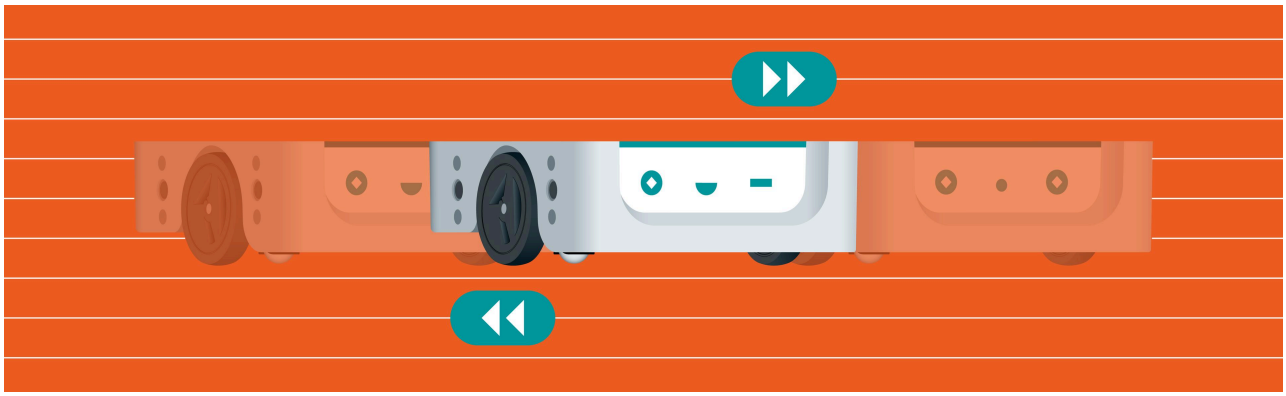
1:30 HR

There and Back Again

Precisely control the forward and backward movements of your robot.



Introduction

In the previous chapter, you learned how to make your robot move around freely and maybe even dance a little! We will now practice moving your robot forward and backward with absolute precision.

This may sound easy at first glance, but it takes a little practice to perfect the art of precision movement. Experiment with wheel rotations, time, and distance as you pursue the path of mechanical movement mastery. Try saying that three times fast!

Good luck little grasshopper.

Learning Objectives

- Precisely move Alvik in a forward/backward direction.
- Understand wheel geometry (circumference, diameter, radius).

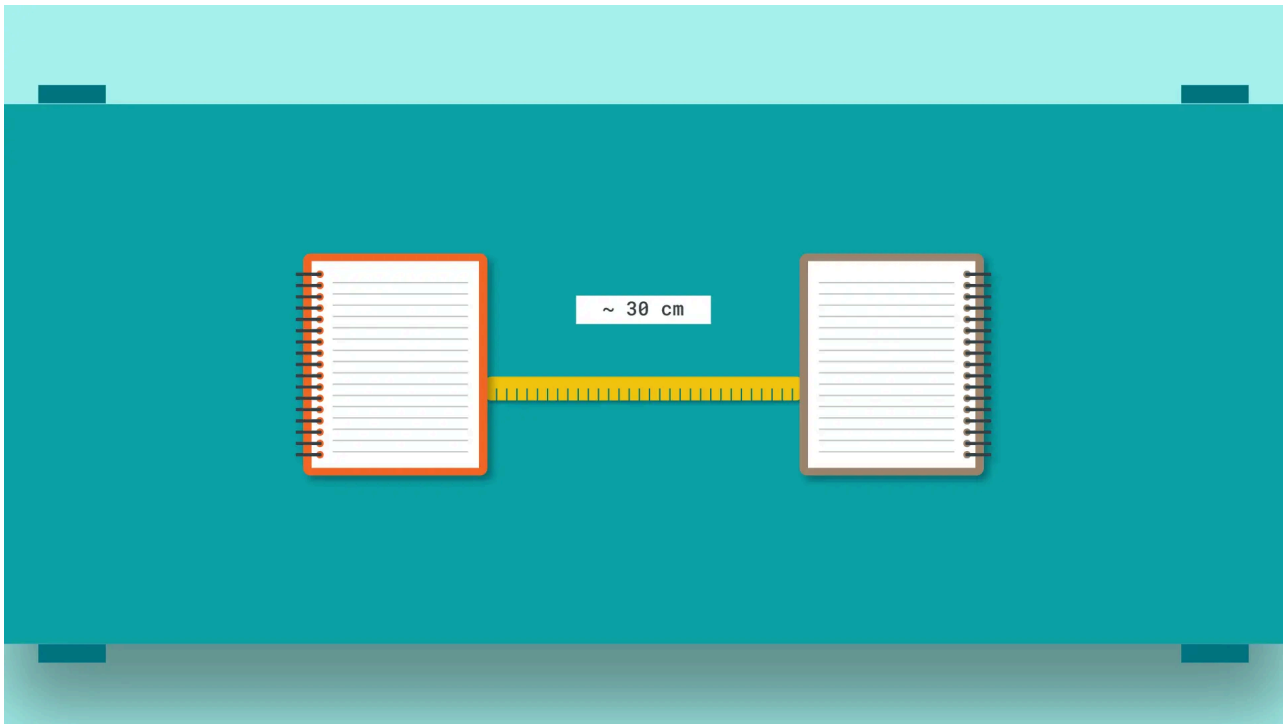
- Calculate distance using time and speed variables.
- Observe and compare measurement outcomes.
- Save and run files on Alvik for cable-free testing.

Setting Up

Let's prepare our practice zone. Make sure you have the following materials available.

- **Barriers** (e.g. - books, boxes, black tape)
- **Tape Measure or Ruler**
- **Calculator**
- **Notebook** (recommended)

Position the barriers as shown in the image below. We are using two notebooks, but you may use any objects you have available. The exact distance between barriers is not important at this time but they should be approximately one ruler-length apart.



Get Moving

Place Alvik between two notebooks with the back of the robot touching one of them. In just a moment, we will write a simple program to move your robot forward and backward. The goal is to get Alvik as close as possible to the next notebook without touching it and then return back to its starting position.

Sound easy?

Give the challenge a try using the code below in a new script. You will quickly notice similarities to the examples from our last lesson.

```
from arduino_alvik import ArduinoAlvik

alvik = ArduinoAlvik()

def setup():
  alvik.begin()
  delay(1000)

def loop():
  alvik.set_wheels_speed(20, 20)
  delay(5000)
  alvik.set_wheels_speed(-20, -20)
  delay(5000)
  alvik.set_wheels_speed(0, 0)
  delay(10000)

def cleanup():
  alvik.stop()

start(setup, loop, cleanup)
```

> **NOTE:** **Remember to save your work often and use filenames that clearly identify your script - for example, *get_moving.py* **

The odds are that you did not get as perfectly close as you wanted, or you may have crashed into the opposite notebook. Let's see if we can do better by adjusting the arguments inside `alvik.set_wheels_speed()` and `delay()`.

As you test different values, recall that we use **Revolutions Per Minute (RPM) to measure Alvik's speed**, which means wheel rotation has a direct relationship with

the length of time inside our `delay()` function.

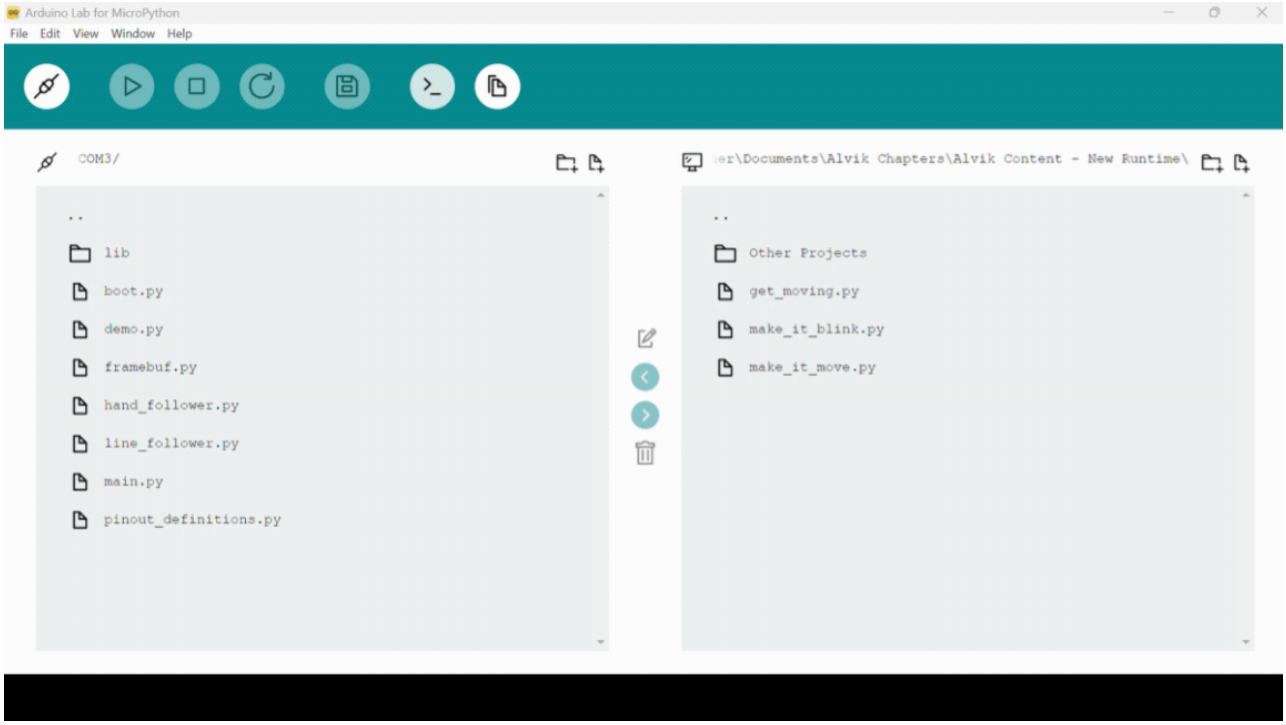
As a simple example of this relationship, imagine changing the time delay to `delay(60000)` while maintaining the speed of the wheels at 20 RPM. Since 60 seconds = 1 minute and our RPM is set to 20, we know that the robot would complete 20 wheel rotations before the next line of code is executed. If we change the length of time to `delay(30000)`, half the time as before, the robot would complete only 10 revolutions.

Cable Free

Up to this point, we have run all our scripts directly from Arduino Lab for MicroPython, which communicates with Alvik via USB® cable. But let's be honest, dealing with a cable while Alvik is moving can be a bit of a headache! Up next, we explore how to save our files to the robot and run them entirely cable-free.

Remember how to manage files in Arduino Lab by clicking on the **File Manager** button? Here, we can see all the files saved locally within a folder on our computer. In the space to the left of local file management, we can manage files that are saved on Alvik. Let's try moving one of the local files on the right to Alvik's memory on the left.

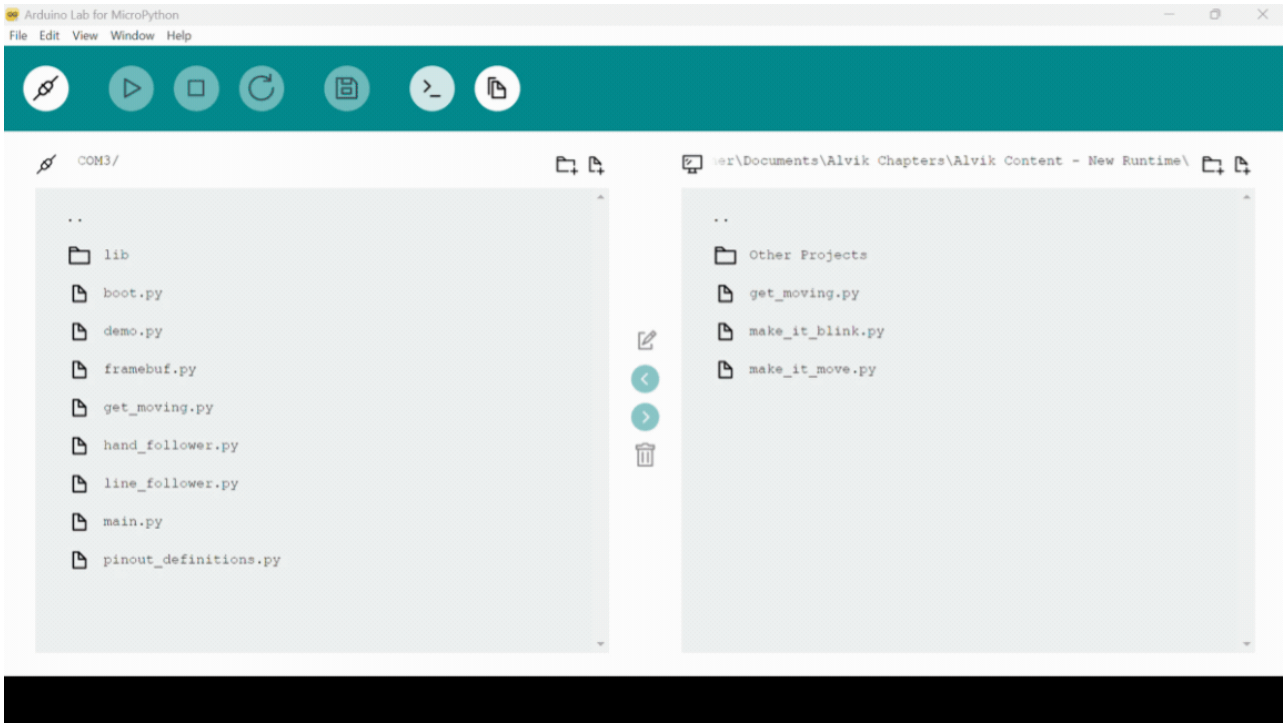
1. Select one of your project files and **click Upload to save a copy to Alvik.**



Alvik now has a copy of your program saved to its memory, but it doesn't understand that it should run the script. The trick is to use a file called ***main.py*** - keep reading to understand how we use it.

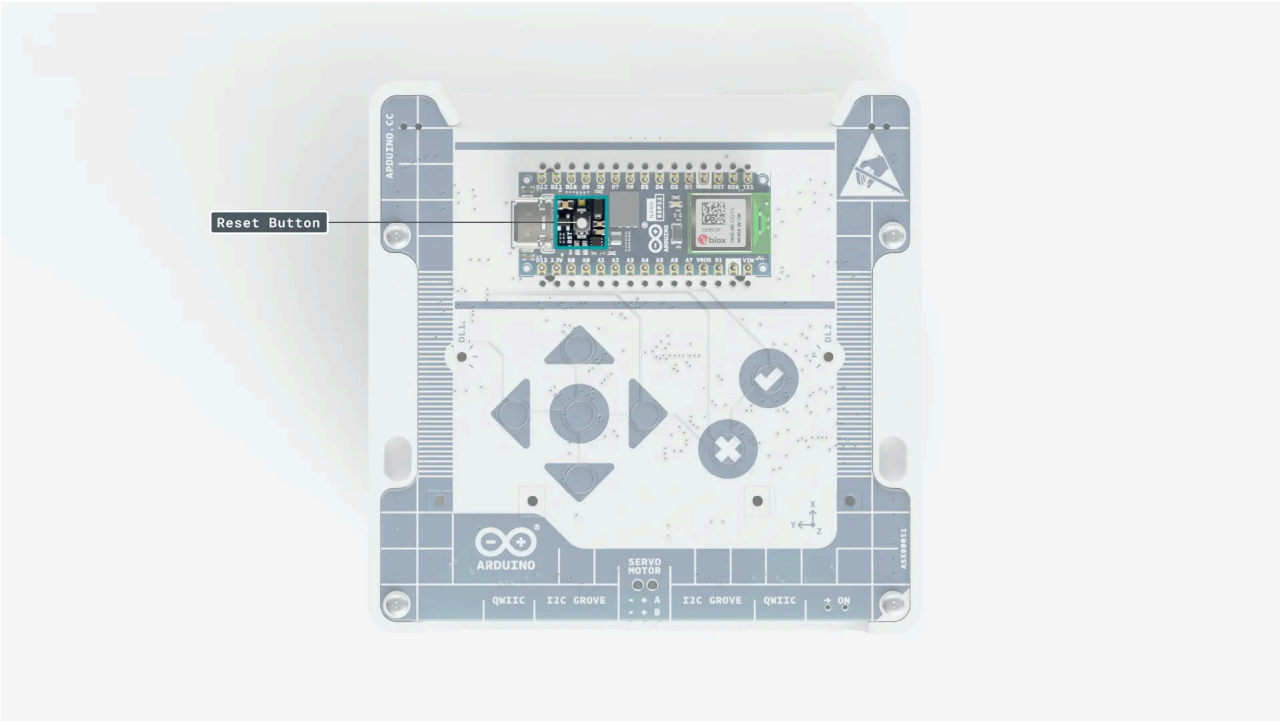
In MicroPython, just like Python, *main.py* is one of the first files to execute when your board powers on. That means any script or modules imported to this file will instantly run once you restart or turn on Alvik. Try for yourself in the next step!

2. Open *main.py* on your board and delete the existing contents, if any. Then, **update the script with** `import get_moving` **and click Save.**

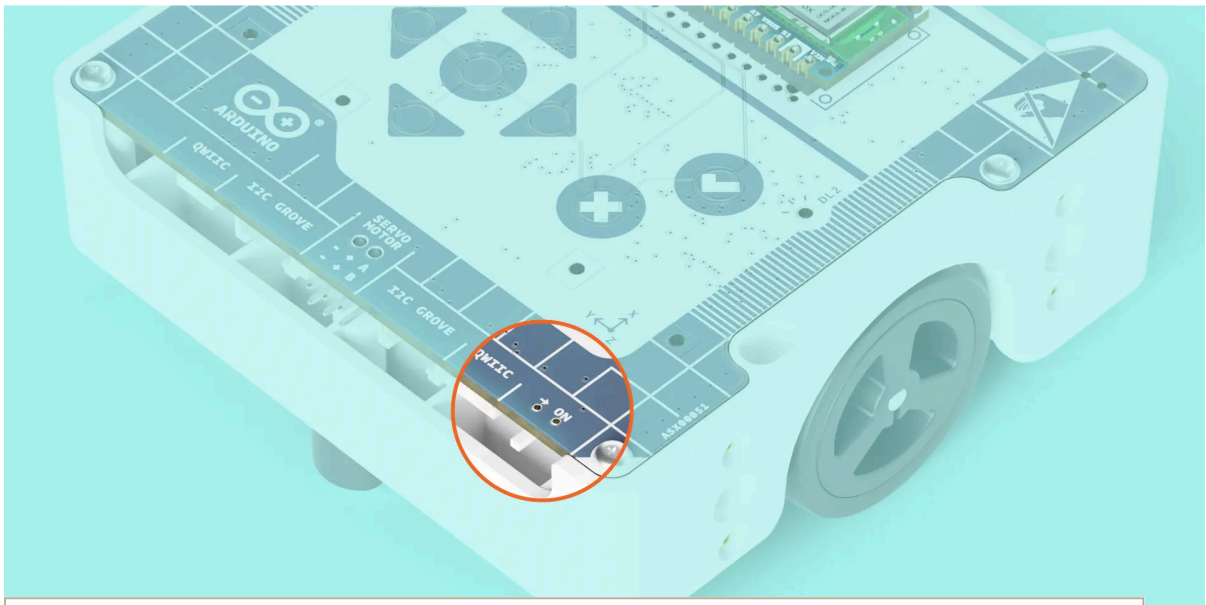


Now, whenever Alvik powers up, *main.py* will import your script as a module and run the code inside.

Time for the first cable-free experience! **Disconnect the USB-C® cable from your Nano ESP32 and push the white reset button on top of the board**. After a few seconds, your script will automatically begin.



Keep in mind that your script will loop forever. When you are ready to stop testing, you must turn off the robot.



If you want to stop your robot, simply slide its power switch to the OFF position

Math Saves the Day

A Better Way

How was the previous movement challenge - easy enough, right? Perhaps it took a little trial and error to reach the goal, but eventually you got there. Maybe you adjusted the speed, timing, or even both - or you used a ruler to measure distances traveled and adapted your estimations accordingly.

But, what would happen if we moved the notebooks or wished to travel exactly the same distance no matter the speed? We would have to start the testing process all over again and that would be horribly time-consuming.

Thanks to basic mathematics, there is a better way to approach this problem. In just a moment, we will deepen our understanding of the relationships between time, distance, and geometry. Math is here to save the day, and you will soon discover a formula that does all the heavy lifting for us!

The Circle

Look at your robot and imagine the wheels are circles. There are various ways we could measure the wheel as a circle, including:

- **Radius:** The distance from the center point of the circle to its outer edge.
- **Diameter:** The distance across a circle, passing through its center point.
- **Circumference:** The length of the outer line that creates the circle itself.

Looking at the image above, we find the Diameter **(d)** of Alvik's wheel is 34 mm across. To calculate the Circumference **(C)** of the wheel, we use diameter and Pi **(π)** as shown in the following formula:

$$C = \pi \cdot d$$

By plugging in actual numbers, we find **C = 3.14 x 34** or **C = 106.8 mm** in length.

So, if the distance around the wheel is 106.8 mm in length, **we can conclude that a complete wheel rotation of 360º will move the robot an approximate distance of 106.8 mm.** And really, that makes a lot of sense. Imagine we could wrap a ruler around the wheel to measure its circumference. Once we lay the ruler flat, we would see the length as a straight path. This path represents the distance our robot travels each time its wheels fully rotate.

Let's use our understanding to build another formula. Since Alvik travels 106.8 mm each time its wheels complete a full rotation, we can easily calculate the total

distance traveled by multiplying the circumference by the total number of rotations. For example, distanced traveled for two rotations would be calculated as 2 x 106.8 = 213.6 mm. This formula is shown below using the variables **Revolutions/Rotations (revolutions), Circumference (C), and Distance (distance)** to define the total distance traveled.

$$distance = C \cdot revolutions$$

But wait, **we are forgetting a very important variable in our equation... time!** Recall that we program movements by setting RPMs for a period of time, which means time affects how many wheel revolutions actually occur.

Think about this example for better clarification. Wheel speeds are set to 1 RPM for a period of `delay(60000)` (60,000 Milliseconds = 60 Seconds = 1 Minute) before finally resetting speeds to zero. How many rotations will be made by the end of this time delay?

```
1   alvik.set_wheels_speed(1, 1)
2   delay(60000)
3   alvik.set_wheels_speed(0, 0)
```

Haha, easy one, right?

We know the Revolutions **per Minute** is set to 1, meaning the wheels make exactly one full rotation when a minute passes. By continuing to play with the delay, you can see how easy it is to precisely control rotations with time. Changing the delay to `delay(120000)` , for example, results in two complete rotations, while `delay(30000)` gives a half rotation of 180º.

By now, it is clearly evident that time plays a major role in our calculations. Where **t** represents the time value in milliseconds, let's update our formula to include it as $\frac{t}{60.000}$.

$$TravelDistance = C \cdot revolutions \cdot \frac{t}{60.000}$$

With this formula, you now have a superpower! As long as you know the value of any two variables in the formula, you can easily find out the third missing variable through quick calculation. Move on to our next script to see it in action.

NOTE: With the help of mathematical formulas, we can make accurate predictions of outcomes, but it's important to keep in mind that real-world results may not perfectly align with our calculations. This is because of **external factors** that could affect our robot's movements such as gravity or friction.

For example, imagine your robot is driving on a slippery wet surface. How would this affect the distance traveled if the wheels occasionally slip?

We should do our best keep external factors in mind, but sometimes it's difficult or impossible to take everything into consideration during experimentation. We won't account for several of these factors in upcoming activities, which is perfectly okay as long as you understand that they may exist.

Try It Yourself

Try testing the new formula with these values:

- **t = 20 seconds**
- **revolutions = 10 RPM**
- **C = 106.8 mm**

Our formula gives us **106.8 x 10 x 20,000/60,000**, resulting in a total **Travel Distance of 356 mm.**

In the code, **t** and **revolutions** takes the following positions:

- delay(**t**)
- alvik.set_wheels_speed(**revolutions, revolutions**)

And, here is how it finally looks inside of the `def loop()` :

```
1  def loop():
2    alvik.set_wheels_speed(20, 20)
3    delay(20000)
```

```
    alvik.set_wheels_speed(0, 0)
    delay(5000) # Optional pause
```

Or even better, we could use variables as shown in this complete version:

```
alvik = ArduinoAlvik()

# Global variables
revolutions = 10
t = 20000

def setup():
  alvik.begin()
  delay(1000)

def loop():
  alvik.set_wheels_speed(revolutions, revolutions)
  delay(t)
  alvik.set_wheels_speed(0, 0)
  delay(5000) # Optional pause

def cleanup():
  alvik.stop()

start(setup, loop, cleanup)
```

- **Experiment:** Place the back of your robot against a notebook once again. Update your script using the example above and run the program. Using a measuring tool, check the total distance from the notebook's edge to the back of your robot.

Compare your measurements to the calculated **Travel Distance** value of 356 mm. How closely does the real measurement match the distance calculated by our formula?

Try again with different values for **revolutions** and **t**. Measure the results and compare them to your estimated **Travel Distance** values.

- **Challenge:** Can you update the formula to solve for the **revolutions** needed to travel 300 mm in 15 seconds? What if we want to travel a distance of 200 mm at a speed of 5 RPM, and we need the value for **t**? If you can solve these, try testing them in your script to see if your calculations are right!

Chicken Run

By now, you should understand how to use formulas to calculate **Travel Distance**, **revolutions**, and **t**, as well as how we include these variables and their values within our scripts to quickly achieve desired outcomes.

To consolidate what we have learned, let's organize a fun competition - a chicken run!

Your teacher will place two barriers on the desk or floor at an exact distance from each other. Your mission is to compete against friends by quickly racing Alvik from one side to the other. Whoever gets closest to the opposite barrier is the winner, but if your robot accidentally touches it, you lose instantly!

On your marks, get set, code!