



**WELCOME TO THE  
CREATIVE ROBOTICS CLUB**

# WHERE CAN I FIND THE SLIDES FROM CREATIVE ROBOTICS CLUB?

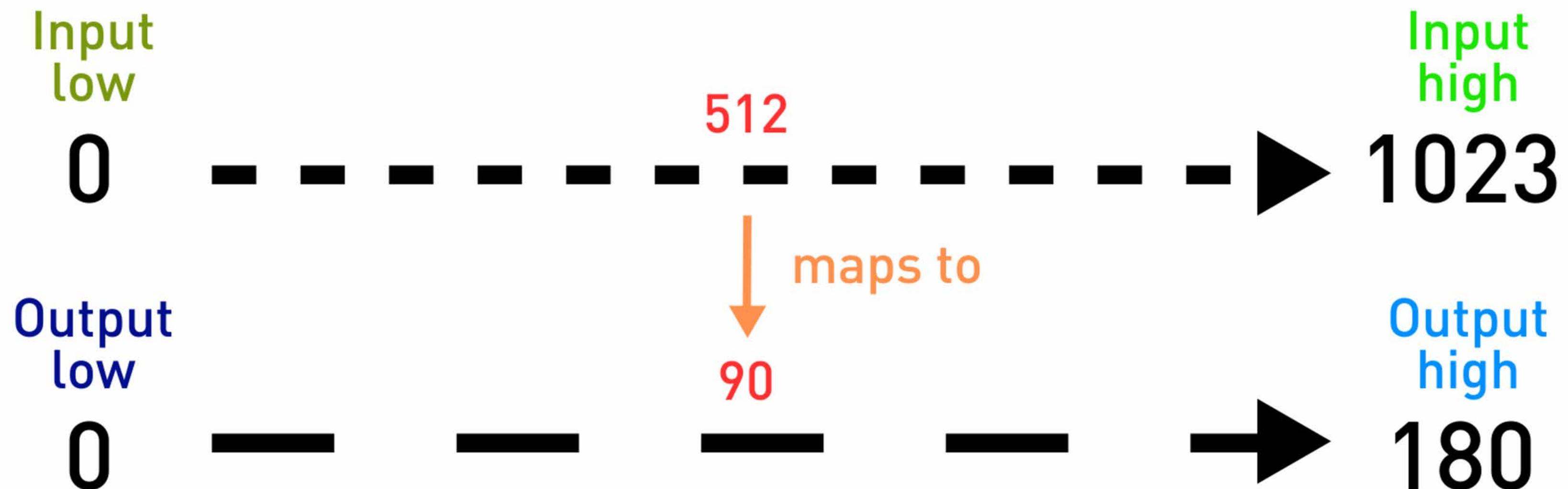


[github.com/instasquid/creativeroboticsclub](https://github.com/instasquid/creativeroboticsclub)

BUT FIRST LETS TALK ABOUT...

LAST WEEK

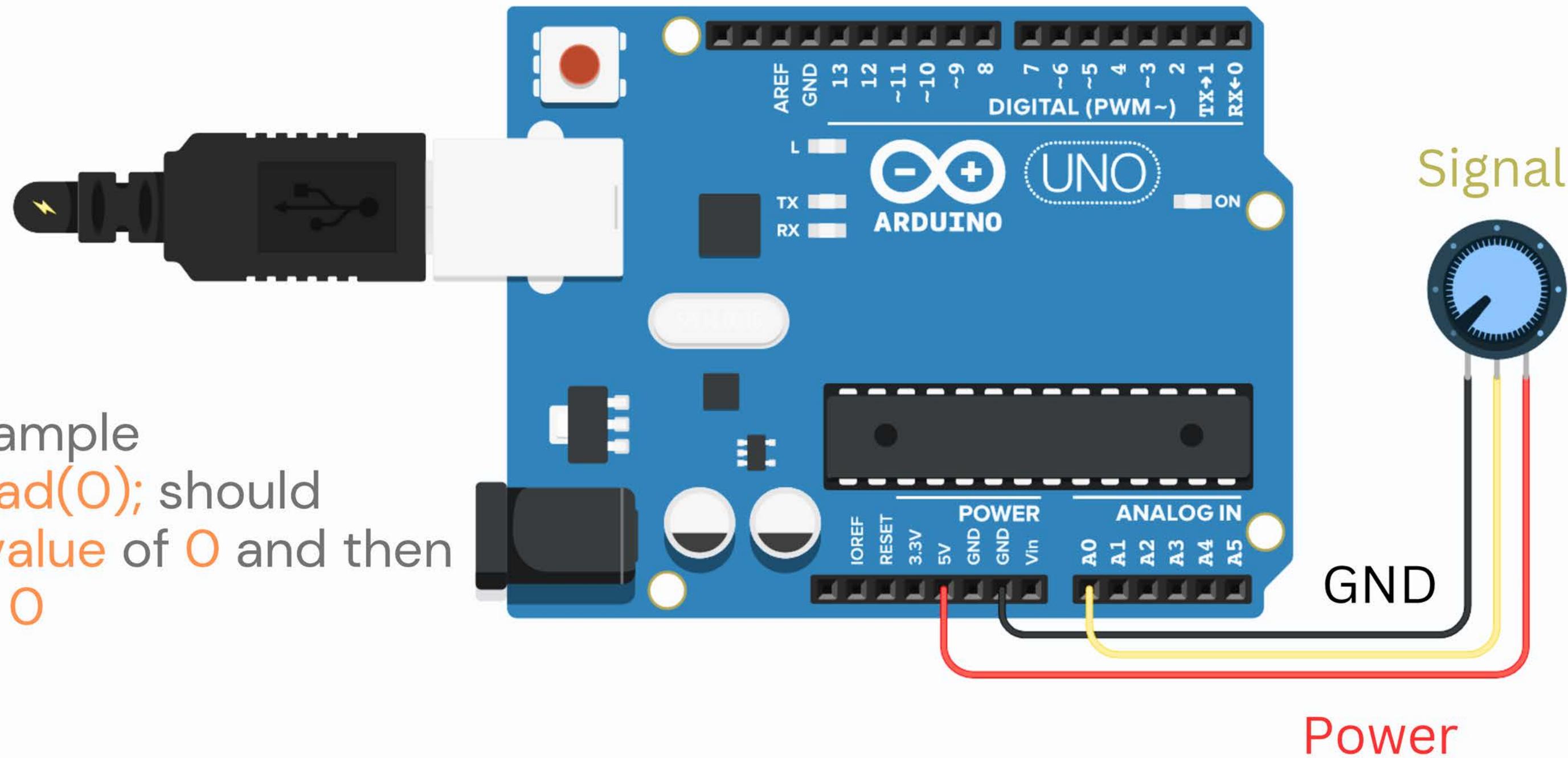
**map( ) lets you take one range of values and scale them to match another**



**map(value, inputLow, inputHigh, outputLow, outputHigh);**

# ARDUINO

In this example  
`analogRead(0);` should  
return a `value` of `0` and then  
map it to `0`



# NOW WHEN YOU RUN THIS CODE...

A list of values will start appearing in the **Serial Monitor**.

Now we know what's going on!

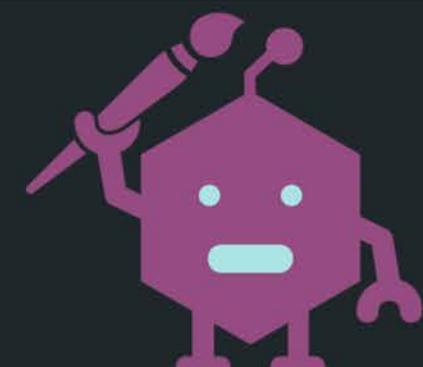
We can use this information to drive interaction in our system.

sketch\_mar2a.ino

```
1 void setup() {
2     Serial.begin(115200); //Start the Serial monitor
3 }
4
5 void loop() {
6     //Read Analog In pin 0
7     int value = analogRead(A0);
8     //Map the reading to a number between 0-180
9     value = map(value, 0, 1023, 0, 180);
10    //Add text so we know what we're looking at
11    Serial.print("Value = ");
12    //Write the value
13    Serial.println(value);
14    //Wait 15ms and do it again
15    delay(15);
16 }
```

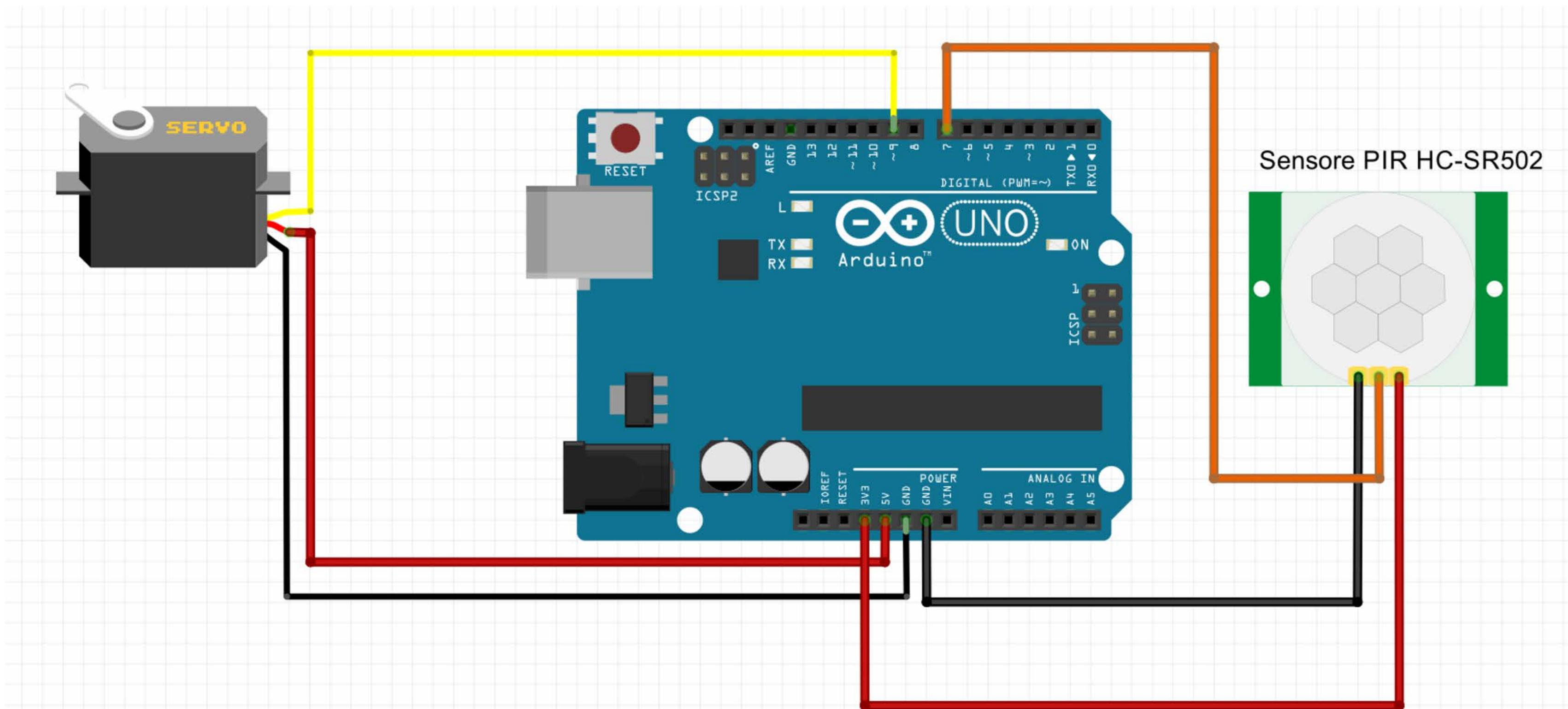
Serial Monitor ×

Message (Enter to send message to 'Arduino U...') New Line 115200 baud



**S = SIGNAL V = VOLTAGE and G = GND**

So, now we know how to wire digital sensors up!



**WHAT ARE WE DOING TODAY AT THE  
CREATIVE ROBOTICS CLUB?**

**ULTRASONIC DISTANCE SENSORS  
REUSING CODE  
SMOOTHING DATA  
IF STATEMENTS (AGAIN)  
WHAT'S NEXT?**

# ARDUINO

So lets talk about other types of sensors.

There Arduino has many different types of sensor for all sorts of applications: temperature, flexibility, sound, to check for the presence of a magnetic field, to check light levels, flame sensors, and more.

Last week we looked at sensors to detect sound, temperature, and light, lets take a quick look at a more complex sensor today.



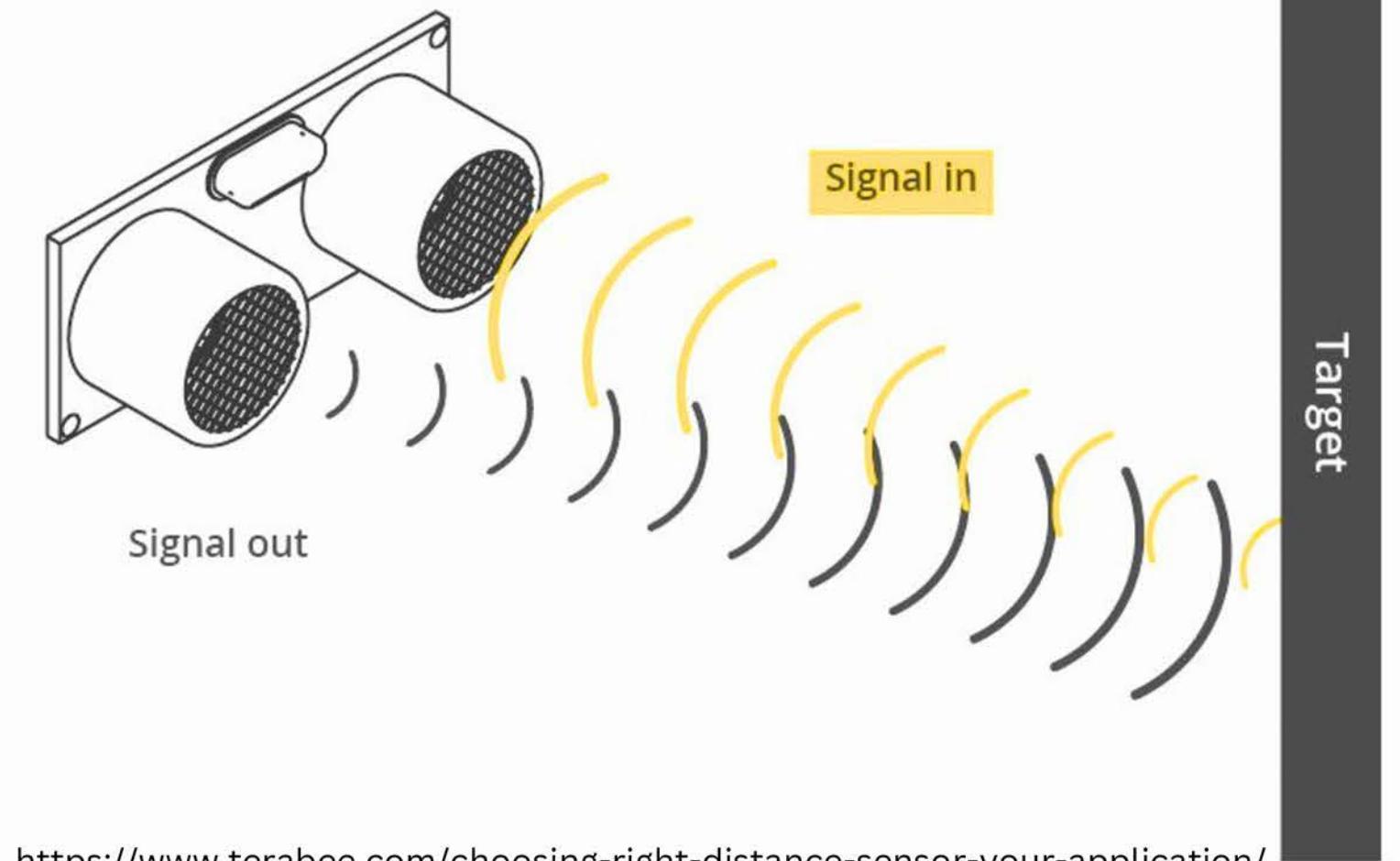
[HTTPS://UK.ROBOTSHOP.COM/PRODUCTS/37-MODULES-SENSOR-KIT-ARDUINO-V2](https://UK.ROBOTSHOP.COM/PRODUCTS/37-MODULES-SENSOR-KIT-ARDUINO-V2)

# ULTRASONIC DISTANCE SENSOR

If we wanted to sense how far something was from our robot we might use an **ultrasonic distance sensor**.

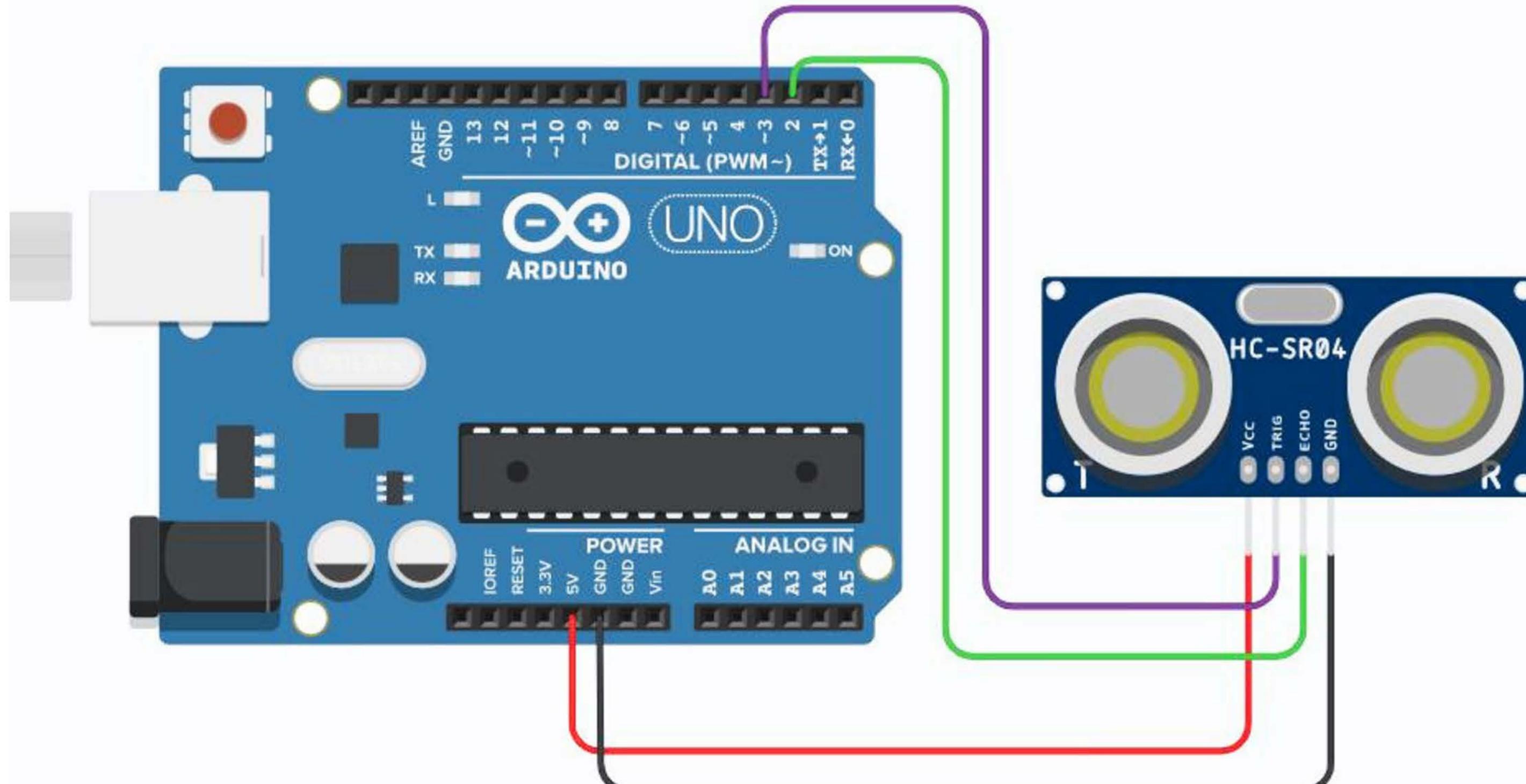
This sensor sends out a burst of sound that is above human hearing and listens for the signal's return. The length of time between sending out the signal and hearing it lets it know just how far away an object in front of it is.

Plus it looks like eyes, and that's cute.



# HOW DO WE WIRE IT UP?

Trig and Echo pins are digital, and you should be able to decipher Vcc and GND by now.



# CODE FOR THE HC-SR04

Before you copy this code, lets take a look at it.

We can use **#define** to declare our pin variables, because we are never going to change their numbers while the program is running. This is a more memory efficient way to code.

The rest of the code sets the **trigger** pin **LOW**, then sets it **HIGH** for **10ms**, and then sets it back to **LOW** again. Simple, but it looks like a mess...

sketch\_mar12a.ino

```
1 //We can use #define to set our pin variables,  
2 //because we are never going to change them  
3 #define trigPin 4  
4 #define echoPin 5  
5  
6 void setup() {  
7     Serial.begin(115200);  
8     //Trig sends the signal  
9     pinMode(trigPin, OUTPUT);  
10    //Echo receives the reflected signal  
11    pinMode(echoPin, INPUT);  
12 }  
13  
14 void loop() {  
15     //Long is a type of variable that holds really big numbers  
16     long duration, distance;  
17     //Make sure the Trig pin is off  
18     digitalWrite(trigPin, LOW); // Added this line  
19     delayMicroseconds(2); // Added this line  
20     //Send the signal  
21     digitalWrite(trigPin, HIGH);  
22     delayMicroseconds(10); // Added this line  
23     //Turn off the signal  
24     digitalWrite(trigPin, LOW);  
25     //Read the reflected signal  
26     duration = pulseIn(echoPin, HIGH);  
27     //Convert it to CM  
28     distance = (duration/2) / 29.1;  
29     Serial.print("Distance: ");  
30     Serial.println(distance);  
31 }
```

LET'S MAKE THE  
CODE EASIER TO  
READ AND REUSE



# FUNCTIONS

So far we have written a lot of code to read sensors and make servos move, but it all gets crowded and it hard to tell apart. Currently, when we reuse code we're either copy and pasting it or typing it out again.

Is there a way we could make it easier to read and easier to reuse later?

**void setup()** and **void loop()** are called **Functions**, and we can write our own.

sketch\_mar12a.ino

```
1 //We can use #define to set our pin variables,  
2 //because we are never going to change them  
3 #define trigPin 4  
4 #define echoPin 5  
5  
6 void setup() {  
7     Serial.begin(115200);  
8     //Trig sends the signal  
9     pinMode(trigPin, OUTPUT);  
10    //Echo receives the reflected signal  
11    pinMode(echoPin, INPUT);  
12 }  
13  
14 void loop() {  
15     //Long is a type of variable that holds really big numbers  
16     long duration, distance;  
17     //Make sure the Trig pin is off  
18     digitalWrite(trigPin, LOW); // Added this line  
19     delayMicroseconds(2); // Added this line  
20     //Send the signal  
21     digitalWrite(trigPin, HIGH);  
22     delayMicroseconds(10); // Added this line  
23     //Turn off the signal  
24     digitalWrite(trigPin, LOW);  
25     //Read the reflected signal  
26     duration = pulseIn(echoPin, HIGH);  
27     //Convert it to CM  
28     distance = (duration/2) / 29.1;  
29     Serial.print("Distance: ");  
30     Serial.println(distance);  
31 }
```

# MY FIRST FUNCTION

Before we setup our USD sensor, lets create a new function to put all our code in. We are also going to learn a new variable type:

**long measureDistance() {}**

Wait, why is this not  
**void distance(){} ??**

01-USCode.ino

```
1 //We can use #define to set our pin variables,
2 //because we are never going to change them
3 #define trigPin 4
4 #define echoPin 5
5
6 void setup() {
7   Serial.begin(115200);
8   //Trig sends the signal
9   pinMode(trigPin, OUTPUT);
10  //Echo receives the reflected signal
11  pinMode(echoPin, INPUT);
12 }
13
14 void loop() {
15   long distance = measureDistance();
16   Serial.print("Distance: ");
17   Serial.println(distance);
18 }
19
20 long measureDistance()
21 {
22   //Long is a type of variable that holds really big numbers
23   long duration, result;
24   //Make sure the Trig pin is off
25   digitalWrite(trigPin, LOW); // Added this line
26   delayMicroseconds(2); // Added this line
27   //Send the signal
28   digitalWrite(trigPin, HIGH);
29   delayMicroseconds(10); // Added this line
30   //Turn off the signal
31   digitalWrite(trigPin, LOW);
32   //Read the reflected signal
33   duration = pulseIn(echoPin, HIGH);
34   //Convert it to CM
35   result = (duration/2) / 29.1;
36   return result;
37 }
```

# MY FIRST FUNCTION

Functions can return information to us, and the first word we use to define our function tells us what information they're going to return.

**void** returns nothing at all  
**long** returns a really long number, too big to fit into a **float** or **int**

We could also make functions using the other variable types we've learned so far: **bool**, **int** and **float** and they would return a number that matches their type.

01-USCode.ino

```
1 //We can use #define to set our pin variables,  
2 //because we are never going to change them  
3 #define trigPin 4  
4 #define echoPin 5  
5  
6 void setup() {  
7   Serial.begin(115200);  
8   //Trig sends the signal  
9   pinMode(trigPin, OUTPUT);  
10  //Echo receives the reflected signal  
11  pinMode(echoPin, INPUT);  
12 }  
13  
14 void loop() {  
15   long distance = measureDistance();  
16   Serial.print("Distance: ");  
17   Serial.println(distance);  
18 }  
19  
20 long measureDistance()  
21 { //Long is a type of variable that holds really big numbers  
22   long duration, result;  
23   //Make sure the Trig pin is off  
24   digitalWrite(trigPin, LOW); // Added this line  
25   delayMicroseconds(2); // Added this line  
26   //Send the signal  
27   digitalWrite(trigPin, HIGH);  
28   delayMicroseconds(10); // Added this line  
29   //Turn off the signal  
30   digitalWrite(trigPin, LOW);  
31   //Read the reflected signal  
32   duration = pulseIn(echoPin, HIGH);  
33   //Convert it to CM  
34   result = (duration/2) / 29.1;  
35   return result;  
36 }  
37 }
```

# MY FIRST FUNCTION

To make sure our **long** function works correctly we need to make sure it ends with an instruction to return a **long** value.

**return result;** achieves this here.

Back in **void loop()** we need to declare a variable that will we will use to save the returned result of our measurement .

**long distance = measureDistance();** is our way of saying “We want to save a number into a variable called “distance” and we want it to be the result of our function.

01-USCode.ino

```
1 //We can use #define to set our pin variables,  
2 //because we are never going to change them  
3 #define trigPin 4  
4 #define echoPin 5  
5  
6 void setup() {  
7   Serial.begin(115200);  
8   //Trig sends the signal  
9   pinMode(trigPin, OUTPUT);  
10  //Echo receives the reflected signal  
11  pinMode(echoPin, INPUT);  
12 }  
13  
14 void loop() {  
15   long distance = measureDistance();  
16   Serial.print("Distance: ");  
17   Serial.println(distance);  
18 }  
19  
20 long measureDistance()  
21 { //Long is a type of variable that holds really big numbers  
22   long duration, result;  
23   //Make sure the Trig pin is off  
24   digitalWrite(trigPin, LOW); // Added this line  
25   delayMicroseconds(2); // Added this line  
26   //Send the signal  
27   digitalWrite(trigPin, HIGH);  
28   delayMicroseconds(10); // Added this line  
29   //Turn off the signal  
30   digitalWrite(trigPin, LOW);  
31   //Read the reflected signal  
32   duration = pulseIn(echoPin, HIGH);  
33   //Convert it to CM  
34   result = (duration/2) / 29.1;  
35   return result;  
36 }  
37
```

# HOW DO I REUSE THIS?

What use is our function?

It makes our code tidier. When we look at `void loop()` we can get a clear idea in a more human language of what it's doing.

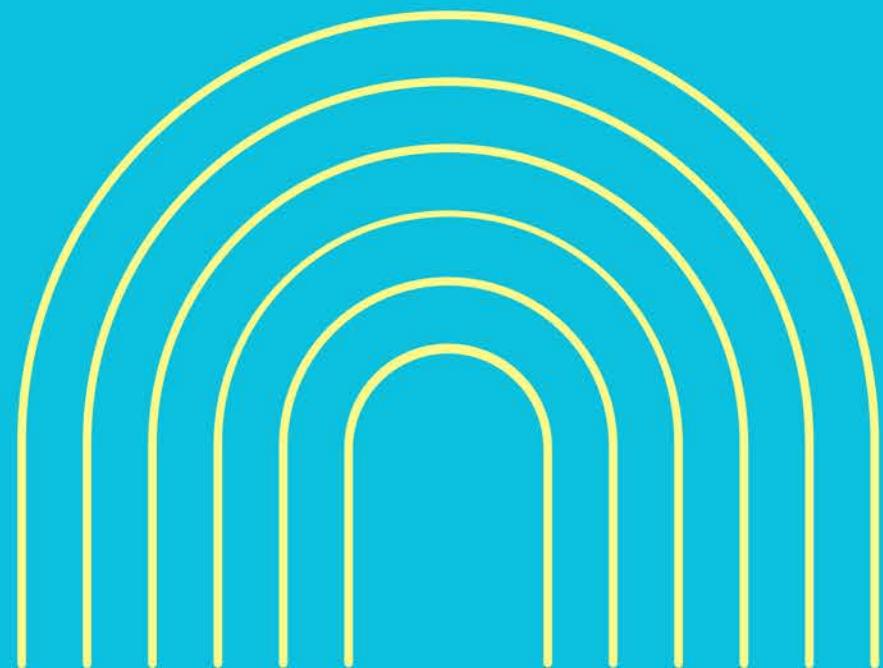
It also makes code easier to reuse. Next time we use a distance sensor we can copy and paste our `measureDistance()` function [and pin definitions] into the new code and know exactly how it works, and what result it will return.

How efficient!

01-USCode.ino

```
1 //We can use #define to set our pin variables,  
2 //because we are never going to change them  
3 #define trigPin 4  
4 #define echoPin 5  
5  
6 void setup() {  
7   Serial.begin(115200);  
8   //Trig sends the signal  
9   pinMode(trigPin, OUTPUT);  
10  //Echo receives the reflected signal  
11  pinMode(echoPin, INPUT);  
12 }  
13  
14 void loop() {  
15   long distance = measureDistance();  
16   Serial.print("Distance: ");  
17   Serial.println(distance);  
18 }  
19  
20 long measureDistance()  
21 { //Long is a type of variable that holds really big numbers  
22   long duration, result;  
23   //Make sure the Trig pin is off  
24   digitalWrite(trigPin, LOW); // Added this line  
25   delayMicroseconds(2); // Added this line  
26   //Send the signal  
27   digitalWrite(trigPin, HIGH);  
28   delayMicroseconds(10); // Added this line  
29   //Turn off the signal  
30   digitalWrite(trigPin, LOW);  
31   //Read the reflected signal  
32   duration = pulseIn(echoPin, HIGH);  
33   //Convert it to CM  
34   result = (duration/2) / 29.1;  
35   return result;  
36 }  
37 }
```

LET'S TRY IT OUT



OKAY, THE VALUES  
FROM THIS  
SENSOR JUMP  
EVERYWHERE



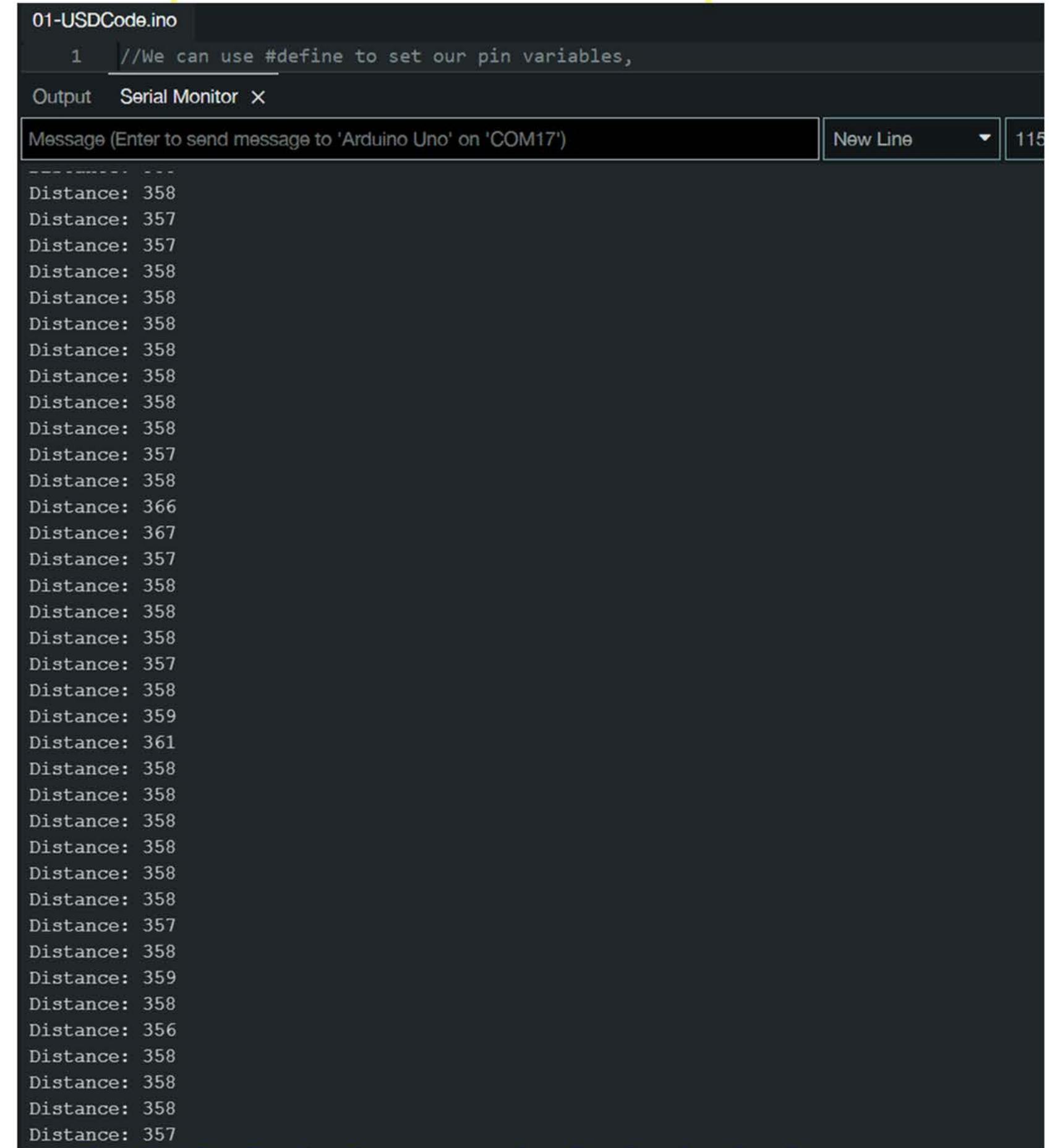
# DATA SMOOTHING

In the example to the right values jump between 357 and 367. That's a lot of variation!

If we wanted to use it with `map()` or an `if()` statement we run the risk of triggering inappropriate responses with it.

What can we do?

Sometimes we might “smooth” data from an input device. To do this we need to keep track of the data we’re getting in and the data that came in before it.



The screenshot shows the Arduino IDE interface. The top bar displays "01-USDCODE.ino". The code area contains a single line: "1 //We can use #define to set our pin variables,". Below the code is a status bar with "Output" and "Serial Monitor X". The "Serial Monitor" tab is active, showing a message field "Message (Enter to send message to 'Arduino Uno' on 'COM17')". The main window displays a continuous stream of data: "Distance: 358", "Distance: 357", "Distance: 357", "Distance: 358", "Distance: 357", "Distance: 358", "Distance: 366", "Distance: 367", "Distance: 357", "Distance: 358", "Distance: 358", "Distance: 358", "Distance: 357", "Distance: 358", "Distance: 359", "Distance: 361", "Distance: 358", "Distance: 358", "Distance: 359", "Distance: 358", "Distance: 357", "Distance: 358", "Distance: 358", "Distance: 358", "Distance: 357", "Distance: 358", "Distance: 359", "Distance: 358", "Distance: 356", "Distance: 358", "Distance: 358", "Distance: 358", "Distance: 357". The status bar at the bottom right shows "New Line" and the number "115".

# DATA SMOOTHING

We need a new global variable that stores our last result. Here I've added `long lastDistance;` for that purpose.

We create our distance reading using **75%** of our last distance reading and **25%** of our new distance reading

`distance = (lastDistance * 0.75) +  
(distance * 0.25);`

Writing `lastDistance * 0.75` is another way of saying **use 75% of lastDistance**

01-USDCODE.ino

```
1 //We can use #define to set our pin variables,  
2 //because we are never going to change them  
3 #define trigPin 4  
4 #define echoPin 5  
5 long lastDistance;  
6  
7 void setup() {  
8   Serial.begin(115200);  
9   //Trig sends the signal  
10  pinMode(trigPin, OUTPUT);  
11  //Echo receives the reflected signal  
12  pinMode(echoPin, INPUT);  
13 }  
14  
15 void loop() {  
16   long distance = measureDistance();  
17   distance = (lastDistance * 0.75) + (distance * 0.25);  
18   lastDistance = distance;  
19   Serial.print("Distance: "); Serial.println(distance);  
20 }  
21  
22 long measureDistance()  
23 {  
24   //Long is a type of variable that holds really big numbers  
25   long duration, result;  
26   //Make sure the Trig pin is off  
27   digitalWrite(trigPin, LOW); // Added this line  
28   delayMicroseconds(2); // Added this line  
29   //Send the signal  
30   digitalWrite(trigPin, HIGH);  
31   delayMicroseconds(10); // Added this line  
32   //Turn off the signal  
33   digitalWrite(trigPin, LOW);  
34   //Read the reflected signal  
35   duration = pulseIn(echoPin, HIGH);  
36   //Convert it to CM  
37   result = (duration/2) / 29.1;  
38 }
```

# DATA SMOOTHING

Next we need to update our **lastDistance** variable for next time.

**lastDistance = distance;** takes care of that.

Now we can load this onto the Arduino and see how it changes the result. Yes, there is still some noise, but it is greatly reduced.

You can experiment with smoothing by changing the amount you \* **lastDistance** and **distance** by, but it should always add up to **1.0**

01-USDCODE.ino

```
1 //We can use #define to set our pin variables,  
2 //because we are never going to change them  
3 #define trigPin 4  
4 #define echoPin 5  
5 long lastDistance;  
6  
7 void setup() {  
8   Serial.begin(115200);  
9   //Trig sends the signal  
10  pinMode(trigPin, OUTPUT);  
11  //Echo receives the reflected signal  
12  pinMode(echoPin, INPUT);  
13 }  
14  
15 void loop() {  
16   long distance = measureDistance();  
17   distance = (lastDistance * 0.75) + (distance * 0.25);  
18   lastDistance = distance;  
19   Serial.print("Distance: "); Serial.println(distance);  
20 }  
21  
22 long measureDistance()  
23 {  
24   //Long is a type of variable that holds really big numbers  
25   long duration, result;  
26   //Make sure the Trig pin is off  
27   digitalWrite(trigPin, LOW); // Added this line  
28   delayMicroseconds(2); // Added this line  
29   //Send the signal  
30   digitalWrite(trigPin, HIGH);  
31   delayMicroseconds(10); // Added this line  
32   //Turn off the signal  
33   digitalWrite(trigPin, LOW);  
34   //Read the reflected signal  
35   duration = pulseIn(echoPin, HIGH);  
36   //Convert it to CM  
37   result = (duration/2) / 29.1;  
38 }
```

“WHAT DO WE DO  
NOW WE HAVE  
MORE RELIABLE  
DATA?”



# REVISITING IF{} STATEMENTS

Last week we looked at if statements. Lets revisit that now.

We learned we can use

```
if (value == true) {  
    //run commands  
} else {  
    //run other commands  
}
```

We make a choice, but we only have two options here. Can we do more?

```
1 #include <Servo.h>  
2  
3 Servo servo;  
4  
5 void setup() {  
6     // put your setup code here, to run once:  
7     Serial.begin(115200);  
8  
9     pinMode(2, INPUT); // we need to set the pin mode this time!  
10  
11    servo.attach(9);  
12 }  
13  
14 void loop() {  
15     // put your main code here, to run repeatedly:  
16     int value = digitalRead(2);  
17  
18     Serial.print("Value = ");  
19     Serial.println(value);  
20  
21     if (value == 1) {  
22         servo.write(90);  
23     } else {  
24         servo.write(0);  
25     }  
26  
27     delay(15);  
28 }
```

# REVISITING IF{} STATEMENTS

Yes, of course we can do more than that!

We can use other mathematic symbols to test a value and decide what to do.

- > [greater than]
- < [less than]
- != [does not equal]
- >= [greater than or equal to]
- <= [less than or equal to]

All these symbols are useful with **if()** statements

```
1 #include <Servo.h>
2
3 Servo servo;
4
5 void setup() {
6     // put your setup code here, to run once:
7     Serial.begin(115200);
8
9     pinMode(2, INPUT); // we need to set the pin mode this time!
10
11    servo.attach(9);
12 }
13
14 void loop() {
15     // put your main code here, to run repeatedly:
16     int value = digitalRead(2);
17
18     Serial.print("Value = ");
19     Serial.println(value);
20
21     if (value == 1) {
22         servo.write(90);
23     } else {
24         servo.write(0);
25     }
26
27     delay(15);
28 }
```

# REVISITING IF{} STATEMENTS

Lets just focus on our **void loop()** for now.

First we test:

**if (distance > 200) {**

We can use this to define our robots behaviour for all values over 200.

Next:

**else if ()**

Until now **else** has just been used as “**in all other cases, do this instead**” but we can add another **if()** statement there with **else if()**! If our first if statement is false, we can try again with a new test.

01-USDCODE.ino

```
15 void loop() {
16     long distance = measureDistance();
17     distance = (lastDistance * 0.75) + (distance * 0.25);
18     lastDistance = distance;
19
20     if (distance > 200) {
21         // Code for if something is far away
22     } else if (distance < 200 && distance > 100) {
23         //Code for if something is a medium distance
24     } else if (distance < 100 && distance > 40) {
25         //Code if something is close
26     } else {
27         //Code if someting is REALLY close
28     }
29
30     Serial.print("Distance: "); Serial.println(distance);
31 }
32
33 long measureDistance()
34 {
35     //Long is a type of variable that holds really big numbers
36     long duration, result;
37     //Make sure the Trig pin is off
38     digitalWrite(trigPin, LOW); // Added this line
39     delayMicroseconds(2); // Added this line
40     //Send the signal
41     digitalWrite(trigPin, HIGH);
42     delayMicroseconds(10); // Added this line
43     //Turn off the signal
44     digitalWrite(trigPin, LOW)
```

## REVISITING IF{} STATEMENTS

The whole line reads:

```
else if (distance > 200 && distance >  
100) { }
```

In English we sometimes use **&** as a shortcut for the word “**and**”.

In programming using **&&** means “**test the values on our left, and the values to the right. If both are true then you can run this code.**”

Any code we put between the curly brackets **{ }** will only run if distance is both **less than 200** and **more than 100**, so **between 101 and 199**.

01-USDCODE.ino

```
15 void loop() {  
16     long distance = measureDistance();  
17     distance = (lastDistance * 0.75) + (distance * 0.25);  
18     lastDistance = distance;  
19  
20     if (distance > 200) {  
21         // Code for if something is far away  
22     } else if (distance < 200 && distance > 100) {  
23         //Code for if something is a medium distance  
24     } else if (distance < 100 && distance > 40) {  
25         //Code if something is close  
26     } else {  
27         //Code if someting is REALLY close  
28     }  
29  
30     Serial.print("Distance: "); Serial.println(distance);  
31 }  
32  
33 long measureDistance()  
34 {  
35     //Long is a type of variable that holds really big numbers  
36     long duration, result;  
37     //Make sure the Trig pin is off  
38     digitalWrite(trigPin, LOW); // Added this line  
39     delayMicroseconds(2); // Added this line  
40     //Send the signal  
41     digitalWrite(trigPin, HIGH);  
42     delayMicroseconds(10); // Added this line  
43     //Turn off the signal  
44     digitalWrite(trigPin, LOW);  
45  
46     //Get the duration of the pulse  
47     duration = pulseIn(echoPin, HIGH);  
48  
49     //Calculate the distance  
50     result = (duration * 0.0343) / 2;  
51  
52     return result;  
53 }
```

# REVISITING IF{} STATEMENTS

Ordering is important here.

We want to test only in ascending or descending values. This makes it easier to read and makes sure we're not accidentally catching the wrong values.

Also, we can only use an **else** after an **if** or an **else if**, but not in-between an **if** and an **else if**.

We write the code as:  
**if** then **else if** then **else**.

01-USDCODE.ino

```
15 void loop() {
16     long distance = measureDistance();
17     distance = (lastDistance * 0.75) + (distance * 0.25);
18     lastDistance = distance;
19
20     if (distance > 200) {
21         // Code for if something is far away
22     } else if (distance < 200 && distance > 100) {
23         //Code for if something is a medium distance
24     } else if (distance < 100 && distance > 40) {
25         //Code if something is close
26     } else {
27         //Code if someting is REALLY close
28     }
29
30     Serial.print("Distance: "); Serial.println(distance);
31 }
32
33 long measureDistance()
34 {
35     //Long is a type of variable that holds really big numbers
36     long duration, result;
37     //Make sure the Trig pin is off
38     digitalWrite(trigPin, LOW); // Added this line
39     delayMicroseconds(2); // Added this line
40     //Send the signal
41     digitalWrite(trigPin, HIGH);
42     delayMicroseconds(10); // Added this line
43     //Turn off the signal
44     digitalWrite(trigPin, LOW);
45 }
```

NEXT WEEK IS WEEK 6

NO CLUB OR CLASSES

NEXT WEEK