Introduction to robotics

6th lab

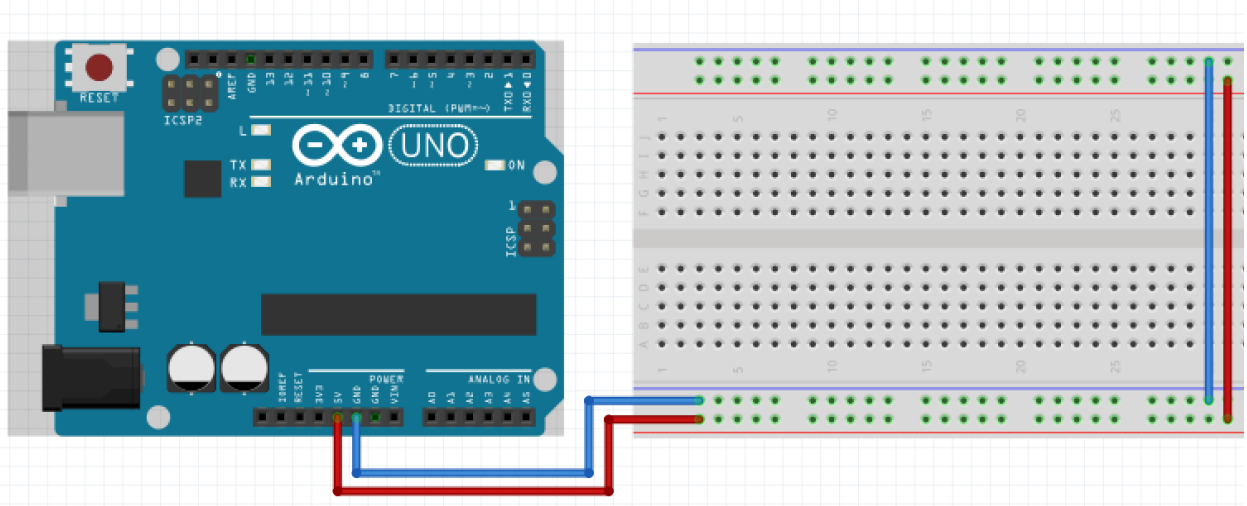
Remember, when possible, choose the wire color accordingly:

- **BLACK** (or dark colors) for **GND**
- **RED** (or colored) for **POWER (3.3V / 5V / VIN)**
- **Remember** than when you use digitalWrite or analogWrite, you actually send power over the PIN, so you can use the same color as for **POWER**
- **Bright Colored** for read signal
- We know it is not always possible to respect this due to lack of wires, but the first rule is **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.



1. LCD Display

1.1 Introduction

Liquid Crystal Display(LCDs) provide a cost effective way to put a text output unit for a microcontroller. As we have seen in the previous tutorial, LEDs or 7 Segments do not have the flexibility to display informative messages. This display has 2 lines and can display 16 characters on each line. Nonetheless, when it is interfaced with the microcontroller, we can scroll the messages with software to display information which is more than 16 characters in length.

The LCD is a simple device to use but the internal details are complex. Most of the 16x2 LCDs use a **Hitachi HD44780** or a compatible controller. Yes, a microcontroller is present inside a Liquid crystal display as shown in figure 2.

The Display Controller takes commands and data from an external microcontroller and drives the LCD panel **(LCDP)**. It takes an ASCII value as input and generates a pattern for the dot matrix. E.g., to display letter 'A', it takes its value **0X42(hex)** or **66(dec)** decodes it into a dot matrix of 5x7 as shown in figure 1.

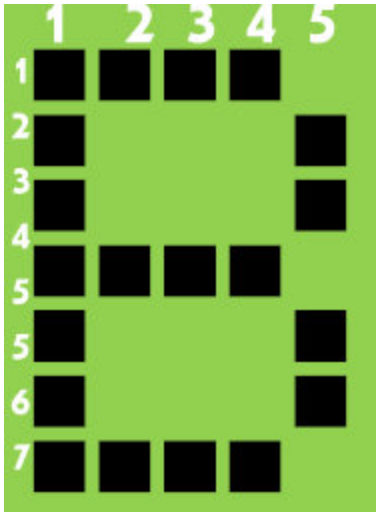
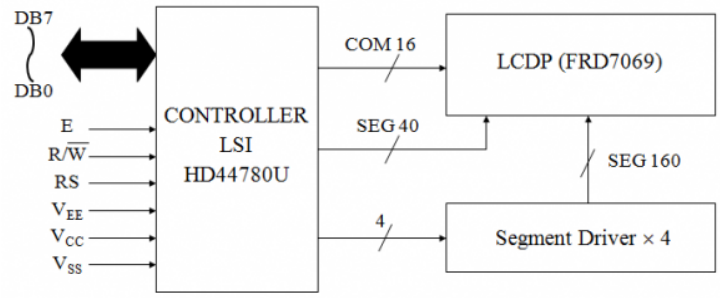


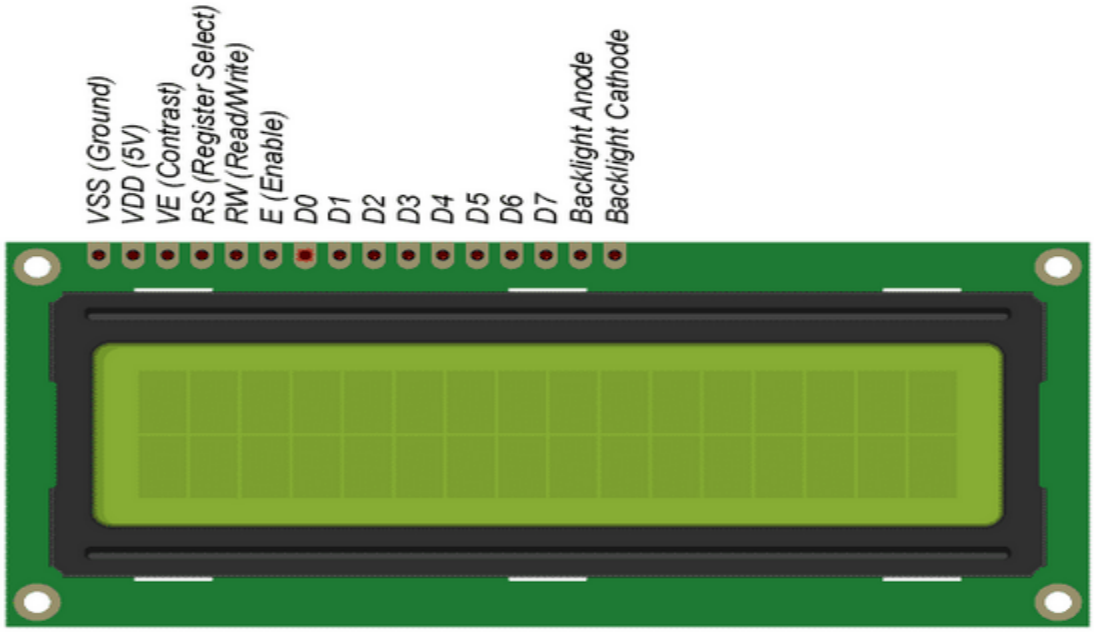
Fig 1: LCD Char 5x7 Matrix
(Each “cell” of the display)

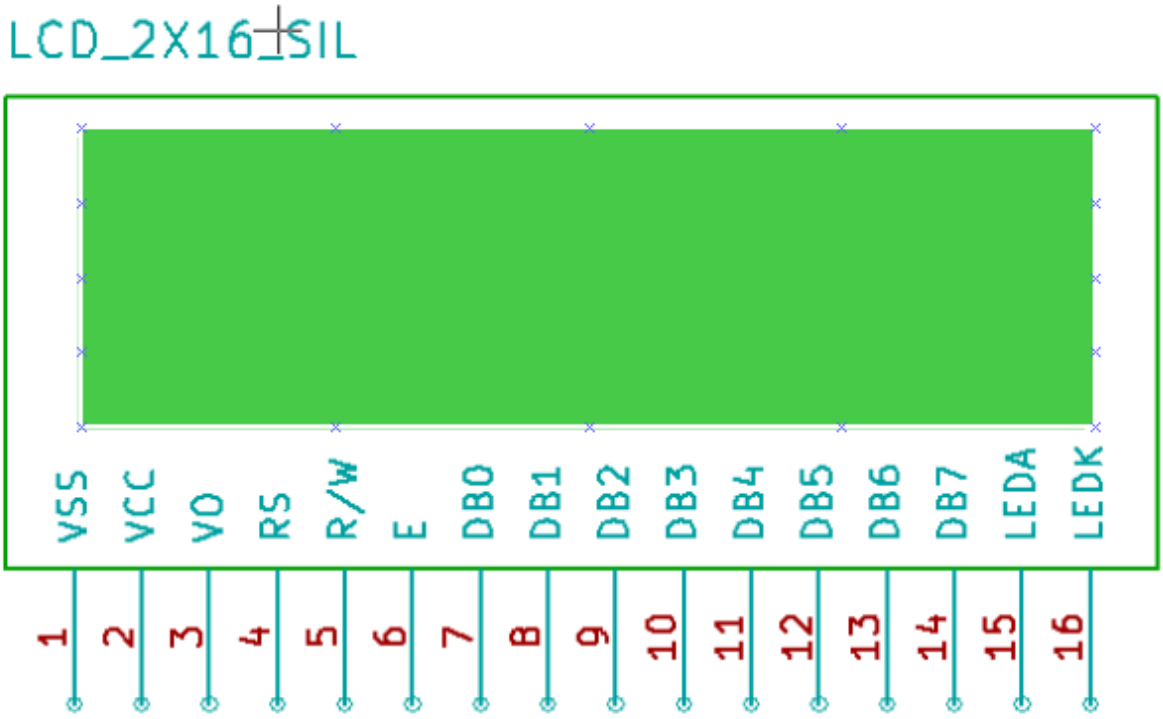
(source:https://exploreembedded.com/wiki/LCD_16_x_2_Basics)



as

Fig 2: LCD Block diagram

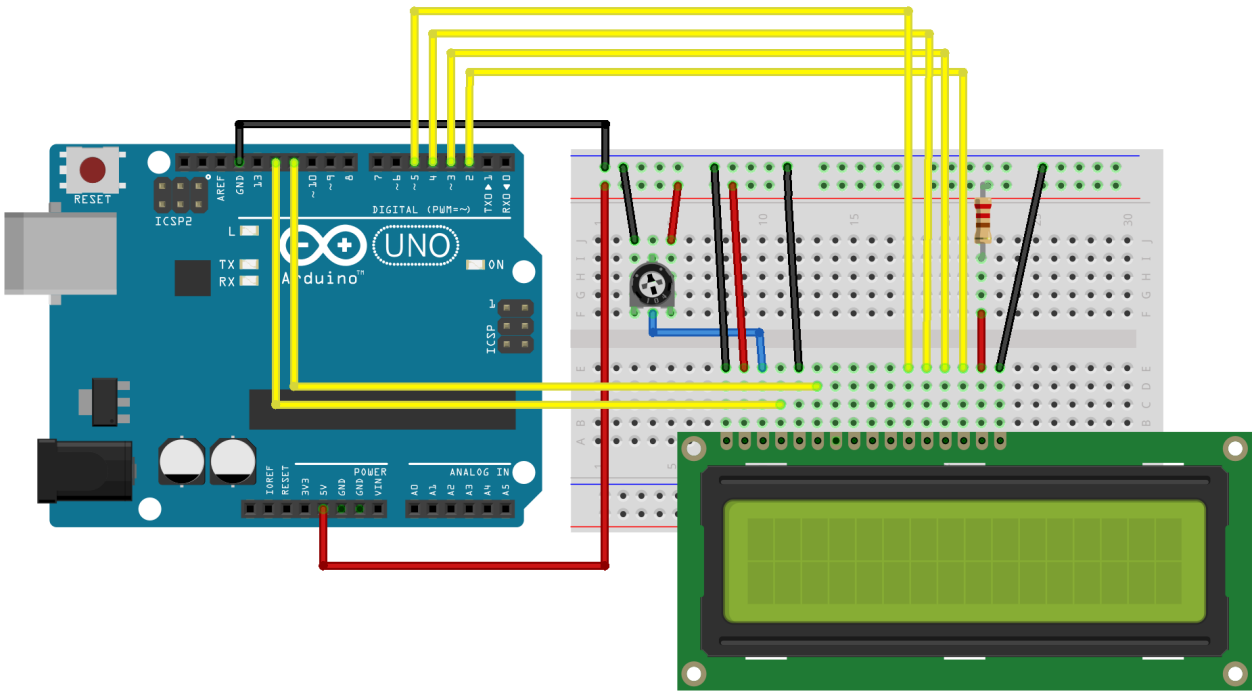




All the pins are identically to the lcd internal controller discussed above

Display PIN	FUNCTION	Arduino PIN
VSS (1)	Ground	GND
VDD (2)	5V	5V
V0 (3)	Contrast adjustment	Potentiometer (analog). PWD after
RS (4)	Register Select. RS=0: Command, RS=1: Data	12
RW (5)	Read/Write (R/W). R/W=0: Write, R/W=1: Read	GND
E (6)	Clock (Enable). Falling edge triggered	11
D0 (7)	Bit 0 (Not used in 4-bit operation)	-
D1 (8)	Bit 1 (Not used in 4-bit operation)	-
D2 (9)	Bit 2 (Not used in 4-bit operation)	-
D3 (10)	Bit 3 (Not used in 4-bit operation)	-
D4 (11)	Bit 4	5
D5 (12)	Bit 5	4
D6 (13)	Bit 6	3
D7 (14)	Bit 7	2
A (15)	Back-light Anode(+)	5V (with 220+ ohm resistor)
K (16)	Back-Light Cathode(-)	GND

Schematic



Resistor is about 220ohms, just like with a regular LED

LCD can be interfaced with the microcontroller in two modes, 8 bit and 4 bit. Today we'll interface with the 4bit. The tradeoff is speed, as we send half as much data in one go, but we gain a reduced number of wires.

Data Lines: In this mode, all of the 8 datalines DB0 to DB7 are connected from the microcontroller to an LCD module as shown in the schematic.
Control Lines:' The RS, RW and E are control lines, as discussed earlier.
Power & contrast: Apart from that the LCD should be powered with 5V between PIN 2(VCC) and PIN 1(gnd). PIN 3 is the contrast pin and is the output of the center terminal of potentiometer(voltage divider) which varies voltage between 0 to 5v to vary the contrast.
Back-light: The PIN 15 and 16 are used as backlights. The led backlight can be powered through a simple current limiting resistor as we do with normal leds.

```
#include <LiquidCrystal.h>
const int RS = 12;
const int enable = 11;
const int d4 = 5;
const int d5 = 4;
const int d6 = 3;
const int d7 = 2;

LiquidCrystal lcd(RS, enable, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis() / 1000);
}
```

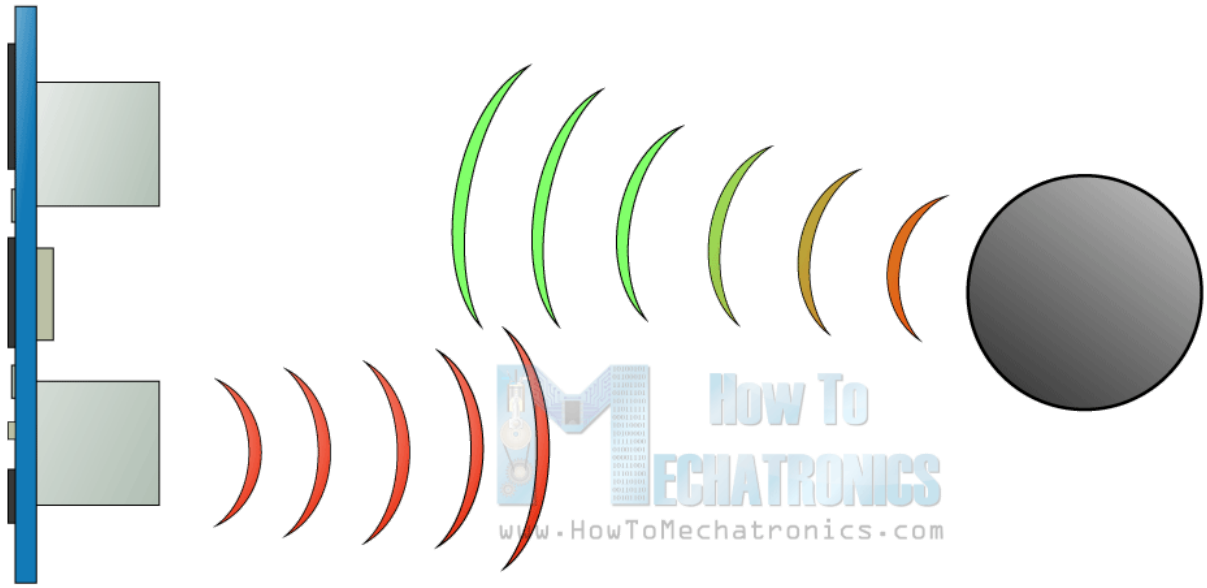
Creating a custom char: https://www.arduino.cc/en/Reference/LiquidCrystalCreateChar

2. Ultrasonic Sensor HC-SR04

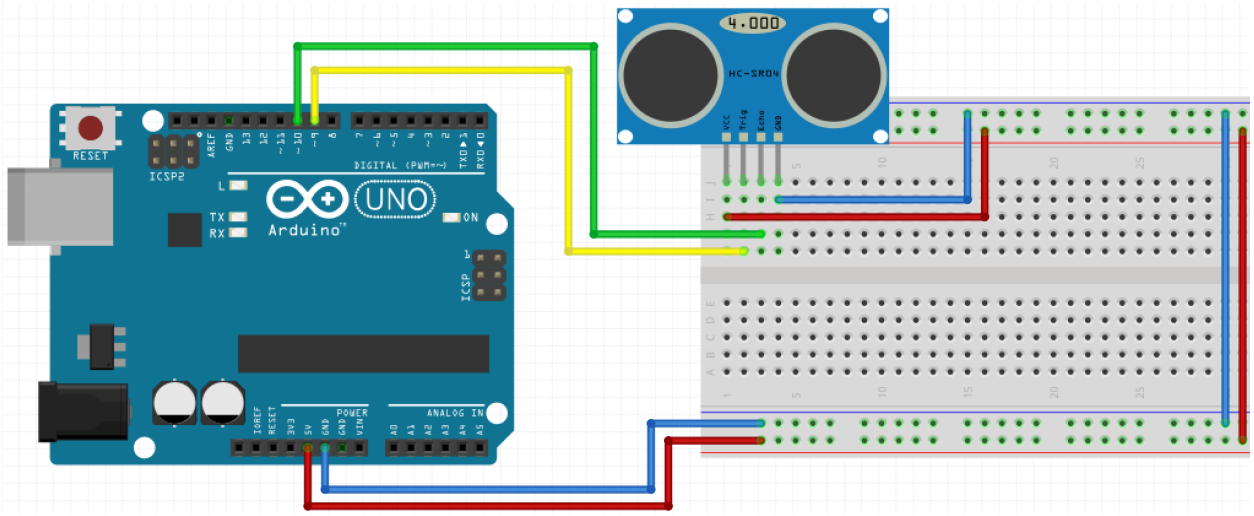
We will learn how the HC-SR04 Ultrasonic Sensor works and how to use it with the Arduino Board. You can watch the following video as well:
https://www.youtube.com/watch?v=ZejQOX69K5M

2.1 How it works

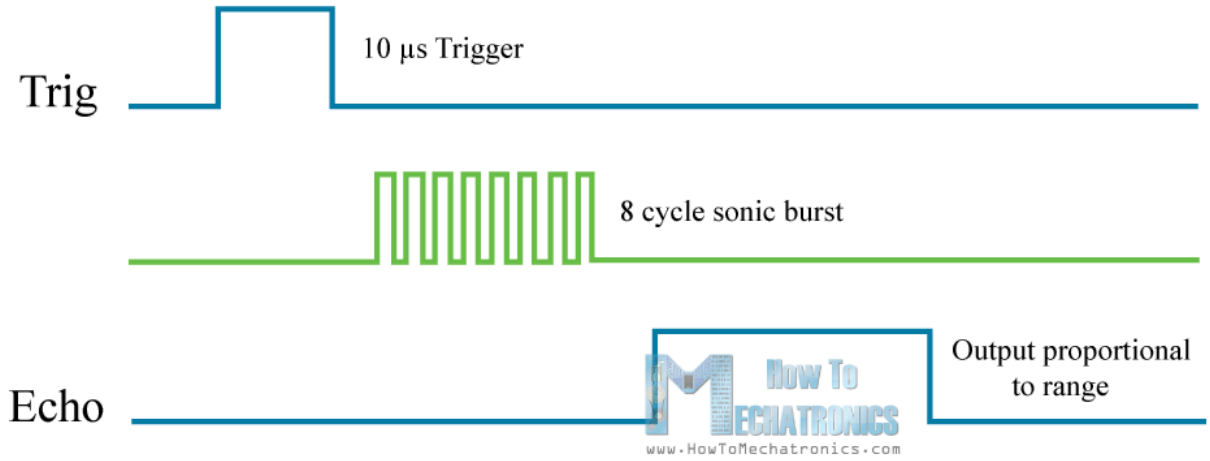
It emits an ultrasound at 40 000 Hz which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.



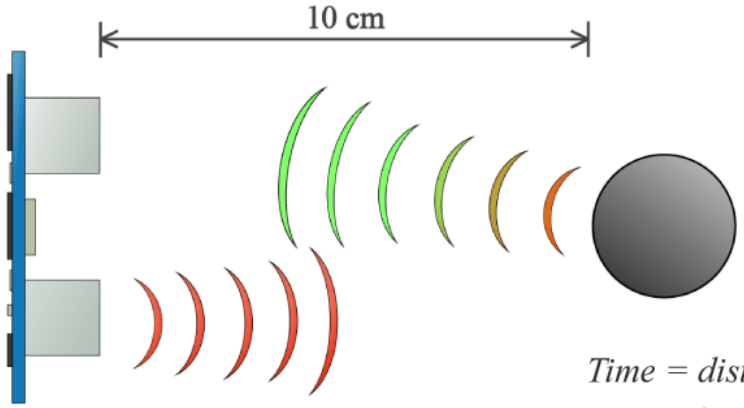
The HC-SR04 Ultrasonic Module has 4 pins, Ground, VCC, Trig and Echo. The Ground and the VCC pins of the module need to be connected to the Ground and the 5 volts pins on the Arduino Board respectively and the trig and echo pins to any Digital I/O pin on the Arduino Board.



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.



speed of sound:

$$v = 340\ m/s$$

$$v = 0{,}034\ cm/\mu s$$

Time = distance / speed:

$$t = s / v = 10 / 0{,}034 = 294\ \mu s$$

Distance:

$$s = t \cdot 0{,}034 / 2$$

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

long duration;
int distance;

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);
}
```

HERE WE PRINT THE VALUE TO THE DISPLAY:

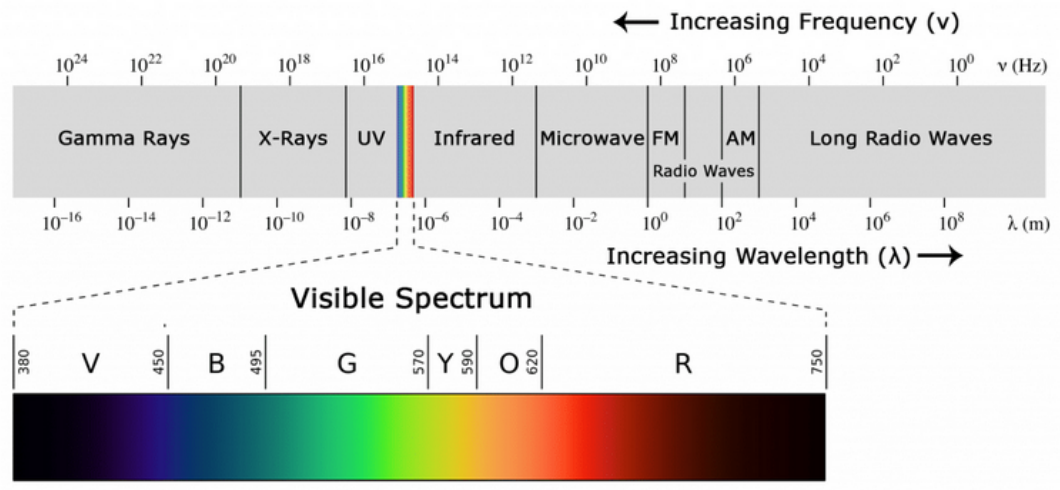
1. SCHEMATIC
2. CODE

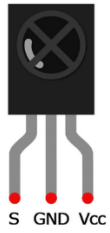
Nu am testat codul inca, dar ar trebui sa mearga. Doar daca sunt probleme de pini.

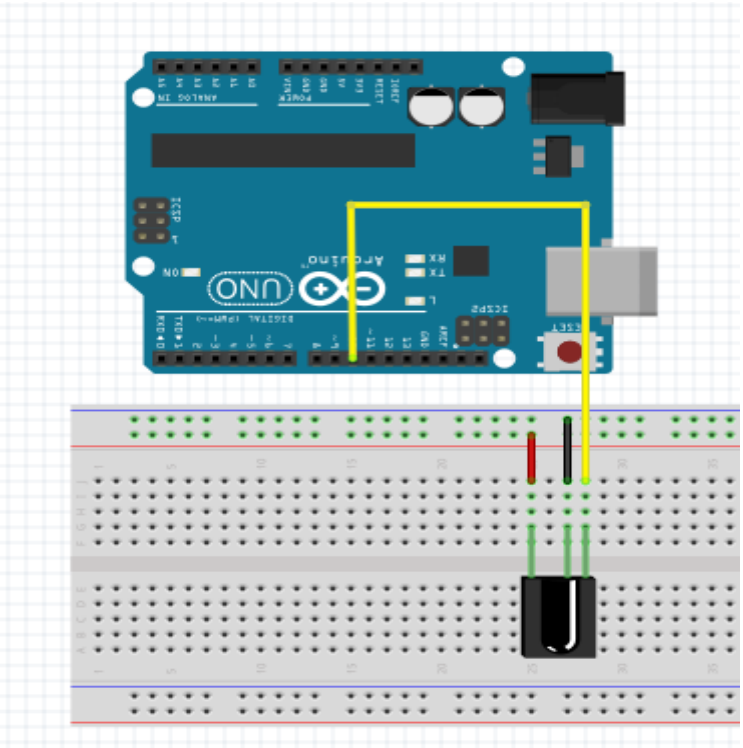
```
#include <LiquidCrystal.h> // includes the LiquidCrystal Library
const int RS = 12;
const int enable = 11;
const int d4 = 5;
const int d5 = 4;
const int d6 = 3;
const int d7 = 2;

LiquidCrystal lcd(RS, enable, d4, d5, d6, d7);
const int trigPin = 9;
const int echoPin = 10;
long duration;
int distanceCm, distanceInch;

void setup() {
  lcd.begin(16,2); // Initializes the interface to the LCD screen, and
specifies the dimensions (width and height) of the display
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
void loop() {
  lcd.clear();
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distanceCm= duration*0.034/2;
  distanceInch = duration*0.0133/2;
  lcd.setCursor(0,0); // Sets the location at which subsequent text written
to the LCD will be displayed
```

```
  lcd.print("Distance: "); // Prints string "Distance" on the LCD
  lcd.print(distanceCm); // Prints the distance value from the sensor
  lcd.print(" cm");
  delay(10);
  lcd.setCursor(0,1);
  lcd.print("Distance: ");
  lcd.print(distanceInch);
  lcd.print(" inch");
  delay(10);
}
```

3. INFRARED(IR)





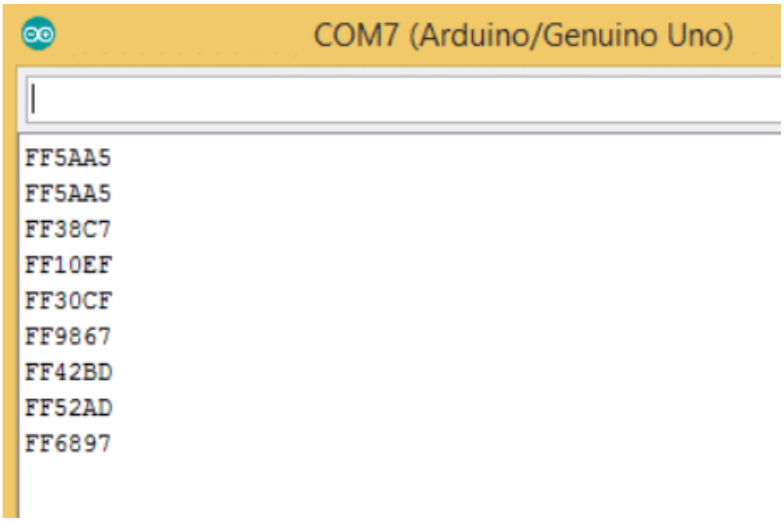


https://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/

We need to download the IR remote library https://www.arduino.cc/reference/en/libraries/irremote/

1. Download ZIP format
2. Link from Arduino IDE Sketch - > Include Library -> Add .ZIP

```
#include <IRremote.h>

const int receiverPin = 10;
IRrecv irrecv(receiverPin);
decode_results results;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  irrecv.blink13(true);
}

void loop(){
  if (irrecv.decode(&results)){
        Serial.println(results.value, HEX);
        irrecv.resume();
  }
}
```



Key	Code
CH-	0xFFA25D
CH	0xFF629D
CH+	0xFFE21D
<<	0xFF22DD
>>	0xFF02FD
>||	0xFFC23D
–	0xFFE01F
+	0xFFA857
EQ	0xFF906F
100+	0xFF9867
200+	0xFFB04F
0	0XFF6897
1	0xFF30CF
2	0xFF18E7
3	0xFF7A85
4	0xFF10EF
5	0xFF38C7
6	0xFF5AA5
7	0xFF42BD
8	0xFF4AB5
9	0xFF52AD

Every remote control uses a specific protocol. It can be useful to identify that protocol in order to know the range of devices to which you have access to.

```
#include <IRremote.h>

const int receiverPin = 10;
IRrecv irrecv(receiverPin);
decode_results results;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  irrecv.blink13(true);
}

void loop(){
  if (irrecv.decode(&results)){
        Serial.println(results.value, HEX);
        switch (results.decode_type){
            case NEC: Serial.println("NEC"); break ;
            case SONY: Serial.println("SONY"); break ;
            case RC5: Serial.println("RC5"); break ;
            case RC6: Serial.println("RC6"); break ;
            case DISH: Serial.println("DISH"); break ;
            case SHARP: Serial.println("SHARP"); break ;
            case JVC: Serial.println("JVC"); break ;
            case SANYO: Serial.println("SANYO"); break ;
            case MITSUBISHI: Serial.println("MITSUBISHI"); break ;
            case SAMSUNG: Serial.println("SAMSUNG"); break ;
            case LG: Serial.println("LG"); break ;
            case WHYNTER: Serial.println("WHYNTER"); break ;
            case AIWA_RC_T501: Serial.println("AIWA_RC_T501"); break ;
            case PANASONIC: Serial.println("PANASONIC"); break ;
            case DENON: Serial.println("DENON"); break ;
          default:
            case UNKNOWN: Serial.println("UNKNOWN"); break ;
          }
        irrecv.resume();
 }
}
```

Now let's print the remote button values

```
#include <IRremote.h>

const int receiverPin = 10;
IRrecv irrecv(receiverPin);
decode_results results;
unsigned long keyValue = 0;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn(); // configure the baud rate and start the IR receiver
  irrecv.blink13(true); // blinks the LED everytime it receives a signal
}

void loop(){
  if (irrecv.decode(&results)){

        if (results.value == 0XFFFFFFFF)
          results.value = keyValue;

        switch(results.value){
          case 0xFFA25D:
          Serial.println("CH-");
          break;
          case 0xFF629D:
          Serial.println("CH");
          break;
          case 0xFFE21D:
          Serial.println("CH+");
          break;
          case 0xFF22DD:
          Serial.println("|<<");
          break;
          case 0xFF02FD:
          Serial.println(">>|");
          break ;
          case 0xFFC23D:
          Serial.println(">|");
          break ;
```

```
        case 0xFFE01F:
        Serial.println("-");
        break ;
        case 0xFFA857:
        Serial.println("+");
        break ;
        case 0xFF906F:
        Serial.println("EQ");
        break ;
        case 0xFF6897:
        Serial.println("0");
        break ;
        case 0xFF9867:
        Serial.println("100+");
        break ;
        case 0xFFB04F:
        Serial.println("200+");
        break ;
        case 0xFF30CF:
        Serial.println("1");
        break ;
        case 0xFF18E7:
        Serial.println("2");
        break ;
        case 0xFF7A85:
        Serial.println("3");
        break ;
        case 0xFF10EF:
        Serial.println("4");
        break ;
        case 0xFF38C7:
        Serial.println("5");
        break ;
        case 0xFF5AA5:
        Serial.println("6");
        break ;
        case 0xFF42BD:
        Serial.println("7");
        break ;
        case 0xFF4AB5:
        Serial.println("8");
        break ;
```

```
          case 0xFF52AD:
          Serial.println("9");
          break ;
        }
        keyValue = results.value;
        irrecv.resume();
  }
}
```

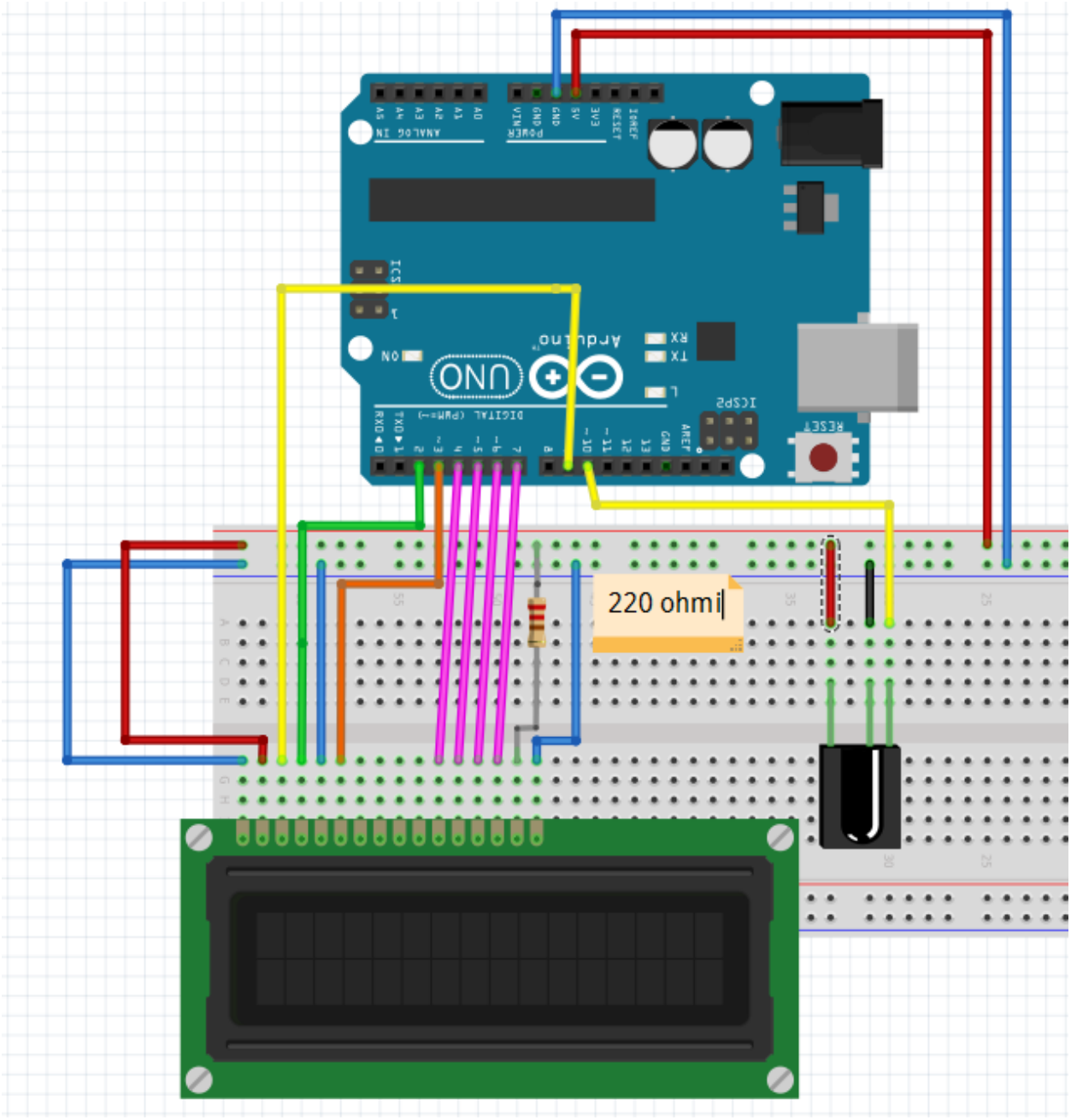
- How does it work?

```
const int receiverPin = 10;
IRrecv irrecv(receiverPin);
decode_results results;
unsigned long keyValue = 0;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  irrecv.blink13(true);
}

void loop(){
  if (irrecv.decode(&results)){
```

For every IR communication that uses the IRremote library we need to create an **irrecv** object and specify the pin on the board to which the receiver is connected. Then we need to create an object named **result** that belongs to the class **decode_result** which will be used by the **irrecv** object in order to communicate with our app.
In the setup function we configure the baud rate and start the IR receiver with the function **enableIRIn().** The function **irrecv.blink13(true)** blinks the LED everytime the receiver receives a signal.
In the **loop(),** the function **irrecv.decode()** will return TRUE if a signal is received and will save the value in **results.value**.
At the end of the loop, **irrecv.resume()** will reset the receiver in order for it to do a new reading.

4. LCD with IR

Printing the IR value on the LCD



```
#include <IRremote.h>
#include <LiquidCrystal.h>

const int receiverPin = 10;
const int V0_PIN = 9; // PWN in loc de POTENTIOMETRU
const int RS = 12;
const int enable = 11;
const int d4 = 5;
const int d5 = 4;
const int d6 = 3;
const int d7 = 2;

LiquidCrystal lcd(RS, enable, d4, d5, d6, d7);
IRrecv irrecv(receiverPin);
decode_results results;
unsigned long keyValue = 0;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  irrecv.blink13(true);
  lcd.begin(16, 2);

  pinMode(V0_PIN, OUTPUT); // PWN in loc de POTENTIOMETRU
  analogWrite(V0_PIN, 90); // PWN in loc de POTENTIOMETRU

}

void loop(){
  if (irrecv.decode(&results)){

        if (results.value == 0XFFFFFFFF)
          results.value = keyValue;
        lcd.setCursor(0, 0);
        lcd.clear();

        switch(results.value){
          case 0xFFA25D:
          lcd.print("CH-");
          break;
          case 0xFF629D:
          lcd.print("CH");
```

```
        break;
        case 0xFFE21D:
        lcd.print("CH+");
        break;
        case 0xFF22DD:
        lcd.print("|<<");
        break;
        case 0xFF02FD:
        lcd.print(">>|");
        break ;
        case 0xFFC23D:
        lcd.print(">|");
        break ;
        case 0xFFE01F:
        lcd.print("-");
        break ;
        case 0xFFA857:
        lcd.print("+");
        break ;
        case 0xFF906F:
        lcd.print("EQ");
        break ;
        case 0xFF6897:
        lcd.print("0");
        break ;
        case 0xFF9867:
        lcd.print("100+");
        break ;
        case 0xFFB04F:
        lcd.print("200+");
        break ;
        case 0xFF30CF:
        lcd.print("1");
        break ;
        case 0xFF18E7:
        lcd.print("2");
        break ;
        case 0xFF7A85:
        lcd.print("3");
        break ;
        case 0xFF10EF:
        lcd.print("4");
```

```
          break ;
          case 0xFF38C7:
          lcd.print("5");
          break ;
          case 0xFF5AA5:
          lcd.print("6");
          break ;
          case 0xFF42BD:
          lcd.print("7");
          break ;
          case 0xFF4AB5:
          lcd.print("8");
          break ;
          case 0xFF52AD:
          lcd.print("9");
          break ;
        }
        keyValue = results.value;
        irrecv.resume();
  }
}
```

5. Serial.read()

5.1 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

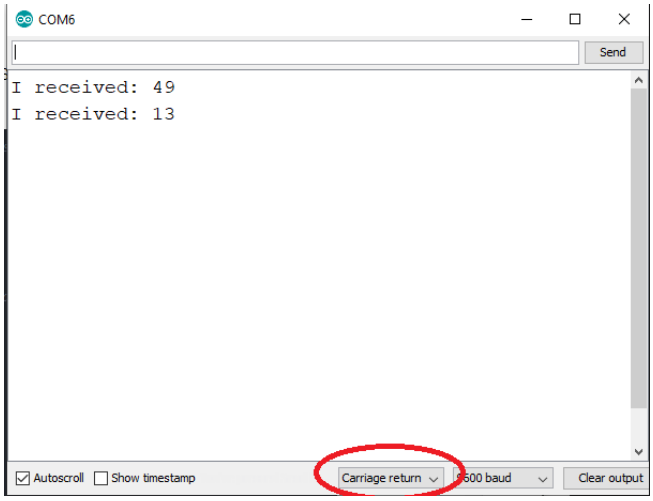
Serial.available()

- 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?**
   a. **A:** First, the ascii value of what they entered. 2ndly, depending if they have newline / carriage return they also have the value for that
2. **If you have multiple values, why is that?**
   a. **A:** Because beside the initial byte, Serial Monitor also adds a newline / carriage return. Switch to no line ending.



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.

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

The received data is printed byte 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);
  }
}
```

5.3 (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);
      }
  }
}
```

6. EEPROM

6.1 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 re-written electronically. In Arduino, you can read and write from the EEPROM easily using the EEPROM library.

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()!

You can easily read and write into the EEPROM using the EEPROM library.
To include the EEPROM library:
#include <EEPROM.h>

6.1 EEPROM 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);
```

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

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

6.2 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 9, which is the value stored in that location.

6.3 EEPROM Read

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.

6.4 Manually write an int (2 bytes) to EEPROM

(check EEPROM.put(), though)

Split the int into 2 bytes and write each of them

```
#include <EEPROM.h>

// address -> 0..1023
void write2BytesEEPROM(int address, int number) {
  // if we have value 10101010 00110011, shifting it goes to 00000000
10101010
  // & oxFF makes sure the number is no larger than 256
  byte byte1 = (number >> 8) & 0xFF; // OxFF = 255 = [...]11111111
  byte byte2 = number & 0xFF;
  EEPROM.update(address, byte1);
  EEPROM.update(address + 1, byte2);
}
int read2BytesEEPROM(int address) {
  byte byte1 = EEPROM.read(address);
  byte byte2 = EEPROM.read(address + 1);
  // byte1 = 10101010 . Shifting it -> 10101010 00000000
  // byte2 = 00110011
  // adding it just gets the original number
  int result = (byte1 << 8) + byte2;
  return result;
```

```
}
void setup() {

  Serial.begin(9600);
//  write2BytesEEPROM(12, 19503);
//  Serial.println(read2BytesEEPROM(12));
Serial.println(255 & 0xFF);
}

void loop() {

}
```

6.5 !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.