
3. Otherwise, prompt user for next heading (call *getNewHeading()* function)
   a. Set Turtle heading to value returned by call to *getNewHeading()*
   b. Movement loop resumes for remainder of *distance* with new Turtle heading

SOLUTION FOR PROGRAMMING PORTION CAN BE DOWNLOADED FROM:

https://ycpcs.github.io/cs100-fall2018/assign/src/turtlegame-solution.py

Name _____

2. **Download the file assign03.py from the course webpage**

   https://ycpcs.github.io/cs100-fall2018/assign/src/turtlegame.py

   Open the file with **PyCharm**. **USING YOUR STRATEGIES FROM PART 1**, add code for each of the TODOs shown in the program.

   - **TODO 1:** Add code to detect a collision with the boundary in the **detectCollisionWithBoundary()** function.
   - **TODO 2:** Add code to detect a collision with a rectangle in the **detectCollisionWithRectangle()** function.
   - **TODO 3:** Add code to detect a collision with a circle in the **detectCollisionWithCircle()** function
   - **TODO 4:** Using some of the implemented headings as a template, complete the conditions for the remaining headings **South**, **Down**, **SE**, **Left**, **Right**, **Forward**, and **Backward** in the **getNewHeading()** function.
   - **TODO 5:** In **main()**, add user validation to the user input for the value for **maxMove**. The code should continue asking the user for input until they enter a valid value between **20** and **200** inclusive.

   In **main()**, the **outer while loop** counts collisions, checks for the end of a move, and prompts the user for a new heading at the end of each move, or when collision has occurred.

   The **inner while loop** moves the turtle forward by 1 pixel at a time, checks for collisions after each single pixel move, and keeps track of the distance covered during the move.

   Inside the **inner while loop** using the collision detection checks for **circle1 and circle2** as a template:

   - **TODO 6:** Add collision detection for the turtle with **circle3** and **circle4.**
   - **TODO 7:** Add collision detection for the turtle with **rectangle2.**
   - **TODO 8:** If no collisions are detected, update the distance for the move.

   In the **outer while loop**, after the end of the inner loop

   - **TODO 9:** When the turtle reaches the maximum move distance, reset the distance, prompt the user for a new heading, and get the new heading from the user.
   - **TODO 10:** If a collision occurred, increment the collision counter, and tell the user that a collision occurred. If the user has not exceeded the max # of collisions, then prompt the user for a new heading, and get the new heading from the user.

     If the user has exceeded the maximum allowable collisions, inform them at the end of the outer loop, before the outer loop exits.