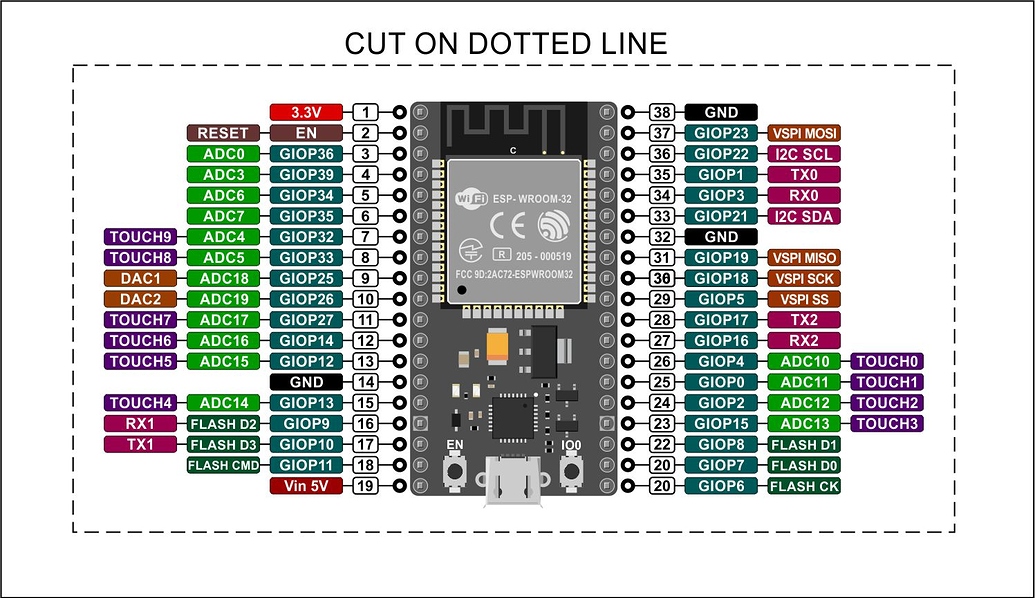
**Overview of the Arduino and ESP32**

Arduinos and ESPS are types of microcontrollers that can control electronics ranging from simple to complex. 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 to the computer. Connecting this provides power to the board.
* Indicated in yellow, are the digital ports. These ports are able to interpret and supply currents in binary signals. That is, they sense or release either ON or OFF.
  + Note that the ports with a tilde (~) are able to give analog outputs.
* Indicated in red are the analog ports. These ports are able to interpret currents 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 analog.
* GND – These ports are always a negative terminal
* 5V / 3.3V – These ports are voltage 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

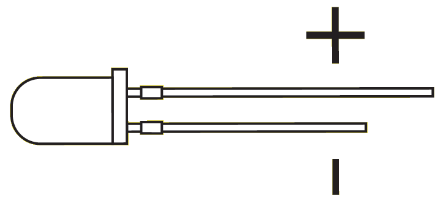
**Lesson 1: Blinking an LED**

What you’ll need:

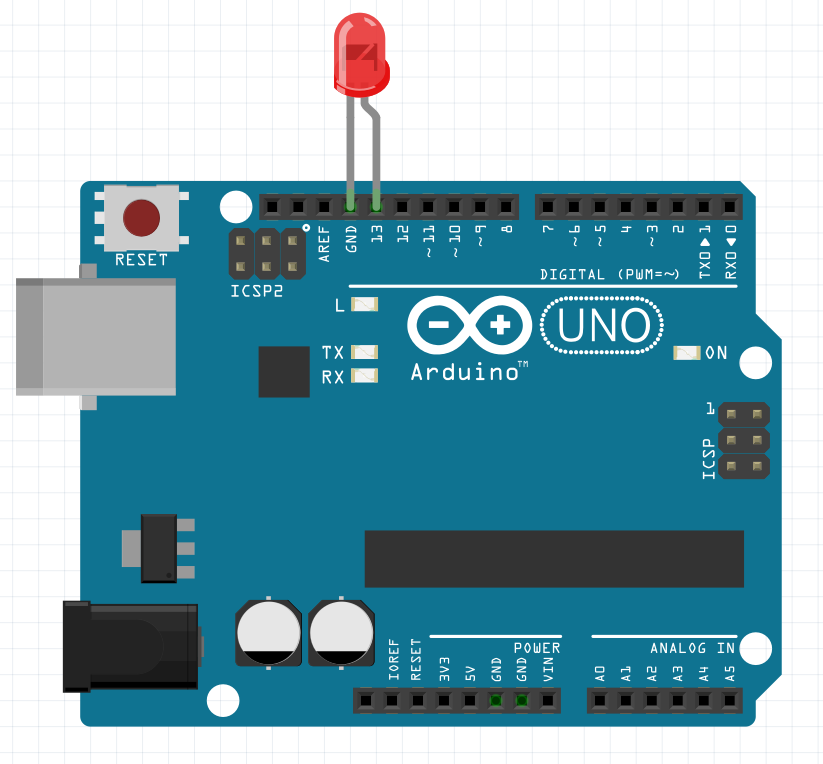
* Arduino / ESP32
* LED (light emitting diode)
* Breadboard

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:

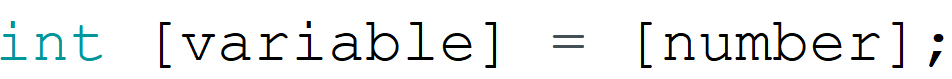
A picture containing text, electronics, circuit

Description automatically generated

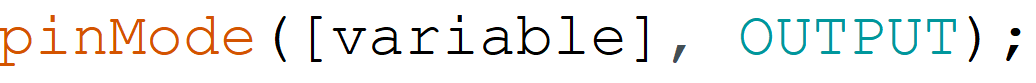
Directions:

|  |  |
| --- | --- |
| 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 |

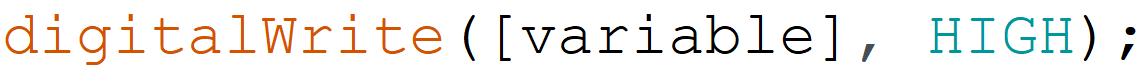
Code (Light the LED):



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.

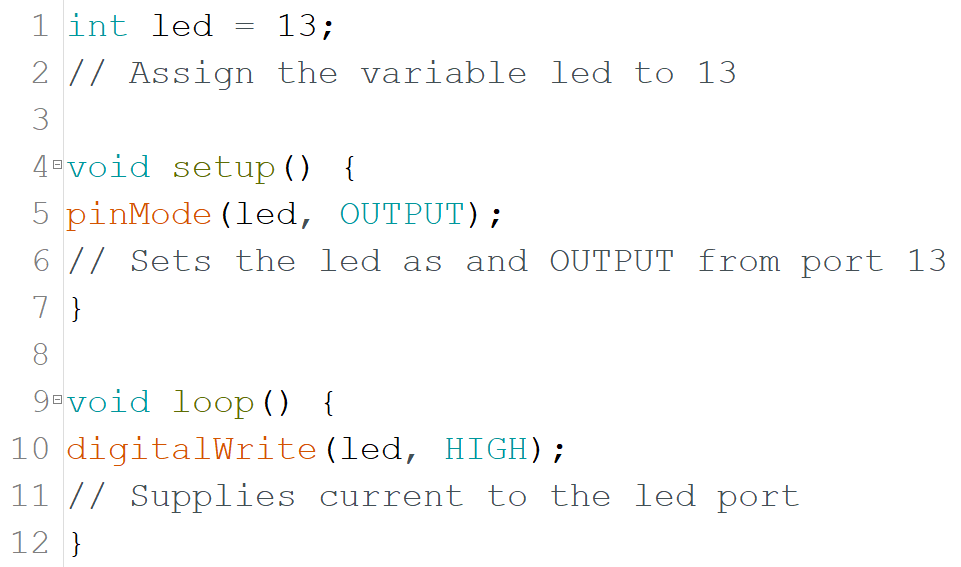


This line assigns a chosen variable as an output or an input on the Arduino. The Arduino takes the information from the variable and uses it as a port address.



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

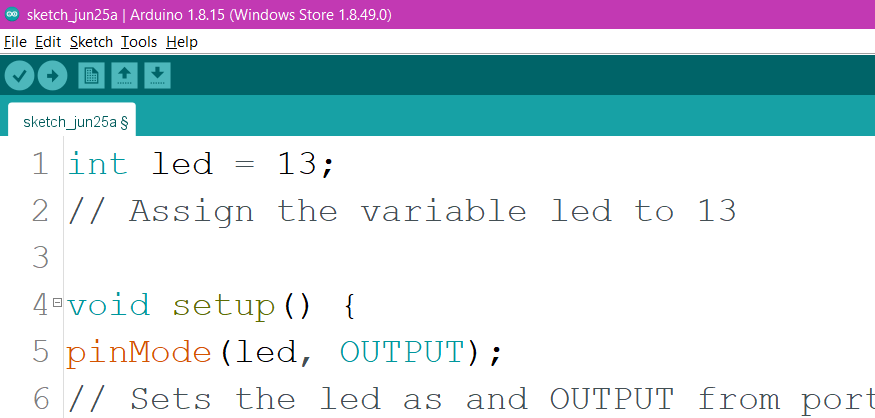
(Use port 23 for ESP32)



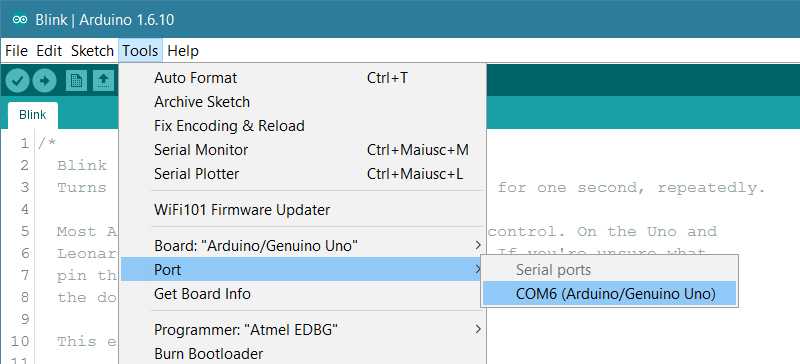
Upload to the Arduino and the LED should light up.

Uploading to Arduino/ESP32:

Plug the Arduino/ESP32 to the computer using the data cable and press the arrow button on the top left corner of the IDE. Make sure the board is correct.



Make sure to have your Arduino/ESP32 selected in ports



For the ESP32, hold down the boot button while uploading. When the image below is seen, release the boot button and press the reset button.

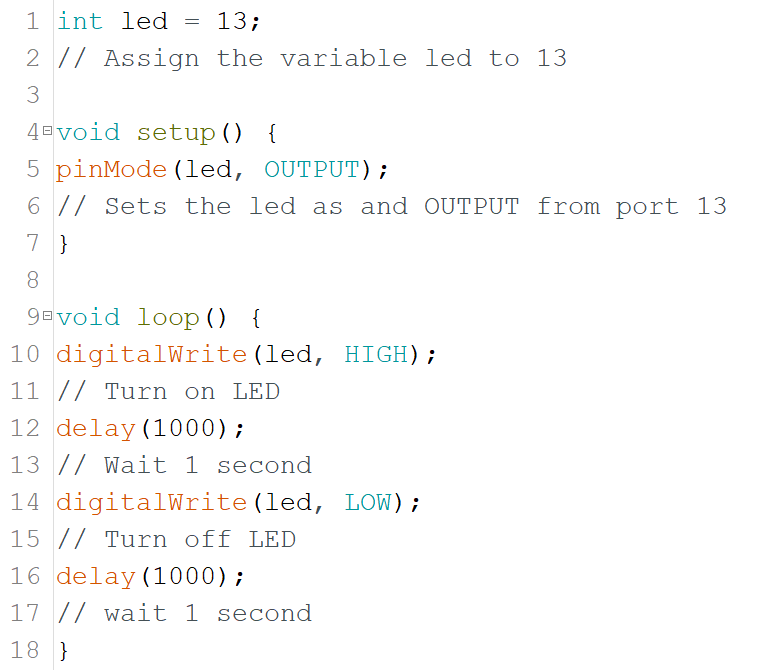
Text

Description automatically generated

Code (Blinking the LED):



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



(Use 23 for ESP32)

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.

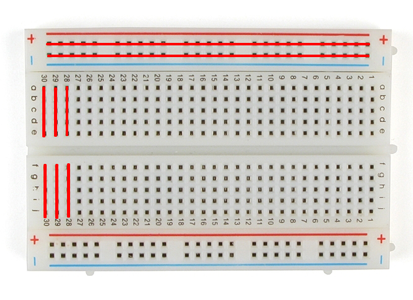
**Lesson 2: LED with a button**

What you’ll need:

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

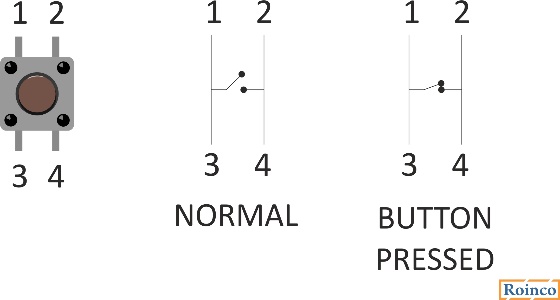
Breadboard:

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.



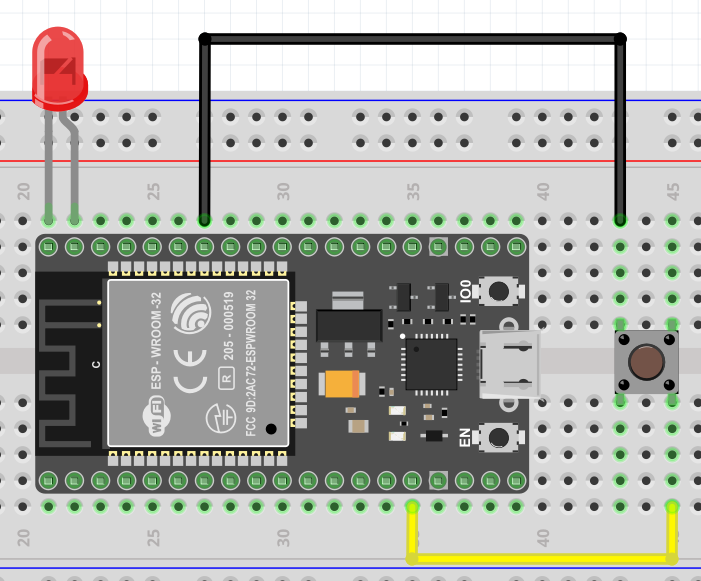
Button:

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:

Diagram

Description automatically generated

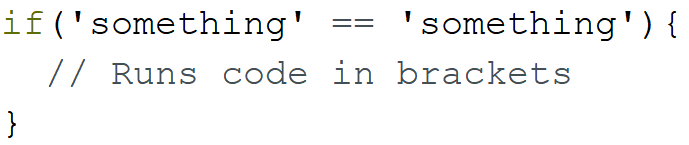
Directions:

|  |  |
| --- | --- |
| 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:



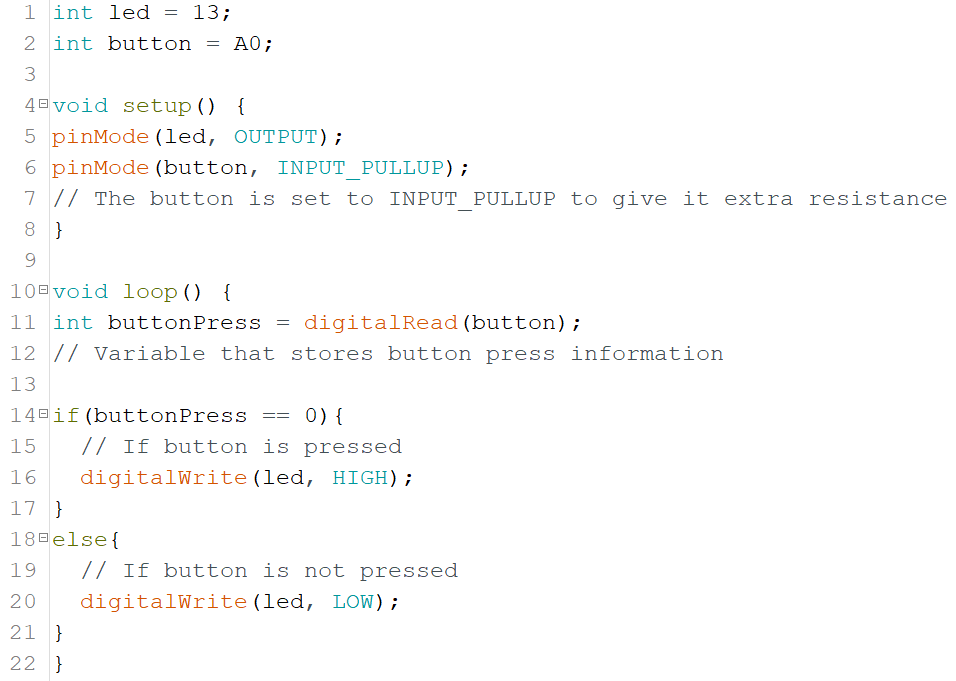
Reads the incoming digital signal from a sensor.



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 |

(Use 23 for ESP32)



(Use 12 for ESP32)

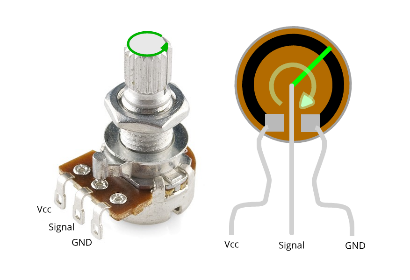
Upload the code to the Arduino/ESP32. When the button is not pressed, the LED should be off, and when the button is pressed, the LED should turn on.

**Lesson 3: Dimming an LED**

What you’ll need:

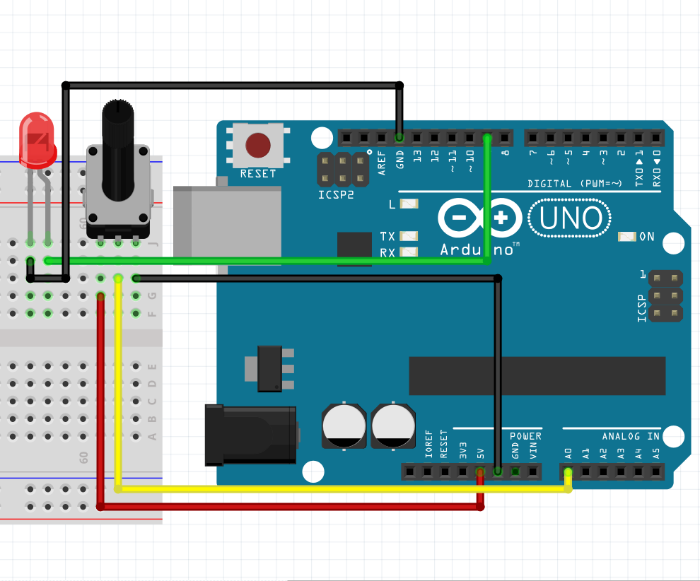
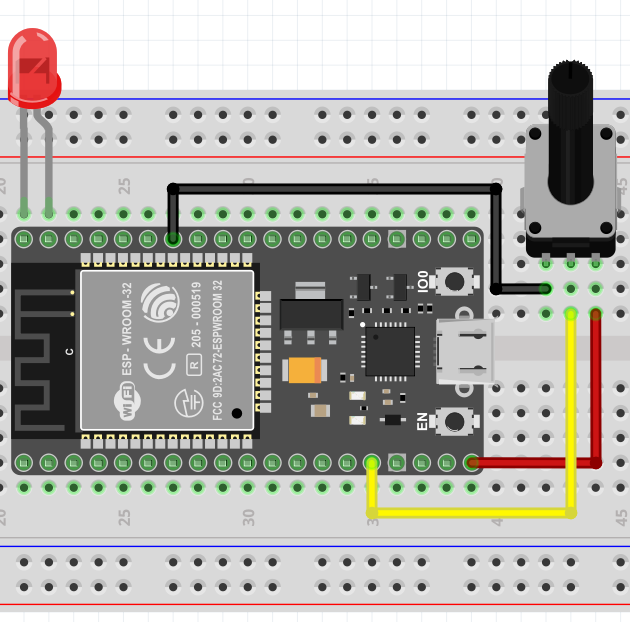
* Arduino/ESP32
* Breadboard
* Potentiometer
* LED
* 6 wires (male to male)

Potentiometer:



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:



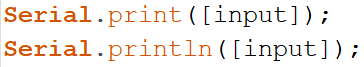
Directions:

|  |  |
| --- | --- |
| 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 |

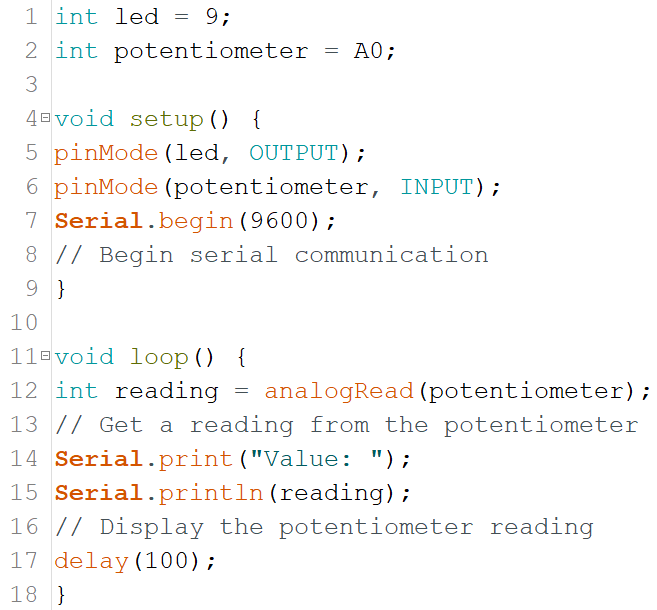
Code (Serial Feedback):



Starts serial communication with the computer



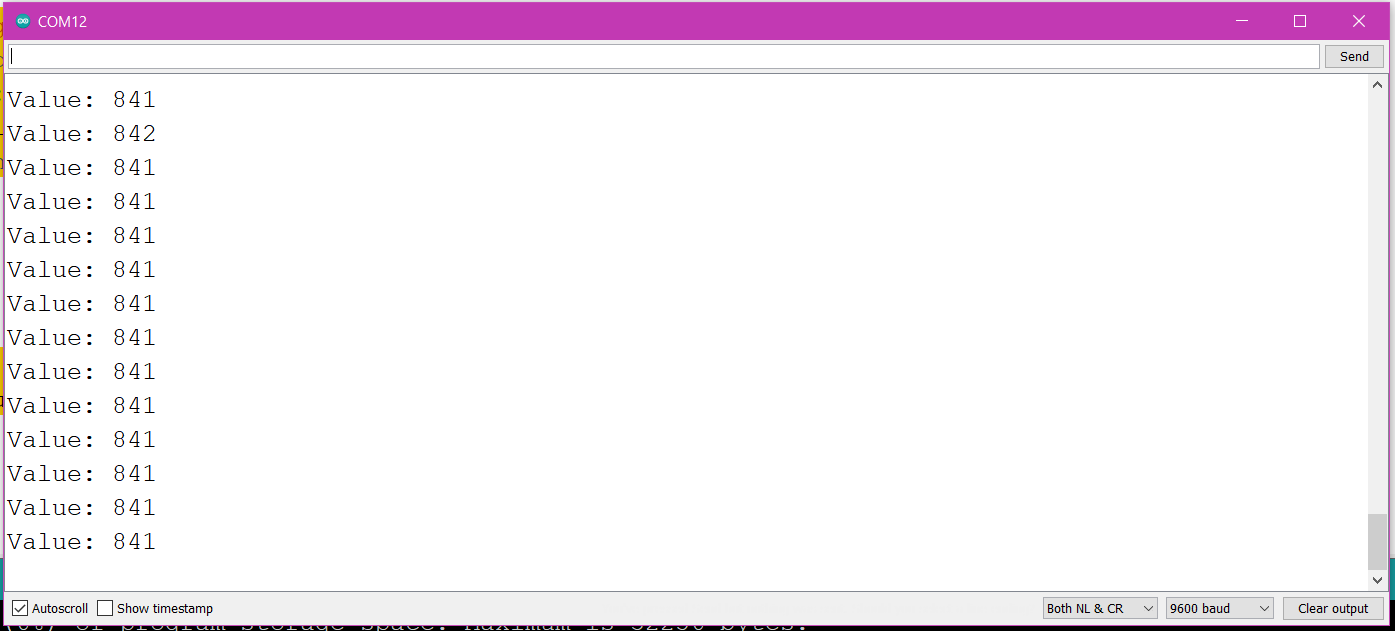
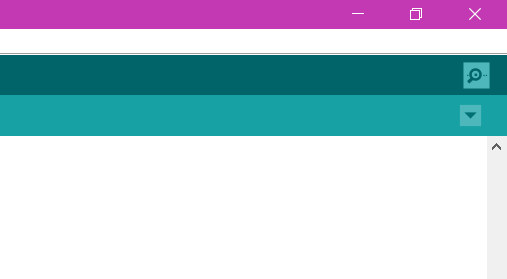
Displays the specified input to the monitor. Using ‘println’ creates a new line of text.



(Use 12 for ESP32)

(Use 23 for ESP32)

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 knob to change the value.



Code (Dimming LED Arduino):



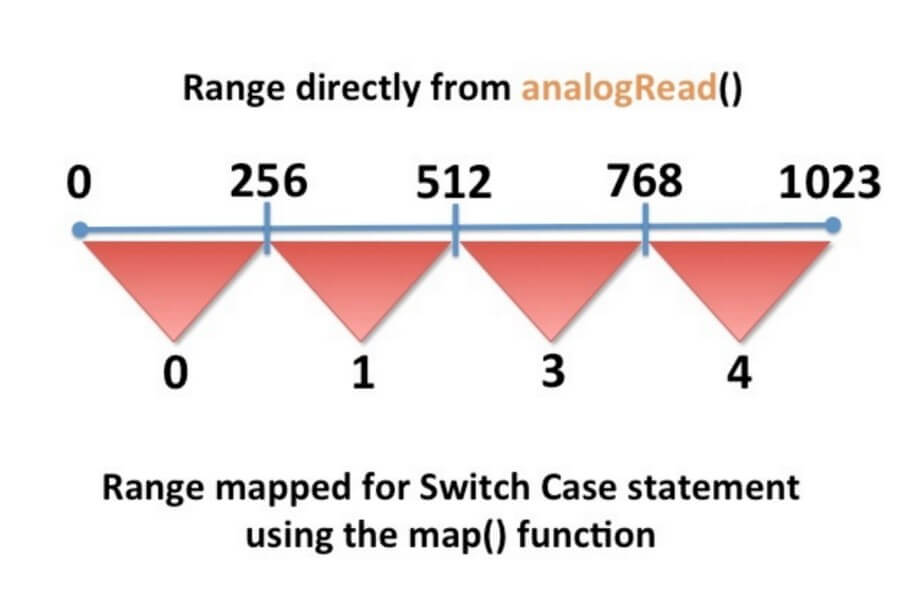
This line reads for an incoming analog information (range of numbers) from a port.

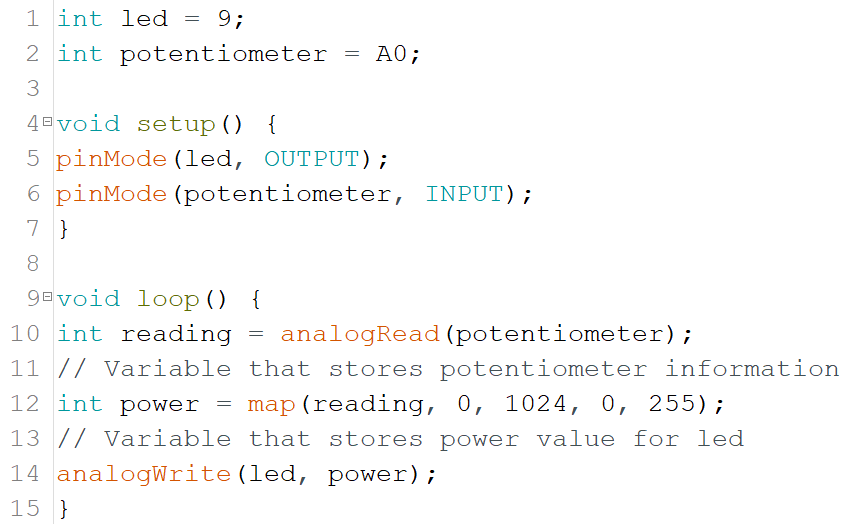


This line sends a power value between 0 and 255 to the specified port. This line can be used to send a range of powers.



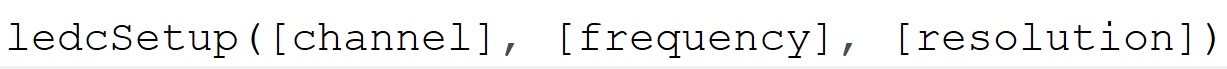
The map function takes in a value and checking its position in the larger range, rewrites its position in a lower range. After the number, the first two numbers are upper and lower bounds of the incoming number range (numbers above are only examples). The second two numbers are the output bounds. Visualisation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



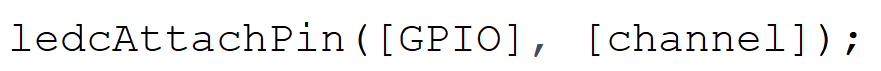


Upload the code to the Arduino and the LED should brighten when the knob is twisted.

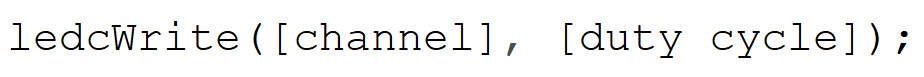
Code (Dimming LED ESP32):



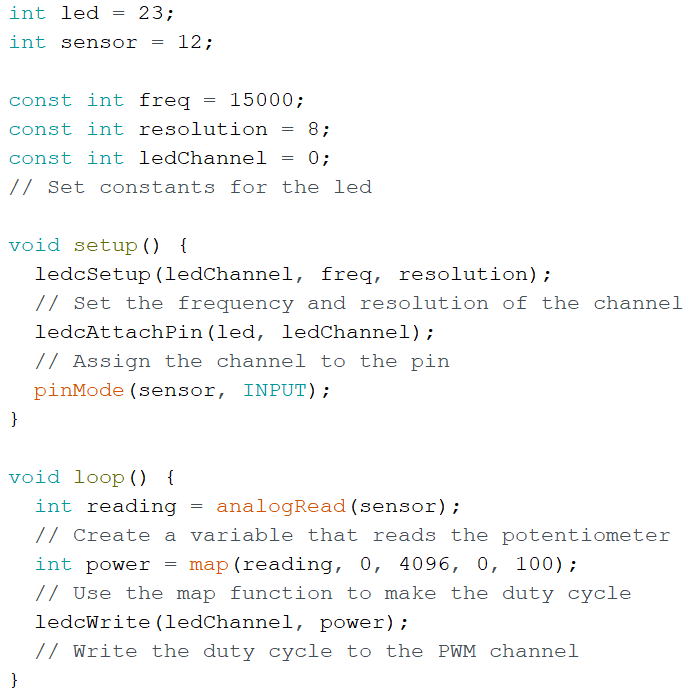
This line is used to set up the properties of the PWM signal, and the channel it will run on. Resolution is how many bits are used for the signal (up to 16 bits). The ESP32 has 16 PWM channels and any GPIO can be used to output PWM using a channel.



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



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.



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