**Table of Contents**

Getting Started……………………………………………………………………………………

Building the Box…………………………………………………………………………

Breadboard………………………………………………………………………

HX711 Scale…………………………………………………………………….

I2C LCD Screen…………………………………………………………………

Pushbutton………………………………………………………………………

Waterpump………………………………………………………………………

Printing the Stand

3D Printing Must-Knows……………………………………………………….

External Components…………………………………………………………..

Writing the Code

Downloading Arduino IDE……………………………………………………..

Navigating Arduino IDE………………………………………………………..

Write the Code………………………………………………………………….

Break the Code…………………………………………………………………

Calling of Libraries……………………………………………………

Definition of Variables…………………………………………………

void setup().............................................................................................

void loop()..............................................................................................

Calibration Code……………………………………………………………….

Testing……………………..……………………………………………………………………..

Scale Testing…………………………………………………………………………….

Pump Testing…………………………………………………………………………….

Starting an Experiment

Setting the scene…………………………………………………………………………

Sources of error………………………………………………………………………….

Daily Life with the Box

Logging Measurements………………………………………………………………….

Installing Data Streamer…………………………………………………………

Connecting to Data Streamer……………………………………………………

During the Experiment……………………………………………………………………

Printable Checklist for Logging Measurements…………………………………………………..

Maintenance

Short-Term Maintenance

Long-Term Maintenance

Improvements

Excel vs. Raspi

Unsolved Mysteries

# **Building the Box**

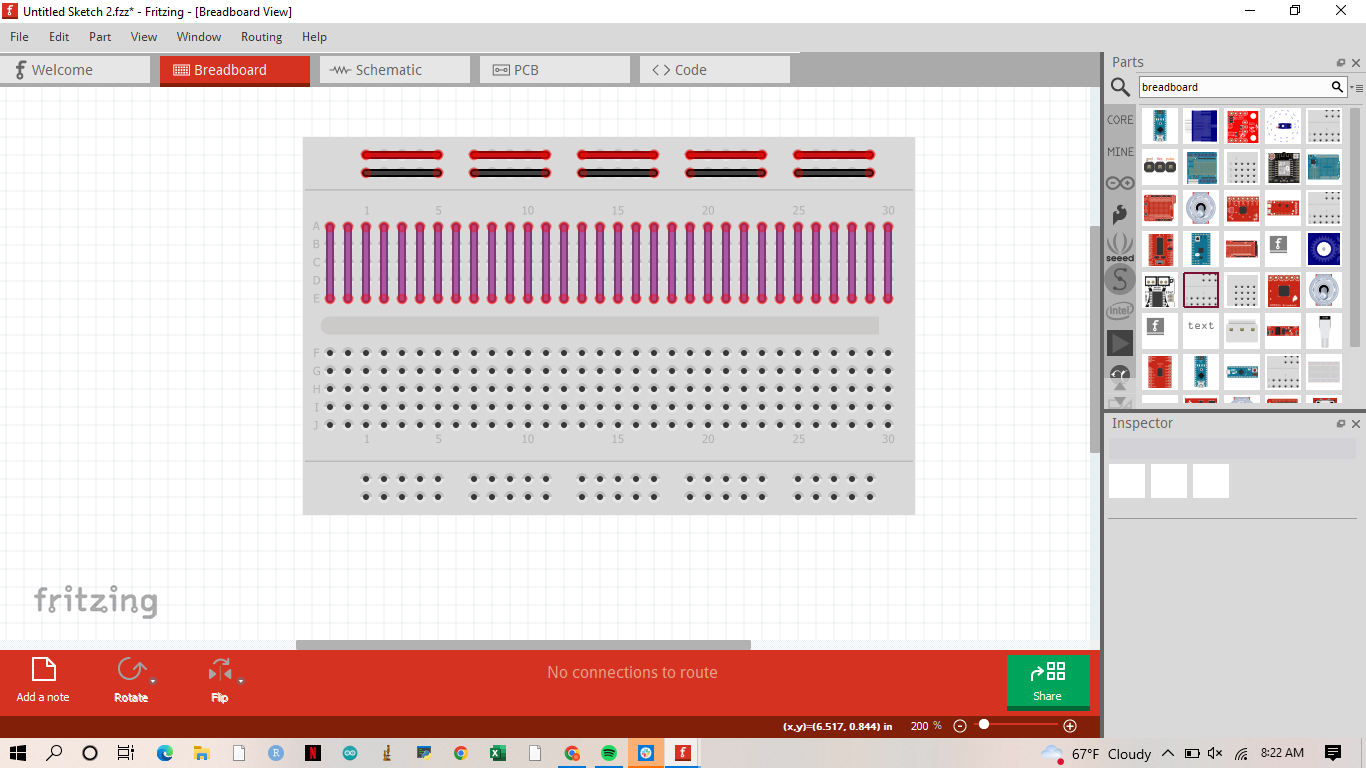
|  | **Arduino Uno R3** |
| --- | --- |
|  | **5kg Load Cell and Scale** |
|  | **Small Breadboard** |
|  | **Male to Male Jumper Wires** |
|  | **Female to Male Jumper Wires** |
|  | **LCD Screen 16x2 characters I2C** |
|  | **Twice 1A 250V AC Push Button** |
|  | **6mm ID X 8mm OD Silicone Rubber Tubing** |
|  | **Large Jar minus the Pickles** |
|  | **Gikifun DC 3V-5V Submersible Water Pump** |
|  | **Black Plastic Plant Saucer** |
|  | **PLA filament** |

## 

## Step 1: Connect Arduino to Breadboard

After you have assembled all of your items, it is time to put together the box itself! If you are just beginning your journey with the world of Arduinos, go one accessory at a time. Connect what is easiest to connect to help you get a feel for how the circuit and computer work. In this guide, we will connect the breadboard to the Arduino.

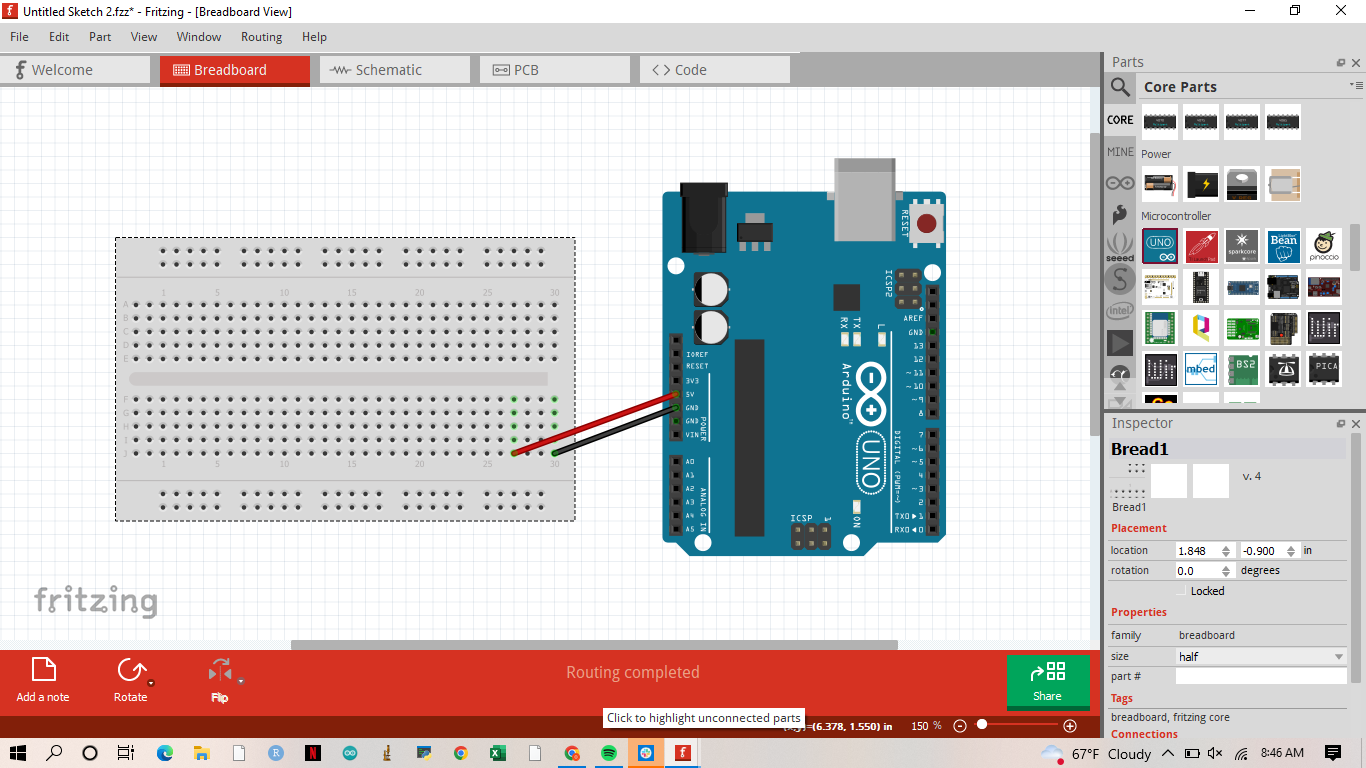
Since we have so many accessories in this project, connecting them directly to the Arduino becomes challenging, since there are only so many power and sensor ports available. This is where the breadboard comes in handy. The way a breadboard works is by having metal strips connecting multiple rows together. This allows you to plug in, for instance, power from the Arduino into the beginning of one row and supply that power to anything else plugged into that row. See below for a schematic of this concept. Connections on the upper half of the breadboard hold true for the lower half as well.



**Figure 1.** Top view of a breadboard. Purple, red and black lines show holes that are connected by metal strips and carry the same electricity between them. This function allows you to plug multiple things into one lane. Note that this figure only shows connections for one half of the breadboard, these connections are mirrored on the other half.

The first connection between the Arduino and the breadboard will be the power supply. This will allow us to continue plugging in accessories as we go along. Another useful tip is to try to keep the wire coloring consistent with all accessories. For example, always use black wires for the ground, red wires for 5V, and some other color for any input/output wires. This will not only help you keep track of which wire is which, but also will help you gain a better understanding of how the circuit works and will make things easier to troubleshoot if something goes wrong.

You want to choose two rows closer to the breadboard's exterior, allowing you to have room to connect everything else you need. At this point, your setup should look like this:

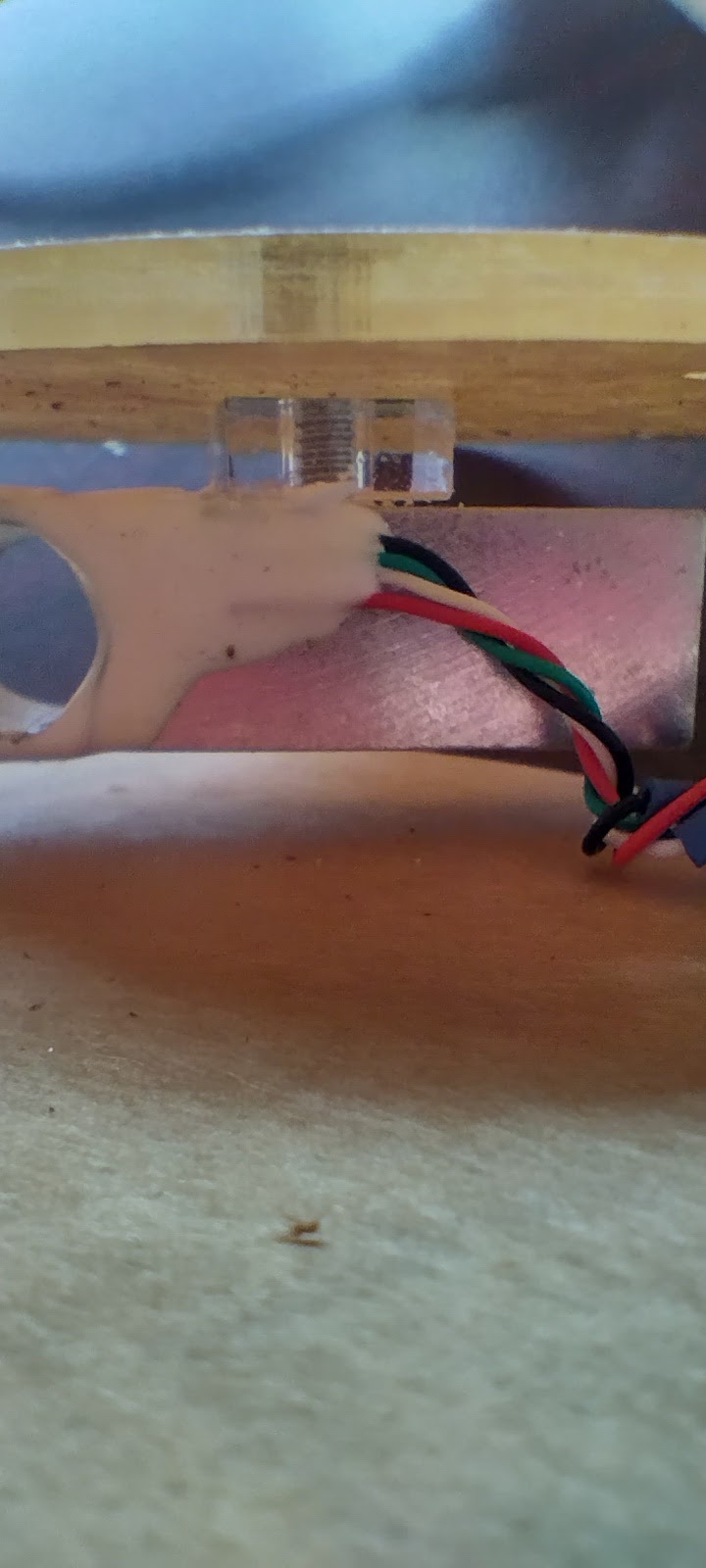


## 

## Step 2: Connect HX711 Scale

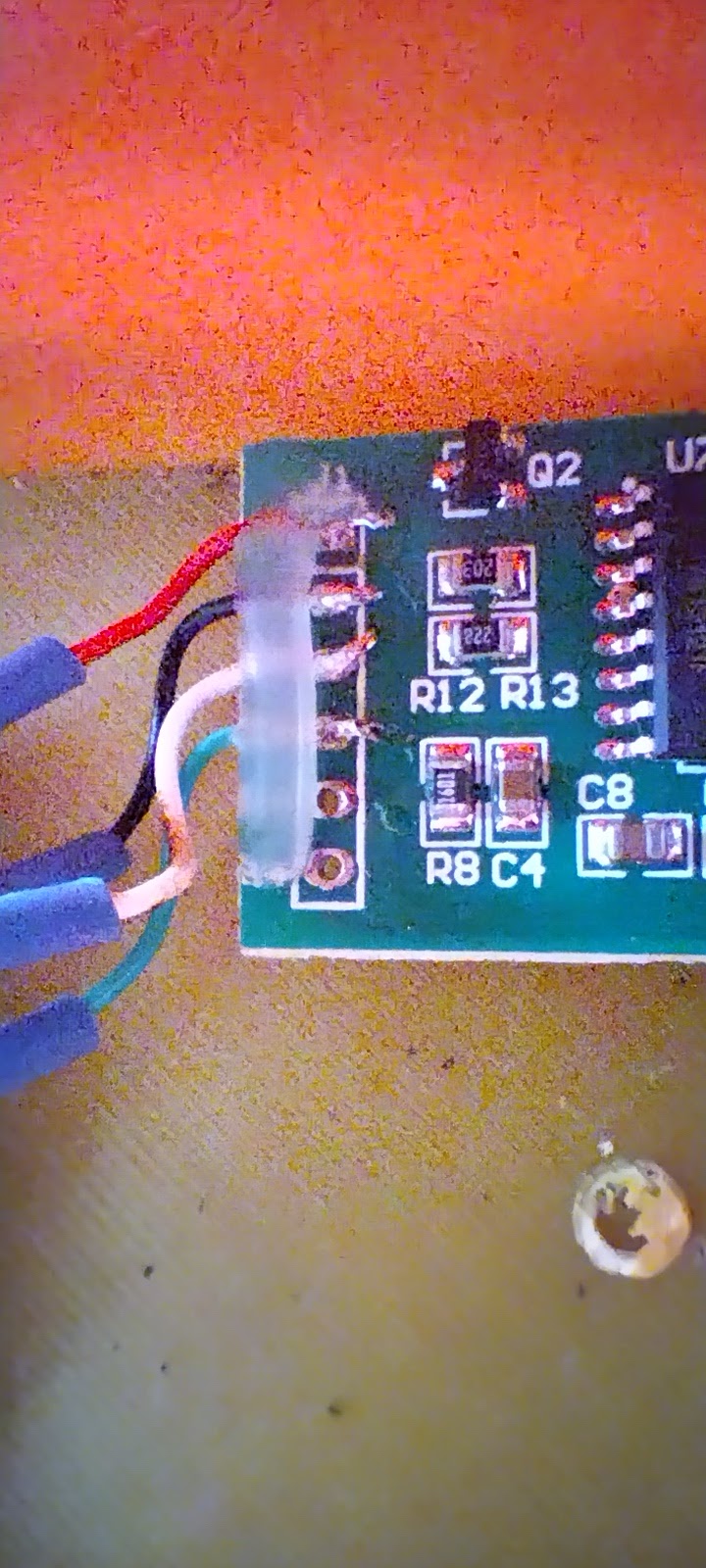
Next, we will work through how to connect the scale. This is an important part since it is the sensor that will be informing the computer's choice of whether to water or not and by how much. Therefore, it will need to be as accurate as possible. In this guide, a 5 kg load cell is used, since it provides the type of resolution needed for the average weight that a given experiment might be measuring. This may vary depending on your experiment details, etc.

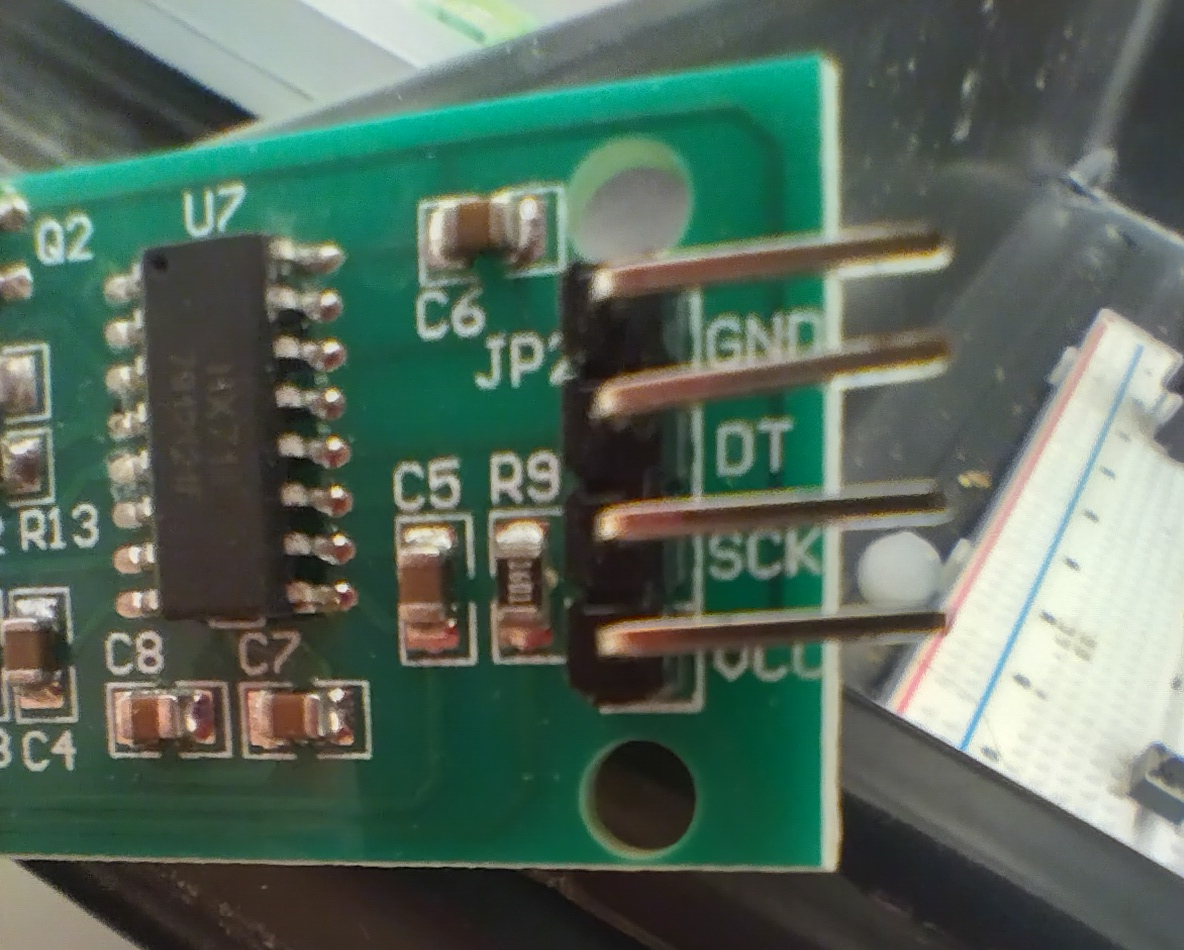
When your scale arrives, there are a couple of things to make sure to do before you begin connecting to the Arduino. First, make sure you screw the bases of the scale onto the load cell in a manner that does not cause the scale to sit unevenly on the surface. Secondly, on one side of the load cell, you will see an arrow, make sure you assemble the bases of the scale so that this arrow is pointing downward. Make sure not to overtighten or leave loose the screws that attach the bases to the load cell. If the load cell does not come with wires pre-attached to it, you will have to attach them to the surface of the load cell like so:



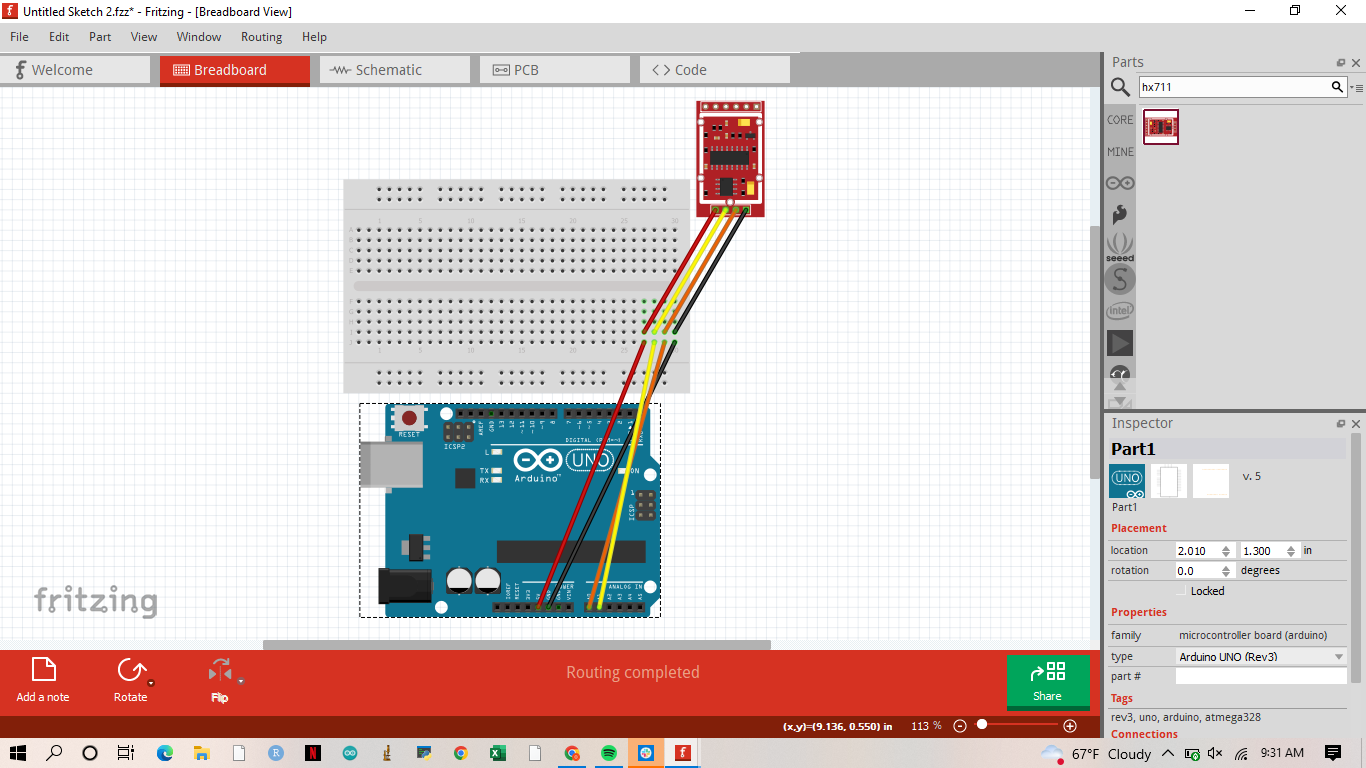
The other component of the scale is the HX711 module. This module converts the voltage that the load cell outputs and turns it into a numerical number. Do not be confused, this numerical number is not the final weight, the code explained in the *Writing the Code* section of the guide will explain how to manipulate this number into being the units that you desire. For now, let's continue with the connections.

The HX711module has four pins connected to it from the load cell itself. If you purchase a scale that is pre-connected you are all set for this step. If you are connecting it on your own, the connections should look like this (keeping the color wires the same as in the figure above, showing them connecting to the load cell):

Similarly, the HX711 module will have 4 output pins. These will be where the wires between the breadboard and the module are connected. There are 4 labels under each pin as follows from top to bottom: GND, DT, SCK, VCC (shown below). To connect these, female-to-male jumper wires are needed. VCC is the 5V connection and GND is the ground connection, when connecting on the breadboard try to keep these two wires red and black, respectively. DT and SCK are output pins that will be generating the sensor output values from the module to the Arduino, to be subsequently transformed into the weight units, and read. For now, connect these to the two empty spaces between the ground and 5V rows on the breadboard. 



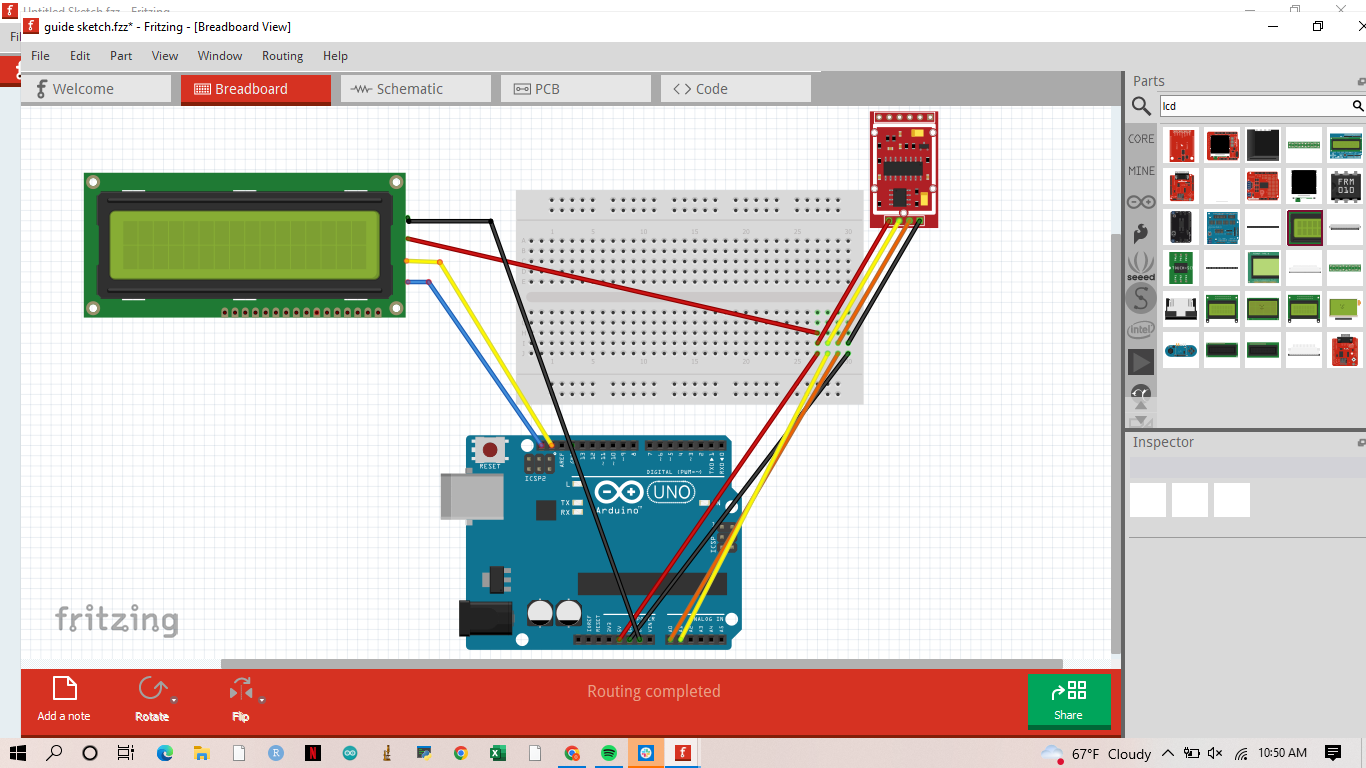
The two sensor outputs DT and SCK will be connected to the Arduino A0 and A1 pins, respectively. You could theoretically since nothing else will be connected to either A0 or A1, plug these wires directly into the Arduino. However, it will be better to keep most of the connections on the breadboard since it will give you flexibility in terms of placing the breadboard and Arduino into the holding compartment which will be covered at the end of the guide in the *3D Printing a Home.* At this point, your setup should look something like this:



## 

## Step 3: Connect I2C LCD Screen

The next accessory we will set up is the LCD screen. The LCD screen is a simple way to create a user interface and will help you design how you want to see the output. The LCD screen will also be useful as a troubleshooting tool later on in the *Testing and Troubleshooting* chapter. The LCD screen used in this guide is a I2C LCD screen. Many tutorials use a regular 16 by 2 LCD screen. The one used in this guide is actually a combination of an I2C module, the 16 by 2 LCD screen and a potentiometer. The largest benefit of using the I2C LCD screen over the regular is the simplification of the wiring that is required. For a regular LCD screen you would have to connect 12 wires to the breadboard and Arduino. For the I2C there are only 4 connections. The four output pins from top to bottom are GND, VCC, SDA, SCL. Similar to the HX711 module, the GND and VCC are the power pins and the other two are the sensors or the input pins that read what the Arduino tells the LCD screen to print. Each of these connections except the 5V (VCC) connection will be connected to the Arduino itself. This is because the Arduino actually has two ports named “SCL” and “SDA” which are made for this I2C LCD screen. At this point, you can connect the SCL pin to the SCL port and the SDA pin to the SDA port. You will also have to connect the GND pin to the second GND port on the Arduino board itself. Since the Arduino board has only one 5-volt port, the VCC pin will have to be connected to the breadboard. Connect the VCC wire to the row that connects with the wire connecting the breadboard to the Arduino 5V port. (Recall the basics of the breadboard connections). At this point, your setup should look something like this:



## 

## Step 4: Connect Pushbutton

The next addition to the system is going to be the pushbutton, which acts as the user input for telling the code whether to move forward in measuring or to stall. The pushbutton has two wires attached to it, a ground (black) wire and a 5V (red) wire. The 5V wire should be connected directly to the line of other 5V connections which are connected to the 5V port on the Arduino. The ground connection is what is going to be controlled by the board. You will need an extra (differently colored wire) and a 220-ohm resistor (comes in the Arduino build kit, and is bright blue). The setup should have the ground connection from the pushbutton followed by one end of the resistor followed by the wire that will lead to pin 7. On the other end of the resistor connect a wire to the 3rd GND port on the Arduino board. At the end of this set up your system should look like this:

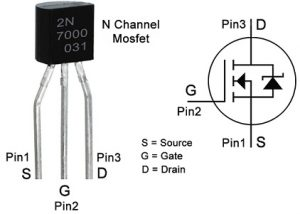
## 

## 

## Step 5: Connect Water Pump

The final addition to the setup is the water pump. The water pump is the most energetically demanding accessory to this system and one of the most important. The water pump, similar to the push button, has two wires colored black and red for ground and 5V, respectively. You will need to extend these wires, by soldering approximately 1 m / 3 ft long cables, to be able to reach the breadboard and give you the flexibility to move around. Please see the YouTube video on soldering the cables together [here](https://www.youtube.com/watch?v=uj_PbRBirkQ&ab_channel=SilverCymbal) to ensure a corrosion-resistant and waterproof connection.

There are a lot of electrical components involved in connecting the water pump. Firstly, you will need an N-channel MOSFET. The MOSFET has three connectors that should be connected as such: the source pin to GND, the gate pin to pin 9 of Arduino (data pin), and the drain pin to the pump. MOSFETS are directional, see the image below:



The next component is the diode. This is an essential component as it suppresses voltage to not damage the Arduino or any components of your system. Particularly, it helps with the voltage to travel in one direction instead of back-traveling. That is why it is occasionally called a back travel suppression diode. This is directional, the little white stripe on it should be going to the 5V connection. This part as well as the MOSFET will have to be replaced after long periods of time, especially long periods of time after intense usage.

Finally, you will be using a 220-ohm resistor between the gate pin and the pin leading to the Arduino board. To assemble the system, use the following diagram as a guideline (this is also the final diagram with all connections):

# 

# **Printing the Stand**

## Step 6: Printing the box

At this point you will have built something that looks like it's out of a sci-fi movie. The mess of wires and computers is not only a hazard for you but also can leave the computer and board open to debris, dust, and water to damage the system. It also leaves room to accidentally unplug connections during data collection. All these reasons are part of why a 3D printed box, customized to the needs of this experiment, was designed.

3D printing is a fantastic tool that offers you flexibility, reliability, and room for creativity. It is also a great way to learn some elementary Computer-Aided Design. This box was developed in TinkerCAD which is a very beginner-friendly and free CAD software. The tutorial and videos that are in the Resources tab are very useful and probably the first place to start if you have never done any CAD before.

You cannot print directly from TinkerCAD, but you can save the file as an .STL file to a program that will be able to format it to be printed.

The 3D printer that printed this stand was PRUSA i3 MK3S+, and the STL files were translated into a 3D printer file using a 3D printing software called Prusa Slicer which has all the tools you will need to prepare your design to be 3D printed.

This stand was printed with generic PLA at a 20% fill-in rate, and 0.2 mm QUALITY print settings.

Please NOTE all the components of the box were printed with the automatically generated support structures, and a brim, to make sure that the design is printed correctly, without the chance of sliding off the printer plate.

Make sure to clean off the support structures prior to assembling the box, using pliers, and be careful of the sharp edges.

**To access the design and to print one yourself, check out the GitHub page associated with this project >>** [**https://github.com/Leon-Yu0320/BTI-Plant-phenotyping**](https://github.com/Leon-Yu0320/BTI-Plant-phenotyping)

## 

## Step 7: Assembling the Box

Along with the 3D Printed stand there are some additional features that are included. You will need a few small zip ties to secure the plastic tubing to the spout support for the stand. Two small levels to insert into the corners of the stand to allow it to be adjusted to the surface you are working on. You will need four screw-in adjustable stand legs to go on each corner. The box doesn't need any extra materials except screws if you wish to screw down the Arduino to the box.

**Supplemental Video on Assembly can be found here:** <https://youtu.be/QlJUVdQT6VA>

# **Writing the Code**

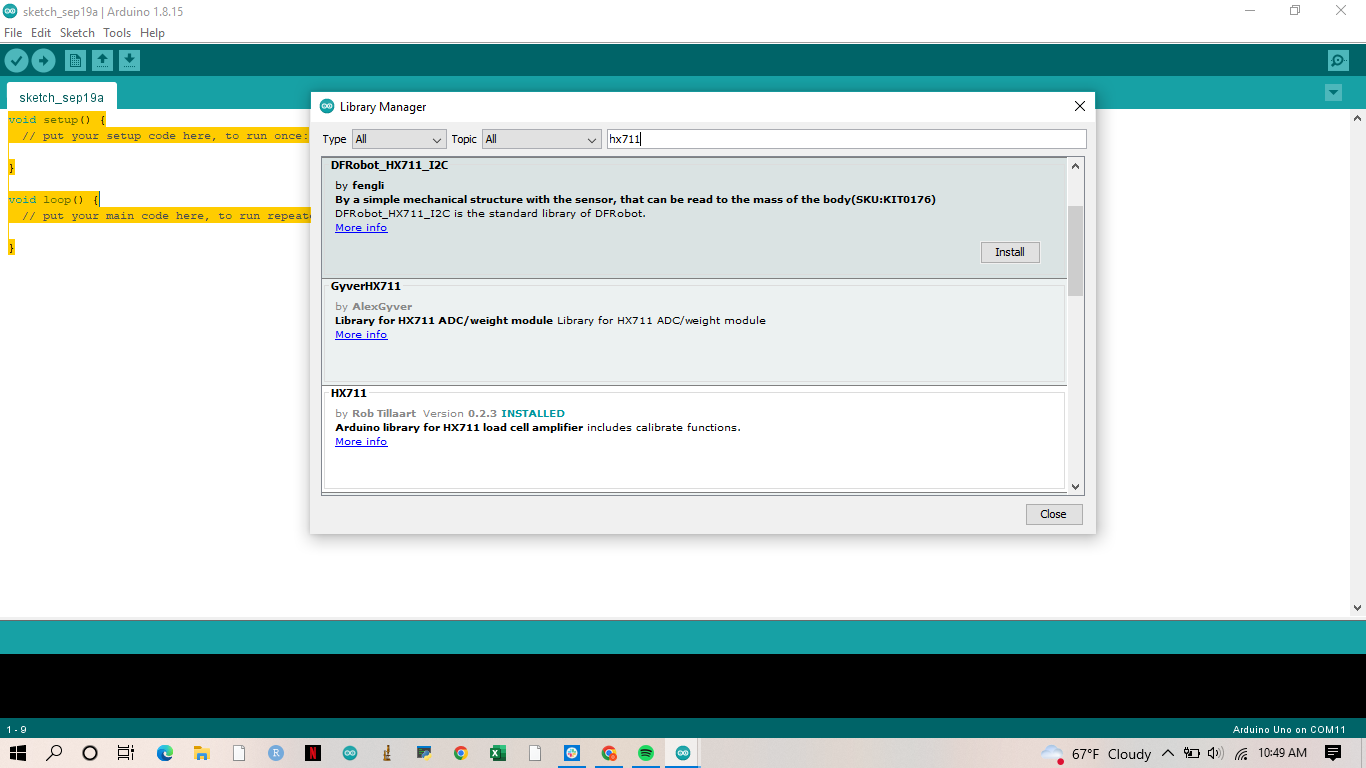
## Step 8: Download Arduino IDE

Arduino uses a robust coding language that is accessed by downloading the Arduino IDE software. You can do this by going to the Arduino site (arduino.cc) and selecting the ‘Software’ tab at the top of the page. Make sure you choose the correct option for your computer's operating system.

Arduino uses a unique type of coding language. It can most closely be compared to C++ however it is still very different. The coding language Arduino uses is robust and allows for a good introduction to coding logic for beginners. You dont have to know how to code beforehand! The more you use the Arduino IDE the more tools you will acquire to help you execute your code. This section of the guide will go over some of the most used tools needed for this project.

*Manage Libraries* **(Tools→ Manage Libraries)**

Similarly to other coding languages, Arduino operates by calling certain libraries. Libraries are collections of code language and documentation. There are some libraries that come preinstalled with every Arduino board however, for this project and others that have uncommon accessories or specialized functions you will need to download the libraries you need manually. This is one of the most vital tools in Arduino IDE. When you open the Manage Libraries tab a window will pop up that looks like this:



When you search up a library you need you will get a lot of results and many times there will be multiple options with the same name. For example, in the above example, the HX711 library was searched. It is hard to be able to tell which library is the correct one to use, especially since each library has slightly different functions and documentation. Each library aside from having a Title will also have who wrote the library below and a description of what the library contains or is used for. Once you have found the correct library click ‘Install” this will install the most recent version of said library. It is good practice to occasionally check your downloaded libraries to see if any updates have been posted. It is important to stay updated with the latest versions of Arduino IDE and all of your libraries to ensure the functionality of the device. Once installed the light blue stamp “INSTALLED” will display next to your library's name. Below is a list of the three libraries, only 2 of which need to be manually downloaded, that are used in the code.

| HX711 (by Rob Tillart) | Library with documentation for the scale |
| --- | --- |
| Wire (included in Arduino) | Library with documentation for serial communications |
| Hd44780 (by Bill Perry) | Library with documentation for I2C LCD display |

*Examples* **(File→Examples)**

The *Examples* option in the Arduino IDE is the beginner coders best friend. When you navigate to this tab you will see the library examples will be split up into four sections. ‘Built-In examples’, ‘Examples for any board’, ‘Examples for Arduino Uno’, and ‘Examples for custom libraries. When you click on any of the libraries it will give you various examples of the code being used. For the built-in libraries or Arduino-sponsored libraries, it will give you the proper commands to type and basic commands that will help you learn the library. But this tool is priceless when it comes to using it for custom/installed libraries. Since the installed libraries are written by people external to Arduino it is important to know how to call certain commands and how the library works. The examples for custom libraries will go over all of the functions contained in the library as well as an example or multiple examples of how to use the library for the most common use. For example, the HX711 library for the scale has an example of how to write code for measuring the weight of a certain object.

*Port* **(Tools→Port→ Serial Ports)**

The port is how the Arduino computer communicates with your computer and the IDE. It is how you send the code to the Arduino and how sensor data is recorded on your computer. The port selections can be found under the Tools tab. Most often, when your Arduino is connected one of the ports will read ‘Arduino Uno’ but if this is not the case a simple way to figure out what port you are connected to is to take note of the names and numbers of the ports while you are connected and then disconnect to see which port changed. Keep in mind you will be connected to different ports depending on which USB port you are using on your computer. This is also the first place to check when your Arduino has an error uploading the code or an error reading the code.

*Board* **(Tools→ Board:)**

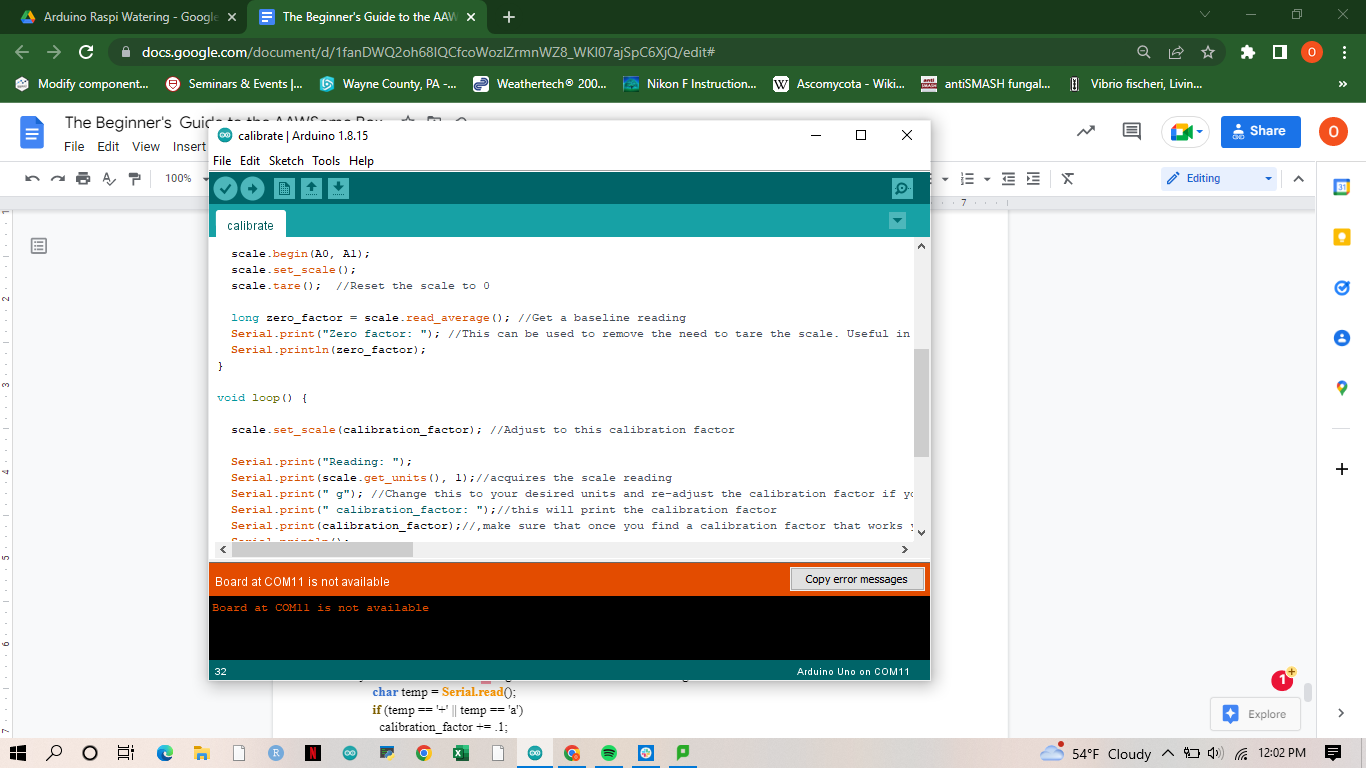
While most of the time the Arduino IDE will sense what boar you are using it is always good to check to make sure the version of your Arduino matches what the IDE has stated. Note: this guide is formatted to work with Arduino UNO and functionality and connections might alter if you are using a different version of the board.

*Copy as..* **(Edit→ Copy as HTML/ Copy for Forum)**

The copy functions are most useful for sharing your code if you need to troubleshoot or just to share your work with others. The Arduino user forum is a great place to start when troubleshooting your board. There is a very specific way that your code must appear on the forum for others to be able to assist you in troubleshooting. This is when you will want to use the ‘Copy for Forum’ option. You can also copy the code as an HTML to make it more transferable to .docx files or other forms of documentation.

## Step 9: Calibration Code

The calibration code should be made in a different sketch than the watering and weighing code. At the beginning of each experiment, you should upload and run the calibration code and calibrate your scale. After you have received a precise reading, you should go to your watering weighing sketch and re-upload that. Remember: The Arduino will only remember and run the code that was last uploaded. To have another sketch run, you must re-upload it manually. The ‘Serial Monitor’ button (red arrow) is where you will give the input of (+/-) to change the calibration factor and to read the weight that the scale gives you. Make sure you are connected to the correct port!



#**include** "HX711.h" //This library can be obtained here http://librarymanager/All#Avia\_HX711

#**define** LOADCELL\_DOUT\_PIN A0//Define the pins that the scale is connected to

#**define** LOADCELL\_SCK\_PIN A1

**HX711** scale;//initialize the library

**float** calibration\_factor = ### ;//play around with this number until you get a value that is in the ball park of your actual weight

//The initial numbers that you will see on your scale will most likely be either way too small or way too large unless they are about right in which case skip this step

**void setup**() {

**Serial.begin**(9600);//begin serial communication

**Serial.println**("HX711 calibration sketch");

**Serial.println**("Remove all weight from scale");//remove all weight from scale

**Serial.println**("After readings begin, place known weight on scale");//place your known weight on the scale, if it is an arbitrarily high or low value adjust the calibration factor significantly because this code will only make minor changes to it. If you are measuring a 100 gram weight and the scale is reading 70000 I recommend going in either direction by 5.

**Serial.println**("Press + or a to increase calibration factor");//open the serial monitor and type in + or -

**Serial.println**("Press - or z to decrease calibration factor");

scale.**begin**(A0, A1);

scale.**set\_scale**();

scale.**tare**(); //Reset the scale to 0

**long** zero\_factor = scale.**read\_average**(); //Get a baseline reading

**Serial.print**("Zero factor: "); //This can be used to remove the need to tare the scale. Useful in permanent scale projects.

**Serial.println**(zero\_factor);

}

**void loop**() {

scale.**set\_scale**(calibration\_factor); //Adjust to this calibration factor

**Serial.print**("Reading: ");

**Serial.print**(scale.get\_units(), 1); //acquires the scale reading

**Serial.print**(" g"); //Change this to your desired units and re-adjust the calibration factor if you like

**Serial.print**(" calibration\_factor: "); //this will print the calibration factor

**Serial.print**(calibration\_factor); //,make sure that once you find a calibration factor that works you note it down for that load cell since it will be used in the set up of the code. Make sure to recalibrate your load cell once in a while if you notice off values

**Serial.println**();

**delay**(2000);

**if** (**Serial.available**())//This part of the code reads the serial communication between your laptop and the load cell.

{ //This next section connects the user input into the serial monitor with an action. This command only changes the calibration factor by .1 so use this to fine-tune your readings once you get them in the correct range. There will be a general variation in the readings that the scale gives you so after you alter the calibration factor wait for about 5-7 readings to gauge how much variance you have. Generally speaking, you want no more than 1-2 grams variation from the true weight.

**char** temp = **Serial.read**();

**if** (temp == '+' || temp == 'a')

calibration\_factor += .1;

**else if** (temp == '-' || temp == 'z')

calibration\_factor -= .1;

}

}

## 

## Step 10: Write the Code

Please copy and paste the code below into a new Arduino sketch. Make sure you have conducted the calibration (**Step 9)** and obtained the calibration factor before you begin this step. This is the skeleton of the code, there are some values that you will need to input. You will need to edit the values for **calibration\_factor (Step 9),i (Step 14),myRefs[](Step 14),and a,b,c (Step 13).**(If you are new to this setup and are unsure of how to change these values, directions for each part of the code can be found in the next section)**.**Click the ‘Verify’ button on the sketch to make sure the code was copied and edited correctly. Once the code was successfully verified, you can upload the code to the box by clicking the ‘Upload’ button which is located next to the ‘Verify’ button.

#**include** <**Wire**.h>

#**include** <**hd44780**.h>

#**include** <hd44780ioClass/**hd44780\_I2Cexp**.h>

**hd44780\_I2Cexp** lcd;

#**include** "HX711.h"

**float** calibration\_factor = ####;

**const int** motorPin = 9;

HX711 scale;

**const int** BUTTON = 7;

**const int** data1 = A0;

**const int** data2 = A1;

**const int** LOADCELL\_DOUT\_PIN = A0;

**const int** LOADCELL\_SCK\_PIN = A1;

**float** weight\_now = 0.0 ;

**float** weight\_prev = 0.0 ;

**float** weight\_before=0.0;

**int** val = 0;

**int** val\_new = 0;

**void setup**() {

**pinMode**(motorPin, OUTPUT);

**Serial**.**begin**(9600);

scale.**begin**(A0, A1);

scale.**set\_scale**();

scale.**tare**();

lcd.**begin**(16, 2);

lcd.**clear**();

**long** zero\_factor = scale.read\_average();

lcd.**print**("Zero factor: ");

lcd.**print**(zero\_factor);

**delay**(1000);

}

**void loop**()

{

**for** (**int** i = 0; i < #; i++)

**while** (i < #) {

**while** (val < 1) {

lcd.**print**("Press button");

val\_new = **digitalRead**(BUTTON);

val = val\_new;

**delay**(750);

lcd.**clear**();

}

lcd.**print**("pot#...")

**Serial.print**(i + 1);

**Serial.print**(",");

lcd.**print**(i + 1);

**delay**(2000);

lcd.**clear**();

lcd.**print**("ref weight");

**int** myRefs[#] = {#, #, #, #, #, #, #, #, etc.};

**int** REF = myRefs[i];

lcd.**print**(myRefs[i]);

**delay**(2000);

lcd.**clear**();

i++;

**delay**(1000);

lcd.**setCursor**(0, 0);

scale.**set\_scale**(calibration\_factor);

lcd.**print**("Weight(g) ");

weight\_now = scale.**get\_units**();

weight\_before=scale.get\_units();

**Serial.print**(weight\_before);

**Serial.print**(",");

lcd.**print**(weight\_now);

**delay**(2000);

lcd.**clear**();

**while** (weight\_now < REF) {

**float** a= REF-weight\_now;

**float** b= a/8;

**float** c=750\*b;

**if** (a<10){

lcd.**print**("Watering...");

**digitalWrite**(motorPin, HIGH;

**delay**(1000);

**digitalWrite**(motorPin, LOW);

**delay**(3000);

lcd.**clear**();

}

**if** (a>10){

lcd.**print**("Watering...");

**digitalWrite**(motorPin, HIGH);

**delay**(c);

**digitalWrite**(motorPin, LOW);

**delay**(3000);

lcd.**clear**();

}

lcd.**print**("Wt ");

weight\_prev = scale.**get\_units**();

lcd.**print**(weight\_prev);

**delay**(2000);

lcd.**clear**();

weight\_now = weight\_prev;

}

**Serial.println**(weight\_now);

lcd.**print**("Next pot...");

**delay**(4000);

lcd.**clear**();

val = 0;

}

}

## Step 11: Annotate the Code

The code consists of a few generalized parts: calling of libraries, definition of variables, assigning ports, setup and loop. In this section, the written code will be broken down and annotated. To annotate code in Arduino simply insert ‘//’ after the line of code and write the comment. This is how the annotations will be written on the code below.

### Calling of Libraries

#**include** <**Wire**.h> //This is the library that facilitates serial connection and communication between the computer and Arduino. Built into the Arduino Uno package and doesn’t need to be downloaded

#**include** <**hd44780**.h>//Downloaded library by Bill Perry and needed to access key functions for the LCD screen

#**include** <hd44780ioClass/**hd44780\_I2Cexp**.h>//include i/o (input/output) class header

**hd44780\_I2Cexp** lcd;// declare lcd object: auto locate & config display for hd44780 chip

#**include** "HX711.h"//Downloaded library by Rob Tillart and is used to access key functions for the scale

**float** calibration\_factor = ####;//The scale is not made to interpret grams or pounds, it simply give a number associated with the weight it reads. **This number is provided by the calibration code in the section titled** ‘Calibrating the Scale’. This factor is what the Arduino uses to convert the output into an interpretable unit.

**const int** motorPin = 9; //Initializes the pin that will be regulating the water pump, the water pump doesn't have any special functions other than turning it on and off which can be done without a library

HX711 scale;//Initialize the scale

### Definition of Variables

There are *four* types of variables that are used in this Arduino code: **int, const int, float, long.** Each of them represents a different type of number and allows for you to do different things with that variable. The [**int**] variable is any round integer and can be positive or negative. It's important to note that the INT variable has bounds of [-32768 to +32768]. If you need to store numbers larger than this, you will need to store them as [**long**] variables. **long** is exactly the same as **int** except a ‘long’ variable can store 4 bytes of info rather than 2 bytes which is what the ‘int’ stores. The limits of a ‘long’ variable are [-2,147,483,647 to +2,147,483,647]. Both of these are for rounded numbers, meaning you cannot store decimals in these variables. To store decimals, you will use **float** variables. ‘Float’ variables offer higher resolution than ‘int’ or ‘long’ however they have slightly less precision than the rounded number options. ‘Float’ variables are essential when working with the scale, since the output will be read as a decimal. If you try to assign the output from the scale to be stored as a rounded number variable such as ‘int’ or ‘ long’ it will throw you an error. The last type of variable is a **const int** type variable. This is the same as the ‘int’ variable except it has the qualifier ‘const’ before it which stands for ‘constant’. This qualifier modifies the behavior of the variable and changes it to *Read Only*. While variables stored with ‘const int’ can generally be stored in ‘int’ it is good housekeeping practice to assign variables that you will be reassigning and changing values for throughout the code to one type of variable and those whose values must stay the same to a different type of variable.

**const int** BUTTON = 7;//This sets the variable named ‘BUTTON’ to 7 for the remainder of the code. This is because the wire connecting the pushbutton to the board is connected to pin 7. Later on in the code a function will call the variable ‘BUTTON’ instead of manually typing in ‘7’ for the pin input

**const int** data1 = A0;//This sets one of the data pins that the scale uses to send information back to the Arduino

**const int** data2 = A1;//Similarly to A0, it is needed to send information back to the Arduino

**const int** LOADCELL\_DOUT\_PIN = A0;

**const int** LOADCELL\_SCK\_PIN = A1;

**float** weight\_now = 0.0 ;//This along with the next two variables are made to store the weight reading from the scale. They are float variables because the readings come as decimal places. We need multiple variables to store the values because they are used in determining the thresholds and allow the code to run smoothly when we get to the loop part.

**float** weight\_prev = 0.0 ;

**float** weight\_before=0.0;

**int** val = 0;//This is is a value set to change when the pushbutton is pressed. When the push button has been pressed the value of this variable will increase to 1.

**int** val\_new = 0;//The value is reset to 0 at the end of the code so that when the loop restarts it is as if the button was not pressed.

### void setup ()

The setup is where you begin to initialize your variables and start to call important functions from your selected libraries. Unlike void loop, which is discussed in the next section, void setup runs only once at each startup of the Arduino.

**void setup**() {//Whenever you start writing a new sketch the blank document will always have the void setup and void loop in place.

**pinMode**(motorPin, OUTPUT);//This sets the pin number that we previously assigned to the variable motonPin as an OUTPUT pin. Telling the Arduino that information will be sent OUT of this pin to the motor.

**Serial**.**begin**(9600);//This starts the serial communication between your computer and the Arduino. This is also where you set the *baud* rate which is the bits per second. It is important especially when you are connecting to a RasPi or other computer that both are on the same baud rate. You can manually set the baud rate to your needs but for this experiment 9600 works well.

scale.**begin**(A0, A1);//This initializes the scale and tells the computer that the A0 and A1 pins are where the information for the scale will be coming in.

scale.**set\_scale**();// sets the scale by beginning communication

scale.**tare**();// sets the scale to 0

lcd.**begin**(16, 2);//Initializes the LCD screen but inputting the size of the LCD screen, in this instance the lcd screen is a 16-character by 2-character screen.

lcd.**clear**();//clears the LCD screen of any information it might have previously stored in its memory or system

**long** zero\_factor = scale.read\_average(); // Gets a baseline reading of the scale. Remember that the scale is not a perfect accessory and varies a bit even when there's no weight on it. Calculating the average can help mitigate the eros. This can be used to remove the need to tare the scale at the beginning of every loop. Useful in permanent scale projects. **It is crucial that you do not have anything on the scale during this step**

lcd.**print**("Zero factor: ");//This tells you what value the scale got as an average and what it set the tare to. This is purely for your information

lcd.**print**(zero\_factor);

**delay**(1000);//It is useful to include some delays in your code as give the Arduino some time to catch up or compute before moving on to the next step. This will become more important in the loop part of the code since Arduino is capable of moving very quickly and it can cause the accessories to struggle in keeping up.

}

### 

### void loop ()

This is the packed portion of the code and the part where the action actually happens. This part of the code will run continuously in the code until the Arduino is disconnected or there is a terminating statement. This code’s void loop () section can be broken down into sequential steps that generalize the overall flow of the experiment.  **Pushbutton Start**, **Pot Number and Reference Weight Information**, **Weighing and Calculating of Difference**, **Operating the Water Pump**, **Rewighing and Recalculating**, **Ending the Cycle**.

**void loop**()

{

**for** (**int** i = 0; i < #; i++)//for this code we are using loops and nested loops to achieve our objective. This is the beginning of the for loop and defines the parameters of the loop which in this case are where i starts at 1 and increases by 1 at the end of every loop until i reaches the value # (which you will input)

**while** (i < #) {//This second ‘nested’ while loop begins the next section of the code, using the for loop is important because it helps set parameters for the variable i to change since we will be using i as our pot number index. The while loop will run the loop *while* the variable is below the threshold

**while** (val < 1) {//This while loop is used for the push button loop within the main code.

lcd.**print**("Press button");//Code with lcd.print functions are useful to create a clearer interface between the user and the computer.

val\_new = **digitalRead**(BUTTON);//the digitalRead function is used to read the data specifically coming out of a certain pin, in this case, the pin we assigned the variable BUTTON to.

val = val\_new;//sets the value of val to the value just read from the pin, if button not pressed the value will be zero if pressed it will be a nonzero number

**delay**(750);

lcd.**clear**();

}

lcd.**print**("pot#...")

**Serial.print**(i + 1);//Arduino is 0 indexed so the first pot is i+1 which is 1

**Serial.print**(",");//These commas are important specifically to have listed in the Serial.print function because they will be how the computer splits the data when you are storing it either on Excel or a RasPi

lcd.**print**(i + 1);

**delay**(2000);

lcd.**clear**();

lcd.**print**("ref weight");

**int** myRefs[#] = {#, #, #, #, #, #, #, #, etc.};//This is where the reference weights of what the plants should be watered to are entered with commas in between each number

**int** REF = myRefs[i];//since when the reference weights are inputted it makes an array you can call certain values from an array by inputting an integer into hard bracelets following the dataframe name.

lcd.**print**(myRefs[i]);

**delay**(2000);

lcd.**clear**();

i++;//increases the pot number by 1 for the next time the loop runs.

**delay**(1000);

lcd.**setCursor**(0, 0); //resets the lcd screen to the 0,0 cursor position

scale.**set\_scale**(calibration\_factor); //set the scale to the calibration factor that will be explained in a following chapter\*this is an important step

lcd.**print**("Weight(g) ");

weight\_now = scale.**get\_units**();//gives a reading from the scale and sets the value weight now to that number

weight\_before=scale.get\_units();//gives a reading from the scale and sets the value weight before to that number

**Serial.print**(weight\_before);

**Serial.print**(",");

lcd.**print**(weight\_now);

**delay**(2000);

lcd.**clear**();

**while** (weight\_now < REF) {//This while loop is the watering pump loop, this loop is saying that it will run as long as the weight now variable is below the reference weight

**float** a= REF-weight\_now;//these calculations are to make the watering “smart” that is to say so the pump doesn't just water a set amount in intervals, these calculations provide a level of precision, these numbers will be different depending on the pump you use so do some troubleshooting just to figure these out.

**float** b= a/8;//divide the grams of difference by an interval to then multiply by a time. This will take some trial and error to figure out what combination of values will not overwater your plant

**float** c=750\*b;//multiply by the number of milliseconds per interval, this number will be different for every waterpump and setup so try out your pump on this code before actually beginning experiments.

**if** (a<10){//This code helps if the difference is smaller than 10. Small differences will not be suitable for the calculated watering (like the code below this one) for these it is best to set the pump to work for a short time and water in intervals

lcd.**print**("Watering...");

**digitalWrite**(motorPin, HIGH;//turns on the water pump

**delay**(1000);//this is the number you edit to control how long the pump stays on for

**digitalWrite**(motorPin, LOW);//turns off the water pump

**delay**(3000);//this allows for the water to seep into the soil and for the scale to accurately reflect the new weight of the plant

lcd.**clear**();

}

**if** (a>10){//the code for the difference being greater than 10 grams is exactly the same as above except the time the pump is on is not a defined interval but the calculated variable ‘c’

lcd.**print**("Watering...");

**digitalWrite**(motorPin, HIGH);

**delay**(c);

**digitalWrite**(motorPin, LOW);

**delay**(3000);

lcd.**clear**();

}

lcd.**print**("Wt ");

weight\_prev = scale.**get\_units**();

lcd.**print**(weight\_prev);

**delay**(2000);

lcd.**clear**();

weight\_now = weight\_prev;//sets the value of the current weight to weight\_now so the loop can re-run and water additionally if needed

}

**Serial.println**(weight\_now);//once the weight matches the conditions of the while loop the computer will exit the loop and move on. The ‘ln’ at the end of the statement means ‘newline’ so in the recording of the code the Arduino will begin a new line of recorded values after this one.

lcd.**print**("Next pot...");

**delay**(4000);//gives the researcher time to switch out samples

lcd.**clear**();

val = 0;//sets value to mean the pushbutton is unpressed

}

}

# 

# **Testing**

After the box is assembled, and you have written the code, it is important to test it to check for any problems. While this guide cannot cover everything that could possibly turn up or go wrong on your Arduino it might be able to help identify some common issues.

## Step 12: Scale Testing

The first test that will be done on the Arduino is figuring out the calibration factor. It is important to note with the calibration factor that you will not get an exact answer for every weight you put on there. The error will increase the larger the difference from your calibrated weight. When calibrating the scale, it is best to have two known weights representative of the upper limit of the weight you will be measuring, as well as the lower limit. This will help prime the scale for the experiment that you will be running. It is crucial to calibrate the scale before starting any experiment and keep a log of when you calibrated it as well as what variance you were able to achieve (a good calibration will have no more than a gram or two of variance in the positive or negative direction of the weight). It is good practice to keep a log of when the scales were calibrated last and to what level of variance they achieve. If you have an experiment that is running for months on end, it might be a good idea to recalibrate to the same weights you calibrated to in the beginning of the experiment to make sure the values aren't being affected artificially.

## 

## Step 13: Pump Testing

Each pump is different and has its own flow rate. The only thing we can control is how long the pump is on and correlate that with the grams of water used to water the plant. Before you begin the experiment, you can upload the watering and weighing code and run it artificially by having a beaker instead of a pot and observe the change in weight as it is being watered. The first value you will need to determine is approximately how many grams of water does the pump disperse during a given amount of milliseconds. The value you are changing to test this is the delay( ) between the motorPin, HIGH command, and the motorPin, LOW command.

Once you have a general value for the grams of water for the selected time, the next step is to calculate the watering time based on the difference between the reference weight and the weight of the pot. Below is a calculation optimized for the specific water pump used in that machine. Firstly, set ‘a’ as the difference between the reference weights and the weight being read by the scale. This will give you how many grams of water the pump needs to water the plant. Secondly, you will need to divide this value by the grams you correlated to the milliseconds in the first paragraph. For example, in this case for this pump, it was calculated that it watered 8 grams every 750 milliseconds. So the difference (a) was divided by 8 to get ‘b’ and then 750 milliseconds was multiplied to get the total time that the pump should water the plants. This calculation is placed within the watering loop so it will be calculated for each pot separately every time the loop starts.

**float** a= REF-weight\_now;

**float** b= a/8;

**float** c=750\*b;

However, this calculation is good for when there is a large difference in the weights, when the differences become small (i.e. 10 grams-20 grams of difference) the watering will be more precise if the pump works not by a calculated value but by a set, small interval which will water the pot a little bit at a time until it reaches the target weight.

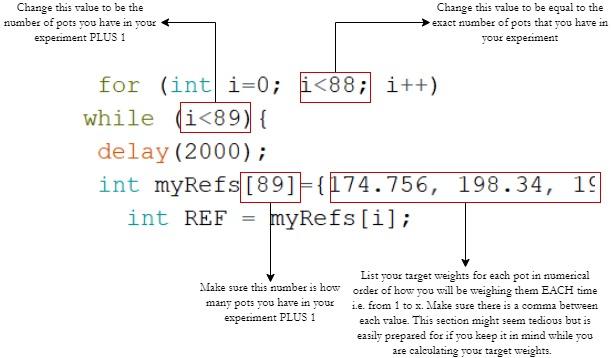
# **Starting an Experiment**

## Step 14: Setting the Scene

***For guidelines on calculating the reference weight per pot reference the protocols.io for this setup. You will find detailed directions under* ‘Loading and Executing Experiment’**

Once you have built the machine and written the code, it is finally time to set up the scene of the experiment. Make sure you set the machine up on an elevated surface where it will be easy for you to see the LCD screen and manage all the components of the process. Leave lots of room around the setup for your laptop and working space for movement. Also, make sure that you leave ample space between the jar of water and the machine. You will most likely be refilling the jar with water during your experiment (Never during a loop since adding water while the machine is watering a pot will change the pressure in the pump and alter the watering rate) and leaving some space will minimize the risk of getting water into the machine.

Aside from the physical setup of the watering weighing system, there is one last thing you need to revise in the code before the final upload to the Arduino. That is to input the reference weights. Refer to the following guidelines:

. 

After you have inputted the reference weights into the code, connect the Arduino to your computer and upload the final sketch onto the Arduino. This will be the sketch that the Arduino uses for the entirety of the experiment. Note: when you start a new experiment, you will need to revise these reference weights.

Occasionally, when watering samples under drought conditions, the water will begin to seep through the drainage holes of the pot. It is important to get rid of this water as any small amount of it will impact the readings of the scale. For this, it is recommended to add an extra, black saucer on top of the saucer that is taped down to the scale. This will allow you to pour the excess water back into the pot after the watering is done to ensure that the pot has gotten all the water that it requires for the experimental condition it is under.

On your stand, there should be two levels that will help you adjust the stand based on the surface you are working on. It is important to check that the scale and stand is level with the surface before starting the experiment, since it will impact the reading of the weights. To adjust the stand, use the four twisting metal stands on the corners of the stand. Make sure the scale is level and firmly supported on the surface on all four corners.

## Sources of Error

Other than human error, there are a few sources of error that have been mentioned sparsely throughout the guide, but are worth mentioning again. Firstly, the scale can only be calibrated down to a few grams of variance from the true value. Since the measurement is being taken at a single time point, the value might be off by anywhere from .1 grams to 2 grams from the true weight of the pot. This error changes from pot to pot unless you have failed to recalibrate your scale, in which case all your values might be significantly higher or lower than your true value. In most cases, depending on the weight of your samples, this variance won’t be significant enough to worry about, but it is always important to mention even the smallest sources of error.

Another error that is not human-caused that has been mentioned sparsely in sections of the guide is the variation in the flow rate of water coming from the pump. The pump was chosen for this project to be suited for the accuracy that the experiment needed. The pump does not have a set flow rate that it maintains while watering or from measurement to measurement, which is why the watering loop has two levels of thresholds to aid in mitigating this error. But it is important to note that occasionally the pump may have a different flow rate and cause the loop to go through its cycle more times than normal or fewer times than normal. The loop is formatted to prevent the water pump from overwatering but on occasion, there have been cases where on the first run of the loop the pump overwaters the plant. However, there have not been cases, other than those mentioned in **Troubleshooting,** where the pump overwaters more than 5 grams.

# **Daily Life with the Box**

## Step 15: Logging Measurements

### Installing Data Streamer

**Note:** You might need to update your version of Microsoft Office, this project works with Microsoft 365.

Once the experiment is set up and the final code is uploaded to the Arduino it's time for collecting measurements. Luckily, Arduino projects are popular enough that Microsoft Excel has a function for logging data from them. To download this function, open the Microsoft Excel app on your computer that you will be connecting to the box for measurements. On the bottom left of the introduction screen, click **Options**. This will take up to a pop-up window with the Excel Options. In the left selection bar at the bottom, select **Add-Ins**. On that screen, there will be an option at the bottom **Manage:** which will lead to a drop-down list of Add-Ins that you can choose from. From the drop-down list, choose **COM Add-Ins** and click **Go>>.** This will lead you to a window with check-boxed options, you will need to select the checkbox listing **Microsoft Data Streamer For Excel** and click **Add**. After this, on your Excel sheets, you should have an option in the tabs titled **Data Streamer,** which you will use to connect to the Arduino and record measurements.

### Connecting to Data Streamer

A data streamer is a useful tool for any type of data collection with sensors. It has a lot of functions that you could use, but for the sake of this experiment, we will only be using a handful. There are shortcomings with using Microsoft Excel to record your data and at the end of the guide, there will be a section on potential alternatives to these inconveniences. The biggest roadblock is that the data streamer can only record data from each day in one sheet. There is no way to compile data from multiple days into one spreadsheet for each day you must start a new spreadsheet and save that spreadsheet (The alternative data recording method mentioned at the end of the guide provides a way to compile data into one spreadsheet). However, the largest advantage of using Microsoft Excel is that it is relatively user-friendly, has lots of troubleshooting forums on the internet, and is accessible to most of the community!

Once you have installed the Data Streamer Add-In and are ready to take your first set of measurements, open a new Excel Spreadsheet and title it with a unique title that you will be able to distinguish for other days that you are recording data. Once Excel opens a new spreadsheet, go to the **Data Streamer** tab and click, on the upper left-hand corner of the drop-down menu, **Connect a Device**. Most often one of the COM ports will be labeled with “ Arduino Uno” but if not you can test which COM port you are connected to by disconnecting and reconnecting the Arduino and seeing which COM port changes. Once you are connected to the Arduino a data logging table should pop up on your screen. Connecting a device will make several new sheets at the bottom of the Excel sheet. The one that you will be focusing on is the Data In sheets. This table is organized to show the most recent data communication at the top and then organize the data from the oldest data communication at the top to the newest at the bottom. **Important** The default on the sheet is 15-20 rows. **Before** you begin collecting data, go to the **Settings** tab at the bottom of the spreadsheet and type in the number of data points you wish to collect, plus a good buffer of rows on top of that. **You must do this before you start recording data. If you noticed, you have forgotten this step, DO NOT ADJUST THE ROWS DURING DATA RECORDING. This will cause data to be lost, since it will regenerate a new table. Save the data you have already collected and open a new spreadsheet and reconnect to the Arduino again.** Once you have set the data rows, you can click **Start Data**. The data streamer will then record the data throughout your experiment. **NOTE: Change your computer sleep settings to *Always On* if the computer is not ON it will alter the power being sent to the pump and the Arduino as well as halt data collection**.

When you are finished logging the data, click the option **Stop Data** on the Excel sheet and disconnect the device and **make sure to save your spreadsheet.**

## During the Experiment

There are a few considerations to keep in mind while the experiment is running.

Immediately when you connect the Arduino to the computer, the Arduino will power up and begin to run the code. This does not impact how and when you connect the device to the data streamer, but the taring and the setting of the zero factor in the code will run automatically. Therefore, **before** you plug in the Arduino to the computer, make sure that the stand is level and the only things on the scale are things that will be there throughout the whole experiment and with every pot. This should be just the two saucers taped down to the scale.

When you first fill up the container with water, make sure the container is relatively clean and free of large particles of soil, which can clog up the pump and cause the flow rate to be altered and eventually cause the pump to malfunction. Do not let the water level drop lower than 25% of the container, and do not pour in more water while the pump is actively watering a sample.

Avoid any of the wire connections touching the water. A good way to manage this is by clipping the wire to the brim of the container with a binder clip at the first connection, or generally giving the wire just enough slack to keep the water pump in the lower fourth of the container.

If water leaks through the pot into the secondary basin, make sure to add that water back into the soil, so the sample has all the water that is allotted to it based on the treatment.

The push button is not made to be like a switch. When you press the button to go onto the next pot at the beginning of the loop, give the code at least 3 seconds to respond to the button being pushed and make sure to push the button for at least 1.5 seconds.

The next page is a Printable Checklist that you can print out and have available to check off at the beginning of every experiment!

# Printable Checklist for Logging Measurements

***Use this checklist at the beginning of every data-collecting session. Adhering to this checklist will help prevent data loss and system malfunctioning due to lack of coffee and sleep!***

* **Level the scale**, make sure it isn't wobbling
* **Clear the scale** of any unnecessary items, and make sure nothing is touching the scale or unexpectedly obstructing the scale
* Fill the container with clean water and observe for any soil particles or debris
* **Is your computer charged?** If not, plug in the computer during the duration of the experiment
* **Change sleep settings on the computer to ‘Always On’**
* Is the area around the Arduino and your computer dry?
* **Add the extra rows in the spreadsheet settings!**
* Name your spreadsheet a unique name to associate with this data collection day

Things that may be useful to have nearby

* **Scale** (for double-checking samples that might have been over or under-watered indicating a problem needed to be fixed)
* **Quick absorbing paper towels** (for quick clean up of any water that gets onto or near electrical components)
* **Extra, small container** (for refilling water from the sink/hose into your large container)
* **Charging cord** (if you are using a laptop, in case the data collection takes longer than planned)
* **Notebook** (for recording observations on the functioning of the machine, this will help you troubleshoot any problems if they arise)

# Maintenance

The system has a lot of working parts and wires. It is not a perfect system and will need short-term and long-term maintenance. In this section, short-term maintenance will detail actions you should take on a daily to weekly timescale and long-term maintenance will detail actions you should take on a monthly to yearly timescale.

## Short-Term Maintenance

One of the most important maintenance issues that have been observed with the system is failing to recalibrate often enough to maintain precise and accurate measurements. You should keep a recalibration log with the date you last recalibrated the system as well as what the final calibration factor was and the variance from the true value that you were able to achieve. You should recalibrate at a minimum before starting each new experiment. If you are doing a long-term experiment and using the box for months on end, recalibrating once for every 1000-2000 measurements is suggested.

This system is made to handle water when it is contained and in use. Very importantly, at the end of each watering session, make sure to empty any lingering water in the tube from the tube. Do not leave the pump in standing water (empty the water container at the end of every session). It is important to keep a clean and dry surface since there are so many electronics and wires involved. Assess for any spilled water droplets and clean them with a paper towel as soon as possible to prevent them from leaking into any components.

At the end of every experiment, make sure to examine for debris that might be stuck under the scale or the stand that might be throwing off your measurements. There is a lot of soil being worked with, and it is essential to clean any debris with a slightly damp towel to ensure that the scale is unaltered.

Avoid leaving the pump to dangle on the wire over the edge of the container. Constant pressure on the connections might cause a malfunction in the functioning of the watering system.

## Long-Term Maintenance

Once you build this setup, you will be using it for a while! There will be some maintenance that you will have to do after several months to years of using the machine. These changes don't have a timeline, rather, you will begin to notice the machine malfunctioning or not working at all and the solution might be to perform these changes.

The box is made to be opened for easy access to the components of the breadboard and electrical system that you need. The first change that you might need to make is replacing the pump. The pumps used for this system are meant to put cost, efficiency, and accessibility first. The pumps are not industrial-strength pumps and will need to be replaced after some use. Luckily, they are delivered in packs of 3. Symptoms that you will need to get your pump changed include: pump stops pumping water, even after attempting all fixes in **Troubleshooting** and consulting with the Arduino Forum.

Similarly, if the scale begins to be impossible to calibrate or shows numbers that make no sense given the calibration factors you have previously calibrated it to, you might want to consider swapping out for a new scale. Before doing so, check if the screws on the scale have been tightened (but not over-tightened). Check all the connections and the directionality of the scale. Before switching, try to flip the scale and see if the values make more sense. If not, replace the scale and recalibrate it to find the new calibration factor.

Additionally, in the wiring, there are elements that are there to aid in the power flow from the Arduino to the pump. These elements are the MOSFET and the diode as described in the **Water Pump**  section of **Building the Box.**  If the pump shuts off mid-experiment, or uncontrollably waters and the solutions in **Troubleshooting** don't work, you might need to replace these elements. Note: if you have these problems and your machine is relatively recently built then it is unlikely that these parts need to be replaced, and the problem is being caused by something else.

# Improvements and Future Work

## Excel vs Raspberry Pi data logging

Recording all measurements in Excel has its benefits and its drawbacks. Firstly, it operates on the assumption that you have access to the Microsoft 365 suite, which in most cases costs a lot of money to purchase if you don't obtain it from an academic institution. There is an alternative that offers some solutions to these drawbacks, but requires the introduction of another coding language: Python. It also involves purchasing another computer device called a Raspberry Pi. However, switching to this system of data recording has serious advantages since it allows for all the days to be stored onto the same document and is easily converted into a table using R. This section will provide a quick overview of the RasPi-Arduino setup. Luckily, you don't need many more extra materials other than the RasPi itself. The cord can be connected to the RasPi’s USB port. You will need a USB-C to connect the RasPi to a monitor to be able to see the program and run the code.

Make sure to go to the RasPi settings and **enable** the serial port and serial console.

Upload the following code as a *bash* script on the RasPi computer:

import serial

import datetime

import csv

ser=serial.Serial(‘/dev/ttyACM0’,9600)

ser.flushInput()

print(“Connected to Arduino”)

while True:

from datetime import datetime

time=(“m% %H:%M:%S”)

read\_serial=ser.readline()

print(read\_serial)

with open(‘weighttest.csv’,”a”) as csvfile:

writer=csv.writer(csvfile, dialect=’excel’)

writer.writerow([read\_serial,time])

print (“File has been created with your stored data”)

Afterwards, go to the terminal (making sure all libraries are uploaded) and type in the command python weighing\_sys.py (yours might be named differently than weighing\_sys.). Note: Once you turn on the power to RasPi, this will also power the Arduino and therefore start the uploaded program. To avoid the program beginning to water before you set up and type in the commands into the Command Prompt: Do NOT plug in the USB (Arduino-RasPi) connection to the RasPi until you have typed in and are ready to press enter on the python weighing\_sys.py. Once you press enter, the Arduino code will restart, so do not worry about the small bit of time that passes between plugging in the Arduino and pressing enter on the Command Prompt to run the code.

During the measuring process, you can see the pot number, pre-watering weight, and post watering weight displayed in the command terminal, so you know that it is recording and working properly. The code is set to display the final values after you press the button to continue onto the next pot, so do not worry if you do not see your values until it says “Next pot…” , they are stored.

This process required a more intermediate to advanced knowledge of Python but offers reliability, stability and storing onto one csv file.

## 

## Troubleshooting and Unsolved Mysteries

This project is a work in progress and still has some issues that have not yet been resolved. The best part about building this watering system is learning coding and electrical design by solving problems that come up. If you build this watering system and encounter these problems let us know! Or let us know if you find a solution to them.

### Uncontrollable Watering

Of the times that we encountered problems with this watering weighing system, the pump watering continuously was the most common cause of those issues. On these occasions, the pump would be working fine until a pot in the middle of the sequence, where it just all of a sudden it fails to turn off. This has been a problem reported by some people who have also purchased the Gikifun pump for scientific experiments. A MOSFET and a diode were added to the breadboard to help with this problem, and it greatly reduced the number of times the problem had occurred, however it did not get rid of it completely. In this case, the only thing that you can do is disconnect the Arduino from the power supply and continue the measurements for that day manually. Just because the watering system threw this problem out one day, it does not necessarily mean that it is unusable. Our experience with this machine is that following days it will work through all the samples.

**Note:**  due to the calculation within the watering loop it may sometimes seem like the pump is watering for a really long time and that this problem has occurred, but in actuality, it will just be working for a really long time to get the weight up to the target weight of your sample. It's always best to give the system the benefit of the doubt. Sometimes, the LCD screen can clue you in on if the system has encountered this problem. Usually, when the pump has malfunctioned and starts to water with no control, you might see nonsense characters and letters pop up on the LCD screen or the “Watering” text might start flickering. The former might be your best clue to shut off the Arduino before it overwaters the sample. This way you can preserve the sample data and finish watering it manually. The LCD screen often acts as a good indicator of what problem the Arduino is facing, so it is important to have the LCD screen viewable to you at all times.

### Stalls in Watering

On the other spectrum of things, the water pump not watering can also come up as a problem. The way this has presented itself is somewhere throughout the experiment a sample will say “Watering..” in the loop, but there will not be any water being pumped to the pot. The loop will continue to cycle through and continue to show “Watering…” but with nothing being watered. Similarly to the aforementioned problem, its occurrence doesn't mean that the watering system is unusable for the coming days. If the LCD screen is showing the loop still cycling through the watering and weighing parts of the loop: wait for a minute and let the Arduino cycle through the loop, sometimes the problem resolves itself and the code continues to run. You might want to try to gently shift the pump and move it within the water up and down or tap the pump, this has occasionally “woken up” the pump, and the code continued onwards. If you try all of these methods to no avail, disconnect the Arduino from the computer and continue your measurements manually.

### Overwatering

In the “Printable Checklist for Logging Measurements”, it is recommended that you have a scale available to you in case of needing to double-check the scale's accuracy. This might not be evident at the beginning of your experiment, but if you are taking measurements at set intervals you might begin to notice what a “normal” amount of water being dispersed might look like. You might then sometimes notice the pump seemed to water for a little too long, in which case it is always good to weigh it on another level scale and compare that value to the value recorded by the Arduino scale. Even though the scale was calibrated at the beginning of each experiment, sometimes the person taking the data noticed that the pump was overwatering the pots. There are two different situations with overwatering. If you noticed that every pot is getting overwatered, the first place to troubleshoot would be to re-upload the calibration code and see if the calibration factor needs to be adjusted. If the scale is sporadically overwatering pots, you need to experiment with the calculations of values for variables described in **Pump Testing**. Preferably, increase the cut-off for when the pump switches from calculated watering to set interval watering -or- decrease the time associated with each set of grams of water -or- increase the number that you are dividing the original difference by. If none of this works, try reconnecting another load cell.

### Leaky Spout

During the first few samples, make sure to pay close attention to all parts of the functioning of the system to catch any mishaps before you get far into your sample measurements. One problem that might happen depending on your setup is that you might notice that the watering system is watering in between loops or leaking continuously. This is actually not a fault of the pump! This can be solved with an alteration in your setup. Generally, the spout of the hose should not be lower than the water level or the brim of the container. This can be solved by obtaining a box with a flat bottom to put the system on. However, be wary of elevating the water pump too much, or else the pump will not be powerful enough to pump water through the hose into the pot. Whatever surface you decide to choose, make sure that it is level and rigid so that the stand is fully supported rather than a soft plastic container which will allow for the stand to ‘sink’ through a little and will alter your measurements.

### Code Errors

Alongside problems that result from the pump, some of the most frustrating problems can come from the code itself. While the code remains the same between each of the experiments, occasionally there might be something that you alter when you change the calibration factor or input the reference weights that throws something off. The Arduino IDE error messages generally are really informative and can lead you to the spot of the error effectively. There is also a large Arduino community and forums that you can post to with errors that feel unsolvable and sometimes people have experienced the same issues as you have!