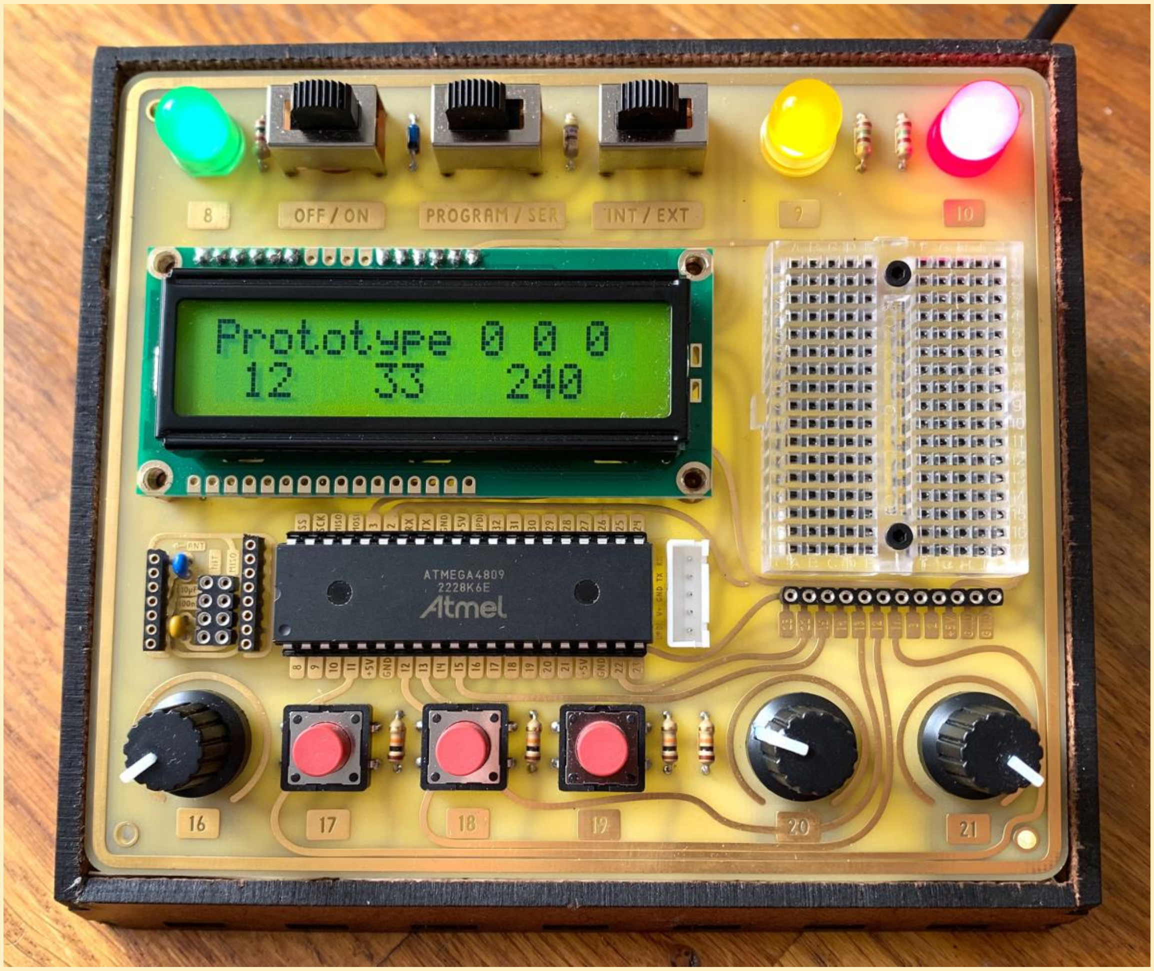
Learn Arduino Using the EZA Board



Available at: EZABoard.com

This manual accompanies the video:

“Learn Arduino: From beginner to wirelessly igniting a model rocket in 12 easy lessons”

Table of Contents

WARNING: When using the external user-accessible pins it is possible to exceed the current rating of your USB port. All computers have some level of protection built in. Some computers shut down when they detect an overcurrent event. It is always best to use a powered USB hub between the EZA Board and your computer.

INSTALLING SOFTWARE AND CONFIGURING THE BOARD

Install Arduino

www.arduino.cc/en/software

Include MegaCoreX

In the Arduino software click on File > Preferences.

In "Additional Board Manager URLs" paste:

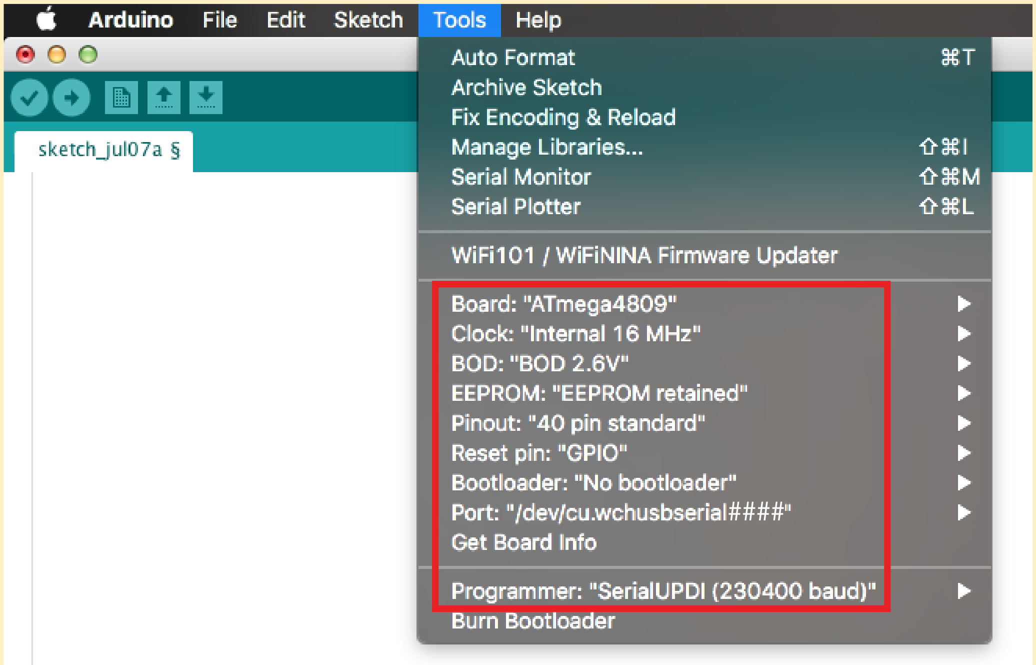
https://mcudude.github.io/MegaCoreX/package_MCUdude_MegaCoreX_index.json

Exit the preferences.

Click on Tools > Board > Board Manager.

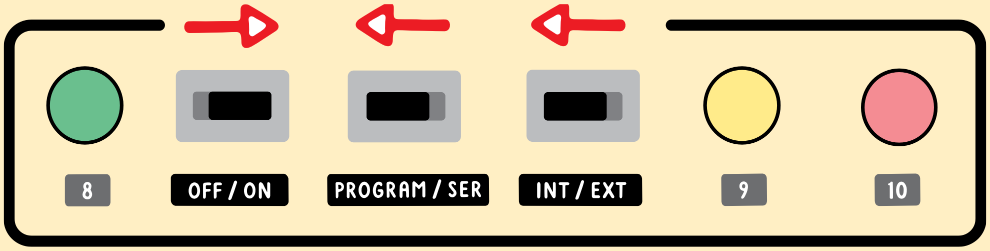
Scroll until you see MegaCoreX. Click it and choose install.

Select these settings:



Board: "ATmega4809"
Clock: "Internal 16 MHz"
BOD: "BOD 2.6V"
EEPROM: "EEPROM retained"
Pinout: "40 pin standard"
Reset pin: "GPIO"
Bootloader: "No bootloader"
Port: "/dev/cu.wchusbserial####"
Programmer: "SerialUPDI (230400 baud)"

Set the switches on the EZA Board as follows:



You are now ready to upload your first program!

LESSON 1: BASICS
THE BASIC SETUP OF AN ARDUINO PROGRAM

A basic Arduino program consists of three main sections:

1. Declarations. An Arduino program starts with a list of items you intend to use later on. Think of this section as a list of supplies and tools you would need before you go on a camping trip. Anything you declare in this section will be accessible to the entire program. The declaration section cannot contain actions, only items.

2. Setup function. All actions must be located inside a function. A function is simply a group of actions with a common purpose. Every Arduino program must contain at least two functions: setup and loop. The setup function is executed upon startup and runs only once. The instructions within the setup function are used to "set up" the chip for operation.

```
    void setup(){}
```

~ Notice that both "void" and "setup" are lowercase.
~ The word "void" indicates that this function is not returning anything.
~ As with any function all instructions go inside the curly brackets.

3. Loop function. The loop function contains your main code. As the name suggest it repeats over and over. This continues as long as the chip has power.

```
    void loop(){}
```

~ Notice that both "void" and "loop" must be lowercase.
~ As with any function all instructions go inside the curly brackets.

COMMENTS

Code is intended to be understood by computers, not by humans. That can make it challenging to understand when you reread your own code. Comments solve this problem. They are used to explain your code in language that makes sense to you, not necessarily to the computer. Adding comments is a very important part of good programming.

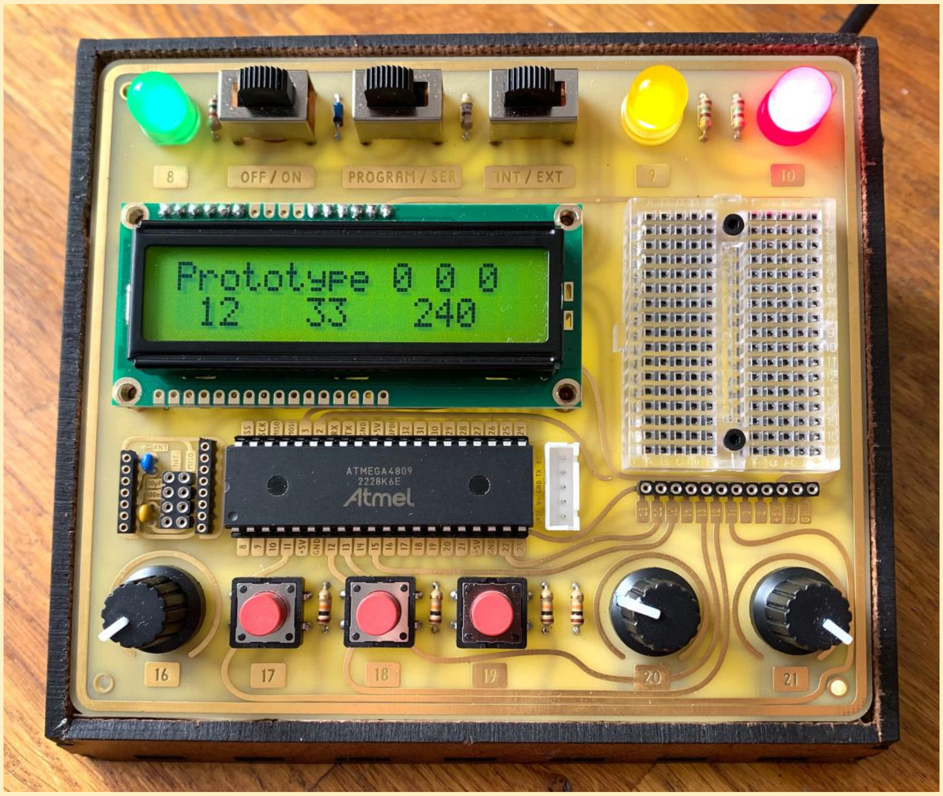
To leave a comment simply add two forward slashes and type the purpose of the code:

```
    byte greenLight = 8;  //Pin 8 connects to the green LED
```

~ Anything after the two slashes will be ignored by the chip; it is only intended for yourself or collaborators.

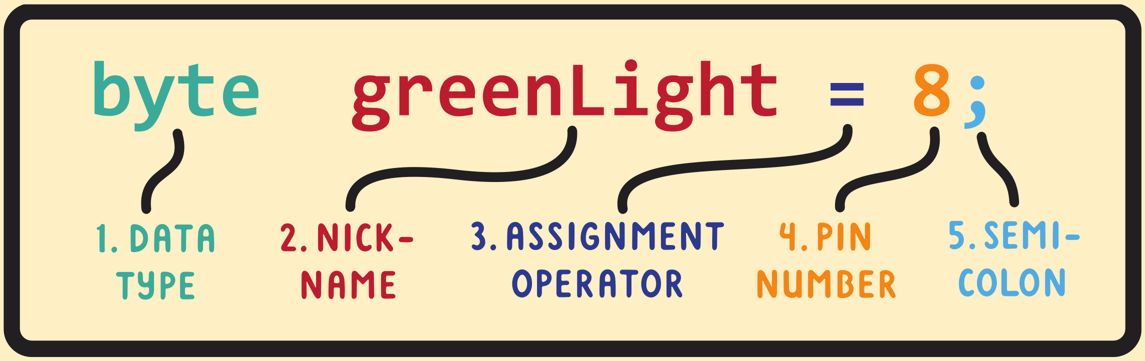
~ Comments are usually placed at the end of a line but they can also be located on a separate line.

~ The black rectangular object below the display is the ATMEGA4809, the chip that runs your code and controls the entire board.



DECLARING PINS

The ATMEGA4809 chip used on your EZA Board has 40 metal legs, referred to as pins. These pins are the way the chip interacts with the outside world. The pins you plan on using are declared at the beginning of the program. A pin declaration consists of five parts:



1. Data type. Before you declare a pin you should type the word “byte”. The reason for that particular word will be made clear later on. Remember that “byte” must be lowercase.

```
    byte greenLight = 8; //Pin 8 is connected to the green LED
    byte button1 = 17; //Button 1 connects to pin 17
```

2. Name. You can name a pin almost anything you like. Pin names cannot start with a number or contain a space.

~ Try to keep your pin names descriptive but short.

~ Arduino and many other computer languages often use “camelCase”. That means a name starts out with a lowercase letter. If the name consists of more than one word all subsequent words will start with an uppercase letter. There are no spaces between the words in a name. It is good practice to follow this convention when naming your pins.

3. Assignment Operator (=). Use a single equals sign to assign the pin number to your pin name.

4. Pin number. The pin numbers are clearly marked on the board, both next to the chip and next to the component connected to each pin.

5. Semicolon. Every declaration or action needs to be terminated with a semicolon;

~ Functions should not be terminated with a semicolon since they are not actions (or declarations), they merely contain actions. So:

```
    void loop(){…}
```

not:

```
    void loop(){…}; //Wrong! No semicolon after a function!
```

SETTING PINS TO BE INPUTS OR OUTPUTS

Every user-accessible pin can interface with the outside world in one of two ways: as an input or as an output. The direction of the signal determines whether you need an input or output. Input pins collect information from the outside world, such as a button press or volume knob position. Output pins manipulate something outside the chip, for example, an LED or motor. In short: *inputs listen and outputs speak*.

Every user-accessible pin can function as either input or output. Setting the input/output mode of a pin is done using the pinMode command. Since this is an action it must go inside a function. The most logical place is the setup function, since setting the pin mode only needs to happen once.

```
    pinMode(greenLight, OUTPUT); //Inside setup function
    pinMode(button1, INPUT); //Inside setup function
```

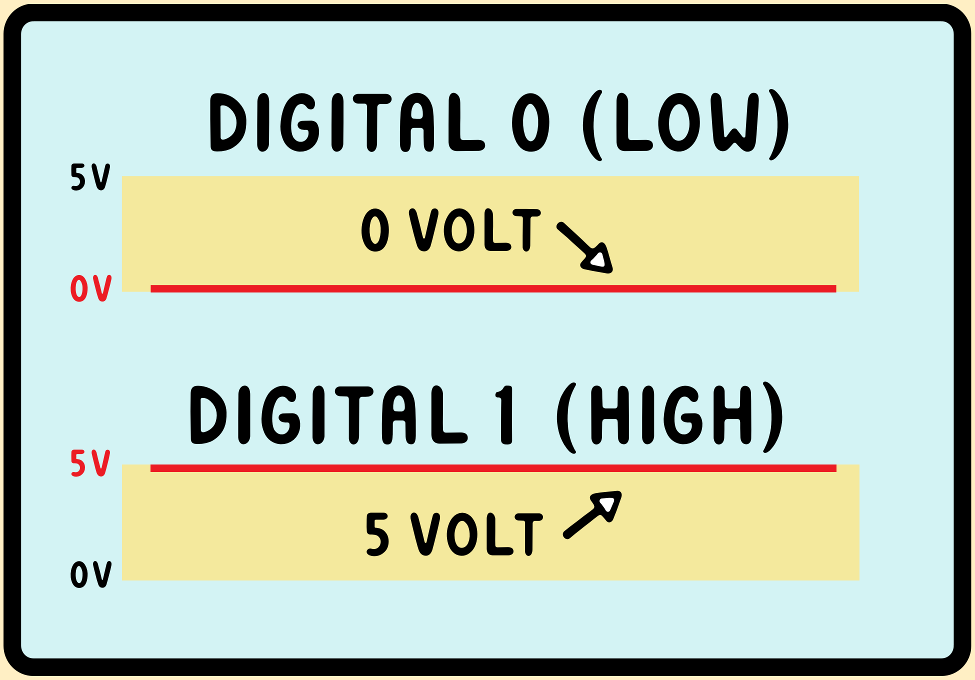
~ Notice the camelCase (pinMode), and capitalization of INPUT and OUTPUT, as well as the semicolon at the end.

~ Remember you must have declared the pin name at the beginning, otherwise the pinMode command does not know what you are talking about.

~ All pins are inputs by default. If an output pin does not act expected you might have forgotten to set it as an output.

WRITING TO AN OUTPUT PIN

The ATMEGA chip on your EZA Board is a digital device. That means it operates exclusively using LOW and HIGH states, represented as 0 and 1. Any digital pin configured as output can have one state at a time: LOW or HIGH. There are no in-between states.



Setting an output pin LOW will connect the pin to 0 volt. Setting an output pin HIGH will connect it to 5 volt, the chip's operating voltage.

To turn any of the EZA Board's lights on you need to send 5 volt to the pin connected to that light. Sending 5 volt to a pin can be accomplished using the digitalWrite command:

```
    digitalWrite(greenLight, HIGH);
```

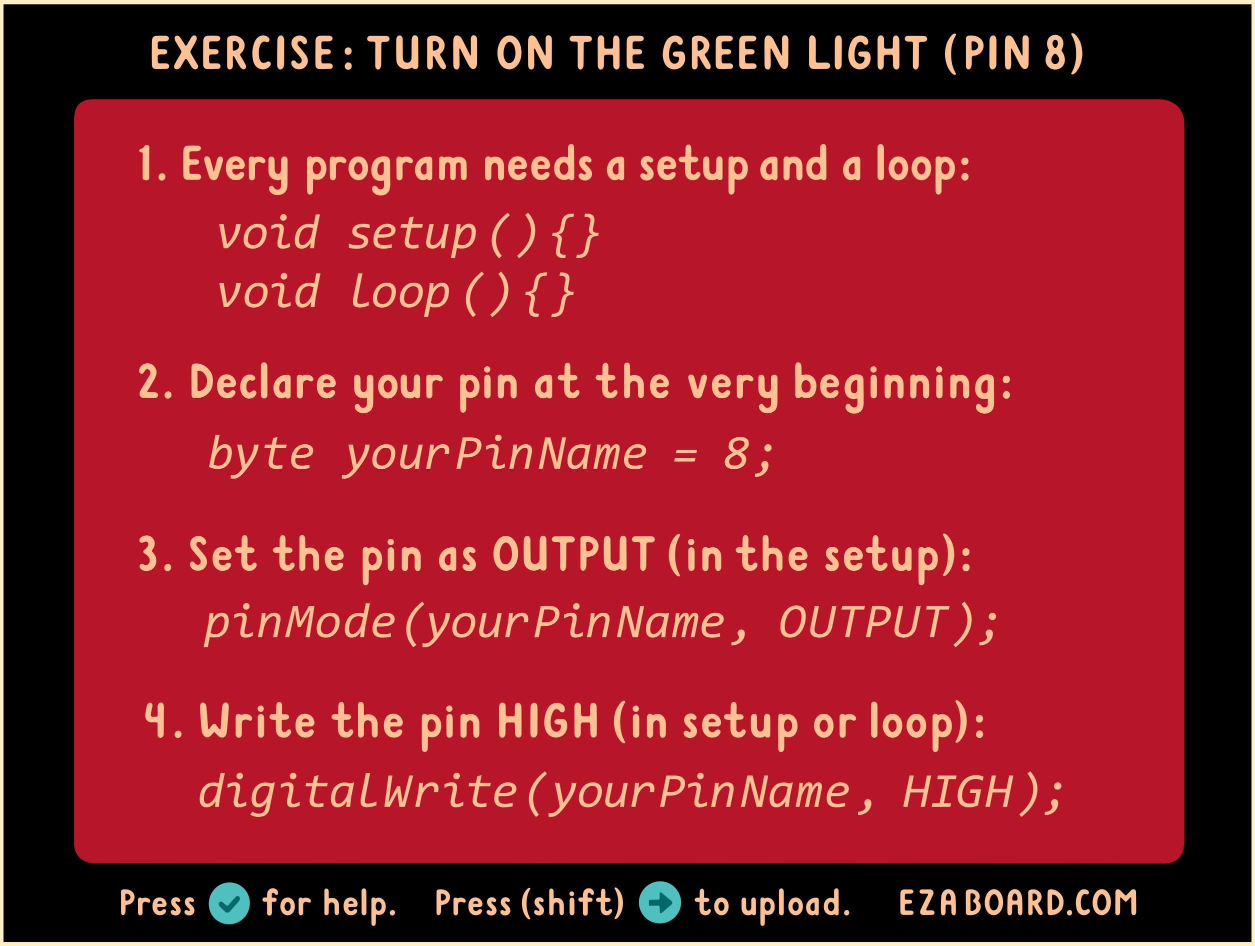
Writing the pin LOW will turn the light off:

```
    digitalWrite(greenLight, LOW);
```

~ Notice the camelCase (digitalWrite), and capitalization of HIGH and LOW.

~ To upload press the arrow on the top left of the Arduino software. Alternatively you might have to use "Upload using programmer" from the "Sketch" menu.

~ If the green light does not turn on you likely forgot to set the pin as an output using the pinMode command. You could also have neglected to declare the pin correctly at the beginning of the program.



EXERCISE: TURN ON THE GREEN LIGHT (PIN 8)

1. Every program needs a setup and a loop:

```
void setup(){}
void loop(){}
```

2. Declare your pin at the very beginning:

```
byte yourPinName = 8;
```

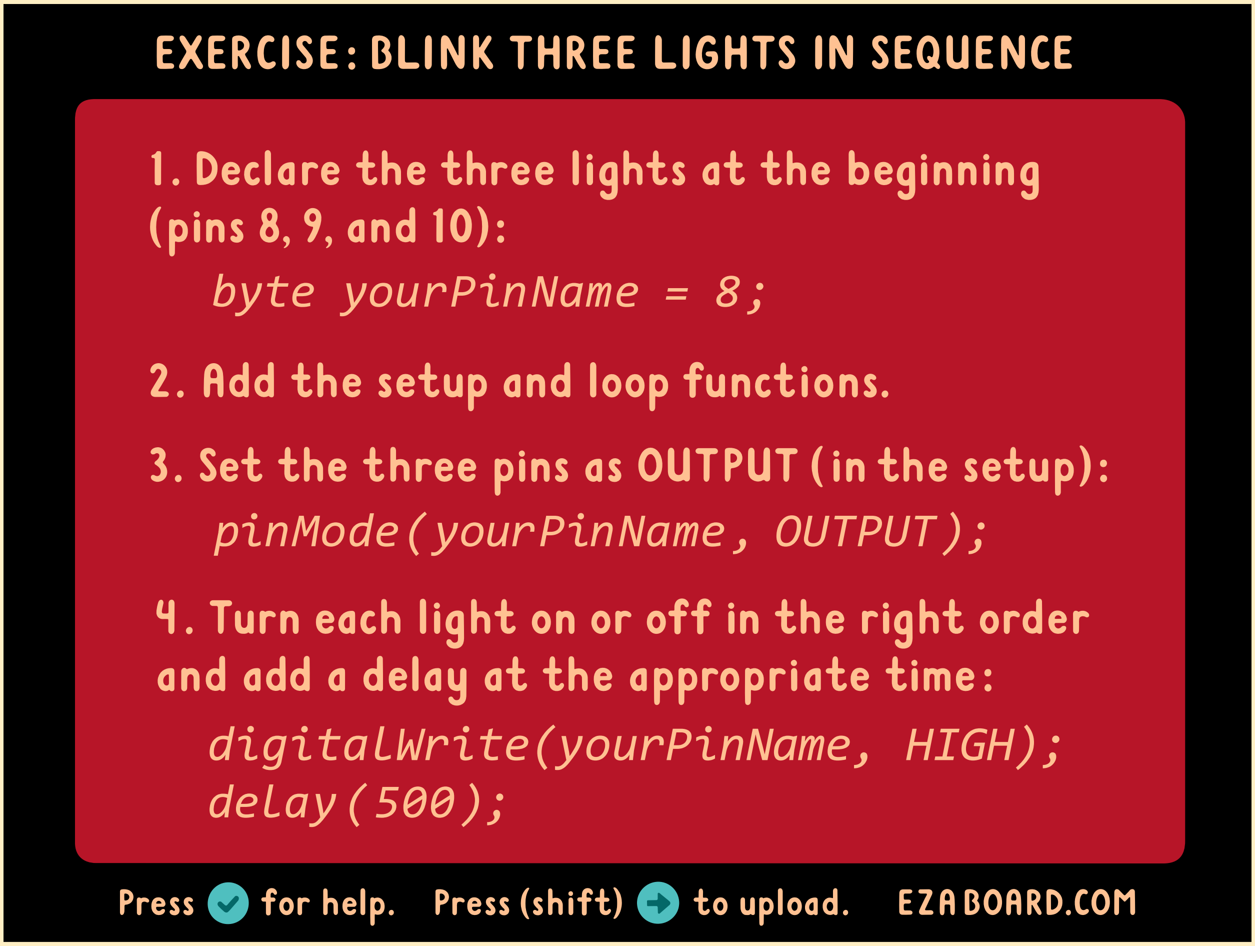
3. Set the pin as OUTPUT (in the setup):

```
pinMode(yourPinName, OUTPUT);
```

4. Write the pin HIGH (in setup or loop):

```
digitalWrite(yourPinName, HIGH);
```

Press ✓ for help. Press (shift) ➜ to upload. EZABOARD.COM



EXERCISE: BLINK THREE LIGHTS IN SEQUENCE

1. Declare the three lights at the beginning (pins 8, 9, and 10):

```
byte yourPinName = 8;
```

2. Add the setup and loop functions.

3. Set the three pins as OUTPUT (in the setup):

```
pinMode(yourPinName, OUTPUT);
```

4. Turn each light on or off in the right order and add a delay at the appropriate time:

```
digitalWrite(yourPinName, HIGH);
delay(500);
```

Press ✓ for help. Press (shift) ➜ to upload. EZA BOARD.COM

LESSON 2: SERIAL MONITOR

The serial bus allows the ATMEGA chip to send text or numbers to your computer, where you can read it on the serial monitor in the Arduino software. This will become extremely helpful when your programs become more complex and you need to know what is going on inside your chip.

~ To access the serial monitor in the Arduino software click on the magnifying glass located at the top right of the window.

Configuring the serial monitor is an action and should be placed in the setup function.

```
     Serial.begin(9600); //9600 is the standard data rate
```

~ Notice that serial commands do not use camelCase but start with “Serial” capitalized followed by a period.

To print exact text:

```
     Serial.print(“Hellooo… Anybody listening?”); //double quotes
```

To print an enter:

```
     Serial.println(); //println = print new line
```

To print the time since startup:

```
     Serial.print( millis() ); //in milliseconds (1/1000th sec)
```

~ millis() needs its own set of parentheses!

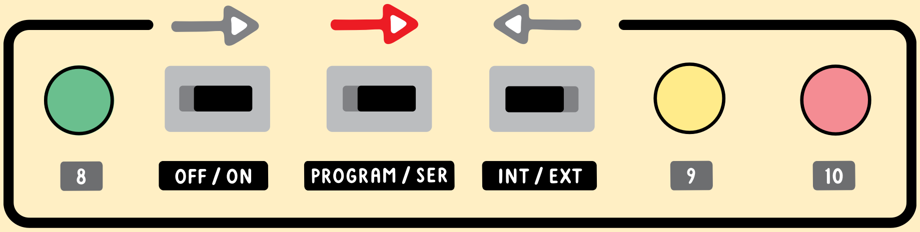
~ Notice there are no quote marks. Putting quote marks around “millis()” would literally print the text “millis()”, instead of the time since startup.

~ A delay is often included after a serial print command to prevent the ATMEGA chip from printing information faster than you can read.

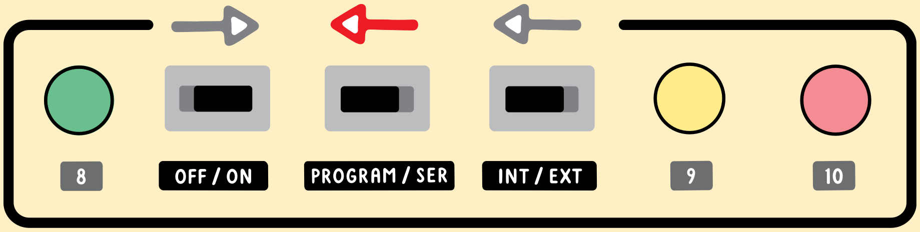
```
    delay(500); //1/1000th sec * 500. delay is lowercase.
```

After you have uploaded your code, switch the center selector from "PROGRAM" to "SER" to read the serial monitor. Don't forget to switch back for programming.

USING SERIAL MONITOR:



PROGRAMMING:



~ Make sure the serial monitor in the Arduino software is set to 9600, otherwise you might get gibberish or nothing at all. This setting is accessible at the bottom of the serial monitor.

~ You might find that the very first thing you print does not show up on the serial monitor after a board restart. This is due to the microcontroller starting up faster than the USB chip. Include a small delay (200 milliseconds) before your first serial print command to avoid this problem.

~ If nothing works you could try closing and reopening the serial monitor by clicking the magnifying glass in the top left corner or restarting the Arduino software.

LESSON 3: INPUTS

So far we have only used pins as outputs. Let's explore inputs next. If you want to read the status of a button you have to declare the pin connected to that button as an input (see ch. 1). What command would be used to read an input? Remember that writing to an output pin is done using the digitalWrite command. Therefore reading the state of an input pin is accomplished using the digitalRead command:

```
     digitalRead(button1); //This code is useless by itself
```

Even though this command is valid, it is useless since the information is read but not processed. It is hanging in mid-air, so to speak. One way of processing the information is to send it to the serial monitor:

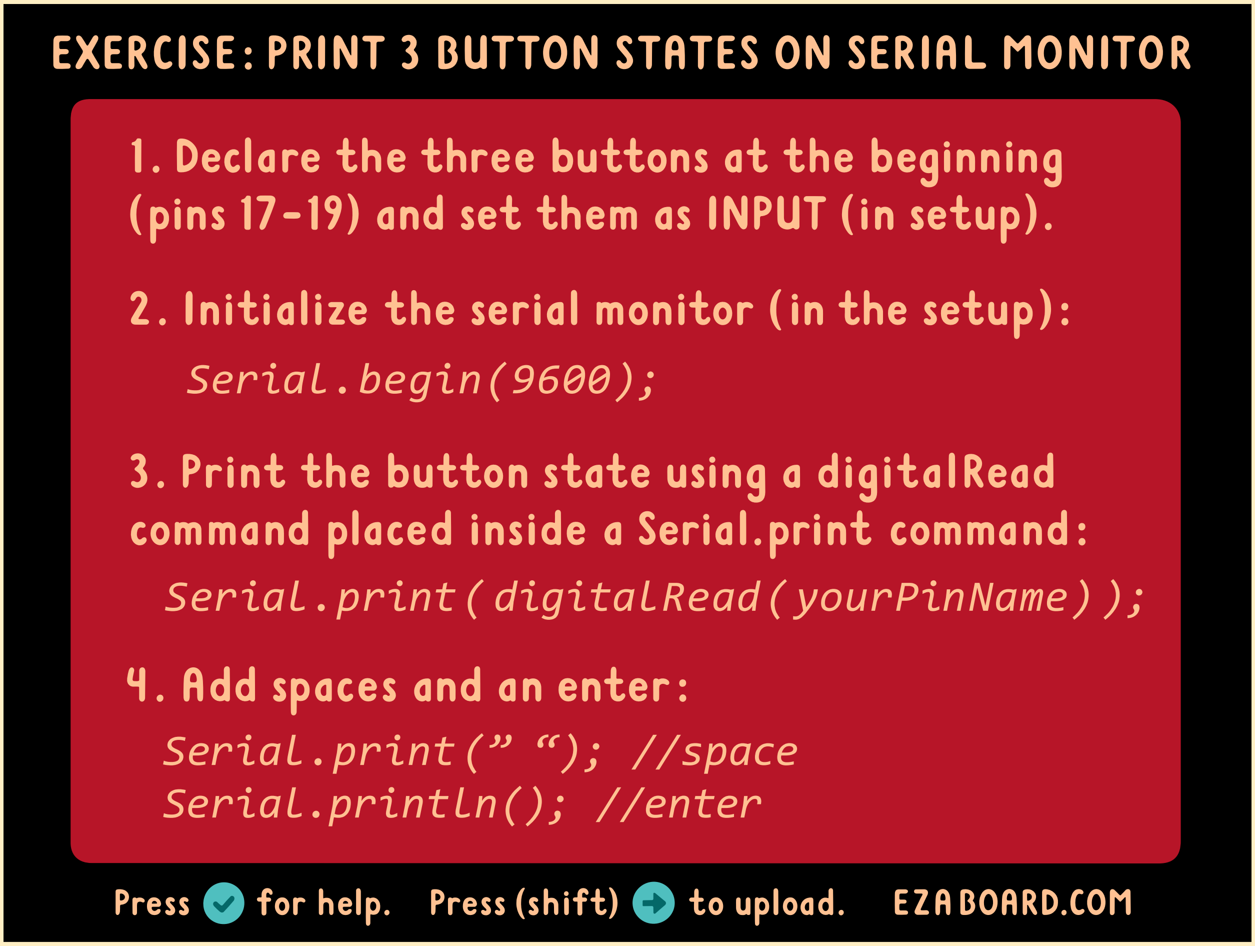
```
     Serial.print( digitalRead(button1) ); //print the state of
button1 to the serial monitor
```

~ Never put a semicolon *inside* a Serial.print statement.

The digitalRead command can only return a value of 0 or 1. If the voltage at the pin is 0 volt the digitalRead command reports 0. When the voltage at the pin is 5 volt (the chip's operating voltage) the digitalRead command will report 1 (not 5!).

~ You might wonder what would happen if you try to do a digitalRead on a pin that is neither at 0 nor 5 volt. Voltages in between 0 and 5 will be rounded up or down to a 0 or 1 but might not always be rounded the way you expect.

To summarize the differences between inputs and outputs:
INPUT: button, microphone, knob - Microcontroller listens – use **digitalRead**
OUTPUT: lights, buzzer, LCD - Microcontroller speaks – use **digitalWrite**



EXERCISE: PRINT 3 BUTTON STATES ON SERIAL MONITOR

1. Declare the three buttons at the beginning (pins 17-19) and set them as INPUT (in setup).

2. Initialize the serial monitor (in the setup):

```
Serial.begin(9600);
```

3. Print the button state using a digitalRead command placed inside a Serial.print command:

```
Serial.print(digitalRead(yourPinName));
```

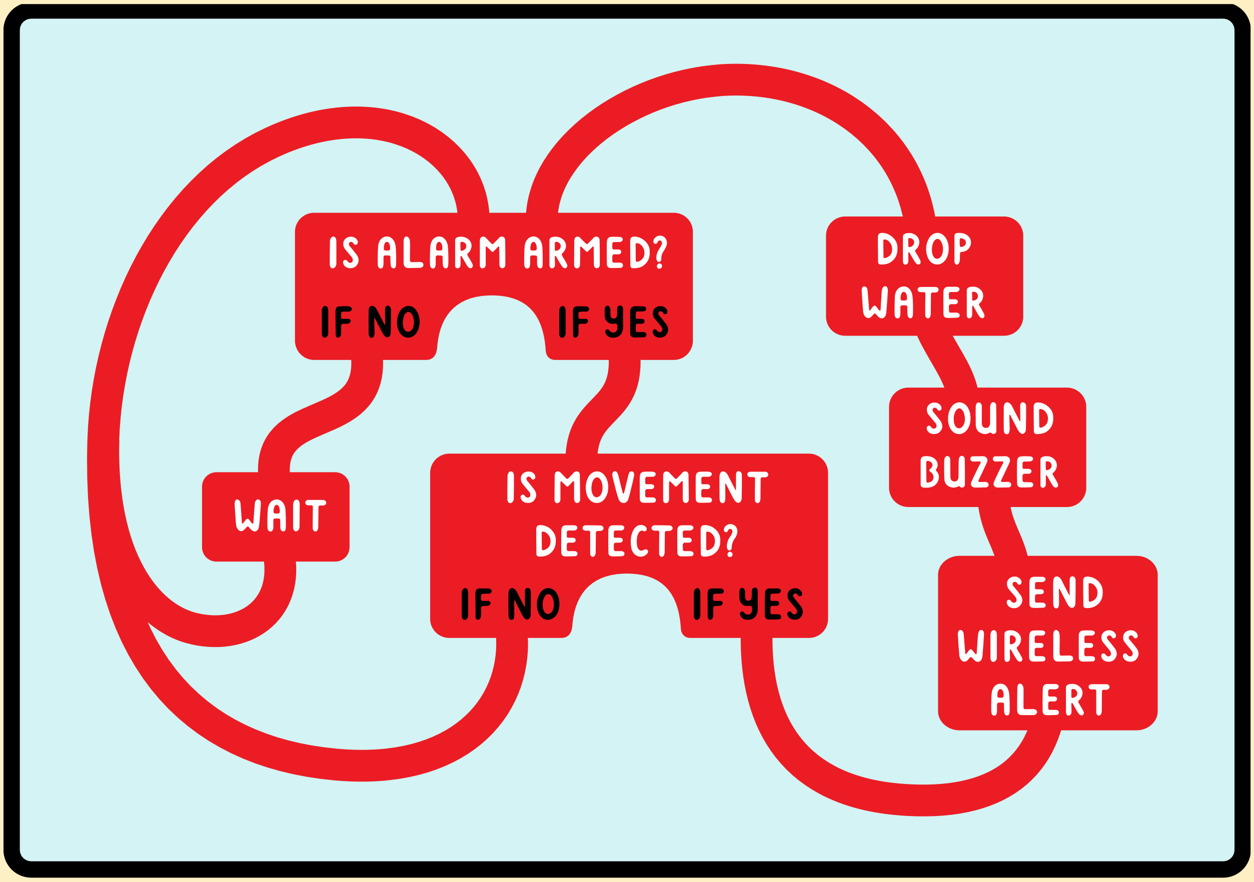
4. Add spaces and an enter:

```
Serial.print(” “); //space
Serial.println(); //enter
```

Press ✓ for help. Press (shift) → to upload. EZA BOARD.COM

LESSON 4: IF-STATEMENTS

If-statements are the key element that makes programming powerful. You can think of your code as a decision tree; when the code comes to an intersection it can go multiple ways and an if-statement determines which branch the code goes down. Try to imagine what this intruder alarm code would do without the if-statements:



If-statements are conditional statements; if the condition is fulfilled the code will execute. The if-statement acts like this:

```
    if (this is true){execute this code} //not real code!
```

~ Notice the parentheses and curly brackets of an if-statement are identical to those in functions, for example: “void loop(){}”. The code goes in the curly brackets.

~ The word “if” needs to be lowercase.

~ An if-statement, like a function, does not need a semicolon at the end.

Let’s turn on an LED when a button is pressed. Remember from last lesson that:

```
     digitalRead(button1); //useless statement by itself…
```

returns a 1 when pressed and a 0 when not pressed. The way you can incorporate that into an if-statement is:

```
     if ( digitalRead(button1) == 1 ){…} //this function will
execute when the button is pressed
```

~ The digitalRead(button1) command does not need a semicolon when it is located inside an if-statement.

Notice the double equals sign. This is a comparison operator. You will need to use it inside an if-statement when *comparing* whether two values are equal. This might seem unnecessarily complicated but comparison operators are used in most computer languages for valid reasons.

Using a single equals sign in an if-statement will result in unexpected behavior!

A single equals sign (assignment operator)

```
numberOfPushups = 17;
```

means: Assign the value of 17 to “numberOfPushups”

A double equals sign (comparison operator)

```
if (numberOfPushups == 17){…
```

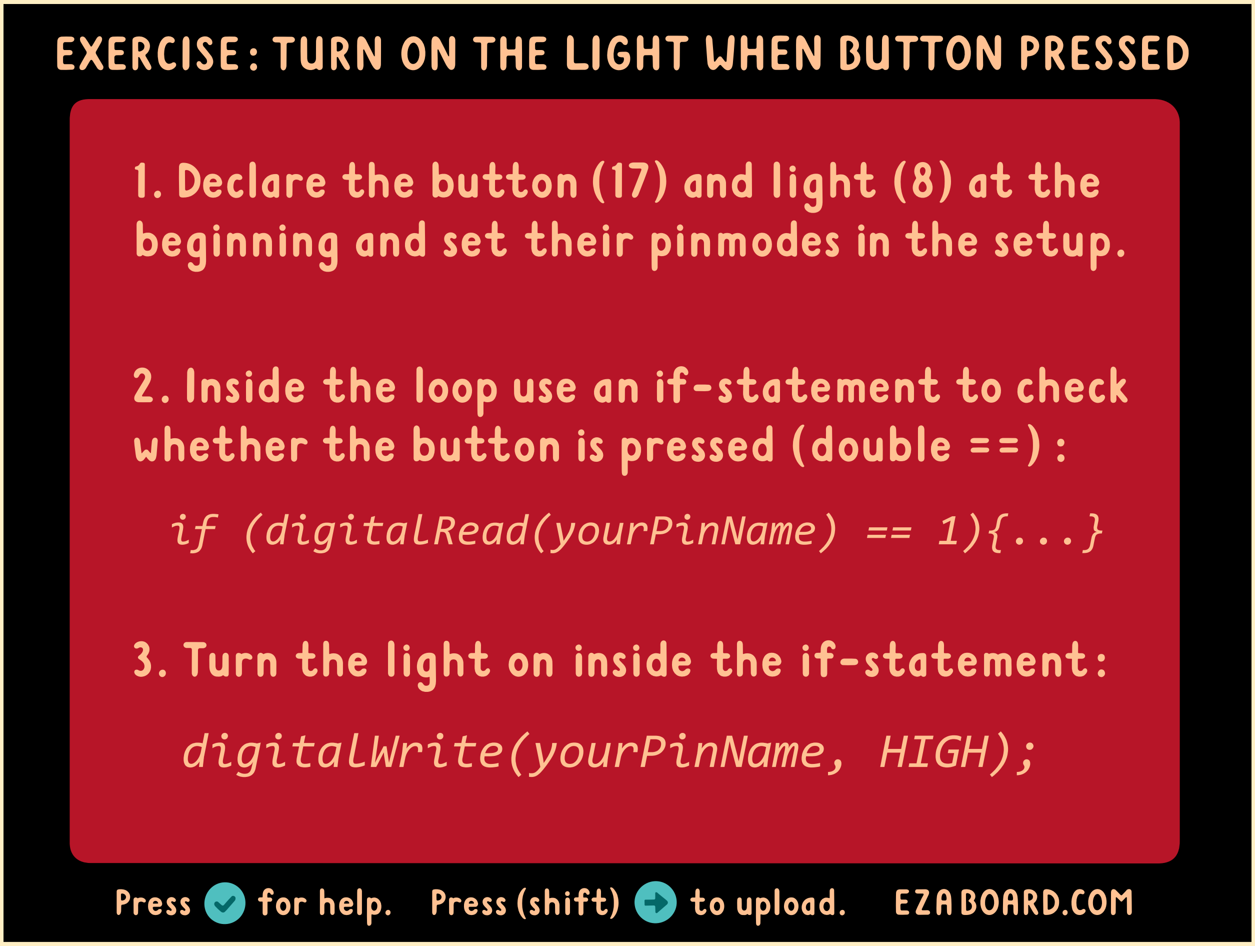
means: test if “numberOfPushups” is equal to 17

If you type:

```
    if ( digitalRead(button1) = 1){… //WRONG!!!
```

the chip will confuse that statement with an assignment operator (set A equal to B), and try to set “digitalRead(button1)” equal to 1, even though that does not make sense to us.

If your if-statement does not work check the use of equals signs first!



EXERCISE: TURN ON THE LIGHT WHEN BUTTON PRESSED

1. Declare the button (17) and light (8) at the beginning and set their pinmodes in the setup.

2. Inside the loop use an if-statement to check whether the button is pressed (double ==):

```
if (digitalRead(yourPinName) == 1){...}
```

3. Turn the light on inside the if-statement:

```
digitalWrite(yourPinName, HIGH);
```

Press ✓ for help. Press (shift) ➜ to upload. EZA BOARD.COM

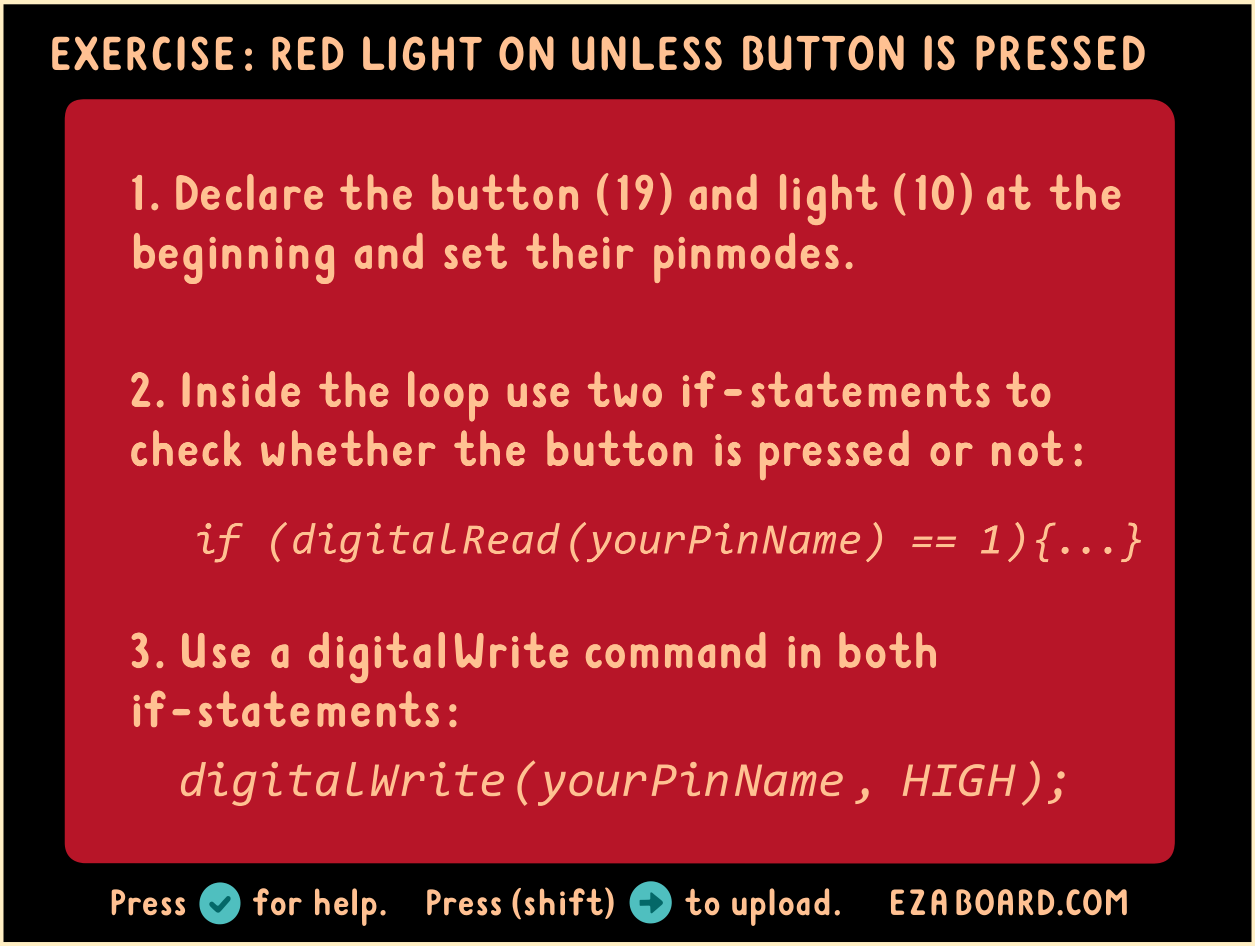
If/else-statements:
When programming an LED to turn on whenever you press a button you might use code like this:

```
if (digitalRead(button1) == 1){…} //Turn light ON
if (digitalRead(button1) == 0){…} //Turn light OFF
```

A faster and more concise way to decide whether the button is pressed or released is the if/else statement:

```
if (digitalRead(button1) == 1){…} //if button is pressed…
else {…} //Whenever the button is NOT pressed…
```

~ Notice the else statement does not need a set of parentheses, only curly brackets.



EXERCISE: RED LIGHT ON UNLESS BUTTON IS PRESSED

1. Declare the button (19) and light (10) at the beginning and set their pinmodes.

2. Inside the loop use two if-statements to check whether the button is pressed or not:

```
    if (digitalRead(yourPinName) == 1){...}
```

3. Use a digitalWrite command in both if-statements:

```
  digitalWrite(yourPinName, HIGH);
```

Press ✓ for help. Press (shift) ➜ to upload. EZABOARD.COM

LESSON 5: LCD (EXERCISE TRACKER)

Using the LCD (Liquid Crystal Display) on your EZA Board is very complicated. Thankfully some kind people have made it very easy by writing a library for the LCD. A library is simply a large amount of code that you can access with a few simple commands. There are libraries for numerous accessories, making Arduino programming accessible to almost everyone.

Remember that the declaration section of your program is like a list of required supplies and tools. A library is one of these tools. Therefore it must be declared at the beginning of your program:

```
#include <LiquidCrystal.h> //Include LCD library
LiquidCrystal lcd(29, 28, 27, 26, 25, 24); //Pin connections
```

This code tells the chip to include the LCD library and informs the library how the LCD pins are connected to the chip's pins.

~ Libraries are very different from pins. Therefore their declarations are very different too. It is sufficient just to copy and paste the declaration lines without understanding the details.

You only need one more command before you can use the LCD! This command goes in the setup. It initializes the LCD and tells the ATMEGA chip what size your LCD is:

```
lcd.begin(16,2); //2 rows of 16 characters each
```

You have now given the EZA Board the ability to "talk":

```
lcd.print(“Ma..ma..”);
```

~ For now place this comment in the setup function, not in the loop.

~ If your LCD does not work the contrast is likely out of adjustment. Turn the adjustment on the back of the board until the contrast is optimal.

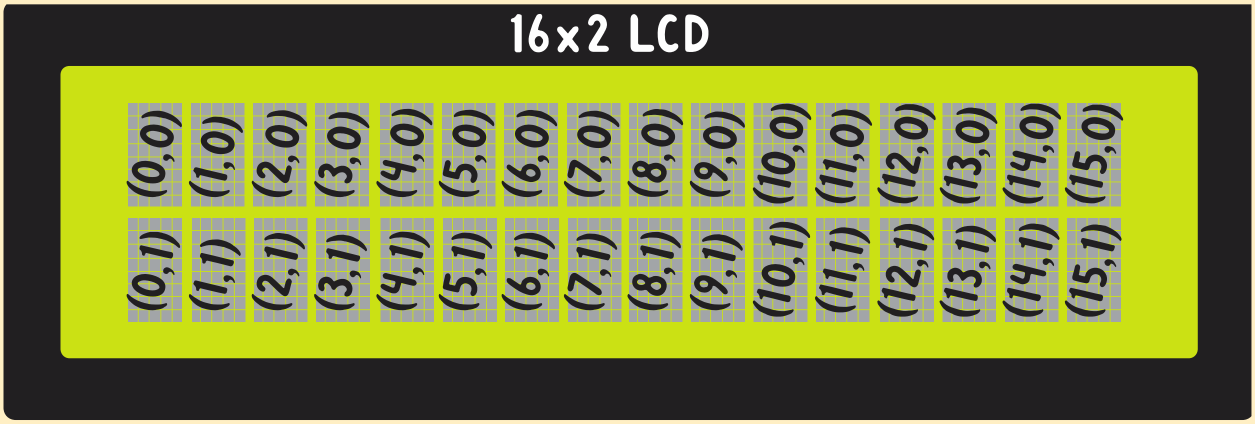
The LCD has a backlight built in. Turning it on will help visibility. The backlight is connected to pin 30. Be sure to declare it as an output and turn it on:

```
     byte backLight = 30; //In declaration section
     pinMode(backLight, OUTPUT); //In setup function
     digitalWrite(backLight, HIGH); //In setup or loop function
```

If you attempt to print to the LCD within the loop you will find that the LCD continues printing wherever it last left off. This is rarely what you want. There are two ways to avoid this. The simply way is to use the lcd.clear(); command. This will erase the entire display and set the cursor at the very beginning. You can then start over.
The advanced way is to manually position the cursor at a specific point. This is done using the command:

```
     lcd.setCursor(0,0); //numbers refer to coordinates
```

Here are the coordinates used to place the cursor at any point:

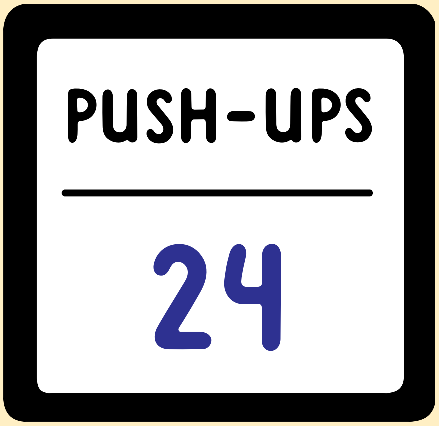


(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	(7,0)	(8,0)	(9,0)	(10,0)	(11,0)	(12,0)	(13,0)	(14,0)	(15,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)	(8,1)	(9,1)	(10,1)	(11,1)	(12,1)	(13,1)	(14,1)	(15,1)

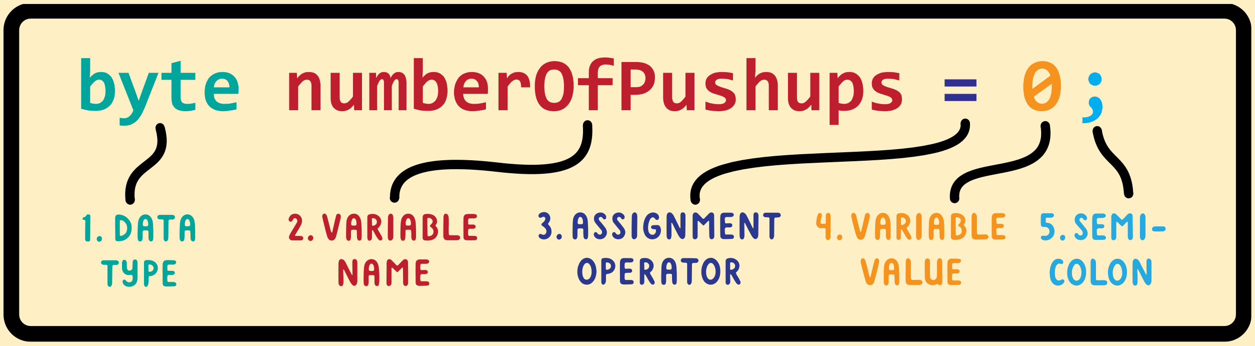
~ There is no erase command for the LCD. Instead you can “erase” text by overwriting it with spaces.

LESSON 6: VARIABLES (EXERCISE TRACKER)

Often you will find the need to keep track of a changing value, in this case the number of pushups. What you need is a variable. A variable is like a whiteboard that contains two pieces of information: a number and the meaning of that number. The value of the number is expected to change during operation; therefore it is called a variable.



Variables are usually introduced in the declaration section. Their declaration is identical to the way pins are declared:



```
byte numberOfPushups = 0;
```

Variables do not necessarily need to be initialized with the value 0. Sometimes it makes sense to initialize the variable at another value:

```
    byte numberOfDartsInMagazine = 12;
```

The data type ("byte" in this case) tells the chip how much memory to allocate to this variable. A byte has a range from 0 to 255 (that is why you declare the pins as bytes; there are fewer than 255 pins). Other variables have larger ranges but take up more memory.

Once you have declared the variable at the beginning of the program you will want to read or manipulate its value later on. When doing so you should NOT *repeat the data type, otherwise the ATMEGA chip will think you are declaring it again! (The chip already knows what the data type is, you shouldn't reiterate it!)*

```
     numberOfPushups = 0; //When referring to a variable already declared
```

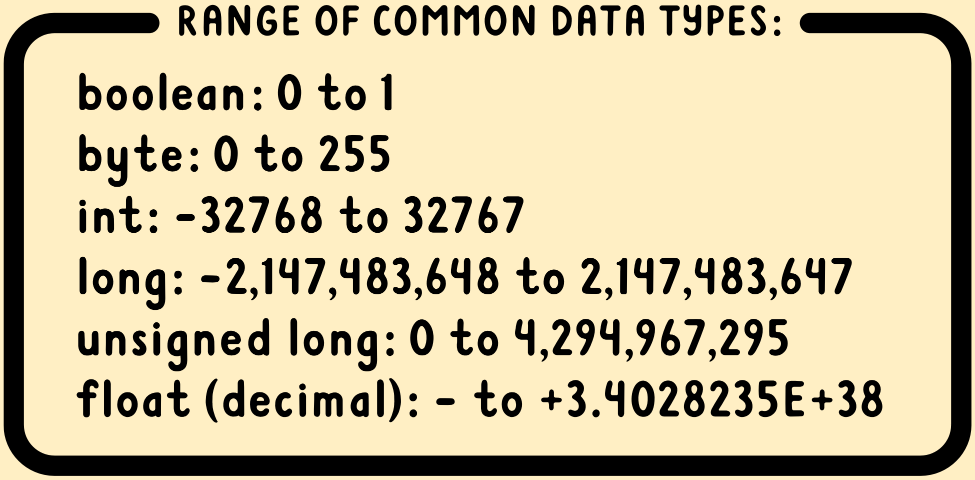
NOT:

```
     byte numberOfPushups = 0; //Only when declaring a variable initially!
```

In case you plan on doing more than 255 pushups you will need a larger data type:

```
     int numberOfPushups = 0; //range up to 32768
```

Here are the ranges of several common data types:



RANGE OF COMMON DATA TYPES:

boolean: 0 to 1
byte: 0 to 255
int: -32768 to 32767
long: -2,147,483,648 to 2,147,483,647
unsigned long: 0 to 4,294,967,295
float (decimal): - to +3.4028235E+38

~ The ranges of data types might seem arbitrary. Why does a byte have a maximum of 255? Why not 250 for example? The reason makes perfect sense to a computer. A computer does not count with numbers from 0-9, but only using 0 or 1. A single "number" is referred to as a "bit". Since one bit can express only two values (0 or 1), a series of 8 bits (called a byte) will be able to express 2*2*2*2*2*2*2*2 = 256 numbers. Since the number 0 is quite useful, computers don't use these 256 values to go from 1 – 256 but instead from 0 – 255.

~ It is good practice to use the smallest data type that will fit the largest value you expect a variable to contain. Using large data types can slow your program down, though the difference might not be noticeable. The float data type is the most likely to slow your program down, especially if you are doing math with it.

You might not want to set a variable to an exact number but simply perform some math on its current value, such as increasing it by one or multiplying by 10:

```
numberOfPushups = numberOfPushups + 1; //Add one
numberOfPushups = numberOfPushups * 10; //multiply by 10
```

LESSON 7: BUZZER AND FOR-STATEMENT (BURGLAR ALARM)

The buzzer is located on the back of your board and is connected to pin 31.

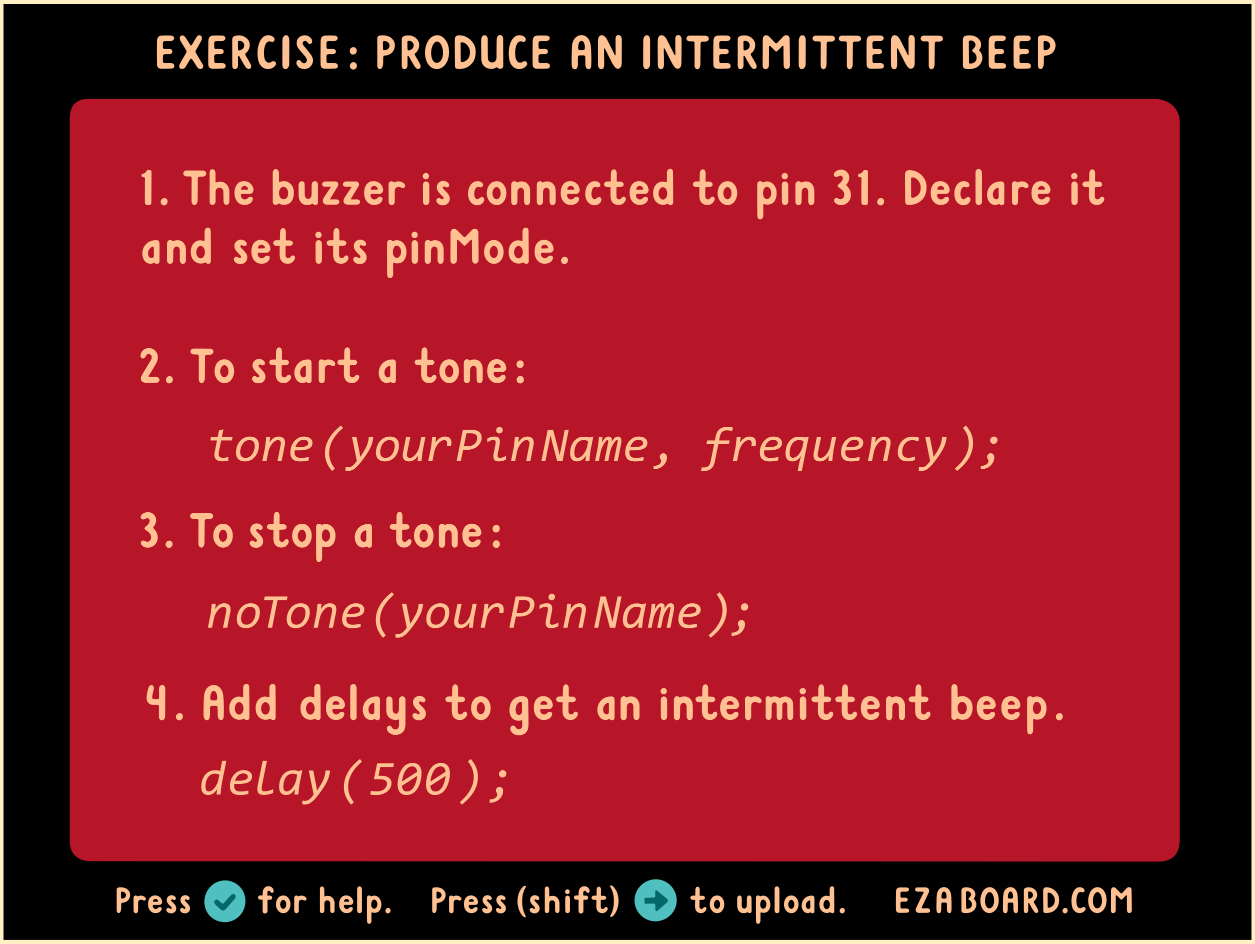
The code is for sounding the buzzer is:

```
     tone(pinName, 432);  //432 refers to frequency in Herz
```

To stop the tone use code:

```
     noTone(pinName);
```

If you type these two commands in the loop and add a delay after each command you will get a beep that repeats as long as the chip has power.



EXERCISE: PRODUCE AN INTERMITTENT BEEP

1. The buzzer is connected to pin 31. Declare it and set its pinMode.

2. To start a tone:

```
tone(yourPinName, frequency);
```

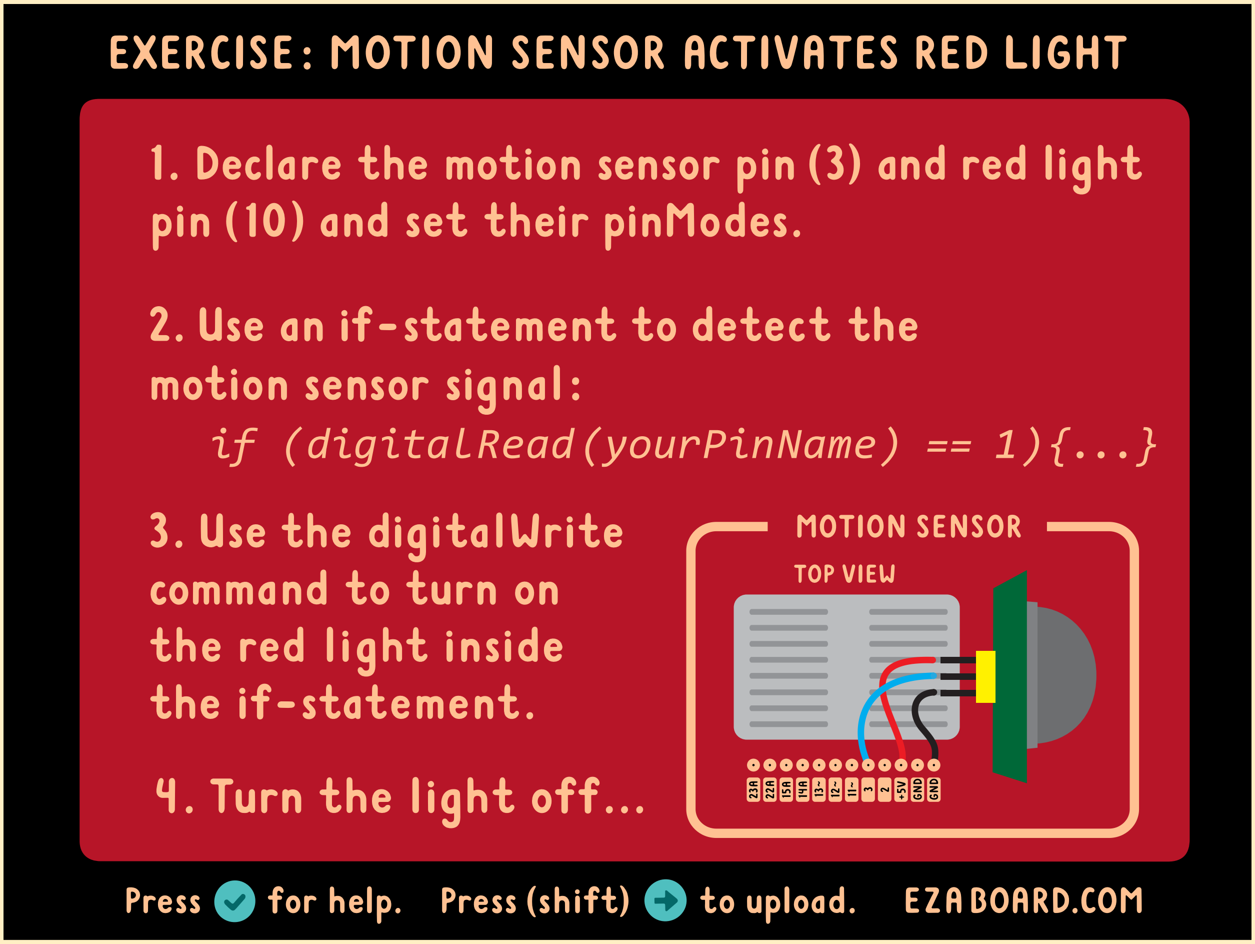
3. To stop a tone:

```
noTone(yourPinName);
```

4. Add delays to get an intermittent beep.

```
delay(500);
```

Press ✓ for help. Press (shift) ➜ to upload. EZABOARD.COM



EXERCISE: MOTION SENSOR ACTIVATES RED LIGHT

1. Declare the motion sensor pin (3) and red light pin (10) and set their pinModes.

2. Use an if-statement to detect the motion sensor signal:

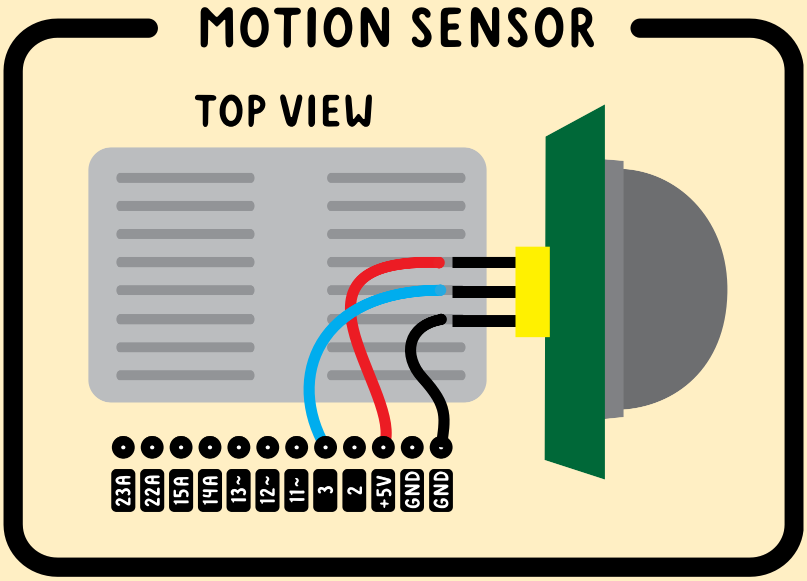
```
    if (digitalRead(yourPinName) == 1){...}
```

3. Use the digitalWrite command to turn on the red light inside the if-statement.

4. Turn the light off...

Press ✓ for help. Press (shift) ➜ to upload. EZA BOARD.COM

~ The color of the wires does not affect functionality but it is standard practice to use red for the positive voltage and black for 0 volt/ground.

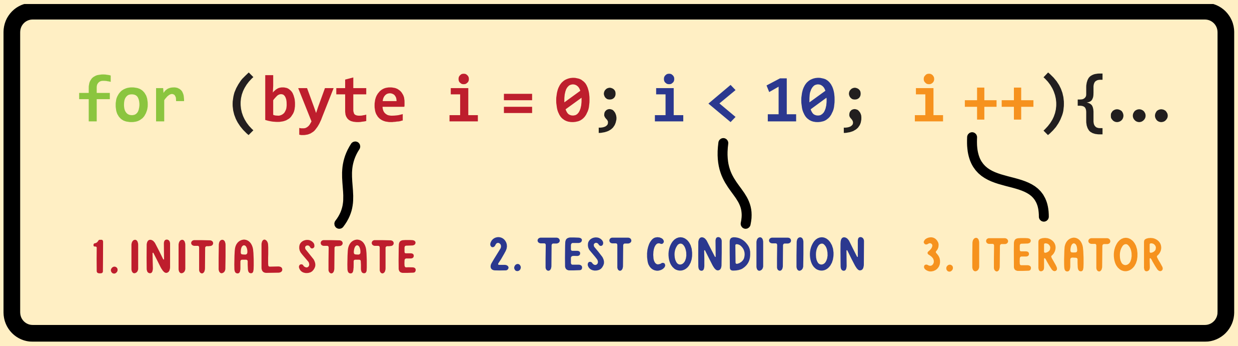


Sometimes you want to repeat some code a specific number of times. You can use a for-statement to accomplish this. A for-statement is simply a repeating block of code:

```
    for (x number of times){repeat this code} //not real code!
```

~ Like a function, a for-statement has a set of parentheses and curly brackets. The word "for" is lowercase. Also notice that, like functions, there is no semicolon at the end.

For-statements can be very powerful. Their setup might seem overly complicated but there is a reason for every part. A for-statement is declared using three conditions separated by semicolons:



1. Initial state. A variable is declared and set to an initial state, usually zero. The variable name traditionally used in for-loops is simply the letter "i".

```
    for (byte i = 0; i < 10; i++){…}
```

~ This for-statement will repeat the code within its curly brackets 10 times.

2. Test condition. As long as the test condition remains true the for-statement will execute. Every time "i" is less than 10 the code inside the for-statement will run.

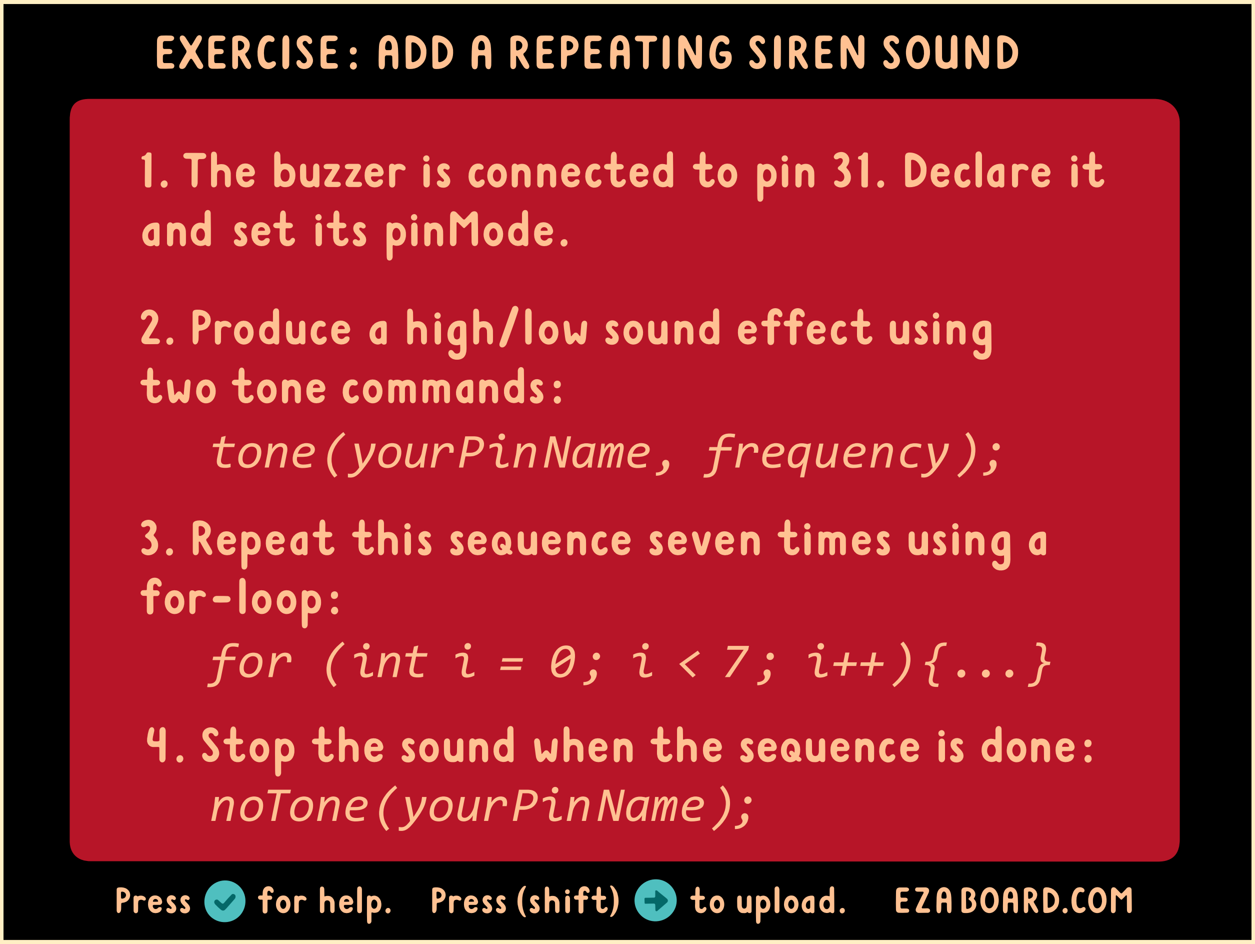
```
    for (byte i = 0; i < 10; i++){…}
```

3. Iterator. An iterator is a number manipulation that happens each time the loop comes around. In this case we want to count up from 0 – 10. The code "i++" simply counts up by one ("i++" is shorthand for saying "i = i + 1").

```
    for (byte i = 0; i < 10; i++){…}
```

~ Remember that the three conditions in a for-statement are placed within the parentheses and separated by semicolons. There is however no semicolon after the last condition.

Now that you know how to repeat code using for-statements you can make a repeating buzzer sound. You can now finish the burglar alarm.



EXERCISE: ADD A REPEATING SIREN SOUND

1. The buzzer is connected to pin 31. Declare it and set its pinMode.

2. Produce a high/low sound effect using two tone commands:

```
    tone(yourPinName, frequency);
```

3. Repeat this sequence seven times using a for-loop:

```
    for (int i = 0; i < 7; i++){...}
```

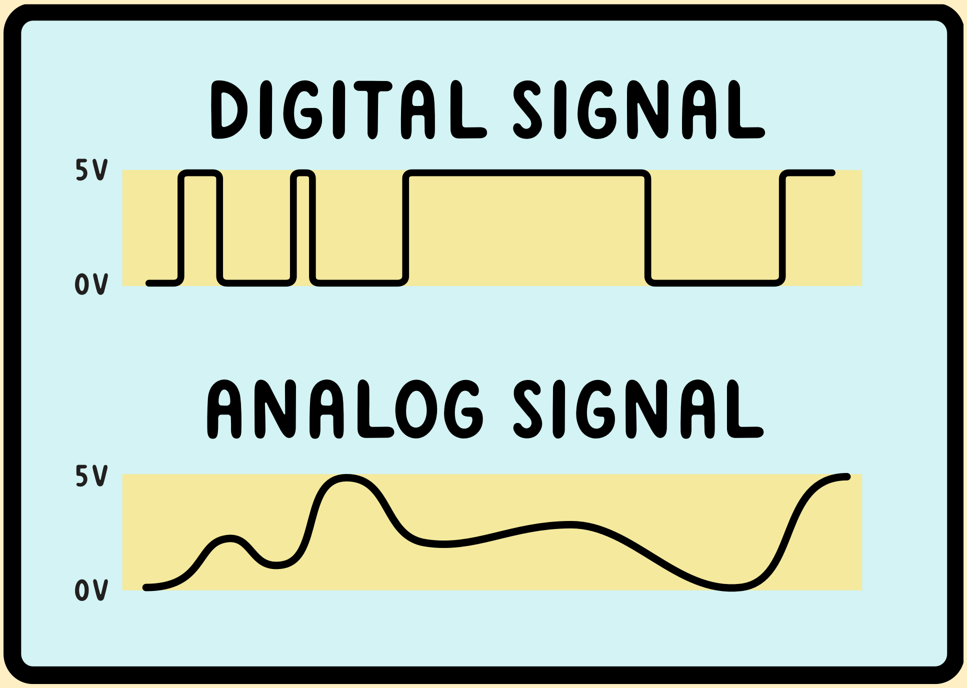
4. Stop the sound when the sequence is done:

```
    noTone(yourPinName);
```

Press ✓ for help. Press (shift) → to upload. EZABOARD.COM

LESSON 8: ANALOGWRITE (DIMMING LEDS)

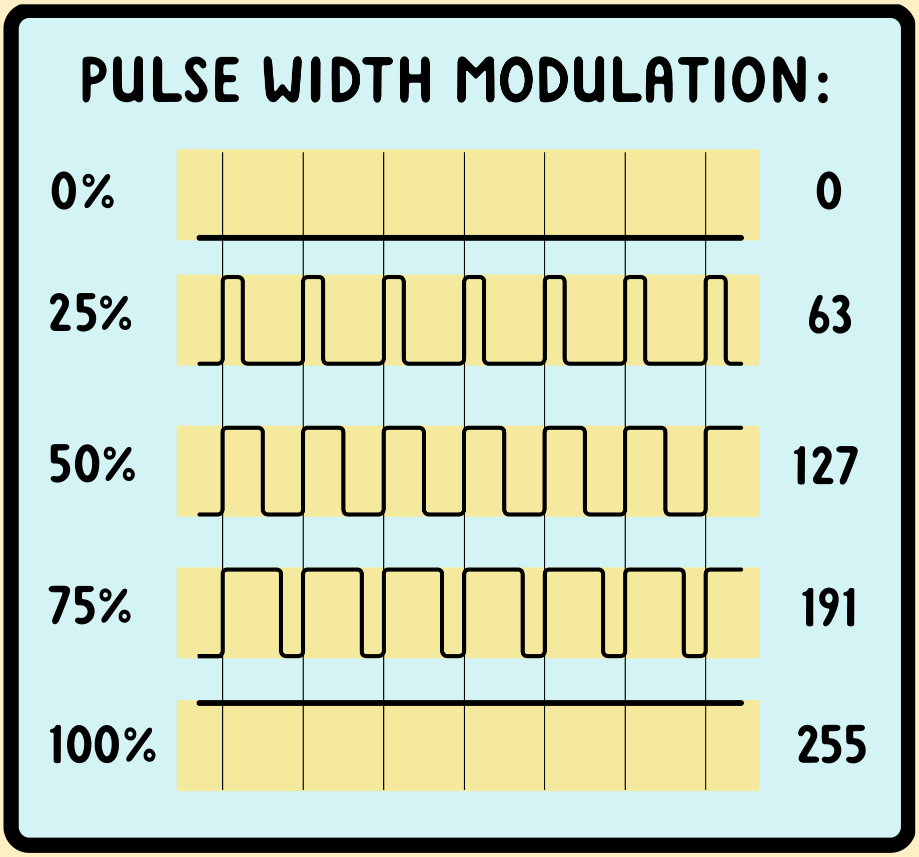
In digital logic outputs can be either 0 or 1; off or on, 0% or 100%. Digital signals act like light switches; there are only two states. But what if you want to do something in between, for example dim a light to half brightness? That would require an analog signal. Analog means there are an infinite number of positions between 0 and 1. Analog signals act like volume knobs; having unlimited number of positions between 0 and 100%.



Since the ATMEGA chip is a digital device it cannot output a real analog signal; it can only be ON or OFF, nothing in between. But there is a way to get an analog effect from a digital device. If you toggled a light switch on and off quickly the light would be on half the time on average, even though it would not really look like a light dimmed to 50%. But if you could toggle the light faster than you could see you would get an effect that looks like a smooth 50% dimmed light. Digital chips can do this dimming far faster than you can see.

Dimming percentages other than 50% are possible as well. Leaving the light on 30% and off 70% of the time would result in a light dimmed to 30%

brightness.



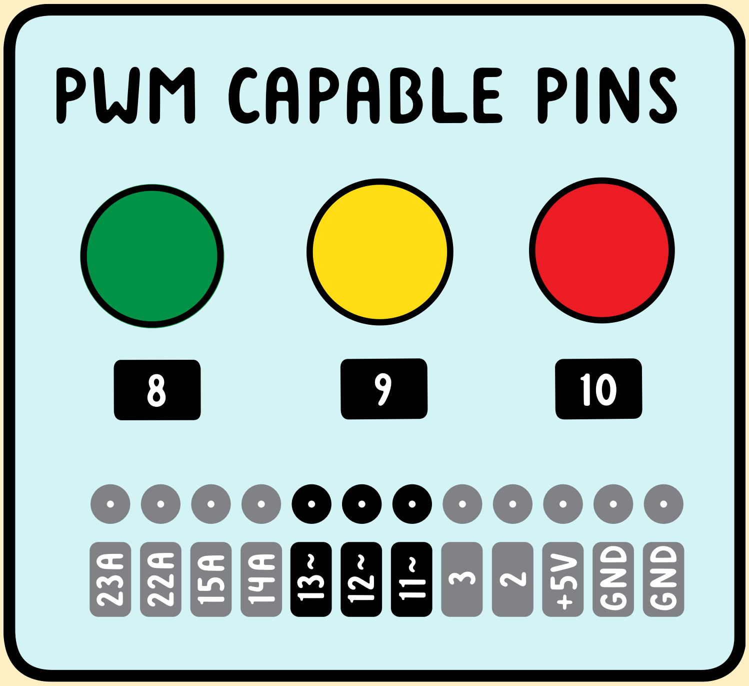
This analog output technique is also called Pulse Width Modulation, or PWM, because the width of the pulses is modulated to achieve the desired brightness.

To generate an “analog” output signal use the command:

```
    analogWrite(greenLight, 127);
```

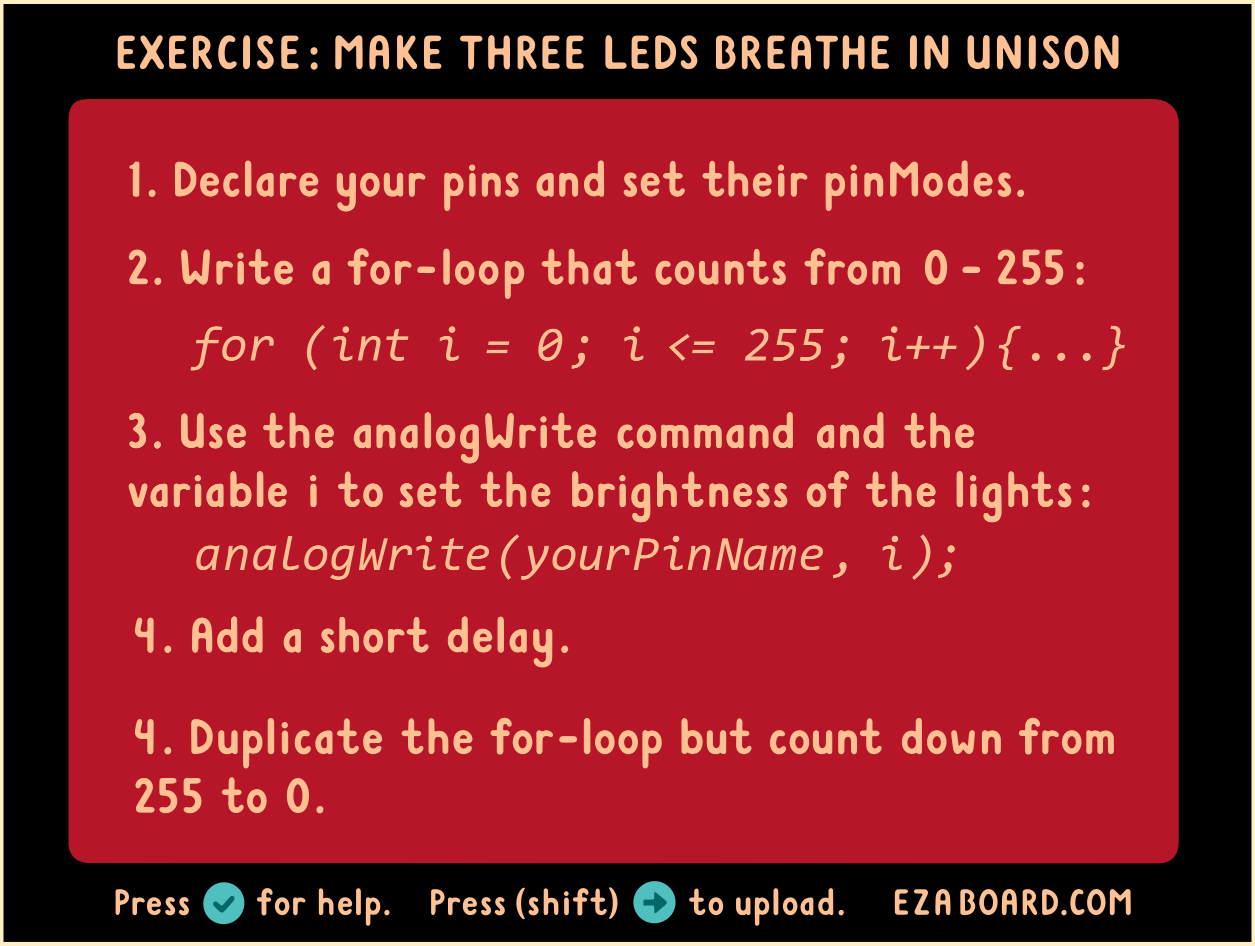
The analogWrite resolution ranges from 0 to 255. 0 is completely off, 255 completely on, and any value in between results in an output intensity between 0 and 100%.

~ The ATMEGA chip can only do analogWrite (PWM) on certain pins. All lights on the board (including the backlight) are connected to PWM capable pins so that you can dim them. In addition pins 11-13 (located below the prototyping board) also have PWM capability. This is indicated by a tilde symbol (~) on the board, because the tilde symbol resembles a PWM square wave.



~ If you send an analogWrite command to a pin not capable of PWM it will simply be rounded up or down to a digitalWrite 1 or 0.

Some notes regarding the capabilities of analog and digital pins:
- All pins (even analog ones) can be digital inputs and outputs.
- Certain pins can also be analog outputs (PWM).
- Only analog pins can be analog inputs.
- There is no relation between pins capable of analog input and pins capable of analog output. That is because pins capable of analog output are not really analog. The phrase "analog pin" refers to a pin capable of analog input.



EXERCISE: MAKE THREE LEDS BREATHE IN UNISON

1. Declare your pins and set their pinModes.

2. Write a for-loop that counts from 0 - 255:

```
for (int i = 0; i <= 255; i++){...}
```

3. Use the analogWrite command and the variable i to set the brightness of the lights:

```
analogWrite(yourPinName, i);
```

4. Add a short delay.

4. Duplicate the for-loop but count down from 255 to 0.

Press ✓ for help. Press (shift) ➜ to upload. EZA BOARD.COM

LESSON 9: ANALOGREAD (RGB LED)

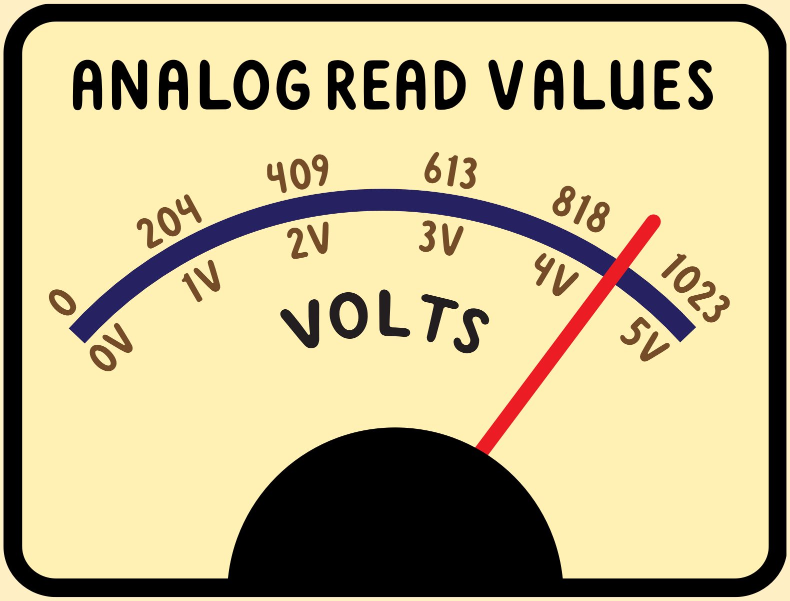
Remember that an analog signal is one with an infinite number of positions between off and on. Even though the ATMEGA chip is a digital device it can read analog signals by converting them into digital representations. A common component that generates an analog signal is a volume knob, also known as a potentiometer. You can read the position of a potentiometer using the analogRead command:

```
    analogRead(knob1); //This code is useless by itself
```

Just like the digitalRead command, this code is pointless since the information is read but not processed. Two common ways of processing the information are by printing it to the serial monitor or saving it to a variable:

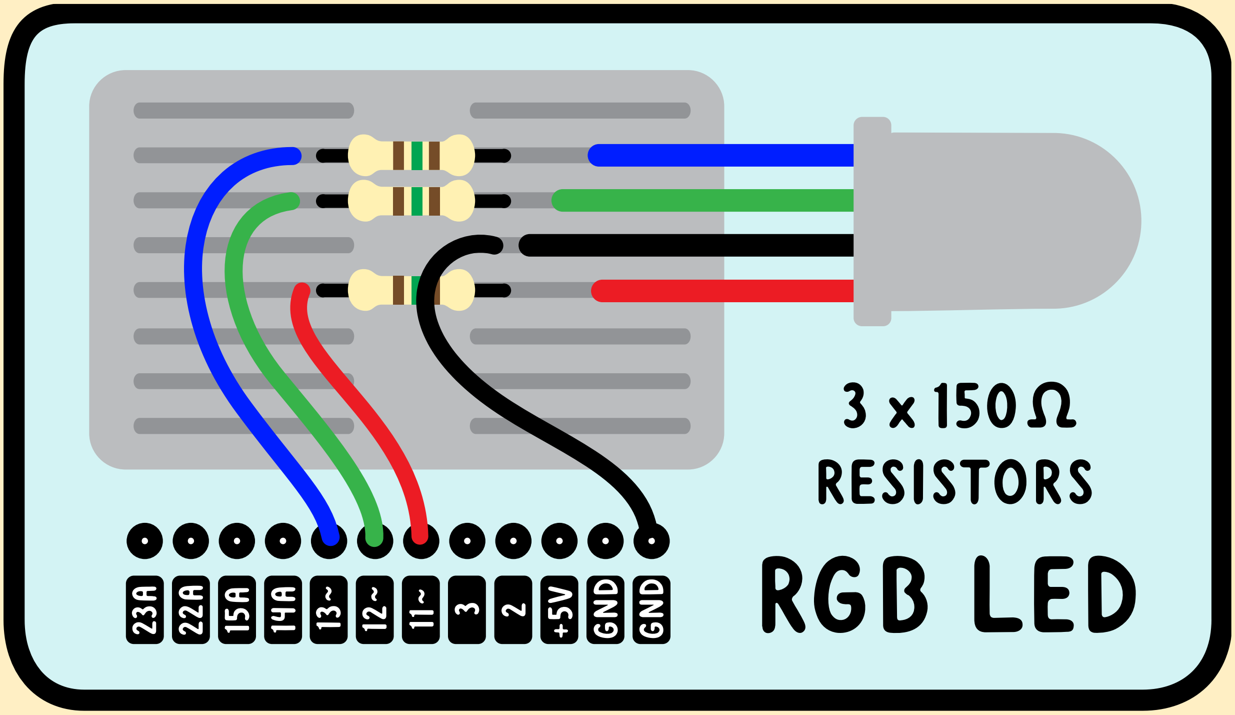
```
    Serial.print( analogRead(knob1) );

    volumeKnobPosition = analogRead(knob1); //int, not byte. See
    below.
```

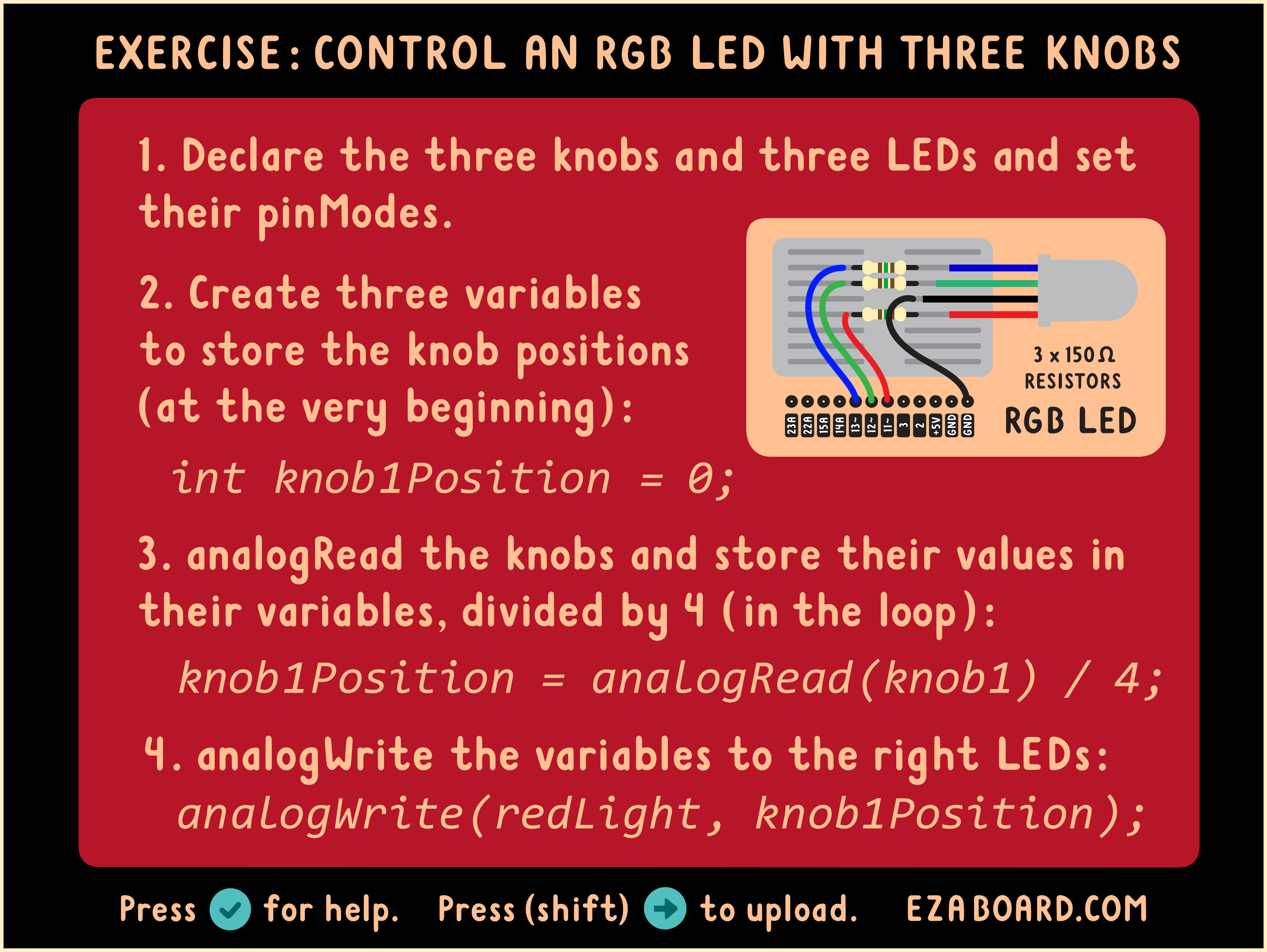
The analogRead command returns values from 0 – 1023. 0 corresponds to 0 volt. 1023 corresponds to 5 volt.



Remember that the analogWrite commands accepts numbers from 0 - 255 to generate an “analog” voltage from 0 – 5 volt. The analogRead command on the other hand, returns values from 0 to 1023 to describe the same 0 – 5 volt range. That just means you get 4 times the precision when using analogRead.

~ A variable used to store an analogRead value should be able to hold a maximum of 1023. Remember that the data type “byte” only goes up to 255. Therefore any variable used to store an analogRead value should be declared as an int (an int can hold numbers up to 32768)!





EXERCISE: CONTROL AN RGB LED WITH THREE KNOBS

1. Declare the three knobs and three LEDs and set their pinModes.

2. Create three variables to store the knob positions (at the very beginning):

```
int knob1Position = 0;
```

3 x 150Ω
RESISTORS
RGB LED
23A
22A
15A
14A
13~
12~
11~
3
2
+5V
GND
GND

3. analogRead the knobs and store their values in their variables, divided by 4 (in the loop):

```
knob1Position = analogRead(knob1) / 4;
```

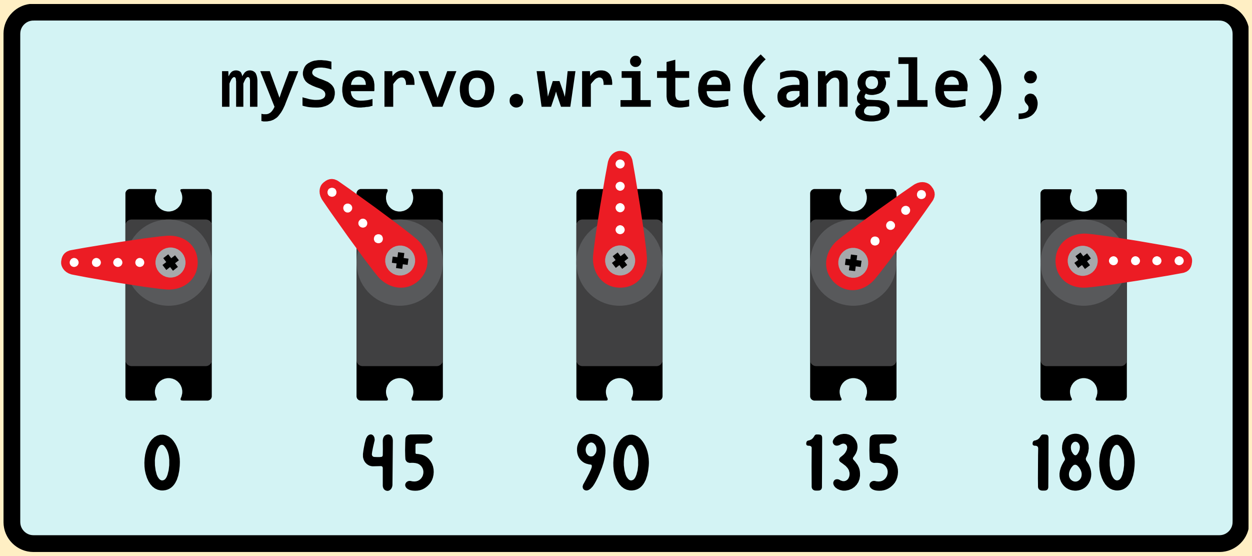
4. analogWrite the variables to the right LEDs:

```
analogWrite(redLight, knob1Position);
```

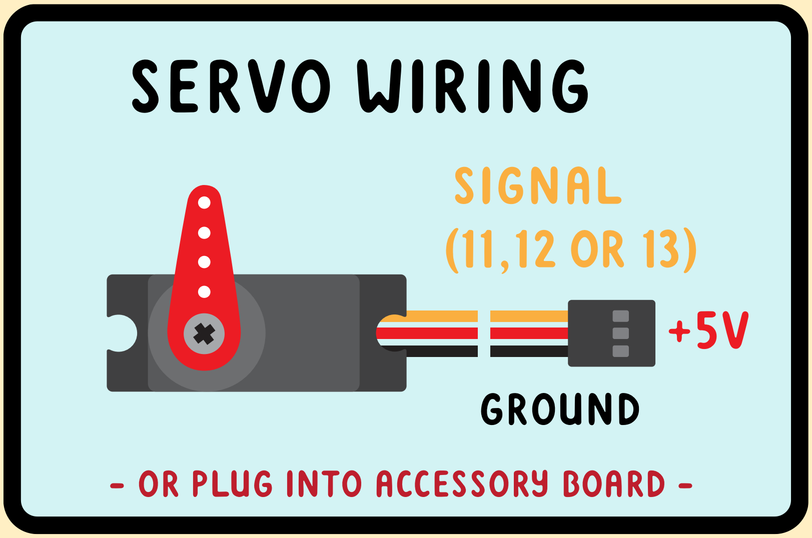
Press ✓ for help. Press (shift) ➜ to upload. EZABOARD.COM

LESSON 10: SERVOS (MOTION-ACTIVATED HOUSE LIGHT)

A servo is a motor that can be commanded to point in a specific direction.



Servos are a great way to act on the physical world. They are easy to use with the EZA Board. Servos only need three wires. If you have the Accessory Board you don't have to wire anything and you can simply plug in up to three servos and power them properly.



In the declaration section include the servo library and name your servo:

```
     #include <Servo.h>
     Servo lightSwitchServo;  //The name of the servo is up to
you
```

In the setup function tell the ATMEGA chip which pin the servo is attached to. This must be a PWM capable pin (11, 12 or 13)!

```
     lightSwitchServo.attach(12);  //The signal wire of the servo
is attached to pin 12 (PWM capable).
```

Command the servo to point to a specific angle:

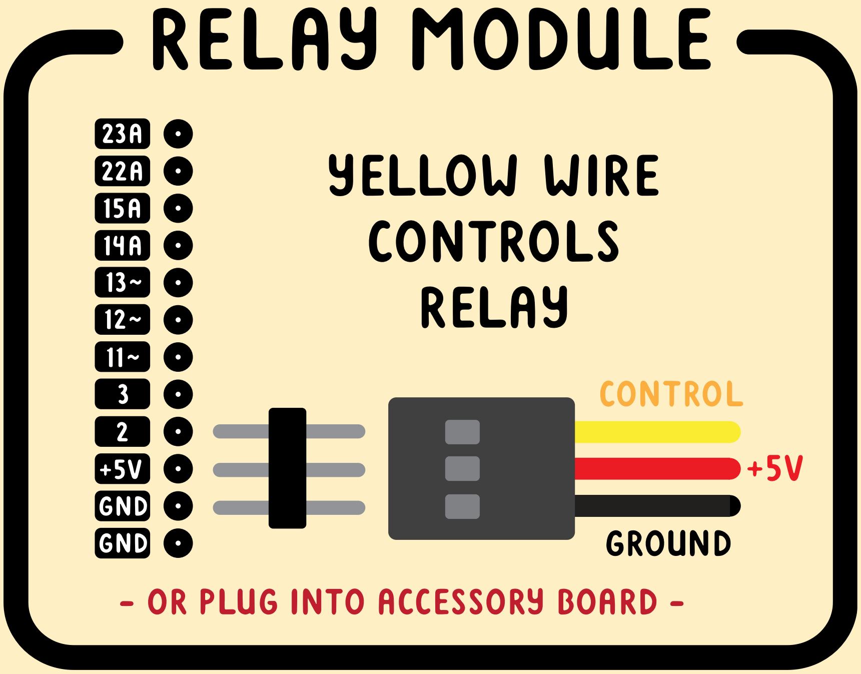
```
     lightSwitchServo.write(90); //Angle between 0 and 180.
```

~ In theory servos have a range from 0 to 180 degrees. Some servos can be damaged by sending them to either end. It is good practice to keep the angle between 10 and 170 degrees.

LESSON 11: TOGGLING A LIGHT (RELAY MODULE)

Let's connect the Relay Module to the EZA Board and experiment with it on pin 2.

If you don't have the Relay Module just substitute one of the on-board LEDs. The lesson is equally useful to you.



In previous lessons we've used code like this:

```
     if (digitalRead(button1) == 1){digitalWrite(relayModule, 
HIGH);}
     else {digitalWrite(relayModule, LOW);}
```

This code is not ideal since you need to keep holding the button down to keep the light on. But what if you want to toggle a light on or off with a single button press?

Please follow along with this reasoning:

- When press the button there are two valid options: either the light should turn ON or it should turn OFF.

- The decision depends on the current state of the light:
- If the light is currently ON, turn it OFF. If the light is currently OFF, turn it ON.
- So our first order of business is to know the current state of the light.

We can keep track of the current state of the light using a variable. Let's call it "lightState". We could implement it like this:

```
     lightState = 0; //means the light is off.
     lightState = 1; //means the light is on.
```

We can then write code like this:

```
     if (lightState == 1){digitalWrite(greenLight, HIGH);}
     else {digitalWrite(greenLight, LOW);}
```

If you toggle the lightState between 0 and 1 you are also toggling the light off and on.

Since this variable can hold only two numbers: 0 or 1, we can use the data type "boolean" to declare the variable:

```
     boolean lightState = 0; //This boolean acts as a flag
```

This type of variable is known as a "flag", a two-state variable whose value acts as a reminder for something else. An American mailbox has a little metal flag attached to it. It is used to indicate to the mailman whether you've got outgoing mail in the box. An upright flag means you do, a downward flag means you don't. In digital terms this would be represented as:

```
     outgoingMailPresent = 1; //We've got outgoing mail! Pick up!
     outgoingMailPresent = 0; //No outgoing mail here!
```

Whenever the button is pressed you want to toggle the light. The decision to toggle the light ON or OFF depends on the current lightState. We need some code like this:

```
if (digitalRead(button1) == 1){ //if button is pressed
     if (lightState == 1){digitalWrite(greenLight, LOW);}
     if (lightState == 0) {digitalWrite(greenLight, HIGH);}}
```

This code works, but only once. The problem is that whenever we toggle the light we also need to update the lightState!

```
if (digitalRead(button1) == 1){ //if button is pressed
     if (lightState == 1){lightState = 0;
digitalWrite(greenLight, LOW);}

     if (lightState == 0) {lightState = 1;
digitalWrite(greenLight, HIGH);}}
```

The lightState is now updated correctly. There is however another insidious problem. Assume that the lightState starts at 1 and go through every line of this simplified example:

```
if (lightState == 1){lightState = 0;}
if (lightState == 0){lightState = 1;}
```

What is your lightState at the end? Still 1! The light when from ON to OFF and immediately back ON! The problem is that we don’t want to consider both options. We only want to take one of the two options. The way to solve this problem is to use an “else if” statement below the “if” statement. With this construction the second statement is only considered if the first one is FALSE. Therefore only one of the two statements can be executed.

```
if (lightState == 1){lightState = 0;}
else if (lightState == 0){lightState = 1;}
```

When you upload and try this code out you will find out that it only works about half the time! Don't give up, we only need one last tweak! A simple delay will fix this issue.

The problem with the code is that the ATMEGA chip is so fast it goes through the entire loop thousands of times per second. Each time it notices the button is still pressed it toggles the lightState. No matter how quickly you press the button, the lightState toggles many times and ends up in an unpredictable position.

A simple delay in the right place can make sure that the code only executes once per button press.

```
if (digitalRead(button1) == 1){ //if button is pressed
     if (lightState == 1){lightState = 0;
          digitalWrite(greenLight, LOW);}
     else if (lightState == 0){lightState = 1;
          digitalWrite(greenLight, HIGH);}
     delay(1000);}
```

~ If your code still doesn't work check the use of equals signs (double equals signs in the comparison portion of if-statements)!

~ For a more elegant solution see the bonus lessons.

You can save yourself some typing by using an if…else statement. In this case there are only two options and one of them has to be true. If the first one is NOT true the second one has to be true. The "else" statement will execute if the first statement is evaluated as false:

```
if (lightState == 1){…} //This code will run if lightState=1
else{…} //This code will execute if lightState is NOT 1
```

Notice that the else statement does not need a set of parentheses!

LESSON 12: WIRELESS COMMUNICATION (ROCKET IGNITER)

When you are programming you don’t always have to write every single line. Sometimes it is sufficient to paste sections or modify examples, especially when you are learning. Using the wireless modules is one of these cases.

Start by opening the Basic_Radio_Transmitter and Basic_Radio_Receiver programs. From here you can write code so that when a button is pressed on the transmitter it will send a message to the receiver to ignite the rocket.

Igniting the rocket requires high current so you will need to use the Accessory Board and a battery. Simply connect around 12V to the input screw terminals and connect the igniter to any of the output screw terminals. Writing the corresponding pin high will send 12V to the igniter and launch the rocket.

BONUS LESSON: WAIT WITHOUT USING DELAY

So far we’ve used delay as a method of waiting. The problem with using delay is that it brings the ATMEGA chip to a halt. The chip has a one-track mind and the delay function ceases execution of code. While in a delay the ATMEGA won’t be able to act on a launch command, for example. As you become a better programmer you will learn to limit the use of the delay command.

The alternative to delay is to use a variable that contains a time in the future. You can then compare this value with the value of millis(), the Arduino function that measures time since startup. If the time of millis() exceeds your memorized time you can execute some code:

```
if (millis() > timeOfNextSomething){
  //Do something like blinking a light
  timeOfNextSomething = millis() + 500;}
```

While this code is running you can do other things, like listening for button presses or radio messages!

~ Remember that millis() always needs its own set of parentheses!!!

The variable “timeOfNextSomething” holds the time in millis. Since millis() counts the time since startup in 1000 steps per second this can quickly turn into a very large number. Therefore “timeOfNextSomething” should be declared using the data type “unsigned long”.

A regular “long” data type can hold values from -2.1 billion to 2.1 billion. Since the millis() measures time since startup it is never negative. The unsigned long is simply a long without the negative portion, measuring numbers from 0 to 4.3 billion instead. “Unsigned” just means no negative sign.

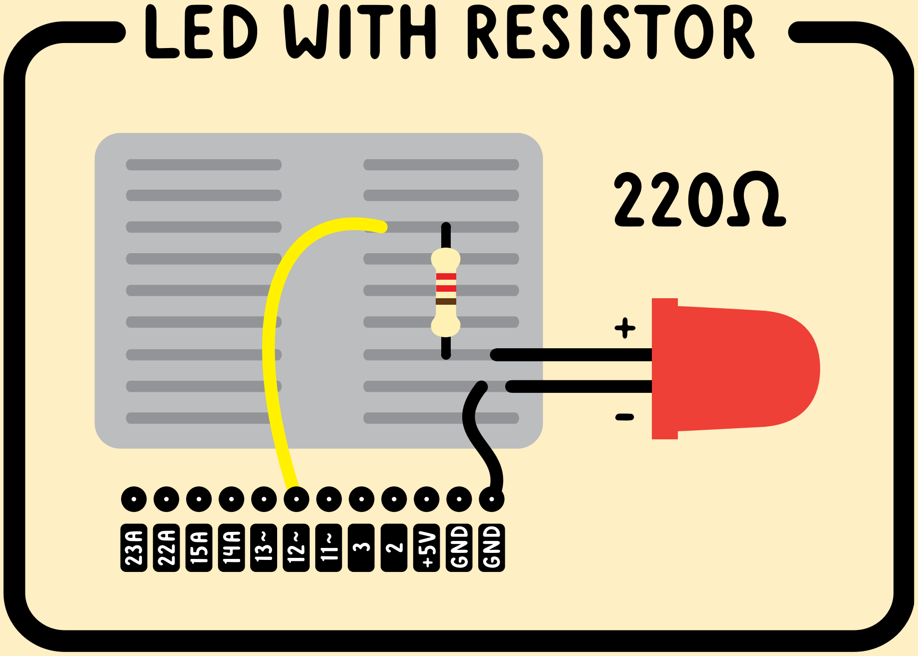
~ Remember that any variable containing time should be declared as an unsigned long!

~ A slightly harder to understand but technically superior way to see if millis() has exceeded a set time is to use the code:

```
if (millis() – startTime > 5000){
     startTime = millis();
```

BONUS LESSON: ADDING AN LED

If you hook up an LED to your EZA Board it is possible to damage either the chip or the LED. You can avoid any damage by using a current limiting resistor. A value between 150 ohm and 330 ohm is usually chosen, and should prevent any damage.

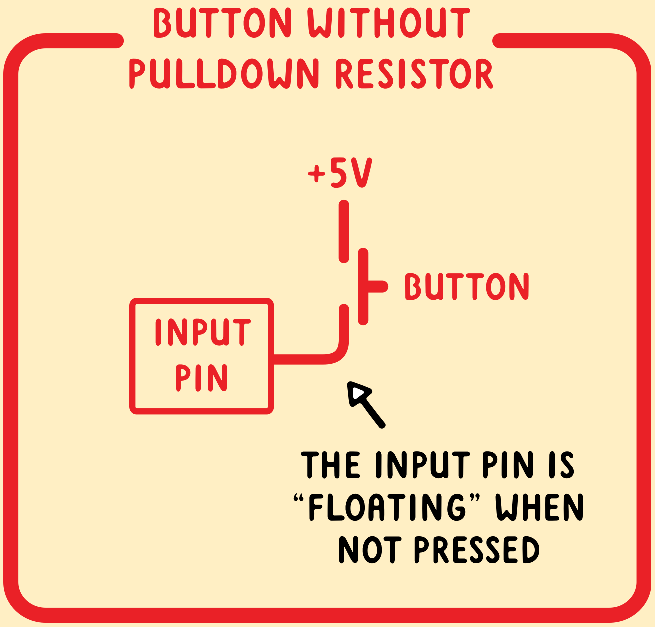

LED WITH RESISTOR

An LED has polarity, meaning that it has to be hooked up one way. Hooking it up backwards will not work, though you won't damage the LED. The "cathode" or negative side is indicated by the shorter leg of the LED and has to be connected closer to ground (0V) than the other leg.

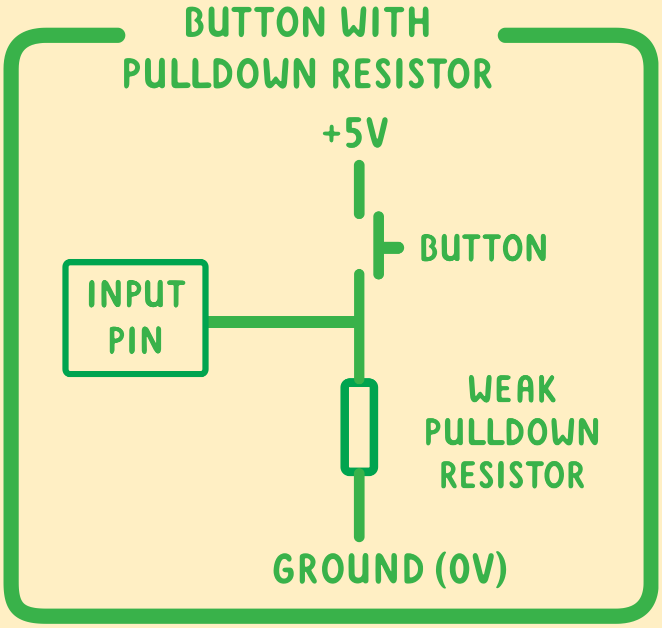
The resistor has no polarity; switching it around does not make a difference. Also, you could swap the order of the resistor and LED.

BONUS LESSON: ADDING A BUTTON

When a beginner adds a button he or she will likely wire it up like the red diagram. Whenever the button is pressed it closes the contact and the input is connected straight up to 5 volt, registering as a digital 1. No problem so far. Whenever the button is not pressed the input pin should measure 0 volt since it is not connected to anything, right? Well, not necessarily. In the real world there is noise. Input pins are very sensitive and act like antennas when not connected to anything. This causes pins to pick up random electrical signals. The ATMEGA will interpret some of this random noise as button presses even though the button is not pressed!

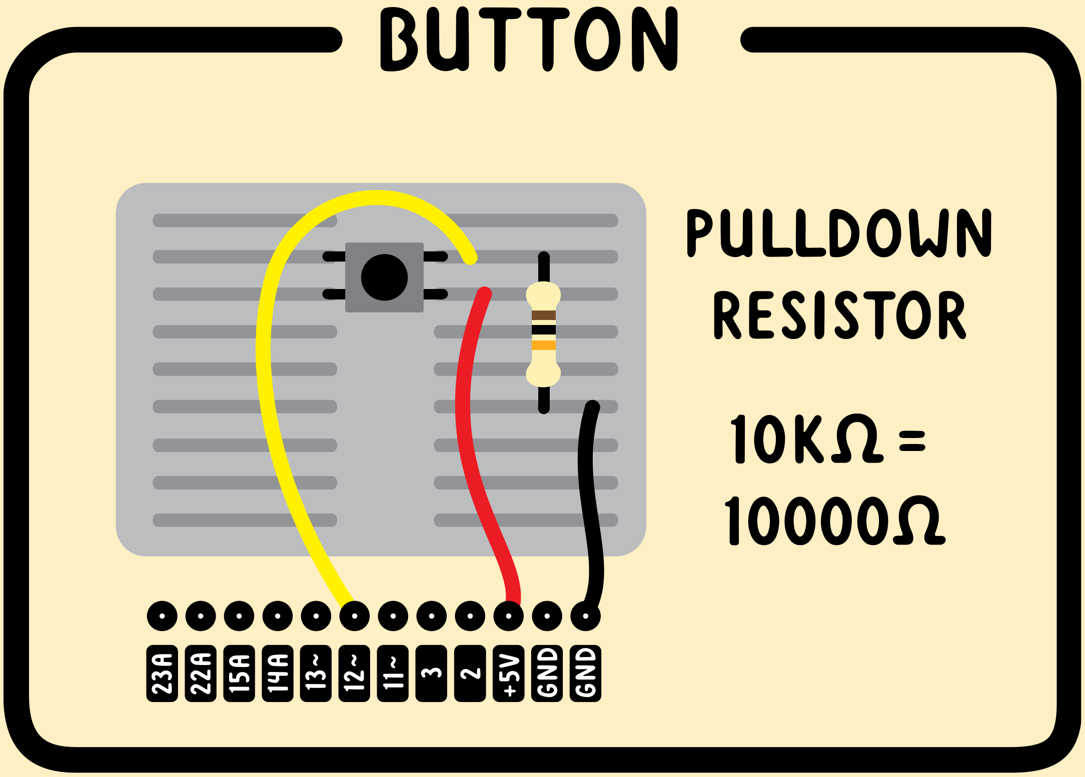

BUTTON WITHOUT PULLDOWN RESISTOR
+5V
BUTTON
INPUT PIN
THE INPUT PIN IS "FLOATING" WHEN NOT PRESSED

Inputs not connected to anything are said to be floating. Floating inputs cannot be trusted since they may randomly register any value. The solution is to add a weak resistor. It overpowers the noise by pulling the input down to 0 volt. This resistor is just strong enough to overpower the noise but not nearly strong enough to overpower the button when pressed. Noise is no longer a problem. The input pin will consistently read 0 unless the button is pressed, then it will read 1.



~ The standard value for a pulldown resistor is 10 000 ohms, usually referred to as 10K ohms or simply a 10K resistor.

In real life this circuit would look like this:

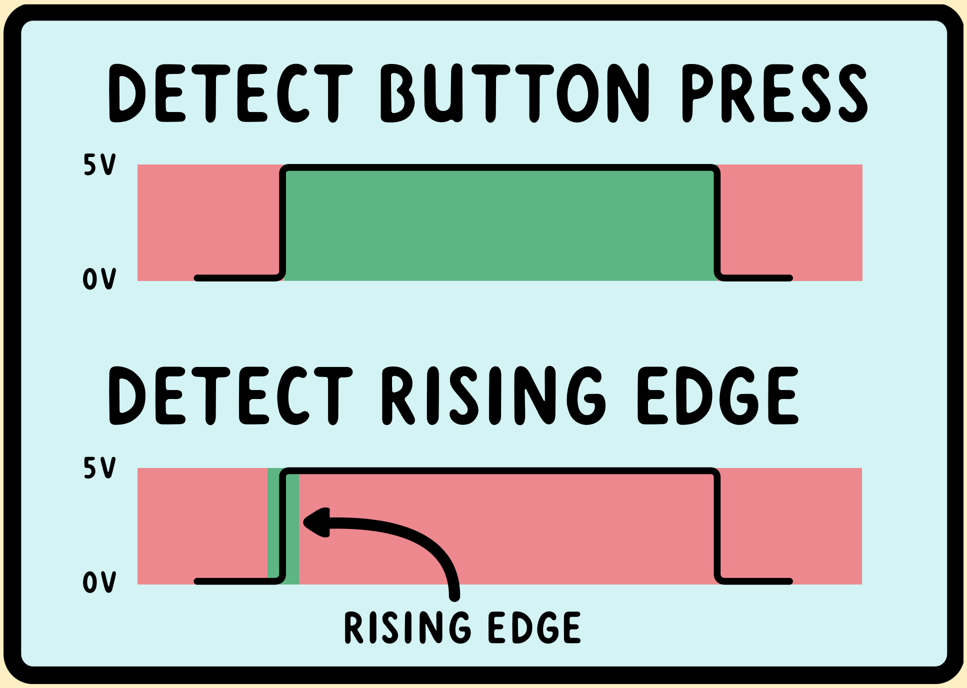


~ Remember that the pulldown resistor has to be located on the input pin side of the button, not on the 5V side of the button!

~ Pins configured as outputs don’t need pulldown resistors. Output pins are internally connected to either 0V or 5V with a low resistance connection. Noise cannot possibly override this “strong” connection.

BONUS LESSON: TOGGLING AN LED (PART II)

Often you would like to trigger an event once every button press, no matter how long the button is held down. You can accomplish this *not* by detecting whether the button is currently pressed, but by detecting the "rising edge" of the button press.



You need some code that can make the Arduino reason like this: "Last time I checked the button it was *not* pressed. Now it *is* pressed. That means someone *just* started pressing it."

This requires some "short-term memory". You could declare a variable (a flag) called "buttonStateLastTime" to act as short-term memory. The ATMEGA chip can then compare "buttonStateLastTime" with another variable called "buttonStateNow". If "buttonStateLastTime" == 0 and "buttonStateNow" == 1, the button was just pressed:

```
buttonStateNow = digitalRead(button3);
if ((buttonStateNow == 1) && (buttonStateLastTime == 0)){...}
```

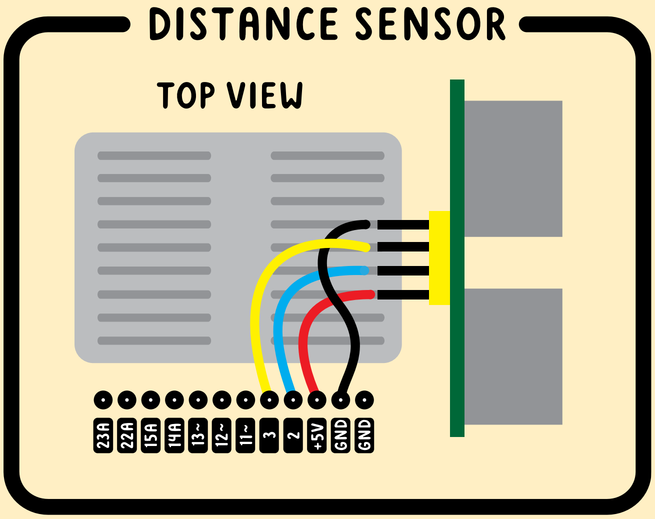
~ After the if-statement you need to update buttonStateLastTime so that it can be compared to a future button state next time around:

```
     buttonStateLastTime = buttonStateNow;
```

~ Occasionally a single button press might be registered twice or more; this happens due to a mechanical phenomenon called "switch bounce" where the switch contacts bounce a few times before making solid contact. This happens too quickly to notice for us but on an ATMEGA it could lead to multiple rising edges. The solution is to add a short delay after a press has been detected, let's say 10 milliseconds. This delay allows the switch contacts to settle before allowing any more button presses to register.

```
     buttonStateNow = digitalRead(button3);
     if ((buttonStateNow == 1) && (buttonStateLastTime == 0)){
          delay(10);
          //Rising edge detected. Do something here...
          }
     buttonStateLastTime = buttonStateNow;
```

BONUS LESSON: ADVANCED IF-STATEMENTS (PARKING SENSOR)



The main component of the parking sensor is an ultrasonic module. It sends out a sound wave and waits for the echo to return after it has reflected of an object. The time between sending and receiving is known as time of flight and can be converted to a distance measurement.

Wiring up the module is easy. It only has a trigger pin, an echo pin, and power.

In order to initiate a measurement you have to pulse the trigger pin high for a short duration:

```
     digitalWrite(triggerPin, HIGH);
     delay(1); //1 millisecond = (1/1000th of a second)
     digitalWrite(triggerPin, LOW);
```

After the trigger pin is pulsed you wait until the echo returns and the echo pin goes high. This needs a precise timing measurement so use the pulseIn command:

```
     timeOfFlight = pulseIn(echoPin, HIGH); //timeOfFlight must
be an int
```

To convert this duration to a distance use the formula:

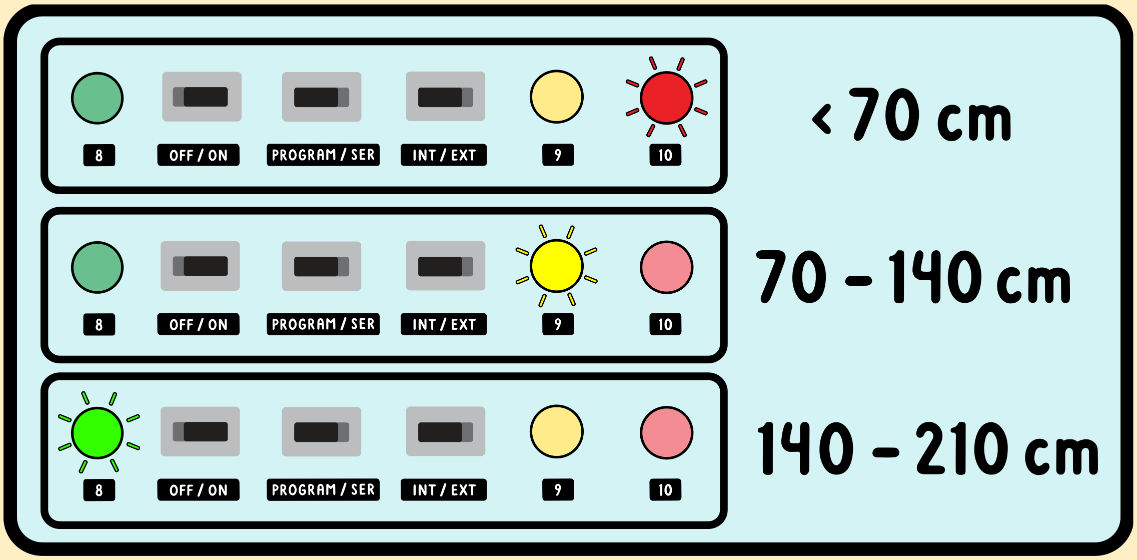
```
     distanceInCm = timeOfFlight * 0.0343 / 2;    //speed of
sound = 343 m / s.
```

~ Distances measured will exceed 255cm so the variable distanceInCm must be an int, not a byte.

Testing the system in the garage by moving the car is cumbersome. You can simulate the changing distance at your desk by using this line:

```
     distanceInCm = analogRead(knob3) / 3; //Turning knob 3 will
simulate a distance measurement
```

Desired functionality of the parking sensor code:



The yellow light should turn on when the distance is *between* 70 and 140 cm. There is no “in between” command in the Arduino language. However you can achieve the same result using an if-statement with multiple conditions. The setup of such a conditional function is as follows:

```
     if ( (A == true) && (B == true) ){…
```

This means that if condition A is true and condition B is true, the if-statement will be executed. The code for AND is the double ampersand (&&).

In the case of the parking sensor:

```
     if ( (distanceInCm > 70) && (distanceInCm < 140) ){…
```

~ Notice that each condition is in its own set of parentheses. The complete set of conditions is also in parentheses.

Since this is an if-statement you might have expected to see the double equals sign (comparison operator). However this particular if-statement did not:

```
     if (distanceInCm > 70){…
```

The double equals sign (standard comparison operator) is only used to detect whether two values are *identical*. There are other comparison operators you can use inside your if-statements:

COMPARISON OPERATORS (compare one value to another)	
Code:	Meaning:
`if (x == 23){…`	if x **equals** 23…
`if (x < 23){…`	if x **is less than** 23…
`if (x > 23){…`	if x **is greater than** 23…
`if (x <= 23){…`	if x **is less than or equal to** 23…
`if (x >= 23){…`	if x **is greater than or equal to** 23…
`if (x != 23){…`	If x **is not** 23…

~ The order of the symbols in the last three comparison operators is important. The equals sign always comes last.

To reiterate, never use a single equals sign to compare anything!

ASSIGNMENT OPERATORS (assign a value to something)	
Code:	Meaning:
`x = 23;`	**Set** variable x to 23

→ FREQUENTLY ASKED QUESTIONS ←

My EZA Board seems dead.
Make sure the USB on the back of the board is plugged in. Make sure the power button (upper left corner) is set to ON. The board can be perfectly functional but show no signs of life if the chip does not contain the right program. Apply settings on page 1 and try uploading the demo program.

My LCD does not work.
Your contrast is likely off. Turn the contrast adjustment on the back of the board until you see characters. If you don't see characters but are able to see all black pixels you likely have a software problem. Try uploading the demo program (and watch the video!).

I'm having upload issues
- <u>Make sure the settings in the tools menu are configured correctly! This is the number one cause of upload issues.</u> See first page for correct settings. (There might be two SerialUPDI 230400 options in the Tools > Programmer menu. Try the other variant.)

- Make sure the switches on the board are configured correctly! See first page.

- Try disconnecting and reconnecting the USB cable.

- Temporarily connect directly to your computer without going through a USB hub. Inexpensive USB hubs can interfere with uploading.

- Instead of selecting the regular upload option you might need to use "Upload using programmer" from the "Sketch" menu.

The Arduino software is giving me mysterious errors when I try to upload.
The Arduino software does a check before allowing you to upload. This catches many mistakes.

A common beginner mistake is failing to put a semicolon at the end of an action or declaration. This will lead to an error messages, usually on the line after the one with the missing semicolon.
Also check your use of brackets. Adding or leaving out one curly bracket can confuse the software. It may think the problem is somewhere else in your code but it is really a missing curly bracket somewhere else. This is especially likely when you start adding external functions. Simplify your code and see if the problem persists.

How can I drive a motor using the EZA Board?
An ATMEGA pin can provide a maximum of 40 mA (milliAmps). Even though that could power a very small motor it is never a good idea to drive a motor directly from one of the pins. Motors generate a lot of electrical noise and can cause your ATMEGA to misbehave or be damaged. Instead use the Accessory Board to drive motors or use the Relay Module to drive mains-powered motors. For more information see the section on accessories.

How can I control a mains-powered device such as coffee maker or lamp?
The Relay Module is the only safe way to control the dangerous voltage of mains power. See the section on accessories.

My external device does not work even though I've hooked up power.
Beginners usually don't realize how important ground (0 volt) is. It is just as important as the positive terminal. Just because ground is at 0 volt does not mean it is unimportant. Remember that electricity can only go around in a circle. If it cannot leave a device it cannot enter either.

What are the INT/EXT selector and white connector for?
Programming external boards. See the section on external boards.

How can I avoid damaging my EZA Board?

Damaging your EZA Board is unlikely as long as you don’t introduce an external voltage above 5 volt or set a pin to output while it is connected to ground or 5V.
Damaging the chip is most likely to occur when you draw more than 40mA from a pin. This is impossible for pins configured as inputs. You can damage the chip by misconfiguring a pin intended to be an input as an output. If a pin connected to a button is set as output and then pressed, the pin will be fried. The reason is that you will connect that output pin (which is at 0 volt) directly to 5V through the button. You create a short circuit from 5 volt to the pin, which is connected to 0 volt. This is also possible with the potentiometers (volume knobs). Before setting a pin as an output just make sure that it cannot be directly connected to either 0 or 5V.

When you play with external components such as LEDs you could damage a pin. You will need a current limiting resistor. See the bonus lessons on LEDs. A value of 150 ohm or higher will prevent you from damaging a pin. These resistors are already installed on all on-board LEDs.

If you do manage to damage the chip you can easily swap it out! The ATMEGA4809 is socketed, meaning you can carefully pry it up and swap it out.

What are the different methods for powering the EZA Board?
The simplest way to run your EZA Board without using your computer is to simply use any cellphone charger plugged into an outlet.
If you want to be completely untethered to the power grid you can use two battery options. The simplest is a portable USB charging bank. These compact devices can even be stored inside the EZA Board stand.
The other alternative is to use the Accessory Board. It can supply 5V to the EZA Board while being supplied with a higher voltage, such as an AA battery pack, 2 or 3 cell LiPo battery, or car battery.

My LCD backlight occasionally flickers when using the wireless module.
Use a USB port with more than 1A capacity. When transmitting the RFM69 transmit module can consume quite a bit of current, and that can pull down

the board voltage just enough to make the backlight reduce in brightness for a fraction of a second. A USB port with enough current capacity will be able to keep up and prevent the backlight from flickering.

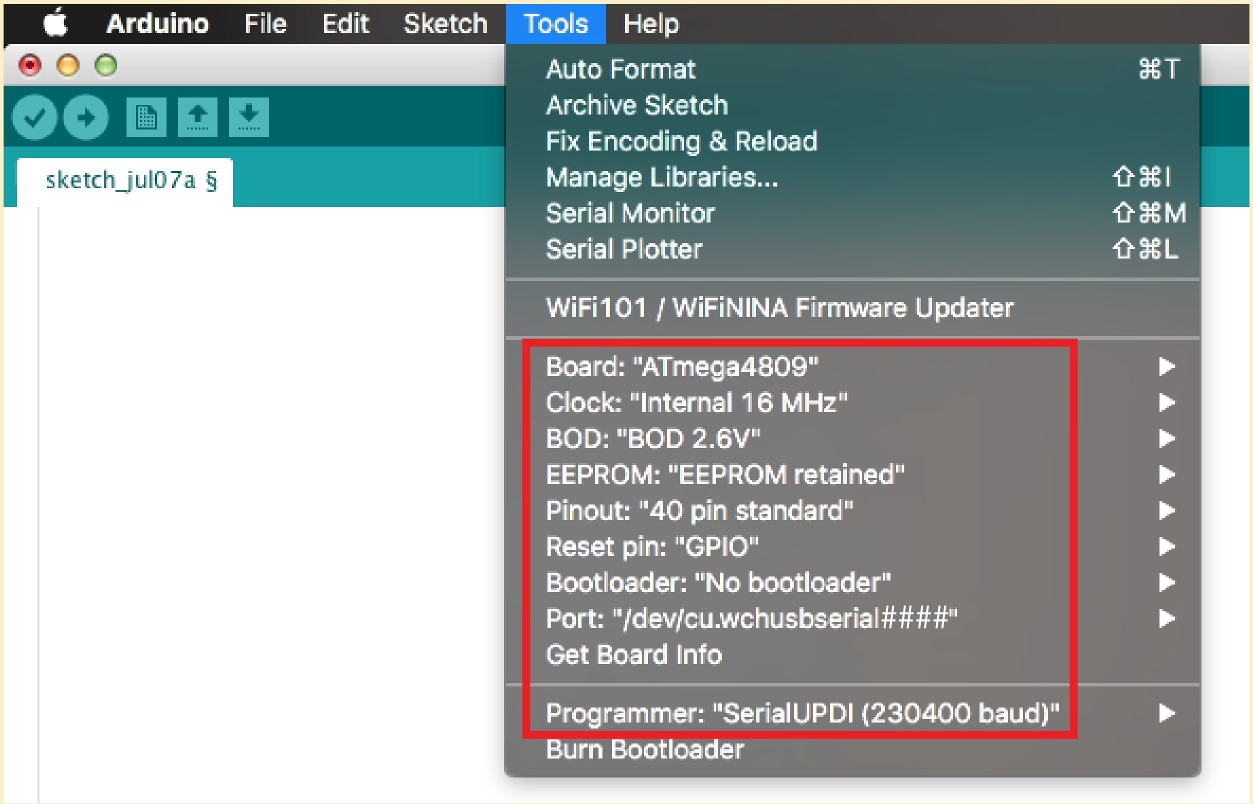
The chip on my board is getting hot.

You most likely somehow applied more than 5V to one of the pins. Your chip is fried. Since the ATMEGA4809 is socketed you can easily swap it out with another ATMEGA4809 chip (be sure to get the through-hole version, not the surface mount version).

What are the mounting hole dimensions:

The mounting hole pattern is 126 x 107mm. The hole diameter is 2.5mm.

What do the upload settings actually do?



Board: "ATmega4809"
Clock: "Internal 16 MHz"
BOD: "BOD 2.6V"
EEPROM: "EEPROM retained"
Pinout: "40 pin standard"
Reset pin: "GPIO"
Bootloader: "No bootloader"
Port: "/dev/cu.wchusbserial####"
Programmer: "SerialUPDI (230400 baud)"

Board: “ATmega4809”

You can find the ATmega4809 under the MegaCoreX heading

Clock: “Internal 16 MHz”
The clock is like the conductor of an orchestra; it makes sure everyone is synchronized with everyone else. Your clock speed determines how fast your chip operates. Be sure to select one of the internal options. 16 MHz is standard.

BOD: “BOD 2.6V”
Chips are designed to run in a specific voltage range. Going under the minimum acceptable voltage may result in random errors and memory corruption. The Brown Out Detector prevents this by putting the chip in reset state while a low voltage condition is detected. Any of the voltages listed is acceptable.

EEPROM: “EEPROM retained”
Your ATMEGA chip has several kinds of memory. The EEPROM is a section of memory accessible to your code that is retained even with power off. If you select “EEPROM not retained” this memory section will be cleared each time you upload. This is rarely necessary.

Pinout: “40 pin standard”
This version of the ATMEGA4809 has 40 pins.

Reset pin: “GPIO”
Pin 32 can be used as a reset pin or an input/output pin. On the EZA Board it is used as an output. It must therefore be set to “GPIO” (General Purpose Input/Output). If you select “Reset” your chip might appear to stop working.

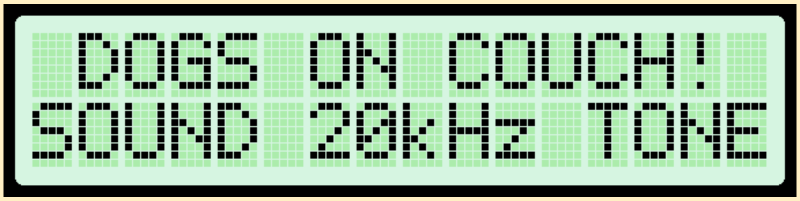
Bootloader: “No bootloader”

Port: “/dev/cu/wchusbSerial####”

Programmer: “SerialUPDI (230400 baud)”
You can attempt a lower rate if you have problems.

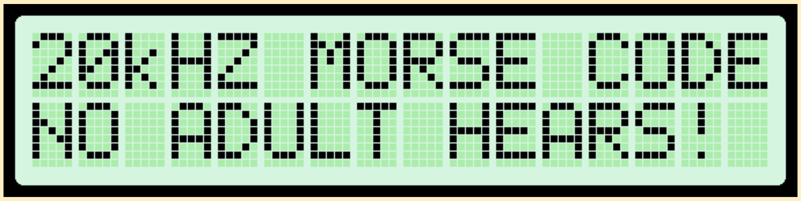
~ Notice there is another entry with a very similar title, that one probably won’t work for you.

100+ PROJECT IDEAS:



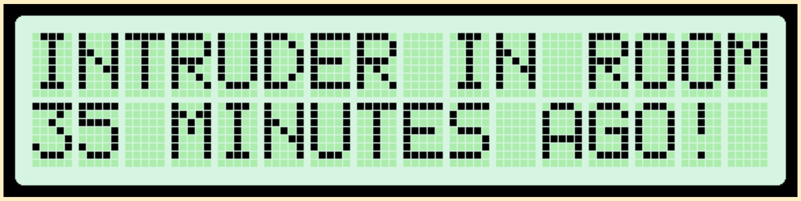
```
 DOGS ON COUCH!
SOUND 20kHz TONE
```

Teach pets not to sit on couch (ultrasonic sensor & buzzer).



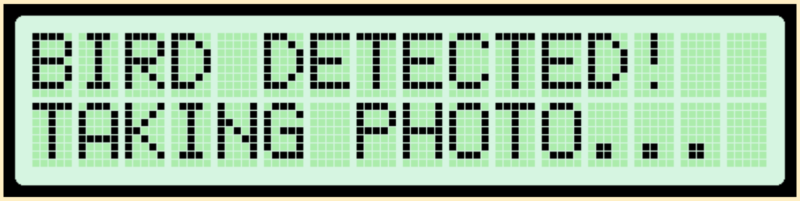
```
20kHZ MORSE CODE
NO ADULT HEARS!
```

Kids send Morse code to each other at high freq so that no adult can hear it.



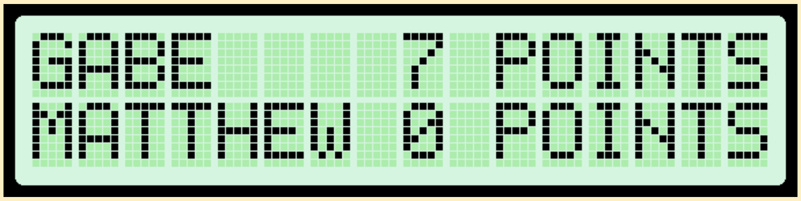
```
INTRUDER IN ROOM
35 MINUTES AGO!
```

Detect if someone has entered the room in your absence (take photo).



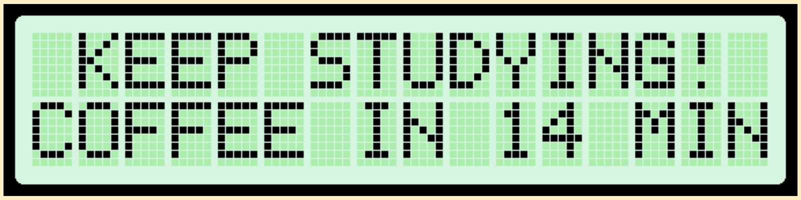
```
BIRD DETECTED!
TAKING PHOTO...
```

Make many DSLRs/professional cameras motion activated (kids: bird feeder cam).



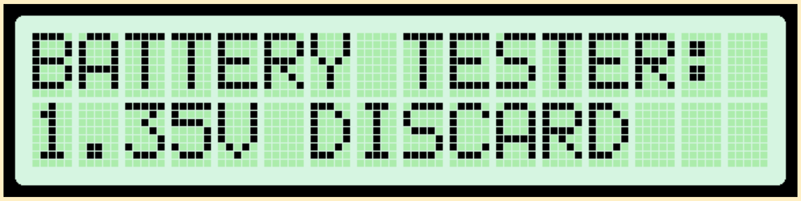
```
GABE    7 POINTS
MATTHEW 0 POINTS
```

Score keeper for games (ping pong).



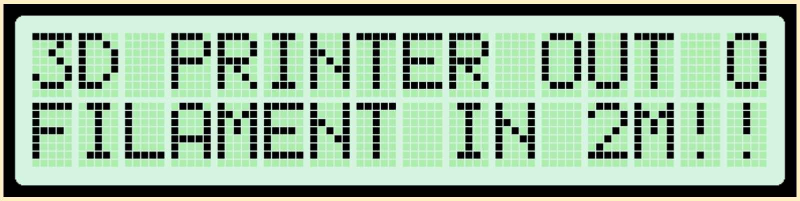
```
 KEEP STUDYING!
COFFEE IN 14 MIN
```

Bible/study timer with automatic reward; when done timer makes you coffee.



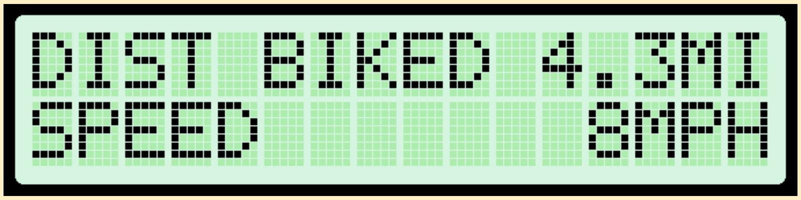
```
BATTERY TESTER:
1.35V DISCARD
```

Battery tester (test 1.5V batteries using analogRead).



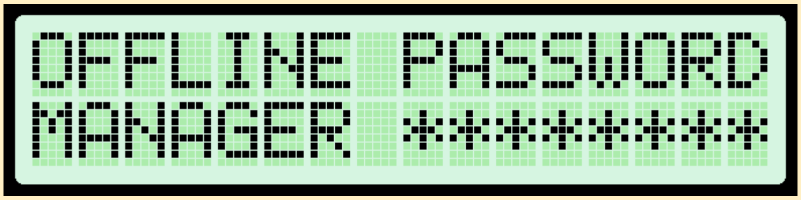
```
3D PRINTER OUT O
FILAMENT IN 2M!!
```

3D printer end of filament warning system.



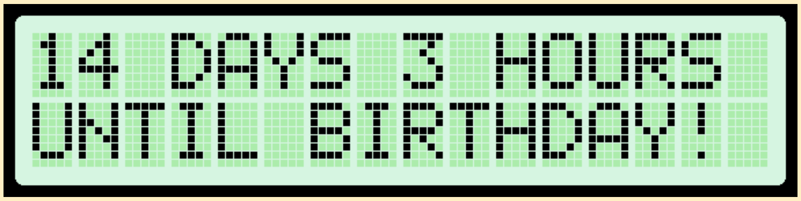
```
DIST BIKED 4.3MI
SPEED       8MPH
```

Kids: measure speed/distance of bike (hall effect sensor & magnet).



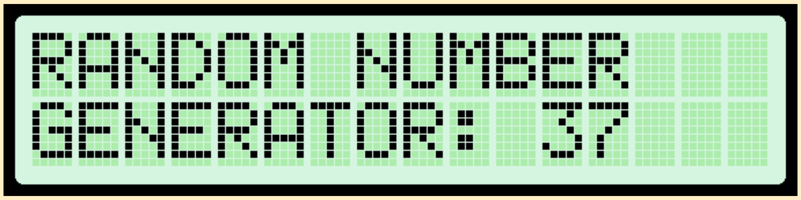
```
OFFLINE PASSWORD
MANAGER ********
```

Offline password storage vault with master password & memory destruction after number of incorrect attempts.



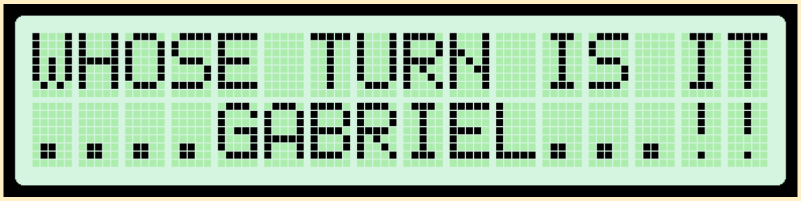
```
14 DAYS 3 HOURS
UNTIL BIRTHDAY!
```

Countdown to important day (game, birthday, release from prison).



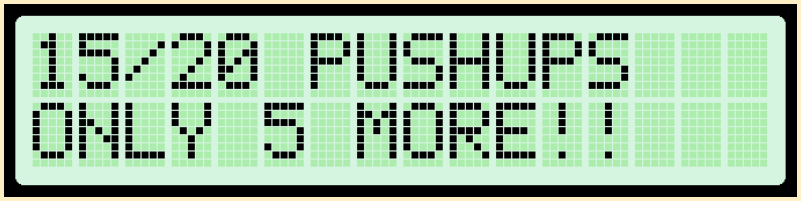
```
RANDOM NUMBER
GENERATOR: 37
```

Random number generator for games (random playing card generator/random die number).



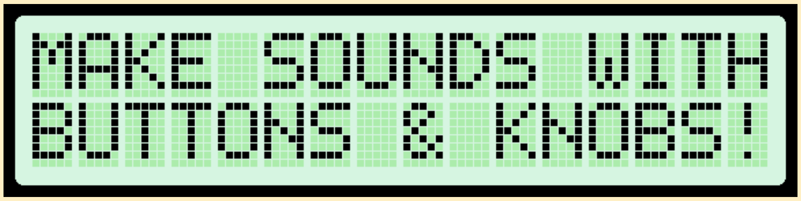
```
WHOSE TURN IS IT
....GABRIEL...!!
```

Random kids’ name picker (to determine whose turn it is).



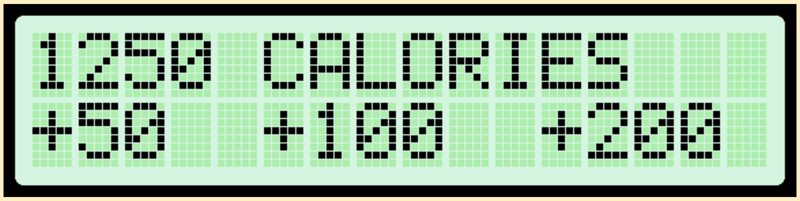
```
15/20 PUSHUPS
ONLY 5 MORE!!
```

Exercise counter: count pushups.



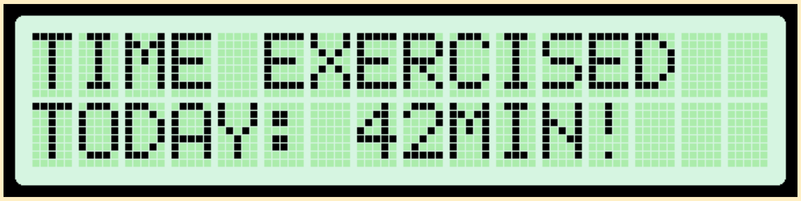
```
MAKE SOUNDS WITH
BUTTONS & KNOBS!
```

Audio synthesizer for kids.



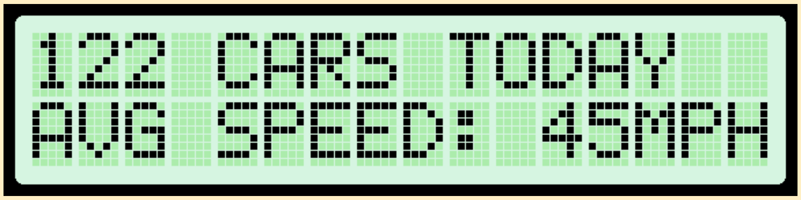
```
1250 CALORIES
+50  +100  +200
```

Calorie tracker.



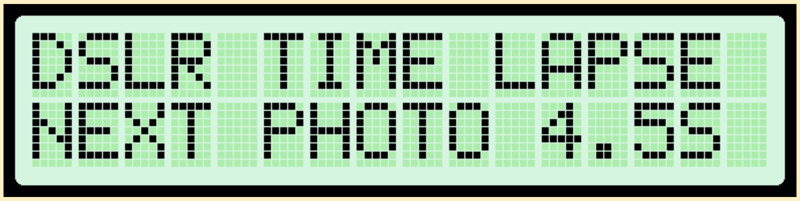
```
TIME EXERCISED
TODAY: 42MIN!
```

Goal tracker.



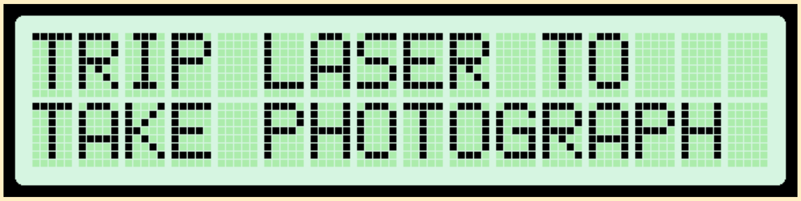
```
122 CARS TODAY
AVG SPEED: 45MPH
```

Count number of vehicles passing by & their speed.



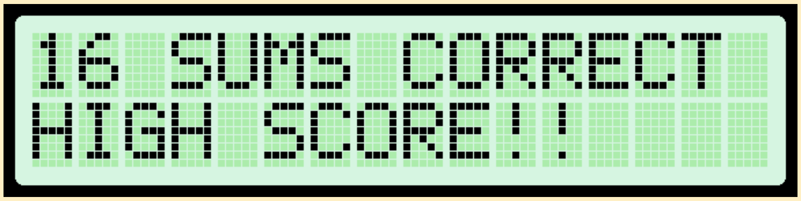
```
DSLR TIME LAPSE
NEXT PHOTO 4.5S
```

Add time-lapse and or delay functionality to many professional cameras (plant growing).



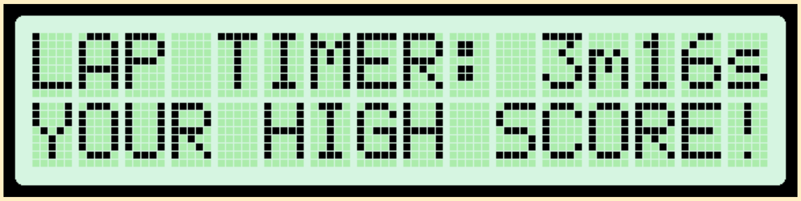
```
TRIP LASER TO
TAKE PHOTOGRAPH
```

Automatically take photograph the very moment you jump (laser and LDR).



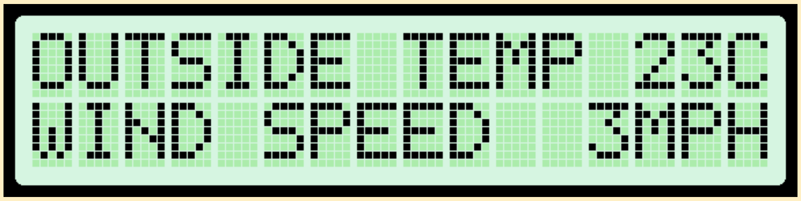
```
16 SUMS CORRECT
HIGH SCORE!!
```

Math trainer with score keeping (addition or multiplication) for kids



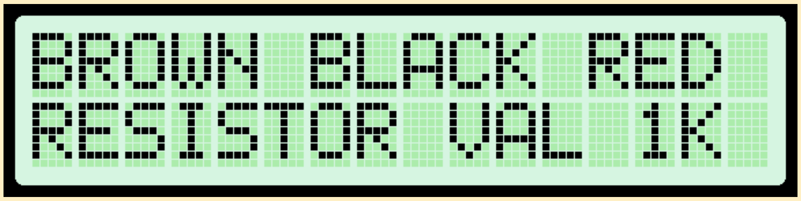
```
LAP TIMER: 3m16s
YOUR HIGH SCORE!
```

Kids' lap timer to run/bike around the house (with personal best memory) (with laser beam?).



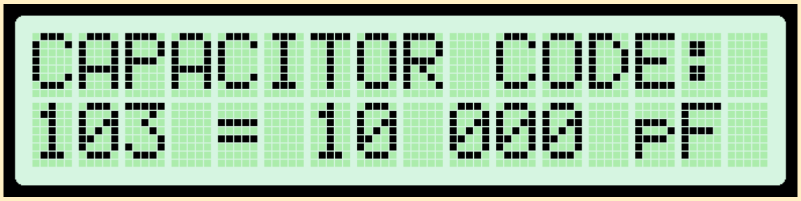
```
OUTSIDE TEMP 23C
WIND SPEED  3MPH
```

Wind speed sensor (a fan moved by the wind will generate a voltage).



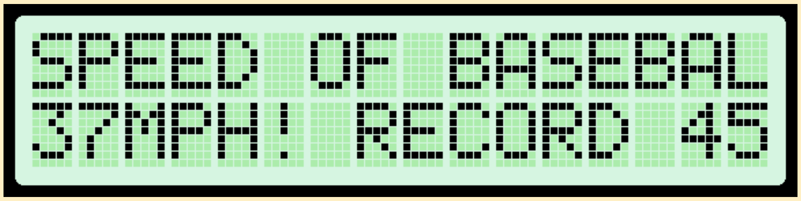
```
BROWN BLACK RED
RESISTOR VAL 1K
```

Resistor color band value calculator.



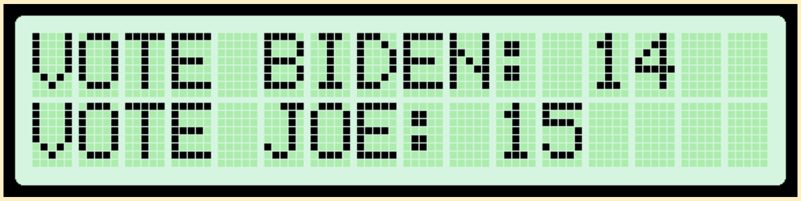
```
CAPACITOR CODE:
103 = 10 000 pF
```

Capacitor code value calculator.



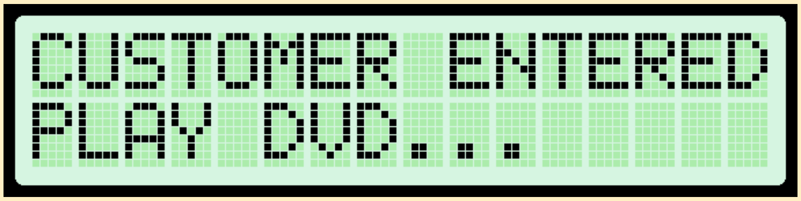
```
SPEED OF BASEBAL
37MPH! RECORD 45
```

Kids project: measure speed of baseball throw (two 2x4 square frames lined on the inside with mirrors. Laser pointer bounces back and forth until it reaches light dependent resistor. Throwing ball through any point in frame interrupts laser beam).



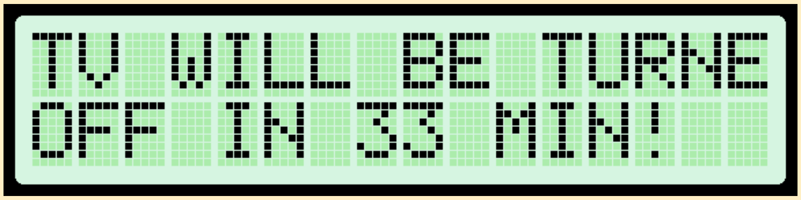
```
VOTE BIDEN: 14
VOTE JOE: 15
```

Anonymous voting for clubs/informal occasions (with audible beep so that double voters will be exposed).



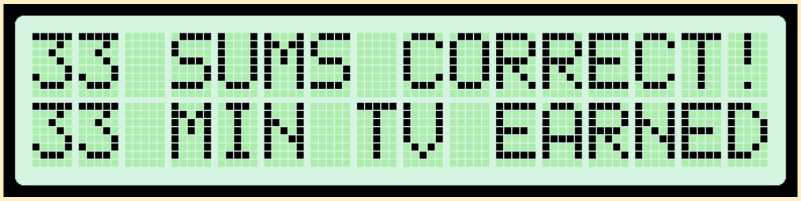
```
CUSTOMER ENTERED
PLAY DVD...
```

TV/DVD player activated by motion (when someone enters waiting room/store).



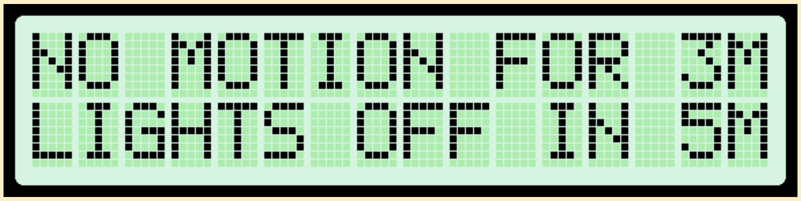
```
TV WILL BE TURNE
OFF IN 33 MIN!
```

Limit kids' use of TV with shutoff timer (enter secret code to allow limited time of TV. Include tamper protection).



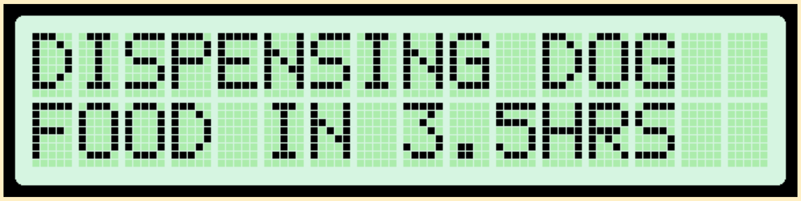
```
33 SUMS CORRECT!
33 MIN TV EARNED
```

Kids can earn TV time by answering a number of math sums correctly.



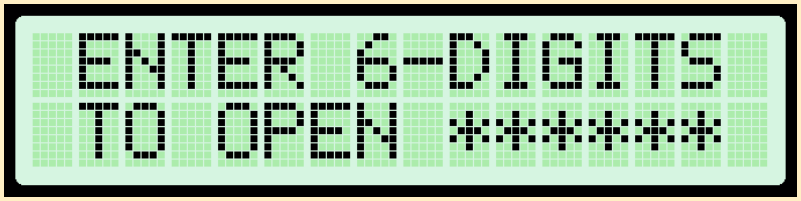
```
NO MOTION FOR 3M
LIGHTS OFF IN 5M
```

Energy saver: turn off light when no presence detected.



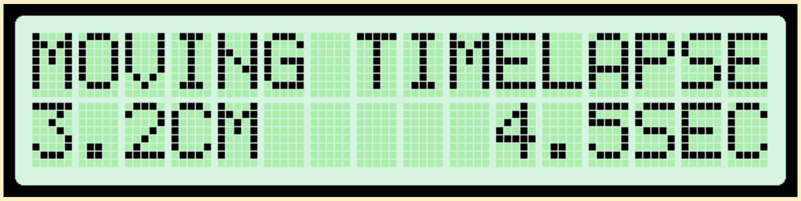
```
DISPENSING DOG
FOOD IN 3.5HRS
```

Automatic pet feeder.



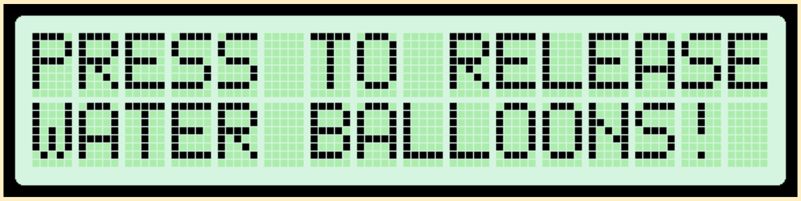
```
 ENTER 6-DIGITS
 TO OPEN ******
```

Locked cabinet with digital combination code.



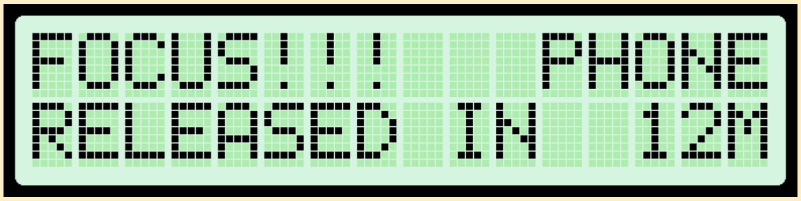
```
MOVING TIMELAPSE
3.2CM      4.5SEC
```

Record moving time lapse (continuous servo to wind up string that pulls camera dolly up a sloping track).



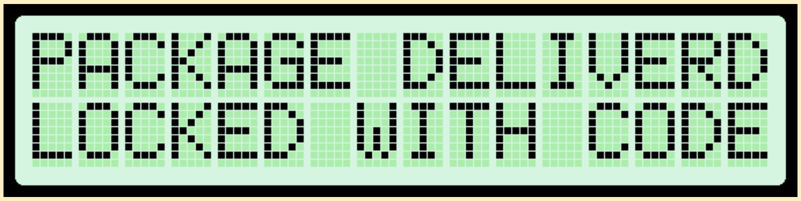
```
PRESS TO RELEASE
WATER BALLOONS!
```

Practical joke: Drop multiple water balloons on someone remotely.



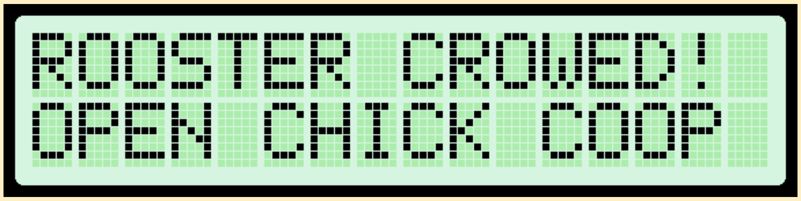
```
FOCUS!!!   PHONE
RELEASED IN  12M
```

Avoid distractions while studying by locking your cellphone inside a timed lock box.



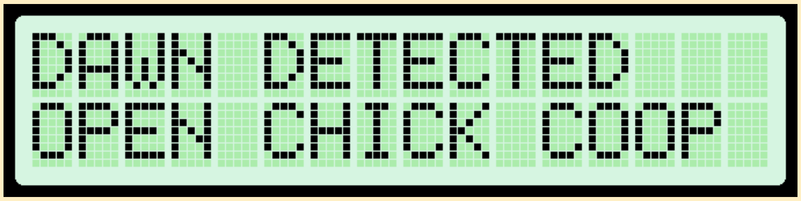
```
PACKAGE DELIVERD
LOCKED WITH CODE
```

Package delivery safe with automatic lock and code to unlock.



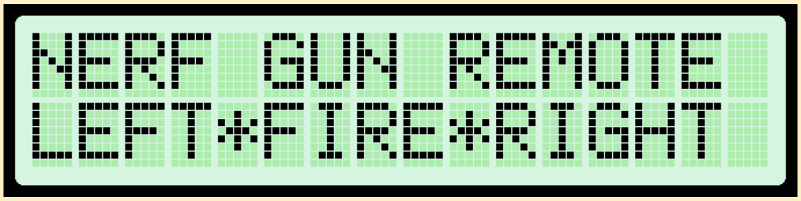
```
ROOSTER CROWED!
OPEN CHICK COOP
```

Open chicken coop when rooster crows (Accessory Board & continuous servo)



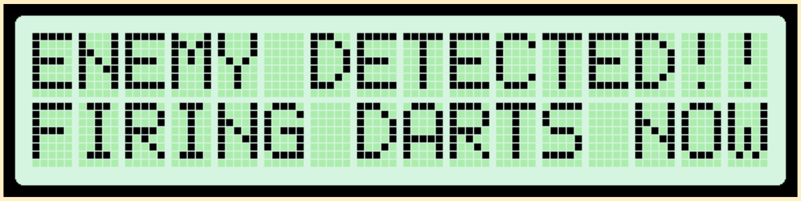
```
DAWN DETECTED
OPEN CHICK COOP
```

Open chicken coop door when light detected (continuous servo & LDR).



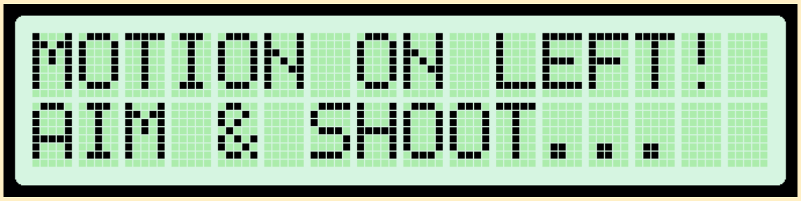
```
NERF GUN REMOTE
LEFT*FIRE*RIGHT
```

Remotely aimed and triggered foam dart gun (two wireless modules).



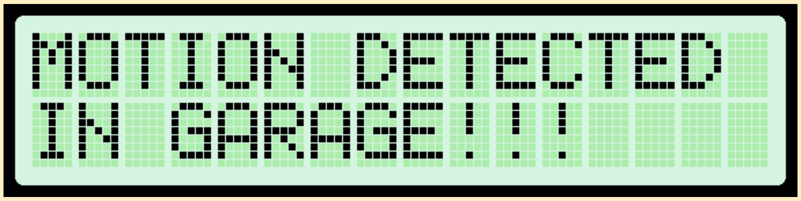
```
ENEMY DETECTED!!
FIRING DARTS NOW
```

Motion detecting foam dart gun. Point foam dart gun at friend when he is near front door. He will go inside house to hide, there he will be shot by motion detecting repeating foam dart gun.



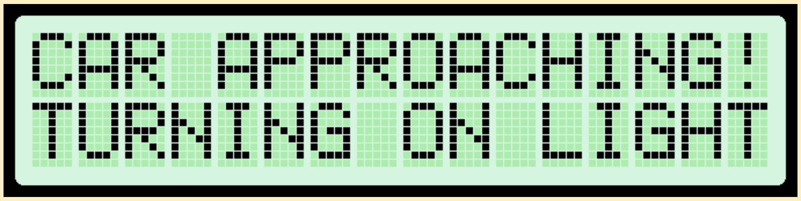
```
MOTION ON LEFT!
AIM & SHOOT...
```

Auto aiming foam dart gun (multiple proximity sensors).



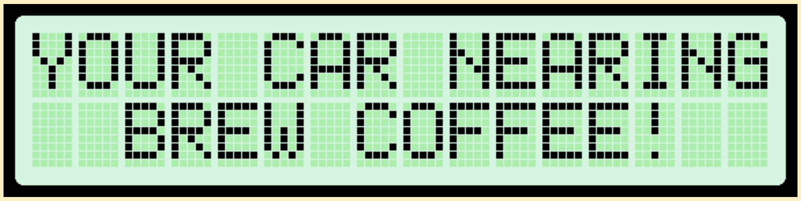
```
MOTION DETECTED
IN GARAGE!!!
```

Wireless burglar alarm (keep kids out of workshop)(alert when customer enters store).



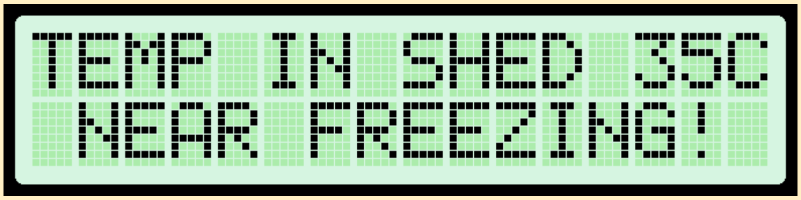
```
CAR APPROACHING!
TURNING ON LIGHT
```

Turn on house/driveway lights when a car approaches.



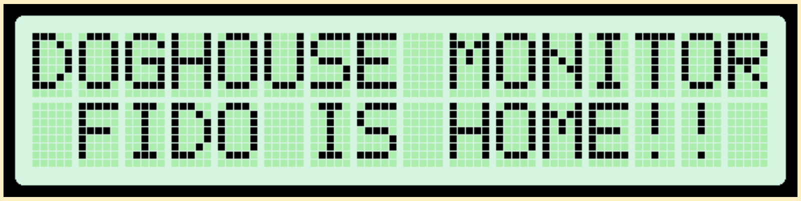
```
YOUR CAR NEARING
  BREW COFFEE!
```

Start brewing coffee by just driving close to your house (Relay Module).



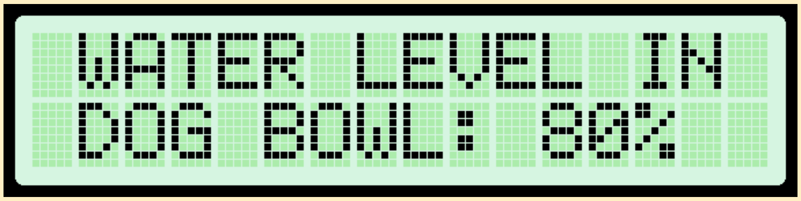
```
TEMP IN SHED 35C
 NEAR FREEZING!
```

Wireless freezing alarm (red, orange, green lights) (temp sensor).



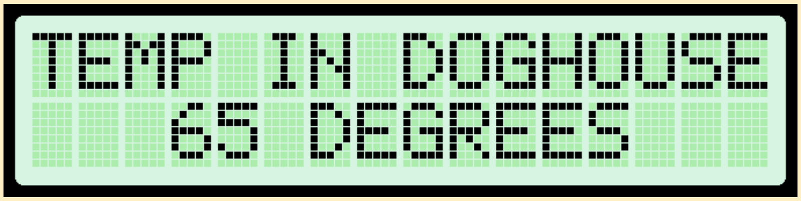
```
DOGHOUSE MONITOR
 FIDO IS HOME!!
```

Detect and wirelessly report if dog/husband is in doghouse.



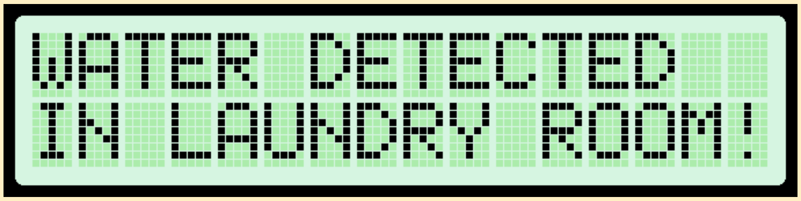
```
 WATER LEVEL IN
 DOG BOWL: 80%
```

Wirelessly monitor level of water in dog’s bowl.



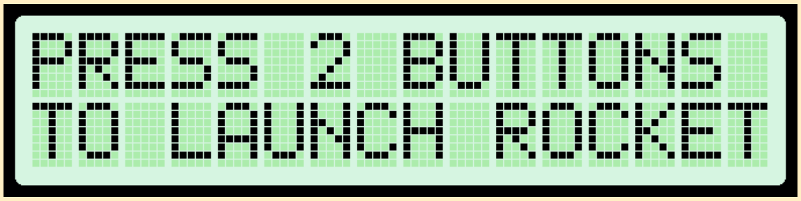
```
TEMP IN DOGHOUSE
   65 DEGREES
```

Wirelessly monitor temperature in dog house (temp sensor).



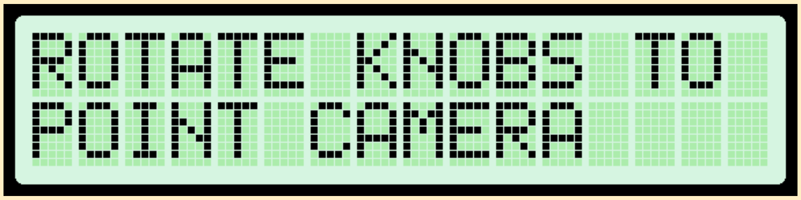
```
WATER DETECTED
IN LAUNDRY ROOM!
```

Wireless flooding alarm.



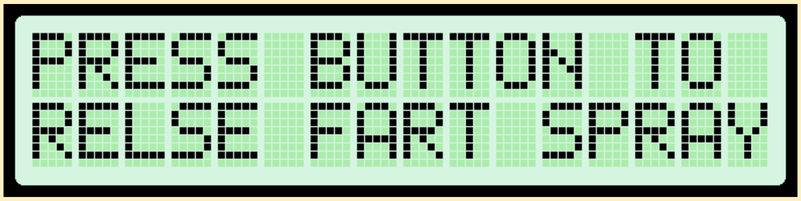
```
PRESS 2 BUTTONS
TO LAUNCH ROCKET
```

Wirelessly ignite rocket (Accessory Board).



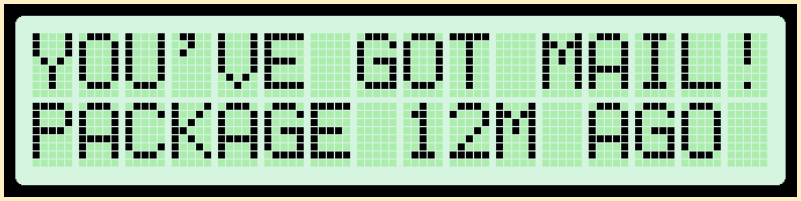
```
ROTATE KNOBS TO
POINT CAMERA
```

Remotely point a spy camera in any direction (servos).



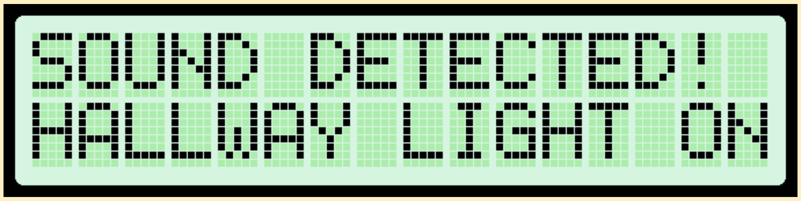
```
PRESS BUTTON TO
RELSE FART SPRAY
```

Practical joke: remote controlled fart spray (with servo).



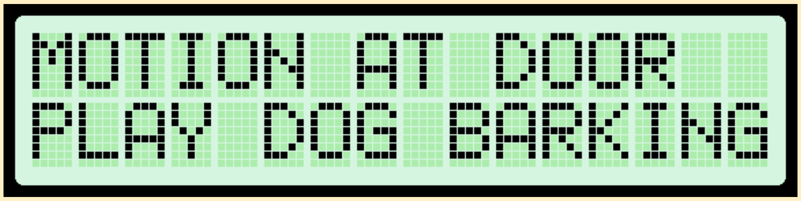
```
YOU’VE GOT MAIL!
PACKAGE 12M AGO
```

Wireless mailbox monitor.



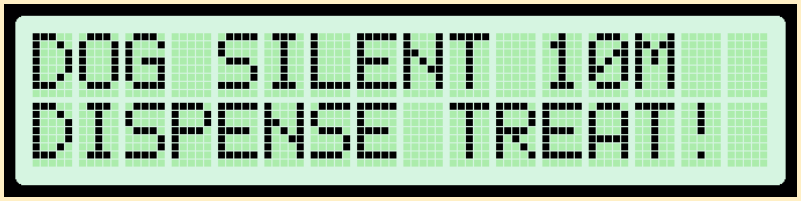
```
SOUND DETECTED!
HALLWAY LIGHT ON
```

Sound activated light switch (burglar deterrent) (servo).



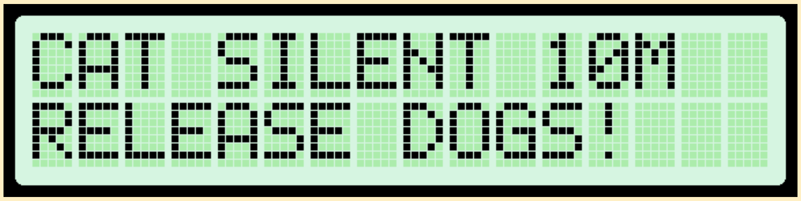
```
MOTION AT DOOR
PLAY DOG BARKING
```

Play barking sound whenever someone rings doorbell to discourage burglar (Infrared LED to activate sound system remotely) (you can also play memorable movie quotes).

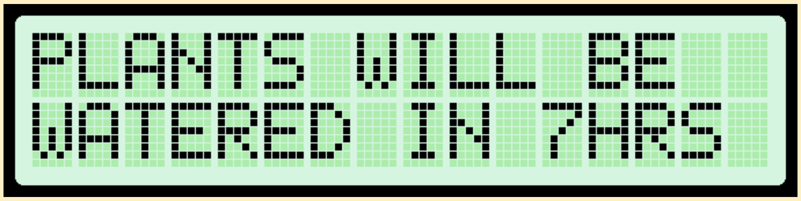


```
DOG SILENT 10M
DISPENSE TREAT!
```

Teach your dog not to bark (buzzer) / servo to dispense treat after silence.

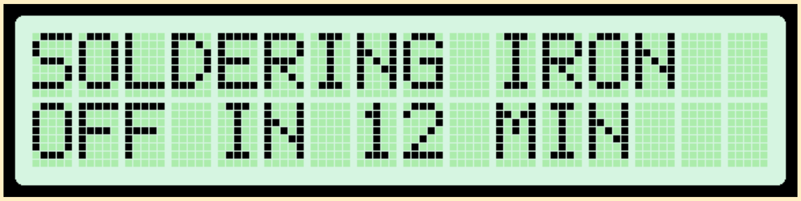


```
CAT SILENT 10M
RELEASE DOGS!
```



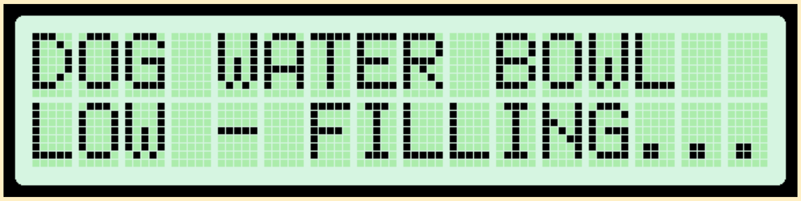
```
PLANTS WILL BE
WATERED IN 7HRS
```

Automatic plant watering (aquarium pump).



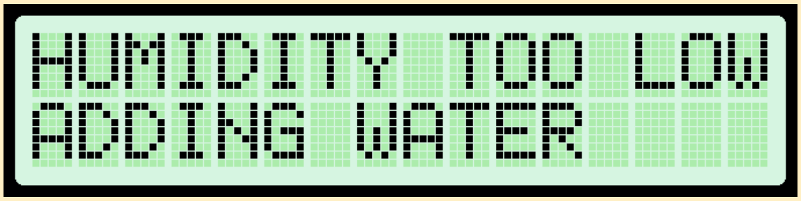
```
SOLDERING IRON
OFF IN 12 MIN
```

Soldering iron/hot glue gun turn off timer (Relay Module).



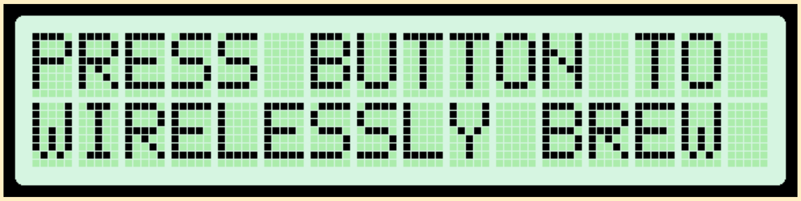
```
DOG WATER BOWL
LOW - FILLING...
```

Automatic pet water bowl top up (from rain barrel?).



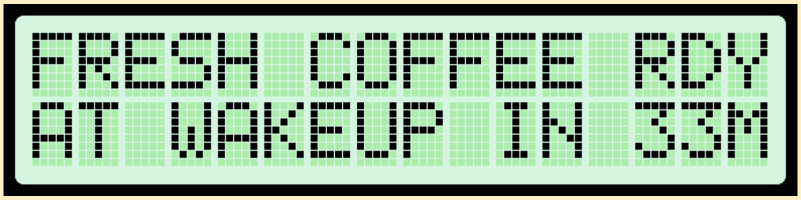
```
HUMIDITY TOO LOW
ADDING WATER
```

Automatic humidity sensor with DIY humidifier (humidity sensor).



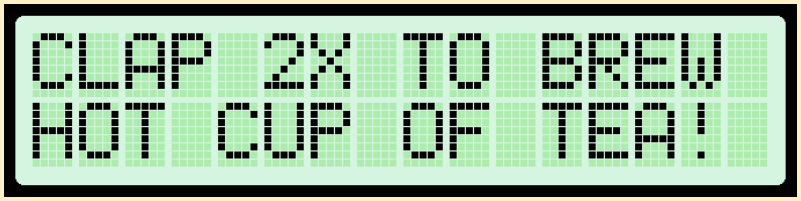
```
PRESS BUTTON TO
WIRELESSLY BREW
```

Wirelessly make a cup of coffee from desk (wireless module).



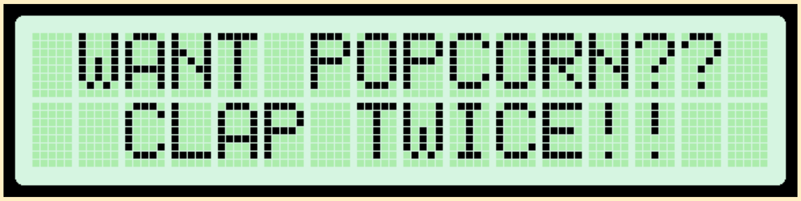
```
FRESH COFFEE RDY
AT WAKEUP IN 33M
```

Wakeup alarm with automatic coffee maker.



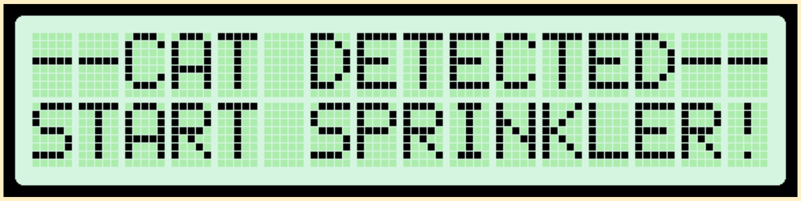
```
CLAP 2X TO BREW
HOT CUP OF TEA!
```

Clap-activated coffee maker (Accessory Board & Relay Module).



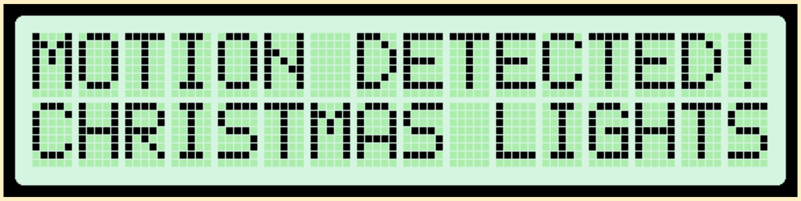
```
 WANT POPCORN??
  CLAP TWICE!!
```

Clap-activated microwave (Accessory Board & servo).



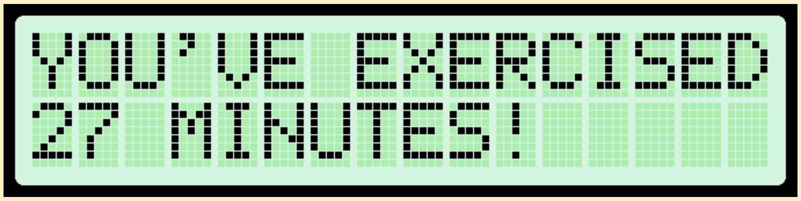
```
--CAT DETECTED--
START SPRINKLER!
```

Cat deterrent from flowerbed (sprinkler).



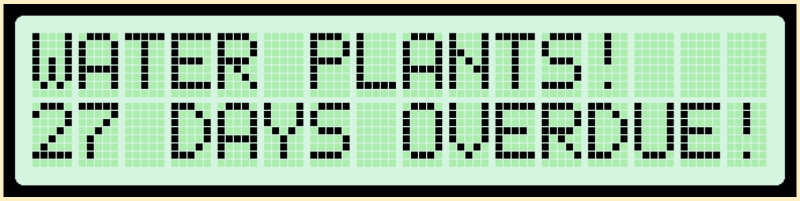
```
MOTION DETECTED!
CHRISTMAS LIGHTS
```

Only turn on Christmas lights when person is present.



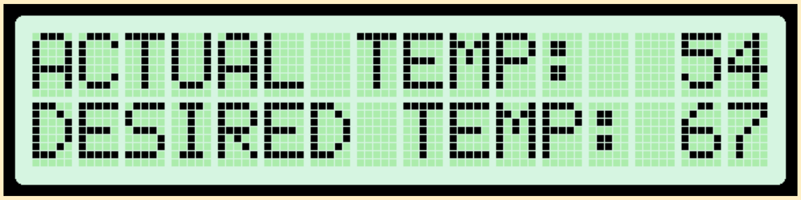
```
YOU'VE EXERCISED
27 MINUTES!
```

Track time spend on computer/exercising (ultrasonic sensor).



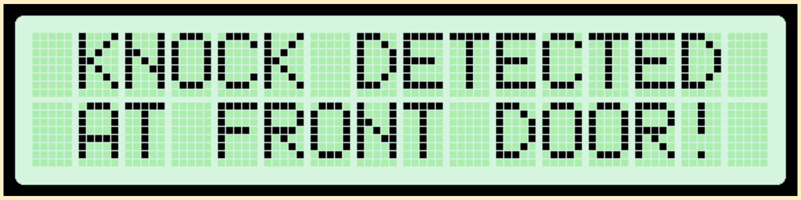
```
WATER PLANTS!
27 DAYS OVERDUE!
```

Reminder to water plants.



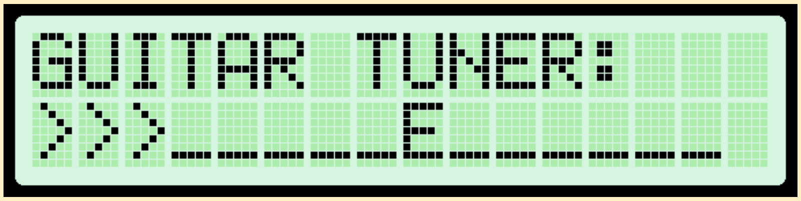
```
ACTUAL TEMP:  54
DESIRED TEMP: 67
```

Automatic heater/fan for doghouse (temp sensor).



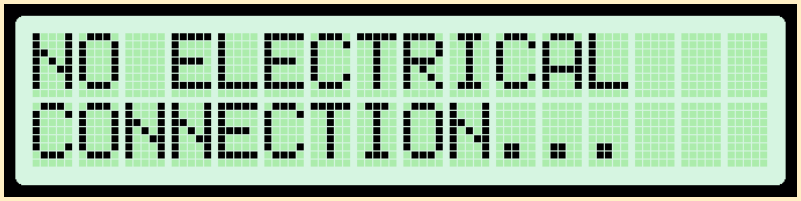
```
 KNOCK DETECTED
 AT FRONT DOOR!
```

Have a knock on your front door be wirelessly transmitted as a doorbell sound to a desktop receiver.



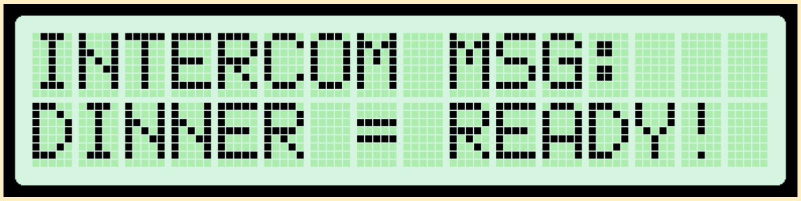
```
GUITAR TUNER:
>>>_____E______
```

Guitar tuner.



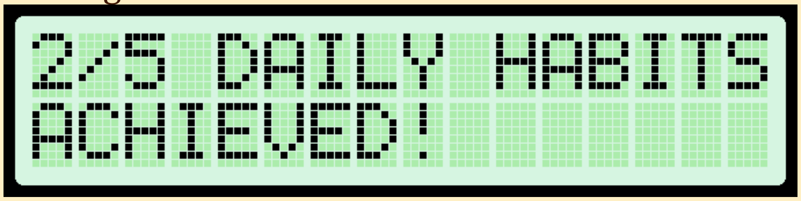
```
NO ELECTRICAL
CONNECTION...
```

Continuity tester with beep (fixing headphones).



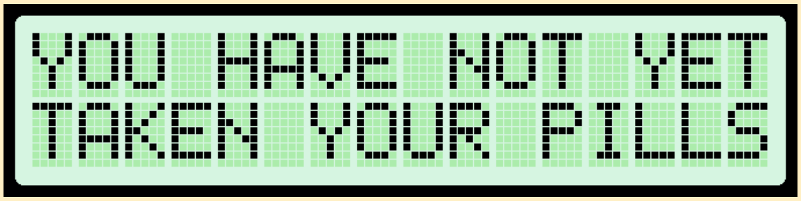
```
INTERCOM MSG:
DINNER = READY!
```

Wireless text-based intercom/dinner call with different tunes for different messages.



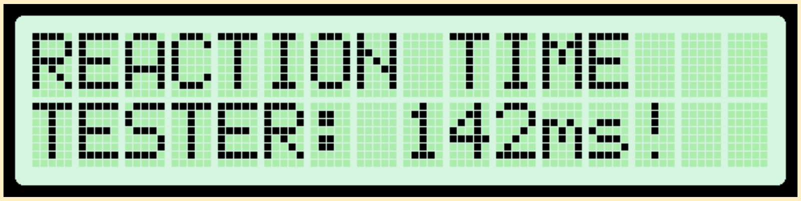
```
2/5 DAILY HABITS
ACHIEVED!
```

Daily habits reminder.



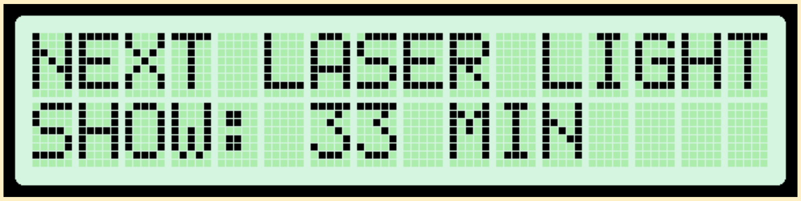
```
YOU HAVE NOT YET
TAKEN YOUR PILLS
```

Medication reminder (refill reminder).



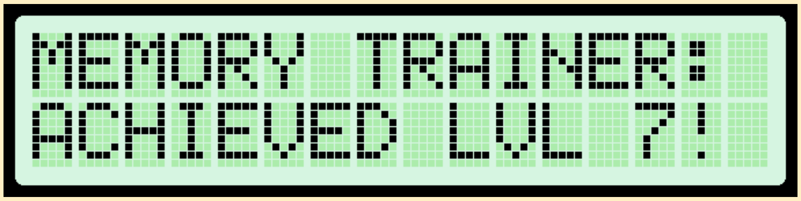
```
REACTION TIME
TESTER: 142ms!
```

Kids project/science lesson: Test your reaction time (press button as quickly as possible after light turns on).



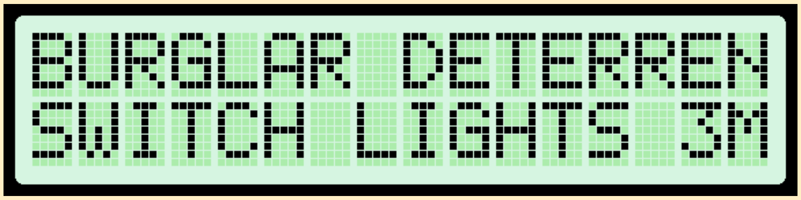
```
NEXT LASER LIGHT
SHOW: 33 MIN
```

Automatically amuse pets with moving laser pointer when you’re not home.



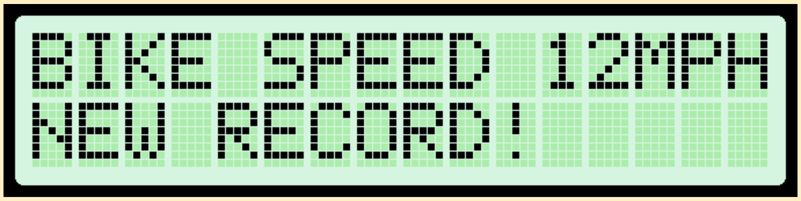
```
MEMORY TRAINER:
ACHIEVED LVL 7!
```

Test memory by repeating a pattern of flashing lights by pressing buttons in the same order (with tune if correct) & score keeping.



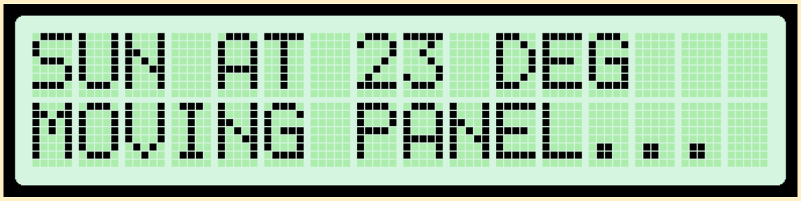
```
BURGLAR DETERREN
SWITCH LIGHTS 3M
```

Randomly turn lights on/off when you are not home to fool burglars. Randomly turn on TV/music to fool burglars (using IR LED as remote).



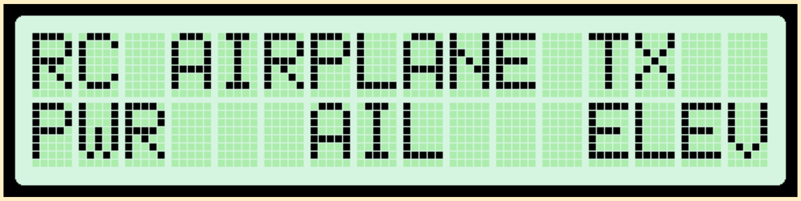
```
BIKE SPEED 12MPH
NEW RECORD!
```

Kids project: Detect speed of bike by measuring travel time between two units (with laser/ultrasonic sensor).



```
SUN AT 23 DEG
MOVING PANEL...
```

Sun tracker for solar panel (light dependent resistors).



```
RC AIRPLANE TX
PWR   AIL   ELEV
```

Proof of concept project: Turn the EZA Board into a remote controlled airplane. This could be done using the wireless modules and Accessory Board. The Accessory Board can control a DC motor attached to propeller and also control two servos. This board can power both the DC motor and the EZA Board from a single 3S LiPo battery. You don't need any other electronics. The second EZA Board will act as the remote.

- Kids: Who can jump over the laser beam (with laser pointer and light sensor) (with sound effects).
- Parking sensor.
- Alibi generator
- Encode a message and send it to a friend who can decode it with his EZA Board.

- Take time-lapse photo in sync with 3D printer (same location each photo).
- Control any light with a TV remote (inexpensive IR detector).
- Practical joke: Turn off light switch as soon as someone turns it on (light dependent resistor).
- Have autonomous car deliver mail from mailbox to house by following laser beam (light dependent resistors).
- Shooting target with automatic reset & score keeping.
- Automatically turn on lights when it gets dark and turn off when it gets light (light dependent resistor).
- Wirelessly ignite campfire by waving your hand (Accessory Board).
- Practical joke: remotely break glass when friend is only person in the room and then blame him (possibly using fishing wire).
- Motion activated lights (use painters tape and hot glue to glue servo to light switch).
- Music player activated by motion.
- Wireless doorbell range extender.
- Clap-activated light switch (servo).
- Locked cupboard that only opens with secret knock-code (servo).
- Automatically send and decode Morse code.
- Science lesson: Measure the speed of sound (two wireless boards & Accessory Boards required).
- Remote controlled boat (with continuous rotating servos acting as paddles, like the old paddle boats).
- Practical joke: annoy friend by randomly changing channel to news station of opposite political party and increasing volume (infrared LED).
- Measure speed of bullet (shoot through two sensors made of aluminium foil).
- Disco lights that light up in sync with the music (Accessory Board).
- Metronome (for practicing a musical instrument)

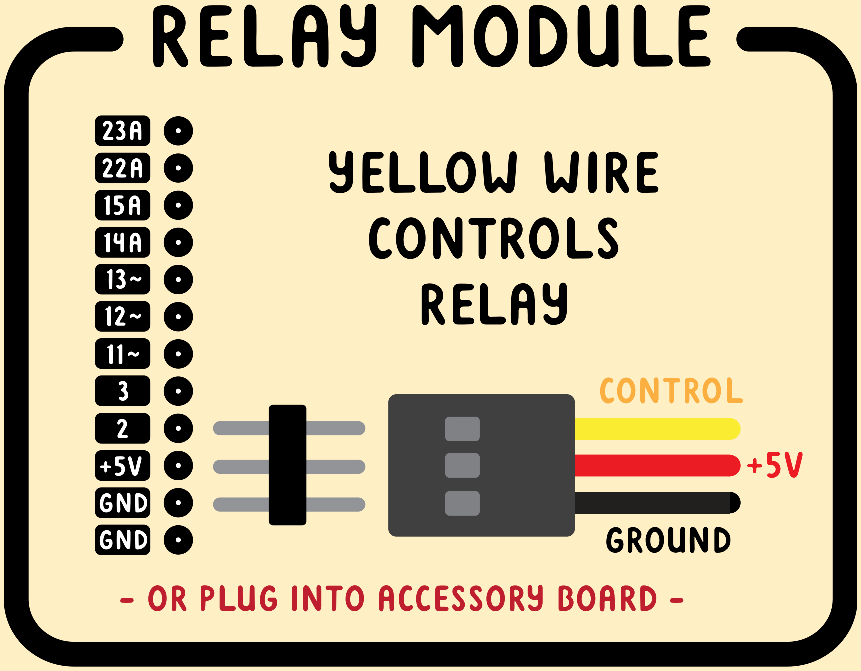
OPTIONAL ACCESSORY: RELAY MODULE

The Relay Module allows you to switch large mains-powered devices. A relay is a mechanical switch that is controlled by an electromagnet. Sending 5V to this magnet will turn on the mains power to the output socket.

The Relay Module allows you to switch devices that consume up to 15 amps. That is up to 1600 watt if you are on a 120 Volt system (USA), or up to 3300 watt if you are on a 220 V system (Europe). Do not try to exceed this rating.

The three control wires of the Relay Module are for ground (black), 5 volt (red) and control (yellow). Sending 5 volt to the yellow wire will turn the relay on. As soon as you stop supplying 5V to this wire the relay will turn off. You cannot damage the Relay Module by connecting its control wires incorrectly.

You can plug the Relay Module directly into the EZA Board using a simple straight pin connector (included) as shown. Even easier is using the three-pin connectors of the Accessory Board. You can connect up to 4 Relay Modules this way.



The yellow LED indicates that the module is receiving 5V power (not 5V control signal). This should be on whenever the module is connected to the EZA Board. The red LED indicates that the control wire is receiving 5V. Whenever the red LED is on the relay should be on as well.

~ If the yellow LED does not turn on as soon as you plug in the module you have likely connected it in reverse! Don’t worry, this will not cause damage.

You might be tempted to try to dim a device connected to a relay using PWM, just as you can with on-board LEDs or the 12-volt circuits on the Accessory Board. This is not possible. Since a relay is a physical mechanism its switching speed is far too slow to respond to PWM.

~ Relays have a lifetime rating of openings and closings. The relay used in this module can endure 100000 cycles at full power. Avoid unnecessarily switching the relay.

Mains power is deadly. Keep the Relay Module away from water and do not attempt to open it.

OPTIONAL ACCESSORY: ACCESSORY BOARD

The pins of the ATMEGA4809 chip on your EZA Board can *not* directly drive power-hungry devices such as motors, LED strips, 12-volt devices or relays. To drive such devices you should use the Accessory Board. It plugs right into the EZA Board and requires no soldering. The Accessory Board has a number of extra abilities:

1. Controlling up to 4 motors or other devices (up to 14 volt).
2. Controlling up to three servos.
3. Measure sound levels.
4. Accurately keep track of time.
5. Powering the EZA Board from a battery.
6. Connect wires to the EZA Board's user-accessible pins in a more reliable way (using the solder points).
7. Controlling up to 4 Relay Modules.

1A. Controlling motors

You can use the Accessory Board to control up to 4 DC motors. (DC motors are the kind of motor with two wires.) You can vary their speed using PWM (analogWrite). Connect the motor to the screw terminals labeled 11, 12, or 13 for variable speed or terminals 14 for on/off control. Whatever voltage you are feeding into the input (labeled 7-14V) will be sent to the appropriate output terminals (11 - 14) when you write the corresponding pin HIGH.

~ Bidirectional control of DC motors is not possible with the Accessory Board.
~ The maximum theoretical current draw is 4 amps per channel.

1B. Controlling devices up to 14 volt

Some categories of devices that are under 14 volt:

- Most LED strips.
- Automobile or camping electronics such as lights, fans or pumps (all 12V devices).
- Almost anything that is battery powered such as laser pointers, electric

foam dart guns, or toys.
- Relays.

Whatever voltage you are feeding into the input (labeled 7-14V) will be sent to the appropriate output terminals (11 - 14) when you write the corresponding pin HIGH. Make sure the pin you are using is set to be an output. You can vary the power going to the device by using PWM (only pins 11-13). Make sure not to send this voltage to any part of the EZA Board. The EZA board is only tolerant of voltages up to 5V.

~ The maximum theoretical current draw is 4 amps per channel.
~ Pins 11-13 are dual purpose; they can be used to control a servo or power devices up to 14 volt. You cannot both control a servo *and* a motor/device on the *same pin* at the same time.

2. Controlling servos
You can control a servo by plugging it directly into the EZA Board's breadboard. This is not ideal, however. Servos can draw a lot of power and generate electrical noise. This can affect the EZA Board's 5V power supply.

You can avoid both these issues using the Accessory Board. It can easily control up to three servos using pins 11, 12 and 13. See section 5 for hints on supplying power to your servos.

~ Pins 11-13 are dual purpose; they can be used to control a servo or power devices up to 14 volt. You cannot both control a servo *and* a motor/device on the *same pin* at the same time.
~ Plugging a servo in backwards does no damage.

3. Measure sound levels
The Accessory Board includes a microphone and preamplifier. This can be used to detect claps, knocks, barking or gunshots. You can measure the sound level on pin 23 using analogWrite.
~ See the example program for details.

4. Accurately keep track of time
The EZA Board has no way to access absolute time (Wednesday 3:14PM). It can only measure elapsed time since startup (4503 milliseconds) and that method might not be particularly accurate. With the Accessory Board you get a very accurate clock that keeps track of the absolute time and date. See the example program for details. The CR2032 battery is installed with the positive pole facing up.

5. Providing 5V to the EZA Board from a battery or your external power supply
If you are powering your Accessory Board with a voltage ranging from 7 to 14 Volt the Accessory Board can regulate this voltage down to 5V required to power the EZA Board. Therefore a single 12V battery or power supply can power both the Accessory Board *and* EZA Board.

If you don't supply 7 – 14V to the Accessory Board the EZA Board can also feed the Accessory Board (with some limitations):

EZA Powered through USB	**Accessory Board powered by 7-14V**	Can I control devices from 7-14V?	Can I control servos?
Yes	**No**	No	May cause problems
No	**Yes**	Yes	Yes (but not ideal)
Yes	**Yes**	Yes	Yes! (remove solder bridge)

~ For either board to provide power to the other the solder bridge needs to be present (default).

~ The solder bridge can be removed with a soldering iron to prevent electrical noise from the servos from interfering with the EZA Board. Both boards will need to be individually powered. Only required when using servos.

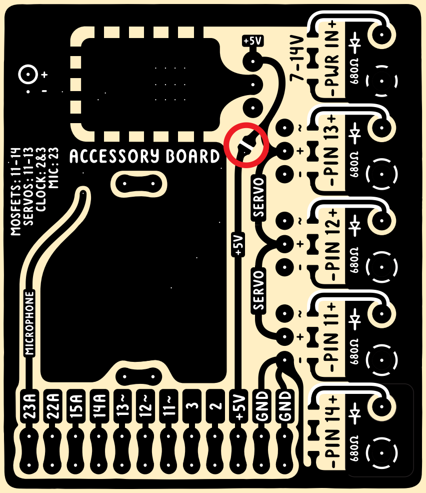
~ The component responsible for the voltage reduction from 7-14V to 5V is the regulator. It may get hot during operation. You are not likely to damage this quality regulator; it has both thermal protection and overcurrent protection. If the regulator gets too hot reduce power consumption (turn off backlight for example) or reduce the input voltage (provide a voltage closer to 7V than to 14V). The regulator is more efficient at lower voltages.

~ Your power supply should be between 7V and 14V, depending on what your device needs. You can go lower than 7 or even 6V but the Accessory Board will no longer provide 5V to the EZA Board.

What power sources provide 7-14 Volt?
Several power sources can provide a voltage in this range. A 12-volt source is especially common. One easy option you likely already have is an AC-DC adapter, also known as a “wall wart” or “power brick”. These have their voltage printed on the case somewhere.

Another good option is a battery pack that takes AA or AAA batteries. You can find these online when you search for “battery holder”. AA or AAA batteries are rated at 1.5V each, so a battery holder with four batteries will provide 6V.

You could also consider using automotive batteries or 2 or 3 cell LiPo batteries, although these have no short-circuit protection and can be dangerous when used incorrectly. These batteries contain a lot of energy and you could potentially start a fire. The cigarette lighter on your car is fused and can be safely used to power the board. When using the Accessory Board with these batteries remember that the EZA Board is not a battery management device and will discharge the battery completely if left on indefinitely.



~ If all you need is 5V for the EZA Board you could use a portable USB power bank. It will shut itself off once the voltage gets too low in order to prevent damage to its internal battery.

Connecting to the Accessory Board can be done using the screw terminals on the side labeled “7-14V IN”. Your ground wire (usually black) will go on the side with the minus sign. The positive wire (often red) will go on the side with the plus sign.

~ Connecting power in reverse will not damage your board.

~ Always turn off or disconnect the power source at the other end before you connect/disconnect wires to the Accessory Board.

6. Connect wires to the EZA Board’s user-accessible pins in a more reliable way (using the solder points).

Sticking wires into the header or breadboard is not the most reliable way to connect electronics. You can solve this problem by soldering to the solder points on the Accessory Board.

~ Even though all pins except 15 and 22 are connected to various electronics on the Accessory Board you can still use many of them for other purposes. If you are not using anything attached to pins 11-14 you can still solder to these pins and use them for other things. The circuitry already connected to these pins is designed to have a minimal impact on any signal present on them.

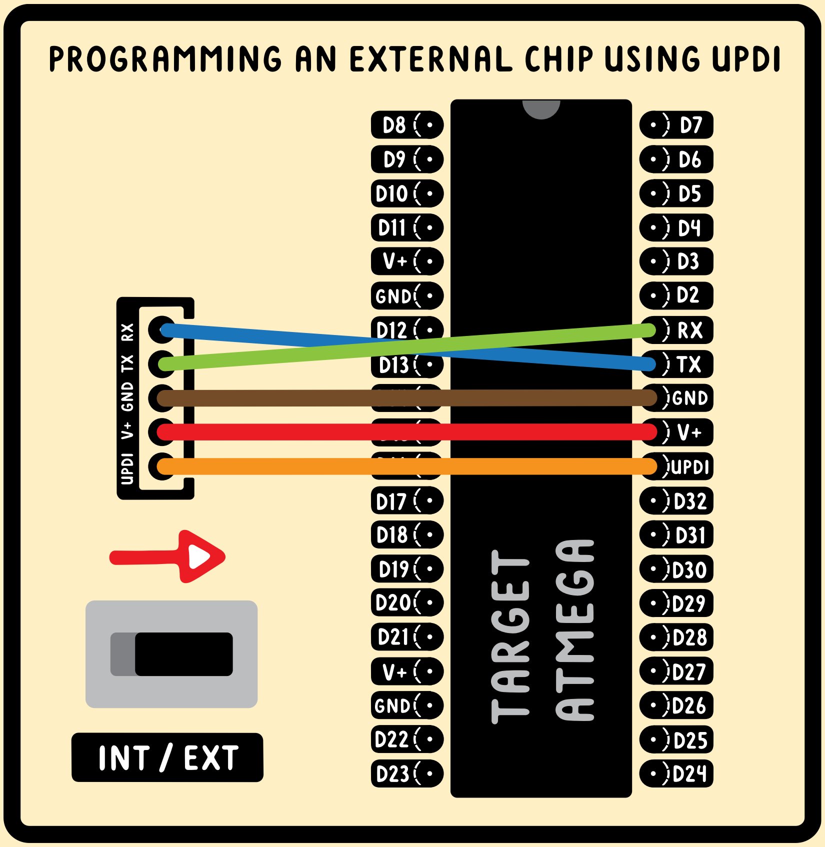
PROGRAMMING EXTERNAL BOARDS

As you get more advanced in your electronics journey you might want to make customized boards running Arduino code, for example a bare-bones compact board. You can use your EZA Board to program any of the AVR-0 chips using the new UPDI interface. You can also read/write serial data to external boards. In order to program external boards move the INT/EXT selector to EXT. In the Arduino software select the settings for the external chip you wish to program, not the settings for the EZA Board.

If your external board is 5V you can let the EZA Board provide power by soldering the solder bridge on the back of the board. This can however ruin an external board running at 3.3V. You can avoid a possible disaster by leaving the solder bridge open and simply providing independent power to your external board.

~ You cannot use one EZA Board to program another EZA Board.

~ The connector used for programming external chips is known as JST XH 5 pin connector with 2.54mm pitch. You might find it advertised as a balance cable for 4S1P LiPo batteries.


PROGRAMMING AN EXTERNAL CHIP USING UPDI

ASSEMBLING THE EZA BOARD CASE

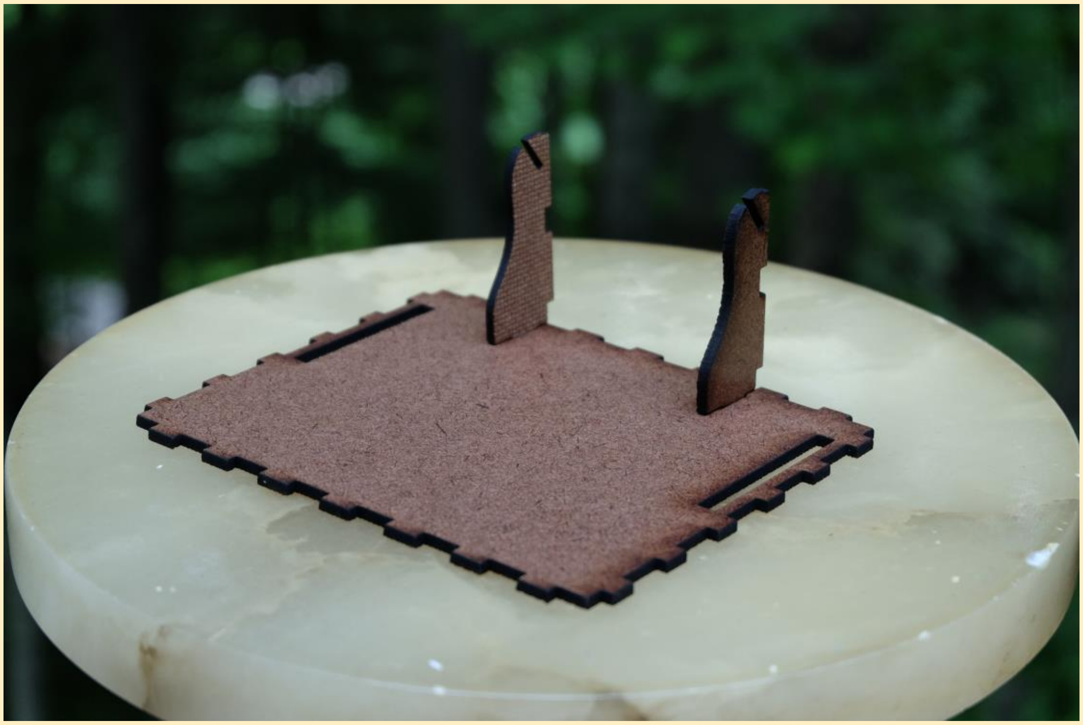
Here are the steps to assemble the EZA Board case. The hardboard has a smooth side and a rough side. When orientation is important it is mentioned.

Occasionally the hardboard edges might need to be beveled using sandpaper to slide in easier.

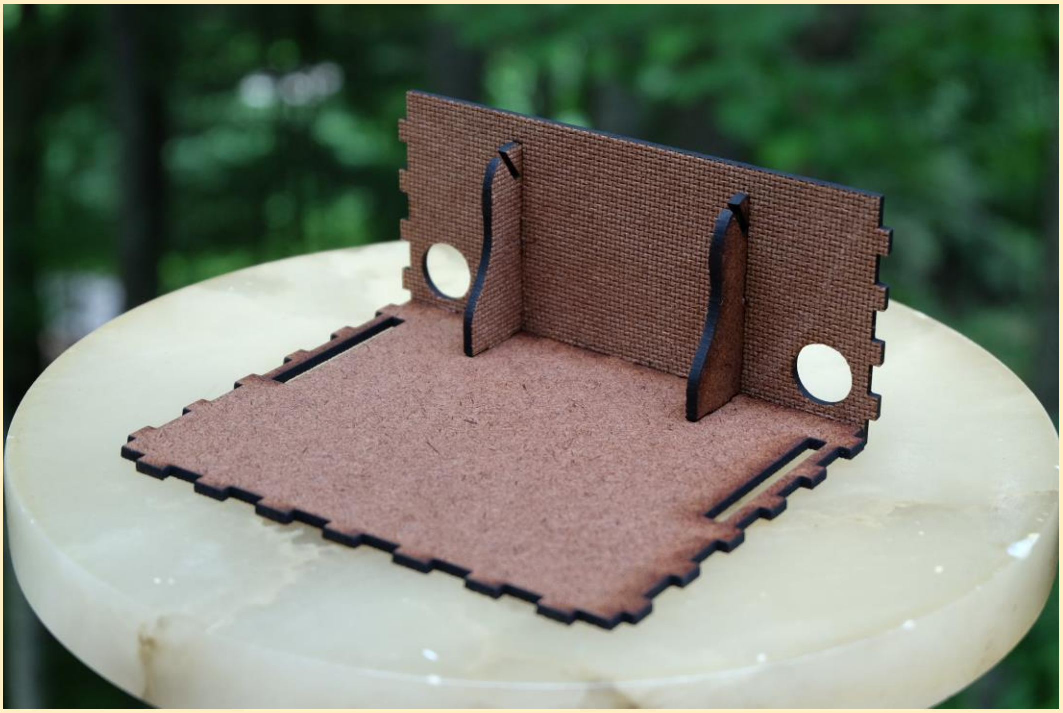
Glue can be omitted. If you choose to glue you could use wood glue, but the best option is contact cement.

~ Cover your work surface because the glue will go everywhere!
~ First dry fit all joints!
~ Disassemble and apply glue.
~ Apply glue and assemble immediately, don't wait for contact cement to become tacky.
~ Wipe off excess contact cement on hardboard. Thin layer of excess contact cement will be absorbed and become nearly invisible.
~ Use rubber bands to keep case under pressure until glue sets.

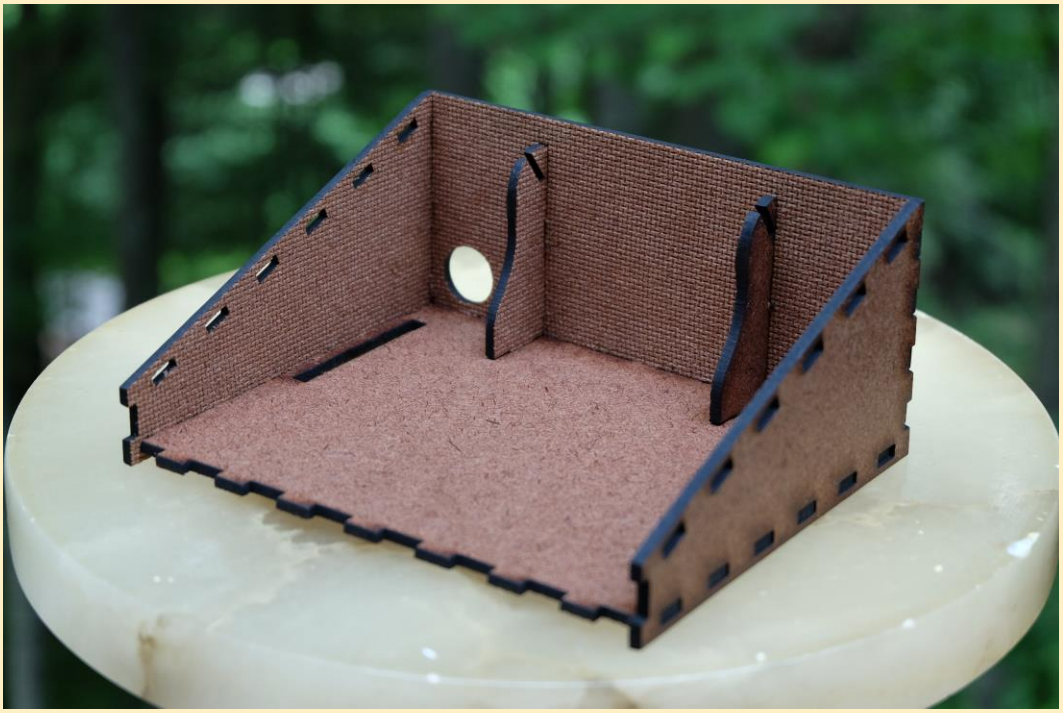
Step 1: Connect the backlight supports to the bottom.



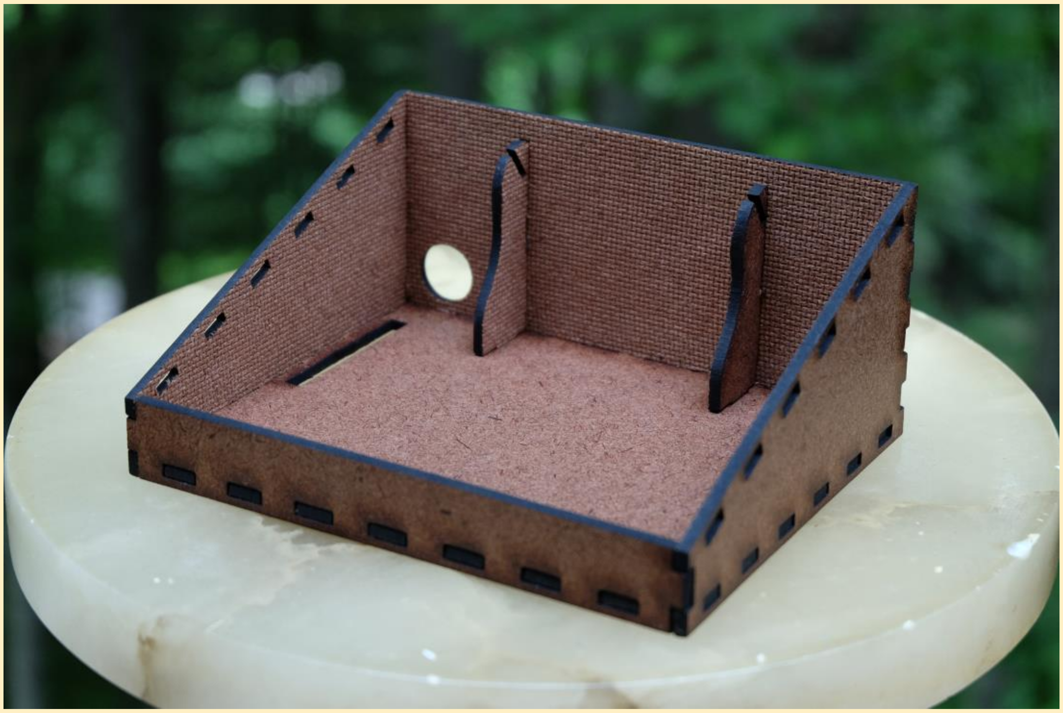
Step 2: Connect the (smooth side facing out).



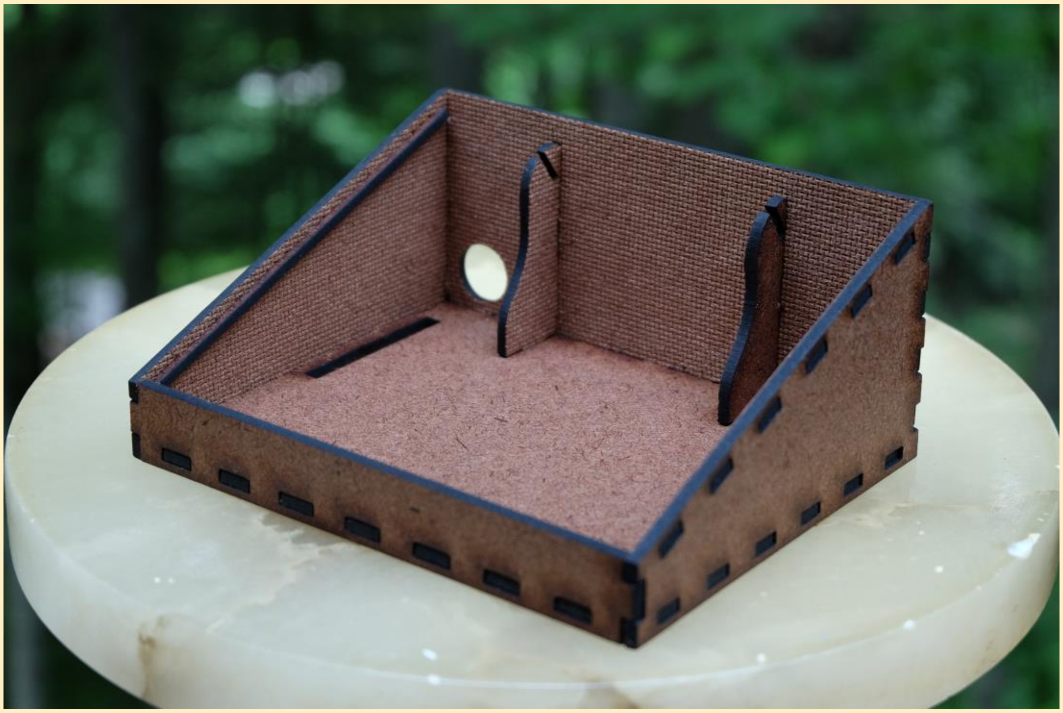
Step 3: Connect the sides (smooth sides facing out).



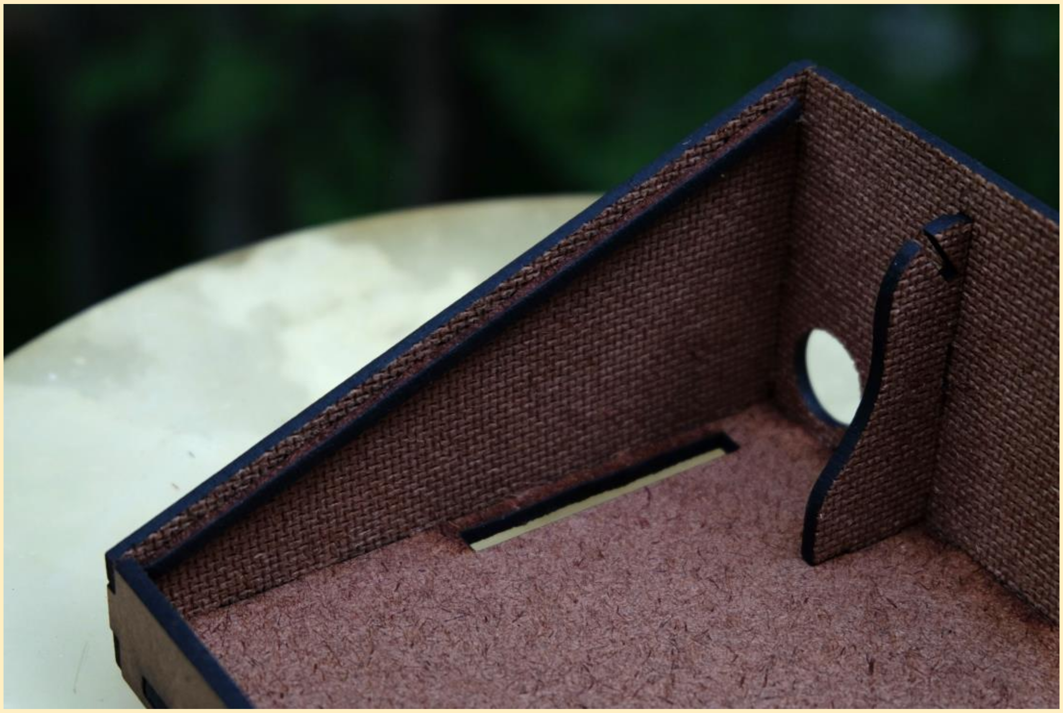
Step 4: Connect the front (smooth side facing out).



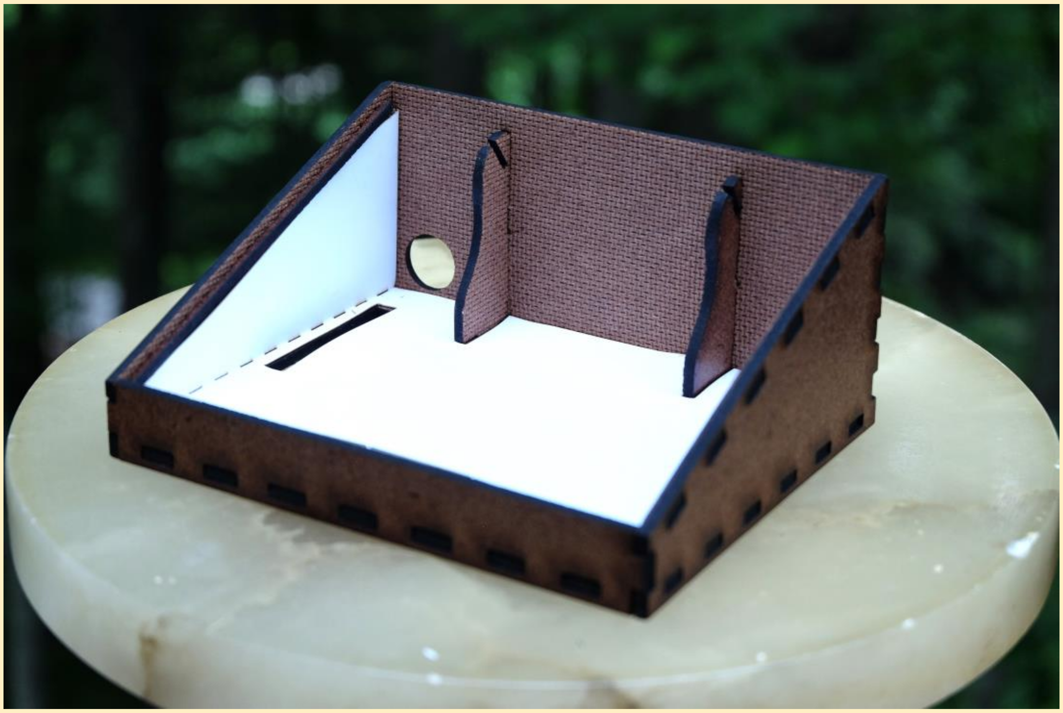
Step 5: Connect the board support ledges.



Detail view of support ledge.



Step 6: Insert paper reflector.



Step 7: Insert backlight board into cutouts.