Workshop IV

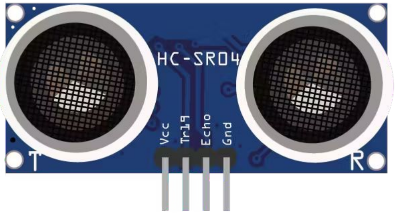
# Catch Up Workshop

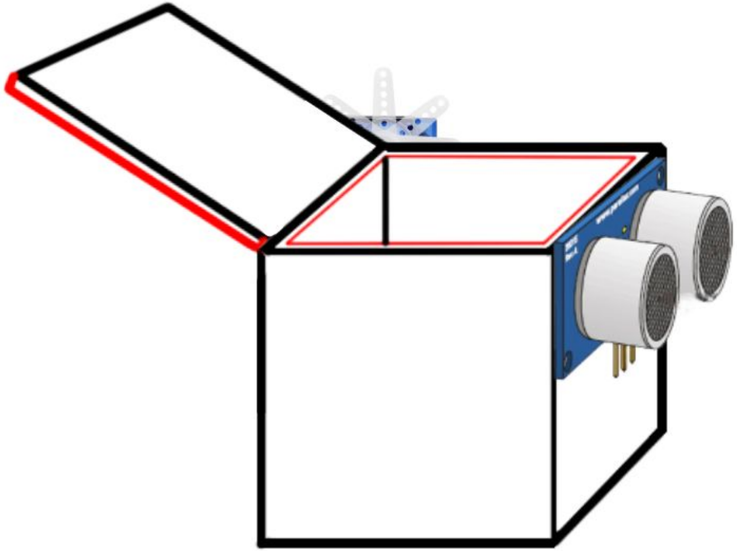
SECTION I

# Project 4 Review/Tips

# Project 4 Overview

- Build a mini trash can that uses an **ultrasonic sensor, micro servo,** and **ESP32**



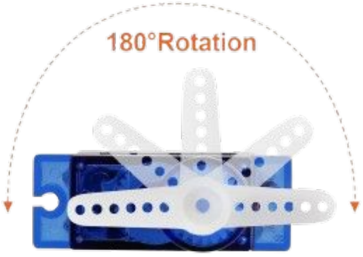


You will learn:

- ESP32 (Continued)
- Servos
- Ultrasonic Sensor
- Arduino Libraries
- Tips for Programming in Arduino IDE

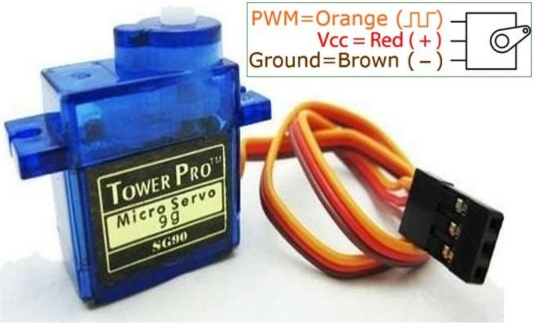
# Servo

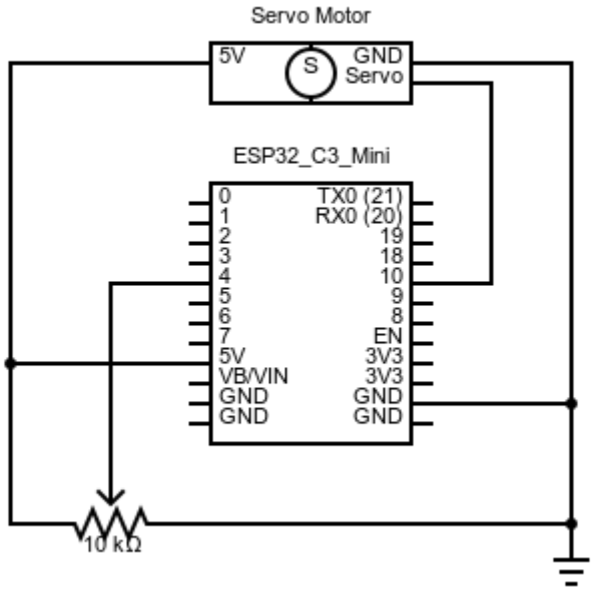
- A **servo motor** is a type of motor that converts electrical energy into mechanical energy to achieve precise control between 0°-180°
- Has three different wires: power (red), ground (brown), and pulse-width modulation (orange)
  - Insert jumper wires into the female end of the micro servo and to connect it to the ESP32 on a breadboard
- The servo will be controlled based on readings from the **ultrasonic sensor**



# Servo Pinout

- To the ESP32, connect:
- Power (red) → 5V pin
- PWM (orange) → any GPIO
  - Remember PWM = Pulse-Width Modulation!
- Ground (brown) → GND



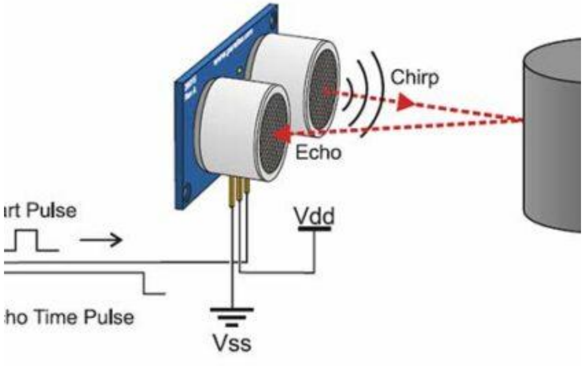


# Ultrasonic Sensor

- A ultrasonic sensor is an electronic device that measures distance via high-frequency waves
- Has four different pins; **power**, **ground**, echo, and trig

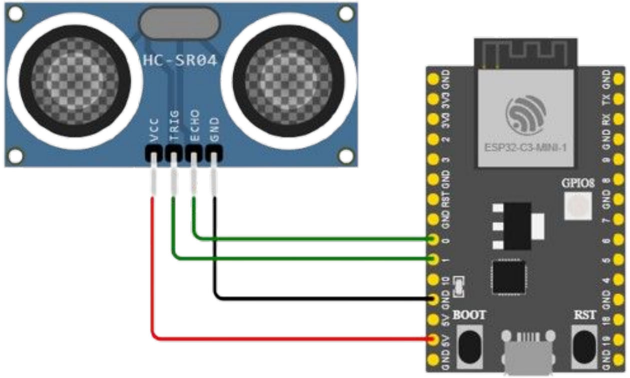
Sequence of events:

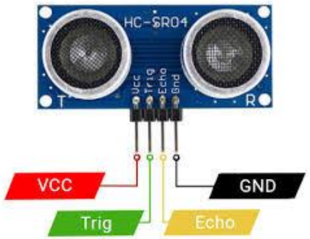
1) **TRIG** pin receives a trigger command, and the sensor sends a sound wave
2) Right when the sound wave is sent, the ECHO pin is flipped to HIGH
3) The sound wave is reflected back and the moment it is detected, the ECHO pin is flipped to LOW
4) Microcontroller measures duration of the ECHO pulse



# Ultrasonic Sensor Pinout

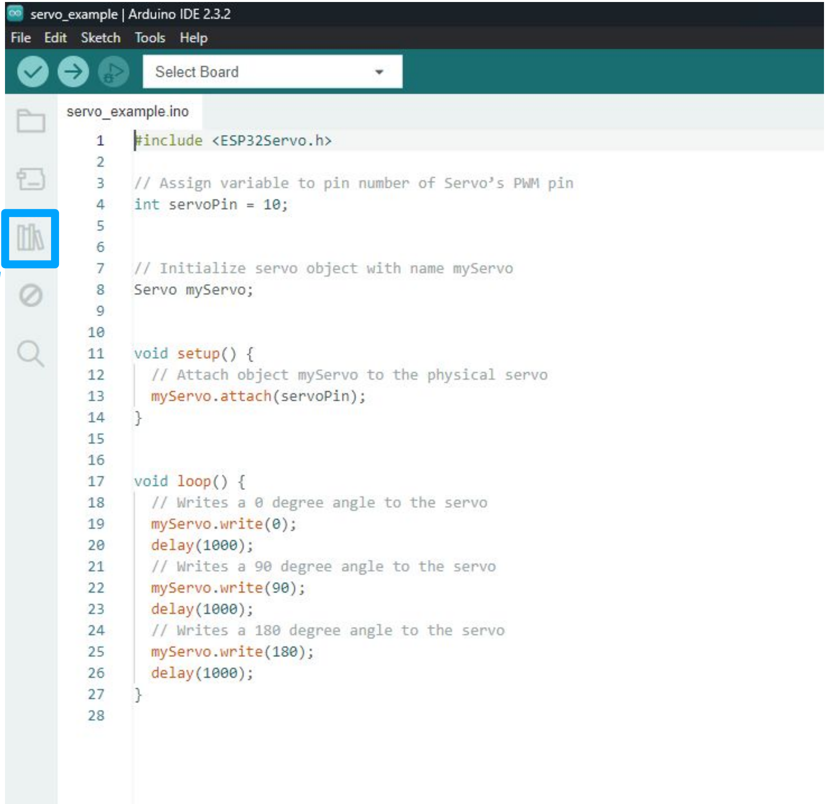
- Connect the **VCC** pin of the ultrasonic sensor to the **5V** pin of ESP32
- Connect the TRIG and ECHO pins to any GPIO pin of the ESP32
- Connect the **GND** of the ultrasonic sensor to the **GND** pin of the ESP32





# Arduino Library

- A collection of pre-written code that makes it easier to interface with hardware or perform specific tasks
- Libraries made by Arduino’s development team or Arduino community members
- Libraries can be downloaded from the library manager

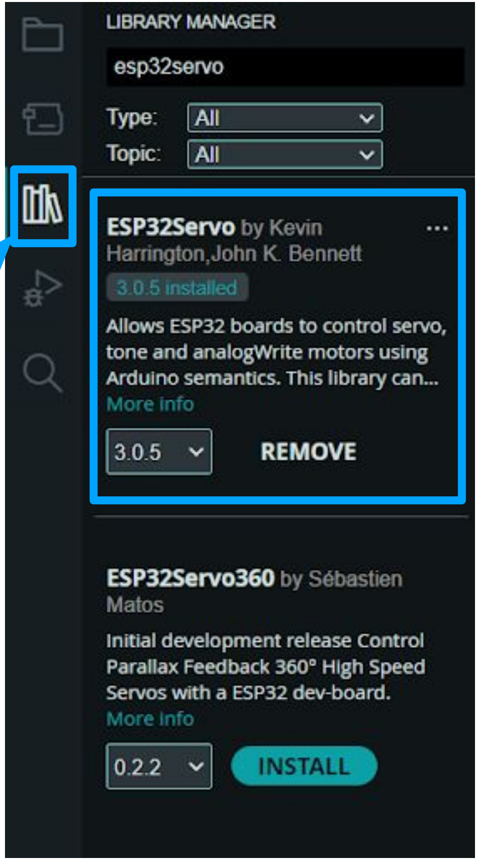


```
#include <ESP32Servo.h>

// Assign variable to pin number of Servo’s PWM pin
int servoPin = 10;


// Initialize servo object with name myServo
Servo myServo;


void setup() {
  // Attach object myServo to the physical servo
  myServo.attach(servoPin);
}


void loop() {
  // Writes a 0 degree angle to the servo
  myServo.write(0);
  delay(1000);
  // Writes a 90 degree angle to the servo
  myServo.write(90);
  delay(1000);
  // Writes a 180 degree angle to the servo
  myServo.write(180);
  delay(1000);
}

```

# Servo Library

- The **ESP32Servo** library will need to be installed to control the Servo in Project 4

1. Click on the Library manager icon
2. Type in ESP32Servo
3. Download the first result by Kevin Harrington, John K. Bennet



LIBRARY MANAGER

esp32servo

Type: All

Topic: All

**ESP32Servo** by Kevin Harrington,John K. Bennett

3.0.5 installed

Allows ESP32 boards to control servo, tone and analogWrite motors using Arduino semantics. This library can...

More info

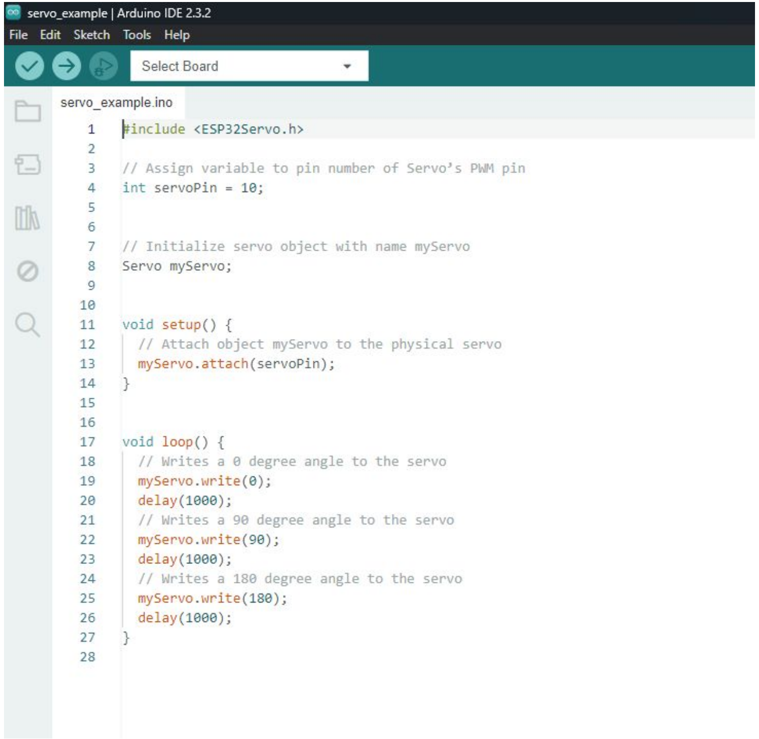
**ESP32Servo360** by Sébastien Matos

Initial development release Control Parallax Feedback 360° High Speed Servos with a ESP32 dev-board.

More info

# Servo Library (Continued)

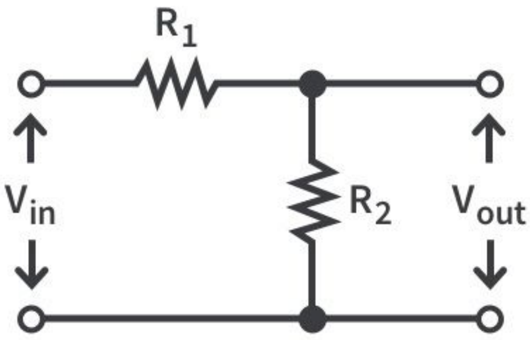
- Must #include <ESP32Servo.h> to use the library
- `Servo your_servo_name`
  - Initializes servo object with whatever you want to name the servo object
- `your_servo_name.attach(int pin)`
  - Links the servo object to physical pin `int pin`
- `your_servo_name.write(int angle)`
  - Moves the servo to a specified degree (0°-180°)



```
#include <ESP32Servo.h>

// Assign variable to pin number of Servo’s PWM pin
int servoPin = 10;


// Initialize servo object with name myServo
Servo myServo;


void setup() {
  // Attach object myServo to the physical servo
  myServo.attach(servoPin);
}


void loop() {
  // Writes a 0 degree angle to the servo
  myServo.write(0);
  delay(1000);
  // Writes a 90 degree angle to the servo
  myServo.write(90);
  delay(1000);
  // Writes a 180 degree angle to the servo
  myServo.write(180);
  delay(1000);
}
```

# Voltage Divider

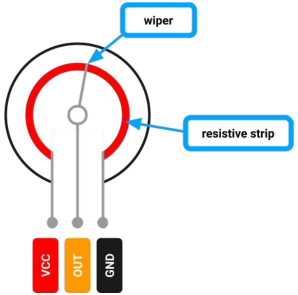
- A circuit configuration with **two resistors** that is used to scale down a voltage (**Vout**).
- Creates a ratio of two resistors to achieve a desired output voltage
  - **R1** and **R2** are connected in series
  - **Vin** is the input voltage
  - **Vout** is the output voltage across **R2**



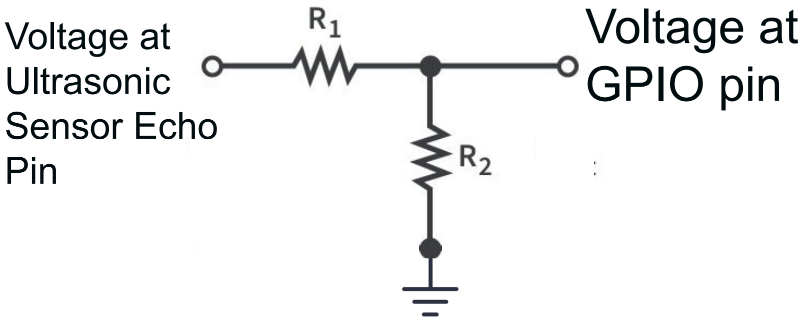
$$V_{out} = \left(\frac{R_2}{R_1 + R_2}\right) V_{in}$$

# Voltage Divider in Project 4

- A **potentiometer** is an adjustable resistor divider
  - A wiper/slider divides the resistive element into two adjustable parts
- The Ultrasonic Sensor is rated for 5V, but the GPIO pins are rated for 3.3V. We need to reduce the voltage at the this pin
  - A Voltage divider solves this



$$V_{out} = \left(\frac{R_2}{R_1 + R_2}\right) V_{in}$$



Voltage at Ultrasonic Sensor Echo Pin

Voltage at GPIO pin

# Map Function

- Re-maps a number from one range to another
- `map(int inputValue, int min_range_1, int max_range_1, int min_range_2, int max_range_2)`
    - `min_range_1` and `max_range_1` are the min and max values of the first range, and `min_range_2` and `max_range_2` are the min and max values of the second range

Map function in Project 4 Checkpoint 1:

- You will need to map the range of the ADC (0-4095) to the servo range (0-180)

RGB LED Ex:

```
potValue = analogRead(potPin)

map(potValue, 0, 4095, 0, 255)
```

SECTION II

# Review: ESP32 Board Setup

# ESP32 and Arduino IDE Setup

https://ieee.ics.uci.edu/ops/esp32_guide.html

SECTION III

# Review: ESP32 Functions

# Digital Pin Functions

- `digitalWrite(int pin, int value)`
  - **Sets the voltage** at the output pin to either a `HIGH` (3.3V) or `LOW` (0V) value
  - Analogy - light switch and light bulb:
    - Like toggling a switch on and off
- `digitalRead(int pin)`
  - **Reads the voltage** at the input pin, returning `HIGH` (3.3V) or `LOW` (0V) as an integer (1 or 0)

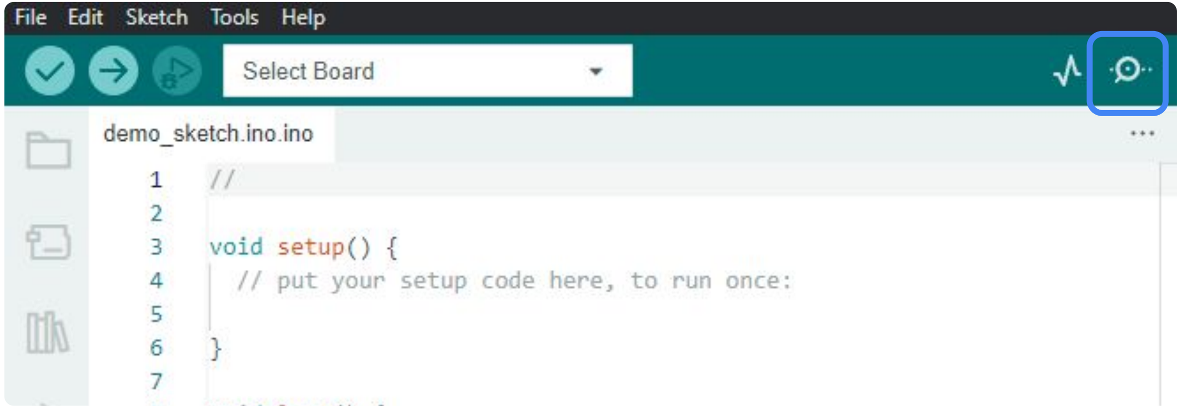
# Analog Pin Functions

- `analogWrite(int pin, int value)`
  - **Sets the average voltage** on digital output pin to a value in the **range** `0-255` (0V to 3.3V)
  - Analogy - light dimmer:
    - You use the slide to set the bulb to anywhere *between* MAX brightness or MIN brightness
- `analogRead(int pin)`
  - **Reads the voltage** at the input pin, maps it to a value in the **range** `0-4095` (0V to 3.3V) and returns that value
  - Use the aliases `A0, A1, A2`… for the pin number

# More Basic Functions

- `delay(int ms)`
  - **Pauses the program execution** by `ms` milliseconds
- `Serial.print("Message")`
  - Sends a string to the computer connected via USB and **displays the string on the Serial Monitor** in the IDE
- `Serial.println("Message")`
  - Sends a string to the computer connected via USB and **displays your string on the Serial Monitor** in the IDE, **followed by a newline**

# Debugging w/ the Serial Monitor (Cont’d)

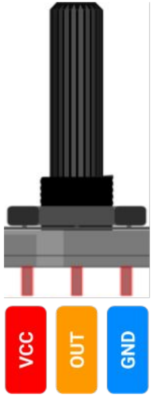


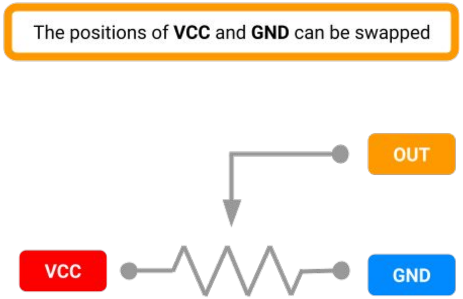
```
//

void setup() {
  // put your setup code here, to run once:

}

```

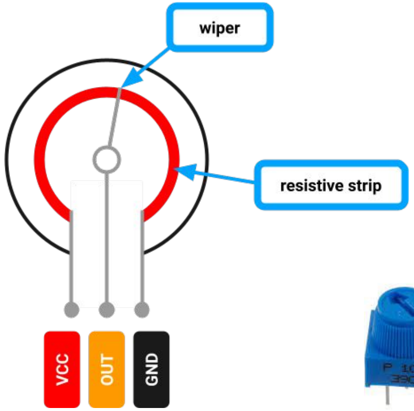
- In the absence of a debugger (the ESP32 is not capable of using one), `Serial.print` is an excellent tool to **help debug programs**
  - Print values to track across parts of your program
    - Unexpected values displayed to the Serial Monitor indicates an error

# What is a Potentiometer?

- A potentiometer is a *variable* resistor with 3 terminals: VCC, OUT, and GND
- We will use it as a voltage divider to only output a fraction of the supply voltage
  - This output pin voltage varies between the VCC and GND pin voltages based on the dial’s position
- **Disclaimer: Don’t turn the wiper too far past its limit (the knob is fragile and can break easily if turned too far)**

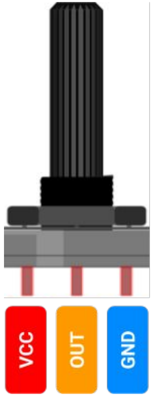



The positions of **VCC** and **GND** can be swapped

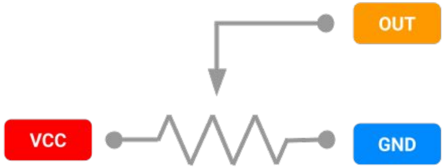


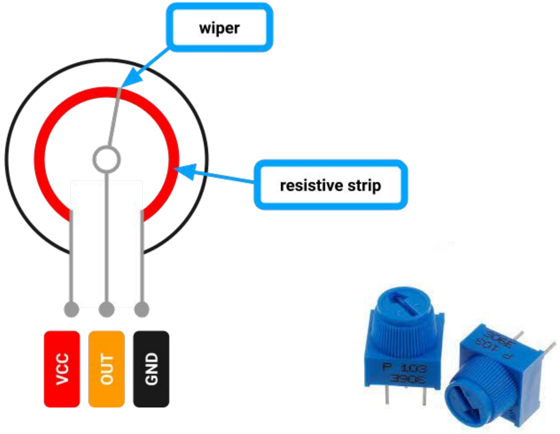
# What is a Potentiometer? (cont.)

- A potentiometer has many different applications, such as:
  - Volume control
  - Light dimming
  - Tuning and calibration
- **Trivia questions!**
  - If the wiper is all the way to the left, what is the voltage at the OUT pin?
  - If the wiper is all the way to the left, what would the value in Arduino IDE be between 0-4095?



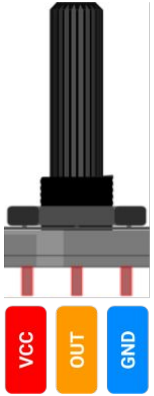
The positions of **VCC** and **GND** can be swapped



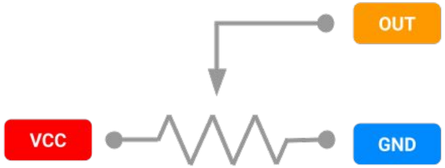


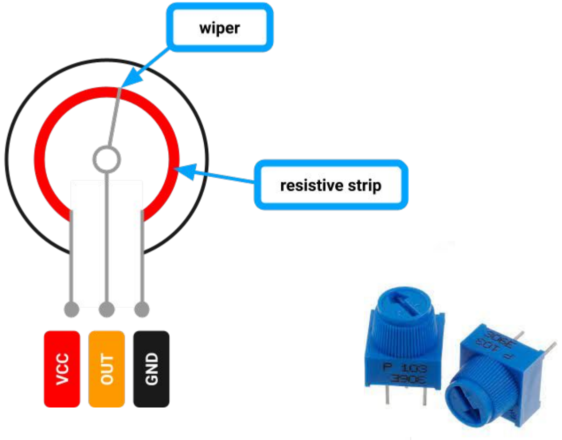
# What is a Potentiometer? (cont.)

- A potentiometer has many different applications, such as:
  - Volume control
  - Light dimming
  - Tuning and calibration
- **Trivia questions!**
  - If the wiper is all the way to the left, what is the voltage at the OUT pin?
  - Answer: **3.3V**
  - If the wiper is all the way to the left, what would the value in Arduino IDE be between 0-4095?
  - Answer: **4095**



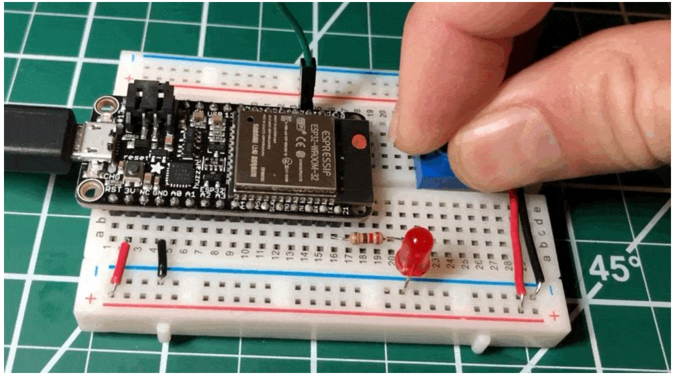
The positions of **VCC** and **GND** can be swapped



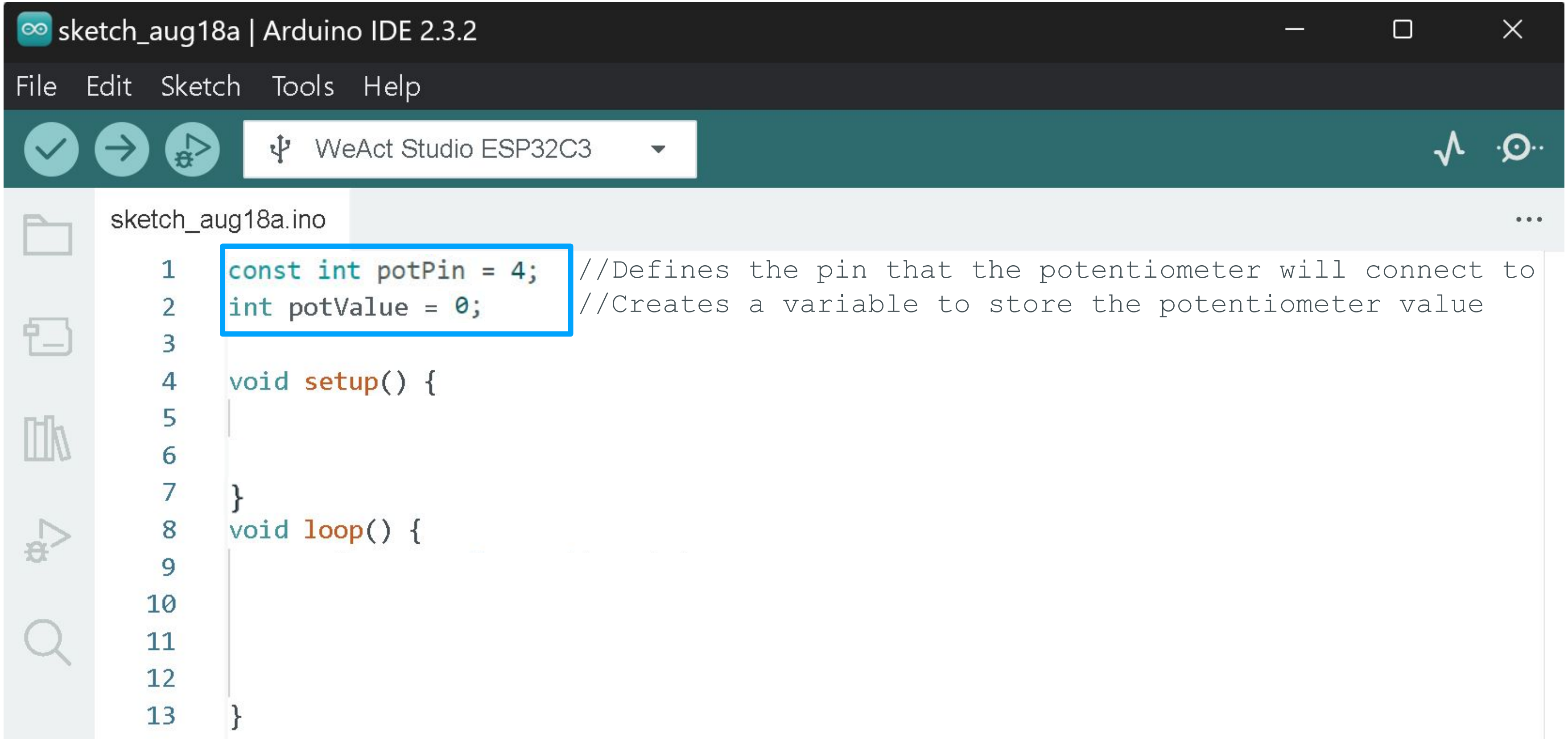


# Using Potentiometers with ESP32

- We can't just use digitalRead() to read values in between 0 - 3.3V off our potentiometer
- Instead, we'll be using `analogRead(int pin)`
    - The analog pin is wired to the ESP32's **analog-to-digital converter (ADC)**
        - Translates the analog signal to a discrete digital signal
- Now, let's look at some code showing this in action!



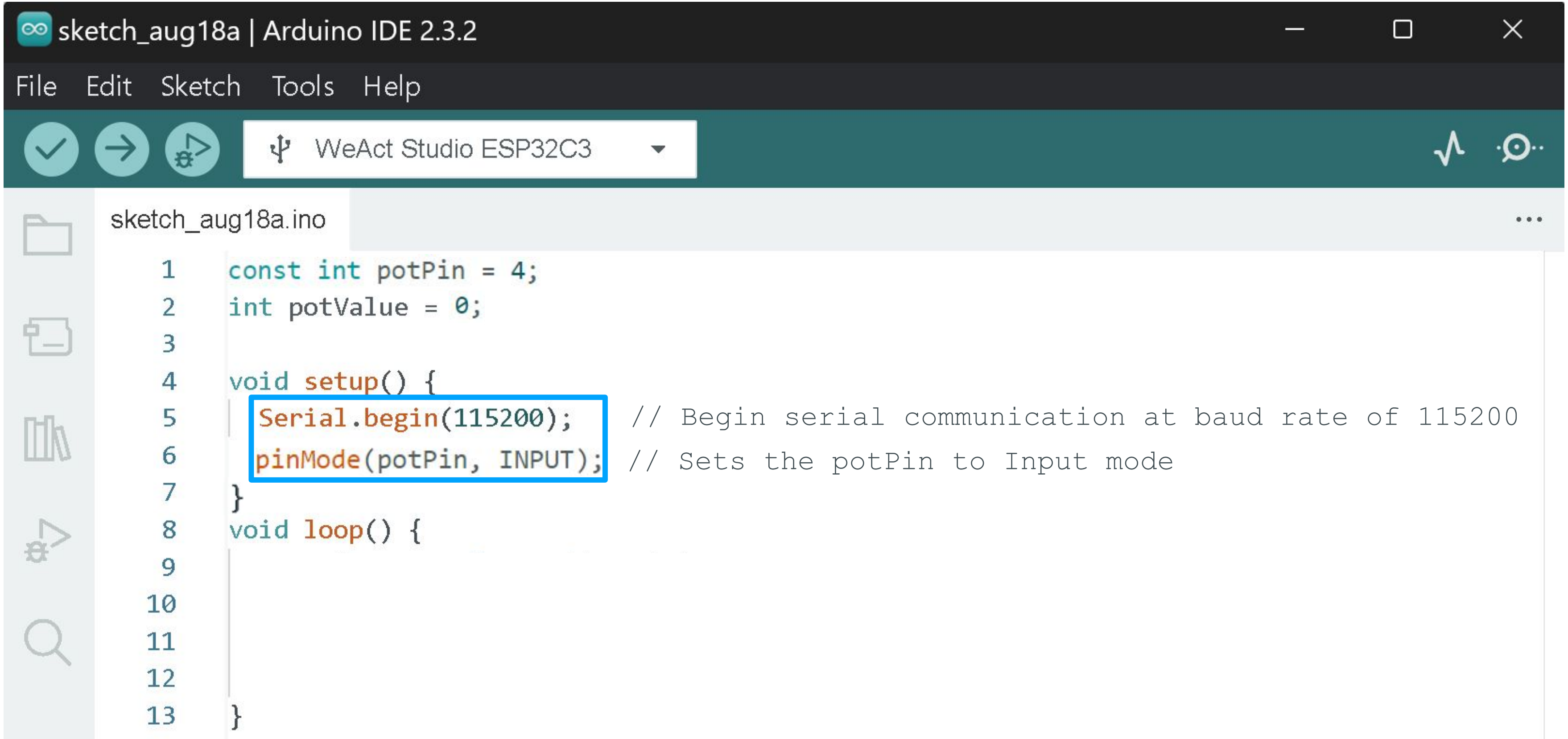
# Potentiometer Code Example



sketch_aug18a | Arduino IDE 2.3.2
File Edit Sketch Tools Help
WeAct Studio ESP32C3
sketch_aug18a.ino

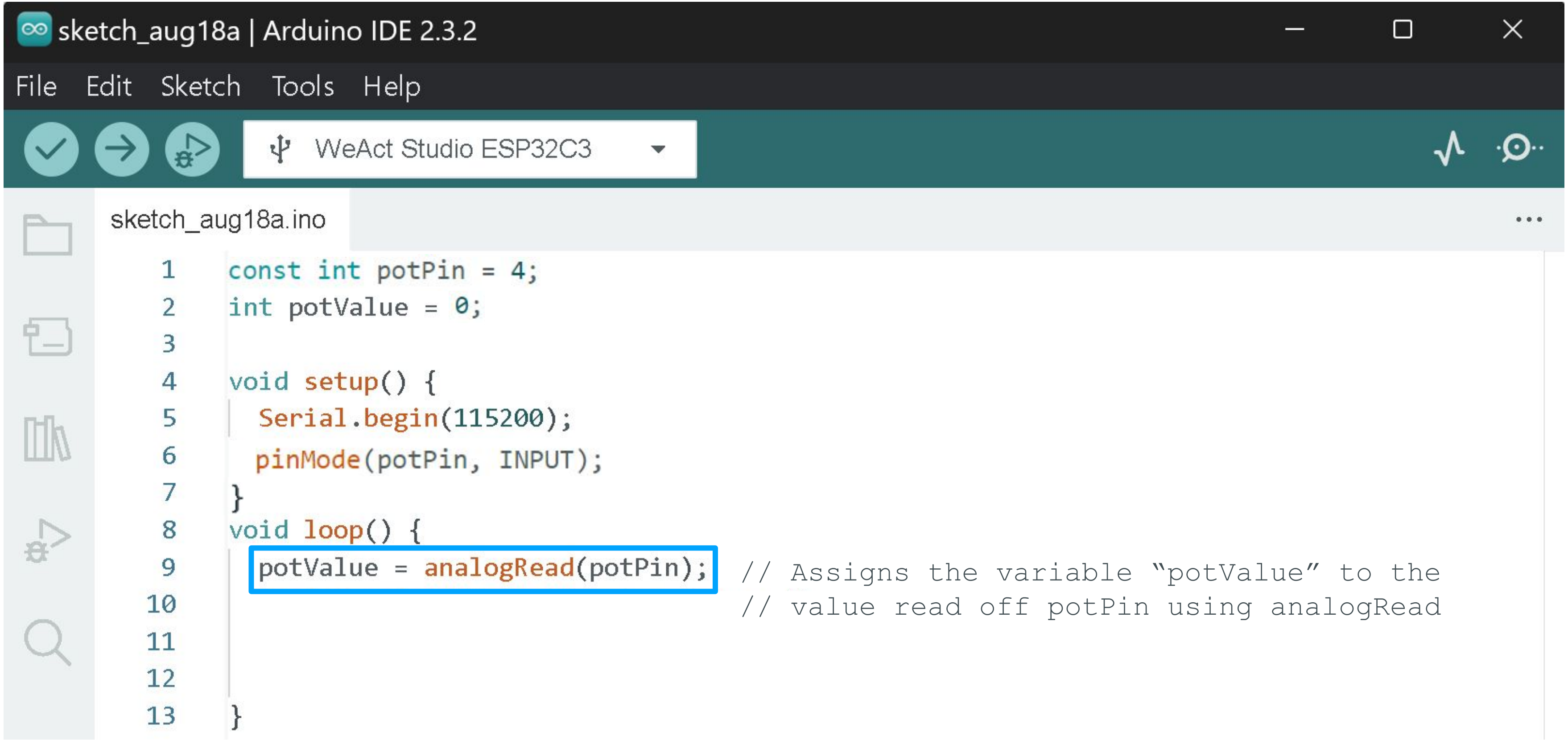
```
const int potPin = 4;  //Defines the pin that the potentiometer will connect to
int potValue = 0;      //Creates a variable to store the potentiometer value

void setup() {


}
void loop() {




}
```

# Potentiometer Code Example



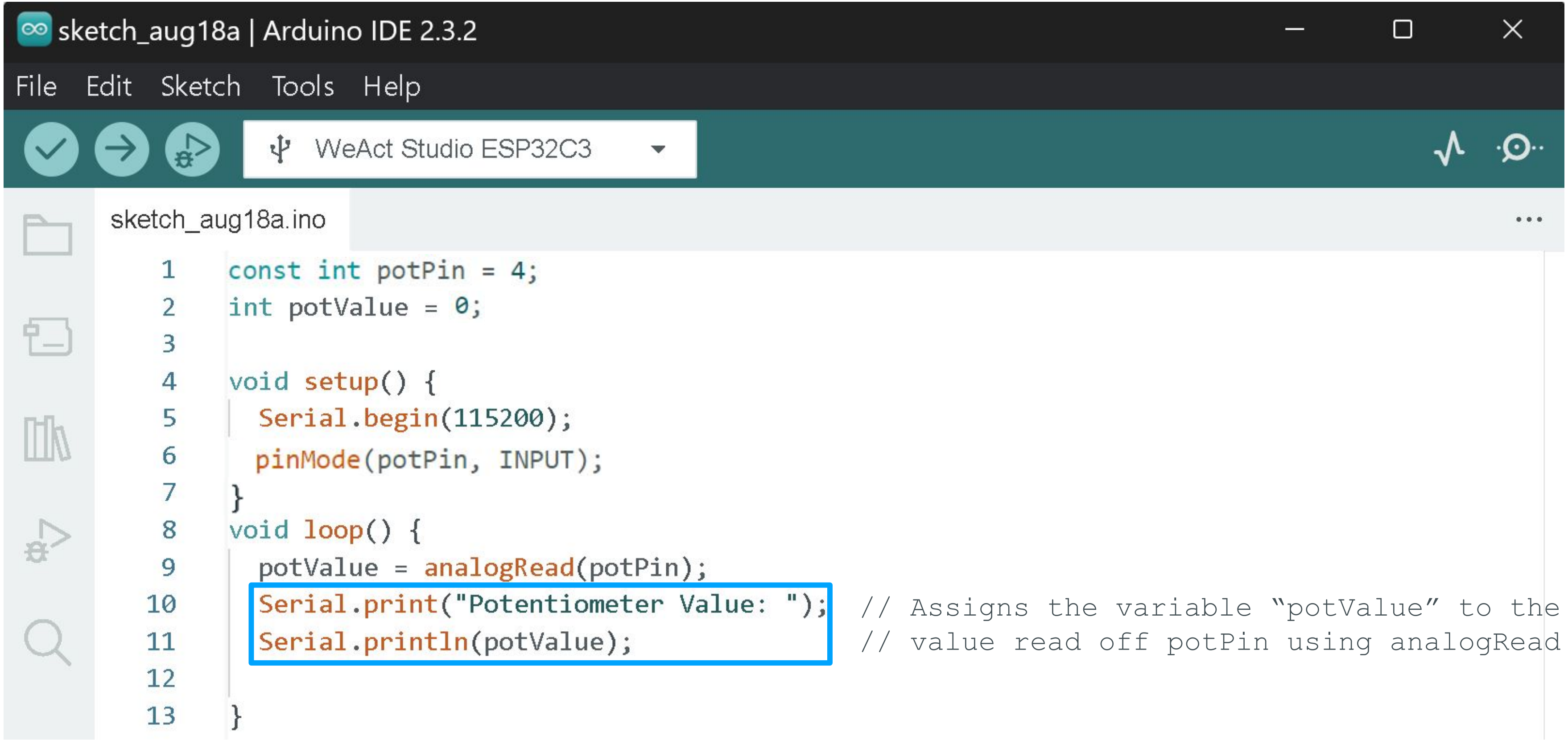
```
const int potPin = 4;
int potValue = 0;

void setup() {
  Serial.begin(115200);   // Begin serial communication at baud rate of 115200
 pinMode(potPin, INPUT);  // Sets the potPin to Input mode
}
void loop() {




}
```

# Potentiometer Code Example



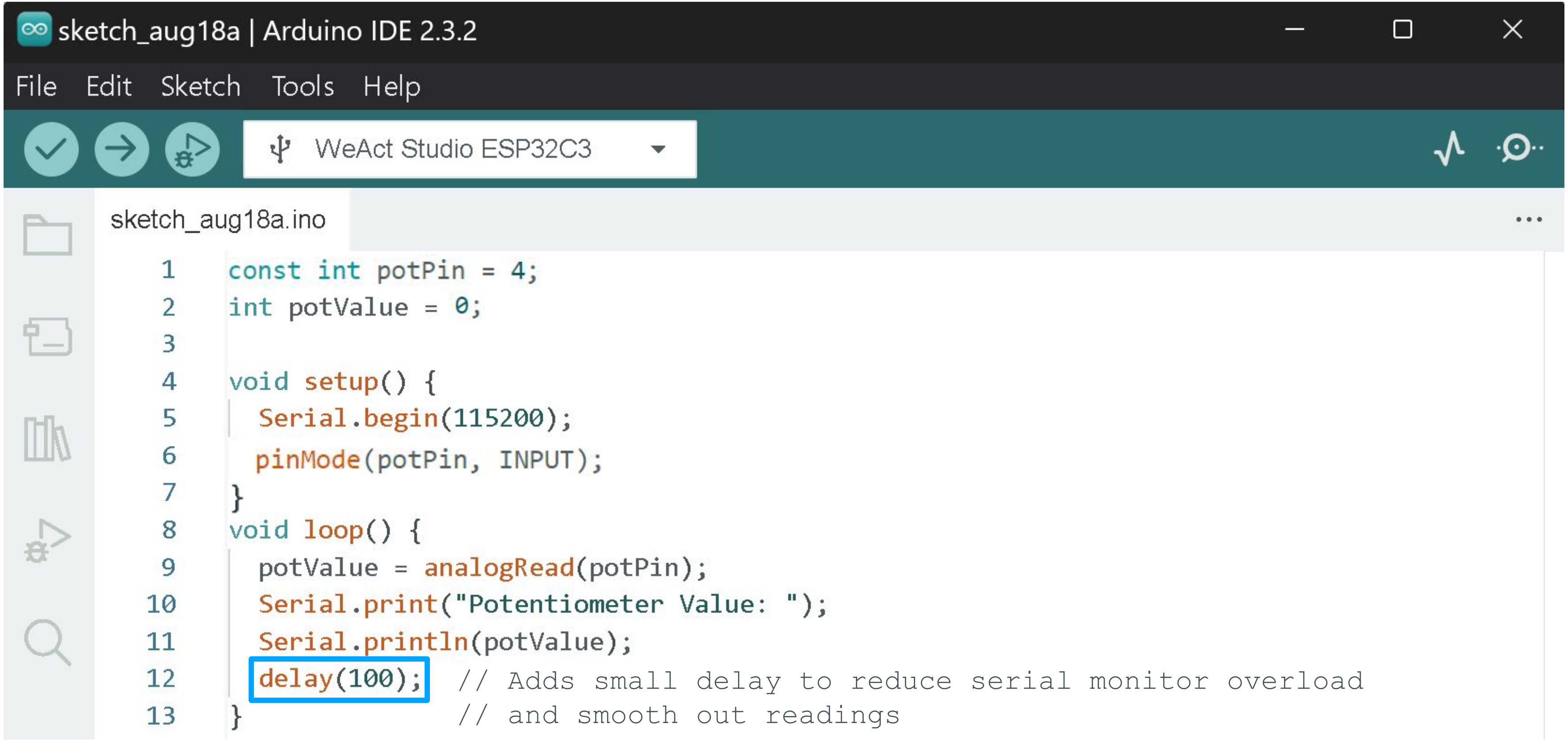
```
const int potPin = 4;
int potValue = 0;

void setup() {
  Serial.begin(115200);
  pinMode(potPin, INPUT);
}
void loop() {
  potValue = analogRead(potPin);   // Assigns the variable “potValue” to the
                                   // value read off potPin using analogRead



}
```

# Potentiometer Code Example



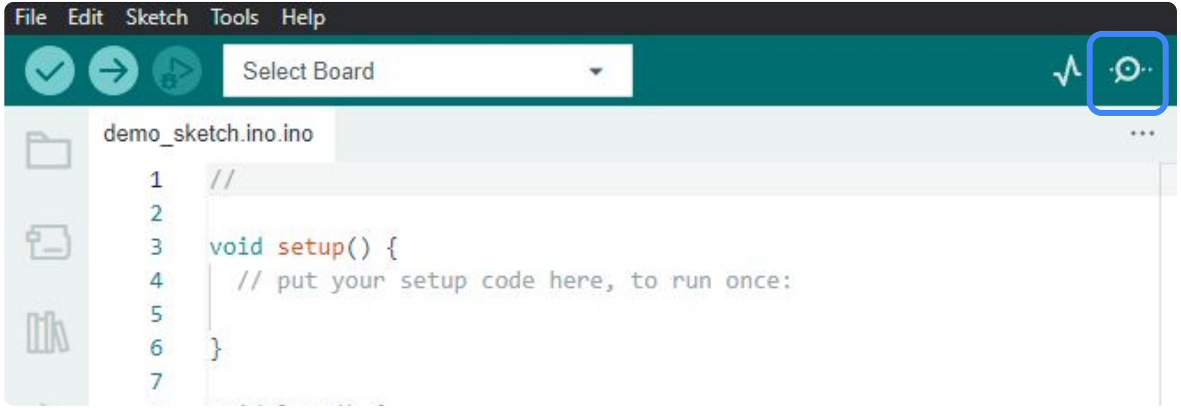
```
const int potPin = 4;
int potValue = 0;

void setup() {
  Serial.begin(115200);
  pinMode(potPin, INPUT);
}
void loop() {
  potValue = analogRead(potPin);
  Serial.print("Potentiometer Value: ");  // Assigns the variable “potValue” to the
  Serial.println(potValue);               // value read off potPin using analogRead

}
```

# Potentiometer Code Example



```
const int potPin = 4;
int potValue = 0;

void setup() {
  Serial.begin(115200);
  pinMode(potPin, INPUT);
}
void loop() {
  potValue = analogRead(potPin);
  Serial.print("Potentiometer Value: ");
  Serial.println(potValue);
  delay(100);   // Adds small delay to reduce serial monitor overload
}               // and smooth out readings
```

# Using the Serial Monitor

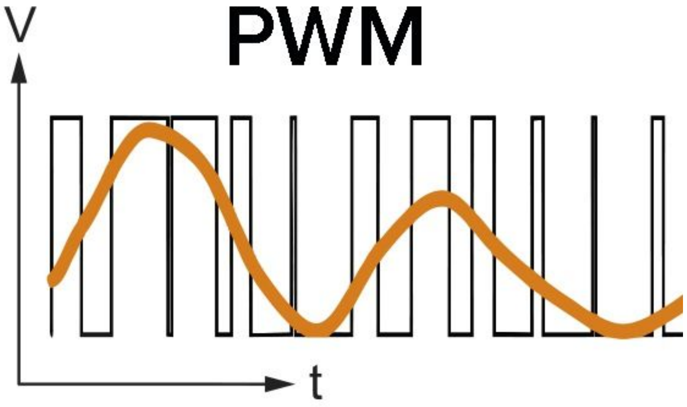


```
//

void setup() {
  // put your setup code here, to run once:

}

```

- While the ESP32 board is connected to the personal computer via USB, select **Serial Monitor** (the **magnifying glass** icon) in the IDE
  - A pane will appear at the bottom of the IDE window which displays all data sent by the ESP32 board using `Serial.print`

# Using Potentiometers with ESP32

- digitalWrite() can only set a pin’s voltage to HIGH or LOW, nothing in between!
- We need to use a new function, `analogWrite(int pin, int value)`
  - With **pulse width modulation (PWM) waves**, we can generate an average voltage anywhere between 0V and 3.3V
  - `int pin` - Reference a specific pin to use
  - `int value` - Any value between 0 and 255 (Inputting 0 outputs 0V, and 127 outputs 1.65V, 255 outputs 3.3V, etc.)



# FAIR USE DISCLAIMER

Copyright Disclaimer under section 107 of the Copyright Act 1976, allowance is made for “fair use” for purposes such as criticism, comment, news reporting, teaching, scholarship, education and research.

Fair use is a use permitted by copyright statute that might otherwise be infringing.

Non-profit, educational or personal use tips the balance in favor of fair use.

# CC BY-NC-SA 4.0