**Course Overview**

|  |  |
| --- | --- |
| Course No. | N/A |
| Course Code | IT 316 |
| Descriptive Title | Embedded Systems |
| Credit Units | 3 |
| School Year | 2020-2021 |
| Mode of Delivery | Synchronous |
| Name of Instructor | Arman M. Masangkay & John Rey C. Cabasisi |
| Course Description | This course provides an in- depth understanding of Arduino and its pluggable components in developing embedded systems. It also introduces the common syntaxes that are used to communicate to each component that will allow it to work as intended. |
| Course Outcomes | (**Think)**  Plan the different types of components to use to create an embedded technology  **(Do)**  Build an embedded technology based on the formulated plan  **(Feel)**  Defend their design choices when planning and building the technology |
| SLSU Vision | A high-quality corporate university of Science, Technology and Innovation. |
| SLSU Mission | SLSU will:  a. Develop Science, Technology and Innovation leaders and professionals;  b. Produce high-impact technologies from research and innovations;  c. Contribute to sustainable development through responsive community engagement programs;  d. Generate revenues to be self-sufficient and financially-viable; |

**Module Guide**

On our previous module, we understand about Tinkercad and how we can use to simulate Arduino components. The interface is self-explanatory. There are some nuances that needs a bit of exploring though, but knowing the basics of it should be enough.

In this module, we will look into existing embedded technologies that are simple and easy to understand. We will try to create it on our own following the instructions that will be provided and at the end of the lesson, there will be tasks that will require you to revise the technology. This means that you can add more functionality to it or create your own based on that same design.

Revising technologies allow you to understand the workflow even better and the reasons why certain techniques are used. This also widens your creativity because it will force you think of how you can innovate an existing technology to make it work even better.

Again, if you have not accessed our LMS, you can do so by following the instructions below.

***Instructions when accessing the LMS (Learning Management System) Moodle:***

* Look for “Embedded Systems (2020-2021)” course
* Self-enroll using the passcode “es2021”

Here’s my contact details if ever you have questions regarding the module or the subject:

Facebook name: Arman Macasuhot Masangkay (doing.art.man)

Email: [vss.arman@gmail.com](mailto:vss.arman@gmail.com)

**Pre-test**

Before we proceed, let’s have some tests to check your understanding of the upcoming topics as well as the previous ones.

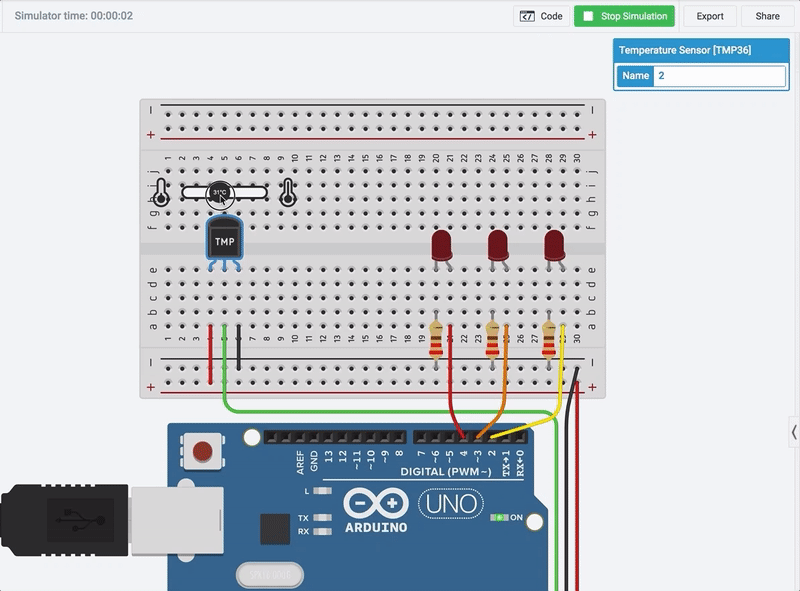
1. What is pinMode for?
2. Is HIGH and LOW constant variable a Boolean data type or integer?
3. What parameter does the delay() function requires?
4. What is TMP36 used for?
5. How does an ultrasonic sensor works?

***By the end of this module, you should be able to:***

* Revise an existing embedded technology
* Build an embedded technology base on the formulated plan

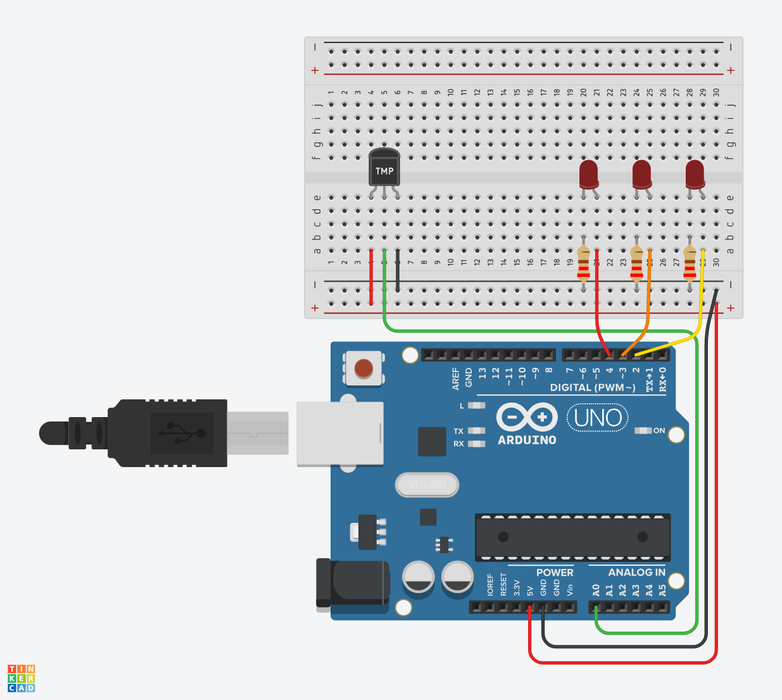
**Lesson 1 – Building your own Thermometer**

In this project, you will turn the Arduino into a thermometer! Use a temperature sensor to measure your skin temperature, and register the output with three LEDs. Even though the Arduino is a digital tool, it can interpret signals from an analog input, like the TMP36 temperature sensor, using the built in Analog-to-Digital (ADC) converter, accessed through the analog pins A0-A5.



To optionally build the physical circuit, gather up your Arduino Uno board, USB cable, solderless breadboard, three LEDs, three alike resistors (any value from 100-1K, 220 ohms preferred), a TMP36 temperature sensor, and breadboard wires.

**Step 1: Build the LED Circuit**



Start by wiring up your Arduino and breadboard with power and ground next to the example circuit, then add the the three red LEDs to the breadboard, as shown. These will be the indicator or "bar graph" lights for the project.

Drag an Arduino Uno and breadboard from the components panel to the workplane, next to the existing circuit.

Connect the 5 volt and ground pins on the Arduino to the power (+) and ground (-) rails on the breadboard with wires. You can change the wire colors if you want to! Either use the inspector dropdown or the number keys on your keyboard.

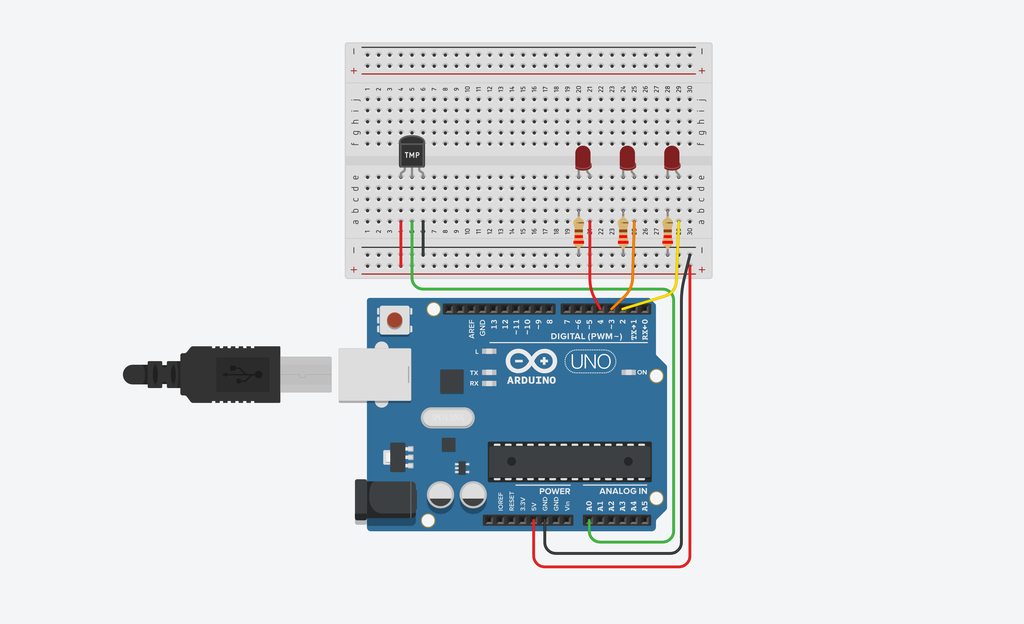
Drag three LEDs on the breadboard in row E, spaced 2 breadboard sockets apart. You can change the LED color using the inspector that pops up when you click on each one.

Use a 220 Ohm resistor to connect each LED's cathode (left leg) to the ground rail (black) of the breadboard. In Tinkercad Circutis, you can change a resistor's value by highlighting it and using the dropdown menu in the inspector.

Connect the LED anodes (right, longer legs) to digital pins 4, 3, and 2 on the Arduino. The LED anode (+) is the terminal that current flows into.

The cathode (-) is the terminal that current flows from. This connects to the ground rail.

**Step 2: Add Temperature Sensor**



A temperature sensor creates a changing voltage signal depending on the temperature it senses. It has three pins: one that connects to ground, another that connects to 5 volts, and a third that outputs a variable voltage to your Arduino, similar to the analog signal from a potentiometer.

There are several different models of temperature sensor. This model, the TMP36, is convenient because its output voltage is directly proportional to temperature in degrees Celsius.

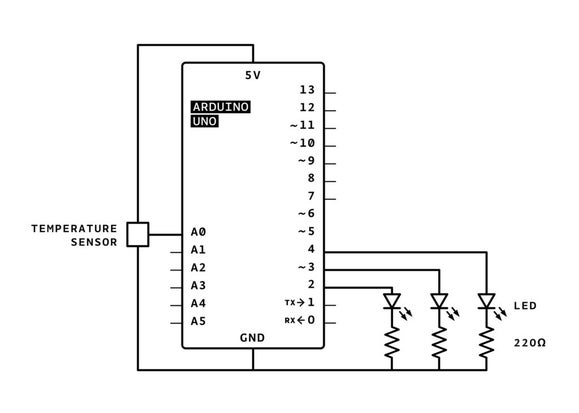
In the circuits editor, find the temperature sensor in the components drawer.

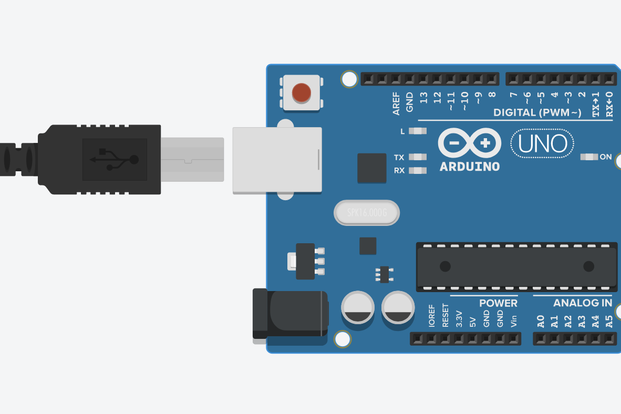
Place the temperature sensor (TMP36) on the breadboard with the rounded part facing away from the Arduino, as shown in the figure (this is the default orientation).

Place the temperature sensor on the breadboard in row E, as shown.

Wire up the temperature sensor so the left pin connects to the 5V voltage rail, the center pin connects to A0 on the Arduino, and the right pin connects to the GND rail.

**Step 3: Analog Input Observation**





In the circuit schematic, you can see that the temperature sensor is connected to power (5 volts) and ground (0 volts) and the analog pin A0. As temperature rises, the pin connected to A0 increases its voltage. You can also see that three LEDs are each connected to their own digital pin.

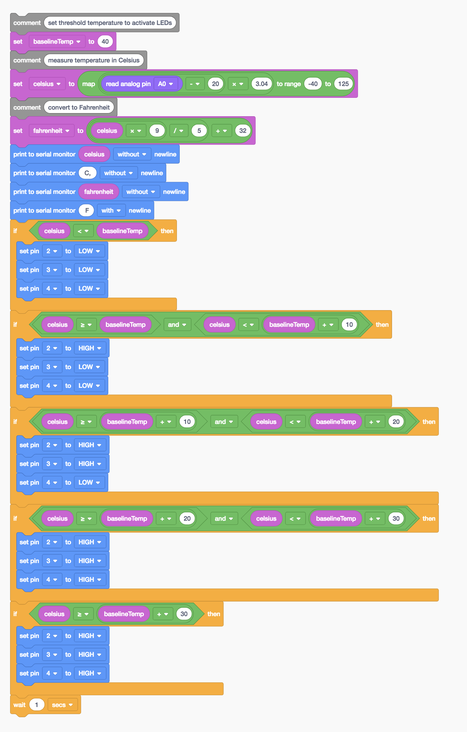
Even though the Arduino is a digital tool, it’s possible for it to get information from analog sensors to measure things like temperature or light. To do this, you’ll take advantage of the Arduino’s built-in Analog-to-Digital Converter (ADC).

Analog-in pins A0 to A5 can interpret voltages between 0 and 5V, and translate that voltage to a value between 0 and 1023 for the Arduino sketch to use. The analog pins are primarily used to read information from sensors (but can also be used as digital outputs 14-19, unrelatedly).

Click "Start Simulation."

Open the Code Editor and find the "Serial Monitor" button to watch the sensor values pour in.

**Step 4: Blocks Code**



Let's use the code blocks editor to listen to the state of the sensor, then make decisions about which LEDs to light up based on the sensor's value.

Click the "Code" button to open the code editor. The grey Notation blocks are comments for making note of what you intend for your code to do, but this text isn't required or executed as part of the program.

Click on the Variables category in the code editor. Create a new variable called baselineTemp and use a "set" block to set it to 40 (degrees C).

To store the sensor value, create a variable named "celsius".

Drag out a "set" block and adjust the dropdown to our new variable celsius.

In the Math category, drag out a "map" block, and nest two arithmetic blocks ("1 + 1") within its first field.

Adjust the range from -40 to 125.

Click on the Input category and drag out an "analog read pin" block, and place it into the first arithmetic field inside the "map" block.

Adjust the arithmetic blocks to "(read analog pin A0 - 20) x 3.04".

Optionally create a new variable for converting the temperature to Fahrenheit with a set block and some arithmetic blocks to read "set fahrenheit to (celsius x 9)/5 + 32".

Add some serial monitoring blocks to print out the temperature in one or both C or F.

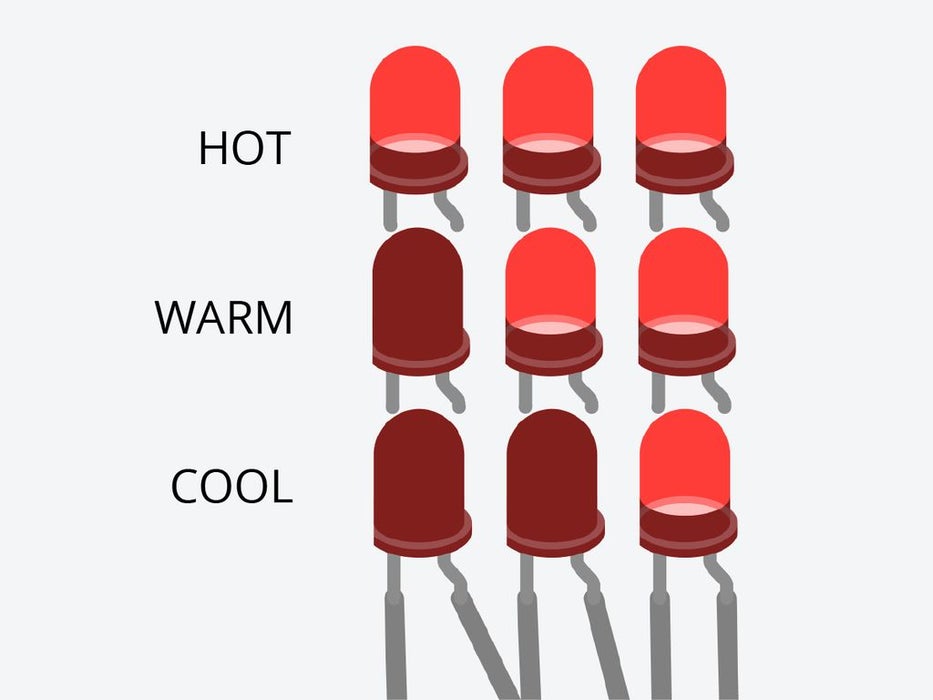
Click the Control category and drag out an if then block, then navigate to Math and drag a comparator block onto the if block.

In the Variables category, grab the celsius variable and the baselineTemp variable and drag them into the comparator block, adjusting the dropdown so it reads "if celsius < baselineTemp then".

Add three digital output blocks inside the if statement to set pins 2, 3, and 4 LOW.

Duplicate this if statement four times and add arithmetic blocks and and/or blocks to create five total state detection if statements. The first state is "the temperature is below our target baseline" so no LEDs light up. When the temperature is greater than or equal to the baselineTemp and less than baselineTemp+10, light up only pin 2's LED. When the temperature is between baselineTemp+10 and baselineTemp+20, light up two LEDs. And so on to account for all the desired states.

**Step 5: Arduino Code Explained**



When the code editor is open, you can click the dropdown menu on the left and select "Blocks + Text" to reveal the Arduino code generated by the code blocks. Follow along as we explore the code in more detail.

int baselineTemp = 0;

int celsius = 0;

int fahrenheit = 0;

Before the setup(), we create variables to store the target baseline temperature, as well as the sensor value. They're called int because they are integers, or any whole number.

void setup()

{

pinMode(A0, INPUT);

Serial.begin(9600);

pinMode(2, OUTPUT);

pinMode(3, OUTPUT);

pinMode(4, OUTPUT);

}

Inside the setup, pins are configured using the pinMode() function. Pin A0 is configured as an input, so we can "listen" to the electrical state of the temperature sensor. Pins 2, 3, and 4 are configured as outputs to control the LEDs.

void loop()

{

// set threshold temperature to activate LEDs

baselineTemp = 40;

// measure temperature in Celsius

celsius = map(((analogRead(A0) - 20) \* 3.04), 0, 1023, -40, 125);

Anything after a set of slashes // is a comment, just for us humans to read, and is not included in the program when the Arduino runs it. In the main loop, baselineTemp is set to its target 40 degrees C.

// convert to Fahrenheit

fahrenheit = ((celsius \* 9) / 5 + 32);

Serial.print(celsius);

Serial.print(" C, ");

Serial.print(fahrenheit);

Serial.println(" F");

The formula for converting between celsius and Fahrenheit is F = (C \* 9) / 5 + 32. Printing to the serial monitor helps you observe the temperature change more granularly than the LED states show alone.

if (celsius < baselineTemp) {

digitalWrite(2, LOW);

digitalWrite(3, LOW);

digitalWrite(4, LOW);

}

if (celsius >= baselineTemp && celsius < baselineTemp + 10) {

digitalWrite(2, HIGH);

digitalWrite(3, LOW);

digitalWrite(4, LOW);

}

if (celsius >= baselineTemp + 10 && celsius < baselineTemp + 20) {

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, LOW);

}

if (celsius >= baselineTemp + 20 && celsius < baselineTemp + 30) {

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, HIGH);

}

if (celsius >= baselineTemp + 30) {

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, HIGH);

}

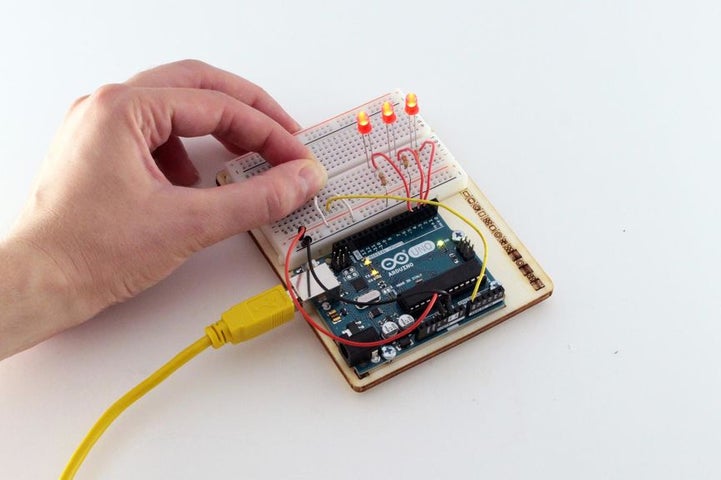
delay(1000); // Wait for 1000 millisecond(s)

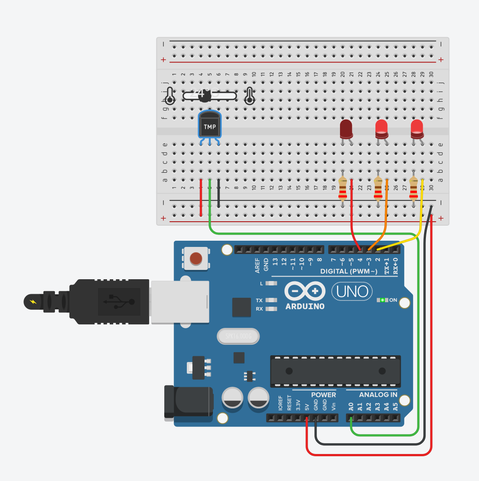
}

The loop's six if statements evaluate for different segments of a certain temperature range between 40 and 46 degrees C, lighting up more LEDs the warmer the temperature.

If you want to see a more obvious change in bar graph lights, you can change the baseline temperature variable and/or the range that you are looking at by changing the arguments in the if() statements. This is called calibration.

**Step 6: Use It!**





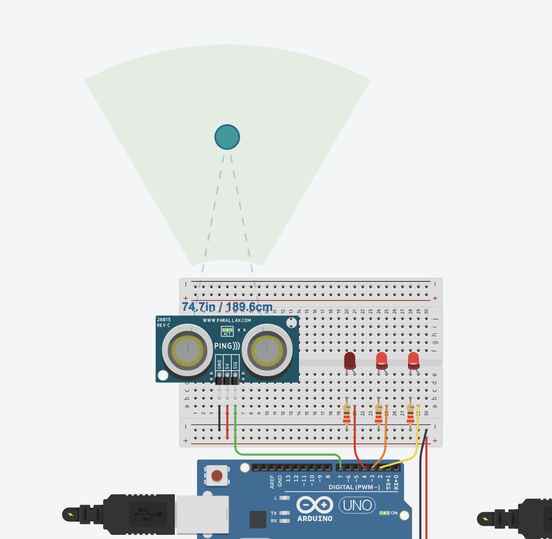
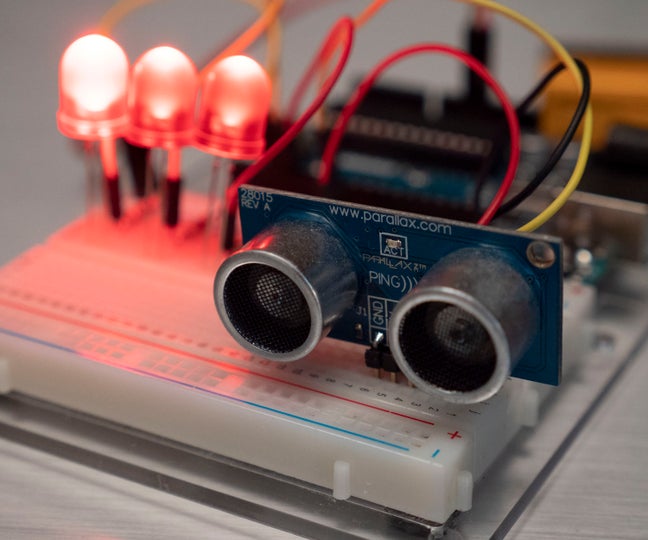
If you built a physical version of this circuit, you can try it out with the Arduino software's serial monitor (magnifying glass button in the upper right of the sketch window), activating the sensor with your fingers. The project might not do what you want it to if the room temperature is really cold or really warm, or if your fingers are cold!

If using a physical board, observe the room temperature using the serial monitor, and set baselineTemp to that value.

Adjust your different temperature threshold "buckets" to a smaller range (2, 4, 6, instead of 10, 20, 30).

Upload your code again, and try holding the sensor in your fingers. As the temperature rises, you should see the LEDs turn on one by one.

**Lesson 2 – Ultrasonic Distance Sensor in Arduino**

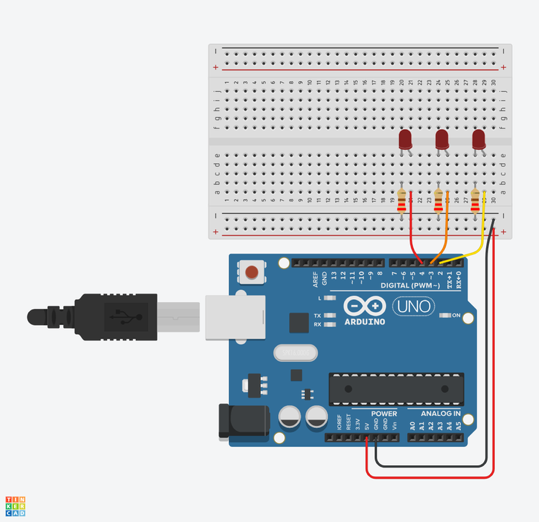
 

Let's measure distances with an ultrasonic rangefinder (distance sensor) and Arduino's digital input. We'll connect up a circuit using a breadboard and use some simple Arduino code to control a single LED.

Ultrasonic rangefinders use sound waves to bounce off objects in front of them, much like bats using echolocation to sense their environment. The proximity sensor sends out a signal and measures how long it takes to return. The Arduino program receives this information and calculates the distance between the sensor and object.

To optionally build the physical circuit, gather up your Arduino Uno board, USB cable, solderless breadboard, three LEDs, resistors (any value from 100-1K), ultrasonic rangefinder, and breadboard wires.

**Step 1: Build the LED Circuit**



Start by wiring up your Arduino and breadboard with power and ground next to the example circuit, then add the the three red LEDs to the breadboard, as shown. These will be the "bar graph" lights for visually indicating the sensor's distance measurement.

Drag an Arduino Uno and breadboard from the components panel to the workplane, next to the existing circuit.

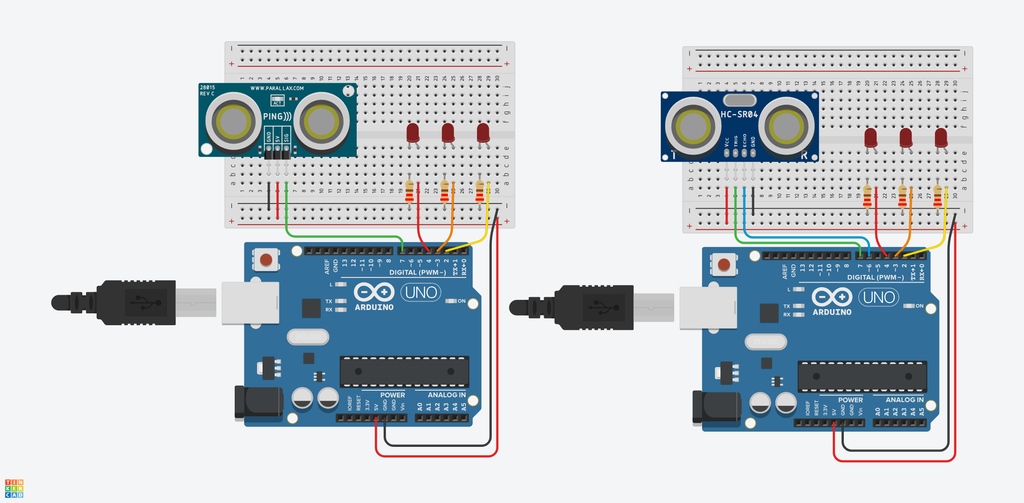
Connect the 5 volt and ground pins on the Arduino to the power (+) and ground (-) rails on the breadboard with wires. You can change the wire colors if you want to! Either use the inspector dropdown or the number keys on your keyboard.

Drag three LEDs on the breadboard in row E, spaced 2 breadboard sockets apart. You can change the LED color using the inspector that pops up when you click on each one.

Use a 220 Ohm resistor to connect each LED's cathode (left leg) to the ground rail (black) of the breadboard. You can change a resistor's value by highlighting it and using the dropdown menu.

Connect the LED anodes (right legs) to digital pins 4, 3, and 2 on the Arduino. The LED anode (+) is the terminal that current flows into. This will connect to the digital output pins on the Arduino. The cathode (-) is the terminal that current flows from. This will connect to the ground rail.

**Step 2: Add Proximity Sensor**



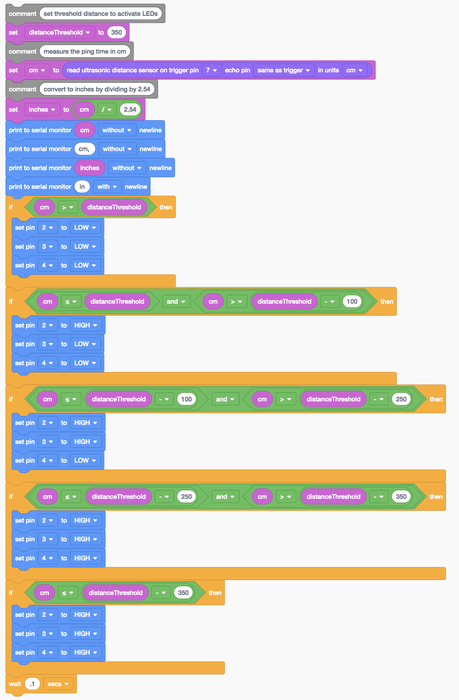
Proximity sensors come in multiple flavors. Here in Tinkercad Circuits, you can choose between a three-pin sensor or a four-pin sensor. In general, ultrasonic rangefinders have one pin that connects to ground, another that connects to 5 volts, a third for sending a signal, and a fourth for receiving a signal. The 'send' and 'receive' pins are combined into one pin on the three-pin flavor.

In the circuits editor, find the ultrasonic rangefinder in the components drawer. To find the four-pin sensor, view "All" in the components panel (using the dropdown menu).

Place the sensor on the breadboard to the left of the LEDs in row E, as shown in the figure.

Wire up the sensor so the 5V pin connects to the 5V voltage rail, the GND pin connects to the ground rail, the SIG or TRIG pin to Arduino pin 7, and, if using the four-pin flavor, the ECHO pin connects to Arduino pin 6.

**Step 3: Code With Blocks**



Let's use the code blocks editor to listen to the state of the sensor, then make decisions about which LEDs to light up based on the sensor's value.

Click the "Code" button to open the code editor. The grey Notation blocks are comments for making note of what you intend for your code to do, but this text isn't required or executed as part of the program.

Click on the Variables category in the code editor. Create a new variable called distanceThreshold and use a "set" block to set it to 350 (centimeters).

To store the sensor value, create a variable named "cm".

Drag out a "set" block and adjust the dropdown to our new variable cm.

In the Input category, drag out a "read ultrasonic distance sensor on" block, and place it inside the set block.

Adjust the dropdown menus inside the input block to set the trigger pin to 7, the echo pin to "same as trigger" and units to cm.

Optionally create a new variable for converting centimeters to inches with a set block and an arithmetic block to read "set inches to (cm / 2.54)".

Add some serial monitoring blocks to print out the sensor distance in centimeters and inches.

Click the Control category and drag out an if then block, then navigate to Math and drag a comparator block onto the if block.

In the Variables category, grab the cm variable and the distanceThreshold variable and drag them into the comparator block, adjusting the dropdown so it reads "if cm > distanceThreshold then".

Add three digital output blocks inside the if statement to set pins 2, 3, and 4 LOW.

Duplicate this if statement four times and add arithmetic blocks and and/or blocks to create five total state detection if statements. The first state is "the distance is farther than our threshold" so no LEDs light up. When the distance is closer than or equal to the distanceThreshold and greater than distanceThreshold-100, light up only pin 2's LED. When the temperature is between distanceThreshold-100 and distanceThreshold-250, light up two LEDs. And so on to account for all the desired states.

**Step 4: Ultrasonic Rangefinder Arduino Code Explained**

When the code editor is open, you can click the dropdown menu on the left and select "Blocks + Text" to reveal the Arduino code generated by the code blocks. Follow along as we explore the code in more detail.

int distanceThreshold = 0;

int cm = 0;

int inches = 0;

Before the setup(), we create variables to store the target distance threshold, as well as the sensor value in centimeters (cm) and inches. They're called int because they are integers, or any whole number.

long readUltrasonicDistance(int triggerPin, int echoPin)

{

pinMode(triggerPin, OUTPUT); // Clear the trigger

digitalWrite(triggerPin, LOW);

delayMicroseconds(2);

// Sets the trigger pin to HIGH state for 10 microseconds

digitalWrite(triggerPin, HIGH);

delayMicroseconds(10);

digitalWrite(triggerPin, LOW);

pinMode(echoPin, INPUT);

// Reads the echo pin, and returns the sound wave travel time in microseconds

return pulseIn(echoPin, HIGH);

}

The next section is a special bit of code for reading the ultrasonic distance sensor. It's called a function. So far you are familiar with setup() and loop(), but in this sketch, the function readUltrasonicDistance() is used to describe the sensor code and keep it separate from the main body of the program. The function definition starts with what type of data the function will return, or send back to the main program. In this case the function returns a long, which is a decimal point number with many digits. Next is the name of the function, which is up to you. Then in parentheses are the arguments the function takes. int triggerPin, int echoPin are the variable declarations for your sensor's connection pins. The pin numbers will be specified when you call the function in the main program loop(). Inside the function, these local variables are used to reference the information you passed to it from the main loop (or from another function). The function itself sends a signal through the triggerPin and reports back the time it takes to get the signal back over echoPin.

void setup()

{

Serial.begin(9600);

pinMode(2, OUTPUT);

pinMode(3, OUTPUT);

pinMode(4, OUTPUT);

}

Inside the setup, pins are configured using the pinMode() function. The serial monitor connection is established with Serial.begin. Pins 2, 3, and 4 are configured as outputs to control the LEDs.

void loop()

{

// set threshold distance to activate LEDs

distanceThreshold = 350;

// measure the ping time in cm

cm = 0.01723 \* readUltrasonicDistance(7, 6);

In the main loop, distanceThreshold is set to its target 350cm.

// convert to inches by dividing by 2.54

inches = (cm / 2.54);

Serial.print(cm);

Serial.print("cm, ");

Serial.print(inches);

Serial.println("in");

To convert centimeters to inches, divide by 2.54. Printing to the serial monitor helps you observe the distance change more granularly than the LED states show alone.

if (cm > distanceThreshold) {

digitalWrite(2, LOW);

digitalWrite(3, LOW);

digitalWrite(4, LOW);

}

if (cm <= distanceThreshold && cm > distanceThreshold - 100) {

digitalWrite(2, HIGH);

digitalWrite(3, LOW);

digitalWrite(4, LOW);

}

if (cm <= distanceThreshold - 100 && cm > distanceThreshold - 250) {

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, LOW);

}

if (cm <= distanceThreshold - 250 && cm > distanceThreshold - 350) {

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, HIGH);

}

if (cm <= distanceThreshold - 350) {

digitalWrite(2, HIGH);

digitalWrite(3, HIGH);

digitalWrite(4, HIGH);

}

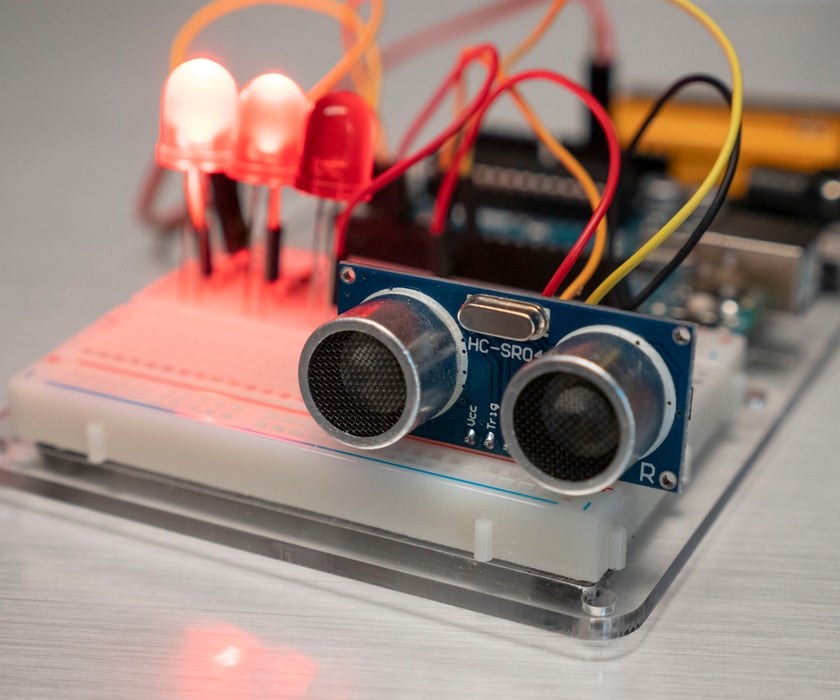
delay(100); // Wait for 100 millisecond(s)

}

The loop's six if statements evaluate for different ranges of distance between 0 and 350cm, lighting up more LEDs the closer the object.

If you want to see a more obvious change in bar graph lights, you can change the distanceThreshold variable and/or the range that you are looking at by changing the arguments in the if() statements. This is called calibration.

**Step 5: Build a Physical Circuit (Optional)**



If you build a physical version of this circuit, you can try it out with the Arduino software's serial monitor (magnifying glass button in the upper right of the sketch window), activating the sensor with your hand, body, notebook, etc.

If using a physical board, put something in front of the sensor and observe the distance reading using the serial monitor, and set distanceThreshold to that value.

Adjust your different distance threshold "buckets" to a range suitable to your initial value, for instance if your hand was 60cm away, your ranges might be 60-40, 40-20, and 20-0.

Upload your code again, and try moving in front of the sensor. As the distance shortens, you should see the LEDs turn on one by one.

**Summary**

Temperature sensors and ultrasonic sensors are quite versatile components. This can be used in many scenarios. The example that we have above is just part of the things that we can do with it. Now, with our understanding of those things, it is time for us to try and revise it and see what we can do to improve it.

**Post-test**

1. How do you define your own function in Arduino?

2. What is *distanceThreshold* variable used for in the second project?

3. What are two parameters that the function *readUltrasonicDistance()*

4. What does delay(100) means?

**Learning Tasks**

**ILO6 :** Based on two projects that we have created on this module, think of a way of how you can innovate it. Please explain why you’ve added that function and how that improves the technology.

**CO2:** Based on the plan that you have created in the past (Module 1), build it using TinkerCad and make a video demonstrating its function and how you build up the code.

**References**

Instructables. (2019a, November 22). *TMP36 Temperature Sensor With Arduino in Tinkercad*. https://www.instructables.com/TMP36-Temperature-Sensor-Arduino-Tinkercad/

Instructables. (2019b, November 22). *Ultrasonic Distance Sensor in Arduino With Tinkercad*. https://www.instructables.com/Ultrasonic-Distance-Sensor-Arduino-Tinkercad/