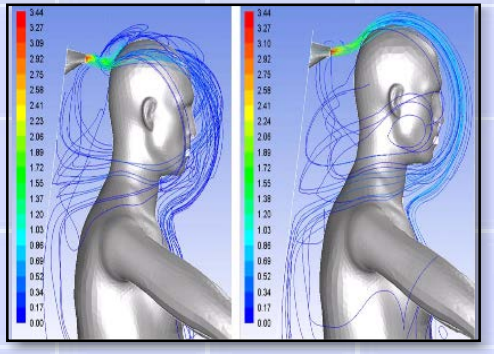



Save Your Breath

MAKE it NASA

Save Your Breath

Agenda

- Introductions and NASA Content Overview
- Applying Design Thinking to a Making Activity
- Building a Sensing System
- Programming the Microcomputer
- Save Your Breath - Building the Test Apparatus
- Save Your Breath – Using CT to Modify the System
- Conclusion

Share Your NASA Experience

Share your NASA experiences, pictures and videos

- Facilitators participating at NASA professional development workshops
- Students using NASA content
- Your organization connecting with NASA Subject Matter Experts

@NASAGRC_Edu – NASA Glenn Office of STEM Engagement on Twitter

@NASAGlenn official accounts:

@NASAedu – NASA Office of STEM Engagement official Twitter account

- Be sure to use the hashtag #NASAGlennSTEM

Save Your Breath

- Students build and test a data-collection system consisting of a sensor, microcontroller, and transmitter that could be used during lunar exploration.
- Students will then use computational thinking practices to modify their system to detect carbon dioxide in model spacesuit helmets and signal an alert if CO2 exceeds acceptable levels.
- Students will test their system through a simulated CO2 build-up incident.

The Student Journal

- This document will help students organize their thoughts and track their progress through the challenge.

- Instructions on the left-side pages; student work on the right-side pages.

Working in Teams

Making projects are all about individual creativity, however the best ideas rarely come from just one person. For this project, it is highly encouraged to have students work in teams of two or more.

Assign roles to team members or have students select their own roles.

- **Design Engineer** – sketches, outlines, patterns, or plans the ideas the team generates
- **Technical Engineer** – assembles, maintains, repairs, and modifies the structural components of the design
- **Operations Engineer** – sets up and operates the prototype to determine what parts work like they should and what can be improved.
- **Technical Writer/Videographer** – records and organizes information, data, and prepares documentation, via pictures and/or video to be reported and published.

Design Your Mission Patch

- Establish a mission/project name – Many NASA missions are named based on the work they do.

- Design a mission patch – Scientists and engineers that work on NASA missions and spacecraft are unified under mission patches that are designed with symbols and artwork to identify the group's mission.

- Create a vision statement – This is a short inspirational sentence or phrase that describes the core goal of the team's work. NASA's current vision statement is:
  - "We reach for new heights and reveal the unknown for the benefit of humankind."

NASA's Past, Present and Future at the Moon

- On October 15, 2019, NASA unveiled its designs for their new Exploration Extravehicular Mobility Unit (xEMU) suit and the Orion Crew Survival System (OCSS).
- The xEMU suit improves on the suits from the Apollo era and spacewalks outside the International Space Station.
- They will be will be worn by first woman and next man as they explore the Moon as part of the agency's Artemis program.

- The Artemis 1 mission, the first integrated test flight of NASA's new Orion spacecraft and the Space Launch System rocket.
- Artemis I will be an uncrewed flight test that will demonstrate capabilities to extend human existence to the Moon and beyond.
- It is currently planned to launch in November 2021.

- On April 30, 2020, NASA announced the selection of three U.S. companies selected to design and develop human landing systems for the moon.
- One of these systems will be chosen as the vehicle to land the first woman and next man on the moon, planned for 2024.
- NASA is returning to the Moon for scientific discovery, economic benefits, and inspiration for a new generation.

Developing New Spacesuits

https://www.nasa.gov/feature/what-are-the-next-generation-spacesuits

The Design Thinking Process

- This process can be used to solve any problem that requires designing and making a solution.

- Adapted for student use from *Design Thinking for Educators*

https://designthinkingforeducators.com/design-thinking/.

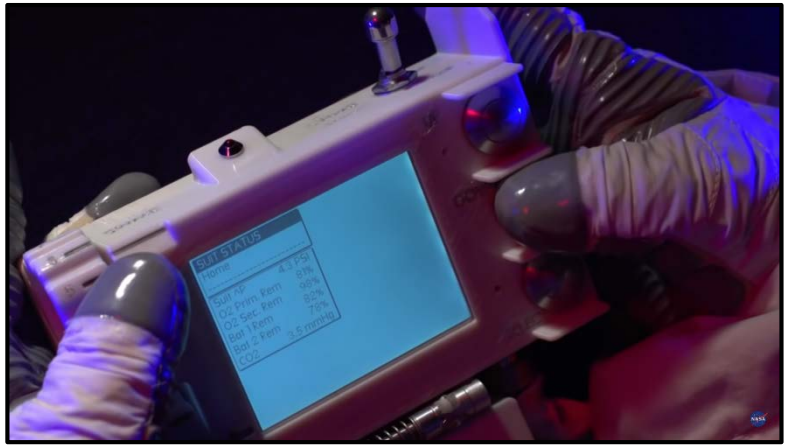
- To make a great solution, first you have to understand the problem.

- Make a list of challenges that need to be solved. These are questions that need answers. (How can…? How would…? How does…?)

- Next, brainstorm any criteria, constraints, and barriers that are part of this problem.
  - **Criteria** are things your solution has to be able to do.
  - **Constraints** are things your solution must not do.
  - **Barriers** are things that could prevent you from finishing your solution.

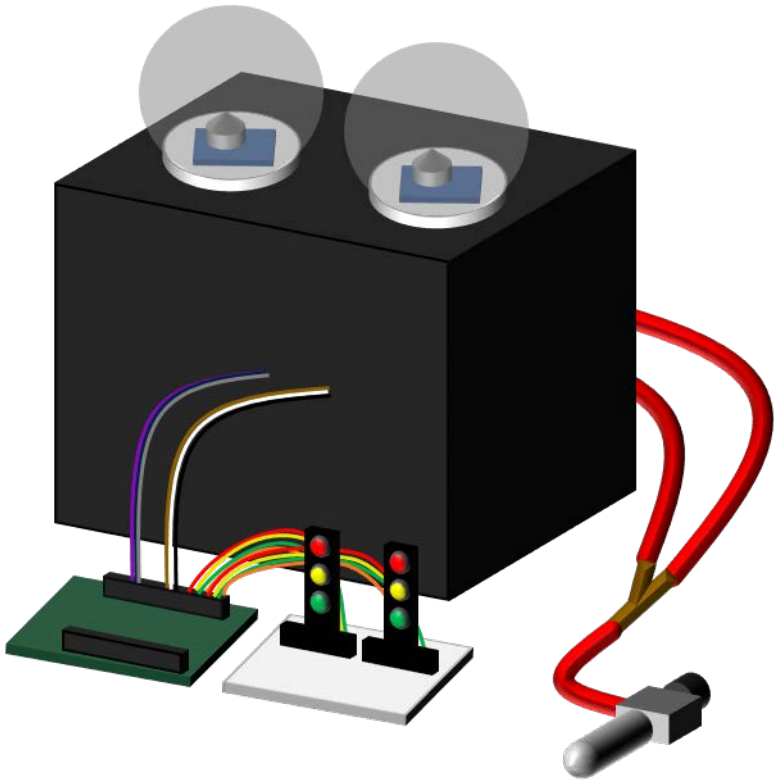
- Student teams plan how to use carbon dioxide sensors inside a helmet to keep astronauts safe.

- They program sensors to alert the user to dangerous or borderline-dangerous levels.



- Students will insert their carbon dioxide sensors into model astronaut helmets.

- They must determine how to alert a user to a dangerous level of carbon dioxide in either one or both helmets.

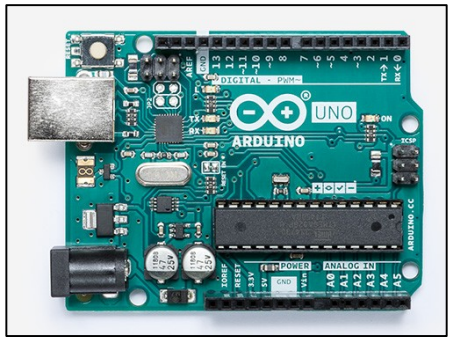


- Students will build a sensor-microcontroller-transmission system using COTS parts.

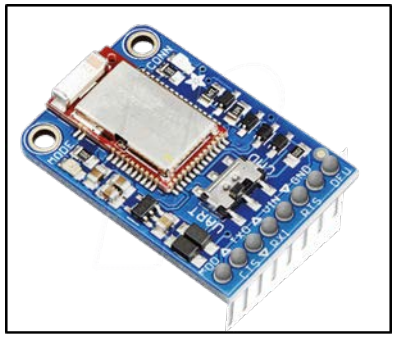
Ambient Light Sensor

Carbon Dioxide Sensor

LED Traffic Light



Microcontroller



Bluetooth Radio

Some examples of challenge questions:

- Organization of Electronic Components
  - How should parts be arranged?
  - How can it stay organized and easy to follow?
- Programming the computer
  - How can we keep the program organized and easy to understand what is going on?
  - How can we test each step along the way?
  - How can we use code efficiently for duplicate sensors?
  - How will we send data to and receive data from the sensor system?
- Transmission of Data
  - What amount of CO2 should be used as a threshold?
  - How will we send data to and receive data from the sensors?

Interpret

• **I learned something.** *How do I interpret it?*

- To begin solving this challenge, you will need to find answers to the criteria, constraints and barriers you listed.

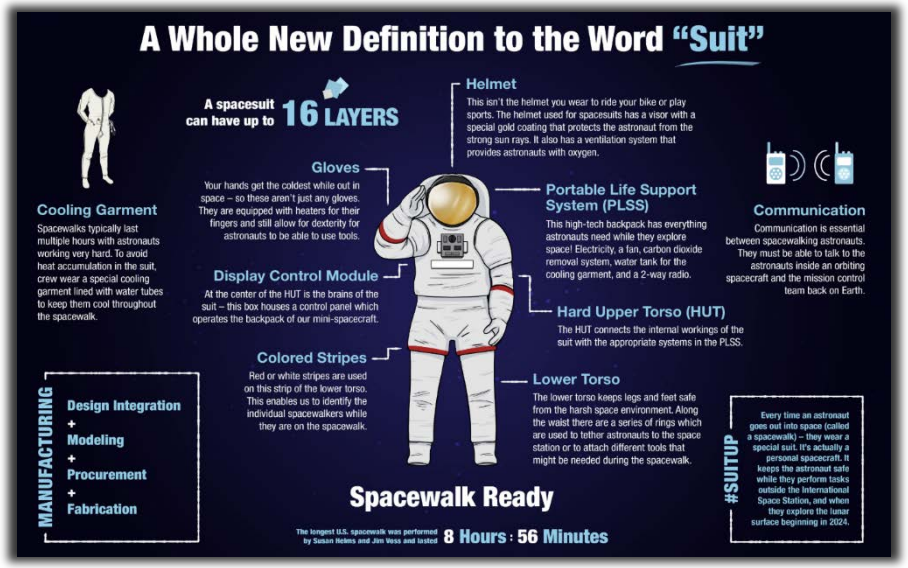
- Brainstorm resources you could use to get more facts about your challenge. You could use books, experts or trustworthy sources on the Internet.

- Access the sources you listed. Write down the key ideas that you learned. Think about how those ideas should affect your solution.

NASA Resources for This Activity

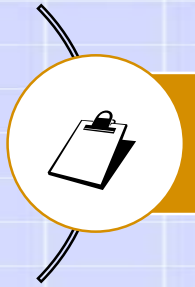
Sample of NASA Resources

- https://www.nasa.gov/specials/artemis/
- https://www.nasa.gov/suitup
- https://www.nasa.gov/feature/spacewalk-spacesuit-basics
- https://www.nasa.gov/content/life-support-systems



NASA Resources for This Activity

- A sheet of NASA website resources has been provided for you.
- Review these resources and determine which will be most helpful for your students. Things to consider:
  - What problems are your students trying to solve? Do the sources help answer their questions?
  - Are they at an appropriate reading level for your students? Will you have to help them through?
- NASA has a lot of information, but also consider additional resources from other reputable sources



Ideate

• **I see an opportunity.** *What do I create?*

- Use the things you learned to create a sketch of your solution. Label all major parts.

- Describe how it works.

- Build your solution for the first time. This model is called a **prototype**.
- Think about what materials you need to make each part of your model.
- Take a picture of your model. Add it to your journal.

Resources can come from a variety of places.

Six Suggested Starter Categories for an Elementary Makerspace (Fontichiaro, 2016)

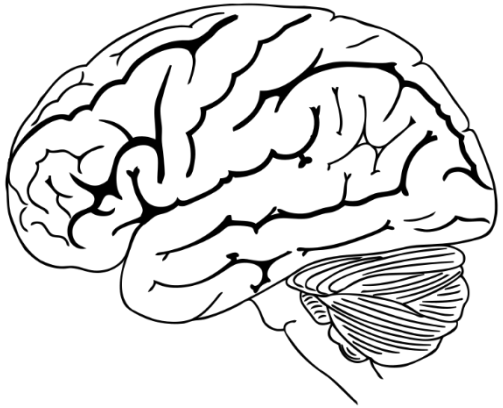
Craft	Engineering	Code	Circuits (& Computing)	Digital Design	Needle & Thread
• Origami • Modeling Clay • Wikki Stix • Scrapbooking • Junk Box creations • Recycled Materials • Challenges	• Tinkertoys • LEGO • K'Nex • BuildWithChrome.com	Robots: • Dash & Dot • Sphero • Ozobot Animation: • Scratch • Blockly • Hour of Code Apps: • Hopscotch • Scratch Jr. • Daisy the Dinosaur	• Arduino • Raspberry Pi • Lego Mindstorms • Snap Circuits • Squishy Circuits • littleBits • K'Nex with electrical components • Circuit blocks	• Canva.com • Picmonkey.com • Makebeliefcomix.com • Pixton.com • 3-D printers • Laser cutters	• Hand sewing • Machine sewing • Knitting • Crochet • Fashion Hacking • Embroidery • Cross Stitch

Building a Sensing System

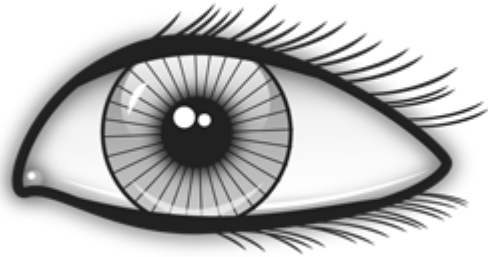
- For the purposes of this project, we will build and program a sensor array from a starter kit. We start with a single sensor.
- Students are encouraged to:
  - Expand the build - add more controls or indicators
  - Modify it – use a different programming sequence
  - Customize it – use a different system altogether!
- Since this is a making activity, you should feel open to make changes based on what works for you.

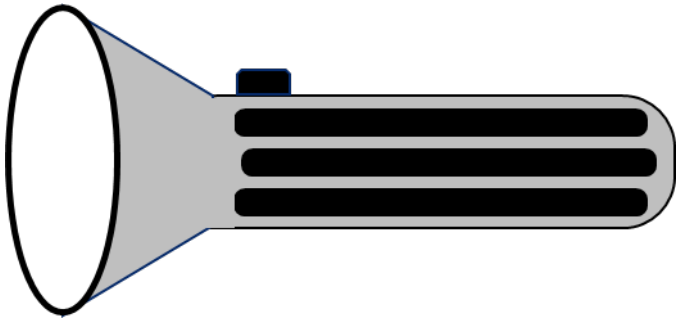
Humans and Robots - Not so Different

Step 3: Eye passes data to brain. Brain responds by signaling pupil to contract.



Step 1: Brain directs eye to open and look towards flashlight.

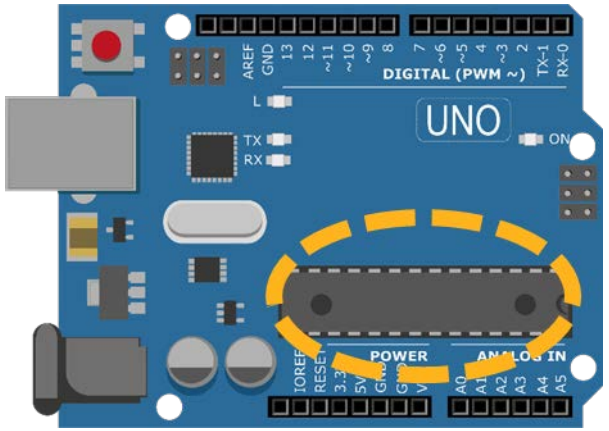




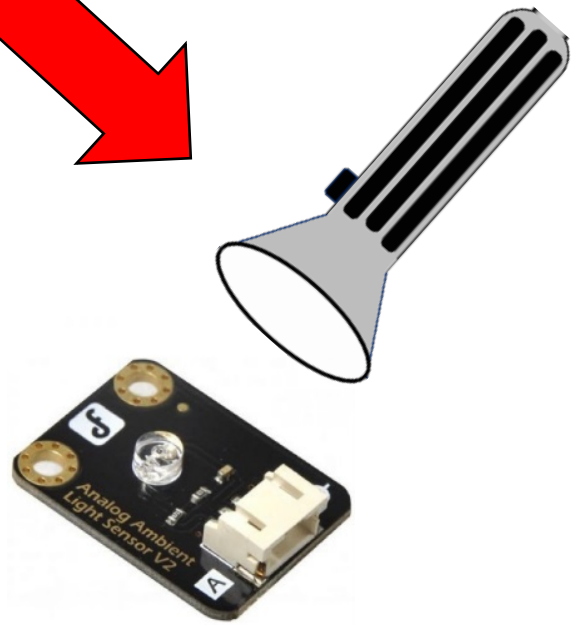
Step 2: Flashlight shines concentrated light into eye.

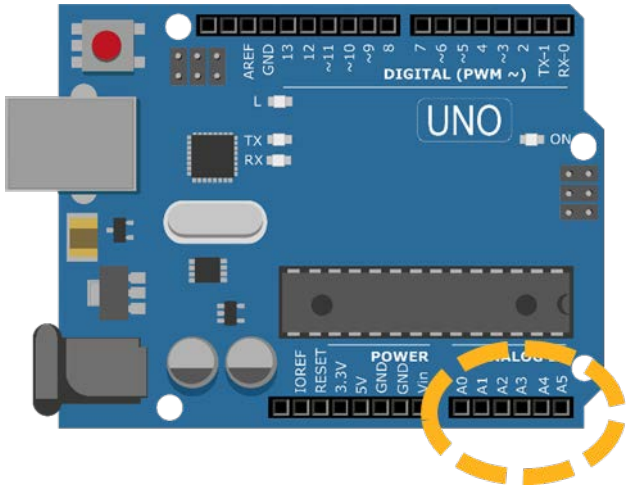
Humans and Robots - Not so Different

Step 3: Port passes data to CPU for analysis. CPU responds with further instructions.



Step 1: Central Processing Unit (CPU) on Arduino directs light sensor to accept data.





Step 2: sensor data passes through port on Arduino

Humans and Robots - Not so Different

Human

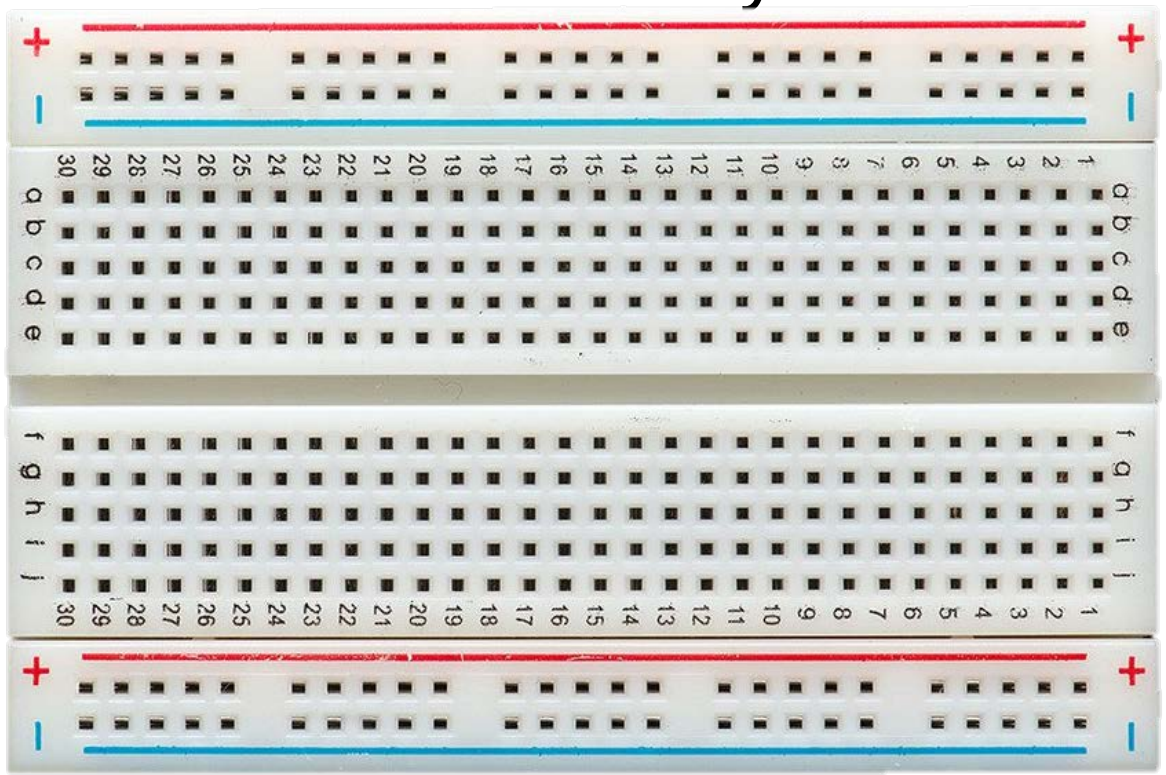
- Blood vessels:
  - Arteries take blood to cells
  - Veins return blood to source
- Nerves
  - Motor nerves send command signals from the brain to the body.
  - Sensory nerves take data from the body to the brain

Robot

- Power Wires:
  - Connected to 5V, give electricity from the source
  - Connected to GND, take electricity back to the source
- Signal Wires
  - Connected to Digital terminals, give start/stop commands to various parts of the robot
  - Connected to Analog terminals, take ranged data from sensors

Building Circuits With Breadboards

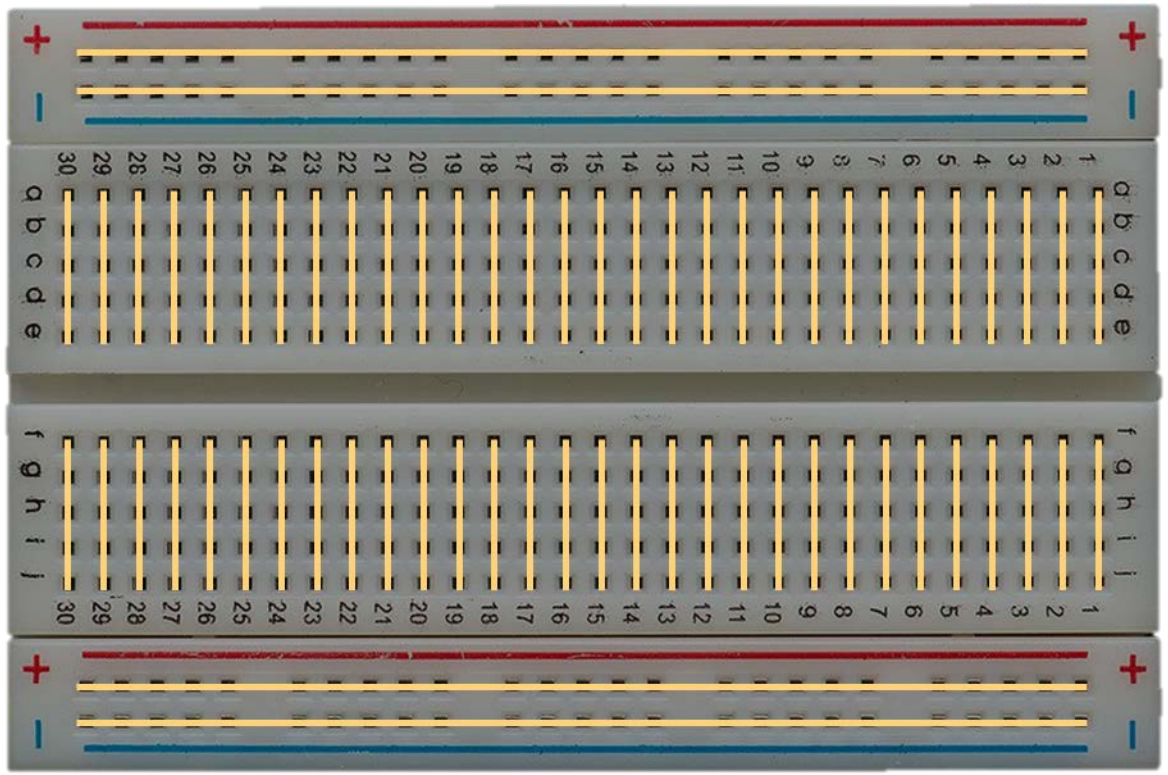
A breadboard is a convenient way to connect everything.



You can’t see them, but there are lots of wires inside it

Building Circuits With Breadboards

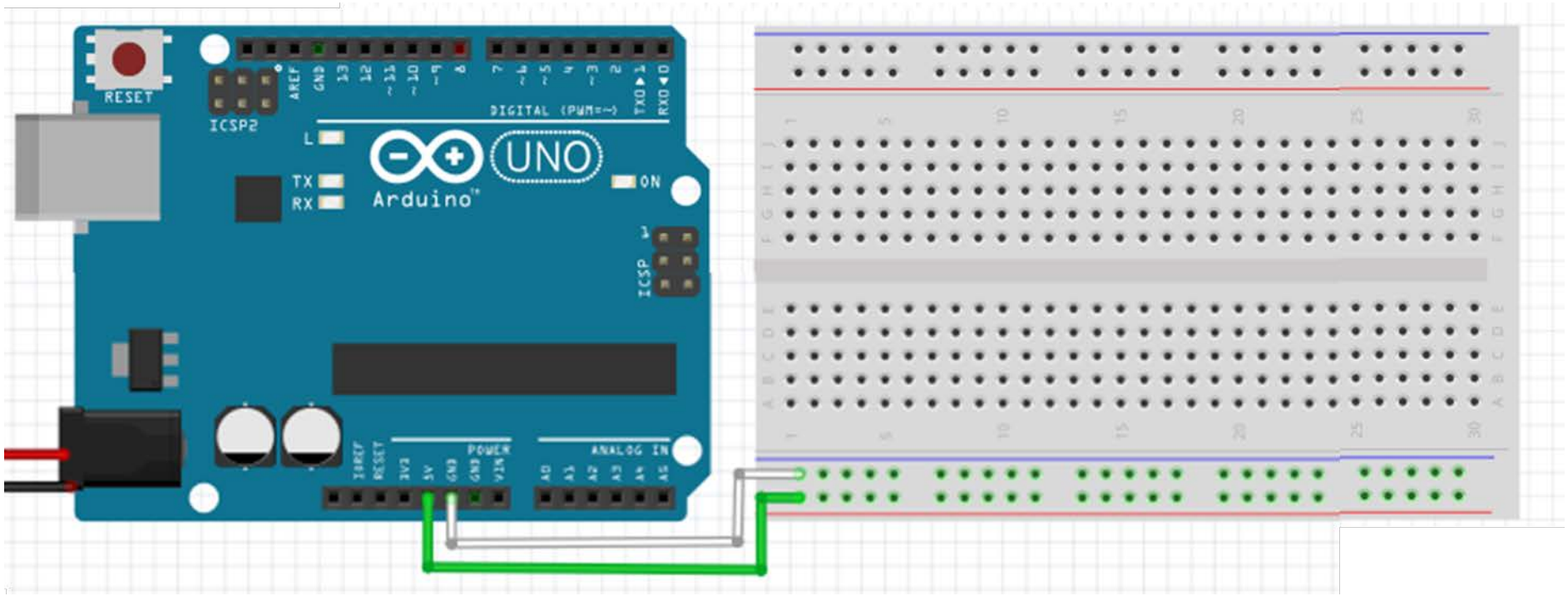
The two top and two bottom rows are connected across the board.



All of the numbered channels are connected a-e and f-j with a gap in the middle. This gap can be bridged with compatible components.

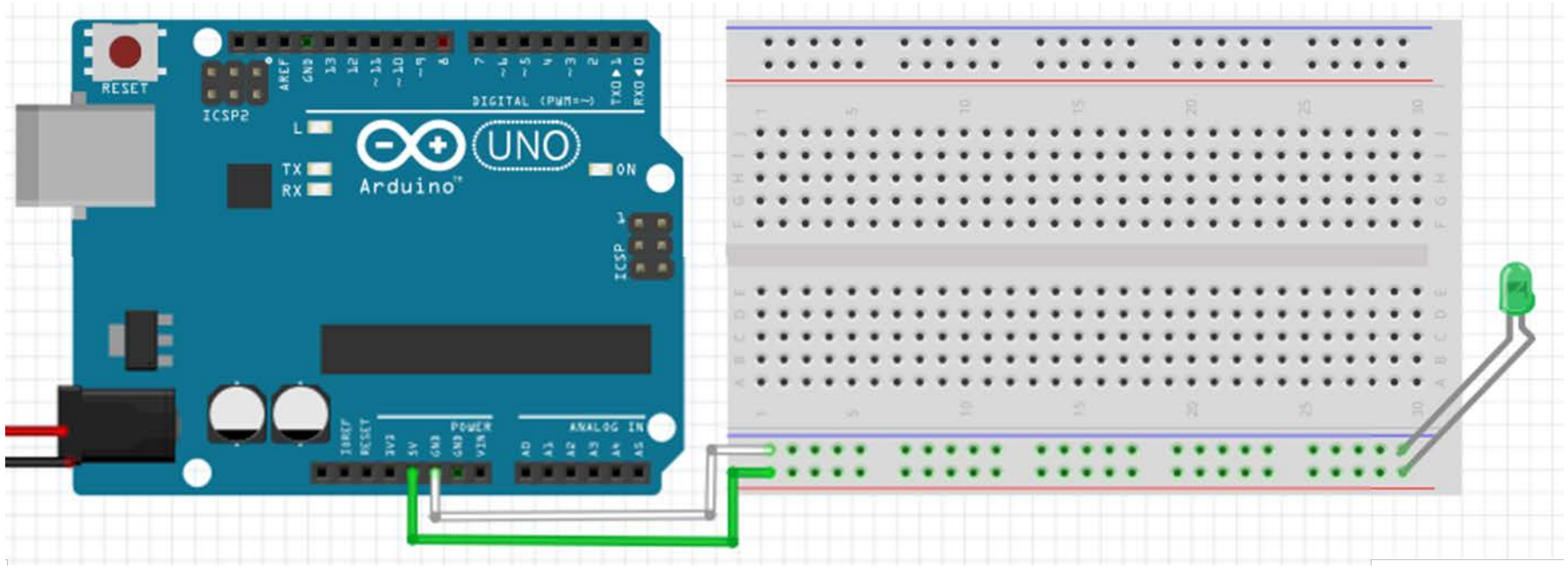
Step 1: Connect 5V and GND

These 5-volt (5V) and Ground (GND) terminals are for things that are always powered (like the Bluetooth radio).



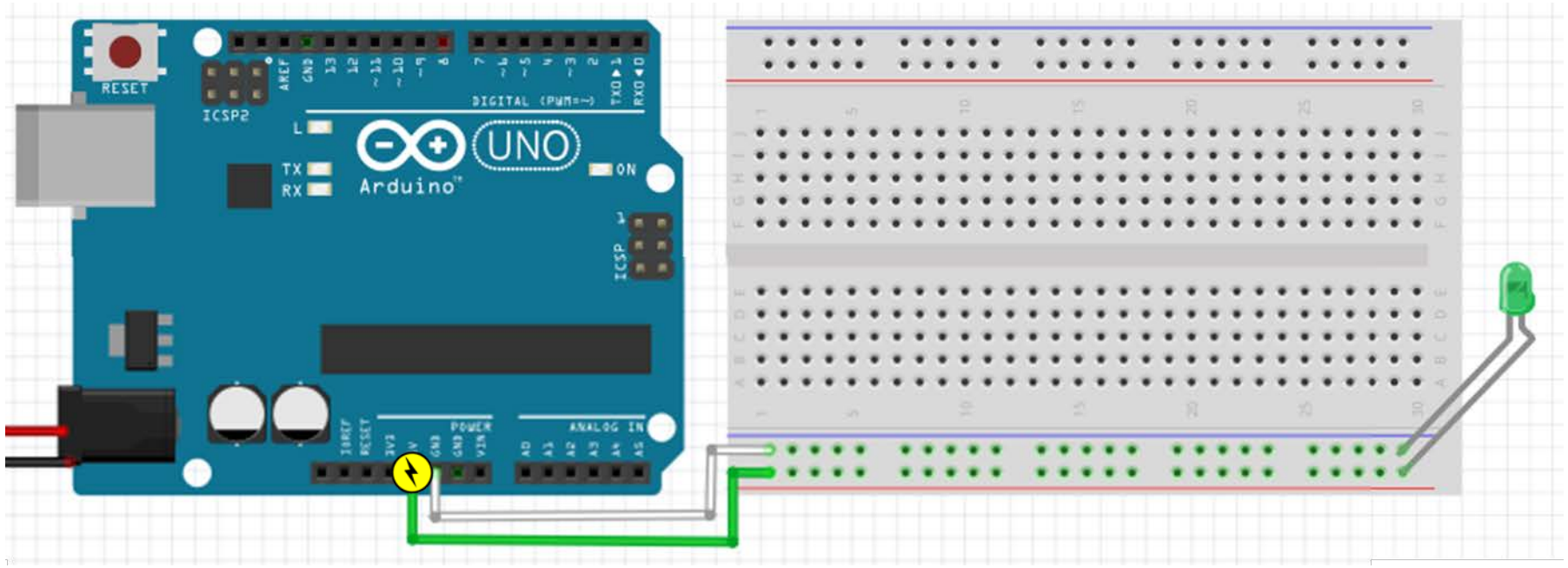
Step 2: Connect a Light Emitting Diode (LED) with Resistor

An LED is an easy way to test if your circuit is working. These LEDs come with a built-in resistor to prevent it from overheating.



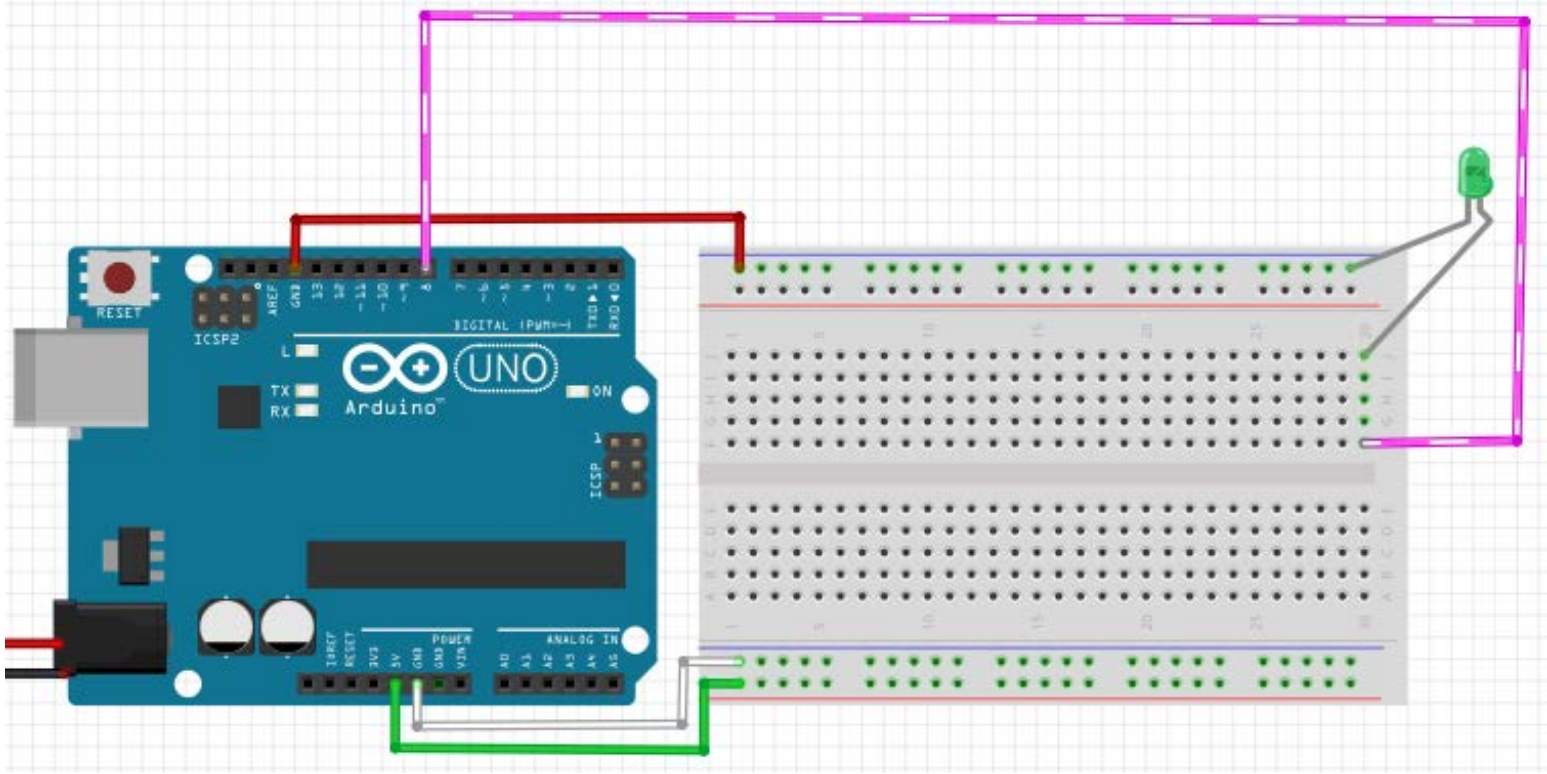
Step 2: Connect a Light Emitting Diode (LED) with Resistor

An LED is an easy way to test if your circuit is working. These LEDs come with a built-in resistor to prevent it from overheating.



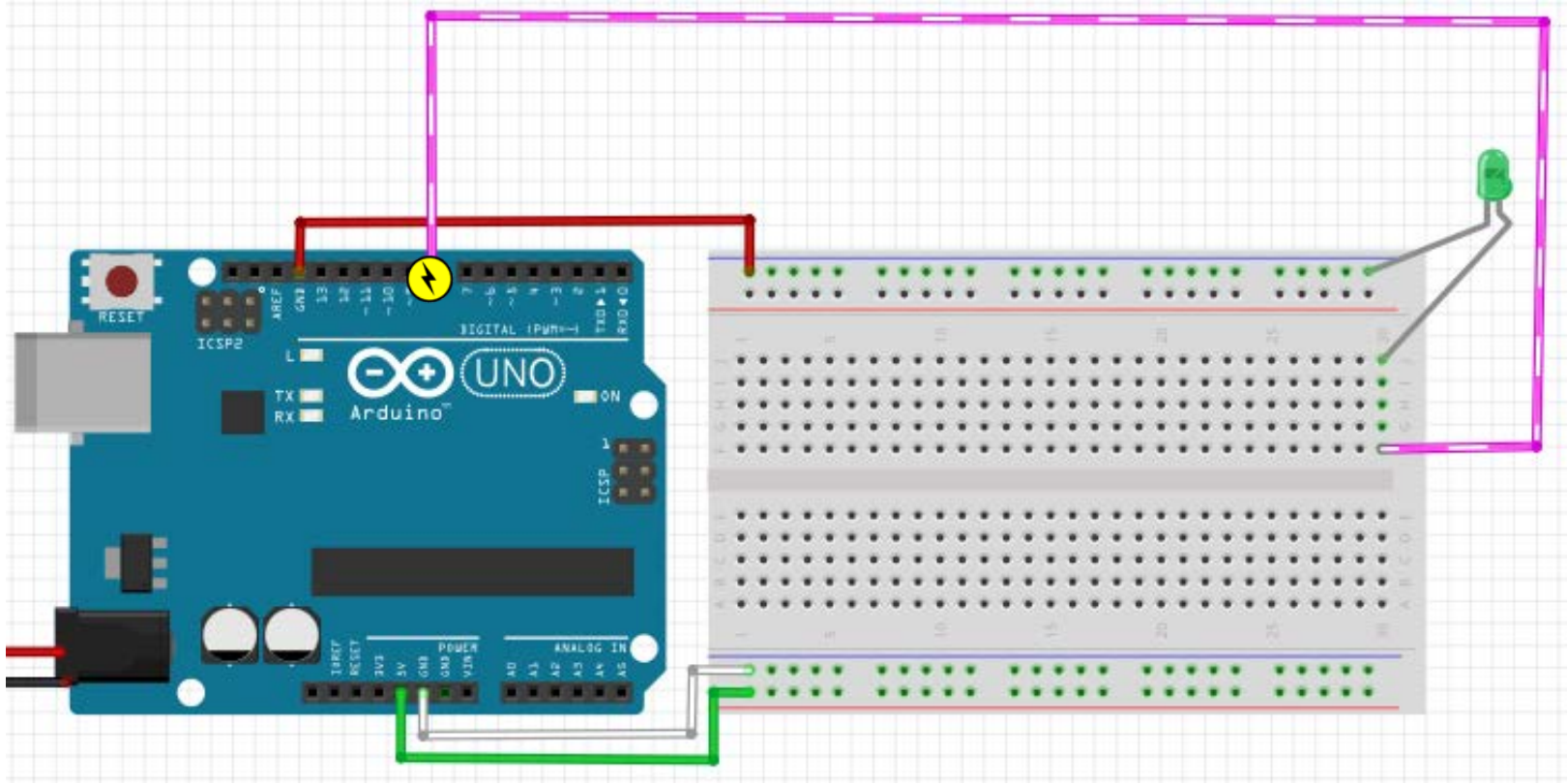
Step 3: Move the LED to Control the Light

Now we will move the light to the other side of the board. We will use pin #8 today, but any pin 2-13 could be used.



Step 3: Move the LED to Control the Light

Rather than be “always on”, this will allow us to control the light with a program we will install on the Arduino.

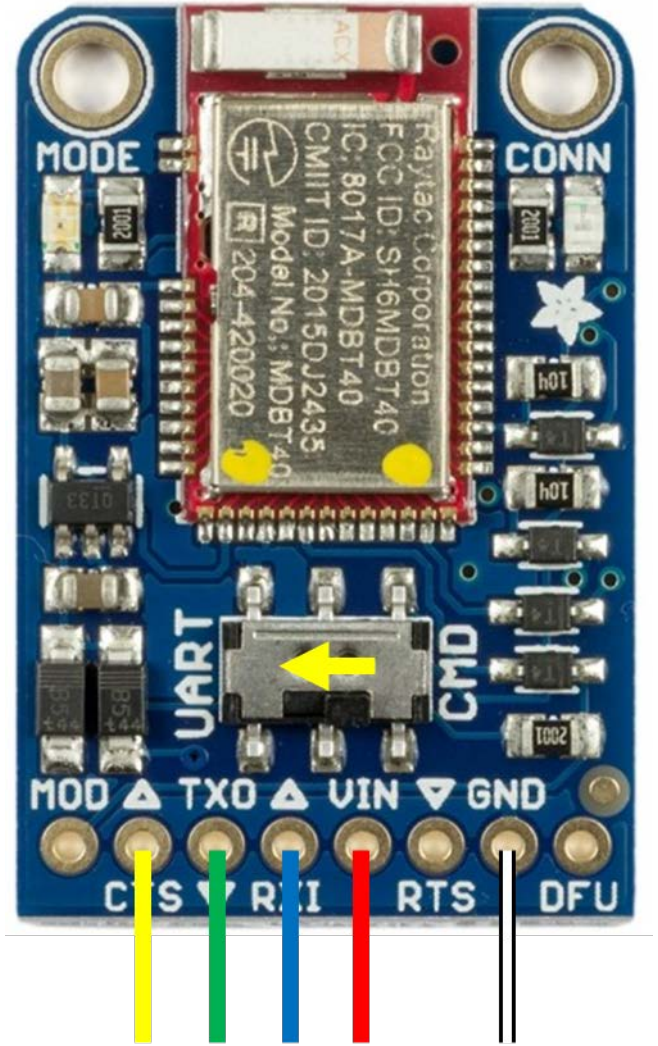


Step 4: Add Bluetooth Transmitter

- We chose this module because it has a ready-made app for both Android and iOS and doesn't require resistors with an Arduino.

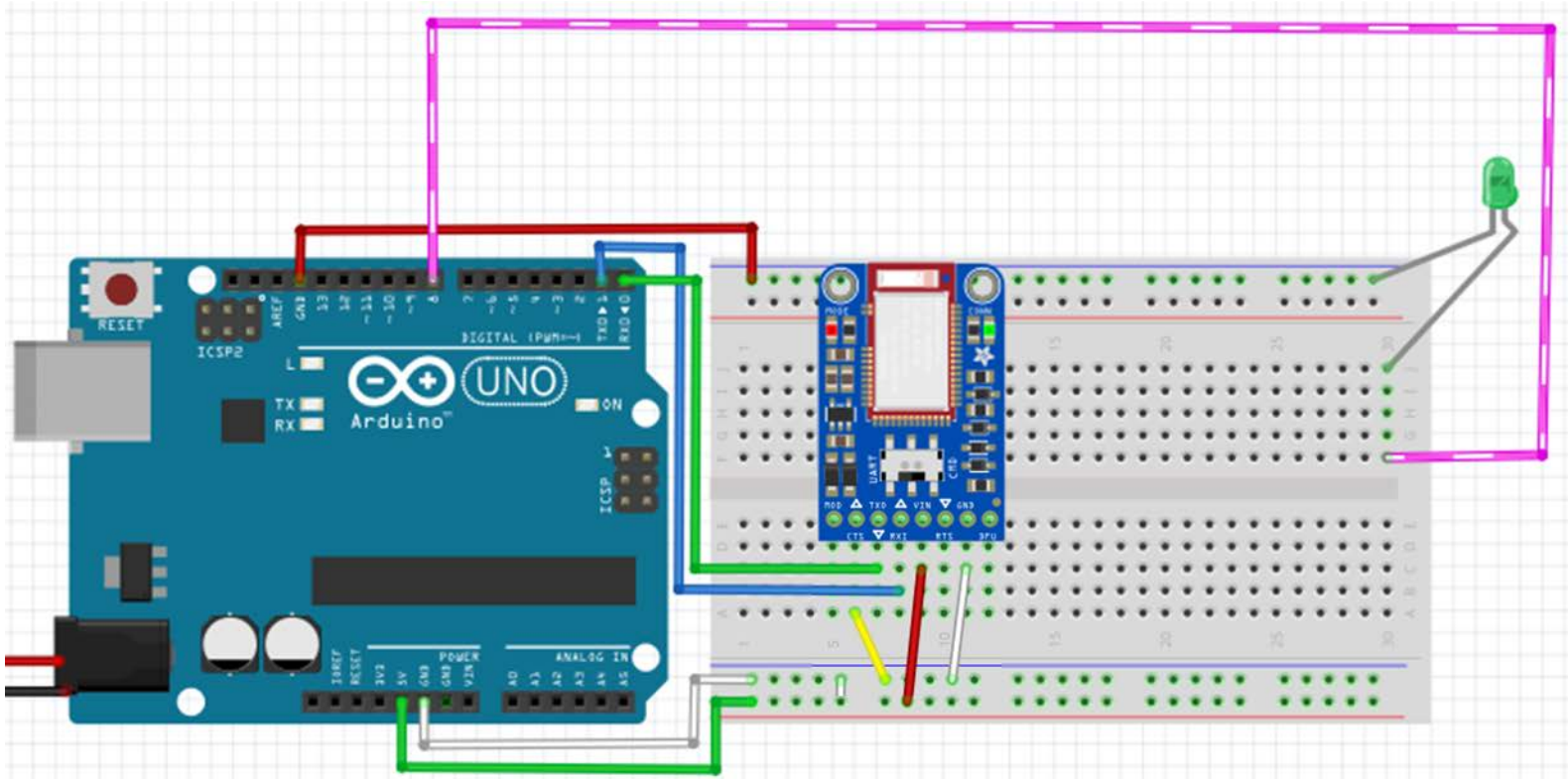
- There is a small physical switch on the module. Set it to Universal Asynchronous Receiver/ Transmitter (UART).

- This module has several pins. The ones we will use are:
  - Voltage In (**VIN**) – Powers the radio
  - Ground (**GND**) – Powers the radio
  - Receive In (**RXI**) – Gets data from robot
  - Transmit Out (**TXO**) – Sends data to robot
  - Clear-to-Send (**CTS**) – Allows radio to control the robot



Step 4: Add Bluetooth Transmitter

The pins on the Bluetooth radio are designed to fit right into the breadboard. Connect the **power** and **signal** wires. Be sure to connect “Transmit” on the Arduino to “Receive” on the Bluetooth and vice versa.

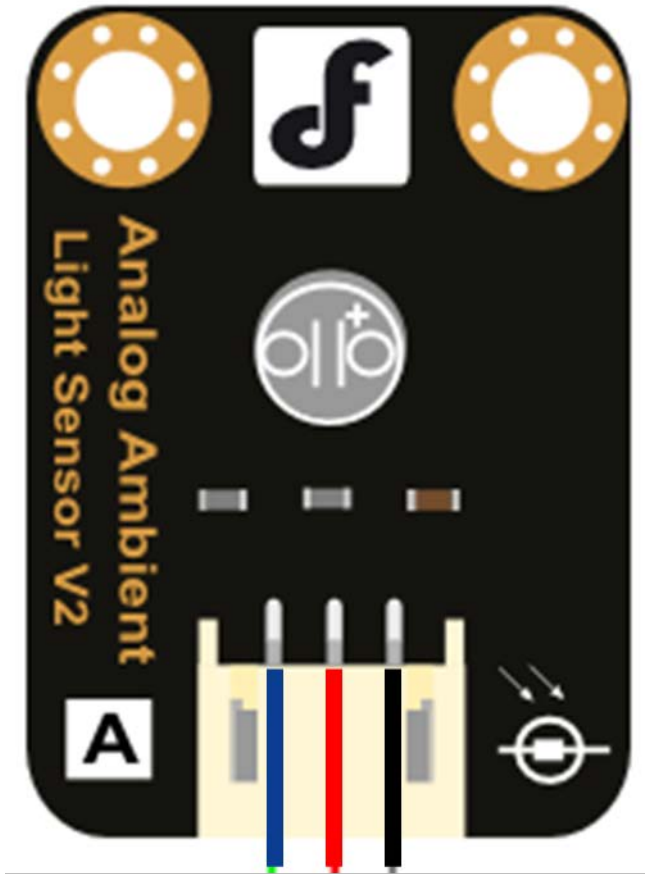


Step 5: Add the Light Sensor

- We chose this sample sensor because it is an easy to program and calibrate sensor.

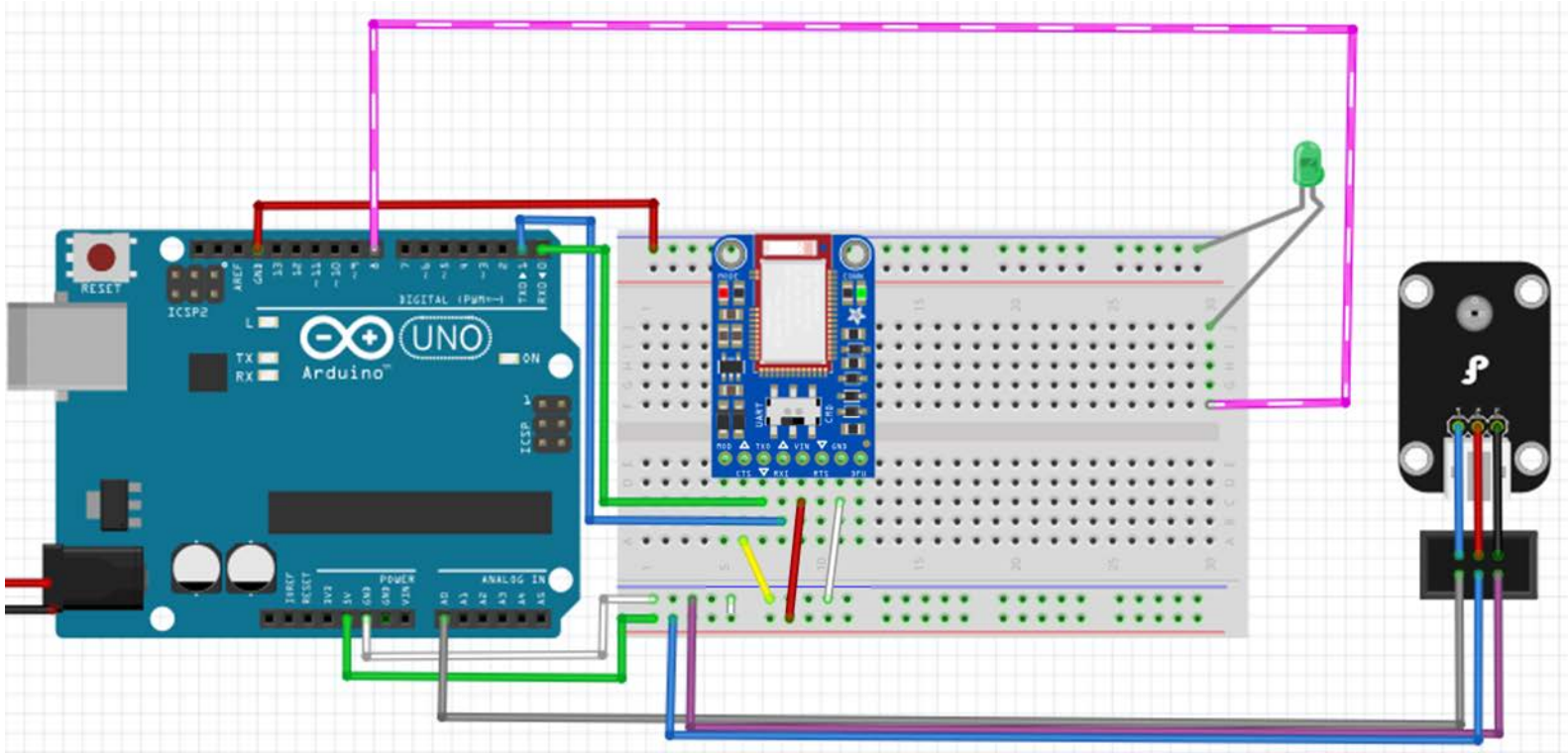
- It has only 3 pins:
  - **5V** – Power in
  - **GND** – Power out
  - **Output** – Analog signal out

- As the ambient light changes around the sensor, it sends a different amount of voltage out of the **output** pin. We can convert this voltage to a percent of intensity value.



Step 5: Add the Light Sensor

The sensor has a ribbon that can be connected to the breadboard. Connect the power wires the same as before. Connect the signal wire to pin A0.



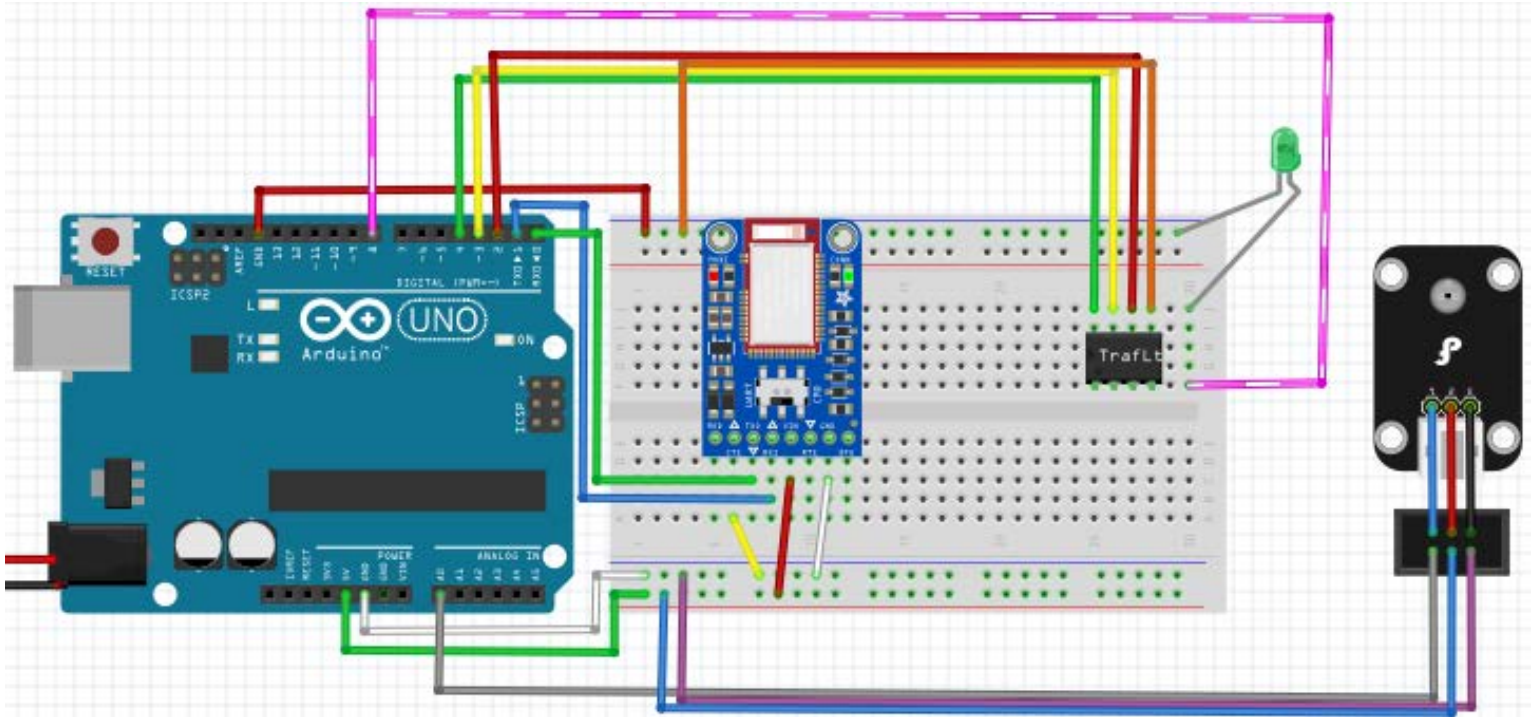
Step 6: Add a Traffic Light

- Add a traffic light to the breadboard.

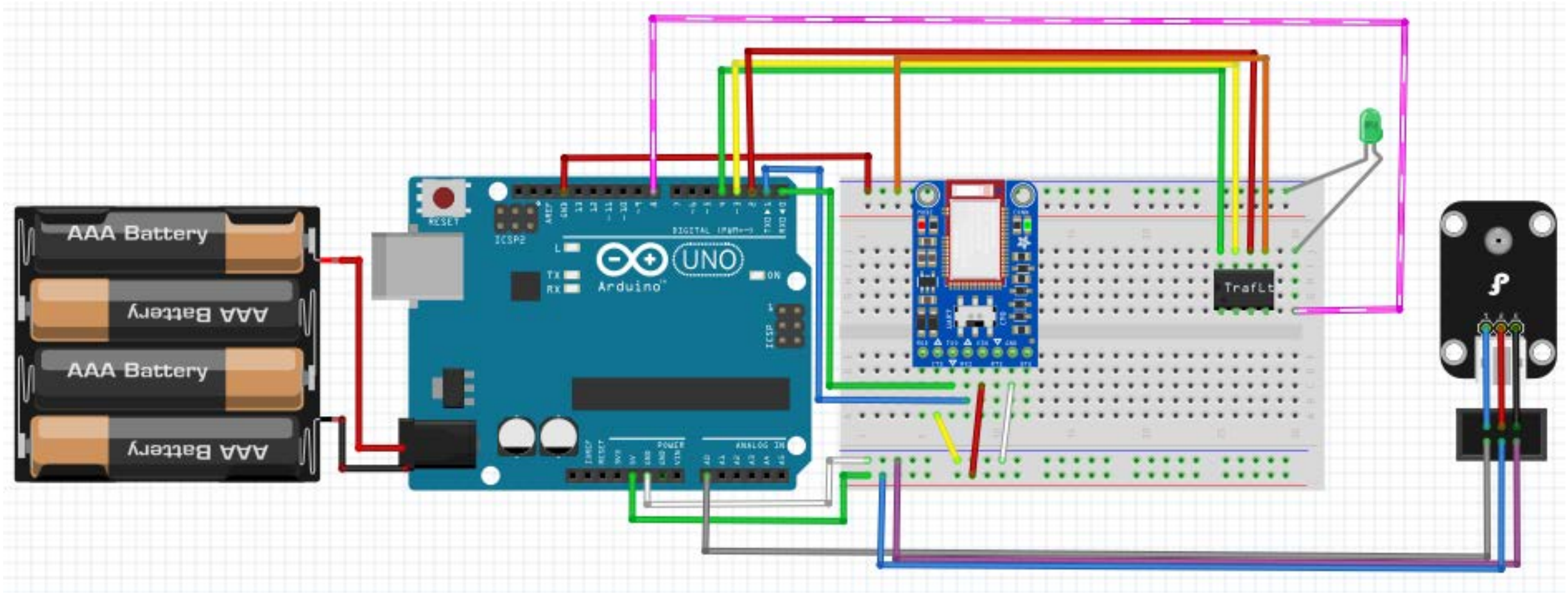
- This part is just 3 LEDs wired together with a single common GND pin.
  - R = Red
  - Y = Yellow
  - G = Green

Step 6: Add a Traffic Light

Connect the Red, Yellow, Green pins using a four-wire ribbon to **D2**, **D3**, and **D4** respectively. Use the fourth wire to connect the **GND** pin to the GND on the breadboard.



Complete Wiring Diagram



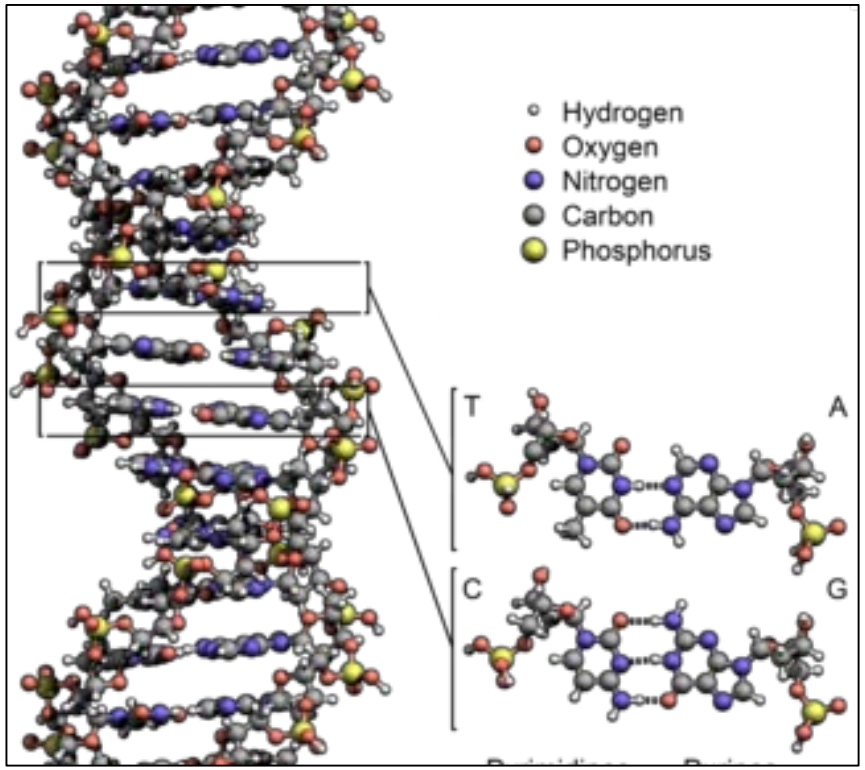
Programming the Sensor

- A program will follow the instructions it has been given.

- It does exactly what it has been instructed to do – no more, no less – even if those instructions are destructive to the system.

Programming the Sensor

Human



Computer

```
package de.wikibooks.javaee;

import javax.naming.InitialContext;
import javax.naming.NamingException;

public class StandaloneMain {

    /**
     * @param args
     */
    public static void main(String[] args) {
        StatelessSessionRemote myEJB = null;
        InitialContext ic;
        try
        {
            ic = new InitialContext();
            myEJB = (StatelessSessionRemote) ic.lookup("de.w

        } catch (NamingException e) {
            e.printStackTrace();
        }

        if (myEJB != null) {
            int result = myEJB.getAplusB(3, 4);
            System.out.println("Ergebnis: " + result);
```

Programming the Sensor: Blink Codes

It can be hard to tell what is going on inside the Arduino. Blinking an LED can be a simple way to know what is going on.

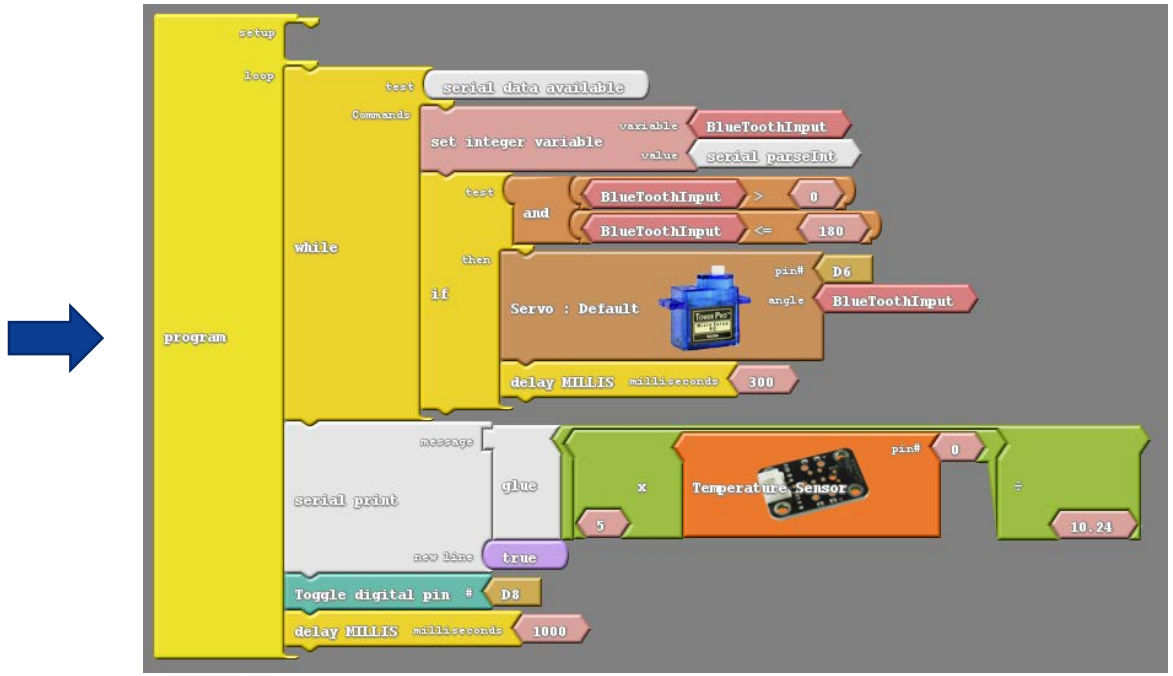
For example:

- On for a second, off for a second = “Everything is fine.”
- Two quick blinks = “Something important just happened.”
- Three long blinks = “Something went wrong.”

Programming the Sensor

Block code helps to visualize the sequence of code more easily than text.

```
#include <Servo.h>

int _ABVAR_1_BTInput = 0 ;
Servo servo_pin_8;
int __ardublockAnalogRead(int pinNumber)
{
  pinMode(pinNumber, INPUT);
  return analogRead(pinNumber);
}



void setup()
{
  Serial.begin(9600);
  servo_pin_8.attach(8);
}

void loop()
{
  while ( Serial.available() )
  {
    _ABVAR_1_BTInput = Serial.parseInt() ;
    if (( ( ( _ABVAR_1_BTInput ) >= ( 0 ) ) && ( ( _ABVAR_1_BT
    {
      servo_pin_8.write( _ABVAR_1_BTInput );
    }
  }

  Serial.print(map ( __ardublockAnalogRead(A0) , 0 , 1023 , 0
  //Serial.print(" ");
```



Block programming can’t do as much, but what it can do is easier!

Step 1: Install Coding Software

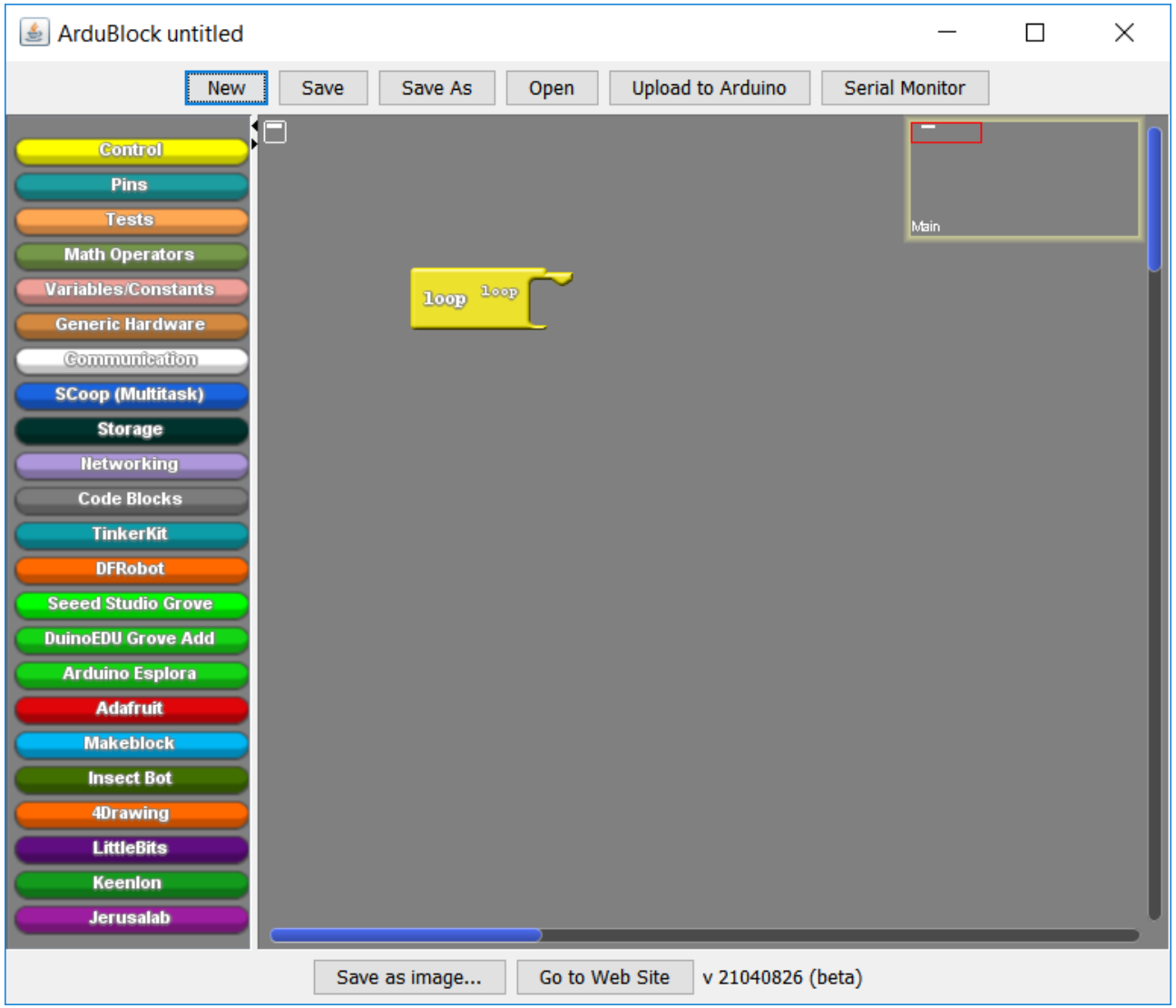
If you have not yet done so:

1. Download and install the Arduino software: https://www.arduino.cc/en/Main/Software

2. Download the latest version of Ardublock-all-master from here: https://github.com/taweili/ardublock/releases

3. Follow the Ardublock installation guide here: http://blog.ardublock.com/engetting-started-ardublockzhardublock/

The Ardublock Interface



The Ardublock Interface: Block Categories

Control - What to do, how often, and in what order

Pins – Working with digital output or analog inputs

Tests - Limits actions to certain conditions

Math Operators - Various arithmetic/ trigonometric calculations

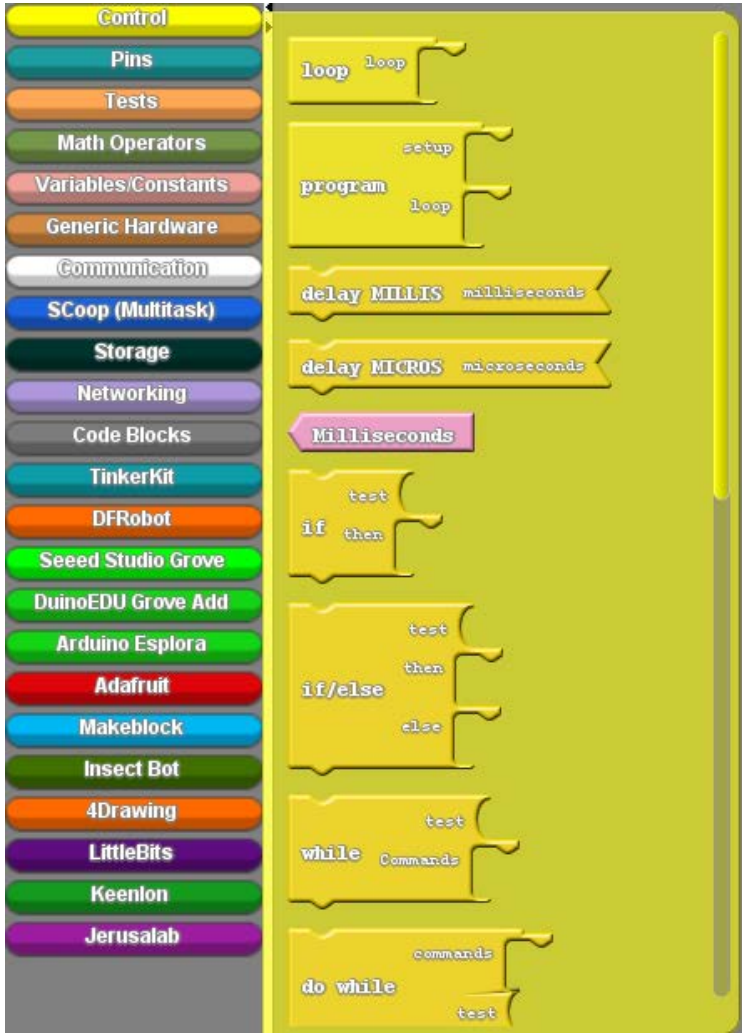
Variables/Constants - Storing, remembering, or changing values

Generic Hardware - Motors and other common hardware

Communication – Transmitting and receiving signals

Custom blocks to operate specific hardware from common suppliers.

The Ardublock Interface: Finding Blocks



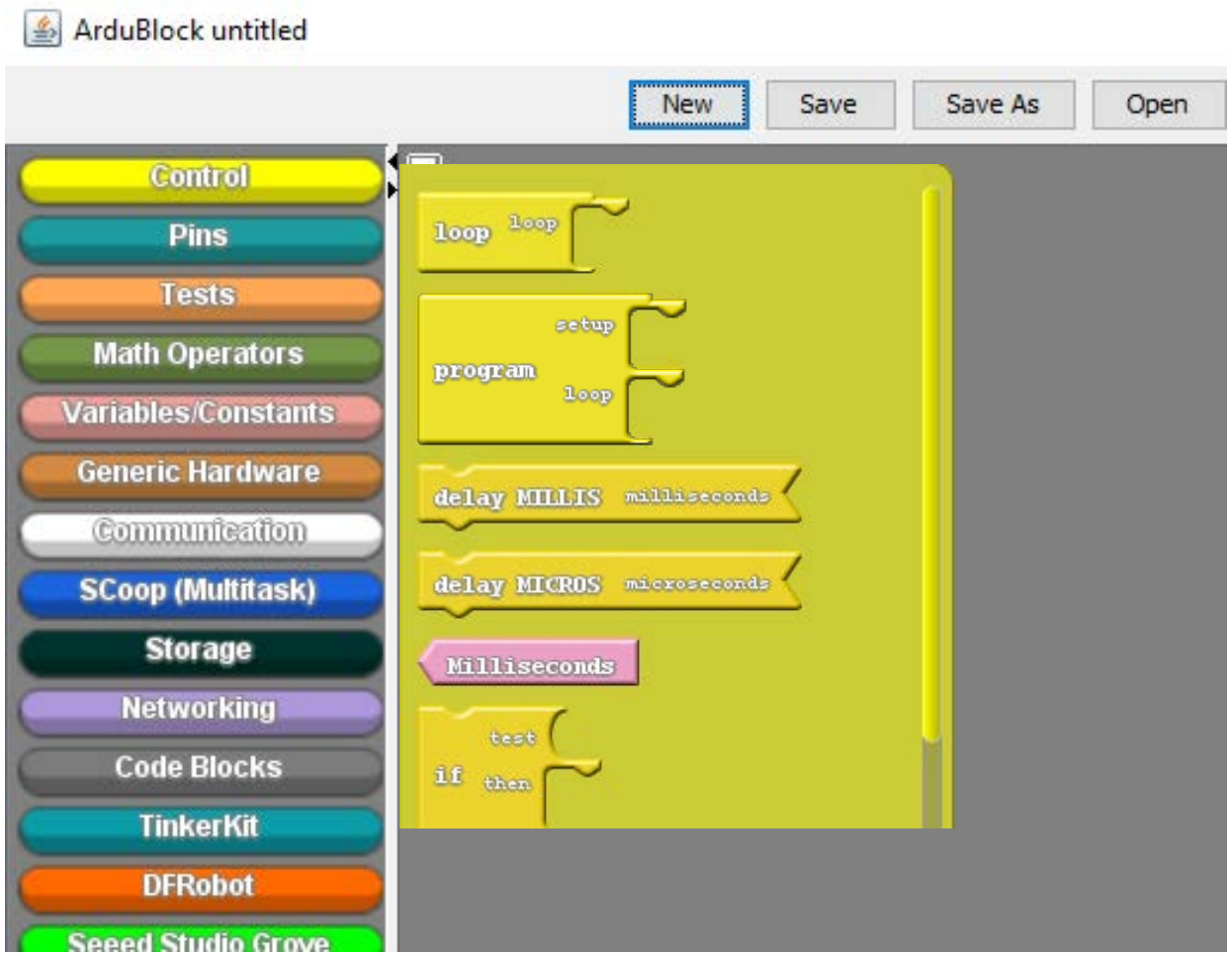
If you are looking for a yellow block, it is most likely in the yellow **Control** category. Likewise cyan blocks are most likely to be in the cyan **Pins** category

Note that not every block in the **Control** category is yellow.

Blocks with “sockets” on the right side will need another block that has a matching “plug” shape on the left side.

Some sockets can take more than one shape of plug. Experiment!

Step 2: Start Building the Program

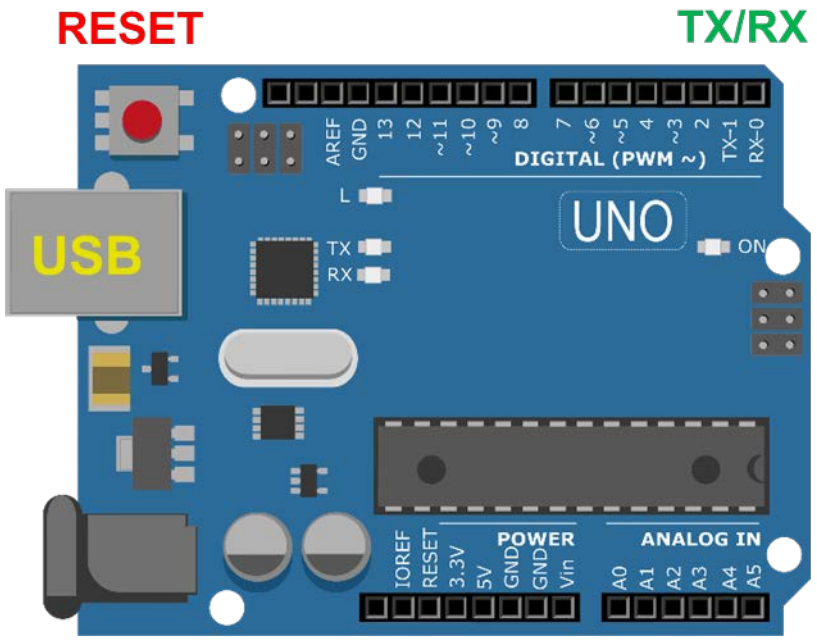


- By default the loop block is on every new Ardublock sketch.
- Drag off the “loop” block, then go to the **Control** menu, then drag the “program” block onto the sketch.
- The program block is the umbrella into which everything goes.
  - “Setup” happens once at the beginning
  - “Loop” happens over and over until the program is stopped.
- Blocks are processed top to bottom.

Step 3: The System Blinks Alive

- The “Toggle digital pin” block is found in the **Pins** category. It toggles a **Digital Output** from HIGH to LOW and vice-versa. Our LED was connected to pin **D8**.
- Normally, the loop runs as fast as the Arduino can manage. The “delay MILLIS” block is found in the **Control** category. It tells the program to wait for a set amount of time before doing anything else. Set it to **1000**.

Step 3: The System Blinks Alive

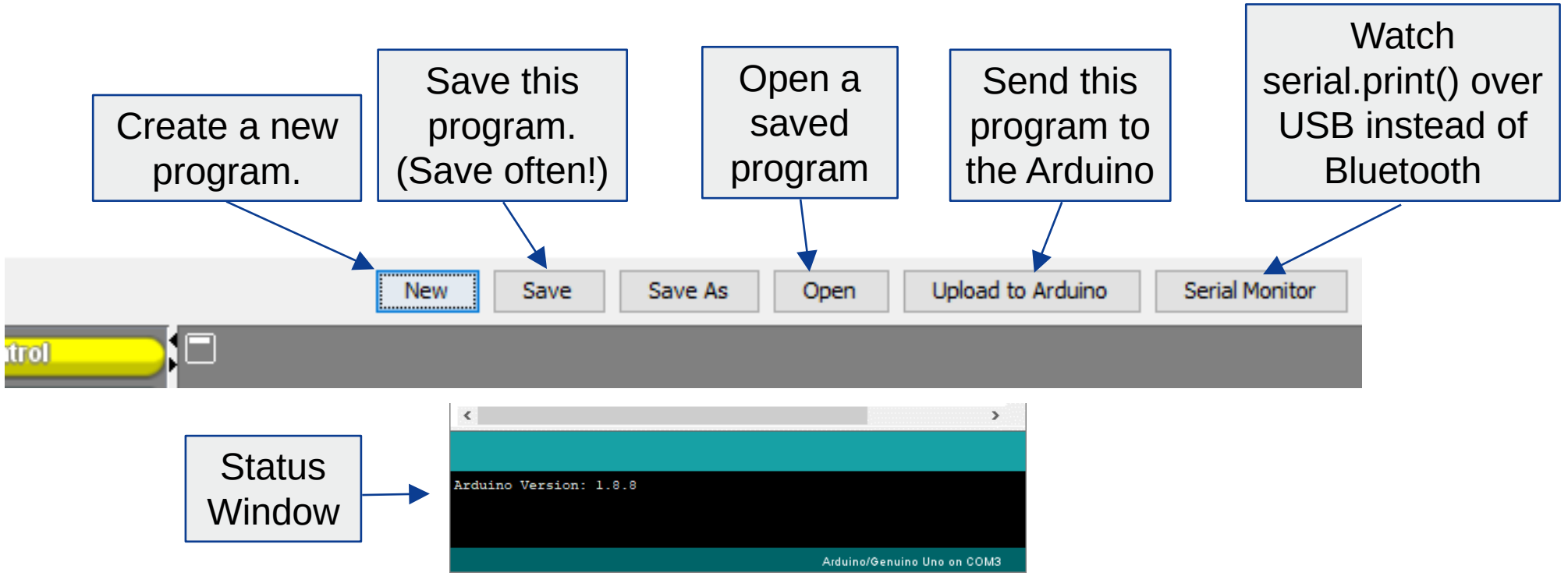


- Connect your Arduino to the computer via **USB** to upload your program. The computer can also power the Arduino through **USB** but not enough for motors or other higher-power hardware.
- The Arduino uses the same data connections to communicate with the computer via **USB** and anything connected via **TX/RX**. Temporarily disconnect the **TX/RX** wires when uploading code to the Arduino.
- Always reboot the Arduino with the **RESET** button after uploading a new program.

More info at: https://www.arduino.cc/en/Guide/ArduinoUno

Step 3: The System Blinks Alive Programming your Arduino

- When you “Upload to Arduino,” Ardublock will convert your block code to text code. Then, Arduino will attempt the upload.
- Status of the upload is listed at the bottom of the Arduino IDE window.
- Your LED should be slowly blinking now. If so, congratulations!



Step 4: Programming the Sensor

- Many sensors use semi-conductive materials to detect changes in properties around them. The way current that passes through them changes when temperature, light, or humidity changes around the sensor.

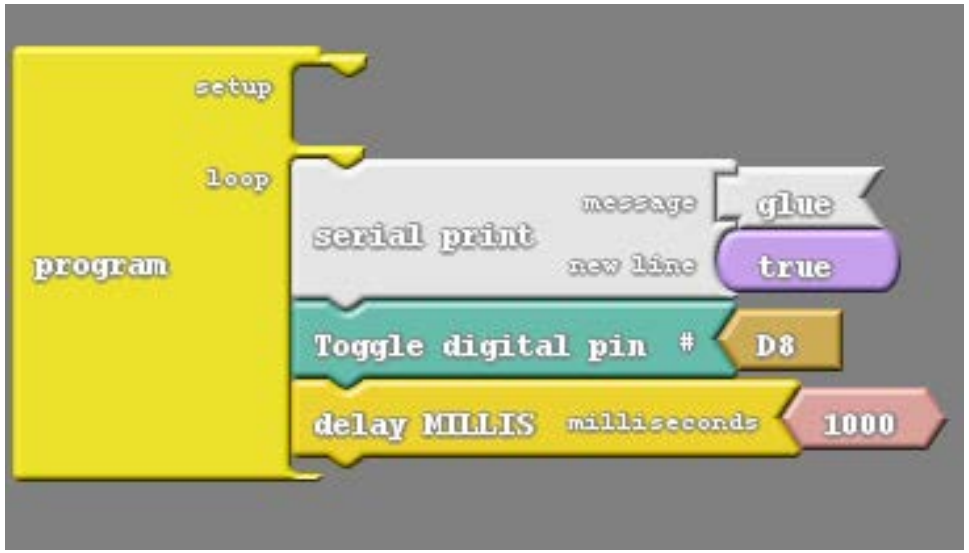
- The PT550 photoreceptor produces a number from 0 to 1023 based on the amount of light and the maximum voltage available to it.

- We'll build code to report out a percent of intensity from 0% to 100%.

Step 4: Programming the Sensor

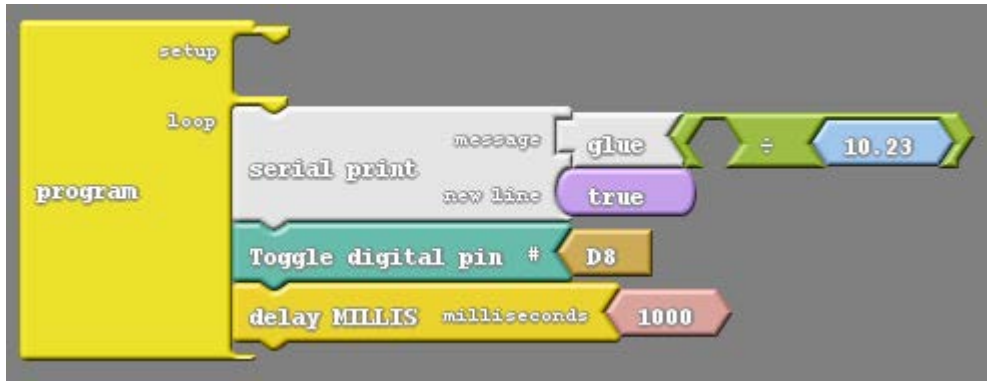
- Add a “serial print” from the Communication category.
- “Message” shows what string of characters are going to be sent (for now, through the **USB** cable).
- Pull off the extra “message” block.
- “New line” is asking if you want the message to end by moving to the next line.

Step 4: Programming the Sensor



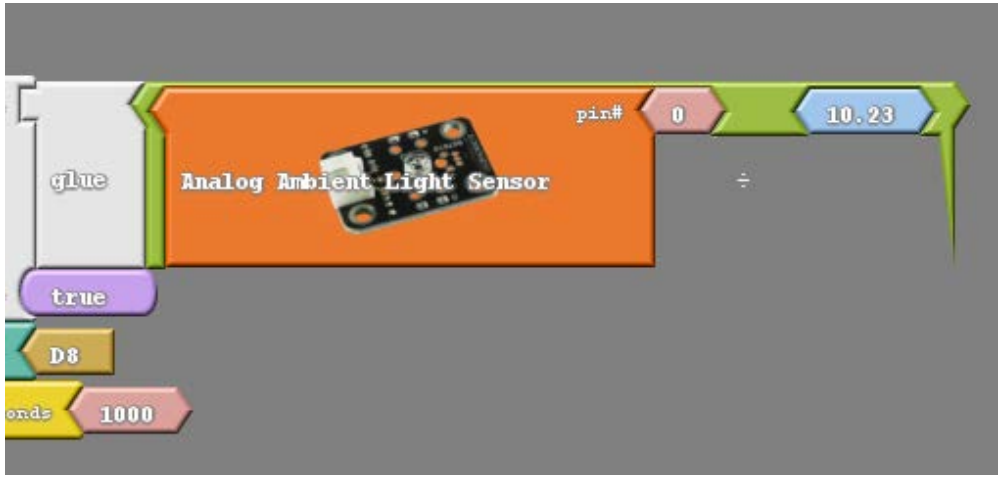
- Add a “glue” block from the Communication category.
- “Glue” allows you to send a number value as a series of characters in a message.

Step 4: Programming the Sensor



- Add a calculation block from the **Math Operators** category.
- If necessary, change the operation to "÷".
- Add a decimal value (starts as **3.1415927**) from the **Variables/ Constants** category.
- Change the value to **10.23**. This is the same as dividing by 1023 and multiplying by 100 to get a percent.

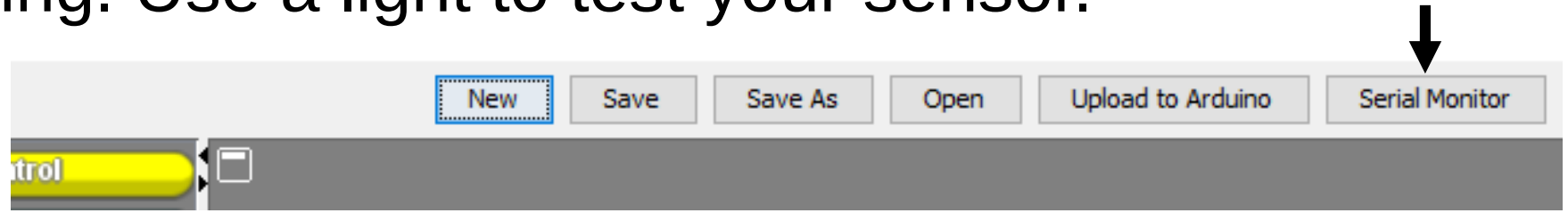
Step 4: Programming the Sensor

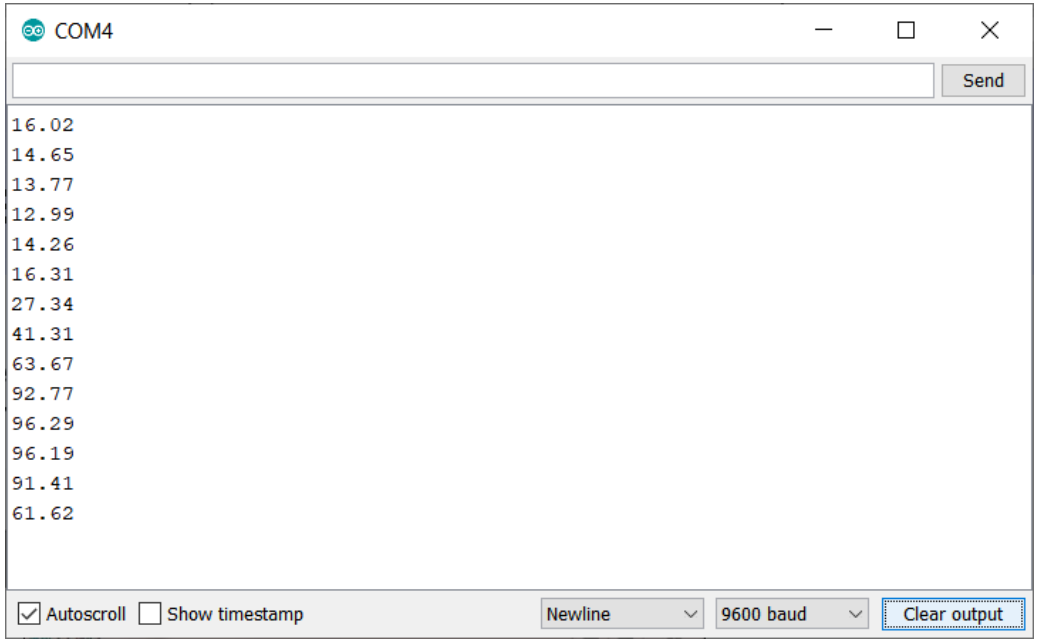


- Find the “Analog Ambient Light Sensor” in the **DFRobot** category. Connect it to the ÷ operation.
- Remember that we connected the sensor to analog pin# 0. Change the value if necessary.
- The program will now take the value from the sensor, and turn it into a percent.
  - No light = 0%
  - Full light = 100%

Step 4: Programming the Sensor Real-time Output with Serial Monitor

- Upload the latest program to your Arduino then click here to open the Serial Monitor. After a moment, numbers should start appearing. Use a light to test your sensor.

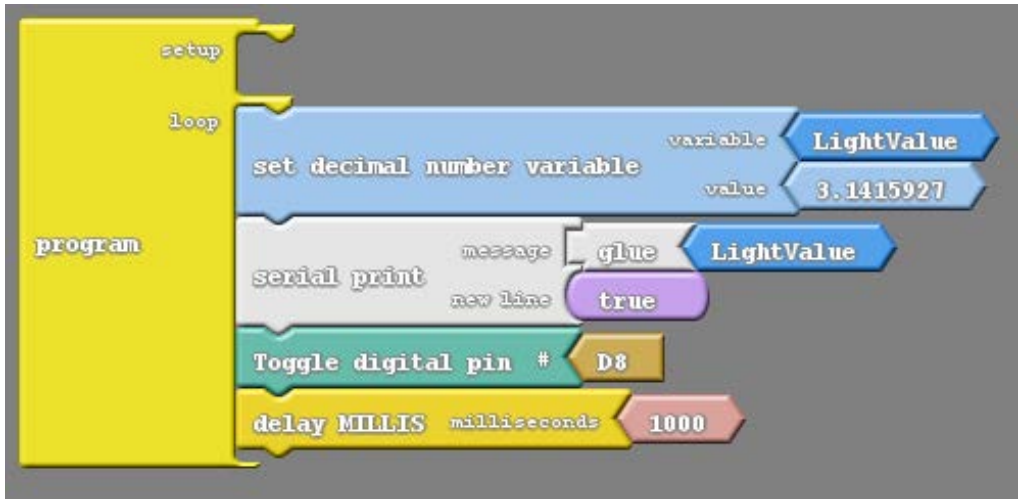




Step 5: Providing Visual Feedback

- We want the system to not just provide numerical data, but also a warning indicator.
- We'll test that out with the traffic light.

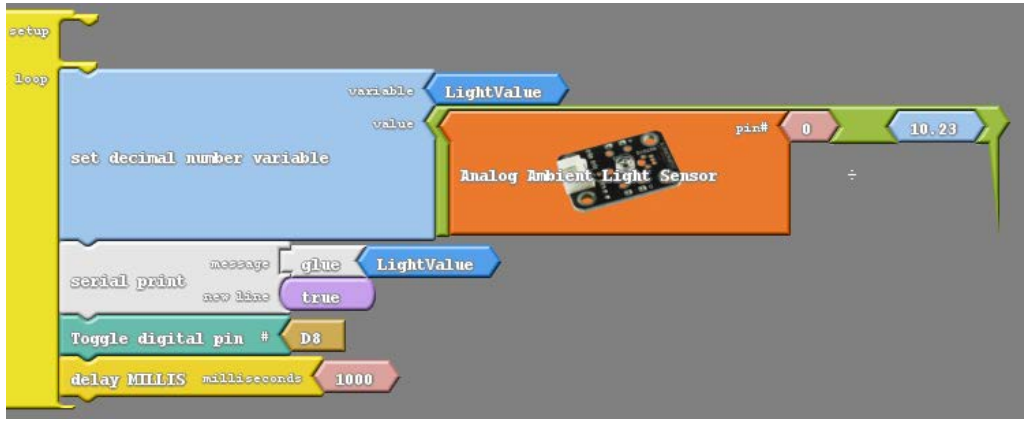
Step 5: Providing Visual Feedback



Since we will be using the sensor value for two things now (the displayed value and determining the traffic light), we'll store the value as a variable.

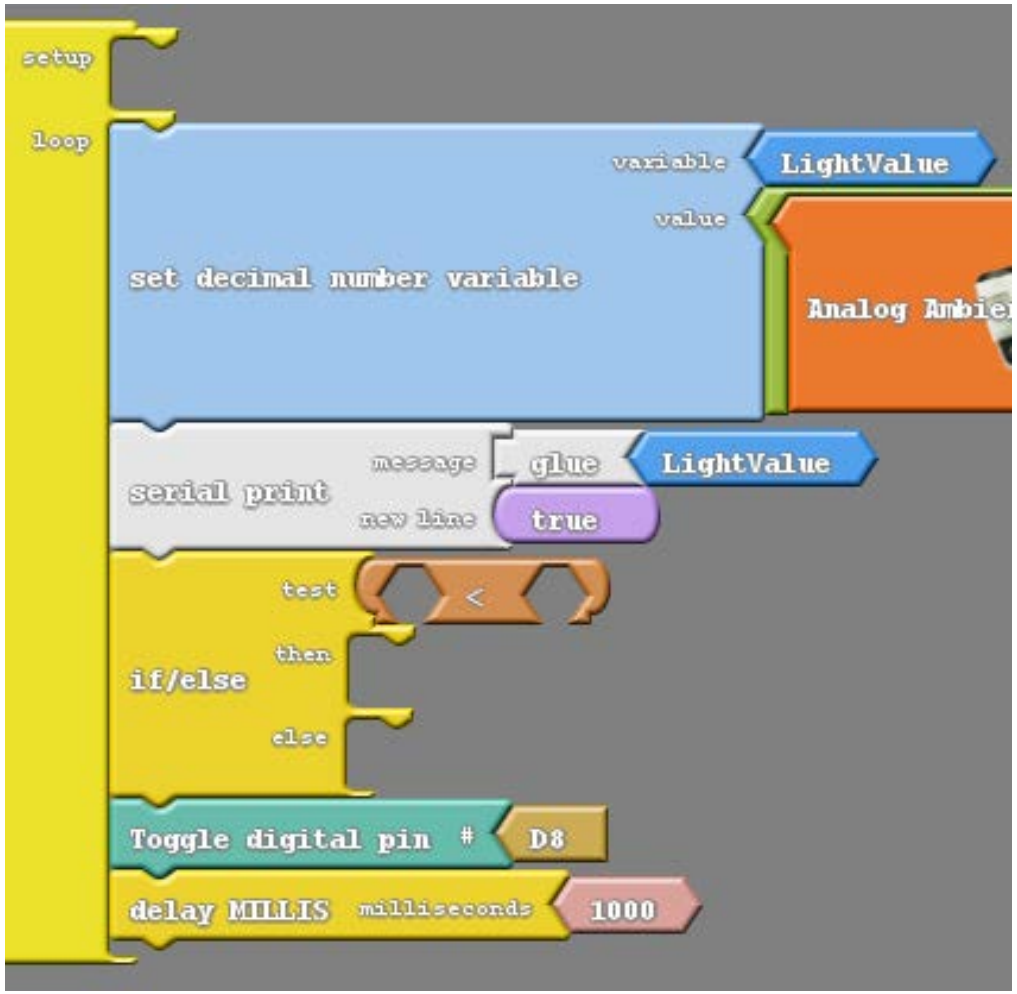
- Add a "set decimal variable" block from the **Variables/ Constants** category.
- Name it something meaningful like "LightValue"

Step 5: Providing Visual Feedback



- Pull off the default number value.
- Grab the blocks that take the light sensor data and convert it to a percent, and move them to the variable value.
  (Now we can use the same LightValue variable any time we need it!)
- Right click the LightValue block, clone the variable, and replace it on the “serial print” command.

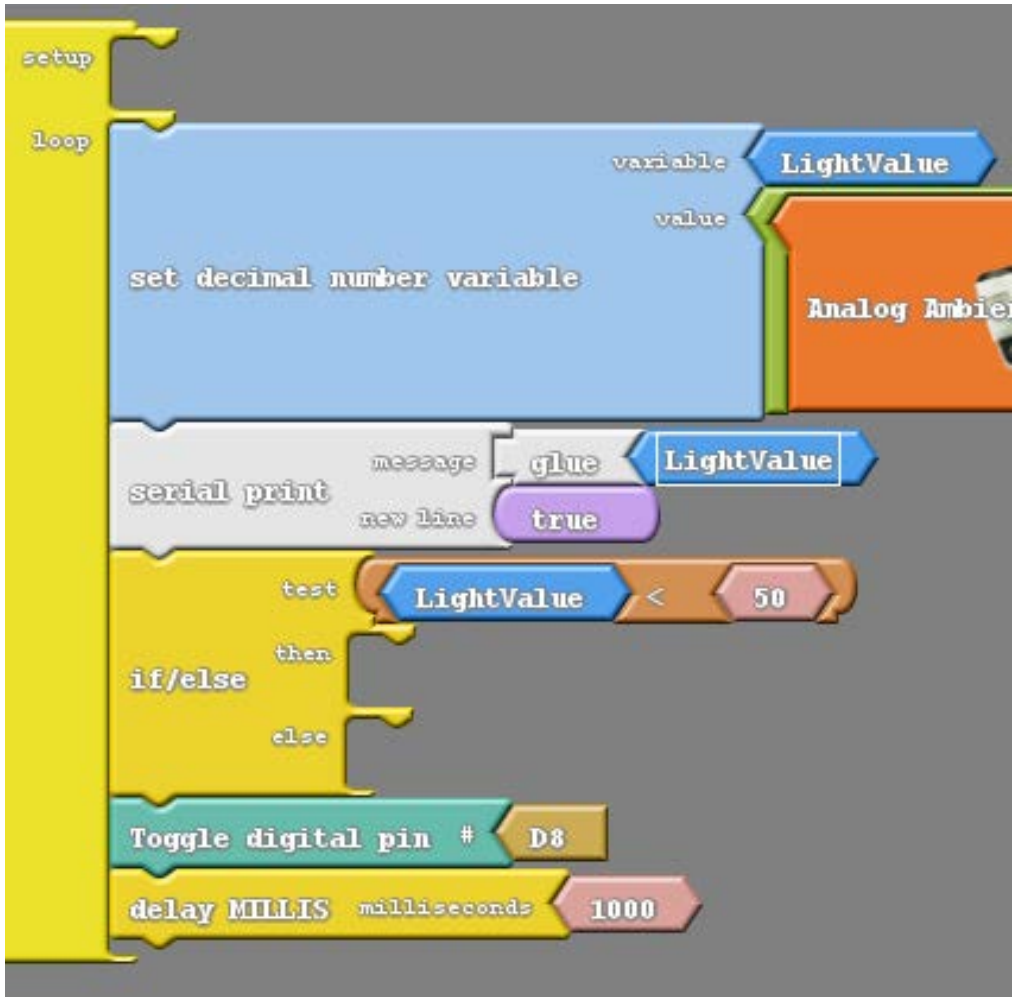
Step 5: Providing Visual Feedback



Next we will use an “if/else” statement to switch the traffic light based on LightValue.

- Add an if/else block from the **Control** category.
- Next, we will compare the LightValue and either shine red or green based on if it is above or below a given value.
- Add a “<” block from the **Tests** category.

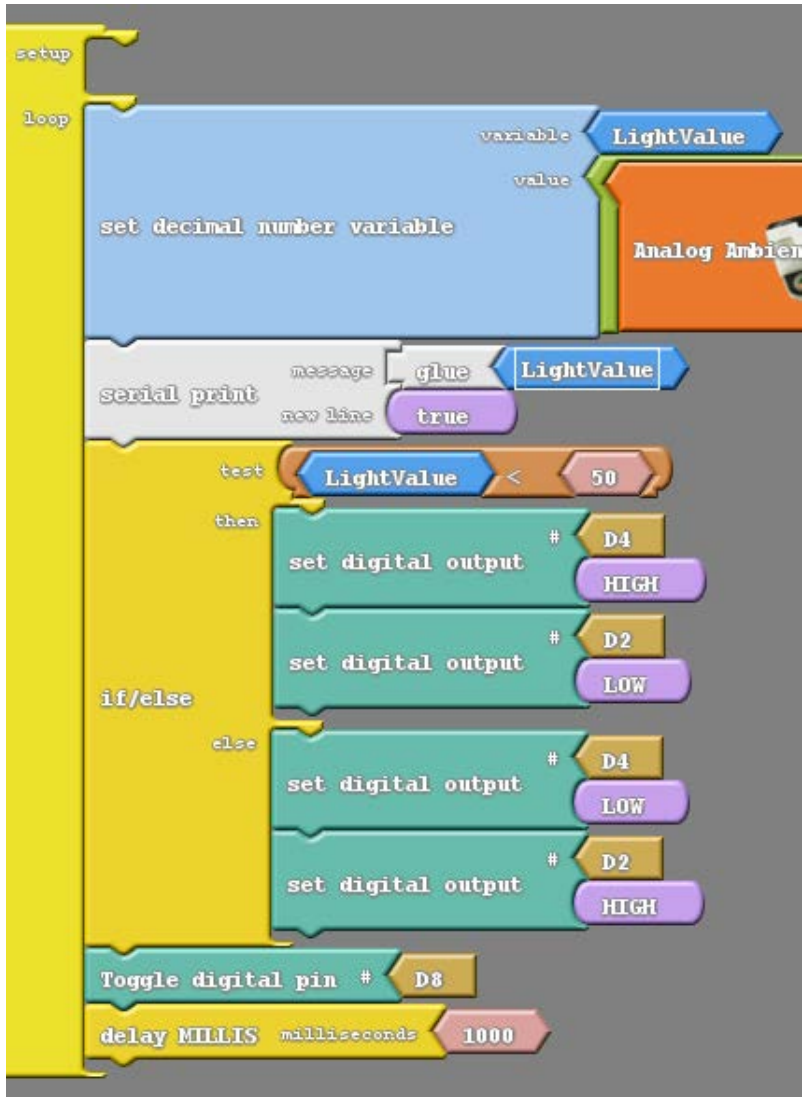
Step 5: Providing Visual Feedback



- Clone the LightValue variable again, and put in in the first test position.

- For now, get an integer constant from the **Variables/ Constants** category and put it in the second test position. Change the value to 50.
  - If LightValue is below 50, it will follow commands in the “then” area.
  - If LightValue is above 50, it will follow commands in the “else” area.

Step 5: Providing Visual Feedback



- Add two “set digital output” blocks from the **Pins** category to the “then” area.
  - The first block will be for pin **D4**. We will set it to “High”.
  - The second pin will be for **D2**. We will set it to “Low”
- Copy the two “set digital output” blocks from the “then” area into the “else” area.
  - Switch the pin outputs for the two pins.
- Save and upload the program to the Arduino.

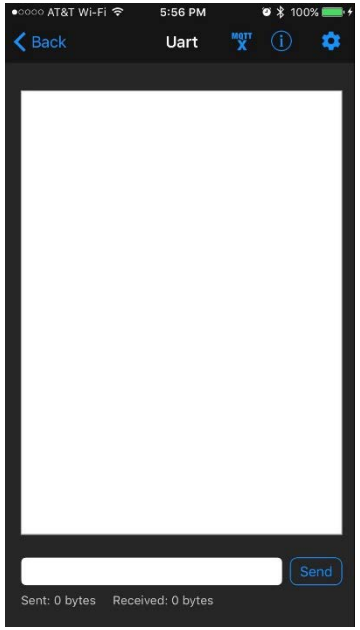
Step 5: Providing Visual Feedback

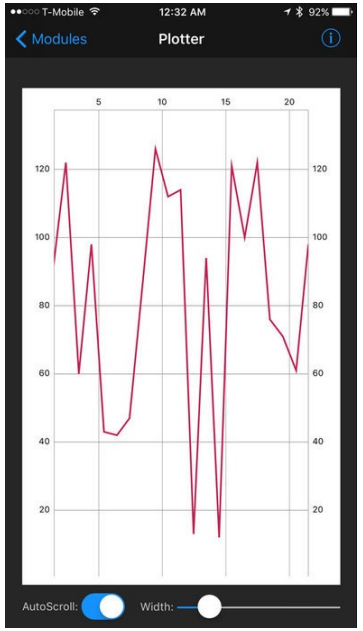
- Now your system will signal any LightValue under 50 as a green light and any LightValue 50 or over as a red light.

- Test it out with a light source.

Step 6: Adding the Bluetooth Radio
Install the Bluefruit Connect App

The main features you'll want to explore are UART (which works like the Serial Monitor in the Arduino program) and the Serial Plotter (which graphs data received from the Arduino, if formatted properly)

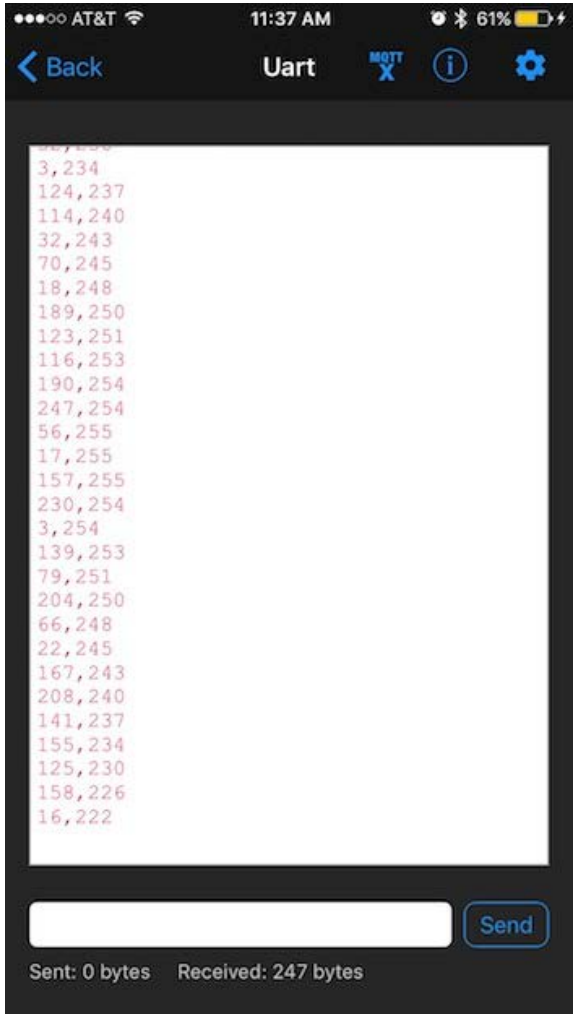




https://learn.adafruit.com/bluefruit-le-connect

Step 6: Adding the Bluetooth Radio

- Now you can receive the output from the light sensor straight on your phone!
- What if you wanted to change the value for making the traffic light switch?

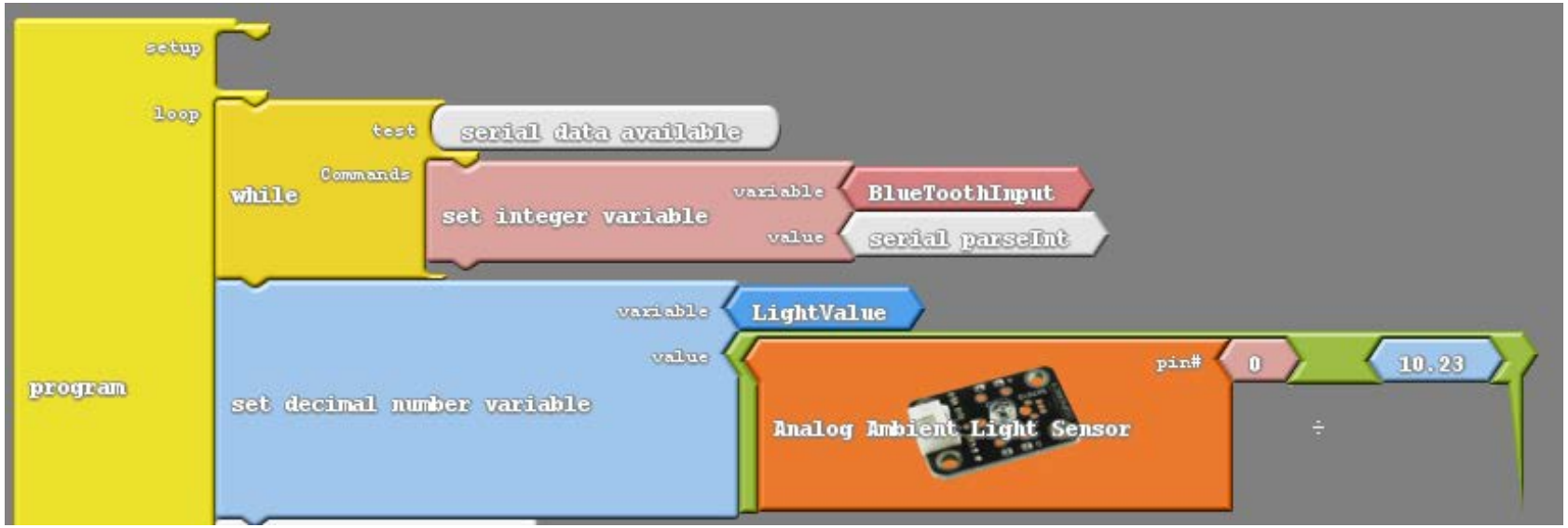


Step 6: Adding the Bluetooth Radio

Now we'll add the code to get input from the Bluetooth radio.

- Add a "while" block from the **Control** category. Use a "serial data available" from the Communication category. This will test if there is data coming from the Bluetooth radio.

- Add a "set integer variable" from the **Variables/Constants** category. Name the variable something helpful like "BlueToothInput" and the value must be set to the "serial parseInt" block.



Step 6: Adding the Bluetooth Radio

ASCII TABLE

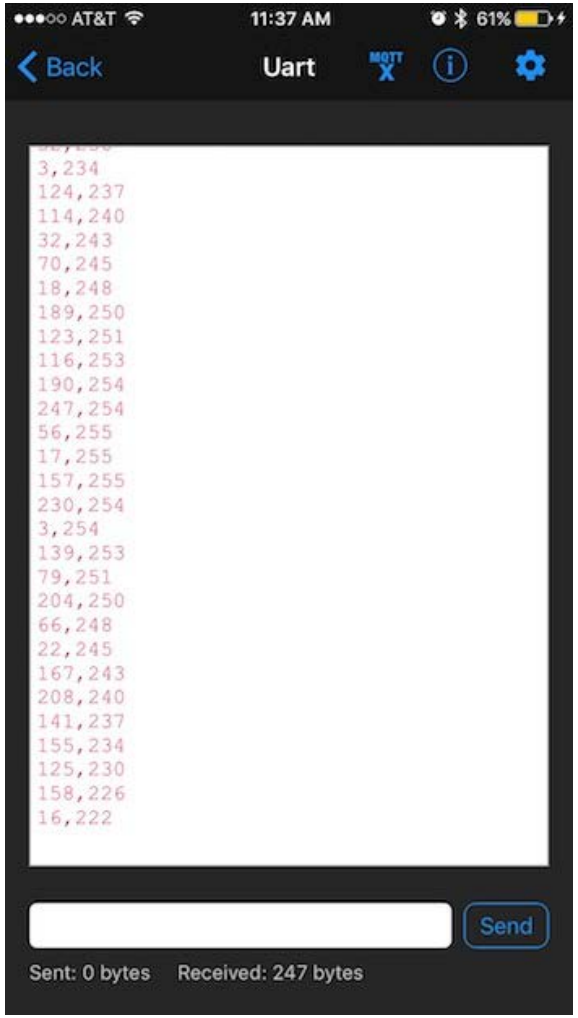
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

- Computers commonly use ASCII code to represent letters, numbers and symbols.
- The “serial read” block in Ardublock would see an input of “28” as “50” then “56” based on the decimal ASCII codes for 2 and 8, respectively.
- We need the “serial parseInt” block to read an input of “28” as an actual “28”.

Step 6: Adding the Bluetooth Radio

- By default, the serial input from your device will send a timeout “null” code after your last input.

- The computer will translate this to a “0”, so we need to account for that in the program.

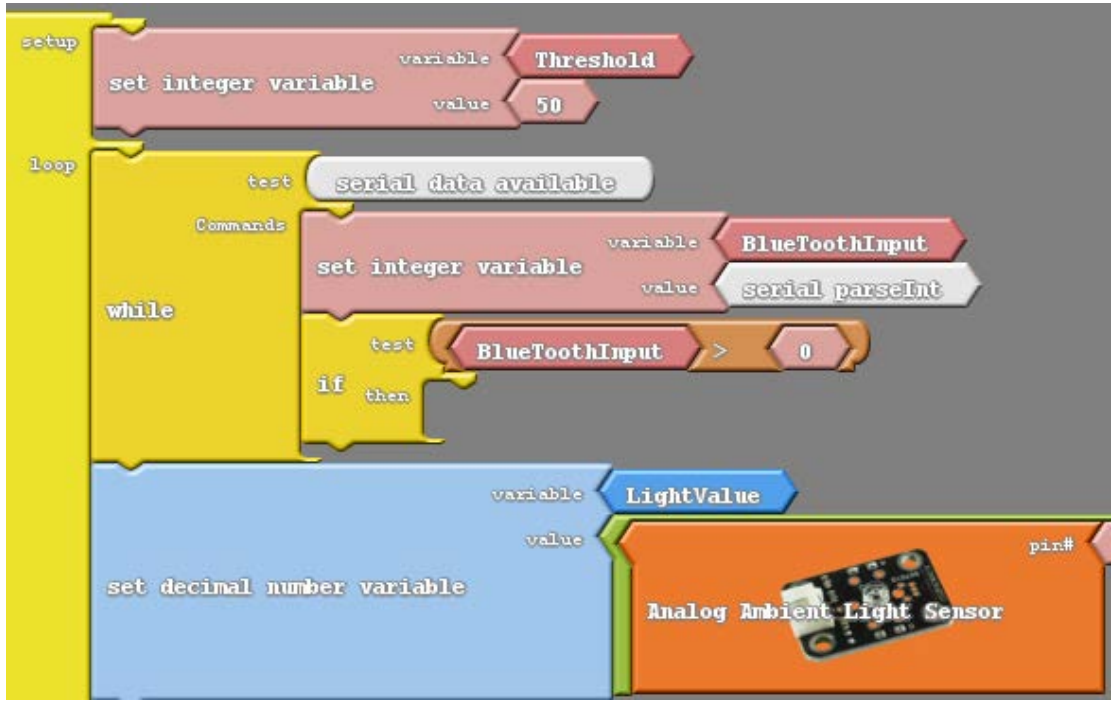


Step 6: Adding the Bluetooth Radio

We will start with a default Threshold of 50. Then, we test if BlueToothInput > 0. If it is, that will become our new Threshold. Otherwise, it will be ignored.

- Add a “set integer variable” from the **Variables/ Constants** category into the “setup” area at the top of the program.
- Name this variable “Threshold.”
- Assign it a value of 50. After the program begins, we will allow that value to change.

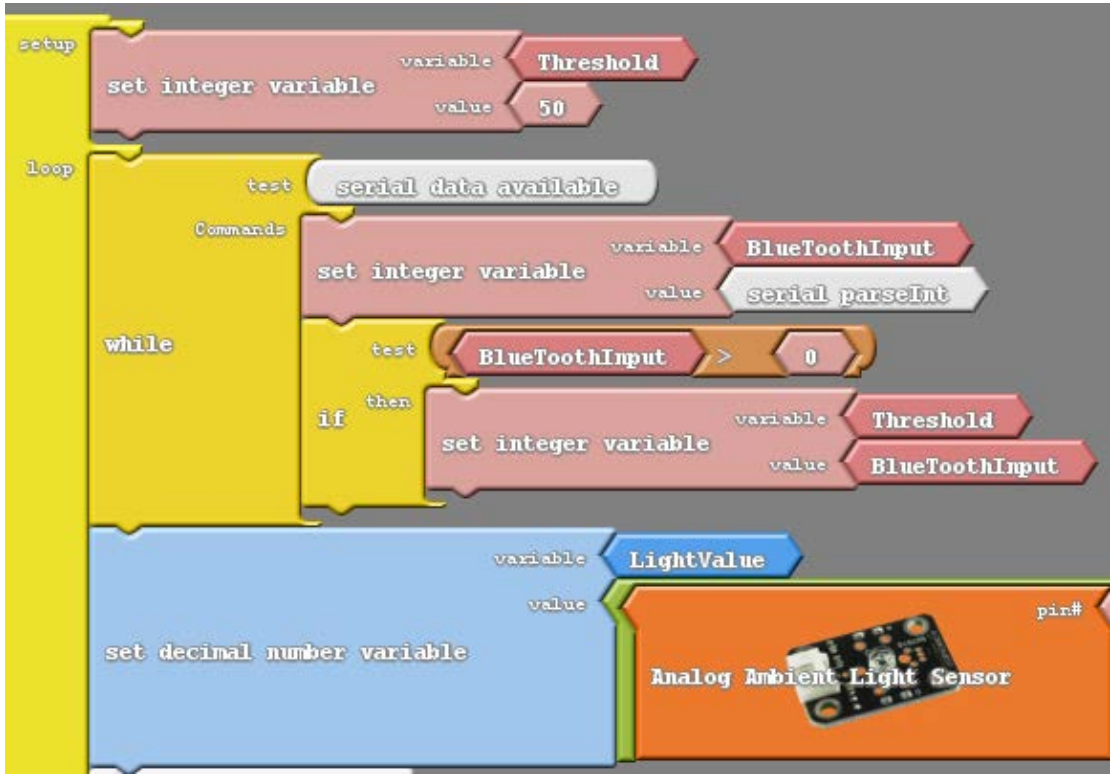
Step 6: Adding the Bluetooth Radio



We will start with a default Threshold of 50. Then we test if BlueToothInput > 0. If it is, that will become our new Threshold. Otherwise, it will be ignored.

- Add an “if” block from the **Control** category.
- Add a “>” block from the **Tests** category.
- Clone the BlueToothInput variable on the left side of the test.
- Add a “0” constant on the right side for the test.

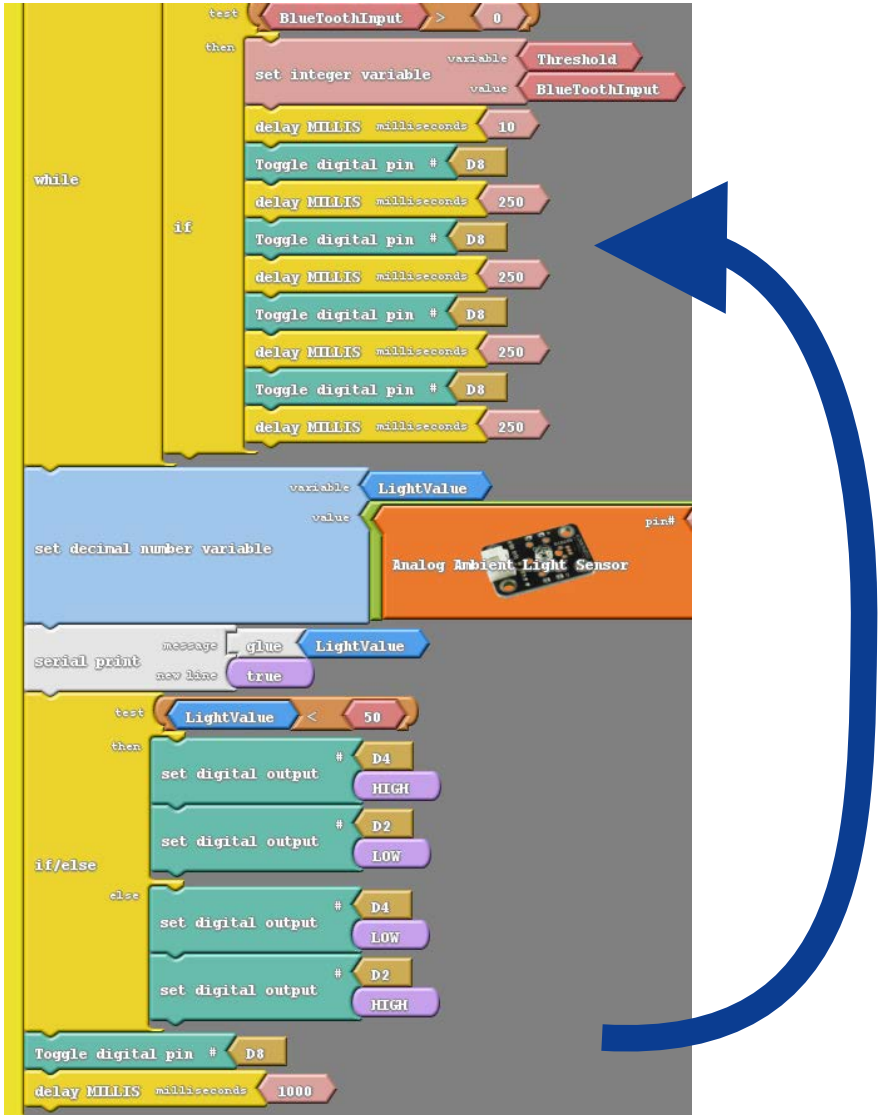
Step 6: Adding the Bluetooth Radio



We will start with a default Threshold of 50. Then we test if BlueToothInput > 0. If it is, that will become our new Threshold. Otherwise, it will be ignored.

- Add a “set integer variable” from the **Variables/Constants** category into the “then” area.
- The new variable will be called “Threshold”. The value will be a cloned “BlueToothInput”.

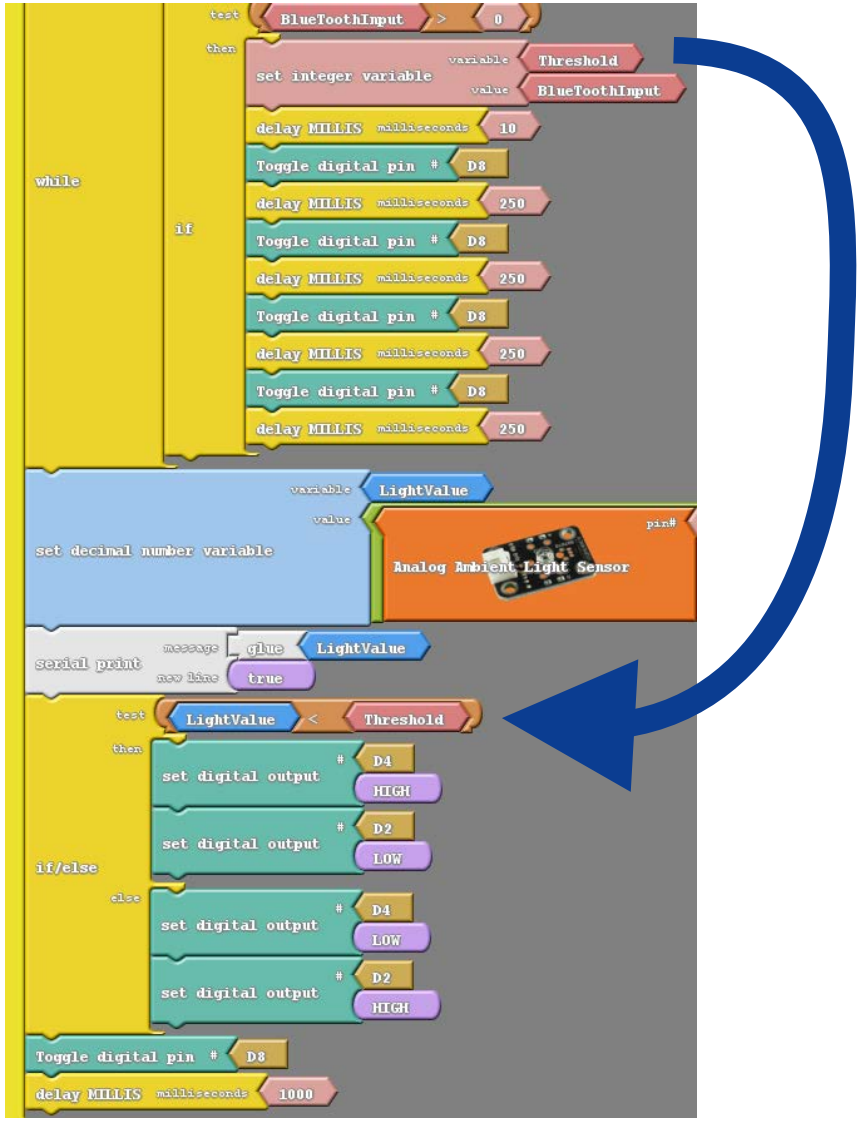
Step 6: Adding the Bluetooth Radio



It's helpful to get a message that the input was received. Let's use our green LED to blink twice quickly when it gets new data.

- Right click the "Toggle Digital Pin" command to clone it. It will clone the delay command as well.
- Change the delay from 1000 milliseconds to 250.
- Clone it 3 more times. Move them all into the "while" block.

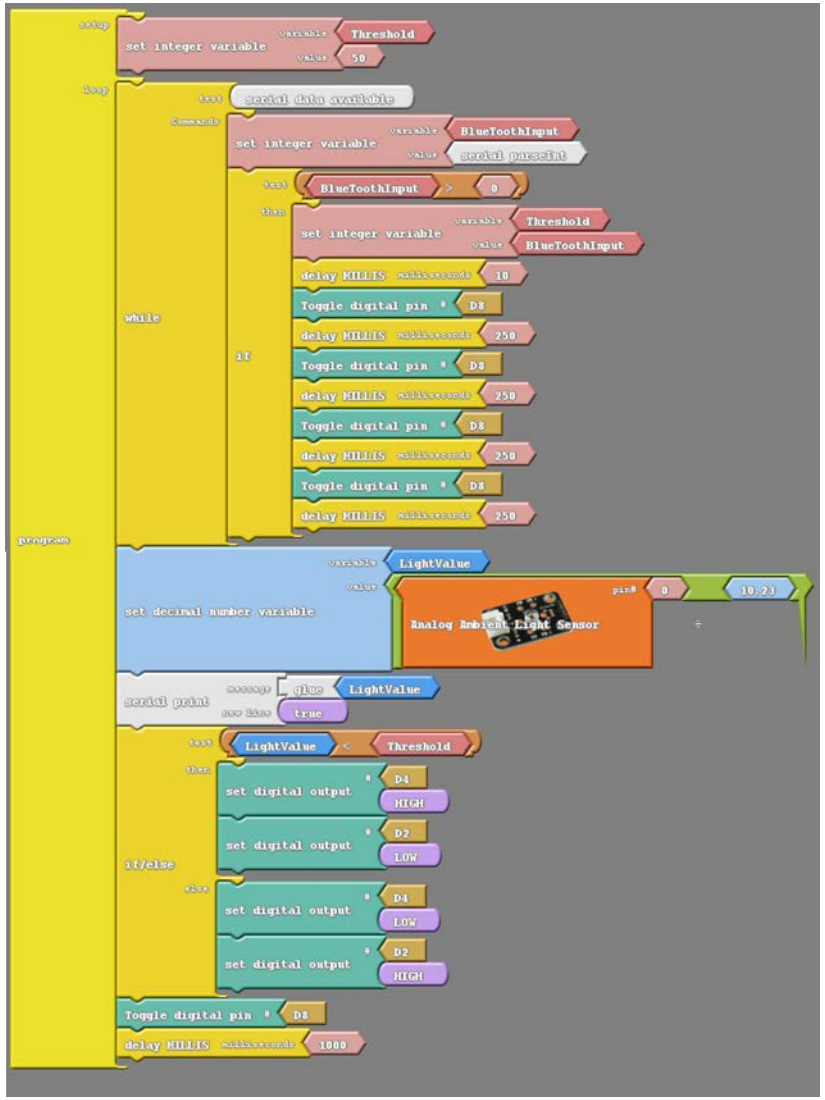
Step 6: Adding the Bluetooth Radio



Finally, we'll use the newly stored "Threshold" variable to change the traffic light threshold.

- Right click the "Threshold" variable to clone it.
- Replace the 50 integer with our variable.

Step 6: Adding the Bluetooth Radio

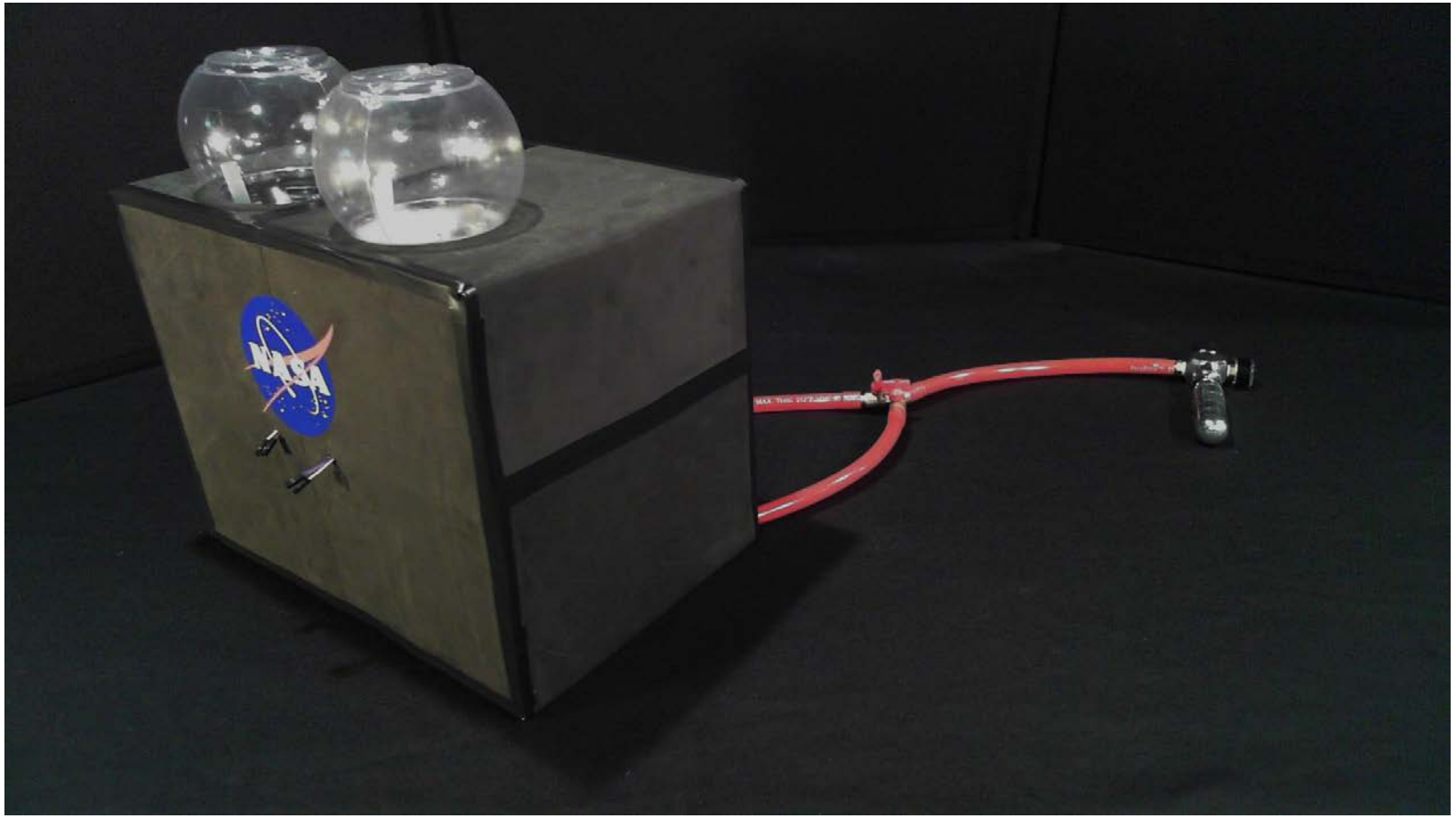


Congratulations! Your light sensor program is complete.

- Now that you have the basic sensor assembled and programmed, it will be up to the students to determine how to use multiple LEDs to show different sensor values.

  - How can the light sensor system be converted to CO2 sensors instead?
  - Which lights on the traffic light will be used?
  - What values could be considered too high or too low?
  - What will the data output look like?

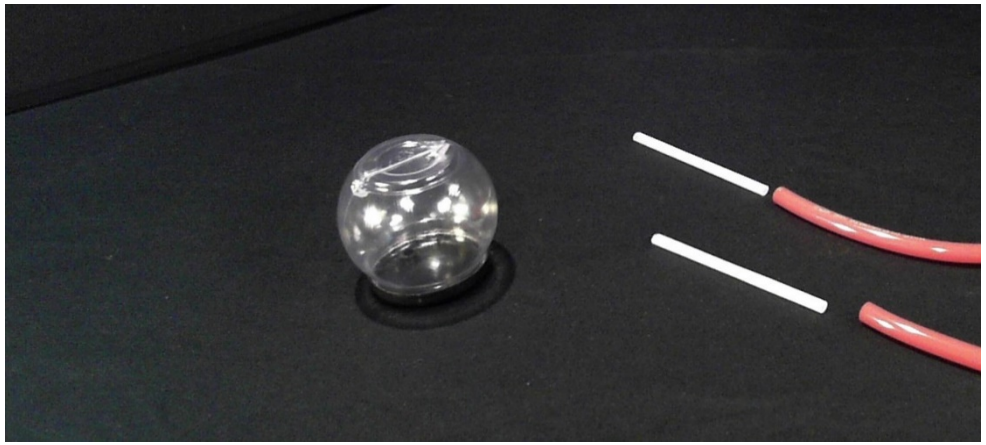
Building the CO_2 Test Apparatus



Building the CO_2 Test Apparatus

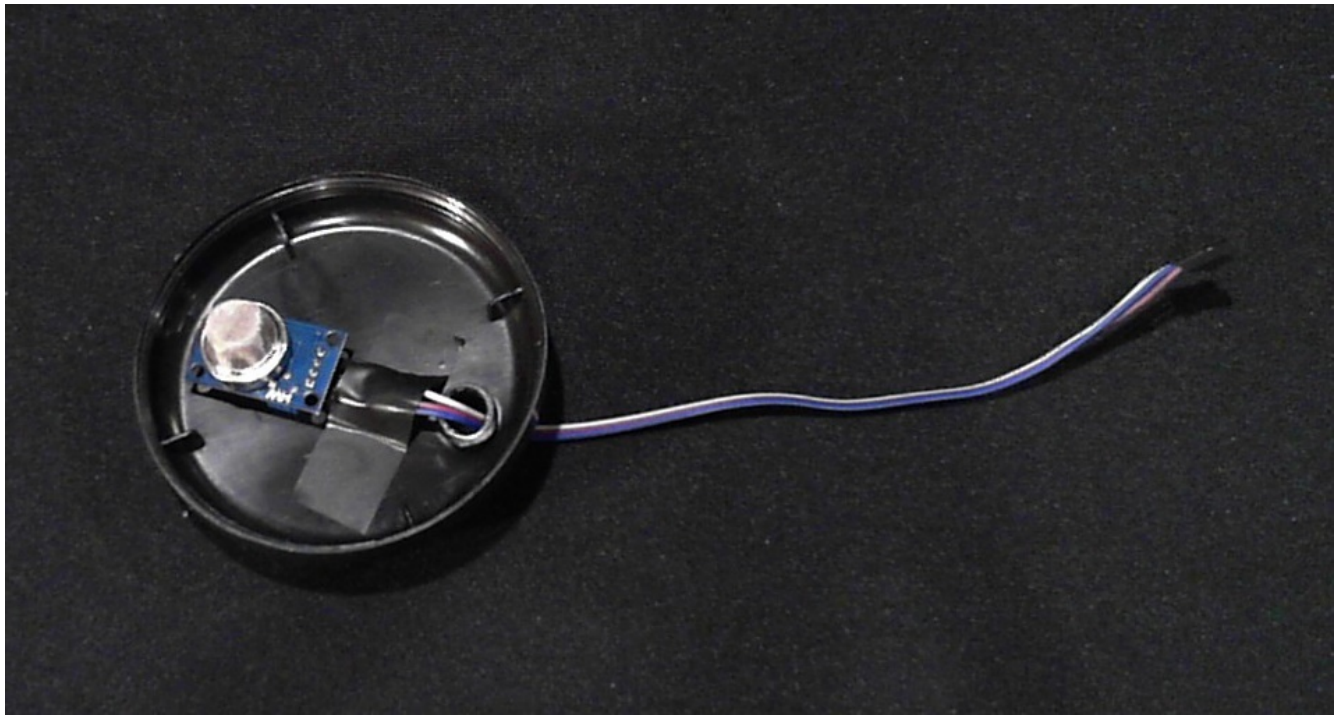
1. Connect CO_2 Tubing.

Distributor
|
1 ft. Tube
|
Y ball Valve
/ \
2 ft. Tubes
| |
Helmet Straws



Building the CO_2 Test Apparatus

2. Use electrical tape to attach a CO2 sensor to the bottom of each helmet “lid”. Slide the wires through the straw hole.

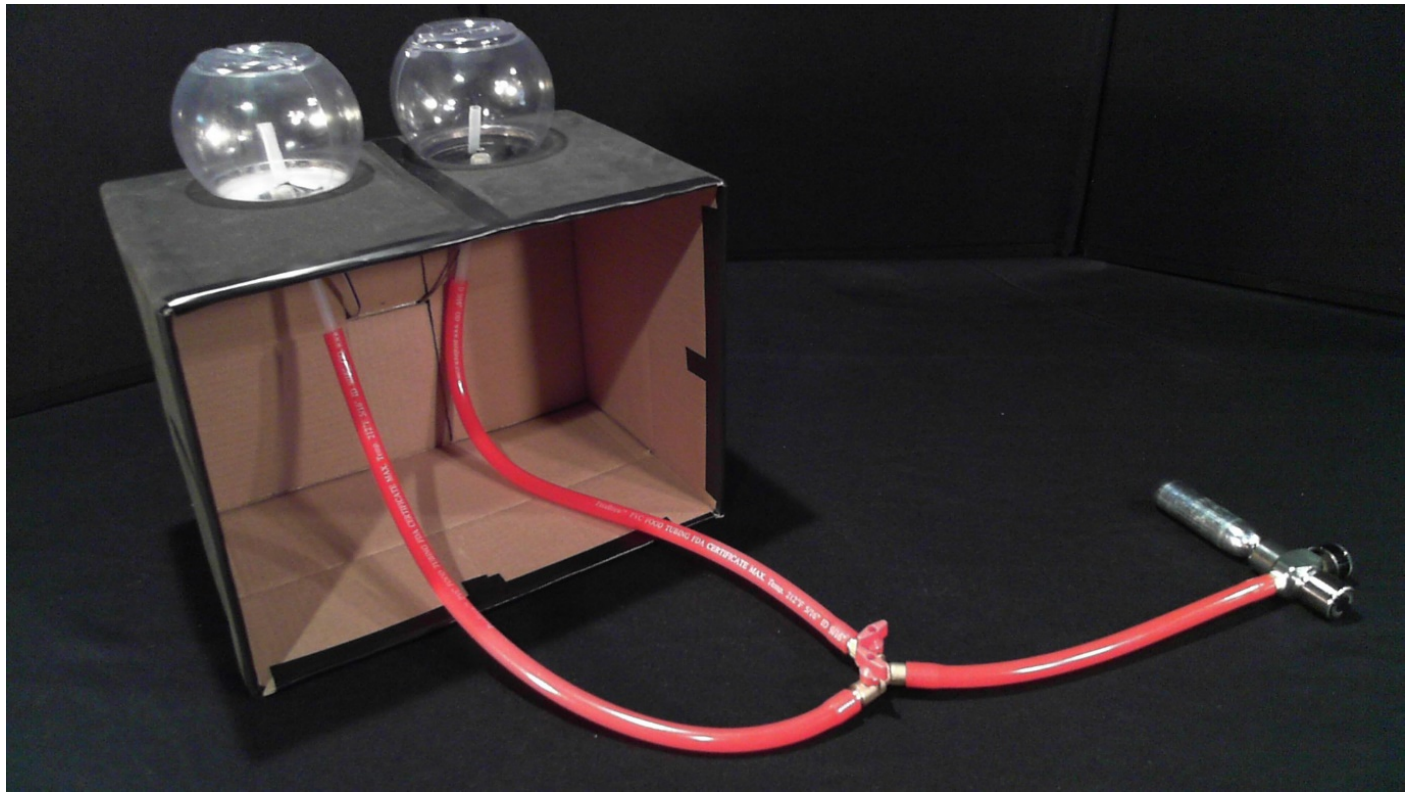


Building the CO_2 Test Apparatus

3. Reinsert straws back into helmet. Use care sliding along the wires.

Building the CO_2 Test Apparatus

4. You can use a regular box to hold up the helmets and conceal which ones are getting CO_2.



Building the CO_2 Test Apparatus

5. Poke holes through the box for the wires of each sensor to connect to the microcomputer.

Building the CO_2 Test Apparatus

6. Use the Y ball valve, to open or close the flow to each helmet.

- Knob facing the same way as the tube = open
- Knob facing across the tube = closed

Building the CO_2 Test Apparatus

7. Make sure the distributor is completely **closed**. Add a CO_2 cartridge to the distributor. Screw until the cartridge is tight.

   This will poke a hole in the cartridge seal. Once the cartridge is started, it cannot be resealed.

Building the CO_2 Test Apparatus

8. When the system has been connected to the helmets and is up and running, **slowly** turn the distributor knob to “increase”. A little flow should cause a well-calibrated system to spike very quickly.

Evolve

- **I tried something.** *How do I evolve it?*

- Look at your prototype, both the hardware and the code. Think about how it will evolve to operate CO2 sensors instead.
  - What parts you want to keep?
  - What parts can be improved?
  - Should anything be added?
  - Should anything be removed?
- Discuss your plan with other creators. What do they think could be improved?

Evolve

• **I tried something.** *How do I evolve it?*

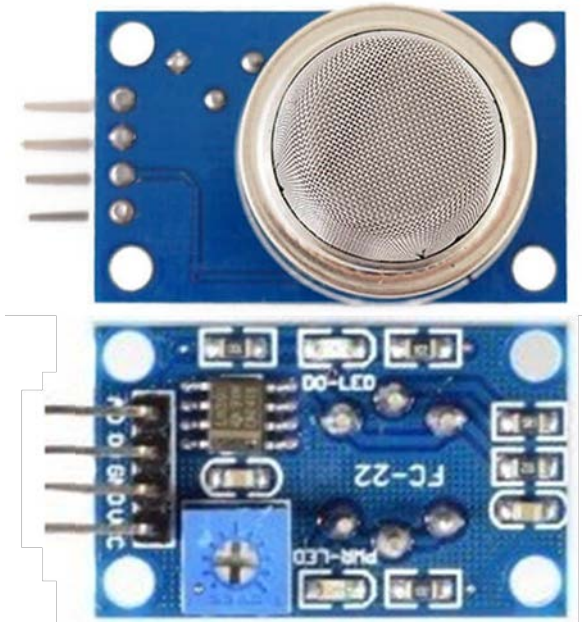
- Make changes and take pictures of each finished model. These are called **iterations**.

- Your pictures should show how your model improved from the first to the last iteration. You will use these pictures on your presentation board

Step 7: Add a Carbon Dioxide Sensor

- It has 4 pins (we will use 3):
  - **AO** – Analog signal output
  - **DO** – Digital signal output
  - **GND** – Power out
  - **VCC** – Power in

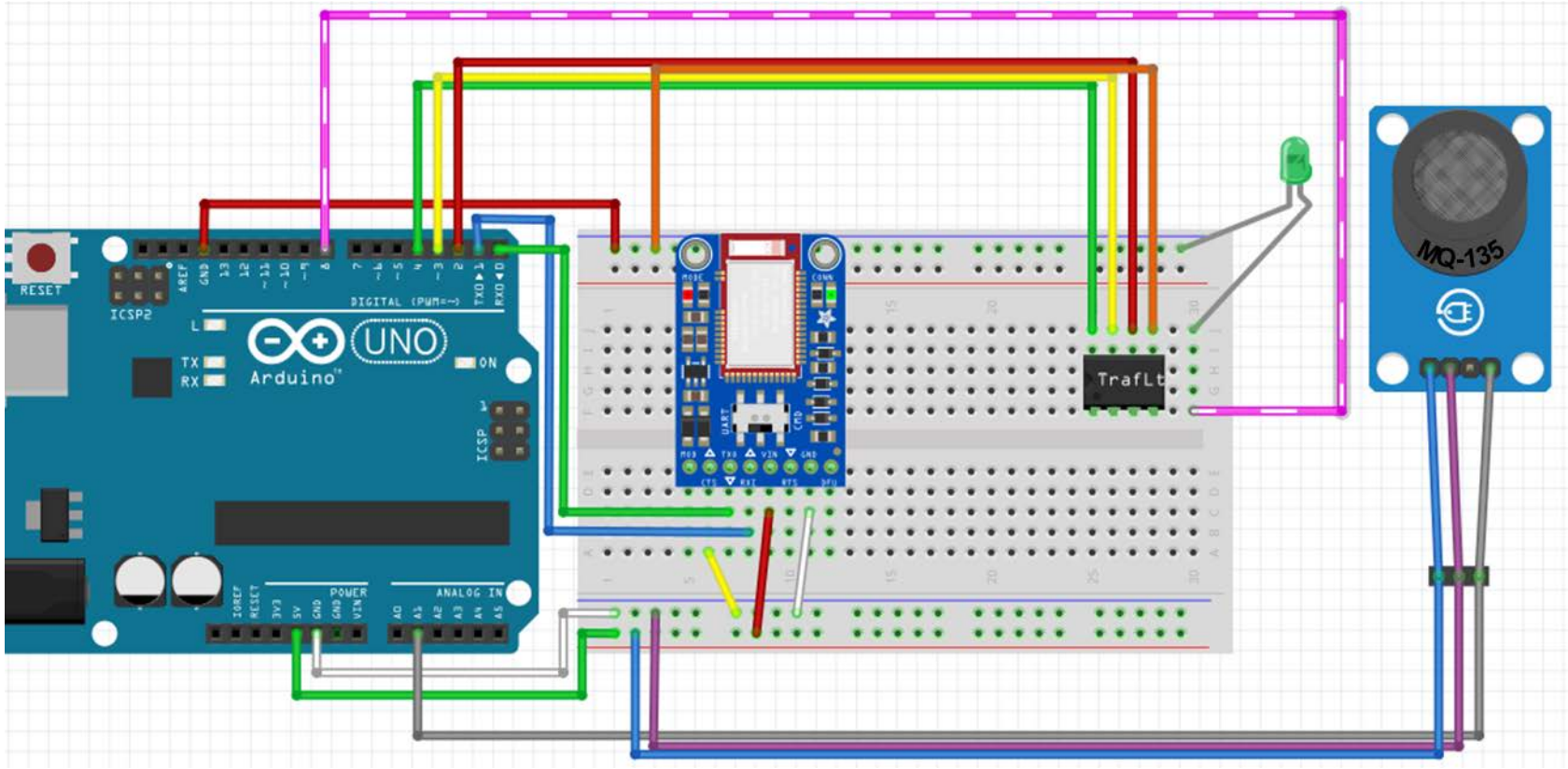
  We will only use the Analog signal output for our system



- As the air quality around the sensor changes, it sends a different amount of voltage out of the **output** pin. We can convert this voltage to a value of parts per million.

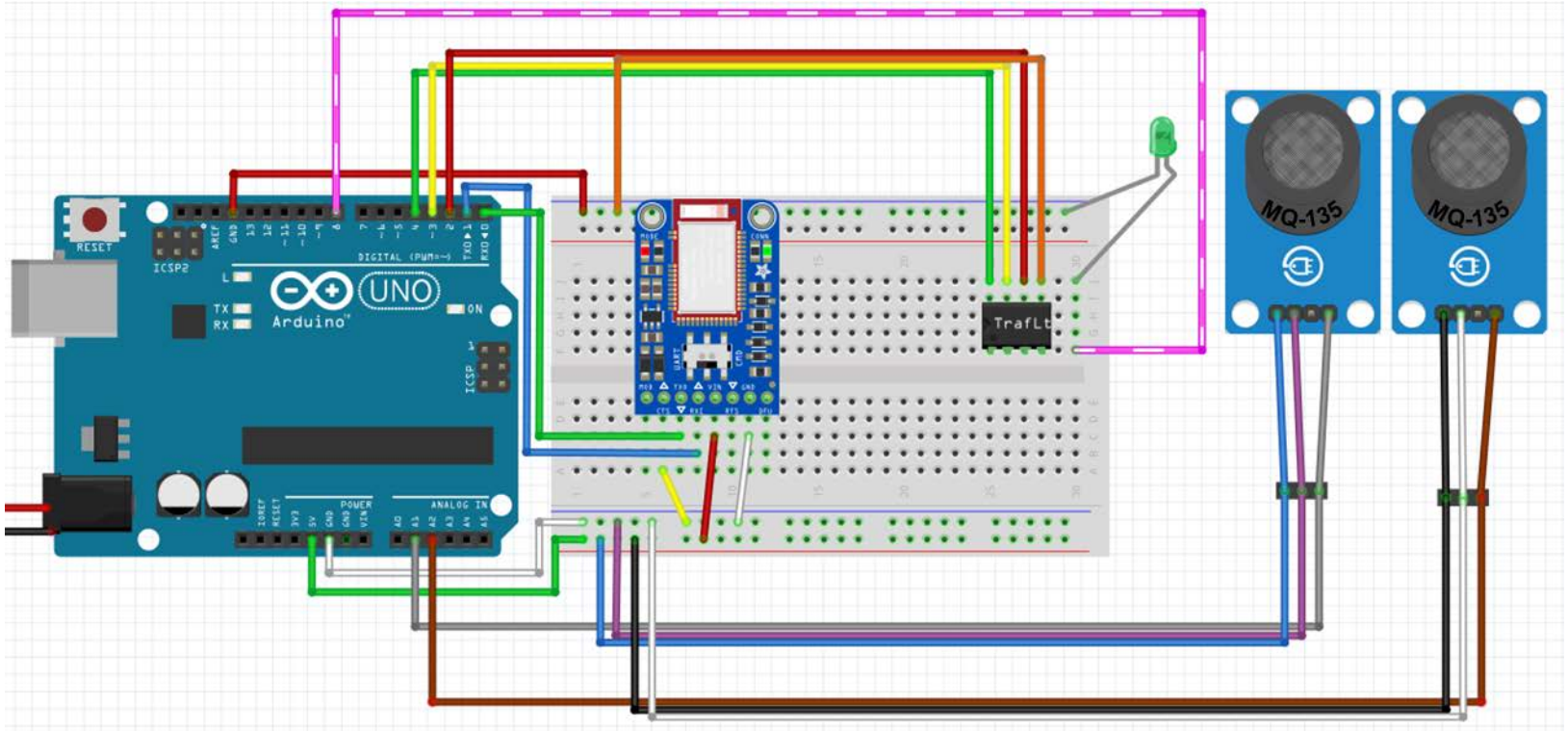
Step 7: Add a Carbon Dioxide Sensor

Remove the light sensor and its ribbon. Replace it with an MQ-135 sensor and a matching color ribbon. Change the signal wire to pin **A1**.



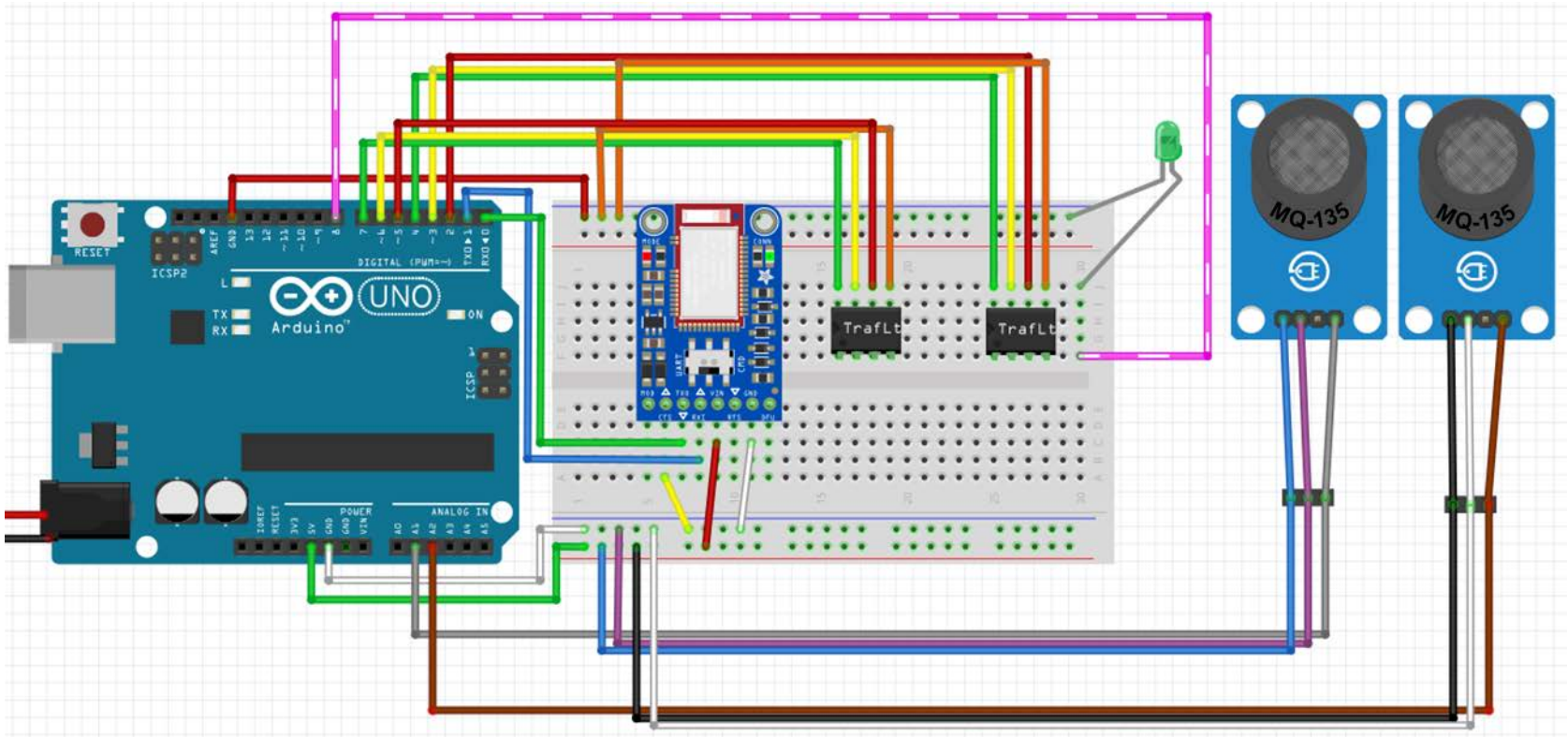
Step 8: Add a Second Carbon Dioxide Sensor

Connect a second carbon dioxide sensor to the **5V** and GND connections on the breadboard. Connect the second signal wire to pin **A2**.

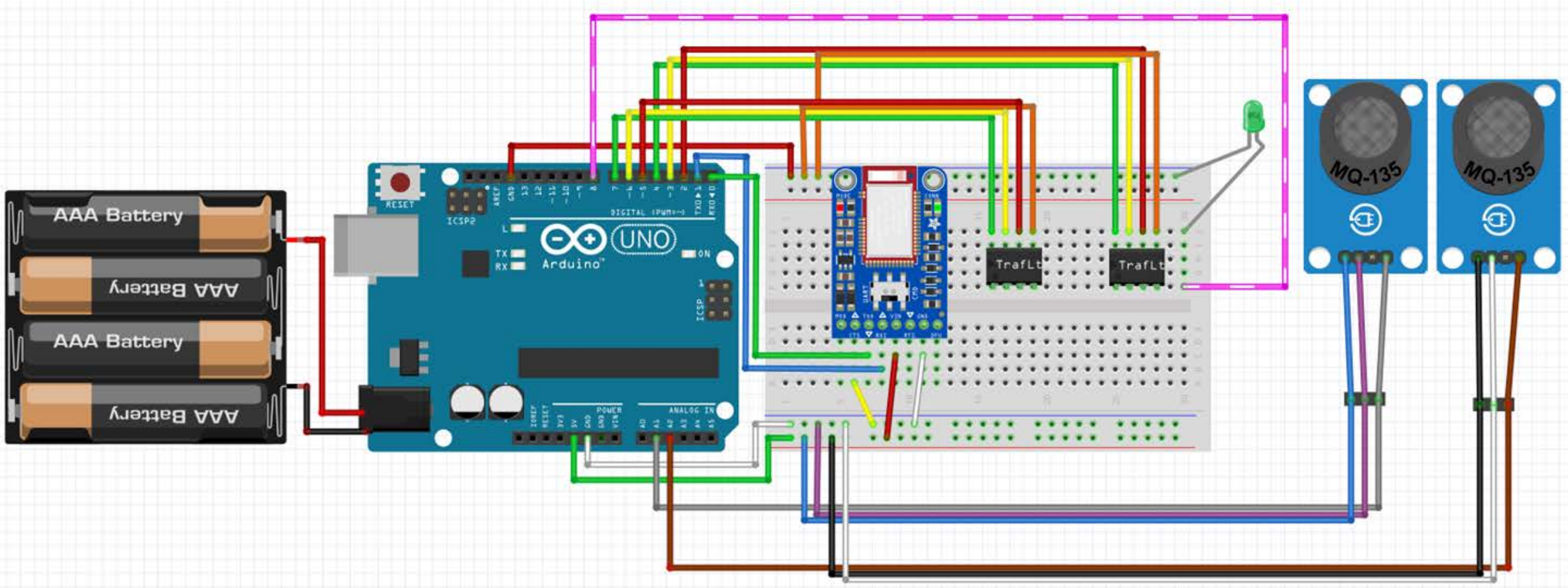


Step 9: Add a Second Traffic Light

Connect the Red, Yellow, Green pins using a four-wire ribbon to **D5**, **D6**, and **D7** respectively. Use the fourth wire to connect the **GND** pin to the GND on the breadboard.



Complete Wiring Diagram



Using Computational Thinking to Enhance the System

Students can use computational thinking skills to expand the original light sensor system to modify the system for multiple CO_2 sensors.

- **Decomposition** - Breaking problems down into smaller parts
- **Pattern recognition** - Iteration, symbolic representation, and logical operations
- **Abstraction** - Logically organizing and analyzing data, removing unimportant details to make problems easier
- **Algorithm design** – determining the appropriate steps to create a set of instructions that solves similar problems the same way.

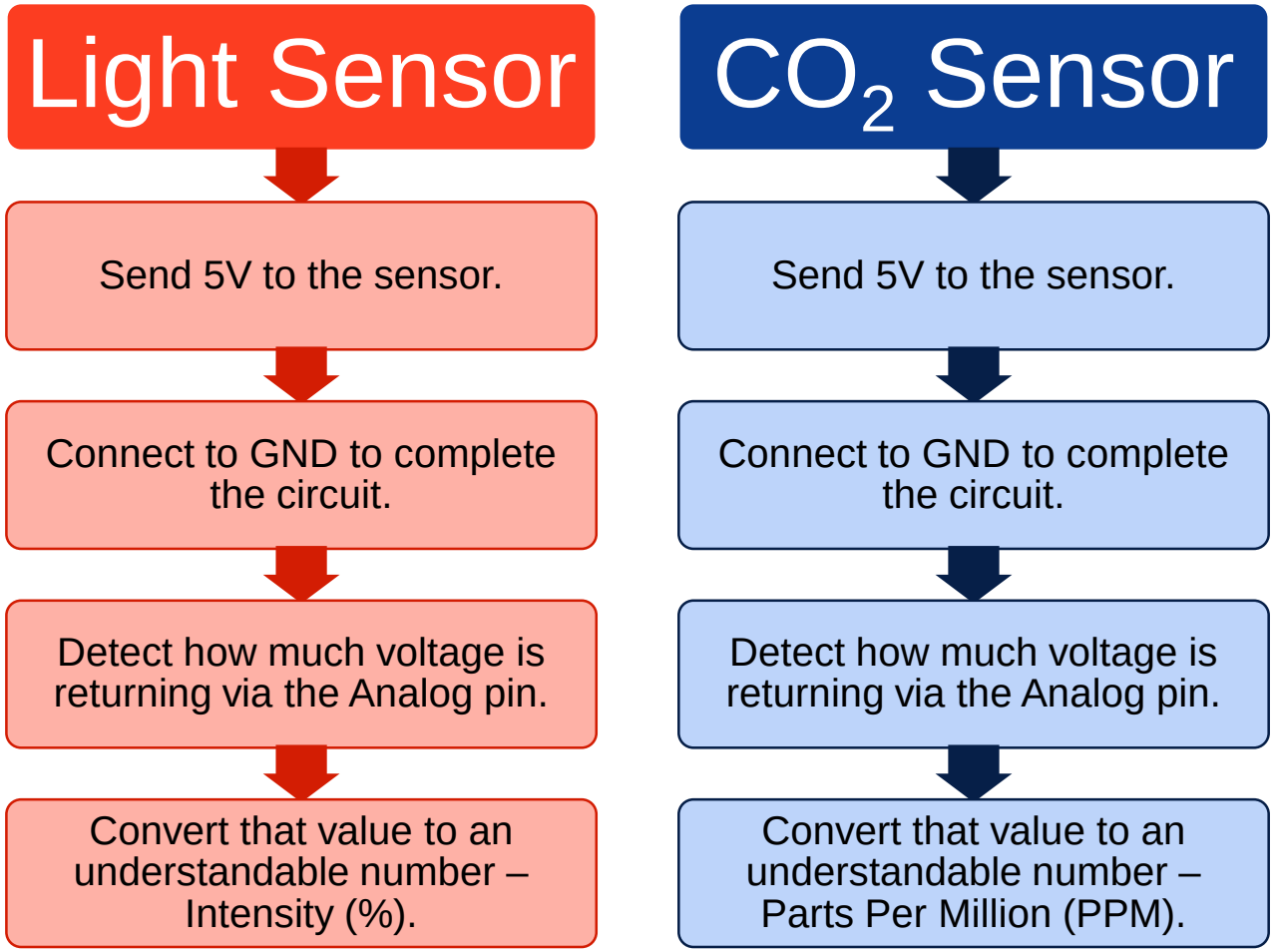
Decomposition

What are the major steps for the light sensor problem?

1. Initiate the system
2. Set up the sensor(s)
3. Determine the threshold for alert
4. Collect data
5. Convert data to meaningful measurements
6. Transmit the measurements
7. Send alerts as necessary
8. Repeat steps 3-7

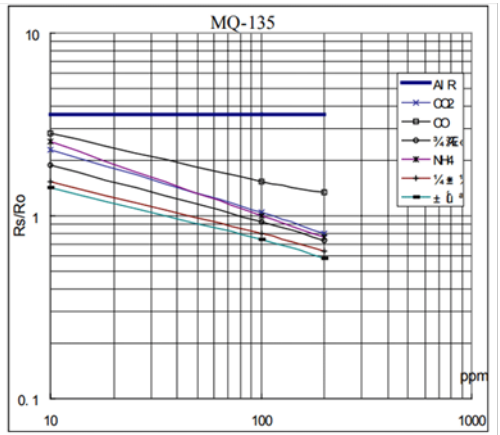
Pattern Recognition

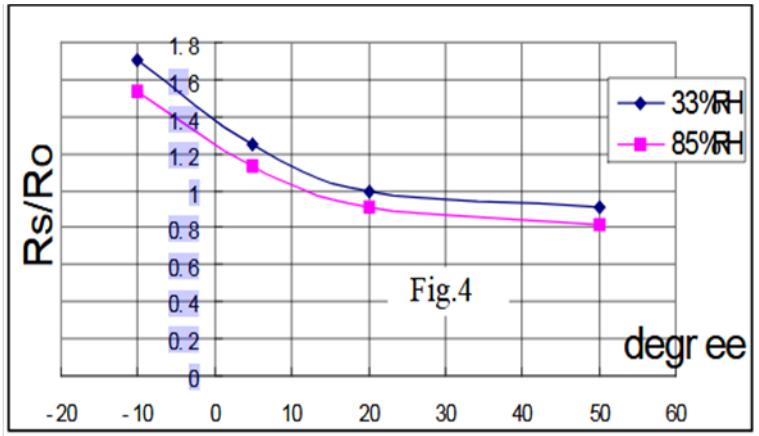
To get sensor data:



Abstraction

- There are some rather complicated mathematics, such as logarithmic functions, to get the CO_2 sensor data to make sense as a PPM readout.

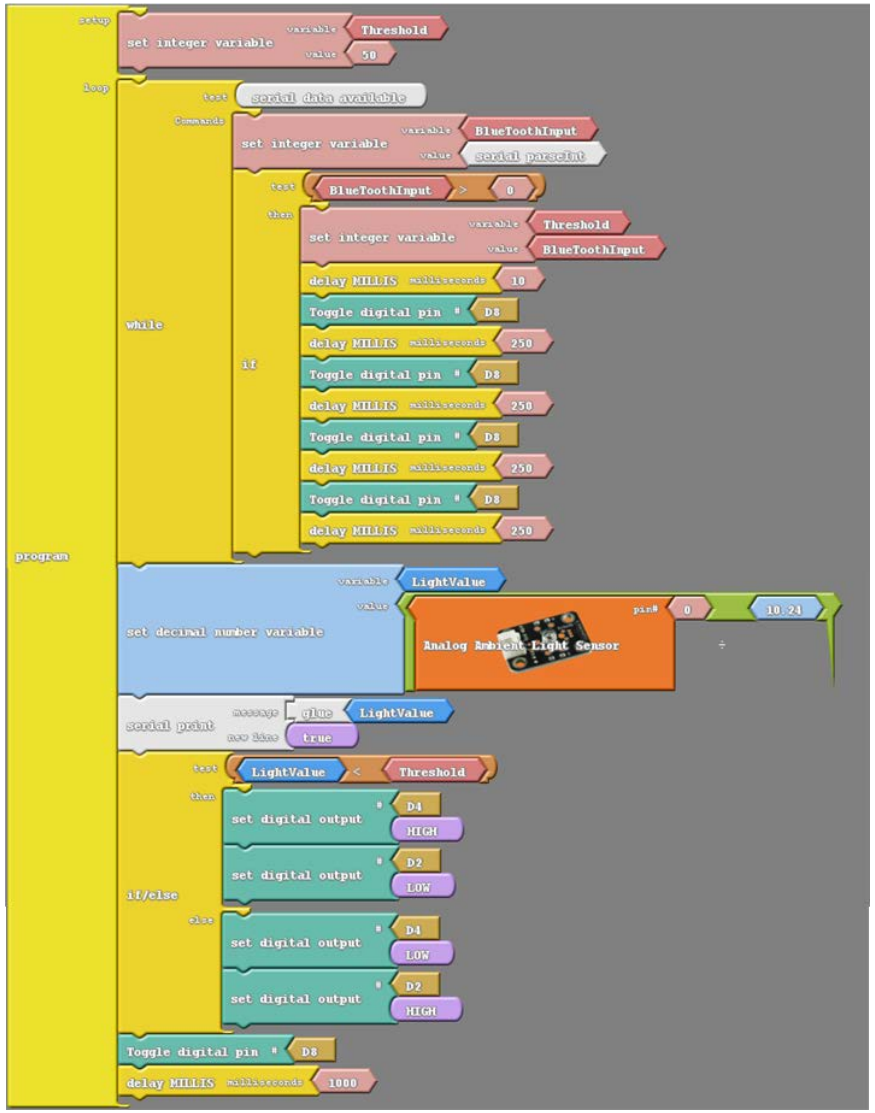




- These are beyond the scope of typical 5th-9th grade students. So we will simplify the process down and provide code that will do the calculations and return a relatively accurate value.

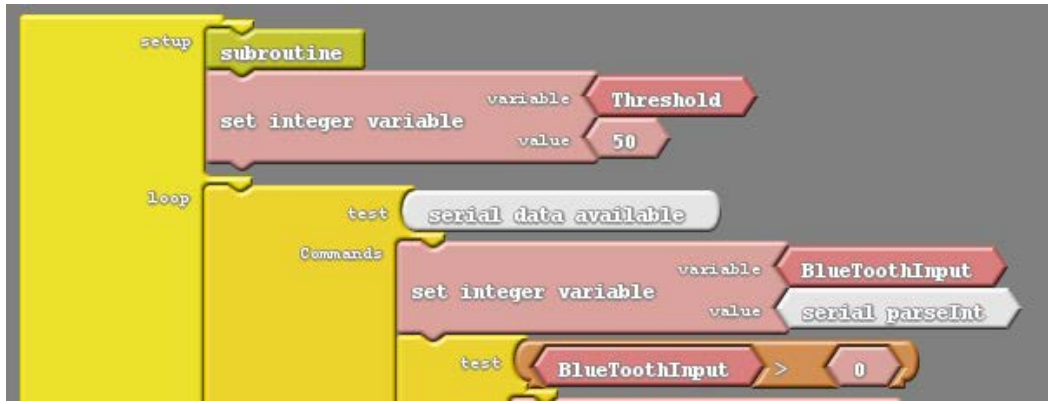
Do not use these systems to measure air quality for human safety!

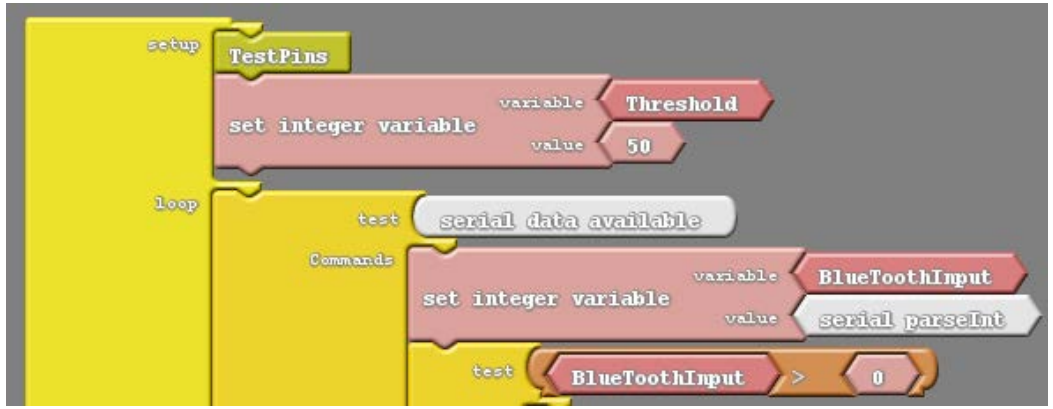
Algorithm Design



- Next, we will return to our block program and modify it to acquire CO_2 data instead.
- Until this point, every command was in a single sequential line. This was ok for a simple program, but as we get more complex, it will get more confusing.
- We need a way to break down our algorithm. We need **subroutines!**

Step 7: Testing the Traffic Lights

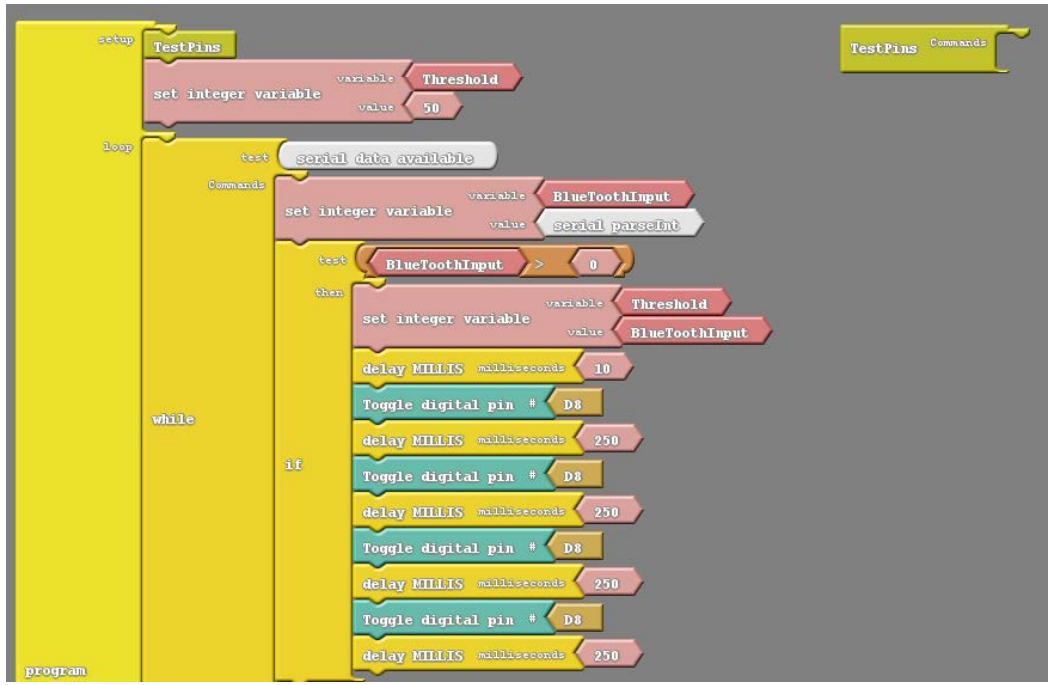




We'll start with a simple one. Let's test all the bulbs by turning them all on and off. This will also tell us that the program has been reset.

- Add an invoke "Subroutine" block from the **Control** category into the "setup" area of the main program.
- Rename it "TestPins"

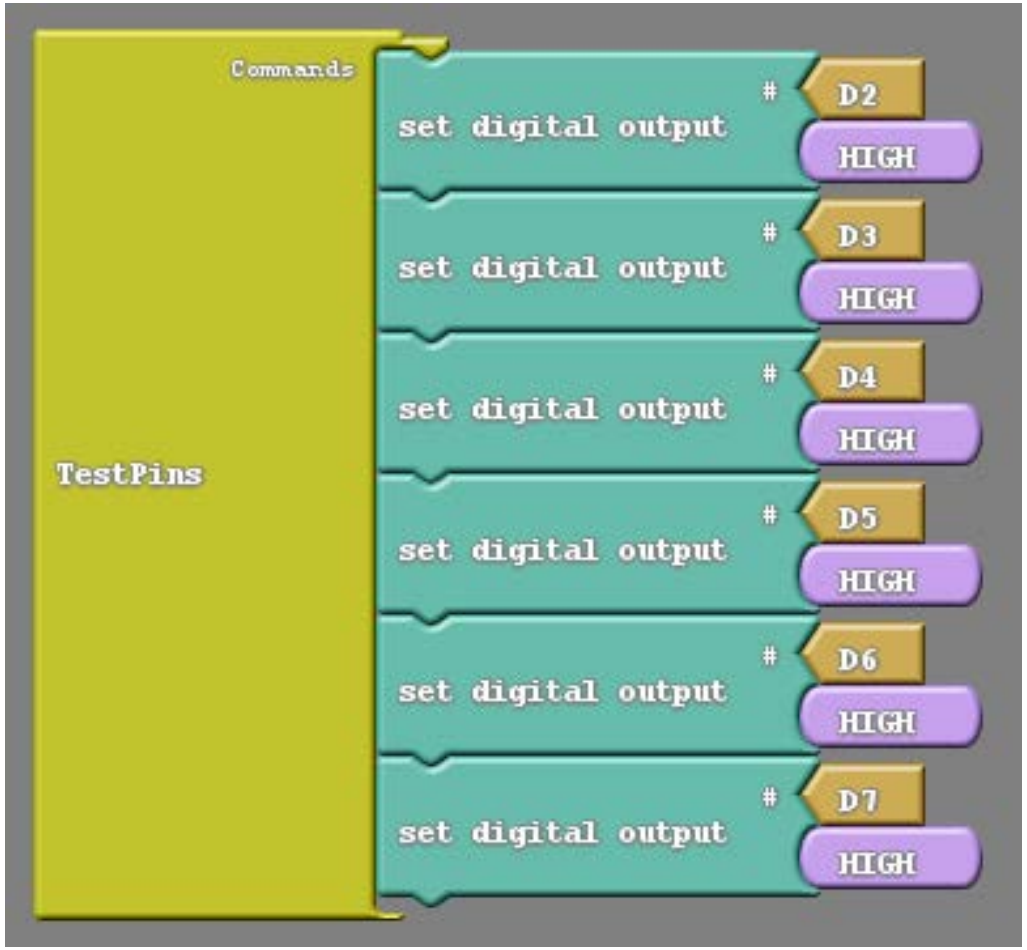
Step 7: Testing the Traffic Lights



Now, we define what the subroutine does.

- Add a define “Subroutine” block from the **Control** category away from your main program.
- Rename it “TestPins” exactly like the other block.

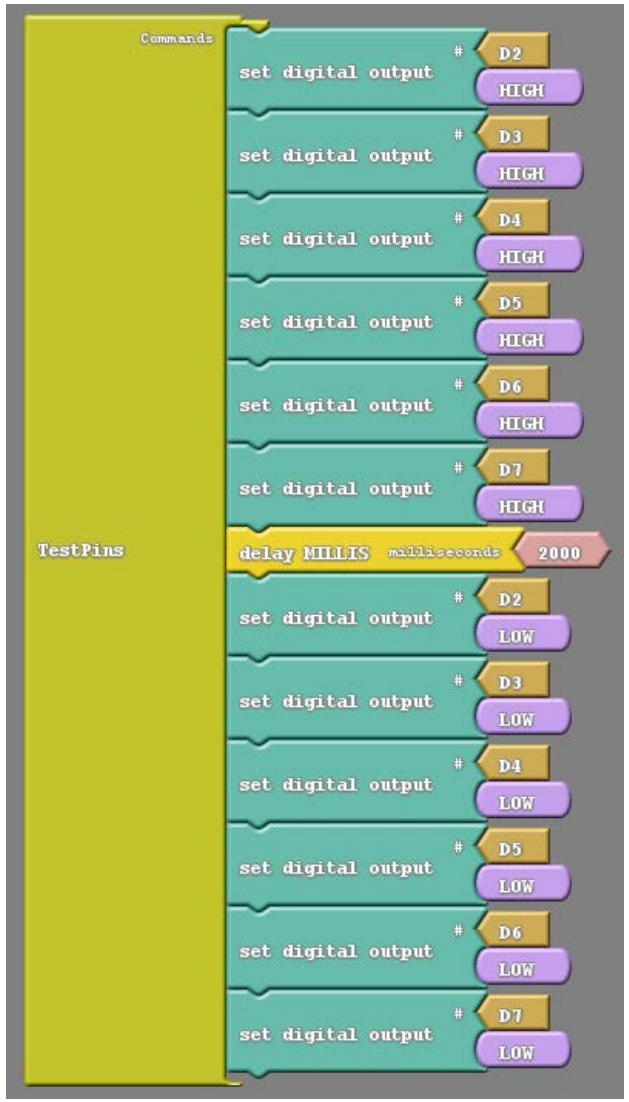
Step 7: Testing the Traffic Lights



Now, we define what the subroutine does.

- Add a “set digital output” block from the **Pins** category and set the D2 pin to “HIGH”.
- Clone this block for each pin, D2-D7.

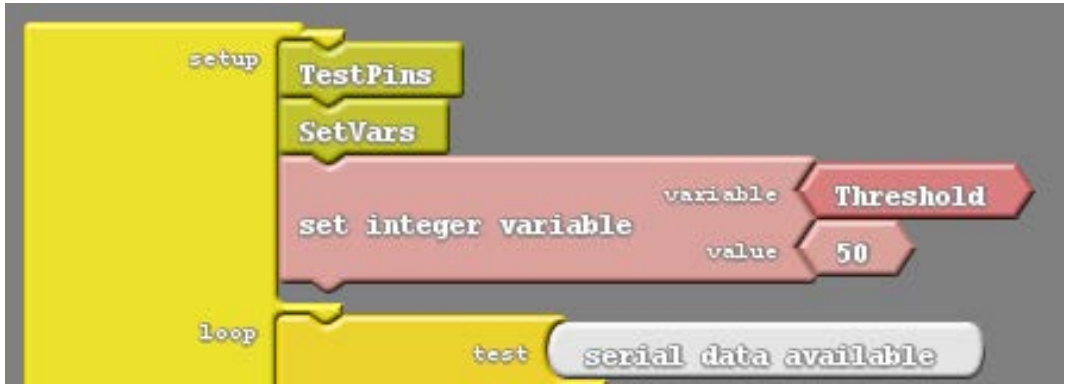
Step 7: Testing the Traffic Lights



Now, we define what the subroutine does.

- Add a “delay MILLIS” block from the **Control** category. Set it to **2000**.
- Clone the “set digital output” blocks for pins, D2-D7. Set them all to “LOW” for this copy.

Step 8: Setting the Default Variables



Next, let's make a subroutine for setting up several necessary variables.

- Add an invoke "Subroutine" block from the **Control** category into the "setup" area of the main program.
- Rename it "SetVars"

Step 8: Setting the Default Variables

Now, define what the subroutine does.

- Add a define “Subroutine” block from the **Control** category away from your main program.
- Rename it “SetVars” exactly like the other block.
- Move the “set integer variable” block for the “Threshold” into this subroutine.

Step 8: Setting the Default Variables

Now, define what the subroutine does.

- Add 5 “set decimal number variable” blocks from the **Variables/ Constants** category.

- Name them the following and assign them the proper values:
  - AvgVRL = 0
  - VoltConst = 5.0
  - LoadResist = 30000
  - VoltLoadResist = 0
  - SensorInput = 0

Step 9: Getting Sensor Data

- When the system boots up, we need a baseline sensor reading in ambient air to compare from.
- We're going to create a subroutine called "GetSensor1"
- This routine is going to get the value from the Analog Output pin of the CO_2 sensor (between 0 and 1023) and compare it to the voltage going into the sensor.

- Single data reads are subject to data fluctuation, so we will take 10 readings over the course of a second and average them out.

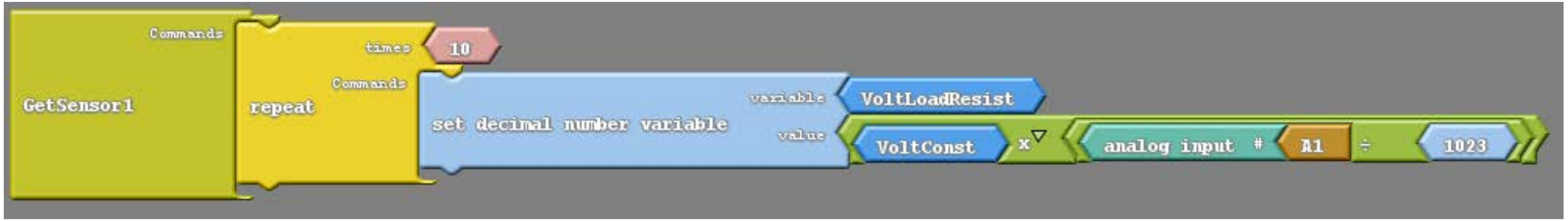
Step 9: Getting Sensor Data

- Create a new subroutine called GetSensor1
- Add a “repeat” block from the **Control** category. Have it repeat 10 times.

Step 9: Getting Sensor Data

- Add a “set decimal number variable” block for the “VoltLoadResist” variable.
- Use operator blocks from the Math Operators category.

Volt Load Resist = VoltConst × (Data from pin A1 ÷ 1023)

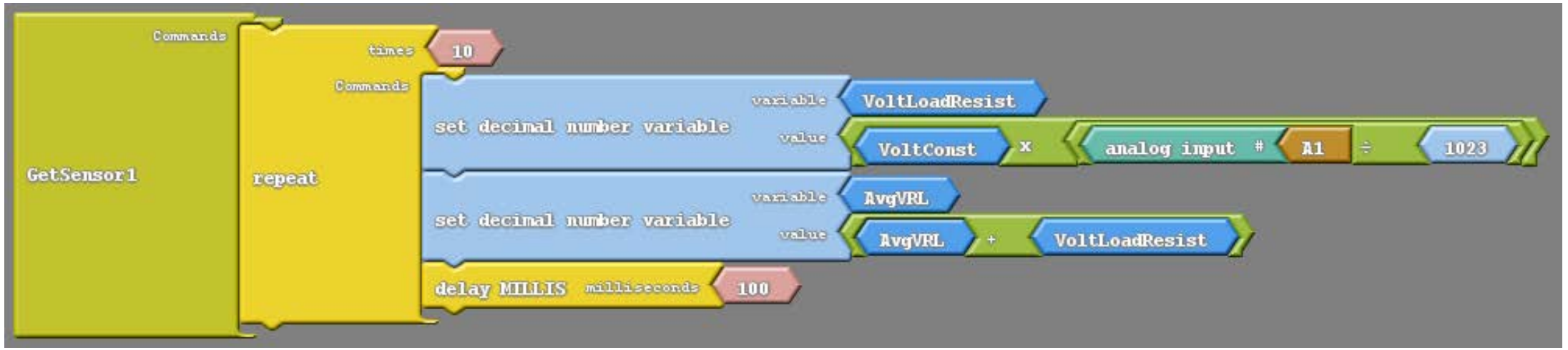


Step 9: Getting Sensor Data

- Take the “VoltLoadResist” and add it to the “AvgVRL” each time we take the data.

(new)AvgVRL = (old)AvgVRL + VoltLoadResist

- Add a 100 millisecond delay to finish the “repeat” block.



Step 9: Getting Sensor Data

- Calculate the average Volt Resistance Load

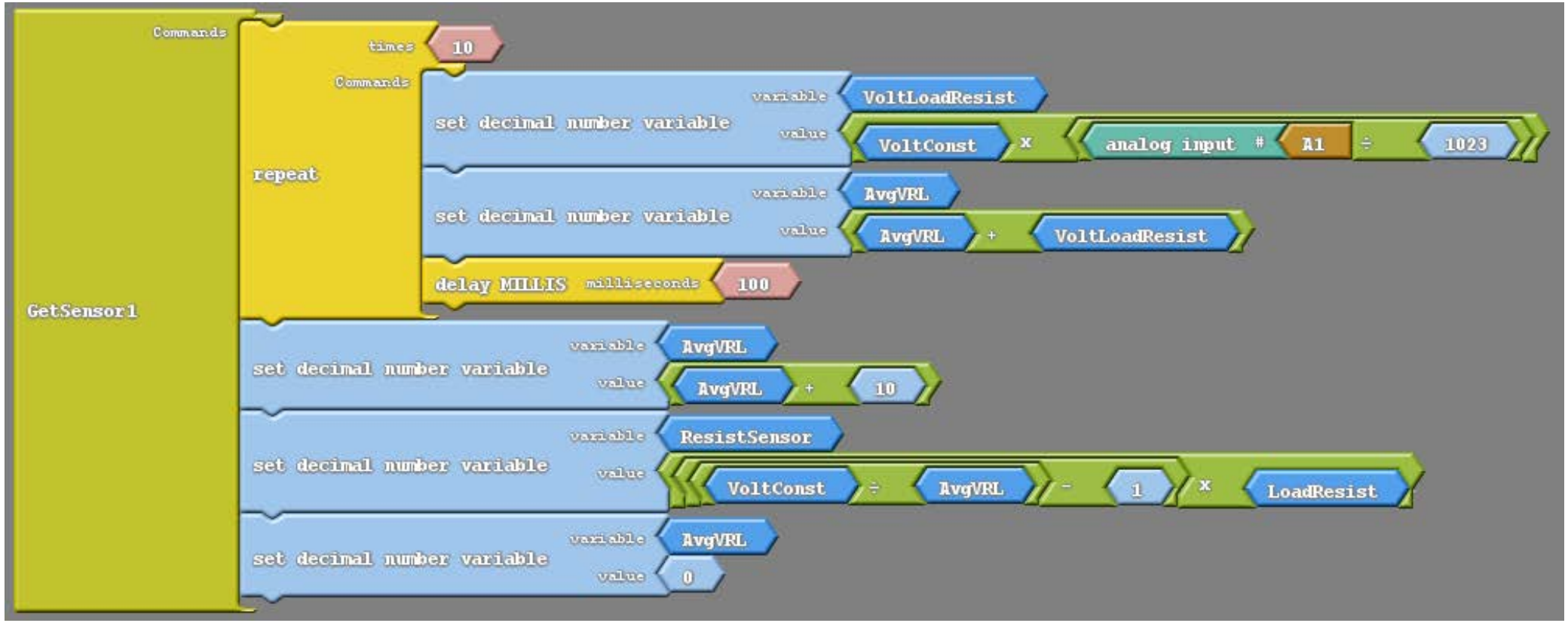
  (new)AvgVRL = (old)AvgVRL ÷ 10

- Next, we calculate the Resistance of the Sensor, which will lead us to getting PPM.

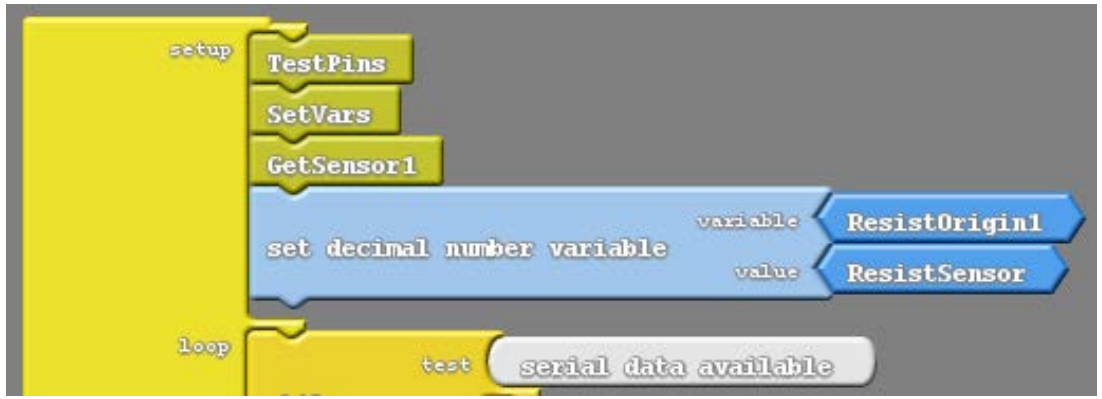
  ResistSensor = ((VoltConst ÷ AvgVRL) – 1) × LoadResist

Step 9: Getting Sensor Data

- A quick housekeeping step, we need to reset the AvgVRL variable back to 0 to be ready for the next sensor data pull.

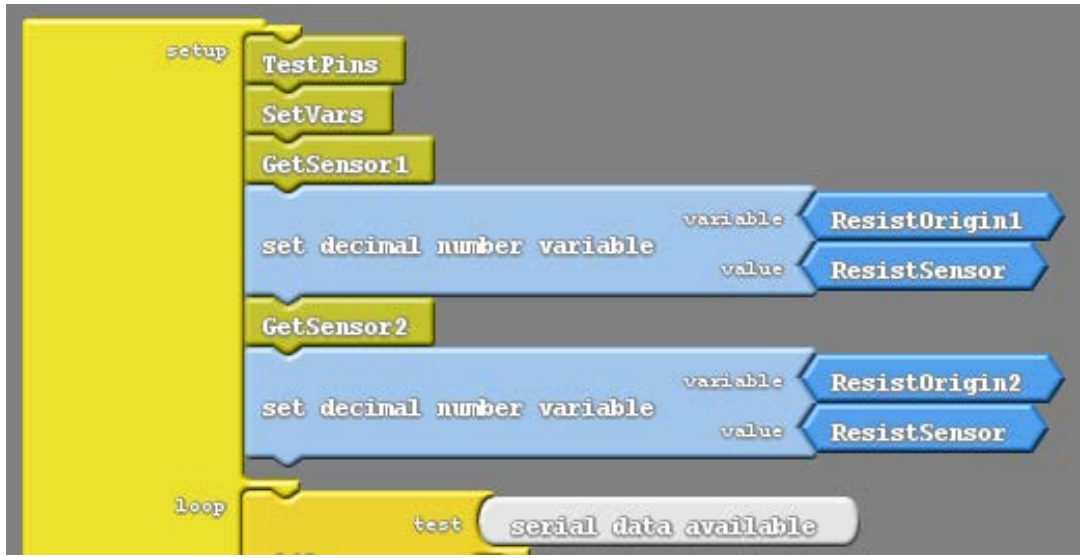


Step 10: Completing the Program Setup



- Back in the “setup” area of the main program, evoke the GetSensor1 subroutine.
- Set a decimal number variable named “ResistOrigin1” and set it equal to the current value of “ResistSensor” that was just calculated from the GetSensor1 subroutine.

Step 10: Completing the Program Setup



This is where pattern recognition and algorithmic thinking come in handy.

- We need to do all the same work for the other CO_2 sensor.
- Rather than start over again from scratch. We can copy the whole subroutine!
- Clone both the full “GetSensor1” subroutine of commands as well as the call for it in the setup area.
- Also clone the set decimal variable command that follows and change its name to “ResistOrigin2”

Step 10: Completing the Program Setup

This is where pattern recognition and algorithmic thinking come in handy.

- Change the name to “GetSensor2” in both, and remember to change the sensor input from Pin A2

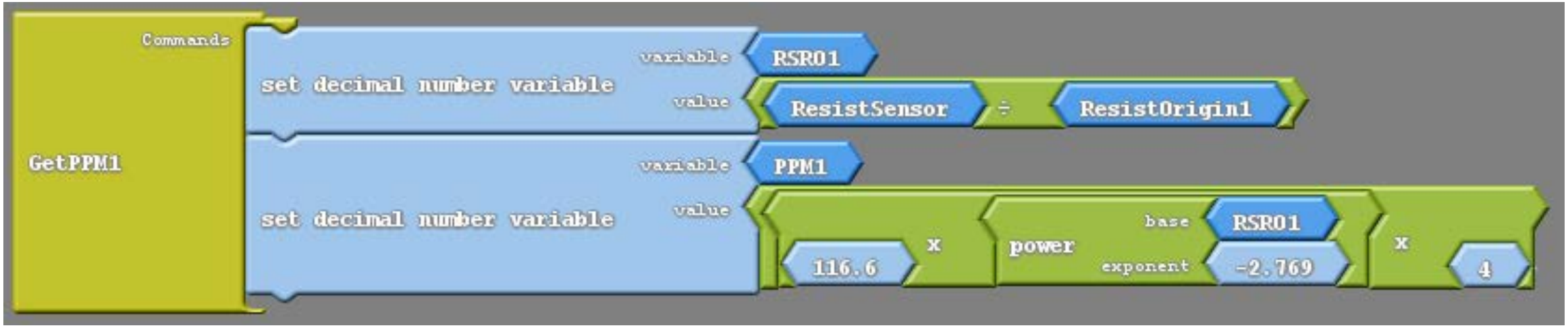
Step 11: Calculating PPM

- Create a new subroutine called “GetPPM1”
- Calculate the ratio of the current Resistance of Sensor 1 compared to the Original Resistance of the same sensor.

RSRO1 = ResistSensor ÷ ResistOrigin1

- To get PPM1, use the RSRO1 variable in this exponential equation:

PPM1 = (116.6 x $RSRO1^{-2.769}$) x 4)



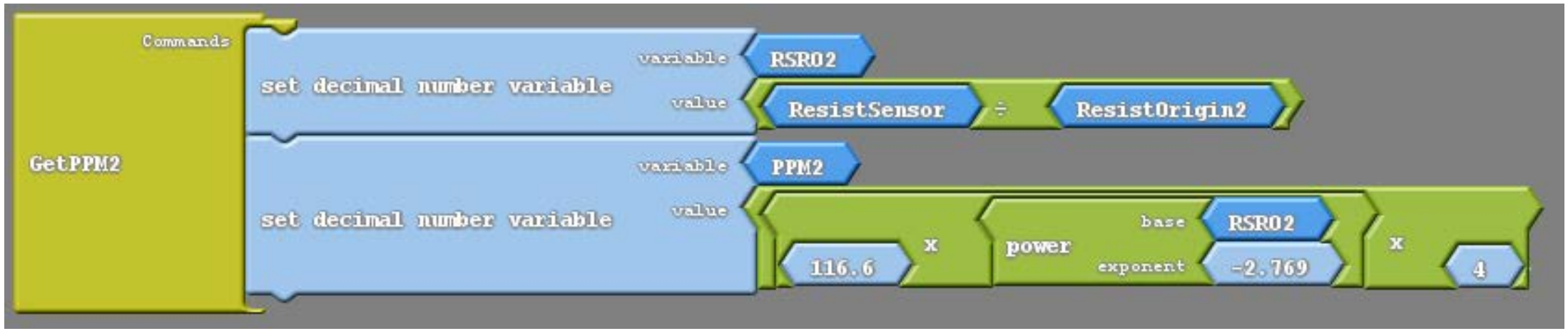
Step 11: Calculating PPM

- Again, we can simply clone the subroutine and call it “GetPPM2”.
- Change the first variable to RSRO2. Calculate the ratio of the current Resistance of Sensor 2 compared to the Original Resistance of the same sensor.

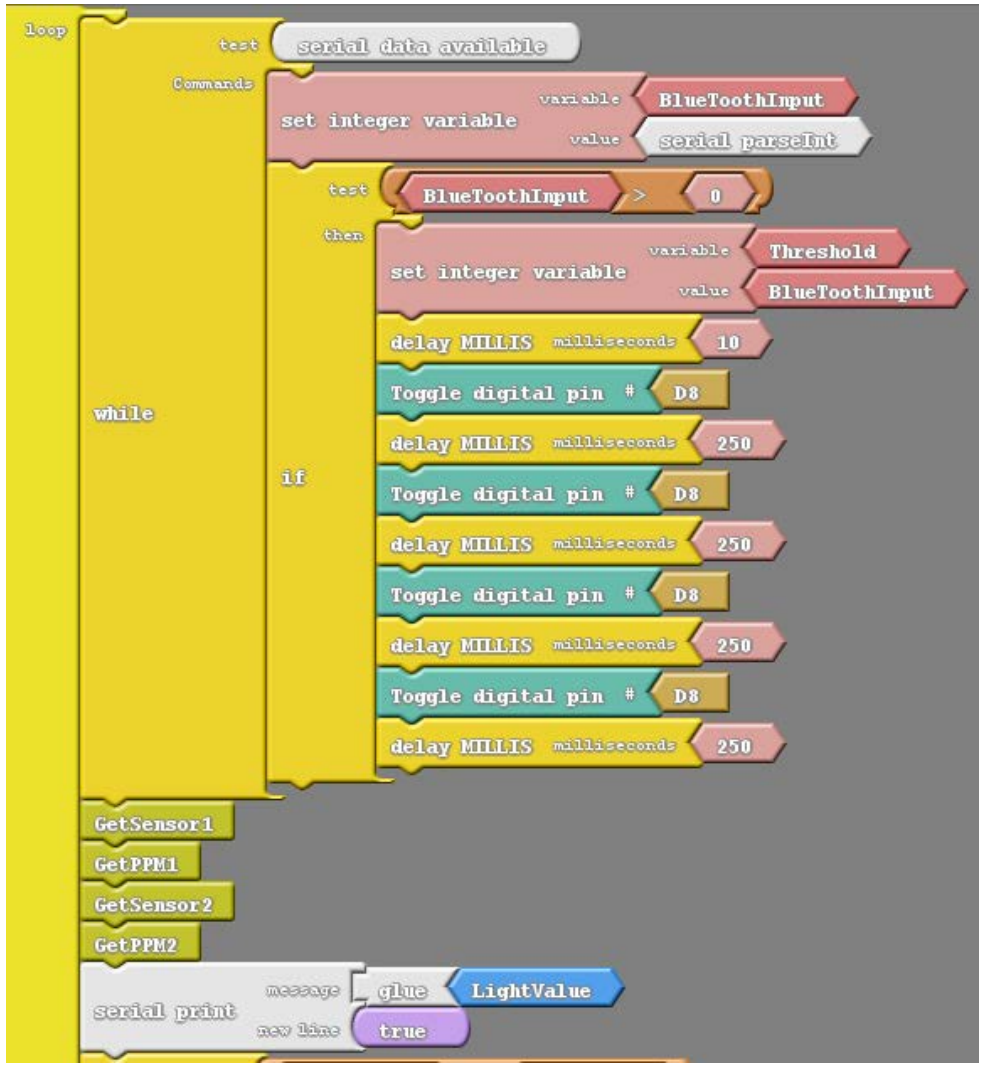
RSRO2 = ResistSensor ÷ ResistOrigin2

- To get PPM2, change to the RSRO2 variable in the same exponential equation:

PPM2 = (116.6 x $RSRO2^{-2.769}$) x 4)



Step 11: Calculating PPM



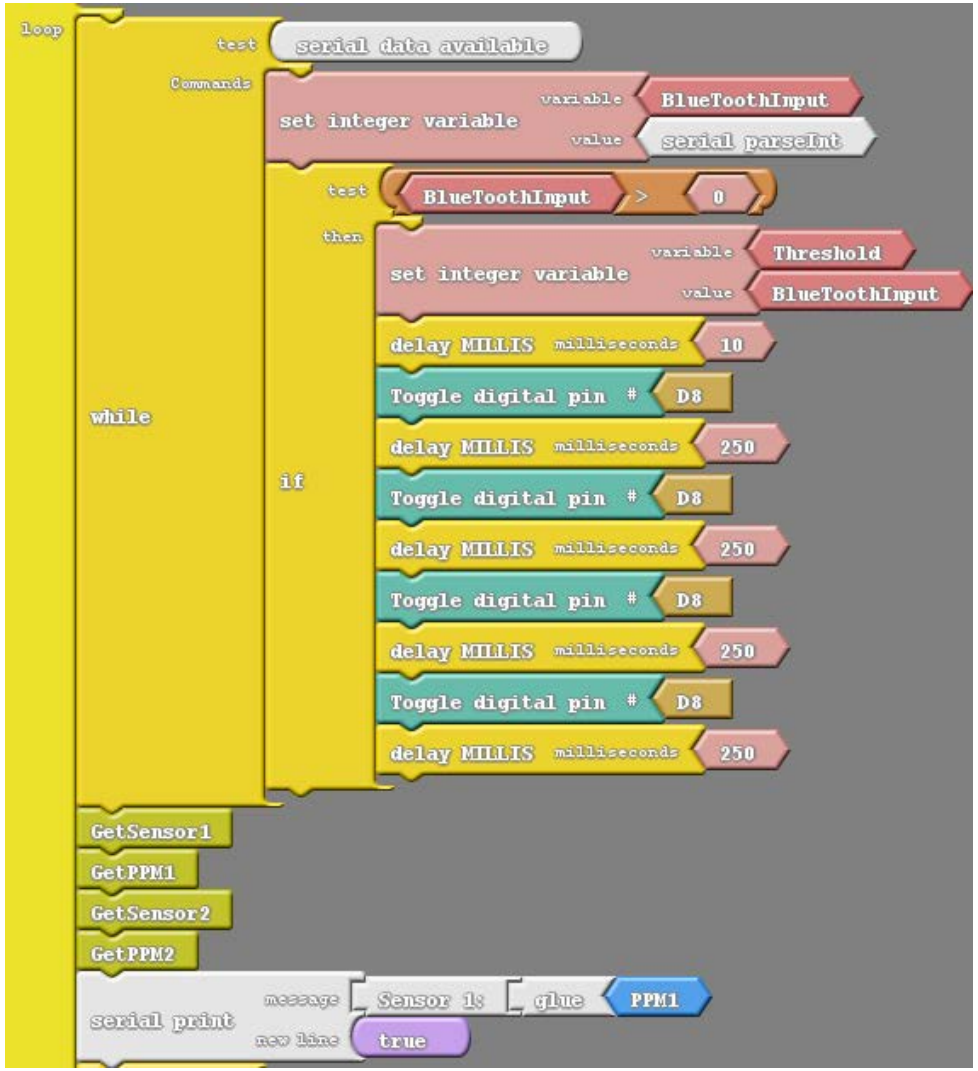
Finish the heavy math portion of the main program by calling the subroutines you created:

- GetSensor1
- GetPPM1
- GetSensor2
- GetPPM2

Use them to replace the command where you pulled data from the Light Sensor.

(If you would like to keep the Light Sensor programming, just pull it off the main program.)

Step 12: Transmitting Data

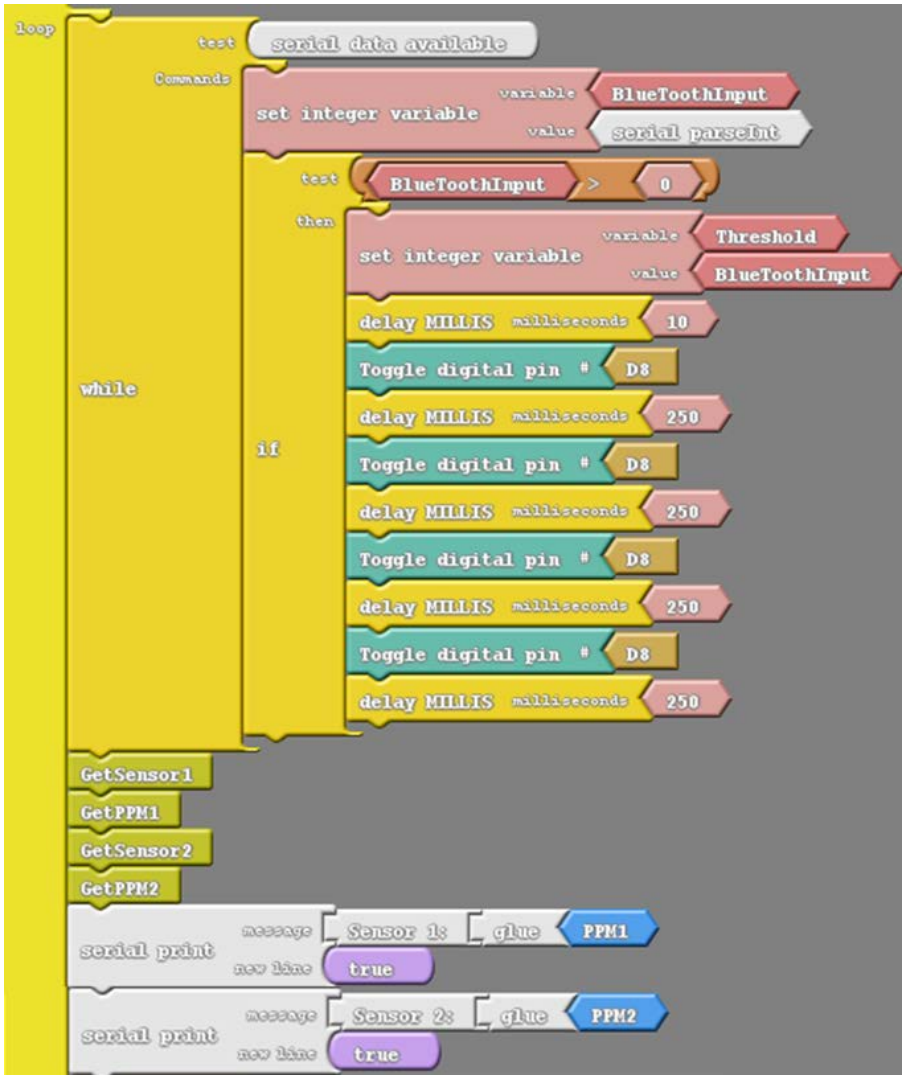


Much like we did with the Light Sensor, the CO_2 program will need to report out the data received.

- Replace the “serial print” command directly following “GetPPM2”
- Change the message to identify “Sensor 1: ”
- Then glue the PPM1 variable to the message.

(Again, if you would like to keep the Light Sensor “serial print” command, just pull it off the main program.)

Step 12: Transmitting Data

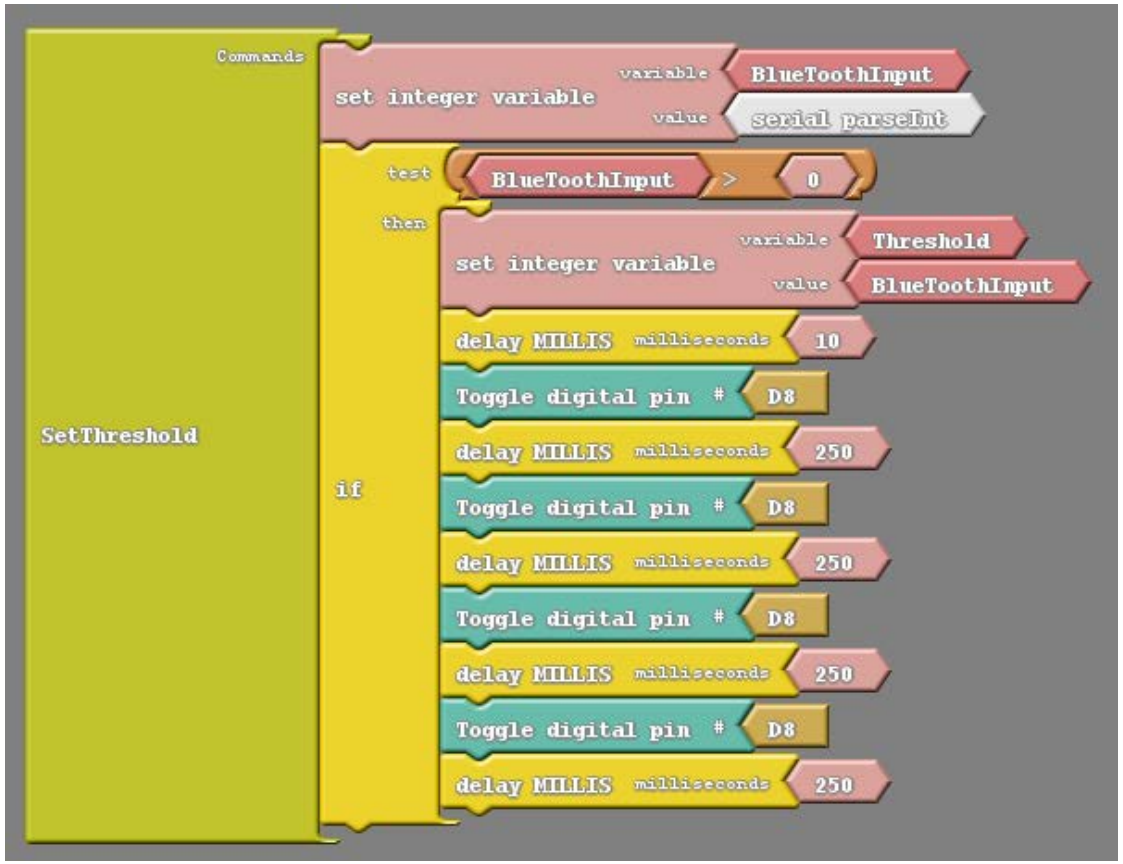


Much like we did with the Light Sensor, the CO_2 program will need to report out the data received.

- Add a second “serial print” command under the one you just placed.
- Change the message to identify “Sensor 2: ”
- Then glue the PPM2 variable to the message.

Now the serial monitor will print out the values of both sensors every time the program takes a set of measurements.

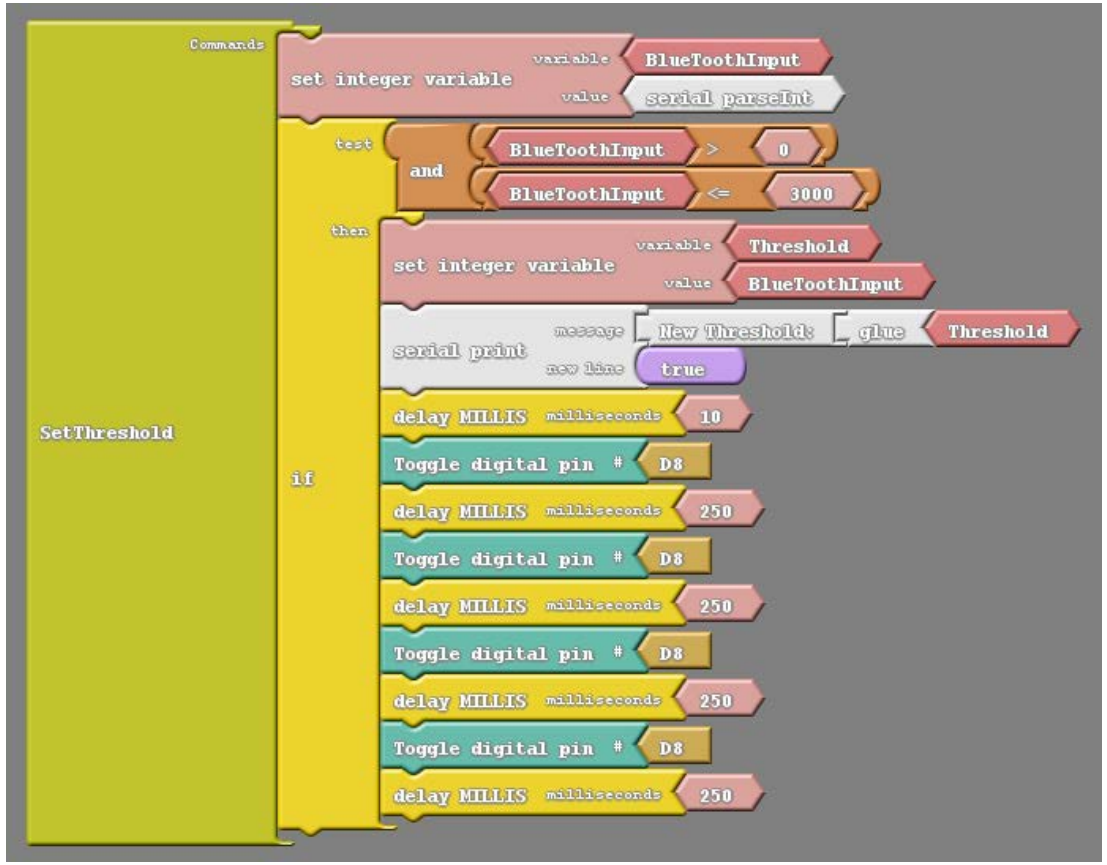
Step 12: Transmitting Data



While we are working on sending and receiving data, let's also work on sending the Threshold.

- Pull the commands out of the while block and place them into a subroutine called "SetThreshold".

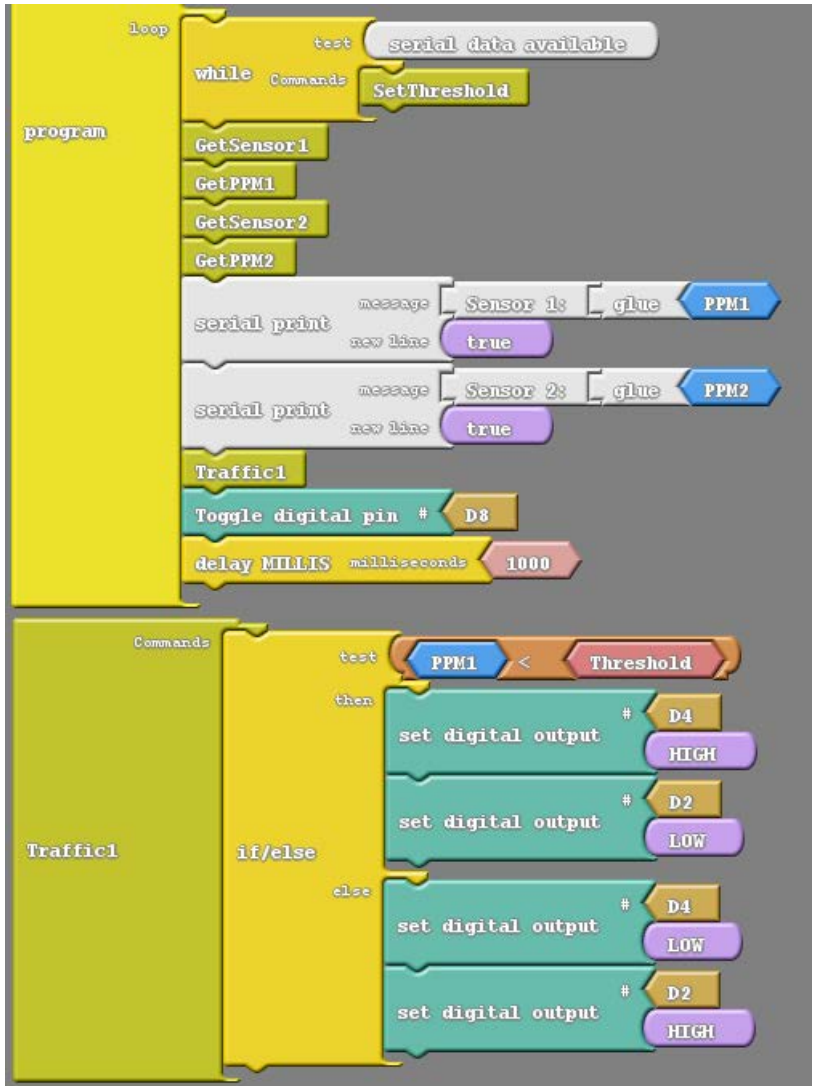
Step 12: Transmitting Data



Since the sensors can only reasonably measure up to 3000 PPM, we need to add another test.

- Add an “and” block from the **Tests** category.
- Put the “BlueToothInput > 0” test in the “and” along with a “BlueToothInput <=3000” test.
- You can also add a “serial print” command to acknowledge the new value.

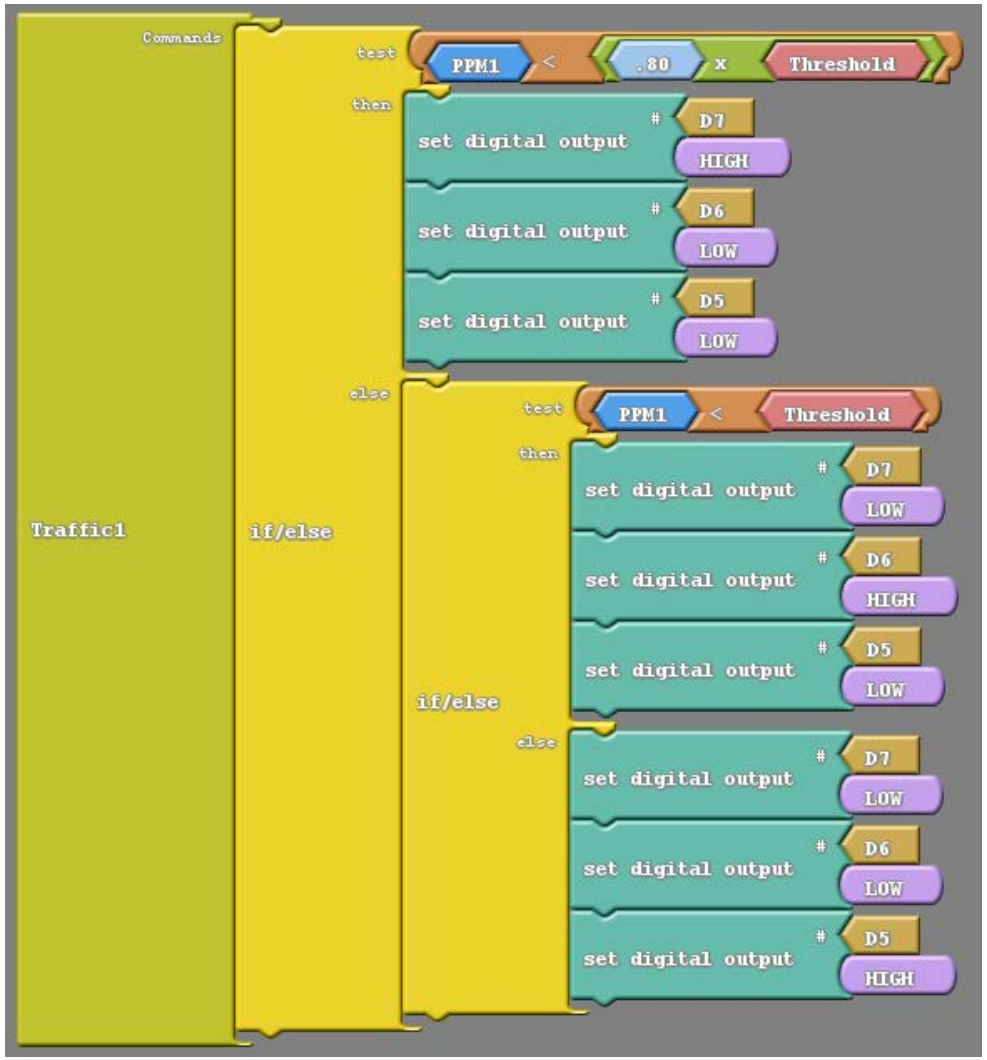
Step 12: Transmitting Data



We will wrap up the program by finishing up the traffic light indicators.

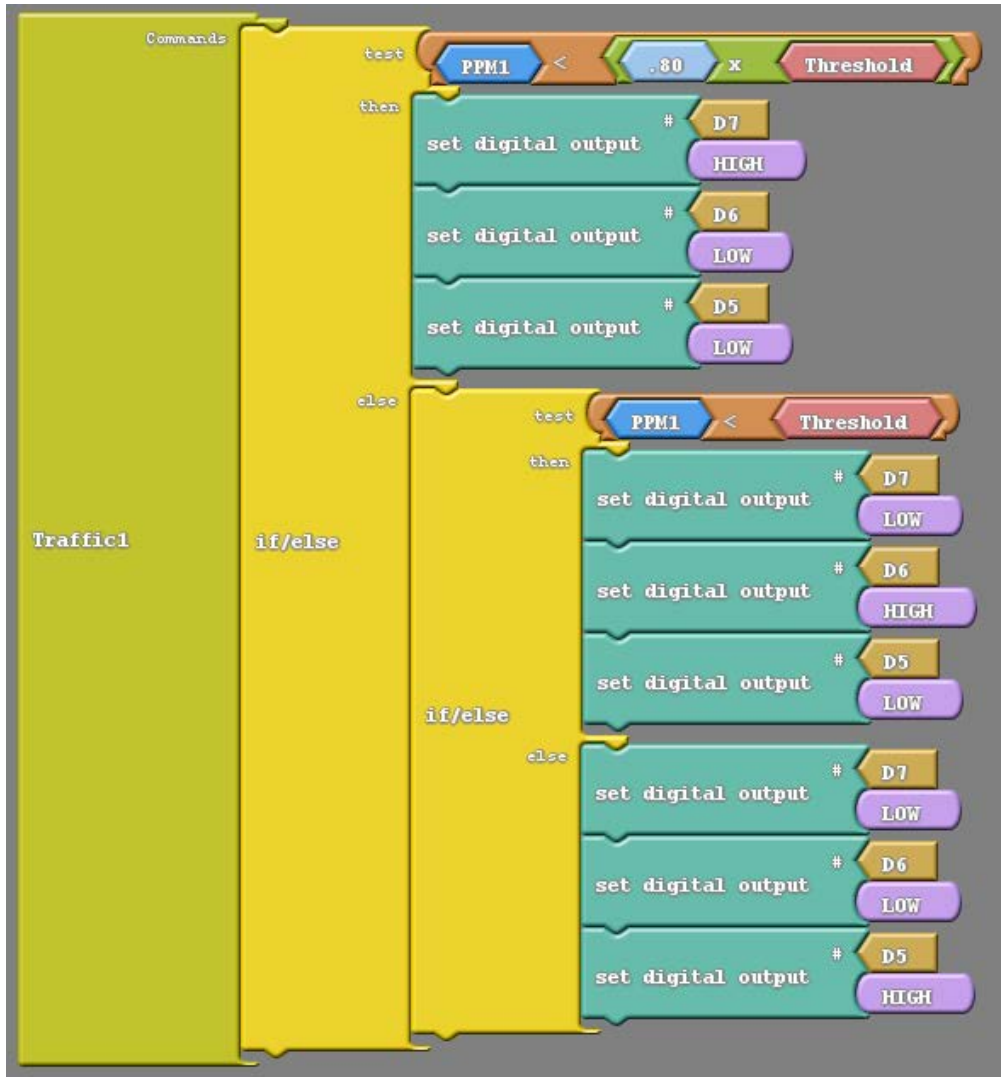
- Pull the if/else that tests the Threshold and add it to a blank subroutine called “Traffic1”.
- Replace the LightValue variable with a cloned PPM1.

Step 12: Transmitting Data



- Change the test to calculate a value that is 80% percent of the Threshold to serve as a warning.
- Add an additional set digital output command for the yellow light pin.
- Change the pins to the proper traffic light:
  - Green = **D7**
  - Yellow = **D6**
  - Red = **D5**
- Add a second nested if/else block to test the full Threshold.

Step 12: Transmitting Data

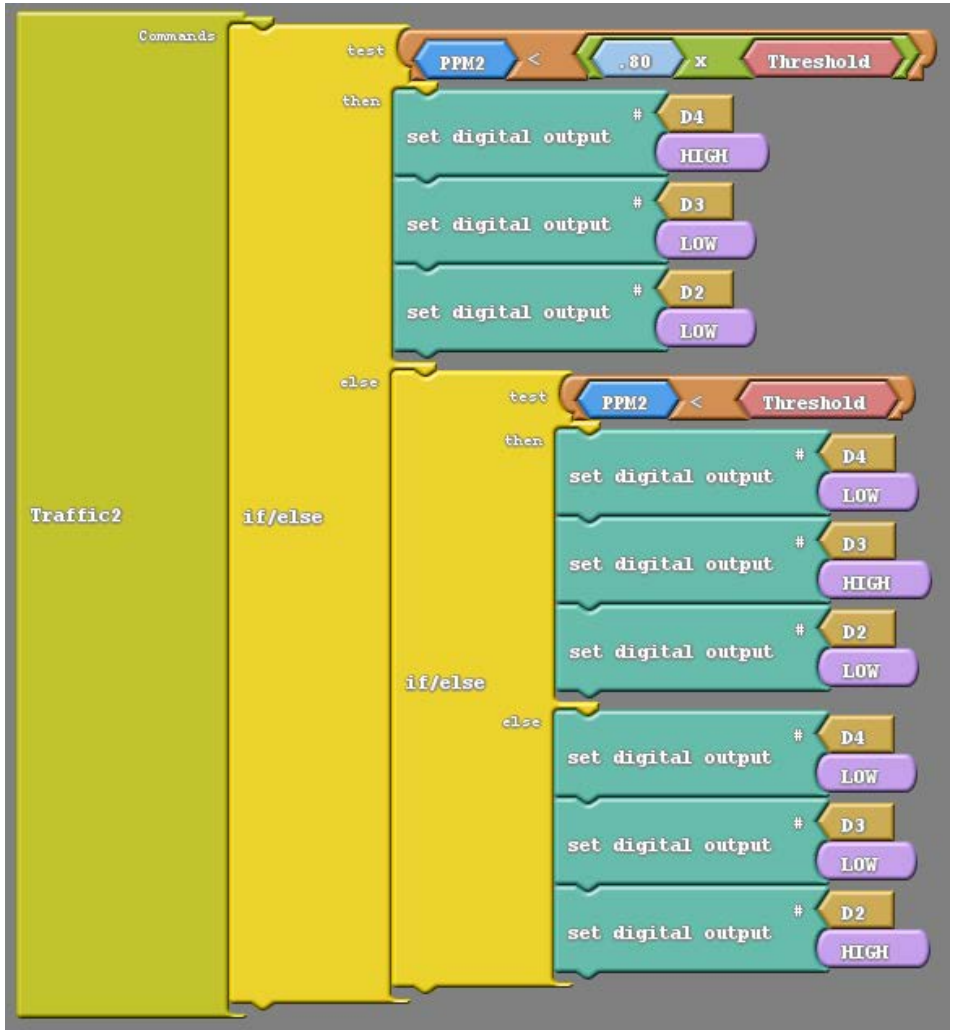


- Now, if the PPM value calculated from Sensor 1 is less than 80% of the Threshold,
  Traffic Light 1 = green

- Otherwise, if the PPM value is less than the full Threshold,
  Traffic Light 1 = yellow

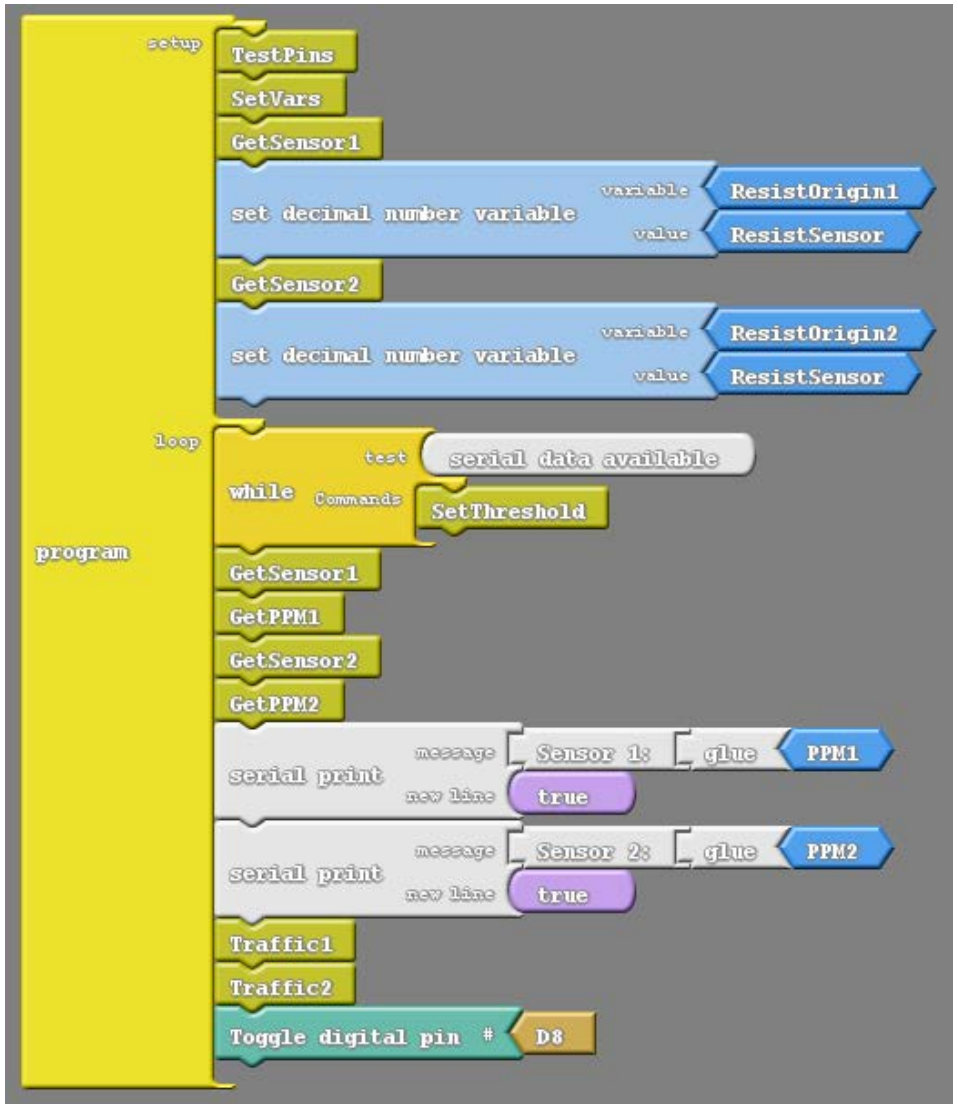
- Otherwise,
  Traffic Light 1 = red

Step 12: Transmitting Data



- Clone the subroutine and name it “Traffic2”.
- Replace the test variable to “PPM2”.
- Change the pins to Traffic Light 2
  - Green = **D4**
  - Yellow = **D3**
  - Red = **D2**

Step 12: Transmitting Data

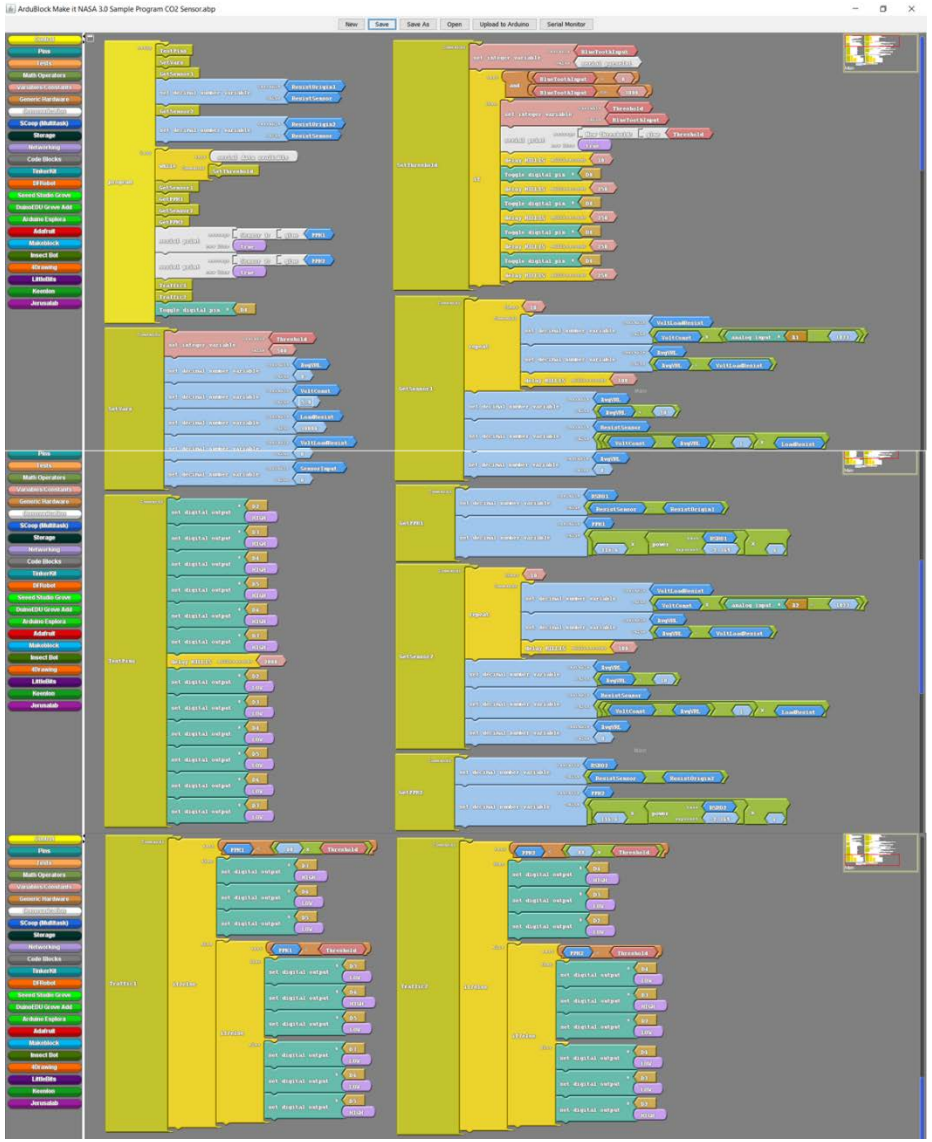


- To finish the program, be sure to evoke the “Traffic1” and “Traffic2” subroutines.
- Also, remove the final delay milliseconds block. It already takes 2 seconds to take the CO_2 data.

Congratulations! You have completed the full program.

Save the program. Upload to the Arduino, and connect the CO2 sensors.

The Completed Program



Share Your Story

- Create a presentation board to present your project.

- Be sure to include:
  - Your original problem
  - What you learned from research
  - Your brainstorm ideas
  - Your first sketch
  - Photos of your prototype
  - Notes from what you improved
  - Photos of all iterations
  - Your final solution model

- Practice presenting your project to others. Be ready to answer questions based on your experience.

Sample Making Project Rubric

	UNSATISFACTORY	COMPETENT	PROFICIENT	DISTINGUISHED
TECHNIQUE/ CONCEPTS	Work lacks understanding of concepts, materials and skills.	Work shows some understanding of concepts, materials and skills.	Work reflects understanding of concepts and materials, as well as use of skills discussed in class.	Work shows a mastery of skills and reflects a deep understanding of concepts and materials.
HABITS OF MIND	Student passively attempts to fulfill assignment without much thought or exploration of possibilities. Student refuses to explore more than one idea.	Developing exploration of possible solutions and innovative thinking. Student has more than one idea but does not pursue.	Student explores multiple solutions and innovative thinking develops and expands during project.	Consistently displays willingness to try multiple solutions and ask thought provoking questions, leading to deeper, more distinctive results. Student fully explores multiple ideas and iterations.
REFLECTION & UNDERSTANDING	Student shows little awareness of their process. The work does not demonstrate understanding of content.	Student demonstrates some self-awareness. Work shows some understanding of content, but student cannot justify all of their decisions.	Student shows self-awareness. Work demonstrates understanding of content and most decisions are conscious and justified.	Work reflects a deep understanding of the complexities of the content. Every decision is purposeful and thoughtful.
CRAFTSMANSHIP	Work is messy and craftsmanship detracts from overall presentation.	Work is somewhat messy and craftsmanship detracts somewhat from overall presentation.	Work is neat and craftsmanship is solid.	Work is impeccable and shows extreme care and thoughtfulness in its craftsmanship.
RESPONSIBILITY	Frequent illegal absences, tardiness, disrespect for classmates and teacher. Disregard for materials and work such as refusal to clean up or throwing out work.	Student is sometimes illegally absent, tardy, or disrespectful. Must be persuaded to assist in clean up and to take work home.	Student is most often present, on time, and respectful. Usually participates willingly in clean up and takes pride in work.	Student is consistently present, punctual, and respectful of classmates and teacher. Self-directed clean up and ownership of work.
EFFORT	Work is not completed in a satisfactory manner. Student shows minimal effort. Student does not use class time effectively.	Work complete but it lacks finishing touches or can be improved with a little effort. Student does just enough to meet requirements.	Completed work in an above average manner, yet more could have been done. Student needs to go one step further to achieve excellence.	Completed work with excellence and exceeded teacher expectations. Student exhibited exemplary commitment to the project.

Created by Lisa Yokana, https://www.edutopia.org/blog/creating-authentic-maker-education-rubric-lisa-yokana

Thank You for Participating!