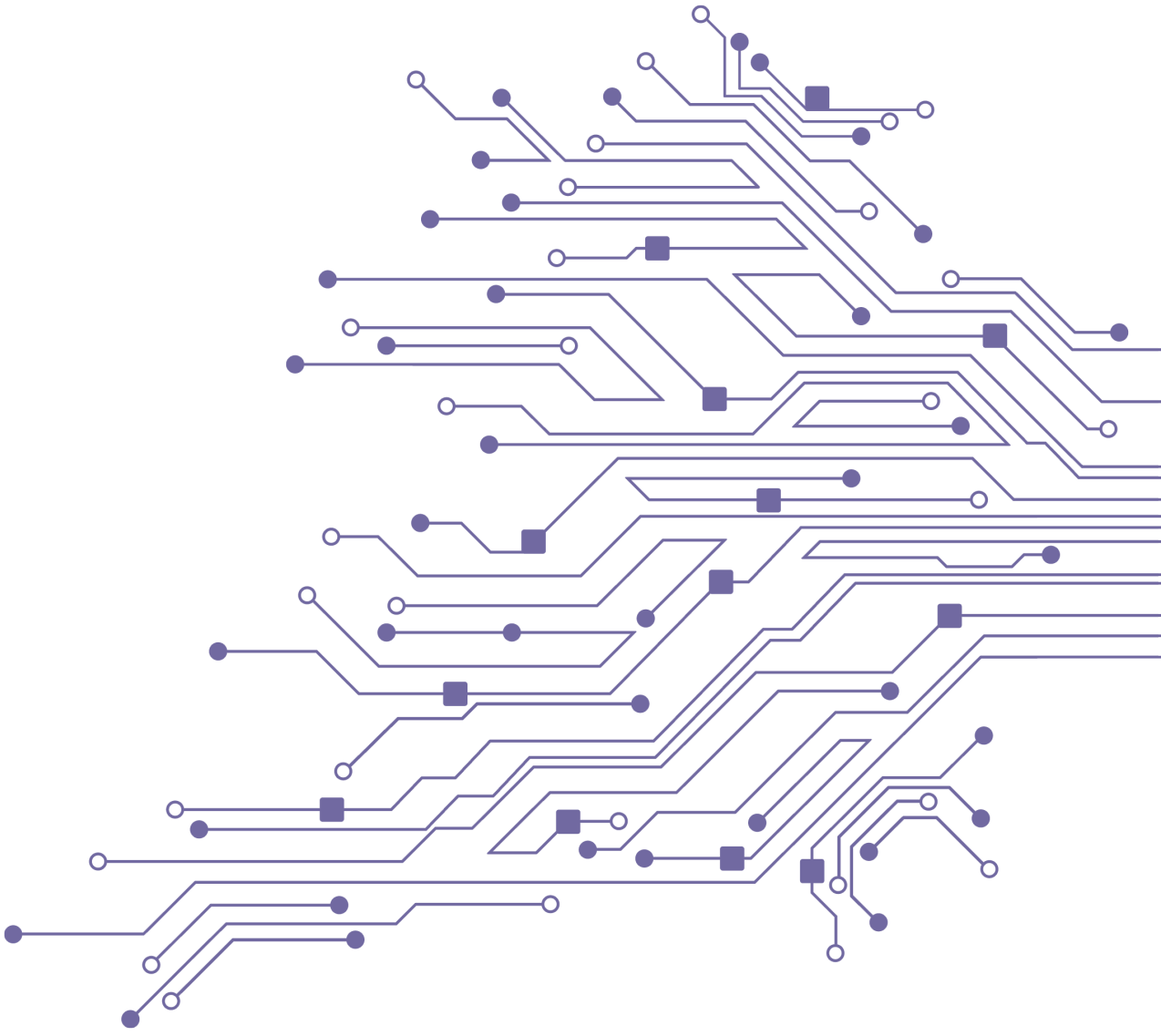
YQM.A.R.S
Mechatronics and Robotics Society

Arduino and ESP32 Sensors Workshop

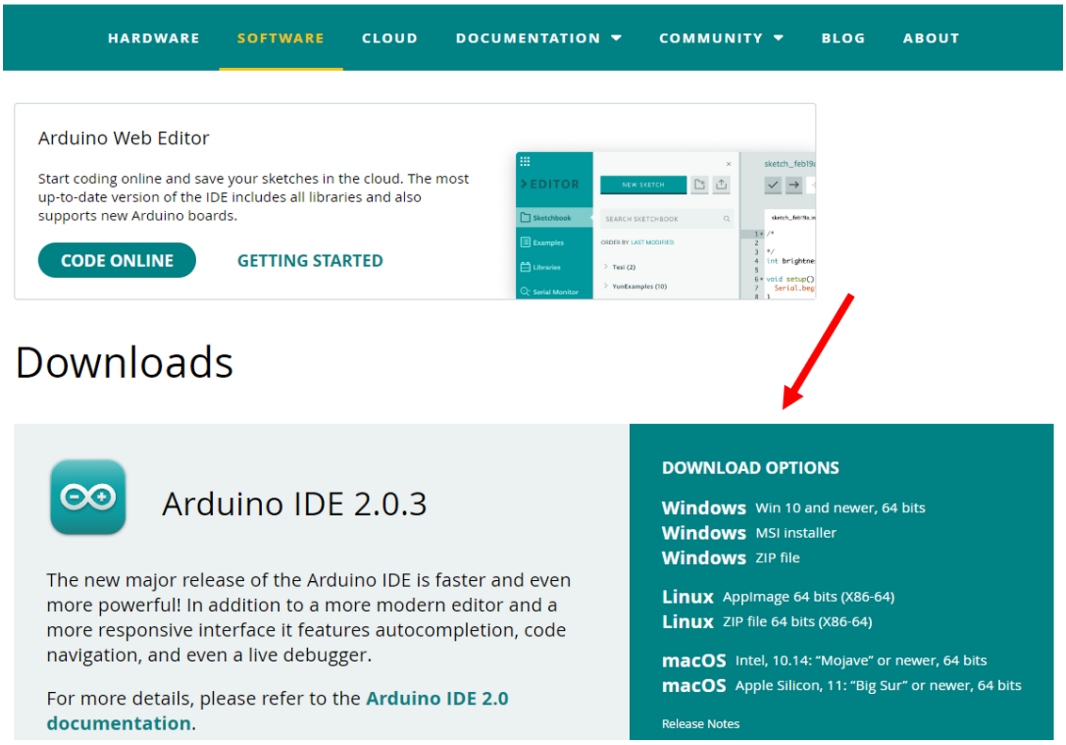
2023



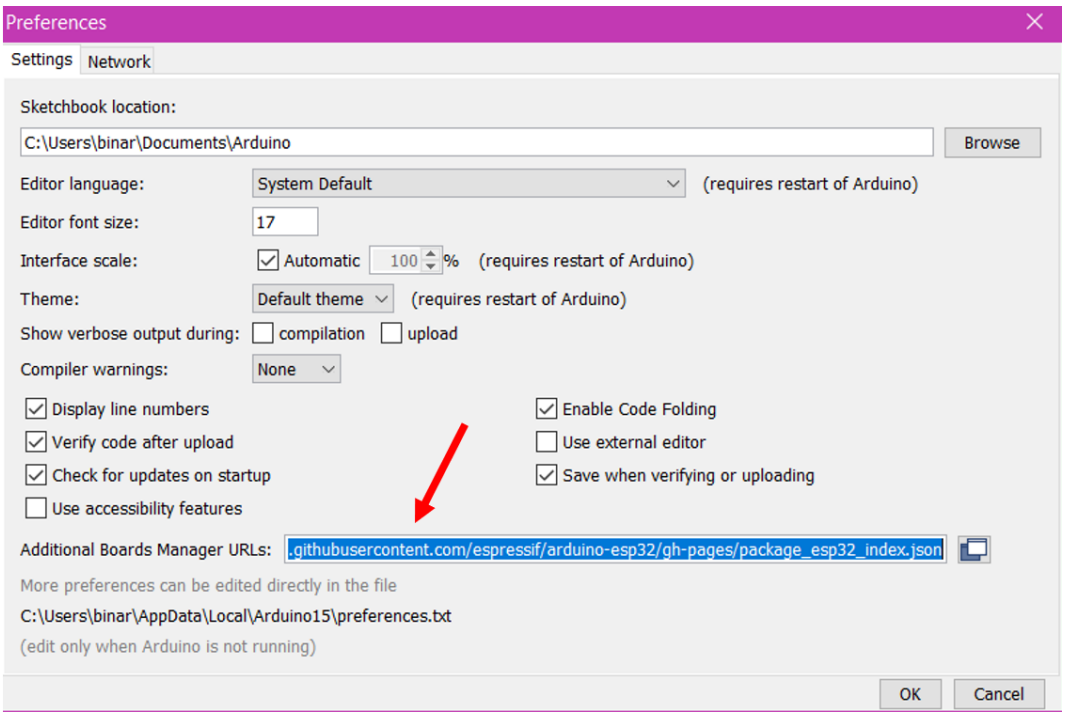
Required Software

The required software for these tutorials is Arduino IDE. To download the software, go through the following steps:

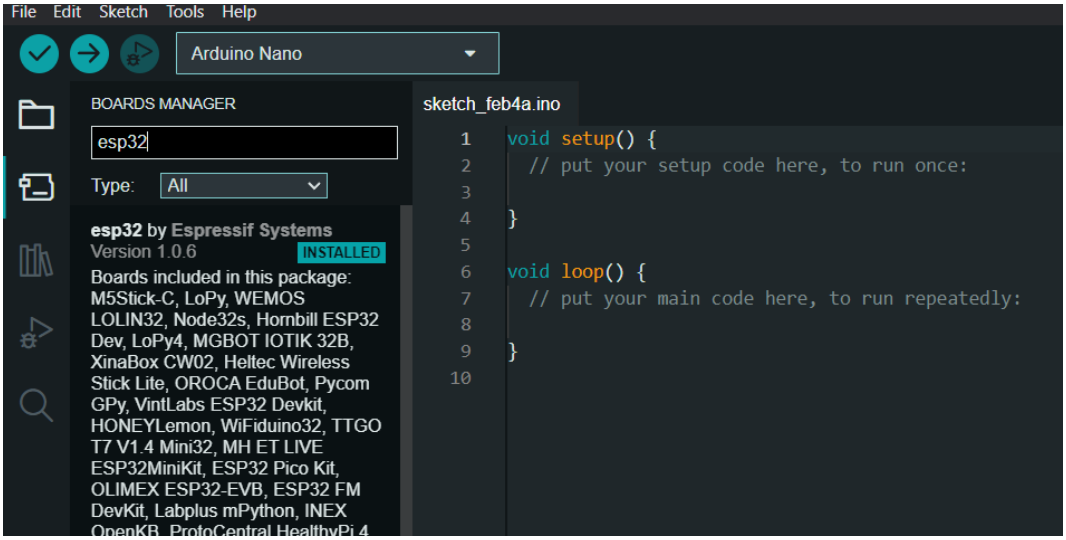
1. Head to the software page on the Arduino website and click on the appropriate installer. Follow the prompts and install on your computer



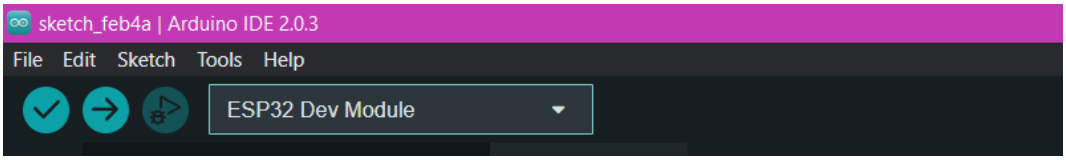
2. Next, open the IDE and head to **File** > **Preferences**, and enter the following URL into the 'Additional boards manager URLs': **https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json**



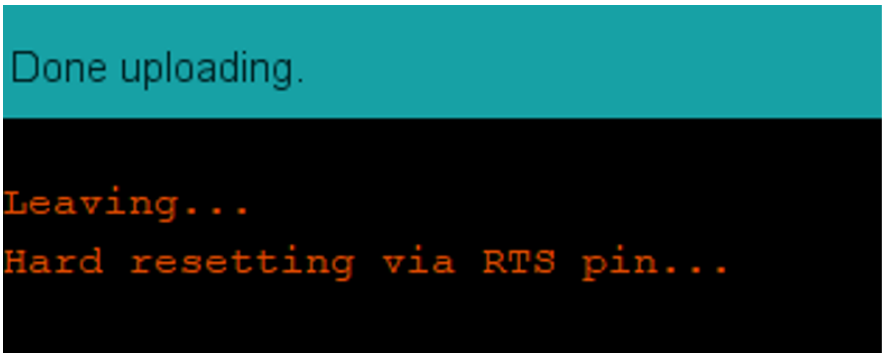
3. Go back to the IDE and head to **Tools** > **Board** > **Boards Manager**. In the old IDE, a new pop-up should appear, and in IDE 2, the toolbox should appear on the left hands side as shown below:



4. To upload code to a board, plug it into the computer and make sure the board is chosen and the port specified (**Tools** > **Port** > **'Select correct port'**). Then, hit the right-facing arrow to upload. Remember to hold down the boot button on the ESP32 while uploading.

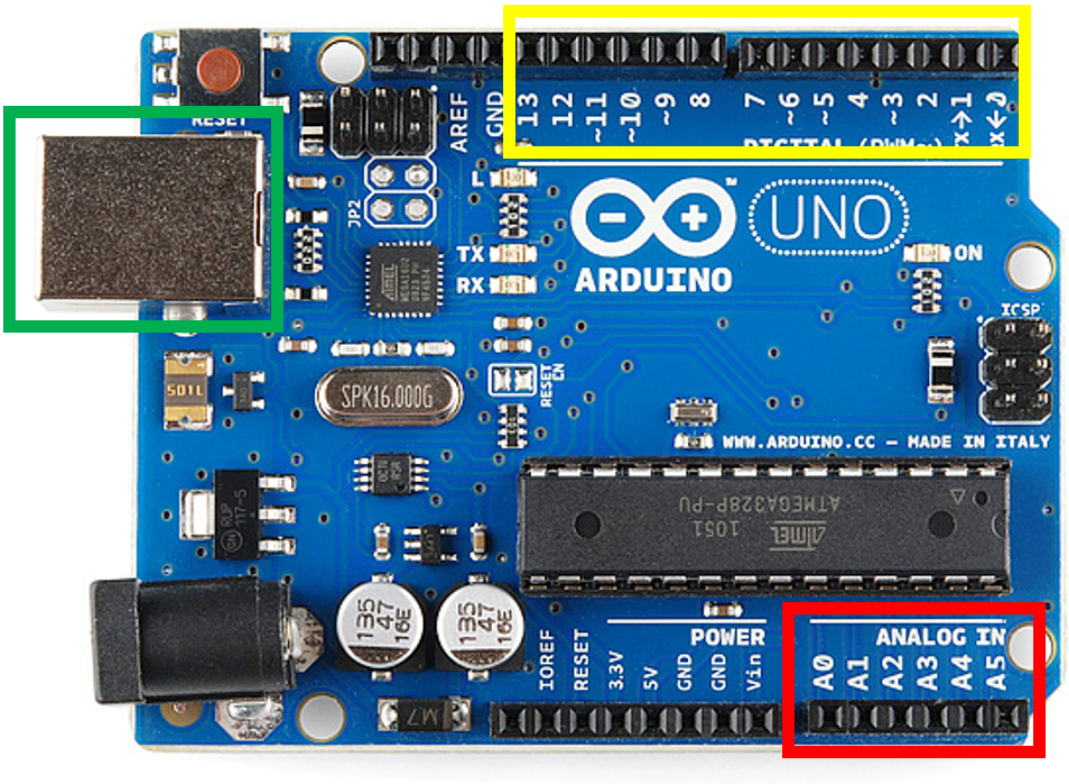


For the ESP32 uploading, once the message below is seen, release the boot button and press the reset button.

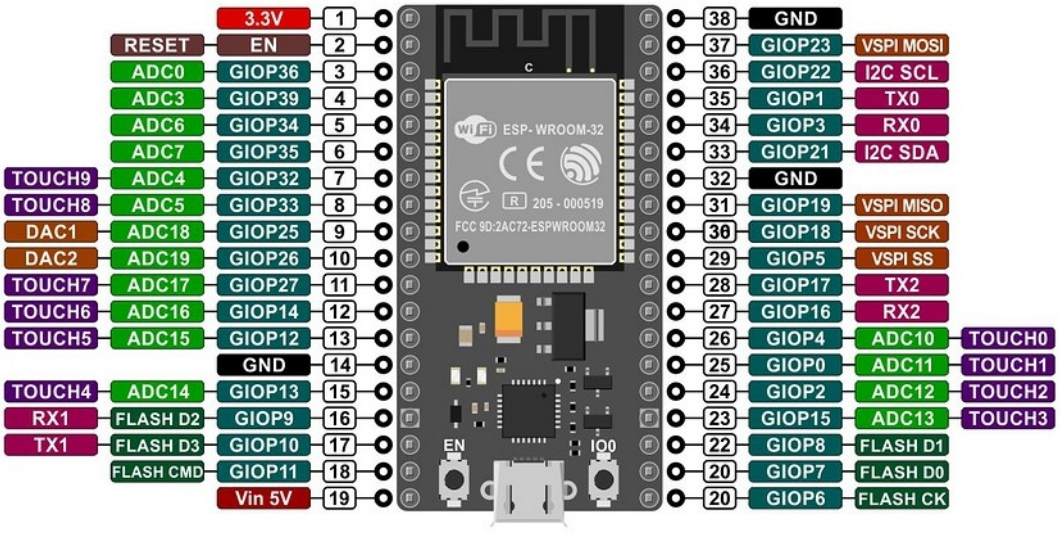


Overview of the Arduino and ESP32

Arduinos and ESPs are types of microcontrollers that can control simple to complex electronics. At a basic fundamental level, they are able to control other electrical devices, and read incoming data. In this tutorial, purple boxes indicate new knowledge on component or code.



- Indicated above in green, is the data cable connection for uploading codes from the computer. Connecting this also provides power to the board
- Indicated in yellow, are the digital ports. These ports are able to interpret and send signals in binary. That is, they sense or output either ON or OFF
  - Note that the ports with a tilde (~) are able to give pwm outputs.
- Indicated in red are the analog ports. These ports are able to interpret signals in a range. For example, they can read a range of 0 - 255.
  - Note that analog ports may also be used as digital ports but not vice versa. Also, analog ports may only read in analog, they cannot write in pwm
- GND - These ports are used as a negative terminal
- 5V/3.3V - These are the voltage supply ports



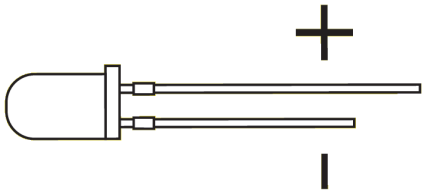
- The pins in green are the GPIO pins which can serve as inputs and outputs unless otherwise signed
- Ports labelled ADC can be used to read analog signals
- Ports labelled TOUCH have internal capacitive touch sensors. For example, these pins can be used as touch sensors using just a finger
- All pins that can use output can use PWM

Lesson 1: Blinking an LED

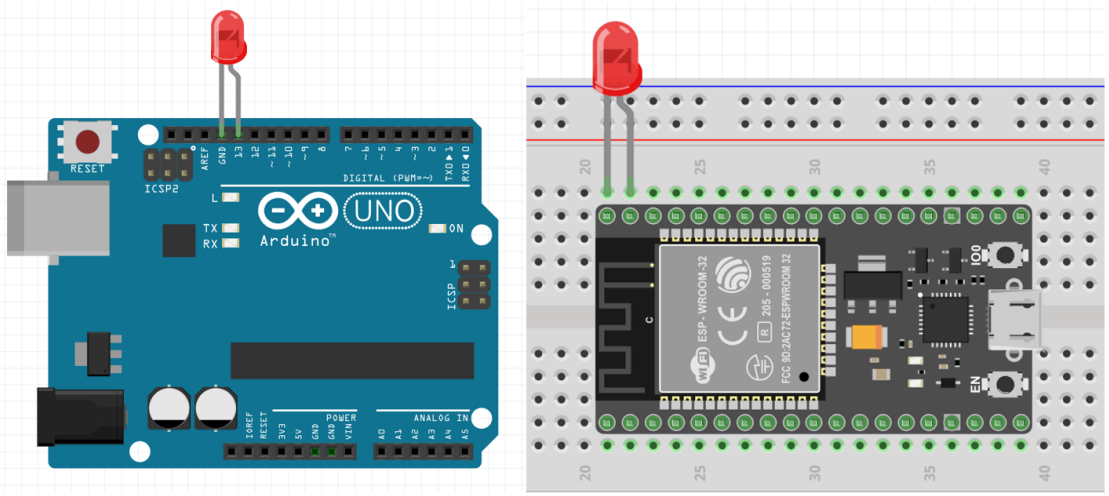
What You'll Need:

- Arduino/ESP32
- LED
- Breadboard

LED:
The new component is a light emitting diode (LED). It is a low-current light source that operates with currents only in a single direction. The longer pin is the positive terminal.



Wiring Diagram:



Directions:

Arduino	ESP32
1. Plug the led with positive terminal port 13, and negative terminal in ground	1. Plug the led with positive terminal port G23, and negative terminal in ground

Code (Light the LED):

```
int [variable] = [number];
```

This line creates a new integer variable of any name and sets it to a specified value. A variable name can be anything (note capitals are identified as different characters). A variable is a sort of container that stores information in it.

```
pinMode([variable], OUTPUT);
```

This line assigns a chosen variable as an output or an input on the Arduino. The Arduino takes the information from the variable and uses it as a port address.

```
digitalWrite([variable], HIGH);
```

This line digitally powers the chosen port to either high or low, meaning current on or off.

```
1  int led = 13;  (Use port 23 for ESP32)
2  // Assign the variable led to 13
3
4  void setup() {
5  pinMode(led, OUTPUT);
6  // Sets the led as and OUTPUT from port 13
7  }
8
9  void loop() {
10 digitalWrite(led, HIGH);
11 // Supplies current to the led port
12 }
```

Upload to the Arduino/ESP32 and the LED should light up.

Code (Blinking the LED):

`delay([number]);`

Pauses code operation for a period of time (milliseconds). The Arduino/ESP32 will continue its previous given operations during this period.

```
1  int led = 13; (Use 23 for ESP32)
2  // Assign the variable led to 13
3
4  void setup() {
5  pinMode(led, OUTPUT);
6  // Sets the led as and OUTPUT from port 13
7  }
8
9  void loop() {
10 digitalWrite(led, HIGH);
11 // Turn on LED
12 delay(1000);
13 // Wait 1 second
14 digitalWrite(led, LOW);
15 // Turn off LED
16 delay(1000);
17 // wait 1 second
18 }
```

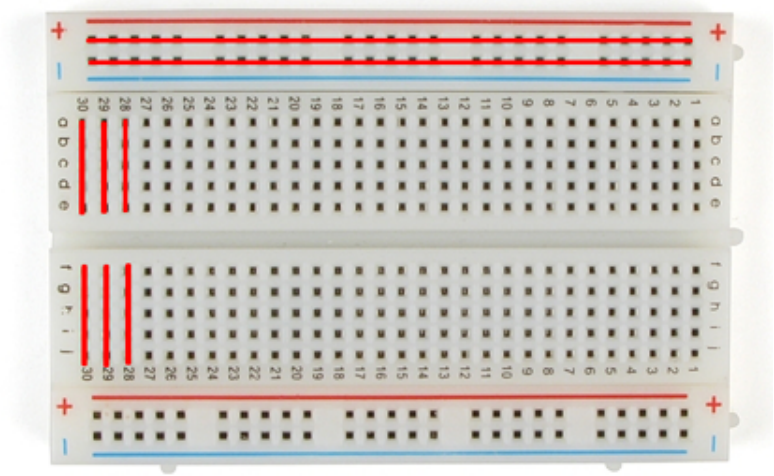
Upload the code to the Arduino/ESP32 and the LED should blink. Changing the value of the delay will change the frequency of the blinking.

Lesson 2: LED with a button

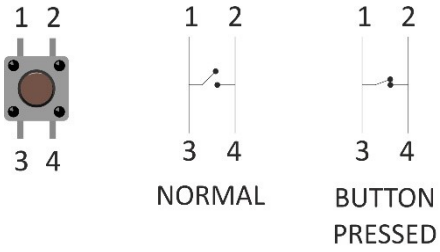
What You'll Need:

- Arduino/ESP32
- LED
- Breadboard
- 2 wires (male to male)
- Button

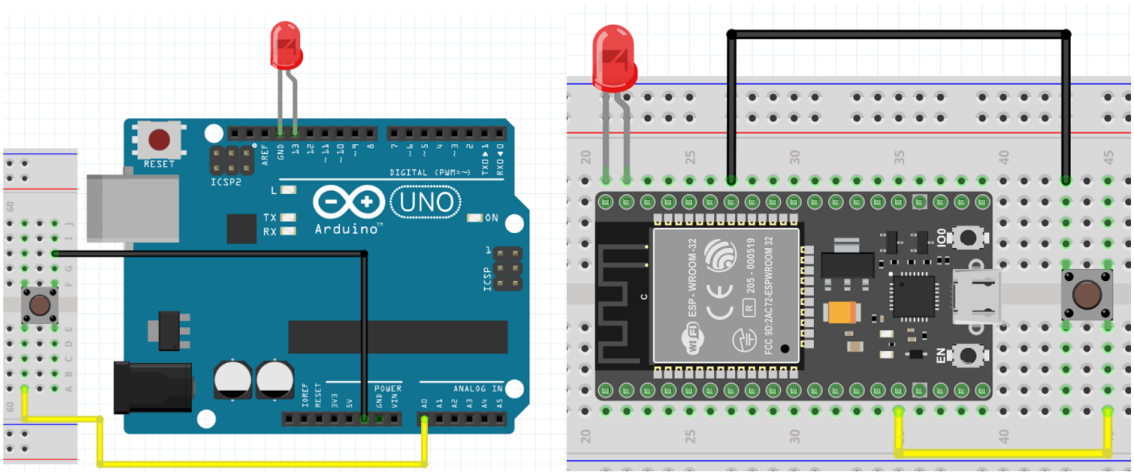
Breadboard:
A breadboard is a non-soldering technique of connecting components. Wires in the same column are connected, and some breadboards have indicated rows connected horizontally as shown below.



Button:
A button is a simple method of breaking a circuit. Standard buttons have 2 pins however, some buttons have 4. Indicated below, pressing a button connects 1 3 to 2 4.



Wiring Diagram:



Directions:

Arduino	ESP32
1. Plug the positive terminal of the Arduino in port 13, and negative to GND. Connect the button to the breadboard as indicated. 2. Using wires, connect the top left terminal to port A0, and connect the bottom right terminal to GND	1. Plug the positive terminal of the ESP32 in G23, and negative to GND. Connect the button to the breadboard as indicated. 2. Using wires, connect the top left terminal to GND, and connect the bottom right terminal to G12

Code:

```
digitalRead([variable])
```

Reads the incoming digital signal from a sensor.

```
if('something' == 'something'){
  // Runs code in brackets
}
```

Logic statements are used to give the boards intelligence. For example, if a variable is equal to a certain number, the microcontroller will run a specific command for those conditions. Operators such as the following may be used:

x == y	x equals y
x != y	x doesn't equal y
x < y	x is less than y
x > y	x is greater than y
x <= y	x is less than or equal to y
x >= y	x is greater than or equal to y

```
int led = 13; (Use 23 for ESP32)
int button = A0; (Use 12 for ESP32)

void setup() {
pinMode(led, OUTPUT);
pinMode(button, INPUT_PULLUP);
// The button is set to INPUT_PULLUP to give it extra resistance
}

void loop() {
int buttonPress = digitalRead(button);
// Variable that stores button press information

if(buttonPress == 0){
  // If button is pressed
  digitalWrite(led, HIGH);
}
else{
  // If button is not pressed
  digitalWrite(led, LOW);
}
}
```

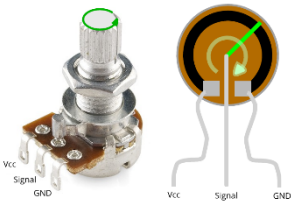
Upload the code to the Arduino/ESP32. When the button is not pressed, the LED should be off, and when the button is pressed, the LED should turn on.

Lesson 3: Dimming an LED

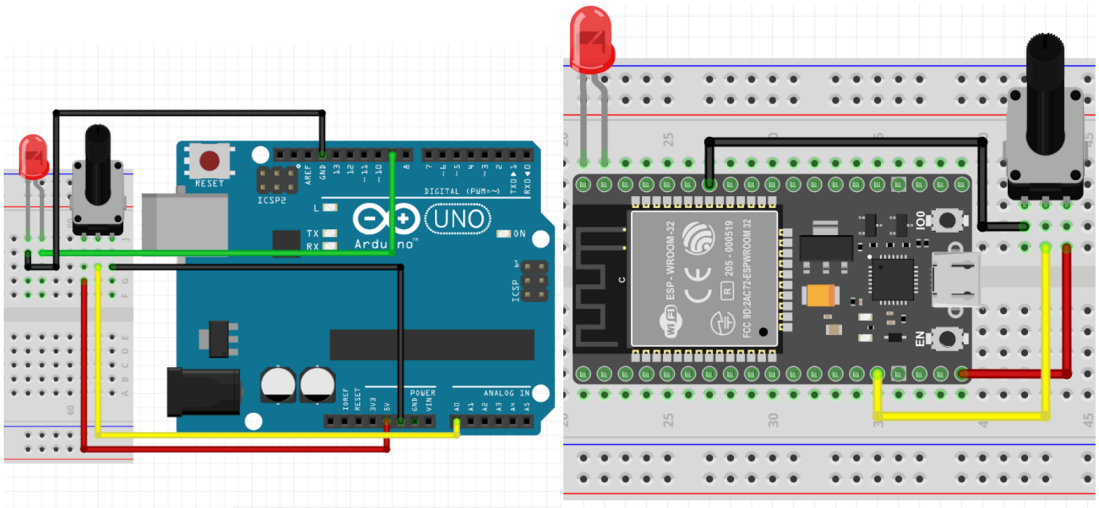
What You'll Need:

- Arduino/ESP32
- LED
- Breadboard
- Potentiometer
- 5 wires (male to male)

LED:
Potentiometers are a sort of variable resistor. They are a sensory input for the Arduino that gives a value between 0 and 1024 depending on how far the knob is twisted. Note that the signal must be used by an analog port. The VCC and GND ports are interchangeable however the signal must always be in the middle.



Wiring Diagram:



Directions:

Arduino	ESP32
1. Plug in the LED and potentiometer into the breadboard. 2. Connect a GND wire to the right side of the potentiometer and a 5V wire to the left side 3. Connect a wire from port A0 to the middle pin on the potentiometer 4. Connect a GND wire and a wire from port 9 correctly to the LED	1. Plug in the LED to port G23 and plug the potentiometer into the breadboard 2. Connect a GND wire to the left side of the potentiometer and a 5V wire to the right side 3. Connect a wire from port G12 to the middle pin on the potentiometer

Code (Serial Feedback):

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

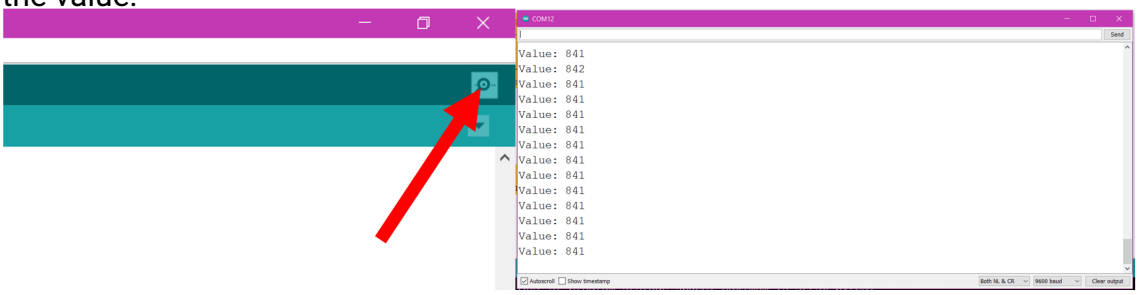
Starts serial communication with the computer

```
Serial.print([input]);
Serial.println([input]);
```

Displays the specified input to the monitor. Using 'println' creates a new line of text.

```
1  int led = 9; (Use 23 for ESP32)
2  int potentiometer = A0; (Use 12 for ESP32)
3
4  void setup() {
5  pinMode(led, OUTPUT);
6  pinMode(potentiometer, INPUT);
7  Serial.begin(9600);
8  // Begin serial communication
9  }
10
11 void loop() {
12 int reading = analogRead(potentiometer);
13 // Get a reading from the potentiometer
14 Serial.print("Value: ");
15 Serial.println(reading);
16 // Display the potentiometer reading
17 delay(100);
18 }
```

Upload the code to the Arduino/ESP32 and open the serial monitor in the top right corner (red arrow). The values should resemble the following. Twist the potentiometer to change the value.



Code (Dimming LED Arduino):

```
analogRead([variable]);
```

This line reads for an incoming analog information (range of numbers) from a port.

```
analogWrite([variable], [number]);
```

This line sends a power value between 0 and 255 to the specified port. This line can be used to send a range of powers.

```
map([number], 0, 1024, 0, 4);
```

The map function takes in a value and checking its position in the larger range, rewrites its position in a lower range. After the number, the first two numbers are upper and lower bounds of the incoming number range (numbers above are only examples). The second two numbers are the output bounds. Visualisation for this example is shown below.


```
int led = 9;
int potentiometer = A0;

void setup() {
pinMode(led, OUTPUT);
pinMode(potentiometer, INPUT);
}

void loop() {
int reading = analogRead(potentiometer);
// Variable that stores potentiometer information
int power = map(reading, 0, 1024, 0, 255);
// Variable that stores power value for led
analogWrite(led, power);
}
```

Upload the code to the Arduino and the LED should brighten when the knob is twisted.

Code (Dimming LED ESP32):

`ledcSetup([channel], [frequency], [resolution])`

This line is used to set up the properties of the PWM signal, and the channel it will run on. Resolution is how many bits are used for the signal (up to 16 bits). The ESP32 has 16 PWM channels and any GPIO can be used to output PWM using a channel.

`ledcAttachPin([GPIO], [channel]);`

This line is used to assign a GPIO pin to a PWM channel. This will be where your PWM channel will output its signal.

`ledcWrite([channel], [duty cycle]);`

This function is essentially the same as the analogWrite() function. It's used to write the duty cycle to the channel. The duty cycle takes a value between 0 and 100.

```
int led = 23;
int sensor = 12;

const int freq = 15000;
const int resolution = 8;
const int ledChannel = 0;
// Set constants for the led

void setup() {
  ledcSetup(ledChannel, freq, resolution);
  // Set the frequency and resolution of the channel
  ledcAttachPin(led, ledChannel);
  // Assign the channel to the pin
  pinMode(sensor, INPUT);
}

void loop() {
  int reading = analogRead(sensor);
  // Create a variable that reads the potentiometer
  int power = map(reading, 0, 4096, 0, 100);
  // Use the map function to make the duty cycle
  ledcWrite(ledChannel, power);
  // Write the duty cycle to the PWM channel
}
```

Upload the code to the ESP32 and the LED should brighten as the potentiometer is twisted.

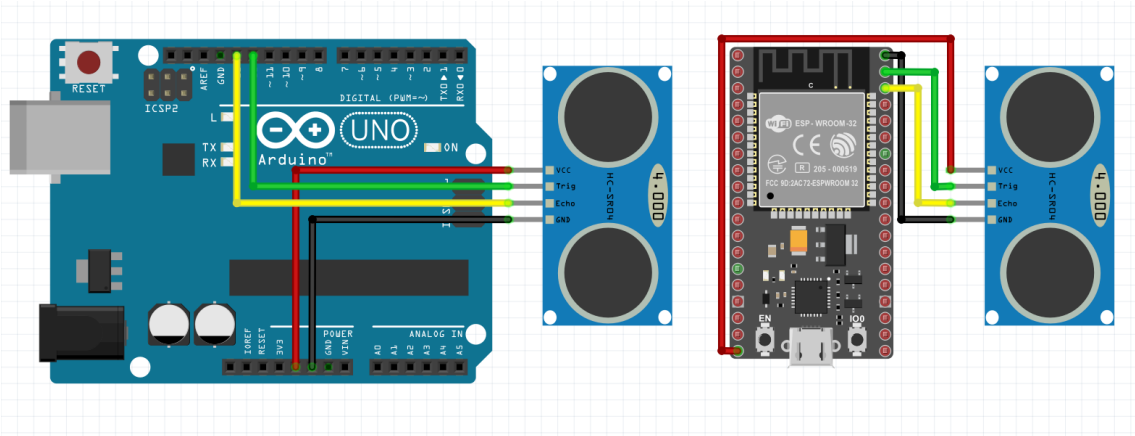
Lesson 4: Ultrasonic Sensor

What You'll Need:

- Arduino/ESP32
- Ultrasonic Sensor (HC-SR04)
- 4 wires (male to female) (Arduino)
- 4 wires (female to female) (ESP32)

Ultrasonic Sensor:
The HC-SR04 ultrasonic sensor sends out a sound and measures the time taken for the to return. It uses this technique to measure distances from objects. Note it has difficulty sensing objects that absorb sound like foam. Though the ultrasonic gives a range of readings, digital ports may be used to read it.

Wiring Diagram:



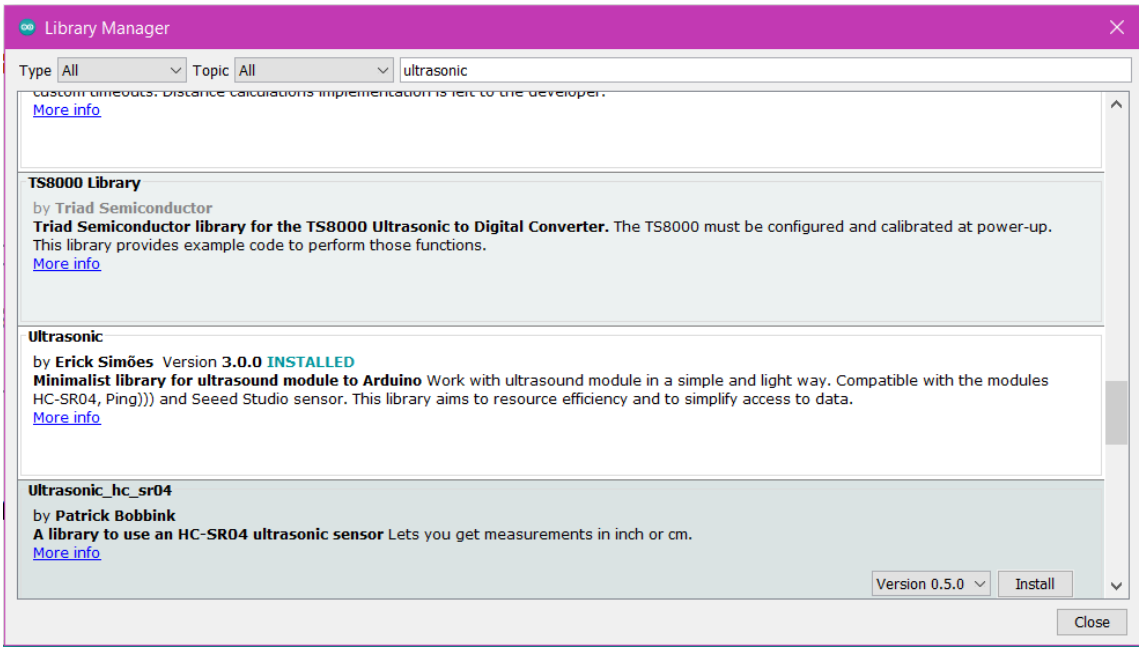
Directions:

Arduino	ESP32
1. Connect the VCC to 5V and GND to GND. Connect the Trig pin to port 12, and connect the Echo pin to port 13	1. Connect the VCC to V5 and GND to GND. Connect the Trig pin to port 23, and connect the Echo pin to port 22.

Code:

```
#include <Ultrasonic.h>
```

This is to introduce an ultrasonic library that does not come default with the software. It must be additionally downloaded. To do this, navigate to Tools -¿ Manage Libraries. In manage libraries, search for ultrasonic, and scroll to locate the indicated library. Once located, click install. When complete, restart the IDE, and the library should be in your Include Library as Ultrasonic.




```
Ultrasonic [variable](Trig, Echo);
```

This line creates an ultrasonic object. Type in the ports of the ultrasonic as indicated by Trig and Echo.

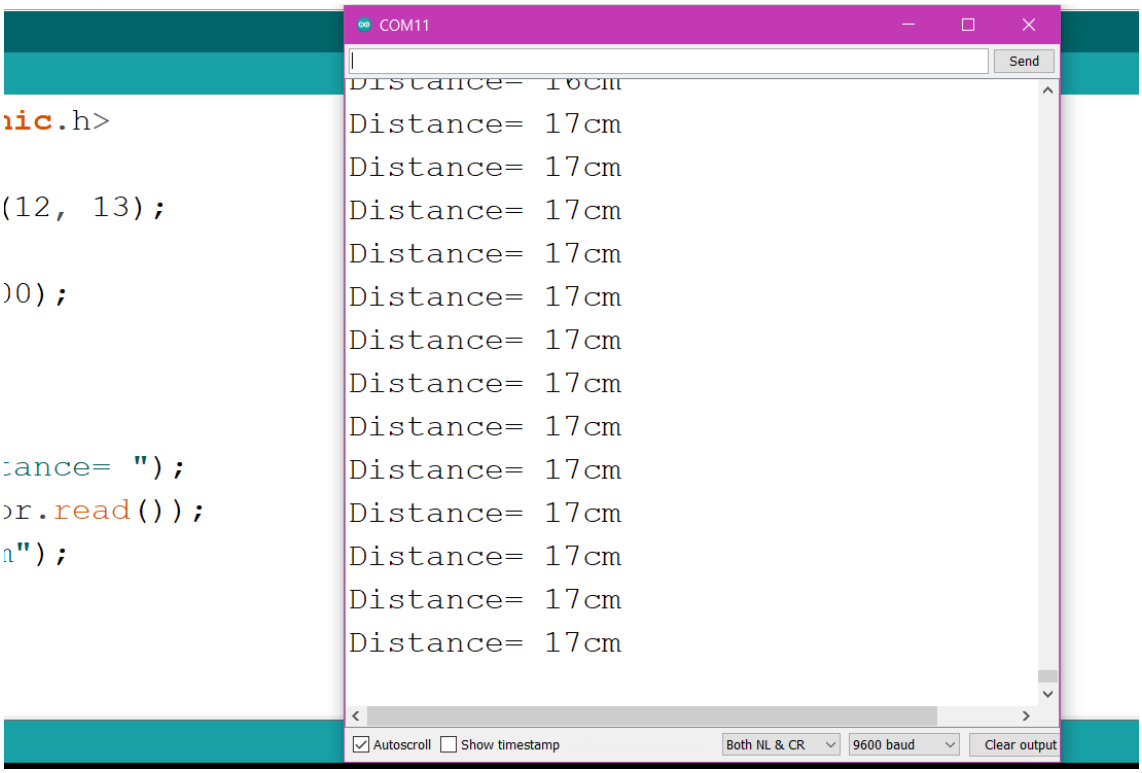
```
[varaible].read();
```

This returns the value currently being read by the sensor.

```ino
#include <Ultrasonic.h>

Ultrasonic sensor(12, 13);
void setup() {
  Serial.begin(9600);
}

void loop() {
Serial.print("Distance= ");
Serial.print(sensor.read());
Serial.println("cm");
delay(100);
}
```



Upload the code to the Arduino/ESP32 and open the Serial Monitor. The ultrasonic should be reading values as seen in the above Serial Monitor.

References

Sponsor 1

Sponsor 2

Sponsor 1

Sponsor 2

Sponsor 1

Sponsor 2