Part One: Modeling a Turn: Point

There are instance where turning in place, or a **point turn** is preferable. Can you list some of these situations?

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

If both the left and right wheels are moving at the same power level, but in the right wheel is moving forward while the left moves backwards (a point turn), what path will the wheels below take? Please draw in the path:

![Image of a robot with two wheels](image)

Hopefully you saw that with a point turn, the robot turns in a circle with a diameter equal to the wheelbase of the robot (illustrated bellow).

![Image of a robot turning in a circle](image)

What is the radius of this circle?

What is the circumference of this circle?
Given the circumference found above, how many rotations must the wheels travel in order to turn 90-degrees to the left?

What would happen if we increased the size of the wheels?

How would the addition of gears change your results?

Which turning option is best for navigating a closed maze with a fixed wall width? Why?
Part Two: Displaying a Value

In part three we are going to write a program to test our calculations from part one. In order to display the number of rotations completed by each motor, it is important that the rotation sensor is zeroed before a reading is collected. Where in the program do you think the rotation sensor should be reset?

a. At the very beginning of the program
b. Just before the turn
c. Just after the turn
d. Just before displaying the value to the screen.

Answer: ________

If you answered 'b', you're right! The number of times the motor turns is constantly being recorded, so it is important that the rotation sensor is reset just before the turn if we are interested in knowing only the number of rotations traveled during the turn. By setting the value of \text{nMotorEncoder[mRight]} to zero, we are resetting the rotation sensor.

Once you zero the rotation sensor, complete a turn and stop, the next step is to display the number of rotations to the NXT display using \text{nxtDisplayTextLine()}, which accepts the following parameters:

\text{nxtDisplayTextLine(line_number, text, var1, var2, var3)};

Here is an example of how we can use \text{nxtDisplayTextLine()} to print to the LCD screen. Notice that in this example the programmer prints the \text{value} of a variable to the screen:

\begin{verbatim}
int x = 5;
nxtDisplayTextLine(4, "The letter x ");
nxtDisplayTextLine(5, "is worth \$d points!", x);
\end{verbatim}

Check your Understanding:
Write a program that displays the name and age of each member of your group on a separate line on the LCD screen. Rather than entering the age as part of the text, create and print a variable for each person's age (as in the example above). For example, variable should look like this:

\begin{verbatim}
int johnAge = 15;
int jeffAge = 14;
int saraAge = 15;
\end{verbatim}

In this example, the final product should print to the screen like this:

John is 15 years old
Jeff is 14 years old
Sara is 15 years old
Part Three: Testing your Turns

Now it’s time to combine the print screen code from part two with the turns discussed in part one. A basic example of this is shown below (please note that this code does not perform all of the functions required for this step, but is meant to given a basic example of how to print rotation sensor values to the screen):

```c
task main()
{
    // reset motor rotation counter to zero
    rMotorEncoder[mRight] = 0;
    rMotorEncoder[mLeft] = 0;

    // perform a turn

    // print to the display for 5 seconds
    nxtDisplayTextLine(3, "motor a rot: %d", rMotorEncoder[motorA]);
    nxtDisplayTextLine(5, "motor c rot: %d", rMotorEncoder[motorC]);
    wait10Msec(500);
}
```

You should now program your robot to make a **90-degree pivot turn**. Your program should print the number of rotations traveled by each motor on the LCD screen (this should update constantly while the program is running). Once you have completed the program, run it twice and record your results below in the columns designated ‘actual’. For comparison, fill in the predicted rotation value from part one in the table. Prior to writing this program in RobotC, please explain in words (or pseudocode) how your program will work:

### Pivot Turn Table:

<table>
<thead>
<tr>
<th></th>
<th>Predicted</th>
<th>Actual from Test 2</th>
<th>Actual from Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Pseudocode for Pivot Turn:

```c
// reset motor rotation counter to zero
rMotorEncoder[mRight] = 0;
rMotorEncoder[mLeft] = 0;

// perform a turn

// print to the display for 5 seconds
nxtDisplayTextLine(3, "motor a rot: %d", rMotorEncoder[motorA]);
nxtDisplayTextLine(5, "motor c rot: %d", rMotorEncoder[motorC]);
wait10Msec(500);
```
Now, write a program to make your robot complete a 90-degree **point turn**. Again, the number of rotations traveled by the motors should be displayed to the LCD screen, and constantly updated as the program is running. *Prior to writing your program, please outline your ideas in the pseudocode box below.*

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Actual from Test 2</th>
<th>Actual from Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Pseudocode for Point Turn:**

Are your predicted and actual values the same for both your point and pivot turns? Are you surprised by the outcome?

_________________________________________________________________________________________________________

_________________________________________________________________________________________________________

_________________________________________________________________________________________________________