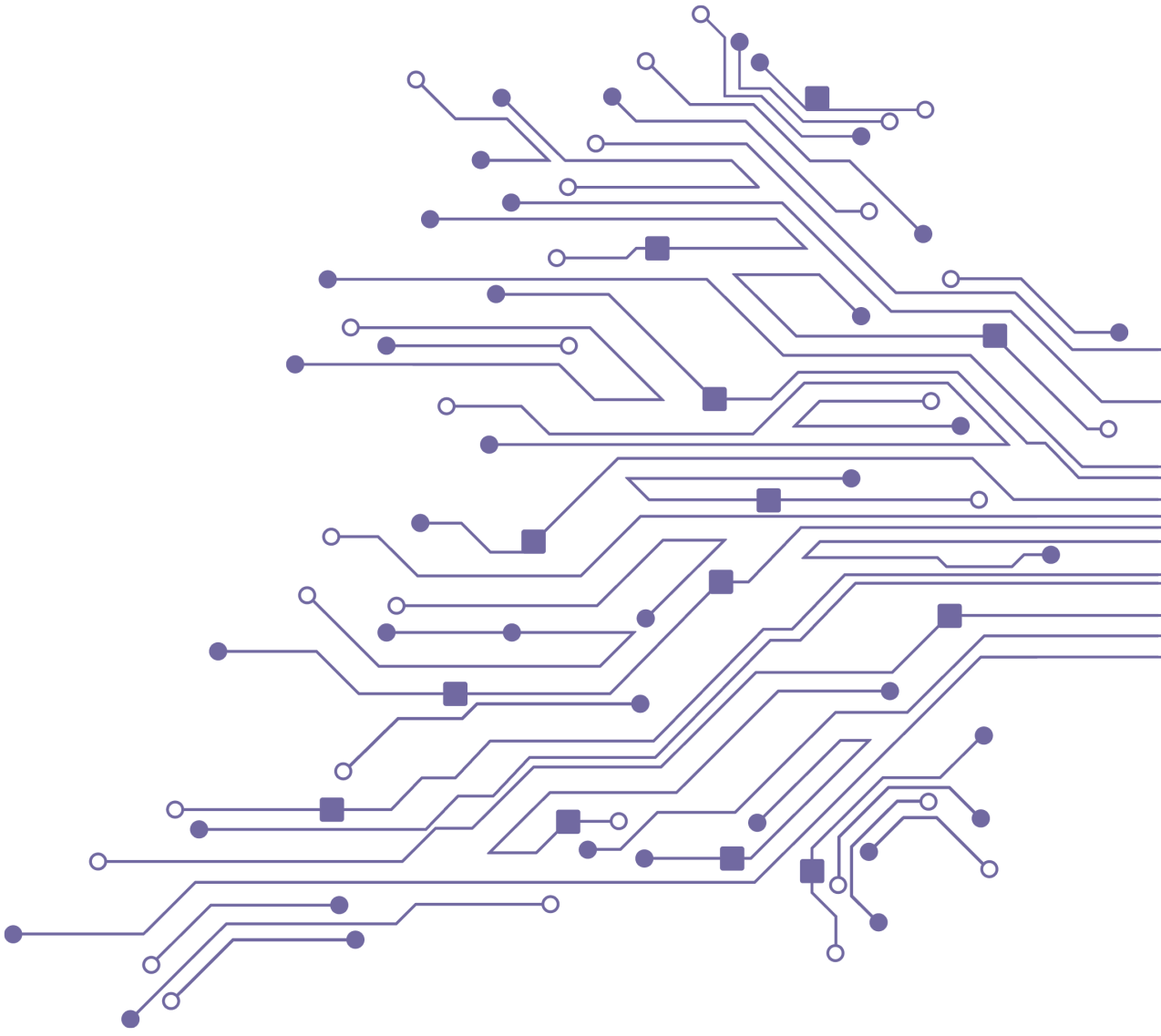
YQM.A.R.S

Mechatronics and Robotics Society

# Microcontroller Workshop

## 2023



# Thank you to our partners!

Turbo Sponsors

BOARD OF PROFESSIONAL ENGINEERS OF QUEENSLAND

COREMATIC

ROBOTICS • COMPUTER VISION • MACHINE LEARNING

jaycar

Gold Sponsors



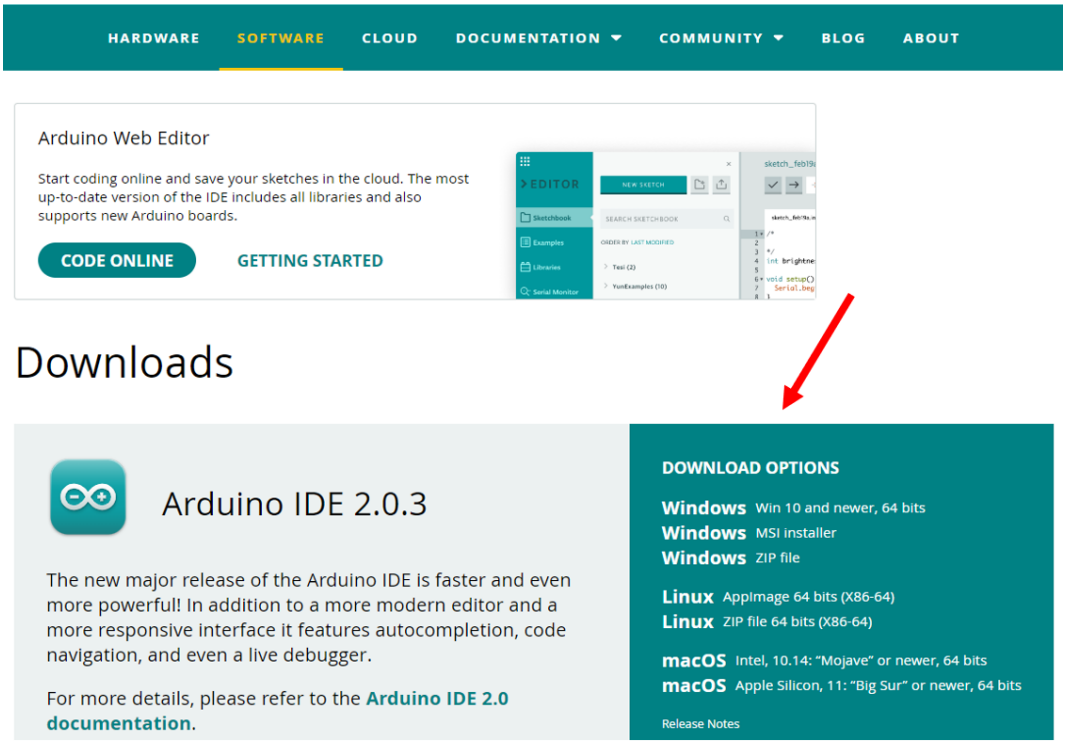

Silver Sponsors

ENGINEERS AUSTRALIA

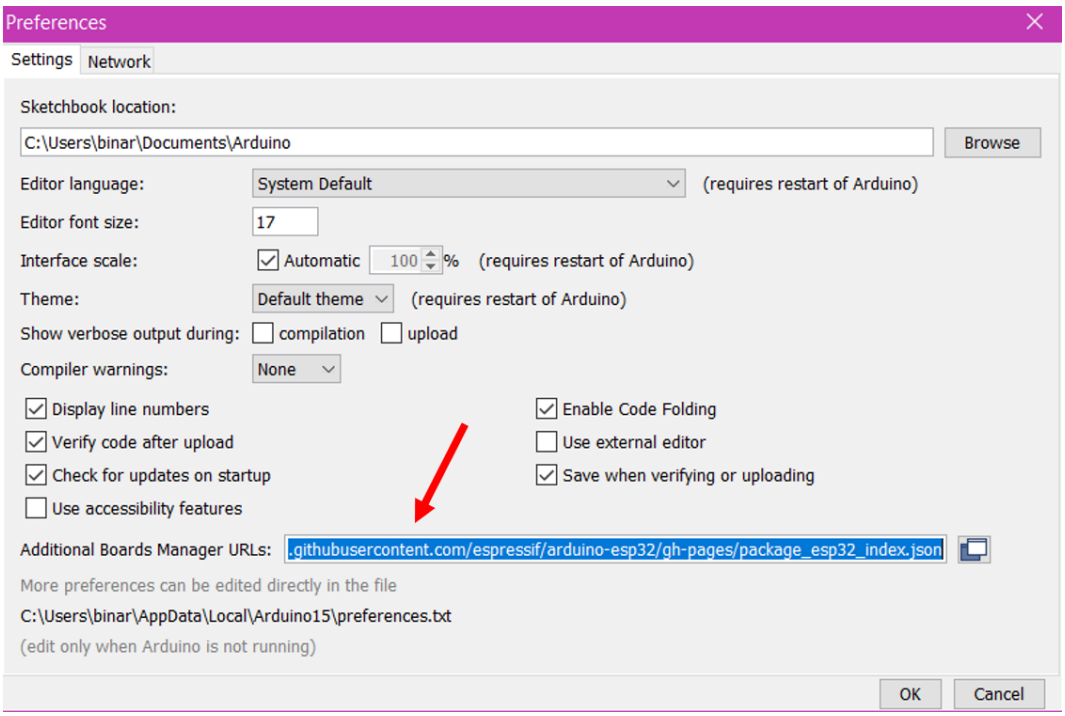
# Required Software

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

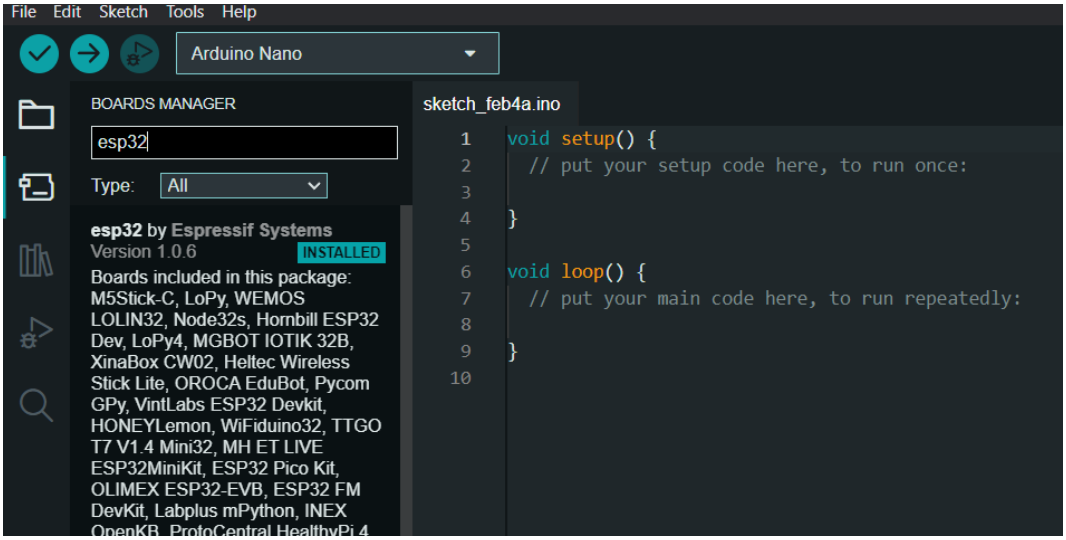
1. Head to the software page on the Arduino website and click on the appropriate installer. Follow the prompts and install on your computer



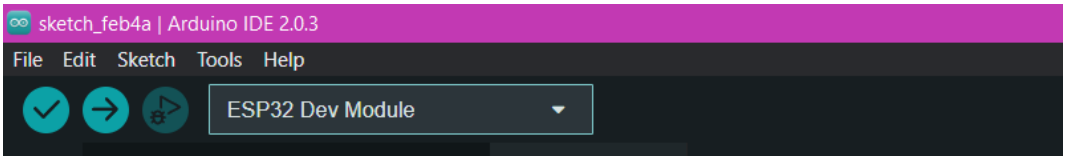

2. Next, open the IDE and head to **File** → **Preferences**, and enter the following URL into the 'Additional boards manager URLs': **https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json**



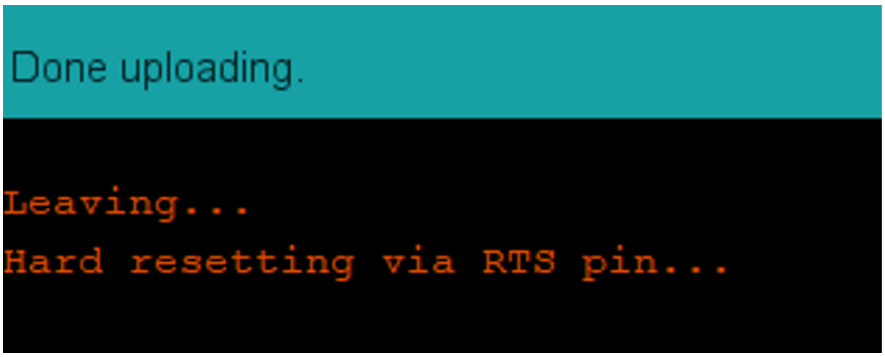
3. Go back to the IDE and head to **Tools** → **Board** → **Boards Manager**. In the old IDE, a new pop-up should appear, and in IDE 2, the toolbox should appear on the left hands side as shown below:




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.



# What Is A Microcontroller

A microcontroller is a small computer on a single chip that controls some or all of the functions of an electronic device or system. It has a processor, memory and input/output peripherals. Microcontrollers are used in many devices, such as cars, robots, home appliances and medical devices.

Microcontrollers usually have features such as:

- A central processing unit (CPU) that performs arithmetic, logic and I/O operations
- Volatile memory (RAM) for data storage
- Non-volatile memory (ROM, EPROM, EEPROM or Flash) for program and operating parameter storage
- Discrete input and output bits that allow control or detection of individual pins
- Serial input/output ports (UARTs) that enable communication with other devices
- Other peripherals such as timers, counters, analog-to-digital converters, etc.

# Hobby Microcontroller Options

- Arduino Family
  - Arduino Uno (ATmega328P)
  - Arduino Mega (ATmega2560)
  - Ardiono Nano (ATmega328P)
- Raspberry Pi Family
  - Raspberry Pi Model B
  - Raspberry Pi Zero
  - Raspberry Pi Pico
- Espressif Family
  - ESP8266 Development Boards
  - ESP32 Development Boards

# Powering A Microcontroller

## Arduino Input Voltage Range

The voltage range of an Arduino board depends on the specific board model. However, most Arduino boards operate on a voltage range of 5V to 12V DC. Some newer boards, such as the Arduino Due, can operate on a wider range of voltages, from 7V to 20V DC. Your laptop USB ports usually supply a well regulated logic-level voltage of 5V, so this

## Main Power Supply Voltage vs Logic-Level Power

Arduino boards have two power input pins: the Vin pin and the 5V pin. The Vin pin is connected to the main power supply voltage, which can be in the range of 5V to 12V or higher, depending on the board. The 5V pin, on the other hand, is connected to the logic-level power supply, which is usually 5V or 3.3V.

The main power supply voltage is used to power external components, such as sensors, motors, and other devices connected to the Arduino board. The logic-level power supply, on the other hand, is used to power the microcontroller and other digital components on the board.

## Drivers and Shields

Drivers and shields are add-on boards that are designed to extend the functionality of an Arduino board. Drivers are used to control motors, relays, and other high-power devices that require more current than the Arduino board can provide. Shields, on the other hand, are add-on boards that provide additional features, such as Wi-Fi connectivity, GPS, and LCD displays.

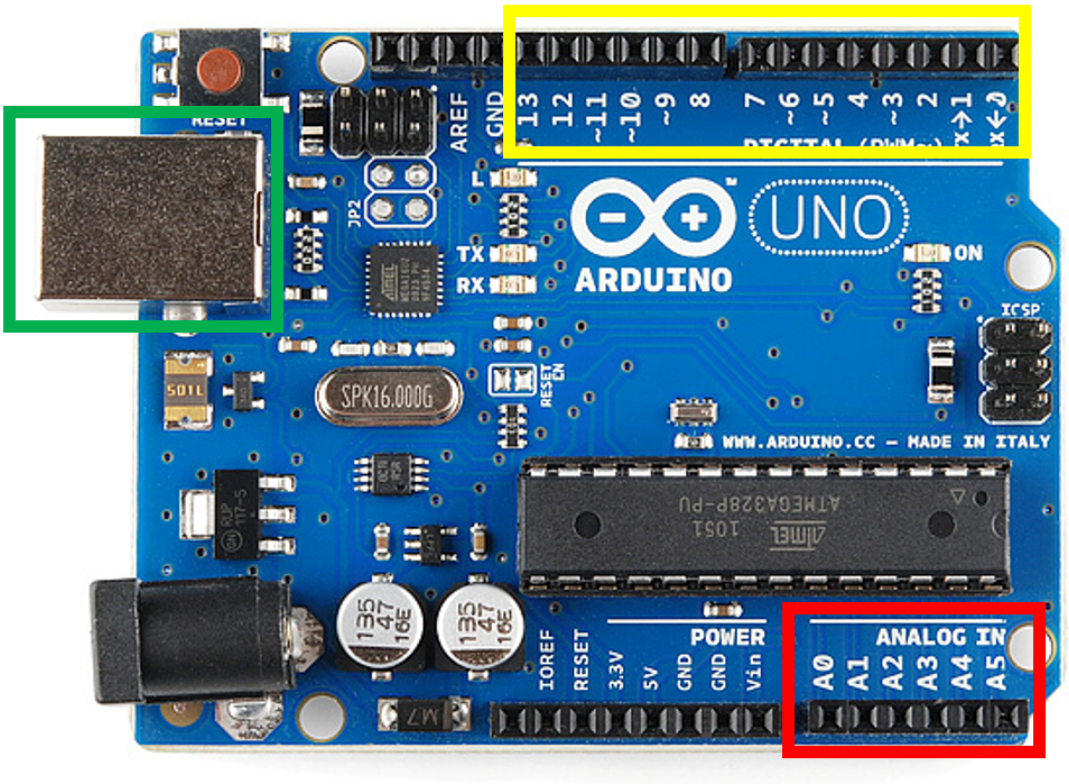
Both drivers and shields may require a separate power supply, depending on the power requirements of the devices they are controlling. In some cases, they may also require level shifting circuitry to ensure that the signals sent between the Arduino board and the driver or shield are compatible.

- Sheilds Available at Jaycar:
  `https://www.jaycar.com.au/search?text=sheild`
- Motor Shield (Usefull for ENGG1100):
  `https://www.jaycar.com.au/arduino-compatible-motor-servo-controller-module/p/XC4472`

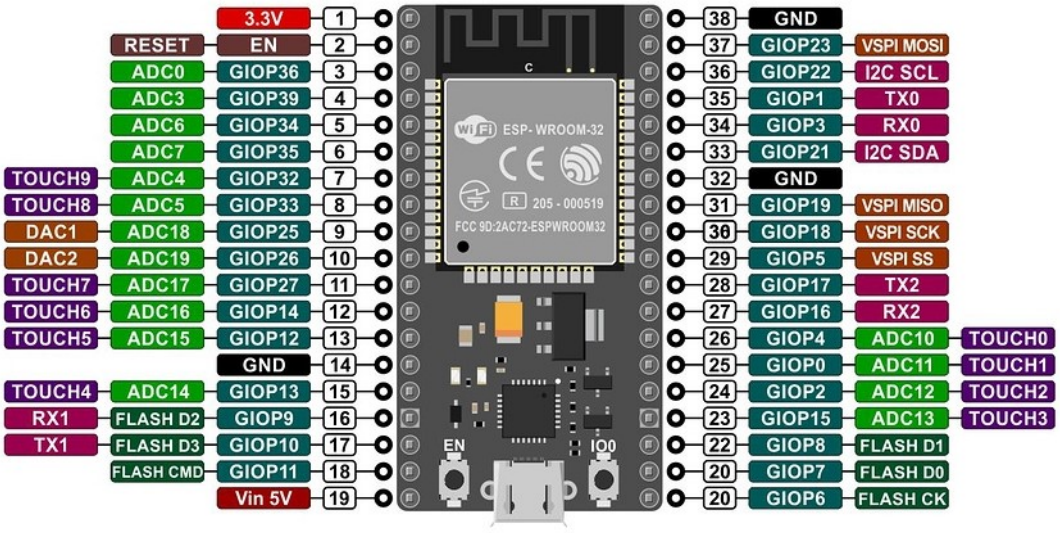
*(These items are usually significantly cheaper online - but JayCar is good in a pinch!)*

# 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



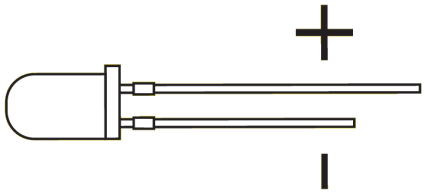

- The pins in green are the GPIO pins which can serve as inputs and outputs unless otherwise signed
- Ports labelled ADC can be used to read analog signals
- Ports labelled TOUCH have internal capacitive touch sensors. For example, these pins can be used as touch sensors using just a finger
- All pins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

# Lesson 1: Blinking an LED

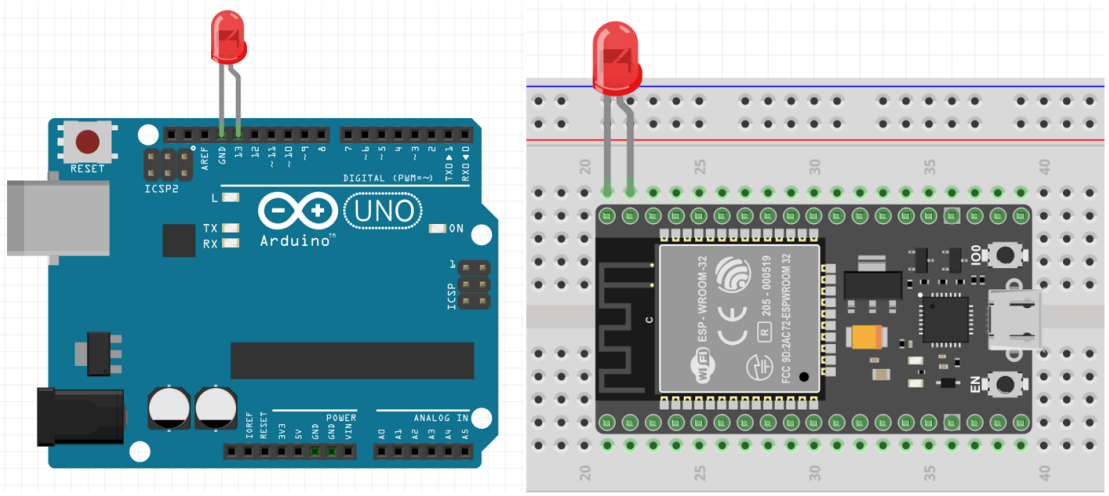
## What You'll Need:

- 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 in port 13, and negative terminal in ground | 1. Plug the led with positive terminal in 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 }
```

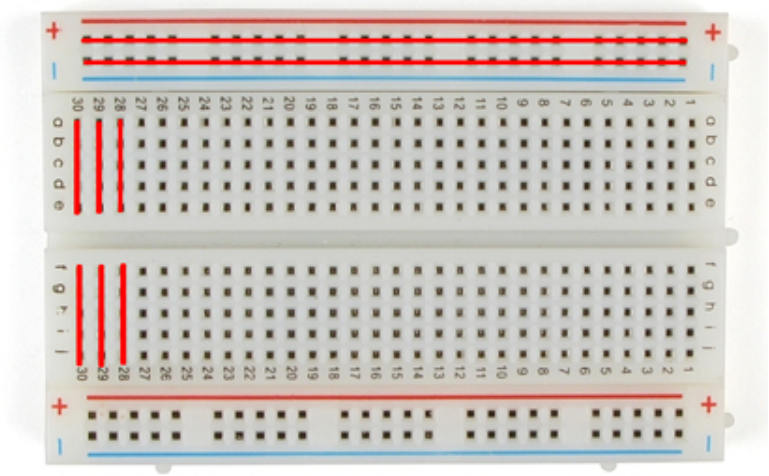
Upload the code to the Arduino/ESP32 and the LED should blink. Changing the value of the delay will change the frequency of the blinking.

# Lesson 2: LED with a button

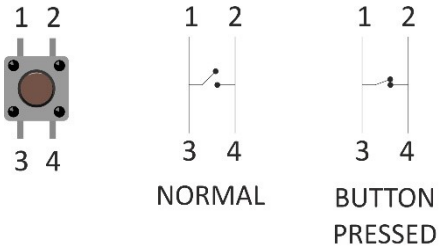
## What You'll Need:

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

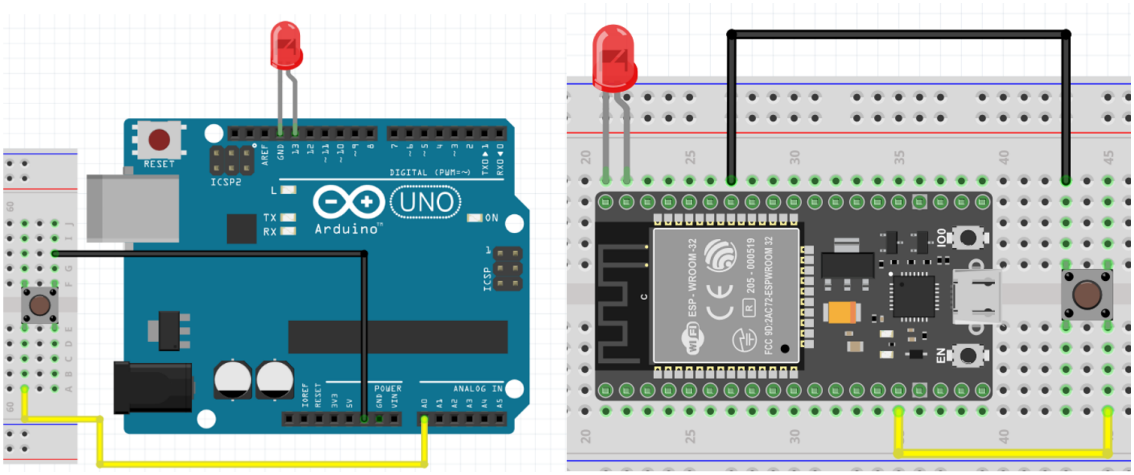
**Breadboard:**
A breadboard is a non-soldering technique of connecting components. Wires in the same column are connected, and some breadboards have indicated rows connected horizontally as shown below.



**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.<br>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.<br>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.

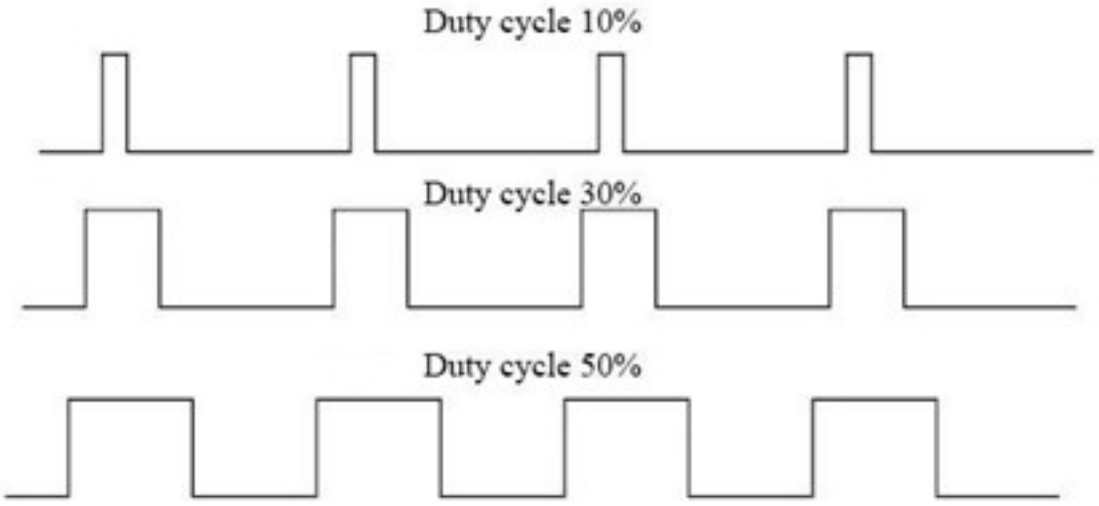
# Lesson 3: PWM Servo Motor Control

## What You'll Need:

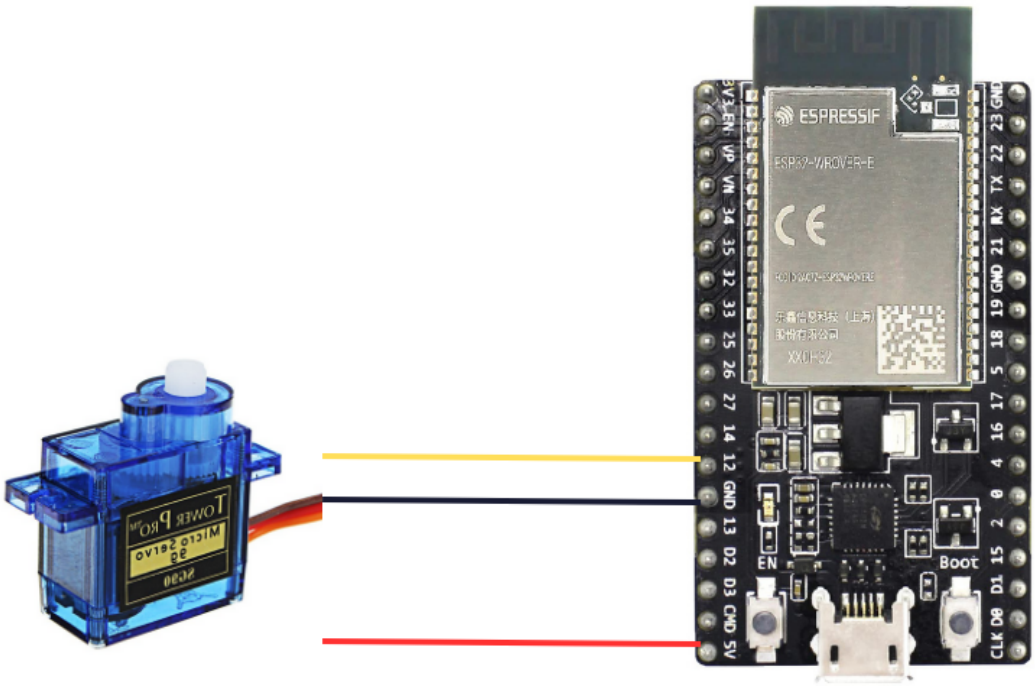
- Arduino/ESP32
- Servo Motor
- LED & Pushbutton
- Breadboard
- 5 wires (male to male)

**Servo Motor:**
A servo motor is a type of motor that can control its position, speed and torque precisely with feedback from a sensor and a controller. It is used for applications that require accurate and dynamic movements, such as robotics, CNC machinery and automated manufacturing. A servo motor consists of a suitable motor, a sensor for position feedback and a controller that interprets the input signal and adjusts the motor accordingly. The servo motors used in this workshop are controlled using a PWM signal.

**Pulse Width Modulation (PWM):**
PWM: Pulse width modululation takes a digital signal (high or low) and rapidly switches the signal on and off. This technique is used to control the amount of power delivered to a load (Can be thought of as analog power). The duration of the on and off periods can determine the average power delivered to the load. This is creates a square wave signal, this signal can be described by its duty cycle and its period. Where duty cycle is how much of the signal is on during one single period.




## Wiring Diagram:



## Directions:

| Arduino | ESP32 |
| --- | --- |
| 1. Instructions pending... | 1. Connect the servos red wire to 5V, black wire to GND and the yellow wire to pin 12.<br>2. Need to import ESP32Servo.h, this library will need to be downloaded from the 'Library Manager'<br>3. (servo).setPeriodHertz this sets the frequency at which the pwm signal occurs<br>4. (servo).attach changes the corresponding pin to output a pwm signal<br>5. (servo).write writes an angle for the PWM<br>6. More useful functions can be found through reading the documentation: `https://madhephaestus.github.io/ESP32Servo/classServo.html#a613aa8f5aa81971357e6ff36df92309d` |

## Code (ESP32):

```
#include <ESP32Servo.h>

Servo servo;

int servoPWM = 12;
int angle;


void setup() {

  servo.setPeriodHertz(50);
  servo.attach(servoPWM);
}
// Main loop moves servo through a range of angles
void loop() {
  angle = 0;
  while (angle < 150) {
    delay(200);
    servo.write(angle);
    delay(200);
    angle += 15;
  }
}


```

# Lesson 4: Wireless Communication

Wireless communication is essential for remotely operated devices/robotics. Although not required, you are usually encouraged to use wireless communication in the ENGG1100 group projects. Wireless control may also come in handy in further group projects such as ENGG2800 and METR2800.

Alternatively, many groups will also just use a long USB / serial cable to remotely operate their machine - yuck. After this tutorial you will hopefully gain the confidence to use wireless communication in your future projects and enjoy the benefits of a more flexible and reliable system.

For this lession we will obviously need a microcontroller with wireless cababilities. UQ MARS has supplied a limited number of ESP32's. These are like cool little arduinos, but are faster, cheaper, have more memory and most importantly feature WiFi (2.4 GHz) and bluetooth capabilites. Alternatively, with a couple of tweaks, you can also use an ESP8266 with this tutorial - ask one of our execs if you have any trouble. **Share one of the ESP32's with a buddy if there isn't enough to go around**.

FYI: JayCar sells ESP32's and various arduino WiFi products which use the ESP8266.

## Overview

We will need to set up a wireless communication link between your Arduino board and your computer. This link is called a "TCP Socket" and it can be thought of as a 'virtual usb port'. A socket is identified using both an IP Address and a port number. you can read more about them here: `https://www.howtouselinux.com/post/tcp-socket`.

In this lesson, we will configure the ESP32 to host its own wireless network (you can think of it like your phone's hotspot). You will then connect your laptop to the ESP32's wireless network. Now that both devices are on the same network - we can connect them using a socket (a virtual USB cable - if you like).

We will need to set up a "socket" on both ends of the connection:

- ESP32 - "Server" Socket
- Computer (python script) - "Client" Socket

Oh - you will also need to install **python 3** and the **pygame** python library using pip. If you don't know how to do this, don't worry, we will go through it in the workshop.
All the code for this workshop can be found here: `LINKTOCODEONGITHUB`

## Code: ESP32 (Server)

```
//#include <ESP8266WiFi.h>
#include <WiFi.h>

WiFiServer serverr(6500);

byte ContrInput;

//--------------------------------------------------------------------------------

void setup(){
  Serial.begin(9600);
  WiFi.softAP("(YOUR NAME)'s ESP32 Network", "thereisnospoon");
  Serial.println(WiFi.localIP());
  //WiFi.setSleep(WIFI_NONE_SLEEP);//WiFi.setSleepMode(WIFI_NONE_SLEEP);
  serverr.begin();
}

//--------------------------------------------------------------------------------

void loop(){
  WiFiClient controllerSock = serverr.available();

  while (controllerSock){
    ContrInput = controllerSock.read();
    while(ContrInput != 255){
      yield();
      //Serial.write(ContrInput);

      //---------------Robot Functionality Starts Here------------------

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



      //---------------Robot Functionality Ends Here--------------------

      ContrInput = controllerSock.read();
    }
    delay(50);
  }

  delay(2000);
}

```

## Code: Laptop (Python Client)):

```
import pygame
import socket

class Main():
    def __init__(self):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.address = socket.gethostname()
        print("Remote IP address: ", self.address)
        print(self.s.connect(("192.168.4.1", 6500))) # Type your ESP Server details here


        pygame.init()
        self.res = (200,200)
        self.screen = pygame.display.set_mode(self.res)
        pygame.display.set_caption("PBL11 Violet Control Pad")
        self.done = False

        self.clock = pygame.time.Clock()

        self.loop()

        pygame.quit();

    def loop(self):
        while not self.done:
            for event in pygame.event.get():

                if event.type == pygame.QUIT:
                    self.done = True

                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_ESCAPE:
                        self.done = True
                    for i in range(1, 127):
                        if event.key == i:
                            print("Pressed Key")
                            self.s.sendall(bytes([i]))
                            if i in range(49, 57, 1):
                                self.s.sendall(bytes([116])) # Send keypress through to esp

                # Advanced key-up functionality:
                if event.type == pygame.KEYUP:
                    if (event.key == pygame.K_w) or (event.key == pygame.K_s):
                        print("Unpressed Key")
                        self.s.sendall(bytes([120])) # turn off motors


            self.clock.tick(30);

# For more advanced control, think about this function:
def output(socket, info):

    print("sending output")

    # Send through lots of bytes at once
    socket.sendall(bytes([142, info[0], info[1], info[2], info[3], info[4], info[5], 10]))

    # See the bytes just sent in python terminal
    print(bytes([142, info[0], info[1], info[2], info[3], info[4], info[5], 10]))

    print("end output")


mainn = Main()
```

## Code: ESP32 (Server) - with servo motor control

Press *w* to set servo position to 90 degrees!
Press *s* to set servo position to 0 degrees.

```
//#include <ESP8266WiFi.h>
#include <WiFi.h>
#include <ESP32Servo.h>

Servo servo;

int servoPWM = 14;

WiFiServer serverr(6500);

byte ContrInput;

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

  servo.setPeriodHertz(50);
  servo.attach(servoPWM);

  WiFi.softAP("(YOUR NAME)'s ESP32 Network", "thereisnospoon");
  Serial.println(WiFi.localIP());
  //WiFi.setSleep(WIFI_NONE_SLEEP);//WiFi.setSleepMode(WIFI_NONE_SLEEP);
  serverr.begin();
}
//--------------------------------------------------------------------------------

void loop(){
  WiFiClient controllerSock = serverr.available();

  while (controllerSock){
    ContrInput = controllerSock.read();
    while(ContrInput != 255){
      yield();

      //---------------Robot Functionality Starts Here-----------------

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

      if (ContrInput == 119) {
        servo.write(90);
      } else if (ContrInput == 115) {
        servo.write(0);
      }
      //---------------Robot Functionality Ends Here-------------------

      ContrInput = controllerSock.read();
    }
    delay(50);
  }

  delay(2000);
}
```

# References


- (Paraphrase from OpenAI's ChatGPT AI language model, personal communication, March 25, 2023)


# Credits

- Binara Wasala
  - "Required Software" section
  - "Overview of the Arduino and ESP32" section
  - "Lesson 1: Blinking an LED" section
  - "Lesson 2: LED with a button" section
- Reuben Richardson
  - "What Is A Microcontroller" & "Hobby Microcontroller Options" sections
  - "Lesson 3: PWM Servo Motor Control" section
  - "Lesson 4: Wireless Communication" section
- Hokul Srikantthan
  - "Lesson 3: PWM Servo Motor Control" section
  - "Powering A Microcontroller" section