The Magical Cube: an Introduction to Arduino

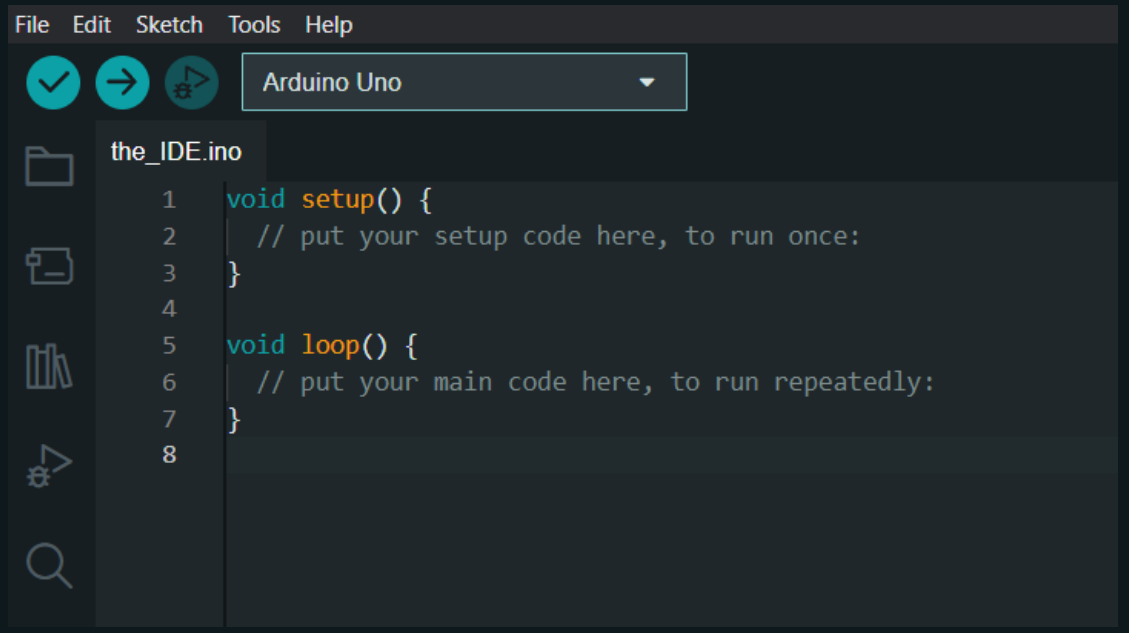
Arun Chauhan, Ishaan Takrani

The Arduino IDE

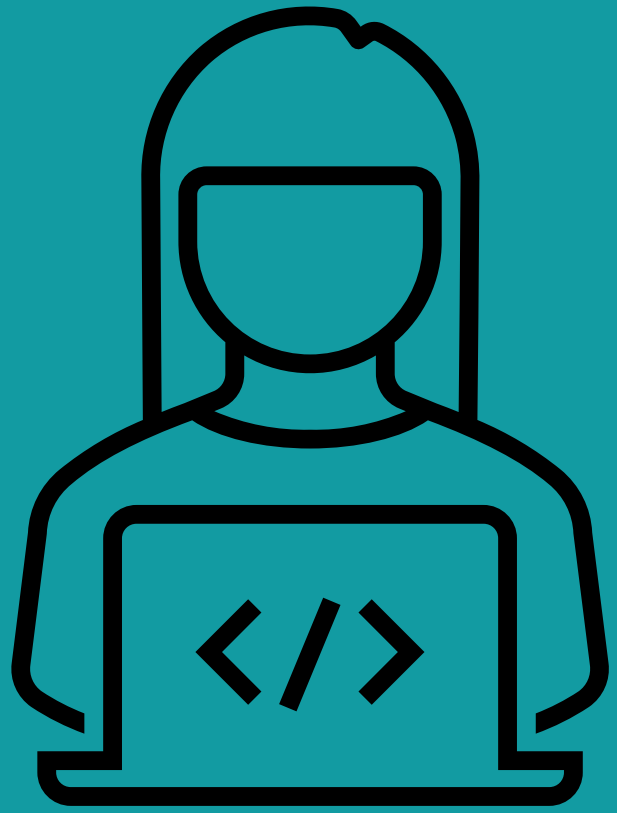
Introduction to the Arduino IDE

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

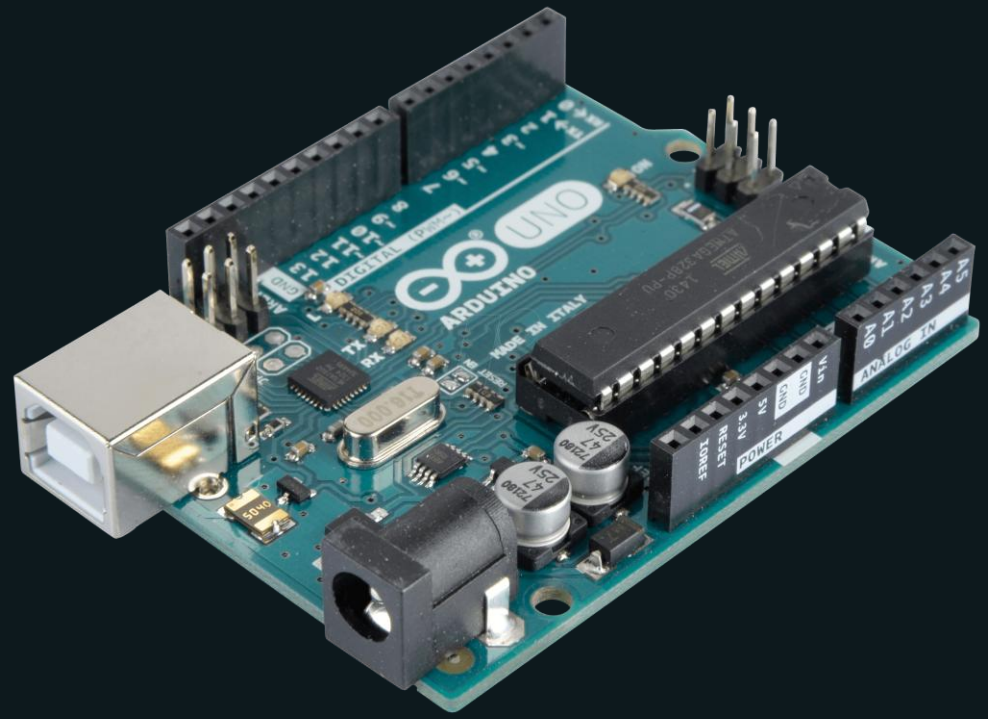
Compiling is the process of converting the code into an executable file that the computer can understand.

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.

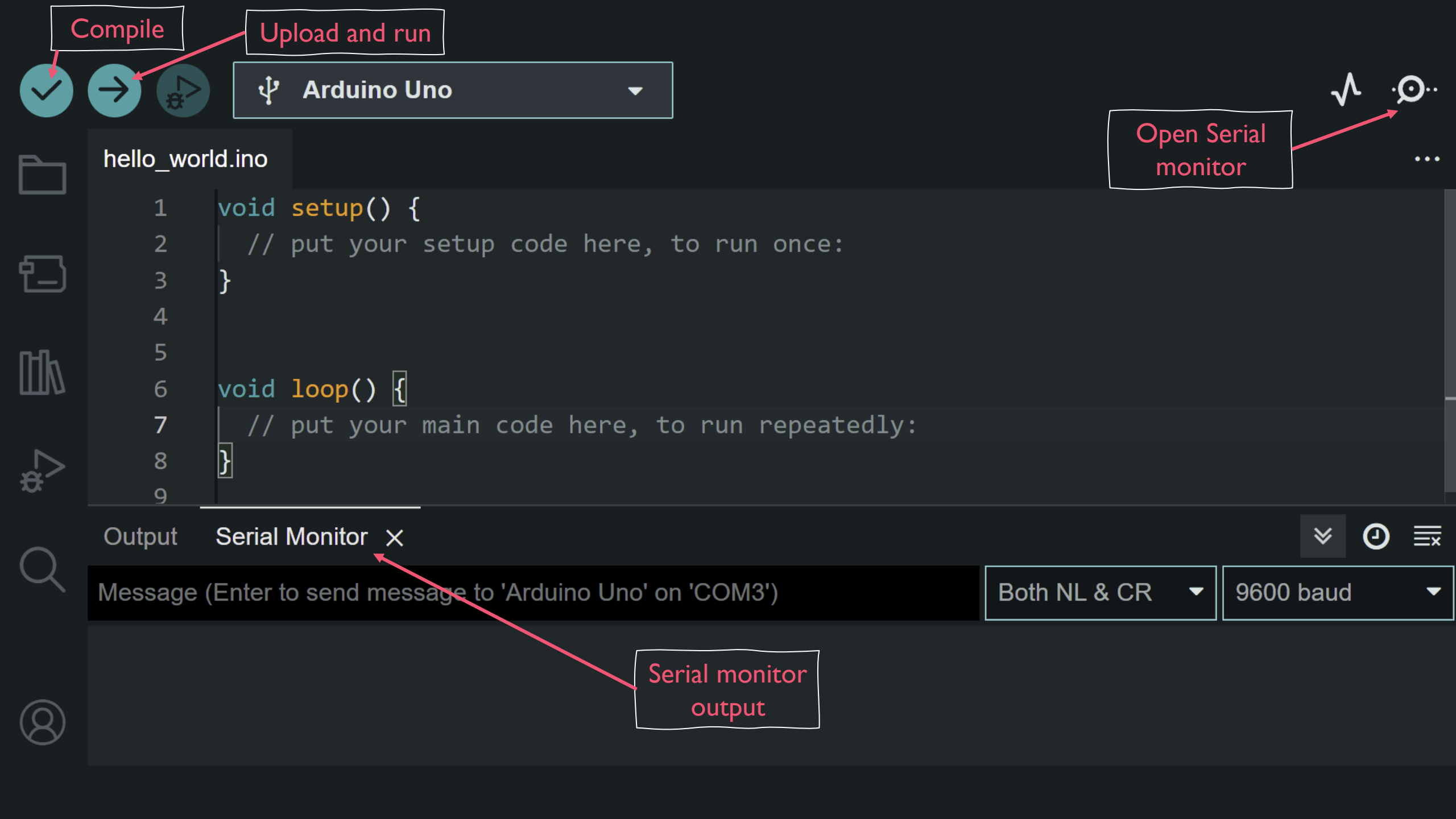
They are used to make software for a vast number of devices.

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 executed, whatever code is in void loop will run ***once***, then whatever is in void setup will run ***repeatedly***.

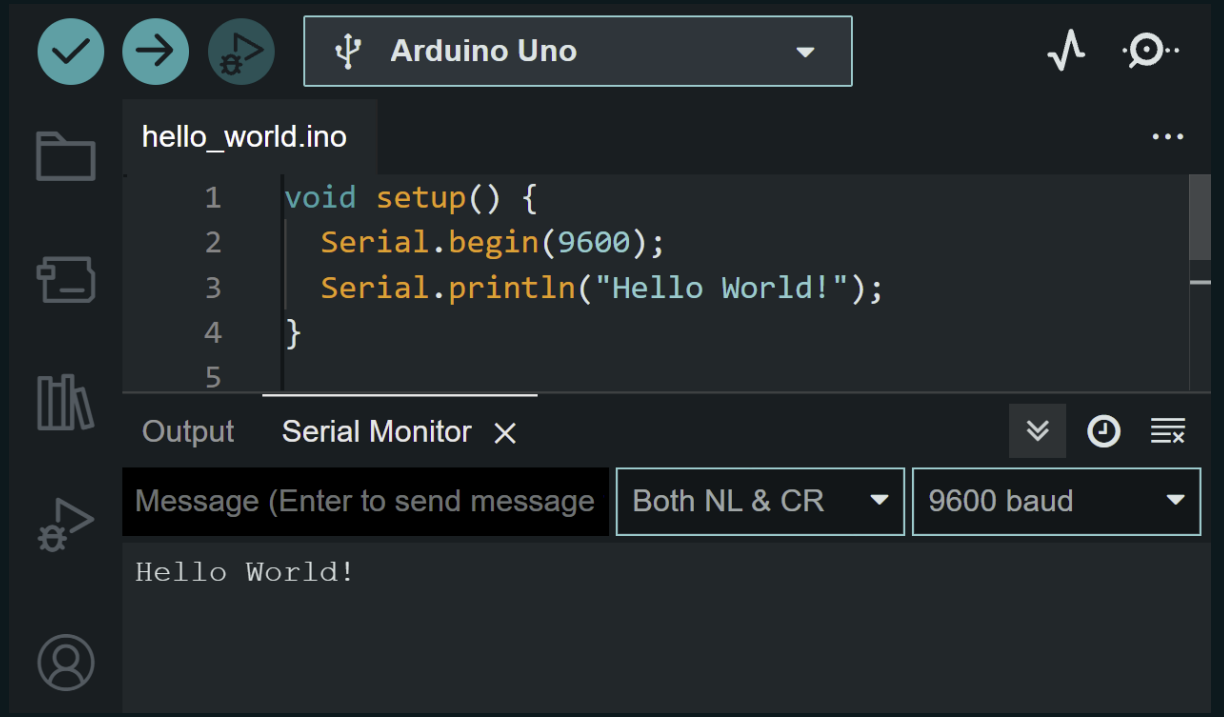

Compile
Upload and run
Arduino Uno
Open Serial monitor
hello_world.ino
1 void setup() {
2 // put your setup code here, to run once:
3 }
4
5
6 void loop() {
7 // put your main code here, to run repeatedly:
8 }
9
Output
Serial Monitor
Message (Enter to send message to 'Arduino Uno' on 'COM3')
Both NL &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

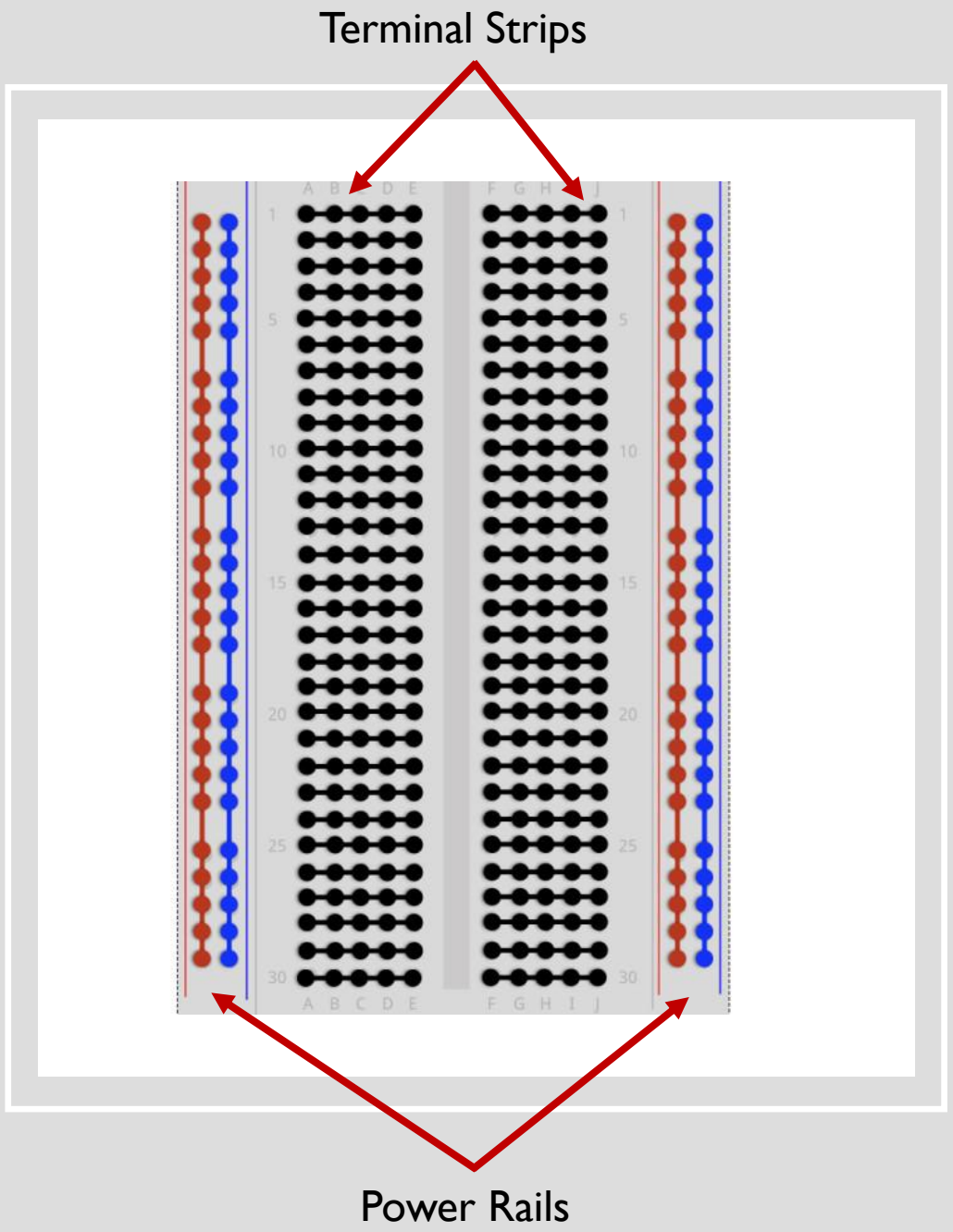
And how they work together to do what you want them to

BREADBOARD

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

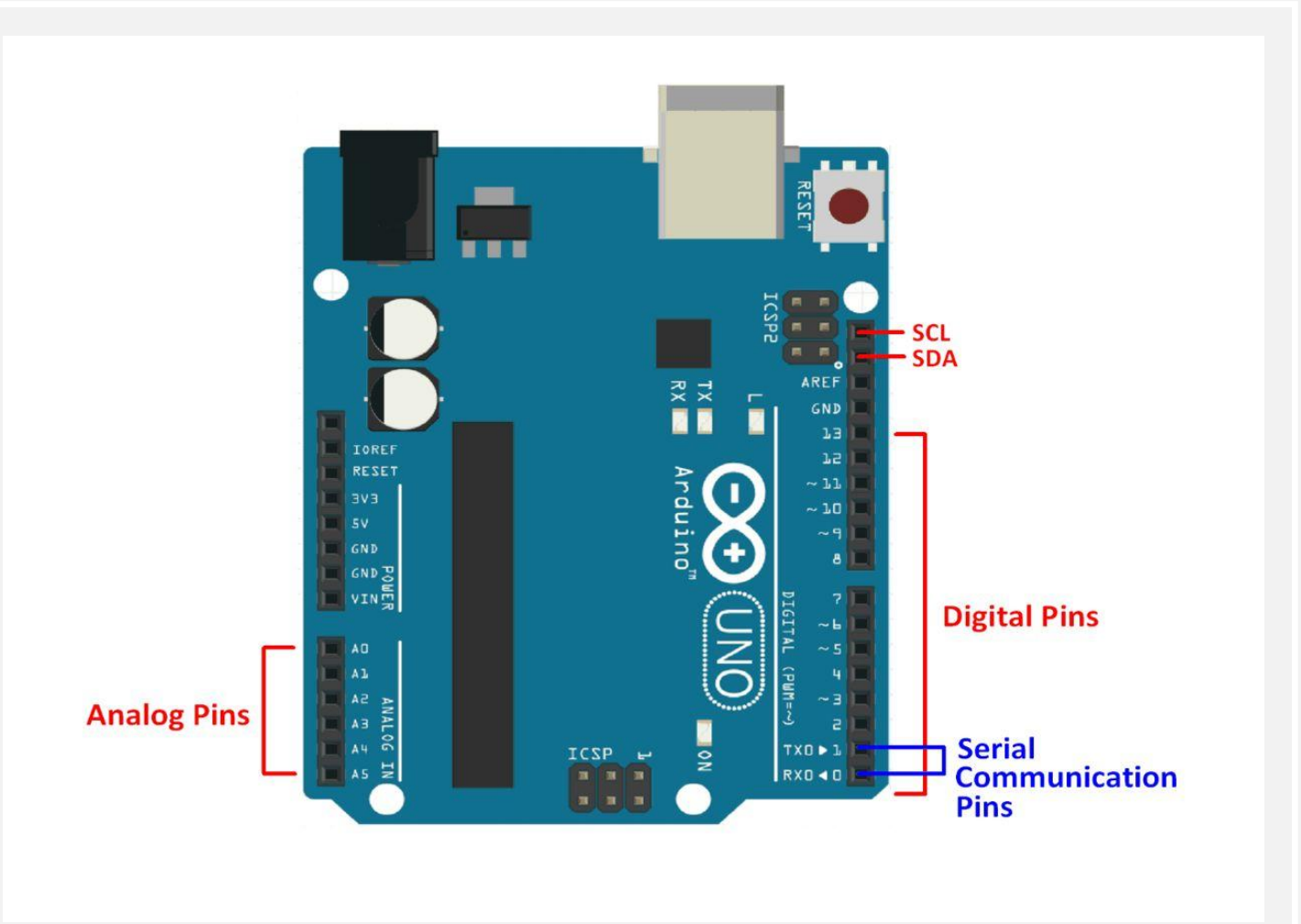
Electrons move through the rails vertically, and through the terminal strips horizontally, as shown by the connections in the image.

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 to and from the Arduino.



SAFETY

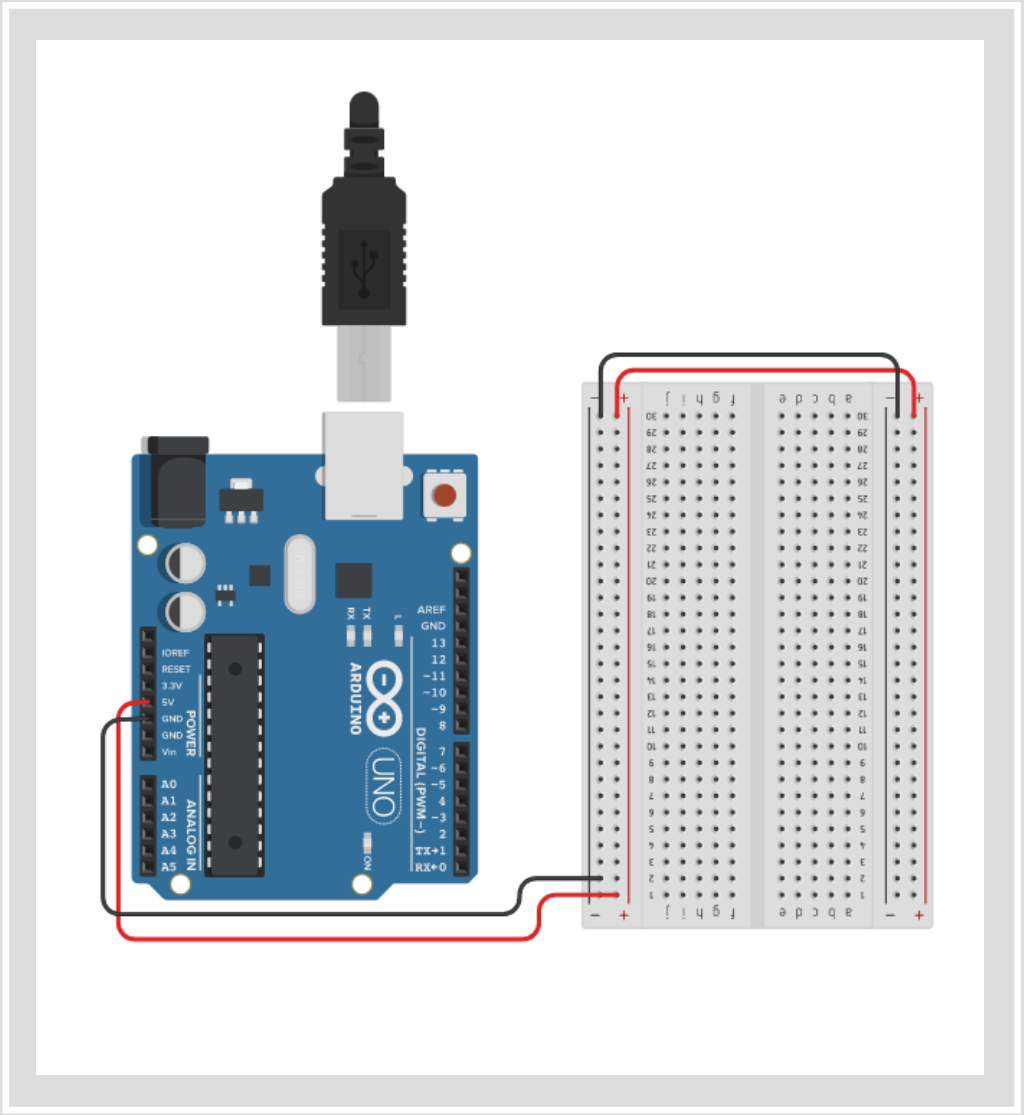
Important: When working, never touch or connect/disconnect wires or tweak parts while the board is connected to power. This could result in serious injury. DO NOT touch circuit components while power is connected.

NEVER – work on a LIVE circuit.

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

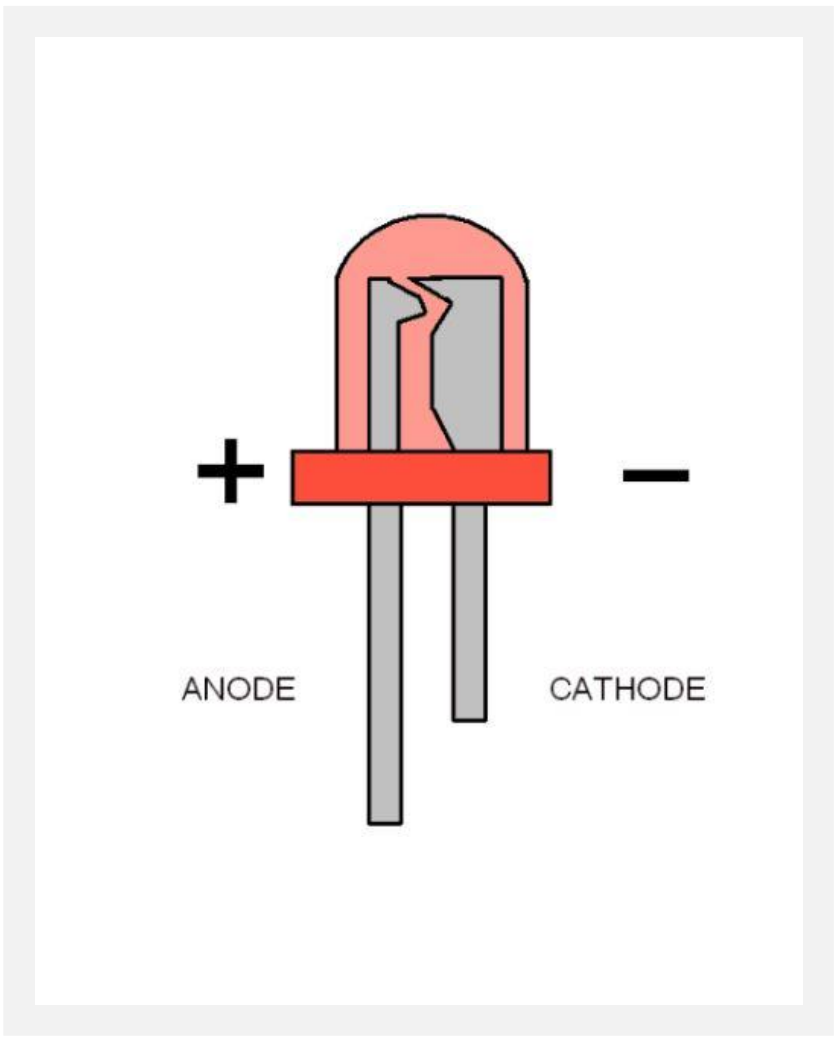
PROVIDE POWER TO THE BREADBOARD

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.



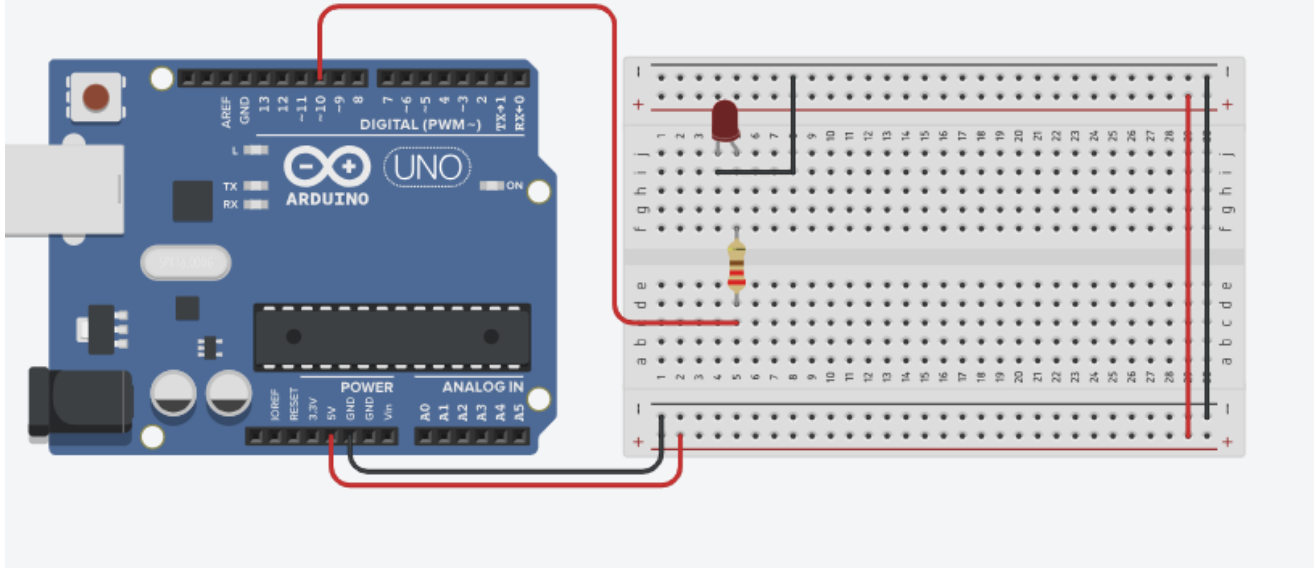
WHAT IS AN LED?

- **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.



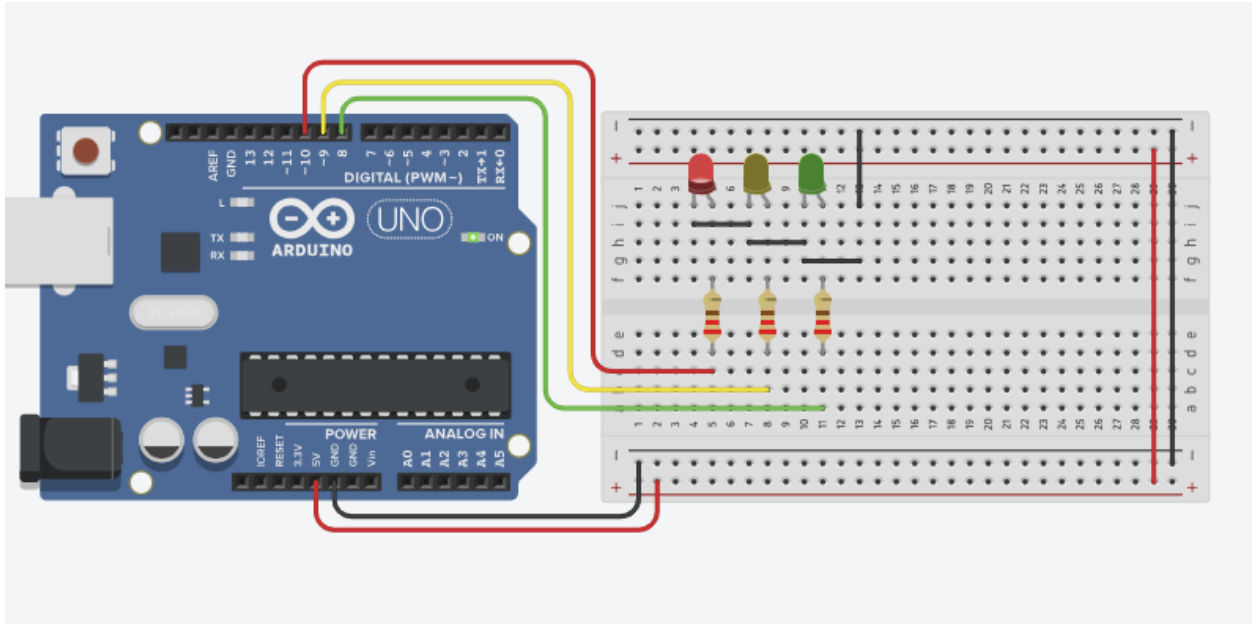
LED → BREADBOARD

- 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)

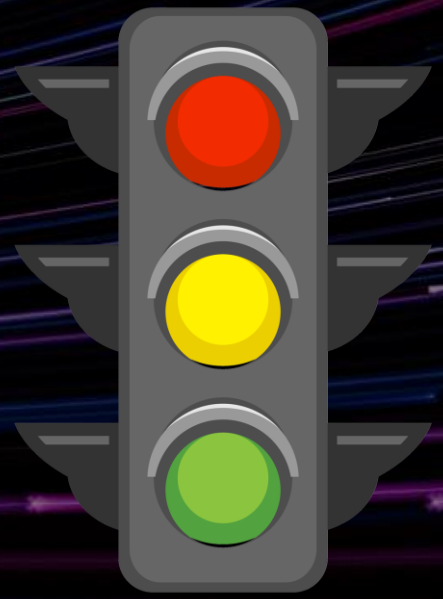
This code turns the green light (pin 8) on for one second, then off for one second. Because this code is in void loop(), this is repeated indefinitely.

```
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

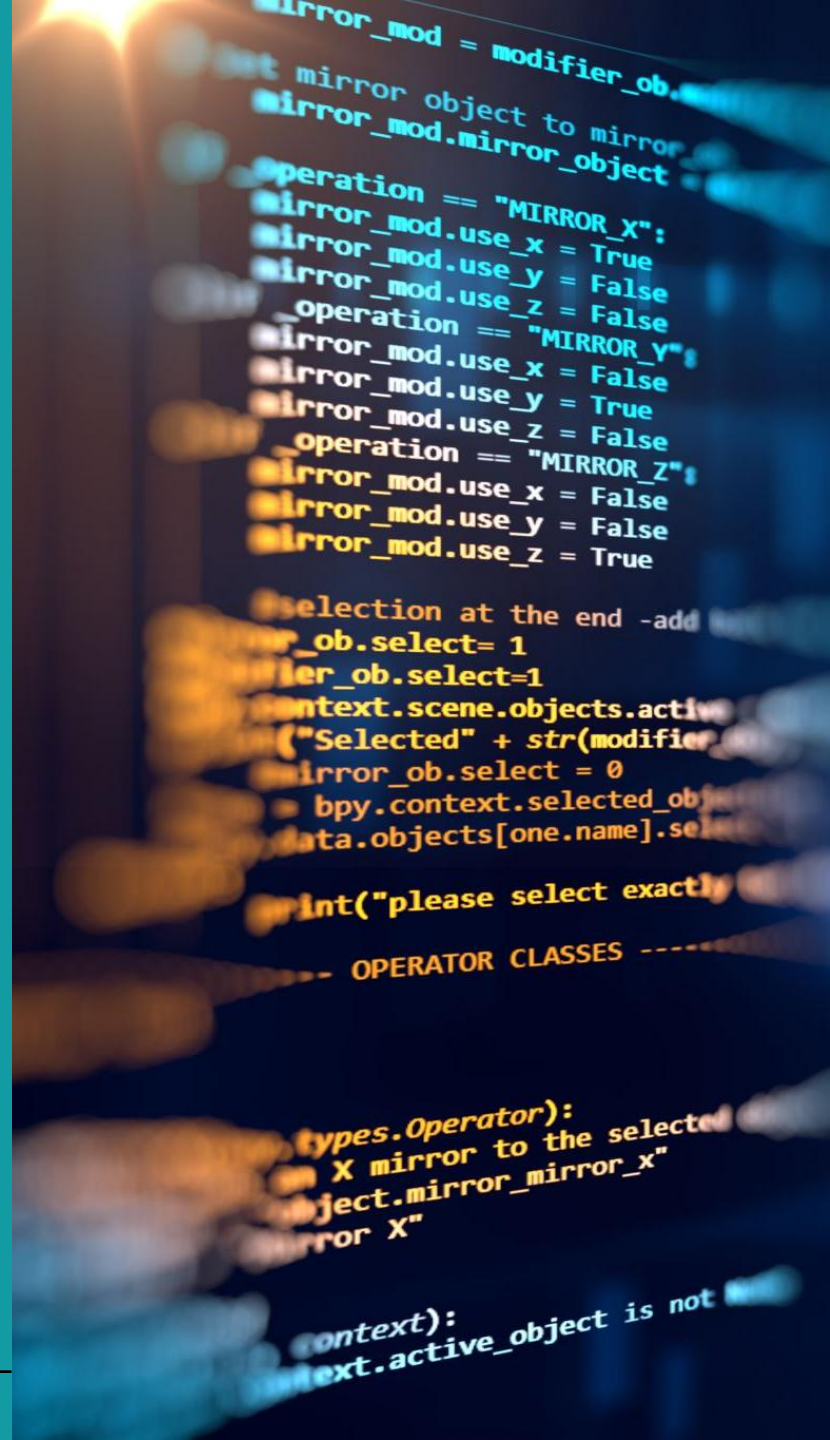
- Using variables for the LEDs, Connect another three LEDs to the Arduino and simulate a two-way traffic light system, much like one in real life.
- The red and green lights should stay on for 5 second intervals, while the yellow light should stay on for 2 seconds.

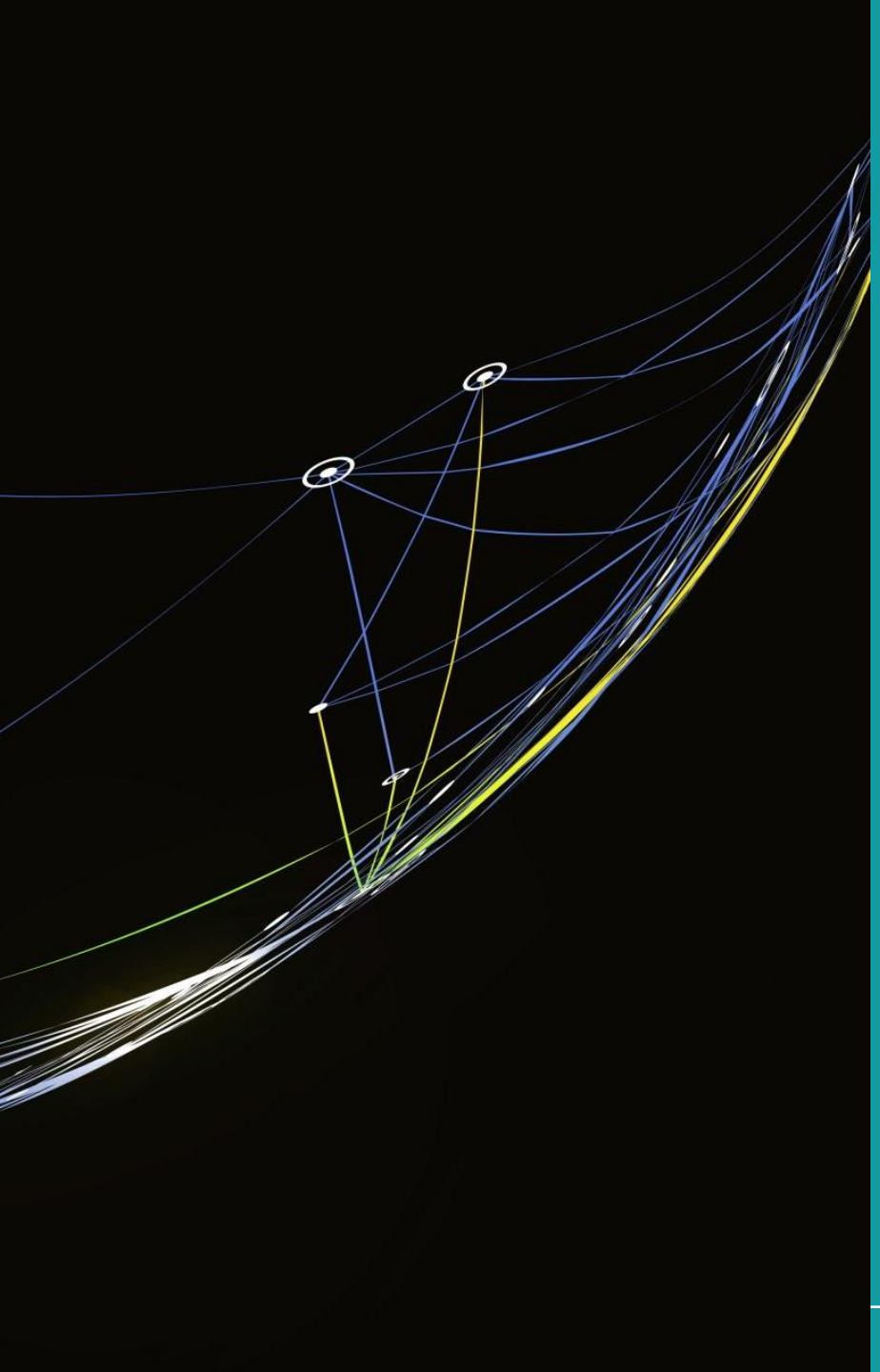
Programming : Functions

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 each 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

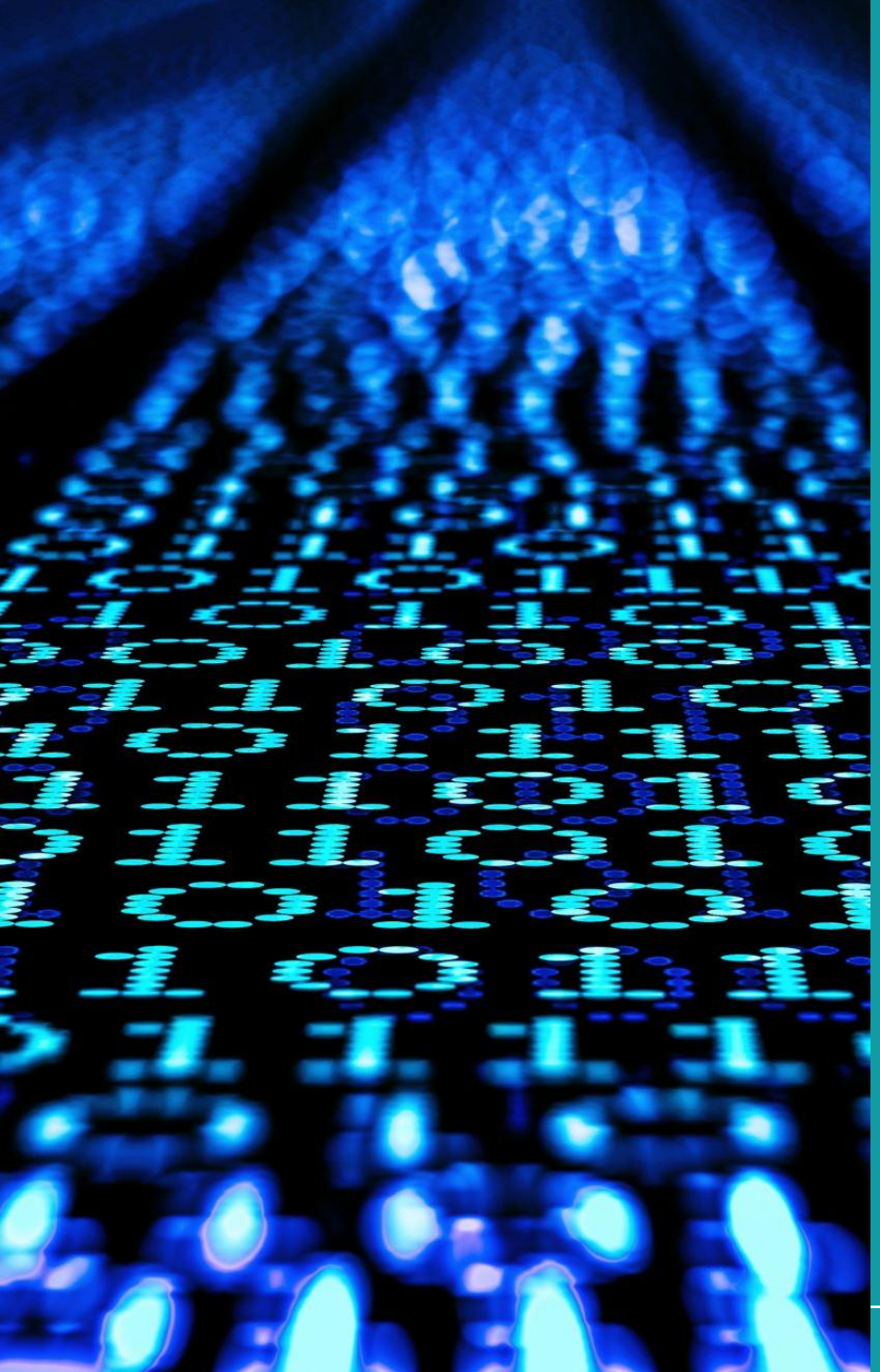
```
void loop() {
  rapidFlash(green);
  rapidFlash(yellow);
  rapidFlash(red);
}
```

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<201;i++){
    Serial.println(i);
}
```

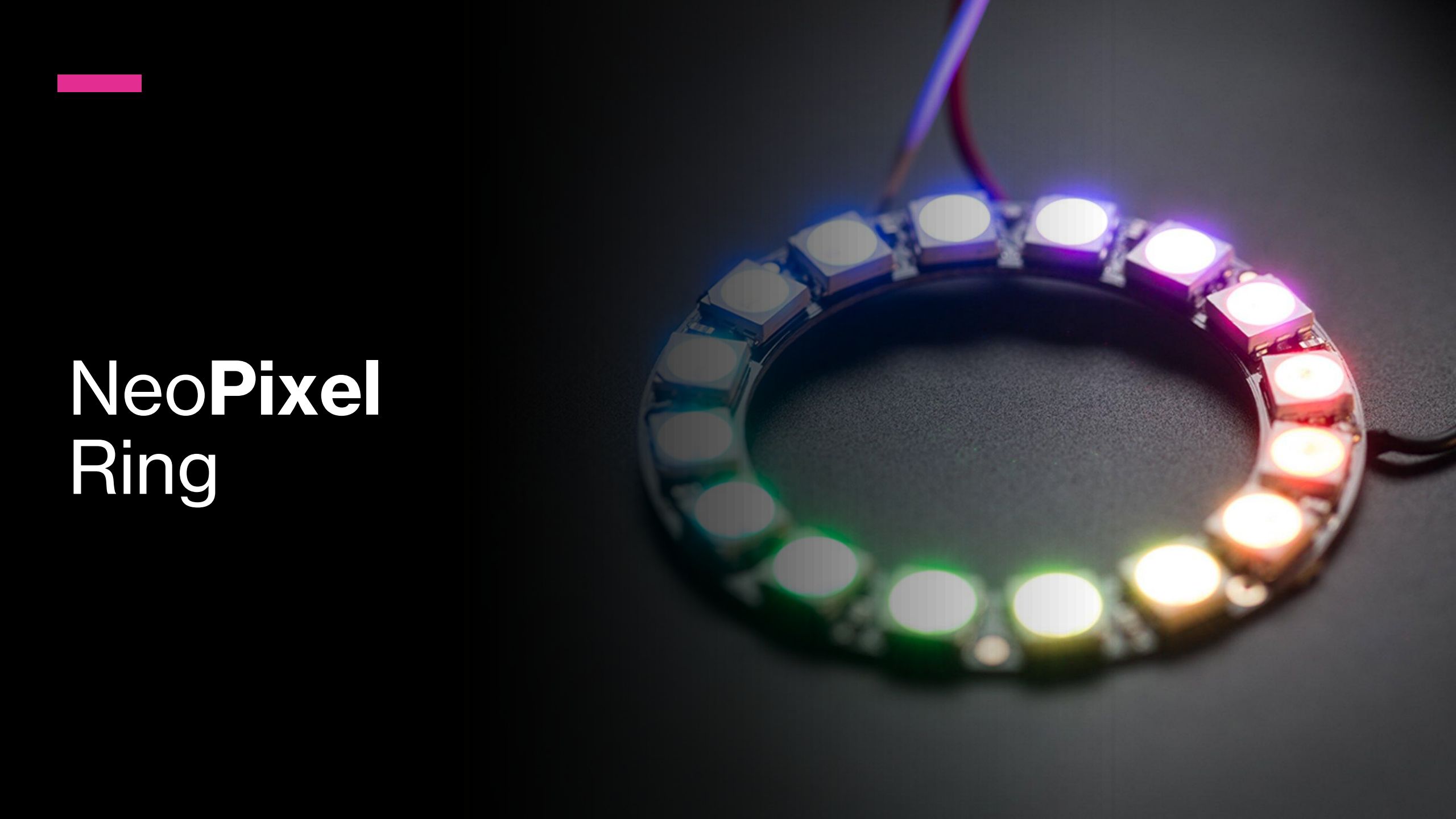
What if we wanted to print every number from 10 to 15?

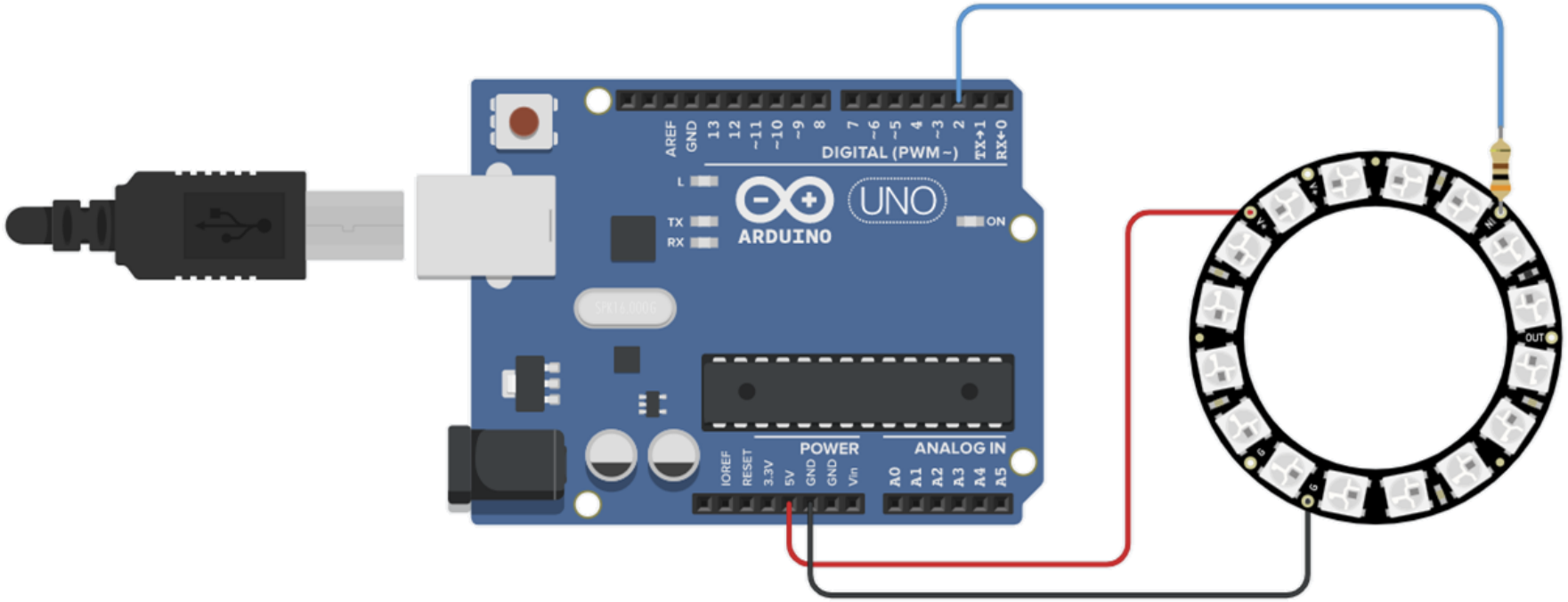
```
for(int i=10;i<16;i++){
    Serial.println(i);
}
```

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

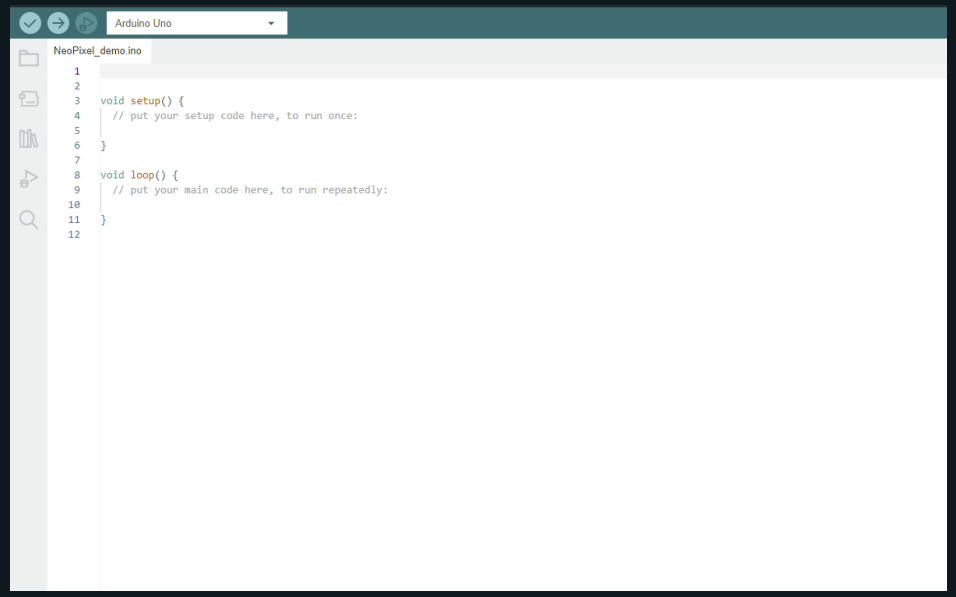
- On the ring, there are three connections to be made.
- V+, G, and IN (soldered).
- V+ connects to the 5V pin on the Arduino, G connects to the GND pin, and IN is an input pin that connects to a digital pin on the Arduino.
- 30Ω resistor in series with the ground connection (some NeoPixel rings already 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>
- A file ending in `.h` is recognized by the Arduino IDE as a *header* file, which contains C language definitions, structures, and classes.

Programming the NeoPixel ing

- 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.

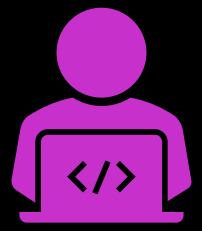
For example, a class can be “ball,” and it can have the functions of bouncing, and characteristics such as size.

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.

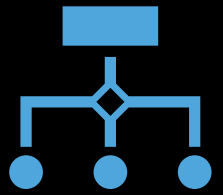
Another member of the “ball” class can be a ***baseball.***

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).

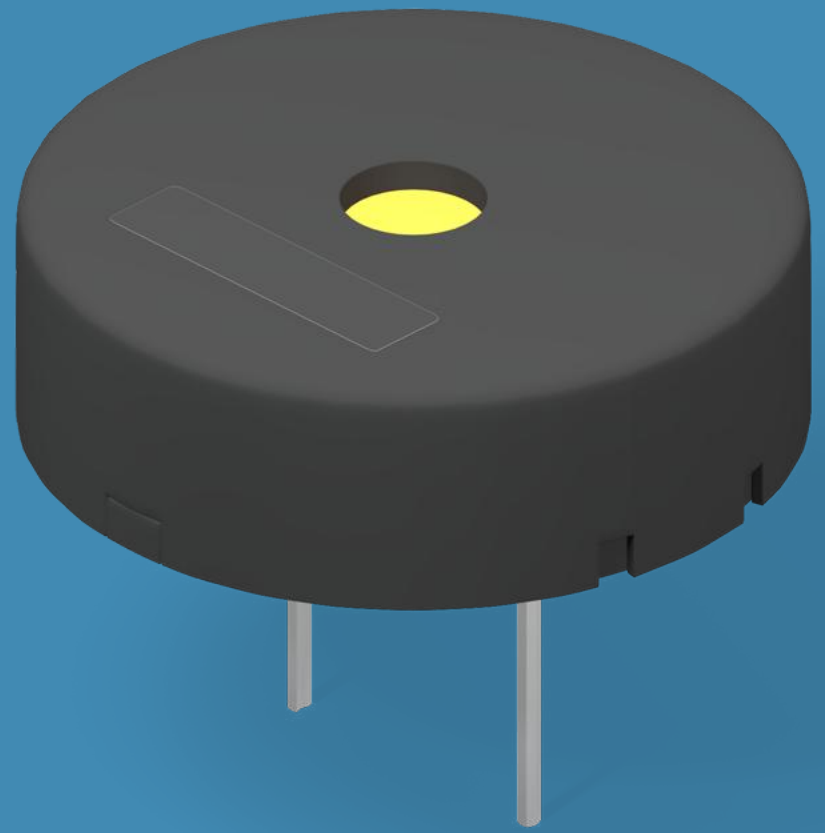
PIEZO BUZZER

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.

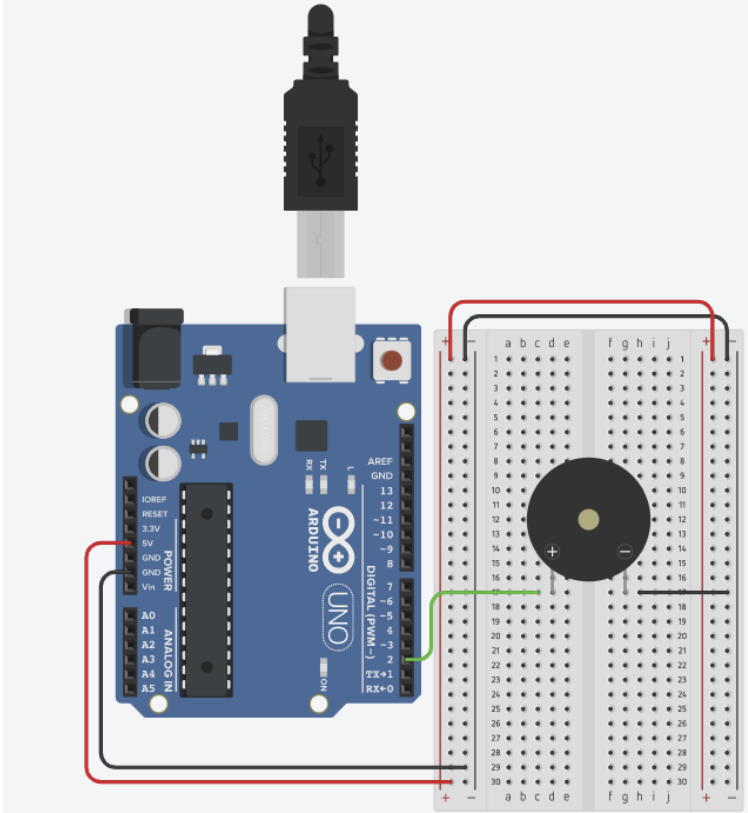
Active buzzers only need a DC power supply to produce sound, while passive buzzers need an AC power supply.

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

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.

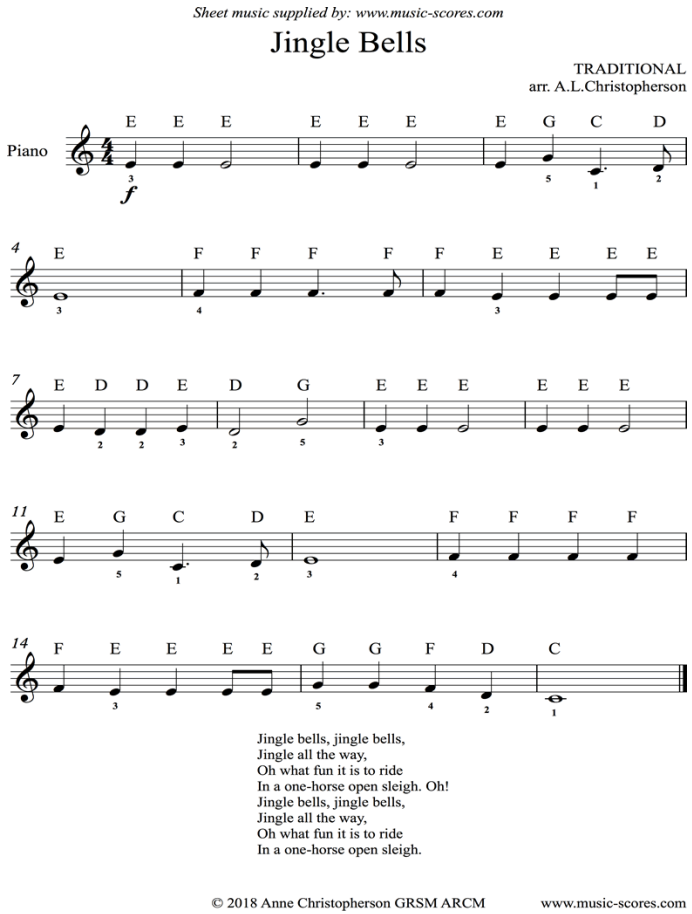
```
void playNote(int note, int
delay_time){

  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.