Introduction to robotics
6th lab

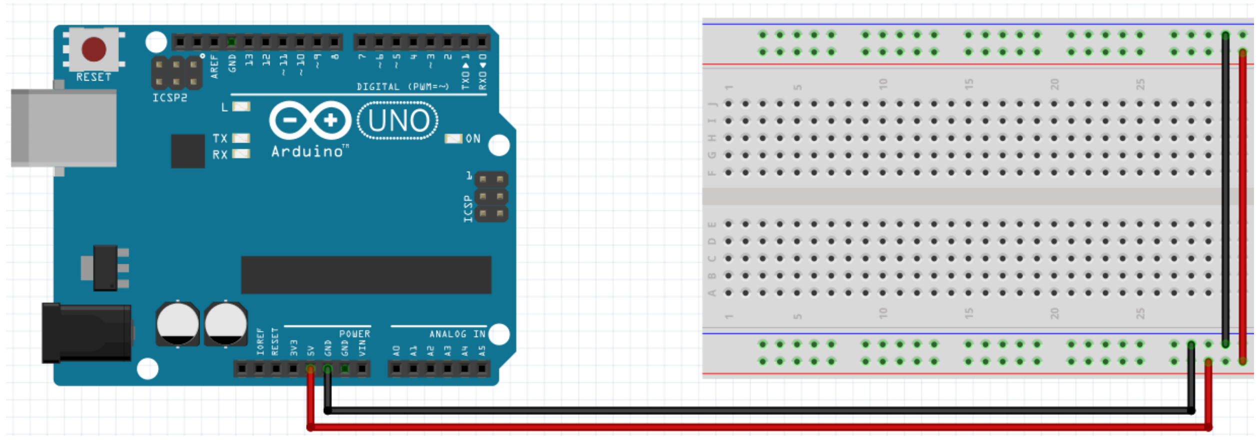
Remember, when possible, choose the wire color accordingly:

- **BLACK** for **GND (dark colors if not available)**
- **RED** for **POWER (3.3V / 5V / VIN) (bright colors if not available)**
- Bright Colored for read and write signal (use **red** when none available and **black** only as a last option)
- We know it is not always possible to respect this due to lack of wires, but the first rule is **DO NOT USE BLACK FOR POWER OR RED FOR GND!**

Now, let's pick it up where we left off…

Pull out your Arduino and breadboard and connect them like in the schematic. This is to "power up" the breadboard so we can easily have access to **5V** and **GND**.

Attention! Remember how the breadboard works. Use correct wire colors.



0. Homework check

Don't be too strict on them regarding how they did it. It is important that they have all the functionalities, but not that they implemented it exactly like we discussed.

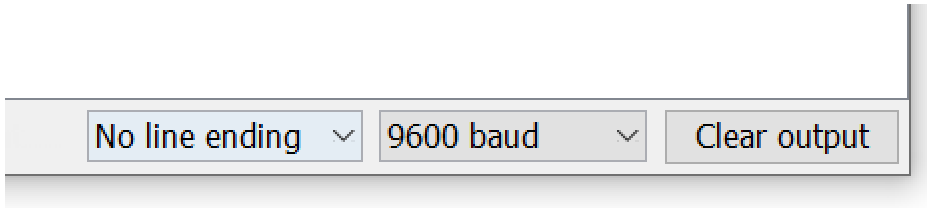
1. Serial Communication

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART), and some have several.

1.1 Introduction

Serial Communication is a method for transferring data over a wire one bit at a time. It's fundamental in microcontroller programming, especially in Arduino projects, for data exchange between the Arduino and a computer or other devices.

Baud Rate: The baud rate is the rate at which information is transferred in a communication channel. In the Serial Communication context, it's the number of symbols per second or pulses per second. For example, a baud rate of 9600 means 9600 bits are transferred per second. It's crucial that both the Arduino and the device it's communicating with agree on the baud rate for successful communication.



```
void setup() {
  // Start serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {
  Serial.println("Hello, world!");
  delay(1000);
}
```

Questions:

1. **What happens if you change the baud rate in the code and/or in the Serial Monitor?**
2. **Why**?

1.2 Serial.read() - reading data from serial

Let’s learn how to read data from the keyboard. Serial Monitor can be used to both send and receive data. Let’s try the following code:

```
int incomingByte = 0; // for incoming serial data

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

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
        // read the incoming byte:
        incomingByte = Serial.read();

        // say what you got:
        Serial.print("I received: ");
        Serial.println(incomingByte);
  }
}
```

First, we check if there is a

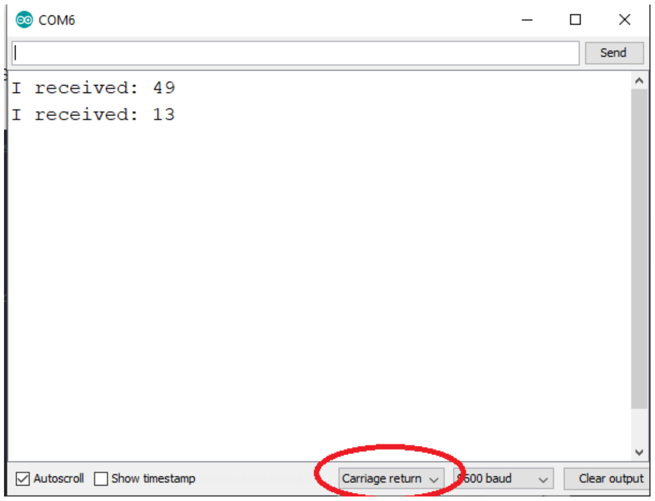
Serial.available()

- Get the number of bytes (characters) available for reading from the serial port. This is data that’s already arrived and stored in the serial receive buffer (which holds 64 bytes).

Serial.read() - Reads incoming serial data.

Questions:

1. **Try writing in Serial Monitor the number “1”. What is the output?**
2. **If you have multiple values, why is that?**



```
I received: 49
I received: 13
```

A carriage return, sometimes known as a cartridge return and often shortened to CR, <CR> or return, is a control character or mechanism used to reset a device's position to the beginning of a line of text. It is closely associated with the line feed and newline concepts, although it can be considered separately in its own right.

(source:https://en.wikipedia.org/wiki/Carriage_return)

Play a bit more with newlines etc.

1.3 (Optional) Using the power of Serial.print(ln)

The received data is printed by default in DEC (the ASCII value) but we can cast to a different type directly or using the power of serial print.

```
int incomingByte = 0; // for incoming serial data

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

void loop() {
 // send data only when you receive data:
 if (Serial.available() > 0) {
	// read the incoming byte:
	incomingByte = Serial.read();

	// say what you got:
	Serial.print("I received (DEC): ");
	Serial.println(incomingByte, DEC);
```

```
        Serial.print("I received (HEX): ");
        Serial.println(incomingByte, HEX);

        Serial.print("I received (OCT): ");
        Serial.println(incomingByte, OCT);

        Serial.print("I received (BIN): ");
        Serial.println(incomingByte, BIN);
  }
}
```

1.4 (Optional) Reading a char and/or a string

So, let’s read the char instead of the ASCII value.

```
char incomingByte = 0; // for incoming serial data

String inputString = "";          // a String to hold incoming data
bool stringComplete = false;  // whether the string is complete

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

void loop() {
 // send data only when you receive data:
 if (Serial.available() > 0) {
        // read the incoming byte:
        incomingByte = (char) Serial.read();
        inputString += incomingByte;
        Serial.print("I received: ");
        Serial.println(incomingByte);
        // if the incoming character is a newline, set a flag so the main loop can
        // do something about it:
        if (incomingByte == '\n') {
         stringComplete = true;
        Serial.println(inputString);
        }
 }
}
```

1.5 Lab Exercise 1: Basic Serial Menu Interface with Arduino

1.5.1 Requirements

Objective:
Create a simple Serial menu interface using Arduino. The program will display a menu in the Serial Monitor and perform actions based on the user's input.
Requirements:
Display a menu with three options.
Read the user's choice via Serial input.
Print a specific message for each menu option.
After performing the action, the menu should be displayed again for the next input.

Example output:

```
Menu Interface:
Select an option:
1. Print a "Hello, World" message
2. Print an "Option 2 selected" message
3. Print a "Custom message of your own"
```

1.5.2 Code template

```
void setup() {
  // TODO: Initialize serial communication at 9600 bps
}

void loop() {
  // TODO: Check if Serial input is available
  // TODO: Read the input and store it in a variable
  // TODO: Call a function to print the message based on the user's choice
  // TODO: Display the menu again
}
void printMenu() {
  // TODO: Print the main menu options to the Serial Monitor
  // The menu should list at least three options for the user to choose from
}
void printMessage(int option) {
  Serial.print("\nPrinted message: ");
  // TODO: Use a switch-case structure to handle different options and print corresponding messages
}
```

1.5.3 Solution

```
void setup() {
  // Initialize serial communication at 9600 bits per second
  Serial.begin(9600);
  // Display the initial menu interface to the Serial Monitor upon startup
  printMenu();
}

void loop() {
  // Continuously checks for incoming serial data
  if (Serial.available()) {
    // Reads an integer value from the serial buffer (user's menu choice)
    int choice = Serial.parseInt();
    // Calls function to print a message based on the user's choice
    printMessage(choice);
    // Re-displays the menu after executing the chosen action
    printMenu();
  }
}

// Function to display a menu of options to the user
void printMenu() {
  Serial.println("Menu Interface:");
  Serial.println("Select an option:");
  Serial.println("1. Print a \"Hello, World\" message");
  Serial.println("2. Print a \"Option 2 selected\" message");
  Serial.println("3. Print a \"Custom message of your own\"");
}

// Function to print different messages based on the user's selection
void printMessage(int option) {
  Serial.print("\nPrinted message: ");
  switch (option) {
    case 1:
      Serial.println("Hello, World\n");
      break;
    case 2:
      Serial.println("Option 2 selected\n");
      break;
    case 3:
      // Prints a placeholder for a custom message for option 3
      Serial.println("Custom message\n");
      break;
    default:
      // Handles any choices outside the defined options
      Serial.println("The selected option is outside the scope of the menu\n");
  }
}
```

2. EEPROM

https://docs.arduino.cc/learn/programming/eeprom-guide

2.1 Introduction

The microcontroller on the Arduino board (ATMEGA328 in case of Arduino UNO) has EEPROM (Electrically Erasable Programmable Read-Only Memory). This is a small space that can store byte variables. The variables stored in the EEPROM are kept there, even when you reset or power off the Arduino. Simply, the EEPROM is permanent storage similar to a hard drive in computers.

The EEPROM can be read, erased and rewritten electronically. In Arduino, you can read and write from the EEPROM easily using the EEPROM library.

The microcontroller on the Arduino Uno board has 1024 bytes (1kB) of EEPROM: memory whose values are kept when the board is turned off (like a tiny hard drive).

Functions in the EEPROM class are automatically included with the platform for your board, meaning you do not need to install any external libraries.

You can easily read and write into the EEPROM using the EEPROM library.
To include the EEPROM library:

```
#include <EEPROM.h>
```

2.2 EEPROM Read

To read a byte from the EEPROM, you use the EEPROM.read() function. This function takes the address of the byte as an argument.

```
EEPROM.read(address);
```

For example, to read the byte stored previously in address 0.:

```
EEPROM.read(0);
```

This would return the value which is stored in that location.

```
#include <EEPROM.h>

byte brightness = 0;        //Create an empty variable

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    // wait for user to open serial monitor
  }
}

void loop() {
  // read a byte from the address "0" of the EEPROM
  brightness = EEPROM.read(0);
  Serial.println(brightness);
  delay(500);
}
```

Questions:

1. **What value do you get? Why?**

2.3 EEPROM Write (Do not run this code. Always use update instead!)

The EEPROM has a finite life. In Arduino, the EEPROM is specified to handle 100 000 write/erase cycles for each position. However, reads are unlimited. This means you can read from the EEPROM as many times as you want without compromising its life expectancy. That is why, when possible, we use update() instead of write()!

To write data into the EEPROM, you use the EEPROM.write() function that takes in two arguments. The first one is the EEPROM location or address where you want to save the data, and the second is the value we want to save:

```
EEPROM.write(address, value); // DO NOT ACTUALLY USE
```

For example, to write 9 on address 0, you'll have:

```
EEPROM.write(0, 9);
```

```
#include <EEPROM.h> //include the library

byte brightness;        //Create an empty integer variable

void setup() {
  Serial.begin(9600);      //Start serial monitor
  EEPROM.write(0, 25);  // write value 25 on address 0
  //Value must be between 0 and 255
}

void loop() {
  // read a byte from the address "0" of the EEPROM
  brightness = EEPROM.read(0);
  Serial.println(brightness);        //Print it to the monitor
  delay(500);
}
```

2.4 EEPROM Update

The EEPROM.update() function is particularly useful. It only writes on the EEPROM if the value written is different from the one already saved.

As the EEPROM has limited life expectancy due to limited write/erase cycles, using the EEPROM.update() function instead of the EEPROM.write() saves cycles.

You use the EEPROM.update() function as follows:

```
EEPROM.update(address, value);
```

At the moment, we have 9 stored in the address 0. So, if we call:

```
EEPROM.update(0, 9);
```

It won't write on the EEPROM again, as the value currently saved is the same we want to write.

```
#include <EEPROM.h> //include the library

byte brightness;       //Create an empty integer variable

void setup() {
 Serial.begin(9600);      //Start serial monitor
 EEPROM.update(1, 25);  // write value 25 on address 0
 //Value must be between 0 and 255
}

void loop() {
 // read a byte from the address "0" of the EEPROM
 brightness = EEPROM.read(1);
 Serial.println(brightness);        //Print it to the monitor
 delay(500);
}
```

2.5 EEPROM GET

While reading and writing a byte is fine, we need to work with data that is more than 1 byte. Read any data type or object from the EEPROM.

```
EEPROM.get(address, data)
```

Parameters
address: the location to read from, starting from 0 (int)
data: the data to read, can be a primitive type (eg. float) or a custom struct

2.5.1 Read (get) an int

```
#include <EEPROM.h>

int brightness;       //Create an empty variable, type: integer

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

void loop() {
  // read an integer starting on address "0" and store the value on "brightness"
  EEPROM.get(1, brightness);    //EEPROM.get(start address, variable);
  Serial.println(brightness);
  delay(500);
}
```

Questions:

1. **You just wrote "25" to byte 1. Now you get a different value, why?**

2.6 EEPROM Put

The purpose of this example is to show the EEPROM.put() method that writes data on EEPROM using also the EEPROM.update() that writes data only if it is different from the previous content of the locations to be written. The number of bytes written is related to the data type or custom structure of the variable to be written.

```
EEPROM.put(address, data)
```

Parameters
address: the location to write to, starting from 0 (int)
data: the data to write, can be a primitive type (eg. float) or a custom struct

2.6.1 Writing (putting) and reading (getting) a float value

```
#include <EEPROM.h>
float brightness;              //Create an empty variable, type: float

void setup() {
  Serial.begin(9600);
  float sensorRead = 56.7;      //Define a variable as float
  EEPROM.put(0, sensorRead);    //Write that value starting on address 0
}

void loop() {
  // read a float starting on address "0" and store the value on "brightness"
  EEPROM.get(0, brightness);
  Serial.println(brightness);
  delay(500);
}
```

EEPROM.put() takes care of writing the entire float (all 4 bytes) starting from the specified address. The get() function can then be used to read the float back correctly.

Questions:

1. **What value do you get if you just read an int from address 1? Last we modified, it was 25. (code from 2.2)**

2.7 EEPROM Iteration

The purpose of this example is to show how to go through the whole EEPROM memory space with different approaches.

```
#include <EEPROM.h>

byte currentValue = 0;

void setup() {
 Serial.begin(9600);
 for (int index = 0 ; index < EEPROM.length() ; index++) {
   // EEPROM[index] += 1; // increment the values of each cell by 1
   currentValue = EEPROM.read(index);
   Serial.print("EEPROM[");
   Serial.print(index);
   Serial.print("] = ");
   Serial.println(currentValue);
 }
}

void loop() {}
```

2.8 Lab exercise 2: EEPROM Counter

2.8.1 Requirements

Create a program that increases a counter stored in EEPROM every time the Arduino restarts. Display the current count via Serial Monitor.

2.8.2. Solution

```
#include <EEPROM.h>

int counterAddress = 0; // Address in the EEPROM to store the counter

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

  // Read the current counter value from EEPROM
  int counter = EEPROM.read(counterAddress);

  // Increase the counter by 1
  counter++;

  // Write the new counter value back to EEPROM
  EEPROM.update(counterAddress, counter);

  // Display the current counter value
  Serial.print("Times this arduino board has been reset (since counting): ");
  Serial.println(counter);
}

void loop() {
  // No need to repeat any action here
}
```

2.9 !important

- Don’t write multiple values on the same address, otherwise you will lose the previously written number (unless that’s what you want to do)
- A memory location can only store one byte of data. So, for numbers between 0 and 255, that’s fine, but for other numbers you’ll have to split the number in several bytes, and store each byte separately. For example, a double value in Arduino Uno takes 4 bytes. Also, that means that you can only store 1024/4 = 256 double values in the EEPROM memory.
- Don’t write a value to the EEPROM inside an infinite loop without any delay or check for user input. **Remember, you only have about 100.000 write cycles available per address.** If you just write to EEPROM in the loop() function with no other code, you might destroy your EEPROM storage pretty fast.

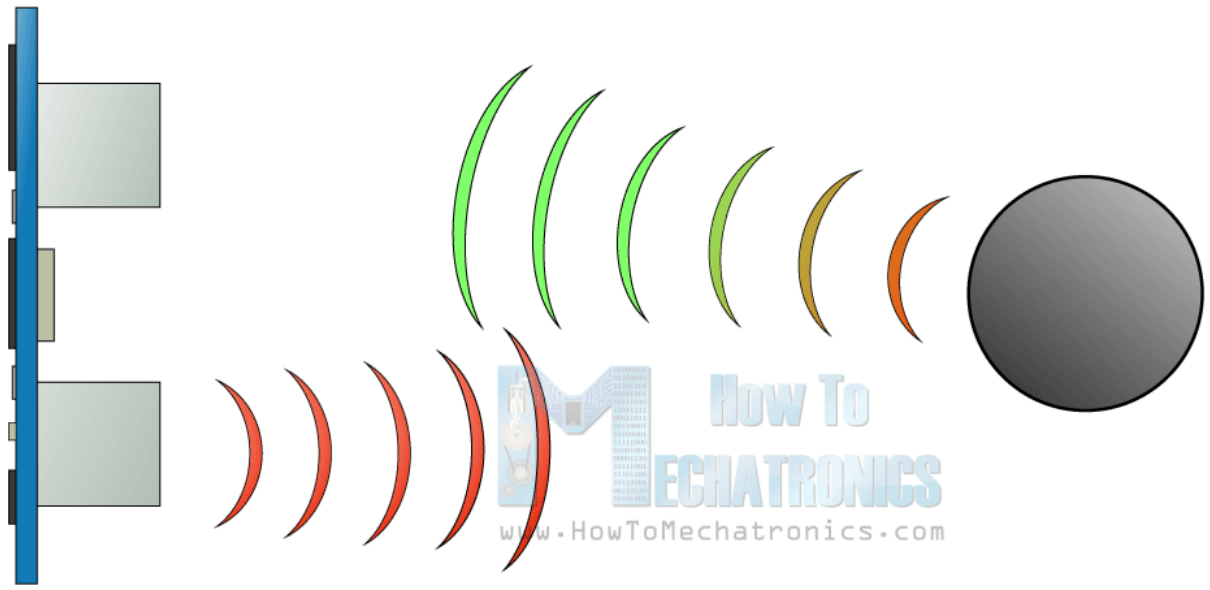
3. Sensors

3.1 Distance sensors: Ultrasonic Sensor HC-SR04

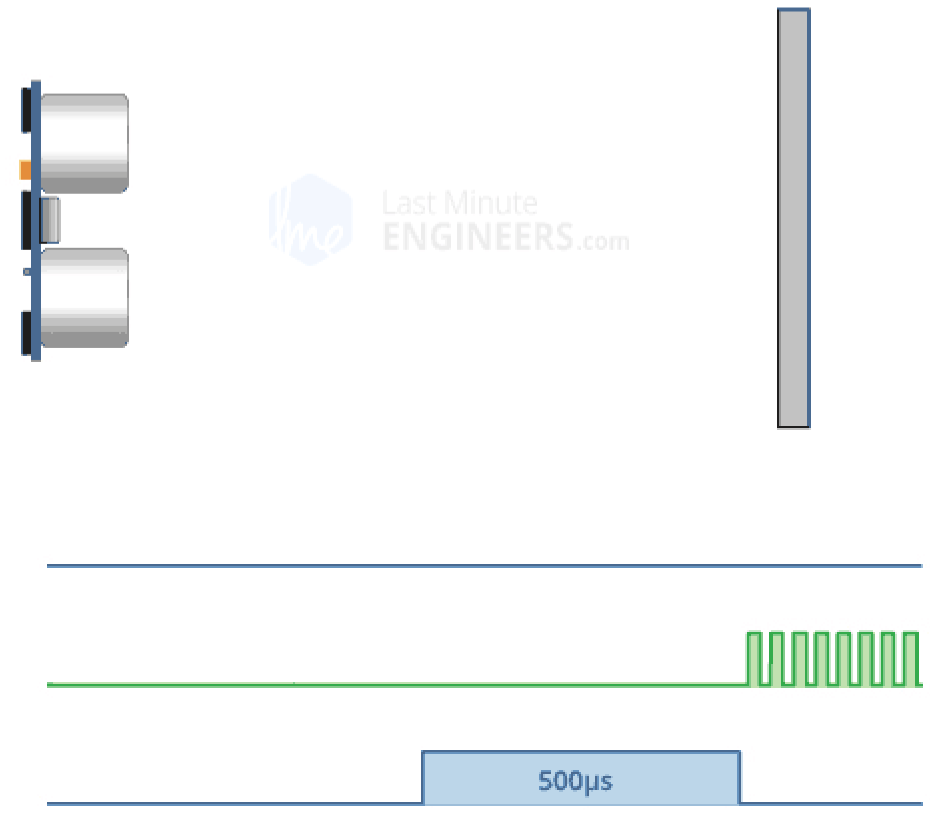
We will learn how the HC-SR04 Ultrasonic Sensor works and how to use it with the Arduino Board. You can watch the following video as well: https://www.youtube.com/watch?v=ZejQOX69K5M

https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/

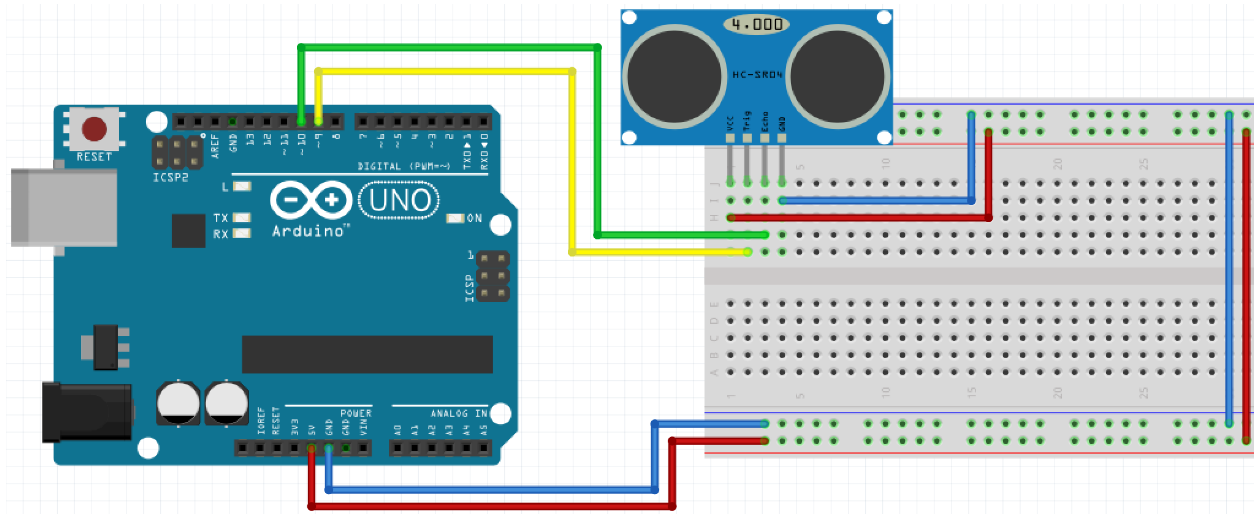
It emits an ultrasound at 40 000 Hz which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.



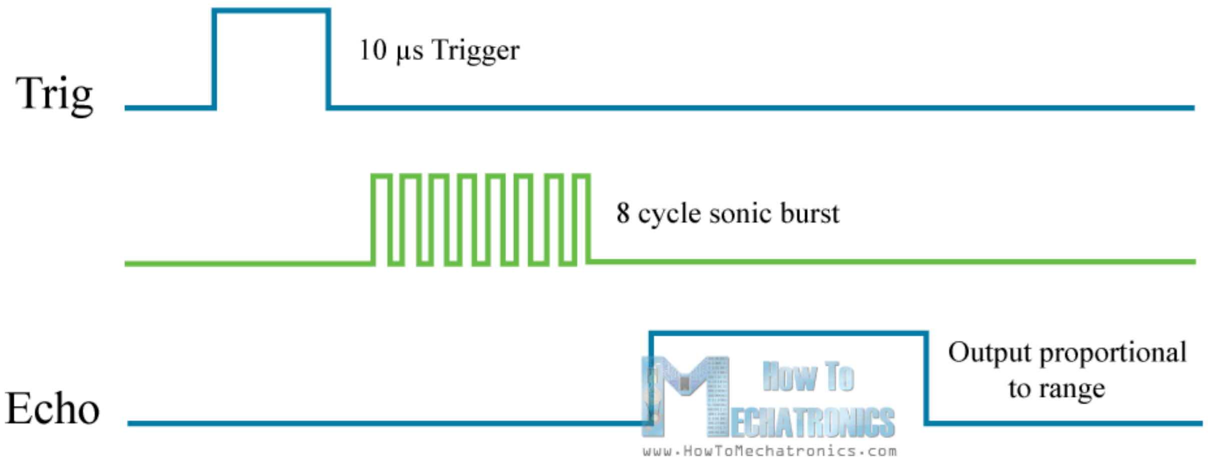
The HC-SR04 Ultrasonic Module has 4 pins, Ground, VCC, Trig and Echo. The Ground and the VCC pins of the module need to be connected to the Ground and the 5 volts pins on the Arduino Board respectively and the trig and echo pins to any Digital I/O pin on the Arduino Board.



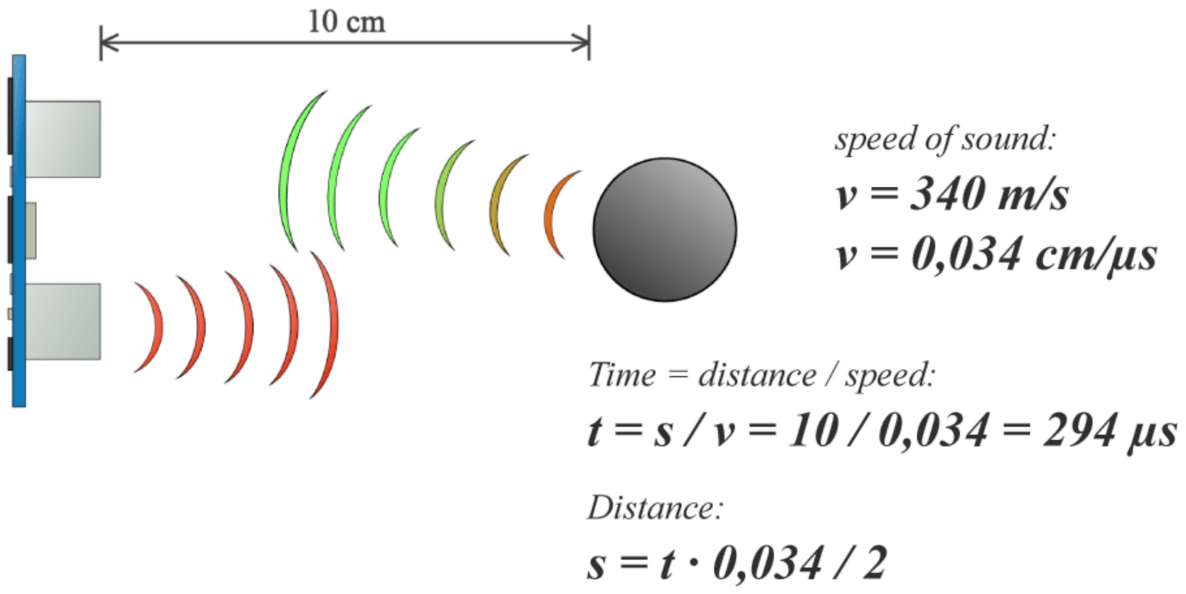
source: https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/



In order to generate the ultrasound you need to set the Trig on a High State for 10 µs. That will send out an 8 cycle sonic burst which will travel at the speed of sound and it will be received in the Echo pin. The Echo pin will output the time in microseconds the sound wave traveled.



For example, if the object is 10 cm away from the sensor, and the speed of the sound is 340 m/s or 0.034 cm/µs the sound wave will need to travel about 294 u seconds. But what you will get from the Echo pin will be double that number because the sound wave needs to travel forward and bounce backward. So in order to get the distance in cm we need to multiply the received travel time value from the echo pin by 0.034 and divide it by 2.



3.2 Operation of the HC-SR04 Ultrasonic Sensor:

1. Triggering the Sensor: The sensor is triggered when a high pulse of at least 10 microseconds is sent to the trigPin.
2. Emitting Ultrasonic Pulses: Upon receiving the trigger signal, the sensor emits a series of 8 ultrasonic pulses at 40 kHz.
3. Switching to Receive Mode: Immediately after emitting the pulses, the sensor switches to receive mode to listen for the echo.
4. Echo Detection: The sensor detects the echo of the ultrasonic pulses when they bounce off an object and return.
5. Setting the echoPin HIGH: The sensor sets the echoPin HIGH upon detecting the echo, marking the beginning of the echo reception.
6. Measuring Echo Duration: The echoPin remains HIGH for the duration it takes for the ultrasonic pulses to travel to the object and back.
7. Setting the echoPin LOW: Once the echo is fully received, the sensor sets the echoPin LOW, indicating the end of the echo detection phase.
8. Distance Calculation: The duration for which the echoPin stays HIGH (measured using the pulseIn function in Arduino) corresponds to the round-trip time of the ultrasonic pulses. This duration is used to calculate the distance to the object, based on the known speed of sound.

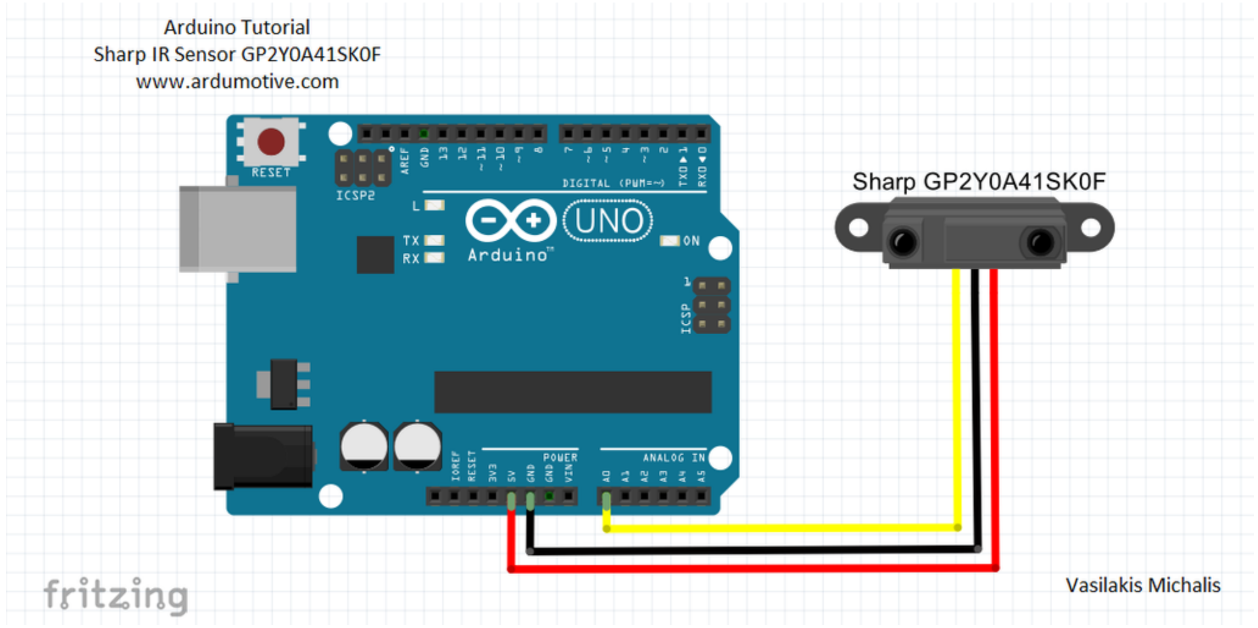
```
const int trigPin = 9;
const int echoPin = 10;

long duration = 0;
int distance = 0;

void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}
void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Sound wave reflects from the obstacle, so to calculate the distance we
  // consider half of the distance traveled.
  distance = duration*0.034/2;
  // Prints the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.println(distance);
}
```

The HC-SR04's electronic circuitry is designed to measure the time interval between the emission of the ultrasonic pulses and the reception of the echo. This timing is converted into the length of the HIGH pulse on the echoPin.

3.2 Infrared sensor (sharp)

The Sharp GP2Y0A41SK0F is an infrared (IR) distance sensor, widely used in robotics and other applications for non-contact distance measurements. Unlike the ultrasonic sensors like the HC-SR04, the Sharp IR sensor uses infrared light to determine the distance to an object.

We will use the Sharp IR sensor (GP2Y0A41SK0F) to measure distance from an object (4~30cm).



Source:https://www.instructables.com/How-to-Use-the-Sharp-IR-Sensor-GP2Y0A41SK0F-Arduin/

Just reading the raw value

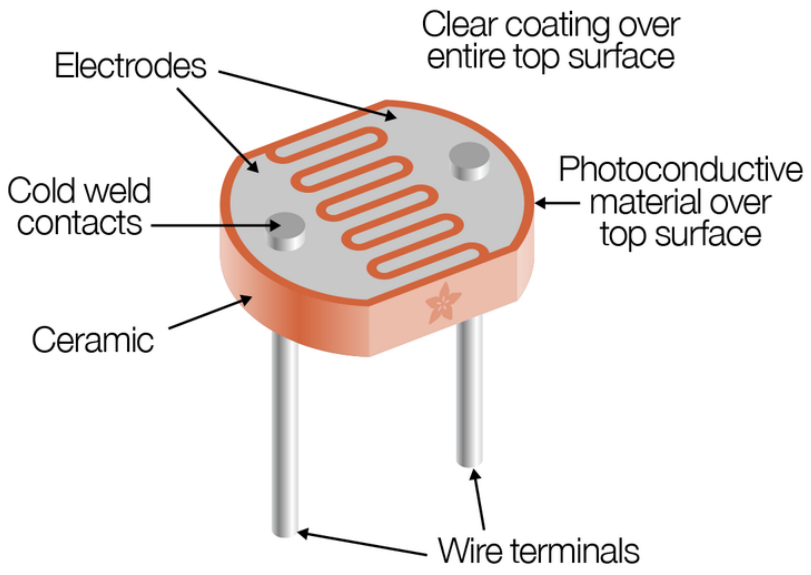
```
const int irSensorPin = A0;
float irSensorValue = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  float irSensorValue = analogRead(irSensorPin);
  delay(1000);
  Serial.println(irSensorValue);
}
```

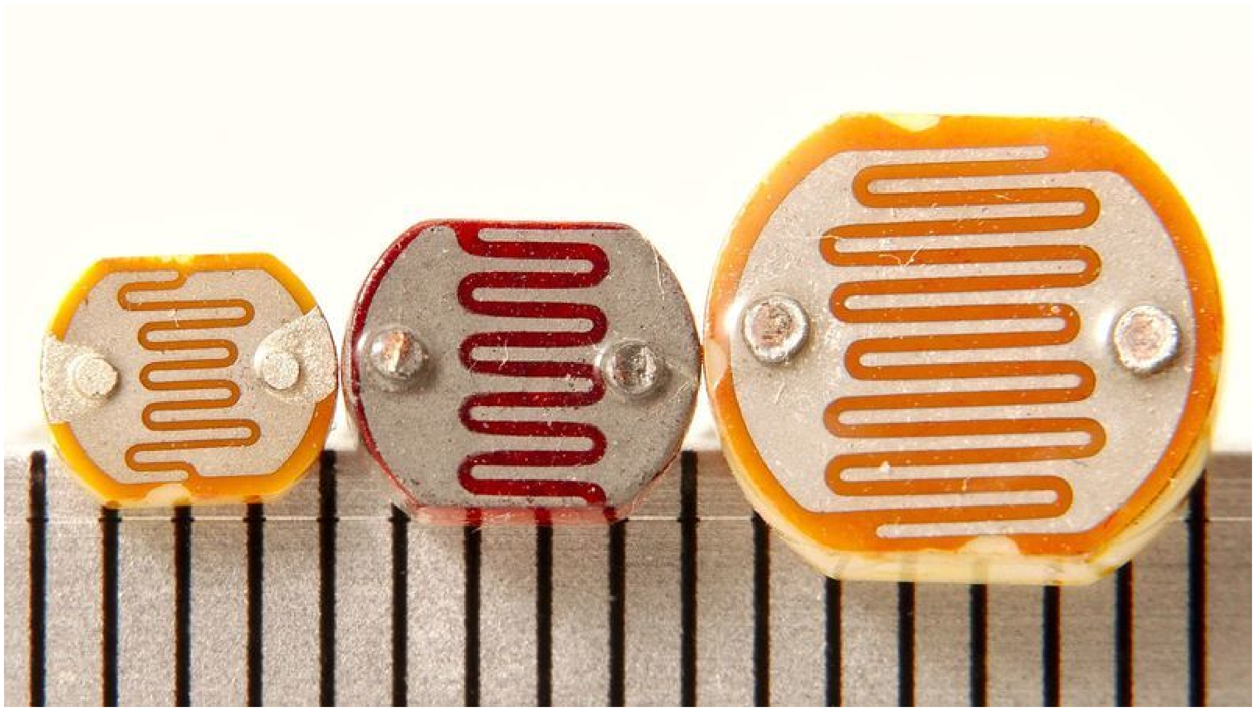
Processing the value into distance. Keep in mind that this is the formula for the current sensor, extracted from the datasheet. The specification changes with each sensor type.

```
const int irSensorPin = A0;

float irSensorvalue = 0;
float volts = 0.0;
int distance = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  float irSensorValue = analogRead(irSensorPin);
  float voltage = irSensorValue * 5 / 1024; // switching from 0..1023 to 0..5V
  float distance = 13 * pow(voltage, -1); // extracted from the datasheet
  delay(100);
  Serial.println(distance);
}
```

3.2 Photocells

Photocells are sensors that allow you to detect light. They are small, inexpensive, low-power, easy to use and don't wear out. For that reason they often appear in toys, gadgets and appliances. They are often referred to as CdS cells (they are made of Cadmium-Sulfide), light-dependent resistors (LDR), and photoresistors.


Clear coating over entire top surface
Electrodes
Cold weld contacts
Photoconductive material over top surface
Ceramic
Wire terminals

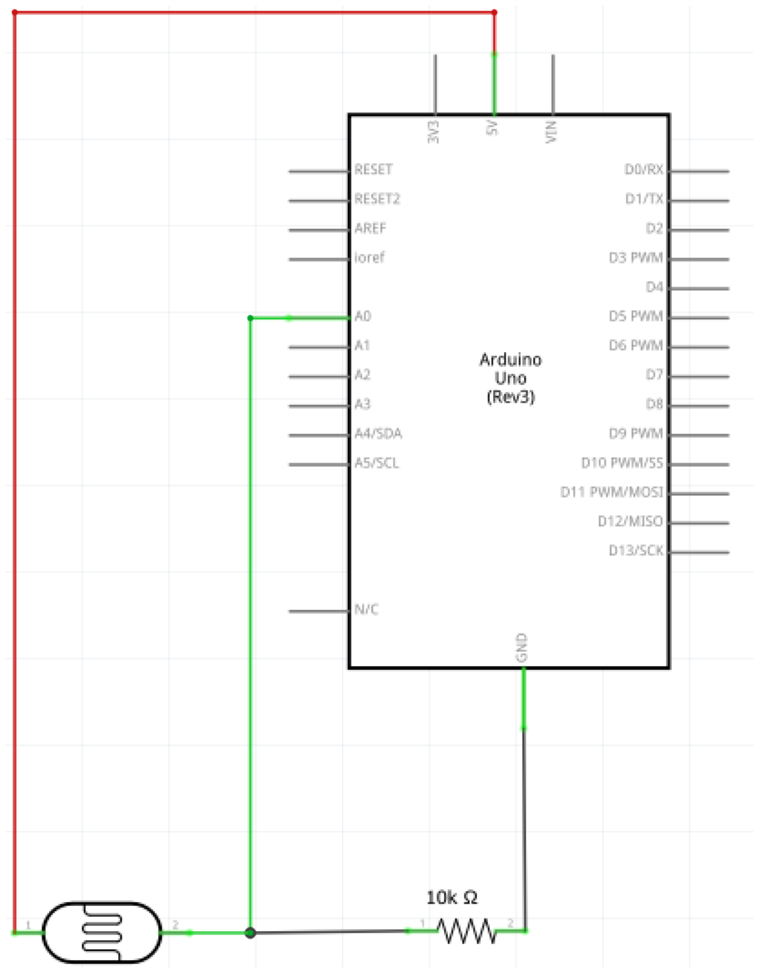


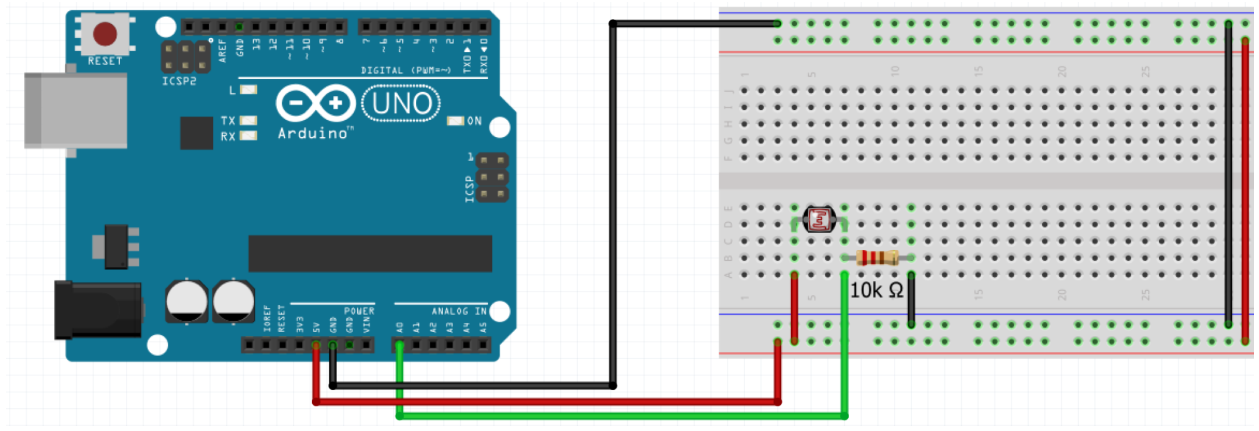
Source: https://learn.adafruit.com/photocells/

3.2.2 Testing a Photocell

Because photocells are basically resistors, they are non-polarized. That means you can connect them up 'either way' and they'll work just fine! The quickest way to test a photocell is to use a multimeter. Switch a multimeter on and put it on a 20k or 200k setting. See how the measured resistance value changes depending on how much light reaches the photocell.

3.3.3 Connecting a Photocell





```
int photocellPin = 0;     // the cell and 10K pulldown are connected to a0
int photocellValue;     // the analog reading from the sensor divider

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

void loop(void) {
  photocellValue= analogRead(photocellPin);

  Serial.print("Analog reading = ");
  Serial.println(photocellValue);     // the raw analog reading

  delay(100);
}
```

3.3.4 Lab exercise 3: turn on a LED as the lights go down

Add an LED to pin 11. Use the reading of the photocell to turn on the LED inverse proportional to the light in the room.

Solution:

```
int photocellPin = 0;     // the cell and 10K pulldown are connected to a0
int photocellValue = 0;     // the analog reading from the sensor divider

int ledPin = 11;
int ledValue = 0;

void setup(void) {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop(void) {
  photocellValue = analogRead(photocellPin);

  Serial.print("Analog reading = ");
  Serial.println(photocellValue);     // the raw analog reading

  // it is useful to invert the reading so that we get a higher value when there is less light
  photocellValue = 1023 - photocellValue;

  // we need to map the value from 0..1023 to 0..255
  ledValue = map(photocellValue, 0, 1023, 0, 255);
  analogWrite(ledPin, ledValue);
  delay(100);
}
```

Extra Exercise: Menu with submenu items

Create a menu with submenu items.

Requirements

Objective:
Create an Arduino program that provides an interactive menu through the Serial Monitor. The menu will allow users to print predefined and custom messages, view sensor readings, and manually control an LED.
Requirements:
Display a menu with three options, each with other suboptions. Make sure you have all the appropriate connections.

1. Print messages
2. Print sensor values.
3. Manually control an LED

After a message is printed, go back to the main menu.
When printing sensor values, read a keyboard input when to stop and return to main menu
After turning the led on or off, return to the main menu.

Example output:

```
Menu Interface:
  Select an option:
  1 Print Messages
    1.1 Print a "Hello, World" message
    1.2 Print an "Option 2 selected" message
    1.3 Print a "Custom message of your own"
    1.4 Back
  2 See Sensor Values
    2.1 Ultrasonic sensor
    2.2 Sharp sensor
    2.3 LDR Sensor
    2.4 Back
  3 Manually Control the LED
    3.1 TURN ON
    3.2 TURN OFF
    3.3 Back
```

Resources:

1. https://docs.arduino.cc/learn/programming/eeprom-guide
2. https://www.arduino.cc/reference/en/language/functions/communication/serial/
3. https://docs.arduino.cc/learn/programming/memory-guide
4. https://omerk.github.io/lcdchargen/
5. https://electronoobs.com/eng_arduino_tut167.php