Arduino Rundown

ASUMobiCar2019G25

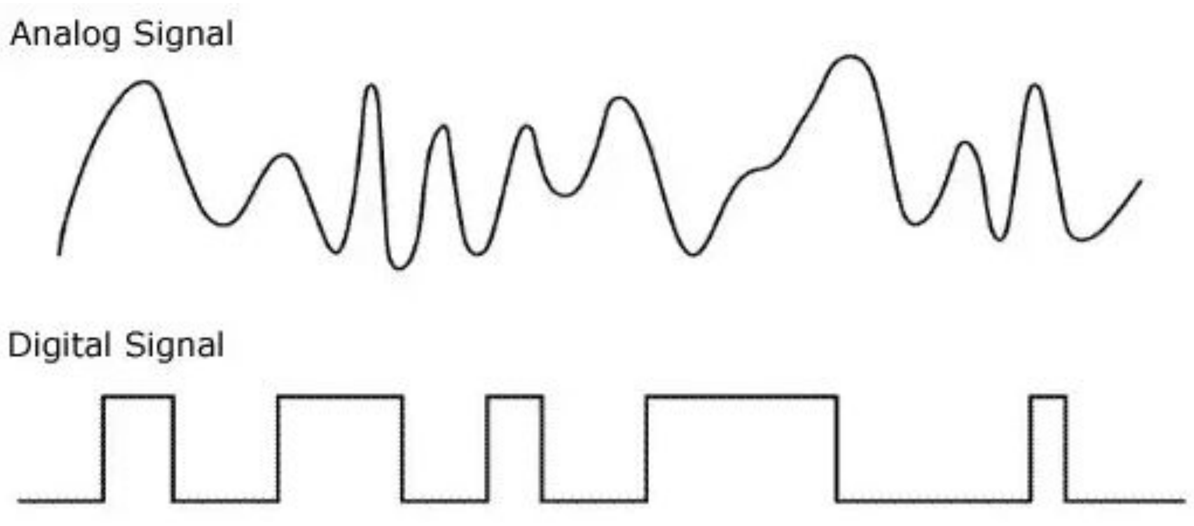
Arduino

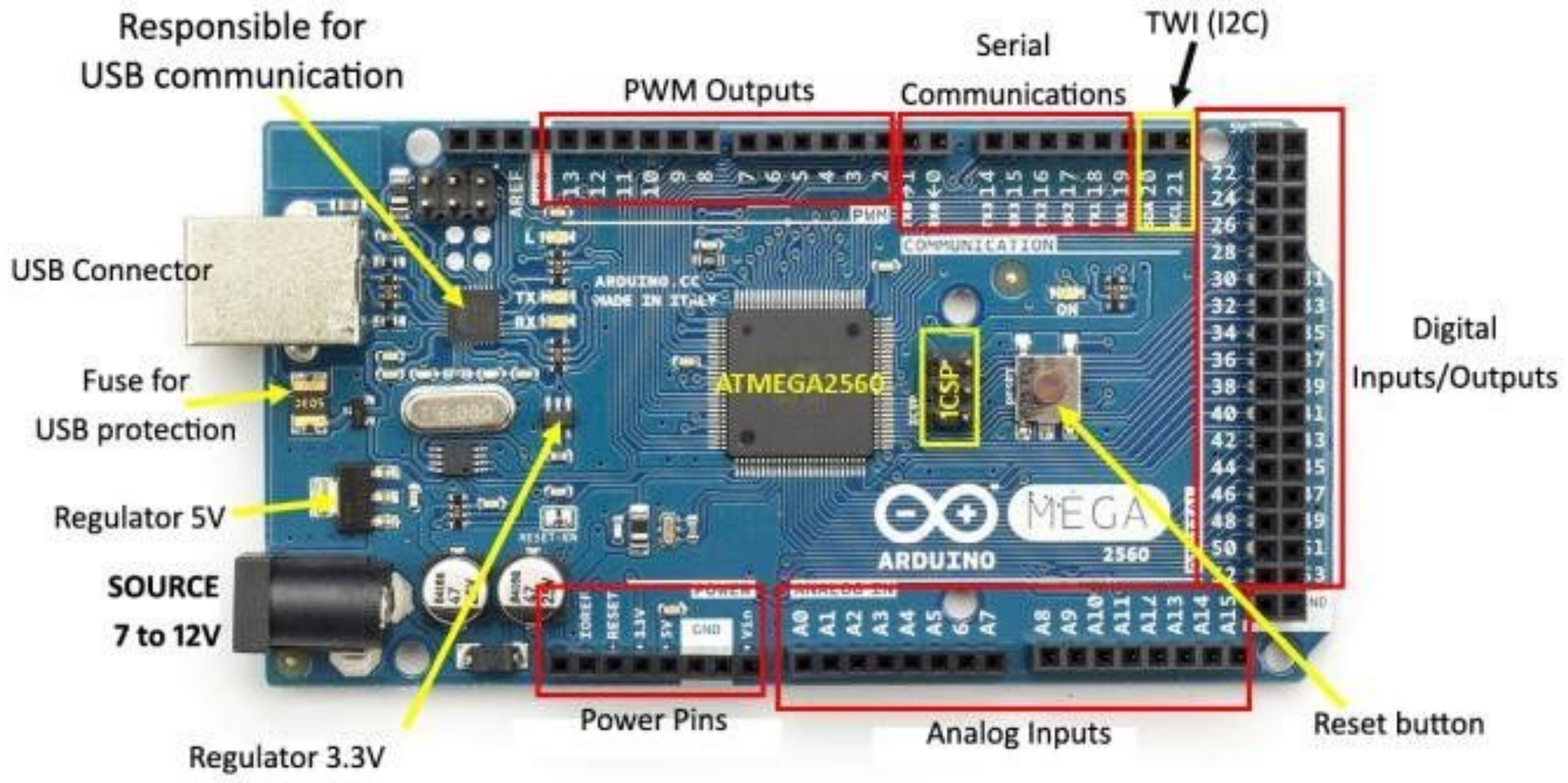
Arduino is a microcontroller board. A microcontroller is a mini-computer on a chip, so it has a memory (where we upload our code), and a processor (which executes the code).

The microcontroller chip (ATMega2560) is wired to a number of digital and analog input/output (I/O) pins that can be connected to external electronics such as push-buttons (input: sensors) or LED's (output: actuators). It also has a USB cable to connect to a computer (for power supply and code upload).

Examples of Digital/Analog Signals

- Digital Signals
  - **Input**: Smoke Sensor
    (0: No Smoke 1: Smoke detected)
  - **Output**: LED
    (0: LED off 1: LED on)
- Analog Signals
  - **Input**: Temperature Sensor (Senses a value between 0℃ to 100℃)
  - **Output**: LED (Varies its brightness with range 0 to 255)


Analog Signal
Digital Signal


Responsible for USB communication
PWM Outputs
Serial Communications
TWI (I2C)
USB Connector
Fuse for USB protection
Regulator 5V
SOURCE 7 to 12V
Regulator 3.3V
ATMEGA2560
ICSP
Digital Inputs/Outputs
Power Pins
Analog Inputs
Reset button

Notes on Arduino board

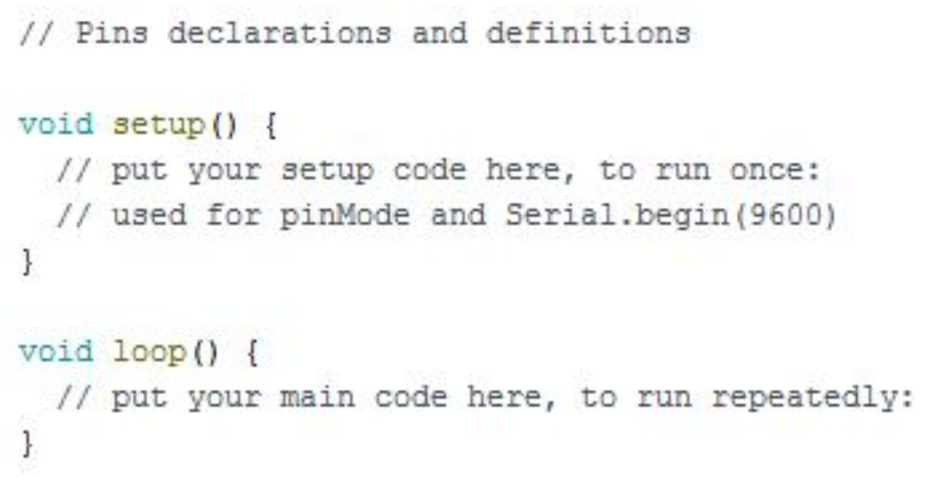
- **Power Pins**: where we get Vcc (5V) and GND.

- **Digital Pins**: 2 to 13 (input/output).
- **Analog In Pins**: A0 to A15 (input only).
- **PWM~ Outputs**: can be used as digital in/out or analog out.

- **Regulator 5V**: regulates whatever voltage (between 7V and 12V) is supplied from the power socket (SOURCE) into a constant 5V.
- **Reset Button**: resets the microcontroller so that it begins the program from the start.

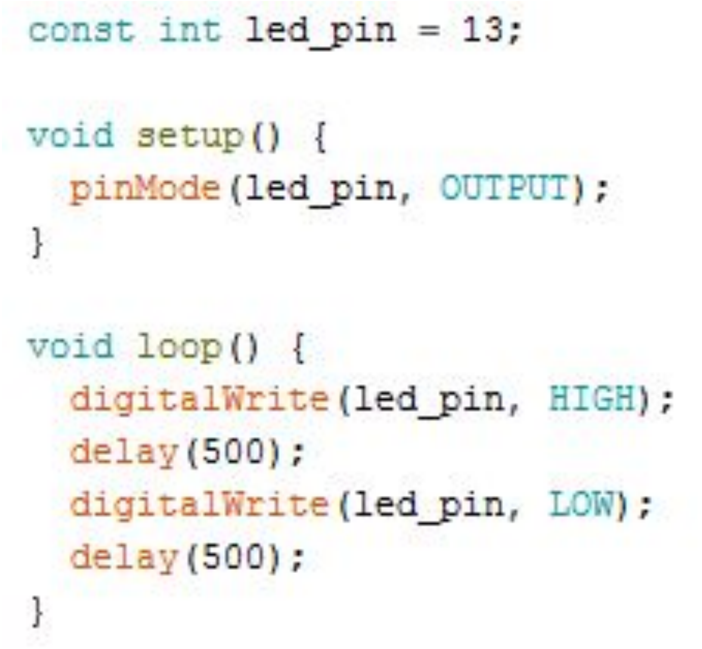
Arduino Software: ide

https://www.arduino.cc/download_handler.php



```
// Pins declarations and definitions

void setup() {
  // put your setup code here, to run once:
  // used for pinMode and Serial.begin(9600)
}

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

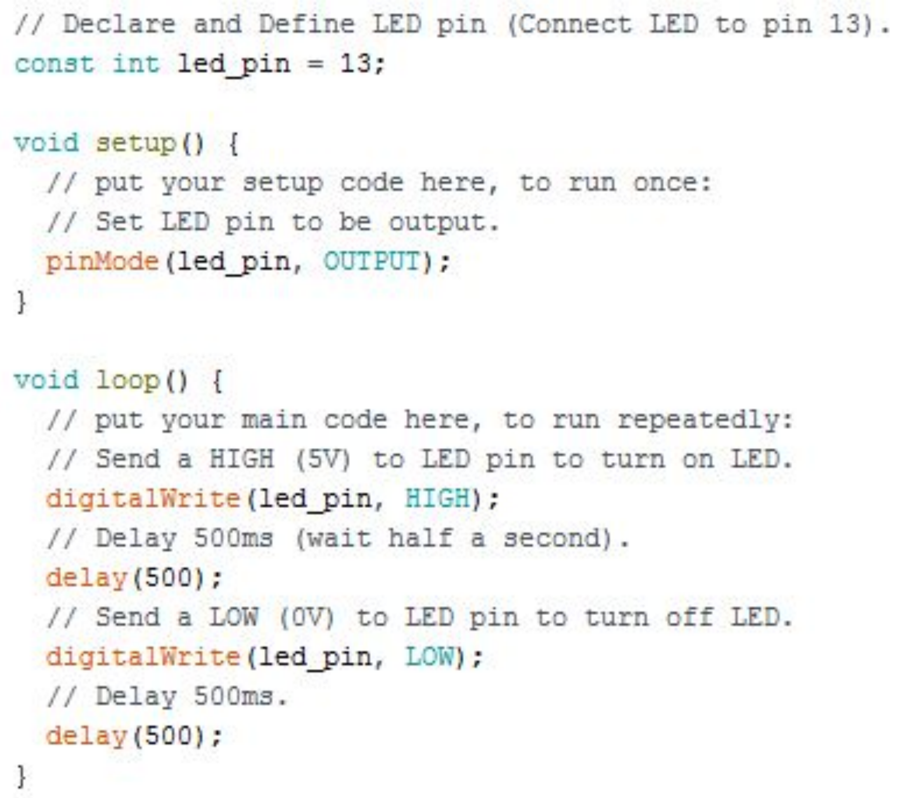
Blinking LED Sketch



```
const int led_pin = 13;

void setup() {
  pinMode(led_pin, OUTPUT);
}

void loop() {
  digitalWrite(led_pin, HIGH);
  delay(500);
  digitalWrite(led_pin, LOW);
  delay(500);
}
```

without comments

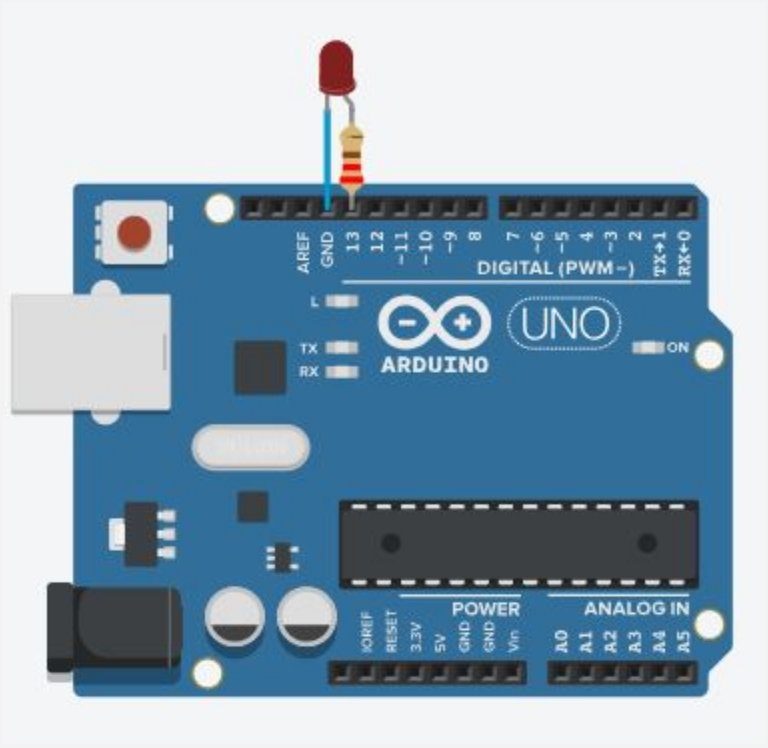
Blinking LED Sketch



```
// Declare and Define LED pin (Connect LED to pin 13).
const int led_pin = 13;

void setup() {
  // put your setup code here, to run once:
  // Set LED pin to be output.
  pinMode(led_pin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  // Send a HIGH (5V) to LED pin to turn on LED.
  digitalWrite(led_pin, HIGH);
  // Delay 500ms (wait half a second).
  delay(500);
  // Send a LOW (0V) to LED pin to turn off LED.
  digitalWrite(led_pin, LOW);
  // Delay 500ms.
  delay(500);
}
```

with comments

Connection Circuit



Sketch Notes

Variables

A **variable** is a container in memory which has:

- Type
- Name
- Value

In this sketch, an integer *int* variable called *led_pin* is being set to the value of 13, the pin that the LED is connected to on the Arduino, and because *led_pin* will always be connected to pin 13, we add *const* to prevent it from ever changing. Throughout the rest of the program, we can simply use *led_pin* whenever we want to control pin 13.

Setting variables is useful because you can just change this one line if you hook up your LED to a different I/O pin later on; the rest of the code will still work as expected.

Sketch Notes
Declarations and Definitions

Above the setup function, we write the declarations and definitions of all pins:

- **Declaration**: const int x;
- **Definition**: x = 0;
- **Declaration with Definition**: const int x = 0;

Example: const int led_pin = 13;

// This allows us to use the variable led_pin throughout the code to refer to digital pin 13.

Sketch Notes
void setup

```
void setup() {

}
```

is one of two functions that must be included in every Arduino program. Code within the curly braces of the setup function is executed only once at the start of the program.

This is useful for one-time settings, such as setting the direction of pins pinMode() and initializing communication interfaces Serial.begin(9600).

Sketch Notes
pinMode

Arduino’ digital pins can function as input or outputs. To configure a specific pin to behave either as an input or an output, use the command:

pinMode(pin_variable, DIRECTION);

DIRECTION: INPUT or OUTPUT

This needs to be done only once at the beginning, so we normally put it in the setup function.

Example: pinMode(led_pin, OUTPUT);

Sketch Notes

void loop

```
void loop() {

}
```

is the second required function in all Arduino programs. The contents of the loop function repeat forever (at the speed of 16 MHz) as long as the Arduino is on.

Sketch Notes
digitalWrite

digitalWrite(pin_variable, STATE); sets the state of a digital output pin (sends out 5V or 0V to the pin). The pin remains in this state until it is changed in the code.

STATE can either be HIGH (5V) or LOW (0V).

Example: digitalWrite(led_pin, HIGH);

// Sends a HIGH signal (5V) to the *led_pin*, so the LED is turned on.

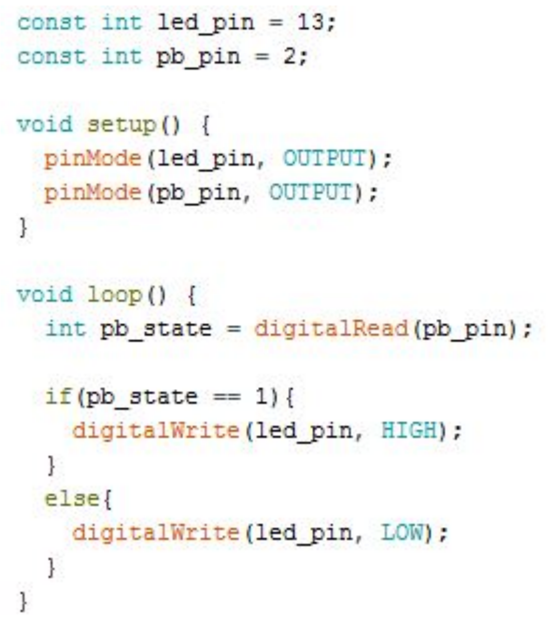
Sketch Notes
delay

delay(delay_period_in_ms); makes the Arduino stop doing anything for the amount of time specified in milliseconds.

Example: delay(500);

// In this sketch, we're delaying the program for 500ms, or half a second. This results in the LED staying on for half a second before you execute the next command.

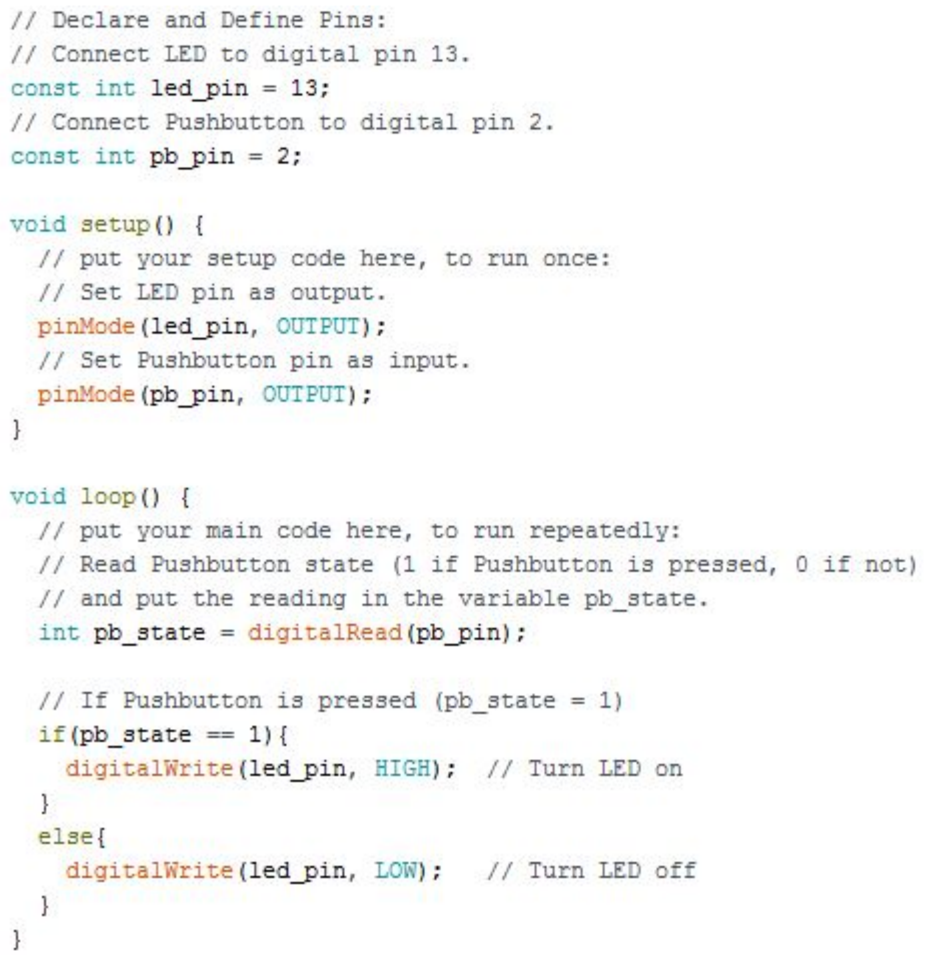
Push-Button Controlled LED Sketch



```
const int led_pin = 13;
const int pb_pin = 2;

void setup() {
  pinMode(led_pin, OUTPUT);
  pinMode(pb_pin, OUTPUT);
}

void loop() {
  int pb_state = digitalRead(pb_pin);

  if(pb_state == 1){
    digitalWrite(led_pin, HIGH);
  }
  else{
    digitalWrite(led_pin, LOW);
  }
}
```

without comments

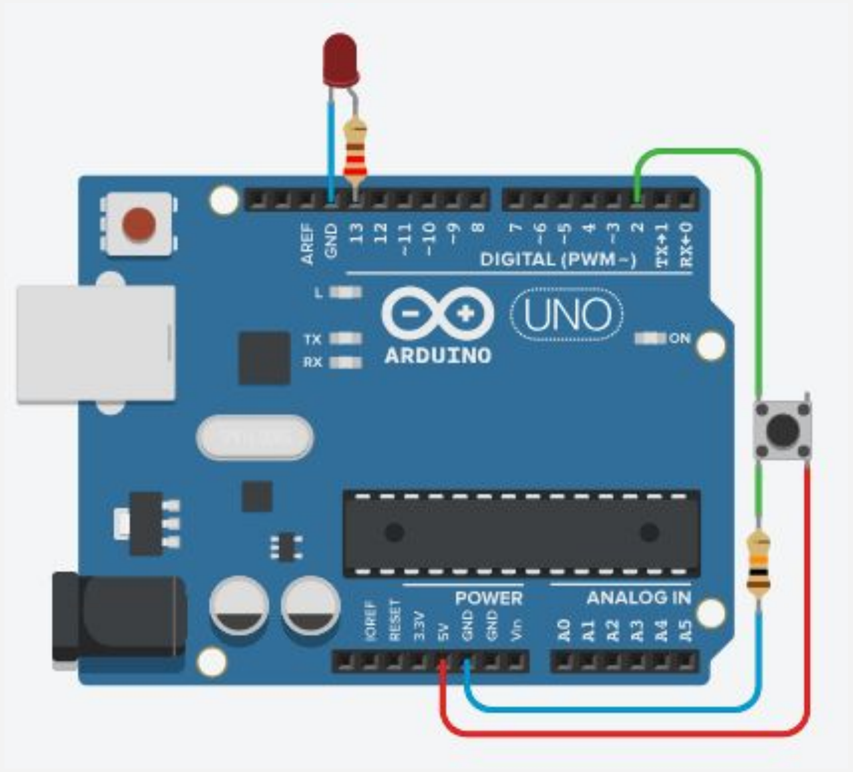
Push-Button Controlled LED Sketch



```
// Declare and Define Pins:
// Connect LED to digital pin 13.
const int led_pin = 13;
// Connect Pushbutton to digital pin 2.
const int pb_pin = 2;

void setup() {
  // put your setup code here, to run once:
  // Set LED pin as output.
  pinMode(led_pin, OUTPUT);
  // Set Pushbutton pin as input.
  pinMode(pb_pin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  // Read Pushbutton state (1 if Pushbutton is pressed, 0 if not)
  // and put the reading in the variable pb_state.
  int pb_state = digitalRead(pb_pin);

  // If Pushbutton is pressed (pb_state = 1)
  if(pb_state == 1){
    digitalWrite(led_pin, HIGH);  // Turn LED on
  }
  else{
    digitalWrite(led_pin, LOW);   // Turn LED off
  }
}
```

with comments

Connection Circuit



Sketch Notes
digitalRead

digitalRead(pin_variable); reads the value of an input digital pin (0 or 1).

Example: int x = digitalRead(pb_pin);

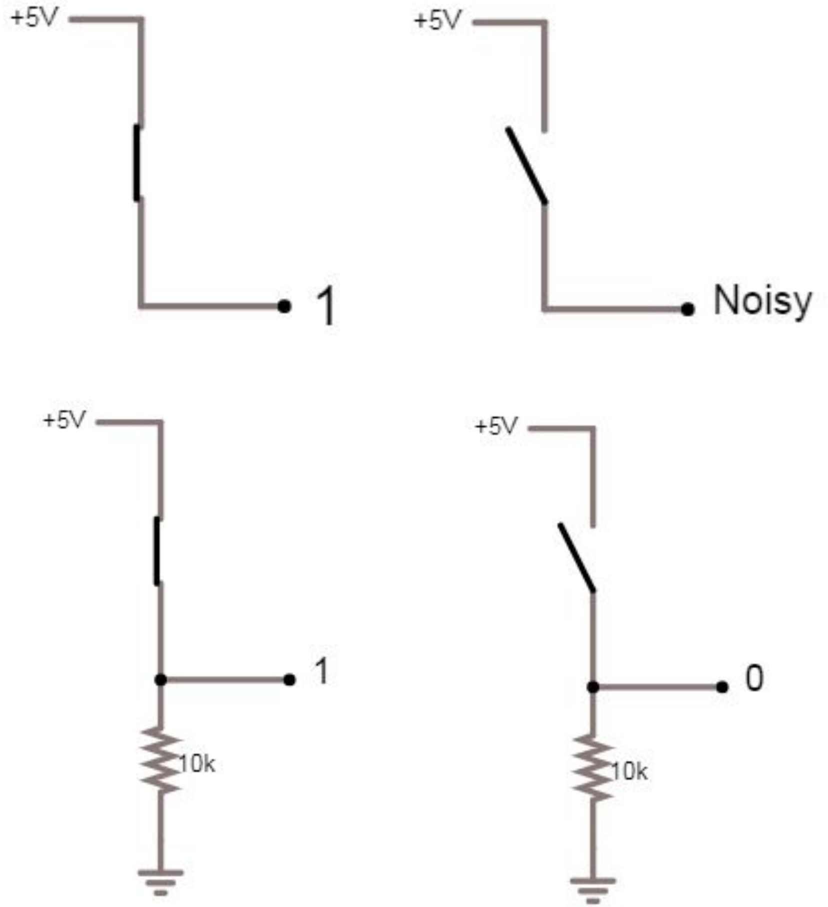
// reads the value of the pin connected to the push-button *pb_pin*, which will be 0 if the button isn't pressed, and 1 if it is. The reading is then stored in the variable x.

Sketch Notes
Pull-down Resistor

When working with push-buttons, it’s important to understand the concept of a **pull-down resistor**.

When the pushbutton is pressed, the output will be Vcc. However, without the pull-down resistor, if the push-button isn’t pressed, nothing guarantees that the output will be GND (the pushbutton will be floating and its output will be noisy). So we use a high value resistance to “pull down” the output (connect it to GND) when the push-button isn’t pressed (high value so it doesn’t take a lot of current).

Sketch Notes
Pull-down Resistor



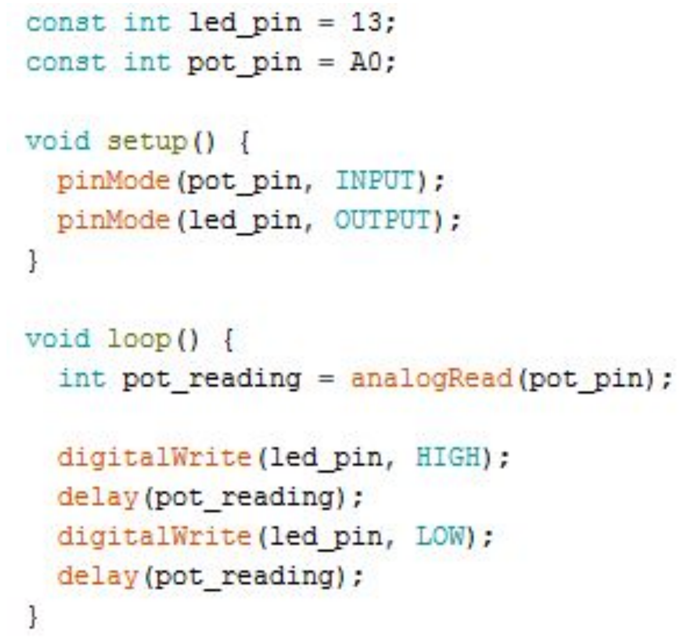
Sketch Notes
If-Statement

The if-statement checks for a condition and executes the statement or set of statements within its curly braces if the condition is true.

```
if                    (CONDITION)                    {
      // Code which is executed if CONDITION is true.
}
```

Example: In this sketch, the if() statement is checking to see if the value returned by digitalRead() is HIGH. The == is a comparison operator that tests whether the first item digitalRead() is equal to the second (HIGH). If this is true (that is, the button is being pressed), the code inside the brackets executes, and the LED set to HIGH. If this is not true (the button is not being pressed), the else statement is executed, and the LED is turned LOW.

Potentiometer Controlled LED Blink Rate Sketch

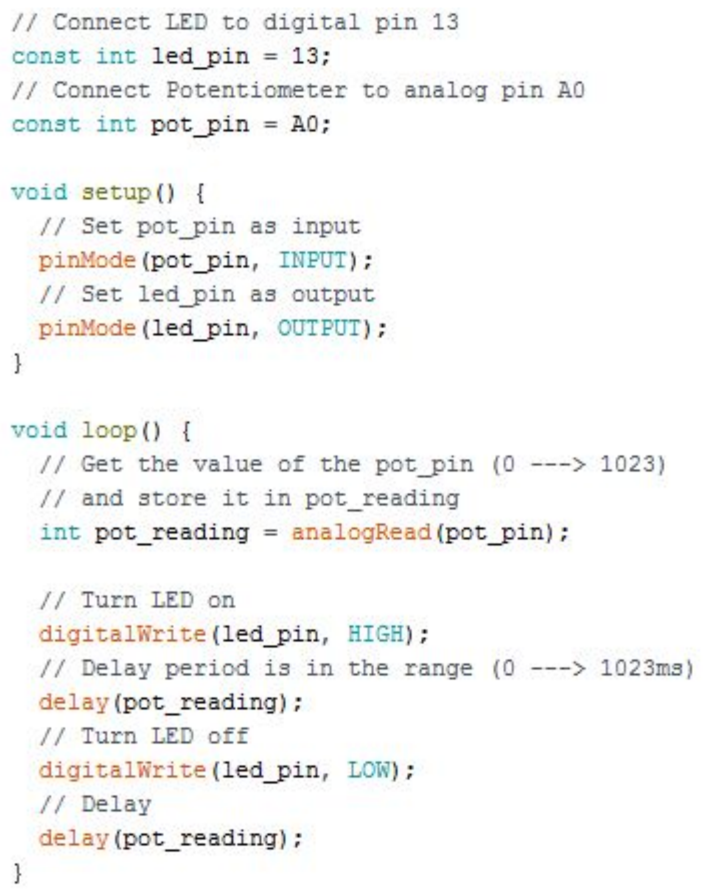


```
const int led_pin = 13;
const int pot_pin = A0;

void setup() {
  pinMode(pot_pin, INPUT);
  pinMode(led_pin, OUTPUT);
}

void loop() {
  int pot_reading = analogRead(pot_pin);

  digitalWrite(led_pin, HIGH);
  delay(pot_reading);
  digitalWrite(led_pin, LOW);
  delay(pot_reading);
}
```

without comments

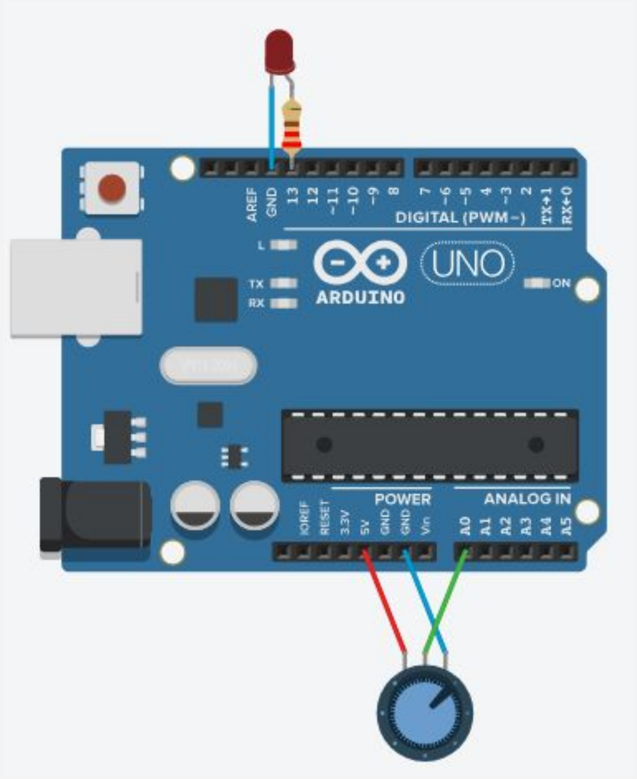
Potentiometer Controlled LED Blink Rate Sketch

with comments



```
// Connect LED to digital pin 13
const int led_pin = 13;
// Connect Potentiometer to analog pin A0
const int pot_pin = A0;

void setup() {
  // Set pot_pin as input
  pinMode(pot_pin, INPUT);
  // Set led_pin as output
  pinMode(led_pin, OUTPUT);
}

void loop() {
  // Get the value of the pot_pin (0 ---> 1023)
  // and store it in pot_reading
  int pot_reading = analogRead(pot_pin);

  // Turn LED on
  digitalWrite(led_pin, HIGH);
  // Delay period is in the range (0 ---> 1023ms)
  delay(pot_reading);
  // Turn LED off
  digitalWrite(led_pin, LOW);
  // Delay
  delay(pot_reading);
}
```

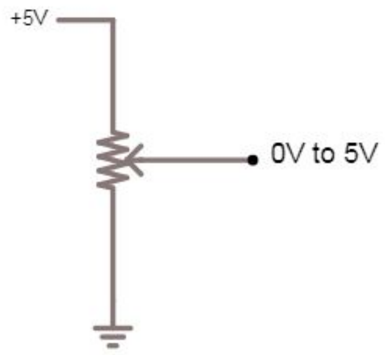
Connection Circuit



Sketch Notes
Potentiometer Voltage Divider

Potentiometers have 3 terminals, the two on either side are connected to Vcc and GND. The middle terminal (called wiper) is wired to an input analog pin on the Arduino board. This creates a voltage divider with the output voltage measured by the Arduino.

As you turn the potentiometer, you vary the voltage between 0V and 5V. This 5V range corresponds to the range 0 to 1023 which is returned by analogRead function.



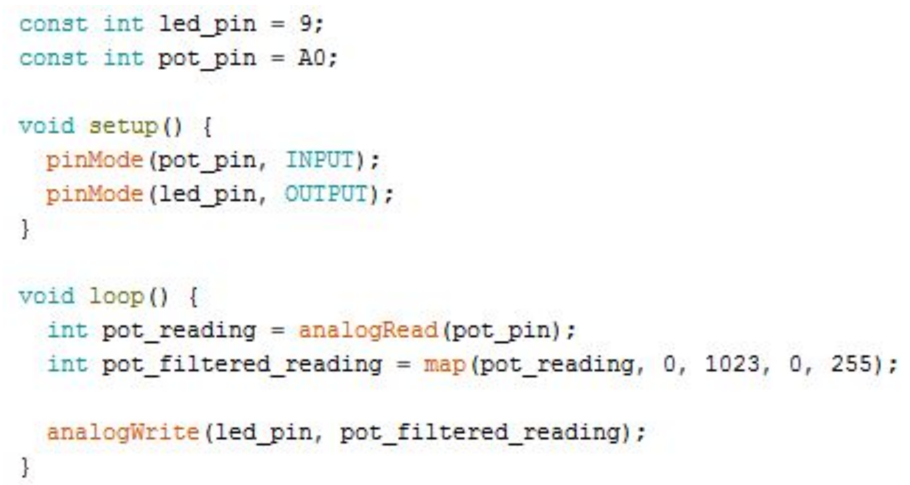
Sketch Notes
analogRead

analogRead(pin_variable); reads the value of an input analog pin, returning a number in the range (0 → 1023).

Example: `int x = analogRead(pot_pin);`

// Reads the analog value of the potentiometer (a number between 0 and 1023) and stores it in the variable x.

Potentiometer Controlled LED Brightness Sketch

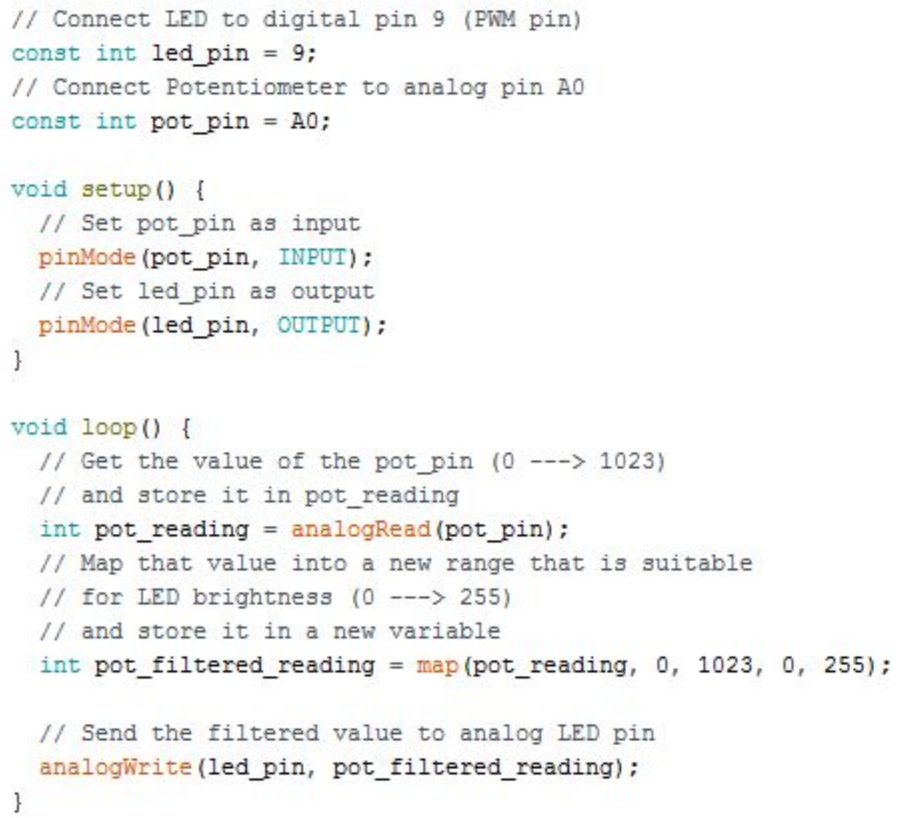


```
const int led_pin = 9;
const int pot_pin = A0;

void setup() {
  pinMode(pot_pin, INPUT);
  pinMode(led_pin, OUTPUT);
}

void loop() {
  int pot_reading = analogRead(pot_pin);
  int pot_filtered_reading = map(pot_reading, 0, 1023, 0, 255);

  analogWrite(led_pin, pot_filtered_reading);
}
```

without comments

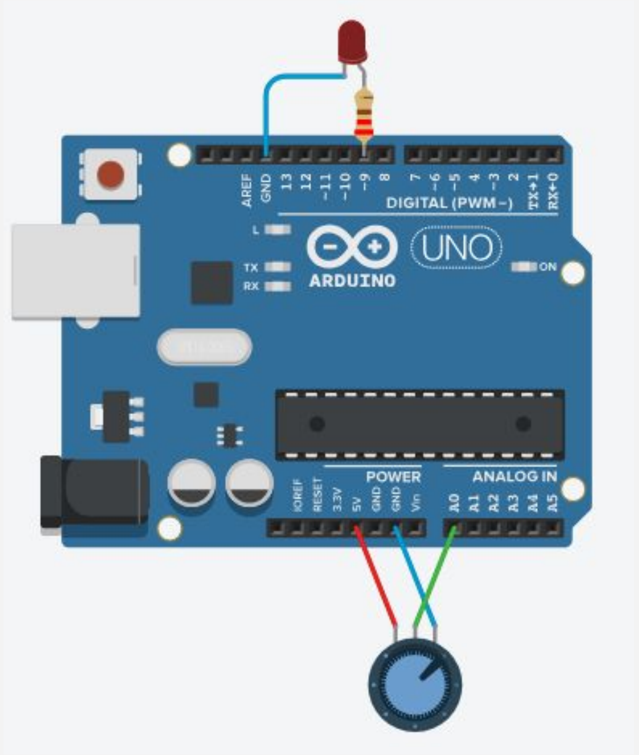
Potentiometer Controlled LED Brightness Sketch

with comments



```
// Connect LED to digital pin 9 (PWM pin)
const int led_pin = 9;
// Connect Potentiometer to analog pin A0
const int pot_pin = A0;

void setup() {
  // Set pot_pin as input
  pinMode(pot_pin, INPUT);
  // Set led_pin as output
  pinMode(led_pin, OUTPUT);
}

void loop() {
  // Get the value of the pot_pin (0 ---> 1023)
  // and store it in pot_reading
  int pot_reading = analogRead(pot_pin);
  // Map that value into a new range that is suitable
  // for LED brightness (0 ---> 255)
  // and store it in a new variable
  int pot_filtered_reading = map(pot_reading, 0, 1023, 0, 255);

  // Send the filtered value to analog LED pin
  analogWrite(led_pin, pot_filtered_reading);
}
```

Connection Circuit



Sketch Notes
map

map() function maps a number from one range to another. When controlling the brightness of an LED, the suitable range is 0 → 255:

- 0: LED is off
- 255: LED is on with maximum brightness

Example: map(variable, 0, 1023, 0, 255);

// Changes the range of the number stored in *variable* to a new range.

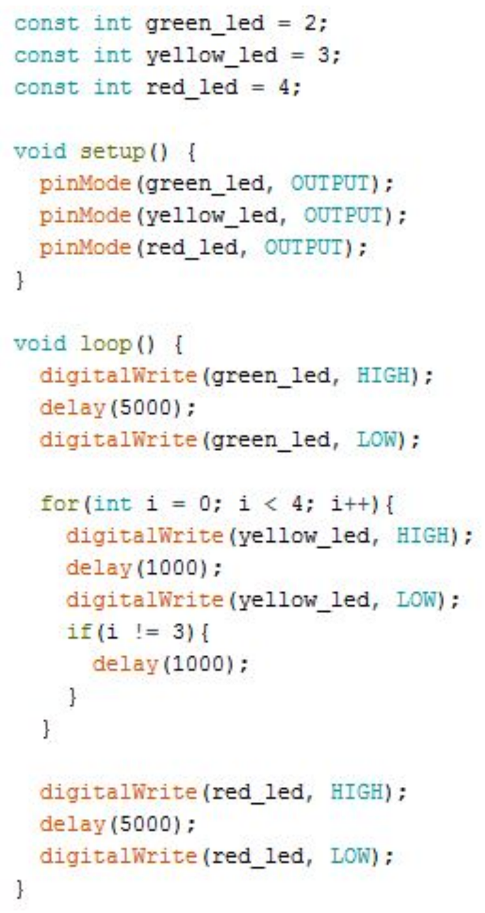
Sketch Notes
analogWrite

analogWrite() sends out an analog value to a PWM pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds.

Example: analogWrite(led_pin, 255);

// Turns on an LED with maximum brightness.

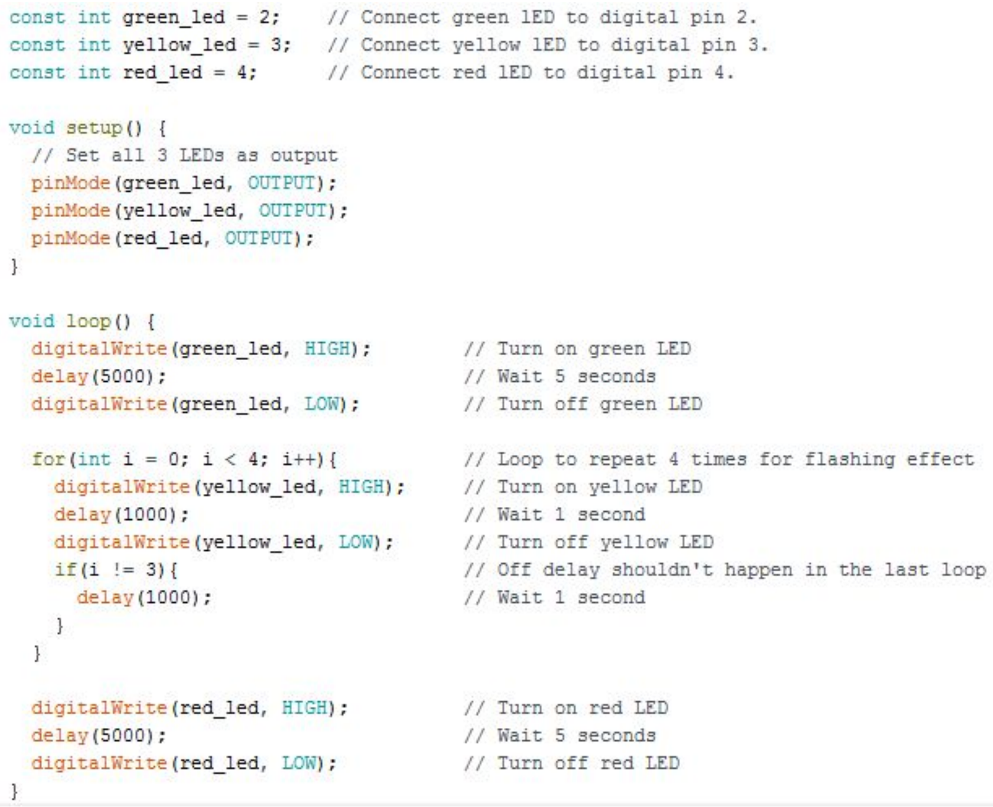
Traffic Light Sketch



```
const int green_led = 2;
const int yellow_led = 3;
const int red_led = 4;

void setup() {
  pinMode(green_led, OUTPUT);
  pinMode(yellow_led, OUTPUT);
  pinMode(red_led, OUTPUT);
}

void loop() {
  digitalWrite(green_led, HIGH);
  delay(5000);
  digitalWrite(green_led, LOW);

  for(int i = 0; i < 4; i++){
    digitalWrite(yellow_led, HIGH);
    delay(1000);
    digitalWrite(yellow_led, LOW);
    if(i != 3){
      delay(1000);
    }
  }

  digitalWrite(red_led, HIGH);
  delay(5000);
  digitalWrite(red_led, LOW);
}
```

without comments

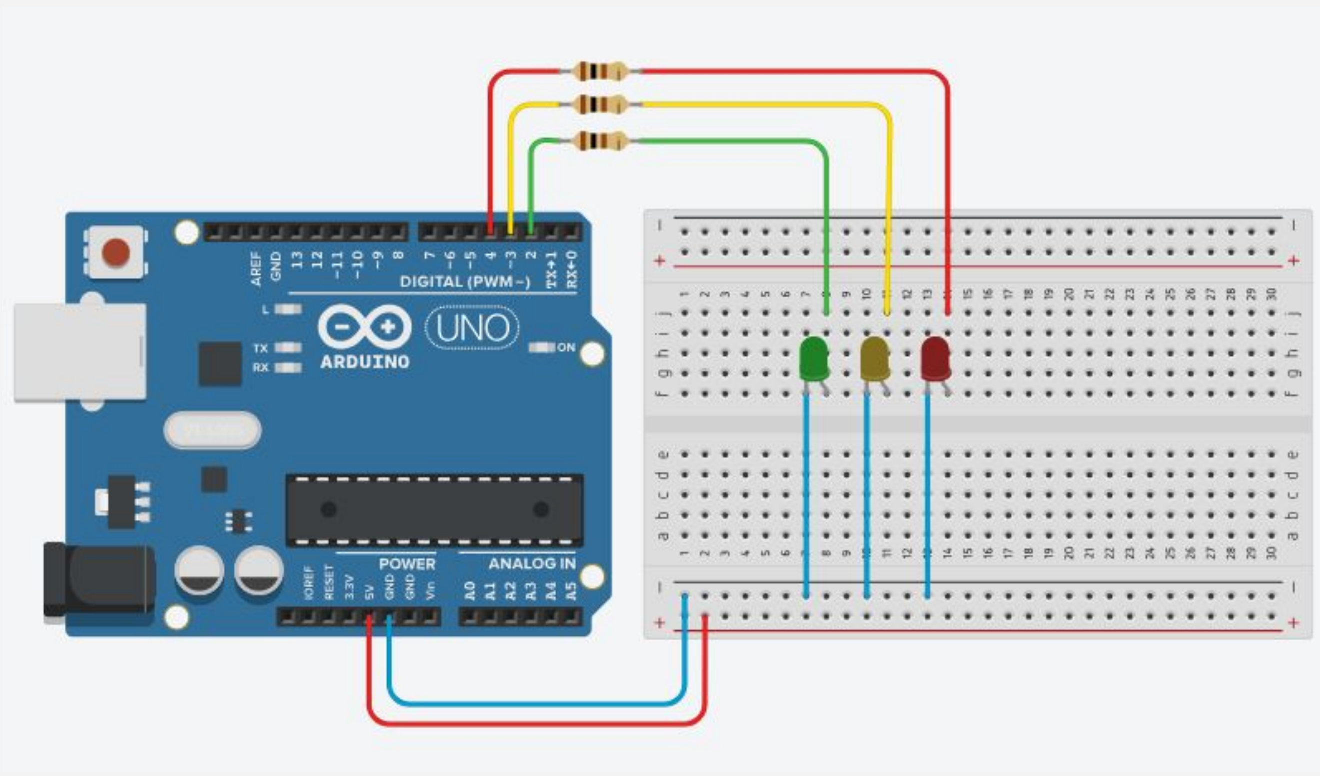
Traffic Light Sketch



```
const int green_led = 2;    // Connect green lED to digital pin 2.
const int yellow_led = 3;   // Connect yellow lED to digital pin 3.
const int red_led = 4;      // Connect red lED to digital pin 4.

void setup() {
  // Set all 3 LEDs as output
  pinMode(green_led, OUTPUT);
  pinMode(yellow_led, OUTPUT);
  pinMode(red_led, OUTPUT);
}

void loop() {
  digitalWrite(green_led, HIGH);         // Turn on green LED
  delay(5000);                           // Wait 5 seconds
  digitalWrite(green_led, LOW);          // Turn off green LED

  for(int i = 0; i < 4; i++){            // Loop to repeat 4 times for flashing effect
    digitalWrite(yellow_led, HIGH);      // Turn on yellow LED
    delay(1000);                         // Wait 1 second
    digitalWrite(yellow_led, LOW);       // Turn off yellow LED
    if(i != 3){                          // Off delay shouldn't happen in the last loop
      delay(1000);                       // Wait 1 second
    }
  }

  digitalWrite(red_led, HIGH);           // Turn on red LED
  delay(5000);                           // Wait 5 seconds
  digitalWrite(red_led, LOW);            // Turn off red LED
}
```

with comments

Connection Circuit

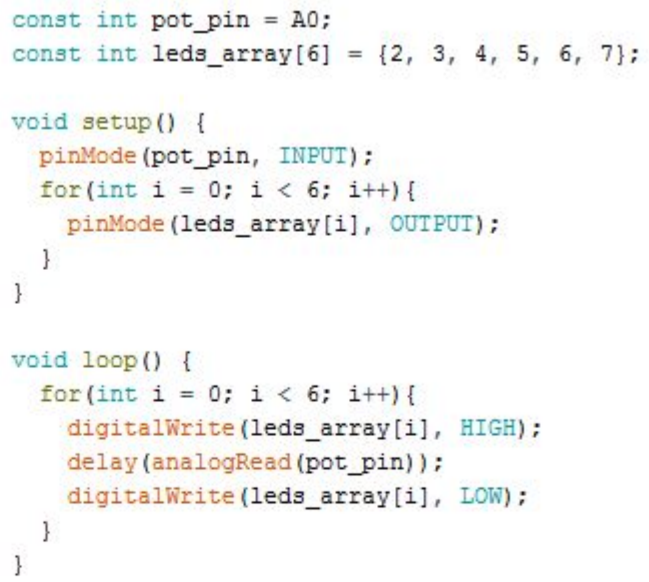


Sketch Notes
For Loop

The for statement is used to repeat a block of code enclosed in curly braces. An increment counter i is usually used to increment and terminate the loop.

The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

Running Light Sketch

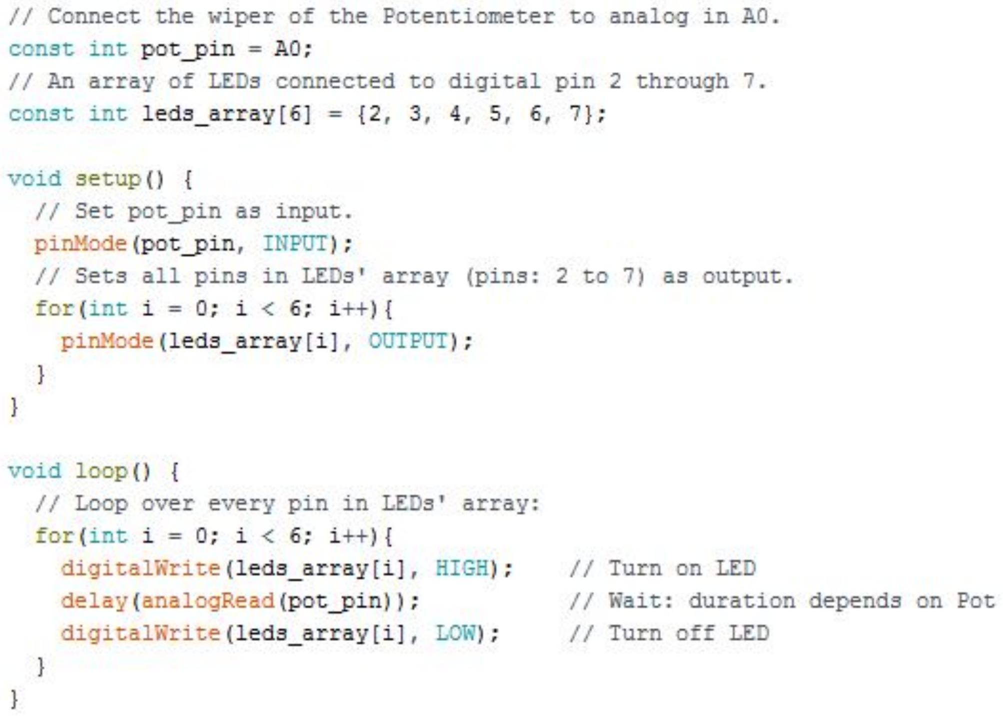


```
const int pot_pin = A0;
const int leds_array[6] = {2, 3, 4, 5, 6, 7};

void setup() {
  pinMode(pot_pin, INPUT);
  for(int i = 0; i < 6; i++){
    pinMode(leds_array[i], OUTPUT);
  }
}

void loop() {
  for(int i = 0; i < 6; i++){
    digitalWrite(leds_array[i], HIGH);
    delay(analogRead(pot_pin));
    digitalWrite(leds_array[i], LOW);
  }
}
```

without comments

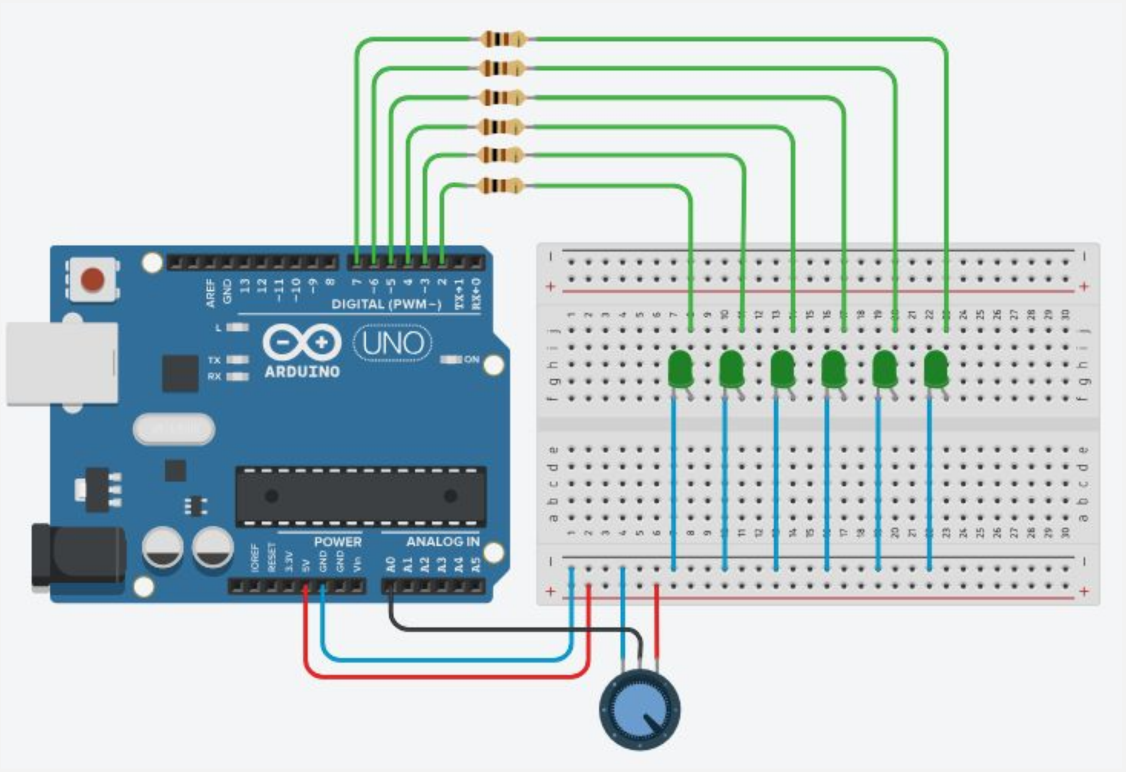
Running Light Sketch

with comments



```
// Connect the wiper of the Potentiometer to analog in A0.
const int pot_pin = A0;
// An array of LEDs connected to digital pin 2 through 7.
const int leds_array[6] = {2, 3, 4, 5, 6, 7};

void setup() {
  // Set pot_pin as input.
  pinMode(pot_pin, INPUT);
  // Sets all pins in LEDs' array (pins: 2 to 7) as output.
  for(int i = 0; i < 6; i++){
    pinMode(leds_array[i], OUTPUT);
  }
}

void loop() {
  // Loop over every pin in LEDs' array:
  for(int i = 0; i < 6; i++){
    digitalWrite(leds_array[i], HIGH);    // Turn on LED
    delay(analogRead(pot_pin));           // Wait: duration depends on Pot
    digitalWrite(leds_array[i], LOW);     // Turn off LED
  }
}
```

Connection Circuit



Sketch Notes
Arrays

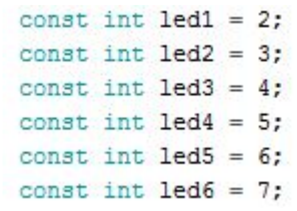
An array is a collection/list of variables that are accessed with an index number (the number between brackets).

Example: int example_array[4] = {1, 2, 3, 4};

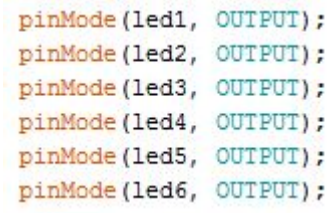
- example_array[0] = 1
- example_array[1] = 2
- example_array[3] = 3
- example_array[4] = 4

Sketch Notes
For-Array Combination

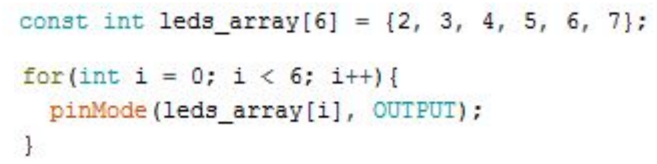
Combining for loops with arrays can be very useful as shown below:



```
const int led1 = 2;
const int led2 = 3;
const int led3 = 4;
const int led4 = 5;
const int led5 = 6;
const int led6 = 7;
```

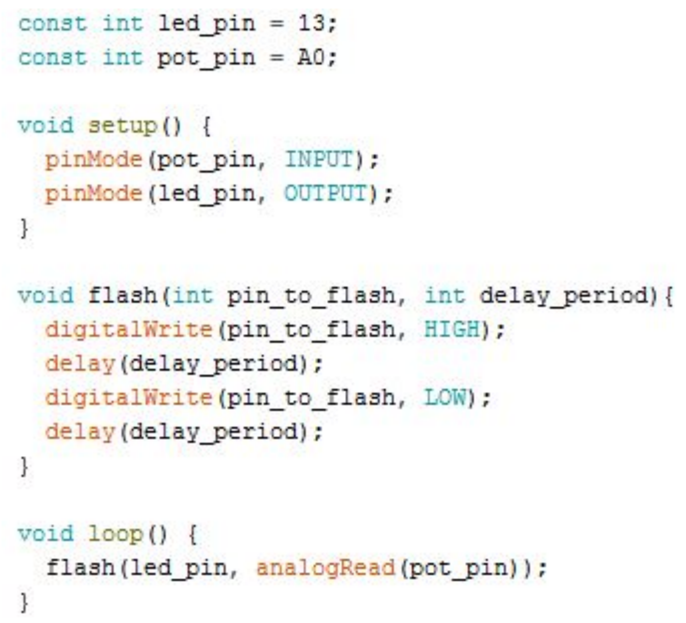


```
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
pinMode(led3, OUTPUT);
pinMode(led4, OUTPUT);
pinMode(led5, OUTPUT);
pinMode(led6, OUTPUT);
```



```
const int leds_array[6] = {2, 3, 4, 5, 6, 7};

for(int i = 0; i < 6; i++){
  pinMode(leds_array[i], OUTPUT);
}
```

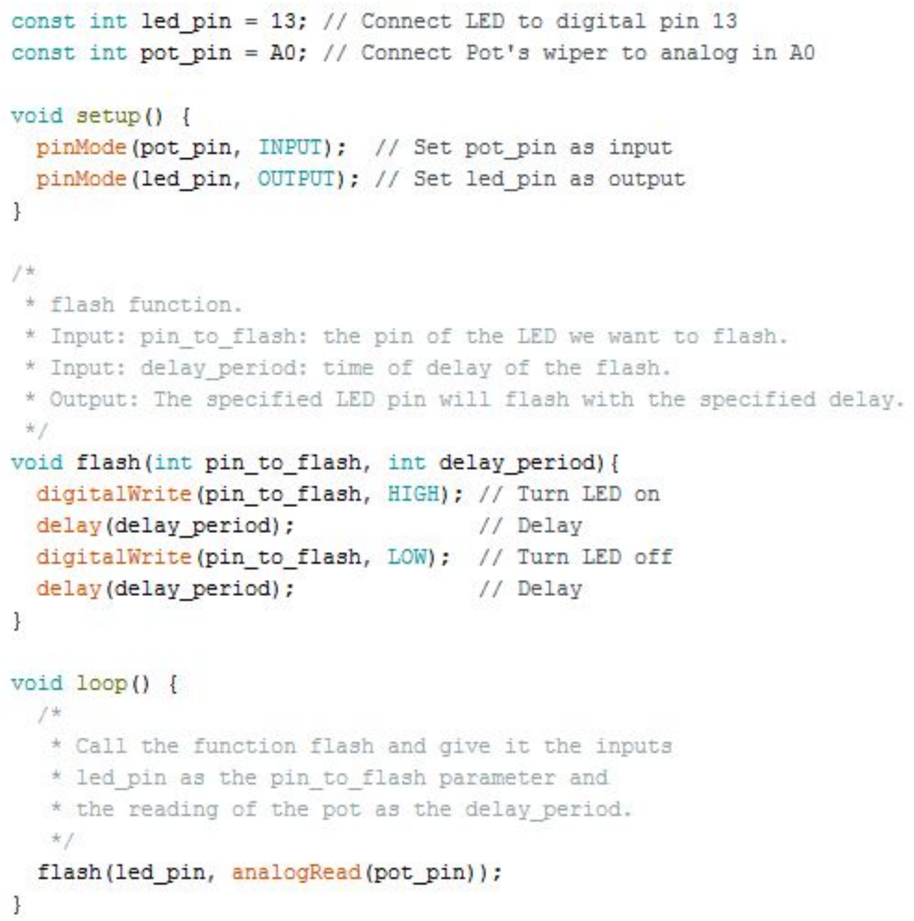
Potentiometer Controlled Blink Rate (V2) Sketch



```
const int led_pin = 13;
const int pot_pin = A0;

void setup() {
  pinMode(pot_pin, INPUT);
  pinMode(led_pin, OUTPUT);
}

void flash(int pin_to_flash, int delay_period){
  digitalWrite(pin_to_flash, HIGH);
  delay(delay_period);
  digitalWrite(pin_to_flash, LOW);
  delay(delay_period);
}

void loop() {
  flash(led_pin, analogRead(pot_pin));
}
```

without comments

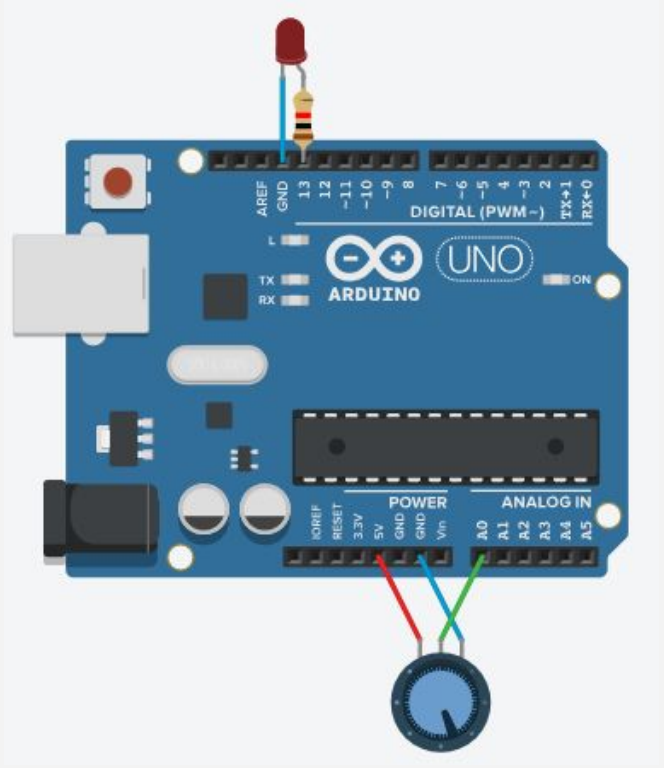
Potentiometer Controlled Blink Rate (V2) Sketch



```
const int led_pin = 13; // Connect LED to digital pin 13
const int pot_pin = A0; // Connect Pot's wiper to analog in A0

void setup() {
  pinMode(pot_pin, INPUT);  // Set pot_pin as input
  pinMode(led_pin, OUTPUT); // Set led_pin as output
}

/*
 * flash function.
 * Input: pin_to_flash: the pin of the LED we want to flash.
 * Input: delay_period: time of delay of the flash.
 * Output: The specified LED pin will flash with the specified delay.
 */
void flash(int pin_to_flash, int delay_period){
  digitalWrite(pin_to_flash, HIGH); // Turn LED on
  delay(delay_period);              // Delay
  digitalWrite(pin_to_flash, LOW);  // Turn LED off
  delay(delay_period);              // Delay
}

void loop() {
  /*
   * Call the function flash and give it the inputs
   * led_pin as the pin_to_flash parameter and
   * the reading of the pot as the delay_period.
   */
  flash(led_pin, analogRead(pot_pin));
}
```

with comments

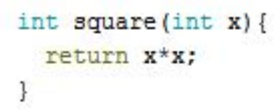
Connection Circuit



Sketch Notes
Functions

A **function** is a box which receives and input and gives a certain output.

Example: the function *f(x)* = x^2 takes as input some x and returns as output that value squared.

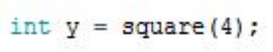


```
int square(int x){
  return x*x;
}
```

Here we created a function named "*square*" and declared it as an int because it returns (outputs) an integer. Between the parentheses we put the inputs of the function which we call the function's parameters. The input of the square function is an int x. The function takes that value and returns its square.

Sketch Notes
Functions

When we want to execute a function, we must call it:



```
int y = square(4);
```

By calling the function as shown above, the code within the function is executed and returns a 16, which is stored in the variable y.

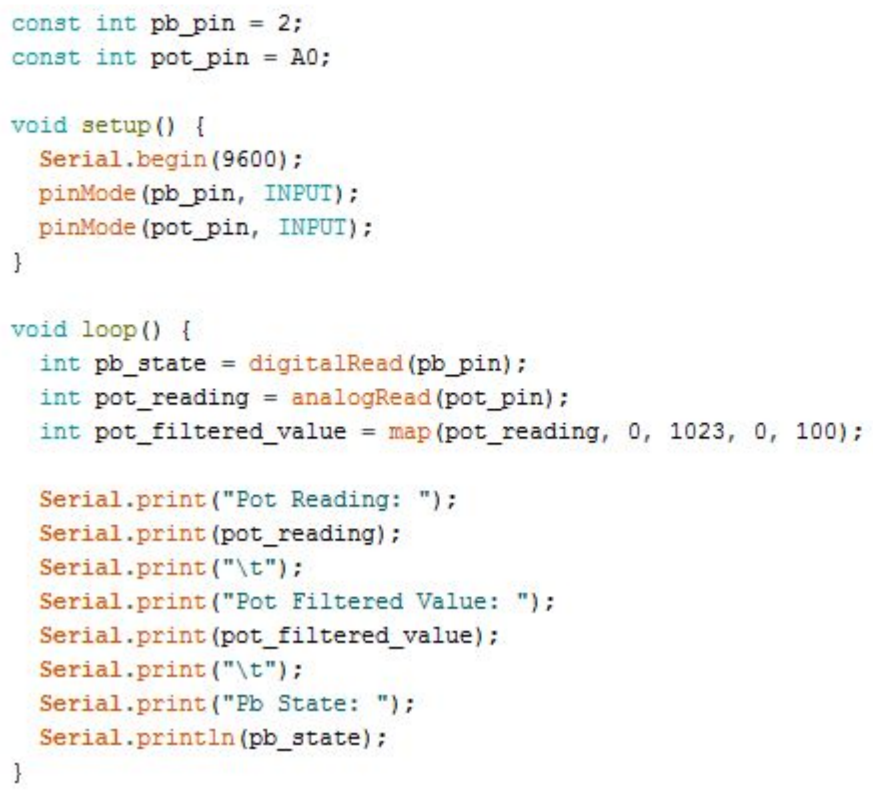
Sketch Notes

Functions

So, all we have really done here is to move the four lines of code that flash the LED from the void loop to be in a function of their own called *flash*. Now you can make the LED flash any time you like by just calling the new function by writing *flash*(pin, period).

However in our sketch, the function doesn't return anything. It simply just executes a chunk of code, which is why we declared it as a void.

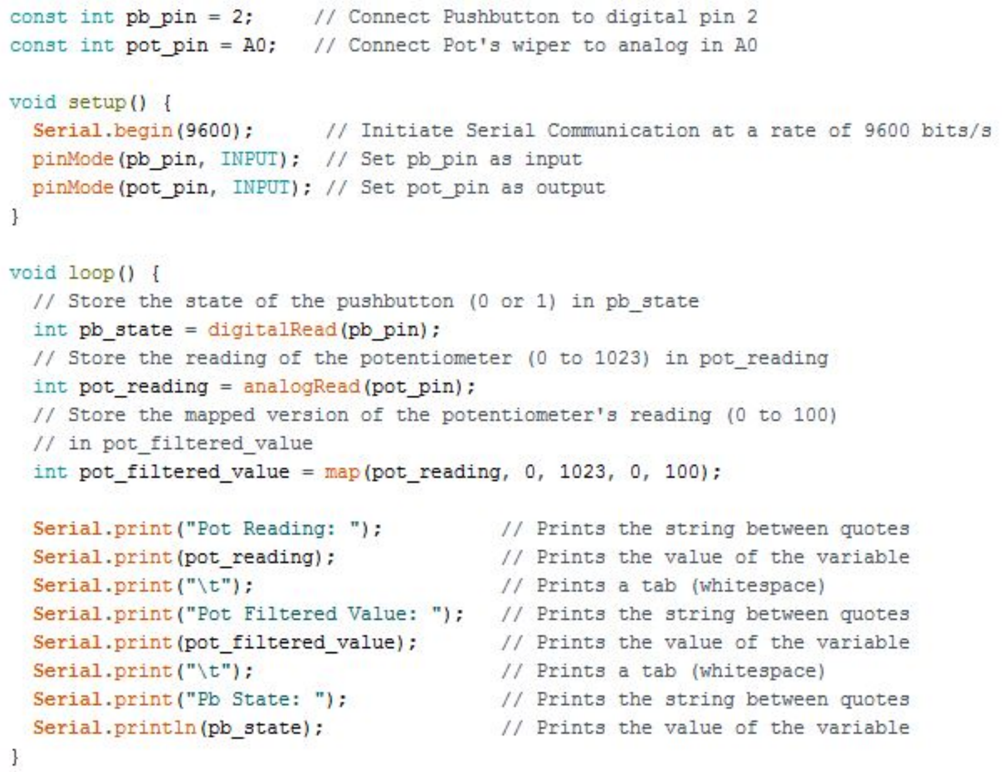
Serial Out Sketch



```
const int pb_pin = 2;
const int pot_pin = A0;

void setup() {
  Serial.begin(9600);
  pinMode(pb_pin, INPUT);
  pinMode(pot_pin, INPUT);
}

void loop() {
  int pb_state = digitalRead(pb_pin);
  int pot_reading = analogRead(pot_pin);
  int pot_filtered_value = map(pot_reading, 0, 1023, 0, 100);

  Serial.print("Pot Reading: ");
  Serial.print(pot_reading);
  Serial.print("\t");
  Serial.print("Pot Filtered Value: ");
  Serial.print(pot_filtered_value);
  Serial.print("\t");
  Serial.print("Pb State: ");
  Serial.println(pb_state);
}
```

without comments

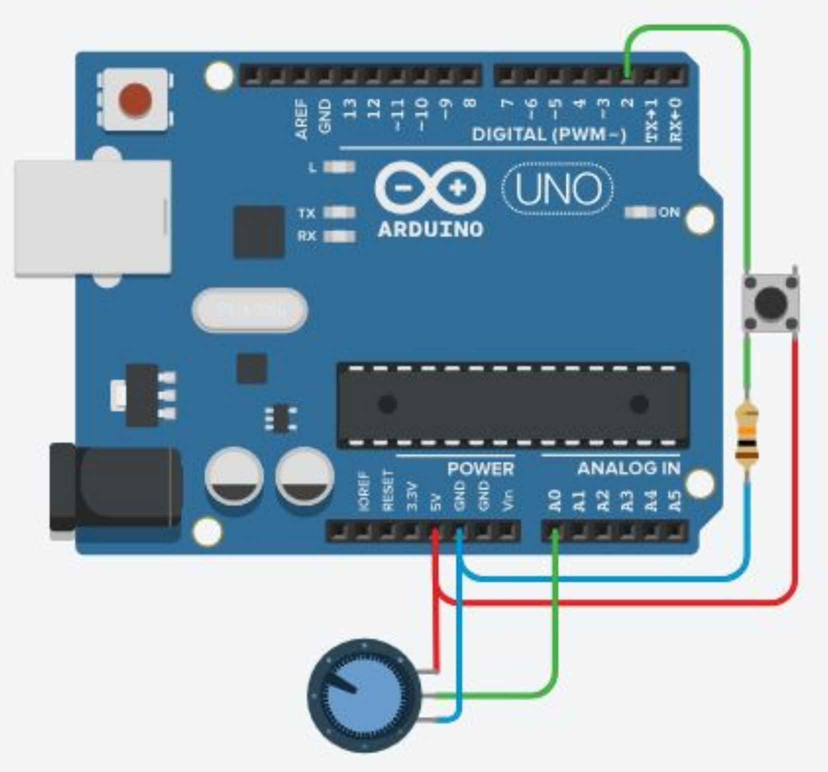
Serial Out Sketch



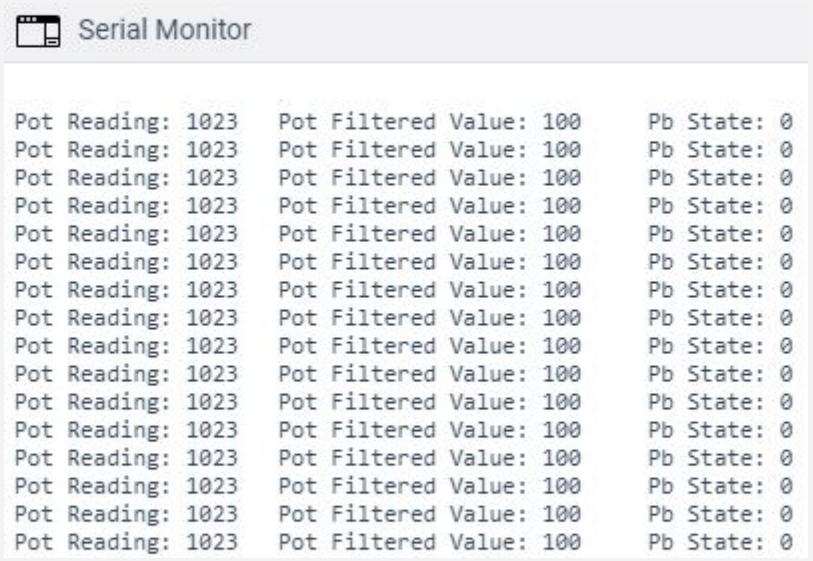
```
const int pb_pin = 2;     // Connect Pushbutton to digital pin 2
const int pot_pin = A0;   // Connect Pot's wiper to analog in A0

void setup() {
  Serial.begin(9600);       // Initiate Serial Communication at a rate of 9600 bits/s
  pinMode(pb_pin, INPUT);  // Set pb_pin as input
  pinMode(pot_pin, INPUT); // Set pot_pin as output
}

void loop() {
  // Store the state of the pushbutton (0 or 1) in pb_state
  int pb_state = digitalRead(pb_pin);
  // Store the reading of the potentiometer (0 to 1023) in pot_reading
  int pot_reading = analogRead(pot_pin);
  // Store the mapped version of the potentiometer's reading (0 to 100)
  // in pot_filtered_value
  int pot_filtered_value = map(pot_reading, 0, 1023, 0, 100);

  Serial.print("Pot Reading: ");          // Prints the string between quotes
  Serial.print(pot_reading);              // Prints the value of the variable
  Serial.print("\t");                     // Prints a tab (whitespace)
  Serial.print("Pot Filtered Value: ");   // Prints the string between quotes
  Serial.print(pot_filtered_value);       // Prints the value of the variable
  Serial.print("\t");                     // Prints a tab (whitespace)
  Serial.print("Pb State: ");             // Prints the string between quotes
  Serial.println(pb_state);               // Prints the value of the variable
}
```

with comments

Connection Circuit



Serial Monitor


Serial Monitor

```
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
Pot Reading: 1023   Pot Filtered Value: 100     Pb State: 0
```

Sketch Notes
Serial Communication

Serial communication refers to a way for Arduino to send and receive data from a laptop or other devices.

The most basic serial communication that you can do with an Arduino is to send data (for example: a sensor's readings) from Arduino to the computer.

To do that we use the following functions:

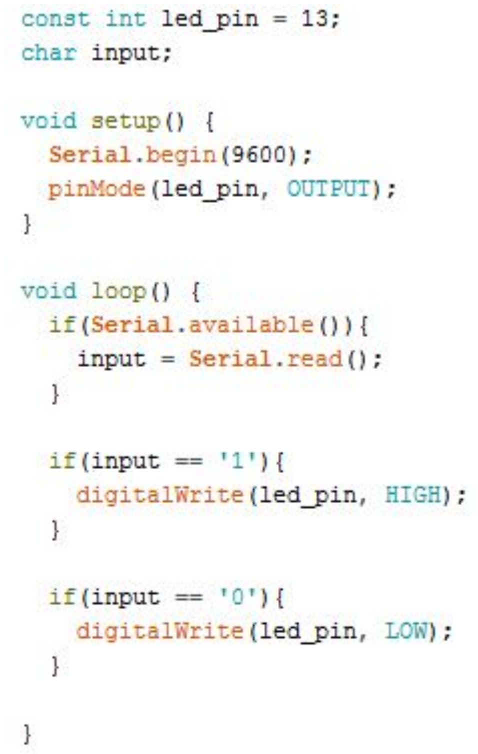
➢ Serial.begin(baud_rate);

// used in void setup() to initiate Serial Communication. Baud_rate is 9600 per standard.

➢ Serial.print("Message");
➢ Serial.println("Message");

// Serial.println does the same as Serial.print but whatever comes after Serial.println is printed on a new line.

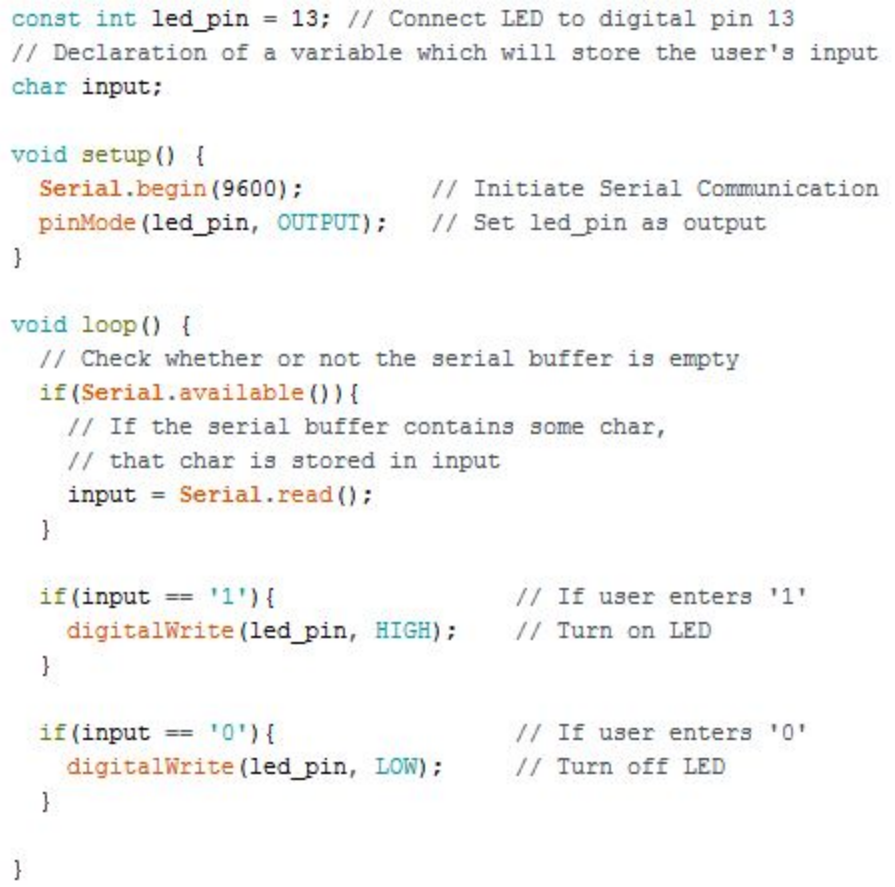
Serial In Sketch



```
const int led_pin = 13;
char input;

void setup() {
  Serial.begin(9600);
  pinMode(led_pin, OUTPUT);
}

void loop() {
  if(Serial.available()){
    input = Serial.read();
  }

  if(input == '1'){
    digitalWrite(led_pin, HIGH);
  }

  if(input == '0'){
    digitalWrite(led_pin, LOW);
  }

}
```

without comments

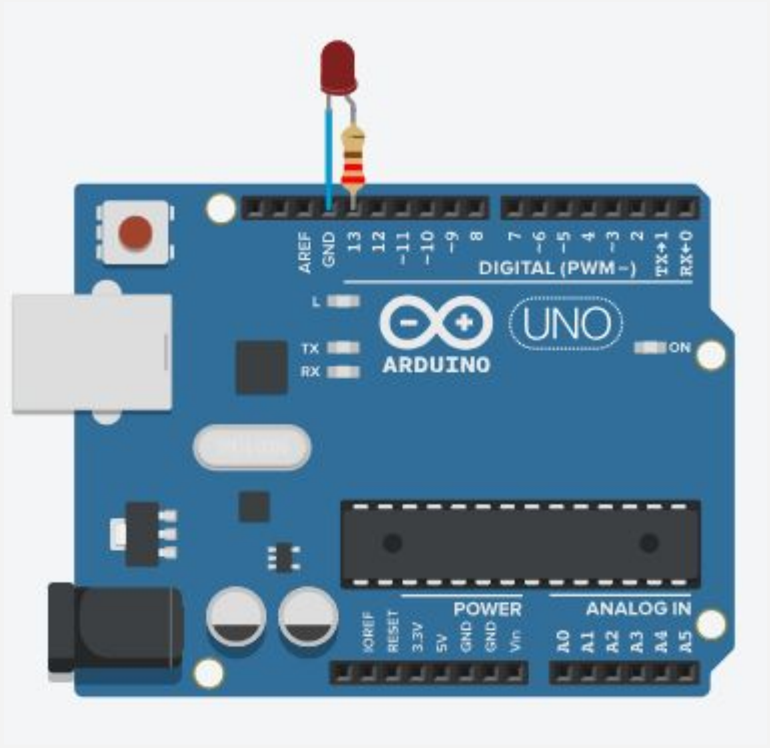
Serial In Sketch



```
const int led_pin = 13; // Connect LED to digital pin 13
// Declaration of a variable which will store the user's input
char input;

void setup() {
  Serial.begin(9600);          // Initiate Serial Communication
  pinMode(led_pin, OUTPUT);    // Set led_pin as output
}

void loop() {
  // Check whether or not the serial buffer is empty
  if(Serial.available()){
    // If the serial buffer contains some char,
    // that char is stored in input
    input = Serial.read();
  }

  if(input == '1'){                  // If user enters '1'
    digitalWrite(led_pin, HIGH);     // Turn on LED
  }

  if(input == '0'){                  // If user enters '0'
    digitalWrite(led_pin, LOW);      // Turn off LED
  }

}
```

with comments

Connection Circuit



Sketch Notes
Serial Input

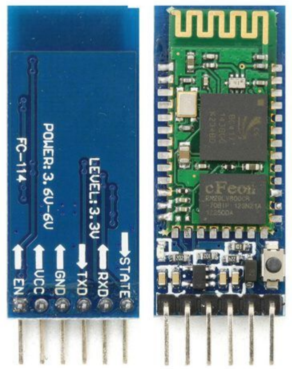
Serial input refers to sending data from a computer or some other device to the Arduino.

This is done by using the following 2 commands:

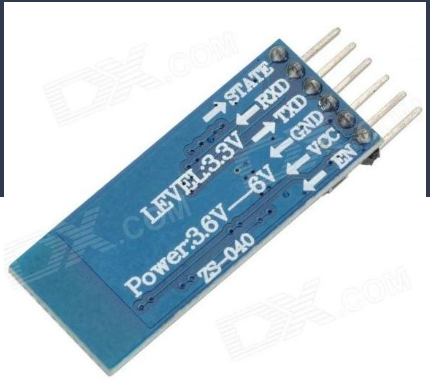
- ➢ Serial.available() // returns the number of characters (or bytes) that are currently stored in the Arduino's incoming serial buffer. If the user doesn't input anything, it returns 0. So we use it to check whether or not the user has entered some character.
- ➢ Serial.read() // reads and returns the character that is available in the buffer (entered by the user), which we typically store in a variable declared as a char.

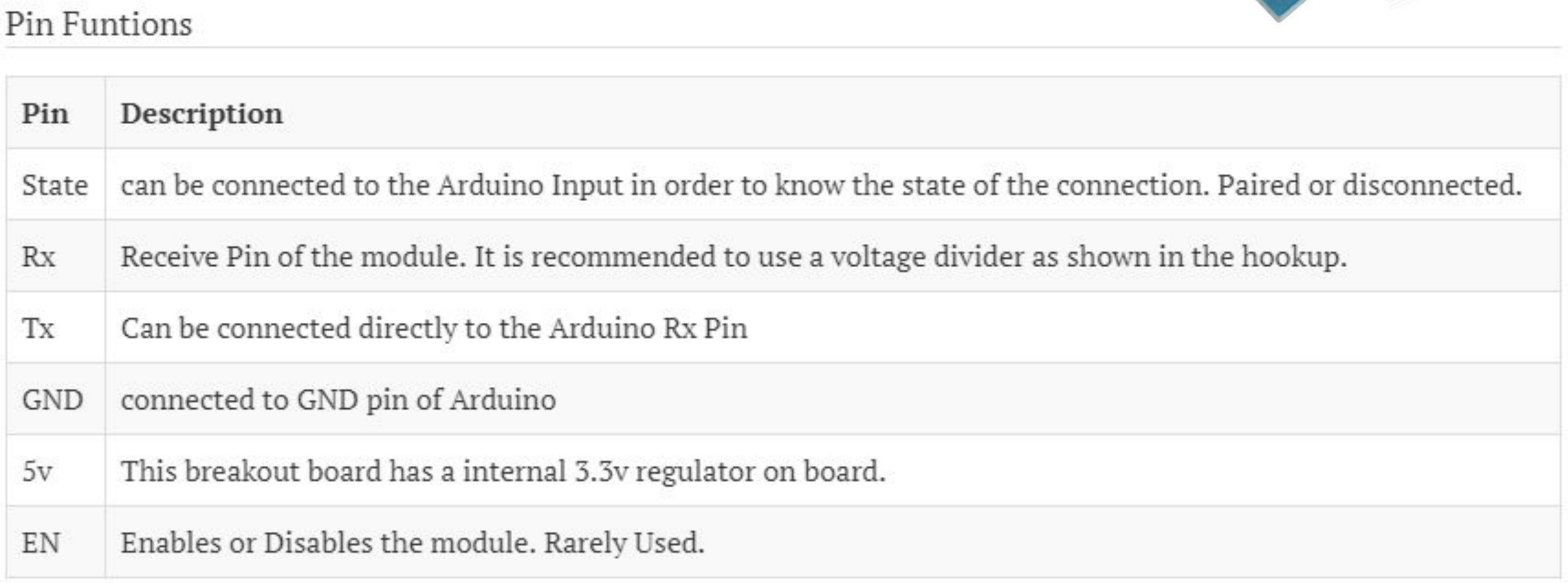
Bluetooth Module HC-05

In order for the Arduino to communicate with our smartphones/laptops, we'll use a Bluetooth module called HC-05.



HC-05 Pinout

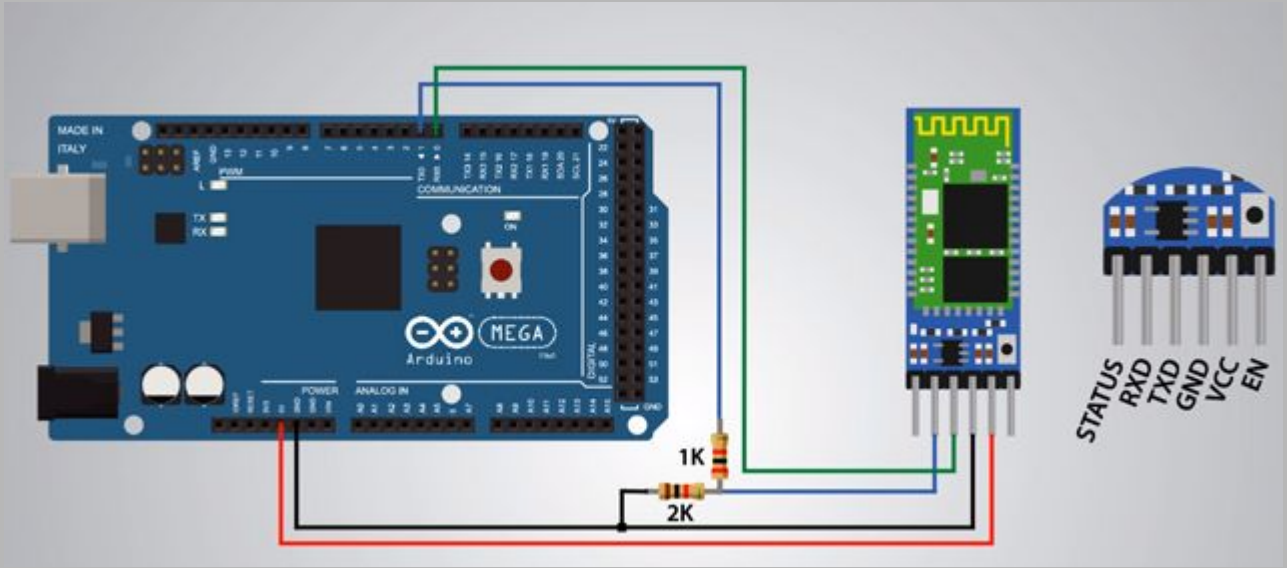




Pin Funtions

Pin	Description
State	can be connected to the Arduino Input in order to know the state of the connection. Paired or disconnected.
Rx	Receive Pin of the module. It is recommended to use a voltage divider as shown in the hookup.
Tx	Can be connected directly to the Arduino Rx Pin
GND	connected to GND pin of Arduino
5v	This breakout board has a internal 3.3v regulator on board.
EN	Enables or Disables the module. Rarely Used.

HC-05 Schematic

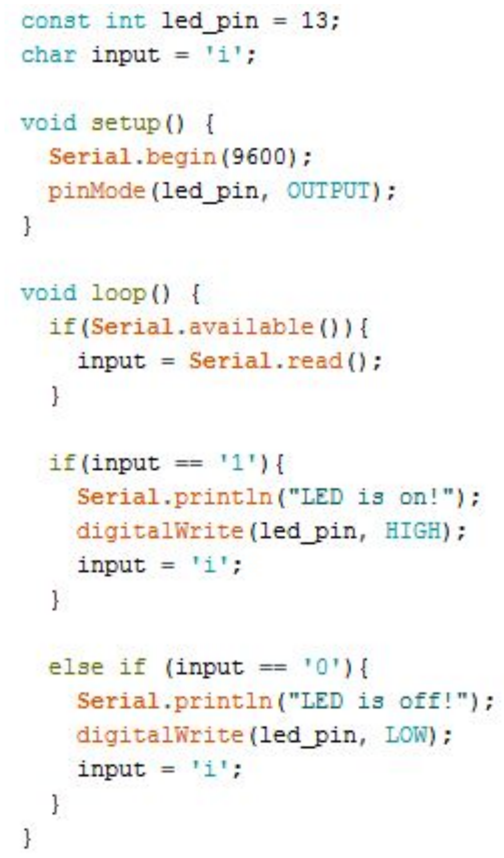


HC-05 Note

Before uploading the code, we should unplug the TX and RX lines because when uploading, the Arduino uses the serial communication, so the pins RX (digital pin 0) and TX (digital pin 1) are busy. However, this is unnecessary when using Arduino Mega, because it has multiple Serial Communication ports.

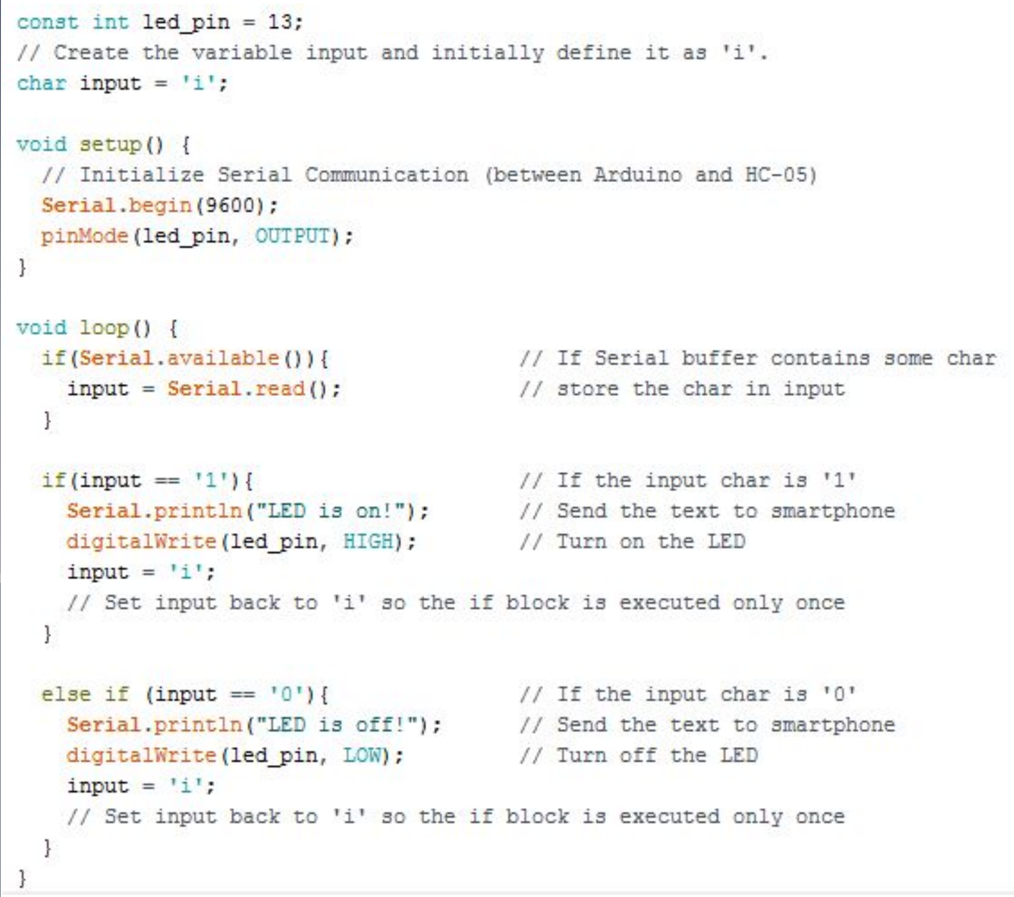
Smartphone Controlled LED

without comments



```
const int led_pin = 13;
char input = 'i';

void setup() {
  Serial.begin(9600);
  pinMode(led_pin, OUTPUT);
}

void loop() {
  if(Serial.available()){
    input = Serial.read();
  }

  if(input == '1'){
    Serial.println("LED is on!");
    digitalWrite(led_pin, HIGH);
    input = 'i';
  }

  else if (input == '0'){
    Serial.println("LED is off!");
    digitalWrite(led_pin, LOW);
    input = 'i';
  }
}
```

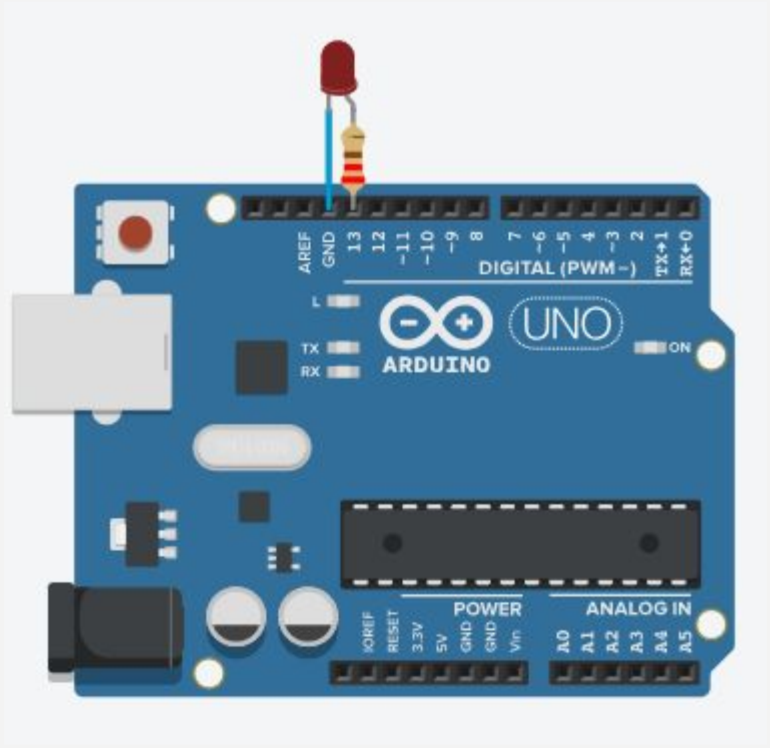
Smartphone Controlled LED



```
const int led_pin = 13;
// Create the variable input and initially define it as 'i'.
char input = 'i';

void setup() {
  // Initialize Serial Communication (between Arduino and HC-05)
  Serial.begin(9600);
  pinMode(led_pin, OUTPUT);
}

void loop() {
  if(Serial.available()){                    // If Serial buffer contains some char
    input = Serial.read();                   // store the char in input
  }

  if(input == '1'){                          // If the input char is '1'
    Serial.println("LED is on!");            // Send the text to smartphone
    digitalWrite(led_pin, HIGH);             // Turn on the LED
    input = 'i';
    // Set input back to 'i' so the if block is executed only once
  }

  else if (input == '0'){                    // If the input char is '0'
    Serial.println("LED is off!");           // Send the text to smartphone
    digitalWrite(led_pin, LOW);              // Turn off the LED
    input = 'i';
    // Set input back to 'i' so the if block is executed only once
  }
}
```

with comments

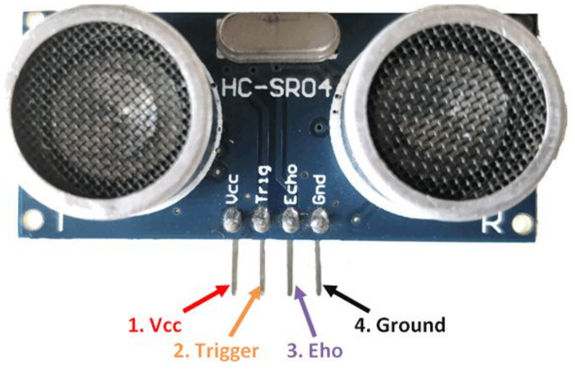
Connection Circuit



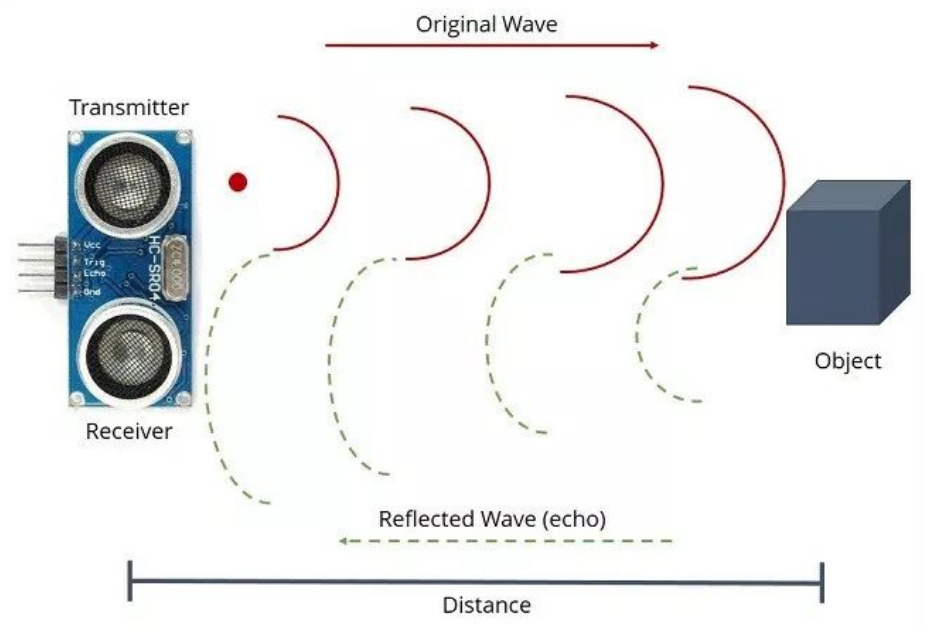
Ultrasonic Distance Sensor HC-SR04

The Ultrasonic module measures the distance between the car and any object in front of it.

The **trig** pin (input) transmits an ultrasonic wave (high frequency sound). When the signal finds an object, it is reflected and the **echo** pin (output) receives it.



Ultrasonic Distance Sensor HC-SR04



Distance = Speed · Time

- **Speed** is the known speed of sound in air (330m/s).
- **Time** is the time between transmission of the wave and its reception (calculated by the sensor).

Ultrasonic Distance Sensor HC-SR04

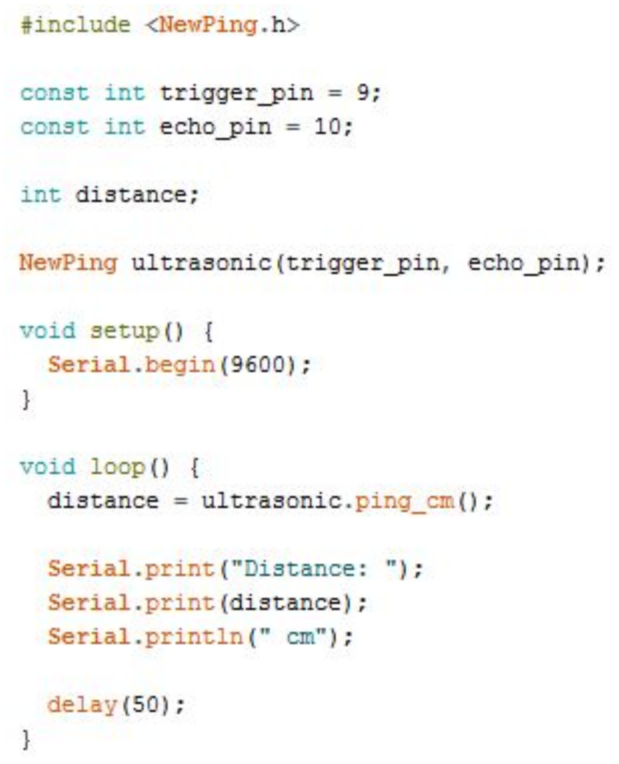
To start the measurement, the **Trigger** pin has to be made high for 10uS and then turned off. This action will trigger an ultrasonic wave from the transmitter and the receiver will wait for the wave to return. Once the wave is returned, after getting reflected by any object, the **Echo** pin goes high for a particular amount of time which will be equal to the time taken for the wave to return back to the sensor. Thus, allowing the sensor to measure the distance.

NewPing Library

By including this library, the code will become much simpler and it’ll make our life easier.

A Ping is simply the process of sending a signal and receiving it.

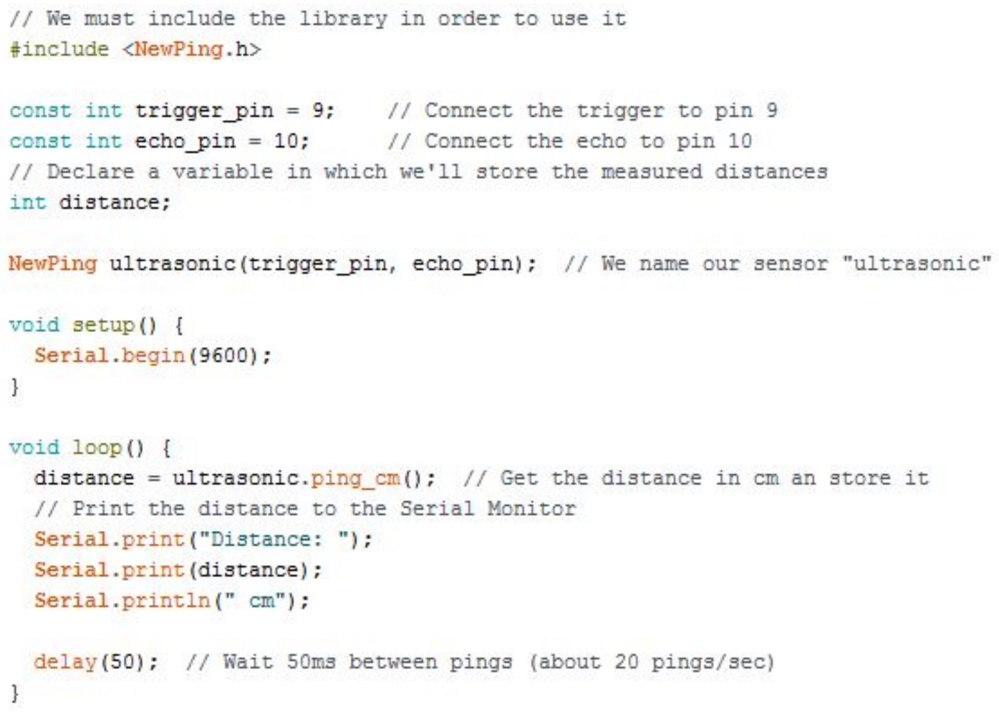
Ultrasonic Serial Out



```
#include <NewPing.h>

const int trigger_pin = 9;
const int echo_pin = 10;

int distance;

NewPing ultrasonic(trigger_pin, echo_pin);

void setup() {
  Serial.begin(9600);
}

void loop() {
  distance = ultrasonic.ping_cm();

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(50);
}
```

without comments

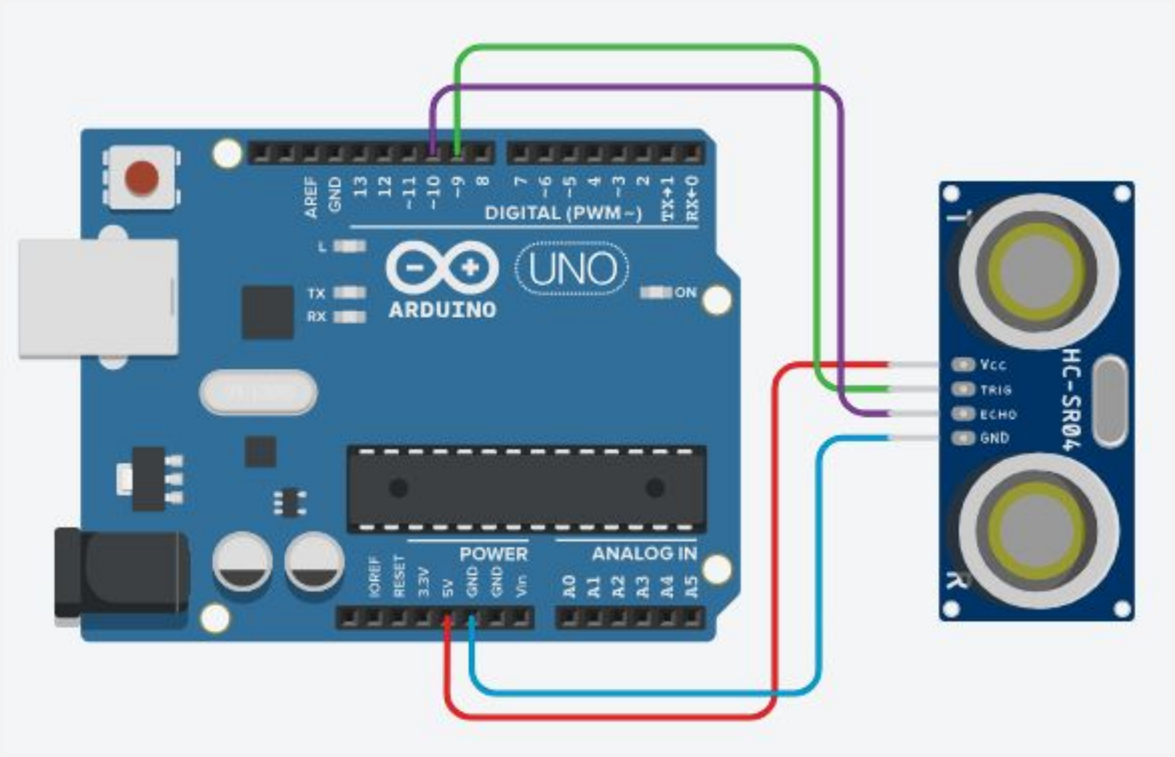
Ultrasonic Serial Out



```
// We must include the library in order to use it
#include <NewPing.h>

const int trigger_pin = 9;    // Connect the trigger to pin 9
const int echo_pin = 10;      // Connect the echo to pin 10
// Declare a variable in which we'll store the measured distances
int distance;

NewPing ultrasonic(trigger_pin, echo_pin);  // We name our sensor "ultrasonic"

void setup() {
  Serial.begin(9600);
}

void loop() {
  distance = ultrasonic.ping_cm();  // Get the distance in cm an store it
  // Print the distance to the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(50);  // Wait 50ms between pings (about 20 pings/sec)
}
```

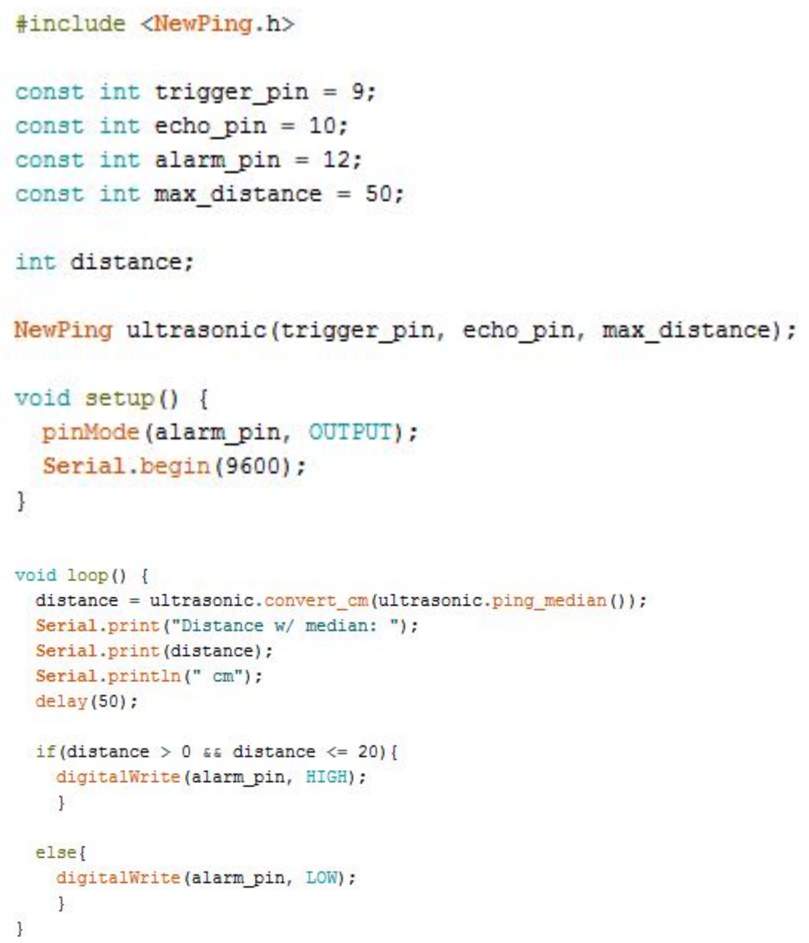
with comments

Connection Circuit



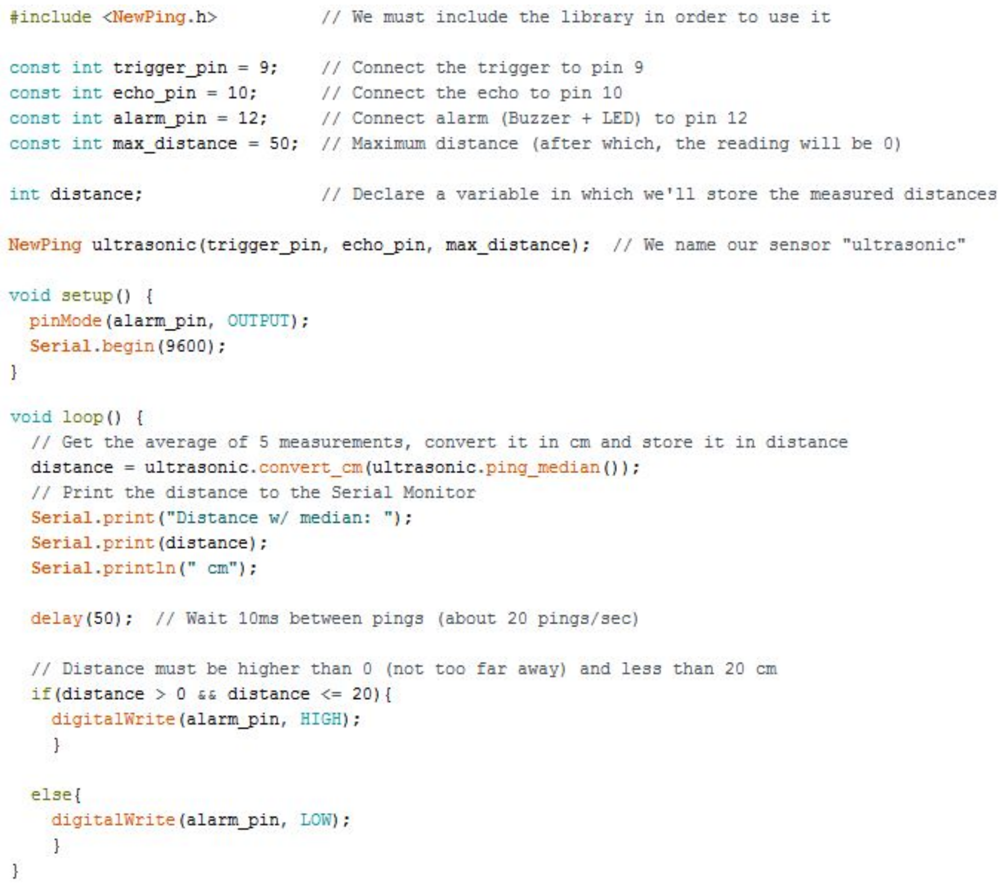
Ultrasonic Alarm

without comments



```
#include <NewPing.h>

const int trigger_pin = 9;
const int echo_pin = 10;
const int alarm_pin = 12;
const int max_distance = 50;

int distance;

NewPing ultrasonic(trigger_pin, echo_pin, max_distance);

void setup() {
  pinMode(alarm_pin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  distance = ultrasonic.convert_cm(ultrasonic.ping_median());
  Serial.print("Distance w/ median: ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(50);

  if(distance > 0 && distance <= 20){
    digitalWrite(alarm_pin, HIGH);
    }

  else{
    digitalWrite(alarm_pin, LOW);
    }
}
```

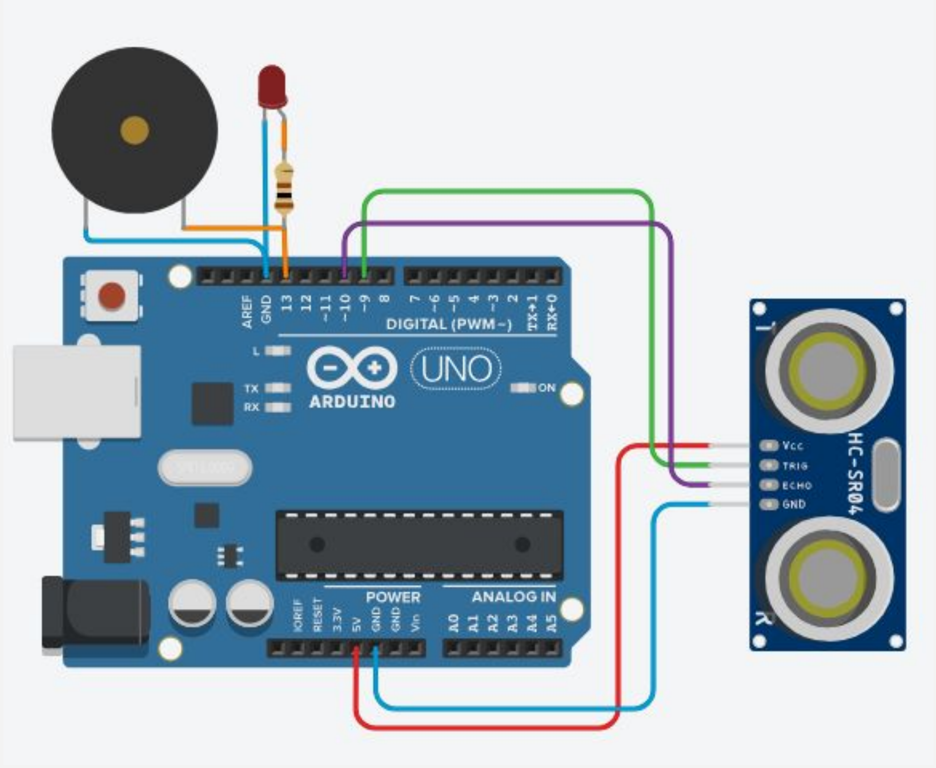
Ultrasonic Alarm



```
#include <NewPing.h>          // We must include the library in order to use it

const int trigger_pin = 9;    // Connect the trigger to pin 9
const int echo_pin = 10;      // Connect the echo to pin 10
const int alarm_pin = 12;     // Connect alarm (Buzzer + LED) to pin 12
const int max_distance = 50;  // Maximum distance (after which, the reading will be 0)

int distance;                 // Declare a variable in which we'll store the measured distances

NewPing ultrasonic(trigger_pin, echo_pin, max_distance);  // We name our sensor "ultrasonic"

void setup() {
  pinMode(alarm_pin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // Get the average of 5 measurements, convert it in cm and store it in distance
  distance = ultrasonic.convert_cm(ultrasonic.ping_median());
  // Print the distance to the Serial Monitor
  Serial.print("Distance w/ median: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(50);  // Wait 10ms between pings (about 20 pings/sec)

  // Distance must be higher than 0 (not too far away) and less than 20 cm
  if(distance > 0 && distance <= 20){
    digitalWrite(alarm_pin, HIGH);
    }

  else{
    digitalWrite(alarm_pin, LOW);
    }
}
```

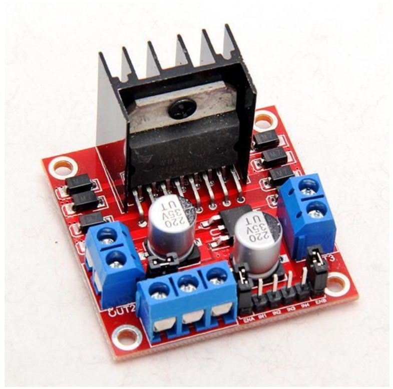
with comments

Connection Circuit



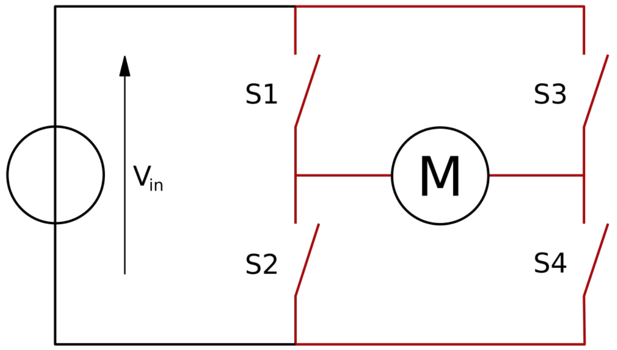
L298N Motor Driver Module

This is a dual H-bridge (explained in the next slide!) motor driver. Thus, capable of control the speed and direction of two DC motors, providing them with up to 2A of current.



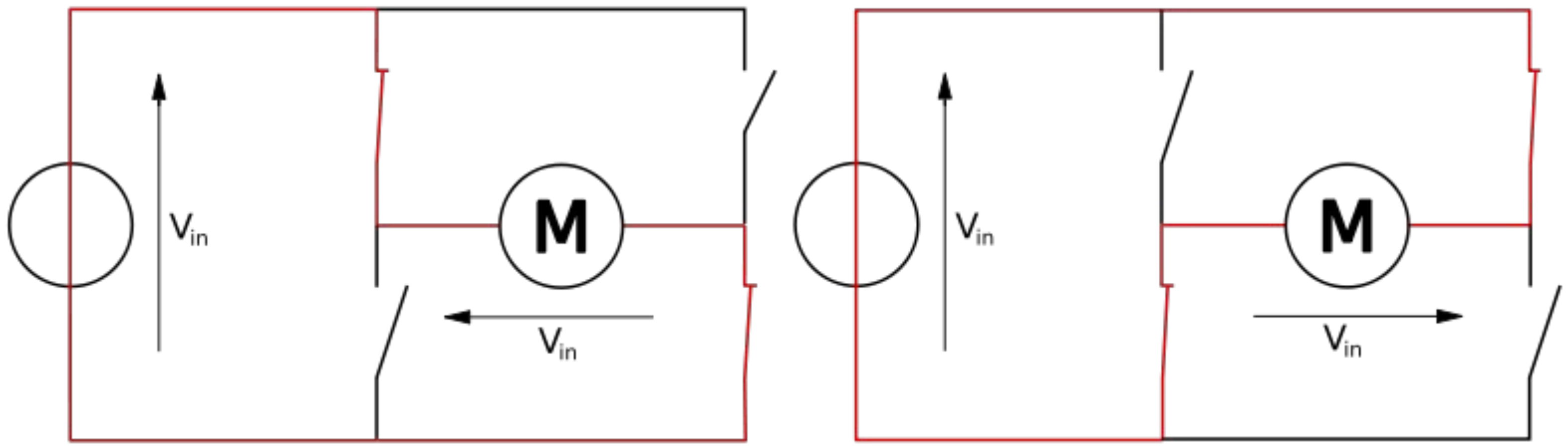
H-Bridge

An H-bridge is an electric circuit that switches the polarity of a voltage applied to a load. These circuits are often used in robotics to allow DC motors to run forwards or backwards.



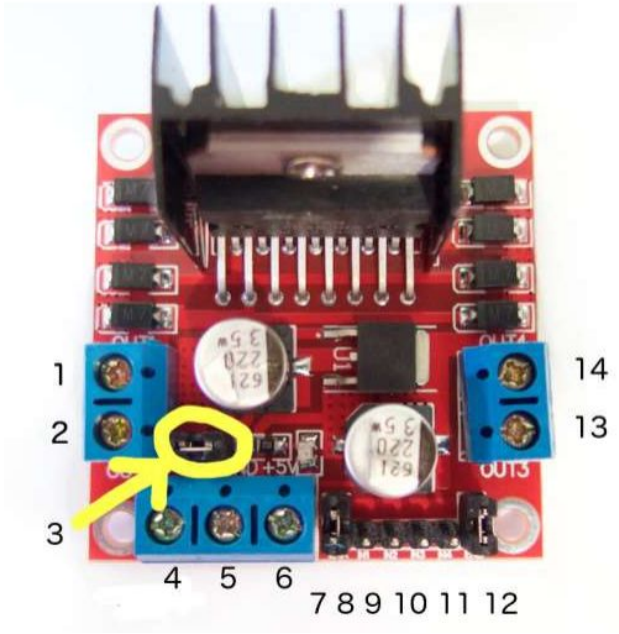
H-Bridge

When the polarity changes, so does the direction of rotation of the motor.



L298N Module Pinout

1. DC motor 1 "+" or stepper motor A+
2. DC motor 1 "-" or stepper motor A-
3. 12V jumper - remove this if using a supply voltage greater than 12V DC. This enables power to the onboard 5V regulator
4. Connect your motor supply voltage here, maximum of 35V DC. Remove 12V jumper if >12V DC
5. GND
6. 5V output if 12V jumper in place, ideal for powering your Arduino (etc)
7. DC motor 1 enable jumper. Leave this in place when using a stepper motor. Connect to PWM output for DC motor speed control.
8. IN1
9. IN2
10. IN3
11. IN4
12. DC motor 2 enable jumper. Leave this in place when using a stepper motor. Connect to PWM output for DC motor speed control.



Controlling two DC-motors with L298N Module

1. Connect each motor to the A and B connections on the L298N module (1/2 and 13/14), making sure the motors have the same polarity.

2. Connect the power supply - the positive to pin 4 and negative/GND to pin 5. If the supply is up to 12V you can leave in the 12V jumper (point 3 in the image) and 5V will be available from pin 6 on the module. This can be fed to the Arduino 5V pin to power it from the motors' power supply. **Don't forget to connect Arduino GND to pin 5 on the module as well to complete the circuit.**

Controlling two DC-motors with L298N Module

3. Connect 4 digital output pins from Arduino to IN1, IN2, IN3, IN4.
    a. IN1: Motor A +
    b. IN2: Motor A -
    c. IN3: Motor B +
    d. IN4: Motor B -

4. Connect 2 analog output pins from Arduino to EnA, EnB.
    a. EnA: Speed Control for motor A
    b. EnB: Speed Control for motor B

L298N Module Direction/Speed Control

The motor direction is controlled by sending a HIGH or LOW signal to the drive for each motor.

For example, for motor A:

- a HIGH to IN1 and a LOW to IN2 will cause it to turn in one direction,
- a LOW to IN1 and a HIGH to IN2 will cause it to turn in the other direction.

However the motors will not turn until a HIGH is set to the enable pin, which can be done by placing the plastic jumper on the EN pin but it will prevent speed control (maximum speed only).

However if we need to control the speed of the motors, the plastic jumper should be removed, and the enable pins should be connected to 2 analog pins.