The Magical Cube: an Introduction to Arduino

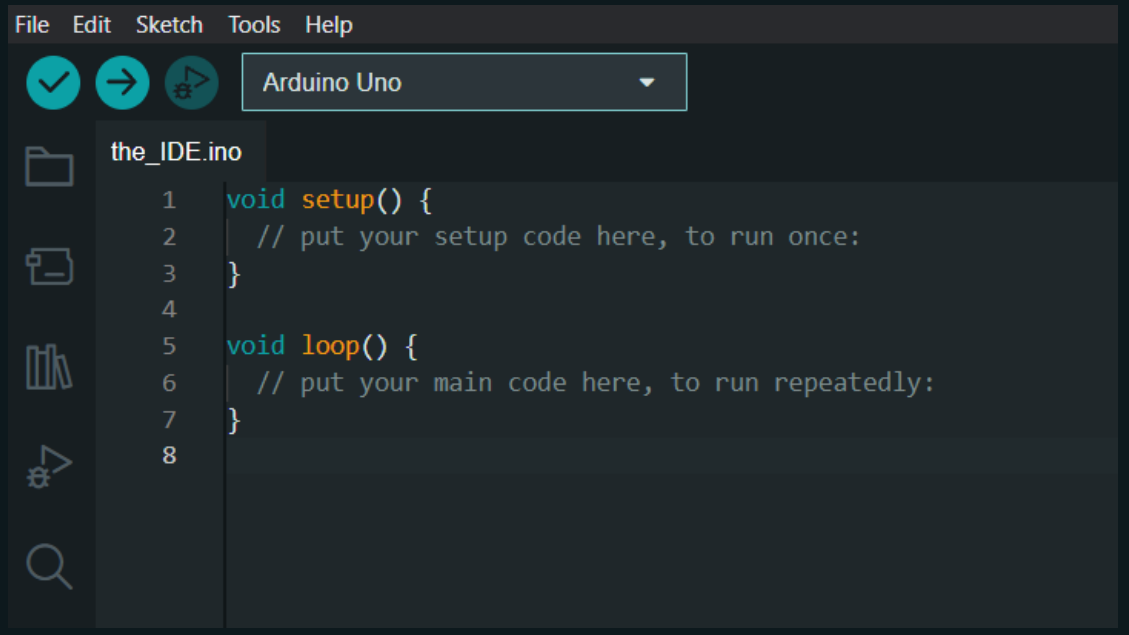
Arun Chauhan, Ishaan Takrani

The Arduino IDE

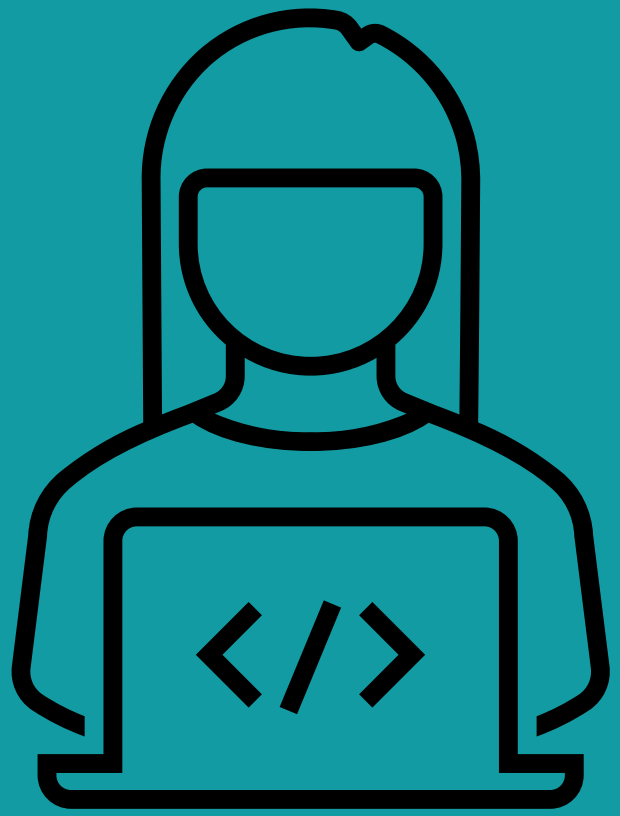
Introduction to the Arduino IDE

What is an IDE?

- IDE stands for ***integrated development environment.***
- It is an application that allows for programmers (like you) to develop code.
- Generally, an IDE allows for the writing, compiling, and executing of code.
- The Arduino IDE allows the user to write, compile, and upload code to an Arduino microcontroller.



What Does it Do?



Writing

- The IDE allows the user to write code.
- Some IDEs allow for multiple languages.
- Many can autocomplete lines of code (intelligent code completion).
- The Arduino IDE uses a language that is a variant of C++.

Compiling

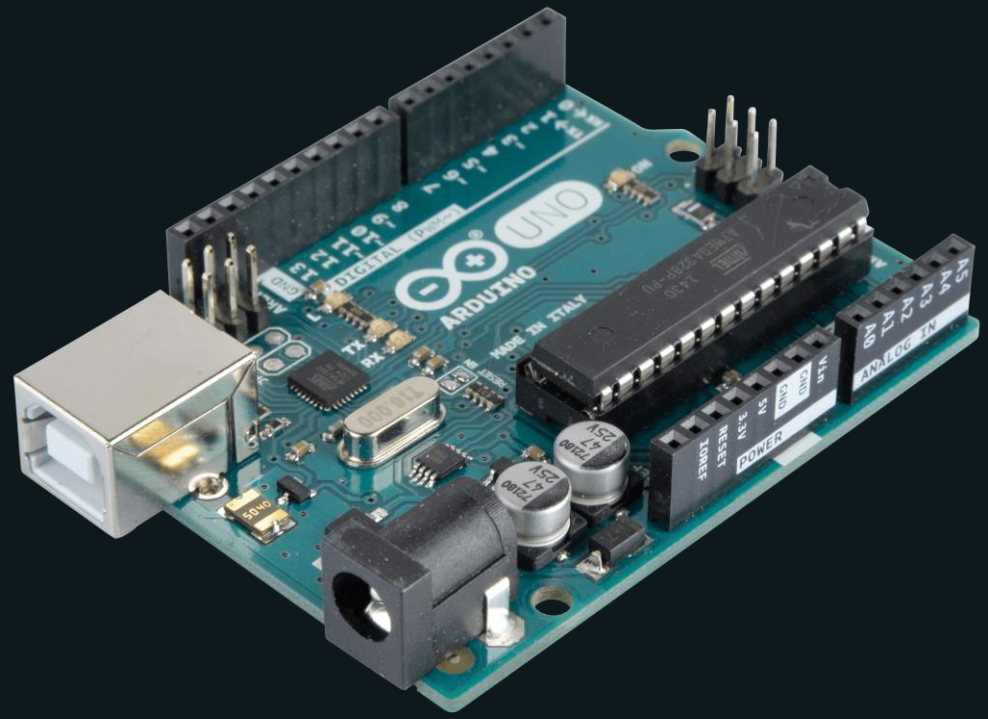
Compiling is the process of converting the code into an executable file that the computer can understand.

Computers can't read English (they don't understand it).

The code must be transformed into machine language for the computer to execute.

Uploading

- Once compiled, the device must execute the program.
- The compiler sends the executable program to the Arduino Microcontroller.
- The file is uploaded through a wired connection from the device with the IDE installed to the Arduino microcontroller.
- The microcontroller executes the code.



Arduino Language Basics

Introduction to the Arduino programming language

What is a Programming Language?

Simply put, a programming language is a notation used for the programming of a computer.

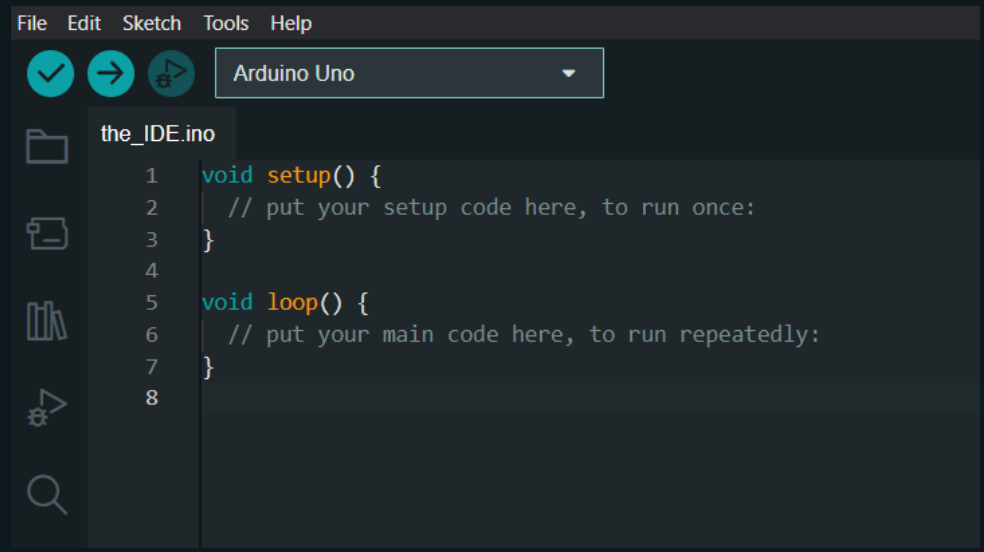
They are used to make software for a vast number of devices.

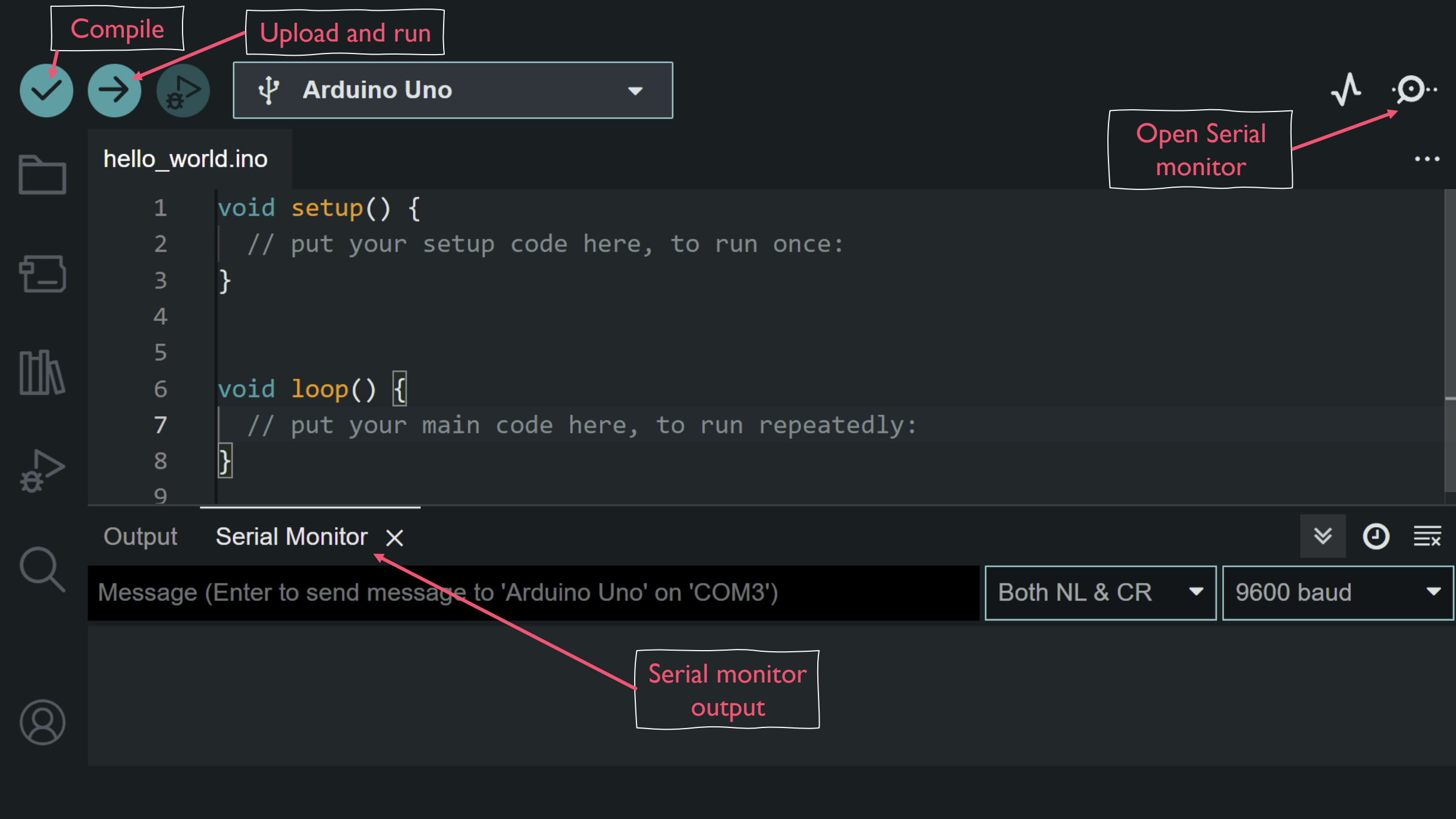
Arduino Programming Language

- The Arduino programming language is similar to C++, with modified functionalities that make it practical for Arduino microcontrollers.

setup() and loop()

New files in the Arduino IDE have two functions pre-defined. Void setup and void loop. When the program is run, whatever code is in void loop will run ***once***, then whatever is in void setup will run ***repeatedly***.



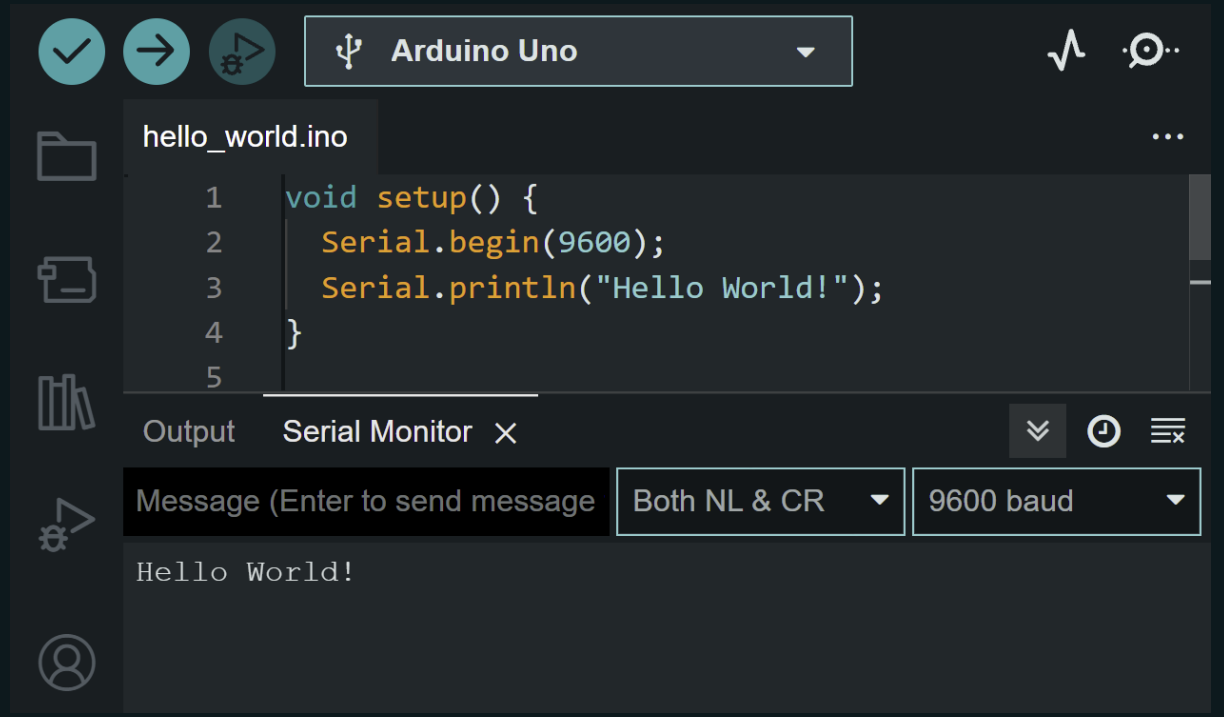

Compile
Upload and run
Arduino Uno
Open Serial monitor
hello_world.ino
1 void setup() {
2 // put your setup code here, to run once:
3 }
4
5
6 void loop() {
7 // put your main code here, to run repeatedly:
8 }
9
Output
Serial Monitor
Message (Enter to send message to 'Arduino Uno' on 'COM3')
Both NL & CR
9600 baud
Serial monitor output

Syntax

- The Arduino programming language is similar to C++, so both have the same rules and conventions.
- https://www.w3schools.com/cpp/ is a helpful resource to learn more about the C++ syntax

Hello World!

- Serial.begin(9600); sets the baud rate, which is the number of signal changes per second.
- Serial.println("hello world"); outputs hello world to the serial monitor.
- begin() and println() are functions of the Serial class. They take arguments in the parentheses and do something with them.



ARDUINO / BREADBOARD

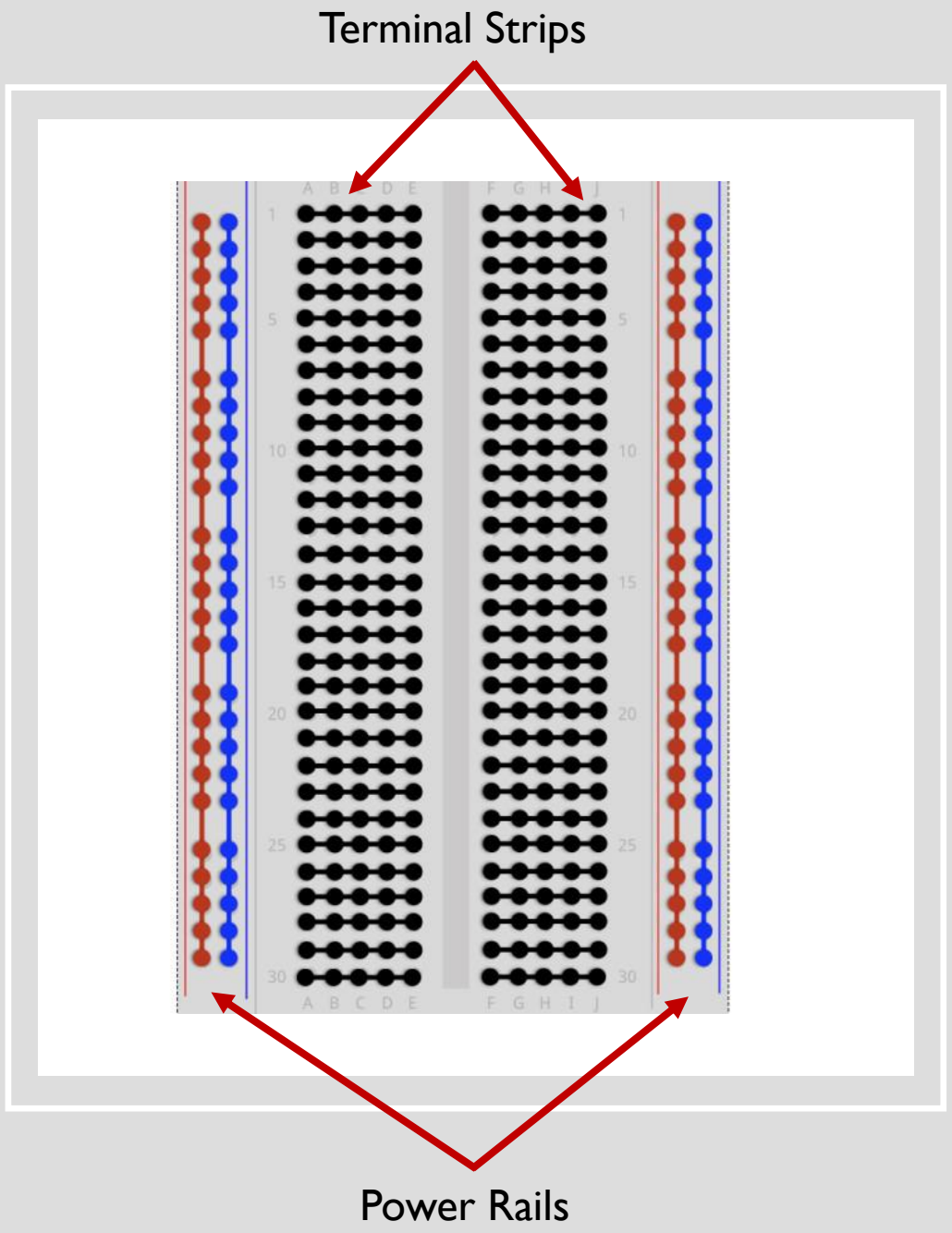
And how they work together to do what you want them to

BREADBOARD

The breadboard is a board that circuits can be built on. It has two power rails, two terminal strips, and a center divider.

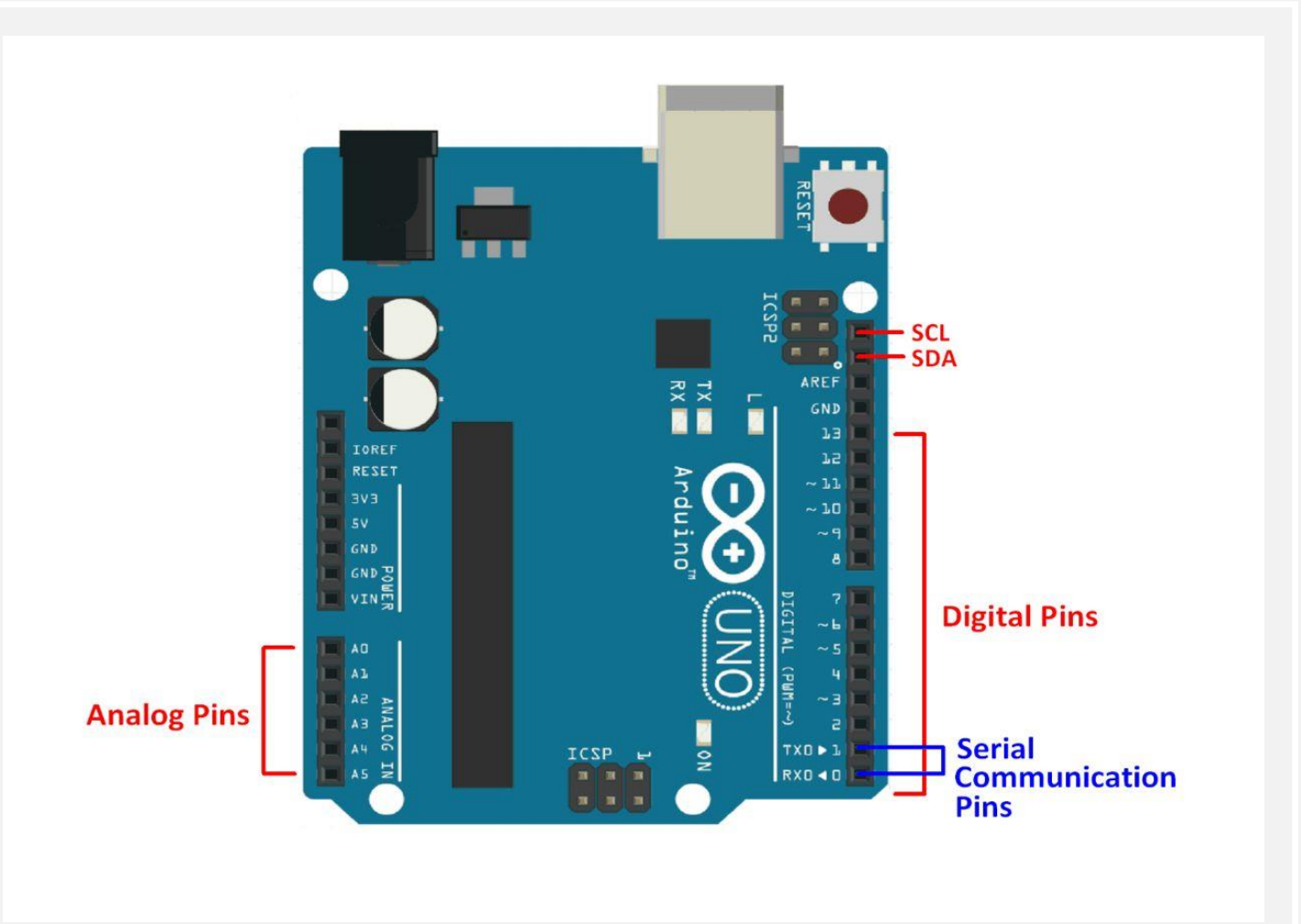
Electrons move through the rails vertically, and through the terminal strips horizontally, as shown by the connections in the image.

Nothing flows across the center divider unless the two strips are connected.



ARDUINO UNO MICROCONTROLLER

- The Arduino Uno is a microcontroller that can be used to build and control circuits.
- It features pins for power, ground, and analog and digital data pins.
- Generally, components are connected to the digital pins or analog pins to send and receive data from the Arduino.



SAFETY

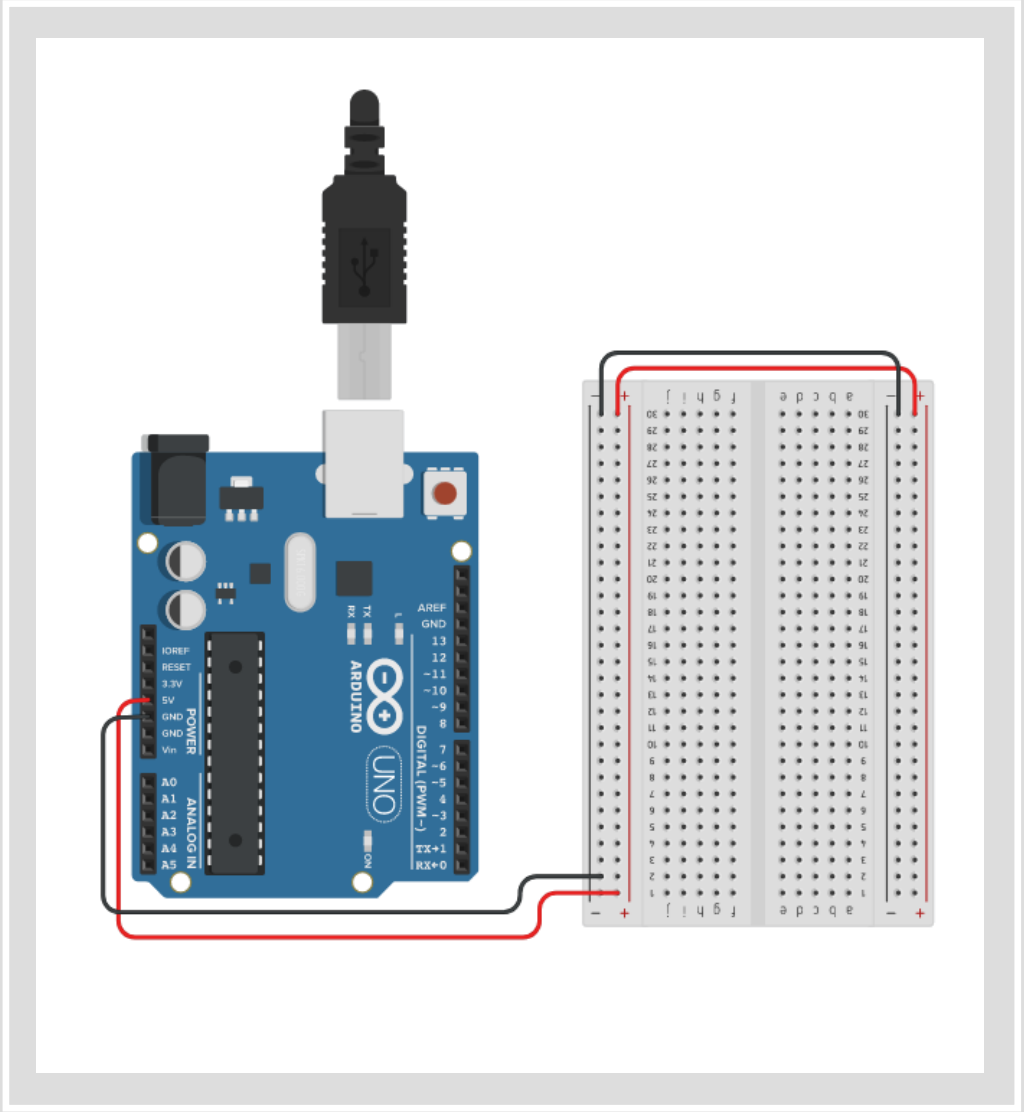
Important: When working, never touch or connect/disconnect wires or tweak parts while the board is connected to power. This could result in serious injury. DO NOT touch circuit components while power is connected.

NEVER – work on a LIVE circuit.

The person depicted is a trained professional. Do not attempt to replicate their actions.

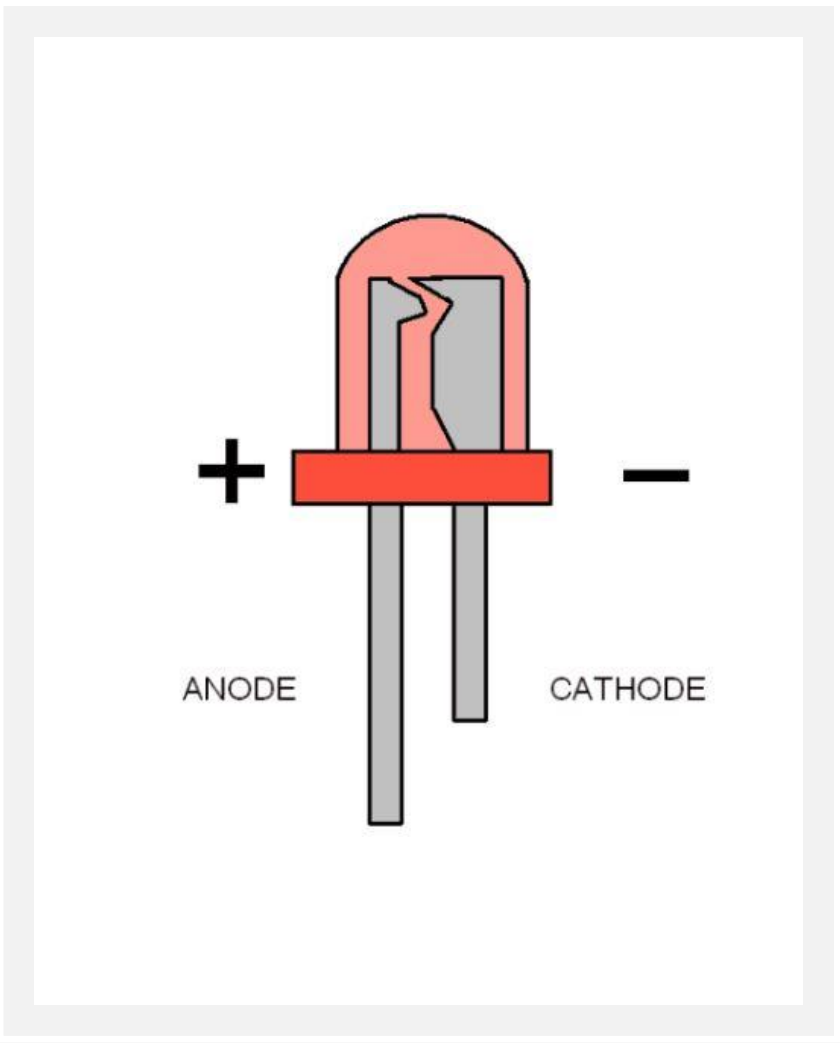
PROVIDE POWER TO THE BREADBOARD

Connecting the 5V output pin to the red power rail and the GND (ground) pin to the black rail and connecting both rails over the center divider allows the rails to be used to power components.



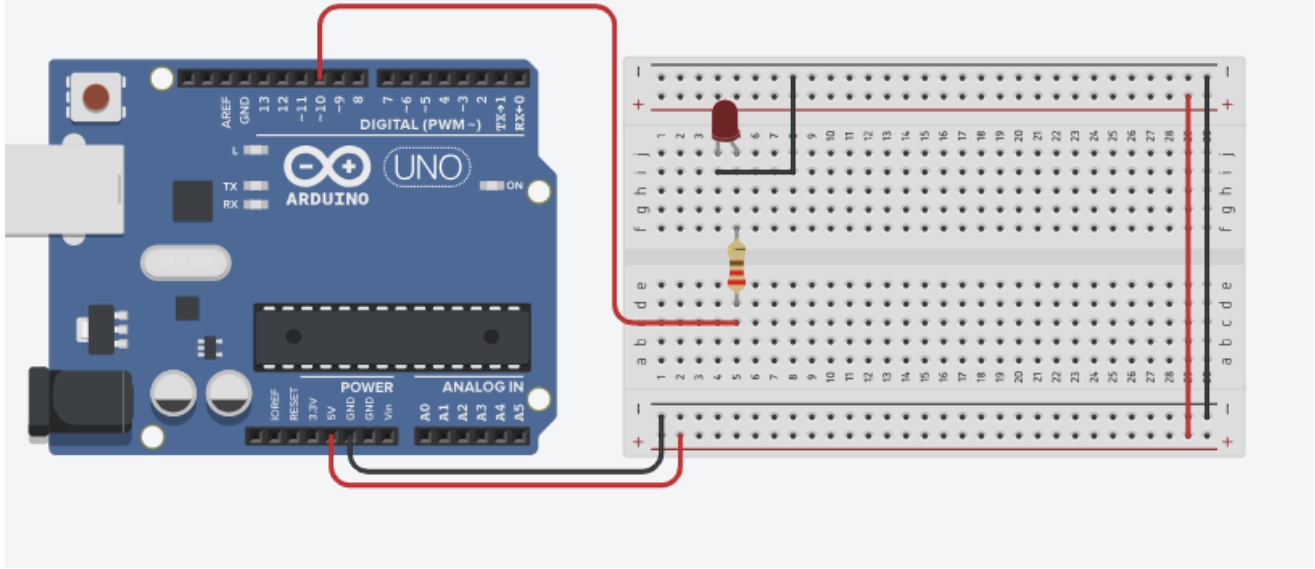
WHAT IS AN LED?

- **LED** stands for **L**ight **E**mitting **D**iode.
- It emits light when current flows through it.
- The colour of the LED is determined by the wavelength of the emitted light.
- Inside an LED, there is an anode and a cathode. Current enters through the anode and leaves through the cathode.
- The anode is connected to the power source, which is either the power rail on the breadboard, or an Arduino digital pin.
- The anode usually has a longer stalk than the cathode.



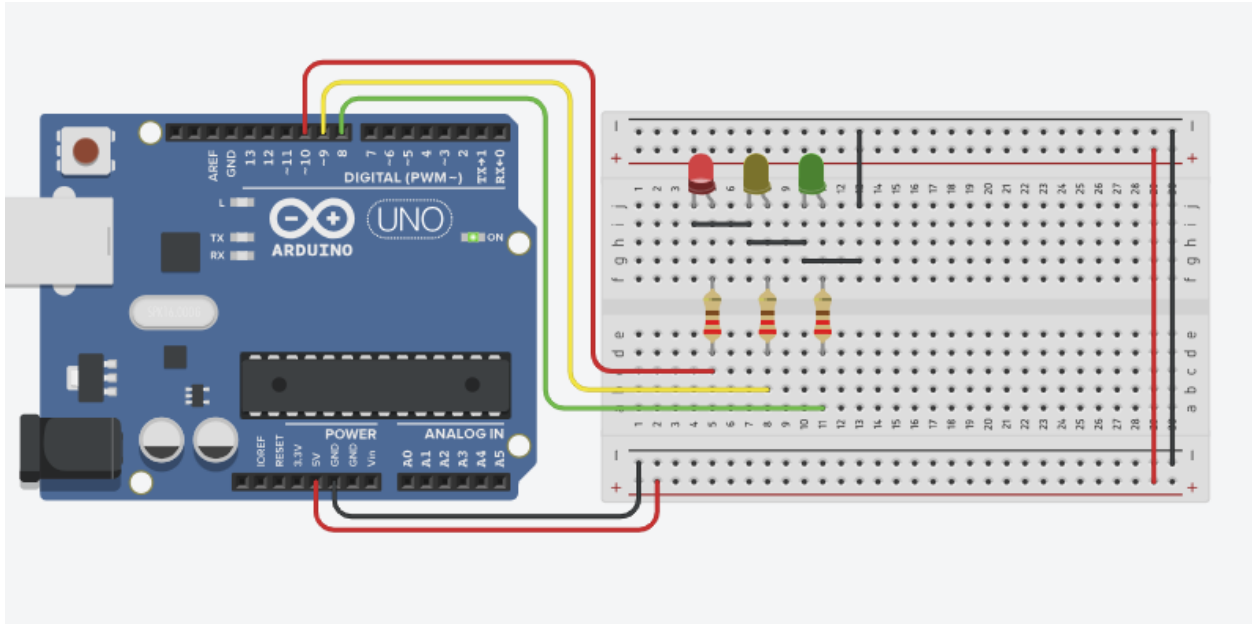
LED → BREADBOARD

- The anode of the LED is connected to a digital pin on the Arduino, and the cathode is connected to the ground rail on the breadboard.
- There is also a 220Ω resistor connected in series with the LED. This resists the current provided by the Arduino and prevents the LED from burning out or exploding.



LEDS

- The three other pins each control a light. Connect each one to an output pin on the Arduino. Since pins 0 and 1 are used for serial communication, use digital pins 2-13
- Example:
  - Green: Pin 8
  - Yellow: Pin 9
  - Red: Pin 10
- In this diagram, the red led (pin 10) is activated.



PINMODE() – ARDUINO IDE

The pinMode function is used to determine Arduino pins as either input or output pins.

Since the Arduino must send a signal to the LEDs, the pins that do this are output pins.

This is done in void setup().

```
void setup() {

  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10,OUTPUT);

}
```

DIGITALWRITE()

- Void loop is where code runs repeatedly. To turn on the lights on the LEDs, use digitalWrite.

digitalWrite writes a digital pin to the request of the programmer. By setting it to HIGH, whatever is connected to the pins will activate. LOW does the opposite.

```
void loop() {
  digitalWrite(8, HIGH);
  digitalWrite(9, HIGH);
  digitalWrite(10, HIGH);
}
```

DELAY

Delay is used to add a delay. The parameter taken is the time, in milliseconds. (1000ms = 1s)

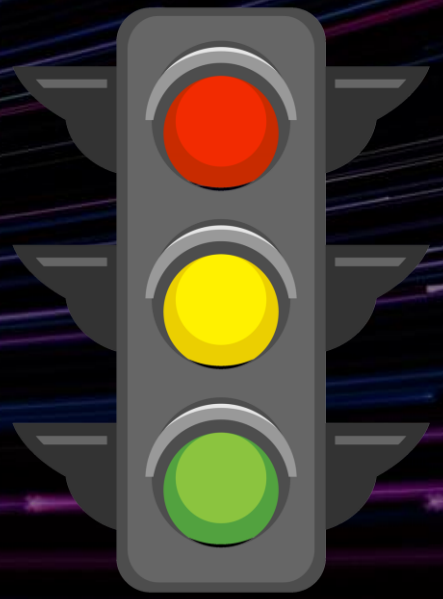
This code turns the green light (pin 8) on for one second, then off for one second. Because this code is in void loop(), this is repeated indefinitely.

```
void loop() {
  digitalWrite(8, HIGH);
  delay(1000);
  digitalWrite(8, LOW);
  delay(1000);
}
```

TASK: TRAFFIC LIGHT

- Simulate a traffic light system using the green, red, and yellow lights using digitalWrite() and delay().

- First, the green light should turn ON for 7 seconds. Then, the yellow light should turn ON for 2 seconds. Lastly, the red light should turn ON for 3 seconds. This sequence should repeat continuously.



VARIABLES

A variable is a storage location that can be assigned a value.

In the Arduino programming language, variables are defined with datatypes. Datatypes can include integers, strings, floats, and more. More at: https://www.javatpoint.com/arduino-data-types

DEFINING A VARIABLE IN THE ARDUINO IDE

- Variables in the Arduino programming language are usually defined outside of the setup and loop functions.
- Notice how each LED corresponds to a number, and to activate the LED, the number must be used.
- Variables can be used to replace the number, to make code easier to write and understand.
- Let's make some variables for the LEDs.

DEFINING A VARIABLE IN THE ARDUINO IDE

- Firstly, we need a datatype for the variable. In this case, Integer would work. Since "int" is the representation for integer in C++, we will write "int" before the variable name. const int (constant integer) can also be used.
- A const int is an integer that can't be modified once it is defined. Because the value of a pin will not be modified, const int is a viable option.
- Next, we need to assign the variable a value by using "=".
- Variable names should be clearly understood and are usually concise.
- Now, instead of needing to remember the number that corresponds to each LED, I can just write the variable.

```
int green_led = 3;
int yellow_led = 4;
const int red_led = 5;
```

TASK 2: DOUBLE TRAFFIC LIGHT SYSTEM

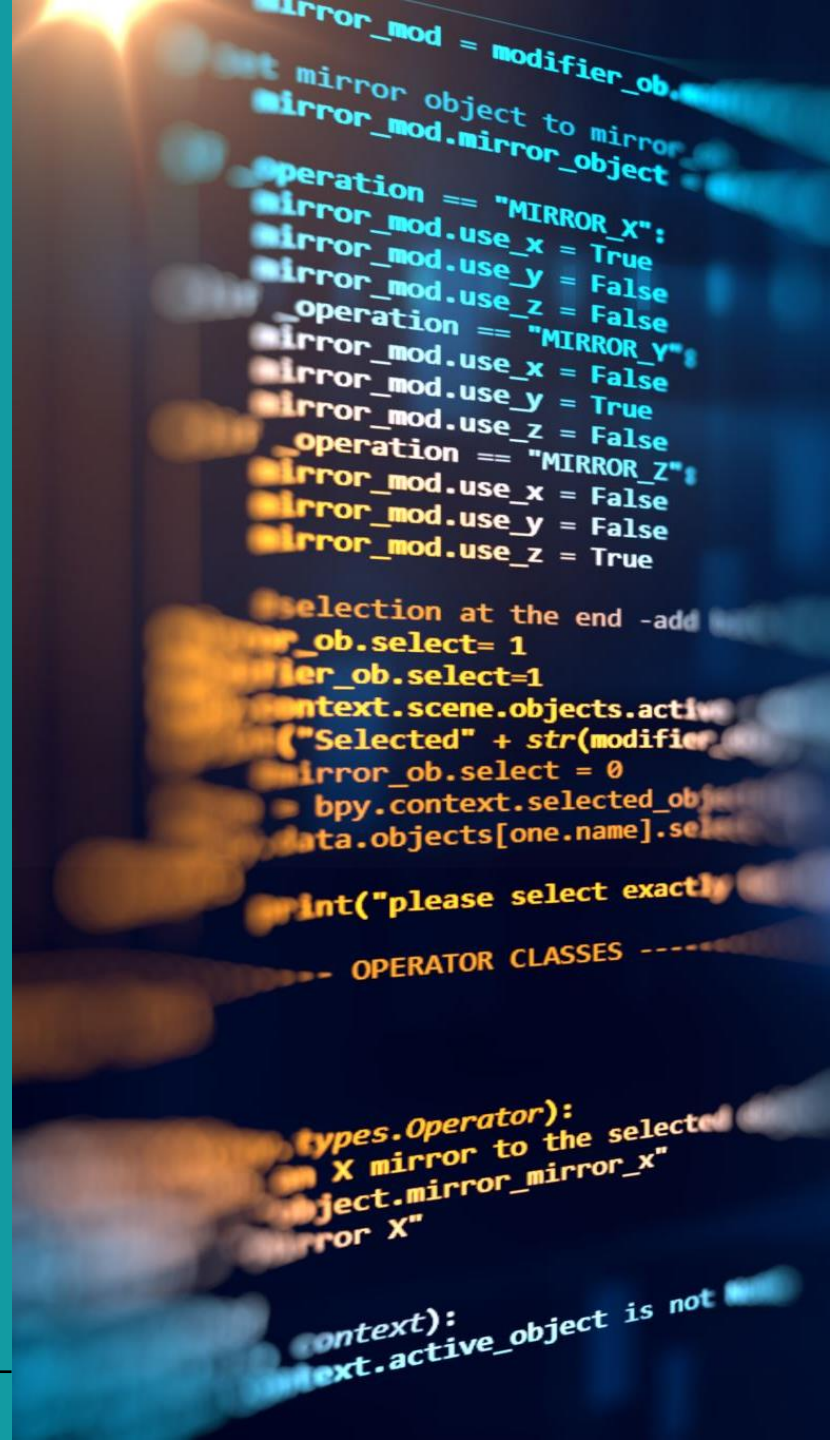
- Using variables for the LEDs, Connect another three LEDs to the Arduino and simulate a two-way traffic light system, much like one in real life.
- The red and green lights should stay on for 5 second intervals, while the yellow light should stay on for 2 seconds.

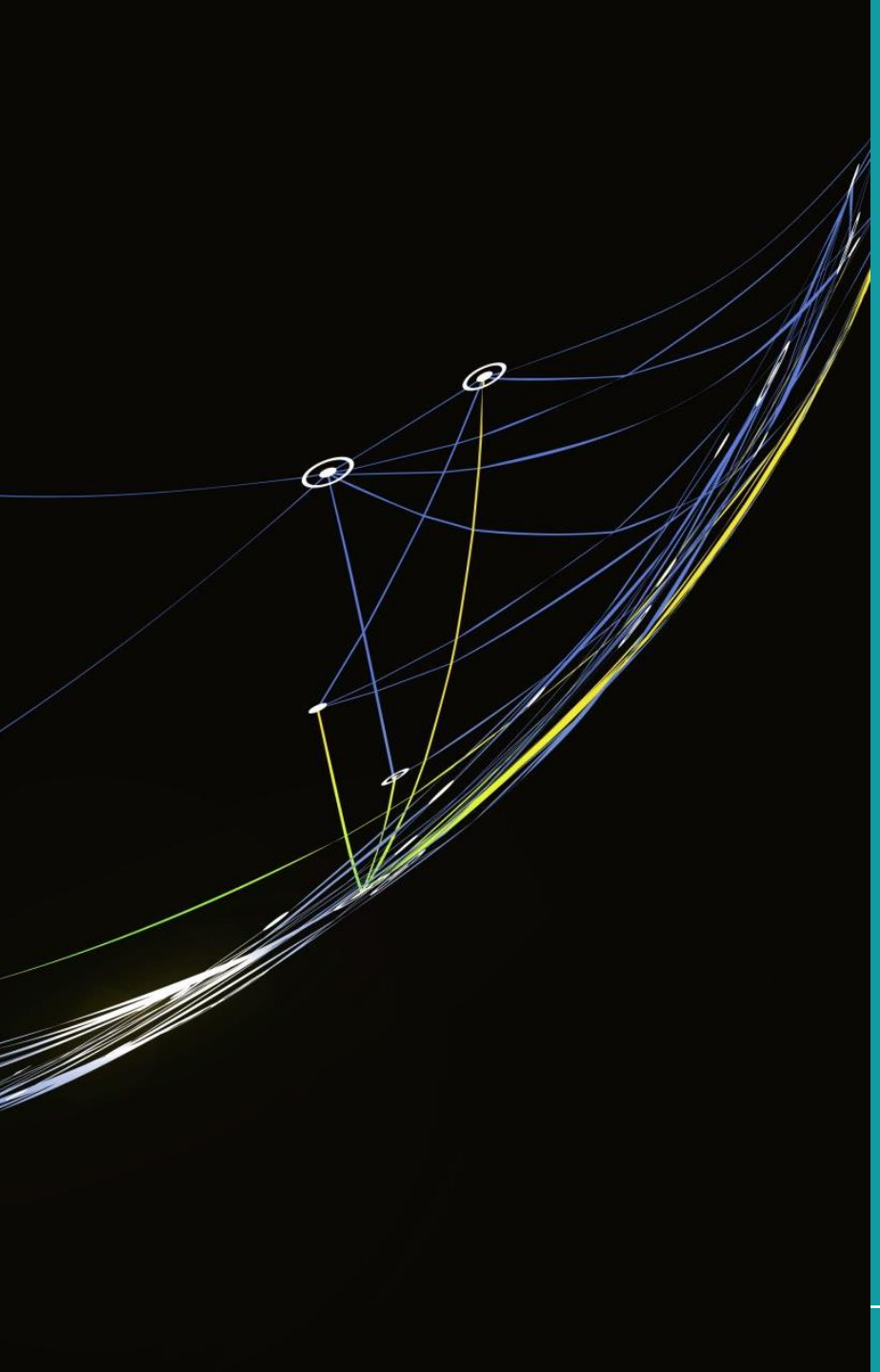
Programming: Functions

In programming, a function is a module that performs a specific task. They are generally used for processes in code that need to be performed more than once.

Functions generally return, or produce, a value that can be used in the program.

Some take values when they are called, which are called arguments, which are referred to parameters inside of the function.





Functions: Example

Each function has a return type, which is the datatype of the variable that is returned. In the case of void loop and void setup, the return type is "void," which means that nothing is returned.

Return types can be int (integer), string, char, double, and any other datatype.

When arguments are passed to the function, a datatype also needs to be defined.

Functions: Example

- Consider this: I want to make an LED flash rapidly.
- I could write code for each LED, but to make it simpler, I can use a function.
- The function takes the LED as an argument, and then turns it on and off rapidly. This can be used for every led.

```
void rapidFlash(uint8_t led){

  digitalWrite(led, HIGH);
  delay(30);
  digitalWrite(led, LOW);
  delay(30);
  digitalWrite(led, HIGH);
  delay(30);
  digitalWrite(led, LOW);
  delay(30);
  digitalWrite(led, LOW);
  delay(30);
  digitalWrite(led, HIGH);
  delay(30);
  digitalWrite(led, LOW);
  delay(30);
}
```

```
const int green = 3;
const int yellow = 4;
const int red = 5;
```

Define variables

```
void rapidFlash(uint8_t led){
  digitalWrite(led, HIGH);
  delay(30);
  digitalWrite(led, LOW);
  delay(30);
  digitalWrite(led, HIGH);
  delay(30);
  digitalWrite(led, LOW);
  delay(30);
  digitalWrite(led, LOW);
  delay(30);
  digitalWrite(led, HIGH);
  delay(30);
  digitalWrite(led, LOW);
  delay(30);
}
```

Define function with return type void and **argument** with datatype uint8_t (8-bit unsigned integer)

Flashes LED rapidly

In the function wherever "led" is used, it is the same as the ***argument*** given in the ***function call***.

```
void setup() {
  pinMode(green, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(red, OUTPUT);
}
```

pinMode for each LED

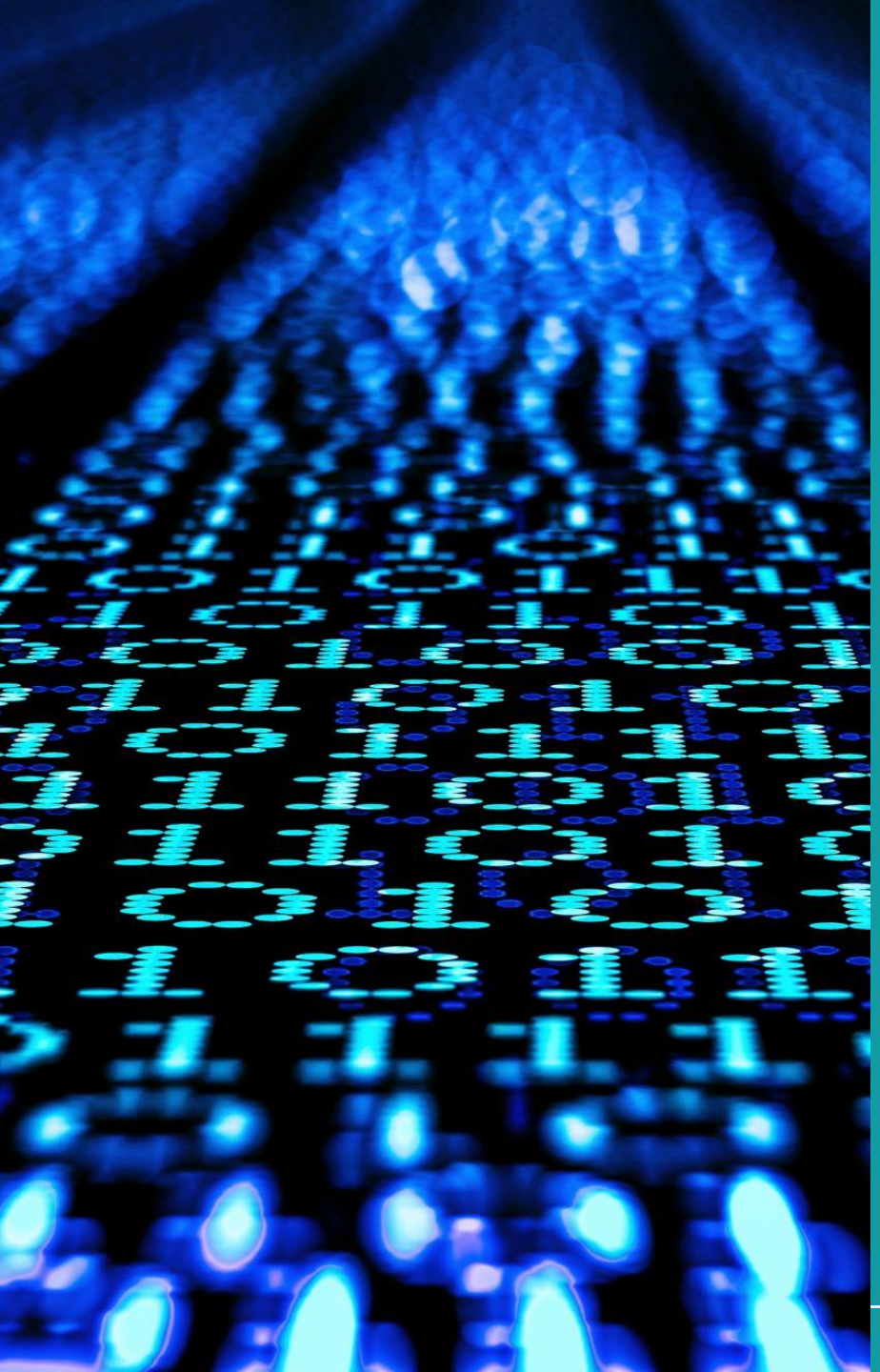
```
void loop() {
  rapidFlash(green);
  rapidFlash(yellow);
  rapidFlash(red);
}
```

Calls the rapidFlash function for each LED (the function calls)

Task 3: Function(ing) Traffic Lights

- Re-create the double traffic light system from before, but this time, use a function to do it.
- The only code in void loop should be the calling of the function, everything else (digitalWrite statements) should be in the new function.
- Hint: the function definition should look *something* like this:

```
void doubleTrafficLightSystem(uint8_t green, uint8_t yellow, uint8_t red, uint8_t green2, uint8_t yellow2, uint8_t red2)
```



Programming: For Loops

A for loop is a chunk of code that runs until a numerical condition is met.

They are generally used to perform a task, usually slightly altered each time, a certain number of times. Let's break down a for loop in void setup:

Also, more at:
https://www.arduino.cc/reference/en/language/structure/control-structure/for/

Programming: For Loops

Initializes a new variable, I, as 0. This variable can have any name, but I is generally used.

Whatever is in the for loop runs until the condition is false. The condition in this case is: i<5, so if I is greater than 5, the loop ends.

Each semicolon is a ***delimiter***. It essentially tells the compiler that the line ends here. After the delimiter, a new line starts.

i++ increments i by one each time the loop runs.

```
for(int i = 0 ; i<5 ; i++){
    Serial.println(i);
}
```

Since i is 0 and gets incremented by one each time the loop runs, and the loop runs until i is equal to 5, this code prints numbers from 0 to 4 to the serial monitor.

For loops: Example

What if we wanted to print every number from 0 to 200?

```
for(int i=0;i<200;i++){
    Serial.println(i);
}
```

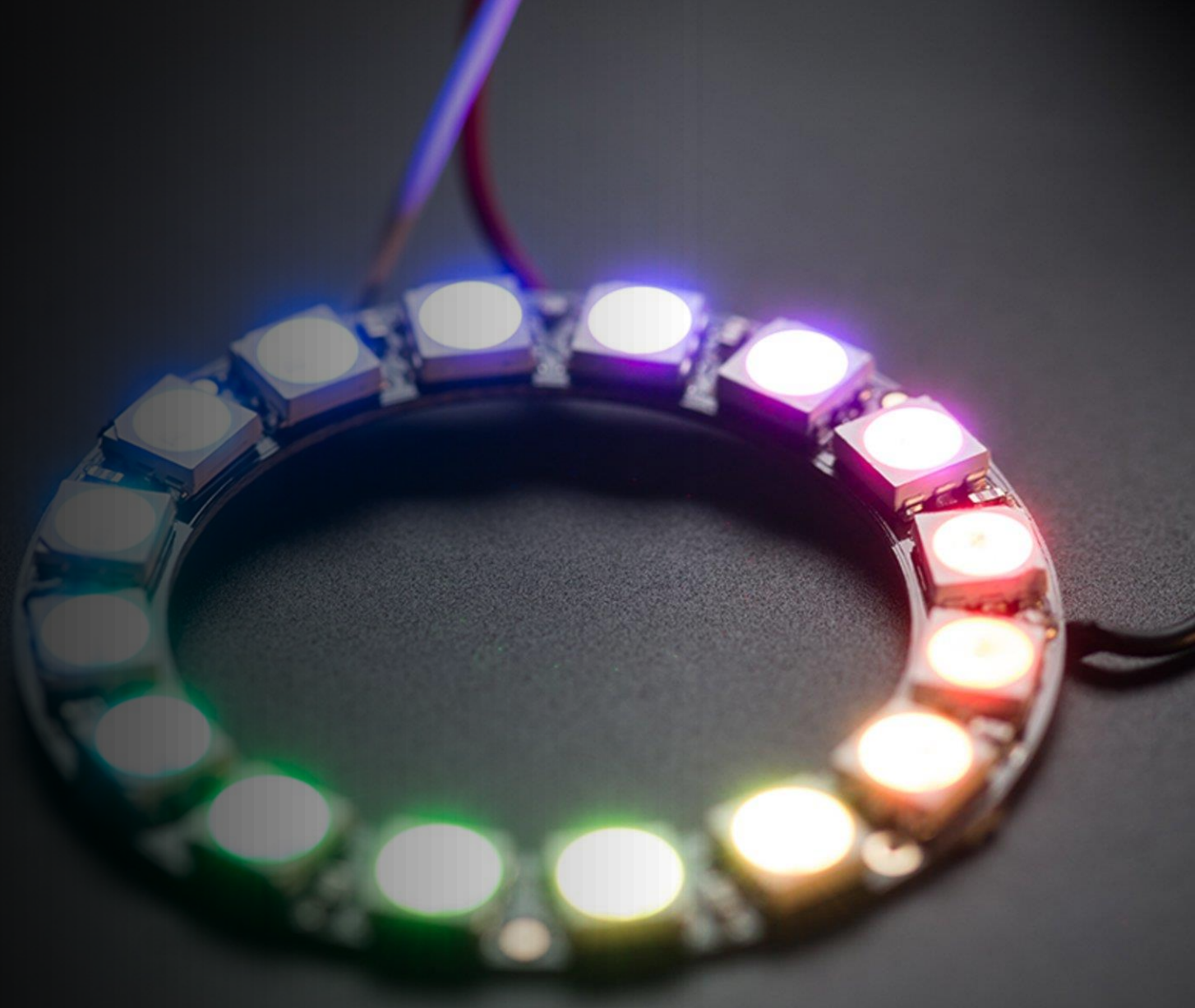
What if we wanted to print every number from 10 to 15?

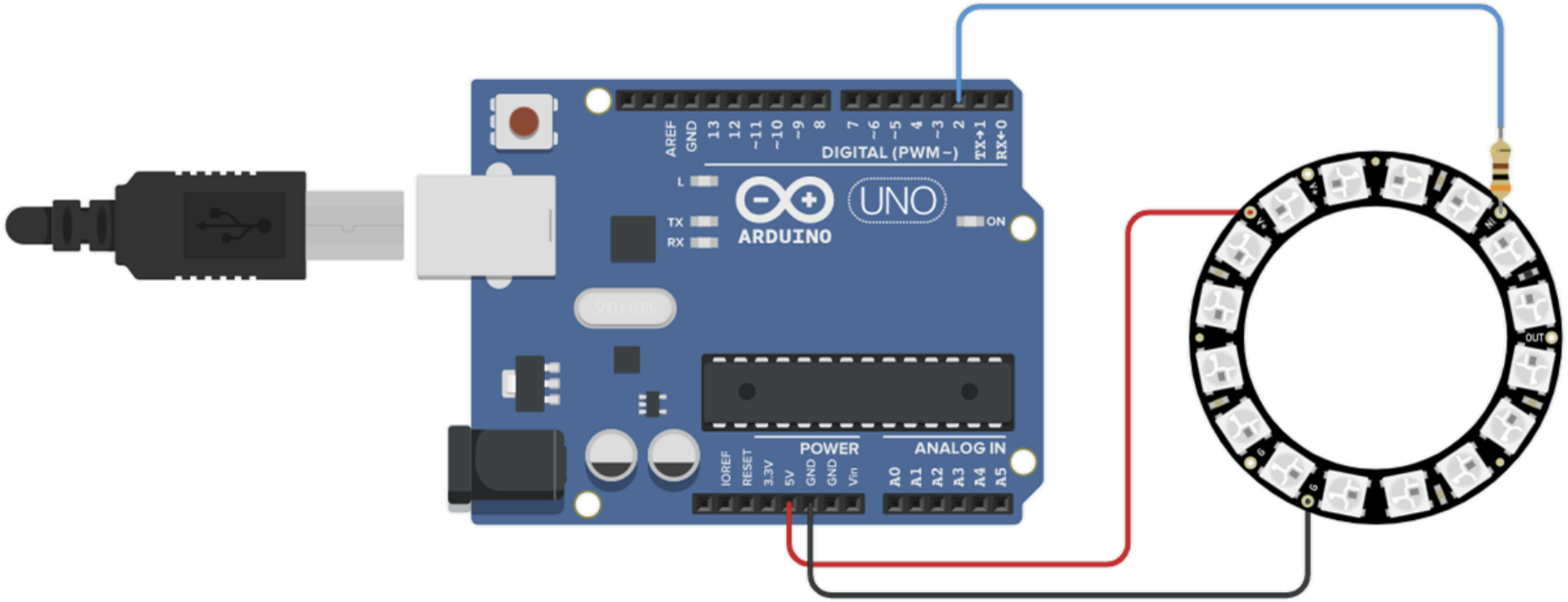
```
for(int i=10;i<16;i++){
    Serial.println(i);
}
```

What if we wanted to print every number from 20 to 5?

```
for(int i=20;i>6;i--){
    Serial.println(i);
}
```

NeoPixel Ring





Wiring

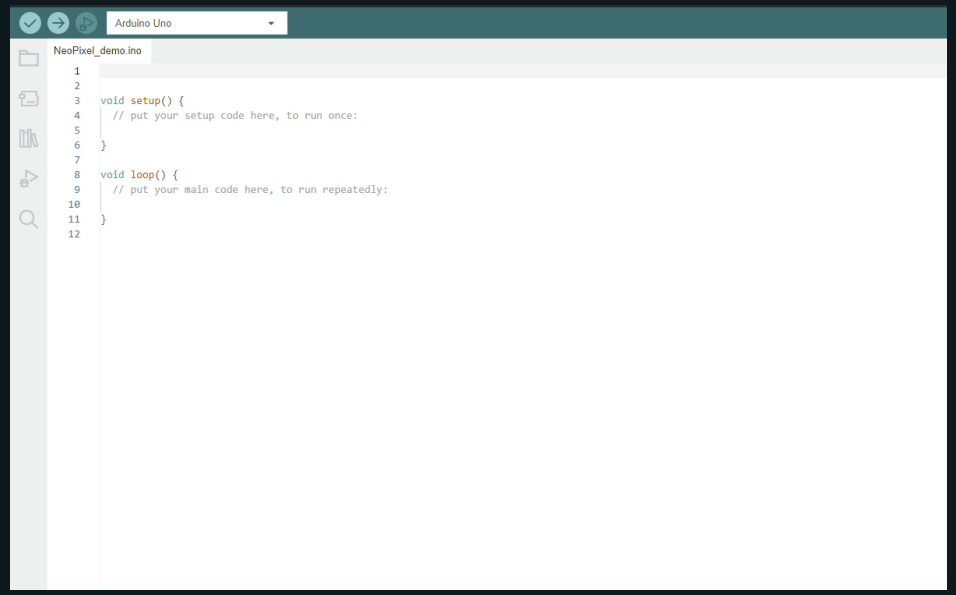
- On the ring, there are three connections to be made.
- V+, G, and IN (soldered).
- V+ connects to the 5V pin on the Arduino, G connects to the GND pin, and IN is an input pin that connects to a digital pin on the Arduino.
- 30Ω resistor in series with the ground connection (some NeoPixel rings already have the resistance required, and don’t need an additional one).

NeoPixel Libraries

- A library is a file/set of files that contain pre-built functions that allow the programmer to control a wide variety of components more efficiently.
- The Adafruit Neopixel library contains many functions that make the usage of the NeoPixel ring easier.
- This library can be installed through the Arduino IDE, and by using the #include keyword, any functions in the included file can be used in your code.

Installation

Video:



- The library can also be downloaded from: https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-installation
- Once downloaded, the library can be included in the file by using the #include keyword in your code, like so:
- #include <Adafruit_NeoPixel.h>

Programming the NeoPixel ring

- The Adafruit NeoPixel library makes programming the NeoPixel ring simpler, as it contains a class and functions specific to the ring.
- To use the ring, a new object of the Adafruit_NeoPixel class must be created using a constructor.

Object-Oriented Programming - Basics

Object oriented programming is a type of programing concept based on objects and classes. A class is a general category, usually with certain functions and characteristics, and an object is something of a certain class.

For example, a class can be “ball,” and it can have the functions of bouncing, and characteristics such as size.

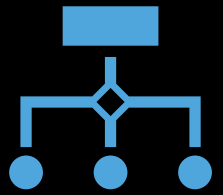
A member of this class (the object) can be a ***basketball***, which can perform the “ball” class functions, and has “ball” characteristics, such as size.

Another member of the “ball” class can be a ***baseball.***

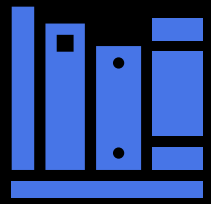
Think of the object as an actual object, and the class as a classification of the object.

Programming the NeoPixel Ring

Object oriented programming (OOP) is used for the NeoPixel ring.



The class is “Adafruit_NeoPixel” and the object is your physical NeoPixel ring. The class is the classification of the object.



The Adafruit NeoPixel library contains the definition of the Adafruit_NeoPixel class, and its functions.

Programming the NeoPixel Ring - Constructor

A constructor in OOP is a function that is used to initialize a new object. It can also take arguments that define the object's characteristics.

The constructor for the NeoPixel ring takes three arguments:

The number of pixels	The Arduino pin it's connected to	LED type

The Constructor - Breakdown

Arg 3: LED type (NEO_GRB) and data stream frequency (NEO_KHZ800)

Note: GRB means green, red, blue, and KHZ800 means 800khz

Class name

Object name

Call constructor function with three arguments (args)

Arg 1: # of LEDs

Arg 2: Arduino output pin

```
Adafruit_NeoPixel my_ring = Adafruit_NeoPixel (16, 13, NEO_GRB + NEO_KHZ800);
```

NeoPixel Functions

Some of the primary functions for the NeoPixel ring are:

- begin(): initialize the object.
- clear(): set all pixels to 0 / off.
- setPixelColor(pixel number, R, G, B): set a pixel's colour to the specified RGB value.
- show(): send the pixel data to the NeoPixel ring.

Example:

- Here is some example code for the ring. It turns all 16 pixels on the ring red at full power (255).
- The LEDs are labeled 0-15, since computers start counting from zero.
- This can also be done in a much easier way by using a for loop.

```
#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel my_ring = Adafruit_NeoPixel(16, 13,
NEO_GRB + NEO_KHZ800);

void setup() {
  my_ring.begin(); // initializes the ring
  my_ring.clear(); // clear ring (blank)
  my_ring.show(); // send instructions to ring
}

void loop() {
  my_ring.setPixelColor(0,255,0,0);
  my_ring.setPixelColor(1,255,0,0);
  my_ring.setPixelColor(2,255,0,0);
  my_ring.setPixelColor(3,255,0,0);
  my_ring.setPixelColor(4,255,0,0);
  my_ring.setPixelColor(5,255,0,0);
  my_ring.setPixelColor(6,255,0,0);
  my_ring.setPixelColor(7,255,0,0);
  my_ring.setPixelColor(8,255,0,0);
  my_ring.setPixelColor(9,255,0,0);
  my_ring.setPixelColor(10,255,0,0);
  my_ring.setPixelColor(11,255,0,0);
  my_ring.setPixelColor(12,255,0,0);
  my_ring.setPixelColor(13,255,0,0);
  my_ring.setPixelColor(14,255,0,0);
  my_ring.setPixelColor(15,255,0,0);
  my_ring.show();
}
```

Example: For Loop

- In void loop, this for loop can be used instead, and It is easily modifiable, which allows for easier usage of the NeoPixel ring.

```
for(int i = 0; i < 16; i++){
  my_ring.setPixelColor(i,255,0,0);
}
my_ring.show();
```

Task 4: Colours

- Using a for loop, set all pixels to red, yellow, green, blue, and purple, with a delay of 1 second for each colour.

Helpful colour combos (RGB):

- Yellow: (255, 255, 0)
- Green: (0, 255, 0)
- Blue: (0, 0, 255)
- Purple: (255, 0, 255)

Task 5: Spiral

- Make each pixel light up red one after the other, with a delay of 50 milliseconds in between each pixel lighting up. Then, turn each pixel off one by one, with a delay of 50 milliseconds in between each pixel turning off. Repeat this continuously to create a type of spiraling pattern.
- Tip: To turn off a pixel, the RGB values must be set to (0,0,0).

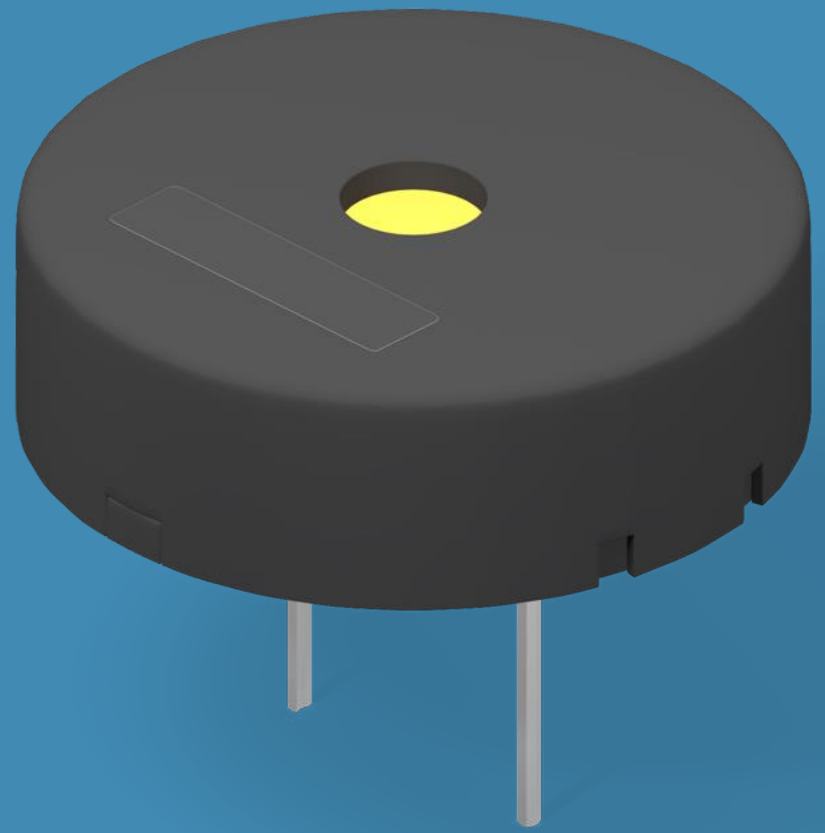
PIEZO BUZZER

beep beep

WHAT IS IT?

A piezo buzzer is a device that is used to create a sound. It contains a ceramic-like disc, called a piezo ceramic element, which is surrounded by a metal vibration disc. When current is applied to the buzzer, the disc vibrates, which creates a sound.

The higher the voltage, the faster the vibration, and this causes a higher-pitched sound.



ACTIVE AND PASSIVE BUZZER

There are two types of piezo buzzers. Active and passive.

Active buzzers only need a DC voltage source to produce sound, while passive buzzers need an AC voltage source.

To tell them apart use a DC voltage source like a battery. If the buzzer makes a continuous buzzing noise, it is an active buzzer.

ACTIVE BUZZERS

Passive buzzers can be activated by using digitalWrite(pin#, HIGH).

There are only two states, HIGH and LOW, which only allows for a single frequency tone.

They can be used for alarms notifiers, and anything that requires a sound that doesn't need to be more than one tone.

PASSIVE BUZZERS

Passive buzzers use the tone(pin#, frequency).

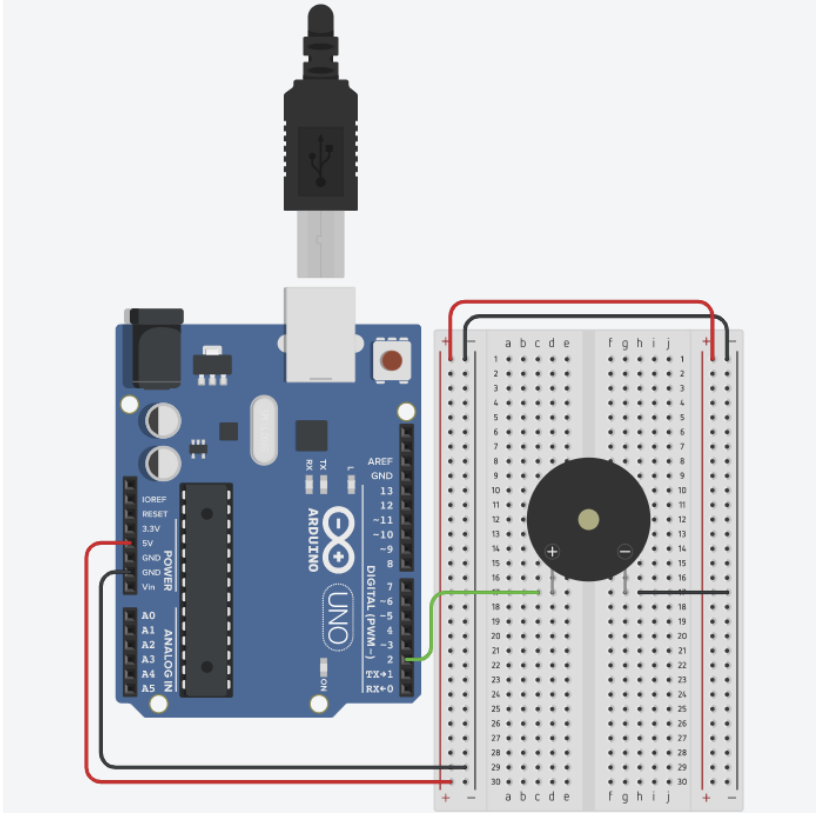
Unlike active buzzers, they can produce multiple frequencies.

This makes them useful for playing songs, or anything that requires multiple frequencies.

BUZZER → BREADBOARD → ARDUINO

Passive buzzers will be used for this section.

The buzzer has two connections. Power and ground. Ground connects to the ground rail on the breadboard, and power connects to a digital pin on the Arduino (pins 2-13).



PROGRAMMING THE BUZZER

- The buzzer is an output, so pinMode(buzzer, OUTPUT) must be used.
- Tone(buzzer, frequency) is used to control the buzzer's output frequency. (higher output number = higher frequency).
- Specific music notes can be played using the buzzer.

```
const int buzzer = 2; // buzzer is in pin 2

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

void loop() {

  tone(buzzer, 523); // Plays note C5
  delay(1000);

  tone(buzzer, 784); // Plays note G5
  delay(1000);

}
```

TASK 6: BEEP BOOP

- The noTone(buzzer_pin) function takes the buzzer's pin as a parameter. It turns off the buzzer. Use it with a delay to make a repeated beeping booping sound.
- The sound should be as follows: 100 milliseconds of a 1000hz tone, then 100 milliseconds of silence, then 100 milliseconds of a 500hz tone, then 100 more milliseconds of silence.
- This should create a "beep boop sound."

MUSIC!

The buzzer can play melodies!

By playing certain frequencies and using set delay lengths, the buzzer can be used to play simple songs.

This chart shows notes and their frequencies. The frequencies must me rounded to whole numbers in order to be used.

Note Frequency Chart

	Octave 0	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7	Octave 8
C	16.35	32.70	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01
C♯	17.32	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	4434.92
D	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.64
D♯	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	4978.03
E	20.60	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	5274.04
F	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
F♯	23.12	46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	5919.91
G	24.50	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	6271.93
G♯	25.96	51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	6644.88
A	27.50	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	7040.00
A♯	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	7902.13

playNote() FUNCTION

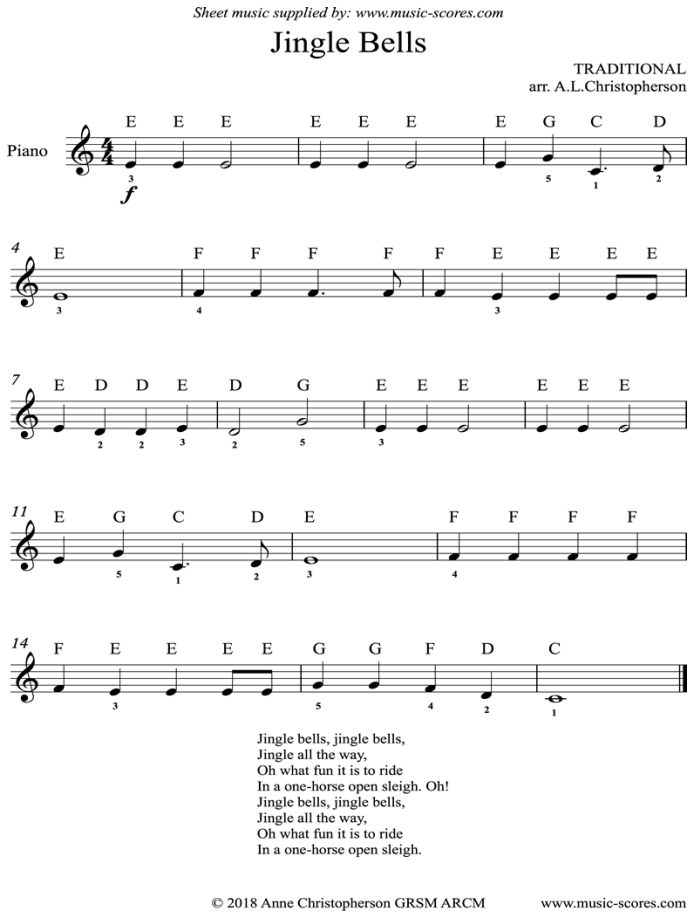
We can make a playNote() function to make playing notes easier.

The function adds a small silent period at the end of each note to differentiate between notes.

The function takes the note and the note duration as arguments.

```
void playNote(int note, int
delay_time){

  tone(buzzer, note);
  delay(delay_time);

  noTone(buzzer);
  delay(25);

}
```

EXAMPLE: JINGLE BELLS



Sheet music supplied by: www.music-scores.com

Jingle Bells

TRADITIONAL
arr. A.L.Christopherson

Piano

Jingle bells, jingle bells,
Jingle all the way,
Oh what fun it is to ride
In a one-horse open sleigh. Oh!
Jingle bells, jingle bells,
Jingle all the way,
Oh what fun it is to ride
In a one-horse open sleigh.

 www.music-scores.com

JINGLE BELLS

```
int buzzer = 2;
int E5 = 659;
int G5 = 784;
int C5 = 523;
int D5 = 587;
int F5 = 699;


void playNote(int note, int delay_time){

  tone(buzzer, note); // plays note in buzzer pin
  delay(delay_time);  // plays note for given time

  noTone(buzzer); // buzzer is silent
  delay(25);  // silent for 25 milliseconds
  // using noTone() allows for you to distinguish between notes

}
```

JINGLE BELLS

```
void setup() {
  pinMode(buzzer, OUTPUT);  // buzzer is output
}


/*
  at 120 bpm (beats per minute) each bar lasts
  for two seconds, which means each quarter note
  lasts for half a second, or 500 milliseconds.
  delay(500) is used for a quarter note.
*/
```

JINGLE BELLS

```
void loop() {

  // bar 1:
  playNote(E5, 500); // the first argument is the note, and the second is the delay
  playNote(E5, 500);
  playNote(E5, 1000);

  // bar 2:
  playNote(E5, 500);
  playNote(E5, 500);
  playNote(E5, 1000);

  // bar 3:
  playNote(E5, 500);
  playNote(G5, 500);
  playNote(C5, 750);
  playNote(D5, 250);

  // bar 4:
  playNote(E5, 2000);

  // bar 5:
  playNote(F5, 500);
  playNote(F5, 500);
  playNote(F5, 750);
  playNote(F5, 250);
```

JINGLE BELLS

```
// bar 6:
playNote(F5, 500);
playNote(E5, 500);
playNote(E5, 500);
playNote(E5, 250);
playNote(E5, 250);

// bar 7:
playNote(E5, 500);
playNote(D5, 500);
playNote(D5, 500);
playNote(E5, 500);

// bar 8:
playNote(D5, 1000);
playNote(G5, 1000);

// bar 9:
playNote(E5, 500);
playNote(E5, 500);
playNote(E5, 1000);

// bar 10:
playNote(E5, 500);
playNote(E5, 500);
playNote(E5, 1000);
```

JINGLE BELLS

```
// bar 11:
playNote(E5, 500);
playNote(G5, 500);
playNote(C5, 750);
playNote(D5, 250);

// bar 12:
playNote(E5, 2000);

// bar 13:
playNote(F5, 500);
playNote(F5, 500);
playNote(F5, 750);
playNote(F5, 250);

// bar 14:
playNote(F5, 500);
playNote(E5, 500);
playNote(E5, 500);
playNote(E5, 250);
playNote(E5, 250);

// bar 15:
playNote(G5, 500);
playNote(G5, 500);
playNote(F5, 500);
playNote(D5, 500);
```

JINGLE BELLS

```
    playNote(D5, 250);

    // bar 12:
    playNote(E5, 2000);

    // bar 13:
    playNote(F5, 500);
    playNote(F5, 500);
    playNote(F5, 750);
    playNote(F5, 250);

    // bar 14:
    playNote(F5, 500);
    playNote(E5, 500);
    playNote(E5, 500);
    playNote(E5, 250);
    playNote(E5, 250);

    // bar 15:
    playNote(G5, 500);
    playNote(G5, 500);
    playNote(F5, 500);
    playNote(D5, 500);

    // bar 16:
    playNote(C5, 2000);
}
```

TASK 7: EXPERIMENT!

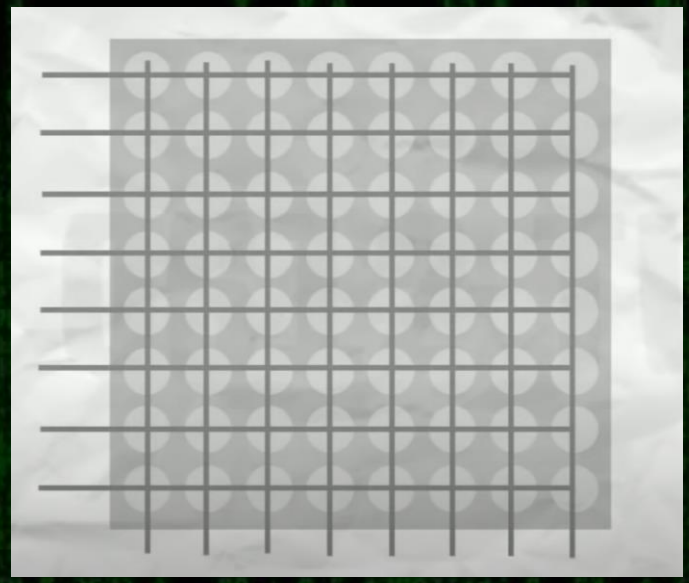
Use the buzzer to play your favourite song.

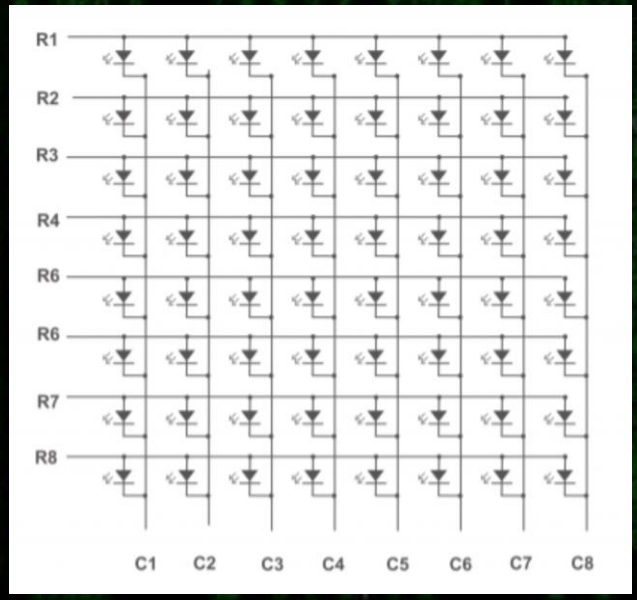
LED Matrix

Flashing Lights

The Matrix

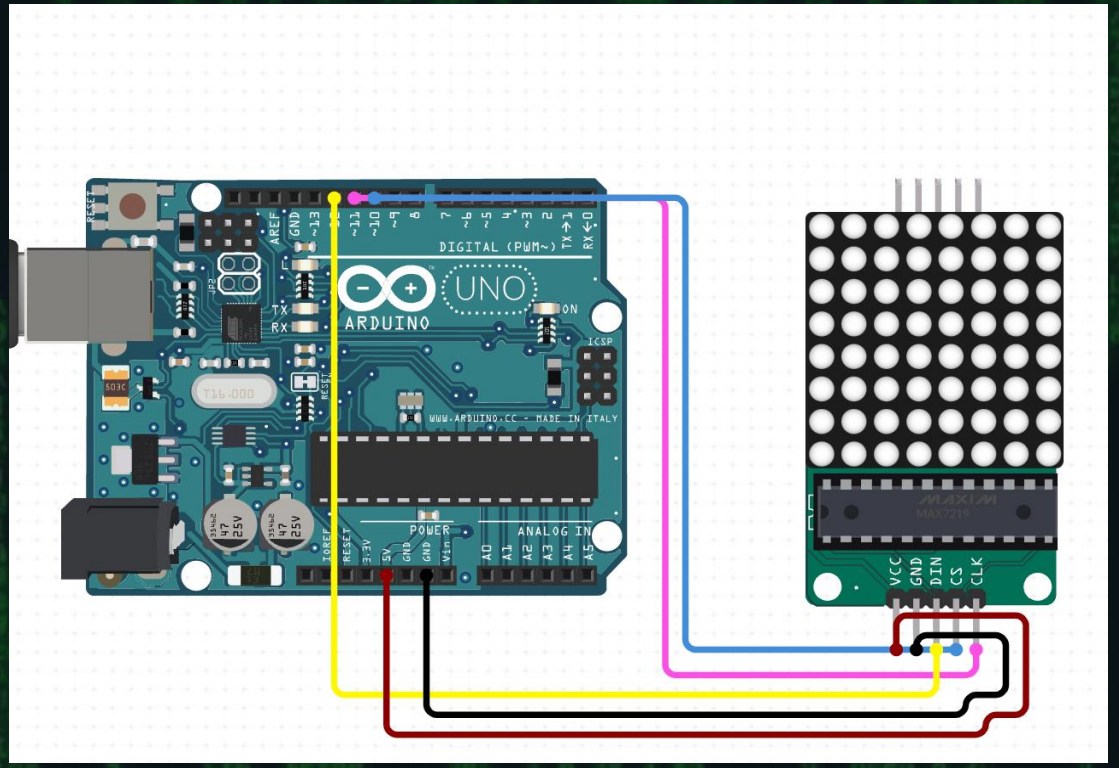
- An LED Matrix is an LED-based display commonly used to display information such as time and scrolling text. The LEDs are located at every row and column intersection of the matrix, which can be seen in the picture to the left. Every grey circle in the picture represents an LED. The schematic also shows you the location of the LEDs inside the matrix.
- Helpful Explanation (1:30 to 6:45): https://youtu.be/G4llo-MRSiY?t=90





Wiring the Matrix

- VCC = 5V pin / rail
- GND = GND pin / Rail
- DIN = any Arduino Pin (2-13)
- CLK = any Arduino Pin (2-13)
- CS = any Arduino Pin (2-13)
- The breadboard can also be used



Library

The MaxMatrix library is used for easy control of the LED matrix.

Resource for installing the MaxMatrix library: https://electronoobs.com/eng_arduino_max_matrix.php#google_vignette

Constructor

Constructor:

MaxMatrix (DIN pin, CS pin, CLK pin, # of MAX7219 modules in use)

Parameter 1: Arduino pin that the DIN pin of LED Matrix is connected to

Parameter 2: Arduino pin that the CS pin of LED Matrix is connected to

Parameter 3: Arduino pin that the CLK pin of LED Matrix is connected to

Parameter 4: number of LED Matrixes (MAX7219) connected together

If DIN is in pin 12, CLK is in pin 11, and CS is in pin 10, and I was only using one LED matrix module, My constructor would be:

```
MaxMatrix my_matrix = MaxMatrix(12, 10, 11, 1);
```

Functions

- There are many functions for the LED matrix.
- init(): Initialize the object
- setIntensity(num): Set LED brightness (0-15)
- setDot(x,y,true): Turn led at (x,y) coordinate on
- clear(): Clear LEDs, turns all LEDs off
- writeSprite(x,y,sprite): Displays sprite (shape) at (x,y) coordinate
    - A sprite is a pointer to a byte or character array containing the data for a shape
- shiftLeft(rotate,fill_zero): shifts the shape to the left by one column
    - if rotate is true, the column that shifts out of the display will shift back in from the opposite side, creating a continuous loop
    - if fill_zero is true, as the shape moves through the display, the LEDs behind the shape will be set to 0 (off)
- shiftRight(rotate, fill_zero): shifts the shape to the right by one column
- shiftUp(rotate): shifts the shape up by one row
- shiftDown(rotate): shifts the shape down by one row

Arrays

- In programming, an array is a series of memory locations that are represented as a single item. They are used to create collections of items with similar datatypes.
- For example, an integer array holds a collection of integers. And a character array holds a collection of characters.
- Here are two examples of how to create an integer array of three integers.

```
int my_int_array[3] = {1,2,3};

// alternatively:

int my_int_array[3] = {
    1,
    2,
    3
};
```

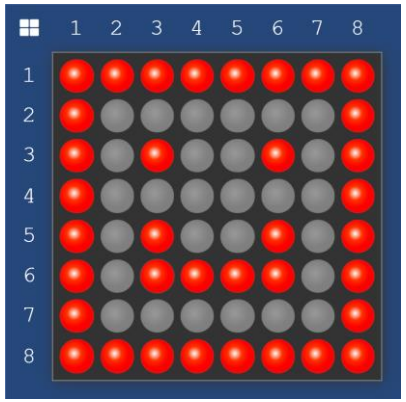
Making an Array for a Sprite

- To make an array for a sprite, you must create a character array. (represented as char in C++).
- The first two items are the dimensions of the Matrix. (8x8) in this case.
- Then, there are 8 items that represent the sprite. Each item is a B(for binary) followed by a series of 0s and 1s that represent an activated or deactivated LED.
- For Example, B10100001 means that LEDs 1, 3, and 8 are on in the current row.
- If the displayed sprite is rotated, the B values can be changed to be oriented correctly.
- To display the array, use writeSprite(x,y,sprite) in void loop like this:

```
my_matrix.writeSprite(0, 0, smiley_face);
```

```
void setup() {
 my_matrix.init();  // initialize the Matrix
 my_matrix.setIntensity(8);  // set brightness to
0 (range is 0-15)
 my_matrix.clear();  // clear the display (turn
off all LEDs)
}
```

```
char smiley_face[] = {
  8, // x dimension
  8,          // y dimension
  B11111111,
  B10000001,
  B10100101,
  B10000001,
  B10100101,
  B10111101,
  B10000001,
  B11111111
};
```



Task 8: Faces

Make character arrays for neutral and sad faces

LED Matrix: Scrolling Sprites

- It is possible for a sprite to move across the matrix.
- To do this, the sprite must be displayed, then cleared, then displayed in a different location. This process must repeat.
- To display a moving sprite of Pac-Man, start by displaying it on coordinated outside of the matrix, then using a for loop, move it one pixel to the right for every iteration.

```
char Pacman[] = {8, 8,
                 B00011100,
                 B00100010,
                 B01000001,
                 B01001001,
                 B01010101,
                 B00100010,
                 B00000000,
                 B00001000};

void setup() {
 my_matrix.init();  // initialize the Matrix
 my_matrix.setIntensity(8);  // set brightness to 0 (range is 0-15)
 my_matrix.clear();  // clear the display (turn off all LEDs)
}

void loop() {
 // start outside the matrix (to the left)
 my_matrix.writeSprite(-8,0,Pacman);

 /* move the Pacman 16 times to the right
  * first 8 times = Pacman is shifted in from the left
  * last 8 times = Pacman is shifted out to the right
  */
 for (int i=1; i<17; i++){
   // update the location of the Pacman
   my_matrix.clear();
   my_matrix.writeSprite(-8+i,0,Pacman);
   delay(100);
 }
 my_matrix.clear();  // clear the display
}
```

Scrolling Sprites

- Alternatively, the MD_MAX72XX and MD_Parola libraries can be used to scroll sprites across the LED matrix.
- To install the libraries:
    1. Sketch → Include Library → Manage Libraries
    2. This will open Arduino's Library Manager, where you can search for each library and install it

Once you have installed the libraries, you will have to "include" them in your Arduino sketch. There are two ways to do this:

1. Sketch → Include Library → "MD_Parola" & "MD_MAX72XX"
2. Use the "#include" keyword: #include <MD_Parola.h> and #include <MD_MAX72xx.h>

MD_Parola Library Constructor

MD_Parola (module type, DIN pin, CLK pin, CS pin, # of MAX7219 modules)

Parameter 1: type of MAX7219 module; ours is "GENERIC_HW"
Parameter 2: Arduino pin that the DIN pin of LED Matrix is connected to
Parameter 3: Arduino pin that the CLK pin of LED Matrix is connected to
Parameter 4: Arduino pin that the CS pin of LED Matrix is connected to
Parameter 5: number of LED Matrixes (MAX7219) connected together

- In the IDE, the constructor would look something like this:

- `MD_Parola my_matrix = MD_Parola(MD_MAX72XX::GENERIC_HW, 12, 11, 10, 1);`

MD_Parola Library Functions

- begin(): initialize the object
- setIntensity(num): set the intensity (brightness) of the display (range 0-15)
- displayClear(): clear the display (turn off all LEDs)
- displayScroll(text, text alignment, effect, speed): scrolling text display
  - text is the text you want to display on the LED Matrix
  - text alignment is how you want to align the text (left, center, right)
  - effect is the type of effect you want to use for the entry and exit of text on the display
  - speed is the time, in milliseconds, between animation frames

Function resource: https://majicdesigns.github.io/MD_Parola/class_m_d___parola.html

Enumerator	Text alignment options
PA_LEFT	The leftmost column for the first character will be on the left side of the display.
PA_CENTER	The text will be placed with equal number of blank display columns either side.
PA_RIGHT	The rightmost column of the last character will be on the right side of the display.

Enumerator	Effect options
PA_NO_EFFECT	Used as a place filler, executes no operation.
PA_PRINT	Text just appears (printed)
PA_SCROLL_UP	Text scrolls up through the display.
PA_SCROLL_DOWN	Text scrolls down through the display.
PA_SCROLL_LEFT	Text scrolls right to left on the display.
PA_SCROLL_RIGHT	Text scrolls left to right on the display.
PA_SPRITE	Text enters and exits using user defined sprite.

Code Breakdown

```
#include <MD_Parola.h>
#include <MD_MAX72xx.h>

MD_Parola my_matrix = MD_Parola(MD_MAX72XX::GENERIC_HW, 12, 11, 10, 1);

const char* text = {“Hello"};

void setup() {
  my_matrix.begin();
  my_matrix.setIntensity(5);
  my_matrix.displayClear();
  my_matrix.displayScroll("Hello", PA_CENTER, PA_SCROLL_LEFT, 100);
}

void loop() {

  if(my_matrix.displayAnimate()){
    my_matrix.displayReset();
  }

}
```

Include Libraries

Constructor

Initialize matrix

Set matrix brightness to 5 and clear display (all LEDs off)

This is an if, then statement. It follows the format:

If(condition){
  Do this
}

If the condition is true, then whatever is inside of it will be executed by the computer. In this case, this statement perpetually resets the matrix, constantly allowing it to use displayScroll()

Display “Hello”, centered in the center of the Matrix, scrolling left, with 100ms of delay each time the display refreshes

```
const char* text = {“Hello"};
```

Pointers

- “text” is a pointer.
- In computer science, a pointer is a variable that stores the memory address of another variable or a value. In other words, it shows where a value is stored, instead of storing the value.
- In this case, text acts as a string, which is a collection of characters.

Task 9: Scroll

- Use the LED matrix to output a smiley face symbol :) and use PA_SCROLL to animate it. Experiment with the arguments of the displayScroll() function.

displayScroll(text, text alignment, effect, speed)

More effect options

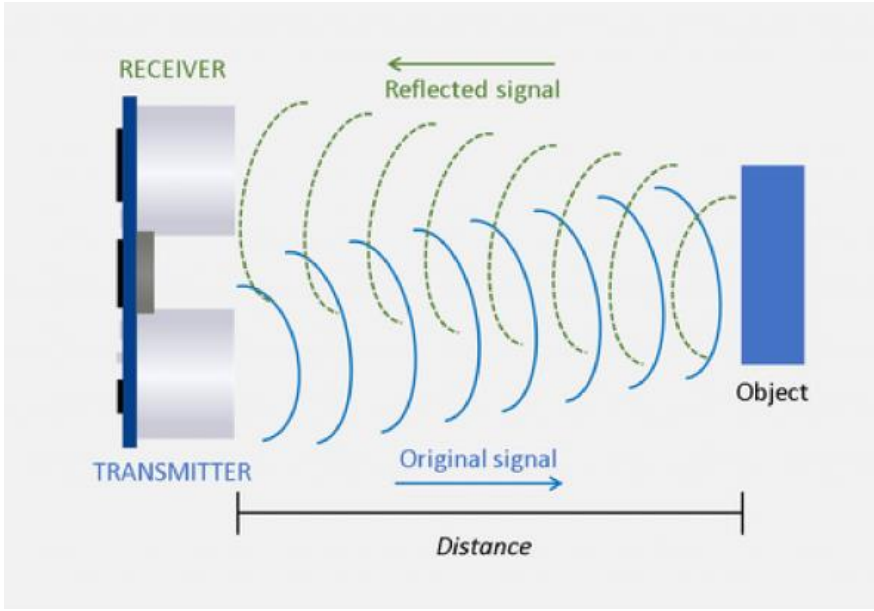
Enumerator	
PA_NO_EFFECT	Used as a place filler, executes no operation.
PA_PRINT	Text just appears (printed)
PA_SCROLL_UP	Text scrolls up through the display.
PA_SCROLL_DOWN	Text scrolls down through the display.
PA_SCROLL_LEFT	Text scrolls right to left on the display.
PA_SCROLL_RIGHT	Text scrolls left to right on the display.
PA_SPRITE	Text enters and exits using user defined sprite.
PA_SLICE	Text enters and exits a slice (column) at a time from the right.
PA_MESH	Text enters and exits in columns moving in alternate direction (U/D)
PA_FADE	Text enters and exits by fading from/to 0 and intensity setting.
PA_DISSOLVE	Text dissolves from one display to another.
PA_BLINDS	Text is replaced behind vertical blinds.
PA_RANDOM	Text enters and exits as random dots.
PA_WIPE	Text appears/disappears one column at a time, looks like it is wiped on and off.
PA_WIPE_CURSOR	WIPE with a light bar ahead of the change.
PA_SCAN_HORIZ	Scan the LED column one at a time then appears/disappear at end.
PA_SCAN_HORIZX	Scan a blank column through the text one column at a time then appears/disappear at end.
PA_SCAN_VERT	Scan the LED row one at a time then appears/disappear at end.
PA_SCAN_VERTX	Scan a blank row through the text one row at a time then appears/disappear at end.
PA_OPENING	Appear and disappear from the center of the display, towards the ends.
PA_OPENING_CURSOR	OPENING with light bars ahead of the change.
PA_CLOSING	Appear and disappear from the ends of the display, towards the middle.
PA_CLOSING_CURSOR	CLOSING with light bars ahead of the change.
PA_SCROLL_UP_LEFT	Text moves in/out in a diagonal path up and left (North East)
PA_SCROLL_UP_RIGHT	Text moves in/out in a diagonal path up and right (North West)
PA_SCROLL_DOWN_LEFT	Text moves in/out in a diagonal path down and left (South East)
PA_SCROLL_DOWN_RIGHT	Text moves in/out in a diagonal path down and right (North West)
PA_GROW_UP	Text grows from the bottom up and shrinks from the top down.
PA_GROW_DOWN	Text grows from the top down and and shrinks from the bottom up.

ULTRA**SONIC** SENSOR

Sound bounces

WHAT IS AN ULTRASONIC SENSOR?

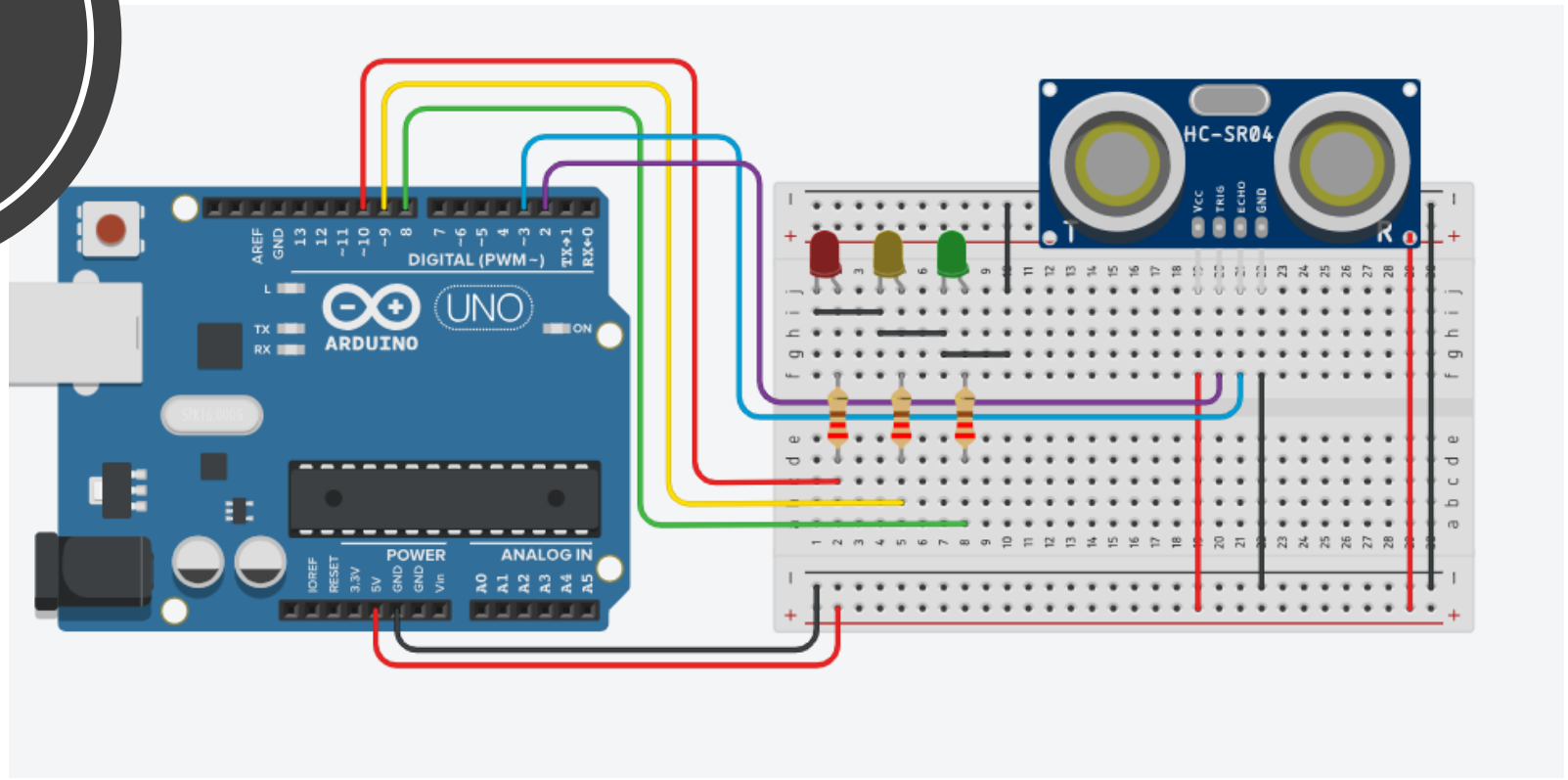
- An ultrasonic sensor is a sensor that uses sound to measure distance from it to an object.
- The sensor consists of a transmitter and a receiver. The transmitter emits ultrasonic sound waves beyond human hearing range, and the receiver catches these waves.
- Distance is determined based on how long the waves take to make a journey from the transmitter to the receiver after bouncing off an object. The farther the object is, the longer it takes for the ultrasonic waves to travel.
- Based on the time, the distance can be computed using this formula: distance = (duration / 2) * 0.034



THE DISTANCE EQUATION

- (duration / 2) * 0.034 is used to calculate the distance. How does it work?
- Duration is divided by 2 because it is the duration for the 2-way trip.
- The speed of sound is 340 m/s. However, the duration measured by the ultrasonic sensor is in microseconds (μs), and we want the distance in centimeters. Therefore, we must convert the speed of sound to cm/μs, which is 0.034 cm/μs. Now, following the formula: distance = speed x time, we derive the following equation: distance = (duration / 2) * 0.034

WIRING – SENSOR & TRAFFIC LIGHT



The trig, echo, and all the traffic light data pins are connected to the Arduino. Vcc and both GND pins are connected to ground

EXAMPLE: PROXIMITY LIGHTS

- What if I want to write code using the sensor where different LEDs light up depending on the distance of a target?
- If the target is within 50 cm of the sensor, the green light would turn on.
- If the target is between 51 and 75 cm of the sensor, the yellow light would turn on.
- If the target is more than 75 cm away from the sensor, the red light would turn on.

```
const int trig = 2;
const int echo = 3;

long duration;
int distance;

const int red = 8;
const int yellow = 9;
const int green =  10;

void setup() {

  pinMode(red, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(green, OUTPUT);

  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);

  Serial.begin(9600);

}

void loop() {

  digitalWrite(trig, LOW);
  delayMicroseconds(2);

  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  duration = pulseIn(echo, HIGH);

  distance = (duration/2) * 0.034;


  if (distance < 50){
    digitalWrite(green, HIGH);
    digitalWrite(yellow, LOW);
    digitalWrite(red, LOW);
  }

  else if (distance > 50 && distance < 75){
    digitalWrite(green, LOW);
    digitalWrite(yellow, HIGH);
    digitalWrite(red, LOW);
  }

  else{
    digitalWrite(green, LOW);
    digitalWrite(yellow, LOW);
    digitalWrite(red, HIGH);
  }

  Serial.println(distance);

}
```

CODE

CODE - VARIABLES

```
const int trig = 2;
const int echo = 3;

long duration;
int distance;

const int red = 8;
const int yellow = 9;
const int green =  10;
```

- The variables for the trig, echo, and LED pins are all const int, because they are integers that are constant, and do not need to be changed.
- Duration is a long, which is a signed 32-bit number. They can range from -2,147,483,648 to 2,147,483,647
- Distance is int rather than const int because it is not constant. It is frequently measured and changed.

CODE - PINMODE

```
void setup() {

  pinMode(red, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(green, OUTPUT);

  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);

  Serial.begin(9600);

}
```

- All of the pins are outputs except for the echo pin, which is an input.
- The trig (trigger) pin is used to trigger an ultrasonic sound pulse, while echo receives it after it bounces off of an object.
- Since echo must measure this pulse and tell the Arduino microcontroller about it, it needs an input pin.

```
void loop() {

  digitalWrite(trig, LOW);
  delayMicroseconds(2);

  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  duration = pulseIn(echo, HIGH);

  distance = (duration/2) * 0.034;
```

CODE - PULSE

- Trig, which triggers an ultrasonic pulse is disabled for 2 microseconds (for reference, there are 1000 microseconds in a millisecond).
- Then, trig is activated for 10 microseconds, sending out an ultrasonic pulse.
- The time that the pulse takes to reach the object then back to the Arduino is recorded and moved to the variable *duration* using the pulseIn function.
- The distance formula is used to calculate the distance based off of the duration.

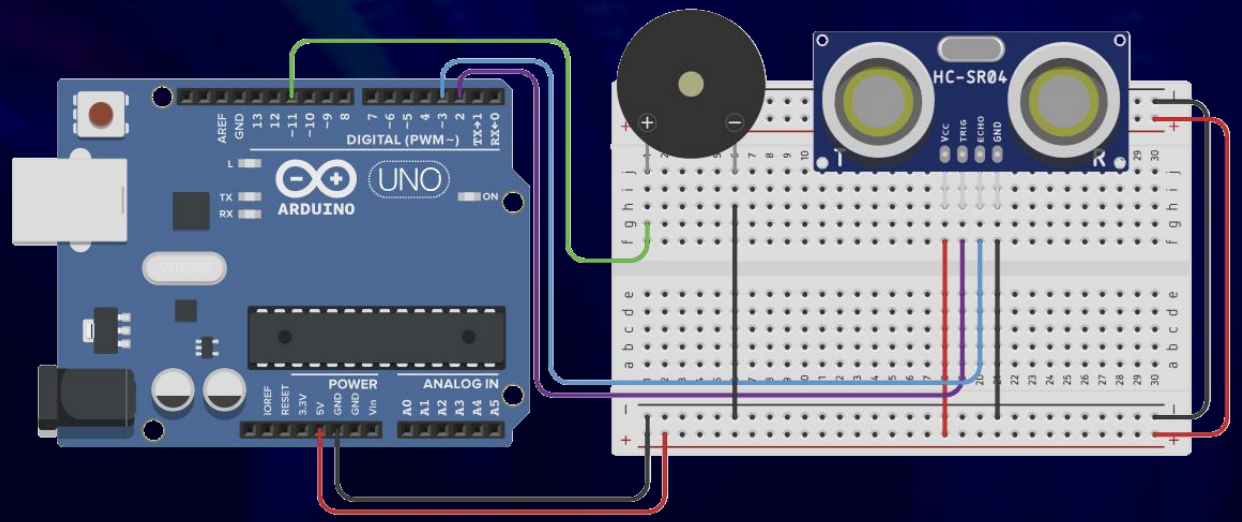
```
  if (distance < 50){
    digitalWrite(green, HIGH);
    digitalWrite(yellow, LOW);
    digitalWrite(red, LOW);
  }

  else if (distance > 50 && distance < 75){
    digitalWrite(green, LOW);
    digitalWrite(yellow, HIGH);
    digitalWrite(red, LOW);
  }

  else{
    digitalWrite(green, LOW);
    digitalWrite(yellow, LOW);
    digitalWrite(red, HIGH);
  }

  Serial.println(distance);

}
```

CODE – IF, ELSE IF, ELSE

- If the distance is less than 50, all three of the LEDs are turned off except for the green one.
- ***else if*** is generally used in situations where more than one ***if*** statement is needed. The ***else if*** statement will only be checked if the ***if*** statement is false.
- This is different from just ***if*** statements because with ***if*** statements, they are all checked separately, whether or not any are false.
- The ***else*** statement is like an ***else if*** statement with no condition. If all of the ***if*** and ***else if*** statements before it are false, whatever is in the ***else*** block will run.
- Serial.println(distance) is used to print the distance to the serial monitor. This is not necessary, but it can be used to keep track of the distance and diagnose potential problems with the setup/code.

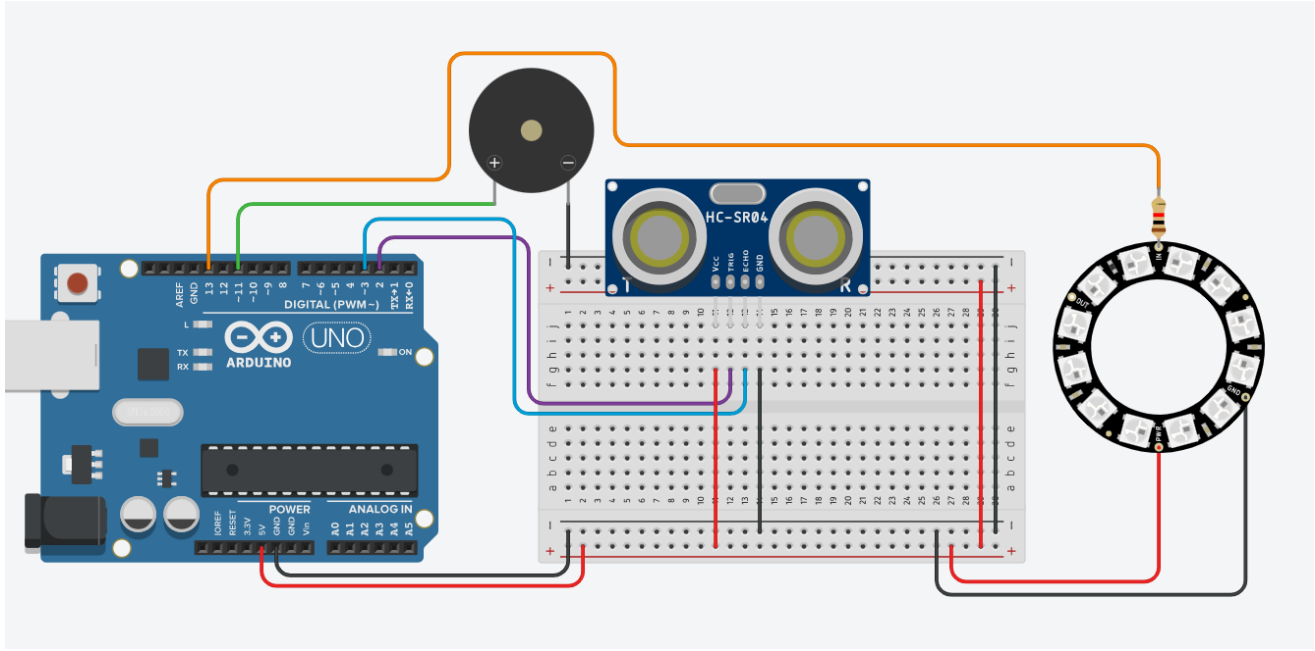
TASK 10: ULTRASONIC BEEPS

- Use a piezo buzzer and an ultrasonic sensor to create a proximity siren.
- The buzzer should play a repeated beeping sound (like the beep boop exercise).
- It should play a higher-pitched tone with more rapid beeps as the target comes closer to the sensor.



TASK 11:LOUD SPIRALS

- Attach a NeoPixel ring, an ultrasonic sensor, and a buzzer to the Arduino.
- Modify your code from task 10 to include the NeoPixel ring.
- Make the ring create a spiral of different colours depending on the distance between the sensor and the target.
- 0-50 cm: red spiral
- 50-100 cm: yellow spiral
- 100+ cm: green spiral



Analog and Digital

Analog and digital are words that are commonly used to differentiate between items that share a purpose.

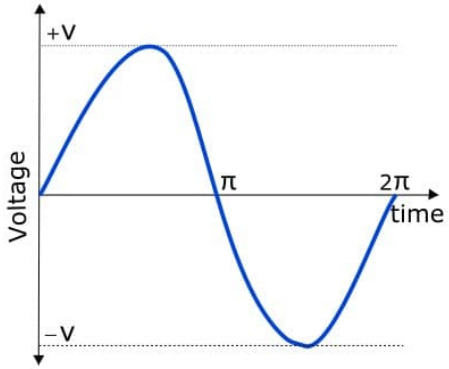
A common example are clocks. Analog clocks use hands to display the time while digital clocks use a digital display.

Another example are analog and digital music formats. An example of a digital audio format is an mp3 file. An example of an analog format would be a vinyl record.

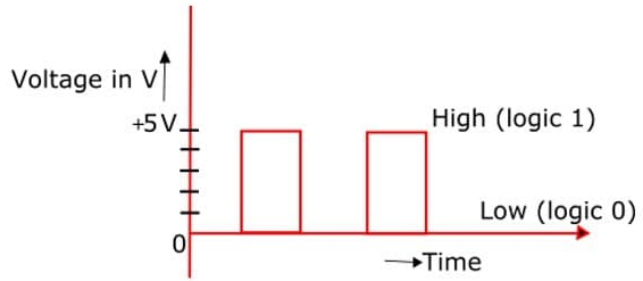
What is the Difference?

- Analog information can be read as electric pulses of varying amplitude, while digital information can only be binary, read as only one of two possible amplitudes.

Analog Circuits vs Digital Circuits



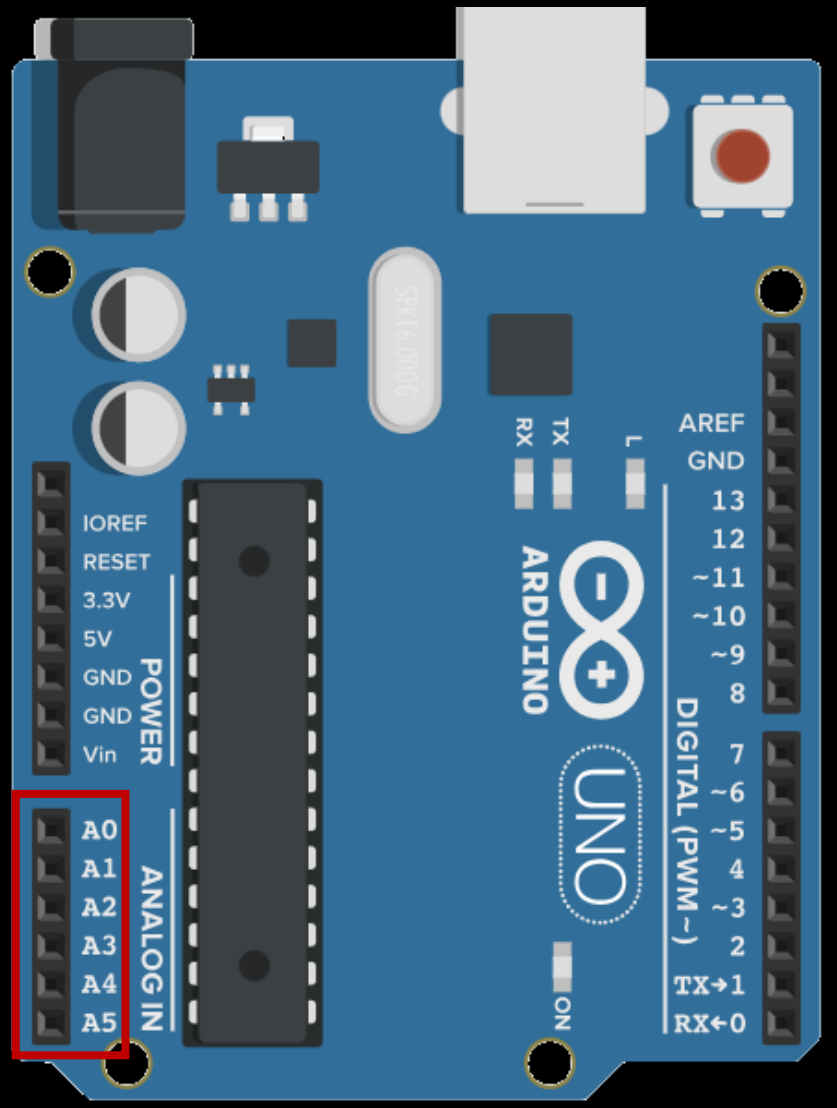
Example of Analog Signal



Example of Digital Signal

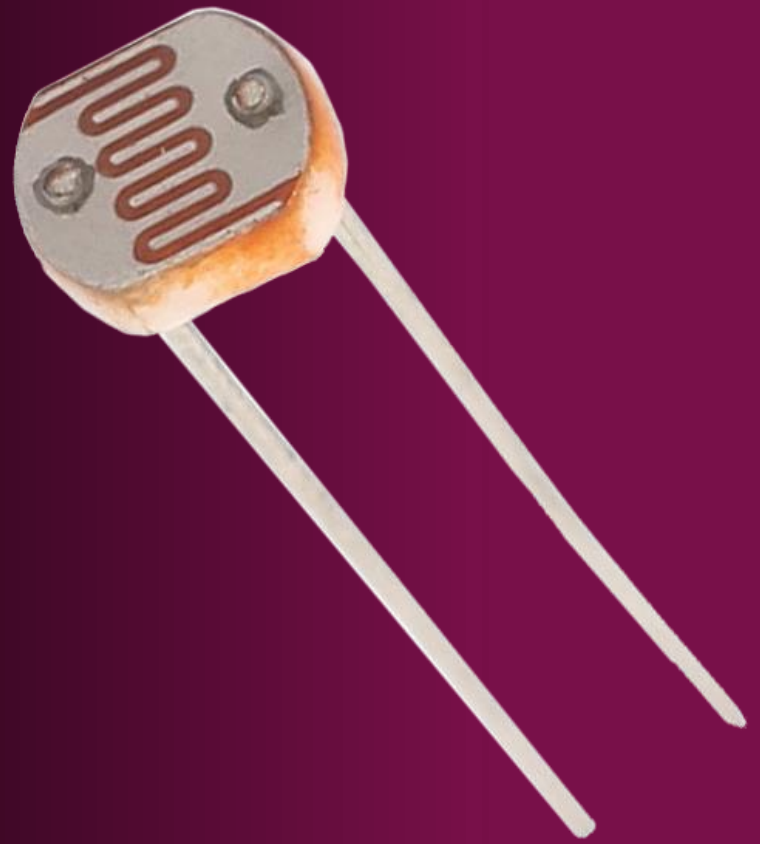
Arduino Analog pins

- The Arduino Uno has 5 analog pins, labeled A0 to A5.
- These pins can be used to input and output analog signals.
- The Arduino has a built-in analog to digital converter that converts analog signals to digital and back for the Arduino to use.
- Using an analog pin is very similar to using a digital pin. The functions simply use *analog* instead of *digital.*
- Analog functions would not use HIGH or LOW, but rather a numerical value between 0 and 1023. This number is limited by the Arduino's 10 bit analog-to-digital converter.
- For example, digitalWrite(3, HIGH); could be analogWrite(A3, 1023);



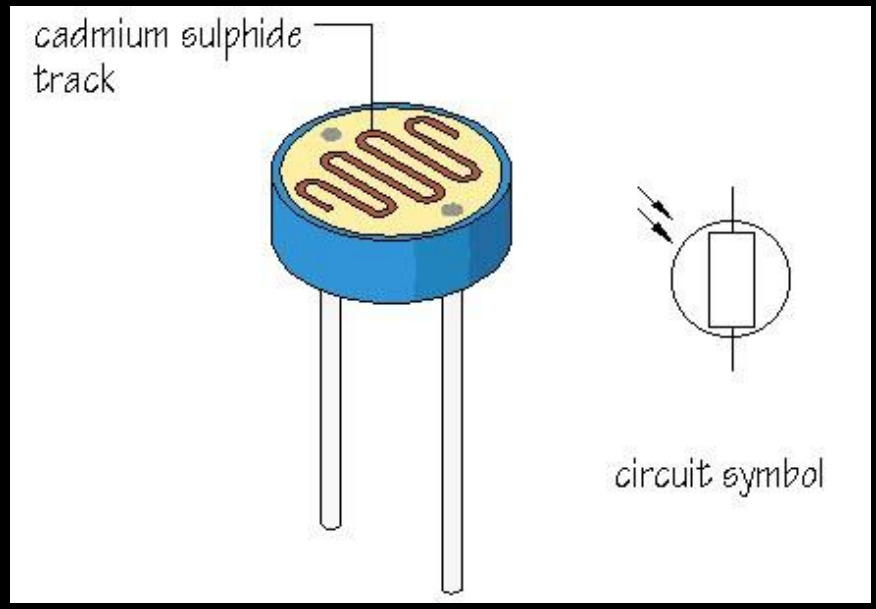
Light **D**ependent **R**esistor

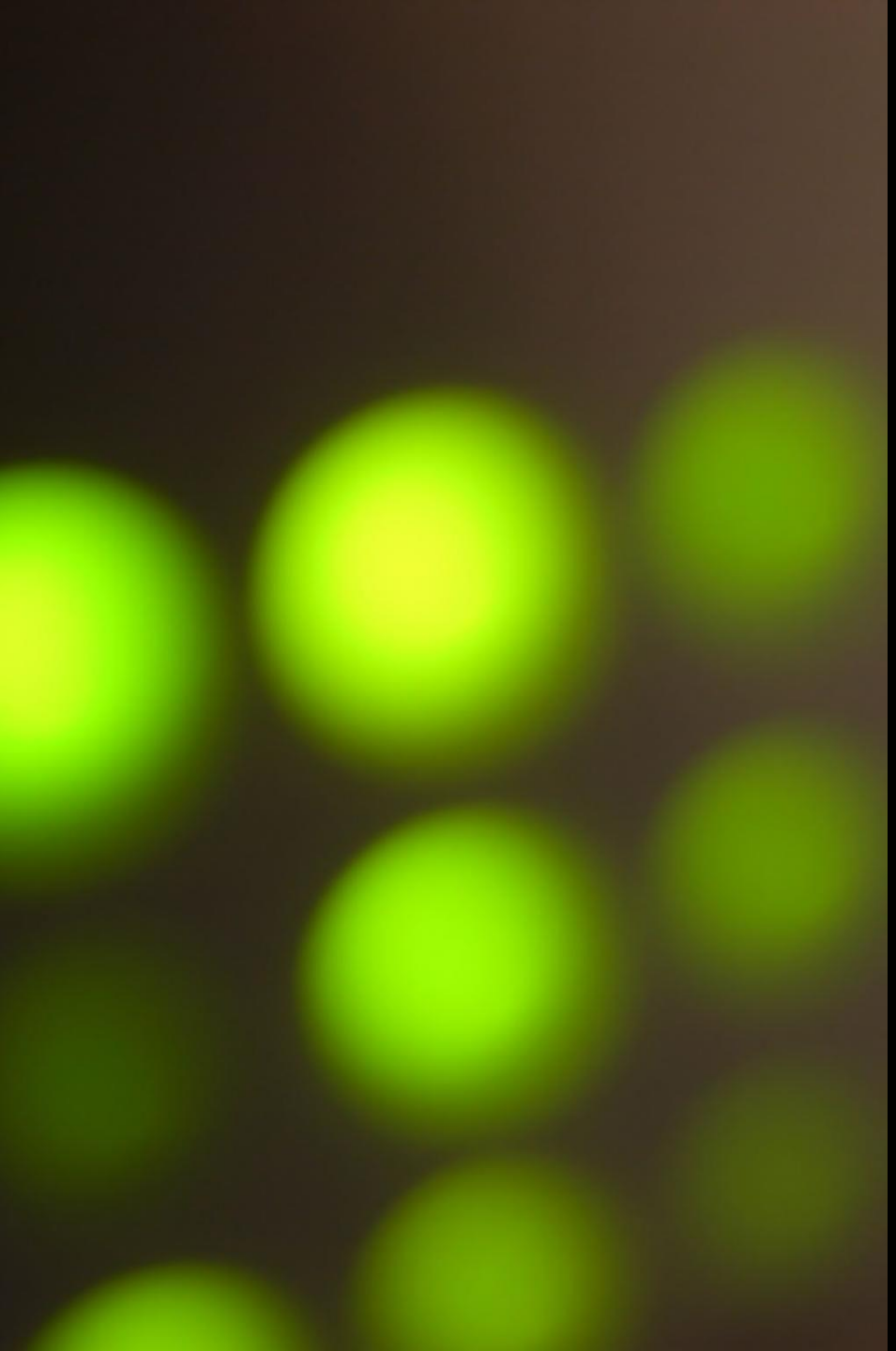
Or LDR,
or photoresistor



What is an LDR?

- An LDR is a resistor that has a variable resistance based on the intensity of the light that it is exposed to.
- For example, if the LDR is in direct sunlight, the resistance will be lower than if it were in a dark room.
- In the LDR is a cadmium sulphide track. When light hits it, the photons of the light excite the valence electrons in the LDR, which reduces the material's conductivity, which then increases the resistance.



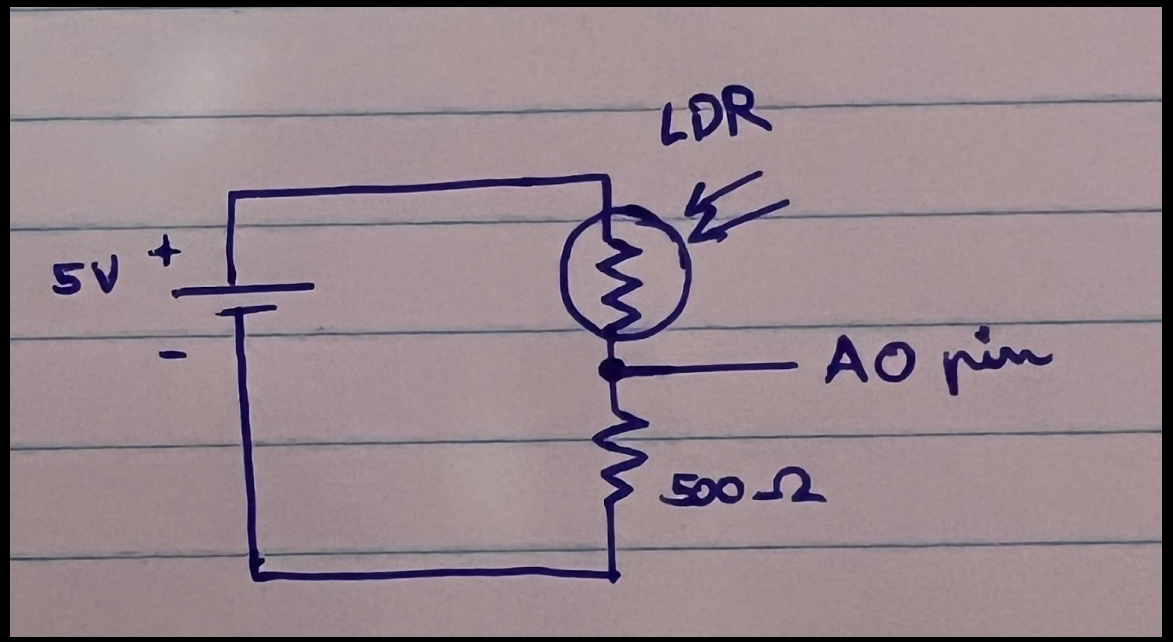


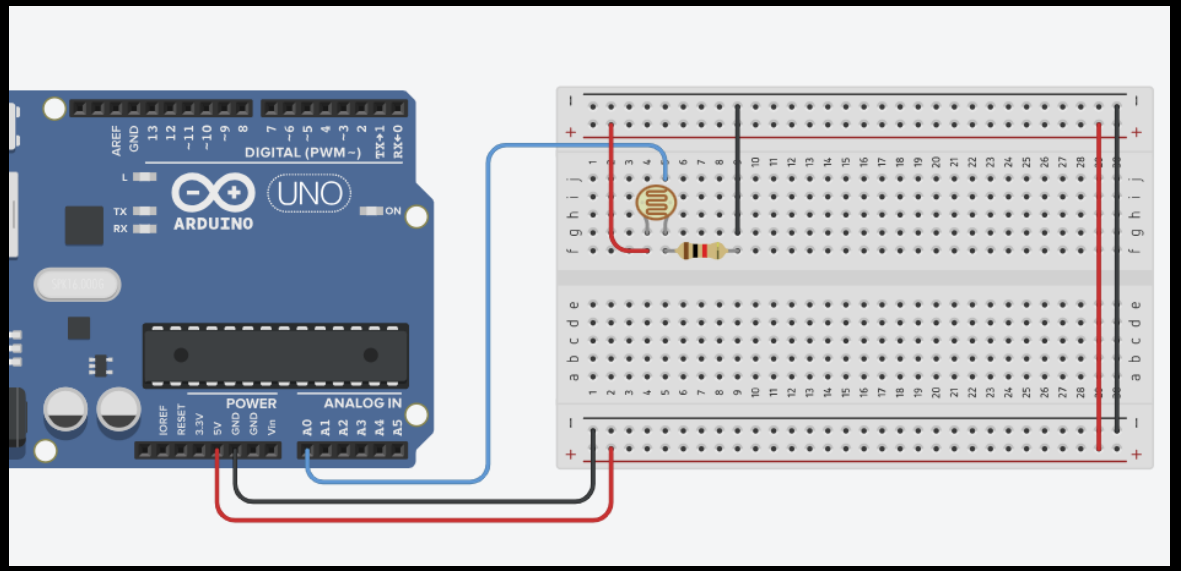
Real World Applications

- LDR’s can be used for many applications:
  - Automatic streetlights
  - Automatic night lights
  - Mobile phone auto-brightness
  - Automatic camera flash control

LDR → Arduino

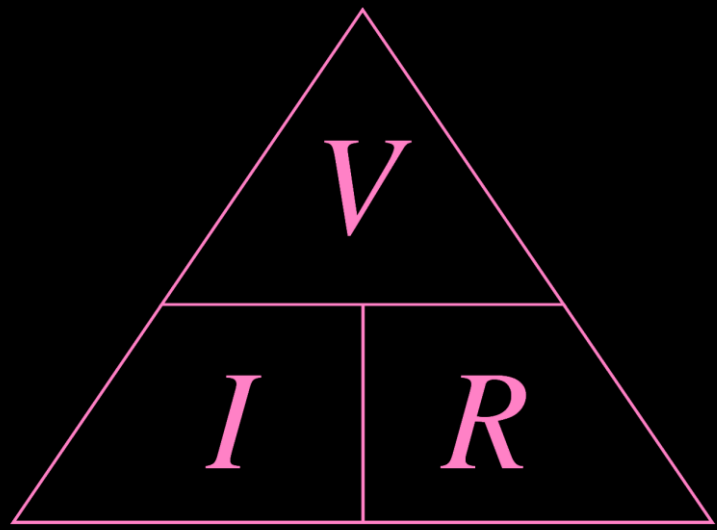
- Connect the LDR in series to an **analog** Arduino pin (labeled A0-A5).
- Include a 550Ω resistor.





LDR

- The LDR value is essentially the voltage drop across the LDR. This gives us an indication of the LDR's resistance, since according to Ohm's Law, current stays the same in a series circuit. Therefore, the voltage drop across a load depends on its resistance. We need to use an additional resistor in our circuit because if the LDR was the only load in the circuit, all 5 volts will be supplied to the LDR, leading to a voltage drop of 5 volts.



Programming the LDR

- The LDR uses the analogRead() function to read a value from a component.
- Because of this, pinMode(pin #, INPUT) will be used.
- The value that the Arduino reads from the LDR can be printed to the serial monitor.

Example: Print to Serial Monitor

```
const int LDR_pin = 0;
int LDR_value = 0;

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

void loop() {
  LDR_value = analogRead(LDR_pin);
  Serial.print("LDR Value is: ");
  Serial.println(LDR_value);
  delay(200);
}
```

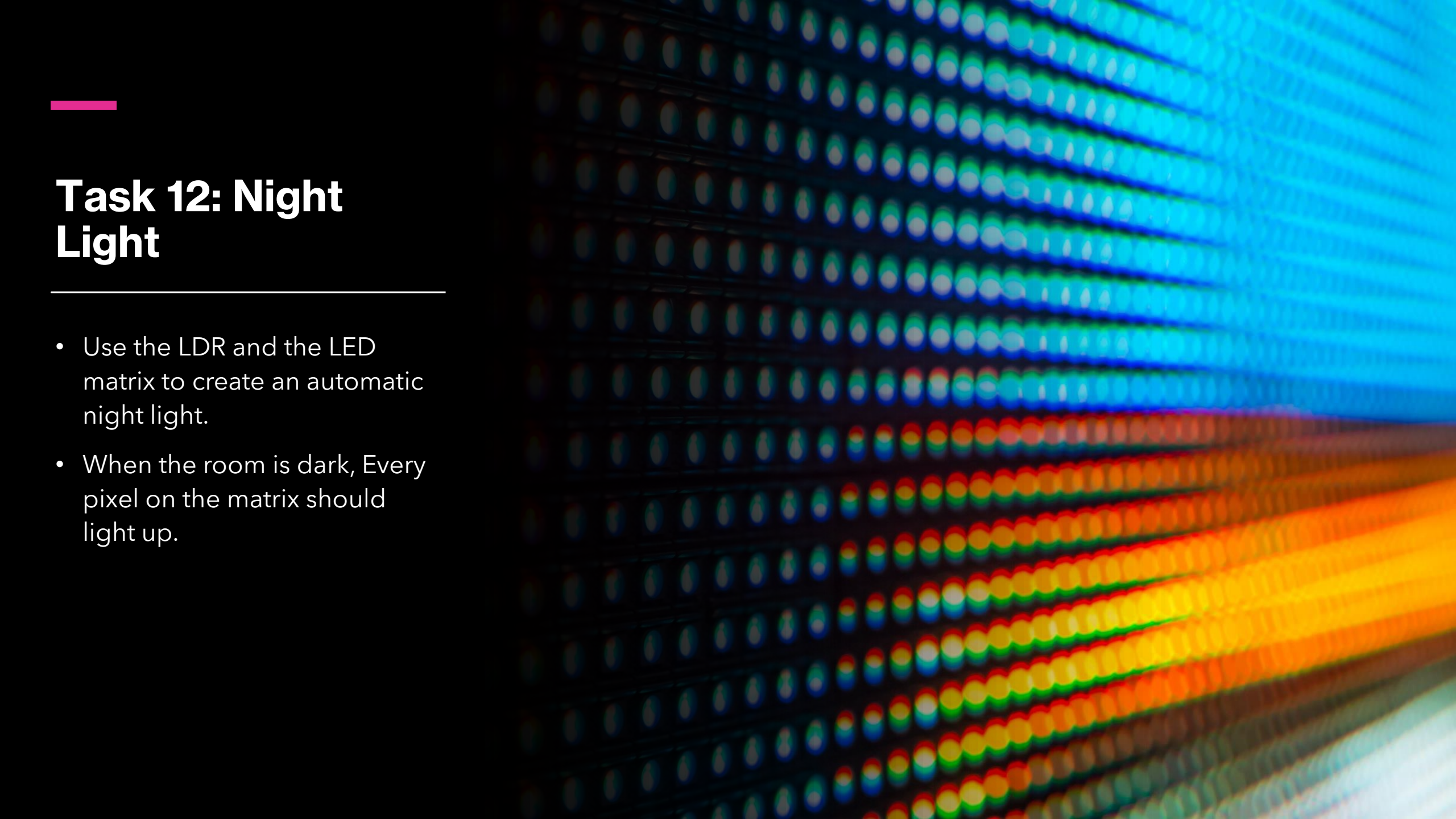
LDR_pin is a const int because it is the LDR pin and doesn't change.

LDR_value is not a const int because the value changes frequently.

The LDR_value is the value that is read from the LDR_pin by the Arduino. It is then printed to the serial monitor. It is refreshed every 200 milliseconds.

Task 12: Night Light

- Use the LDR and the LED matrix to create an automatic night light.
- When the room is dark, Every pixel on the matrix should light up.



Task 13: Adaptive Night Light

- Use the LDR and the LED matrix to create an automatic night light that changes the brightness of the LED matrix based on the brightness of the room.
- Include three different brightness modes.

I2C LCD

Readable text

WHAT IS AN LCD?

LCD stands for **L**iquid **C**rystal **D**isplay.

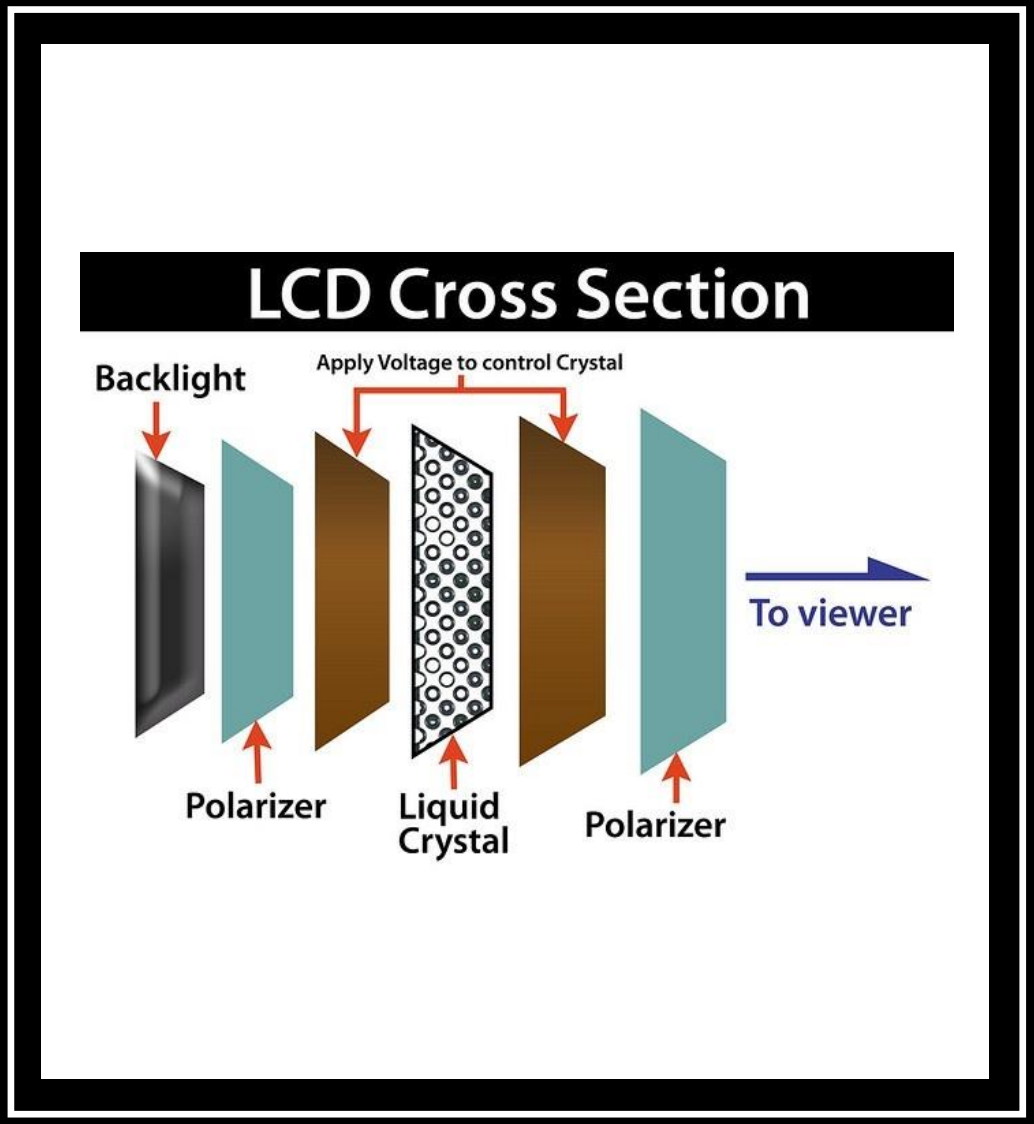
In an LCD, there is a layer of liquid crystal material that are sandwiched between electrodes. On either side of this is a polarizer that only lets certain light waves out.

The liquid crystals bend the light for it to pass through the polarizers.

Colour LCDs In front of this system is an RGB filter that is used to display either of the three colours.

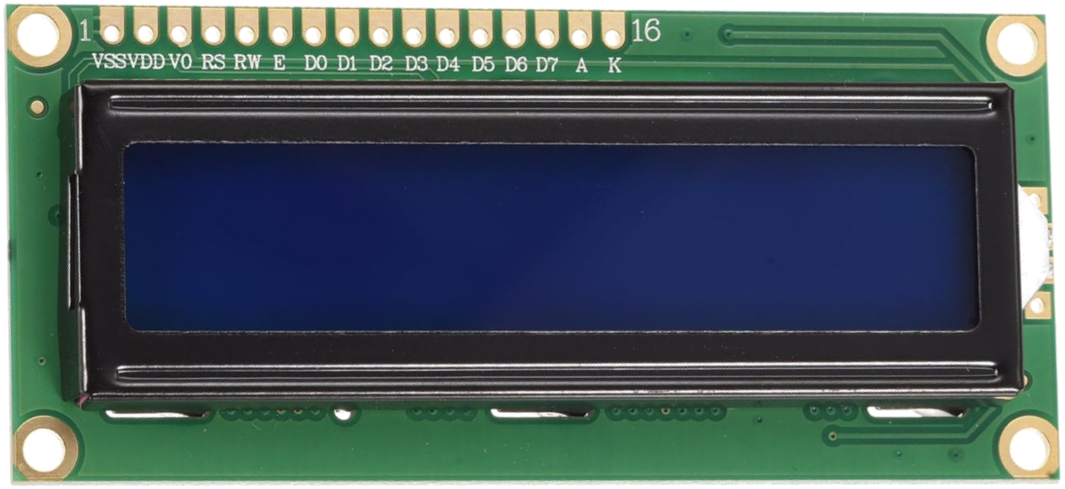
There is also a backlight that illuminates the display. Otherwise, it would be too dark to see.

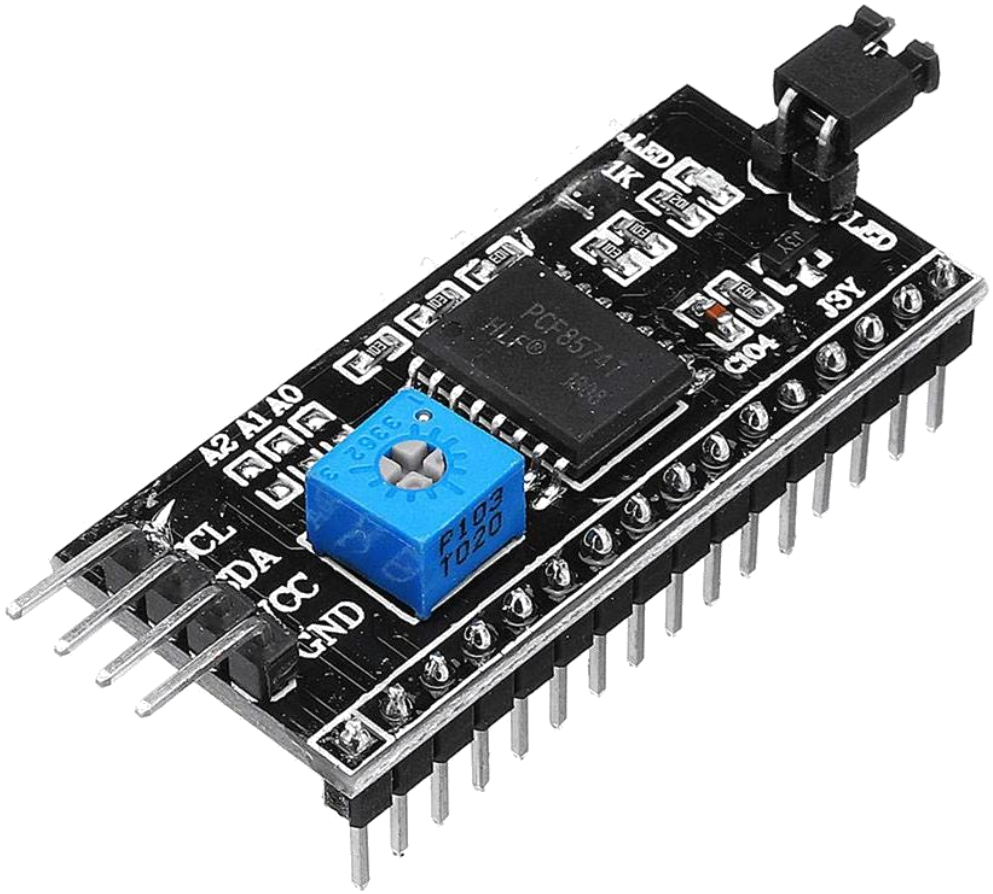
More: https://www.youtube.com/watch?v=VamqtyatBss&ab_channel=AKIOTV



16 x 2 LCD

- This LCD has a 16x2 grid of spaces, each containing a smaller 5x8 grid that is used for displaying characters.
- It is used with an I2C for easy operation.

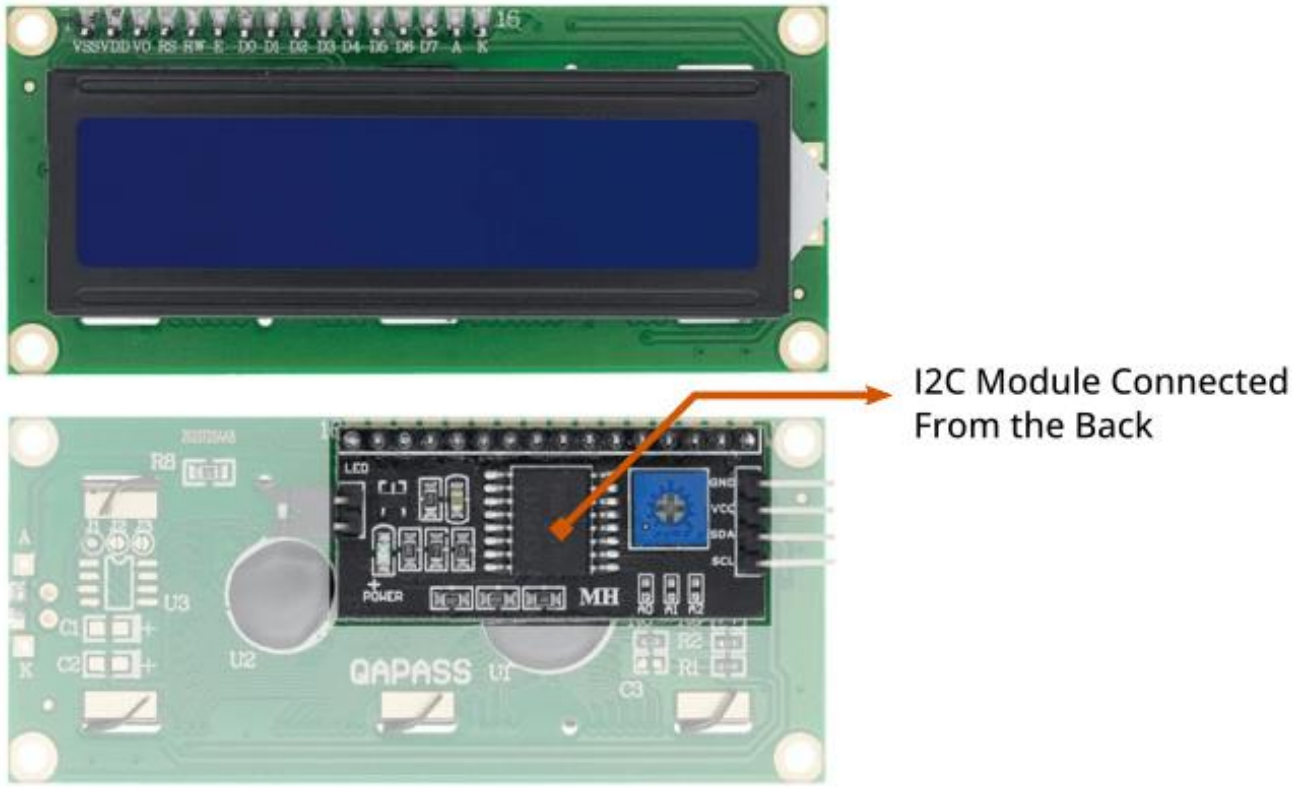




WHAT IS AN I2C?

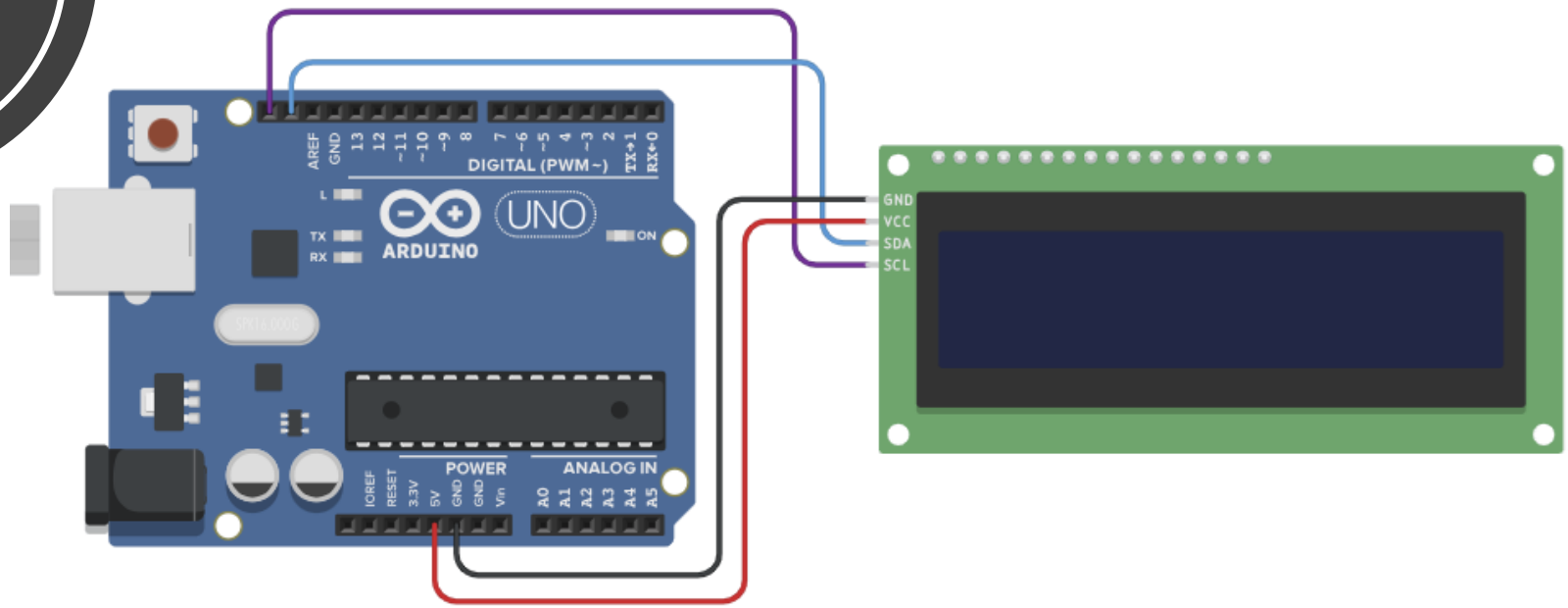
- **I2C** stands for **I**nter-**I**ntegrated **C**ircuit.
- It is used for easier communication between the Arduino and the LCD.
- It utilizes the serial data (SDA) and serial clock (SCL) pins on the Arduino microcontroller.

LCD + I2C

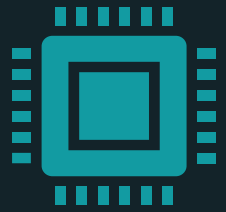


- An I2C bus can be attached to a 16x2 LCD for easier control of it. The module can then be connected to the SDA and SCL pins on the Arduino.

LCD → ARDUINO


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5
GND
VCC
SDA
SCL

LIQUIDCRYSTAL_I2C LIBRARY



The LCD can be programmed by using the LiquidCrystal_I2C library.

The library has functions that will make the LCD easier to control.

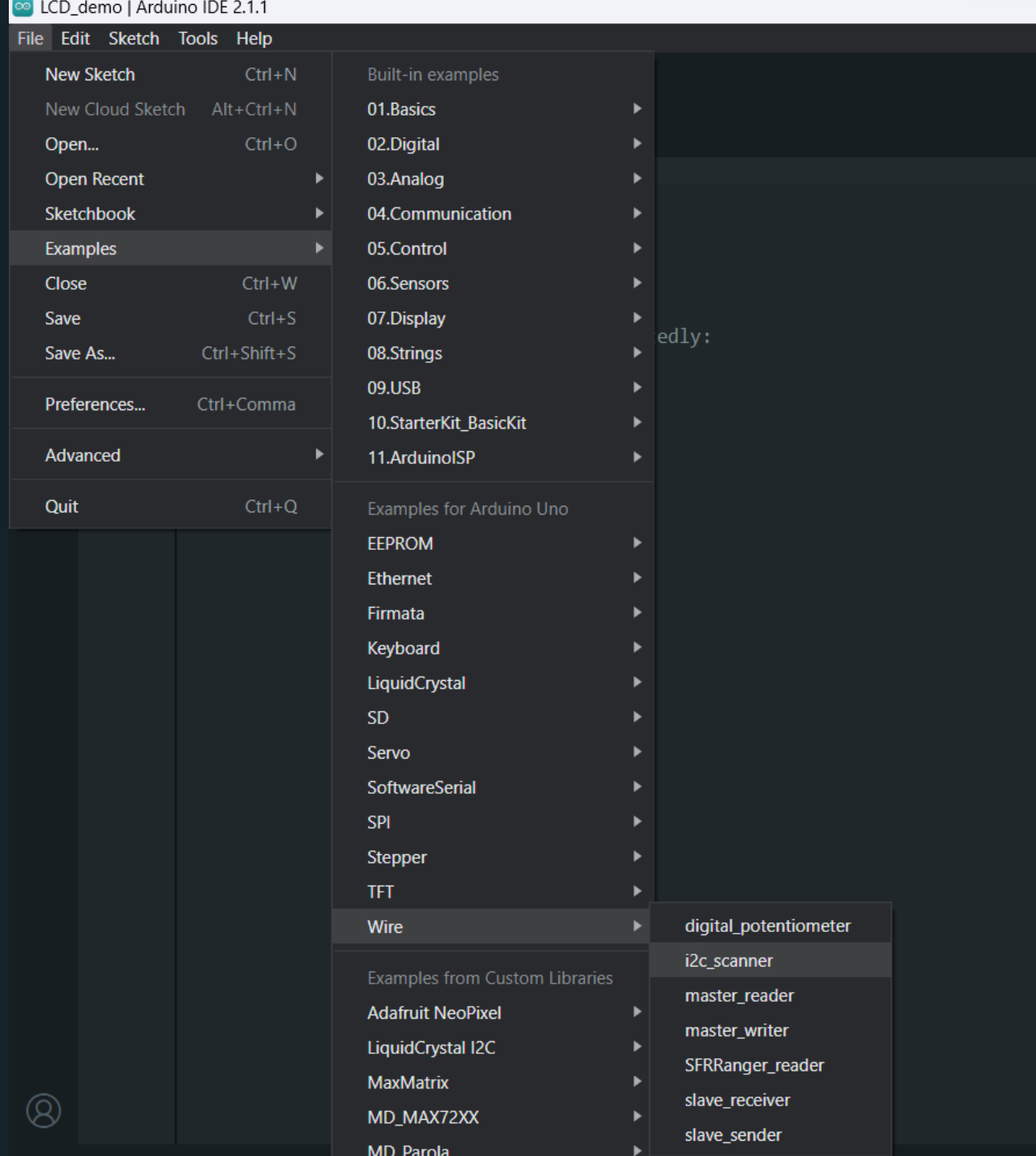
To install the library:

Sketch → Include Library → Manage Libraries.

This will open Arduino's Library Manager, where you can search for the library and install it.

ADDRESS

- Each I2C has an address, and this address can differ based on the manufacturer.
- The address can be found by running a program that can be found in the IDE.
- To access it, navigate to:
  - File > Examples > Wire > i2c_scanner
- The address of the I2C will be outputted to the serial monitor.
- It should look *something* like this: ***0x27***



PROGRAMMING THE LCD

```
#include <LiquidCrystal_I2C.h>  // Including the library

LiquidCrystal_I2C my_lcd = LiquidCrystal_I2C(0x27,16,2);

/*
  Above: This initializes an object named "my_lcd" using a constructor of three arguments
  Argument 1: Address
  Argument 2: Width (16 spaces)
  Argument 3: Height (2 spaces)
*/

void setup() {

  my_lcd.init();    // Initializes the LCD
  my_lcd.clear();   // Clears the LCD (empty display)
  my_lcd.backlight();   // Activates the LCD's backlight

  my_lcd.setCursor(2,0);  // Sets cursor position (x, y) (2nd space, 1st line)
  my_lcd.print("Hello world!"); // Displays "Hello world!" on the display
}
```

CUSTOM CHARACTERS

- Much like with the LED Matrix, custom characters can be made and displayed on each 5x8 sector on the LCD.
- They can be made using an array with the ***byte*** datatype and initialized by using the createChar() function.
- The createChar function takes two arguments,
  - The first argument is a number that is used to store the character. This number is then used in the write() function as an identifier.
  - The second argument is the name of the byte array that is used to create the character.

```
byte heart[8] = {
0b00000,
0b00000,
0b01010,
0b11111,
0b11111,
0b01110,
0b00100,
0b00000
};

byte key[8] = {
0b00000,
0b00110,
0b00100,
0b00110,
0b00100,
0b01110,
0b01010,
0b01110
};

LiquidCrystal_I2C my_lcd = LiquidCrystal_I2C(0x27,16,2);

void setup() {
  my_lcd.init();
  my_lcd.clear();
  my_lcd.backlight();

  my_lcd.createChar(0, key);
  my_lcd.createChar(1, heart);

  my_lcd.setCursor(7, 0);
  my_lcd.write(0);  // displays the key on the LCD

  my_lcd.setCursor(8, 0);
  my_lcd.write(1); // displays the heart on the LCD
}
```

TASK 14: MOVING CHARACTER

- Create any custom character and have it repeatedly move across the top row of the display, then the bottom row.
- Helpful resource for custom characters: https://maxpromer.github.io/LCD-Character-Creator/.
- Hint: Use a for loop.

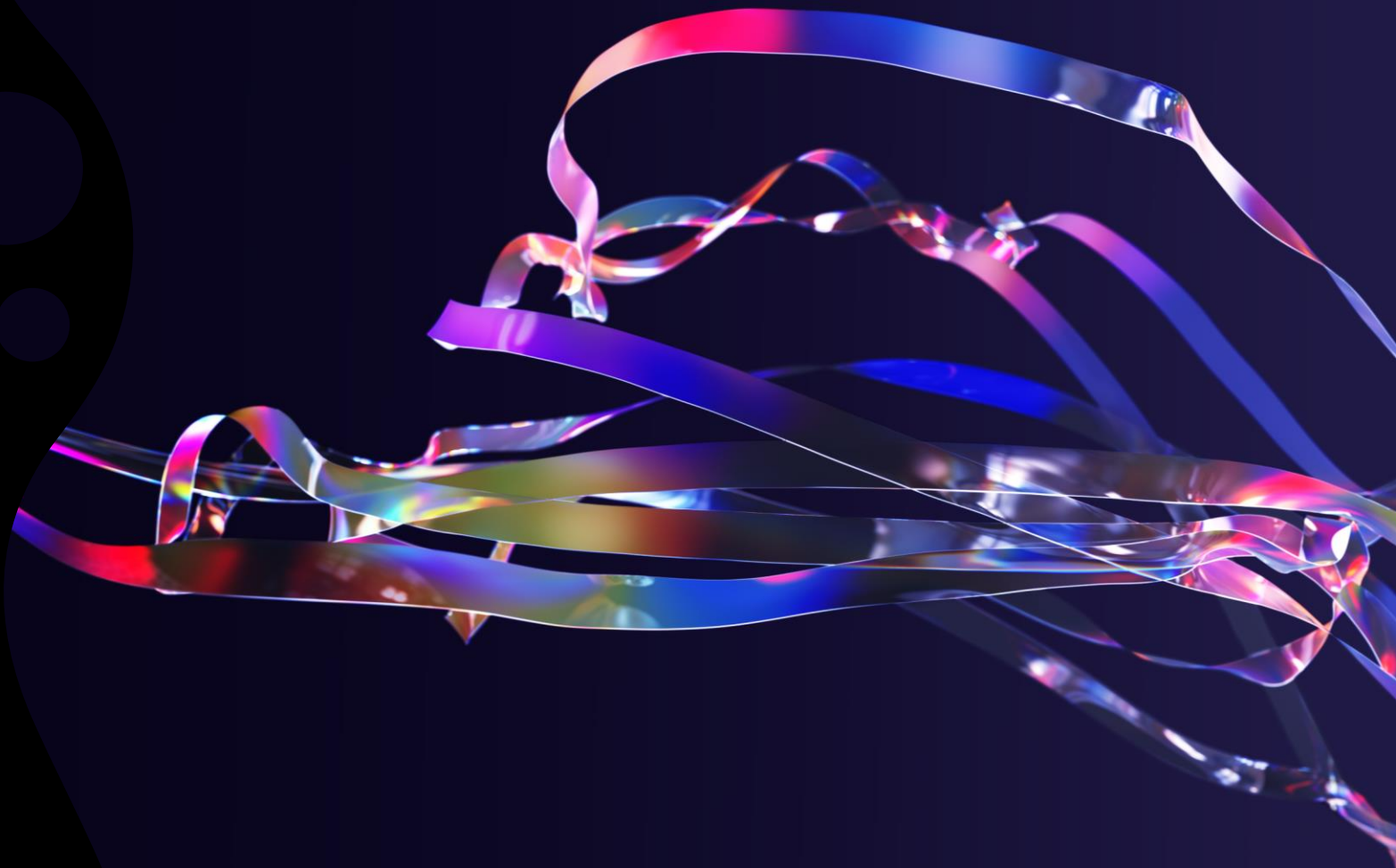
- ***Take it further (optional):*** Alternate between the top and bottom rows so that the character looks like it's simultaneously jumping and moving across the display.

TASK 15: LIGHT INDICATOR

- Use an LDR alongside the LCD display to create a light level indicator.
- If the surroundings are bright, or if the LDR has a light shone on it, the LCD should display "Bright."
- If the surroundings are of average brightness, the LCD should display "A Little Bright."
- If the surroundings are dark, or if the LDR is covered, the LCD should display "Dark."

Bluetooth

Bluetooth and Arduino

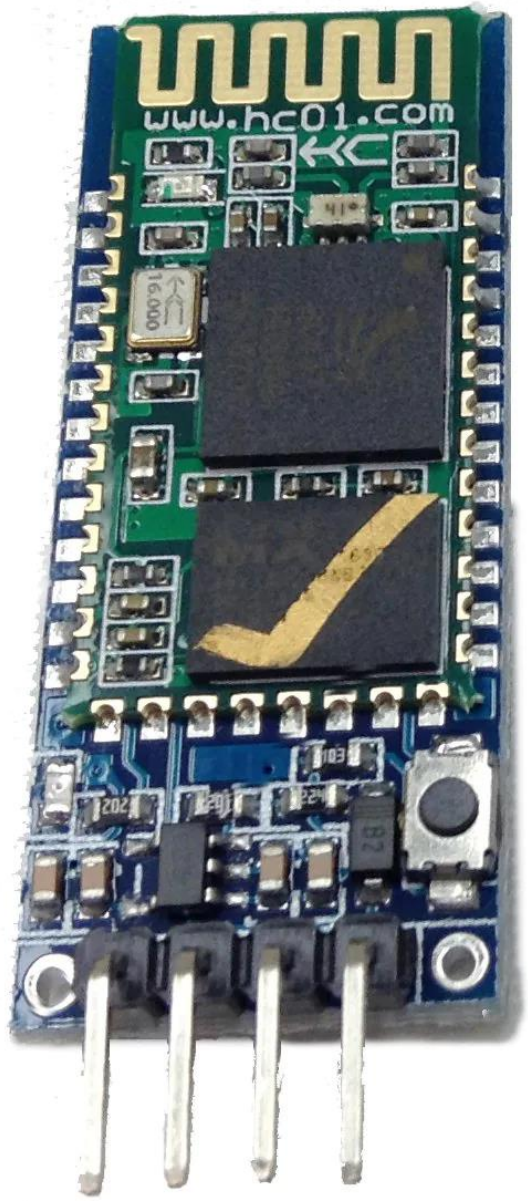


What is Bluetooth?

- Bluetooth is a wireless technology that uses ultra-high frequency (UHS) radio frequencies that are used for communication between a master (leader) and a slave (follower) device.
- Master (leader, or central) devices search for other devices and initiate connection.
- Slave (follower, or peripheral) devices wait for connections.
- For example, when using a phone with Bluetooth-enabled headphones, the phone is the master (leader) device, and the headphones are the slave (follower) device.
- Note: The master/slave naming convention found in many technological settings is a controversial topic because they could be considered offensive. Terms like leader/follower have been implemented on other products as replacements. Be an ethical programmer 👍
- Read more here: https://www.wired.com/story/tech-confronts-use-labels-master-slave/

HC-05 Module

- The HC-05 module can be used for communication between the Arduino Uno and an android mobile device.
- The HC-05 will take the role of a follower, while your mobile device will take the role of leader. This means that the mobile device will tell the HC-05 module what to do.
- The module's name, passkey, and role can be changed in the Arduino IDE.
- It is not compatible with Apple devices.



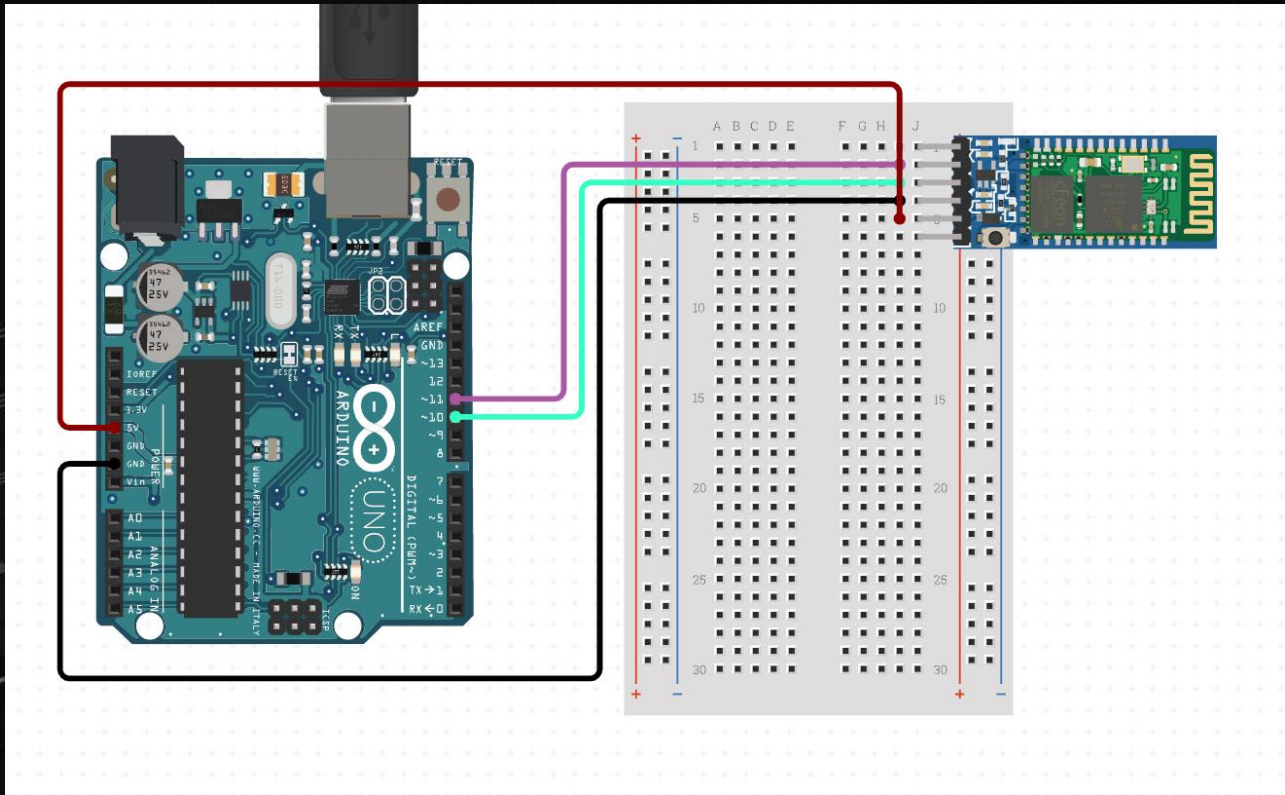
Arduino → HC-05

VCC = 5V pin

GND = GND pin

TXD = pin 10

RXD = pin 11



When you power the Arduino, you will see a red flashing light on the bottom left corner of the front side of the HC-05 module. This indicates that the HC-05 is discoverable as a Bluetooth device and is waiting for connections.

AT Command Mode

- To enter AT command mode, disconnect power from the Arduino, then hold down the button on the bottom right of the HC-05 module as you supply power to the Arduino again.
- If the red LED on the HC-05 is blinking at 2 second intervals, the module is in AT command mode.
- This mode can be used to change the module's name, password, and role (leader, follower).
- Use this code to access the serial monitor to use AT commands.
- It checks if information from the Bluetooth module is available, and if it is, the info is printed onto the serial monitor.
- If input information from the user is available, it will be sent to the Bluetooth module.
- This allows for communication between the HC-05 and the serial monitor command line.

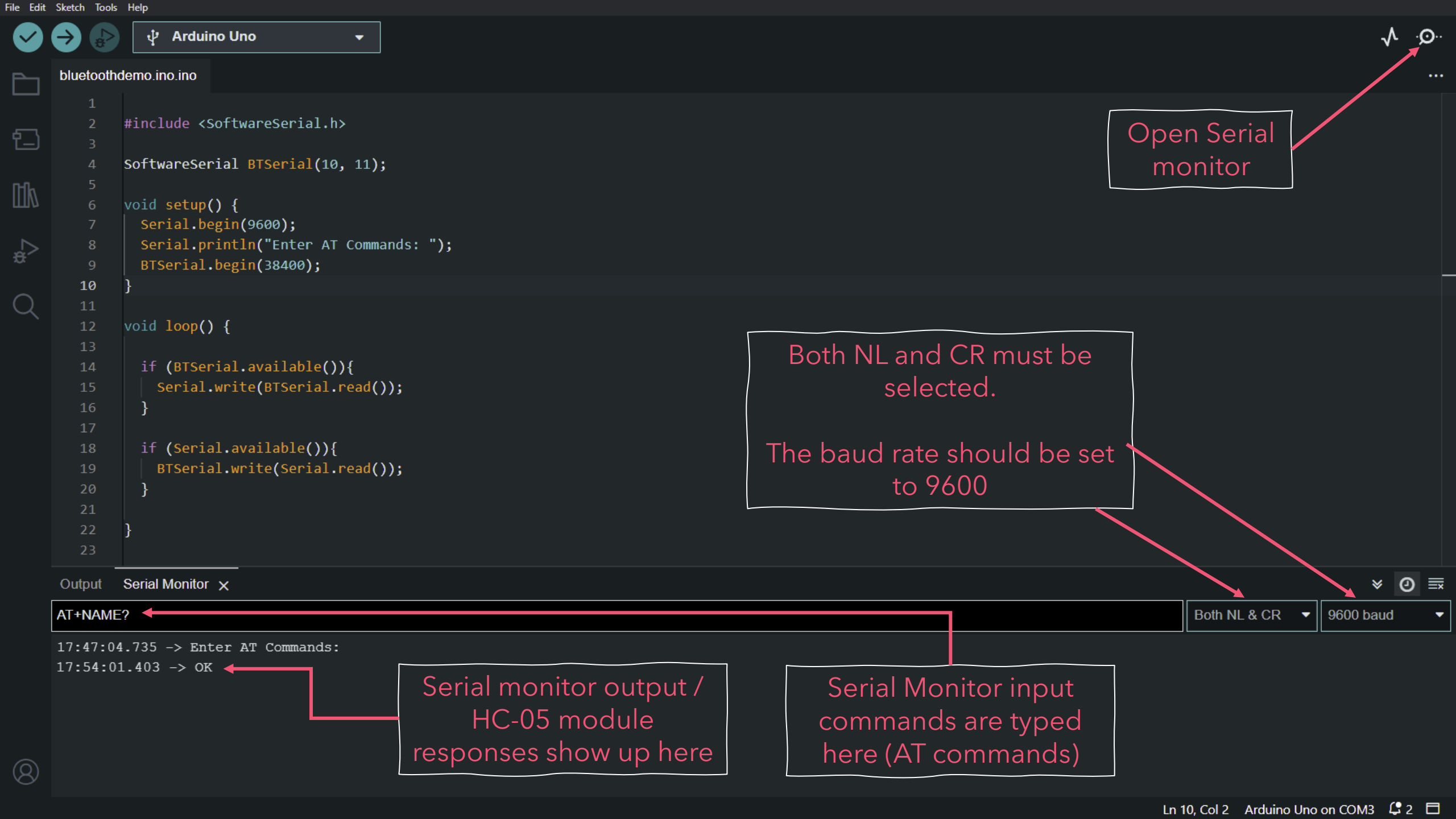
```
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(10, 11);

void setup() {
  Serial.begin(9600);
  Serial.println("Enter AT Commands: ");
  BTSerial.begin(38400);
}

void loop() {

  if (BTSerial.available()){
    Serial.write(BTSerial.read());
  }

  if (Serial.available()){
    BTSerial.write(Serial.read());
  }

}
```

How to Use AT Command Mode

AT command mode allows a user alter the settings of the device. This happens through certain commands that are used in the serial monitor.

“

These commands use the keyword AT, then a plus sign to define a function.

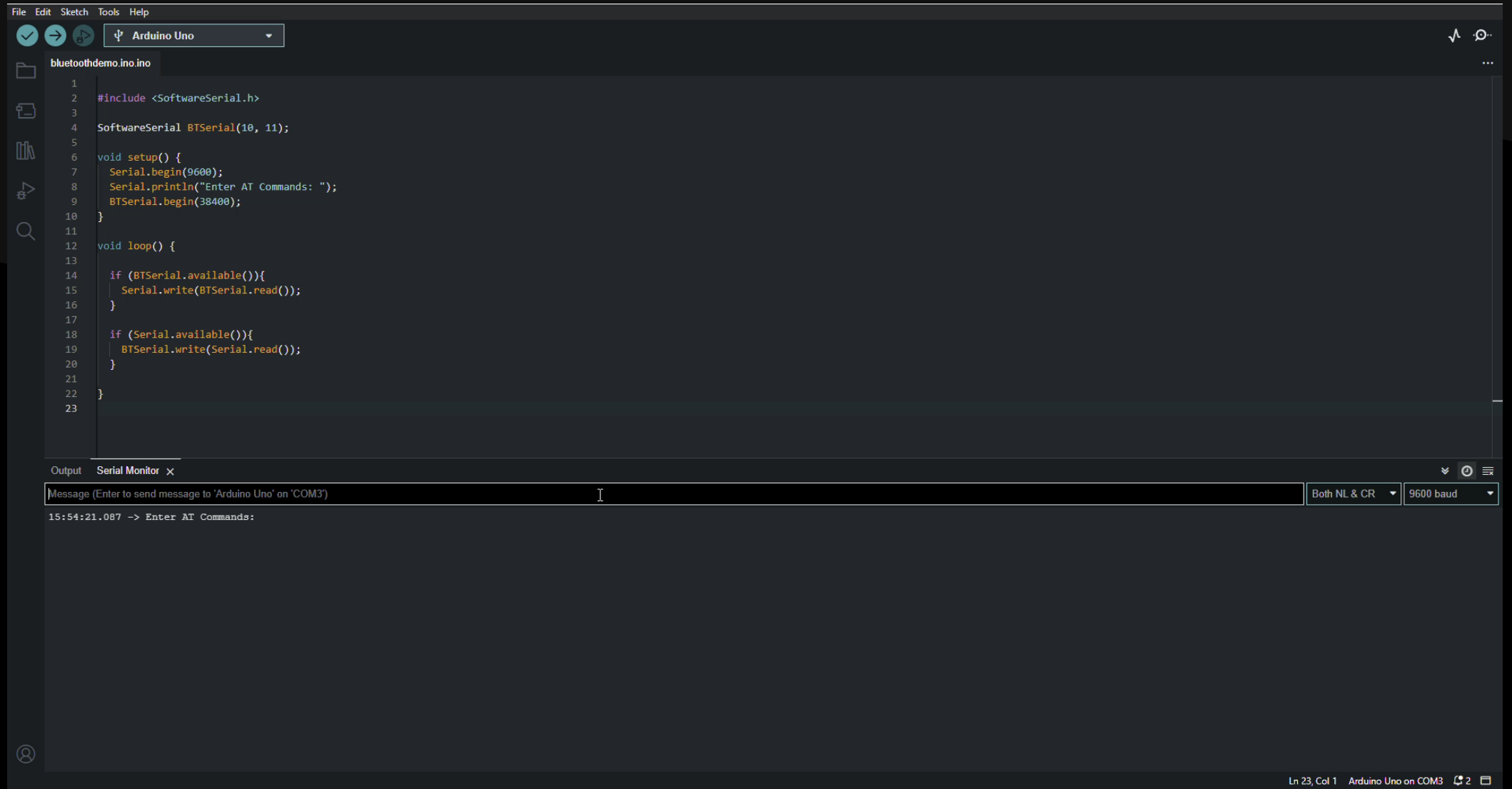

File Edit Sketch Tools Help
Arduino Uno
bluetoothdemo.ino.ino
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(10, 11);
void setup() {
Serial.begin(9600);
Serial.println("Enter AT Commands: ");
BTSerial.begin(38400);
}
void loop() {
if (BTSerial.available()){
Serial.write(BTSerial.read());
}
if (Serial.available()){
BTSerial.write(Serial.read());
}
}
Open Serial monitor
Both NL and CR must be selected.
The baud rate should be set to 9600
Output
Serial Monitor
AT+NAME?
Both NL & CR
9600 baud
17:47:04.735 -> Enter AT Commands:
17:54:01.403 -> OK
Serial monitor output / HC-05 module responses show up here
Serial Monitor input commands are typed here (AT commands)
Ln 10, Col 2
Arduino Uno on COM3

The Commands

Command	What it does
AT+NAME?	Returns HC-05 module's name
AT+NAME=enter name	Changes name to whatever is after the equals sign
AT+PSWD?	Returns HC-05 module's password
AT+PSWD=0000	Changes password to 0000, or whatever is after the equal's sign
AT+ROLE?	Returns the role of the HC=05 module. 0 means follower, 1 means leader, 2 means follower loop role
AT+ROLE=1	Changes the role to 1, or follower. (Should be 0 for usage)

If the command is properly executed, ***OK*** will be displayed on the serial monitor.

AT Commands - Example



MIT App Inventor

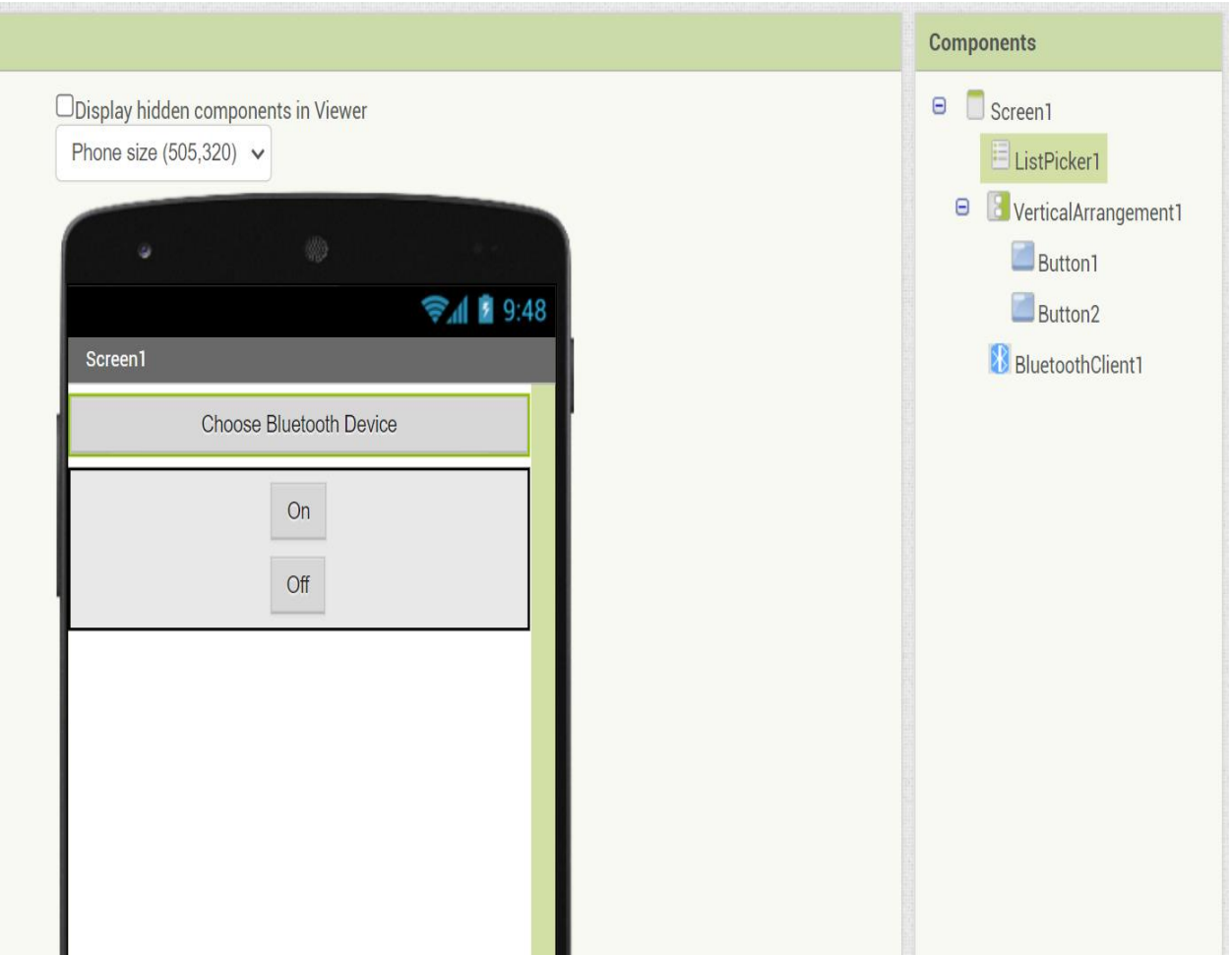
MIT Appinventor is an application that can be used to create mobile applications.

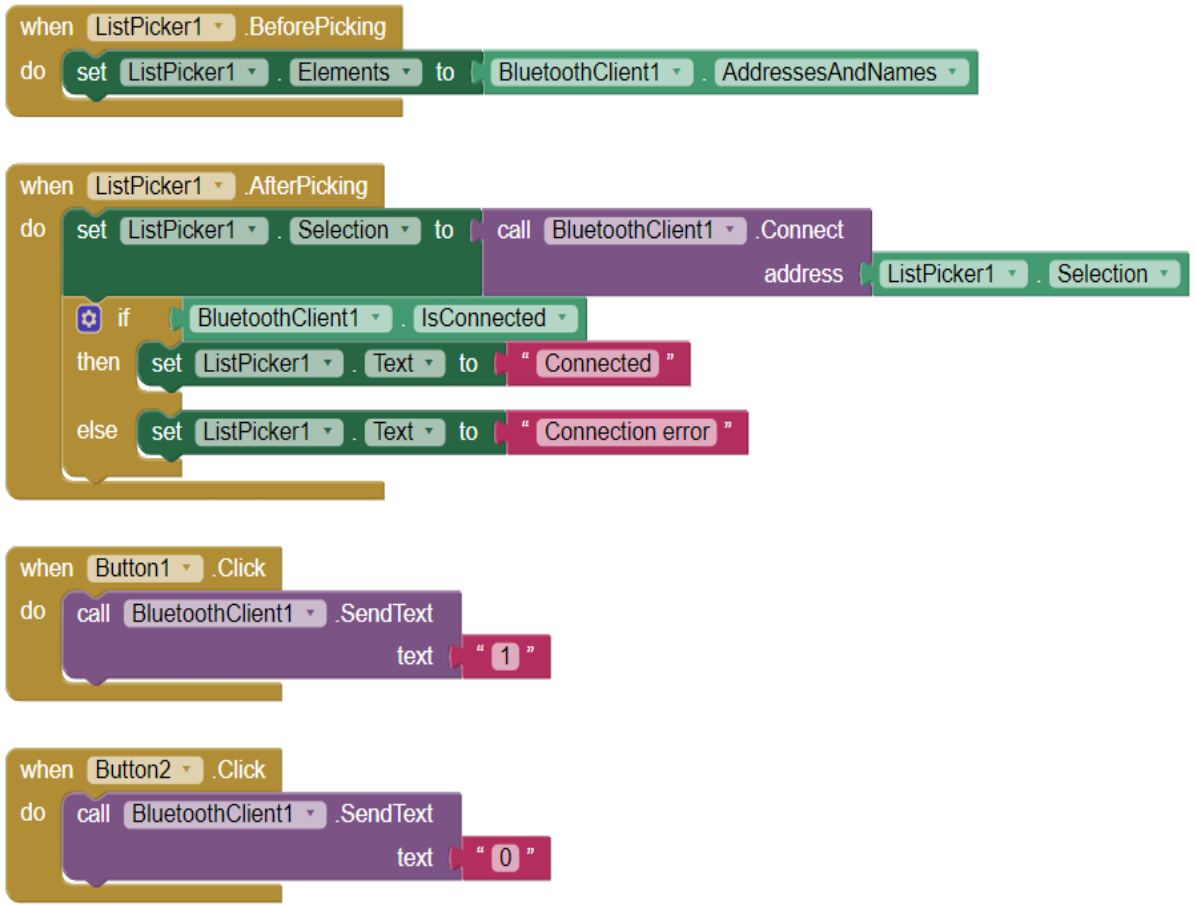
Use MIT App Inventor for Bluetooth Communication

- https://drive.google.com/file/d/1O4FCcTkG1zKdCsYDUSv45Ha5edHzi1OW/view?usp=sharing
- https://drive.google.com/file/d/1VVH9OZae49PYFzL54RHkMReUaaXMl8Vk/view?usp=sharing
- These videos detail the process of using MIT Appinventor to connect your device to the HC-05 module.
- The text sent to the Bluetooth module can be processed and used in the Arduino IDE to control the Arduino peripherals.
- There needs to be code in both MIT App Inventor and the Arduino IDE.
- Your device will tell the HC-05 module what to do, then the module will tell the Arduino what to do
- Code is on the next few slides.

MIT App Inventor – Designer and Blocks

NOTE: If the Bluetooth selection screen on the app is blank, ensure that the app has access to view Bluetooth devices in the app permissions.





Example: Controlling The NeoPixel ring with the HC-05 module and MIT App Inventor

```
#include <SoftwareSerial.h> // Includes the SoftwareSerial class for the code to work
#include <Adafruit_NeoPixel.h> // Includes code for Adafruit_NeoPixel class

Adafruit_NeoPixel my_ring = Adafruit_NeoPixel(16, 2, NEO_GRB + NEO_KHZ800);
// Above: Constructor for NeoPixel ring. The ring is in pin 2.


SoftwareSerial BTSerial(10, 11);  // Pin 10 is the TX pin and pin 11 is the RX pin on the HC-05 module

char incoming = 0; // Stores the value sent by MIT App Inventor

void setup() {
  Serial.begin(9600); // Sets the baud rate for both the Serial and Bluetooth device
  BTSerial.begin(9600);

  my_ring.begin(); // initializes the ring
  my_ring.clear(); // clear ring (blank)
  my_ring.show(); // send instructions to ring
}

  ------------------------------------------------------------------------------------------

void loop() {

  if (BTSerial.available()){  // If the Bluetooth module is available
```

```
void loop() {

  if (BTSerial.available()){  // If the Bluetooth module is available

    incoming = BTSerial.read(); // incoming is the value sent to the Bluetooth module by the app

    Serial.println(incoming); // Outputs the incoming value to the Serial monitor (not necessary)

    if (incoming == '1'){ // If incoming is 1,

      for(int i = 0; i < 16; i++){
        my_ring.setPixelColor(i,255,0,0); // sets pixel to red
        delay(50);  // 50 millisecond delay
        my_ring.show(); // updates ring, displays the light
      }

    }

    else if (incoming == '0'){  // If incoming is 0,

      for(int i = 0; i < 16; i++){
        my_ring.setPixelColor(i,0,0,0); // sets pixel to blank
        delay(50);  // 50 millisecond delay
        my_ring.show(); // updates ring, displays the light
      }
    }
  }
}
```

Congratulations!

You’re an Arduino expert!