LECTURE III

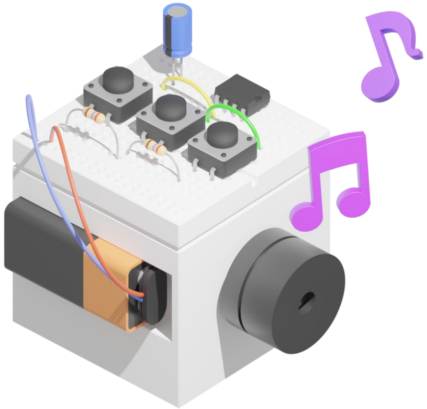
Introduction to Embedded Systems and Microcontrollers

SECTION I

Project II Review

Project 2 Review

- Official Due Date: Friday, November 15th by 11:59PM
- Ask questions and seek help (online or @Lab Hours), so you can finish it on time!
- Note: To get the $20 refund for Fall Quarter, you must complete all Projects by the end of Fall Quarter



Learning Concepts:

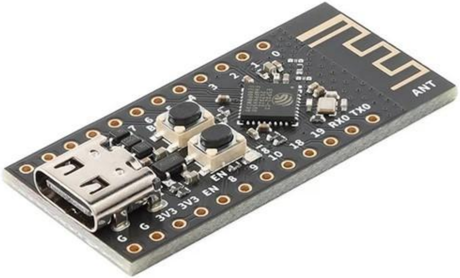
- Circuit Analysis (Nodes, Voltage Drop)
- Circuit Troubleshooting
- Multimeters
- 555 Timer
- Breadboarding
- Soldering

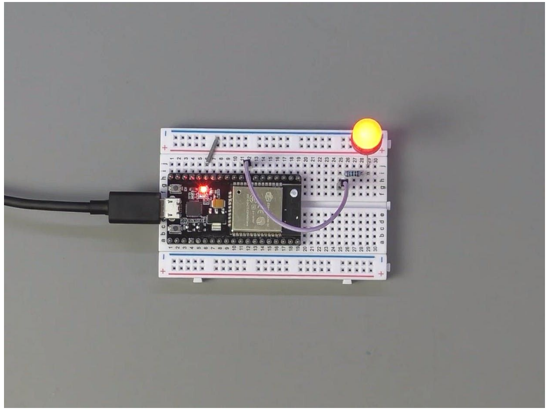
SECTION II

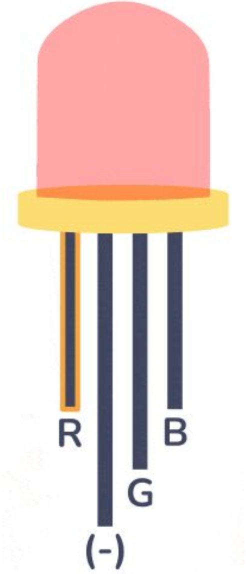
Project III

Project 3 Overview

- Build and program a dimmable RGB LED using the **ESP32** and **potentiometers**.







You will learn:

- Embedded Systems
- Microcontrollers
- ESP32
- Arduino IDE
- Pulse Width Modulation (PWM)

SECTION III

What is an Embedded System?

Embedded Systems

- An **embedded system** is a combination of hardware and software designed for a specific purpose
  - Ex) alarm clock, camera, or MP3 player

  - Contrasts from a **general-purpose system**, like a smartphone or laptop
    - These devices can act as an alarm, camera, and a media player *combined*
    - They typically have much more functionality than an embedded system

Embedded Systems

- Large-scale mechanical and electrical systems often consist of **multiple, smaller embedded systems**
  - Each embedded system has a function that supports the larger system
  - Ex) **Airplanes** - in-flight entertainment system, temperature control, speed control, flight management, flight data recorder
- OPS Projects 3-7 and the Capstone project are all embedded systems

SECTION IV

Microcontrollers & ESP32

Microcontrollers

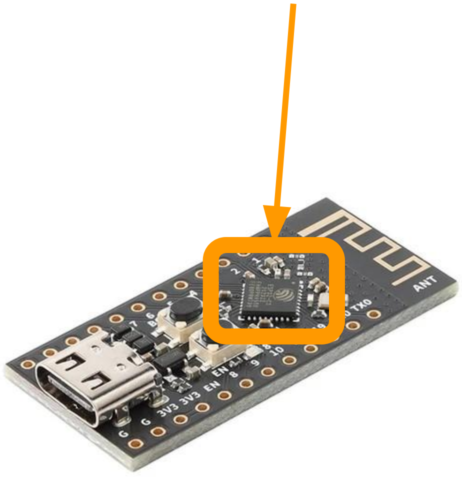
- A **microcontroller** (or **MCU**) is a **small computer** on a single integrated circuit
  - Complete with functionality to run and store programs
- An **architecture** defines the organization of hardware within the computer
  - **MCUs have different architectures**
    - ex. ESP32, ATmega, etc.

ESP32 Microcontroller Family

ESP32-C3Fx4 Board

- The ESP32-C3Fx4 board in your OPS kits was developed by WeAct Studios
- The ESP32 board is **hobbyist-friendly**
    - Wi-Fi and Bluetooth capabilities
    - ESP32 chip itself contains the processor, RAM, and flash storage
    - In OPS, we will refer to the **ESP32-C3Fx4 board as an ESP32** from now on for convenience

The MCU is **an SOIC** soldered to the board



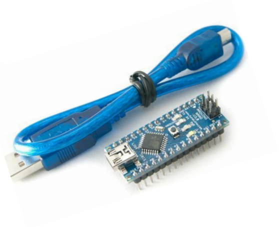
ESP32-C3Fx4

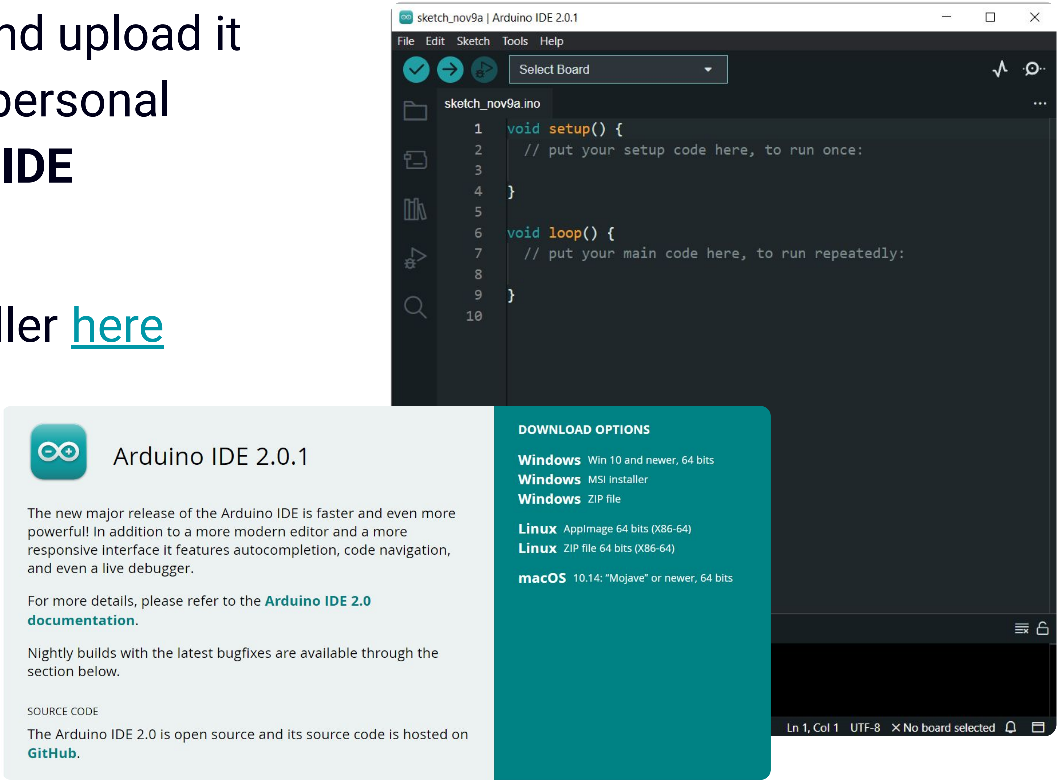
SECTION V

Arduino IDE

Arduino IDE

- We will write code, compile, and upload it to the ESP32 board from our personal computers using the **Arduino IDE** software kit
- Download the latest IDE installer here





sketch_nov9a | Arduino IDE 2.0.1
File Edit Sketch Tools Help
Select Board
sketch_nov9a.ino
Ln 1, Col 1 UTF-8 ✕ No board selected
DOWNLOAD OPTIONS
Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file
Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)
macOS 10.14: "Mojave" or newer, 64 bits

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

}

void loop() {
  // put your main code here, to run repeatedly:

}

```

Arduino IDE 2.0.1

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the **Arduino IDE 2.0 documentation**.

Nightly builds with the latest bugfixes are available through the section below.

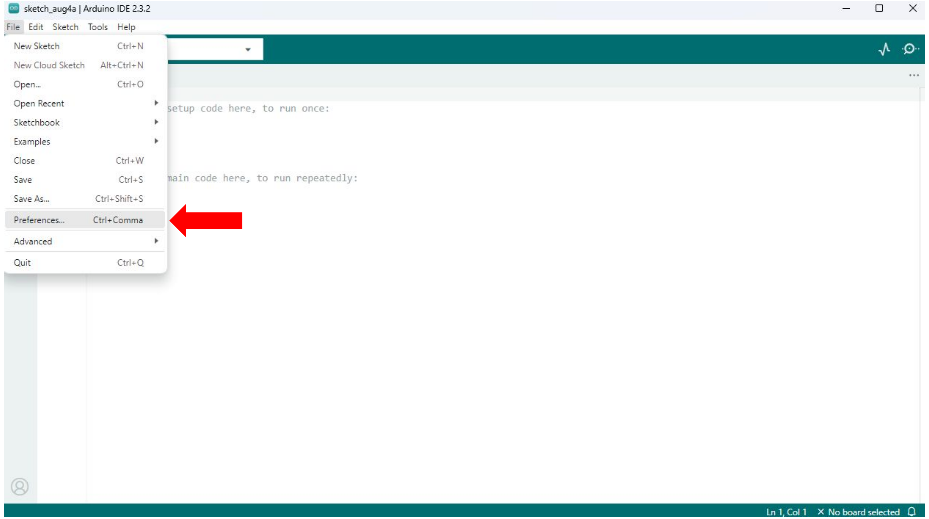
SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on **GitHub**.

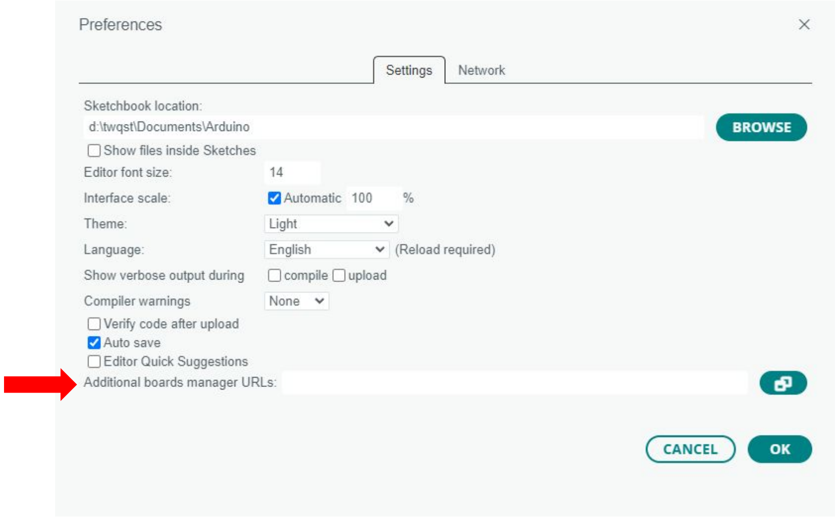
Arduino IDE Setup for ESP32 (cont.)

Step 2: Once it's installed, open Arduino IDE

- Go to File -> Preferences -> paste in this link under "Additional Boards Manager URLS", then click OK: https://dl.espressif.com/dl/package_esp32_index.json


sketch_aug4a | Arduino IDE 2.3.2
File Edit Sketch Tools Help
New Sketch Ctrl+N
New Cloud Sketch Alt+Ctrl+N
Open... Ctrl+O
Open Recent
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Preferences... Ctrl+Comma
Advanced
Quit Ctrl+Q
Ln 1, Col 1 × No board selected

```
setup code here, to run once:

main code here, to run repeatedly:
```



Sketchbook location:
d:\twqst\Documents\Arduino

☐ Show files inside Sketches

Editor font size: 14

Interface scale: ☑ Automatic 100 %

Theme: Light

Language: English (Reload required)

Show verbose output during ☐ compile ☐ upload

Compiler warnings None

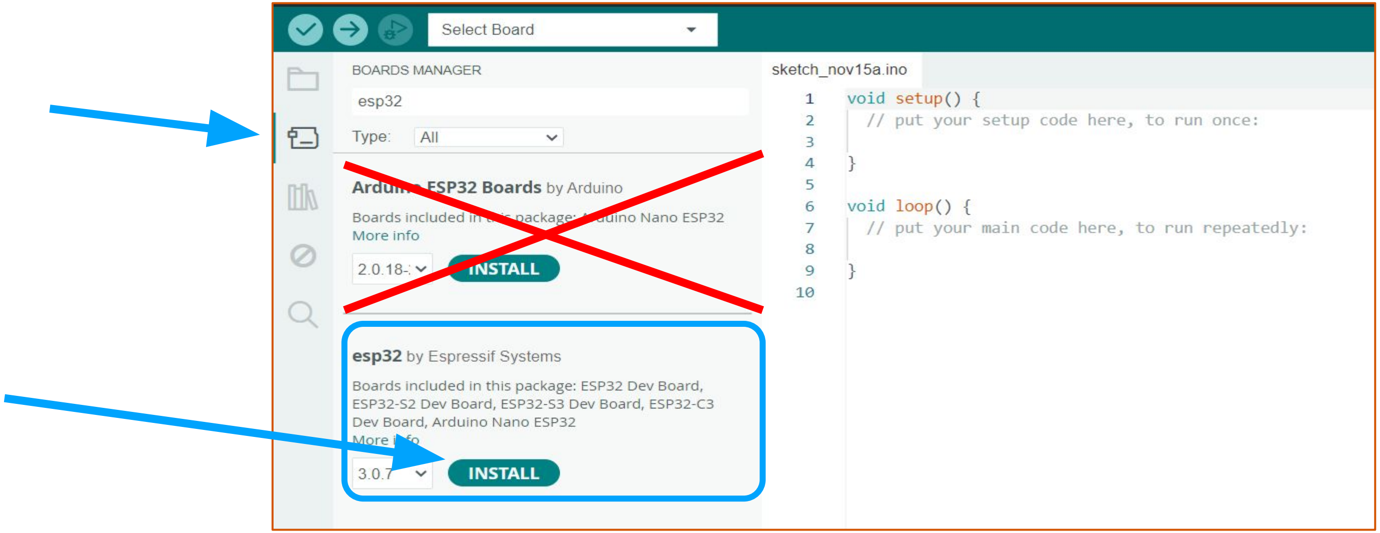
☐ Verify code after upload

☑ Auto save

☐ Editor Quick Suggestions

Additional boards manager URLs:

Download ESP32 Board library



BOARDS MANAGER

esp32

Type: All

Arduino ESP32 Boards by Arduino

Boards included in this package: Arduino Nano ESP32
More info

2.0.18-: INSTALL

esp32 by Espressif Systems

Boards included in this package: ESP32 Dev Board, ESP32-S2 Dev Board, ESP32-S3 Dev Board, ESP32-C3 Dev Board, Arduino Nano ESP32
More info

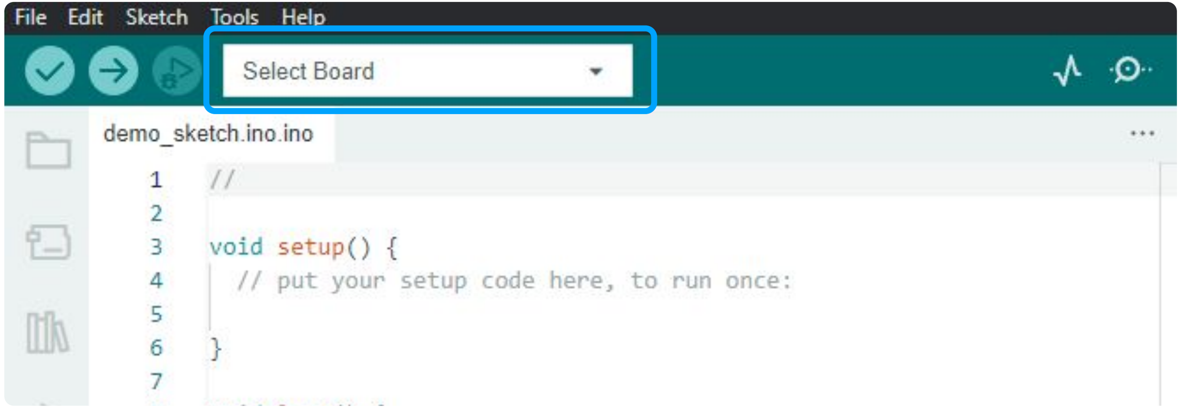
3.0.7 INSTALL

sketch_nov15a.ino

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

}

void loop() {
  // put your main code here, to run repeatedly:

}

```

- Click on the board manager tab on the right hand side
- Type in “esp32” into the search bar
- Install the library made by “Espressif Systems”

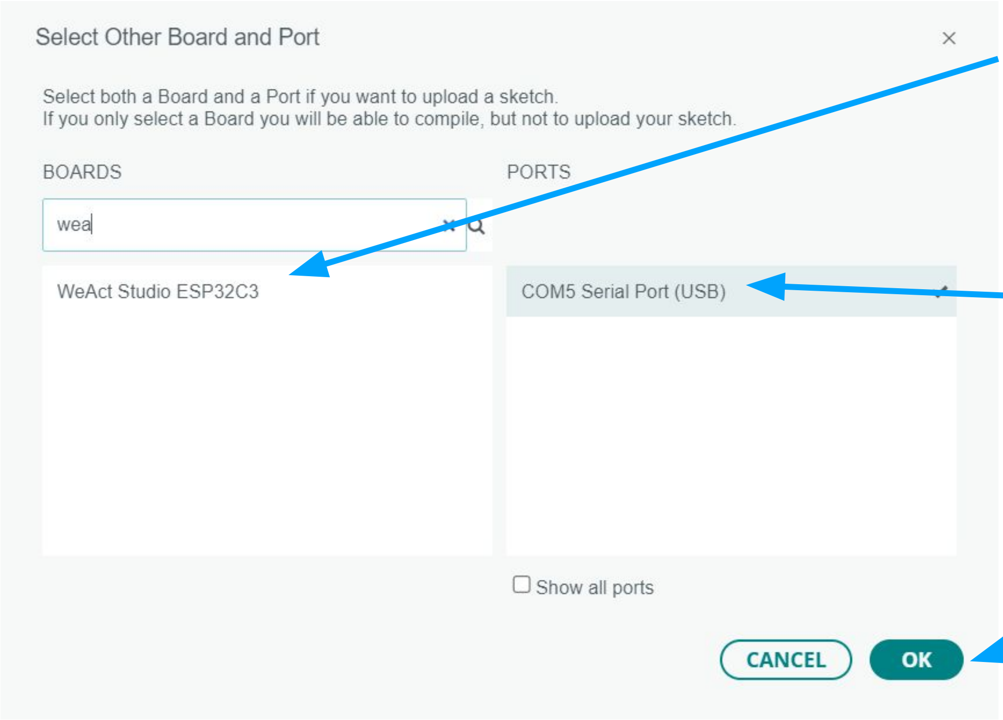
Setting the Target Board



```
//

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

}

```

- Before we can upload a sketch, we must **connect the ESP32 board via the USB-C to USB-A cable to the computer** and configure the IDE to the correct board and USB port
- Open the **Select Board** dropdown and select an available ESP32 board
  - If no option appears, click **Select other board and port…**

Setting the Target Board (Cont'd)



Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

BOARDS: wea

WeAct Studio ESP32C3

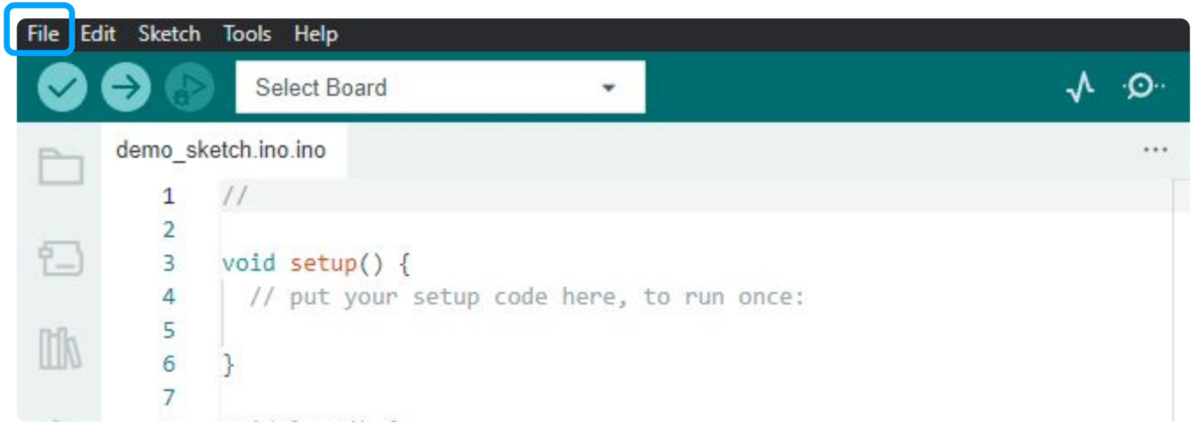
PORTS

COM5 Serial Port (USB) ✓

□ Show all ports

- In the popup window, search **Boards** and select **WeAct Studio ESP32C3**
- From the available items under **Port**, select the USB port to which the board is currently connected
- Confirm the selections by clicking **OK**

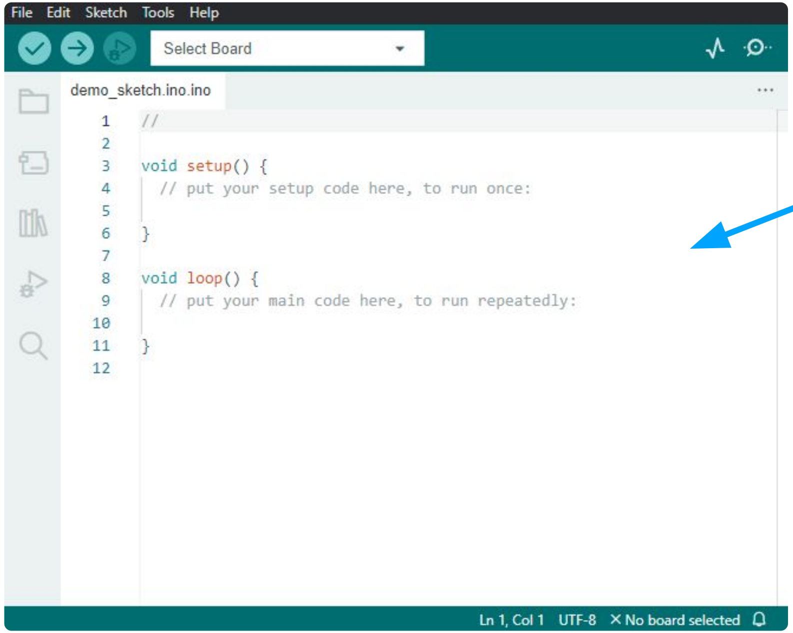
Creating a Sketch



```
//

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

}

```

- An Arduino source code file is called a **sketch** and has the **.ino** extension
- When the IDE opens, a new sketch will be created automatically or a previous sketch will open
    - To create a new sketch, select **File → New Sketch** from the menu

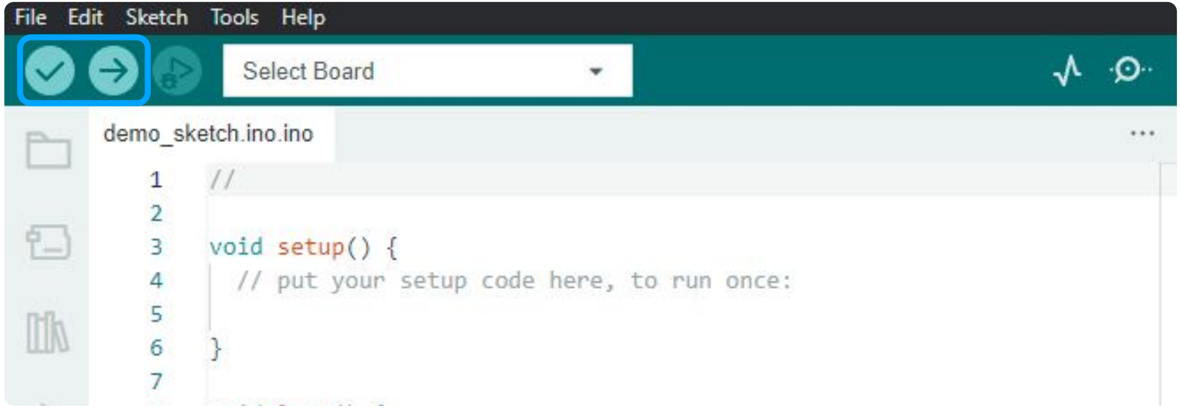
Editing a Sketch



```
//

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

}

void loop() {
  // put your main code here, to run repeatedly:

}

```

- Sketches can be modified from the **code editor**, which appears just below the menu
- New sketches opened in the editor come with **template code**

Verifying and Uploading a Sketch



```
//

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

}

```

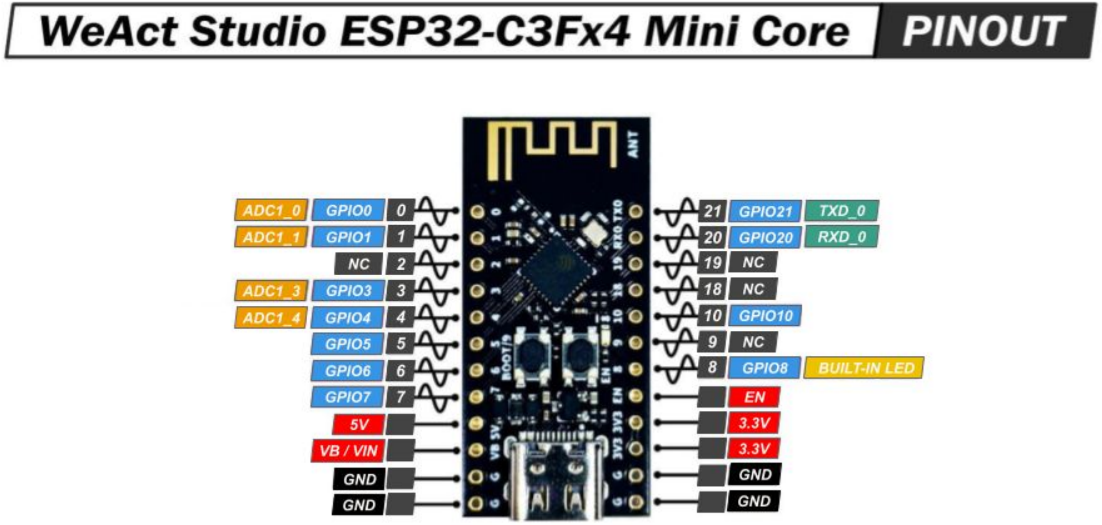
- Before uploading the sketch, select **Verify** (the **checkmark** icon) to compile the sketch
- Once the sketch compiles successfully, select **Upload** (the **right arrow** icon) to upload the compiled sketch to the Arduino board

SECTION VI

Basic ESP32 Pins

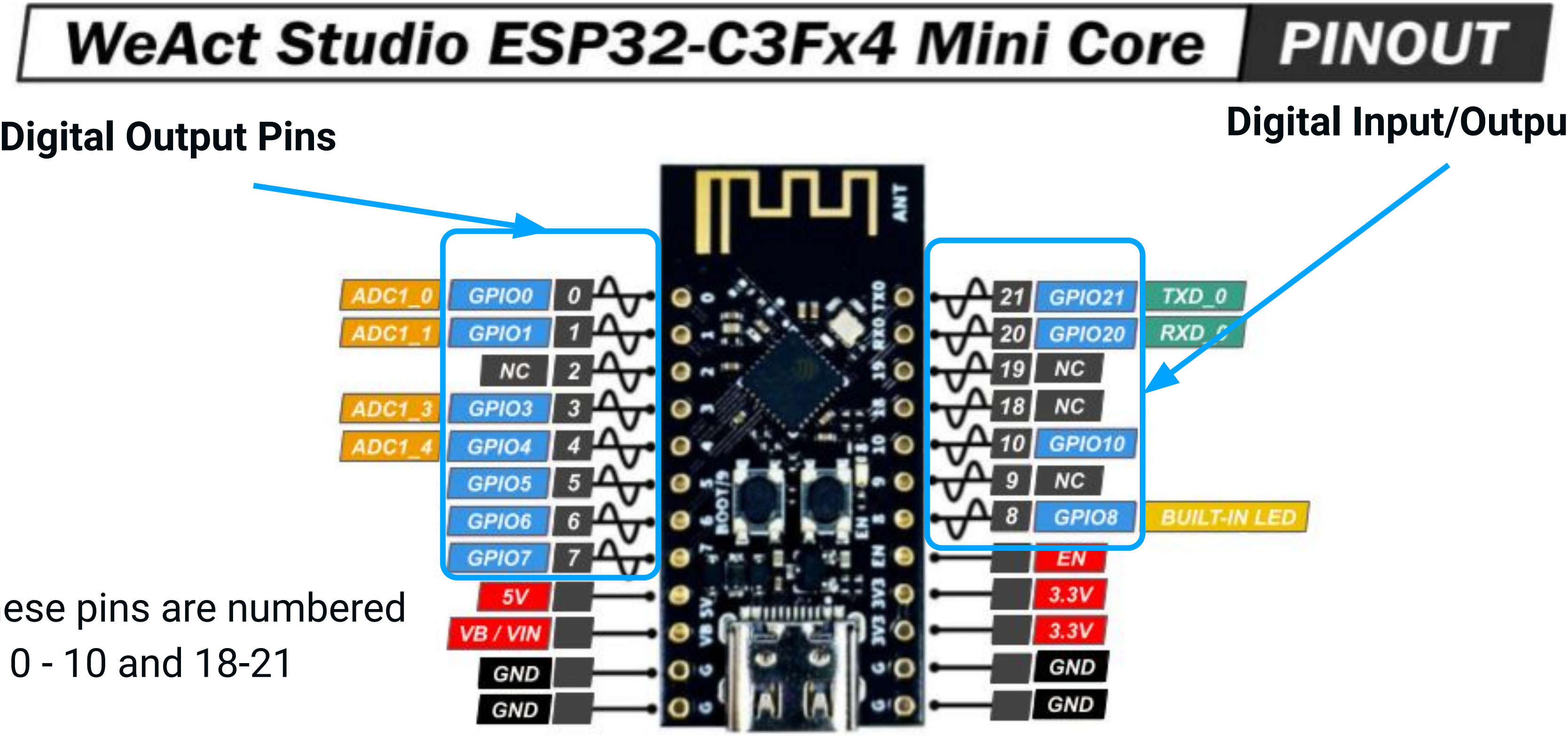
ESP32 Pinout Diagram

- A pinout diagram is included for the ESP32
- **General Purpose Input/Output** (**GPIO**) pins are pins that can be set as either INPUT or OUTPUT.



  - These are the **analog** and **digital pins** marked by the pinout diagram
  - We will design an Arduino sketch to control these pins

Digital Pins



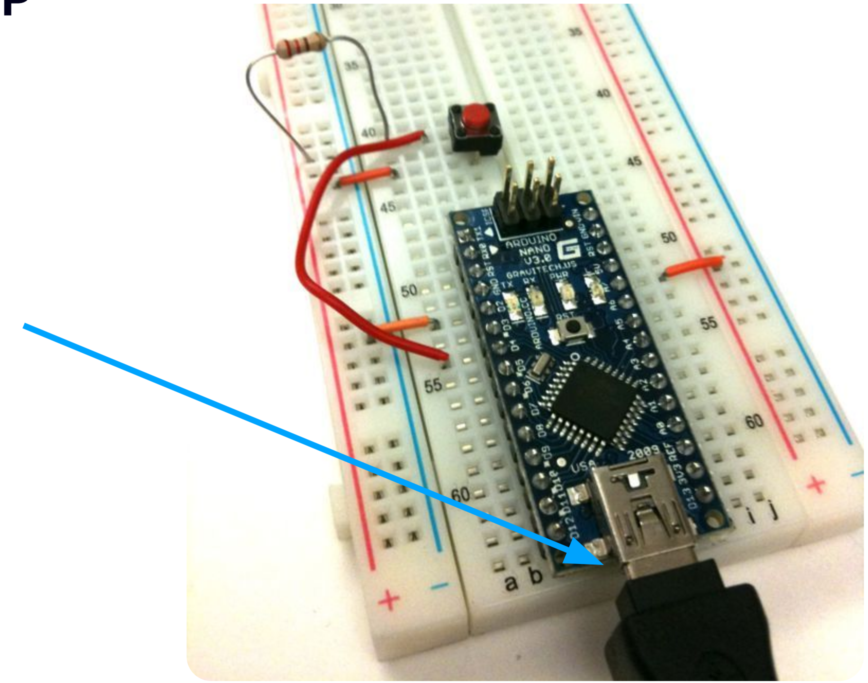
Digital Output Pins

Digital Input/Output Pins

These pins are numbered as 0 - 10 and 18-21

Prototyping with the ESP32

- The ESP32 can be **seated along the DIP channel** of a standard breadboard just like a DIP IC
- The **ESP32's USB port should be oriented away from the board**, so the connected cable doesn't obstruct the breadboard
- Circuits that interface with the ESP32 board must **share a common ground with the GND pin of the board**



SECTION VII

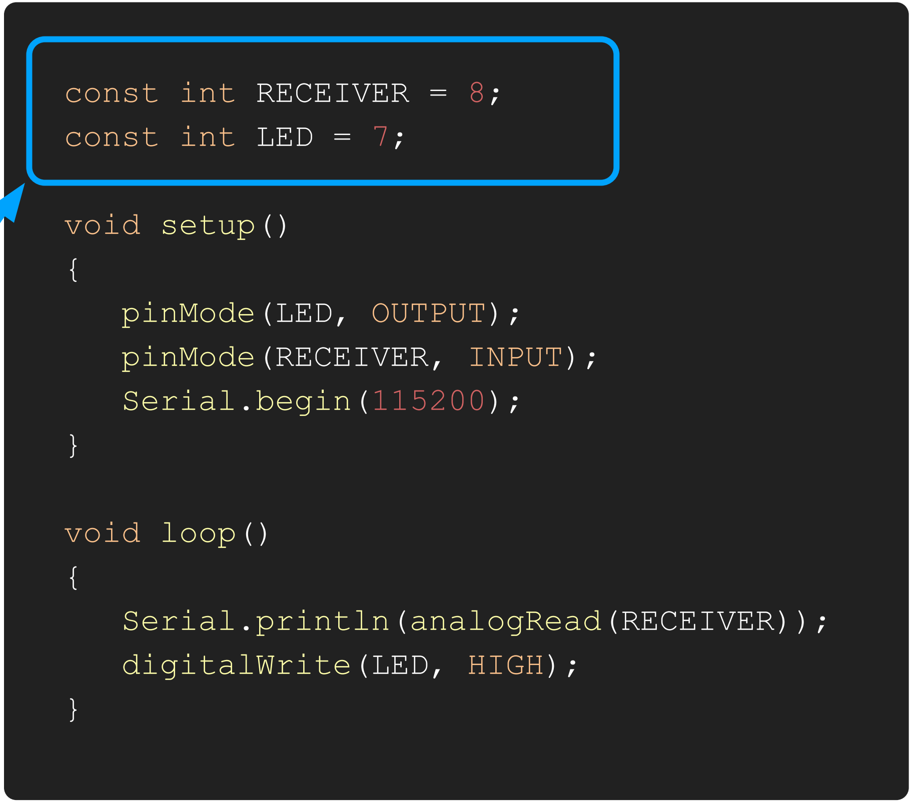
Sketch Structure

Sketch Structure

- Sketches are written in the **Arduino language**, which is much like C++
- **Assign pins as global variables** at the beginning of the sketch:

  `const int pinName = pin#;`

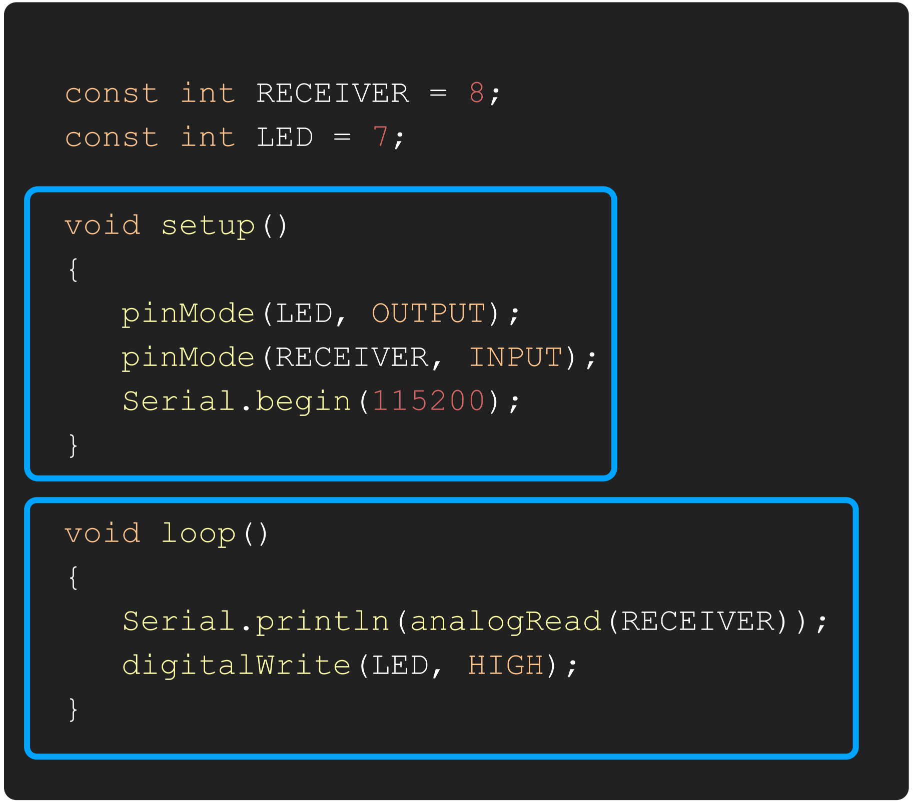
  - Each GPIO pin is associated with an integer
- Don’t place the global variables inside functions (otherwise it won’t be a global variable!)



```
const int RECEIVER = 8;
const int LED = 7;

void setup()
{
    pinMode(LED, OUTPUT);
    pinMode(RECEIVER, INPUT);
    Serial.begin(115200);
}

void loop()
{
    Serial.println(analogRead(RECEIVER));
    digitalWrite(LED, HIGH);
}
```

Sketch Structure (Cont'd)

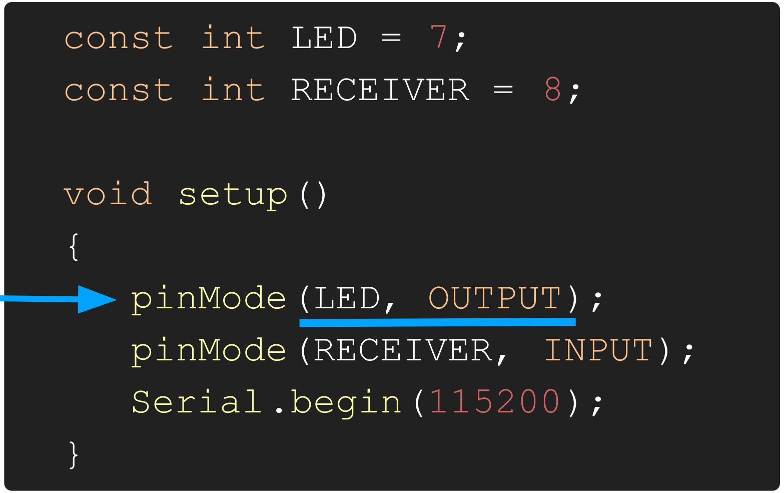
- Just like the C++ **`main`** function, Arduino IDE has built-in functions **`setup`** and **`loop`**, which must be defined by the programmer
- **`setup()`** runs once at the beginning of program execution
  - Used to initialize pin modes
- **`loop()`** runs repeatedly in an infinite loop



```
const int RECEIVER = 8;
const int LED = 7;

void setup()
{
   pinMode(LED, OUTPUT);
   pinMode(RECEIVER, INPUT);
   Serial.begin(115200);
}

void loop()
{
   Serial.println(analogRead(RECEIVER));
   digitalWrite(LED, HIGH);
}
```

Things to do in `setup`

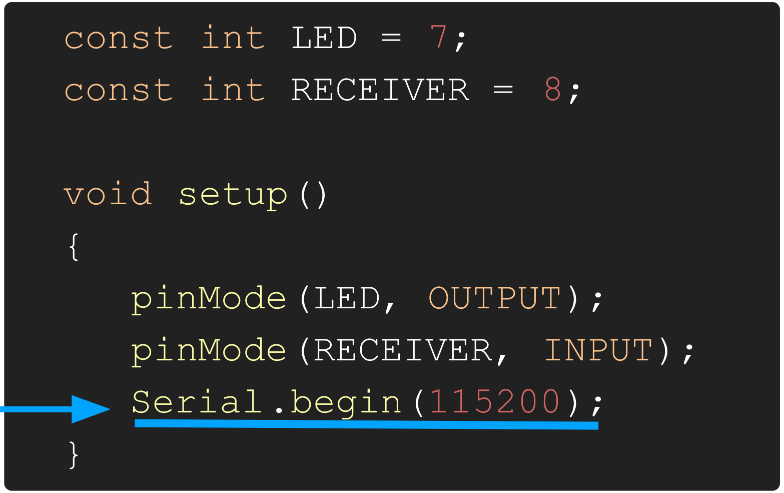
- `pinMode(int pin, int mode)`
    - Call this function for each digital pin you want to use
    - Pass the global variable assigned to each pin to the `pin` parameter
    - Specify if your pin is **`INPUT`** or **`OUTPUT`** mode
        - We **read data from input pins** and **send data to output pins**



```
const int LED = 7;
const int RECEIVER = 8;

void setup()
{
   pinMode(LED, OUTPUT);
   pinMode(RECEIVER, INPUT);
   Serial.begin(115200);
}
```

Things to do in `setup` (Cont'd)

- `Serial.begin(115200)`
  - `Serial` is an object that facilitates communication between the ESP32 board and the computer via USB
  - Used to print statements to the computer
  - Pass **`115200`** bits per second as the argument for this member function
    - Known as the **baud rate**



```
const int LED = 7;
const int RECEIVER = 8;

void setup()
{
    pinMode(LED, OUTPUT);
    pinMode(RECEIVER, INPUT);
    Serial.begin(115200);
}
```

SECTION VIII

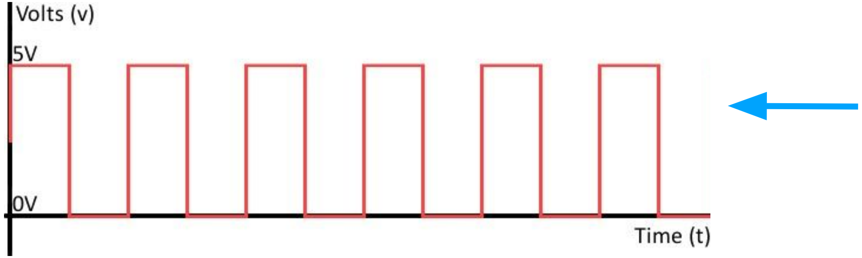
Arduino IDE Functions

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
- `digitalRead(int pin)`
  - **Reads the voltage** at the input pin, returning `HIGH` (3.3V) or `LOW` (0V) as an integer (1 or 0)
- Analogy - light switch and light bulb:
  - You use the switch to set the bulb to either MAX brightness or MIN brightness

Digital Signals

- **Computers (and microcontrollers) transfer data** across wires/lines as **digital** (discrete) **voltage signals**
  - These signals are either a **HIGH** or **LOW** voltage
  - Contrasts from **analog signals** which can be values in a continuous range



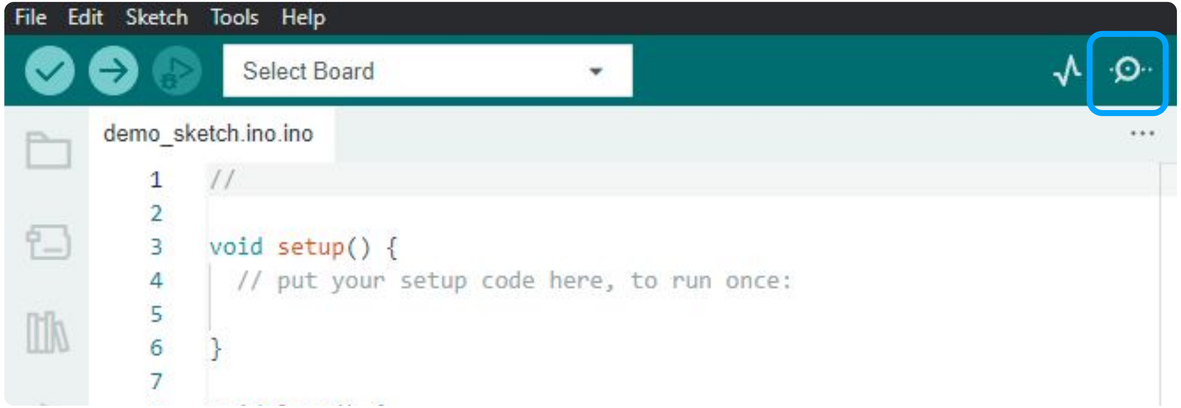
This is a **digital signal** where the waveform is either **5V** or **0V** (two *discrete* values)

- **Digital signals are translated** to the **Binary** number system (1s and 0s)

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**

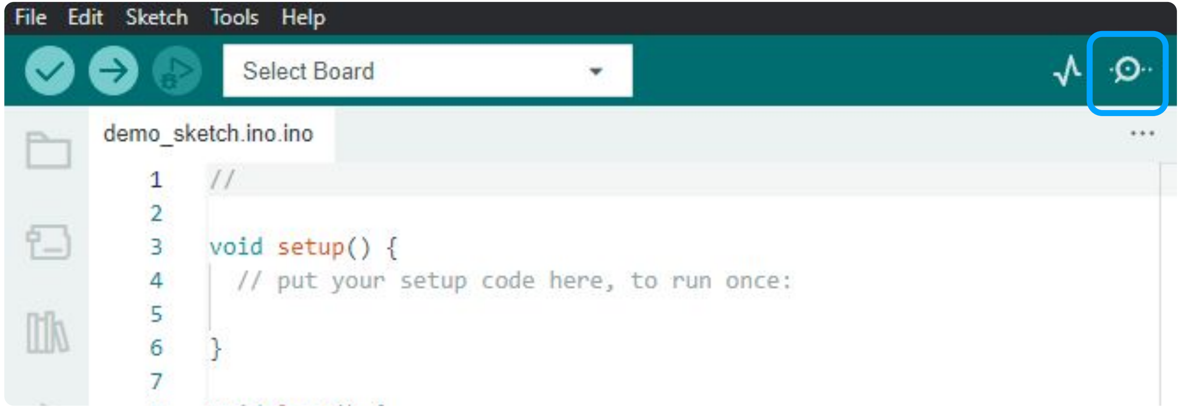
Using the Serial Monitor



```
//

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 the Serial Monitor (Cont'd)



```
//

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

}
```

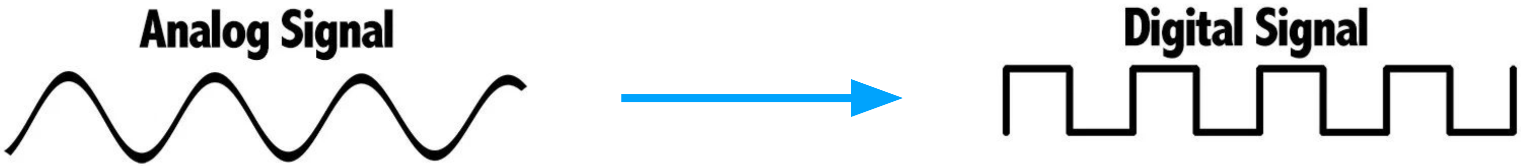
- In the absence of a debugger (the ESP32 is not capable of using one), `Serial.print` is an excellent tool to **help debug programs**
  - Print values to track across parts of your program
    - Unexpected values displayed to the Serial Monitor indicates an error

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)
  - This is a function for **~ PWM pins only**
- `analogRead(int pin)`
  - **Reads the voltage** at the input pin, maps it to a value in the **range** `0-1023` (0V to 3.3V) and returns that value
  - Use the aliases **`A0`**, **`A1`**, **`A2`**… for the pin number
- Analogy - light dimmer:
  - You use the slide to set the bulb to anywhere *between* MAX brightness or MIN brightness

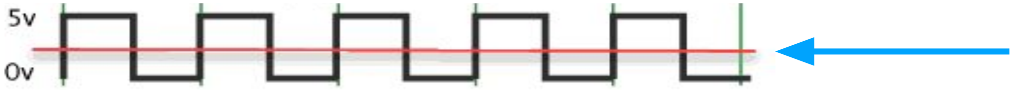
ADC and `analogRead`

- `analogRead` **utilizes the analog-to-digital converter** inside the microcontroller to **measure the real-world, analog signal** and **convert it to a digital signal**
  - The measurement resolution is 10 bits, which is why the function returns values from `0-1023`



PWM

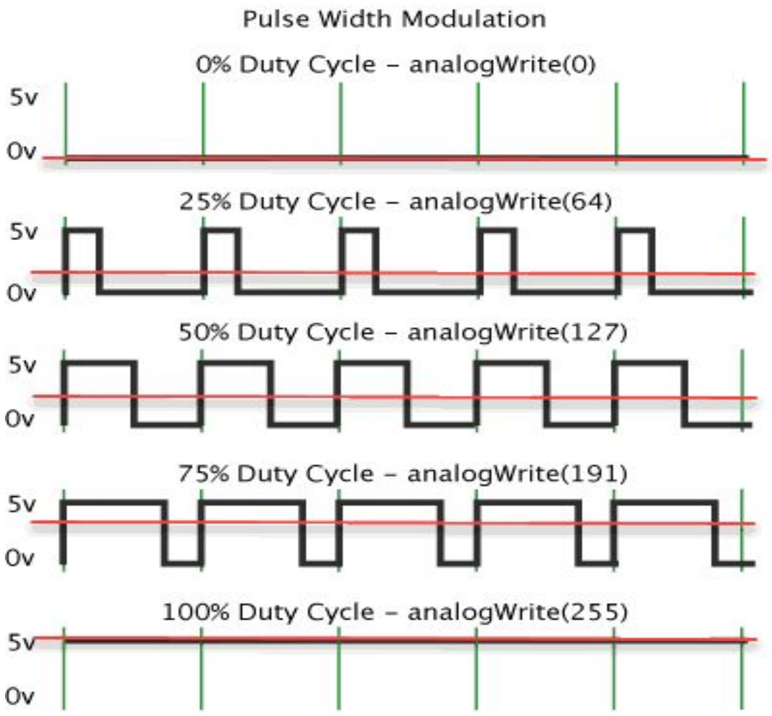
- The ESP32 board (the underlying AVR microcontroller) is a **digital source**, meaning it can only output a `HIGH` (5V) or `LOW` (0V) voltage
    - Then how does `analogWrite` output analog signals?
- Pulse Width Modulation (**PWM**) is an oscillating digital waveform that emulates an analog output
    - By oscillating a signal from `HIGH` to `LOW` quickly, the average voltage over time will be *between* `HIGH` and `LOW` - an analog value



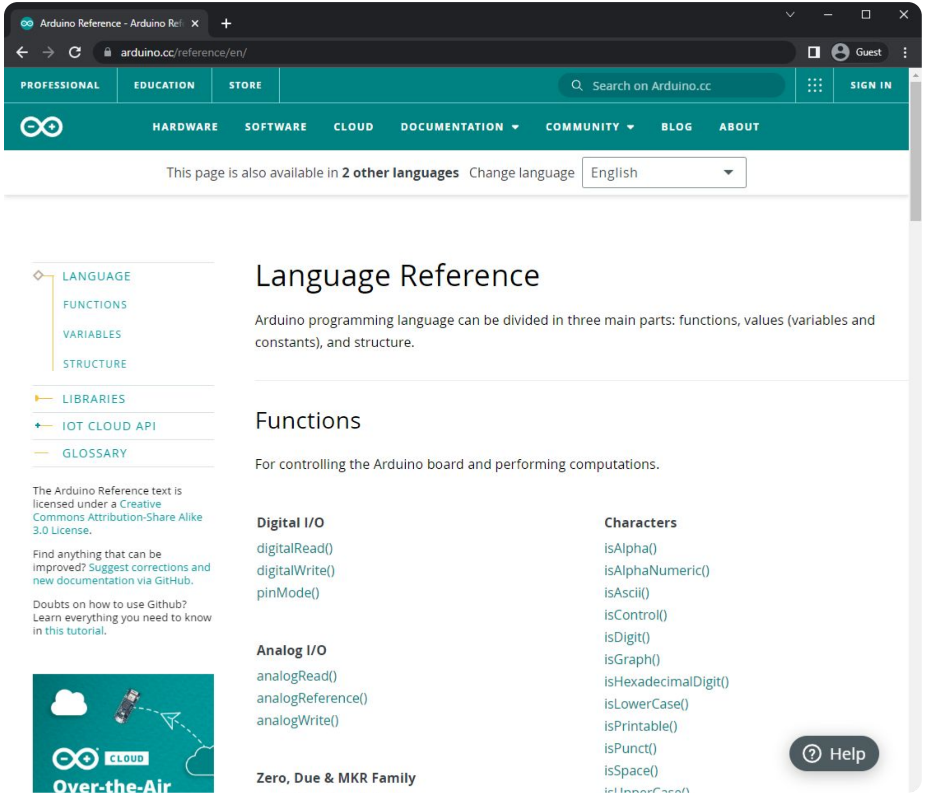
The average value, the *analog value*, of this waveform is **2.5V**

PWM Duty Cycle

- The **duty cycle** of the PWM wave is the percentage of time where the signal is `HIGH`
  - For example, a 50% duty cycle translates to an average value of 2.5V (50% of 5V)
  - Allows us to output a continuous range of voltages between `HIGH` and `LOW`
  - Duty cycle is **controlled by a timer** inside the AVR microcontroller


Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
25% Duty Cycle – analogWrite(64)
50% Duty Cycle – analogWrite(127)
75% Duty Cycle – analogWrite(191)
100% Duty Cycle – analogWrite(255)
5v
0v

Arduino Reference Library



This page is also available in **2 other languages** Change language

Language Reference

Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure.

Functions

For controlling the Arduino board and performing computations.

Digital I/O

digitalRead()
digitalWrite()
pinMode()

Analog I/O

analogRead()
analogReference()
analogWrite()

Zero, Due & MKR Family

Characters

isAlpha()
isAlphaNumeric()
isAscii()
isControl()
isDigit()
isGraph()
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()

The Arduino Reference text is licensed under a Creative Commons Attribution-Share Alike 3.0 License.

Find anything that can be improved? Suggest corrections and new documentation via GitHub.

Doubts on how to use Github? Learn everything you need to know in this tutorial.

- Learn more about Arduino functions and libraries here
- The **Arduino Reference Library** includes support for devices like LCDs, Sensors, and WiFi modules

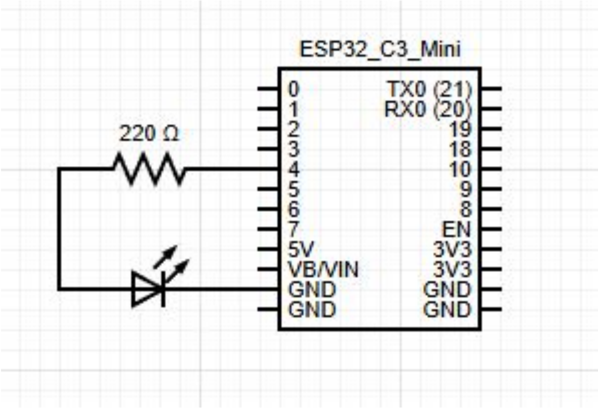
SECTION IX

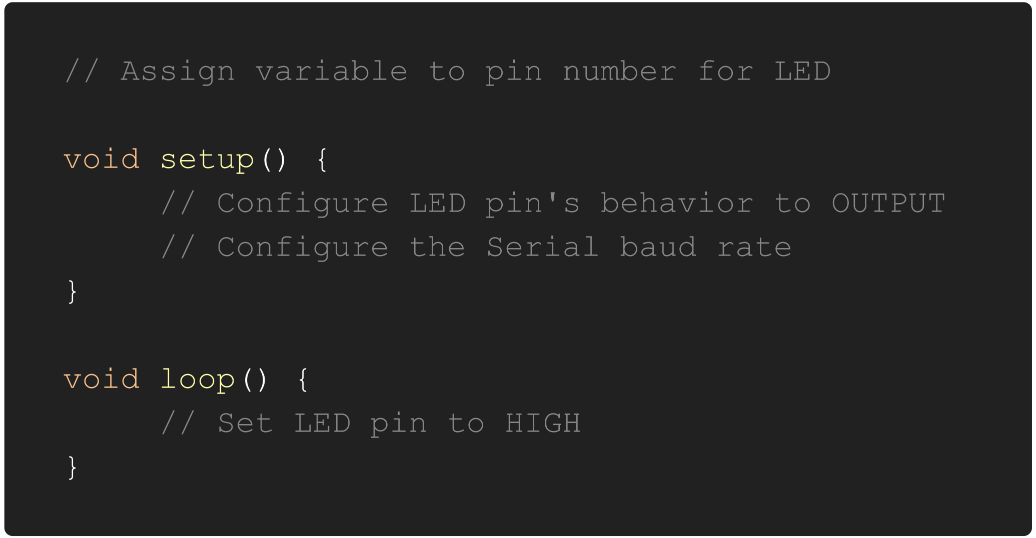
Digital LED Circuit Exercises

Digital LED Circuit

I/A

Build the circuit below from the schematic. Then, complete the template code, flash it to the ESP32 board, and verify the circuit.



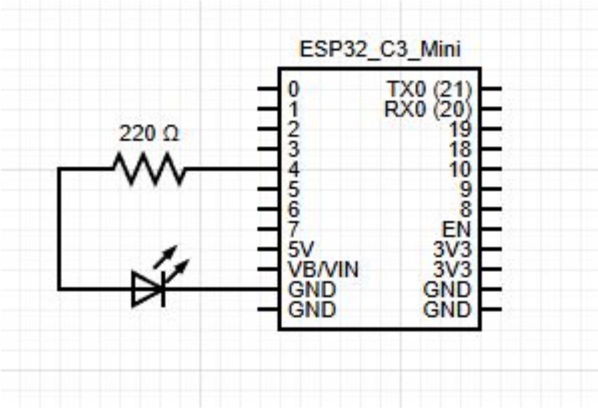


```
// Assign variable to pin number for LED

void setup() {
     // Configure LED pin's behavior to OUTPUT
     // Configure the Serial baud rate
}

void loop() {
     // Set LED pin to HIGH
}
```

Digital Blinking LED Circuit

I/A

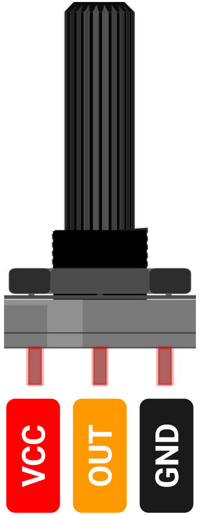
Build the circuit below from the schematic. Then, complete the template code, flash it to the ESP32, and verify the circuit.



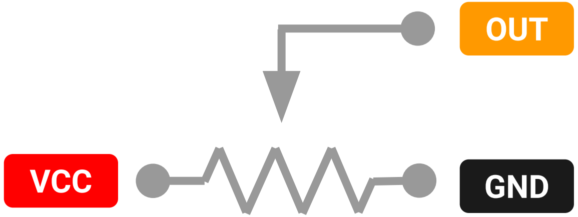
```
// Assign variable to pin number for LED

void setup() {
     // Configure LED pin's behavior to OUTPUT
     // Configure the Serial baud rate
}

void loop() {
     // Set LED pin to HIGH
     // Delay
     // Set LED pin to LOW
     // Delay
}
```

Potentiometers

- A **potentiometer** is a **three-terminal variable resistor**
- We will use the potentiometer as a **voltage divider** - a circuit which accepts a supply voltage and outputs a voltage which is a fraction of the supply voltage
- The voltage of the potentiometer's output pin ranges between the VCC and GND pin voltages

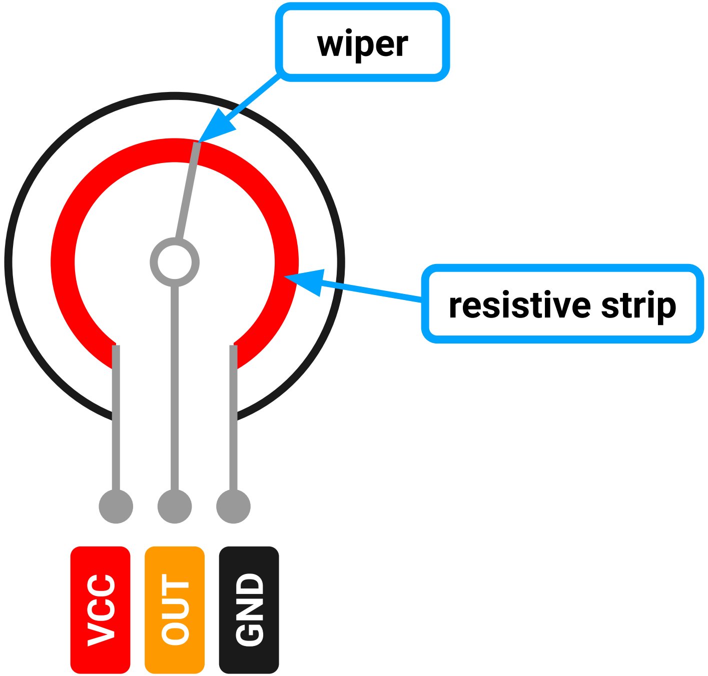


The positions of **VCC** and **GND** can be swapped



Potentiometers (Cont'd)

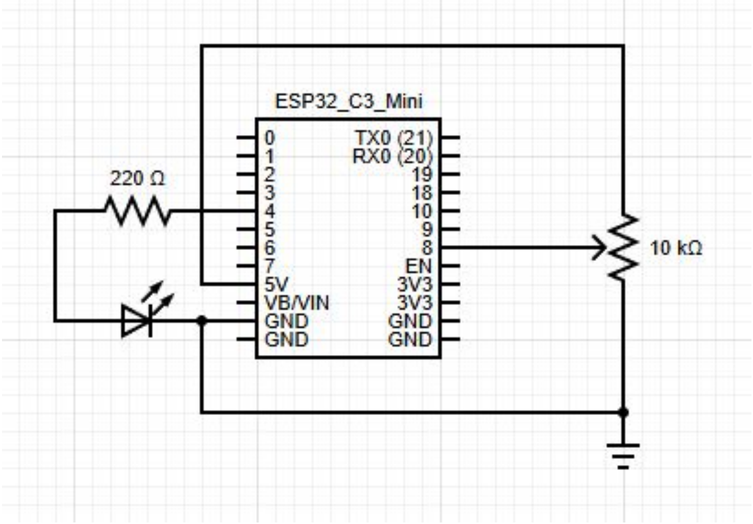
- Internally, a **resistive strip** connects its VCC and GND pins
    - A rotating **wiper** connects the output pin to the strip
- **The greater the distance** along the strip between the wiper and the VCC pin, **the greater the resistance** between the wiper and VCC
- The wiper **reduces the voltage** at the output pin the further it is **turned clockwise** (toward GND)



Digital Dimmable LED Circuit

I/A

Build the circuit below from the schematic. Then, complete the template code, flash it to the ESP32, and verify the circuit.



```
// Assign variable to pin number for LED
// Assign variable to pin number for Pot

void setup() {
  // Configure the LED pin's behavior to OUTPUT
  // Configure the Pot pin behavior to INPUT
  // Configure the Serial baud rate
}

void loop() {
      // Read pot pin value
      // Set LED pin to the pot pin value
}
```

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