

Sensing for Science: Exploring the World of Arduino and Sensors In the Classroom

The Instructor's Guide

By Kayce Williams



This material is based upon work supported in part by the National Science Foundation EPSCoR Cooperative Agreement OIA-1757351. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Welcome to the Sensing for Science Professional Development Program for Teachers. We will be exploring how to use Arduino and create an electrical device that will help us collect data on water quality. This simulates the research being done by Montana scientists and engineers who are part of the National Science Foundation-supported [CREWS project](#) (Consortium for Research on Environmental Water Systems). This guide contains a list of materials and steps in order to learn and teach Sensing for Science in your classroom.

The suggested grade level for this content is grades 5-12. But grades K-4 can be possible, adjustments need to be made.

The total time commitment for this content is 14 x 45-minute periods or roughly two weeks.

All lessons with video can be found by going to the Montana State University Science Math Resource Center Website <https://www.montana.edu/smrc/sensingforscience/> . Follow the QR code below to open the Sensing for Science web page.



Science Math
Resource Center



Table of Content

Level 1 the Basics	Page 5
Level 2 Sensors.....	Page 12
Level 3 Displays.....	Page 20
Level 4 Data Loggers.....	Page 36
Level 5 GPS modules.....	Page 52
Level 6 TDS Sensor Project	Page 61

This guide has the potential to achieve the following standards:

Montana Science standards:

Fourth-grade Physical Science—make observations to provide evidence of the transfer of energy from place to place by sound, light, heat, and electric currents

Six-Eighth Grade Physical Science—gather information to describe that synthetic materials come from natural resources and impact society; ask questions about data to determine the factors affecting electric and magnetic force strengths

Six-Eighth Grade Earth and Space Science—apply scientific principles to design a method for monitoring and minimizing a human impact on the environment;

Ninth-Twelfth Grade Physical Science—communicate through scientific and technical information roles of molecular-level structure in the functioning of designed materials; plan and conduct investigations to provide evidence that electric currents can produce magnetic fields and changing magnetic fields can produce electric currents; create a computational model to calculate the change in the energy of one component in a system when the change in energy of the other component and energy flows in and out of the system are known; design, build, and refine a device that works within given constraints to convert one form of energy into another form of energy; develop and use a model of two objects interacting through electric or magnetic fields to illustrate the forces between objects and the change in energy of the objects due to the interaction

Ninth-Twelfth Grade Life Science—design, evaluate, and refine a solution for reducing the direct and indirect impacts of human activities on the environment and biodiversity and analyze scientific concepts used by American Indians to maintain healthy relationships with environmental resources

ITEEA Standards of Technological Literacy:

The Nature of Technology

Standard 1—Scope of Technology

Standard 2— Core Concepts of Technology

Standard 3— Relationships Among Technologies and Other Fields

Technology and Society

Standard 5—Effects on the Environment

Standard 6—Impact of Society on Technology

Design

Standard 8—Attributes of Design

Standard 9—Engineering Design

Standard 10—Other Problem-Solving Approaches

Abilities for a Technological World

Standard 11—Apply Design Processes

Standard 12—Use and Maintain Products and Systems

Standard 13—Assess the Impact of Products and Systems

The Designed World

Standard 15—Agricultural and Related Biotechnologies

Standard 16—Energy and Power Technologies

Standard 17—Information and Communication Technologies

This is the content of one Sensing for Science Kit. It is recommended to have one kit per pair of students. All the items can be found on Amazon but sometimes it is cheaper to find them in other places. Buying in bulk is cheaper than buying individually. Some items can be substituted with different items. For example, as of writing this, Arduino Uno is about \$15 per unit, while Arduino Micro Pro is \$5 per unit. It would be cheaper to use the Arduino Micro Pro. I would need to check the pinouts and adjust the instructions. There are forums to help with this process, but you can always email: kayce.williams@montana.edu for assistance.

Full Kit Material lists

Arduino board (uno)
Bread Board
Jumper wire (20)
Resistors (220 and 10k ohm)
Potentiometer
LEDs (6)
9v battery harness
USB cable for Arduino
Photoresistor
LCD
LCD I2C adaptor
DF Robot TDS sensor
BN-180 GPS module
Micro SD module
Micro SD (8GB or less)
USB Micro SD Card Reader
buttons

Sensing for Science

Level 1

Level Summary:

This level will cover the basics of using Arduino to create electrical circuits. Topics covered in this lesson are electrical components, ohm's law, and coding.

Objectives:

Students will be able to:

- Identify electrical components and use them correctly in a given circuit.
- Create coding to control the function of a given circuit

Materials: (ideally you would need enough for each student, but enough for groups of two works too)

- computer or device to run Arduino IDE (a pc is preferred)
- Development boards (Arduino Uno is what we use at the level)
- Breadboards (any size will work)
- LEDs (2-3 volt dip packages)
- Resistors (package of 220 ohms will work)
- Jumper wires (at least two for each circuit)
- Random crafting supply for the activity

Estimated Time needed:

2 x 45 periods (20 minutes for lecture 25 for building circuits)

Preparation:

- Test Arduino IDE on all the PCs
- Check Uno boards and clear them with barebone code
- Have patience, things will not always work the way they should
- Go through this level before teaching it, get used to how everything works

Code Words to know:

- Void- returns nothing
- Loop- repeats
- Setup- starts up the code and electronics, runs once
- digitalWrite—write HIGH or LOW to a digital pin
- pinMode—sets a pin as an input or output
- Delay—default in milli-seconds, waits for a set time before moving on to the next line



More Resources Here:
www.montana.edu/smrc/sensingforscience/

Sources:

Arduino. (2018, February 5). What is Arduino? Retrieved July 8, 2022, from <https://www.arduino.cc/en/Guide/Introduction/>

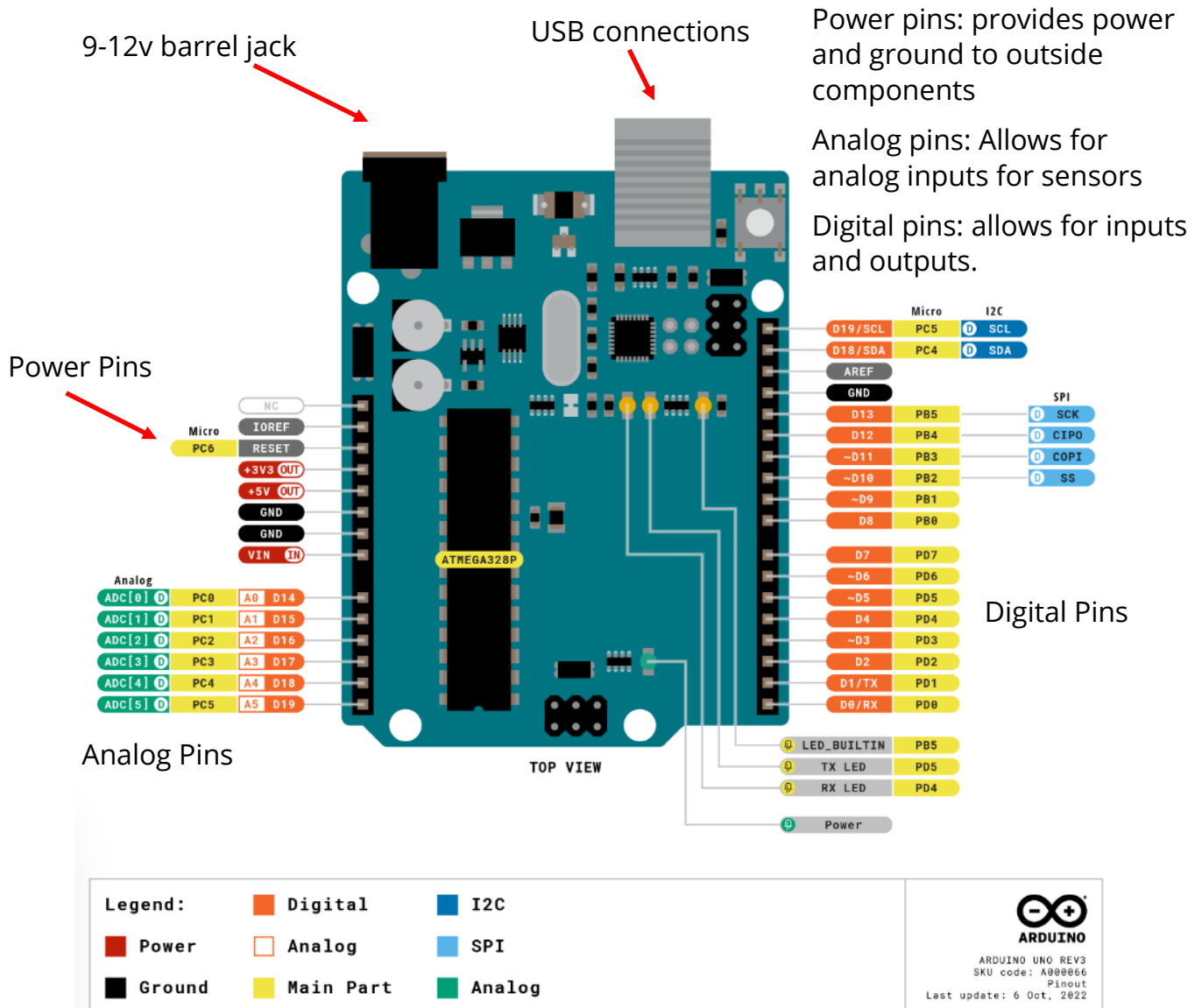
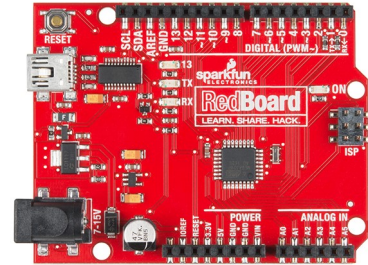
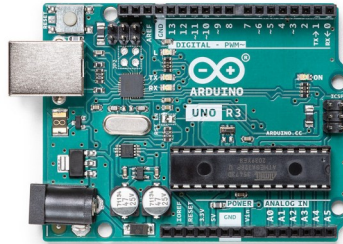
Electrical4U. "Light Emitting Diode (LED): What Is It & How Does It Work? | Electrical4U." <https://www.electrical4u.com/>, 28 Oct. 2020, www.electrical4u.com/led-or-light-emitting-diode/.

Science Buddies. (2019). How to Use a Breadboard. Science Buddies; Science Buddies. <https://www.sciencebuddies.org/science-fair-projects/references/how-to-use-a-breadboard>

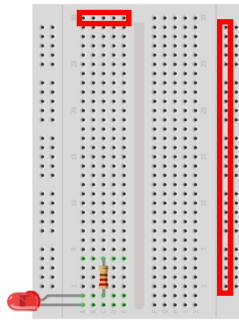
SparkFun Inventor's Kit for Arduino. Vol. 4.0, SparkFun electronics, Inc., 2017.

What is Arduino?

Is a company born of the Ivrea Interaction Design Institute, their goal was to provide a pathway for non-engineers to have the ability to develop electrical devices. Users of any skill can use Arduino. Arduino software and hardware is open sourced, meaning Arduino provides all their code and schematics for anyone to use. Therefore, there are many Arduino clones available to purchase as seen below. Many cool projects can be done with Arduino: DIY 3D printers, heart sensors, auto-watering gardens, and many more.



Breadboards are a fast and easy way to prototype circuits with the ability to make alterations and a low chance of damaging electric components. There are many types and sizes of breadboards. Our kits use a half-size breadboard that includes power rails and component rails. All power rails are connected vertically and component rails are connected horizontally in groups of 5 pins.

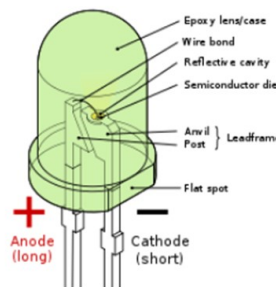
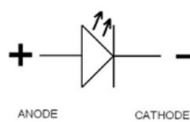
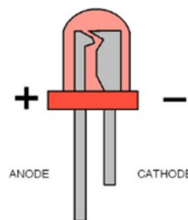


What is happening on the board? The connectors inside the board are friction clips that are all connected to provide continuity for the circuits. Photo from Sciencebuddies.com

Breadboards get their name from the old style of prototyping circuits. To test circuits they used slabs of wood (like a cutting board) and used nails and wire to build and test circuits before sending them off for manufacturing. It was a cheap method, but time-consuming and not easy to make changes if needed.

LEDs or light-emitting diodes are a semiconductor made from silicon when given the right amount of voltage will produce light. With too little voltage the LED does not light up. With too much voltage the LED is ruined. LED's light is directional, meaning light is the brightest in one direction of the device. LEDs also have polarity, meaning they only allow electrons to flow in one direction. To make sure we have the polarity right we need to have the negative leg going to the negative side of the battery or ground. The negative side can be identified in three ways: 1) the short leg side, 2) the flat edge of the bulb, and 3) the anvil (a larger piece of metal) inside the bulb.

What is a LED?



Electrical 4 U

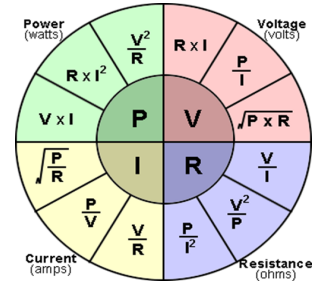
Resistors reduce the flow of electrons. They do not have polarity and come in all shapes and sizes. The color bands tell us what the R-value of the resistor is. They are very inefficient light bulbs, instead of making light, they dissipate energy by heat. We will use them to help control the voltage in our LED circuit.



STEM Moment-Science

This is a great opportunity to teach students about how different elements' valance electrons determine if a material will be a conductor, insulator, or semiconductor.

Ohm's Law is foundational in dealing with electronics. The law describes the relationship between voltage, current, and resistance. This relationship can be shown in this equation: $V=I \times R$. Check the chart to see all the different ways to write out ohms law based on what your knowns are. We can use Ohm's Law to find a resistor value we need for the LED circuit. The LED I am using has a forward voltage (the voltage required to work) of 3 volts with a current of .02 amps. Since we are using the LED in series with the resistor the amperage will be the same. We will be using a 5 volts power supply so the resistor needs to use up 2 volts. So we have two knowns for the Ohm's Law equation: $V=2$ volts and $I = .02$ amps. So now we use some algebra to figure out the R (resistance) value of our resistor: $2 = .02 \times R$. For the LED I am using I will need a 100-ohm resistor. Be sure to check the volt requirements for your LED and find the resistor value you will need. It is common to use 100 to 300 ohms worth of resistance in a LED circuit. Most kits will have a 220-ohm resistor for LED, you can use that.



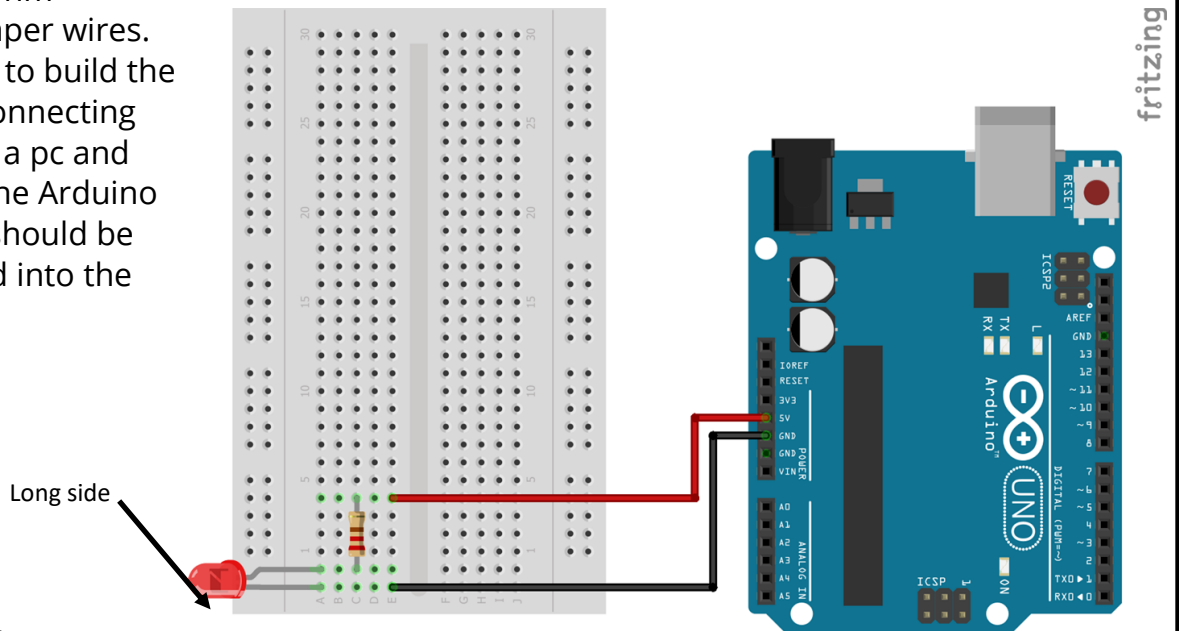
STEM Moment—Math

Ohm's Law is a great way to show students how we can use algebra in a real-world application. If you add serial and parallel circuits, then you can reinforce adding and subtracting fractions and using decimals.

Circuit 1

This circuit will make use of the information we learned so far and also test the power pins of the Arduino board. We will be using the following materials: Arduino board, Breadboard, USB cable, LED, 220-ohm resistor, and Jumper wires.

Use the diagram to build the circuit. Test by connecting the USB cable to a pc and plugging it into the Arduino board. The LED should be on when plugged into the USB cable.



Troubleshooting:

- My LED is not glowing but board is powered -> Check connections -> Check your polarity -> Try another LED
- My Arduino board is not turning on -> check for short circuits (power to ground connecting) -> check cable -> check PC USB port -> try another board

Code for Circuit 2 using the Arduino IDE

For us to start using the Arduino board more fully now we need to use the Arduino IDE (Integrated Development Environment). If you don't have Arduino IDE on a computer yet go to www.Arduino.cc/downloads and follow the instructions on installation. It is free! There are other coding programs out there (some work better than others) but use them at your own risk. Open the Arduino IDE and let's start coding. See the Sensing for Science Introduction video for help on installing the Arduino IDE.

When you open Arduino and start a new document, you will see the "bare bones" code needed to work our Arduino. Let's break down each part.

"Barebones" code

```
void setup() {  
  // put your setup code here, to run once:  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
}
```

Explore More

Check out all the Arduino Projects done by going to the link in the QR code.
projecthub.arduino.cc



Void setup

This section of code runs only once when the Arduino board is restarted (by a power-up or reset button). It is used for starting up any operation such as serial monitors, LCD screens, external modules, and pin settings. We are going to use this area to set up our digital pin 9 as an output. We do this by typing `pinMode(9, OUTPUT);` The semicolon is required when using this type of coding (C++). This tells the program end of the sentence and will give you an error if you forget.

```
void setup() {  
  pinMode (9,OUTPUT);  
}
```

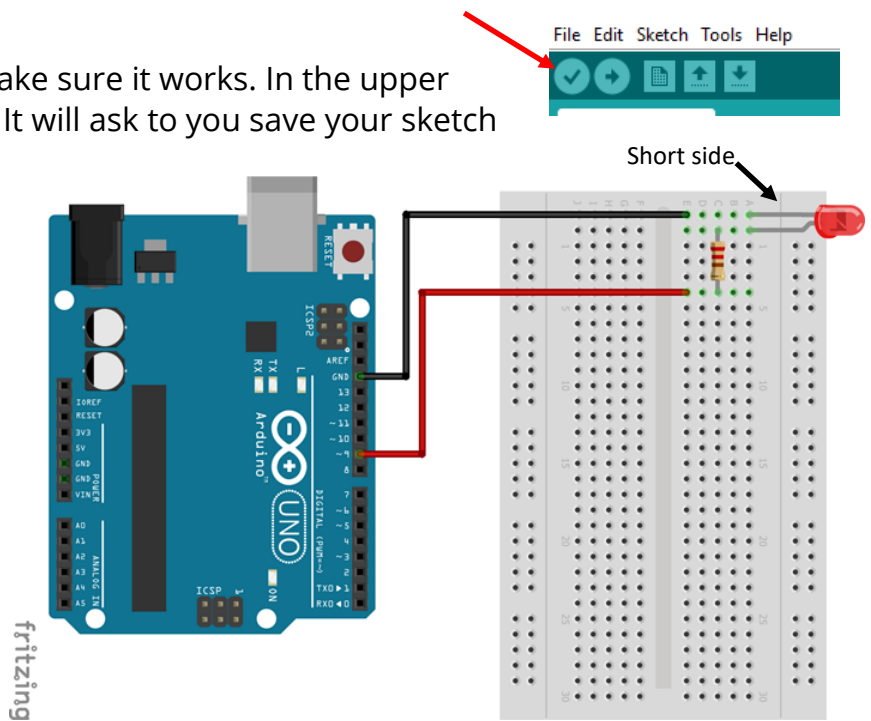
Void loop

This section of code is considered the main code. It is repeated over and over again until stopped. There are many things we can put into this section. We are going to use two commands for the next circuit `digitalWrite` and `delay`. The `digitalWrite` command is used to turn a digital pin HIGH (on) and LOW (off). The `delay` command is used to pause the program for milliseconds. Remember to place your semicolons after each line. Copy the code to the right into your IDE. This will turn the LED on for 1 second and off for 1 second and repeat.

```
void loop() {  
  
  digitalWrite (9,HIGH);  
  delay(1000);  
  digitalWrite (9,LOW);  
  delay (1000);  
}
```

Verify and Build Circuit 2

Now we should check our code to make sure it works. In the upper left-hand corner is the verify button. It will ask to you save your sketch (sketch is the name of the document) when you click it and then it will run through your code. If everything is correct you should see "compiling is complete" with no orange warnings. If something is wrong it will give you a warning in orange. Do not freak out, the error code might be overwhelming but with practice it can help you pinpoint your error (most likely you forgot a semicolon). Once everything is good, build the circuit by following the circuit diagram to the right.



It's time to upload! Plug your USB cable back into your Arduino and go to your code on Arduino IDE. At the top is the "Tools" menu. There is a "board:" area, check your board is selected here. Next, go to the "Port:" and select the port that the board is connected to, for Windows it is COM# (Arduino Uno). If you have multiple ports open try one and see if it works. Once your port is selected go back to the left-hand corner next to verify and click the upload. Your Arduino board should start blinking and you should see a loading bar in the IDE. If everything is working correctly you should get an "upload completed message" and your LED should start turning on and off. If not check the troubleshooting guide.

Troubleshooting:

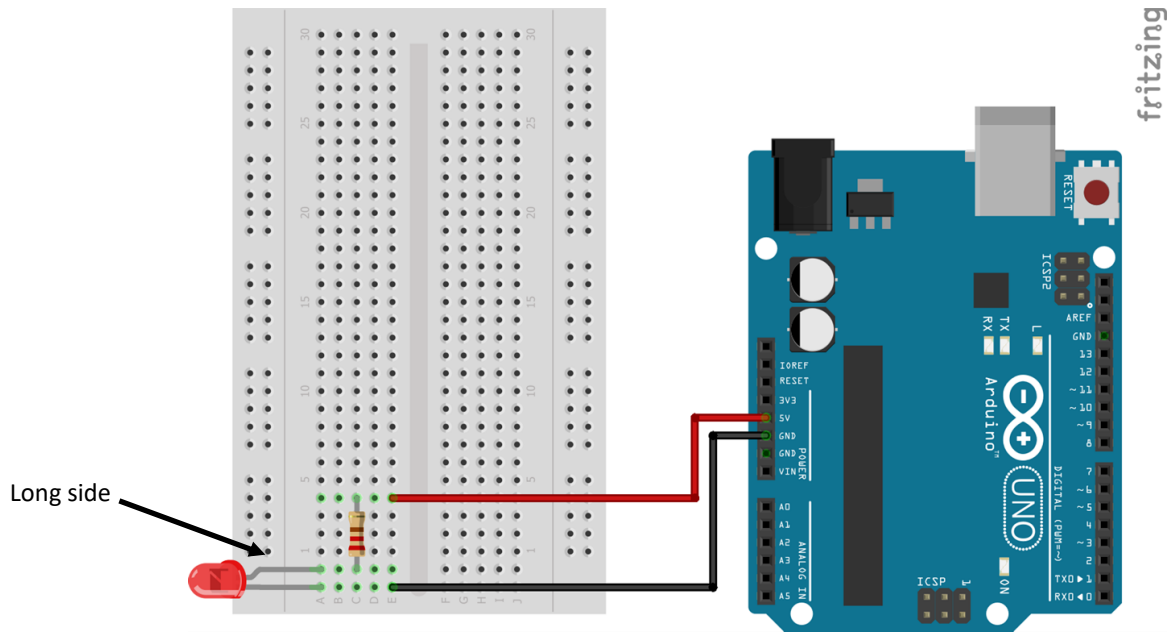
- I got an COM error -> go back to your port options and try another port -> still not working restart computer
- My LED is not on -> check your connections
- LED not blinking -> check code -> check connections

Activity Time—LED light show

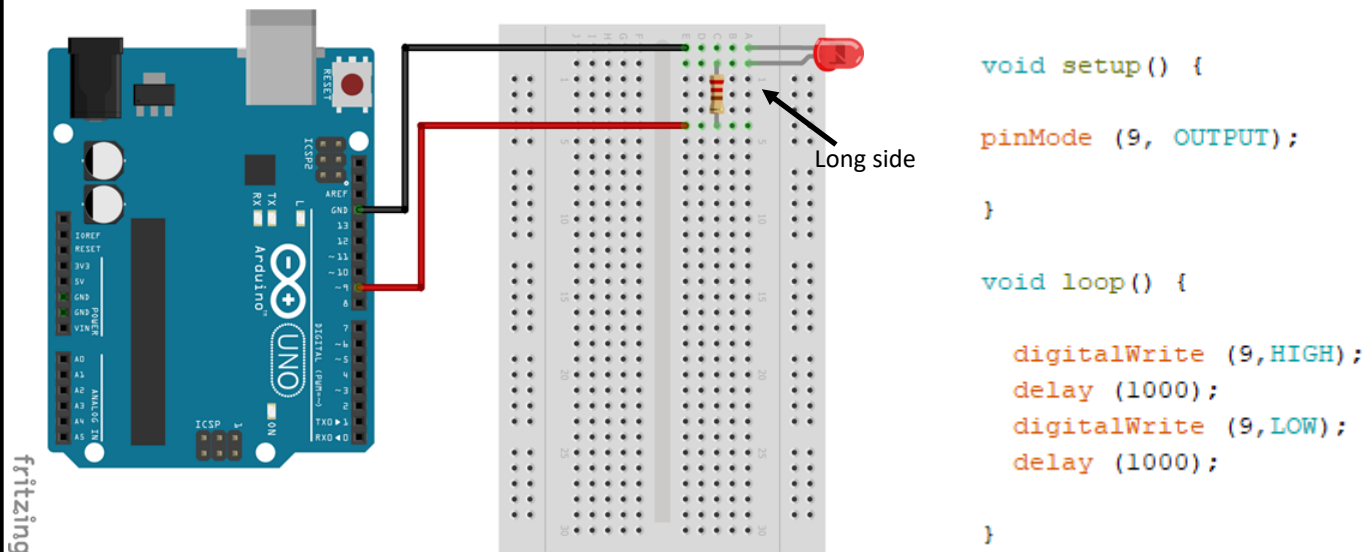
As a way to assess your students and to invite creativity, have your student design and build a light show. This activity requires students to add more LEDs to their circuits and think about their coding to make the light show the way they want it to work. Random craft items (straws, Ping-Pong balls, cups, etc.) can be used for this project. The students can design their light show to blink with music (a school song is a great example) or just create pretty patterns. Don't be afraid of students failing their designs, this is a great learning opportunity. Guide them but do not do the work for them.

Level One Student Printout cards: print enough for each group

Circuit #1



Circuit #2



Level Summary:

This level will use sensors in Arduino. Topics covered in this lesson are sensors, analog and digital signals, and variables.

Objectives:

Students will be able to:

- Describe what a sensor is and how we use them to control an electrical device
- Create coding with inputs controlling the outputs of a given circuit.

Materials: (ideally you would need enough for each student, but enough for groups of two works too)

- computer or device to run Arduino IDE (a pc is preferred)
- Development boards (Arduino Uno)
- Breadboards
- LEDs (2-3 volt dip packages)
- Resistors (package of 220 ohms will work)
- Jumper wires (at least two for each circuit)
- Photoresistor
- Potentiometer
- Flashlights (testing photoresistor circuit)

Estimated Time needed:

2 x 45 periods (20 minutes for lecture 25 for building circuits)

Preparation:

- Test Arduino IDE on all the PCs
- Check Uno boards and clear them with barebone code
- Have patience, things will not always work the way they should
- Go through this level before teaching it, get used to how everything works
- Test photoresistors and potentiometer (sometimes they don't work)

Code Words to know:

- int—integer, stored value between -32,768 to 32,797
- Serial.begin(9600) - starts the serial interface between the pc and Arduino board, baud rate is set in the parentheses.
- analogRead—reads data coming into the analog pins, range 0 -1023
- analogWrite—writes data to go out on the analog pins, range 0 -1023
- Serial.print—sends data from the Arduino board to the serial monitor
- Serial.println—sends data from the Arduino board to the serial monitor, auto enters down to the next line.
- If—this test a condition and runs code if the condition is true
- Else—this runs in the condition in the if statement is false



More Resources Here

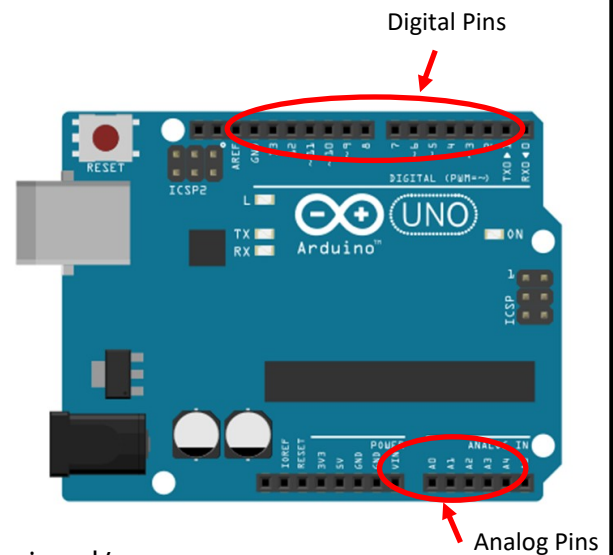
[www.montana.edu/smr/
sensingforscience/](http://www.montana.edu/smr/sensingforscience/)

Introducing Sensors

A sensor is a device that takes input from the environment and converts it into data for us to interpret. Sensors can be complex (dissolved oxygen) or simple (temperature sensor). On an Arduino board, most sensors will be set up the same way. We will explore two sensors at this level that will translate to more complex sensors.

All sensors will need to be connected to an input pin on the Arduino board. These inputs can be analog (number value ranging from 0 to 1023) or digital (on or off). Both have their advantages and disadvantages, these can be found at:

<https://www.softwaretestinghelp.com/analog-vs-digital-signal/>



Activity: Human Sensors

Here is an activity that uses your senses to identify possible issues with water quality. The materials required are a set of water samples that you collect. I recommend getting samples from the tap, local ponds/streams, and pools, or making your own (mixing salt into the water). Break the students into groups and give each group the first sample. Using sight, smell, and feel, have the student decide if the water is safe to drink or not. Repeat with each sample. Reveal where the samples came from after all the groups have reported their decisions. Explain how we can use our senses as sensors, but other sensors are required to better determine if water is safe to drink. You can use pH strips or premade sensors you may have in your labs to demonstrate.

Potentiometer

A potentiometer is a 3-pin variable resistor. The two outside pins are connected to the ground and power. The middle pin is the wiper, this pin's value changes when you turn the knob, giving us a different voltage value. Potentiometers are sensors, they convert the physical rotation of the knob to an electrical input that we can send to the Arduino and get a numerical value for us to interpret. The pins on a potentiometer are very delicate, so be careful when trying to place them on the breadboard.

Potentiometers are used anywhere you see a knob, such as volume control and speed control. They come in many types and configurations.



Circuit 1

In the first circuit, we will be creating a circuit that will read analog data from the potentiometer and output an digital signal to a LED. Let's get the code written out first. This code will introduce new commands for us to use.

Variables—before the void setup we can place lines of code here to create variables. We use variables to give the data (in or out) a name that we can use for interpreting. There are many variables we can create. We will be using the “int” variable. Typing “int” before setting a variable, tells the code that the variable is an integer. Variable names are case-sensitive. Keep variables simple and easy to recognize what data they are holding. The variables we will be setting are shown in the image to the right. The “sensor value” will be the data coming from the potentiometer and “output value” for the LED. We could give pins a variable name as well, but it is not required for this circuit.

```
int sensorValue = 0;
int outPutValue = 0;

void setup() {
```

Void setup

In this circuit, we will be using the void setup area to start up our serial monitor. The serial monitor is an addition to the Arduino IDE that allows us to see data read by the Arduino board for visual inspecting of data, calibrating of sensors, and debugging issues that we may have with the board. As we continue through the different levels of Sensing for Science we will be using the serial monitor. To start the serial monitor we will put Serial.begin(9600); This tells the Arduino board to start the serial monitor when the board is restarted and the serial monitor is opened. Our code for void setup should look like the code to the right.

```
void setup() {
  Serial.begin(9600);
}
```

This is the speed of communications between the board and the computer.

The Loop code

In level one we had pretty simple loop code, but now this area will start getting complex. But we will take it one line at a time and understand what each command does for us. See printout cards at the end of the level for the full code.

```
sensorValue = analogRead(A0);
```

Calling the variable we created earlier

analogRead is a command that tells the board to read the data coming in from A0 pin

```
outPutValue = map(sensorValue, 0, 1023, 0, 255);
```

Calls the variable we created earlier

The map commands take values from one range and convert them to another range. So the values from our sensorValue variable have a range from 0 to 1023 (the range analog pins read) and converts it to a range of 0 to 255 (the range of the digital pin output)

```
analogWrite(9, outPutValue);
```

This command will take the outPutValue and write it as an analog signal through a digital pin (9).

This command tells us to print onto the serial monitor. Using quotes creates text in the monitor. Writing a variable name or a pin will show the value in the serial monitor.

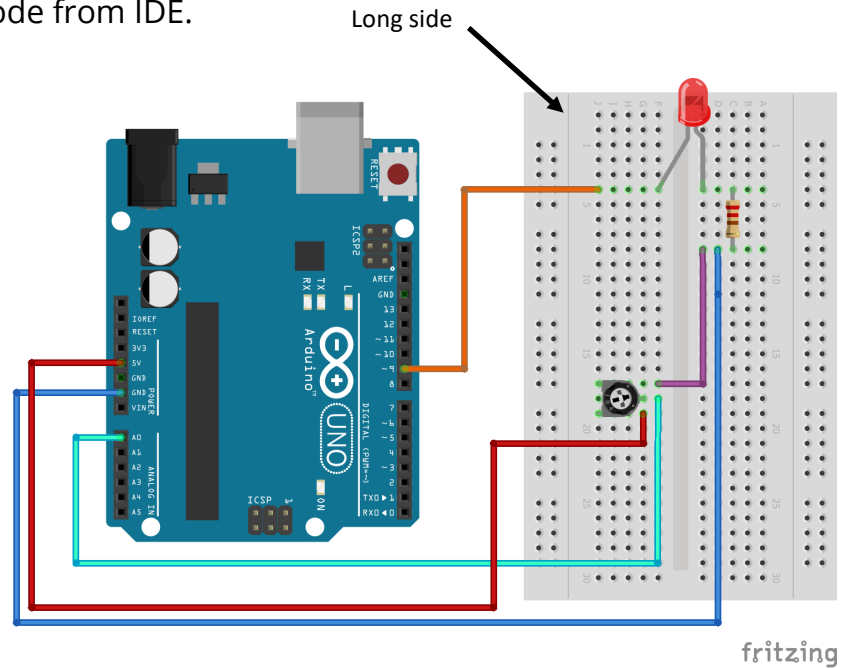
```
Serial.print ("sensor =");  
Serial.print (sensorValue);  
Serial.print ("\t ouput =");  
Serial.println (outPutValue);  
  
delay (2);
```

Adding the "ln" to the end of the print will create a new line in the monitor.

A delay allows time for the data values to settle when there are changes

That is all the code for this circuit. At this time it would be good to verify your code to see if you missed anything. Remember it is not the end of the world if you get an error message. Most likely it would be a missed semicolon or you have a character not the right case in your variables.

Construct the circuit below using the diagram. When you are finished connect your board to the USB cable and upload your code from IDE.



Once you have successfully uploaded your code to the Arduino board, keep the USB plugged in and open the serial monitor in the IDE. The serial monitor is in the upper right-hand corner and looks like a spyglass. A dialog box should open and within a few seconds, you should start seeing your data roll in. When turning the knob on the potentiometer you should see the values change on the serial monitor and your LED change in brightness. We have completed our first sensor circuit. Play around with this one before moving on to the next circuit for this level.



```
COM3
sensor =1      ouput =0
sensor =2      ouput =0
sensor =2      ouput =0
sensor =1      ouput =0
sensor =2      ouput =0
sensor =1      ouput =0
sensor =2      ouput =0
sensor =1      ouput =0
sensor =2      ouput =0
sensor =2      ouput =0
sensor =2      ouput =0
sensor =1      ouput =0
sensor =1      ouput =0
sensor =2      ouput =0
sensor =2      ouput =0
sensor
```

☒ Autoscroll ☐ Show timestamp Newline 9600 baud Clear output

Circuit 2

The second circuit we will be creating at this level is a light sensor. We can see this sensor used in automated greenhouses and newer cars that turn on headlights automatically. This sensor will be constructed with two components, a standard resistor, and a photoresistor. A photoresistor is a resistor that changes its resistance based on the level of ambient light. We are going to use the change in value for our sensor. Photoresistors do not have polarity and be used in either orientation.



Take a look at the code we will need for the circuit.

```
int lightLevel;
int threshold = 300;

void setup() {
  Serial.begin(9600);
}

void loop() {
  lightLevel = analogRead(A0);

  if(lightLevel < threshold){
    digitalWrite(9,HIGH);
  }
  else{
    digitalWrite(9,LOW);
  }
  Serial.print("light level =");
  Serial.println(lightLevel);

  delay(200);
}
```

This will be the placeholder for the data coming from the light sensor

Tell the board that analog data from A0 is the lightLevel.

Digitalwrite is either off or on.

This command writes the lightLevel value in the serial monitor for us to inspect

This is the level we set as a threshold. It is used to decide when the circuit will light up the LED. This can be adjusted to fit your photoresistor values.

Starting the serial port for monitoring data and calibrating.

If else statements are logic commands that run a certain code "if " the condition is true. Otherwise, use the other code. This code is saying if the lightLevel value is less than (<) 300, turn the LED on, otherwise keep the LED off.

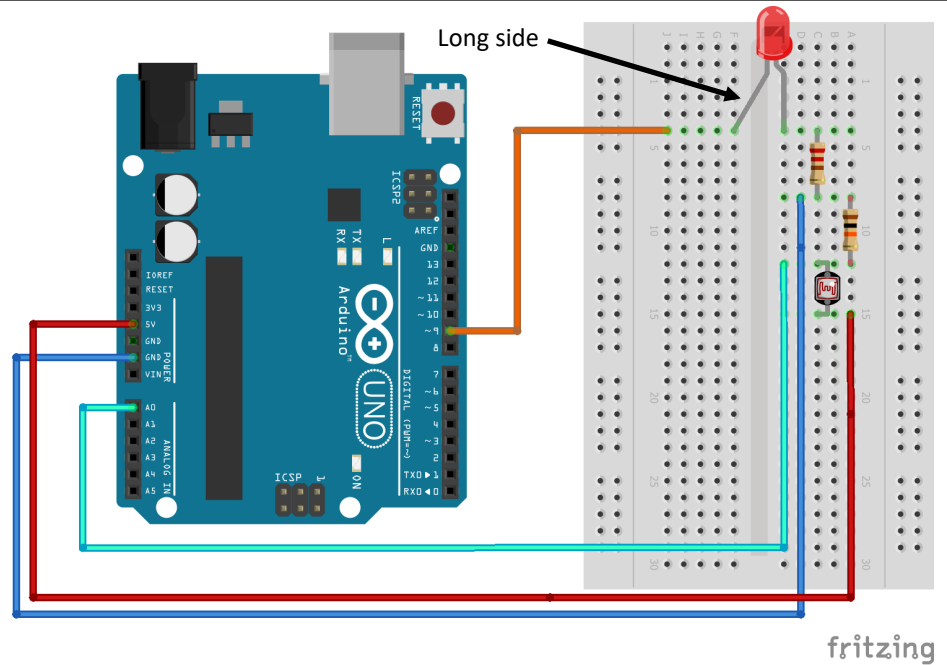
This delay allows time for us to debug the serial monitor but allows the code to repeat fast enough to change output with changes to input

Explore More

Want to explore other light-sensing devices. Here is a YouTube short from Collin's Lab about using phototransistors instead of a photoresistor. youtube.com/shorts/5HKvRrVWgYs



After you have verified your code and fixed any errors, it is time to build the circuit. Use the diagram to aid of the construction of the circuit.



Once you have confirmed you have constructed the circuit correctly, upload the code to the Arduino board. Once it is finished uploading, open the serial monitor and observe the values. Using your finger, cover and uncover the photoresistor and take note of the values. Also, take note of the LED, does it turn off and on? All photoresistors have some variant in them and the threshold might have to be changed for the circuit to work correctly. Play around with the threshold and the delay to see how the circuit reacts. You can take components or lines of code from Circuit 1 and use them in Circuit 2.

Troubleshooting guide:

- My LED doesn't turn on when I cover the photoresistor -> check connections -> check serial monitor to see if you are reaching thresholds -> replace LED
- Serial monitor going in a straight line not entering down -> missing a 'ln' in the last line of the serial print.
- Serial monitor showings nothing -> check you port
- Serial monitor values are not changing with the photoresistor -> bad photoresistor try another one
- Values are not changes when turning the nob of the potentiometer -> pins are not in the breadboard-> misaligned the pins of the potentiometer->damaged potentiometer try another one

Sources:

Analog Vs Digital Signal - What Are The Key Differences. (n.d.). www.softwaretestinghelp.com. <https://www.softwaretestinghelp.com/analog-vs-digital-signal/>

Jost, Danny. "What Is a Sensor?" FierceElectronics, 2019, www.fierceelectronics.com/sensors/what-a-sensor.

SparkFun Inventor's Kit for Arduino. Vol. 4.0, SparkFun electronics, Inc., 2017.

Level Two Student Printout cards: print enough for each group

C Potentiometer Code

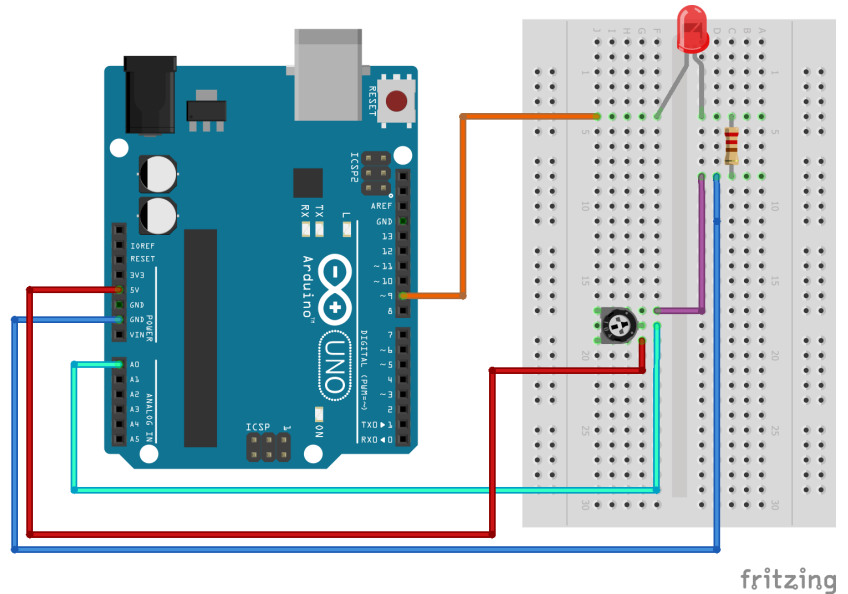
```
int sensorValue = 0;
int outPutValue = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  sensorValue = analogRead(A0);
  outPutValue = map(sensorValue, 0, 1023, 0, 255);
  analogWrite(9, outPutValue);

  Serial.print ("sensor =");
  Serial.print (sensorValue);
  Serial.print ("\t output =");
  Serial.println (outPutValue);

  delay (2);
}
```



fritzing

Light Sensor Code

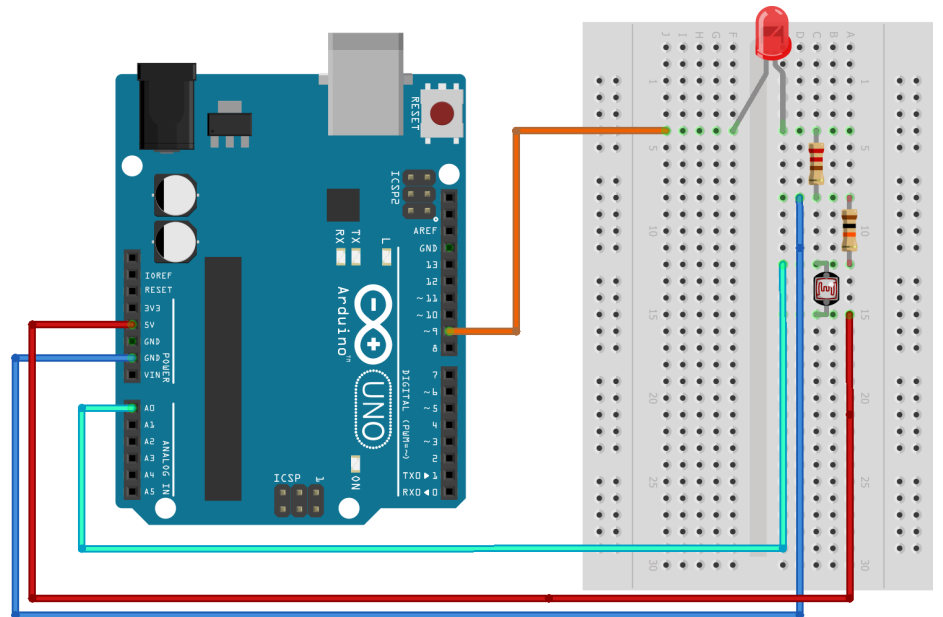
```
int lightLevel;
int threshold = 300;

void setup() {
  Serial.begin(9600);
}

void loop() {
  lightLevel = analogRead(A0);

  if(lightLevel < threshold){
    digitalWrite(9,HIGH);
  }
  else{
    digitalWrite(9,LOW);
  }
  Serial.print("light level =");
  Serial.println(lightLevel);

  delay(200);
}
```



fritzing

Level Summary:

This level will use visual displays in Arduino. Topics covered in this lesson are LCD displays, SPI and I2C hookups, and sending data from sensors to displays.

Objectives:

Students will be able to:

- Display text on the LCD with information about the data a given sensor is collecting.

Materials: (ideally you would need enough for each student, but enough for groups of two works too)

- computer or device to run Arduino IDE (a pc is preferred)
- Development boards (Arduino Uno is what we use in this book)
- Breadboards (any size will work)
- Resistors (package of 220 ohms will work)
- Jumper wires (at least 20 per kit)
- TMP36 (temperature sensor)
- Potentiometer
- LCD (Liquid Crystal Display)
- I2C adaptor

Estimated Time needed:

3 x 45 periods (20 minutes for lecture 25 for building circuits)

Preparation:

- Test Arduino IDE on all the PCs
- Check on uno boards and clear them with barebone code
- Have patience, things will not always work the way they should
- Go through this level before teaching it, get used to how everything works
- Test LCDs and tmp36 (sometimes they don't work)
- There are a lot of wires at this level. It is easy to mess up the wiring. Be prepared for troubleshooting many boards.

Code Words to know:

- `#include`—includes libraries with prebuilt code
- `const`—holds variables values
- `LiquidCrystal LCD ()` - this is an object created for the code. The first part calls the object type from the library. The second part is the name of the new object.
- `LCD.begin ()` or `LCD.init()` - starts up the display
- `LCD.clear()` - clears the display
- `LCD.setCursor()` - moves the cursor to different areas of the LCD.
- `LCD.print`—sends data from the Arduino board to the LCD.
- `LCD.println`—sends data from the Arduino board to the LCD, auto enters down to the next line.
- `while`—this will run code only when a condition is true.
- `LCD.backlight`—turns on the LED of the LCD when using i2c.
- `Float`—is a variable type that can use larger numbers and decimals



More Resources Here

www.montana.edu/smr/sensingforscience/

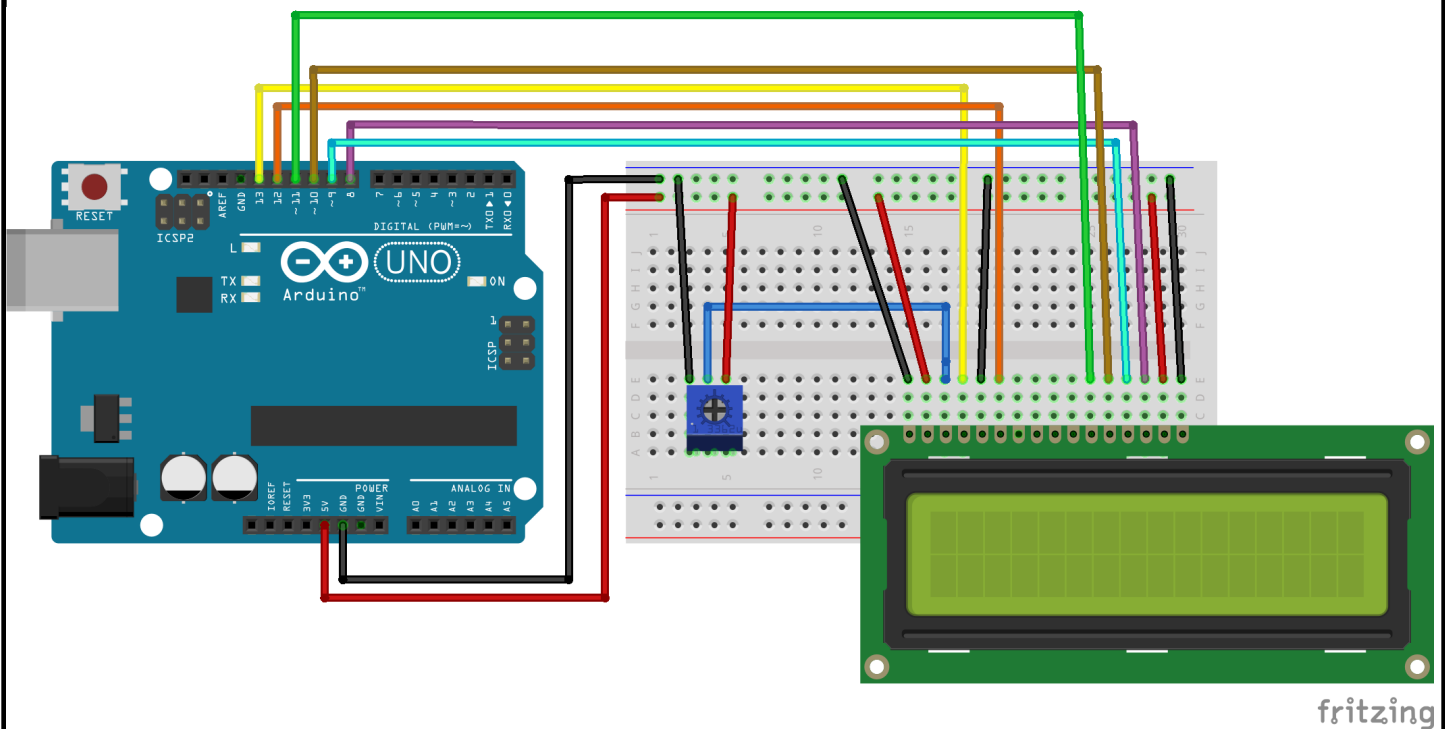
Displays

So far in this series we have used the Serial Monitor to display data. But if we want to take our circuit out in the field to collect data, it would be better to not have to carry the computer along. To solve this problem we can add a display to our circuit. There are many different types and sizes of displays we can purchase for our Arduino. We are going to use the liquid crystal display (LCD) for our display. This display will allow us to have 32 (2 rows of 16 characters) characters. We will construct the LCD using two different methods.



Circuit 1

Let's start with the basic functionality of the LCD. Looking at the diagram below wire up the LCD screen. There are a lot of wires, if you mess up it is ok. This is a great exercise in troubleshooting your wiring.



fritzing

Explore More

Here is a silly video about the history and function of liquid crystal displays.

<https://youtu.be/eGQQWlbD-nM>



Once you have the LCD screen wired up, we can write some code to test the circuit. Copy the code below and upload it to the Arduino board.

Here we are declaring that we are using the prebuilt library called LiquidCrystal. This allows us to use LCD commands in our code.

```
#include <LiquidCrystal.h>

const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  lcd.begin(16, 2);
  lcd.print("hello, world!");
}

void loop() {
}
```

Here we are creating an object called LCD that uses the following pins to work.

Starts sending data to the LCD. We must include how many rows and columns we have on the LCD

Here we are matching the pin out of the LCD to the pin out of the Arduino board.

Just like printing in the serial monitor, we can print to the LCD screen by using the LCD.print command.

Troubleshooting

My LCD is not lighting up -> check you wiring connects-> check for broken wires

My LCD is lighting up but I don't see anything-> turn the knob on potentiometer to change contrast-> check wiring

My LCD screen has a bunch of mumbo jumbo-> reset the Arduino board->check wiring

Now that you have successfully sent data to the display, let us explore how we can use the LCD. We are going to make a revolving meal menu and a temperature display. These are just a few things you can use displays with. There are more uses and different types of displays out there and they all work pretty much the same.

Code 2 for Circuit 1

This next section of code to copy will produce a revolving menu in a dining hall. This code should start up and display "today's menu") and then start looping through the different meals. Once finished writing code, upload and test.

```
#include <LiquidCrystal.h>

const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {

  lcd.begin(16, 2);

  lcd.print("Today's Menu");

  delay(3000);

  lcd.clear();
}

void loop() {

  lcd.setCursor(0,0);
  lcd.print("breakfast");
  lcd.setCursor(0,1);
  lcd.print("pancakes");
  delay(3000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("lunch");
  lcd.setCursor(0,1);
  lcd.print("sandwich");
  delay(3000);
  lcd.clear(); lcd.setCursor(0,0);
  lcd.print("dinner");
  lcd.setCursor(0,1);
  lcd.print("tacos!");
  delay(3000);
  lcd.clear();

}
```

This new command clears the display. Otherwise, the LCD will stack on top of other characters.

Setting a cursor on the top row.

Setting a cursor on the second row.

Noticed I messed up here and didn't enter down between codes, but it still works.

Code 3 for circuit 1

This next code allows us to send data to the LCD from the serial monitor. It is almost like sending instant messages, but it is only on the LCD. Copy the code below and follow the instructions. This code can also be found in the file->examples->liquidcrystal->serialdisplay. If you use the example code, make sure you make changes to the pinouts for the LCD.

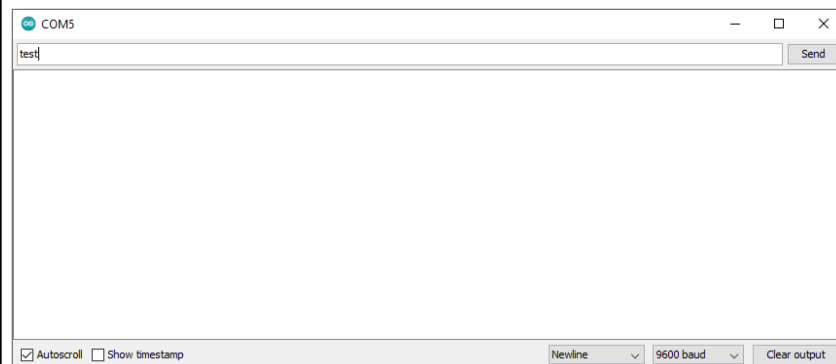
```
#include <LiquidCrystal.h>

const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
void setup()
{
  lcd.begin(16, 2);
  Serial.begin(9600);
}

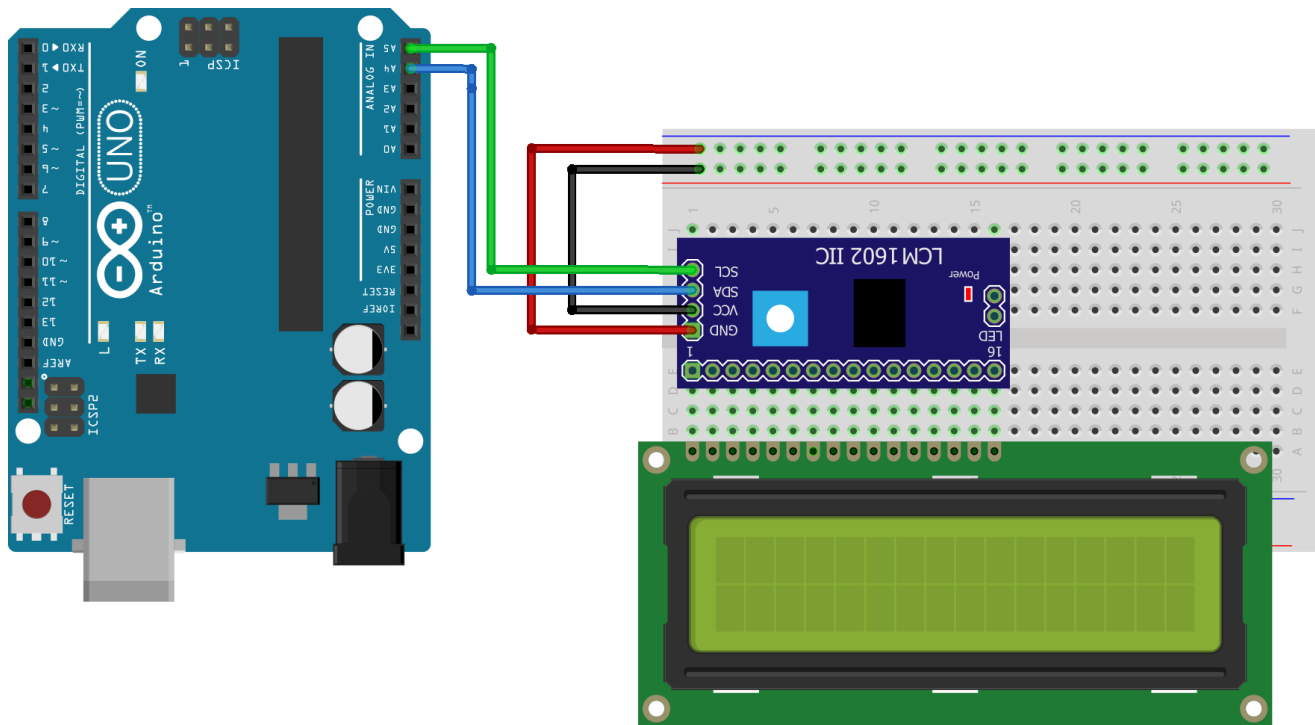
void loop()
{
  // when characters arrive over the serial port...
  if (Serial.available()) {
    // wait a bit for the entire message to arrive
    delay(100);
    // clear the screen
    lcd.clear();
    // read all the available characters
    while (Serial.available() > 0) {
      // display each character to the LCD
      lcd.write(Serial.read());
    }
  }
}
```

Upload the code to your Arduino board and open the serial monitor. At the top of the serial monitor is a command bar where we can type in. Type "test" and click send. You should see "test" show up on your LCD screen. Remember we can only have 16 characters and spaces count as a character.



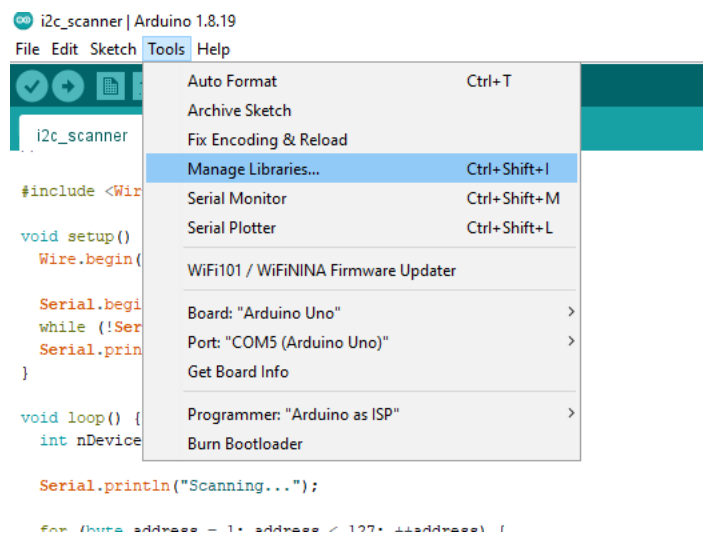
Circuit 2

Now that we have the basic functionality of the LCD screen down, let us look at a way to simplify the wiring. We are going to use the I²C ([inter-integrated circuit](#)) adaptor for the LCD to condense the wires down to 4. To start find your I²C adaptor and attach it to your LCD. Next, create the circuit below. See Sensing for Science Level 3 video for using I2C adapter.

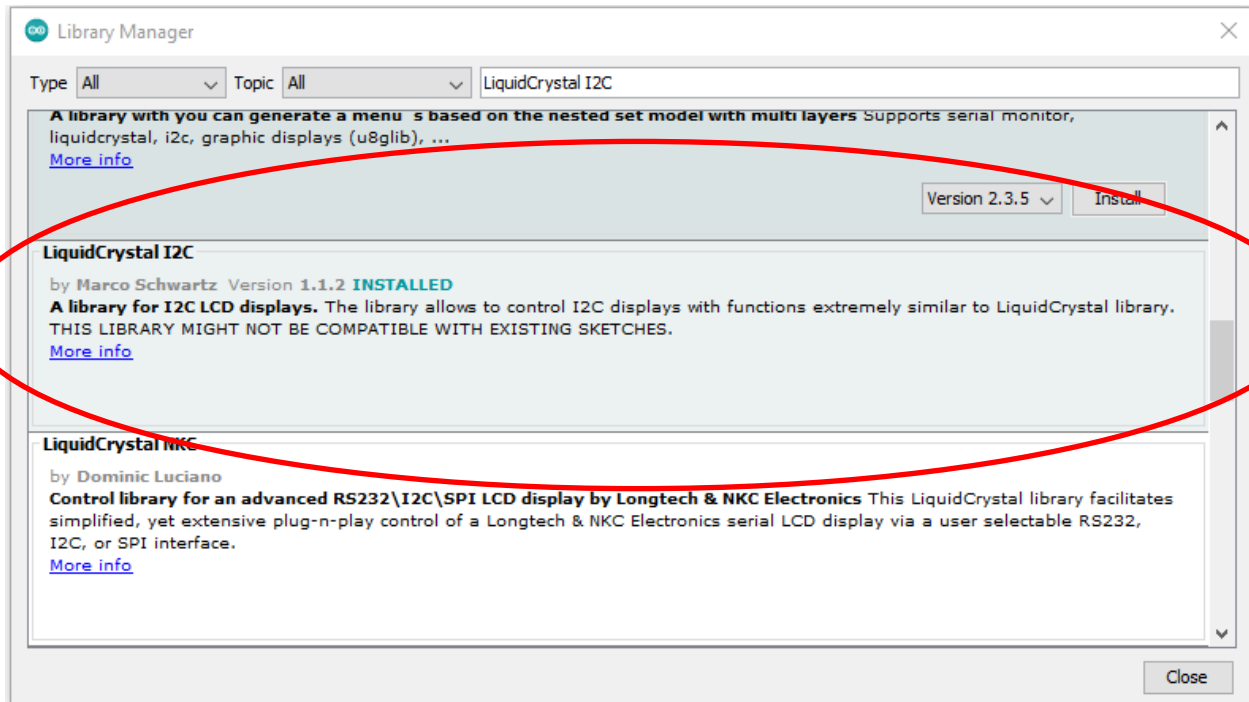


fritzing

Once you have the circuit built, let's look at the first code we are going to use. When using I²C we need to find the address of the device. Most devices you purchase will come with an address labeled. But what if you need to find the address? Well, there is a code prebuilt in the Wire library. We will need to install the LiquidCrystal I2C library at this time as well. In Arduino go to Tools->manage libraries.



When the library manager opens, search for “liquidcrystal I2C”. You will need to scroll down a bit and find the right library by Marco Schwartz. Click the install button and wait for the library to be installed and close out of the library manager.



Find the i2c scanner under File->Example->Wire->ic2_scanner. Open that sketch and upload it to your board. Once the code is uploaded, open the serial monitor and watch the code do its job. You will be given a hex address that you should write down, we will be needing it later. Mine for this device is 0x27. In more complex circuits there would be a need to change addresses of devices but that is outside the scope of this series.



Now that we have the device address we can start playing with the LCD again using the I²C adaptor. Most of the code will still be the same as before with differences. Let us look at the code on the next page and see what is different. This code can be found in examples under the LiquidCrystal I2C library labeled hello world. You will need to make some changes to the example code to work with our display of 16x2. The example uses a 20x4 display.

Includes the library for I2C LCD displays.

```
#include <LiquidCrystal_I2C.h>
```

```
LiquidCrystal_I2C lcd(0x27,16,2);
```

Here we are creating an object like before, with minor differences. The first number is the address that we found before. The second is the number of characters per line. The third is the number of rows.

```
void setup()
```

```
{
```

```
  lcd.init();
```

```
  // Print a message to the LCD.
```

```
  lcd.backlight();
```

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

```
  lcd.print("Hello, world!");
```

```
  lcd.setCursor(0,1);
```

```
  lcd.print("How are you?");
```

```
}
```

Instead of LCD.begin we use lcd.init to start up the LCD now.

This turns on the LED backlight.

```
void loop()
```

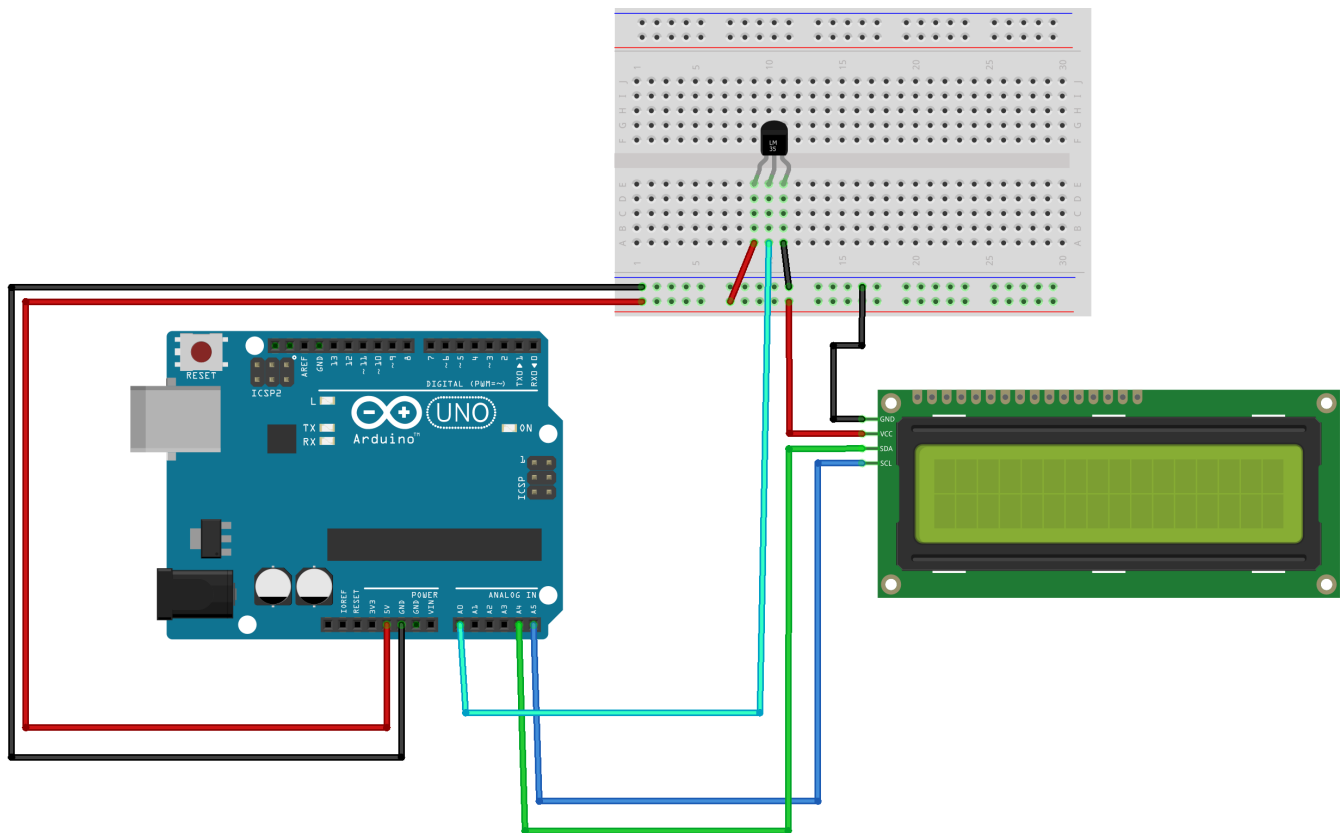
```
{
```

```
}
```

As you can observe, with only a few differences, the LCD still functions the same, but with only 4 wires. This will help simplify circuits when we start adding more components and will need to have pins available to us.

Circuit 2

Now that we are comfortable with the LCD screen, its time to add a sensor. We are going to create a temperature reader. We will be using a TMP36 sensor. These are cheap but pretty precise sensors. If you look at it with the flat side towards you the left pin needs to be connected to 5v. The middle pin needs to be connected to A0. The right pin needs to be connected to the ground. Now if you have the 5v and ground flipped you will know, the sensor



fritzing

gets very hot when this happens and will burn you and start melting the plastic around it. When you power up the circuit and you feel the heat, power down right away and flip the connections. Use the diagram below to help with construction.

The next page has the code that we will be using. One of the things you might notice is that we can use arithmetic in our code. We use arithmetic in this project to convert the voltage reading from the A0 pin to a value we can understand (temperature). You may notice that we can print variable values into the LCD.

Troubleshooting

- Sensor is very hot-> gnd and 5v needs to be flipped
- LCD is showing jumbo->check connects-> check code->reset board

Code for Circuit 2

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);
float voltage = 0;
float degreesC = 0;
float degreesF = 0;
void setup()
{
  lcd.init();
  lcd.backlight();
  lcd.clear();
}

void loop()
{
  voltage = analogRead(A0) * 0.004882813;
  degreesC = (voltage - 0.5) * 100.0;
  degreesF = degreesC * (9.0 / 5.0) + 32.0;

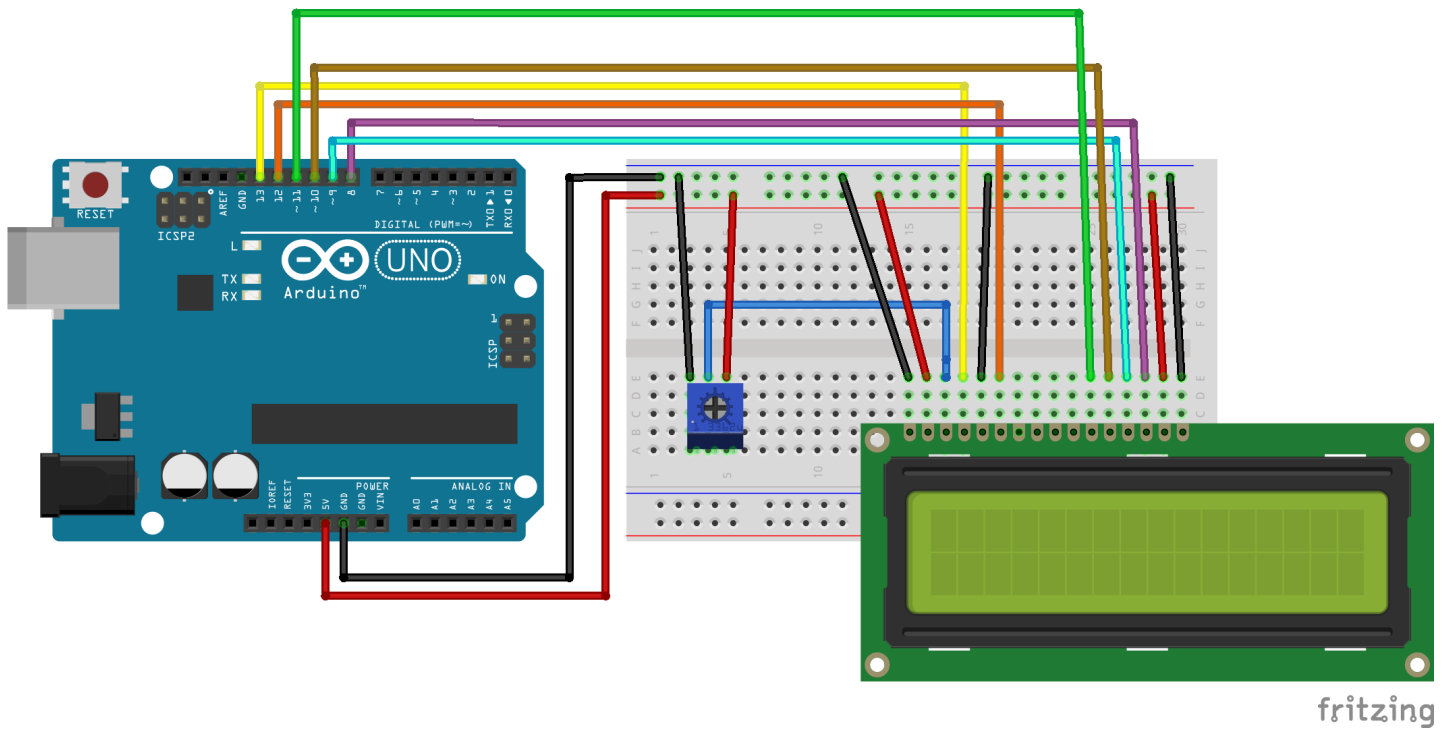
  lcd.clear();

  lcd.setCursor(0, 0);
  lcd.print("Degrees C: ");
  lcd.print(degreesC);

  lcd.setCursor(0, 1);
  lcd.print("Degrees F: ");
  lcd.print(degreesF);

  delay(1000);
}
```

Level Three Student Printout cards Circuit 1 and Code 1



Hello World Code

```
#include <LiquidCrystal.h>

const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {

  lcd.begin(16, 2);

  lcd.print("hello, world!");
}

void loop() {

}
```

Menu Code

```
#include <LiquidCrystal.h>

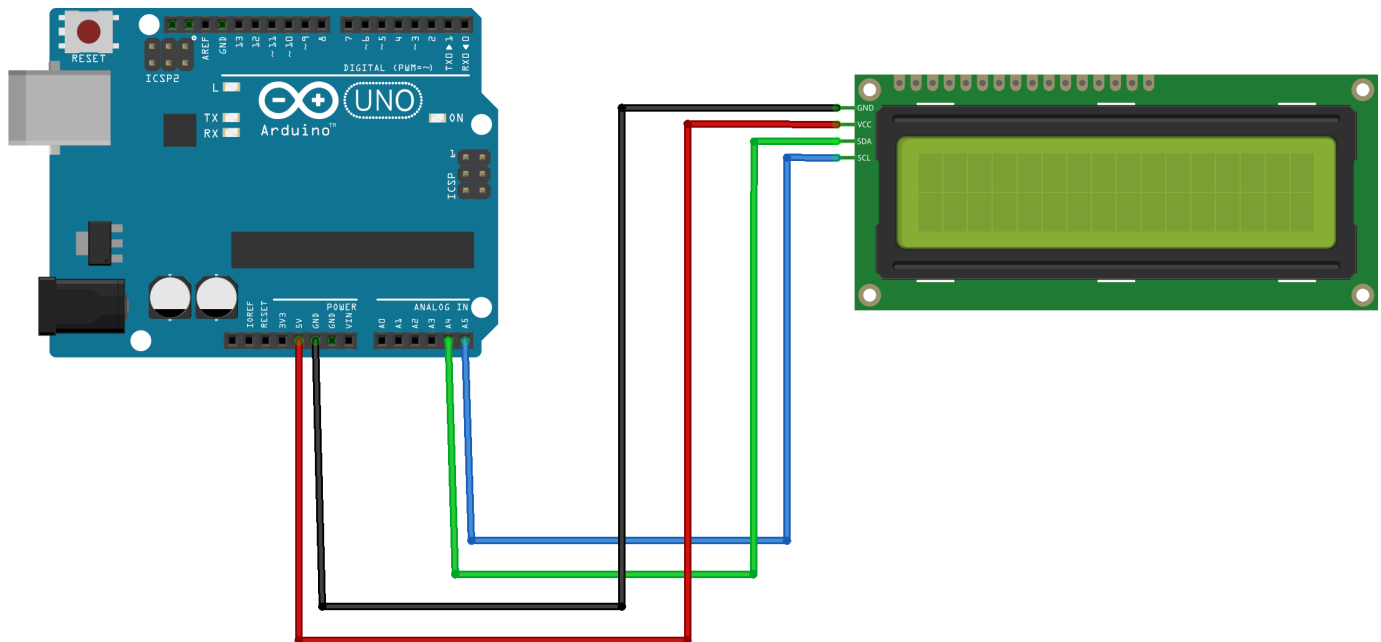
const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  lcd.begin(16, 2);
  lcd.print("Today's Menu");
  delay(3000);
  lcd.clear();
}

void loop() {
  lcd.setCursor(0,0);
  lcd.print("breakfast");
  lcd.setCursor(0,1);
  lcd.print("pancakes");
  delay(3000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("lunch");
  lcd.setCursor(0,1);
  lcd.print("sandwich");
  delay(3000);
  lcd.clear(); lcd.setCursor(0,0);
  lcd.print("dinner");
  lcd.setCursor(0,1);
  lcd.print("tacos!");
  delay(3000);
  lcd.clear();
}
```

<pre> #include <LiquidCrystal.h> const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8; LiquidCrystal lcd(rs, en, d4, d5, d6, d7); void setup() { lcd.begin(16, 2); Serial.begin(9600); } void loop() { // when characters arrive over the serial port... if (Serial.available()) { // wait a bit for the entire message to arrive delay(100); // clear the screen lcd.clear(); // read all the available characters while (Serial.available() > 0) { // display each character to the LCD lcd.write(Serial.read()); } } } </pre>	<pre> #include <LiquidCrystal.h> const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8; LiquidCrystal lcd(rs, en, d4, d5, d6, d7); void setup() { lcd.begin(16, 2); Serial.begin(9600); } void loop() { // when characters arrive over the serial port... if (Serial.available()) { // wait a bit for the entire message to arrive delay(100); // clear the screen lcd.clear(); // read all the available characters while (Serial.available() > 0) { // display each character to the LCD lcd.write(Serial.read()); } } } </pre>
--	--



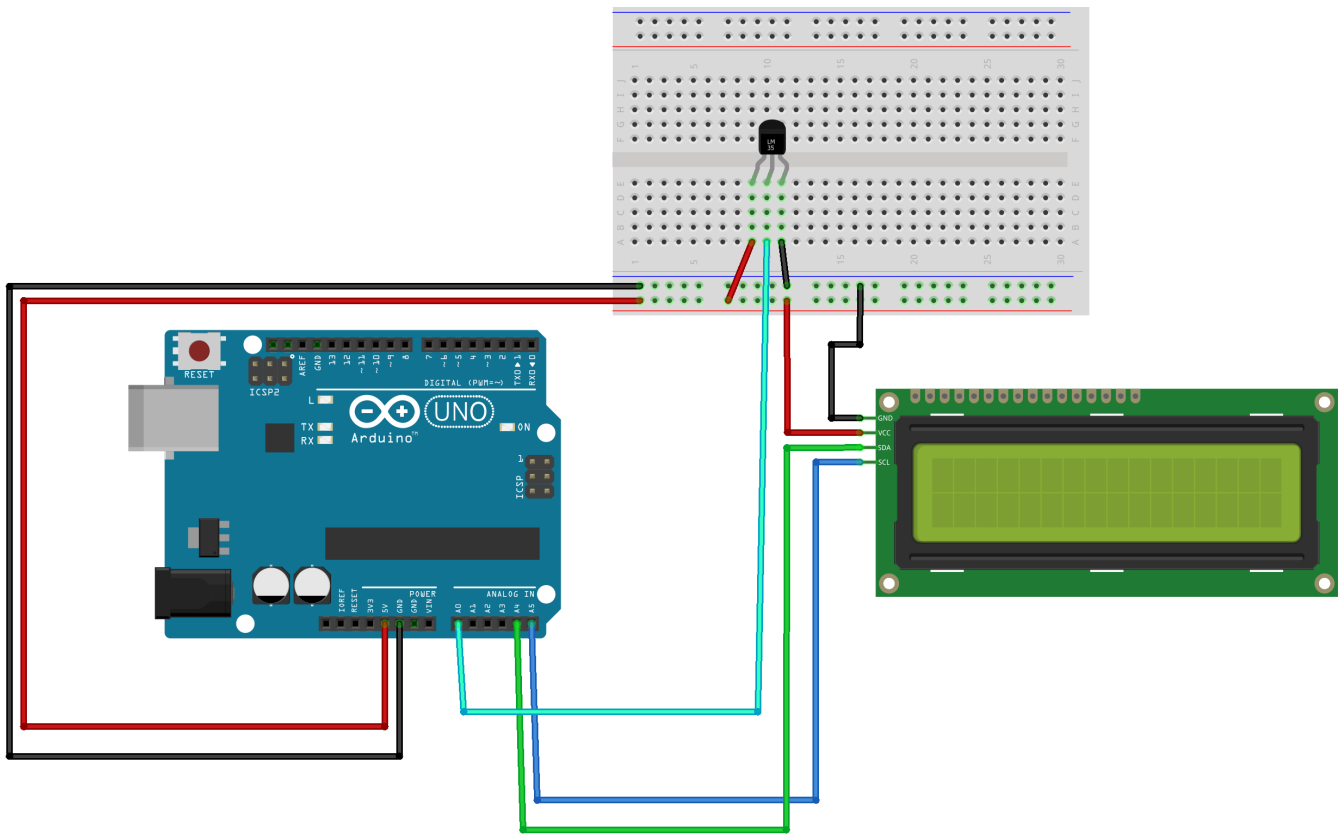
```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
  lcd.init();
  // Print a message to the LCD.
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Hello, world!");
  lcd.setCursor(0,1);
  lcd.print("How are you?");
}

void loop()
{
}
```

Level Three Student Printout cards Circuit 3



fritzing

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);
float voltage = 0;
float degreesC = 0;
float degreesF = 0;
void setup()
{
  lcd.init();
  lcd.backlight();
  lcd.clear();
}

void loop()
{

  voltage = analogRead(A0) * 0.004882813;
  degreesC = (voltage - 0.5) * 100.0;
  degreesF = degreesC * (9.0 / 5.0) + 32.0;

  lcd.clear();

  lcd.setCursor(0, 0);
  lcd.print("Degrees C: ");
  lcd.print(degreesC);

  lcd.setCursor(0, 1);
  lcd.print("Degrees F: ");
  lcd.print(degreesF);

  delay(1000);
}
```

Level Summary:

This level is all about creating a data logger. Topics covered in this lesson are SD card reader shields, and reading/writing data.

Objectives:

Students will be able to:

- Create a data logger using given sensors and a SD card module.

Materials: (ideally you would need enough for each student, but enough for groups of two works too)

- computer or device to run Arduino IDE (a pc is preferred)
- Development boards (Arduino Uno is what we use in this book)
- Breadboards (any size will work)
- Resistors (package of 220 ohms will work)
- Jumper wires (at least 20 per kit)
- TMP36 (temperature sensor)
- Potentiometer
- LCD (Liquid Crystal Display)
- I2C adaptor
- USB SD card reader
- SD Card Shield
- Micro SD card



More Resources Here:

<https://www.montana.edu/smrc/sensingforscience/>

Estimated Time needed:

3 x 45 periods (20 minutes for lecture 25 for building circuits)

Preparation:

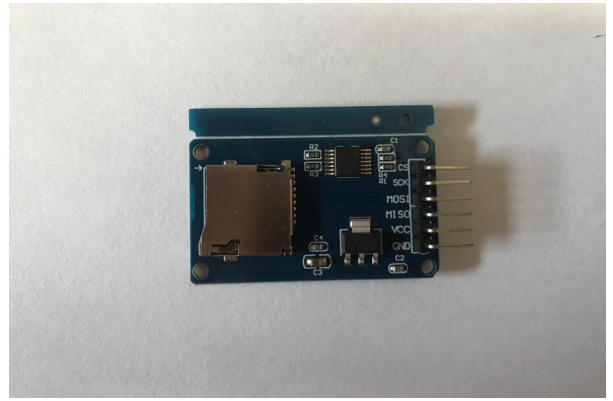
- Test Arduino IDE on all the PCs
- Check on uno boards and clear them with barebone code
- Reformat sd cards (this clears data and prepares the cards, FAT32 works)
- Go through this level before teaching it, get used to how everything works
- Test LCDs and tmp36 (sometimes they don't work)

Code Words to know:

- `myfile.open`—when using the sd library, this opens the sd card
- `myfile.close`—when using sd library, this closes the sd card
- `Myfile.available` - this is used with if or while to check if sd card and its file is ready.
- `Myfile.read` - reads the data off sd card file

Data loggers

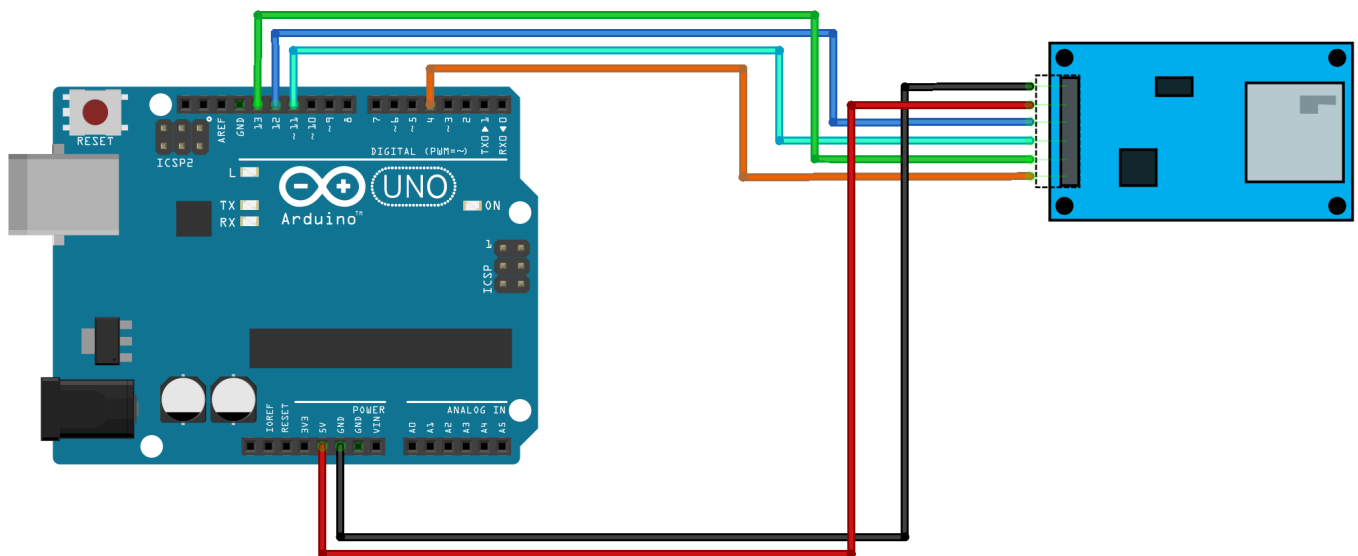
We want to go out and collect data from the world. We could take our Arduino circuit and a laptop out to collect data, but that gets challenging with all the equipment and the fear of dropping your laptop. There is the current trend of using IoT (Internet of Things) to send data via the Internet, which works great when there is good internet. In Montana, an Internet connection is not guaranteed. So we will save data locally by using a data logger SD card module. This device will allow us to save data to a micro SD card for us to view later.



Circuit 1

Let's start with testing the module and the SD card. Wire up the following circuit .

CS->4, SCK->131, Mosi->11, Miso->12



fritzing

Explore More

Arduino has an Internet of Things (IoT) cloud system that can be used to create cool circuits and control them remotely via wifi or Bluetooth. This requires an IoT Arduino board and a device (Ipad or laptop). Here is an example of an IoT project by DroneBot Workshop. youtu.be/pgGpuws7f9o

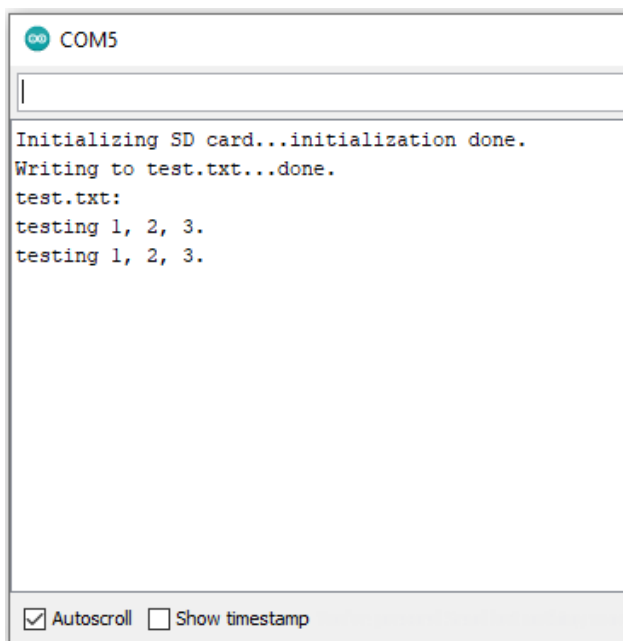


Once you have the SD card module wired up, we can write some code to test the circuit. Copy the code and upload it to the Arduino board. This code is also under examples in the SD card library.

Upload the code to the Arduino board and open the serial monitor. If everything is working correctly, you should see the results shown below. If you get an error message you might have to reformat your SD card (see formatting steps in the instruction video) or your SD card is damaged.

If you have a successful test you should be able to pull the SD card and open the file on the pc. It will show up as a .txt file.

We will break down the code in the next circuit.



The screenshot shows the serial monitor window for COM5. The output text is as follows:

```
Initializing SD card...initialization done.
Writing to test.txt...done.
test.txt:
testing 1, 2, 3.
testing 1, 2, 3.
```

At the bottom of the window, there are two checkboxes: "Autoscroll" (checked) and "Show timestamp" (unchecked).

```
#include <SPI.h>
#include <SD.h>

File myFile;

void setup() {

  Serial.begin(9600);
  while (!Serial) {
    ;
  }

  Serial.print("Initializing SD card...");

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    while (1);
  }
  Serial.println("initialization done.");

  myFile = SD.open("test.txt", FILE_WRITE);

  if (myFile) {
    Serial.print("Writing to test.txt...");
    myFile.println("testing 1, 2, 3.");

    myFile.close();
    Serial.println("done.");
  } else {

    Serial.println("error opening test.txt");
  }

  myFile = SD.open("test.txt");
  if (myFile) {
    Serial.println("test.txt:");

    while (myFile.available()) {
      Serial.write(myFile.read());
    }

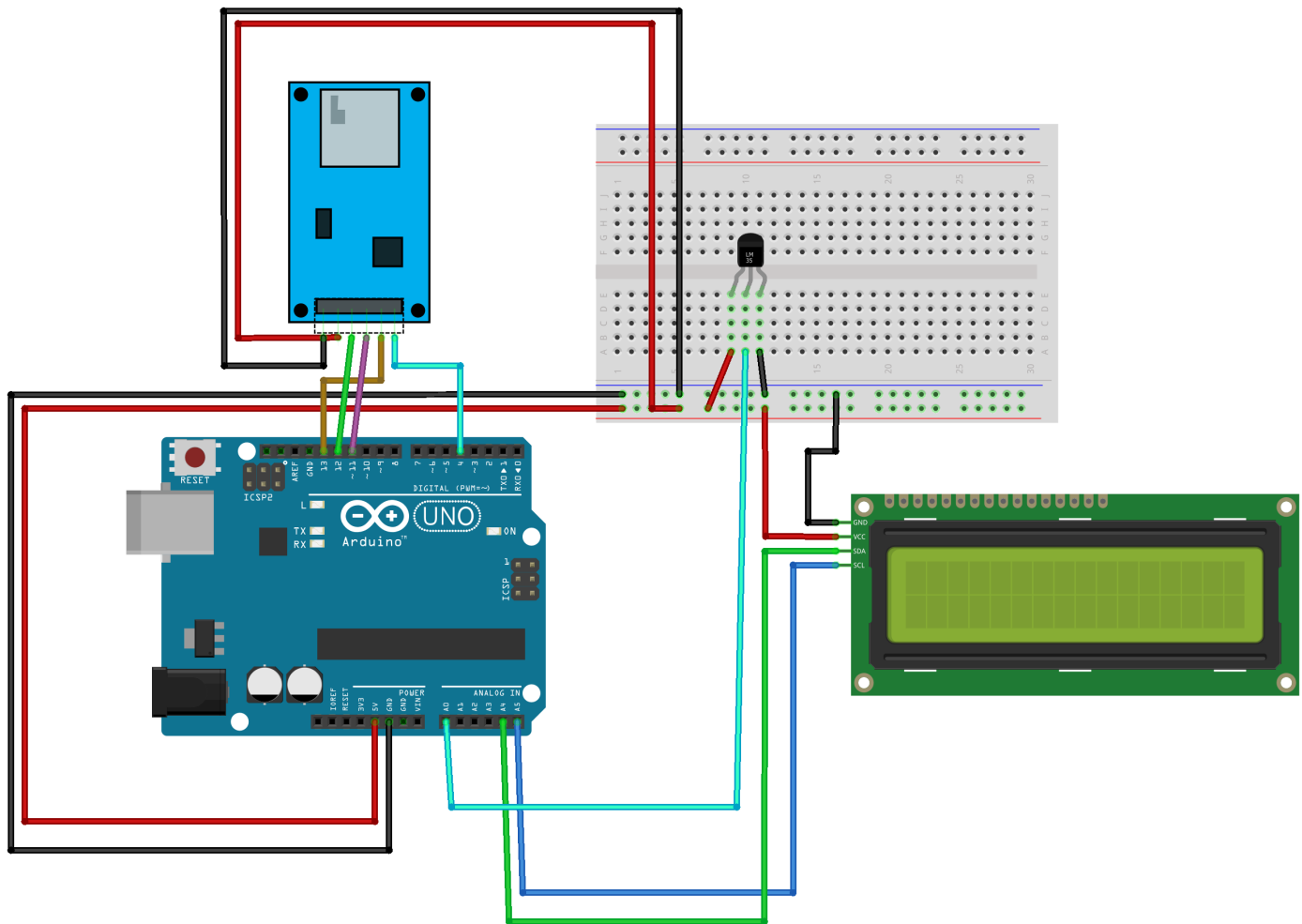
    myFile.close();
  } else {

    Serial.println("error opening test.txt");
  }
}

void loop() {
  // nothing happens after setup
}
```

Circuit 2

This next circuit we are going to build will collect temperature data for us at a set interval of time (roughly 10 seconds). This circuit is great when you need to step away but need continuous data collection. You will need the TMP36, LCD (with I2C), and the data logger. For most of the circuit, you will recognize as it is the same temperature sensor from the previous level.



fritzing

Once you have the circuit built, we can move on to the code. You will notice that the code is starting to get pretty complex. I will break down sections of the code that are new to this project. The code will be on the next 2 pages.

STEM Moment—Science and Technology

This is a great time to teach about Thermoelectric materials. Such things to cover are the Peltier effect, the Thomson effect, and the Seebeck effect.

All the libraries we will be using in this code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>
```

Setup of the LCD object

```
LiquidCrystal_I2C lcd(0x27,16,2);
```

```
float voltage = 0;
float degreesC = 0;
float degreesF = 0;
const int chipSelect = 4;
```

Declaring variables for the code

```
void setup() {
```

```
  lcd.init();
  lcd.backlight();
```

Start up the LCD and turn on the LED back light

```
  if (!SD.begin(chipSelect)) {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("card initialized.");
  delay(5000);
  lcd.clear();
}
```

This is a check of the SD card. If the card is good and ready, you will get a card-initialized message. But if the card is missing or bad you will get a card failed, or not present message, and the program stops until the error is fixed.

```
void loop() {
```

```
    voltage = analogRead(A0) * 0.004882813;  
    degreesC = (voltage - 0.5) * 100.0;  
    degreesF = degreesC * (9.0 / 5.0) + 32.0;
```

Converting the voltage to useful values

```
    lcd.clear();
```

```
    lcd.setCursor(0, 0);  
    lcd.print("Degrees C: ");  
    lcd.print(degreesC);
```

Display the temperature reading

```
    lcd.setCursor(0, 1);  
    lcd.print("Degrees F: ");  
    lcd.print(degreesF);
```

Small delay for read outs

```
    delay(5000);
```

```
File dataFile = SD.open("datalog.txt", FILE_WRITE);
```

```
//if the file is available, write to it:
```

```
if (dataFile) {  
    dataFile.print("Degrees C: ");  
    dataFile.print(degreesC);  
    dataFile.print(",");  
    dataFile.print("Degrees F: ");  
    dataFile.println(degreesF);  
    dataFile.close();  
}
```

This block of codes is the saving to the SD card. Notice how the print commands work just like the serial port.

```
lcd.clear();  
lcd.setCursor(0,0);  
lcd.print("info saved");  
delay(5000);
```

Notification of saved information and delay for 5 seconds. Each reading should be roughly 10 to 11 seconds before the next reading.

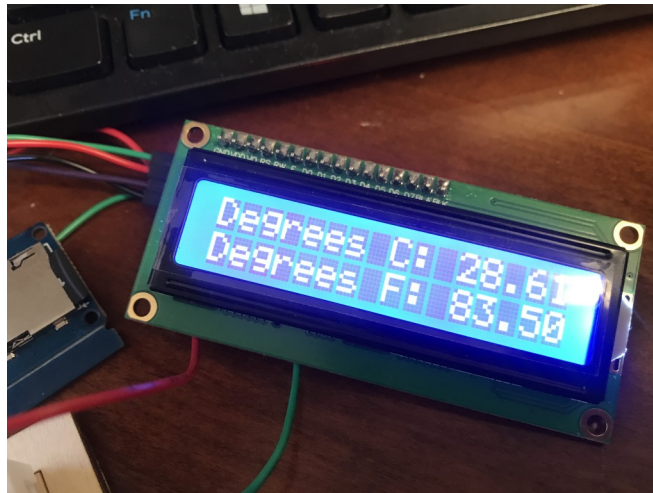
```
    }  
    // if the file isn't open, pop up an error:  
    else {  
        lcd.clear();  
        lcd.setCursor(0,0);  
        lcd.print("error SD");  
        delay(5000);  
    }  
    delay(1000);  
}
```

If the SD has issues this will give us a error.

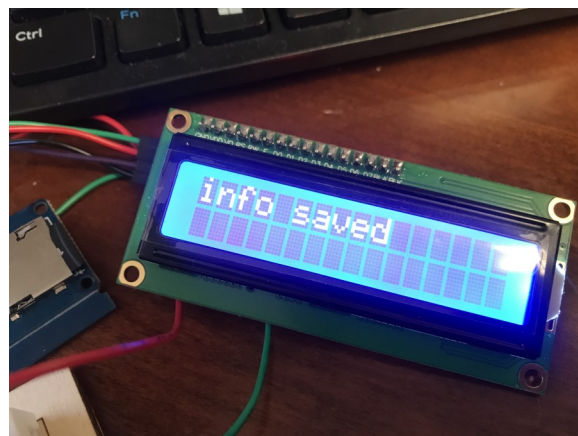
When you have the code ready and uploaded, the circuit will start working right away. You should see the circuit go through its setup code. If everything is ready you should see something like in the image below.



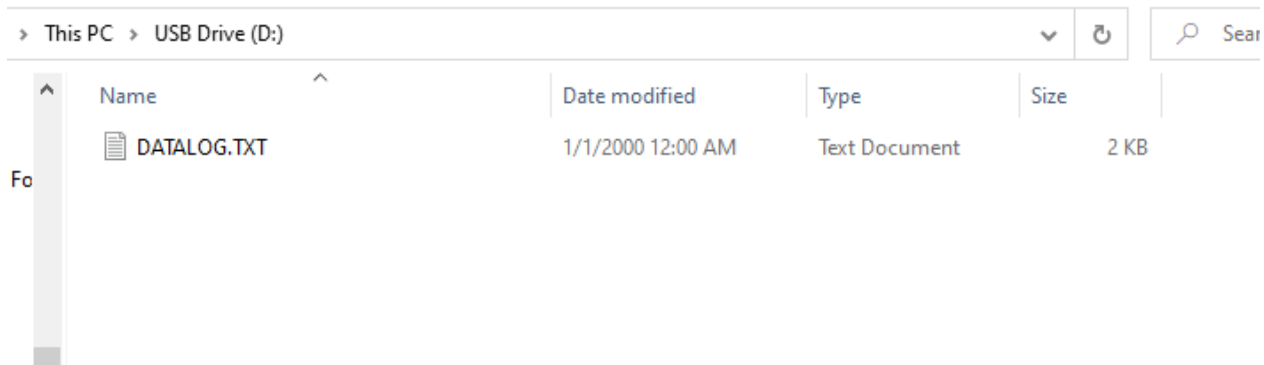
After the setup, the circuit will take a temperature reading and will display it on the LCD.



The circuit will save the data to a SD card and you will get a message that the information was successfully saved.



After we wait for awhile we can turn off the circuit and pull the SD card. Using the USD SD card reader we can plug in to the pc and open up the drive. You should see a data log file as a .txt file. We can open it and see our data.



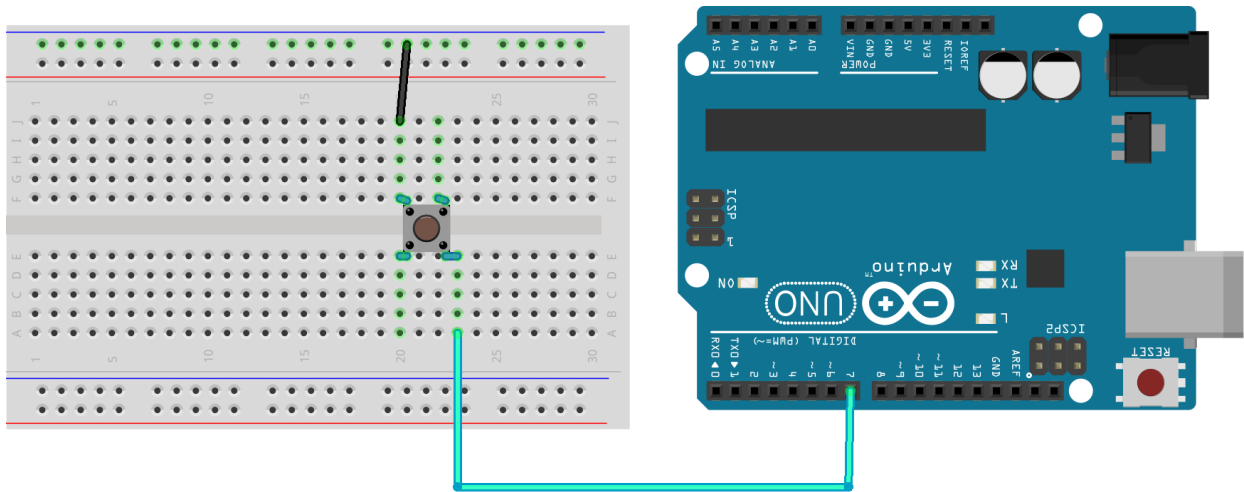
Here is an example of the code in the Notepad app. You can import this info into Excel to better analyze the date.

```
Degrees C: 32.52,Degrees F: 90.54
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 31.05,Degrees F: 87.90
Degrees C: 31.05,Degrees F: 87.90
Degrees C: 31.05,Degrees F: 87.90
Degrees C: 31.05,Degrees F: 87.90
Degrees C: 30.57,Degrees F: 87.02
Degrees C: 30.57,Degrees F: 87.02
Degrees C: 30.57,Degrees F: 87.02
Degrees C: 30.08,Degrees F: 86.14
Degrees C: 30.08,Degrees F: 86.14
Degrees C: 30.08,Degrees F: 86.14
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
```

What are some downfalls of the code we created? How can we improve this code to get us more reliable data?

Adding to the Circuit

Now that we have created a data logger, let us create a circuit that will only collect data when we want it to by pushing a button. We can keep everything from the previous circuit connected. We will just add the button to the design. See image below for button hookup.



fritzing

This setup will make the Arduino sense a button push. When the button is not being pushed it sits in the "HIGH" status. When pushed current runs through the onboard resistor and sets the status as "LOW". We will use that to trigger the saving of our data. Here is the new code for our circuit.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>

LiquidCrystal_I2C lcd(0x27,16,2);

float voltage = 0;
float degreesC = 0;
float degreesF = 0;
const int chipSelect = 4;

int run;
int buttonPin;
```

New variables declared. One for the button and the other for the run command. Run turns orange here because it is used as a keyword in one of the libraries we are using. We are using it as a variable here.

```
void setup() {  
  
  run = 0; //starts stopped  
  buttonPin = 7; //whatever pin your button is
```

Setting values to our variables

```
  pinMode(buttonPin, INPUT_PULLUP);  
  
  lcd.init();  
  lcd.backlight();  
  
  if (!SD.begin(chipSelect)) {  
    lcd.clear();  
    lcd.setCursor(0,0);  
    lcd.print("Card failed, or not present");  
    // don't do anything more:  
    while (1);  
  }  
  lcd.clear();  
  lcd.setCursor(0,0);  
  lcd.print("card initialized.");  
  delay(5000);  
  lcd.clear();  
}
```

Telling the Arduino that we are using pin 7 as an input and also using the onboard resistor too.

```
void loop() {  
  
  voltage = analogRead(A0) * 0.004882813;  
  degreesC = (voltage - 0.5) * 100.0;  
  degreesF = degreesC * (9.0 / 5.0) + 32.0;  
  
  lcd.clear();  
  
  lcd.setCursor(0, 0);  
  lcd.print("Degrees C: ");  
  lcd.print(degreesC);  
  
  lcd.setCursor(0, 1);  
  lcd.print("Degrees F: ");  
  lcd.print(degreesF);  
  
  delay(50);
```

```

if(digitalRead(buttonPin) == LOW)
{
File dataFile = SD.open("datalog.txt", FILE_WRITE);

//if the file is available, write to it:
if (dataFile) {
  dataFile.print("Degrees C: ");
  dataFile.print(degreesC);
  dataFile.print(",");
  dataFile.print("Degrees F: ");
  dataFile.println(degreesF);
  dataFile.close();
lcd.clear();
lcd.setCursor(0,0);
lcd.print("info saved");
delay(5000);

}
// if the file isn't open, pop up an error:
else {
lcd.clear();
lcd.setCursor(0,0);
lcd.print("error SD");
delay(5000);
}}
  delay(1000);
}

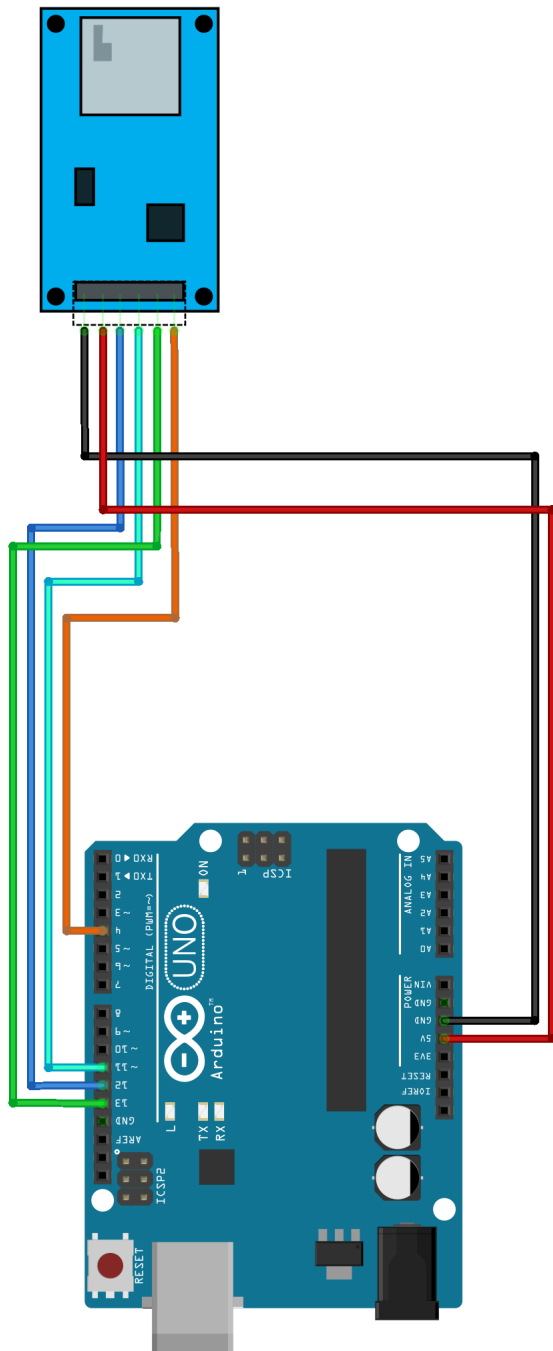
```

When the button is pushed, it will run the code to save the data. Otherwise, it will continue on looping until we push the button.

After saving data, the loop will continue until pushed again.

Once the code is ready, upload to the Arduino board. When the LCD displays the temperature readings, click the button. Should see "info saved" message. Do this a few times, then power down the Arduino and check the data on the SD via the USB card reader.

Level Four Student Printout card Circuit 1 and Code 1



fritzing

```
#include <SPI.h>
#include <SD.h>

File myFile;

void setup() {

  Serial.begin(9600);
  while (!Serial) {
    ;
  }

  Serial.print("Initializing SD card...");

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    while (1);
  }
  Serial.println("initialization done.");

  myFile = SD.open("test.txt", FILE_WRITE);

  if (myFile) {
    Serial.print("Writing to test.txt...");
    myFile.println("testing 1, 2, 3.");

    myFile.close();
    Serial.println("done.");
  } else {

    Serial.println("error opening test.txt");
  }

  myFile = SD.open("test.txt");
  if (myFile) {
    Serial.println("test.txt:");

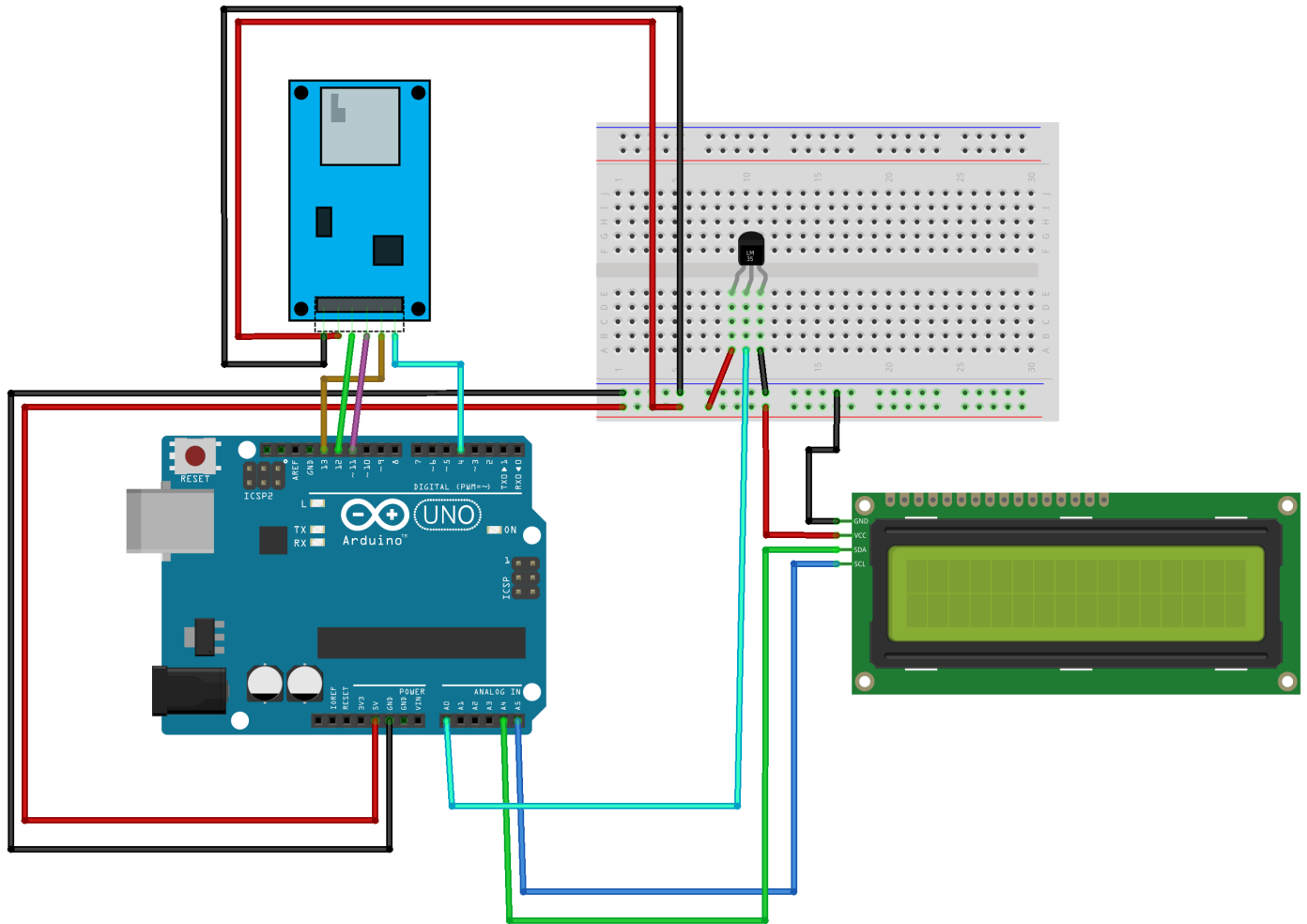
    while (myFile.available()) {
      Serial.write(myFile.read());
    }

    myFile.close();
  } else {

    Serial.println("error opening test.txt");
  }
}

void loop() {
  // nothing happens after setup
}
```

Level Four Student Printout card Circuit 2



fritzing

Level Four Student Printout card Code 2

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

float voltage = 0;
float degreesC = 0;
float degreesF = 0;
const int chipSelect = 4;

void setup() {
  lcd.init();
  lcd.backlight();

  if (!SD.begin(chipSelect)) {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("card initialized.");
  delay(5000);
  lcd.clear();
}
```

```
void loop() {
  voltage = analogRead(A0) * 0.004882813;
  degreesC = (voltage - 0.5) * 100.0;
  degreesF = degreesC * (9.0 / 5.0) + 32.0;

  lcd.clear();

  lcd.setCursor(0, 0);
  lcd.print("Degrees C: ");
  lcd.print(degreesC);

  lcd.setCursor(0, 1);
  lcd.print("Degrees F: ");
  lcd.print(degreesF);

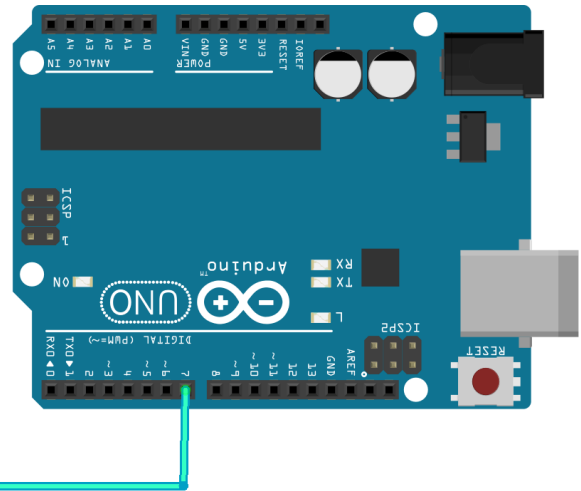
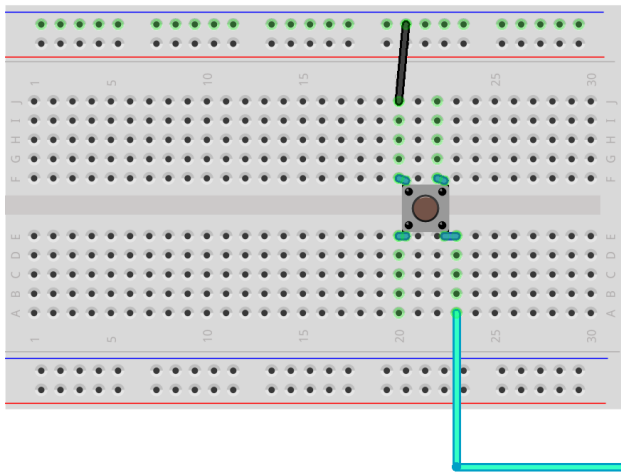
  delay(5000);

  File dataFile = SD.open("dataLog.txt", FILE_WRITE);

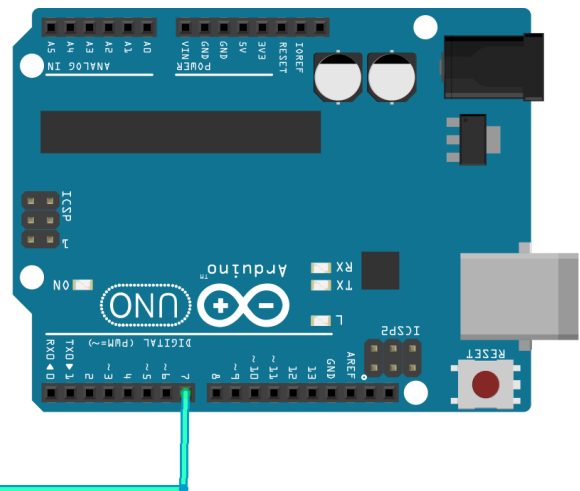
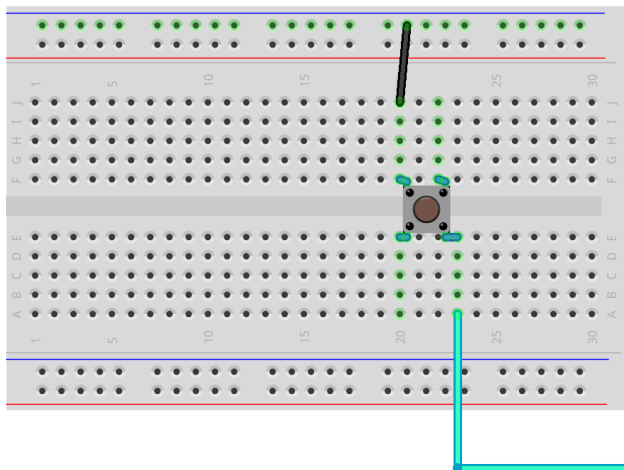
  //if the file is available, write to it:
  if (dataFile) {
    dataFile.print("Degrees C: ");
    dataFile.print(degreesC);
    dataFile.print(",");
    dataFile.print("Degrees F: ");
    dataFile.println(degreesF);
    dataFile.close();
  }
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("info saved");
  delay(5000);

  }
  // if the file isn't open, pop up an error:
  else {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("error SD");
    delay(5000);
  }
  delay(1000);
}
```

Level Four Student Printout card add on to previous circuit



fritzing



fritzing

Level Four Student Printout card add on to previous circuit

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>

LiquidCrystal_I2C lcd(0x27,16,2);

float voltage = 0;
float degreesC = 0;
float degreesF = 0;
const int chipSelect = 4;

int run;
int buttonPin;

void setup() {

  run = 0; //starts stopped
  buttonPin = 7; //whatever pin your button is

  pinMode(buttonPin, INPUT_PULLUP);

  lcd.init();
  lcd.backlight();

  if (!SD.begin(chipSelect)) {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("card initialized.");
  delay(5000);
  lcd.clear();
}
```

```
void loop() {

  voltage = analogRead(A0) * 0.004882813;
  degreesC = (voltage - 0.5) * 100.0;
  degreesF = degreesC * (9.0 / 5.0) + 32.0;

  lcd.clear();

  lcd.setCursor(0, 0);
  lcd.print("Degrees C: ");
  lcd.print(degreesC);

  lcd.setCursor(0, 1);
  lcd.print("Degrees F: ");
  lcd.print(degreesF);

  delay(50);

  if(digitalRead(buttonPin) == LOW)
  {
    File dataFile = SD.open("datalog.txt", FILE_WRITE);

    //if the file is available, write to it:
    if (dataFile) {
      dataFile.print("Degrees C: ");
      dataFile.print(degreesC);
      dataFile.print(",");
      dataFile.print("Degrees F: ");
      dataFile.println(degreesF);
      dataFile.close();
    }
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("info saved");
    delay(5000);

  }
  // if the file isn't open, pop up an error:
  else {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("error SD");
    delay(5000);
  }
  delay(1000);
}
```

Level Summary:

This level is all about GPS modules. Topics covered in this lesson are GPS modules, NMEA data, and GPS libraries.

Objectives:

Students will be able to:

- Display latitude and longitude from a current location using a GPS module.

Materials: (ideally you would need enough for each student, but enough for groups of two works too)

- computer or device to run Arduino IDE (a pc is preferred)
- Development boards (Arduino Uno is what we use in this book)
- Breadboards (any size will work)
- Jumper wires (at least 6 per kit)
- GPS Modules (BN 180s are in the kits, but anything higher will work too)



More Resources Here:

www.montana.edu/smrc/sensingforscience/

Estimated Time needed:

2 x 45 periods (20 minutes for lecture 25 for building circuits)

Preparation:

- Test Arduino IDE on all the PCs
- Check on uno boards and clear them with barebone code
- Check that your uno boards are similar to the ones in this project. Some boards moved their pc interrupt pins. Check your manufacturer's data sheets.
- Go through this level before teaching it, get used to how everything works
- Test GPS modules (sometimes they don't work)

Code Words to know:

- string—this is a collection of characters and data (which can be very large), that can be added to.
- readstringuntil—part of the software serial library, reads a string, usually to be printed later
- static - used when setting variables that are used from other classes (code outside yours)
- Uint32_t - a numeric type that guarantees 32 bits

GPS modules

GPS modules are devices available to us that include a GPS antenna and receiver. This device is able to receive data from the GPS satellites and calculate the current location of the module. The GPS module then sends us NMEA data to our Arduino board.

We will be using the Beitina BN-180 GPS. This is a small yet accurate module. It is very easy to hook up and run on our Arduino boards.

Explore More

This is a great opportunity to explore NMEA code and how to understand what the satellites are sending to our GPS. aprs.gids.nl/nmea/



The hookup for our GPS module is very simple. The VCC connects to 5v on the Arduino. TX connects to pin 6 on the Arduino. RX will go to pin 5 on the Arduino. GND will go to GND on the Arduino.



Once the circuit is built, it is time to test it. The code we are using is simple and will use a new prebuilt library called SoftwareSerial.

This library allows us to create new serial ports using pins. Since we will need the serial port to read the data, being able to create another port to read the GPS is essential. We will also use the string function that will collect many characters of data.

```
#include <SoftwareSerial.h>
```

Including the SoftwareSerial library

```
static const int RXPin = 6, TXPin = 5;  
static const uint32_t GPSBaud = 9600;  
SoftwareSerial ss(RXPin, TXPin);
```

Declaring some variables for the GPS. Most GPS units use a Baud rate of 9600. If you get a different module you might have to change this.

```
void setup() {
```

```
ss.begin(GPSBaud);
```

Here we are creating an object which is our new serial port for the GPS.

```
Serial.begin(115200);
```

Starts the GPS unit

```
}
```

```
void loop() {
```

```
String msg = ss.readStringUntil('\r');
```

Start serial monitor so we can see our data. We are going to run it at 115200 to transfer all the data in time

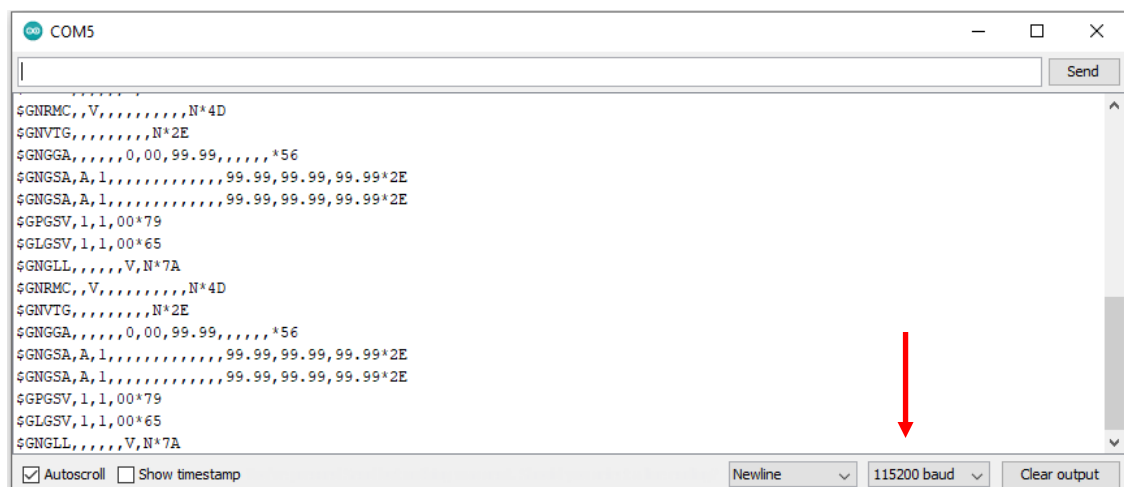
```
Serial.print(msg);
```

This code will read and collect the characters of data from the satellites until it gets to a return.

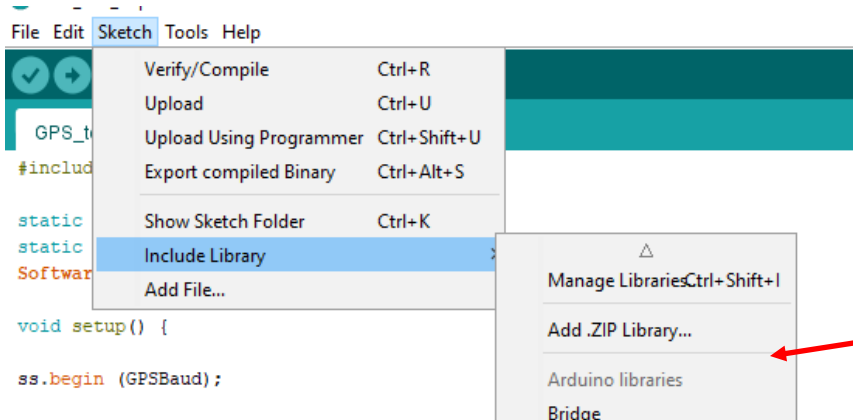
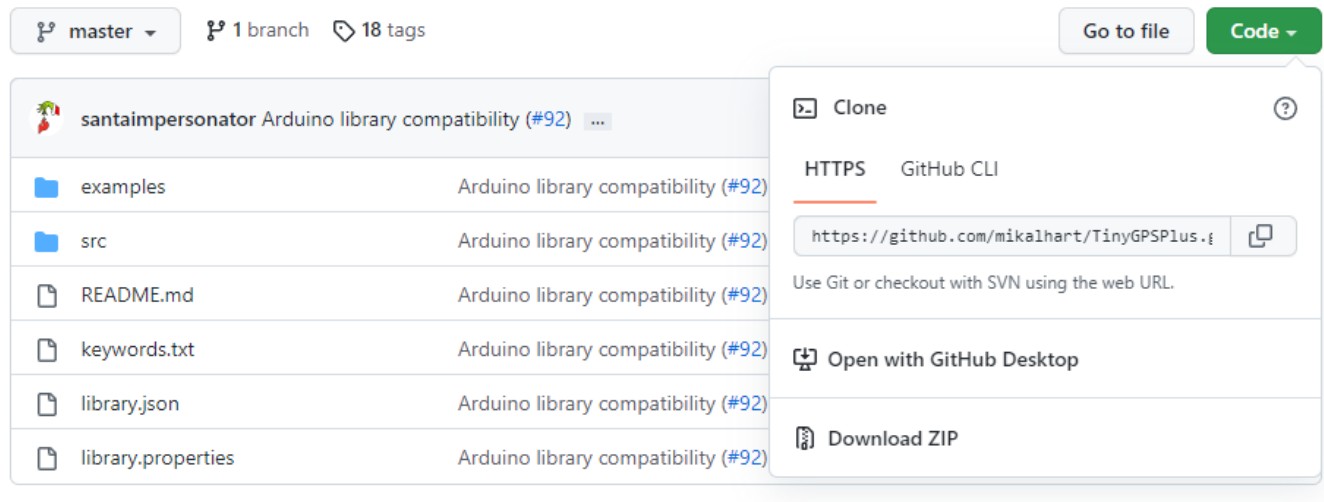
Prints the string data to the serial port

```
}
```

Upload the code and open the serial monitor. Change the baud rate to 115200 and watch the data come in. You might get trash at first, but after a while, if you have sight satellites will get useful data. Buildings and trees can block satellites. The lines of code you are seeing in the monitor are the NMEA data.

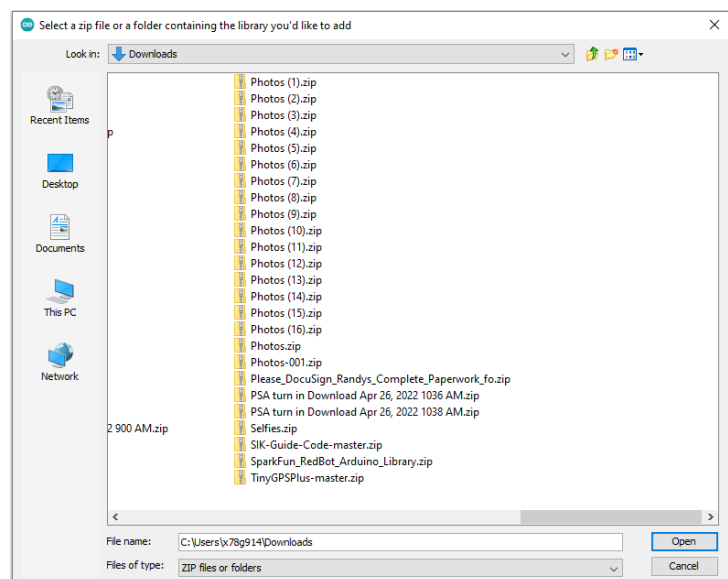


Now that we know the GPS module works, let us turn the NMEA data into something we can easily understand. We are going to use a custom-made library that is not available for installation via the Arduino library. The library we need can be found here at the following link: <https://github.com/mikalhart/TinyGPSPlus>. On the webpage is a green code button. Click it and click "download ZIP".



When the download is finished, open Arduino and go to sketch->include library->add .ZIP library

Find the downloaded zip file and open it. The TinyGPSPlus library is now available for us to use in our code.



We are now ready for start writing the code.

```
#include <TinyGPS++.h>
#include <TinyGPSPlus.h>
#include <SoftwareSerial.h>
```

The libraries we will need.

```
String gpstext;
static const int RXPin = 6, TXPin = 5;
static const uint32_t GPSBaud = 9600;
SoftwareSerial ss(RXPin, TXPin);
TinyGPSPlus gps;
```

Declaring variables

```
void setup() {
  ss.begin(GPSBaud);
  Serial.begin(115200);
}
```

Creating two objects: SS and GPS

Our setup code to start the serial monitor and GPS

```
void loop() {
  while (ss.available() > 0)
    if (gps.encode(ss.read()))

    // See if we have a complete GPS data string
    if (displayInfo() != "0")
    {

      // Get GPS string
      gpstext = displayInfo();
      Serial.println(gpstext);
    }
}
```

Our loop code is fairly simple. When the SS serial is available, read the gps and get data from the displayInfo() function and print the gpstext from displayInfo(). What is the displayInfo?

```

String displayInfo()
{
    // Define empty string to hold output
    gps.encode(ss.read());
    String gpsdata = "";

    // Get latitude and longitude
    if (gps.location.isValid())
    {
        gpsdata = String(gps.location.lat(), 6);
        gpsdata += ("," );
        gpsdata += String(gps.location.lng(), 6);
        gpsdata += ("," );
    }
    else
    {
        return "0";
    }

    // Get Date
    if (gps.date.isValid())
    {
        gpsdata += String(gps.date.year());
        gpsdata += ("-");
        if (gps.date.month() < 10) gpsdata += ("0");
        gpsdata += String(gps.date.month());
        gpsdata += ("-");
        if (gps.date.day() < 10) gpsdata += ("0");
        gpsdata += String(gps.date.day());
    }
    else
    {
        return "0";
    }

    // Space between date and time
    gpsdata += (" ");

    // Get time
    if (gps.time.isValid())
    {
        if (gps.time.hour() < 10) gpsdata += ("0");
        gpsdata += String(gps.time.hour());
        gpsdata += (":");
        if (gps.time.minute() < 10) gpsdata += ("0");
        gpsdata += String(gps.time.minute());
        gpsdata += (":");
        if (gps.time.second() < 10) gpsdata += ("0");
        gpsdata += String(gps.time.second());
    }
    else
    {
        return "0";
    }

    // Return completed string
    return gpsdata;
}

```

This is the displayInfo function. It is outside of the VOID setup and loop. Sense we desire this function to give us information we will use String (data with many characters) instead of VOID (nothing returned).

Read the SS serial data and encode it. Start adding to the string

If the GPS gives a valid location, then add the latitude and longitude to the string up to 6 characters each. Otherwise, return a 0.

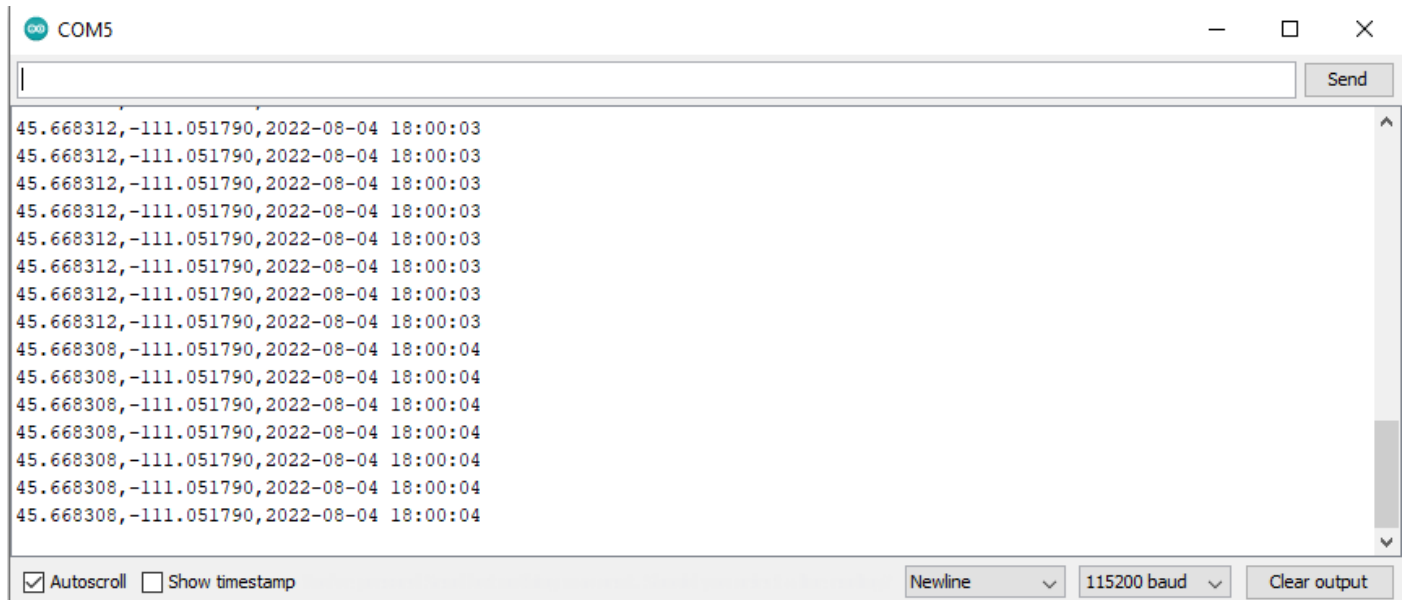
If we have a valid date from the GPS, add it to the string. This date will be in GST. Otherwise return a 0

IF we have a valid time from the GPS, add it to the string. This time will be in GST. Otherwise return a 0.

If the GPS is not reading send a 0

Return the String back to the loop.

Upload the code and see if it works. When the upload is complete open the serial monitor and see what the data looks like. It might take a moment for data to start coming in. If after a while you still not getting a location try moving outside (laptops make this easy). This data can be copied to the notepad and saved. If you import this data into Excel or Google Maps you can see how accurate the module is.

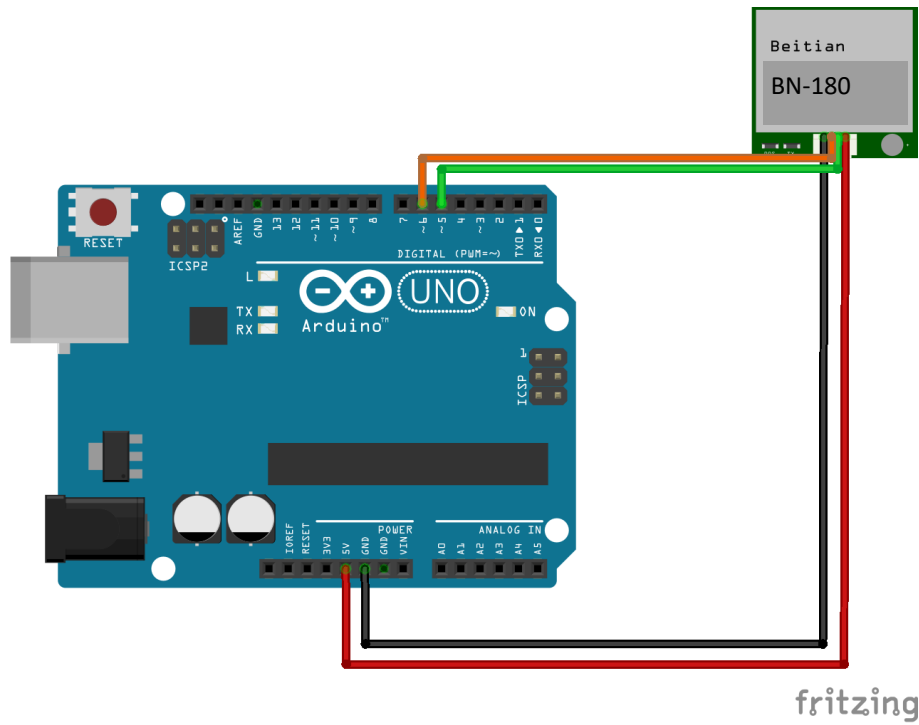


Sources:

Building a GPS System - SparkFun Electronics. (n.d.). [Www.sparkfun.com. https://www.sparkfun.com/gps](https://www.sparkfun.com/gps)

Workshop, DroneBot. "Using GPS Modules with Arduino & Raspberry Pi." DroneBot Workshop, 26 June 2021, dronebotworkshop.com/using-gps-modules/. Accessed 5 Aug. 2022.

Level Five Student Printout card Circuit 1 and Code 1



```
#include <SoftwareSerial.h>

static const int RXPin = 6, TXPin = 5;
static const uint32_t GPSBaud = 9600;
SoftwareSerial ss(RXPin, TXPin);

void setup() {
  ss.begin(GPSBaud);
  Serial.begin(115200);
}

void loop() {
  String msg = ss.readStringUntil('\r');
  Serial.print(msg);
}
```

```

#include <TinyGPS++.h>
#include <TinyGPSPPlus.h>
#include <SoftwareSerial.h>

String gpstext;

static const int RXPin = 6, TXPin = 5;
static const uint32_t GPSBaud = 9600;
SoftwareSerial ss(RXPin, TXPin);
TinyGPSPPlus gps;

void setup() {

  ss.begin(GPSBaud);

  Serial.begin(115200);

}

void loop() {
  while (ss.available() > 0)
    if (gps.encode(ss.read()))
      // See if we have a complete GPS data
      if (displayInfo() != "0")
      {
        // Get GPS string
        gpstext = displayInfo();
        Serial.println(gpstext);
      }
}

```

```

String displayInfo()
{
  // Define empty string to hold output
  gps.encode(ss.read());
  String gpsdata = "";

  // Get latitude and longitude
  if (gps.location.isValid())
  {
    gpsdata = String(gps.location.lat(), 6);
    gpsdata += (" ");
    gpsdata += String(gps.location.lng(), 6);
    gpsdata += (" ");

  }
  else
  {
    return "0";
  }

  // Get Date
  if (gps.date.isValid())
  {
    gpsdata += String(gps.date.year());
    gpsdata += ("-");
    if (gps.date.month() < 10) gpsdata += ("0");
    gpsdata += String(gps.date.month());
    gpsdata += ("-");
    if (gps.date.day() < 10) gpsdata += ("0");
    gpsdata += String(gps.date.day());

  }
  else
  {
    return "0";
  }

  // Space between date and time
  gpsdata += (" ");

  // Get time
  if (gps.time.isValid())
  {
    if (gps.time.hour() < 10) gpsdata += ("0");
    gpsdata += String(gps.time.hour());
    gpsdata += (":");
    if (gps.time.minute() < 10) gpsdata += ("0");
    gpsdata += String(gps.time.minute());
    gpsdata += (":");
    if (gps.time.second() < 10) gpsdata += ("0");
    gpsdata += String(gps.time.second());

  }
  else
  {
    return "0";
  }

  // Return completed string
  return gpsdata;
}

```

Sensing for Science

Level 6

Level Summary:

This level takes all the content we have learned and uses it to help us collect data. The data we are collecting will be total dissolved solids (TDS) in water. TDS is a great way to see if there is a potential issue in water quality.

Objectives:

Students will be able to:

- Develop an electronic device that will collect and save data on water quality for analysis later.

Materials: (ideally you would need enough for each student, but enough for groups of two works too)

- computer or device to run Arduino IDE (a pc is preferred)
- Development boards (Arduino Uno is what we use in this book)
- Breadboards (any size will work)
- Jumper wires (at least 6 per kit)
- GPS Modules (BN 180s are in the kits, but anything higher will work too)
- LCD with I2C
- Button
- TDS sensor module
- Potentiometer or tmp36 (depends on if you want temperature to be sensed or inputted)
- SD card reader module (card and reader too)
- 9v batter with connector



More Resources Here:

www.montana.edu/smrc/sensingforscience/

Estimated Time needed:

2 x 45 periods (45 minutes for building circuits, 45 for collecting and analyzing data)

Preparation:

- Test Arduino IDE on all the PCs
- Check on uno boards and clear them with barebone code
- Check that your uno boards are similar to the ones in this project. Some boards moved their pc interrupt pins. Check your manufacturer's data sheets.
- Go through this level before teaching it, get used to how everything works
- Test TDS modules (sometimes they don't work)
- This project can go in any direction you want to go, feel free to change up sensors and what kind of data you want to use.
- If you are sending students to ponds or streams to collect data, make sure safety protocols are put in place for student safety.

The Final Project

At this level, we are going to create a circuit that will test the total dissolved solids (TDS) in water using the GravityTDS sensor from DFRobot. It will also collect the GPS location and save the data to an SD card. We will have an LCD display too. There is a lot going on here with hardware and software, so take time to tape wires and do some organizing. It will help with troubleshooting when out in the field. There are many ways this circuit can be constructed. I will show you the way I did it, but you can build it in the way you think is good.

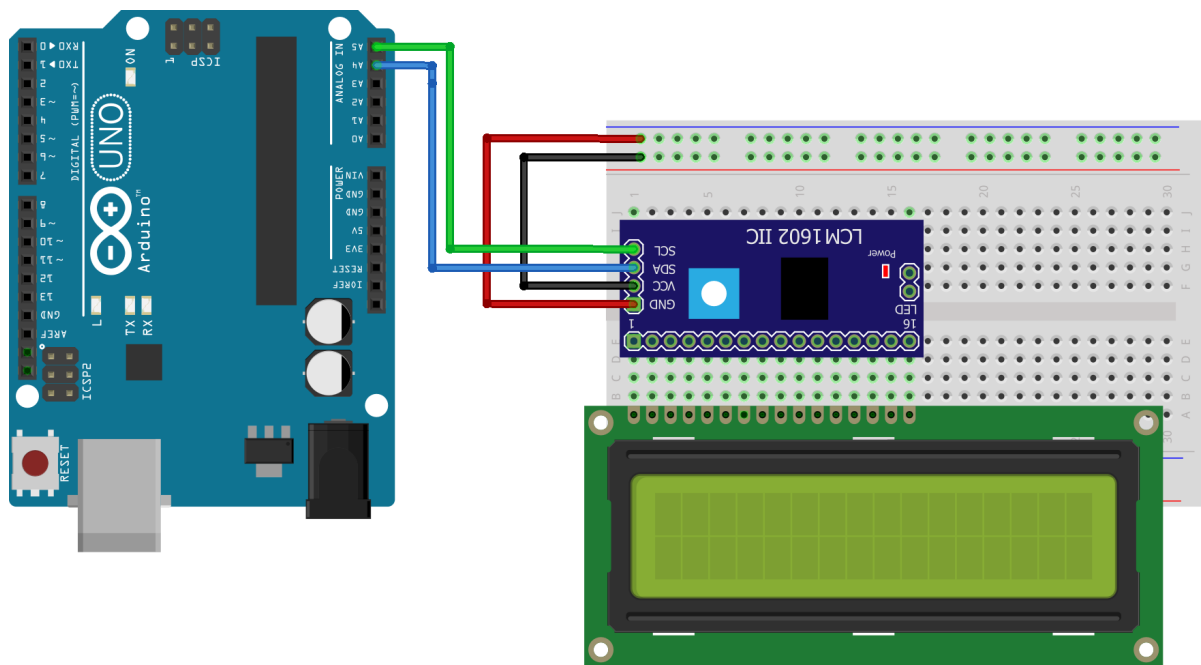
NSF EPSCoR CREWS

See how scientists are using sensors to monitor the Upper Clark Fork River for sediment metals. They are using similar sensors as the TDS sensor we are using at this level.



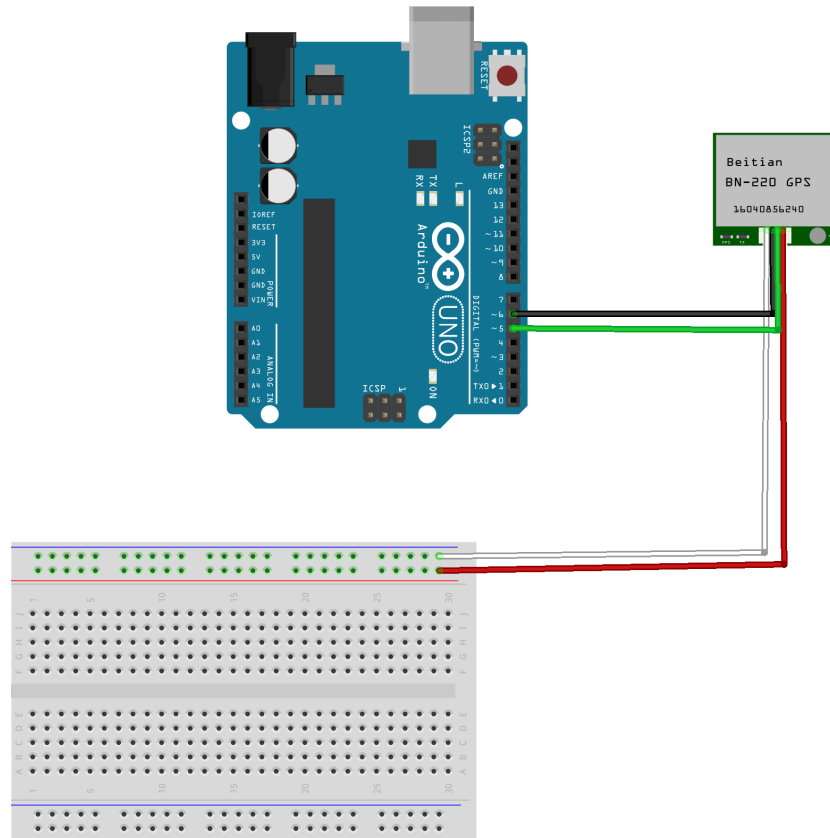
Here is how I wired up this circuit. I will do a schematic for each device connected to the Arduino board. Take note, I will use the power rails on the breadboard for all 5v and GND connections.

LCD hook with I2C adaptor.



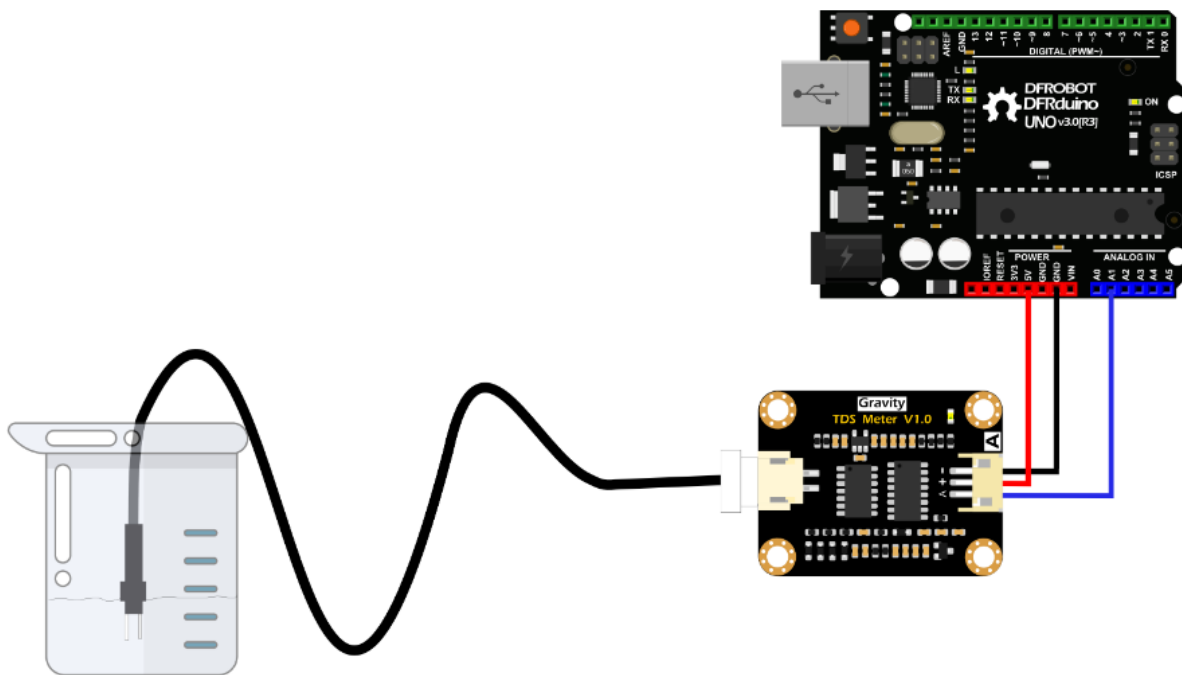
fritzing

GPS Hook up (Pin 1 = GND, Pin 2 = 6, Pin 3 = 7, Pin 4 = 5V)

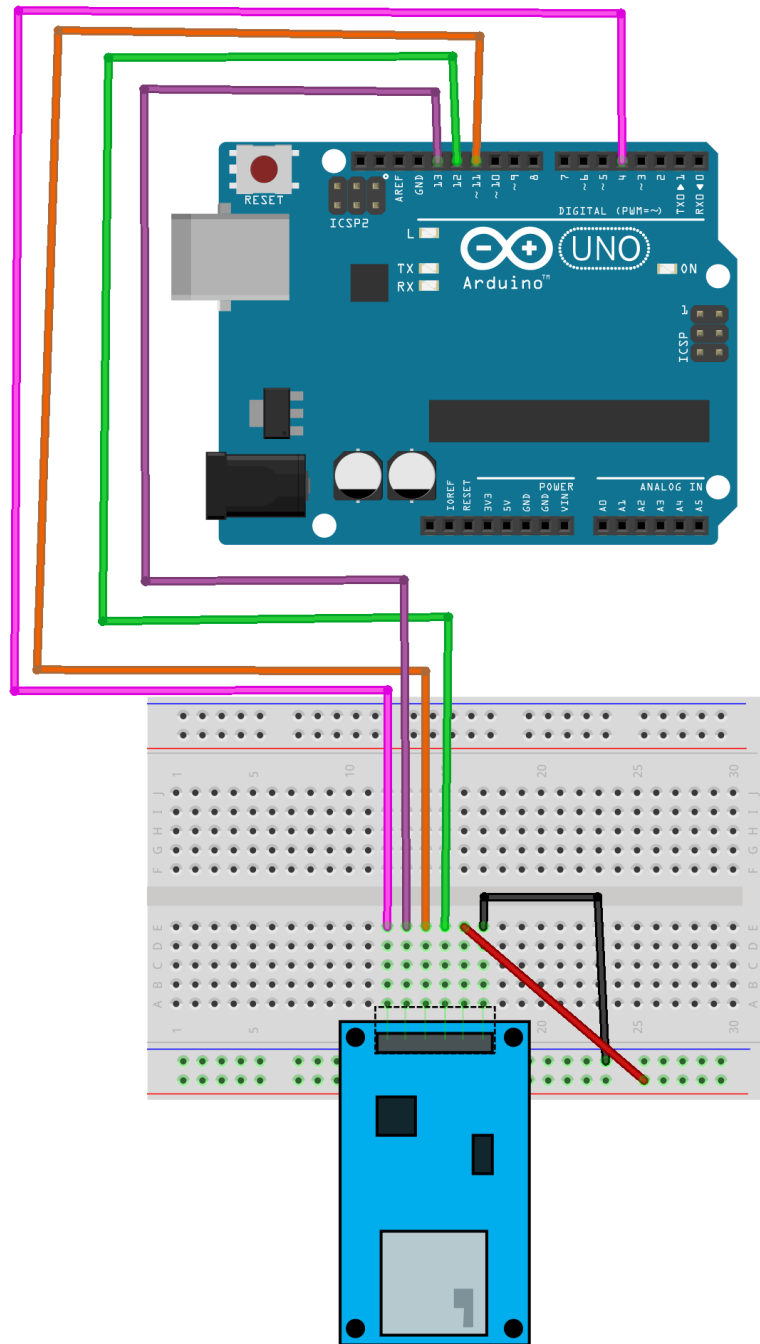


fritzing

TDS hook up (Image is from DFrobot's wiki page)



SD car reader Hook Up



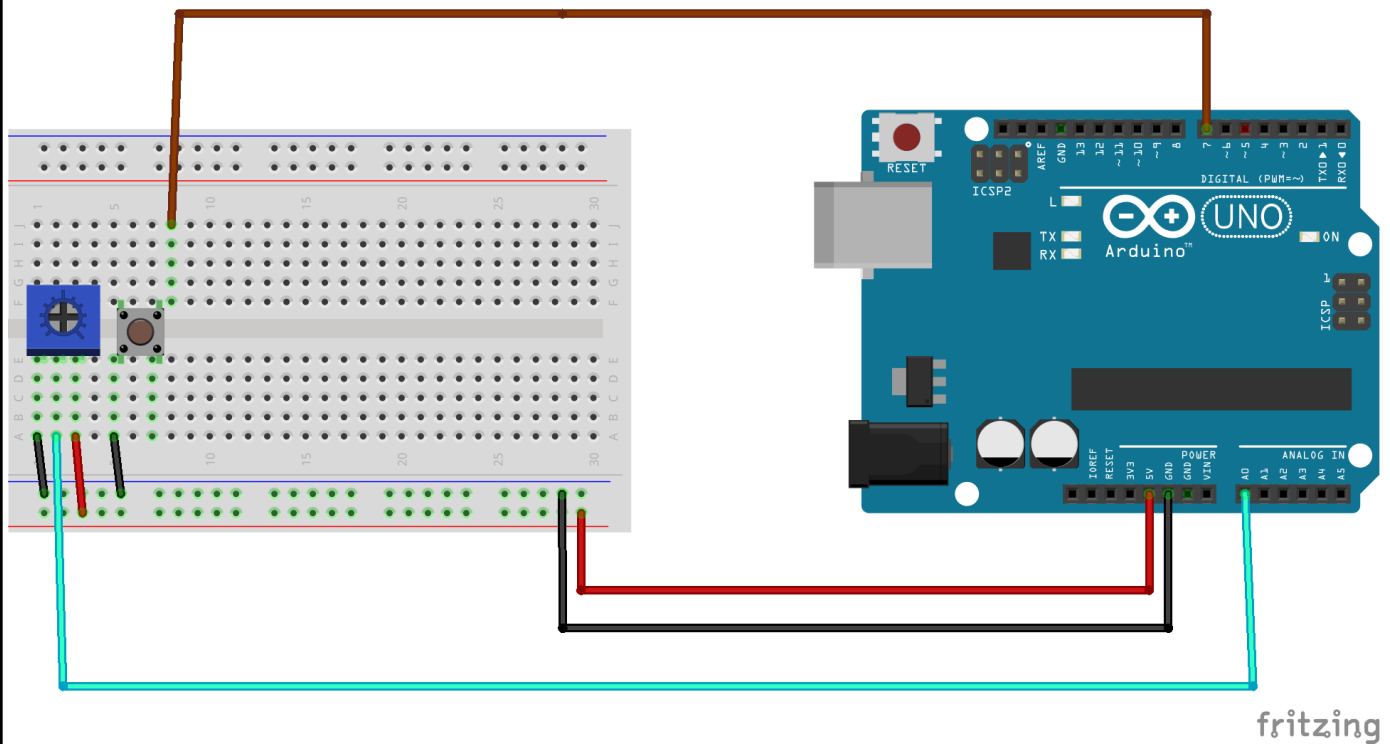
fritzing

NSF EPSCoR CREWS

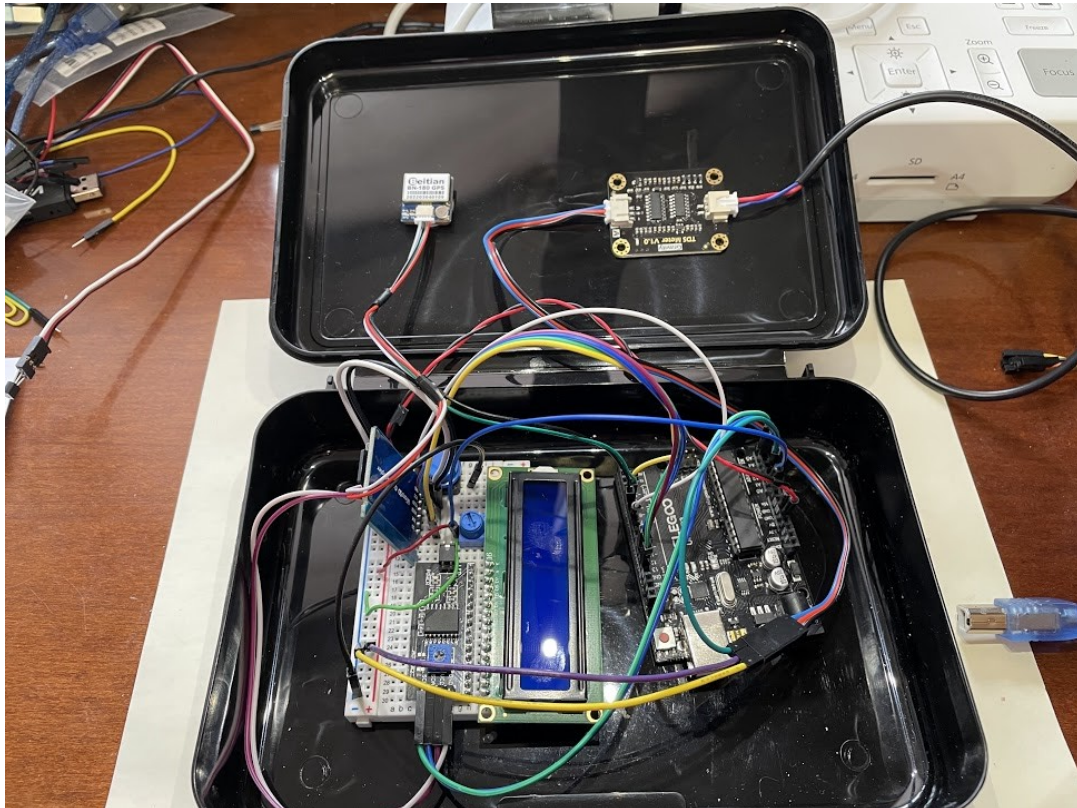
See how scientists are using sensor to study the nitrate and herbicide levels in the ground water of the Judith River Basin. Their focus is to develop new technologies for removing nitrates and other contaminants. They are using similar sensors as the TDS sensor we are using in this level. www.mtnsepsc.org/projects/crews/research/ucfr



Button and Potentiometer hook up (note you can replace the potentiometer with a temperature sensor if you want, just make sure it is waterproof and the wires are long enough.)



Here is what mine looks like as a prototype (yes, it's a mess). I used tape to organize wires and tape down different modules. Once I know it works, I'll organize everything to make it easier to carry around.



It is now time to code this circuit. You can copy it from the worksheets or download it from the Sensing for Science website. You will need the TDS library from DFRobot which can be found here:

https://wiki.dfrobot.com/Gravity__Analog_TDS_Sensor___Meter_For_Arduino_SKU__SEN0244

```
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include <EEPROM.h>
#include "GravityTDS.h"

// String to hold GPS data
String gpstext;

// GPS Connections
static const int RXPin = 6, TXPin = 5;

// GPS Baud rate (change if required)
static const uint32_t GPSBaud = 9600;

// TinyGPS++ object
TinyGPSPlus gps;

// SoftwareSerial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);

//I2C LCD
const int chipSelect = 4;

LiquidCrystal_I2C lcd(0x27,16,2); // set the LCD address to 0x3F for a 16 chars and 2 line display

//Button Push
int run;
int buttonPin;

//TDS
#define TdsSensorPin A1
GravityTDS gravityTds;
float temperature = 25, tdsValue = 0;

//TMP
float degreesC = 0; //the temperature in Celsius, calculated from the voltage

void setup() {

// Start Serial Monitor for debugging
Serial.begin(115200);

// Start SoftwareSerial
ss.begin(GPSBaud);

//Prep for Button Push
run = 0; //starts stopped
buttonPin = 7; //whatever pin your button is plugged into
pinMode(buttonPin, INPUT_PULLUP);
pinMode(2, INPUT);

//TDS startup
gravityTds.setPin(TdsSensorPin);
gravityTds.setAref(5.0); //reference voltage on ADC, default 5.0V on Arduino UNO
gravityTds.setAdcRange(1024); //1024 for 10bit ADC; 4096 for 12bit ADC
gravityTds.begin(); //initialization
```



Link to this code. Copy and paste into
Arduino IDE

```

//LCD startup and SD Check
lcd.init();
lcd.clear();
lcd.backlight();      // Make sure backlight is on

if (!SD.begin(chipSelect)) {
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Card failed, or not present");
  // don't do anything more:
  while (1);
}

lcd.clear();
lcd.setCursor(0,0);
lcd.print("card initialized.");
delay(1000);
lcd.clear();
lcd.print("Finding Sats");
}

void loop() {

while (ss.available() > 0)
  if (gps.encode(ss.read()))

    // See if we have a complete GPS data string
    if (displayInfo() != "0")
    {

      // Get GPS string
      gpstext = displayInfo();
      Serial.println(gpstext);

degreesC = map(analogRead(A0),0,1023,0,100);

gravityTds.setTemperature(degreesC); // set the temperature and execute temperature compensation
gravityTds.update(); //sample and calculate
tdsValue = gravityTds.getTdsValue(); // then get the value
Serial.print(tdsValue,0);
Serial.println("ppm");

  lcd.clear();
  lcd.setCursor(0, 0);           //set the cursor to the top left position
  lcd.print("Degrees C: ");      //print a label for the data
  lcd.print(degreesC);          //print the degrees Celsius

  lcd.setCursor(0, 1);          //set the cursor to the lower left position
  lcd.print(tdsValue,0);        //Print a label for the data
  lcd.print("ppm");             //print the degrees Fahrenheit

```

```

//check button pressed, if so enter program condition (inside if statement)
if(digitalRead(buttonPin) == LOW) //functions based off of button pulling input pin LOW
{

File dataFile = SD.open("datalog.txt", FILE_WRITE);

//if the file is available, write to it:
if (dataFile) {
    dataFile.print("Degrees C: ");
    dataFile.print(degreesC);
    dataFile.print(",");
    dataFile.print(tdsValue,0);
    dataFile.print("ppm");
    dataFile.print(",");
    dataFile.println(gpstext);
    dataFile.close();
}
lcd.clear();
lcd.setCursor(0,0);
lcd.print("info saved");
delay(1000);
}
// if the file isn't open, pop up an error:
else {
lcd.clear();
lcd.setCursor(0,0);
lcd.print("error SD");
delay(1000);
}}

    delay(1000);
}

String displayInfo()
{
    // Define empty string to hold output
    gps.encode(ss.read());
    String gpsdata = "";

    // Get latitude and longitude
    if (gps.location.isValid())
    {
        gpsdata = String(gps.location.lat(), 6);
        gpsdata += ("," );
        gpsdata += String(gps.location.lng(), 6);
        gpsdata += ("," );
    }
    else
    {
        return "0";
    }

    // Return completed string
    return gpsdata;
}

```

Testing the circuit

After the circuit is built and the code is uploaded, take a walk around outside and press the button to collect data points to make sure all the modules are working correctly. You should have the LCD show TDS reading in ppm and the temperature (either inputted by you or a temperature sensor). When the button is pushed the LCD should say "data saved". After walking around you can pull the SD card and check the data and make sure the GPS is reading correctly.

Calibrate the TDS

You can also calibrate the sensor the TDS sensor. To calibrate you will need a standard solution of 707 ppm. With the circuit plugged into the pc, open the serial monitor. Place the probe into the solution and wait for the reading to stabilize. Once stabilize, use the command bar inside the serial monitor and type "enter" and click send. You should see that the calibration mode has started. Next type in the command bar "cal:707" and click send. You should see a success message. More about working with the TDS sensor can be found at https://wiki.dfrobot.com/Gravity__Analog_TDS_Sensor___Meter_For_Arduino_SKU__SEN0244

Collecting data

Now that your circuit is complete and ready for data collection. Remember when working around bodies of water to be safe. Follow all safety rules when working around water.

To collect proper data, place the probe into the water and wait for the data on the LCD to level out. Once the data levels, press the button to save the data. It is good practice to carry a bottle of distilled water to clean the probe between collection areas. It is also important to not stir up the water, it will affect the readings. When working with lakes and ponds, avoid stepping into the water along the shore. Ideally, a boat or raft works best for lakes and ponds. When working with streams and rivers work upstream to prevent silt from affecting the readings. If possible try to get a reading every 5 to 10 feet for a small section of stream. Adjust the distance according to the distance of the stream you are testing.

The Data

After collecting data, you can import the data into an Excel sheet. This makes it easy to analyze the data. If you want to see how I looked at the data, check out Level 6 of Sensing for Science video found on the Sensing for Science website at the QR code below.

For reference, the limit of TDS for drinking water is 500 ppm or less. Ideally, 150 ppm or less is the best for drinking.

www.montana.edu/smrc/sensingforscience/



Conclusion

This project can lead to other projects and extend the use of Arduino. The student may find after looking at the data that there is a high TDS or find a spike of TDS in the area of the pond or along a stream. Why is that the case? Can the students design and build another Arduino circuit that can help answer that question?

Arduino can be used for other science projects. Such project could be:

- Bionic hands
- Automated Greenhouse
- Motion Detector
- Self-driving robots

If there is a project that your students want to do, feel free to email me at kayce.williams@montana.edu. I can help you with finding resources and parts for the project.

Lastly, check out the cool resource being done in Montana on water quality and other Montana NSF EPSCoR projects by going to the website using the QR code below.

Thank you

