



Microcontroller Handbook

Except as permitted under the Canada Copyright Act of 1997, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without prior written permission of the publisher.

Designations used by companies to distinguish their products are often claimed as Trademarks. All brand names and product names used in this book are trade names, service marks, trademarks, or registered trademarks of their respective owners. The publisher and author or designing team are not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate information regarding the subject matter covered.

Arduino is a Trademark of the Arduino Team.

Information contained in this handbook has been obtained from resources believed to be reliable. However, Exceed Robotics cannot guarantee the accuracy or completeness of any information published and neither Exceed Robotics nor its contributors to this handbook shall be responsible for any errors, omissions, or damages arising out of use of this information.

Acknowledgements

Thank you to the following people who spent countless hours and resources to help put this book together:

- Rob Hattam
- Micheal Em
- Kiran Patel
- Ethan Goldentuler
- Brandon Da Costa
- Kiran Patel

Table of Contents

CHAPTER 1: GETTING STARTED WITH ARDUINO	8
1.1 What is an Arduino?	9
1.2 Hardware Pinout	10
1.3 TinkerCad Circuits	12
1.3.1 Creating a TinkerCad Account	12
1.3.2 Create a New Design	13
1.3.3 Learn How to use TinkerCad Circuits	13
1.4 Download Software	15
1.5 Example Code	16
1.6 Exploring the Arduino IDE	17
1.7 Libraries and Troubleshooting	19
1.7.1 Testing the Device	21
1.8 Serial Monitor	22
1.9 What is a Digital Signal?	22
1.10 What is an Analog Signal?	23
1.11 Digital Vs Analog Signal	24
1.12 What is Pulse Width Modulation (PMW)?	25
CHAPTER 2: INTRODUCTION TO BASIC ELECTRIC CIRCUITS	26
2.1 What is Electricity?	27
2.1.1 Electricity in a Circuit	27
2.2 How a Circuit Works	28
2.2.1 Circuit Diagrams	28
2.2.2 Circuit Diagram Symbols	29
2.3.1 Using The Multimeter	31
2.3.2 Using The Oscilloscope	33
2.4 Ohms Law	33
2.5 Kirchhoff's Voltage and Current Law	34
Homework Questions:	35
CHAPTER 3: KEY CODING FUNDAMENTALS	37
3.1 Data Types	38

3.1.1 Boolean	38
3.1.2 Char	38
3.1.3 Integers	39
3.1.4 Long	39
3.1.5 Short	39
3.1.6 Float / Double	39
3.1.7 Array	40
3.1.8 Variable Creation	41
3.2 Arduino Code and the IDE	42
3.2.1 Introduction to Code	43
3.3 Arduino Math/Arithmetic	44
3.3.1 Arithmetic Operators	44
3.3.2 Comparison Operators	46
3.3.3 Boolean Operators	49
3.4 Making Decisions in Code	50
3.4.1 If Statements	50
3.4.2 Else Statements	51
3.5 Using While Loops	53
3.6 Using For Loops	56
3.7 Nested Loops	58
3.8 Defining Functions	59
3.9 Mapping Function	62
CHAPTER 4: INTRODUCTION TO ARDUINO BASICS	63
4.1 Parts You Will Learn	64
4.1.1 Arduino UNO	64
4.1.2 Arduino MEGA	68
4.1.3 Arduino NANO	70
4.1.4 Breadboard	71
4.1.5 LED (Light-Emitting Diode)	73
4.1.6 RGB LED	76
4.1.7 Jumper Wires	79
4.1.8 Resistors	80
4.1.9 Pull-Down Resistors	81

4.1.10 Pull-Up Resistors	82
4.1.11 Transistors	82
4.1.12 Voltage Divider	84
4.1.13 Buzzers	85
4.2 Buzzer Library	87
4.3 Building Your First LED Circuit	88
4.4 Code Breaker Challenge	89
4.5 Homework Questions	90
CHAPTER 5: WORKING WITH SENSORS	100
5.1 Parts You Will Learn	101
5.1.1 Push Button	101
5.1.2 Reed Switch (Magnetic Sensor)	103
5.1.3 PIR Motion Sensor	106
5.1.4 Potentiometer	109
5.1.5 Photoresistor	110
5.1.6 Joystick	113
5.2 Complex Sensors	115
5.2.1 Ultrasonic Sensors	115
5.2.2 Temperature Sensor	117
5.2.3 Flex Sensor	119
5.2.4 Force Sensor	121
5.2.5 Keypad	123
5.2.6 Sound Sensor	125
5.2.7 Gyroscope and Accelerometer (MPU6050)	128
5.2.8 Sensor Shield	131
5.3 Sensor Library	132
5.4 Practice Questions	133
CHAPTER 6: WORKING WITH MOTORS	147
6.1 Parts You Will Learn	148
6.1.1 Servo Motor	148
6.1.2 DC Motor	150
6.1.2.1 Controlling DC Motors Method 1: Transistor	151
6.1.2.2 Controlling DC Motors method 2: L293D Chip	153

6.1.2.3 Controlling DC Motors method 3: L298N Driver	155
6.1.2.4 Controlling DC Motors Method 3: Arduino Motor Shield	160
6.1.3 Stepper Motor	163
6.2 Servo Library	166
6.3 Practice Examples	167
6.4 Homework Questions	169
CHAPTER 7: Display Modules	174
7.1 LCD Shield	175
7.2 LCD Library	176
7.3 Practice Question	177
7.4 Homework Question	178
7.5 I2C LCD	180
7.6 8X8 Dot Matrix	182
CHAPTER 8: WIRELESS COMMUNICATION	188
8.1 HM-10 Bluetooth Module	189
8.2 IR Remote and Receiver	191
8.3 RF Transceiver (NRF2L401):	193
CHAPTER 9: ADDITIONAL ARDUINO HARDWARE	203
9.1 Parts You Will Learn	204
9.1.1 DS 1307 RTC Module Pinout	204
9.1.2 RELAY	207
9.2 Practice Examples	208
9.3 Homework Questions	210
CHAPTER 10: HELPFUL BEGINNER TIPS	213
10.1 Common Issues and Troubleshooting	214
10.1.1 Problems in Circuits	214
10.2 Problems in Programming	215
10.2.1 Variable	215
10.1.3 Equals, Assignments and Syntax:	215
10.3 Common Errors	218
10.4 Good Programming Techniques	223
CHAPTER 11: ARDUINO RESEARCH SKILLLS	225
11.1 Engineering Design Process	226

11.1.1 WHAT ARE THE REQUIREMENTS?	227
11.1.2 HOW CAN IT BE DONE?	227
11.1.3 RESEARCH ON POSSIBLE SOLUTIONS	228
11.1.4 CHOOSING A PROMISING SOLUTION	230
11.1.5 BUILD A PROTOTYPE	230
11.1.6 EVALUATE THE PROTOTYPE	230
11.1.7 REDESIGN AS NEEDED	231
11.2 How Do You Start?	231
11.3 Types of Useful Resources	234
11.3.1 API's	235
11.3.2 Datasheets	236
11.3.3 Schematics	237
11.3.4 Code Sample	238
CHAPTER 12: ARDUINO CODING REFERENCE	239
12.1 Arduino Command Quick Reference	240
12.3 Structures Reference	243
CHAPTER 13: TOOLS, COMPONENTS, AND ADDITIONAL INFORMATION.....	246
13.1 Starter Kits	247
13.1.1 KeyeStudio Starter kit	247
13.1.2 Arduino Starter Kit	248
13.2 Additional Information on the Web	249
13.2.1 The Blog	249
13.2.2 Videos	251
12.2.3 Websites	251
CHAPTER 14: GLOSSARY	252

CHAPTER 1: GETTING STARTED WITH ARDUINO

1.1 What is an Arduino?

An **Arduino** is an open-source platform used to create electronic projects. When you say “Arduino,” it can refer to three different things, either a physical piece of hardware, a programmable environment, or an Integrated Development Environment.

The Arduino is a small microcontroller board with several connection sockets that can be wired to external parts such as motors, light sensors, LEDs etc. Arduino also refers to the software development field to program the Arduino Board by writing code for it. The I.D.E (Integrated Development Environment) refers to the text editing software on the computer that lets you write code and then upload it to the Arduino board.



Figure 1-1. An Arduino Uno board.

The Uno is one of the most popular boards in the Arduino family. It is a great choice for beginners who have just started learning how to use Arduino.

An Arduino IDE with code typed in it. The ten lines of code are all you need to blink the onboard LED on your Arduino!

```

Blink
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * Most Arduinos have an on-board LED you can control. On the Uno and Leonardo, it is attached to digital pin 13. If you're unsure what pin the on-board LED is connected to on your Arduino model, check the documentation at http://www.arduino.cc
 *
 * This example code is in the public domain.
 *
 * modified 8 May 2014
 * by Scott Fitzgerald
 */

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}

```

Figure 1-2. The IDE with code.

1.2 Hardware Pinout

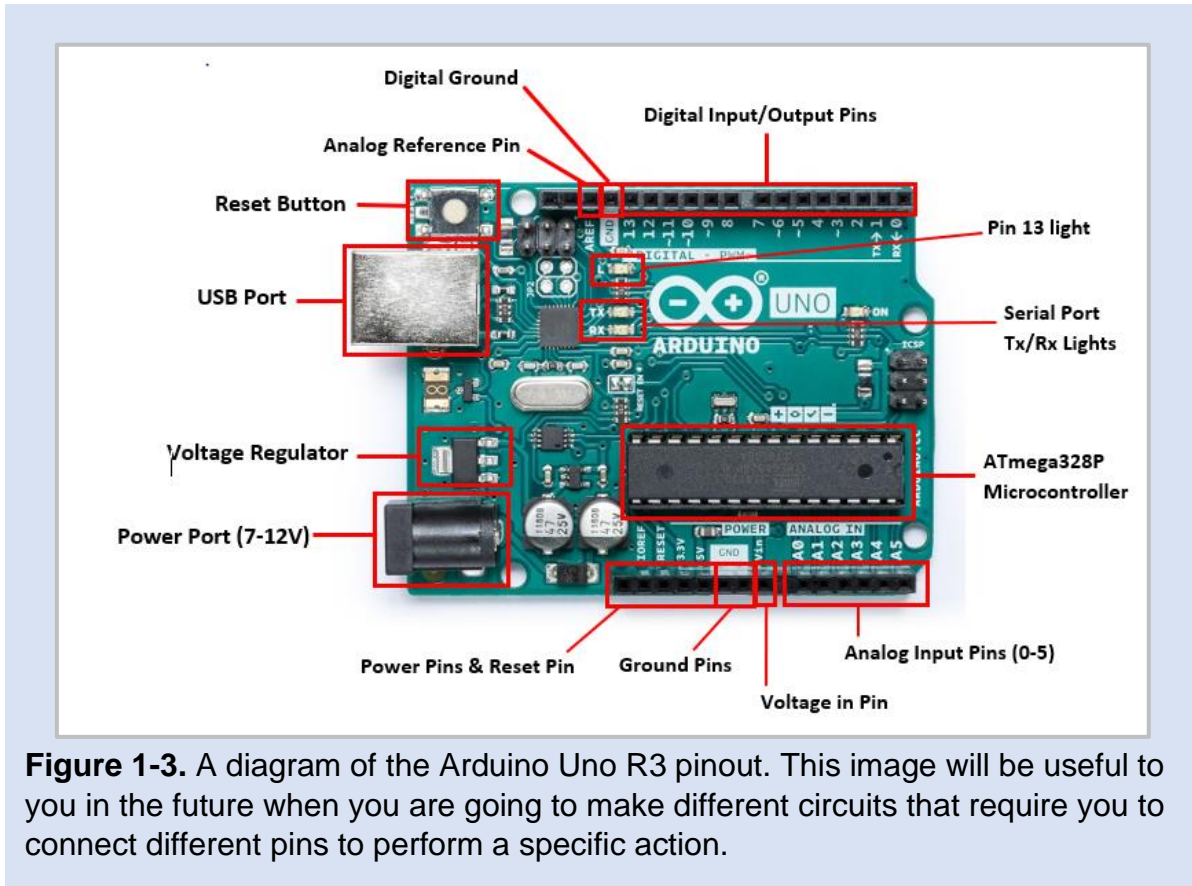


Figure 1-3. A diagram of the Arduino Uno R3 pinout. This image will be useful to you in the future when you are going to make different circuits that require you to connect different pins to perform a specific action.

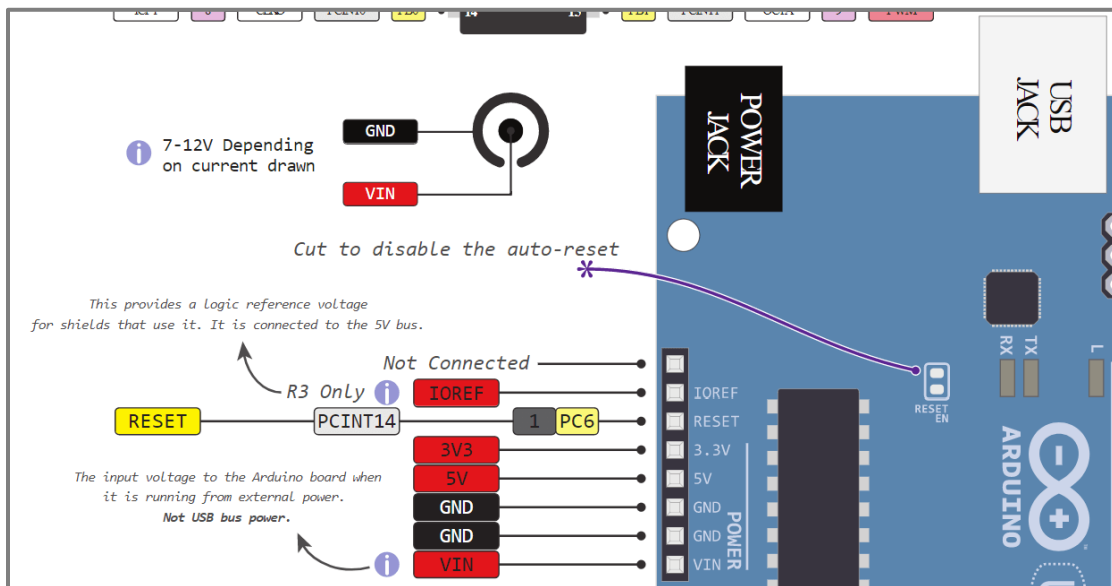


Figure 1-4. Arduino Uno Pinout of the power and grounding pins.

Power Jack – The DC Power Jack can be used to power your Arduino board. The power jack is usually connected to a wall adapter.

VIN Pin - This pin is used to power the Arduino Uno board using an external power source. It is recommended to be between 5-12 volts.

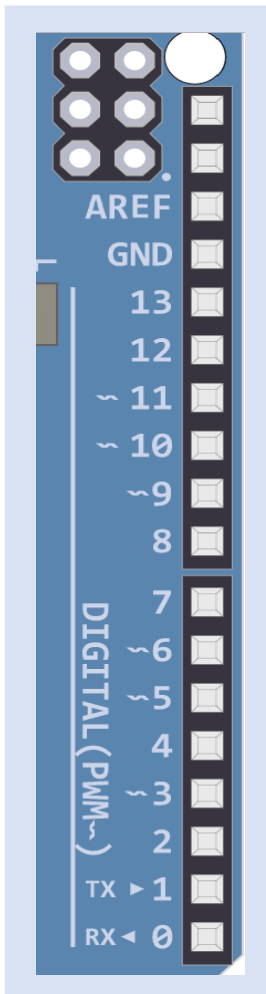
USB cable - when connected to the computer, provides 5 volts at 500mA.

5v and 3v3 - They provide regulated 5 and 3.3v to power external components.

GND pins - Are all interconnected. The GND pins close the electrical circuit. Always make sure that all GNDs (of the Arduino and attached components) are connected and have a common ground.

RESET - Resets the Arduino.

IOREF - The input/output reference. It provides the voltage reference with which the microcontroller operates.



Pins 0-13 of the Arduino Uno serve as digital input/output pins.

Pin 13 of the Arduino Uno is connected to the built-in LED.

Digital is a way of representing voltage in 1 bit: either 0 or 1. Digital pins on the Arduino are pins designed to be an inputs or output based on the needs of the user. Digital pins are either on or off. When ON they are in a HIGH voltage state of 5V and when OFF they are in a LOW voltage state of 0V.

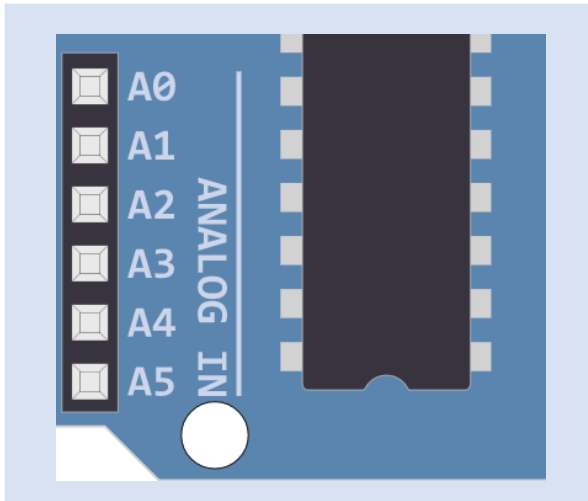
On the Arduino, When the digital pins are configured as output, they are set to 0 or 5 volts.

When the digital pins are configured as input, the voltage is supplied from an external device. This voltage can vary between 0-5 volts which is converted into digital representation (0 or 1). To determine this, there are 2 thresholds:

Below 0.8v - considered as 0.

Above 2v - considered as 1.

Figure 1-5. An Arduino Uno's Digital Pins.



The Arduino Uno has **6 analog pins**, which utilize ADC (Analog to Digital converter). These pins serve as analog inputs but can also function as digital inputs or digital outputs. Arduino Pins A0-A5 can read analog voltages.

ADC stands for **Analog to Digital Converter**. ADC is an electronic circuit used to convert analog signals into digital signals in which the micro-processor can understand and use it through its operation.

Figure 1-6. An Arduino Uno's Analog Pins

All the inputs and Outputs use standard jumper sockets connected to the microcontroller. Therefore, you can plug components straight in and see results immediately!

1.3 TinkerCad Circuits

Now that you know what an Arduino is and different pinouts that the Arduino board can have, you can now make a virtual circuit using the Arduino board! The website that will help you with this task is **TinkerCad**. Follow the steps below to create a TinkerCad account:

- 1) Go to <https://www.tinkercad.com/#/>
- 2) Click "Join Now"
- 3) Enter the requested information for registration.
- 4) Click "Create Account"

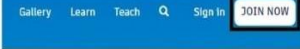


1.3.1 Creating a TinkerCad Account

3. Enter the requested info and click "Create Account"

1. Go to <https://www.tinkercad.com/#/>

2. Click on "Join Now"



Note: Make sure you use a 'real' email address you can access (you will get a conformation email with a link you need to open)



Note: Pick a year before 1999

1.3.2 Create a New Design

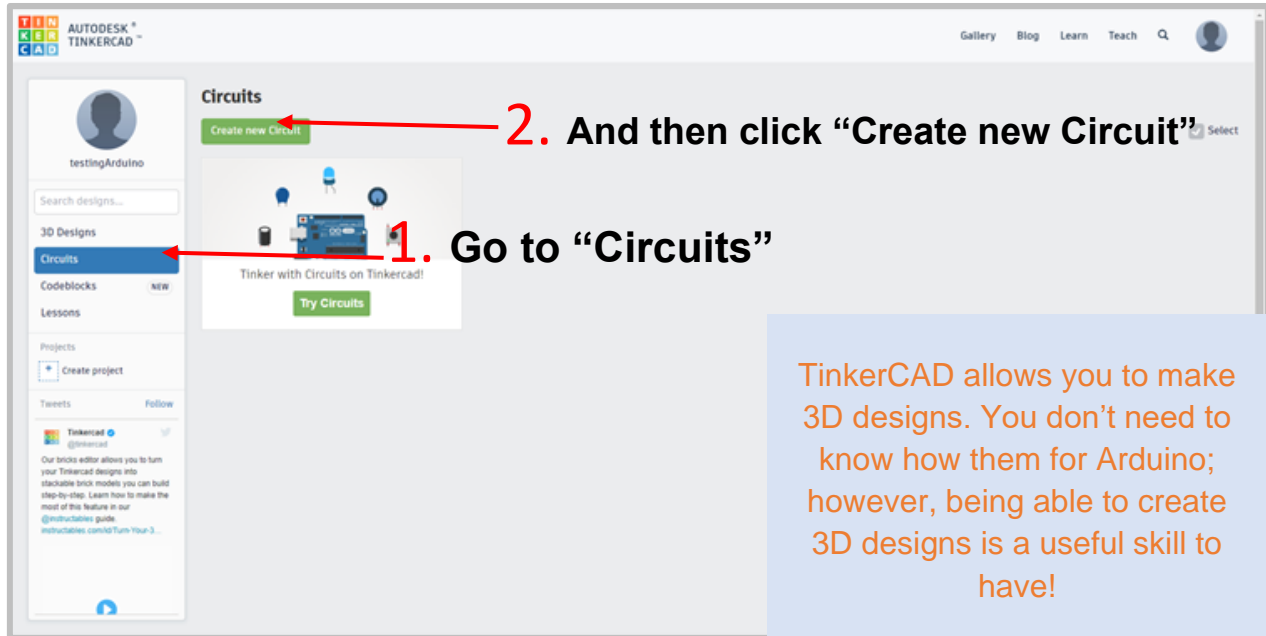
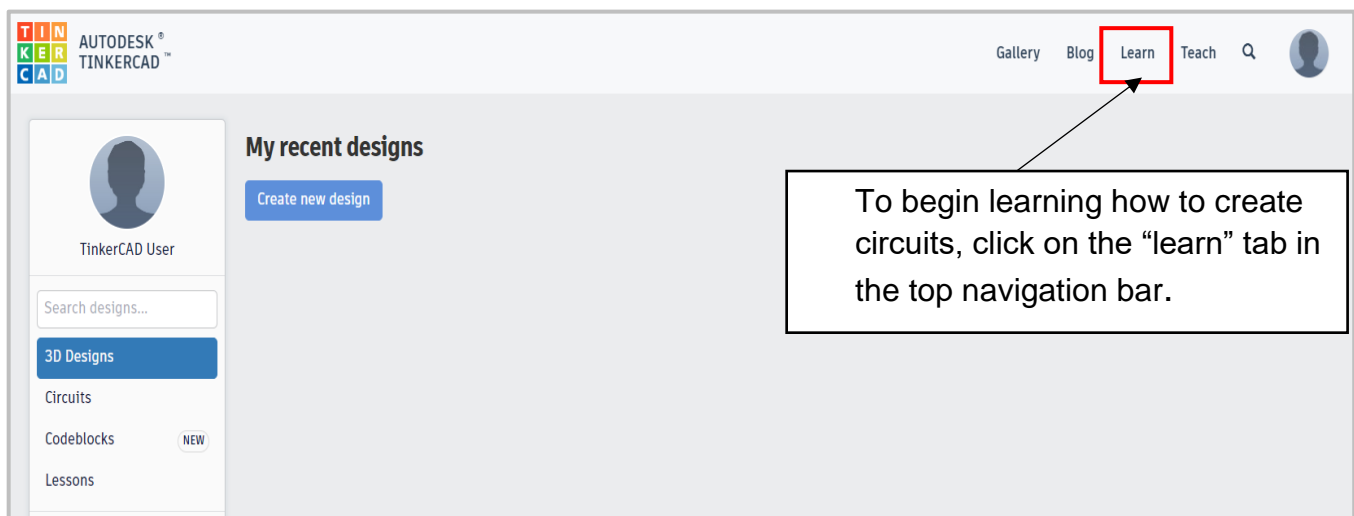


Figure 1-8. Inside the TinkerCad home menu for circuits.

1.3.3 Learn How to use TinkerCad Circuits



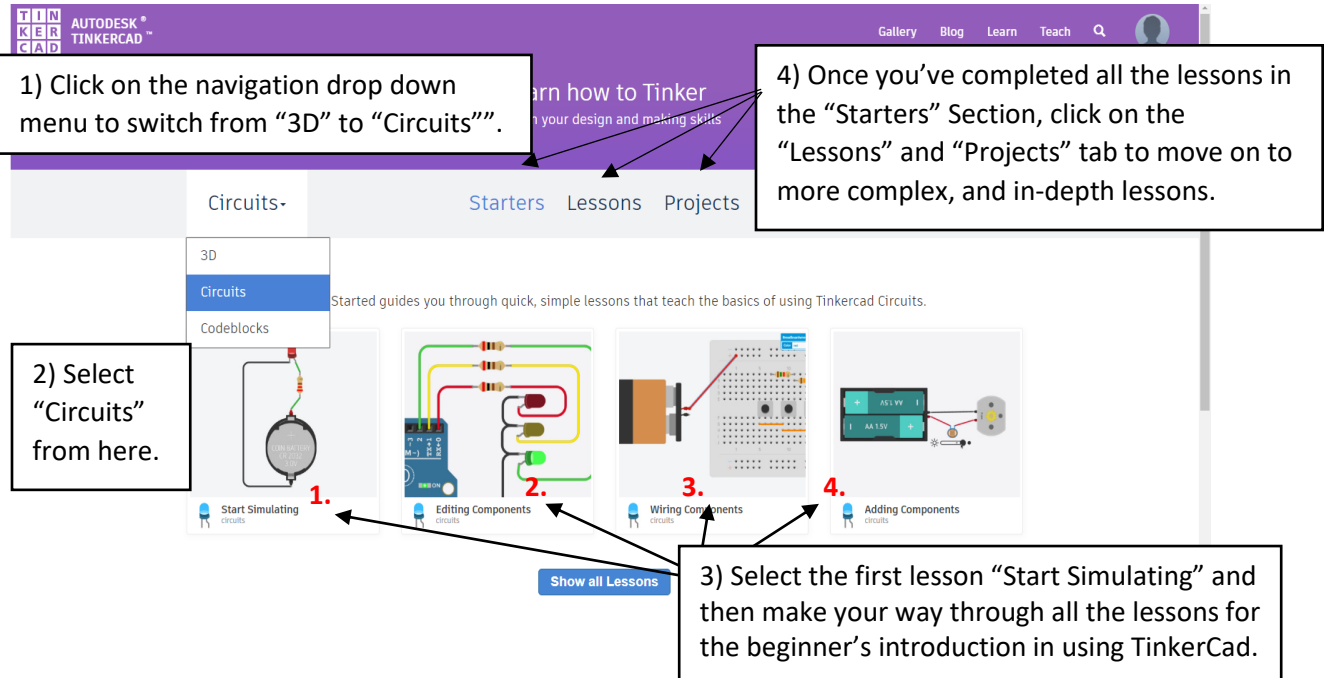


Figure 1-10. The Tinker Cad Learn menu. You can find the lessons by clicking on the Circuits tab then completing all "Starters", "Lessons" and "Projects" examples provided.

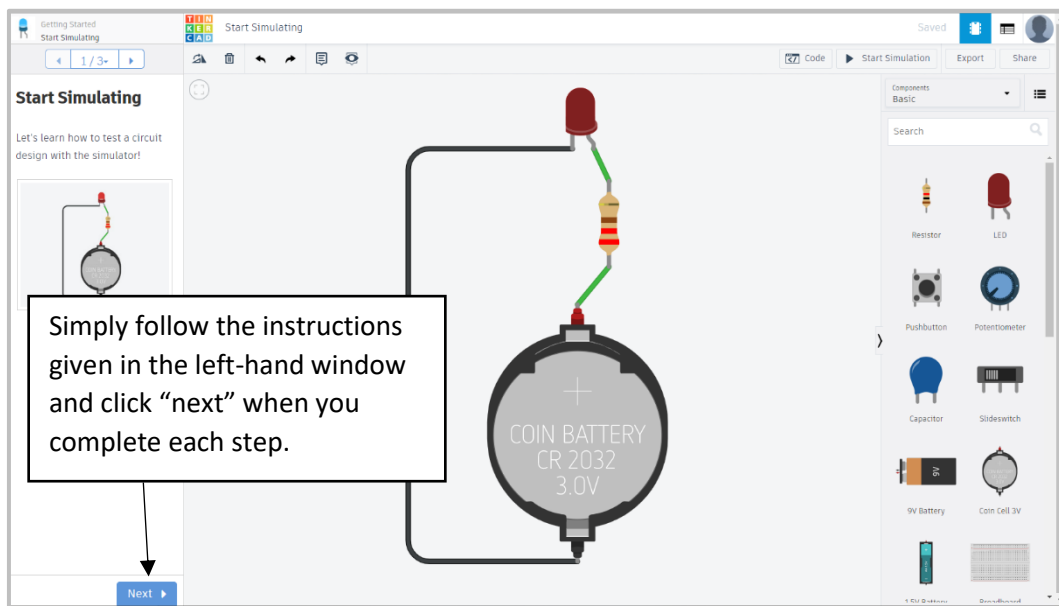


Figure 1-11. The TinkerCad Lessons for circuits. You can simply click the next button and follow the instructions given to complete each lesson.

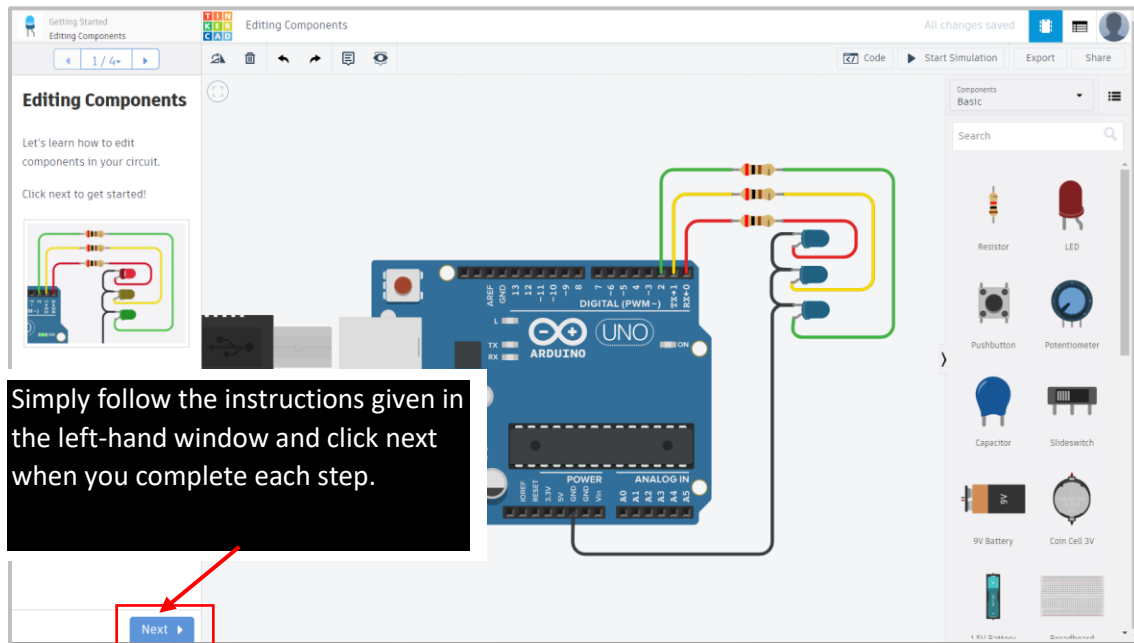


Figure 1-12. Another example of the TinkerCad Lessons for circuits. You can simply click the next button and follow the instructions given to complete each lesson.

1.4 Download Software



Open the Arduino download page at <http://arduino.cc/en/Main/Software>.

Select the download file for the operating system you are using from the list of current Arduino downloads. As of making this handbook, the current version of the Arduino IDE

After you have installed the software, plug one end of your USB cable into your Arduino board and the other end into your computer, to ensure the program is running correctly.



If you need more information or you still have questions, you can visit the link below:

If you have any problems for Mac downloads, visit
<http://arduino.cc/en/Guide/MacOSX> for more help.

To see a video about how to install the Arduino IDE on a computer running Mac OS X, visit the companion site at www.wiley.com/go/adventuresinarduino.

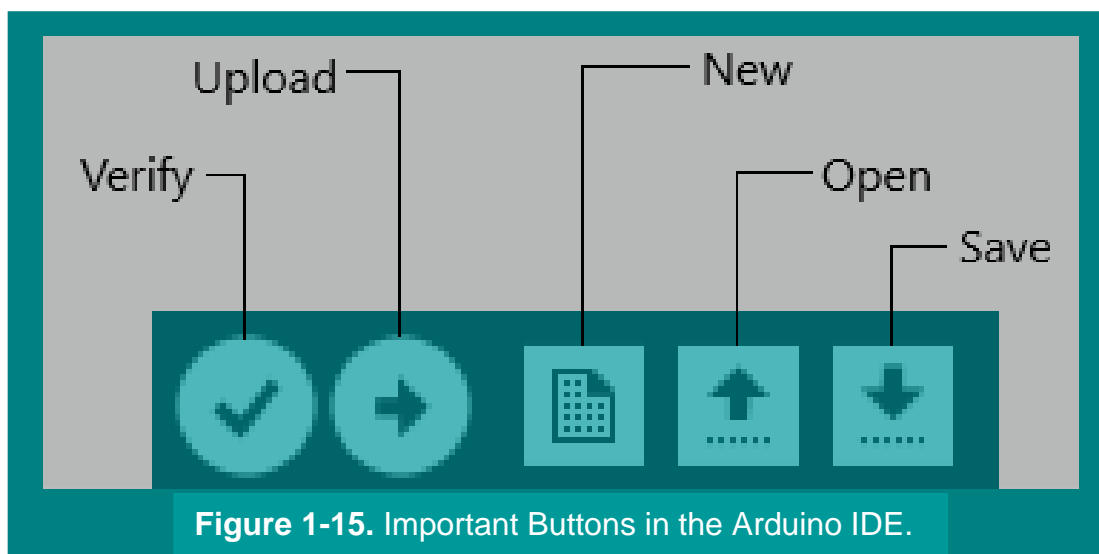
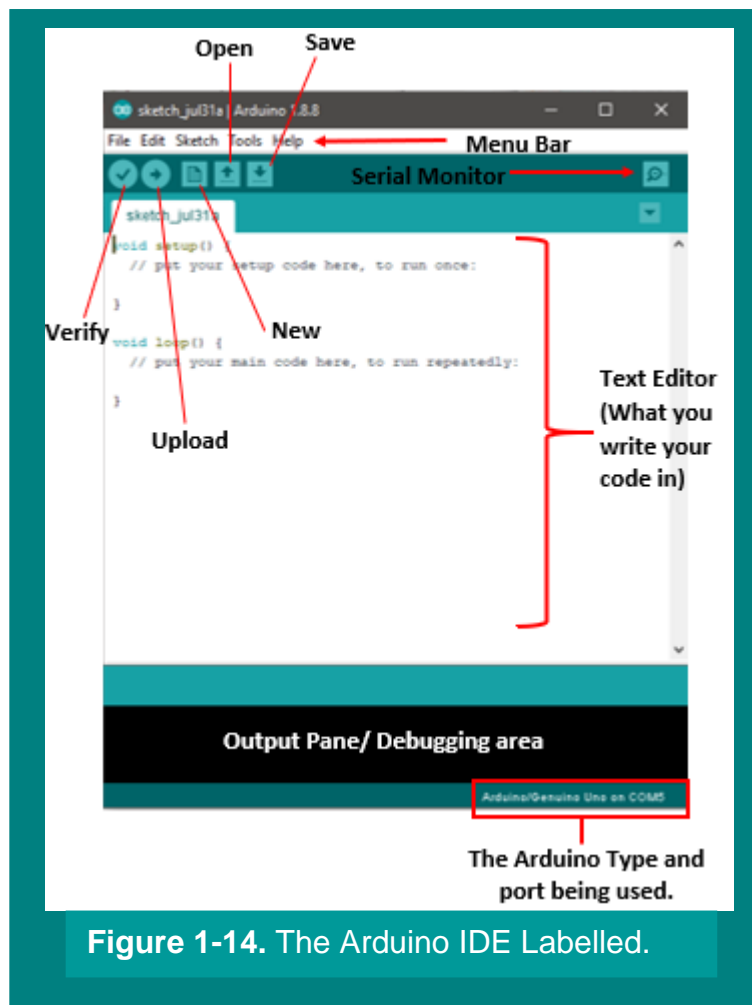
If you experience any problems for Windows OS download, visit:
<http://arduino.cc/en/Guide/Windows>

If your computer runs Linux, you should first visit the online documentation for Linux and Arduino at <http://playground.arduino.cc/Learning/Linux>

1.5 Example Code

Click Here to Download all code examples <https://exceedrobotics.com/wp-content/uploads/2022/01/Arduino-Handbook-Code.zip>

1.6 Exploring the Arduino IDE



The “**Check Mark**” is the Verify button. When you click this button, the Arduino compiles the code; in other words, it takes the code written and turns it into instructions the Arduino Board can understand. This process is called “**Compiling**.” If there are parts of the code the Arduino’s compiler does not understand (e.g. missing semicolon or typo in code) it prints an error message at the bottom of the IDE window.

The button that looks like an “**Arrow to the Right**” is an Upload button. The Upload button compiles the code and uploads it onto the Arduino UNO through a USB. The shortcut key for verifying is Ctrl + R, and key for uploading is Ctrl + U.

The button that looks like a “**File Icon**” is the New file button. By clicking this button, you will immediately open a new file.

The button that looks like an “**Arrow Up**” icon is the Open button. The open button allows you to open previous files.

The button that looks like an “**Arrow Down**” icon is the Save button. This button allows you to save the code so that you won’t lose all your progress.

Note: Arduino boards only holds one program, called a *sketch* at a time. Therefore, each time you upload a sketch, the Arduino loses the old sketch and only runs the new one.

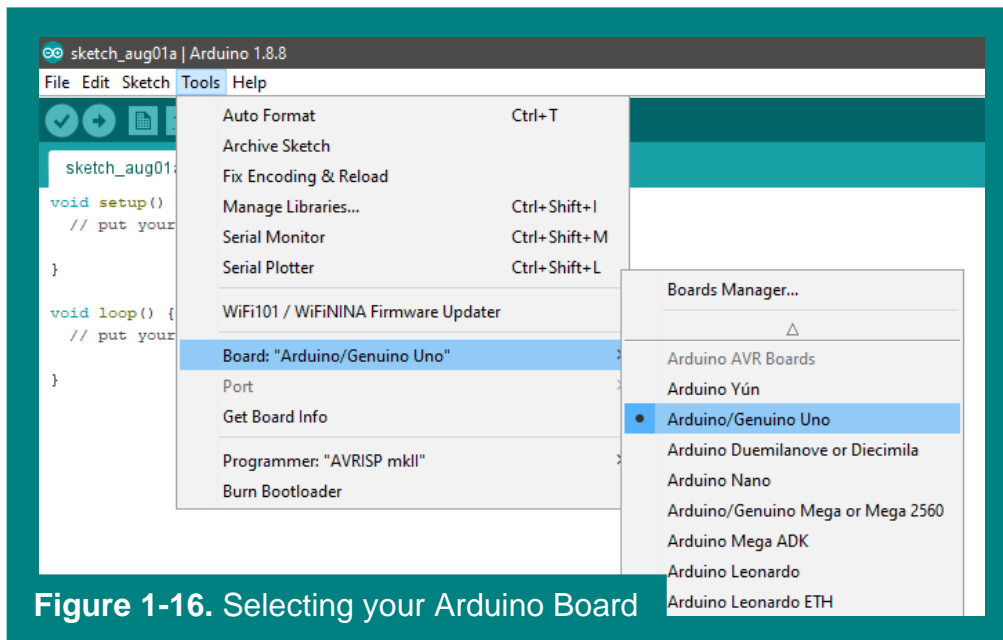


Figure 1-16. Selecting your Arduino Board

Open the IDE, go to the menu bar: Tools > Board, then select the Arduino Board type you are using.

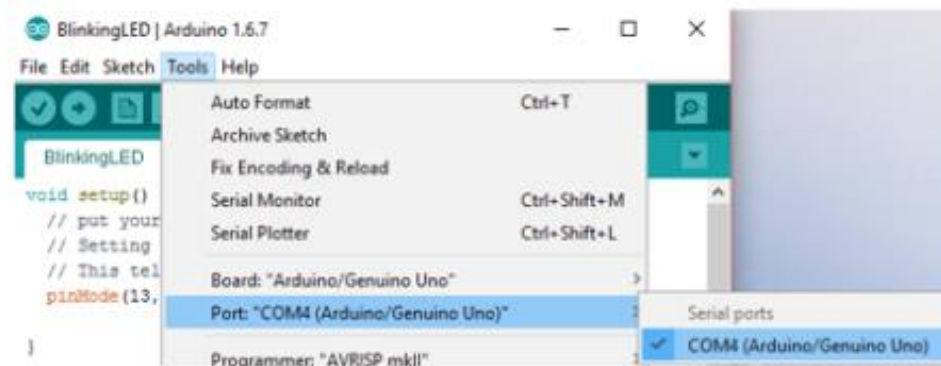


Figure 1-17. Selecting the port.

2) Go to menu: Tools > Port and select the port available:

- Windows is usually: "COM" port
- Mac is usually: "/dev/tty.usbmodem"
- Linux: usually: "/dev/ttyACMO"

3) Once that is done, you are ready to send code written in the IDE to the Arduino. Any orange error messages from the IDE may indicate some errors such as your board not being connected properly, selecting the wrong board, or software drivers needing to be installed.

Note: When you are uploading, the RX and TX lights should blink on the Arduino, and the code should start running. If not, you may have to do some troubleshooting to fix the problem.



1.7 Libraries and Troubleshooting

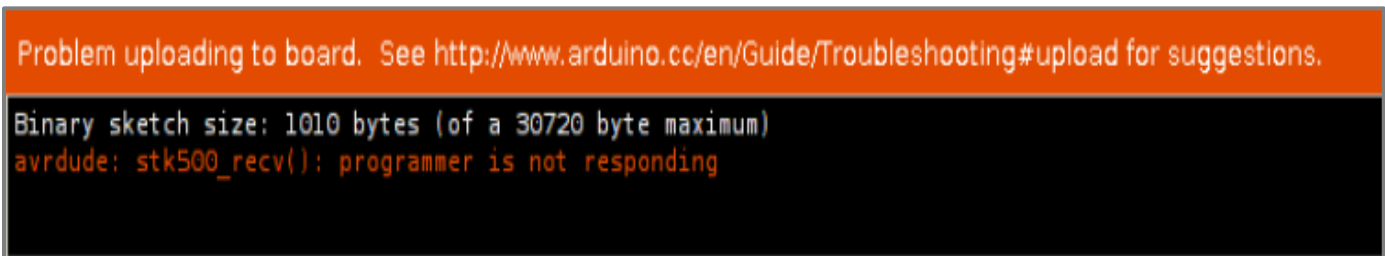


Figure 1-18. A Common Error Message to the Arduino IDE.

When something goes wrong when you're trying to upload code to an Arduino board, a message called "avrdude" might be printed at the bottom of the Arduino IDE. Usually, it's that the computer

is trying to use avrdude to send a new sketch to the Arduino Uno, but the computer can't find it. The problem could be caused by selecting the wrong port.

Arduino Command Quick Reference Table	
Command	Description
<i>setup()</i>	A function that runs once when the Arduino first starts. See also http://arduino.cc/en/Reference/Setup .
<i>loop()</i>	A function that is repeatedly run after the <i>setup()</i> is completed and until the Arduino is turned off. See also http://arduino.cc/en/Reference/Loop .
<i>pinMode()</i>	Sets the PIN entered as the argument to either output electricity. See also http://arduino.cc/en/Reference/PinMode .
OUTPUT	Keyword set in the second argument of <i>pinMode()</i> that says the pin will output electricity. See also http://arduino.cc/en/Reference/Constants .
<i>digitalWrite()</i>	Turns on or off the electricity at the specified pin. See also http://arduino.cc/en/Reference/DigitalWrite .
<i>analogWrite()</i>	Controls the exact amount of electricity flowing out of a specified pin https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/
<i>digitalRead()</i>	detects if power is flowing in to a specified pin https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/
<i>analogRead()</i>	detects how much power is flowing into a specified pin https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/
HIGH	Keyword used to turn on the electricity in <i>digitalWrite()</i> . See also http://arduino.cc/en/Reference/Constants .
LOW	Keyword used to turn on the electricity in <i>digitalWrite()</i> . See also http://arduino.cc/en/Reference/Constants .
<i>delay()</i>	Pauses the Arduino Uno for a specified number of milliseconds. See also http://arduino.cc/en/Reference/Delay .
<i>tone()</i>	generates a frequency in a specified pin https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/
<i>noTone()</i>	ends the use of a frequency in a specified pin https://www.arduino.cc/reference/en/language/functions/advanced-io/notone/
<i>map()</i>	converts the ranges of inputted values to a controlled ratio https://www.arduino.cc/reference/en/language/functions/math/map/

1.7.1 Testing the Device

- Find the Arduino application on your desktop
- Double click to open
- Open the Arduino Software you will see the IDE on the right
- Connect the USB to the computer
- Click on tools
- Click on port and select the Arduino port
- Click on the Board and Select

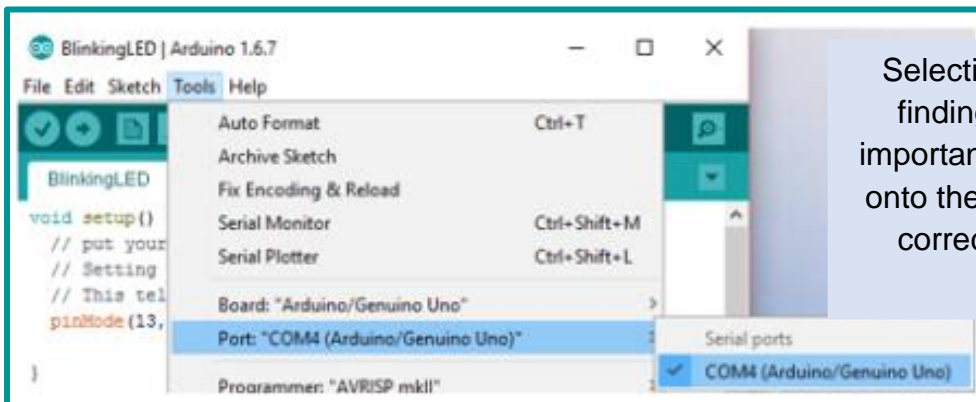
Uno



Helpful Steps to follow!



Figure 1-19. The Arduino IDE



Selecting the Arduino port and finding the Arduino device is important to properly upload code onto the board and to ensure the correct drivers and ports are working.

Figure 1-20. Inside Arduino.

1.8 Serial Monitor

The Serial Monitor is a “tether” between the computer and your Arduino. It has a separate pop-up window that acts as a different terminal that communicates by receiving and sending Serial Data. The Serial Monitor is handy when you have a continuous output of information from sensors, and you want to see them on a computer instead of the LCD. Serial Data is sent over a single wire (but usually travels over USB in our case) and consists of a series of 1's and 0's. Data can be sent in both directions (In our case on two wires).

How to set-up Serial Monitor

1) In the set-up, you need to begin the Serial Monitor

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

2) In the main loop, you need to print the info to the Serial Monitor

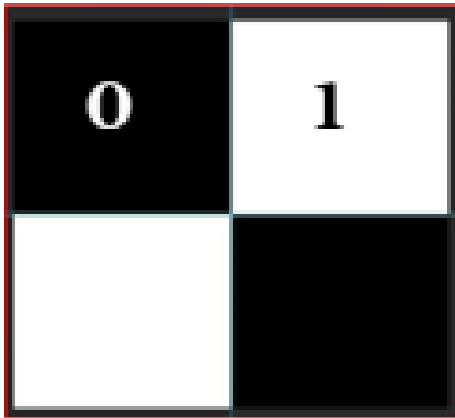
```
void loop ()
{
    Serial.println(“Hello”)
}
```

3) In tools, select Serial Monitor.

Now you can use the Serial Monitor. The Serial Monitor will help you with learning Arduino by letting you observe outputs to the different types of code you will write for your Arduino circuit.

1.9 What is a Digital Signal?

Digital signals are values that have a definite state. In the case for Arduino digital pins, they can only have two states “LOW” or “HIGH”, “ON” or “OFF”. (“0” or “5v” which makes “0” /”1”).



Let us assume we only have the colours black and white with no grey shade in between. This would be a Digital Signal where black is a 0 and white is a 1 with no other numbers in between.

Digital signals can only be one of 2 states ON/OFF in the case for the shades, only Black or White and nothing in between.

Figure 1-21. A colour-based example of digital signals.

Digital Input Examples:

- ON/OFF Switch
- Black and white colour sensor

Digital Outputs allow a microcontroller to output 2 logic states. Like ON/OFF.

1.10 What is an Analog Signal?

Analog signals are values that can have a range of many states.

There are many grey shades between white and black. We could not explain the colour “grey” using black or white or using 2 definitive values “0” / “1.”

In this example; we need a big range of numbers, let’s say the range of 0 -10 to represent white as well as all grays with different brightness and Black.

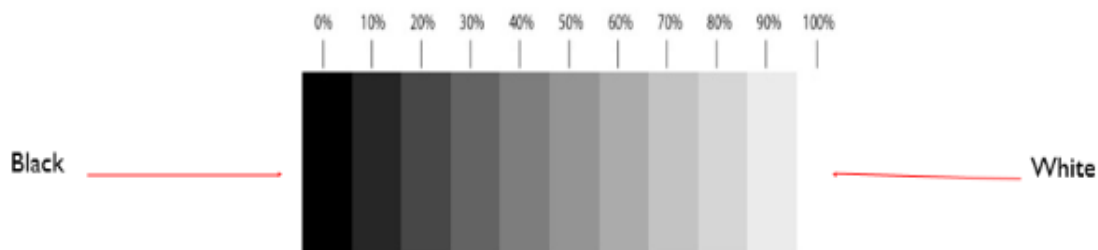
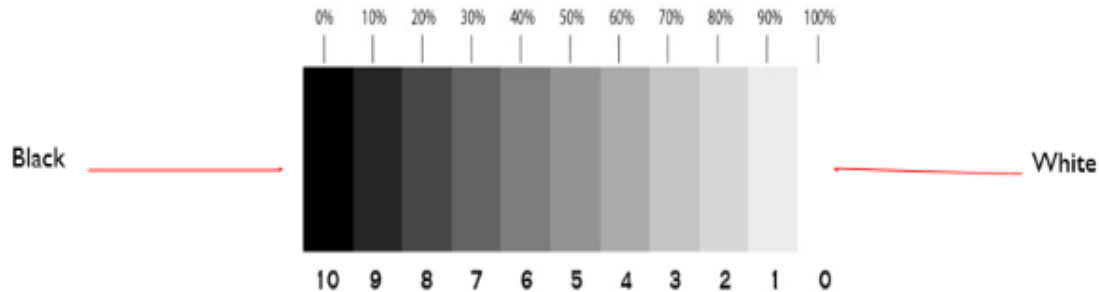


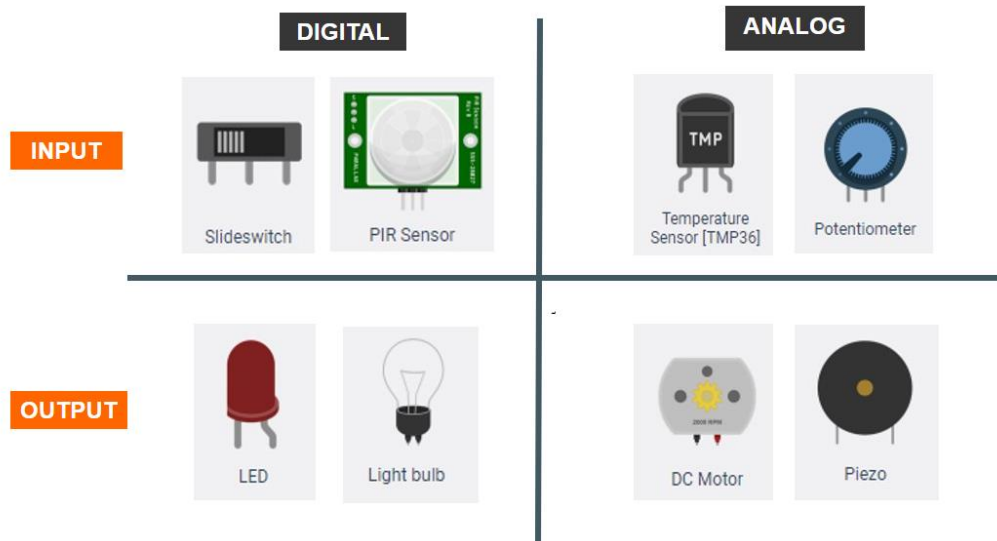
Figure 1-22. A colour-based example of analog signals. Analog signals can be a range of states. In the case for the shades example above, the analog signals include Black or White, and the different shades of grey in between.

1.11 Digital Vs Analog Signal

Once more we need to look at this gray scale for reference. digital signals can only be high or low or black and white but analog lets you access the many shades of gray, this adds versatility that can be both useful but also adds unnecessary work.

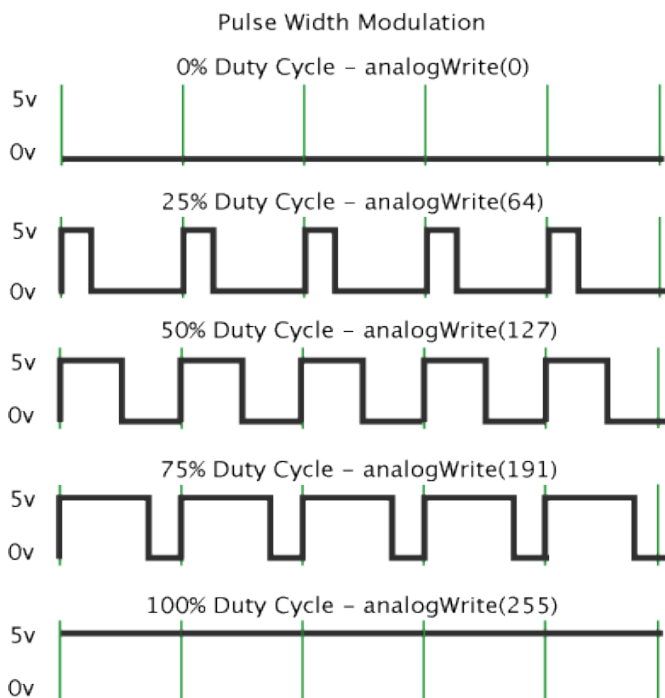


Here we see a few of the devices that the Arduino is used with. We see certain devices fall under digital or analog signals for very specific reasons. digital outputs do not need direct voltage control, they only need a constant high voltage signal to function, just as those under analog input need specific voltage signals to produce different results. but that doesn't mean these devices are exclusive to analog or digital signals. a motor would run on a digital signal, it would just always run at full speed. just as a light bulb could be controlled with an analog signal as the light would be as dim or bright as the signal allowed.



1.12 What is Pulse Width Modulation (PMW)?

Pulse Width Modulation, or PMW is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full “on” (5 Volts) and “off” (0 Volts) by changing the portion of the time the signal is spent “on” versus the time that the signal spent “off”. The duration of "on time" is called the **pulse width**. To get varying analog values, you change, or **modulate**, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.



The green lines represent a regular time-period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each.

A call to `analogWrite()` is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time).

CHAPTER 2: INTRODUCTION TO BASIC ELECTRIC CIRCUITS

2.1 What is Electricity?

Electricity is the flow of electrons. Atoms are made up of electrons which surround protons and neutrons. The electrons can “jump” from atom to atom. When a lot of electrons “jump,” a flow is created, like water droplets in a river. If you observe 2.1.1, then you will see the visual representation of electricity and the flow of electricity in atoms.

2.1.1 Electricity in a Circuit

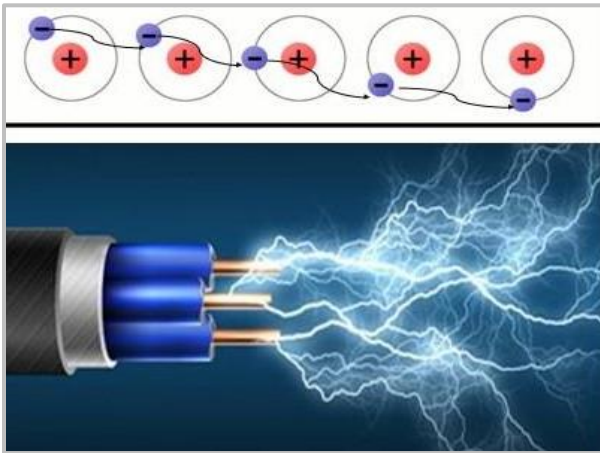


Figure 2-1. Electricity Flow in Circuits.

In a circuit, electrons flow from an area of “more electrons” (the negative terminal of the battery) to an area of “fewer electrons” (the positive end of the battery).

Electrons or electricity flows when the negative and positive sides are connected to a working circuit. Batteries are usually made of some acid, either in liquid or solid form. Therefore, converting chemical energy into electrical energy by a chemical reaction.

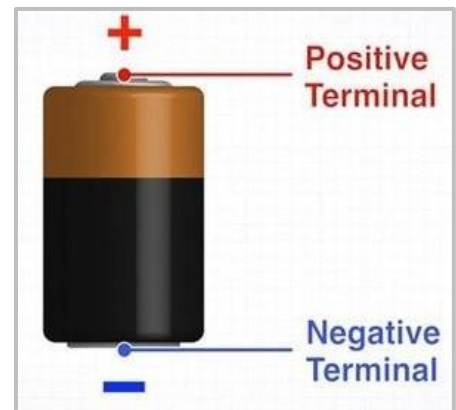
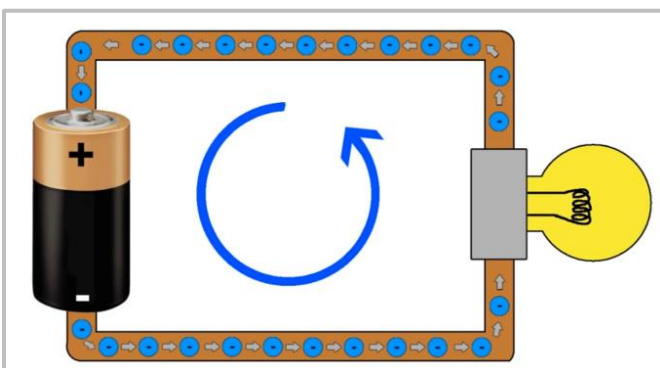


Figure 2-2. Electricity Flows out from the negative terminal to the positive terminal when connected to a circuit.



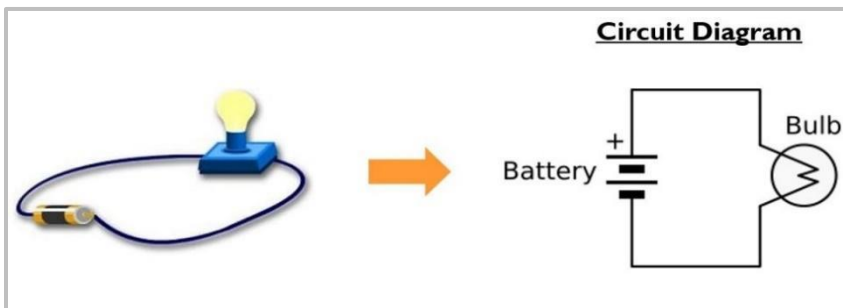
If we connect components together using wires and have a power source (battery), we can create a working circuit or a flow of electron (a **current**) through the circuit to power a device (e.g. lightbulb).

Figure 2-3. A simple circuit.

2.2 How a Circuit Works

During a project while working with circuits, instead of drawing the various electronic components, we use symbols to represent them in a **circuit diagram**.

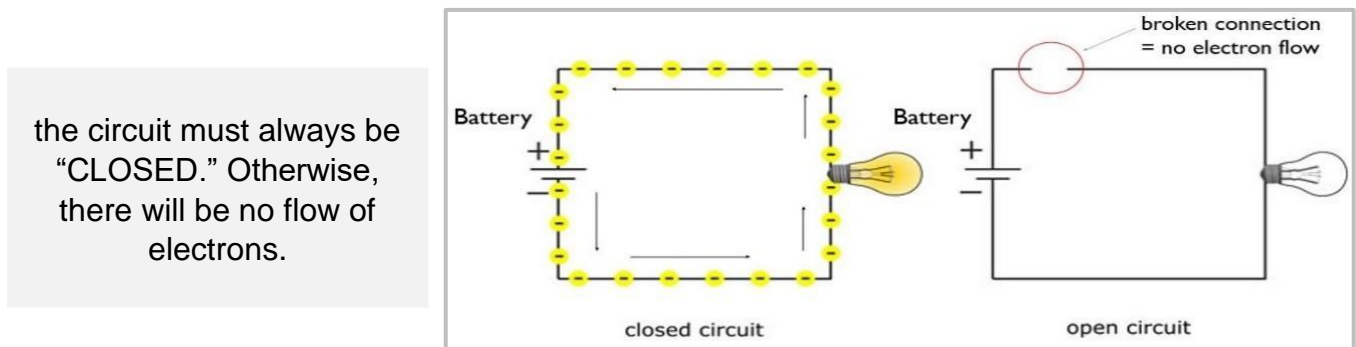
2.2.1 Circuit Diagrams



A circuit diagram shows how components are connected in a circuit. Each component / device is represented by a symbol. (Can be seen in 2.2.2).

Figure 2-4. A circuit diagram uses symbols to draw a circuit rather than drawing all the physical components.

Circuit diagrams show the connections with wires drawn as neat straight lines. The actual layout of the components is different from the circuit diagram and this can be confusing for beginners. It is better to concentrate on the *connections*, of the components and not the actual positions of them in the diagram.



the circuit must always be "CLOSED." Otherwise, there will be no flow of electrons.

Figure 2-5. A circuit diagram can be opened or closed. A circuit must be closed (wires are connected no gap) for the current to flow through and power the circuit.

Drawing circuit diagrams is not difficult. This is a useful skill for science and when working with electronics. You will need to draw circuit diagrams if you design your own circuits.

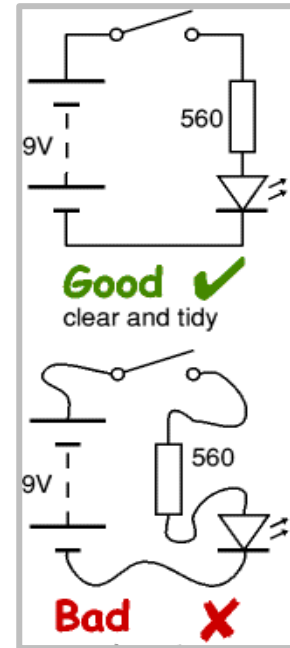
The following steps and tips will help in drawing:

- Use the correct symbol for each component.
- Draw wires using a ruler to have straight lines.
- Put dots (•) at junctions.
- Label components such as resistors, capacitors, LEDs etc. with their values.
- The positive (+) supply should be pointed up and the negative (-) supply pointed down.



Helpful Steps to follow!

Figure 2-6. Good and bad form of drawing a circuit diagram.



2.2.2 Circuit Diagram Symbols

Shown below are a few of the many symbols used in making circuit diagrams. These are the most common ones you will come across while working with Arduino.

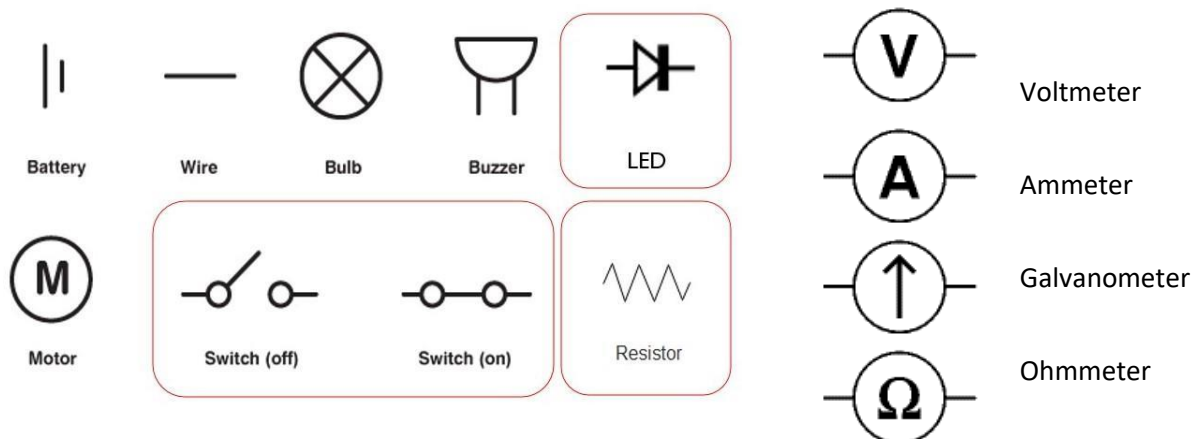


Figure 2-7. Circuit diagram symbols and what each symbol represents. These symbols are a good reference for future use in designing your own circuits.

2.3 Current and Voltage

Current is the rate of flow of electrons. Higher current means “more electrons” are flowing through the circuit at a given time. Current is measured in Amps (A) using an Ammeter.

Voltage is the electrical force that drives an electric current between two points.

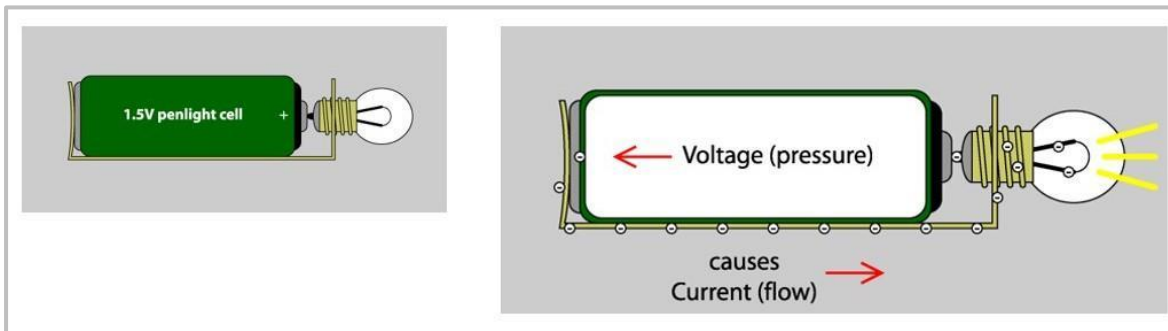
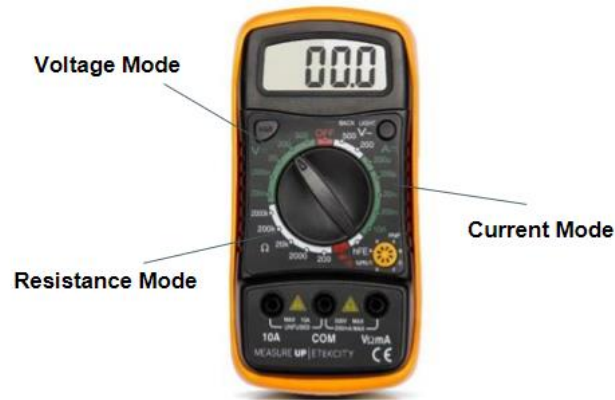


Figure 2-8. A higher voltage will result in more current.

	Current	Voltage
Symbol	I	V
Definition	Current is the rate at which electric charge flows past a point in a circuit. In other words, current is the rate of flow of electrons	Voltage is the potential difference between two points in a circuit. In other words, voltage is the "energy per unit charge".
Unit	(A), amps, amperage	(V), volts, voltage
Relationship	Current cannot flow without Voltage.	Voltage can exist without current.
Measuring Instrument	Ammeter	Voltmeter
SI Unit	1 ampere = 1 coulomb/second. ($I = V/R$)	1 volt = 1 joule/coulomb. ($V=W/C$)
In series circuits	Current is the same through all components connected in series.	Voltage gets distributed over components connected in series.
In parallel circuits	Current gets distributed over components connected in parallel.	Voltages are the same across all components connected in parallel.

2.3.1 Using The Multimeter

We can measure the **Voltage**, **current**, or **resistance** in a circuit using a **multimeter**. To make circuit measurements we need to make sure the multimeter is the **correct mode**. To do this, turn the dial to the desired setting and connect the meter to the circuit you want to measure.

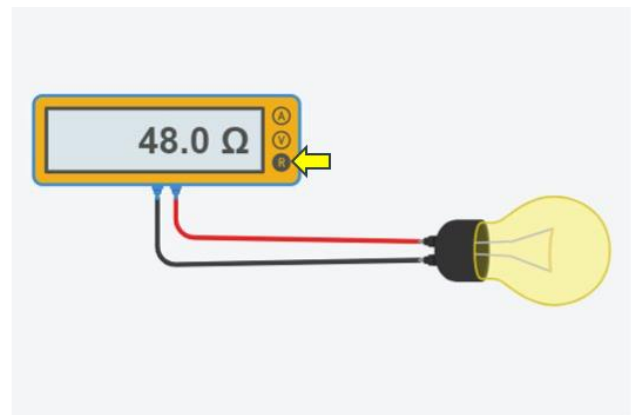


Note: Each type measurement requires the meter to be connected differently

Measuring Resistance:

Resistance is measured across device in the absence of a power supply

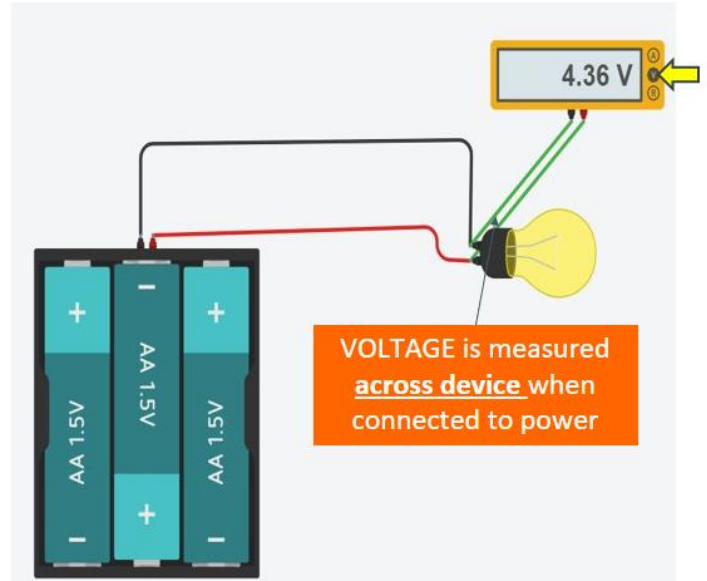
Make sure multimeter is set to “R” for resistance



Measuring Voltage:

VOLTAGE is measured **across device** when connected to power

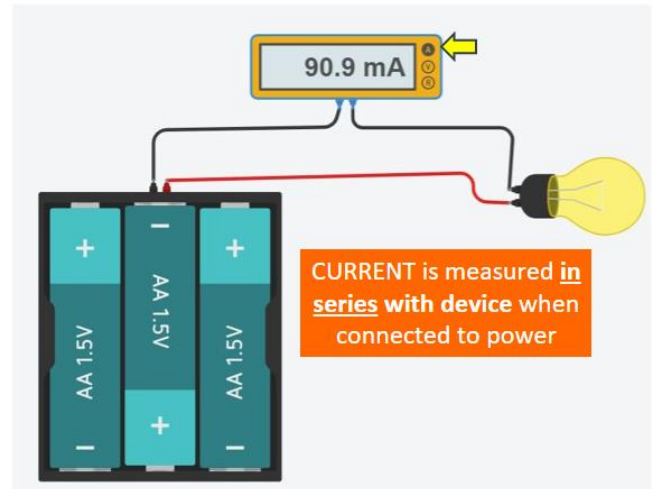
Make sure multimeter is set to “V” for voltage



Measuring Current:

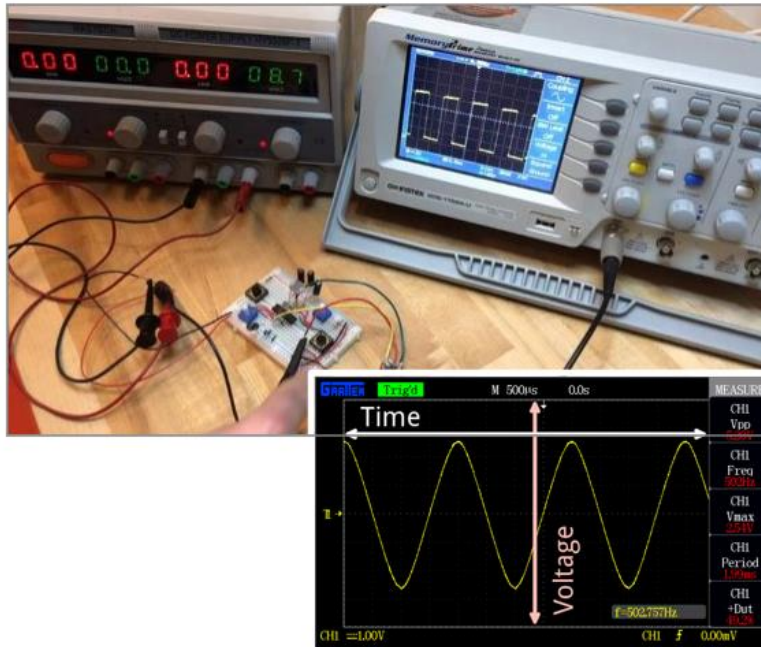
CURRENT is measured **in series** with device when connected to power

Make sure multimeter is set to “A” for current



2.3.2 Using The Oscilloscope

The main purpose of an oscilloscope is to graph an electrical signal as it varies over time. Most scopes produce a two-dimensional graph with time (s) on the x-axis and Voltage (V) on the y-axis.



2.4 Ohms Law

Ohms is the unit that is used to measure electrical resistance, resistance is represented by the symbol Ω . The electrical resistance determines how difficult it is for electrons to flow through a given material. Ohm's law is the rules that govern the relationship between Voltage, Current and Resistance. We can determine an equation from this law that is used to determine the amount of any three of these units within an electrical component given two of the three values.

$$V = I \times R$$

Find voltage using
current and resistance

$$I = V / R$$

Find current using
voltage and resistance

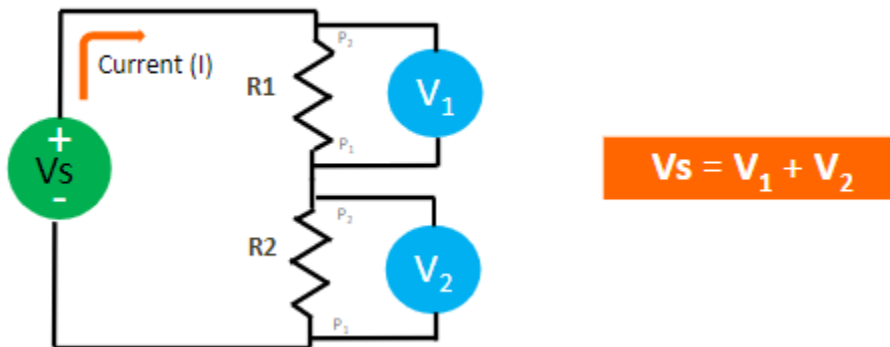
$$R = V / I$$

Find resistance using
voltage and current

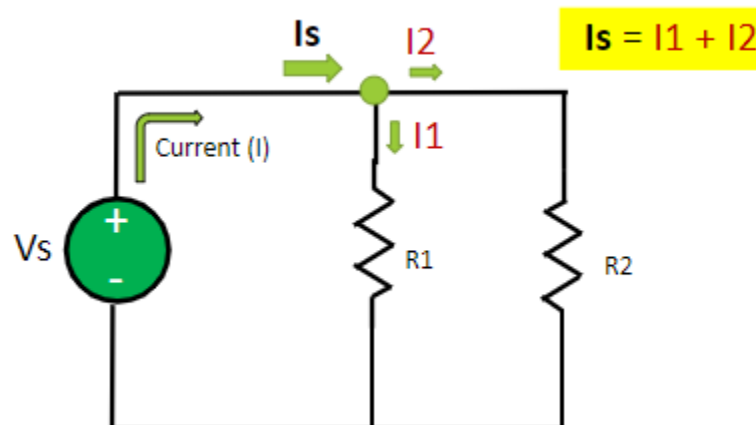
2.5 Kirchhoff's Voltage and Current Law

Voltage Drop is the loss in potential energy by a given component, if there is only one device in a circuit then the voltage drop across the device would be equal to the voltage source.

Kirchhoff's Voltage Law (KVL) says that if we add the voltage drop across all components in a loop within a circuit, they will add up to the voltage source. We may use this law to calculate the voltage drop across a resistor.



Kirchhoff's Current Law says that the number of electrons entering any point in a circuit must be equal to the number leaving that point. This means that if we count the number of electrons entering and those exiting, the value will be the same. this law lets us see how much current is flowing through a path given the current of the other paths



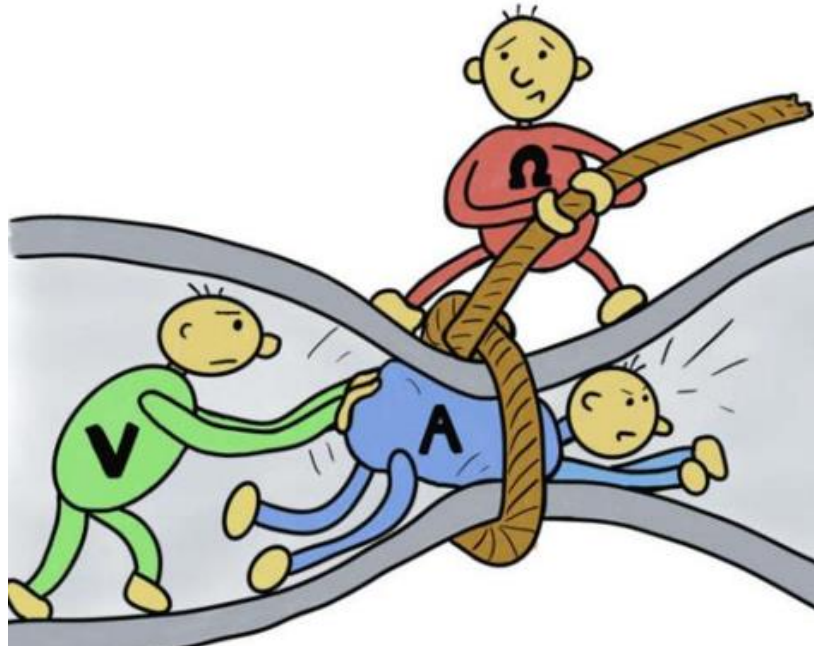
Homework Questions:

1. What is **current**:

- The push behind the electrons
- The measure of how hard it is for electrons to flow
- Total number of electrons that are flowing

2. What would happen to the amount of **current** if the **resistance increase**? (refer to picture below):

- Current decrease
- Voltage decrease
- Current increase



3. What is **Voltage**?

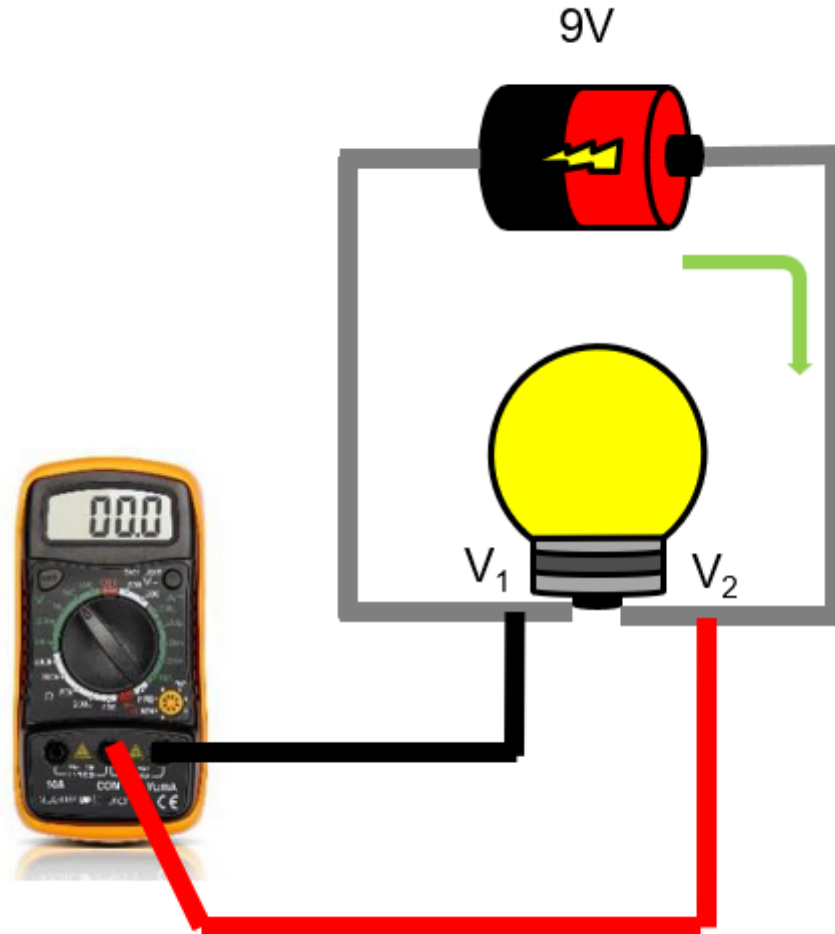
- It is a measure of energy stored in a system
- The energy between two points
- It is a measure of the potential energy each electron has

4. In which of these circuits can **voltage** be measured?

- A battery connected to a motor
- A microcontroller connected to a lamp
- An LED connected to a resistor

5. The voltage **across the lamp** is?

- 6V
- 7V
- 9V



6. Which of these voltages can be considered as **1 (5V)** for the **digital input of Arduino**?
 - a. 2.6V
 - b. 2.4V
 - c. 2V
7. **Push button** is an example of an **input device**
 - a. True
 - b. False
8. Which of these is an **incorrect pin**:
 - a. An output pin connected to an LED
 - b. An input pin connected to a push button
 - c. An input pin connected to a pullup resistor and a push button

CHAPTER 3: KEY CODING FUNDAMENTALS

3.1 Data Types

Data types, in programming is a classification that specifies the type of value a variable has and what type of mathematical, relation or logical operations can be applied to it without causing an error. Every piece of information has its own data. The type of a variable determines how much space it occupies in the storage and how the value of the variable is interpreted.

Byte – A byte stores an 8-bit value from, 0 to 255.

3.1.1 Boolean

A **Boolean** holds one of two values, true or false. This can be represented in the “C” coding language by using the word ‘true’ or ‘false’ but it can also be described as a 1 or 0.

This can be interpreted as a light or a switch. One representing when the switch is pressed, or the light is on, giving a result of True or 1. 0 representing when the switch is open, or the light is off, giving a result of false. Each boolean variable occupies one byte of memory.

Boolean Value Example

```
boolean val = false; // declaring a Boolean type value with false
boolean val = true; // declaring a Boolean type value with true
```

3.1.2 Char

A **Char** Data type holds a single character value and takes up one byte of memory.

Characters are written in single quotes like this: ‘K’. Characters are stored as numbers. The specific values for each character can be seen in the ASCII Chart. This makes it possible to do arithmetic operations on characters. For example, ‘K’ + 1 has a value of 76 because the ASCII value of the capital letter K is 75.

Char Value Example

```
Char val = ‘b’; // declaring a char type value initialized with character a
Char val = 99; // declaring a char type value initialized with character 97
```

3.1.3 Integers

Integers called “**int**” in code are the primary data type for number storage. They are any whole number and can be positive, negative or zero. On the Arduino Uno an **int** stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 number values that **int** can store.

Int Value Example

```
Int num = 99; // declaration of type int variable and initialized with 99
```

3.1.4 Long

Long variables are large size variable data types for number storage, and stores 32 bits (4 bytes), range from -2,147,483,648 to 2,147,483,647.

Long Value Example

```
Long acceleration = 2145678910 ;  
//declaration type Long variable initialized with 2145678910
```

3.1.5 Short

Short variables are a 16-bit data type. A short can store up to 16-bit value (2-byte). This yields a range of -32,768 to 32,767.

Short Value Example

```
Short num = 13 ; //declaration of type short variable initialized with 13
```

3.1.6 Float / Double

Floating-point numbers, are numbers that have a decimal point.

Floating-point numbers are often used to approximate analog and continuous values because they have greater accuracy than integers. Floating-point numbers can be as large as

3.4028235E+38 and as low as -3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

Floats have only 6-7 decimal digits of precision. That means the total number of digits, not the number to the right of the decimal point. Unlike other platforms, where you can get more precision by using a double (e.g. up to 15 digits), on Arduino, **double** is the same size as float.

Floating point numbers are not exact and may yield strange results when compared. For example 6.0 / 2.0 may not equal 3.0. You should instead check that the absolute value of the difference between the numbers is less than some small number.

Float/Double Value Example

```
float num = 2.71828; //declaration of variable with type float and initialize  
it with 2.71828  
double num = 3.1415
```

3.1.7 Array

An **array** is list of variables that are accessed with an index number. Arrays help you to format the code as well as, access a certain variable inside of a list. Think of an array as a pouch of variables or data (float, integer, long), which help you to organize the data. When you need something, you get the certain data inside this pouch, allowing it to be accessed how you want.

Arrays start counting from 0, therefore in an array of 10, they don't end at 10 but at 9. For example; `int myArray[10] = {9, 5, 6, 6, 7, 7, 2, 4, 1, 3};` contains 10 integers, however the last integer will end at 9 because arrays start counting from 0.

For Example:

- myArray[0] contains 9
- myArray[1] contains 5
- myArray[2] contains 6
- myArray[9] containing 3

3.1.8 Variable Creation

To create a variable, we need to include three pieces of information related to its use:

We must specify the type of information that we intend to store (the data type).

We must assign our variable a name, usually one that references its purpose or the information it stores. Rules of Variable naming must be followed

(must start name with letters, no spaces, must be unique so their can't be duplication, is case sensitive, can't use arduino function names)

We have the option of initializing the variable by including a value to be stored.

However, we can also leave this out and assign or change the stored value later. The value put into the variable stored must be consistent with the data type we selected.



Helpful Steps
to follow!

Practice Examples

1. Declare and initialize the number 9 to an int value with the variable name 'num'.

- 1) Specify the type of information that you intend to store. 9 is an integer, so you need to write **int** before your variable name.
- 2) You need to assign a variable name that usually stores the information. The name "num" is given to us. For future reference variable names should represent what the value is storing to follow proper coding standards.
- 3) We need to include a value that will be stored inside of the variable, in this case this would be a **9**. Combine everything together, we will have

```
Answer: int num = 9;
```

2. Declare and initialize a variable to store value of 3.14.

- 1) Assign a data type to the variable that we are going to create. Our data type contains decimals so in this case we need to use a **float** data type.

- 2) Next we need to name our variable that is easily recognizable. In this case we can name our variable *pi*, because the variable is 3.14 which is the first few numbers of Pi.
- 3) Finally, we need to include a value that will be stored inside of the variable which is **3.14**.

Answer: `float pi = 3.14;`

3.2 Arduino Code and the IDE

This section introduces the concept of coding and the language used while coding. These are the Key coding Fundamentals to begin writing your own code for Arduino to create the software to run your own projects.

```

p07_Keyboard
// This program code is part of the public domain.
*/
// create an array of notes
// the numbers below correspond to the frequencies of middle C, D, E, and F
int notes[] = {262, 294, 330, 348};

void setup() {
  //start serial communication
  Serial.begin(9600);
}

void loop() {
  // create a local variable to hold the input on pin A0
  int keyVal = analogRead(A0);
  // send the value from A0 to the Serial Monitor
  Serial.println(keyVal);

  // play the note corresponding to each value on A0
  if (keyVal == 1023) {
    // play the first frequency in the array on pin 8
    tone(8, notes[0]);
  } else if (keyVal >= 990 && keyVal <= 1010) {
    // play the second frequency in the array on pin 8
    tone(8, notes[1]);
  } else if (keyVal >= 980 && keyVal <= 515) {
    // play the third frequency in the array on pin 8
    tone(8, notes[2]);
  } else if (keyVal >= 5 && keyVal <= 10) {
    // play the fourth frequency in the array on pin 8
    tone(8, notes[3]);
  } else {
    // if the value is out of range, play no tone
    noTone(8);
  }
}

```

Figure 3-1. Arduino IDE with code.

“setup()” Function – Any code written in this function runs once when the Arduino is turned on and running.

Comments – Regular text added throughout the code for helping people understand the code written and what each part of the code does.

“loop()” Function – Any code written in this function is run over and over again as long as the Arduino is on, and the code must be written in void loop for the Arduino to run the code and perform it.

“;” semicolons: It is proper syntax and coding rules that every line of code not starting/ending with a curly brace or a comment must end with a semicolon at the end

3.2.1 Introduction to Code

Arduino uses the “C” programming language. All the code written for an Arduino program is simply a list of instructions in this C language that tells the Arduino to do something. Writing code must follow a specific syntax to make the code follow standards in programming.

Every Arduino sketch must have the setup & loop functions. The setup function is triggered only when the Arduino starts up and runs just that once. The loop function continuously runs the code types in it endlessly.

Syntax: the set of rules, principles, and processes that govern the structure of a given language, in this case, the structure of the “C” programming language

Sketch: The text/code of written for the Arduino. The IDE is used to write, save, edit, and load sketches.

Function: Computer instructions that perform an action or respond to triggers. a group of statements that together perform a task

Setup: The first function in most Arduino sketches. This code runs before anything else and is only run once when the Arduino powers up

Loop: The most important function. Required for any sketch, where the Arduino does all the work. Looping the same code continuously to perform the code you wrote for it.

Variable: Simply names used to refer to some location in memory – a location that holds a value the code is working with. It may help to think of a variable as a placeholder for a value. It represents a value such as a number or letter that can change.

Variables do not require, but should follow a proper naming convention when they are more than one word:

Spaces are not allowed in file, variable and function names. The 2 naming conventions that allow multi-word names are:

camelCase: first letter lowercase, but

allOtherLettersAfterStartsWithAnUppercase

all_lowercase: use_an_underscore_with_no_capital_letter_after_each_word

3.3 Arduino Math/Arithmetic

3.3.1 Arithmetic Operators

% (remainder)

Remainder operations calculate the remainder when one integer is divided by another. The % (percent) symbol is used to carry out remainder operations.

Syntax: remainder = dividend % divisor;

Example

```
int x = 0;

x = 7 % 5; // x now contains 2
x = 9 % 5; // x now contains 4
x = 5 % 5; // x now contains 0
x = 4 % 5; // x now contains 4
x = -4 % 5; // x now contains -4
x = 4 % -5; // x now contains 4
```

* (multiplication)

The Multiplication operator * (asterisk) operates on two operands to produce the product.

Syntax: product = operand1 * operand2;

Example

```
int a = 3;
int b = 12;
int c = 0;
c = a * b;

// the variable 'c' gets a value of 36 after this statement is executed
```

+ (addition)

The Addition operator + (plus) operates on two numbers to produce the sum.

Syntax: sum = operand1 + operand2;

Example

```
int a = 7;
int b = 8;
int c = 0;
c = a + b;

// the variable 'c' gets a value of 15 after this statement is executed
```

- (subtraction)

The **Subtraction** operator - (minus) operates on two numbers to produce the difference of the second number from the first number.

Syntax: **difference = operand1 - operand2;**

Example

```
int a = 5;
int b = 10;
int c = 0;
c = a - b;

// the variable 'c' gets a value of -5 after this statement is executed
```

/ (division)

The **Division** operator / (slash) operates on two numbers to break a number up into an equal number of parts and produce the result.

Example

```
int a = 50;
int b = 10;
int c = 0;
c = a / b;

// the variable 'c' gets a value of 5 after this statement is executed
```

Syntax: **result = numerator / denominator;**

= (assignment operator)

The equal sign (=) is called the assignment operator. The assignment operator tells the Arduino board to evaluate whatever value or expression is on the right side of the equal sign and store it in the variable to the left of the equal sign.

Example

```
int analogVal; // declare an integer variable named sensVal

analogVal = analogRead(0);
// store the (digitized) input voltage at analog pin 0 in SensVal
```

3.3.2 Comparison Operators

!= (not equal to)

The (!=) not equal to sign compares the variable on the left with the value or variable on the right of the operator. Returns true when the two operands are not equal.

Syntax: x != y; // is false if x is equal to y and it is true if x is not equal to y

Example

```
// if x is not true
if (!x)
{
    // statements
}
```

< (less than)

Compares the variable on the left with the value or variable on the right of the operator. Returns true when the operand on the left is less (smaller) than the operand on the right.

Syntax: a < b;

Example

```
if (x < y)
{
    // tests if x is less (smaller) than y
    // do something only if the comparison
    //result is true
}
```

<= (less than or equal to)

Compares the variable on the left with the value or variable on the right of the operator. Returns true when the operand on the left is less (smaller) than or equal to the operand on the right.

// is true if x is smaller than or equal to y and it is false if x is greater than y

Syntax: $x \leq y$;

Example

```
if (x <= y)
{
    // tests if x is less (smaller) than or equal to y
    // do something only if the comparison result is true
}
```

== (equal to)

Compares the variable on the left with the value or variable on the right of the operator. Returns true when the two operands are equal.

Syntax $x == y$; // is true if x is equal to y and it is false if x is not equal to y

Example

```
if (x == y)
{
    // tests if x is equal to y
    // do something only if the comparison result is true
}
```


> (greater than)

Compares the variable on the left with the value or variable on the right of the operator. Returns true when the operand on the left is greater (bigger) than the operand on the right.

Syntax: `x > y`; // is true if x is bigger than y and it is false if x is equal or smaller than y

Example

```
if (x > y)
{
    // tests if x is greater (bigger) than y
    // do something only if the comparison result is true
}
```

>= (greater than or equal to)

Compares the variable on the left with the value or variable on the right of the operator. Returns true when the operand on the left is greater (bigger) than or equal to the operand on the right.

Syntax: `x >= y`; // is true if x is bigger than or equal to y and is false if x is smaller than y

Example

```
if (x >= y)
{
    // tests if x is greater (bigger) than or equal to y
    // do something only if the comparison result is true
}
```

3.3.3 Boolean Operators

! (logical not)

Logical NOT results in a true if the operand is false and vice versa.

Example

```
if (!x)
{
    // if x is not true
    // statements
}
```

It can also be used to invert the boolean value.

```
x = !y; // the inverted value of y is stored in x
```

&& (logical and)

Logical AND results in true **only** if both operands are true.

Example

```
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH)
{
    // if BOTH the switches read HIGH
    // statements
}
```

|| (logical or)

Logical OR results in a true if either of the two operands is true.

Example

```
if (x > 0 || y > 0)
{
    // if either x or y is greater than zero
    // statements
}
```

3.4 Making Decisions in Code

3.4.1 If Statements

An **if statement** is a logic command that allows you to direct the flow of your program. Using a conditional statement that can be answered as either a true or false can lead to your program to either execute or skip over a block of code.

This can be interpreted logically as an if/then statement. For example, you're given the condition: if you eat your dinner, then you get dessert. In this example, you only get to have dessert if you ate your dinner. Otherwise, dessert is skipped or ignored.

If Statement Example

```
If (distance < 10)
{
    stopAllMotors();
}
```

We can read this as: if the distance is less than 10, then stop all the motors. This could have been part of the program for a rover type robot where it is autonomously driving. Given some mechanism for measuring distance from objects, we could use something like this to prevent collisions.

In general, an if statement will have the following form:

```
if ( condition ) {
    statement
}
```

3.4.2 Else Statements

Else/Else-If statements: We can expand on the “if” structure to include multiple blocks of code to be considered. By adding either an ‘else’ or “else if” clause, we can add more complex flow control to our programs than a basic ‘if’ statement. If either one or more ‘else if’ clauses are added, they are evaluated in order from top to bottom, starting with the initial if statement condition.

When checking the clauses, the first one to be found true is the one that will be selected, and its containing block executed, all subsequent blocks will be ignored, regardless of whether their conditional statement would have been found true.

Finally, an ‘else’ clause can be attached. In this case, the code contained within the ‘else’ clause will always be executed if none of the preceding conditions are met.

Example Code

```
if (temperature >= 20)
{
    Serial.print("It is hot outside");
}
else if (temperature >= 10)
{
    // 10 <= temperature < 20
    Serial.print("It is nice outside, may need a sweater!");
}
else
{
    // temperature < 10
    Serial.print("It is cold outside, grab a jacket!");
}
```

The above code may be used for a temperature sensor. If it reads a temperature above or equal to 20°C, it will print “It is hot outside”. However, if the temperature is not above 20°C it will move to the next statement, which checks to see if it is above 10°C. This “else if” creates a range to check if the temp is between 10°C and 20°C because in the initial if statement, it has determined the temp to be less than 20°C. Finally, if both conditions return a false, then it is determined the temperature is below 10°C because the first 2 conditions determined the temperature to not be above 20°C or 10°C and the statement in the else part of the code will print.

Note: An if...else if statement can have an unlimited amount of “else if” conditions but must end with an “else” condition or else the code will not run properly.

In general, an if statement with multiple clauses will have the following form:

```
if (condition)
{
    statement
}
else if (condition)
{
    statement
}
else
{
    statement
}
```

3.5 Using While Loops

While Loop: While loop in programming repeatedly executes a target statement inside the brackets if a given condition is true, until the condition becomes false. The false condition can come from your code, such as an incremented variable or a Boolean value, or an external condition, such as from a sensor.

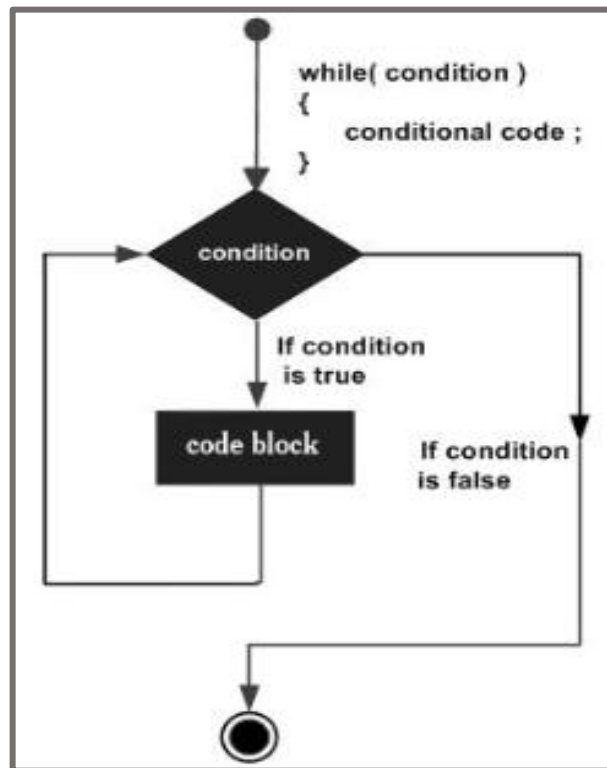


Figure 3-2. While loop flow chart.

In general, while loop will have the following form:

```

while (condition)
{
    statement
}
  
```

Simple While Statement Example

```
int count = 0
while (count < 10)
{
  rotateMotors()
  count += 1
}
```

In this example, we can read this code as while the count is less than 10, rotateMotors() works and add 1 to count variable until the count is equal 10.

Break: If you use break at the end of your condition; function will reach to the end of the loop.

Example of using a break

```
while (a < 10)
{
  if (a == 9)
  {
    break;
  }
  analogWrite(12, a);
}
```

This code can be read as follow:

While “a” is less than 10, the body functions are going to work which is to give an analog signal of variable a to port 12 if it is not equal to 9, because if a is equal to 9, the loop will break and end the loop.

Continue: Continue function will skip one loop of the whole, doesn't reach the end of the loop unless it worked when it was right before the end.

Example of using Continue

```
for (int count = 0; count < 20; ++count)
{
  if ((count % 4) == 0)
  {
    continue;
  }
  Serial.println(count);
}
```

This code can be read as follow:

Count variable is defined as 0, and it is increasing by 1 until it is greater than 20, and every change in count variable, it is going to print its count variable on the serial monitor, but not numbers that can be divided by 4, as if variable count integer divided by 4 is equal to 0, skip the loop.

Return: Return function can be used in any loop or when defining a function. It gives a return value of a loop or a function.

Example of using Return

```
int breakOrReturn() {
    while (True) {
        if ( a == 1) {
            break;
        }
        else if ( a == 2) {
            Return 1;
        }
    }
    Serial.println("We broke out")
    Return 0;
}
```

This example can be read as follow:

'BreakOrReturn' function is defined with a return type of integer. Part of this function, while loop is running until it is False. If "a" is equal to 1, then it will reach the end of the while loop and returns the integer value of 0. If "a" is equal to 2, the function is going to return the value of 1, and because it returns a value, the function is going to reach the end.

3.6 Using For Loops

For loop: A for loop is a repetition control structure that allows you to efficiently loop a condition that needs to be executed a specific number of times. An increment counter is commonly used to increment and eventually end the loop. The “for” statement is useful for repetitive operations, and is often used with arrays to operate on collections of data.

For loops are more efficient and effective than while loop. In for loops, you know the specific amount of times that the condition must be met. However, in a while loop, you have no idea when to stop the code. By having a specific amount, this allows the code to run efficiently. You can use while loops every time as it is simpler and easier to use than a for loop. However, the efficiency of your code will drop drastically.

In general, the for loop will have the following forms:

```
for (initialization; condition; increment)
{
    Statement(s)
}
```

The **initialization** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.

The **condition** is evaluated next. If it is true, “the statement block” or the body of the loop is executed. If it is false, the body of the loop does not perform, and the flow of control jumps to the next statement after for loop.

After the body of the for loop executes, the flow of control jumps back up to the increment statement to be increased then the **condition** is tested again. This statement allows you to update any loop control variables. When the **condition** becomes false, the loop ends.

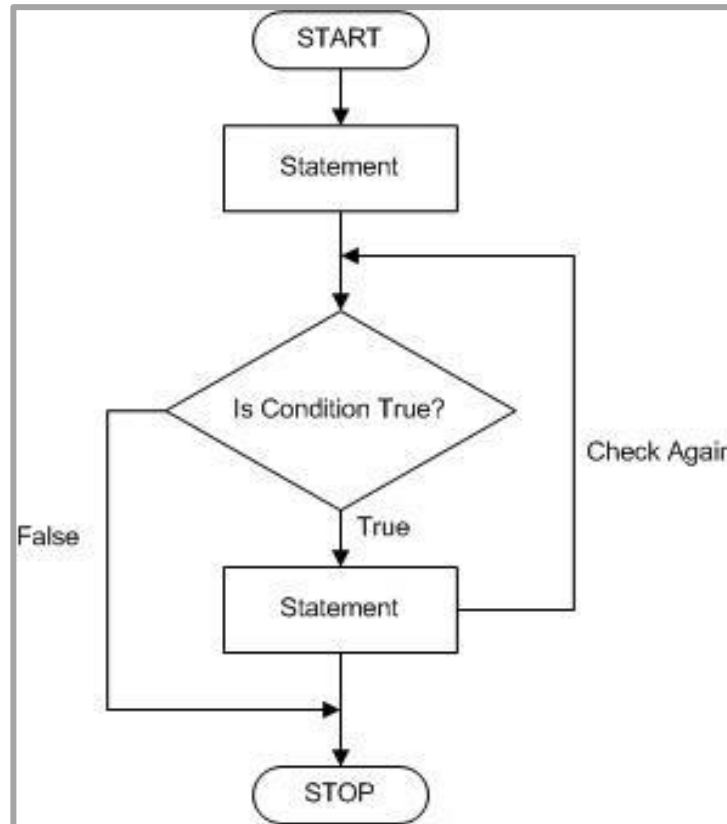


Figure 3-3. The flow diagram showing how a for loop works

Example of a For Loop

```

for (a = 10; a < 20; a = a + 1)
{
  analogWrite(motor1, a);
  delay(1000);
}
  
```

This example can be read as follow:

Motor named 'motor1' rotates the angle of the variable, "a," which is counted 10 to 19 increasing by 1 degree every second.

3.7 Nested Loops

Nested Loops are loops within a loop, an inner loop within the body of an outer loop.

In general, the nested loops will have the following form:

```
for (initialize; control; increment or decrement)
{
    // statement block
    for (initialize; control; increment or decrement)
    {
        // statement block
    }
}
```

Example of a Nested Loop

```
for (counter = 0; counter <= 9; counter++)
{
    // statement block will be executed 10 times
    for (i = 0; i <= 99; i++)
    {
        //statement block will be executed 100 times
    }
}
```

This example can be read as follow:

The counter is equal to 0. If the counter is less or equal than 9, then the counter will add 1 to itself and the statement block will be executed ten times, because the counter goes from 0 to 9. or the second loop, "i" is equal to 0. If "i" is less or equals to 99, then "i" will add 1 to itself and statement block will be executed 100 times. As the counter starts from 0 and goes all the way to 99.

3.8 Defining Functions

Functions are segments of code which allows a programmer to create modular pieces of code to perform a defined task and returns to the area of code from which the function was “called.” The typical case for creating a function is when one needs to perform the same action multiple times in a program. Functions also help the programmers stay organized, and they make the code run more smoothly and efficiently.

There are two required functions in an Arduino sketch, the setup() and loop(). Other functions must be created outside the brackets of these two functions.

Before a function can be used in a sketch, it must be created. The following code is an example of a function that was created to print a dashed line in the Arduino IDE.

```
Void DashedLine()
{
    Serial.println("-----");
}
```

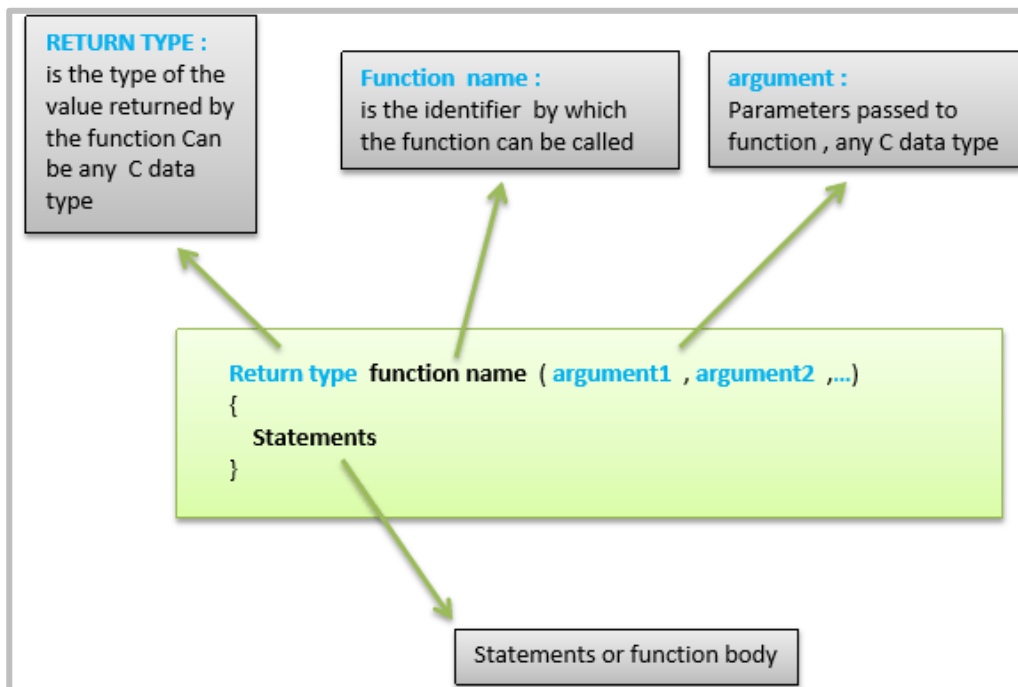


Figure 3-4. Structural Diagram showing the common syntax for creating a function.

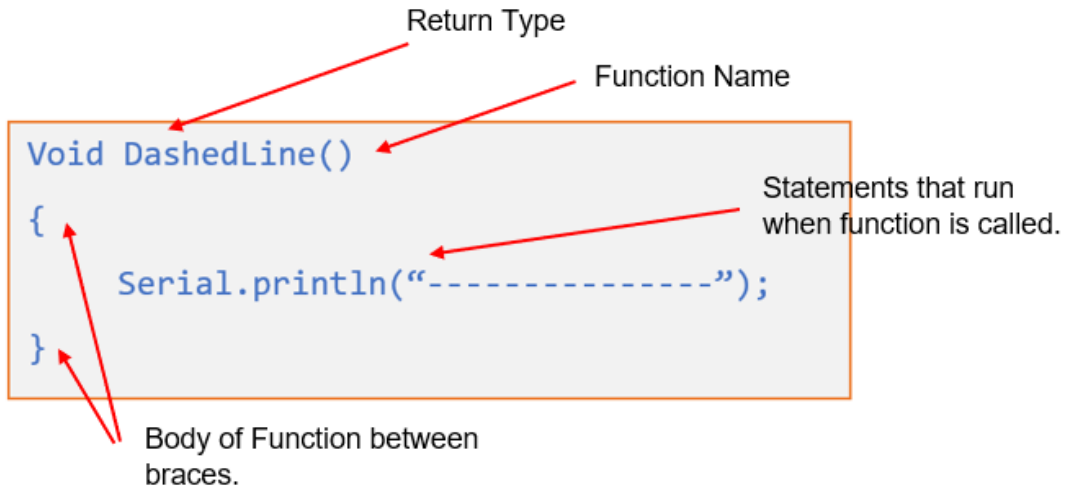


Figure 3-5. Components of a simple Arduino Function

“Void” is a Return type.

“DashedLine” is a Function Name.

“Serial.println” is a statement that runs when the function is called.

When creating a function, it must be given a name and ends with parentheses ().

A function must have a return type. The example function does not return anything, so has a return type of void.

The function body is made up of statements placed between braces {}. The statements make up the functionality of the function (what the function will do when it is called).

When a function is used, it is said to be "called". To use the function that we have created, it must be called in a sketch.

To call a function, use the function name followed by opening and closing parentheses.

The code example below shows calling and using the function made:

```

Void setup()
{
  Serial.begin(9600);
  DashedLine();
  Serial.println("| Program menu |");
  DashedLine();
}

Void loop()
{
}

Void DashedLine()
{
  Serial.println("-----");
}

```

The function above should print this:

```

-----
| Program Menu |
-----

```

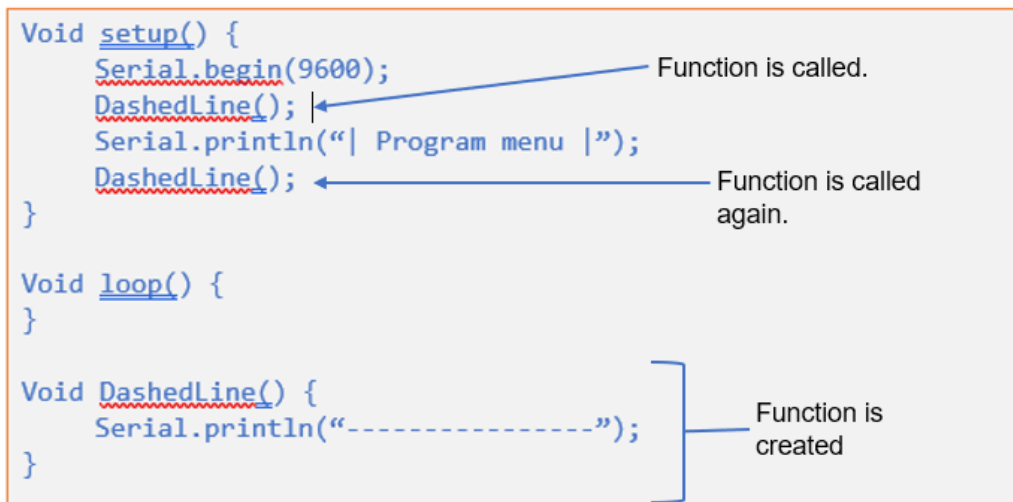


Figure 3-6. Calling a function in Arduino.

3.9 Mapping Function

The **map** command is a built-in function used for the conversion of values into a proportional value set by the ratio between the inputted range of values. by using the **map** function, there is no longer a need to manually calculate a conversion ratio as the function does it for you.

The function has 5 components, first is the value being converted, the next 2 values are the original range in values, from smallest to largest value. the last two components are the range of values you want the original value converted to, smallest to largest.

```
{  
  map(value, fromLow, fromHigh, toLow, toHigh)  
}
```

Below is an example of calling and using the map function

```
void setup() {  
}  
void loop() {  
  int val = analogRead(0);  
  val = map(val, 0, 1023, 0, 255);  
  analogWrite(9, val);  
}
```

CHAPTER 4: INTRODUCTION TO ARDUINO BASICS

4.1 Parts You Will Learn

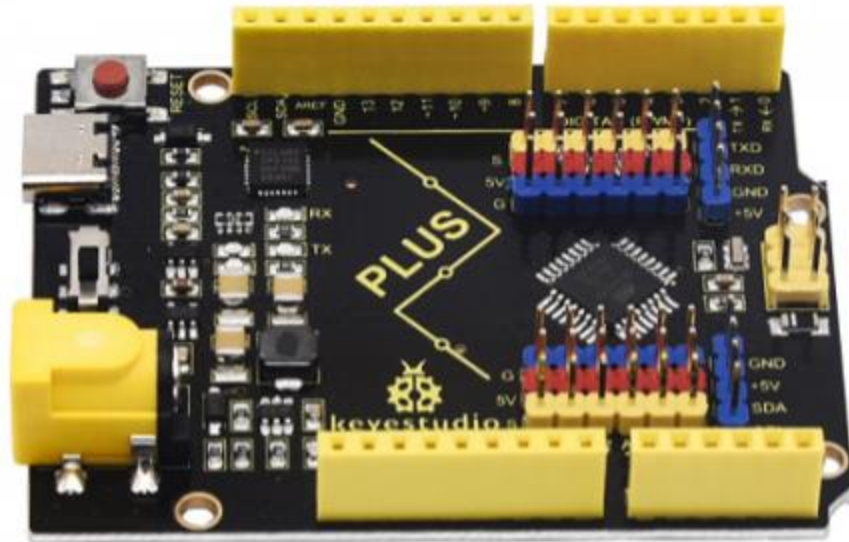
In this part of the chapter, you will learn about the basic parts Arduino has to offer. We will study the Arduino UNO board because it is the most popular board in the Arduino board family.

The Arduino UNO board is a microcontroller that can create many cool projects such as full robotic arms, or an autonomous car. The advantages of Arduino include it being inexpensive, it is cross-platform (multiple people can work on it) and it is open source (source code is made freely available and may be redistributed and modified so you can modify and build upon another person's code).

Some boards look a bit different from the one given below, but most Arduinos have most of these components in common. It has 14 digital input/outputs pins, six analog inputs, a 16MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; directly connect it to the computer with a USB cable or power it with a battery to get started. Think of the Arduino UNO as the brains of your machine and your operations.

4.1.1 Arduino UNO



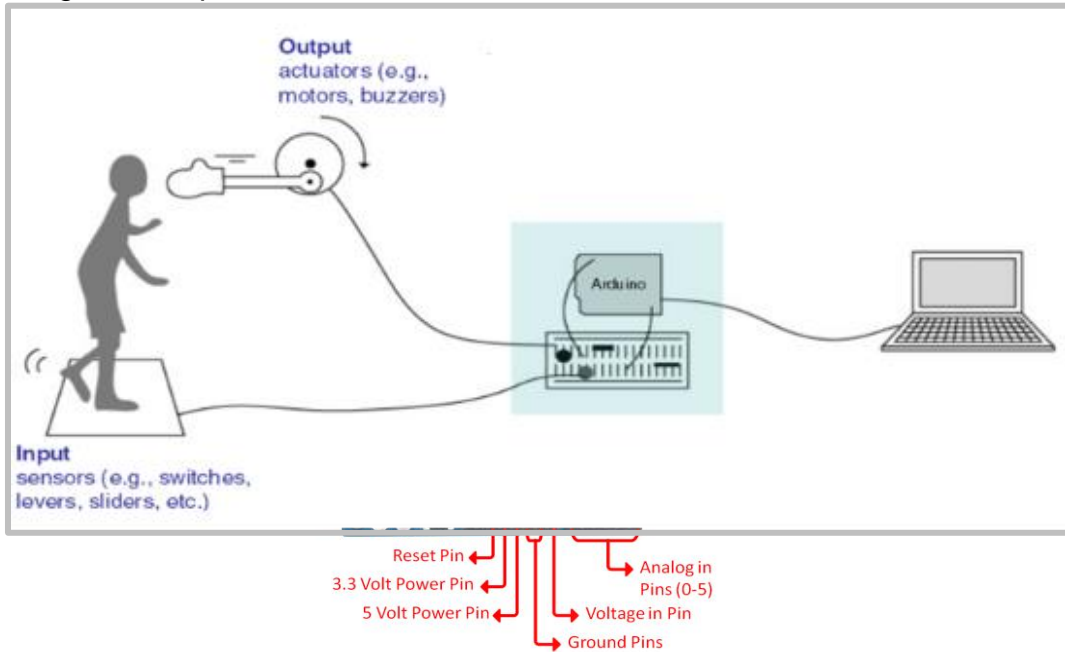


Benefits of using the Arduino Uno Plus:

- More stable USB serial chip
- 3.3V or 5V can be connected to 3.3V sensors
- More IO ports
- Extended Serial communication
- Special DC-DC design that can drive high current loads
- Input voltage of 6-15V
- Type-C interface

I/O stands for Input and Output

Arduino can take some data from INPUTs and decided what to do using its OUTPUTs. There are also two types of Outputs in Arduino; one is digital and the second one is analog output. Compare them with human input. Eyes, ears, taste and feeling are inputs to our brain. Hands and legs are outputs.



Arduino UNO Input and Output Limits:

<i>INPUT</i>	<i>Limits</i>
Input Voltage via USB cable	5.0V
Input Voltage via Barrel Jack	7.0 - 12.0V
Input Voltage via Vin pins	6.0 - 12.0V
Maximum Input Current protected by fuse	500mA

<i>OUTPUT</i>	<i>Limits</i>
Output Voltage controlled by voltage regulator	5.0V / 3.3V
Max. Current for any I/O pin	40mA
Maximum Total Current from all I/O pins together	200mA
Maximum Current from V_{CC} (5V output pin)	~400mA on USB

This Arduino UNO board chart will give you all the necessary information needed for you to work on future projects. This chart contains different kinds of information.

Arduino Uno Information Table			
Pinouts		Physical Specs	
Price Points	\$17.99-\$22.00	DC Current for 2.3V Pin	50 mA
USB Connectivity	Standard A/B USB	SRAM	2 KB
Voltage Level	5V	EEPROM	1 KB
Digital I/O Pins	14 (of which 6 provides PWM output)	Clock Speed	16 MHz
Ethernet/Bluetooth	No(a Shield/module can enable it)	Length	66.6 mm
Analog Input Pins	6	Width	52.4 mm
DC Current per I/O Pin	20 mA	Flash Memory	32
Processor	ATmega328P	Shield Compatibility	Yes

4.1.2 Arduino MEGA

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; just connect it to a computer with a USB cable or power it with an AC to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

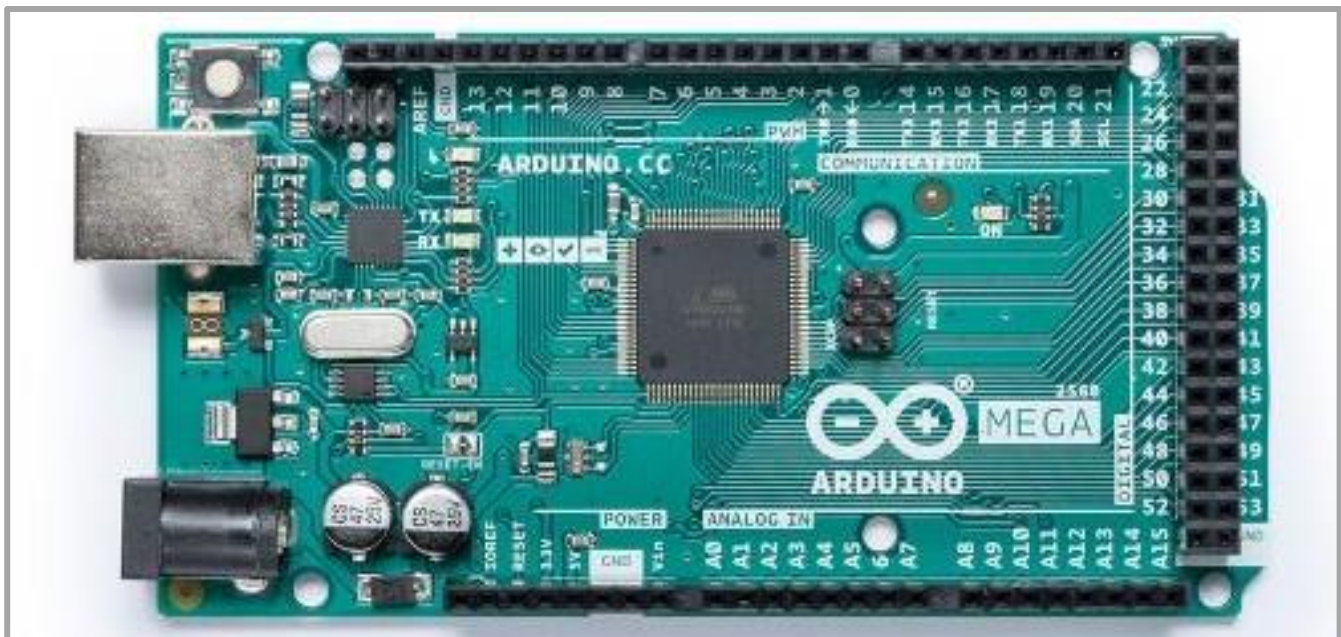
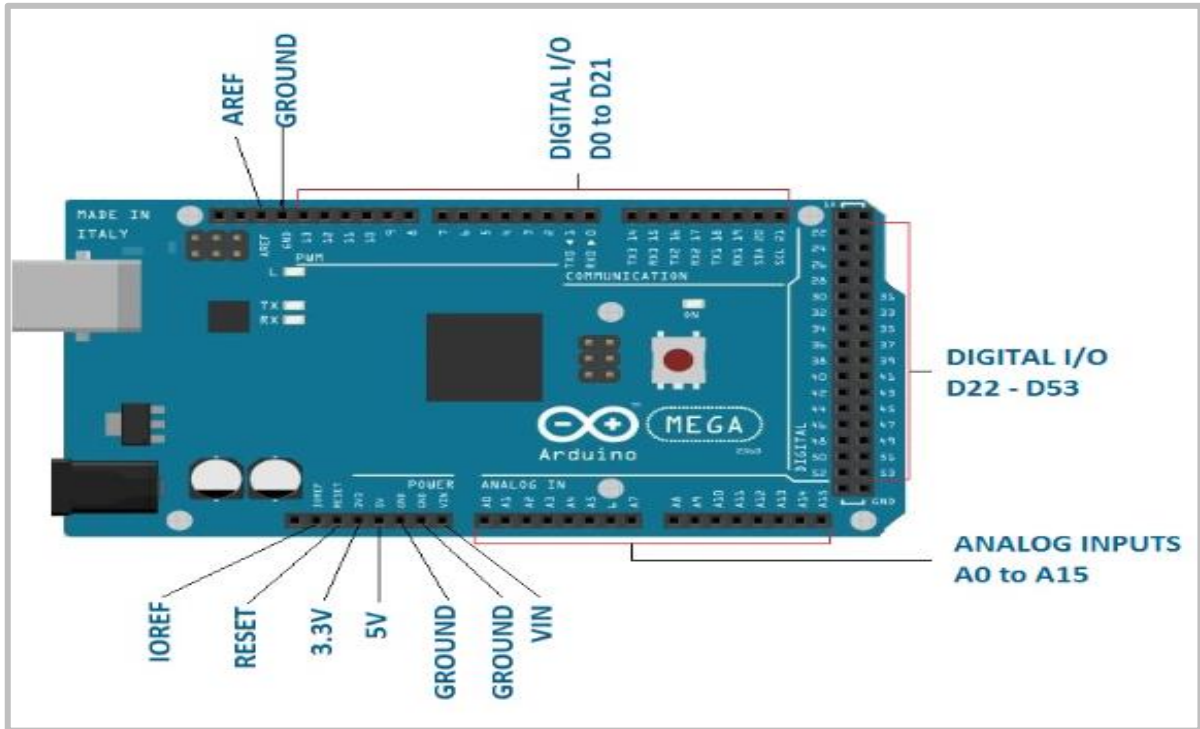


Figure 4.4. The Arduino Mega Board.



This Arduino MEGA board chart will give you all the necessary information needed for you to work on future projects.

Arduino Mega Information Table			
Pinouts		Physical Characteristics	
Price Points	\$36.61 - \$37.00	DC Current for 2.3V Pin	50 mA
USB Connectivity	Standard A/B USB	SRAM	8 KB
Voltage Level	5V	EEPROM	4 KB
Digital I/O Pins	54 (of which 15 provides PWM output)	Clock Speed	16 MHz
Ethernet/Bluetooth	No (a Shield/module can enable it)	Length	101.6 mm
Analog Input Pins	16	Width	52.4 mm

DC Current per I/O Pin	20 mA	Flash Memory	256
Processor	ATmega2560	Shield Compatibility	Yes

4.1.3 Arduino NANO

Arduino NANO is very similar to Arduino UNO, they use the same processor (ATmega328P). However, the main difference between these two would be their size. Arduino NANO is much smaller than Arduino UNO. This can give an advantage and a disadvantage depending on the situation. Arduino NANO is much easier to use as it can fit anywhere. Its size allows it to be more flexible in terms of structures and builds. Arduino NANO is breadboard compatible. It lacks only a DC power jack and works with a Mini-B USB cable instead of a standard one.

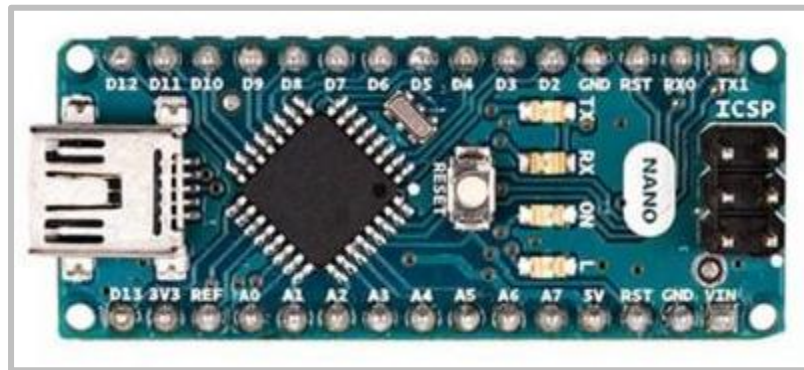
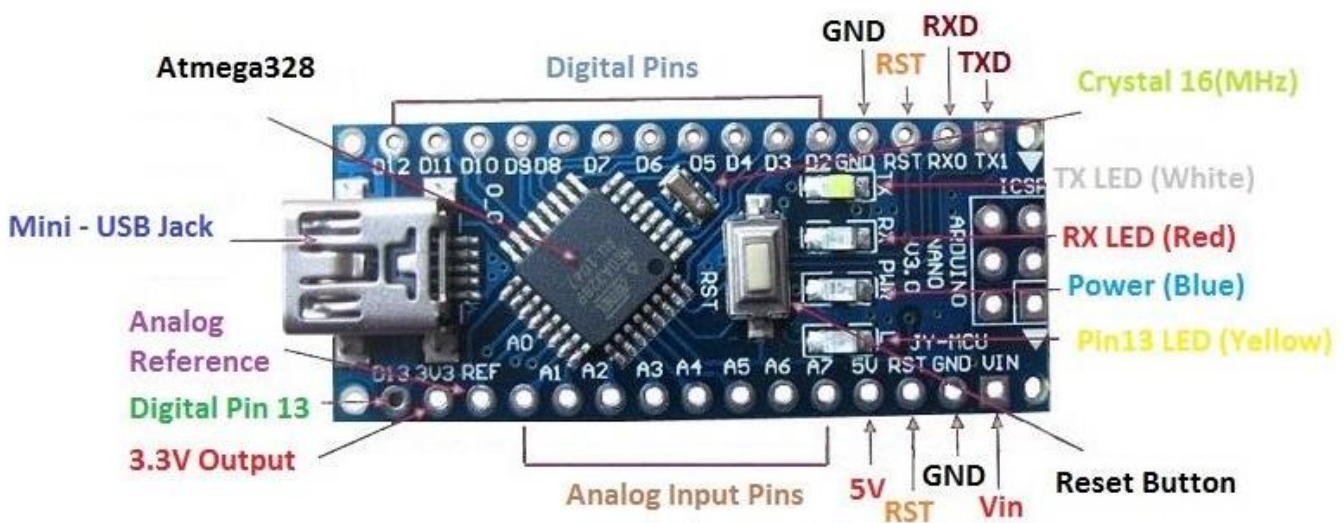


Figure 4-6. An Arduino NANO.



Arduino NANO Information Table			
Pinouts		Physical Characteristics	
Analog Pins	A0 – A7	DC Current for 3.3V Pin	50 mA
Analog Input Pins	A0 – A5	SRAM	2 KB
SPI	10 (SS), 11 (MOSI), 12 (MISO) and 13 (SCK)	EEPROM	1 KB
Digital I/O Pins	14 (Out of which 6 provide PMW output)	Clock Speed	16 MHz
External Interrupts	2, 3	Recommended Input Voltage for Vin pin	7-12V
Inbuilt LED	13	Operating Voltage	5V
DC Current per I/O Pin	40 mA	Flash Memory	32 KB (2 KB is used for Bootloader)
PMW	3, 5, 6, 9, 11	Processor	ATmega2560

4.1.4 Breadboard

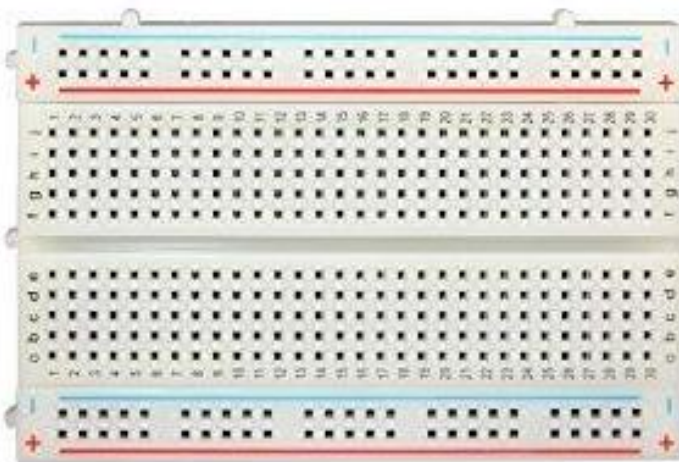


Figure 4-8. A breadboard top view.

A **breadboard** is a board used for making an experimental model of an electric circuit. It helps us build circuits by avoiding tedious and complicated wiring. It makes it easy to make circuits connections as the breadboard has internal connections, so we don't have to connect every one of our various components using wires physically or require soldering.

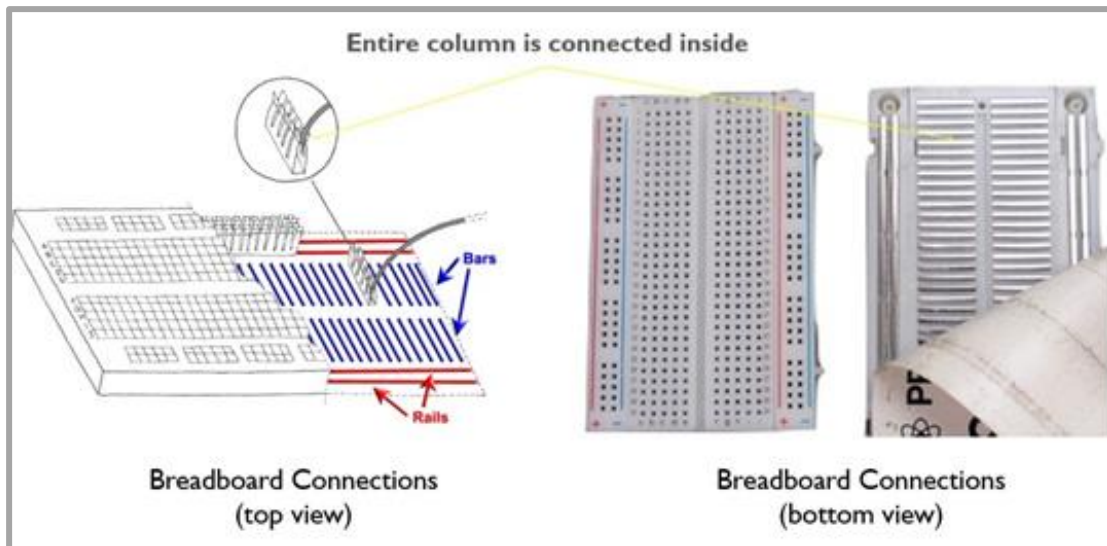


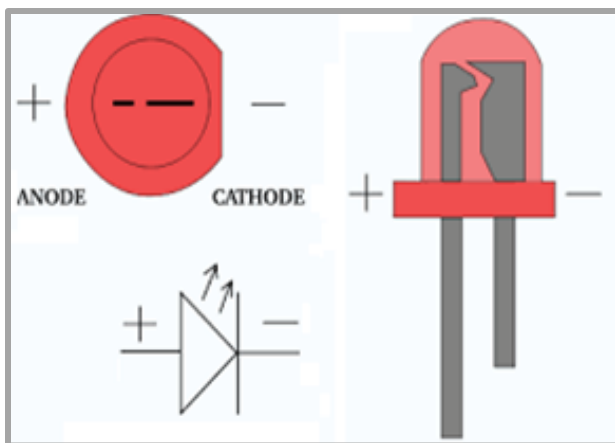
Figure 4-9. A breadboard with the internal wiring shown to see the connections.

A breadboard is connected like this image depicts it. You merely place the components and wires in their rows or columns, depending on how they should be connected. Be careful to not accidentally set the multiple pins of a single component in the same row. The bar rows of metal strips on the bottom of the breadboard have little clips that allow you to stick a wire or the leg of a component into the exposed holes on a breadboard, which then holds the component in place.

Once inserted that component will be electrically connected to anything else placed in that row. This is because the metal rows are conductive and allow current to flow from any point in that strip.

When building a circuit, you tend to need power in lots of different places. The power rails give you lots of easy access to power wherever you need it in your circuit. Usually they will be labeled with a '+' and a '-' and have a red, blue or black stripe, to indicate the positive and negative sides.

4.1.5 LED (Light-Emitting Diode)



A **light-emitting diode (LED)** is a semiconductor device that emits visible light when an electric current pass through it.

Figure 4-10. An LED and its circuit symbol.

Current passing through a diode can only go in one direction, called the forward direction. Current trying to flow the reverse direction is blocked.

If the voltage across a diode is negative, no current “Can flow”, and the ideal diode looks like an open circuit. In such a situation, the diode is said to be off or reverse biased. If the voltage across the diode isn’t negative, it’ll “turn on” and conduct current. Ideally, a diode would act like a short circuit (0V across it) if it was conducting current. When a diode is conducting current, it’s forward biased.

Every diode has **two terminals** -- connections on each end of the component -- and those terminals are **polarized**, meaning the two terminals are distinctly different. It's important not to

mix the connections on a diode up. The positive end of a diode is called the **anode**, and the negative end is called the **cathode**. Current can flow from the anode end to the cathode, but not the other direction. The short leg (cathode, or negative lead) always connects to ground.

Depending on the voltage applied across it, a diode will operate in one of three regions:

1. **Forward bias:** When the voltage across the diode is positive the diode is "on" and current can run through. The voltage should be greater than the forward voltage (V_F) for the current to be anything significant.
2. **Reverse bias:** This is the "off" mode of the diode, where the voltage is less than V_F but greater than Breakdown Voltage. In this mode current flow is (mostly) blocked, and the diode is off.
3. **Breakdown:** When the voltage applied across the diode is very large and negative, lots of current will be able to flow in the reverse direction, from cathode to anode.

What is Positive and Negative Voltage?

Negative voltage in a circuit is voltage that is more negative in polarity than the ground of the circuit.

A voltage source has positive or negative polarity depending on its orientation in a circuit. In the case when a voltage source has negative voltage, it just means the negative terminal of the battery is connected to the positive side of the circuit and the positive terminal of the battery is connected to the negative side of the circuit.

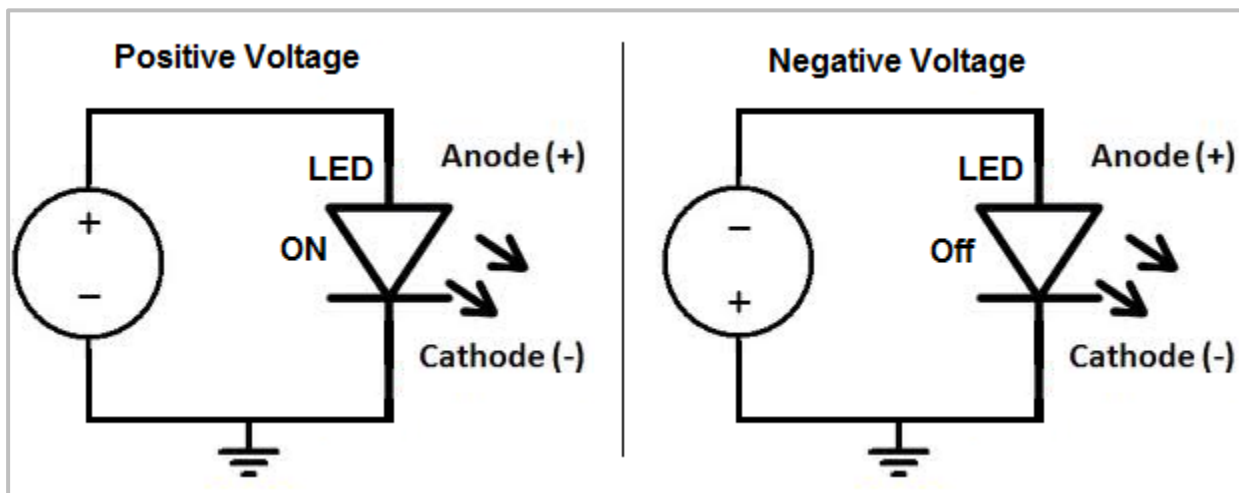


Figure 4-11. An illustration of positive and negative voltage. The first circuit is oriented one way and the other circuit is oriented the same way but with the voltage source flipped around, so that it has opposite polarity orientation than the first.

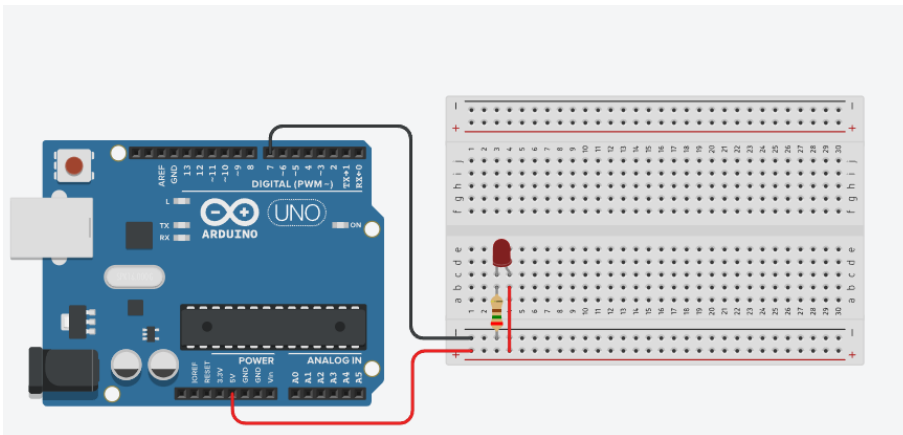
The first circuit is positive voltage because the positive end of the voltage is connected to the positive end of the source is not connected to ground.

The second circuit is the exact opposite, where the positive end of the voltage source is connected to ground, while the negative end is not. An LED is a device that only turns when the anode of the LED is supplied to positive voltage and the cathode with negative voltage. Therefore, only the first circuit's LED turns on, while the second circuit's LED doesn't.

If you need more information or you still have questions, you can visit the link below:

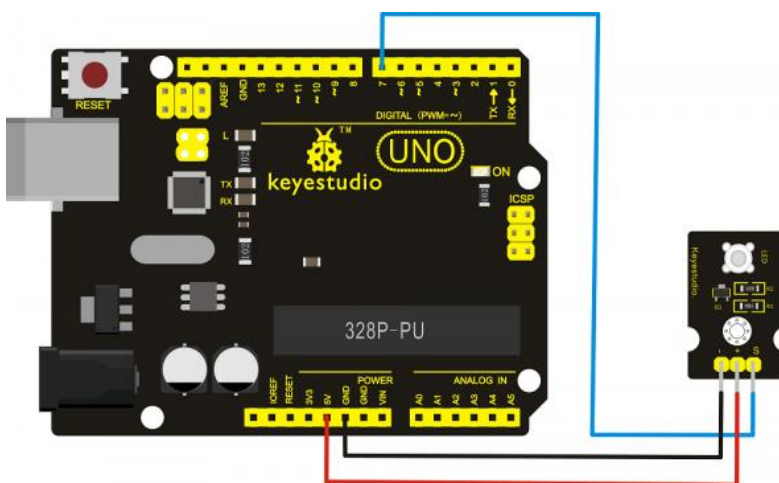
Reference: <https://learn.sparkfun.com/tutorials/diodes> & <http://www.learningaboutelectronics.com/Articles/What-is-negative-voltage>

2 Pin LED example:



<i>Led</i>	<i>Arduino</i>
+	5V
-	7

3 Pin LED example:



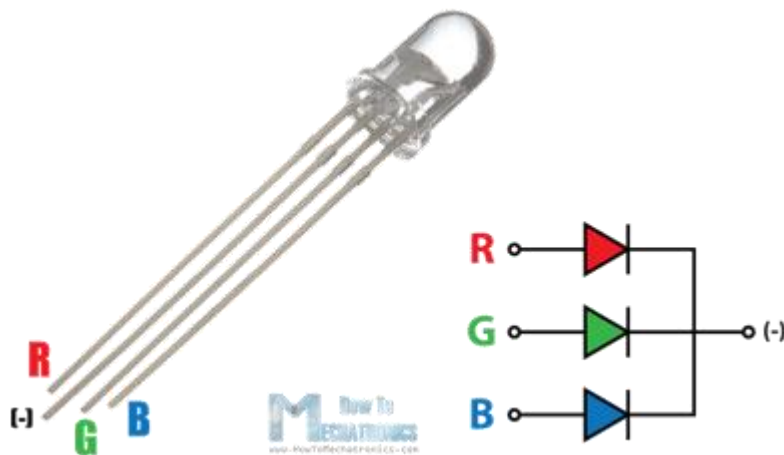
<i>Led</i>	<i>Arduino</i>
+	5V
-	GND
S	8

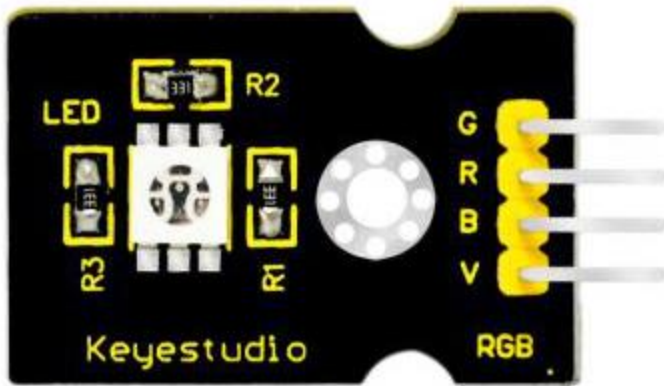
LED Blink Code Example:

```
void setup() {
  pinMode(8, OUTPUT); //Set pin 8 as an output
}

void loop(){
  digitalWrite(8, HIGH); //Turn on the LED
  delay(1000);
  digitalWrite(8, LOW); //Turn off the LED
}
```

4.1.6 RGB LED

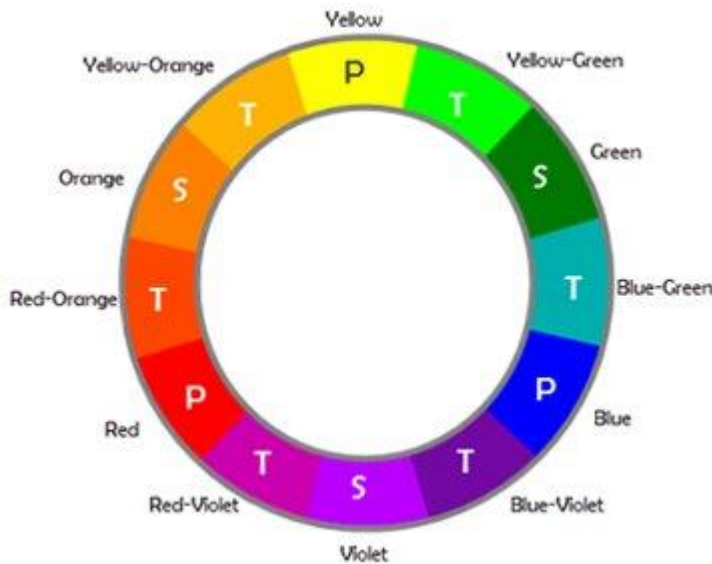




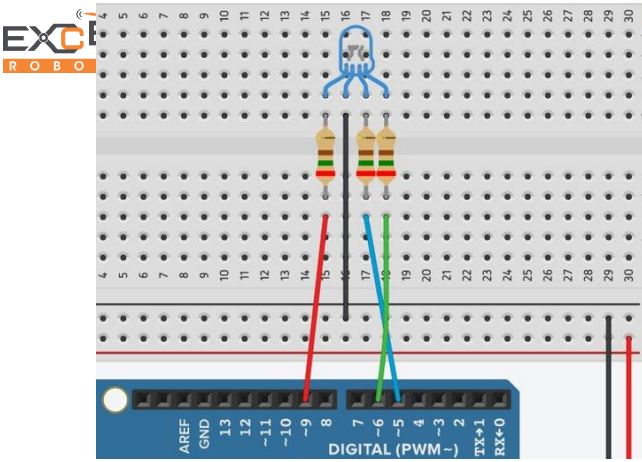
A **Red-Green-Blue light-emitting diode (RGB LED)** is a combination of 3 LED'S in one component. By sending Negative Voltage through the Red, Green and Blue cathods, the colors mix based on the amount of power given creating different colours and hues

Just like a regular LED, an RGB LED follows the same rules with one difference. The RGB LED has multiple inputs so unlike a regular LED where all that can be controlled is the brightness, the RGB LED allows for control over the color and hue of the omitted light.

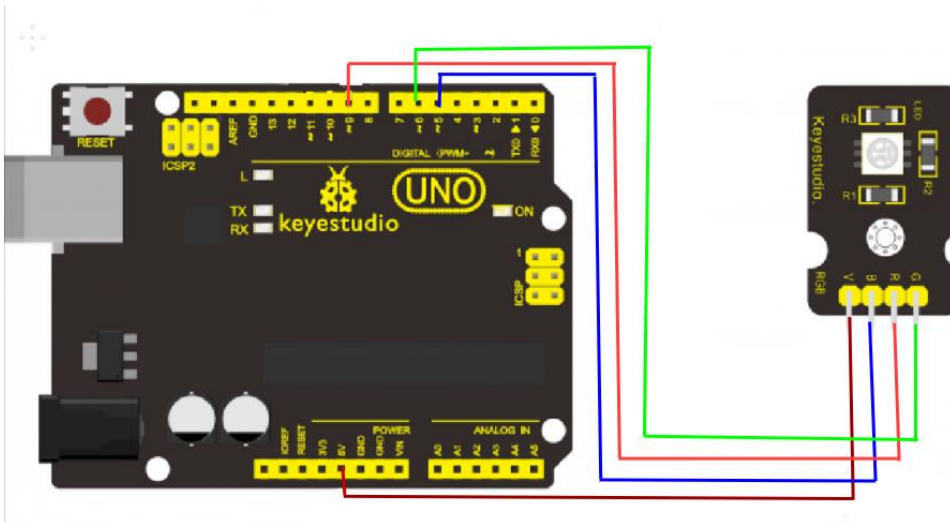
A RGB LED has four terminals, three Cathodes and one Anode. The Anode will be the longest of the four terminals and must be connected to ground. The three cathode terminals represent the primary colors, Red, Green and Blue. Sending Negative Voltage through each of their respective Cathodes will produce their color. they can be identified by their position in relation to the Anode.



By controlling the intensity of each of the 3 primary colors through a microcontroller, any color on the spectrum can be created



<i>RGB Led</i>	<i>Arduino</i>
R	9
-	GND
G	6
B	5



<i>RGB Led</i>	<i>Arduino</i>
V	5V
R	9
G	6
B	5

RGB Code example:

```
void setup() {
  pinMode(9, OUTPUT); //set pin 9 as output (red)
  pinMode(5, OUTPUT); //set pin 5 as output (blue)
  pinMode(6, OUTPUT); //set pin 6 as output (green)
}

void loop(){
```

```
for(int val = 0; val < 255; val++){
  analogWrite(6, val); //set green value to val
  analogWrite(9, 255-val); // set red value to 255 - val
  analogWrite(5, 128-val); //set blue value to 128 - val
  delay(1);
}
}
```

4.1.7 Jumper Wires



Figure 4-12. Multi-coloured jumper wires.

Jumper wires are wires you use when you build prototype circuit to try out new concepts. They may be short pieces of stiff wire as shown in the picture, or they may be a flexible wire with pins on either end. They connect the circuit, allowing the electricity to flow through to different parts of the Arduino.

As you can see, there are three types of wires in Arduino. There are Male-Male wires (left-side bunch), Female-Female wires (the middle bunch) and Male-Female wires (right-side bunch). Each of them plays a specific role when creating an Arduino circuit.



Figure 4-13. Jumper Wires

Jumper wires typically come in three versions: male-to-male, male-to-female and female-to-female. The difference between each of the different wires are their endpoints of the wire. Male ends have a pin that can plug into things, while female ends do not. The female wires are used to plug things into. Male-to-male jumper wires are the most common. When connecting two ports on a breadboard, a male-to-male wire is what you will need.

4.1.8 Resistors

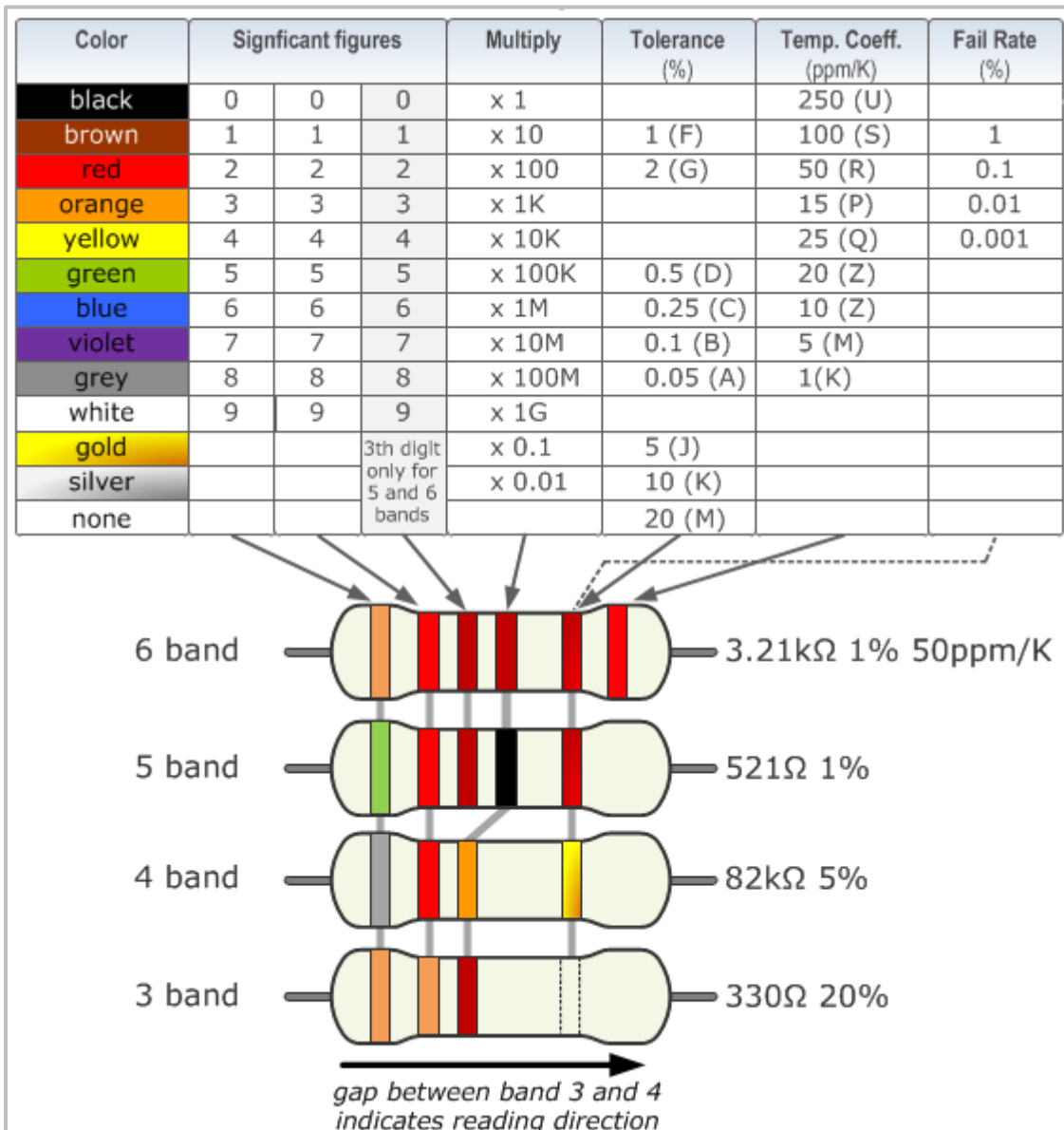
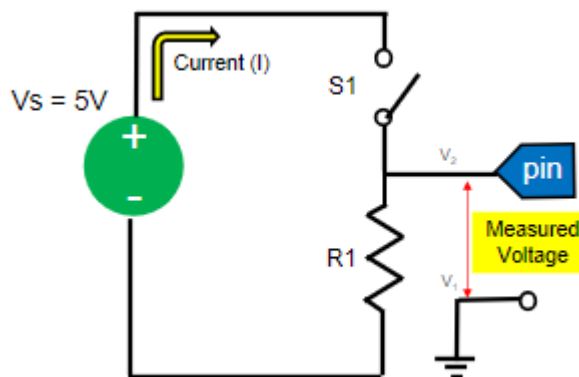


Figure 4-14. Resistors are electronic components which have a specific, never-changing electrical resistance. The resistor's resistance limits the flow of electrons through a circuit.

Note: The chart above and images of resistors shown can be used as a guide to learn how to calculate the resistance of any resistor based on the band colours.

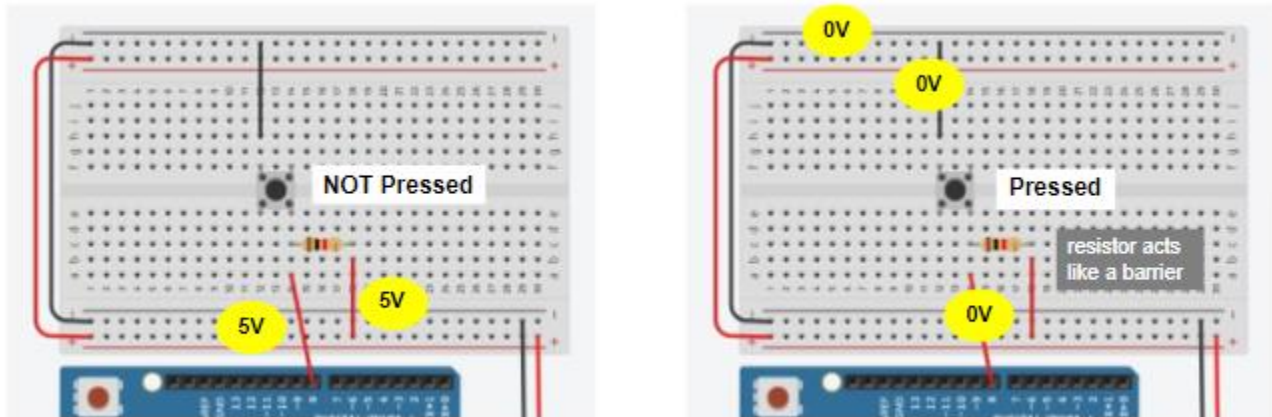
Resistors are passive components, meaning they only consume power (and can't generate it). Resistors are usually added to circuits where they complement active components like microcontrollers, and other integrated circuits. Commonly resistors are used to limit current, divide voltages, and pull-up I/O lines.

4.1.9 Pull-Down Resistors



A Pull-Down Resistor ensures a known state for a signal. This is needed due to the pin being vulnerable to electrical noise (outside electrical interference) which can result in a change of state due to the pin receiving a signal. This is prevented by R1 so that even when the circuit is open, any electrical interference is grounded through the resistor. A resistor is used instead of a straight to ground wire because the electrons will flow straight to ground when the button is pressed and not flow to the pin at all creating a short circuit, with a resistor the electrons choose the path of least resistance allowing power to flow to the pin when the button is pressed.

4.1.10 Pull-Up Resistors



A Pull-up resistor just like a Pull-Down resistor is a resistor that ensures a known state for a signal. When the switch is open, then the circuit disconnects, and the current is not able to flow through the circuit. When the switch is closed, the circuit is connected, and current can flow. For a switch that connects to ground, a pull-up resistor ensures a well-defined voltage (i.e. V_{CC} , or logical high) across the remainder of the circuit when the switch is open.

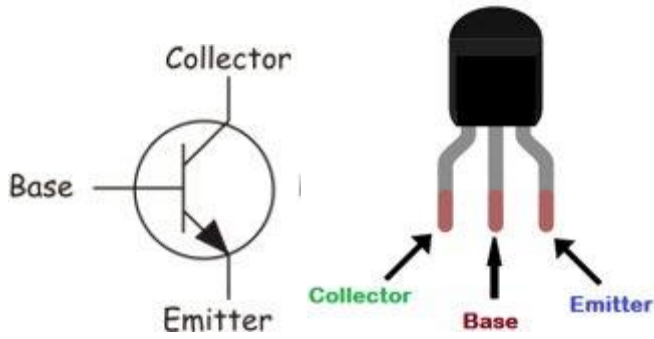
4.1.11 Transistors

A **Transistor** is a semiconductor device used to **amplify or switch electronic signals and electrical power**. Think of it as an electric switch. A **small current** at the Base/Gate terminal **can control or switch a much larger current** between the Collector/Source and Emitter/Drain terminals.

Types Of Transistors

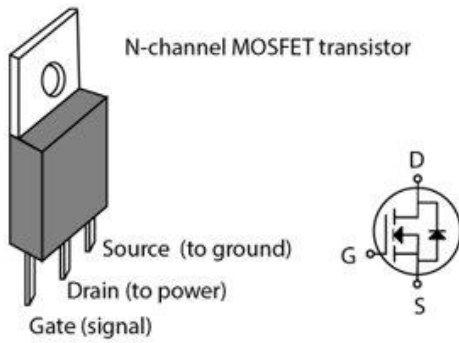
Bipolar Junction Transistor (BJT)

A **BJT** transistor has three legs: **Base (control)**, **Collector (supply +)**, **Emitter (supply -)**

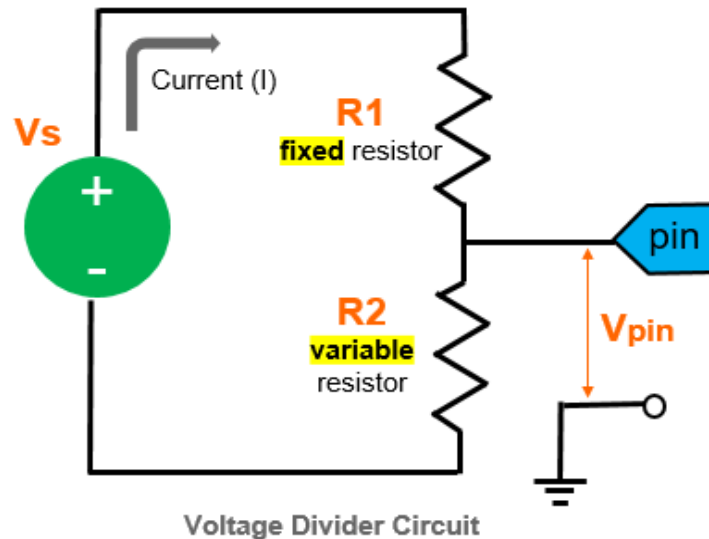


Metal-Oxide Field-Effect Transistor (**MOSFET**)

A **MOSFET** transistor has three legs: **Gate (control)**, **Source (supply +)**, **Drain (supply -)**



4.1.12 Voltage Divider



A Voltage Divider is a circuit designed to divide the drop in voltage across two resistors, this circuit is made by having two resistors be in series in a closed loop. The purpose of a Voltage Divider is to be able to determine the voltage drop across the second resistor. If R_1 were to be removed from the above diagram, then the voltage drop would be across only a single resistor and would always be the same value as V_s . No matter what the resistance of the resistor was, the voltage drop would always be the same and only the current would be different. By adding a fixed resistor (R_1) before the variable resistor (R_2), it is now possible to determine the varying voltage drops of R_2 thanks to Kirchhoff's Voltage Law (KVL). By following KVL an equation can be used to determine the voltage drop of R_2

$$V_{pin} = V_s \cdot \frac{R_2}{R_1 + R_2}$$

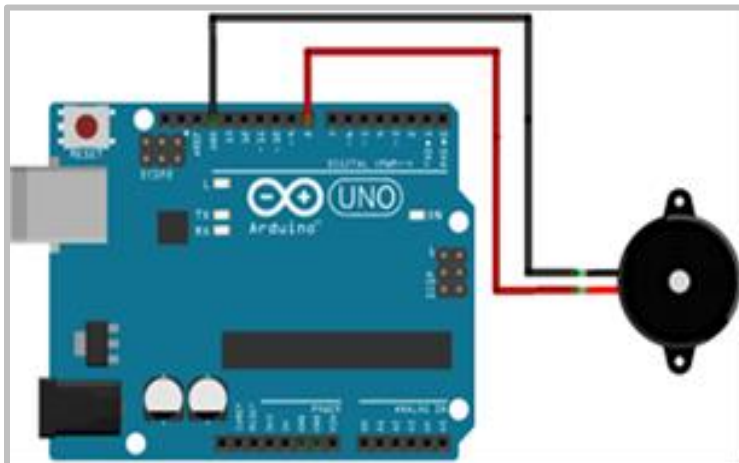
By controlling the value of R_1 , the range of values read by the Arduino can be adjusted, which allows for the voltage divider to be used for a control circuit with fairly precise control of voltage.

4.1.13 Buzzers



A **buzzer** or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric (piezo for short). Typical uses of buzzers include alarm devices, timers, and confirmation of user input such as a mouse click or keystroke. It is an output device that takes the digital pulse and converts it to a monotone sound by vibrating.

2 Pin Buzzer Example:



<i>Buzzer</i>	<i>Arduino Board</i>
+	8
-	GND

3 Pin Buzzer Example:

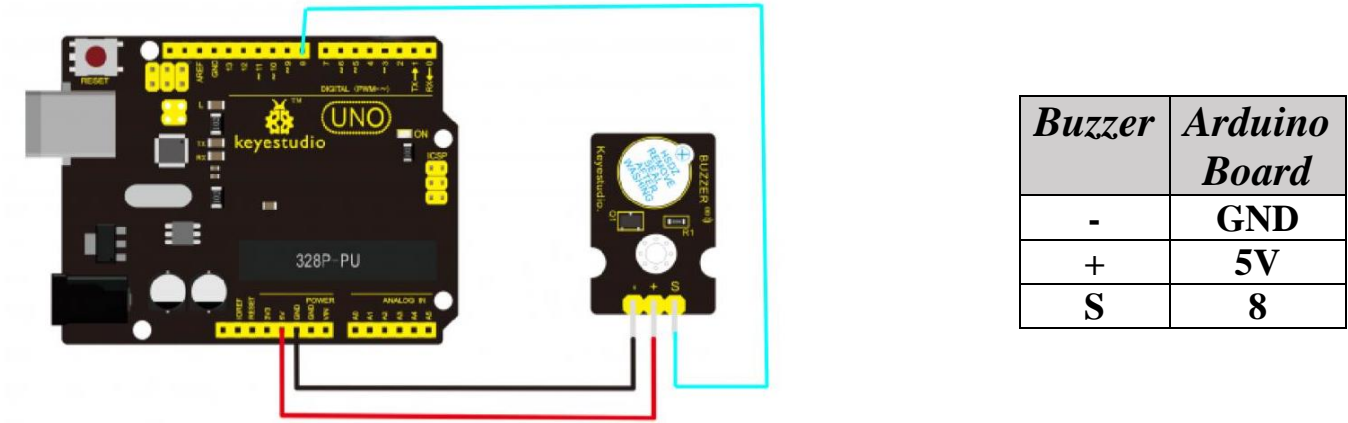


Figure 4-16. Wiring Diagram and Pin Connections for a simple buzzer.

Sample Code to Program a Buzzer:

```
const int buzzer = 8; // Set the buzzer as pin 8

void setup()
{
  pinMode(buzzer, OUTPUT); // Set pin 8 as an Output
}

void loop()
{
  tone(buzzer, 440); // Make noise for the buzzer at pin 8
  delay(1000); // Wait for 1000 millisecond(s)
  noTone(buzzer); // Stop the noise for the buzzer at pin 8
  delay(1000); // Wait for 1000 millisecond(s)
}
```

4.2 Buzzer Library

Functions to sound a Buzzer:

`Tone (pin#, Frequency)`

This function generates a square wave of specified frequency (and 50% duty cycle) on a pin. A duration can be specified. Otherwise, the wave continues until a call to:

`noTone(pin#)`

Library name:

```
#include <NewTone.h>
```

Declaration:

None

Functions:

```
NewTone( pinNumber, frequency);  
NewTone( pinNumber, frequency, Length);  
noNewTone( pinNumber);
```

If you need more information or you still have questions, you can visit the link below:

Reference: <https://bitbucket.org/teckel12/arduino-new-tone/wiki/Home>

4.3 Building Your First LED Circuit

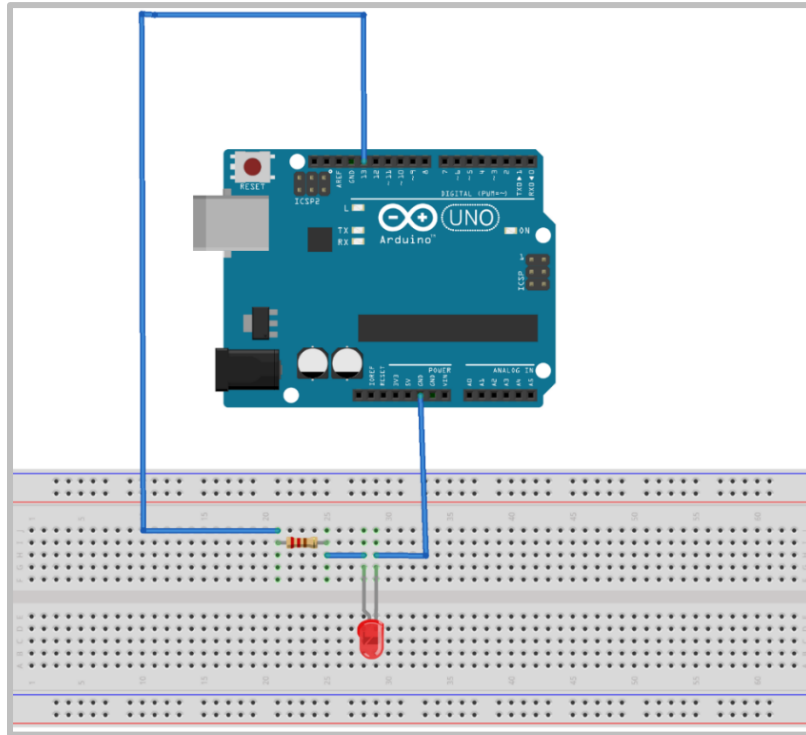


Figure 4-17. A simple circuit with an LED.

This diagram shows a basic circuit to light up an LED. Go on TinkerCad and try building the circuit yourself. Then use the sample code below to program it and experiment with the code itself to see how you can change it.

Sample Code

```
int led = 13; // the pin the LED is connected to

void setup()
{
  pinMode(led, OUTPUT); // Declare the LED as an output
}

void loop()
{
  digitalWrite(led, HIGH); // Turn the LED on
}
```

4.4 Code Breaker Challenge

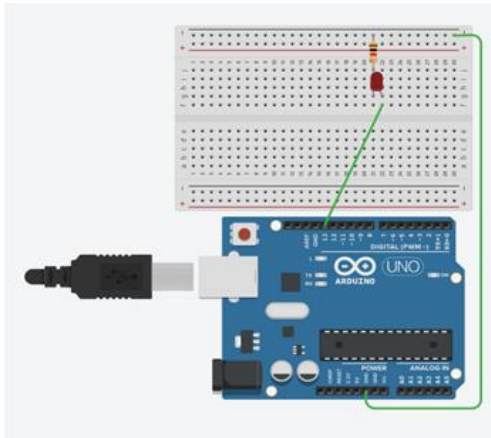


Figure 4-18. Simple LED circuit.

1. Blinking LED

Build the code for this circuit, which makes the LED blink every second.

2. Blinking Buzzer

Build the code for the circuit which makes the buzzer make a sound every second.

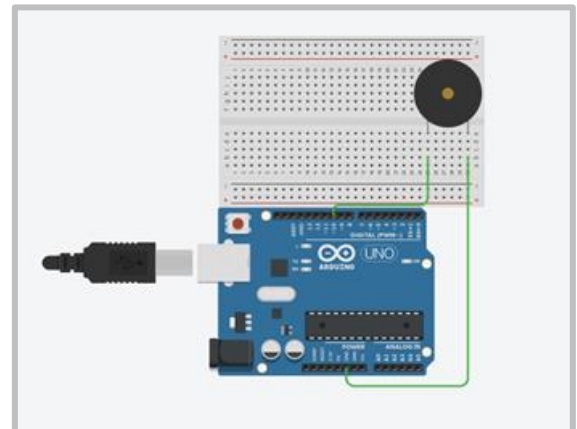


Figure 4-19. Simple Buzzer circuit.

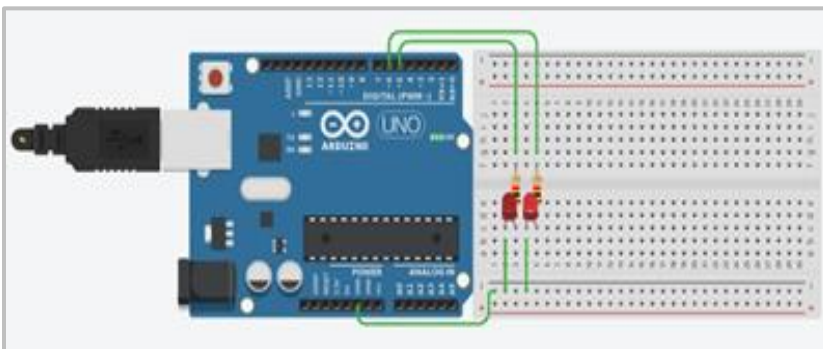


Figure 4-20. Simple Buzzer circuit.

3. Flip flop LED

Build the code for the circuit above which the LED is alternately turned on and off using this circuit.

4. Siren

Build the code which the red and blue LEDs are alternately blinking, and buzzer makes the siren noise using this circuit.

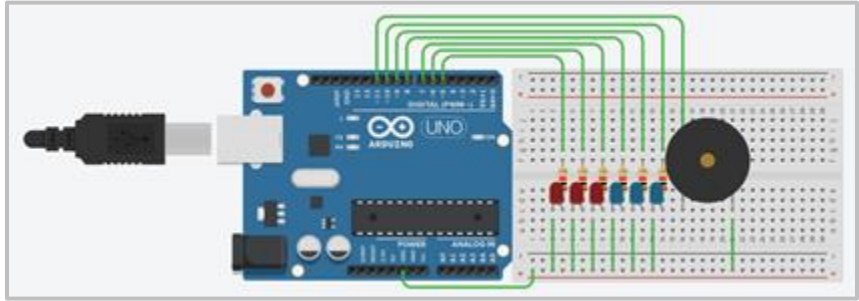


Figure 4-21. An LED and Buzzer combined circuit.

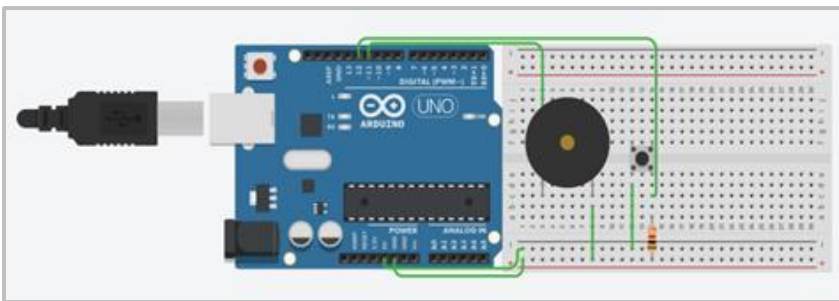


Figure 4-22. Buzzer circuit.

5. Morse Code Sender

Build the code which the buzzer works with 523Hz when the pushbutton is pressed using this circuit above.

4.5 Homework Questions

Q1.

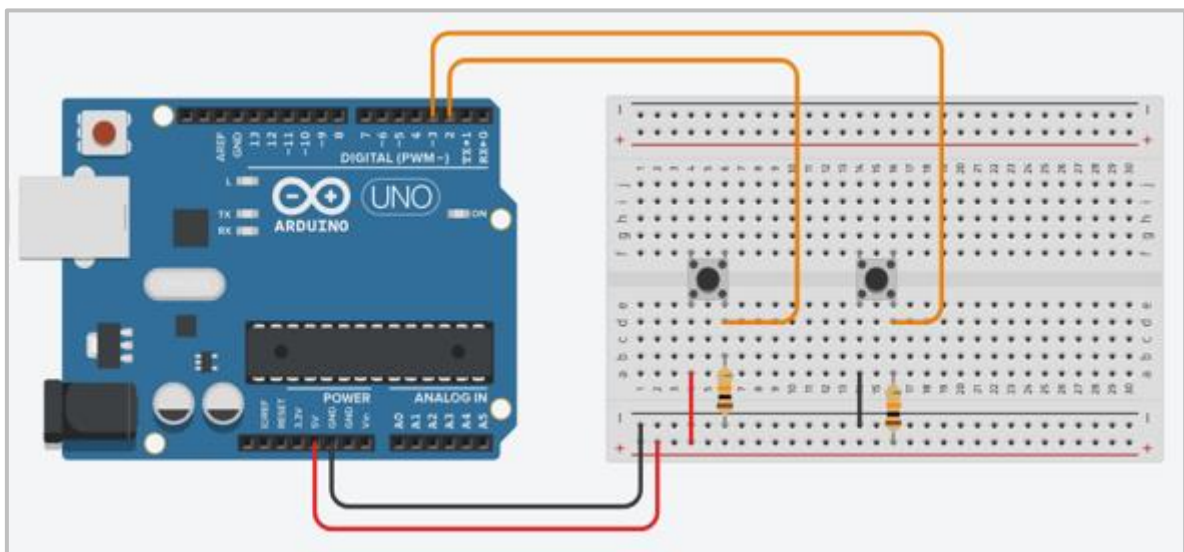


Figure 4-23. A two Buzzer circuit.

```

1 void setup()
2 {
3   pinMode(2, INPUT); //sets pin 2 to input mode
4   pinMode(3, INPUT); //sets pin 3 to input mode
5
6   Serial.begin(9600); //sets up serial monitor
7
8 void loop()
9 {
10  bool button1 = digitalRead(2); //reads digital information on 2
11  bool button2 = digitalRead(3); //reads digital information on 3
12
13
14  Serial.print("b1: "); //prints to serial monitor
15  Serial.print(button1); //prints to serial monitor
16  Serial.print(" , b2: "); //prints to serial monitor
17  Serial.println(button2); //prints to serial monitor
18
19  delay(10); //delay to increase performance of simulator
20 }

```

Figure 4-24. The code for the Arduino Circuit Above.

Construct the following circuit.

Using the Serial monitor output the readings from the digital pins connected to the buttons.

Note: the serial monitor is used for the Arduino to communicate with the computer. It can also be used to display information on the computer screen.

To initialize the serial monitor, use (put in the setup):

`Serial.begin(9600);`

At any point to output information to the screen use: `Serial.println(" ");`

Compare the results from the serial monitor to the circuit design. Can you explain what happens?

Q2. Create Your Own Working Arduino Piano!

The project should include:

- 1 Arduino
- Eight push buttons
- Eight resistors
- One buzzer
- One breadboard

Reminder: To use the buzzer you can make use of the built-in function's 'tone' and 'noTone.'

To turn the buzzer on:

```
tone(pin, frequency); //pin refers to where the buzzer is a plugin, frequency
refers to the sound
```

To turn the buzzer off:

```
noTone(pin);
```

Useful information: To produce a simple scale on a piano, each key must be tuned correctly.

Frequencies for the c Major scale are listed below:

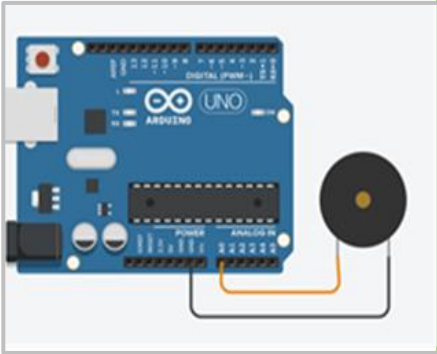
Note	Frequency	Simple Buzzer Circuit
c	261 Hz	
d	294 Hz	
e	329 Hz	
f	349 Hz	
g	392 Hz	
a	440 Hz	
b	493 Hz	
C	523 Hz	

Figure 4-25. Buzzer Circuit

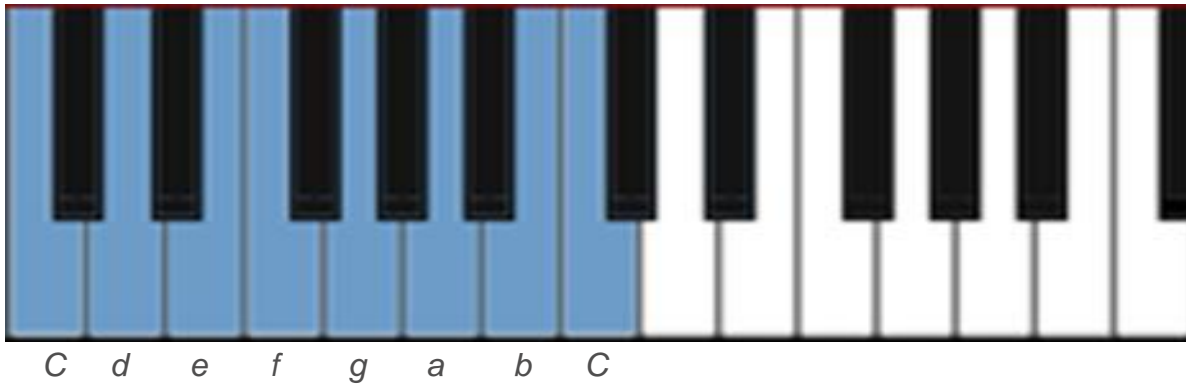


Figure 4-27. The layout of note on the keyboard (from left to right):

Q3. (Challenge)

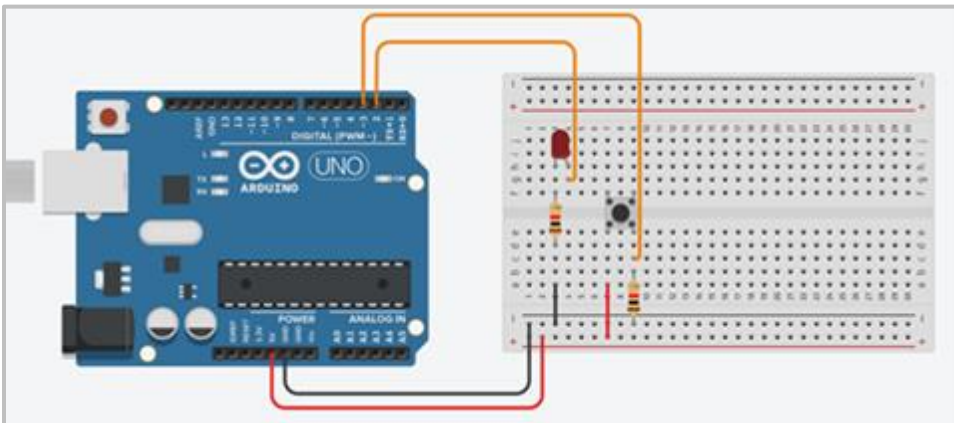


Figure 4-28. Circuit Layout of the Button.

Create the following circuit. Write a program that toggles the state of the LED when the button is pressed. In other words, if the LED starts, it remains off until the button is pressed. Then the LED remains off until the button is pressed again.

Hint: To achieve this it is likely that you will need to create two additional variables. One to keep track of whether the LED is on or off, and one to keep track of whether the button has been released.

Multiple Choice Questions:

1. What is the **voltage range** of the **Vin** pin?
 - a. 2.3 - 5.0V
 - b. 6.0 - 12.0V
 - c. 7.0 - 10.0V

2. The **Arduino CAN NOT be powered** by _____.
 - a. Barrel jack
 - b. USB cable
 - c. Digital I/O (input/output) pins

3. The integrated **Voltage Regulator** on the Arduino board **CANNOT convert** ____ to a stable 5V supply
 - a. USB cable voltage
 - b. Barrel jack voltage
 - c. Vin pin voltage

4. If the **total output current** is higher than _____, the integrated fuse on your Arduino will trip
 - a. 500mA.
 - b. 200mA
 - c. 400mA

5. What is the **output voltage and maximum current** of a digital or analog I/O pin?
 - a. Output voltage=5V, max current=40mA
 - b. Output voltage=3.3V, max current=400mA
 - c. Output voltage=5V, max current=200mA

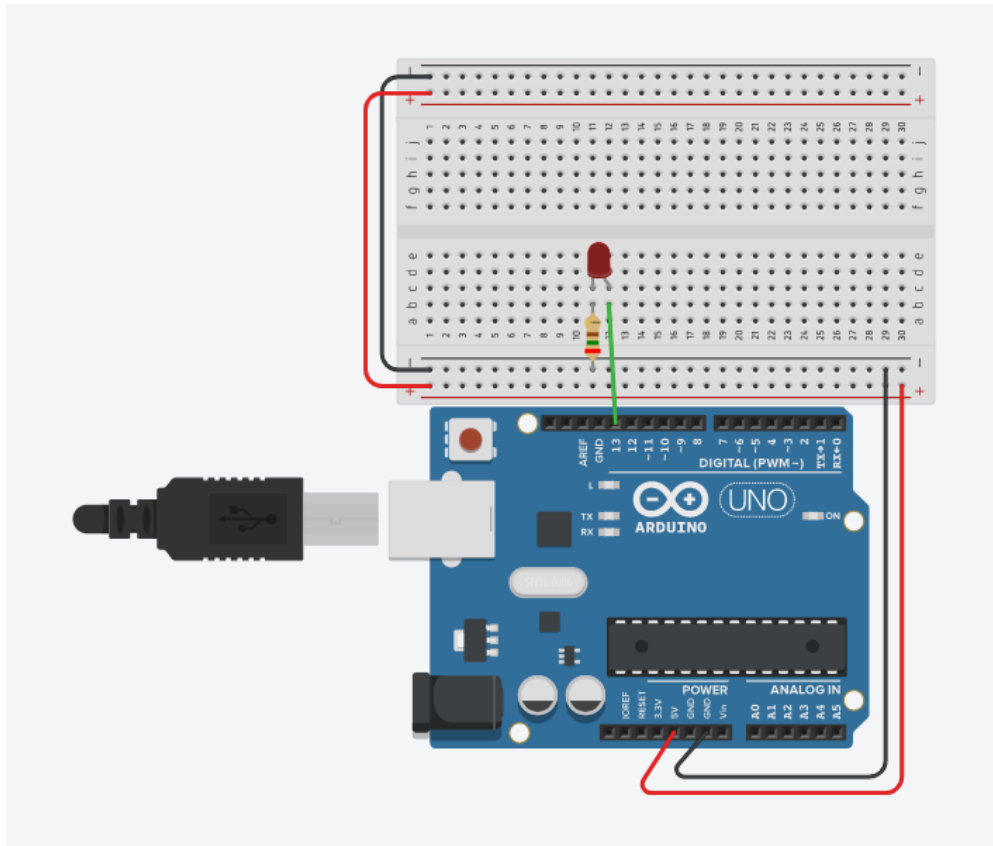
6. Which of these devices are considered an **Output device**?
 - a. Temperature Sensor
 - b. Keyboard
 - c. LCD Screen

7. In which of these **current/voltage** values will an **LED** turn on?
 - a. Current>100mA and Voltage >=1.8V
 - b. Current<20mA and Voltage >=1.8V
 - c. Current>10mA and Voltage < 1.8V

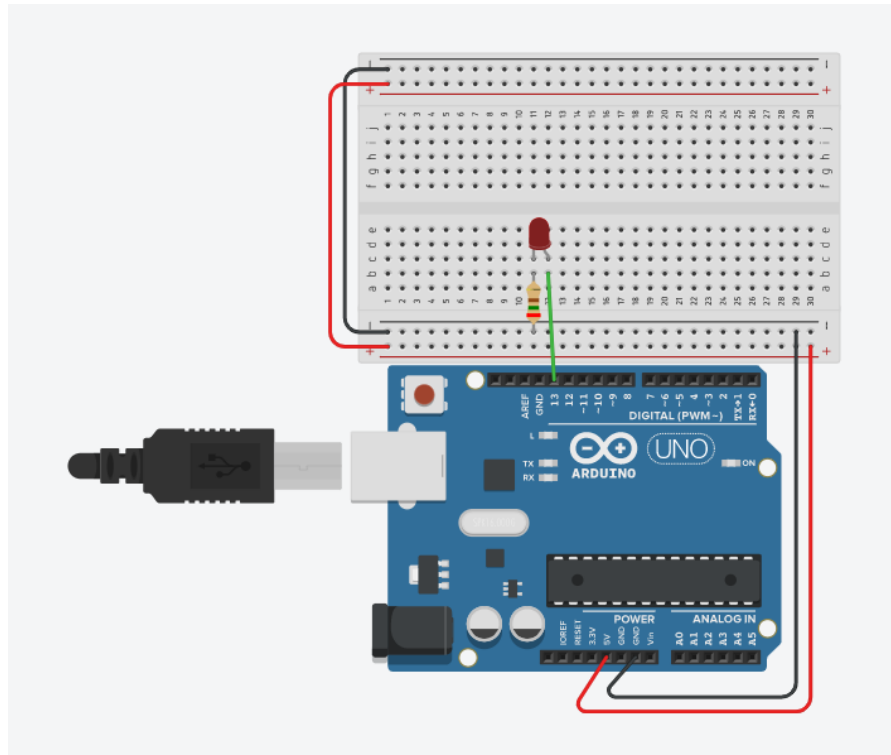
8. Which component(s) could be used to **limit current**?
 - a. Resistor
 - b. Battery

c. LED

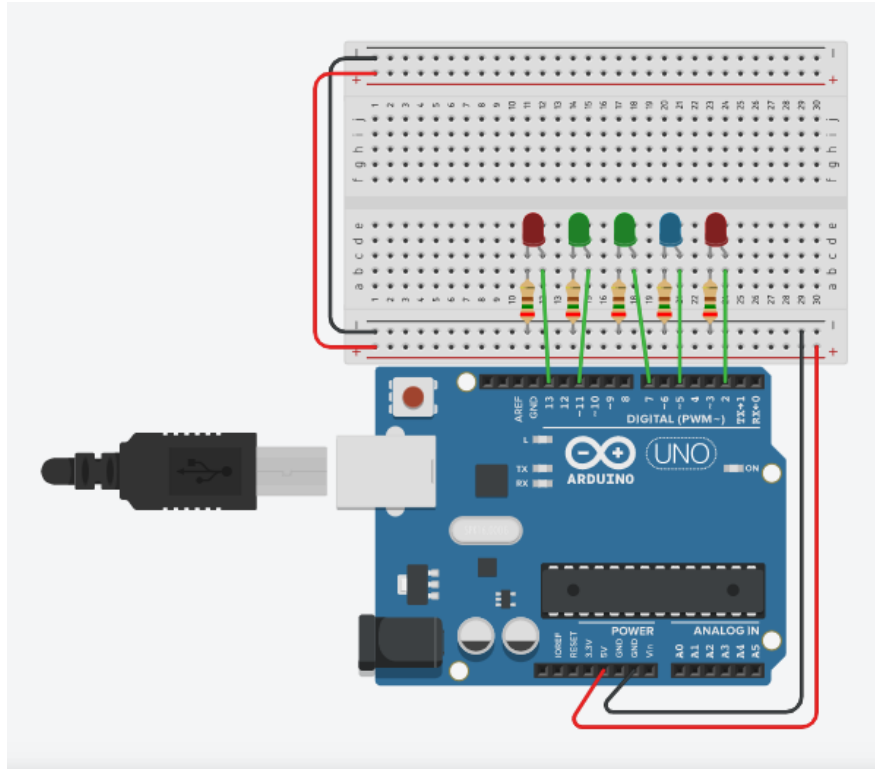
TASK BASED



1. Calculate the **resistor value** needed to turn on an LED using a **12V** battery?
use Ohm's law to calculate with max current the Arduino could provide
2. In TinkerCAD, design a circuit that **turns on an LED using a 3V battery**



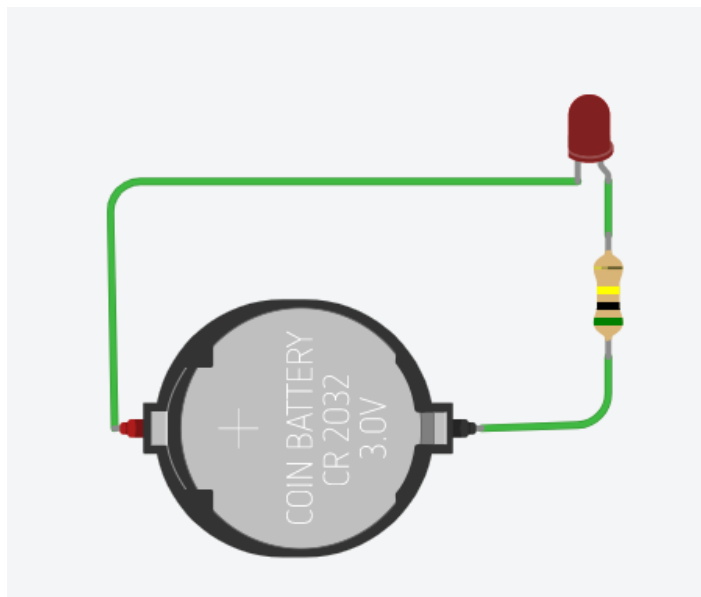
3. Write a **program to turn on an LED** using the resistor from Question #1 above.
4. Given the same circuit above, modify your program to **turn on** the LED for **1 second** and **turn it off for 500 milliseconds**.
5. Modify your program to complete the actions below:
 - a. Turn ON all LEDs for 1000 milliseconds
 - b. Turn OFF all LEDs for 1000 milliseconds
 - c. Turn ON just the Green LEDs for 1000 milliseconds
 - d. Turn OFF all LEDs for 1000 milliseconds
 - e. Turn ON just the red LEDs for 1000 milliseconds
 - f. Turn OFF all LEDs for 1000 milliseconds
 - g. Turn ON just the blue LED for 1000 milliseconds
 - h. Turn OFF all LEDs for 1000 milliseconds



FIND THE ERROR(TROUBLESHOOT)

In this section, the objective is to find the error in the code snippet and the circuits.

1. The following circuit should **turn on the LED**, but it does not work. Can you **find the error and fix it?**



2. The following code should set pin 13 as an output pin. When you run the code, **you get an error**. What is the error? Can you fix this error?

```

1 // C++ code
2 //
3 void setup()
4 {
5     pinMode(13, OUTPUT)
6
7 }
8

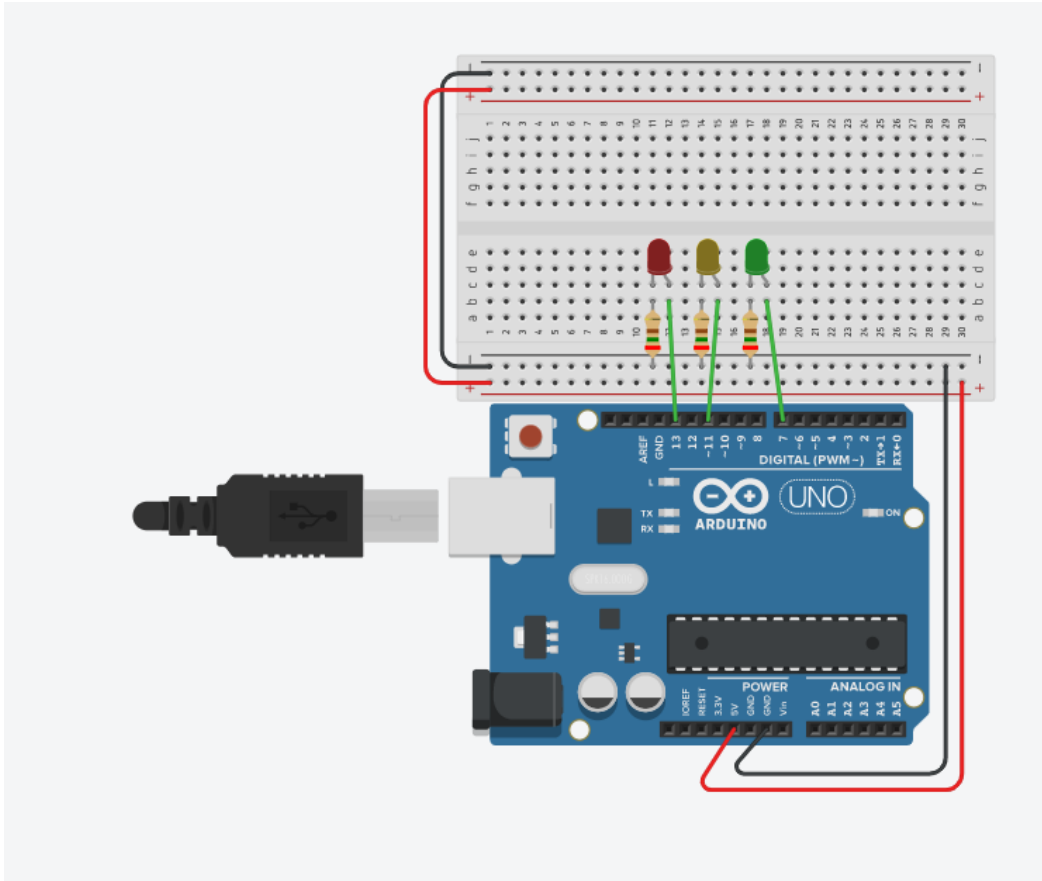
```

3. The following code should **turn on each LED** one by one in the circuit below. It should turn on each LED for **1000 milliseconds**, turn it off for **500 milliseconds**, and **repeat** the same sequence for the other LEDs. However, it doesn't work. Can you find the error and fix it?

```

1 // C++ code
2 //
3 void setup()
4 {
5     pinMode(13, OUTPUT);
6
7 }
8
9 void loop()
10 {
11     digitalWrite(13, HIGH);
12     digitalWrite(11, HIGH);
13     digitalWrite(7, HIGH);
14     delay(1000); // Wait for 1000 millisecond(s)
15     digitalWrite(13, LOW);
16     digitalWrite(11, LOW);
17     digitalWrite(7, LOW);
18     delay(500); // Wait for 1000 millisecond(s)
19 }

```



CHAPTER 5: WORKING WITH SENSORS

5.1 Parts You Will Learn

Now that you have learned the basics parts of the Arduino. You can start learning more about different types of Sensors that the Arduino can provide you with. Each sensor has different outputs and different functions. With the sensors, you can make more complex and exciting builds, with higher and greater outputs. Think of the Arduino sensors as sensors that you have in your body. They are similar. Humans have eyes that can see, those eyes send a signal to your brain, and then your brain converts this signal so that you can have a vision and sight. Arduino is pretty much the same. It takes input from a sensor and then sends this input to the Arduino circuit board. After that, you can tell the Arduino to light the LED when the data from the sensor is True. Sensors are very crucial towards the Arduino structure as they are the inputs of your circuit.

5.1.1 Push Button

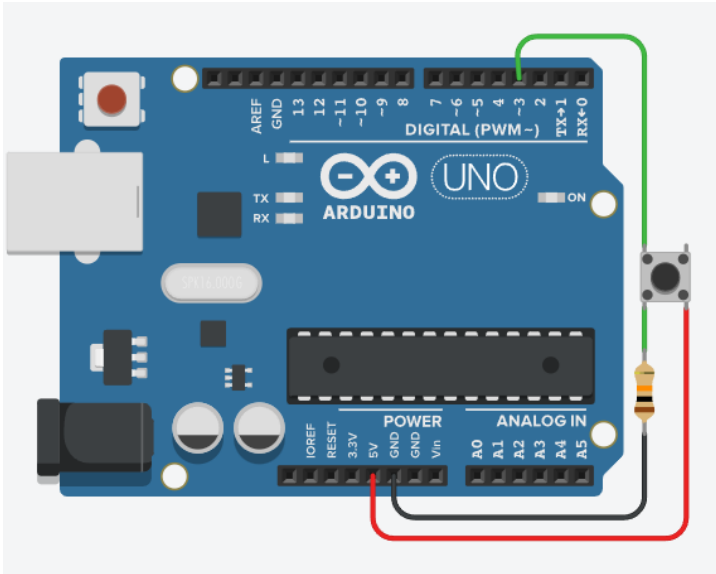
Specification Table	
Working Voltage:	DC 5V
Rating:	50mA / 12VDC
Contact Resistance:	100m max
Insulation Resistance:	100M min
Dielectric Strength:	250VAC for 1 minute
Operating Life:	300,000 cycles per minute
Operating Temperature:	-25C + 70C
Size:	6.2*4.5mm



Figure 5-1. Push Buttons

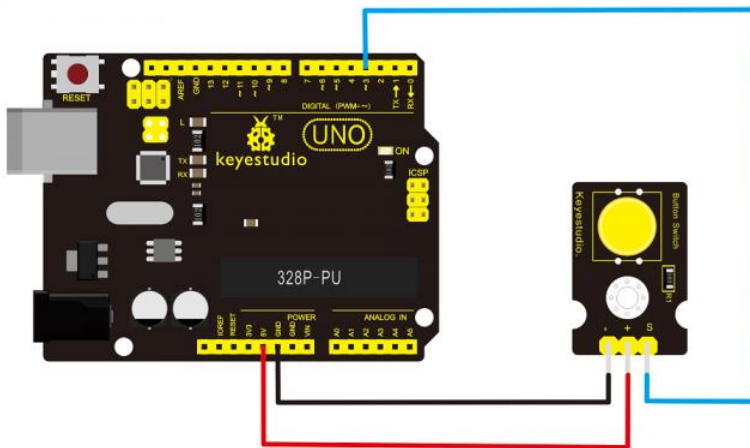
A **push button** is a simple switch mechanism for controlling some aspect of a machine or a process. A switch is an electrical component that can "make" or "break" an electrical circuit, interrupting the current or diverting it from one conductor to another. The mechanism of a switch removes or restores the conducting path in a circuit when it is operated.

4 pin push button Example



<i>Push Button</i>	<i>Arduino</i>
1	3
3	GND
4	5V

3 Pin Push button Example:



<i>Push Button</i>	<i>Arduino</i>
+	5V
-	GND
S	3

Push Button Sample Code:

```
int buttonState = 0; // Set ButtonState as 0

void setup()
{
  pinMode(3, INPUT); // Set pin 3 as an Input
}
```

```

pinMode(13, OUTPUT); // Set pin 13 as an Output
}

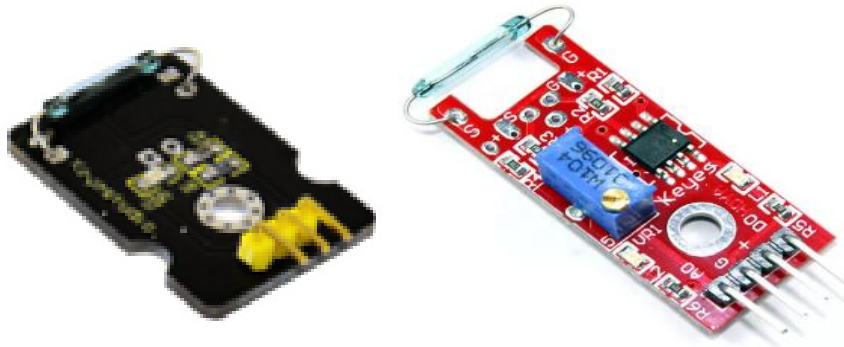
void loop()
{
  buttonState = digitalRead(3); // Read the value assign it to buttonState

  if (buttonState == HIGH) // If buttonState is equals to high
  {
    digitalWrite(13, HIGH); // Turn on the electricity on the pin 13
  }

  else // if not, then
  {
    digitalWrite(13, LOW); // Turn off the electricity on the pin 13
  }
  delay(10);
}

```

5.1.2 Reed Switch (Magnetic Sensor)



Specification Table	
Working Voltage:	DC 2.3V-5V
Working Current	≥20mA
Operating Temperature:	-10°C—+50°C
Size:	30*20mm

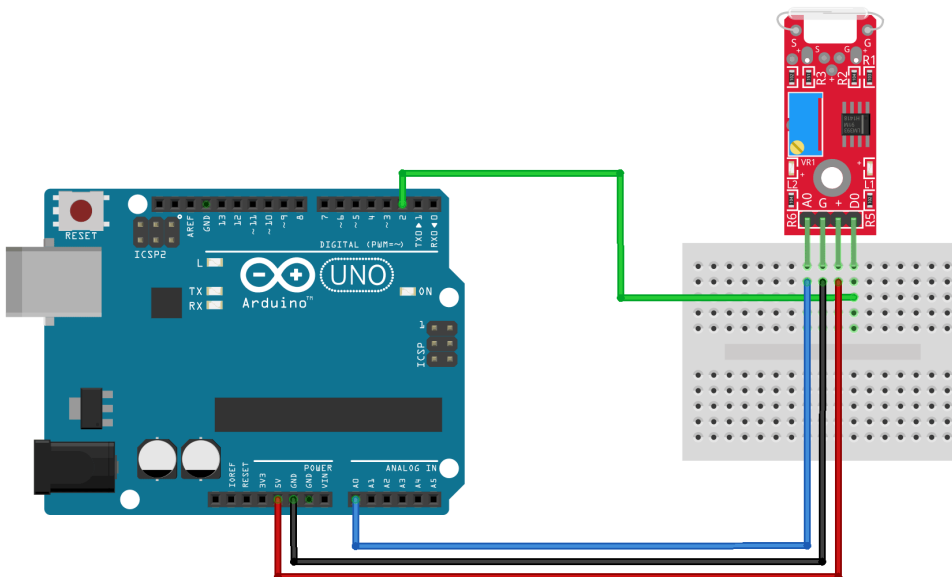
Detection Distance:	≤10mm
---------------------	-------

Figure 5-2. Reed Switch (Magnetic Sensor)

The **reed switch** is an electrical switch operated by an applied magnetic field. The contacts may be normally open, closing when a magnetic field is present, or normally closed and opening when a magnetic field is applied. The switch may be actuated by a coil, making a reed relay or by bringing a magnet near to the switch. Once the magnet is pulled away from the switch, the reed switch will go back to its original position.

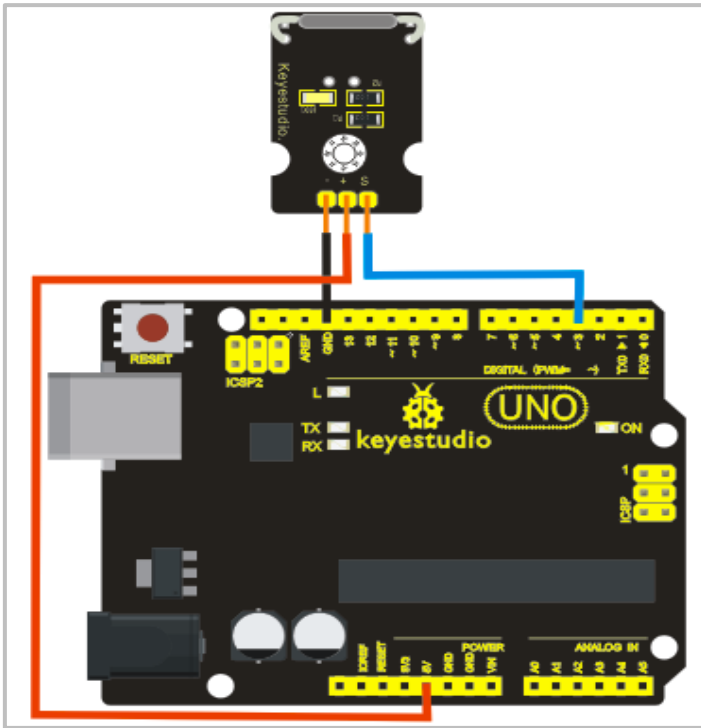
An example of a reed switch application is to detect the opening of a door when used as a proximity switch for a burglar alarm.

4 Pin Code Example:



<i>Reed Switch</i>	<i>Arduino</i>
A0	A0
G	GND
+	5V
D0	3

3 Pin Code Example



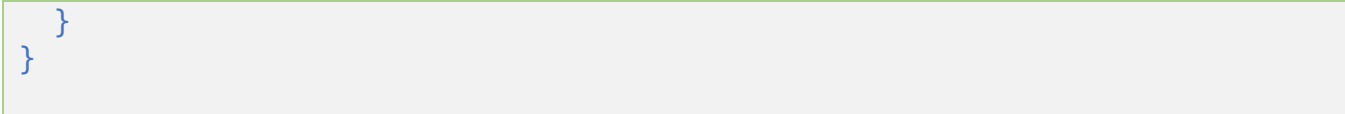
<i>Reed Switch</i>	<i>Arduino</i>
+	5V
-	GND
S	3

Reed Switch Sample Code:

```
int led = 13; // Set LED pin as 13
int reelSwitch = 3; //Set reed switch pin as 3
int switchState; // variable to store reel switch value

void setup()
{
  pinMode (led, OUTPUT); // Set pin 13 as an OUTPUT
  pinMode (reelSwitch, INPUT); // Set pin 3 as an INPUT
}

void loop()
{
  switchState = digitalRead(reelSwitch);
  // read the value of digital interface 2 and assign it to switchState
  // when the magnetic sensor detect a signal, LED is flashing
  if (switchState == HIGH)
  {
    digitalWrite(led, HIGH);
  }
  else
  {
    digitalWrite(led, LOW);
  }
}
```



5.1.3 PIR Motion Sensor

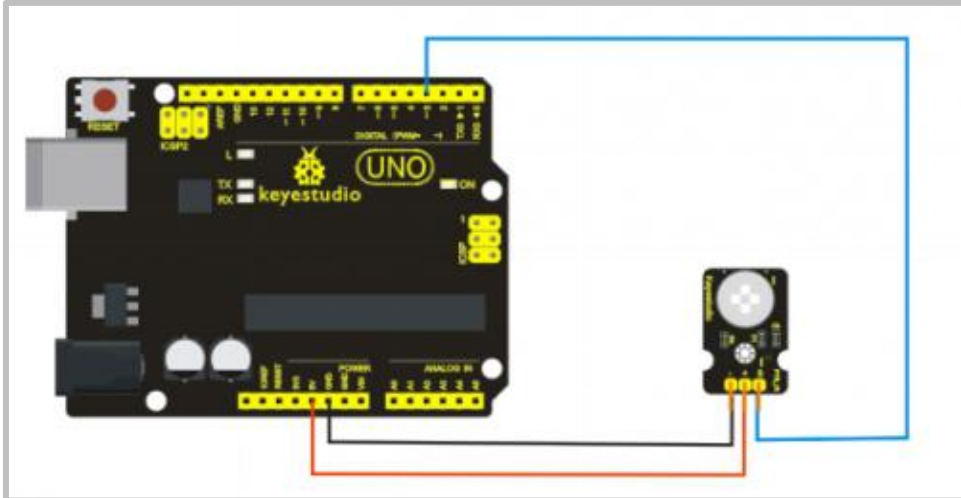


Specification Table	
Input Voltage:	2.3 ~ 5V, 6V Maximum
Working Current	15uA
Operating Temperature:	-20 ~ 85 °C
Output Voltage:	High 3V, low 0V
Detection Distance:	≤10mm
Output Delay Time (High Level):	About 1.3 to 3 Seconds

Detection angle:	100 °
Detection distance:	7 meters
Output Indicator LED	(When output HIGH, it will be ON)
Pin limit current:	100mA
Size:	30*20mm
Weight:	4g

Figure 5-5. PIR Motion Sensors.

A **passive infrared (PIR)** sensor measures infrared light emitted from objects that generate heat, in its field of view. Crystalline material at the center of a rectangle on the face of the sensor detects the infrared radiation. The sensor is split into two halves to identify not the radiation itself, but the change in condition that occurs when a target enters its field. These changes in the amount of infrared radiation on the element, in turn, change the voltages generated, which are measured by an onboard amplifier. When motion is detected the PIR sensor outputs a high signal on its output pin, which can either be read by an MCU or drive a transistor to switch a higher current load.



<i>Motion Sensor</i>	<i>Other</i>
+	5V
-	GND
S	3

PIR Sample Code:

```
int sensor = 3; // Set sensor as a pin 3

void setup()
{
  pinMode(sensor, OUTPUT); // Set pin 3 as an OUTPUT
  pinMode(13, OUTPUT)
  Serial.begin(9600);
}

void loop()
{
  int state = digitalRead(sensor);
  // If the state of the sensor is equals to 1
  if (state == 1)
  {
    Serial.println("True"); // print True
    digitalWrite(13, HIGH); // turn on the on board LED
  }

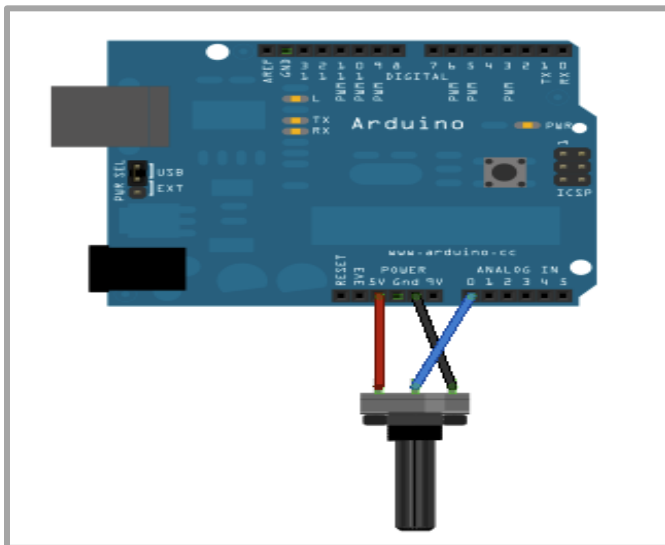
  else //if not
  {
    Serial.println("False"); // print False
  }
}
```

5.1.4 Potentiometer



Specification Table	
Working voltage:	5V
Size:	16*34mm
Weight:	7g

A **potentiometer** is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor. Potentiometers are commonly used to control electrical devices such as volume controls on audio equipment.



<i>Potentiometer</i>	<i>Arduino</i>
+	5V
-	GND
S	A0

Potentiometer Sample Code:

```
int sensorValue = 0; // define a sensorValue as 0

void setup()
```

```

{
  pinMode(A0, INPUT); //define pin A0 as an Input
  pinMode(13, OUTPUT); //define pin 13 as an OUTPUT
}

void loop()
{
  sensorValue = analogRead(A0);
  //sensorValue equals to input from analogRead

  if(sensorValue >= 1){ //if the sensor is greater or equal to 1
    digitalWrite(13, HIGH); //turn on the on board LED
    delay(sensorValue); //delay to the amount that the sensorValue has
    digitalWrite(13, LOW); //turn off the on board LED
  }
}

```

5.1.5 Photoresistor



Specification Table

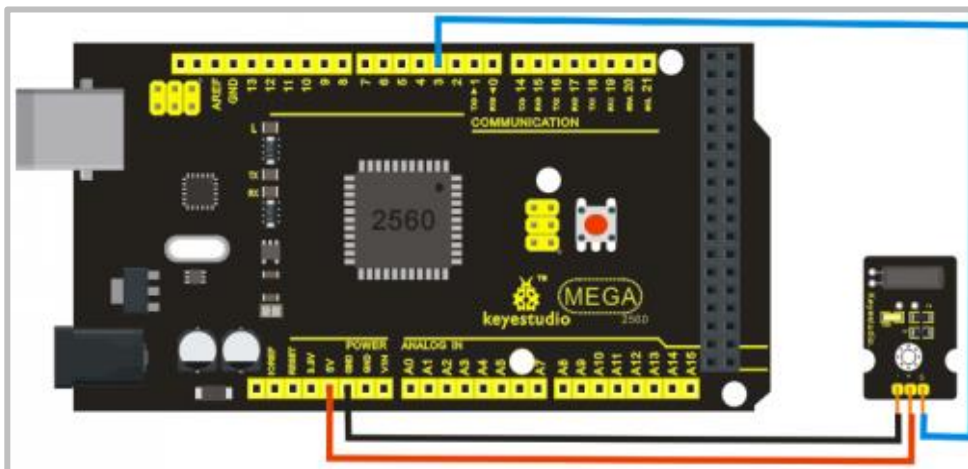
Working voltage:	5V
Size:	38*20mm
Weight:	5g

Figure 5-9. Photosensors.

Photoresistor (Photoresistor) is a resistor whose resistance varies from different incident light strength. It's made based on the photoelectric effect of a semiconductor. If the incident light is intense, its resistance reduces; if the incident light is weak, the resistance increases. Photoresistor is commonly applied in the measurement of light, light control and photovoltaic conversion (convert the change of light into the change of electricity).

A photoresistor is also widely applied to the various light control circuit, such as light control and adjustment, optical switches, etc. Photovaristor is an element that will change its resistance as light strength changes. So, it needs to read the analog values.

The three pronged Photoresistor is not the only version, a photoresistor can also have two prongs. The difference between the two is that the three pronged resistor has an internal voltage divider, this allows for direct measurements of the change in resistance. The two pronged resistor does not have an internal voltage divider, so an external voltage divider must be used to measure a change in voltage.

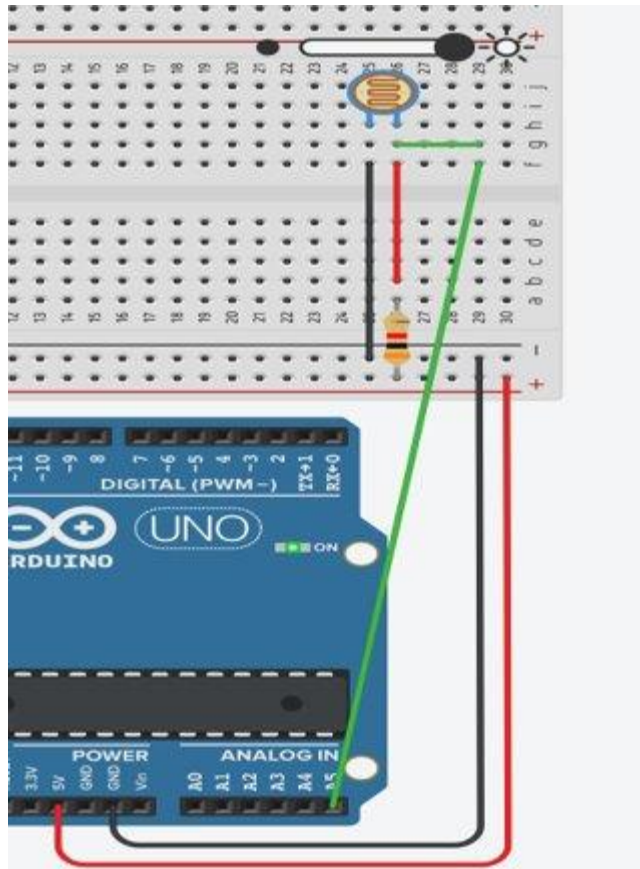


Pin Connections

[+] → [+5V]

[-] → [GND]

[S] → [3]



Photoresistor Sample Code:

```
int lightPin = 3; //define a pin for Photo resistor
int ledPin = 13; //define a pin for LED

void setup()
{
  Serial.begin(9600); //Begin serial communication
  pinMode( ledPin, OUTPUT );
}

void loop()
{
  Serial.println(analogRead(lightPin));
  //Write the value of the photoresistor to the serial monitor.

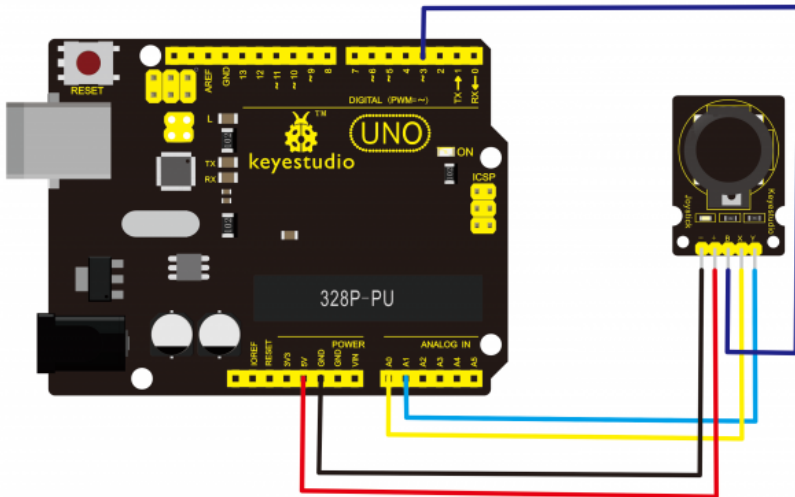
  analogWrite(ledPin, analogRead(lightPin) / 4);
  //send the value to the ledPin. Depending on value of resistor
  delay(10); //short delay for faster response to light.
}
```

5.1.6 Joystick



Specification Table	
Supply Voltage:	2.3V to 5V
Size:	40*28mm
Weight:	12g
Interface	Analog x2, Digital x1

A 2-Axis Joystick contains two independent potentiometers (one per axis) that can be used as dually adjustable voltage dividers, providing 2-Axis analog input in a control stick form. The modular form factor allows you to plug the 2-Axis Joystick directly into a breadboard for easy prototyping. The 2-Axis Joystick includes spring auto return to center.



<i>Joystick</i>	<i>Arduino</i>
+	5V
-	GND
B	3
Y	A0
X	A1

Joystick Sample Code:

```

const int SW_pin = 3; //set switch pin to 3
const int X_pin = 0; //set x pin to 0
const int Y_pin = 1; //set y pin to 1

void setup()
{
  pinMode(SW_pin, INPUT); //set switch pin to input
  digitalWrite(SW_pin, HIGH); //turn on the switch pin
  Serial.begin(115200); //begin the serial monitor
}

void loop()
{
  Serial.print("Switch: ");
  Serial.print(digitalRead(SW_pin));
  Serial.print("\n");
  Serial.print("X-axis: ");
  Serial.print(analogRead(X_pin)); //read and print value from the x pin
  Serial.print("\n");
  Serial.print("Y-axis: ");
  Serial.println(analogRead(Y_pin)); //read and print value from the y pin
  Serial.print("\n\n");
  delay(500);
}

```

5.2 Complex Sensors

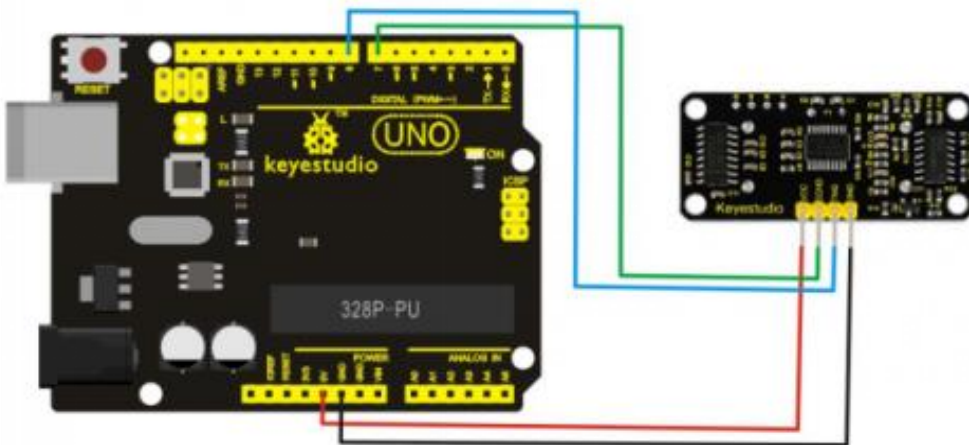
5.2.1 Ultrasonic Sensors



Specification Table	
Working Voltage:	DC 5V
Working Current:	15mA
Working Frequency:	40Hz
Max Range:	5m
Min Range:	2cm
Measuring Angle:	15 degree
Size:	49*22mm

Figure 5-13. Ultrasonic Sensors

An **Ultrasonic sensor** is a device that can measure the distance to an object by using sound waves. It measures distance by sending out a sound wave at a specific frequency and listening for that sound wave to bounce back. By recording the elapsed time between the sound wave being generated and the sound wave bouncing back, it is possible to calculate the distance between the sensor and the object.



<i>Ultrasonic Sensor</i>	<i>Arduino</i>
VCC	5V
GND	GND
trig	8
echo	7

Figure 5-14. The wiring Diagram and Pin Connections for the Ultrasonic Sensor.

Ultrasonic Sensor Sample Code:

```
#include "NewPing.h" // Include NewPing Library

#define TRIGGER_PIN 8 // Set Trig to Arduino Pin 8
#define ECHO_PIN 7 // Set Echo to Arduino pin 7

// Maximum distance we want to ping for (in centimeters).
#define MAX_DISTANCE 400

// NewPing setup of pins and maximum distance.
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
float duration, distance;

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

void loop()
{
  // Send ping, get distance in cm
  distance = sonar.ping_cm();

  // Send results to Serial Monitor
  Serial.print("Distance = ");

  if (distance >= 400 || distance <= 2)
  {
```

```

Serial.println("Out of range");
}
else
{
  Serial.print(distance);
  Serial.println(" cm");
}
delay(500);
}

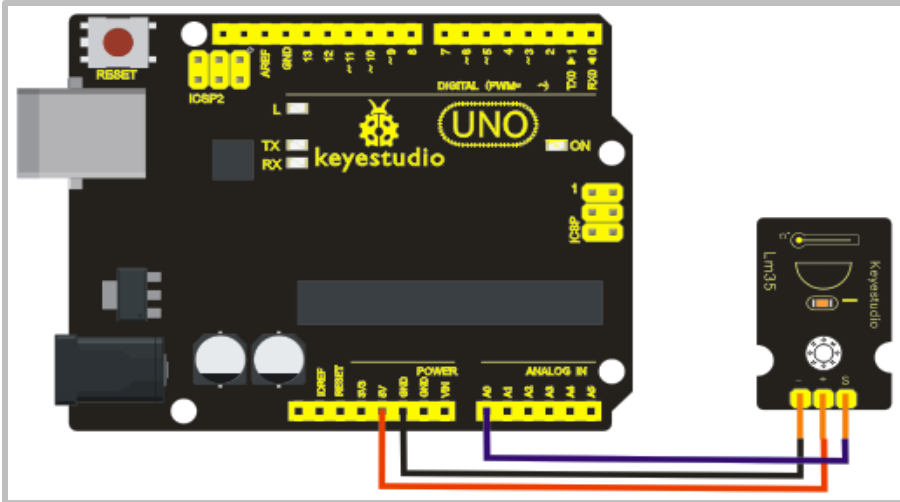
```

5.2.2 Temperature Sensor



Specification Table	
Supply Voltage:	2.3V to 5V
Temperature range:	-55 °C ~ +125 °C
Size:	30*20mm
Weight	3G

A **temperature sensor** is a device, typically, a thermocouple and an RTD, that provides for temperature measurement through an electrical signal. An RTD (Resistance Temperature Detector) is a variable resistor that will change its electrical resistance in direct proportion to changes in temperature in a precise, repeatable and nearly linear manner. There are two kinds of temperature sensors, those with internal voltage dividers and those with external. If the voltage divider is internal, the sensor will have three prongs. If the sensor has two prongs a external voltage divider is needed,



<i>Temperature Sensor</i>	<i>Arduino</i>
+	5V
-	GND
S	3

Figure 5-16. The Wiring Diagram and Pin Connections for a 3 pin temperature sensor.

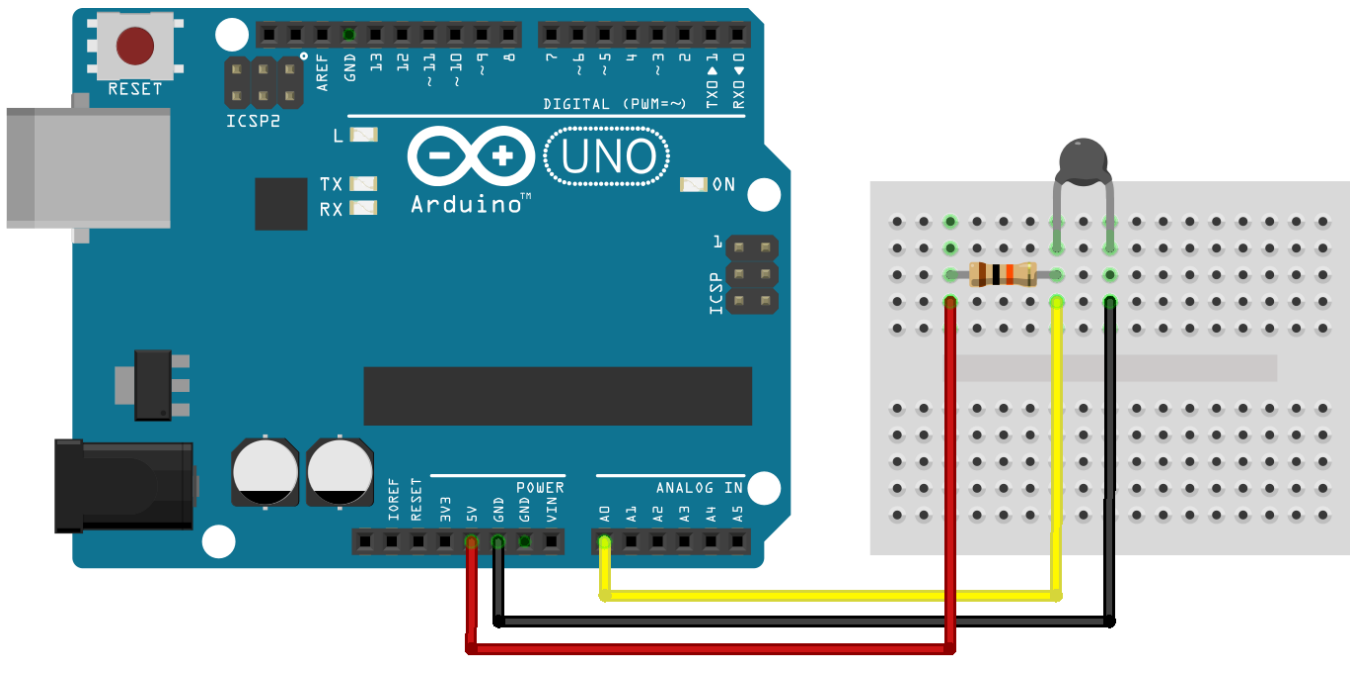


Figure 5-17. The Wiring Diagram and Pin Connections for a 2 pin temperature sensor.

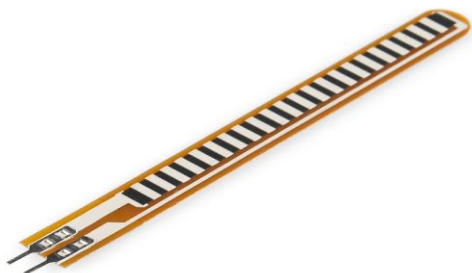
Temperature Sensor Sample Code:

```
float temperature;
int tempPin = 3; // Set temperature pin as 3

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

void loop()
{
  temperature = analogRead(tempPin);
  // Set the temperature as an analogRead pin 3
  temperature = temperature * 0.48828125;
  Serial.print("Temperature = ");
  Serial.print(temperature);
  Serial.println("C");
}
```

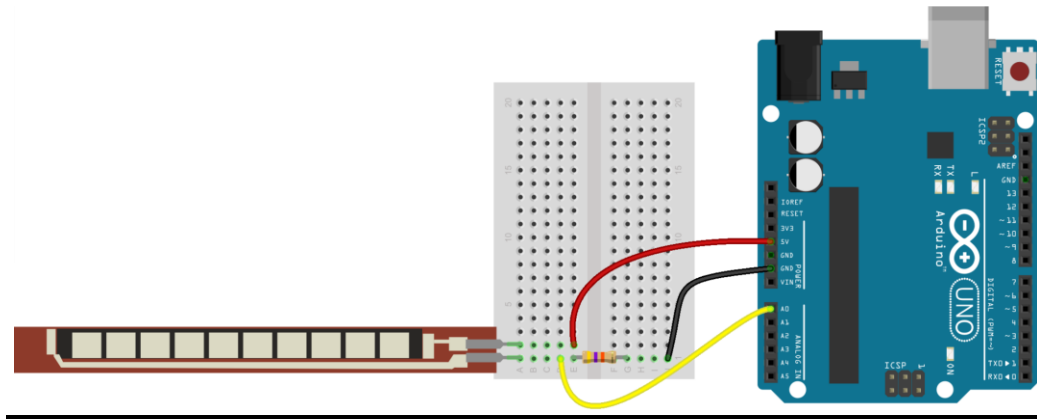
5.2.3 Flex Sensor



Specification Table

Working voltage:	5V
Size:	6.35*73.66mm
Weight:	0.27g

A **Flex Sensor** is a device that measures the amount of bending applied to it, it does so by sending an electrical signal that can vary. The Flex Sensor is a form of variable resistor that will change its electrical resistance in relation to the degree of bending done upon it. To get an actual measurement from the sensor, an external voltage divider circuit must be used.



Flex Sensor Sample Code:

```
int flex = 0; //set to zero void

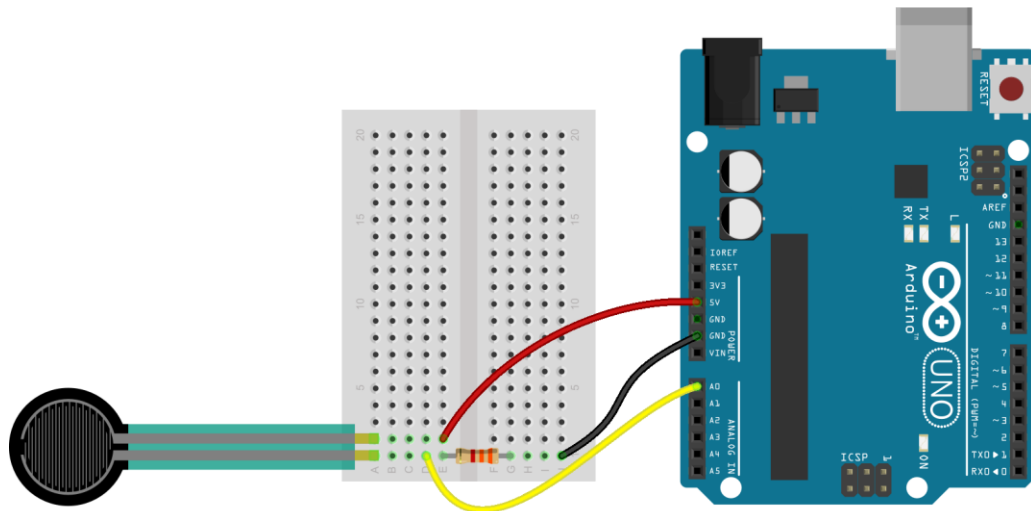
void setup()
{
  pinMode(A5, INPUT);
  Serial.begin (9600);
}
void loop()
{
  flex = analogRead (A0);
  Serial.println ("flex value = ")
  Serial.println (flex);

  if(flex > 0){ //if the flex sensor is bent
    Serial.println("flex sensor is bent");
  }
}
```

5.2.4 Force Sensor



A **Force Sensor** is a device that measures the amount of pressure pressed on to it, The Force Sensor is a form of variable resistor that will change its electrical resistance in relation to the weight / pressure placed upon it. To get an actual measurement from the sensor, an external voltage divider circuit must be used.



Force Sensor Smaple Code:

```
int force=0;

void setup()
{
  pinMode(A0, INPUT);
  Serial.begin(9600);
}

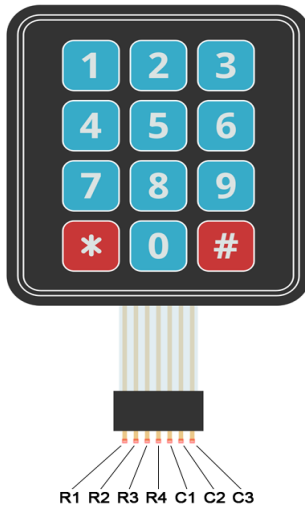
void loop()
{
  force = analogRead(A5);
  Serial.println(force);

  if(force > 0){ //if a force is sensed
    Serial.println("force detected");
  }
}
```

```
1  int force=0;
2  int vibe=0;
3
4  void setup()
5  {
6    pinMode(A0, INPUT);
7    pinMode(3, OUTPUT);
8    Serial.begin(9600);
9  }
10
11 void loop()
12 {
13   force = analogRead(A0);
14   vibe = map(force, 109, 1023, 255, 0);
15   digitalWrite (3,vibe);
16
17   Serial.print("force = ");
18   Serial.println(force);
19   Serial.print("vibration = ");
20   Serial.println(vibe);
21 }
22
```

Serial Monitor

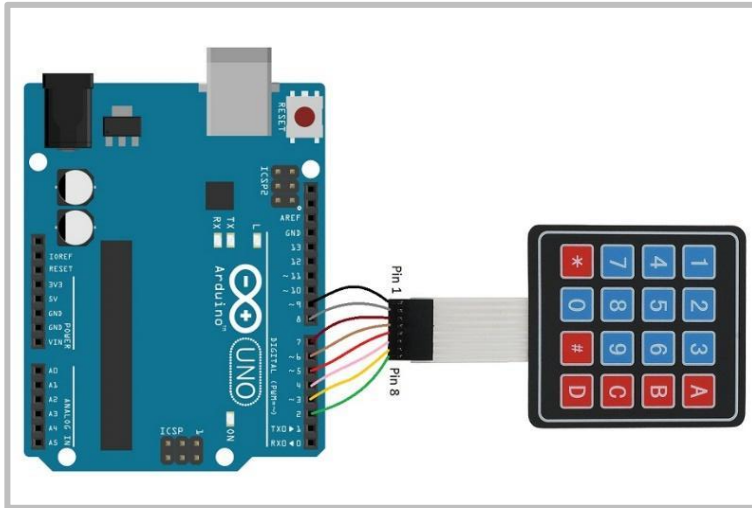
5.2.5 Keypad



Specification Table	
Max. circuit rating:	35VDC, 100Ma
Pad Size:	66.9 x 76 x 0.8mm
Cable length:	85mm (include connector)
Connector:	dupont 7 pins, 0.1-inch (1.54mm) Pitch
Contact bounce:	≤5ms
Weight:	7g

Figure 5-17. The Keypad

Arduino allows you to read a matrix type keypad. You can find these keypads from old telephones or from almost any electronics parts store. They come in 3x4, 4x4 and various other configurations with words, letters and numbers written on the keys. They are internally wired with switches in a matrix configuration (see below), which allows when a key is pressed to determine the x and y location of that key.



<i>Keypad</i>	<i>Arduino</i>
1	9
2	8
3	7
4	6

Keypad Sample Code:

```
#include <Keypad.h>

const byte numRows = 4; //number of rows on the keypad
const byte numCols = 4; //number of columns on the keypad

//keymap defines the key pressed according to the row and columns just as
//appears on the keypad

char keymap[numRows][numCols] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

//Code that shows the keypad connections to the Arduino terminals

byte rowPins[numRows] = {9, 8, 7, 6}; //Rows 0 to 3
byte colPins[numCols] = {5, 4, 3, 2}; //Columns 0 to 3

//initializes an instance of the Keypad class

Keypad myKeypad = Keypad(makeKeymap(keymap), rowPins, colPins, numRows,
numCols);

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

```

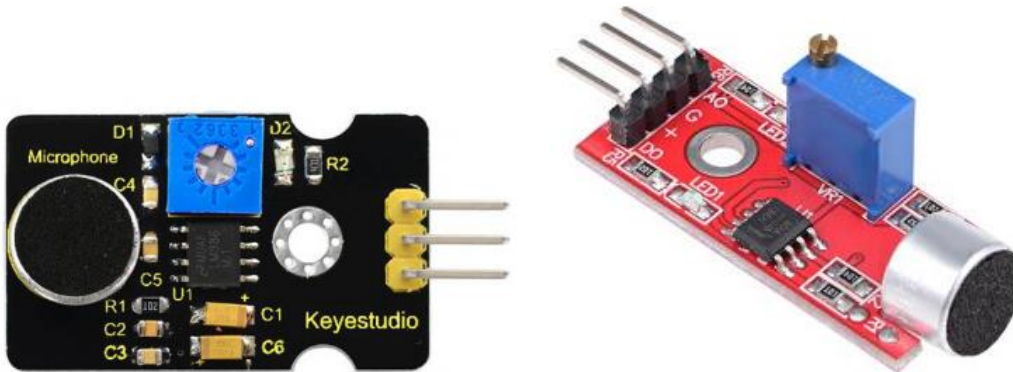
}

//If key is pressed, this key is stored in 'keypressed' variable
//If key is not equal to 'NO_KEY', then this key is printed out
//if count=17, then count is reset back to 0 (this means no key is pressed
//during the whole keypad scan process)

void loop()
{
  char keypressed = myKeypad.getKey();
  if (keypressed != NO_KEY)
  {
    Serial.print(keypressed);
  }
}

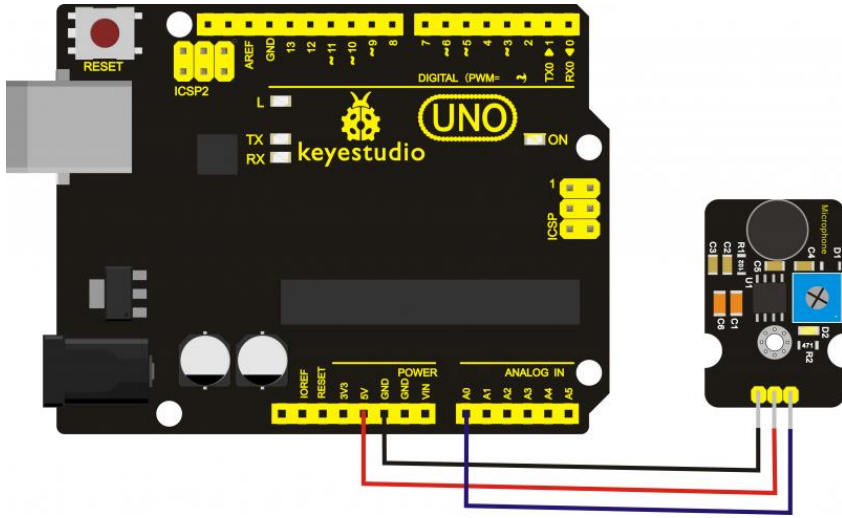
```

5.2.6 Sound Sensor



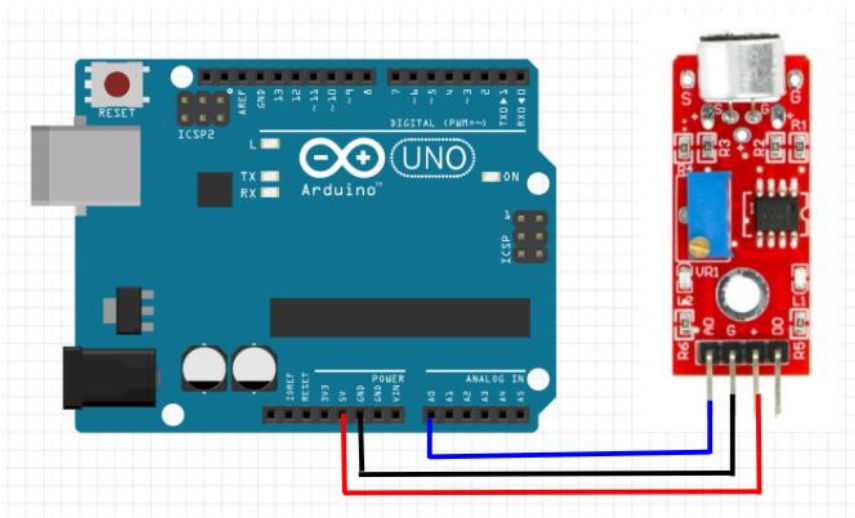
The sound sensor is an Arduino module that allows users to incorporate sound detection into their projects. It uses a microphone to detect the loudness in the surrounding environment and sends the appropriate signals to the Arduino board.

3 pin example:



<i>Sound sensor</i>	<i>Arduino</i>
V	5V
G	GND
S	A0

4 pin example: (can use DO pin for digital out or AO pin for analog out):

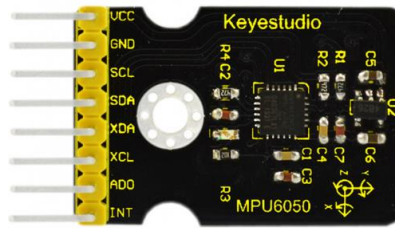


<i>Sound sensor</i>	<i>Arduino</i>
AO	A0
G	GND
+	5V

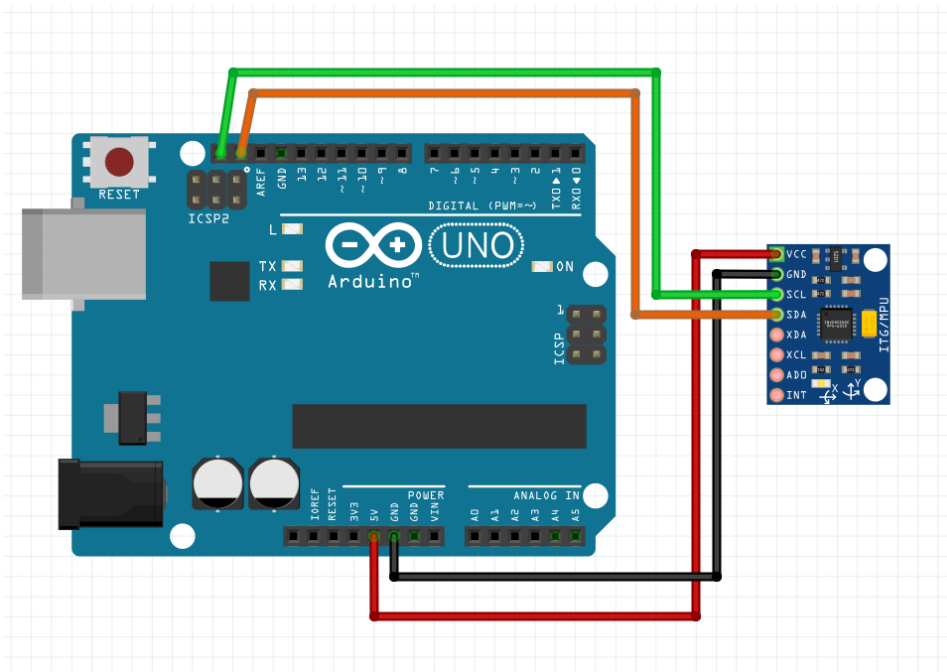
Sound Sensor Code Example:

```
void setup() {  
  Serial.begin(9600); //Start the Serial monitor  
  
}  
  
void loop() {  
  int value = analogRead(A0); //Set the value variable to the analog read  
  from pin A0  
  
  Serial.println(value, DEC); //Print the value that was read from the  
  serial monitor  
  
  if(value >0){ //Run code if a sound was detected  
    Serial.println("Sound detected!"); // Print sound detected to serial  
    monitor  
  }  
  
  delay(100); //Wait 0.1 seconds  
}
```


5.2.7 Gyroscope and Accelerometer (MPU6050)



The gyroscope and accelerometer module allows users to measure rotational velocity and the rate of change of the angular position in the X, Y, and Z axis over time. The MPU6050 has a 3 axis gyroscope and a 3 axis accelerometer integrated on the chip. **Note:** the `Adafruit_MPU6050.h` and `Adafruit_Sensor.h` libraries will need to be added to the Arduino IDE.



<i>Gyroscope</i>	<i>Arduino</i>
VCC	5V
GND	GND
SCL	SCL
SDA	SDA

Gyroscope Code Example:

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

Adafruit_MPU6050 mpu; //create object from Adafruit_MPU6050 class

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

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G); //set accelerometer
sensitivity to 8

  mpu.setGyroRange(MPU6050_RANGE_500_DEG); //set gyroscope range to 500
degrees per second

  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ); //set bandwith to 21Hz

  delay(100);
}

void loop() {

  sensors_event_t accel, gyroscope; //create objects to hold the results
read from sensor

  mpu.getEvent(&accel, &gyroscope, NULL); //reads the result read from the
sensor

  Serial.print("X Acceleration: ");
```



```
Serial.print(accel.acceleration.x); //print the X acceleration to serial
Serial.print(", Y Acceleration: ");
Serial.print(accel.acceleration.y); //print Y acceleration to serial
Serial.print(", Z Acceleration: ");
Serial.print(accel.acceleration.z); //print Z acceleration to serial
Serial.println(" m/s^2");

Serial.print(" X Rotation: ");
Serial.print(gyroscope.gyro.x); //print x rotation to serial
Serial.print(", Y Rotation: ");
Serial.print(gyroscope.gyro.y); //print y rotation to serial
Serial.print(", Z Rotation: ");
Serial.print(gyroscope.gyro.z); //print Z rotation to serial
Serial.println(" rad/s");

delay(1000); //wait 1 second
}
```

5.2.8 Sensor Shield

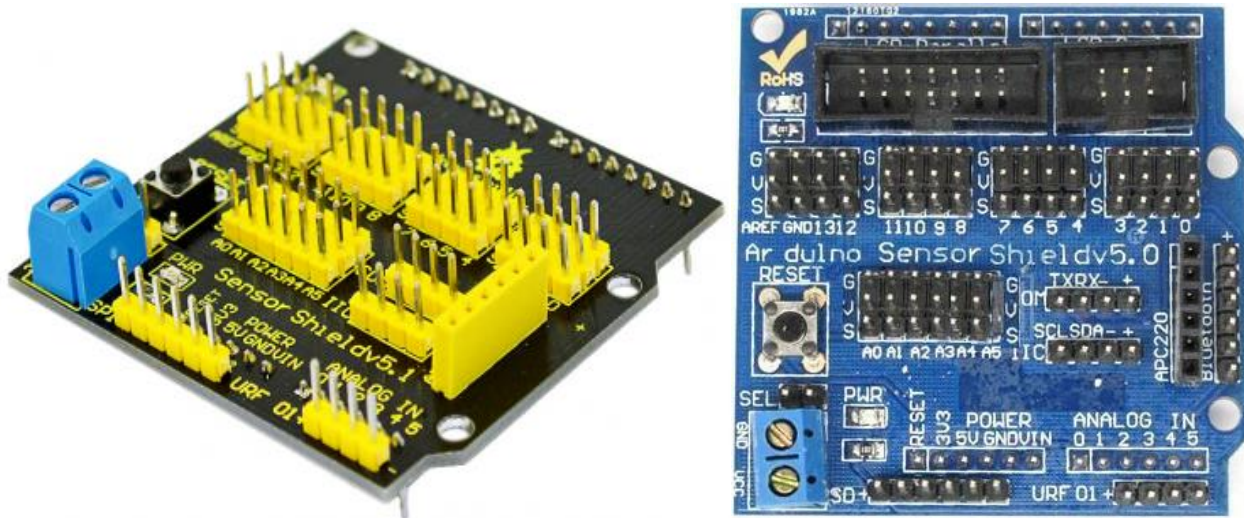


Figure 7-1. The Arduino Sensor Shield.

This type of Sensor Board Shield simplifies the circuit in the sense to make the commonly used sensors easily connected. It would be best if you only connected a sensor and after finishing the circuit connection, compile the corresponding Arduino program and download it to the Arduino MEGA controller to read the sensor data, or receive **7.1.1** returning data of wireless module and finally finish your interactive project. This type of sensor shield draws 800mA of power.

https://wiki.keyestudio.com/Ks0004_keyestudio_Sensor_Shield_V5#Introduction

5.3 Sensor Library

Ultrasonic Sensor Library

Library name:

```
#include <Keypad.h>
```

Declaration:

```
NewPing sonar( Trigger, Echo, maxDistance);
```

Functions:

```
ping_cm();
```

```
ping_in();
```

If you need more information or you still have questions, you can visit the link below:

Reference: <https://playground.arduino.cc/Code/NewPing>

KeyPad Library

Library name:

```
#include <NewPing.h>
```

Declaration:

```
const byte rows = 4; //four rows
```

```
const byte cols = 3; //three columns
```

```
char keys[rows][cols] = {
```

```
    {'1', '2', '3'},
```

```
    {'4', '5', '6'},
```

```
    {'7', '8', '9'},
```

```
    {'*', '0', '#'}
};
```

```
byte rowPins[rows] = {A8,A9,A10,A11}; //connect to the row pinouts of the keypad
```

```
byte colPins[cols] = {A12,A13,A14}; //connect to the column pinouts of the keypad
```

```
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, rows, cols );
```

Functions:

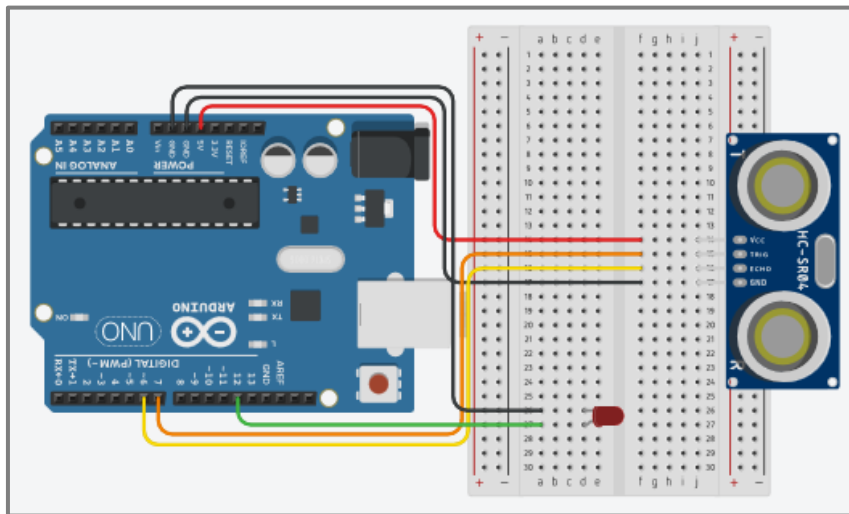
```
waitForKey();
```

If you need more information or you still have questions, you can visit the link below:

Reference: <http://playground.arduino.cc/Code/Keypad>

5.4 Practice Questions

Build the code for this circuit, which makes the LED light up if the distance between the ultrasonic sensor and an object is less than 10cm. If the distance is more than 10cm then the LED will turn off.



Ultrasonic Sensor and LED Practice Code Solution:

```
#include <NewPing.h>
NewPing mysonar (7, 6, 200);
int mydistance;

void setup()
{
  pinMode (12, OUTPUT);
}

void loop()
{
  mydistance = mysonar.ping_cm();

  if (mydistance < 10)
  {
    digitalWrite(12, HIGH);
  }
}
```

```

}

if (mydistance > 10)
{
  digitalWrite(12, LOW);
}
}

```

PIR Sensor Practice:

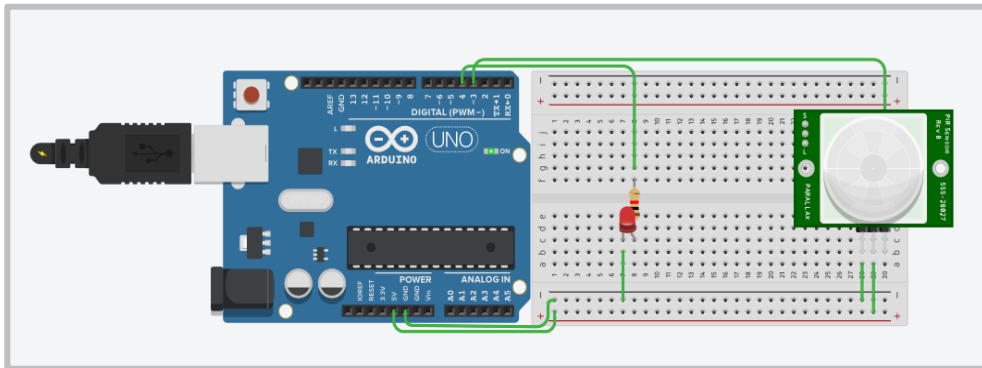


Figure 5-19. Circuit Diagram for an LED and PIR Sensor

Create a code using the circuit above. If the sensor senses that there is movement, then the LED will turn on.

PIR Sensor Practice Code:

```

int sensor = 3;
int indicator = 4;

void setup()
{
  pinMode(sensor, INPUT);
  pinMode(indicator, OUTPUT);
}

void loop()

```

```

{
  int state = digitalRead(sensor); //get valule from PIR sensor

  if (state == 1 )
  {
    digitalWrite(indicator, HIGH); //if true, indicator is on
    delay(100);
  }

  else if (state == 0)
  {
    digitalWrite(indicator, LOW); //if false, indicator is off
    delay(100);
  }
}

```

2) PIR sensor with two indicators.

Create the code for the circuit above. If the sensor senses that there is movement, then the LED and the buzzer will turn on.

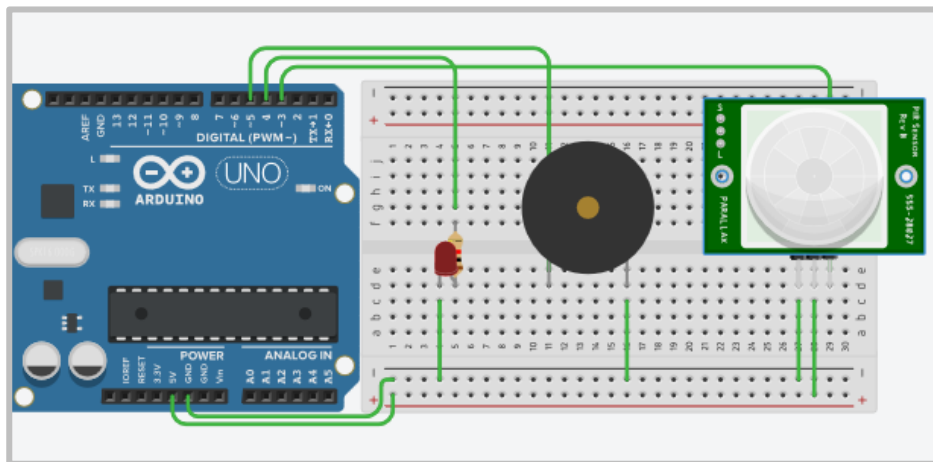


Figure 5-20. Wiring Diagram for an LED, PIR Sensor, and a Buzzer.

PIR Sensor with two indicators Code:

```
int sensor = 3;
int indicator = 4;
int buzzer = 5;

void setup() {
  pinMode(sensor, INPUT);
  pinMode(indicator, OUTPUT);
  pinMode(buzzer, OUTPUT);
}

void loop() {
  int state = digitalRead(sensor);
  if (state == 1) {
    digitalWrite(indicator, HIGH);
    tone(buzzer, 300);
  }

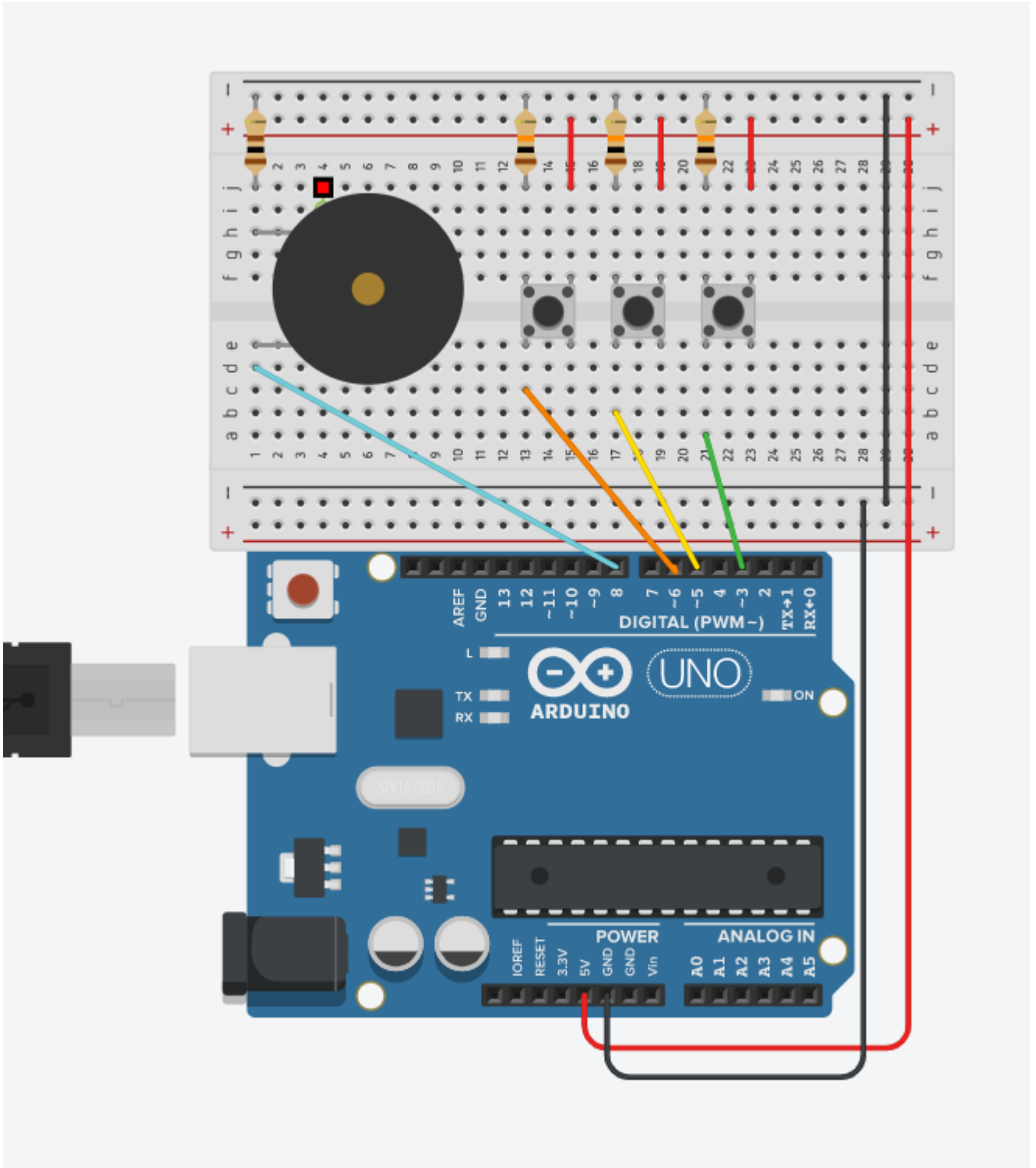
  else {
    digitalWrite(indicator, LOW);
    noTone(buzzer);
  }
}
```

1. Which of these is an example of an **output** device?
 - a. Switches
 - b. Temperature sensors
 - c. Buzzer
2. What **don't** we need to know to generate a sound?
 - a. Pitch
 - b. Frequency
 - c. Voltage Level
3. Which animal can hear the **largest frequency range**?
 - a. cat
 - b. bat
 - c. elephant
4. human can hear frequencies ranging between _____.
 - a. 0Hz to 20000Hz
 - b. 20Hz to 20000Hz
 - c. 20000Hz to 40000Hz
5. Which of these is **correct**:

- a. Elephants can only hear infrasound range
 - b. Humans can hear infrasound range
 - c. Dolphin can hear ultrasounds range
6. If the frequency of a signal **increases** the period will _____?
- a. decrease
 - b. increase
 - c. not change
7. which of these commands is **not used** to control a buzzer?
- a. tone (7, 1000)
 - b. noTone(7)
 - c. digitalWrite(7, HIGH)
8. in which frequency range can Arduino generate sound on a buzzer?
- a. 20Hz- 2000Hz
 - b. 20Hz- 20000Hz
 - c. 0Hz- 20000Hz

TASK BASED

1. Create a circuit using a buzzer. Program it in a way to generate a 2000Hz sound. Monitor the signal across the buzzer using an oscilloscope.
2. For the circuit below, write a program that makes each pushbutton generate different sound pitch, for instance 50Hz, 500Hz and 1000Hz.



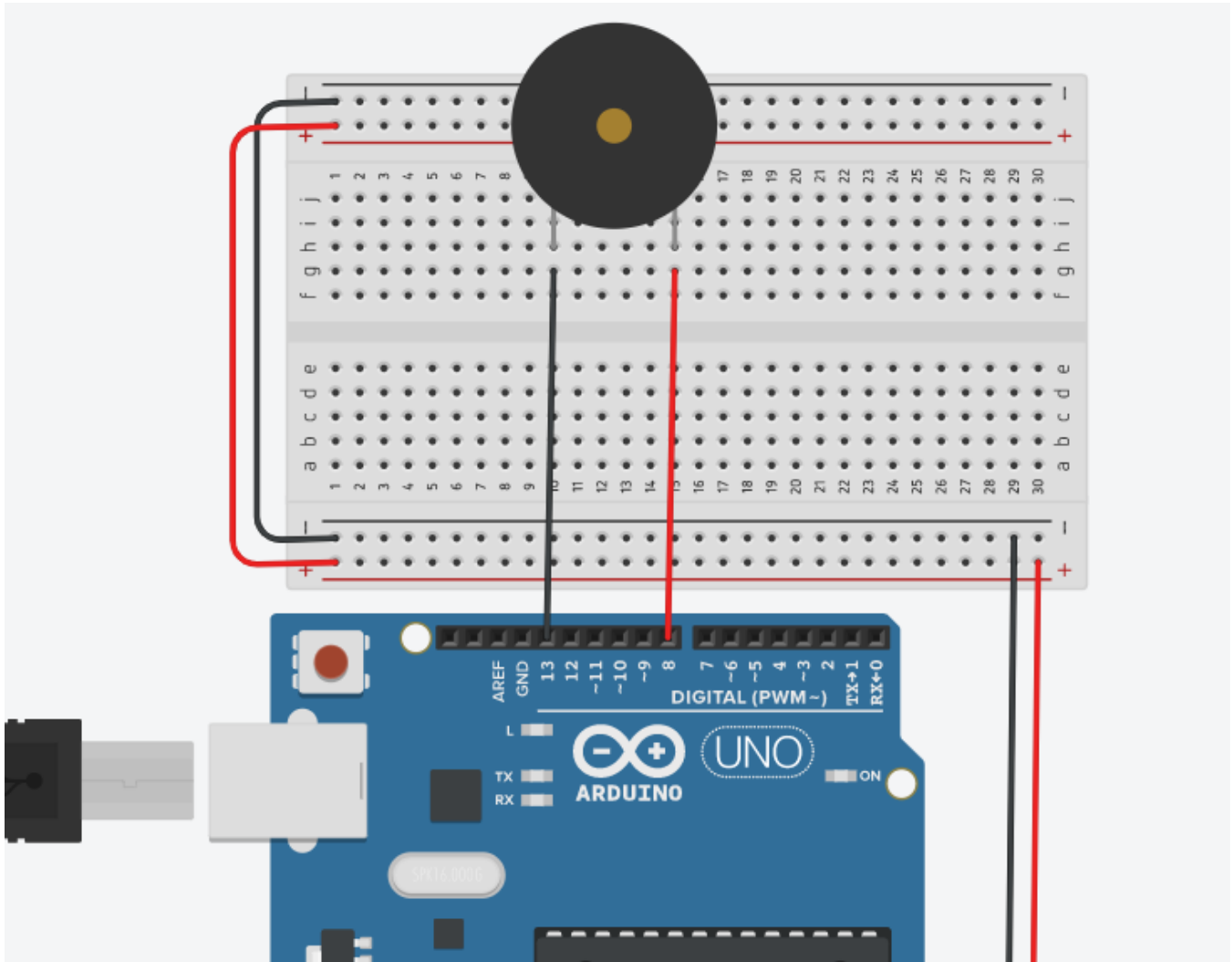
3. For the circuit in the previous question, add another pushbutton which can stop the sound of the buzzer.

- Design a circuit with a buzzer and a RGB LED. Write a program that generate 2 different sound frequency on the buzzer and displays 3 different colors on the RGB LED.

FIND THE ERROR(TROUBLESHOOT)

In this section, the objective is to find the error in the code snippet and the circuits.

- Find the problem in circuit below.



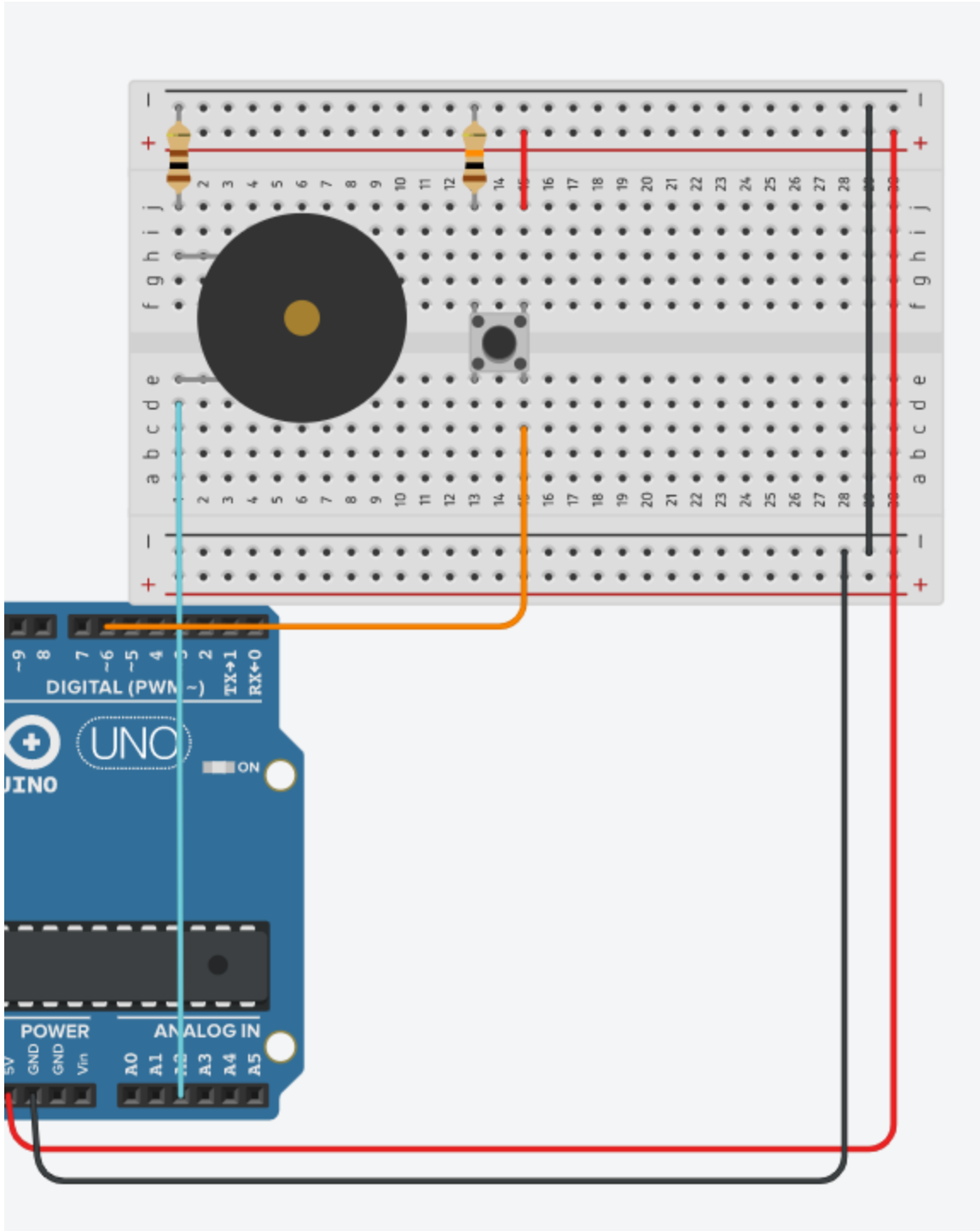
- The following code should generate 3 different sound pitches, but when you run the code only one frequency can be heard! Can you find the error and fix it?

```

19 |
20 |
21 void setup()
22 {
23     pinMode(8, OUTPUT);
24     pinMode(6, OUTPUT);
25     pinMode(7, OUTPUT);
26 }
27
28 void loop()
29 {
30     // turn off tone function for pin 8:
31     noTone(8);
32     tone(6, 880); // play tone 69 (A5 = 880 Hz)
33     // turn off tone function for pin 6:
34     noTone(6);
35     tone(7, 988); // play tone 71 (B5 = 988 Hz)
36     // turn off tone function for pin 7:
37     noTone(7);
38     tone(8, 1047); // play tone 72 (C6 = 1047 Hz)
39     delay(300); // Wait for 300 millisecond(s)
40 }

```

3. In the following circuit, whenever the button is pushed, the buzzer should generate a 1000Hz sound. But it does not work, can you find the problem and fix it?



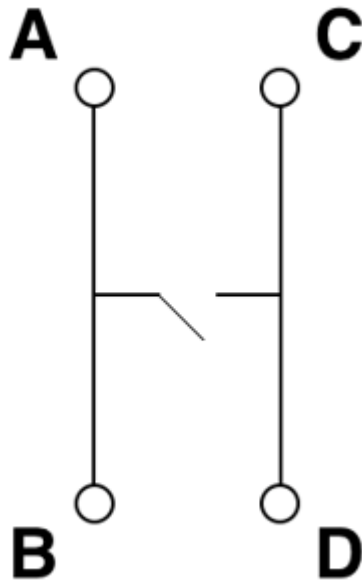
Page Break

Class 7: Design Challenge

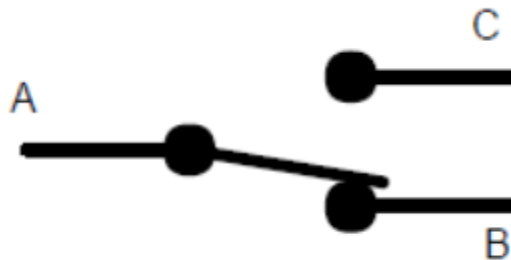
MULTIPLE CHOICE QUESTIONS

1. Which of these is correct?
 - a. 2V connected to a digital input is equivalent to 1 (5V)
 - b. 3V connected to a digital input is equivalent to 3V
 - c. 4.5V connected to an analog input is equivalent to 4.5V

2. Which of these can generate a 30% duty cycle PWM signal?
 - a. `analogWrite(10, 77)`
 - b. `analogWrite(10,130)`
 - c. `analogWrite(10, 60)`
3. which of these is the circuit symbol of pushbutton?



a.



b.



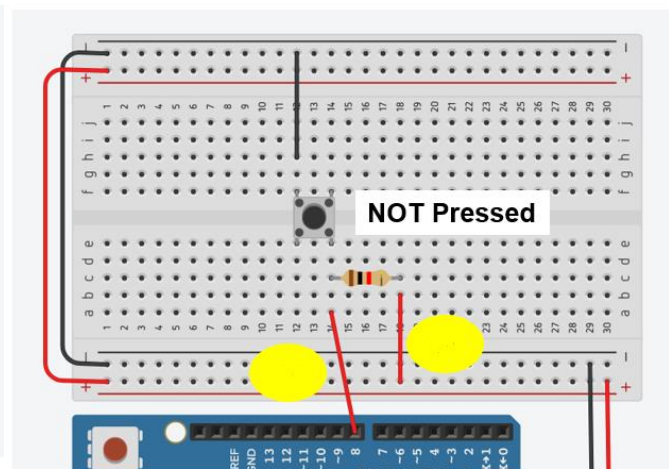
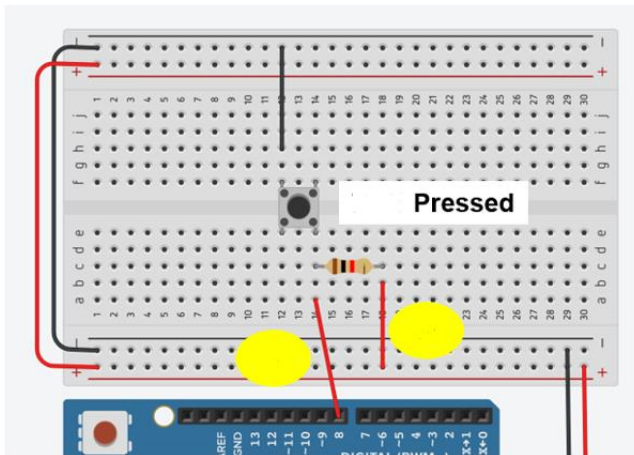
c.

4. What is the Pull up/pull down resistor functions?
 - a. limiting the current through the pins
 - b. limiting the voltage of the pins
 - c. protecting floating pins
5. RGB LEDs could be programmed as a _____.
 - a. Digital output.

- b. Analog output
- c. Digital or Analog output
- 6. What is the function of pinMode() command?
 - a. Power to digital output
 - b. Read analog input
 - c. Define pin
- 7. Which of these devices is an Output device?
 - a. Sensors
 - b. Keyboard
 - c. LCD

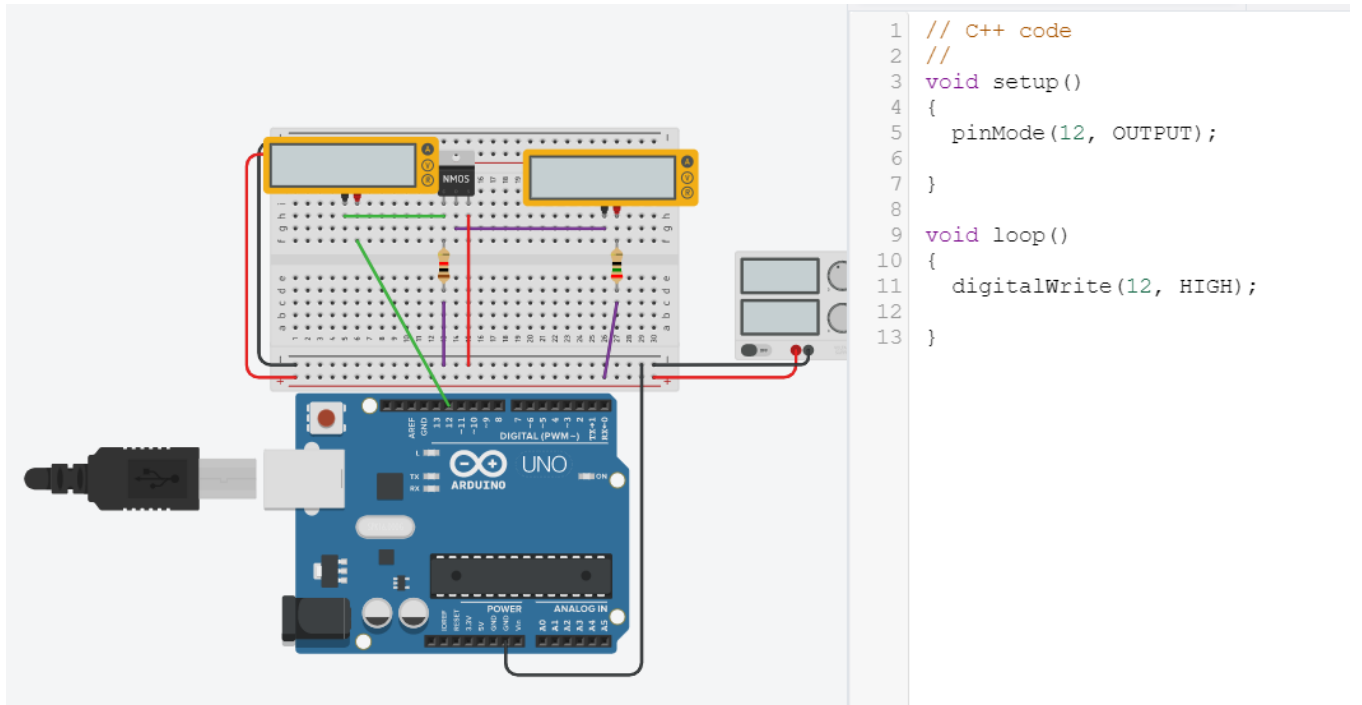
TASK BASED

1. Fill the yellow circles with the voltage of the nearby signals.



2. In the TinkerCAD, implement the circuit and code of the following picture. Using the resistor values in the table, change the value of the resistor connected to the drain of MOSFET and complete the table below: (external power supply = 12V)

Resistor Value	Gate Current	Drain current
1kR		
500R		
100R		
25R		

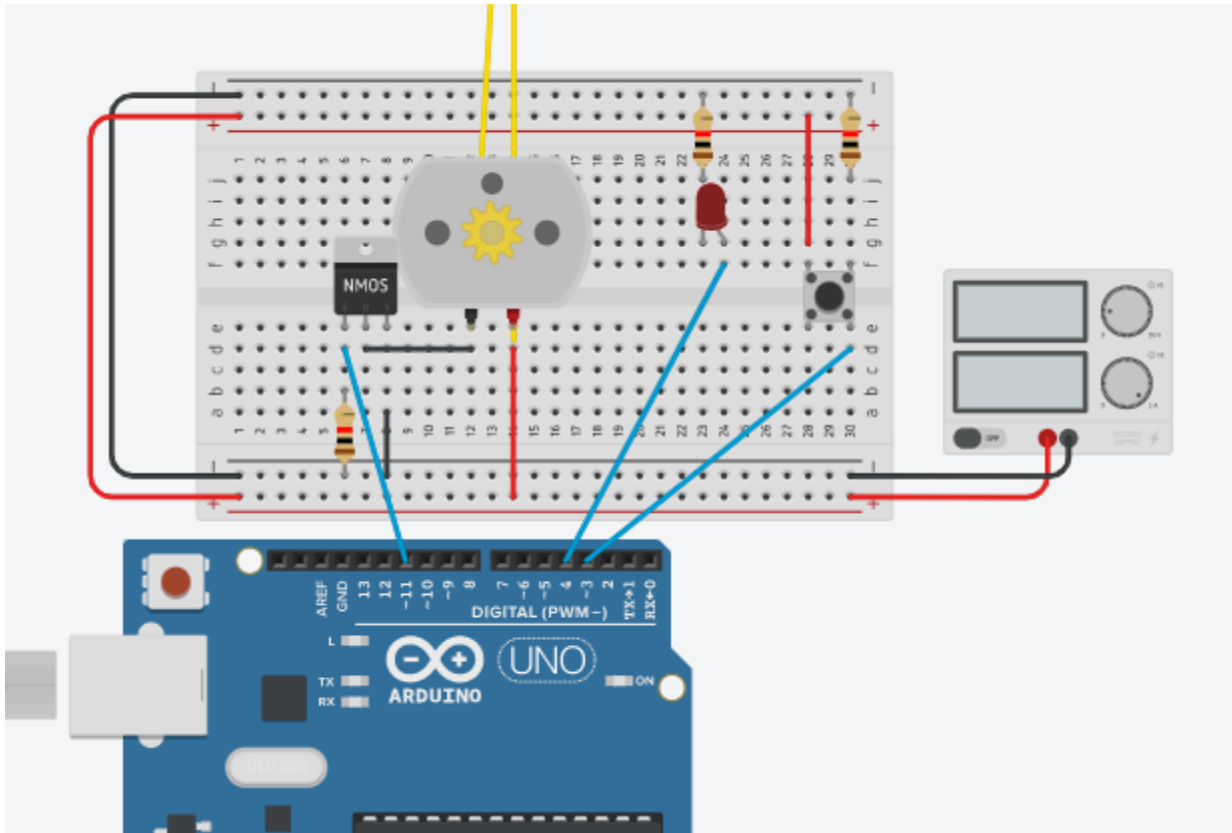


3. Design a circuit with a motor, buzzer and a pushbutton. Write a program which can run the motor with a 75% duty cycle, and when the pushbutton is pushed, the motor will stop spinning and the buzzer will generate a 4000Hz sound.

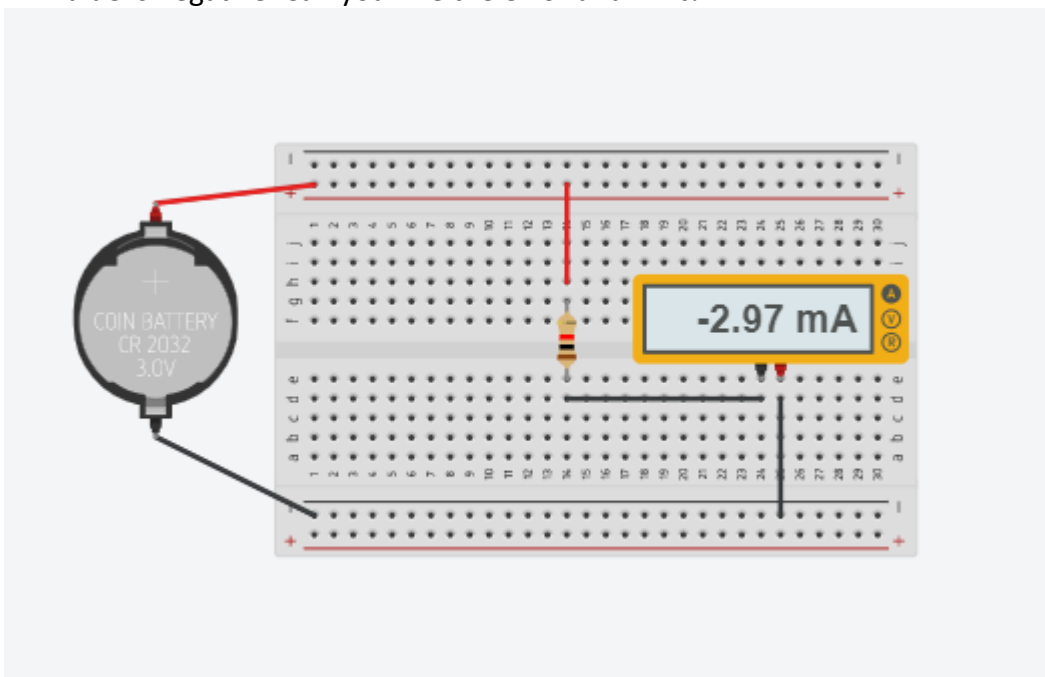
FIND THE ERROR(TROUBLESHOOT)

In this section, the objective is to find the error in the code snippet and the circuits.

1. The following circuit should control an LED and a motor, that is connected to an external power supply. But it does not work. Can you find the error and fix it?



2. In the following design we need to measure current value through the resistor, but the value is negative. Can you find the error and fix it?



3. The following code should generate a PWM signal when a pushbutton (connected to pin 2) is pushed. However, it doesn't work. Can you find the error and fix it?

```
1 // C++ code
2 //
3
4
5 void setup()
6 {
7     pinMode(8, OUTPUT);
8 }
9
10 void loop()
11 {
12
13     if (digitalRead(2) == HIGH) {
14         analogWrite(8, 125);
15     }
16
17     delay(10);
18 |
```

CHAPTER 6: WORKING WITH MOTORS

r

6.1 Parts You Will Learn

Now you will learn different types of motors and their functions. As previously mentioned, the sensors are the inputs of your circuit. The motors will be the outputs of the circuit. Think of the motors as your arms and legs. When your brain takes an input from the sensors such as eyes, your brain makes a process and then it outputs this process onto your legs to move forward.

For example, using an ultrasonic sensor, you can take the input of the distance between the circuit and the obstacle and using a servo motor, you can stop the movement or turn around to avoid the obstacle. The motors are the outputs of your circuit, so they are as crucial as sensors or any other part of the Arduino.

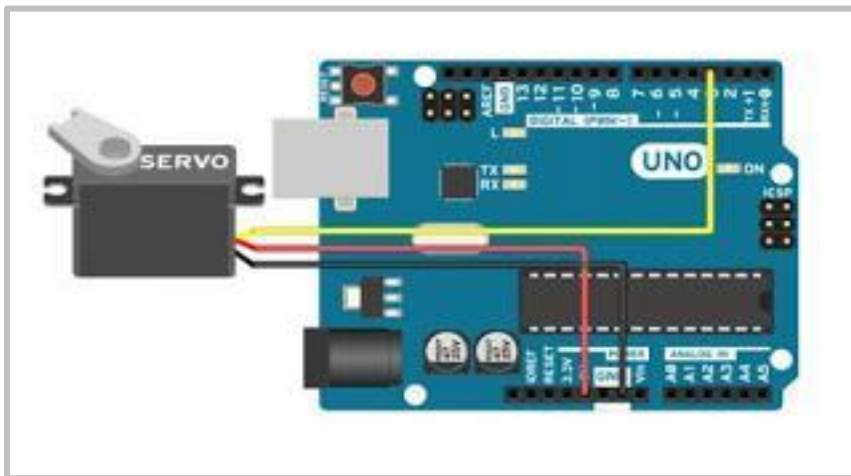
6.1.1 Servo Motor

Servo Motor Specifications	
Weights:	18g, including the cables
Motor speed:	0.14sec/60degree (4.8V)
Voltage:	4.8V - 6V
Stall Current	$\leq 850\text{mA}$
Weight	3G



Figure 6-1. Servo Motor.

A **servomotor** is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback.



Pin Connection

[red (VCC)] → [+5V]

[black (GND)] → [GND]

[yellow (PWM)] → [3]

Figure 6-2. Wiring Diagram of a Servo Motor and the Pin Connections on an Arduino.

Servo Motor Code Example:

```
#include <Servo.h>

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

void setup()
{
  myservo.attach(3); // attaches the servo on pin 3 to the servo object
}
```

```

void loop()
{
  for (pos = 0; pos <= 180; pos += 1)
  {
    // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);
    // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }

  for (pos = 180; pos >= 0; pos -= 1)
  {
    // goes from 180 degrees to 0 degrees
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }
}

```

6.1.2 DC Motor

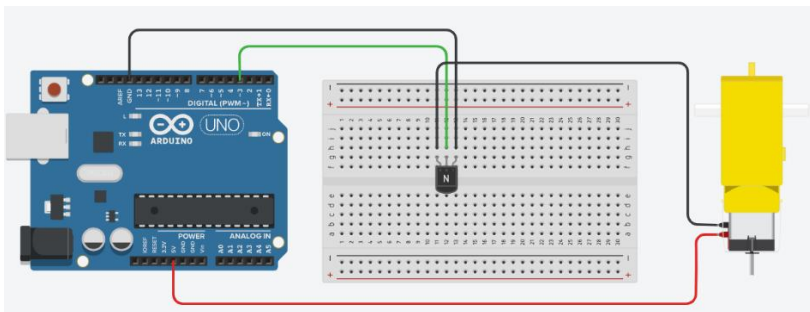
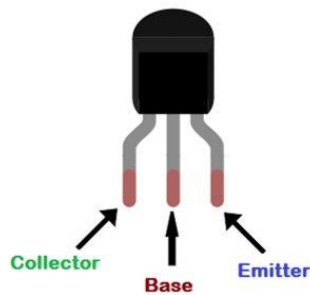


Figure 6-3. DC Motor.

DC Motor Specifications	
Rated Voltage:	6V DC
No load speed:	12000±15%rpm
No load current:	≤280mA
Stall Current	500mA
Operating voltage:	1.5-6.5V DC

A **DC motor** is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor.

6.1.2.1 Controlling DC Motors Method 1: Transistor



<i>Transistor</i>	<i>Other</i>
Emitter	- Motor
Base	3 (Arduino)
Collector	GND (Arduino)

<i>Motor</i>	<i>Other</i>
- Motor	Emitter (Transistor)
+ Motor	5V (Arduino)

DC Motor Code Example:

```

void setup(){
  pinMode(3, OUTPUT); //Set pin 3 as output (Connected to negative end of motor)

}

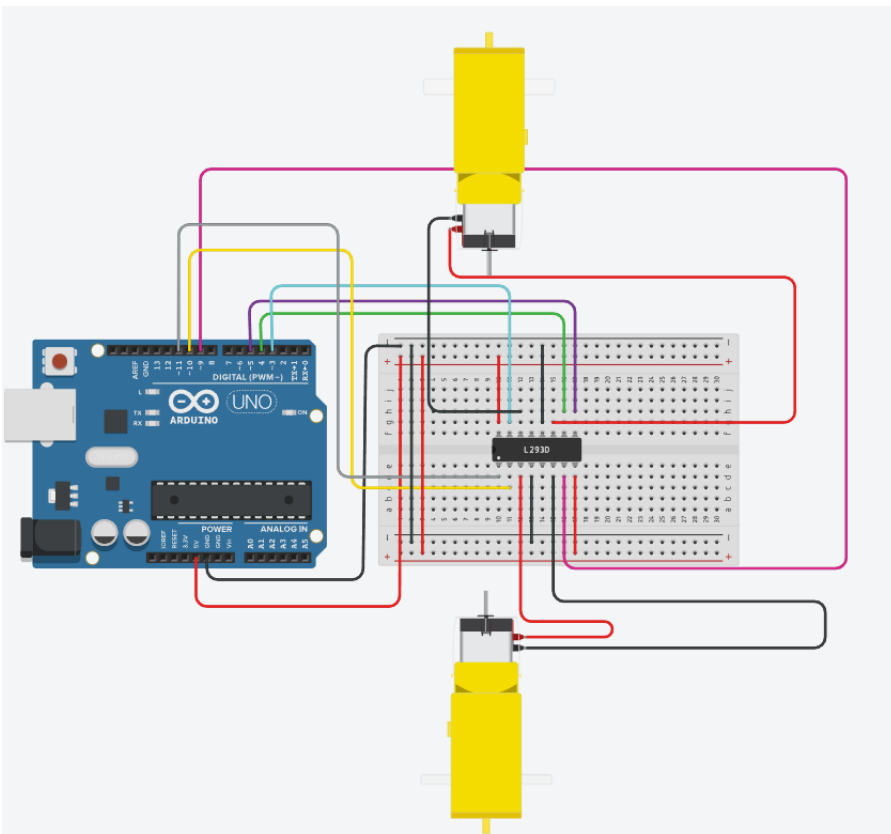
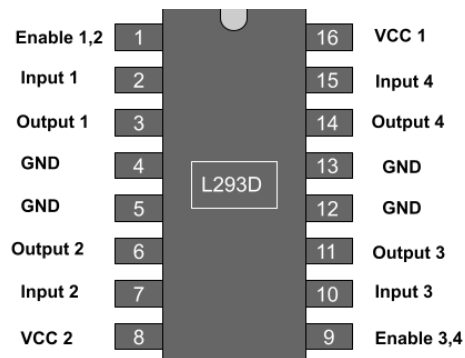
void loop(){

  analogWrite(3, 55); //Spin the motor slowly
  delay(1000);
  analogWrite(3, 105); //Speed motor up
  delay(1000);
  analogWrite(3, 155); //Speed motor up
  delay(1000);
  analogWrite(3, 205); //Speed motor up
  delay(1000)l
  analogWrite(3, 255); //Speed motor up
  delay(1000);
}
  
```

# of motors able to control	1
Speed Control (PWM):	Yes
Rotational Direction Control:	No
Max current (A)	0.5A
Operating Voltage	3V - 20V

6.1.2.2 Controlling DC Motors method 2: L293D Chip

The L293D is a 16 pin IC chip that can control up to 2 DC motors independently at once. The L293D chip works by taking input signals from the Arduino and outputting correlating signals to the DC motor (s). This chip can allow you to manipulate DC motors in many different ways, some of which include, making a DC motor spin clockwise and counterclockwise and controlling the speed of a DC motor.



<i>L293D Chip</i>	<i>Other</i>
Enable 1,2	11 (Arduino)
Input 1	10 (Arduino)
Output 1	+ (Motor 2)
GND	GND
Output 2	- (Motor 2)
Input 2	9 (Arduino)
VCC 2	+
VCC 1	+
Input 4	3 (Arduino)
Output 4	- (Motor 1)
GND	GND
Output 3	+ (Motor 1)
Input 3	4 (Arduino)
Enable 3,4	5 (Arduino)

L293D Motor Code Example:

```
void setup() {
  pinMode(3,OUTPUT); //set pin 3 as output (connected to input 4 on L293D chip)
  pinMode(4, OUTPUT); //set pin 4 as output (connected to input 3 on L293D chip)
  pinMode(5, OUTPUT); //set pin 5 as output (connected to enable 3,4 on L293D
chip)
  pinMode(9, OUTPUT); //set pin 9 as output (connected to input 2 on L293D chip)
  pinMode(10, OUTPUT); //set pin 10 as output (connected to input 1 on L293D chip)
  pinMode(11, OUTPUT); //set pin 5 as output (connected to enable 1, 2 on L293D)
}

void loop() {
  //following code uses PWM to spin motor at full speed and gradually slow down

  digitalWrite(3,HIGH); //spin first motor in one direction
  digitalWrite(4,LOW);
  analogWrite(5,255); //spin first motor at full speed
  delay(1000);
  analogWrite(3,180); //slow first motor down
  delay(1000);
  analogWrite(4,128); //slow first motor down
  delay(100);
  analogWrite(5, 50); //slow fist motor down
  delay(100);
  analogWrite(5,20); //slow first motor down

  digitalWrite(9,HIGH); //spin second motor one direction
```

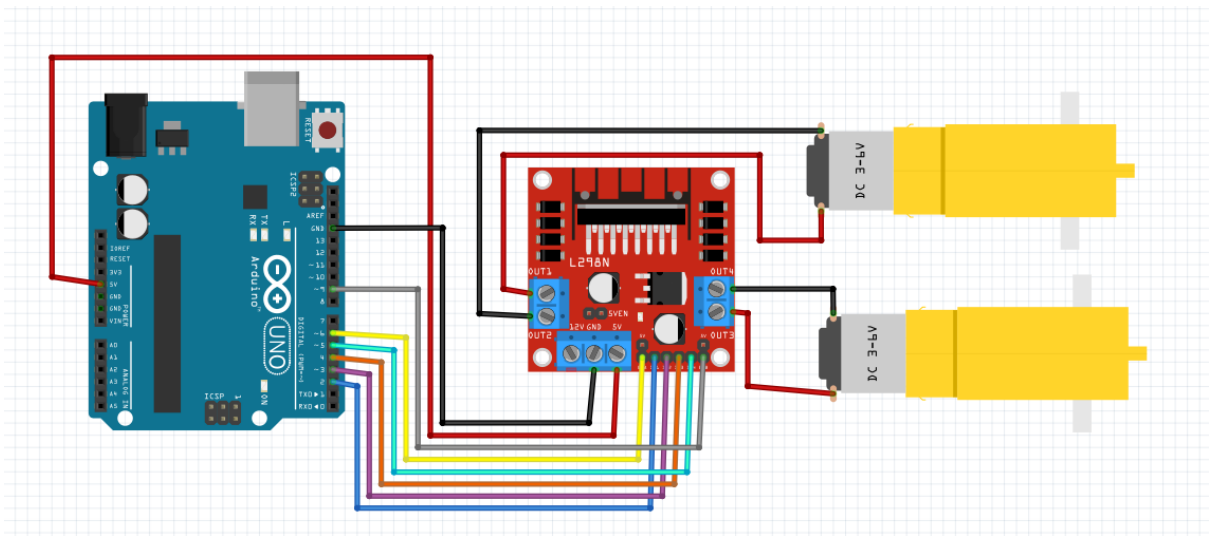
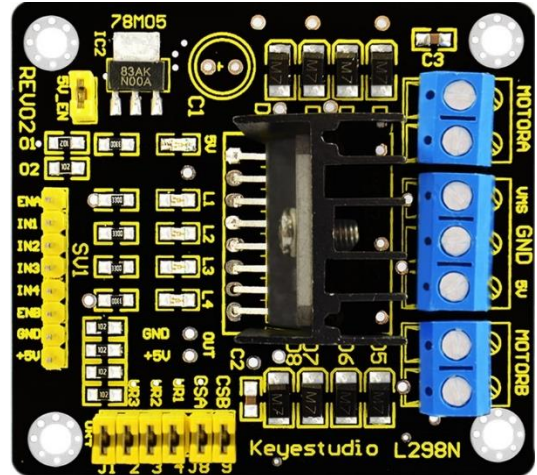
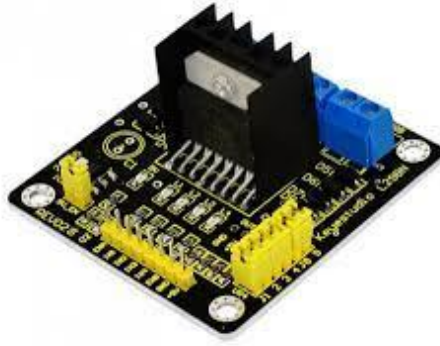
```
digitalWrite(10,LOW);
analogWrite(11,255); //spin second motor at full speed
delay(1000);
analogWrite(9,180); //slow second motor down
delay(1000);
analogWrite(10,128); //slow second motor down
delay(100);
analogWrite(11, 50); //slow second motor down
delay(100);
analogWrite(11,20); //slow second motor down

}
```

# of motors able to control	2
Speed Control (PWM):	Yes
Rotational Direction Control:	Yes
Max current (A)	0.6A
Operating Voltage	4.5V-36V

6.1.2.3 Controlling DC Motors method 3: L298N Driver

The L298N driver is a motor driver that can control up to 2 DC motors simultaneously. The driver can use H-Bridge to control the rotational direction or PWM to control the speeds of the 2 DC motors.



<i>L298N Driver</i>	<i>Other</i>
GND	GND (Arduino)
5V	5V (Arduino)
ENA	6 (Arduino)
IN1	2 (Arduino)
IN2	3 (Arduino)
IN3	4 (Arduino)
IN4	5 (Arduino)
ENB	9(Arduino)
+ Motor A	+ Motor 1
- Motor A	- Motor 1
+ Motor B	+ Motor 2
- Motor B	- Motor 2

L298N Motor Code Example:

```

void setup() {
  pinMode(2, OUTPUT); //Set pin 2 as output (connected to IN1 on L298N)
  pinMode(3, OUTPUT); //Set pin 3 as output (connected to IN2 on L298N)
  pinMode(4, OUTPUT); //Set pin 4 as output (connected to IN3 on L298N)
  pinMode(5, OUTPUT); //Set pin 5 as output (connected to IN4 on L298N)
  pinMode(6, OUTPUT); //Set pin 6 as output (connected to ENA on L298N)
  pinMode(9, OUTPUT); //Set pin 9 as output (connected to ENB on L298N)
}

void loop() {

  digitalWrite(2, HIGH); //Make first motor spin in one direction

  analogWrite(6, 55); //Spin motor slowly
  delay(1000);
  analogWrite(6, 105); //Speed motor up
  delay(1000);
  analogWrite(6, 155); //Speed motor up
  delay(1000);
  analogWrite(6, 205); //Speed motor up
  delay(1000);
  analogWrite(6, 255); //Speed motor up
  delay(1000);
  digitalWrite(2, LOW); //Stop first motor from spinning

  digitalWrite(4, HIGH); //Make second motor spin in one direction

```

```

analogWrite(9, 55); //Speed motor up
delay(1000);
analogWrite(9, 105); //Speed motor up
delay(1000);
analogWrite(9, 155); //Speed motor up
delay(1000);
analogWrite(9, 205); //Speed motor up
delay(1000);
analogWrite(9, 255); //Speed motor up
delay(1000);
digitalWrite(4, LOW); //Stop second motor from spinning

}

```

# of DC motors able to control	2
Speed Control (PWM):	Yes
Rotational Direction Control:	Yes
Max Current (A)	2A
Operating Voltage	5V-35V

6.1.2.4 Controlling DC Motors Method 3: Arduino Motor Shield

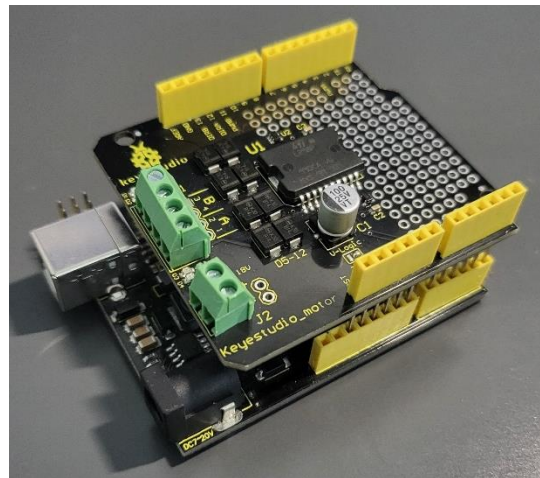
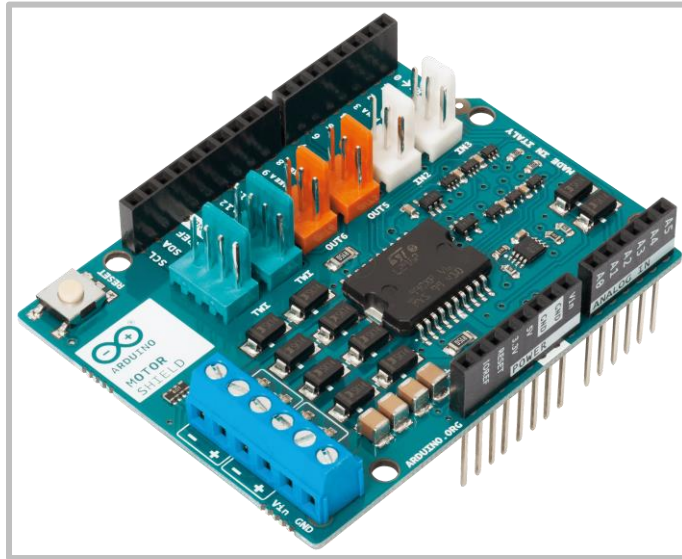
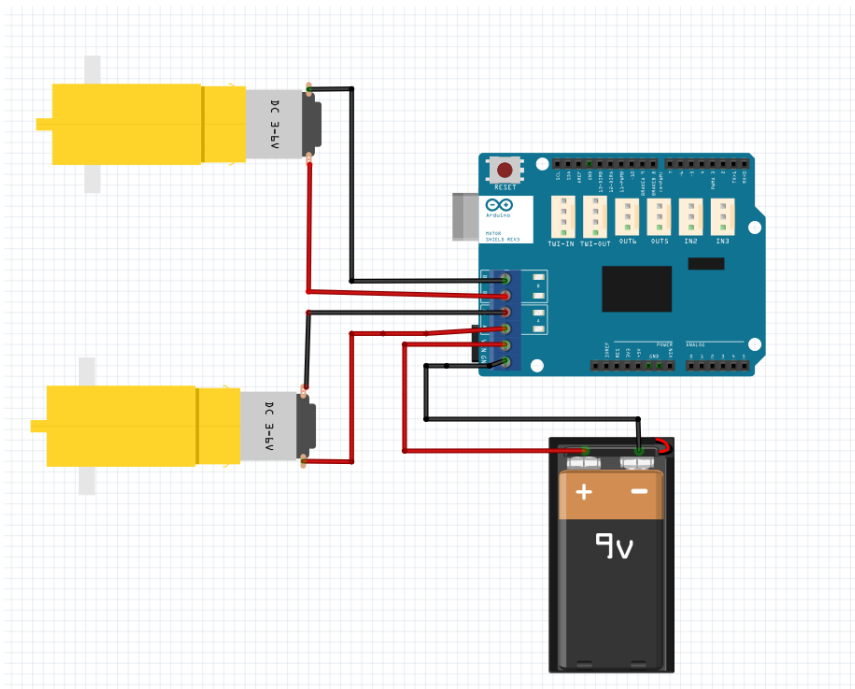


Figure 7-5. Arduino Motor Shield.

Note: Arduino Motor shields are modular circuit boards that you can attach on top of an Arduino board to add additional functionality.

The Arduino motor Shield is a dual full-bridge driver designed to drive inductive loads such as relays, DC and stepping motors. It lets you drive two DC motor with your Arduino, controlling the speed and direction of the vehicle. The shield is also TinkerKit compatible, meaning that you can quickly create projects by plugging TinkerKit modules to the board.



<i>Motor Shield</i>	<i>Other</i>
A+	+ Motor 1
A -	- Motor 1
B+	+ Motor 2
B -	- Motor 2
+	+ External Power Supply
-	- External power supply

Motor Shield Code Example:

```
void setup()
{
  pinMode(12, OUTPUT); //Set pin 12 as output
  pinMode(13, OUTPUT); //Set pin 11 as output
}

void loop()
{
  digitalWrite(12,HIGH); //Spin first motor in one direction

  analogWrite(3, 55); //Spin first motor slowly
  delay(1000);
  analogWrite(3,105); //Spin first motor slowly
  delay(1000);
  analogWrite(3, 155); //Speed up first motor
  delay(1000);
  analogWrite(3, 205); //Speed up first motor
  delay(1000);
  analogWrite(3, 255); //Speed up first motor
  delay(1000);

  digitalWrite(13, HIGH); // Spin second motor in one direction

  analogWrite(11, 55); //Spin second motor slowly
  delay(1000);
  analogWrite(11,105); //Speed up second motor
  delay(1000);
```

```

analogWrite(11, 155); //Speed up second motor
delay(1000);
analogWrite(11, 205); //Speed up second motor
delay(1000);
analogWrite(11, 255); //Speed up second motor

}
    
```

# Of DC motors able to control	2
Speed Control (PWM):	Yes
Rotational Direction Control:	Yes
Max Current (A)	4A (2A per channel)
Operating Voltage	7V - 18V

6.1.3 Stepper Motor

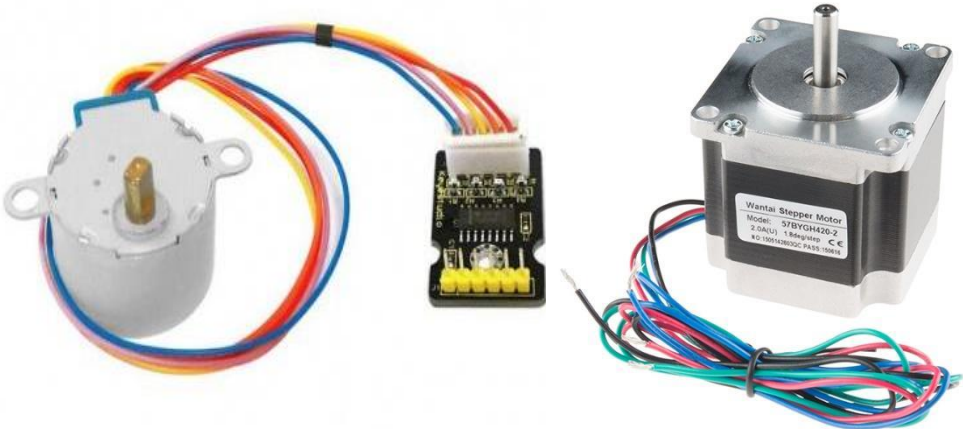
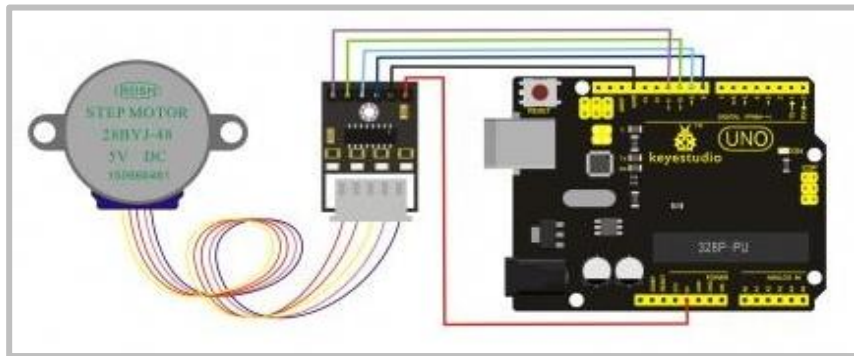


Figure 6-5. Stepper Motors.

A **stepper motor** (step motor or stepping motor) is a brushless DC electric motor that divides a full rotation into some equal steps. The motor's position can then be commanded to move and hold at one of these steps without any position sensor for feedback, if the motor is carefully sized to the application in respect to torque and speed.

Typically, a motor controller is required to control the pulse rate to the stepper (as demonstrated in the diagram below).



<i>Stepper Motor</i>	<i>Arduino</i>
STEP	8,9,10,11
VCC	5V
GND	GND

Figure 6-6. Stepper Motor wiring diagram and Pin connections for the diagram.

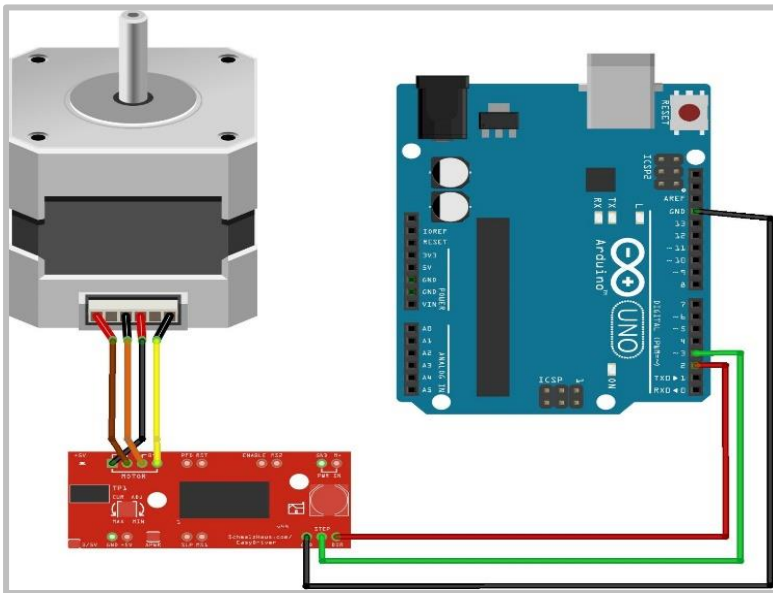
Stepper Motor Code Example:

```
#include <Stepper.h> // Include the Stepper library
#define STEPS 100 // Define STEPS

Stepper stepper(STEPS, 8, 9, 10, 11); // Set pins 8,9,10,11,STEPS as
//stepper
int previous = 0;
```

```
void setup()
{
  stepper.setSpeed(90); // Set stepper speed as 90
}

void loop()
{
  int val = analogRead(0);
  stepper.step(val - previous);
  previous = val;
}
```



<i>Stepper Motor</i>	<i>Arduino</i>
DIR (red)	2
STEP (green)	3
GND (black)	GND

Figure 6-7. Another Version of a Stepper Motor wiring diagram and Pin connections for the diagram.

STEPPER Motor Code Example:

```
#include <Stepper.h> // Include the header file

#define STEPS 32

Stepper stepper(STEPS, 2, 3);
int val = 0; // Set val as 0

void setup()
```

```

{
  Serial.begin(9600);
  stepper.setSpeed(200); // Speed of the stepper is 200
}

void loop()
{
  if (Serial.available() > 0)
  {
    val = Serial.parseInt();
    stepper.step(val);
    Serial.println(val); //for debugging
  }
}

```

6.2 Servo Library

Servo Library allows an Arduino board to control servo motors. Servos have integrated gears, and shafts that can be precisely controlled. The shaft is positioned at various angles, usually between 0 and 180 degrees.

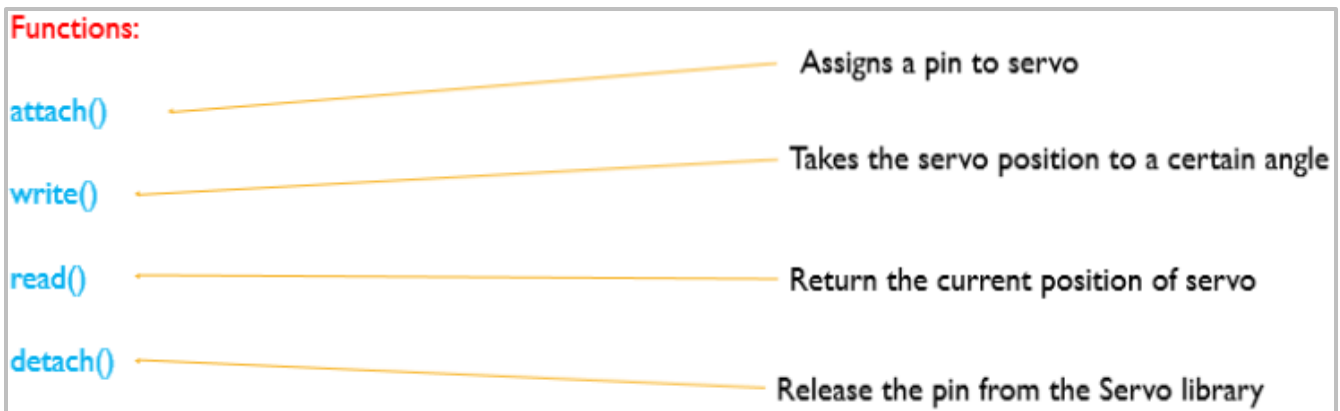


Figure 6-8. Functions apart of the Servo Library and what they can do.

Library name:
`#include <Servo.h>`

Declaration:

```
Servo myServo;
```

Functions:

```
attach(pinNumber);
detach(pinNumber);
write(angle);
read();
```

If you need more information or you still have questions, you can visit the link below:

Reference: <https://www.arduino.cc/en/Reference/Servo>

Fun Fact: Servomotors are used in applications such as robotics, CNC machinery or automated manufacturing.

6.3 Practice Examples

Servo Motor Practice Example

1) Servo Motor

Movement

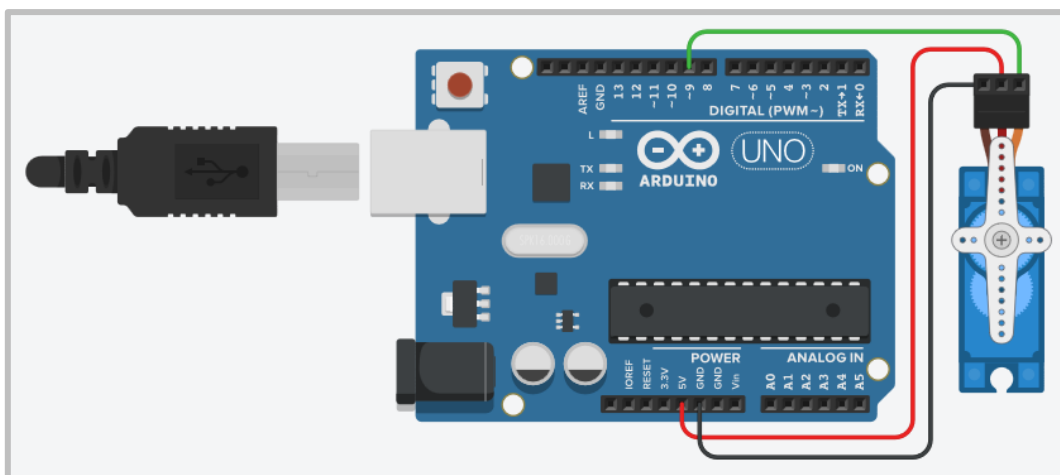


Figure 6-9. Servo Motor wiring diagram.

Servo Motor Movement Example Code:

```
#include <Servo.h>

Servo myservo;
Servo servo_9;

void setup()
{
  myservo.attach(9); //Attach servo to the pin 9
}

void loop()
{
  myservo.write(0); //Set the angle of the servo to 0
  delay(1000); //Wait 1 second
  myservo.write(45); //Set the angle of the servo to 45
  delay(1000); //Wait 1 second
  myservo.write(90); //Set the angle of the servo to 90
  delay(1000); //Wait 1 second
  myservo.write(135); //Set the angle of the servo to 135
  delay(1000); //Wait 1 second
  myservo.write(180); //Set the angle of the servo to 180
  delay(1000); //Wait 1 second
}
```

2) Control Servo Motor with Potentiometer

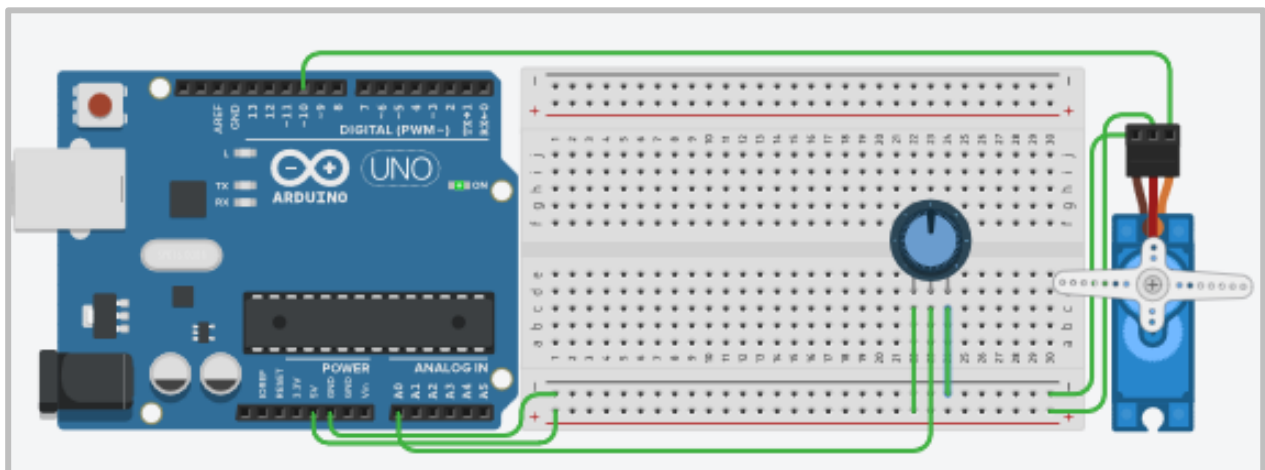


Figure 6-10. Servo Motor wiring diagram with a Potentiometer attached as well.

Servo Motor Movement Example Code:

```
#include <Servo.h>

int potentiometer = 0;
int val;

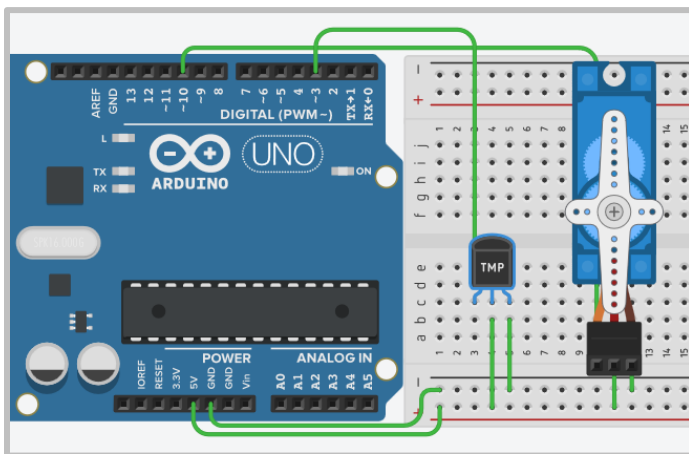
Servo myservo;

void setup()
{
  myservo.attach(10);
}

void loop()
{
  val = analogRead(potentiometer); //read from potentiometer
  val = map(val, 0, 1023, 180, 0); //convert the value to 180-based
  myservo.write(val); //control servo motor
}
```

6.4 Homework Questions

Q.1) Servo motor with a temperature sensor

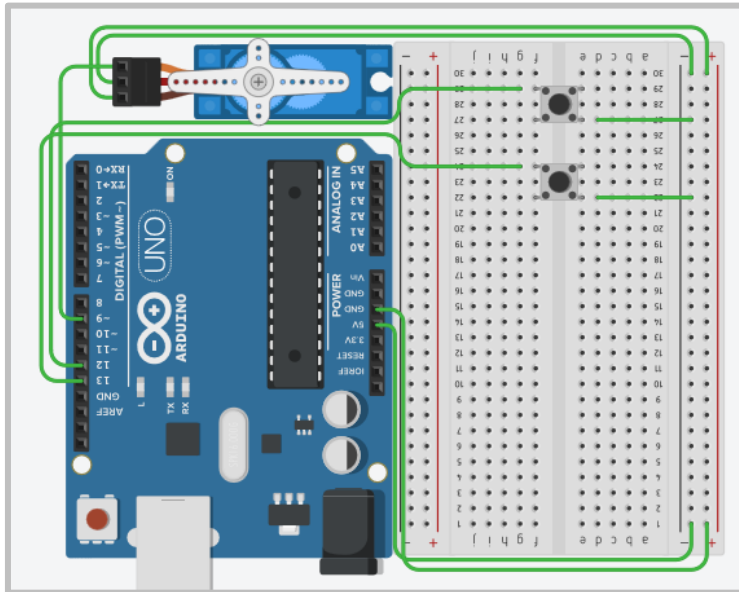


Build a code in which servo motor moves as temperature increases.

HINT: map() function will be required to convert the temperature value to the angle value.

Figure 6-11. Servo Motor wiring diagram with a temperature sensor attached as well.

Q. 2) Servo motor controlled by two push buttons



Build the code which servo motor changes its angle by + or – 20 degrees when the button is pressed. Left one should be negative, and the right one should be positive.

Figure 6-12. Servo Motor wiring diagram controlled by 2 push buttons.

1. The **resistor** cannot be used as a ____?
 - i. Pull down/up to protect floating pins
 - j. Current limiter through LED
 - k. Voltage source
2. Arduino behaves as a ____ When it comes to **measuring the input signal**?
 - a. Switch
 - b. Voltage Regulator
 - c. Multimeter
3. Which of these can only be monitored by **analog input of Arduino**?
 - a. Temperature sensor
 - b. Switches
 - c. PIR sensor
4. Which of these is **not** correct:
 - a. 2.6V is equivalent to 1 in digital signals.
 - b. 2.6V is equivalent to 1 in analog signals
 - c. 2.6V is equivalent to 2.6V in analog signals
5. What is the **voltage level of analog signals** in Arduino?
 - a. 0V-3.3V
 - b. 0V-5V

- c. 0V-12V
- 6. Which of these is **not correct** about analog signals in Arduino?
 - a. The analog value measured by Arduino converts to 0 to 1/10123.
 - b. In the analogWrite() command, the values can range from 0 to 255.
 - c. The analog value measured by Arduino converts to 0 to 1023
- 7. The **PWM pins** can be used as _____.
 - a. Digital input pins
 - b. Analog output pins
 - c. Power pins
- 8. Which of these PWM signals provide **more power** to a motor?

75% duty cycle



a.

25% duty cycle



b.

50% duty cycle



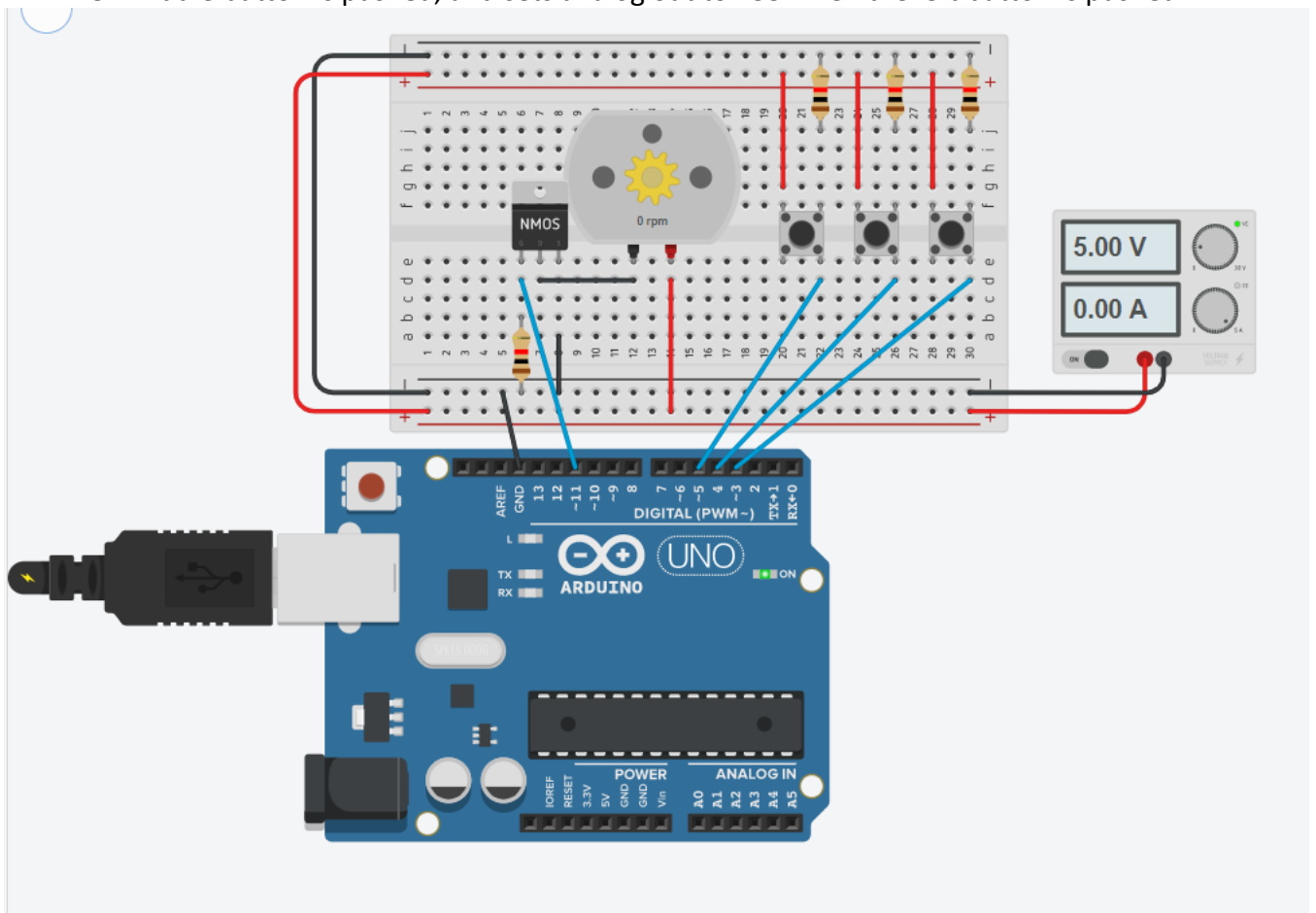
c.

- 9. Which of these defines the **duty cycle**?
 - a. On and off time of the signal
 - b. Voltage level
 - c. Current amount
- 10. The transistors **cannot be used for**:
 - a. Amplify electrical power
 - b. Switch electronic signals

c. Regulating voltage

TASK BASED

1. Explain why the motor **cannot** be driven directly using Arduino pins. What could be the solution to this problem?
2. In the TinkerCAD, connect a DC motor to a 5V power supply and write down the current consumption of the motor. Swap the +5V and 0V connections and report your observations.
3. Design a circuit that can appropriately drive a DC motor by an Arduino.
4. For the circuit below, we need to control the speed of the motor. Write a program that sets the analog output to 50 when the right push button is pushed, sets analog out to 0 when middle button is pushed, and sets analog out to 255 when the left button is pushed.



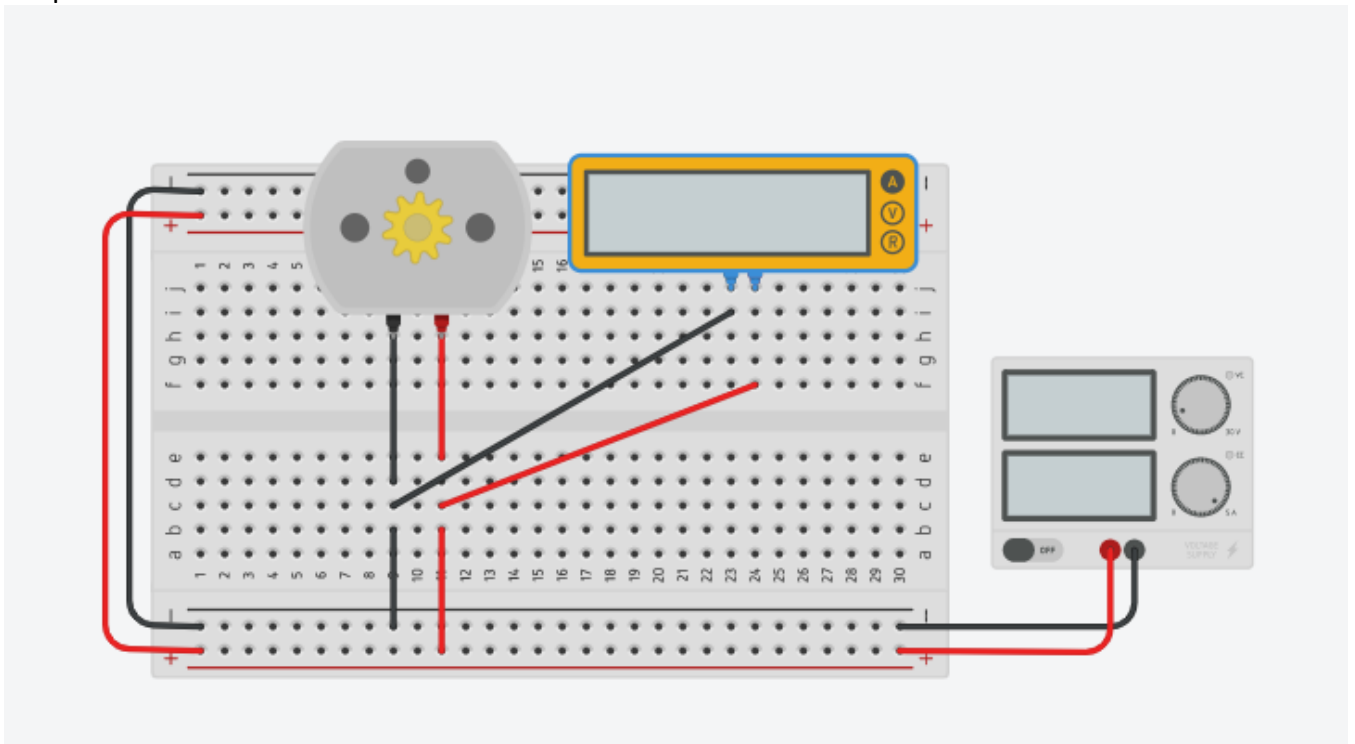
FIND THE ERROR(TROUBLESHOOT)

In this section, the objective is to find the error in the code snippet and the circuits.

1. For the given voltage and resistance, the current is calculated. Is it correct? If not, what is the correct current?

Resistor	Current	Voltage
2.2K	2 A	5V

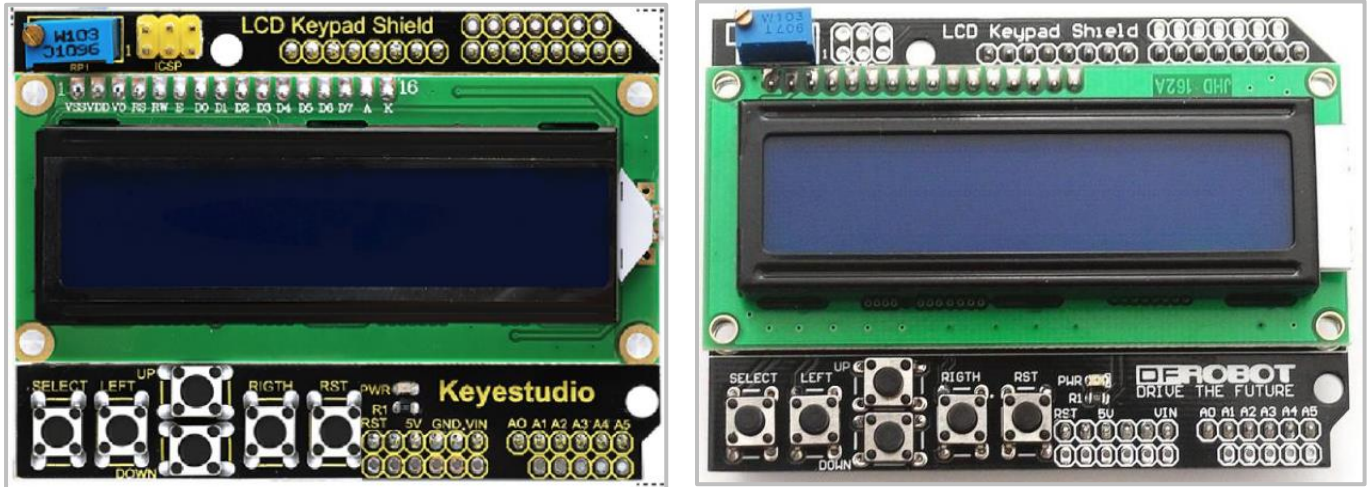
2. As it can be seen in the picture below, we are trying to measure the current flow through the motor, but the multimeter is not displaying the correct value. Can you find the problem? How can this be solved?



Page Break

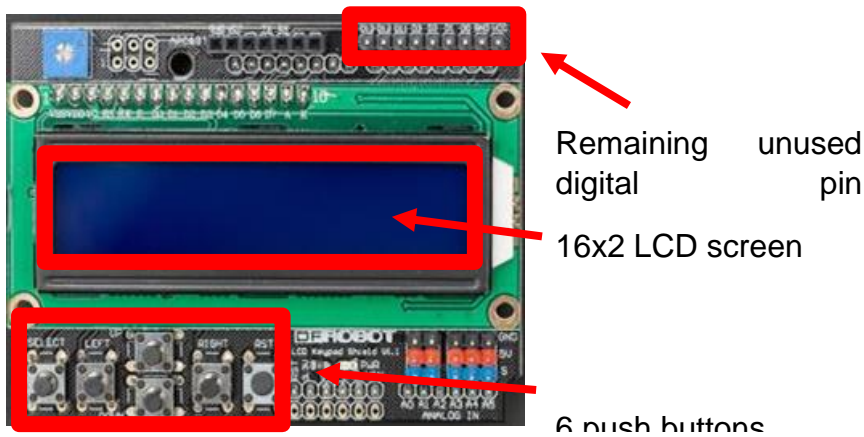
CHAPTER 7: Display Modules

7.1 LCD Shield



Includes a 2x16 LCD and six momentary push buttons. Pins 4, 5, 6, 7, 8, 9 and ten are used to interface with the LCD. Just one Analog Pin 0 is used to read the five pushbuttons. The LCD shield supports contrast adjustment and back-lit on/off functions. It also exposes five analog pins with DFRobot colour code for easy analog sensor plugging and display. The onboard LED indicates power on.

https://wiki.keystudio.com/Ks0256_keyestudio_LCD1602_Expansion_Shield



This LCD Arduino shield has five keys — select, up, right, down and left which allow you to move through menus and make selections straight from one board attached to your Arduino project without requiring a massive tower of shields.

This design allows you to keep connecting sensors to the rest of the pins and use it for monitoring or menu selection with the push buttons even for gaming.

If you need more information or you still have questions, you can visit the link below:

Reference: <https://www.dfrobot.com/product-51.html>

7.2 LCD Library

LCD Library Information

Library name:

```
#include <LiquidCrystal.h>
```

Declaration:

```
LiquidCrystal lcd(8,9,4,5,6,7);
```

Functions:

```
begin(x,y);  
setCursor(x,y);  
print( data );  
clear();
```

If you need more information or you still have questions, you can visit the link below:

Reference: <https://www.arduino.cc/en/Reference/LiquidCrystal>

7.3 Practice Question

Blinking LCD Practice Example

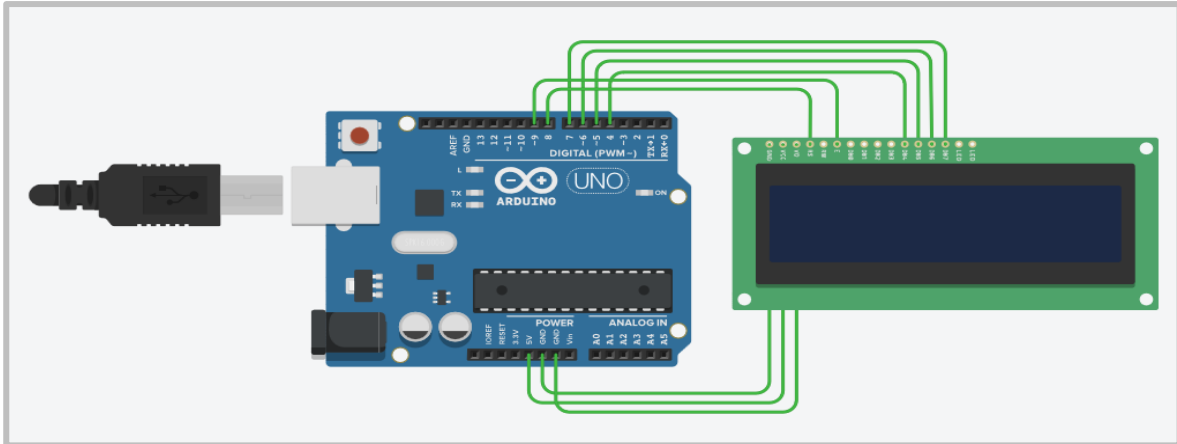


Figure 7-6. The Arduino LCD Shield wiring diagram.

Solution Code:

```
#include <LiquidCrystal.h>

LiquidCrystal mylcd(8, 9, 4, 5, 6, 7);

void setup()
{
  mylcd.begin(16, 2);
}

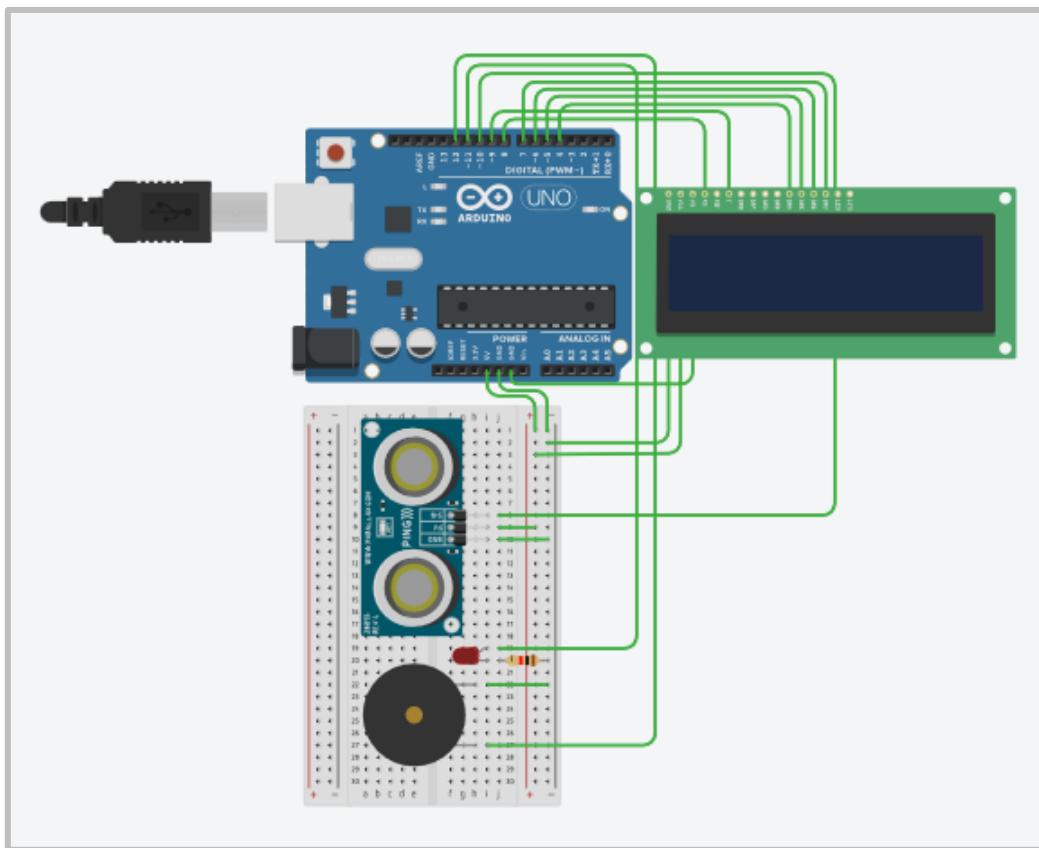
void loop()
{
  mylcd.setCursor(3, 0);
  mylcd.print("I LEARNT");
  mylcd.setCursor(6, 1);
  mylcd.print("LCD");
  delay(1000);
  mylcd.clear();
  delay(1000);
}
```

7.4 Homework Question

Q.1. [Challenge]

1) Build this circuit as shown below.

2) Write the code that displays the distance between the sensor and a detected object in centimeters. The LED turns on to display “Close” when the distance between the sensor and an object is less than 40cm. When the object is less than 25cm away, The Buzzer works on 500hz and the LCD displays “Too Close”.



Solution Code:

```
#include<NewTone.h>
#include <LiquidCrystal.h>

LiquidCrystal myLCD (8, 9, 4, 5, 6, 7);

int cm = 0;
```

```

int LED = 11;
int Buzzer = 12;

long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT); // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Sets the trigger pin to HIGH state for 10 microseconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  // Reads the echo pin, and returns the sound wave travel time in
  //microseconds
  return pulseIn(echoPin, HIGH);
}

void setup()
{
  myLCD.begin(16, 2);
  pinMode(LED, OUTPUT);
  pinMode(Buzzer, OUTPUT);
}

void loop()
{
  cm = 0.01723 * readUltrasonicDistance(10, 10);
  myLCD.setCursor(0, 0);
  myLCD.print(cm);
  myLCD.print("cm");

  if (cm < 40)
  {
    digitalWrite(LED, HIGH);
    myLCD.setCursor(0, 1);

    else if (cm < 25)
    {
      tone(Buzzer, 500);
      myLCD.print("Too Close");
    }

    else
    {
      myLCD.print("Close");
    }
  }
}

```

```

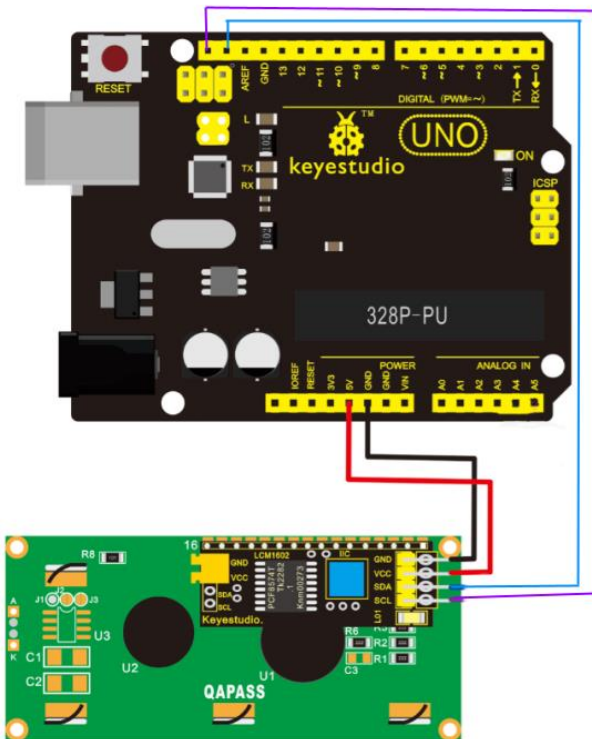
}
}
delay(1000);
myLCD.clear();
noTone(Buzzer);
}

```

7.5 I2C LCD



The I2C LCD is a simplified, easy-to-use module for displaying information. The I2C only has 4 pins, VCC, GND, SDA and SCL. **Note:** To use this module, the `LiquidCrystal_I2C.h` library will need to be downloaded and added to the Arduino IDE.



<i>I2C LCD</i>	<i>Arduino</i>
GND	GND
VCC	5V
SDA	SDA
SCL	SCL

I2C LCD Code Example:

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2); //set the LCD address to 0x27 (16 chars, 2
lines)

void setup(){
  lcd.init(); // initialize the lcd

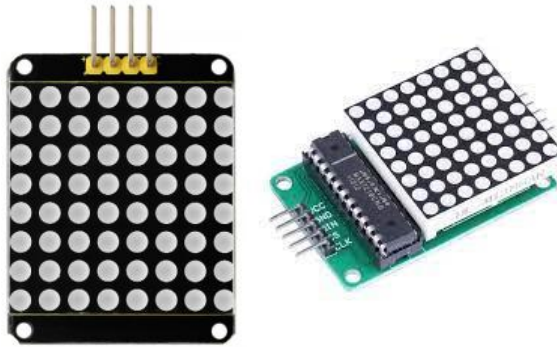
  lcd.backlight(); //turn backlight on

  lcd.setCursor(1,0); //Set cursor to starting position on line 1
  lcd.print("hello, ");
  lcd.setCursor(1,1); //Set cursor to starting position on line 2
  lcd.print("world!");
}

void loop(){

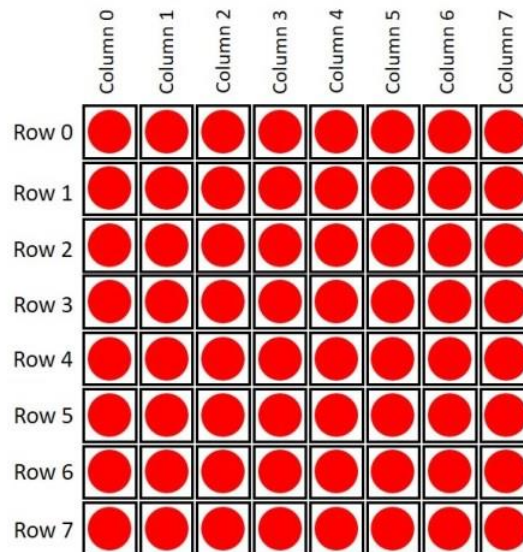
}
```

7.6 8X8 Dot Matrix

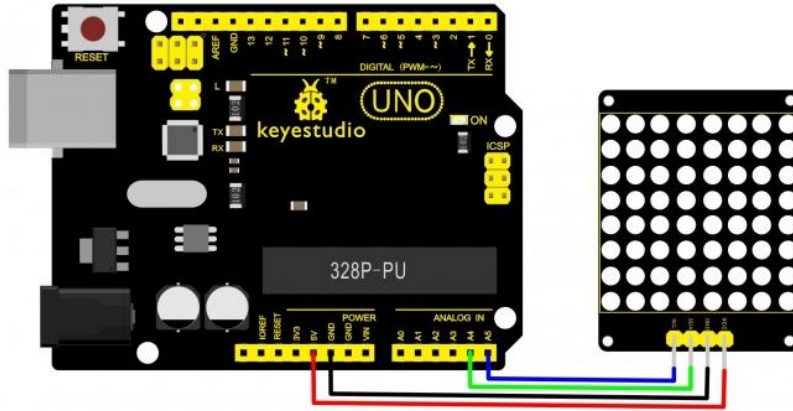


The 8x8 dot matrix is a display system that allows you to control up to 46 LEDs. Normally, to control that many LEDs at once with an Arduino, you will need to use a lot of pins. Thankfully, the 8x8 dot matrix allows us to do with only using 4 or 5 pins. **NOTE:** To use this module, the **LedControl** library will need to be downloaded and added to the Arduino IDE.

Example of how the rows and columns are numbered:

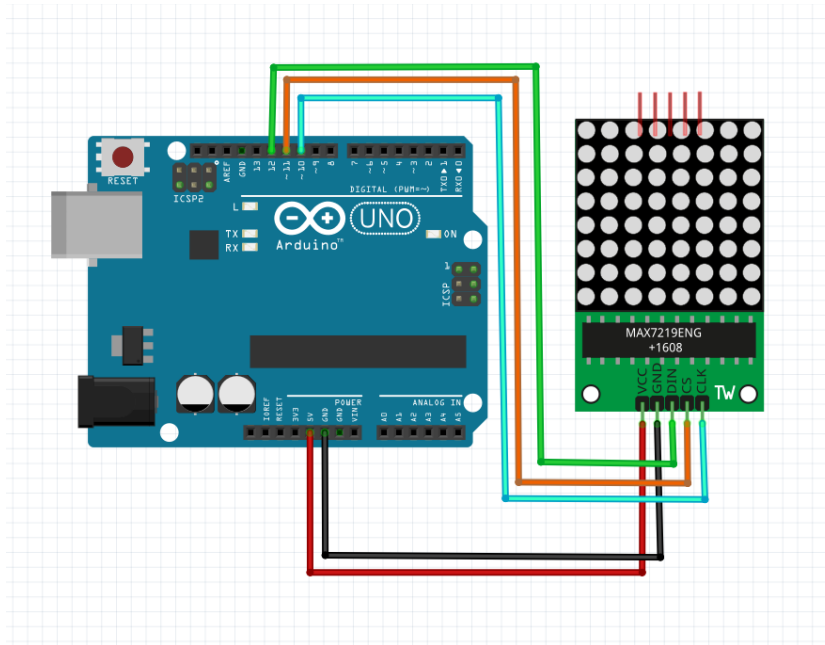


4 Pin Module Example:



<i>Dot Matrix</i>	<i>Arduino</i>
VCC	5V
GND	GND
SDA	A4
SCL	A5

5 pin Module Example:



<i>Dot Matrix</i>	<i>Arduino</i>
VCC	5V
GND	GND
DIN	12
CS	11
CLK	10

Dot Matrix 5 pin Code Example:

```
#include "LedControl.h"

LedControl lc =LedControl(12,10,11);

void setup() {

    lc.shutdown(0,false); //Turn on the Dot matrix
    lc.setIntensity(0,8); //set brightness to 8
    lc.clearDisplay(0); //clear the dot matrix display
}

void printLetter(byte arr[]){ //function to print out the letters onto the dot
matrix
    for(int i =0; i < 8; i++){
        lc.setRow(0,0,arr[0]); //print the letter array of 0
        lc.setRow(0,1,arr[1]); //print the letter array of 1
        lc.setRow(0,2,arr[2]); //print the letter array of 2
        lc.setRow(0,3,arr[3]); //print the letter array of 3
        lc.setRow(0,4,arr[4]); //print the letter array of 4
        lc.setRow(0,5,arr[5]); //print the letter array of 5
        lc.setRow(0,6,arr[6]); //print the letter array of 6
        lc.setRow(0,7,arr[7]); //print the letter array of 7

    }

}
```

```

void loop() {

    //NOTE: 1 = true, 0 = False

    byte H[]
    {B00000000,B00100100,B00100100,B00111100,B00100100,B00100100,B00100100,B00000000}
    ; //set byte values for the letter "H"

    byte E[] =
    {B00000000,B00111100,B00100000,B00111000,B00100000,B00100000,B00111100,B00000000}
    ; //set byte values for the letter "E"

    byte L[] =
    {B00000000,B00100000,B00100000,B00100000,B00100000,B00100000,B00111100,B00000000}
    ; //set byte values for the letter "L"

    byte O[] =
    {B00000000,B00111100,B01000010,B01000010,B01000010,B01000010,B00111100,B00000000}
    ; //set byte values for the letter "O"

    byte W[] =
    {B00000000,B10000010,B10010010,B01010100,B01010100,B00101000,B00000000,B00000000}
    ; //set byte values for the letter "W"

    byte R[] =
    {B00000000,B00111000,B00100100,B00100100,B00111000,B00100100,B00100100,B00000000}
    ; //set byte values for the letter "R"

    byte D[] =
    {B00000000,B00111000,B00100100,B00100010,B00100010,B00100100,B00111000,B00000000}
    ; //set byte values for the letter "D"

    printLetter(H); //call printLetter function with the array for the letter H
    delay(1000);
    printLetter(E); //call printLetter function with the array for the letter E
    delay(1000);
    printLetter(L); //call printLetter function with the array for the letter L
    delay(1000);
    printLetter(L); //call printLetter function with the array for the letter L
    delay(1000);
    printLetter(O); //call printLetter function with the array for the letter O

```

```
delay(1000);  
printLetter(W); //call printLetter function with the array for the letter W  
delay(1000);  
printLetter(O); //call printLetter function with the array for the letter O  
delay(1000);  
printLetter(R); //call printLetter function with the array for the letter R  
delay(1000);  
printLetter(L); //call printLetter function with the array for the letter L  
delay(1000);  
printLetter(D); //call printLetter function with the array for the letter D  
delay(1000);  
  
}
```

Dot Matrix 5 pin Code Example:

```

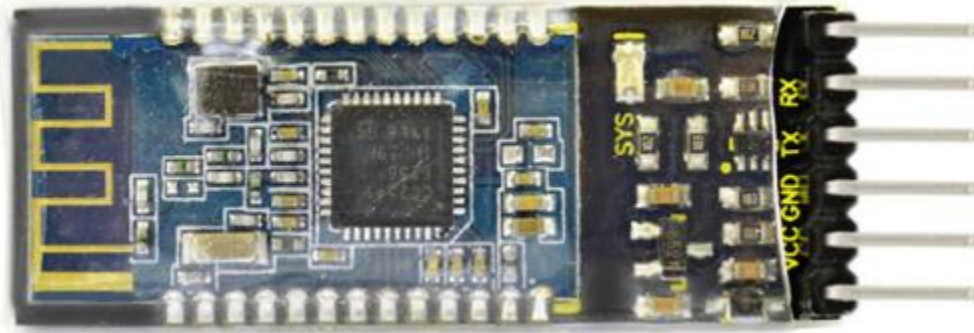
#include <Wire.h>
#include "Adafruit_LEDBackpack.h"
#include "Adafruit_GFX.h"
#ifndef _BV
#define _BV(bit) (1<<(bit))
#endif
Adafruit_LEDBackpack matrix = Adafruit_LEDBackpack();
uint8_t counter = 0;
void setup() {
  Serial.begin(9600);
  Serial.println("HT16K33 test");
  matrix.begin(0x70); // pass in the address
}
void loop() {
  // paint one LED per row. The HT16K33 internal memory looks like
  // a 8x16 bit matrix (8 rows, 16 columns)
  for (uint8_t i=0; i<8; i++) {
// draw a diagonal row of pixels

    matrix.displaybuffer[i] = _BV((counter+i) % 16) | _BV((counter+i+8) % 16) ;
  }
  // write the changes we just made to the display
  matrix.writeDisplay();
  delay(100);
  counter++;
  if (counter >= 16) counter = 0;
}

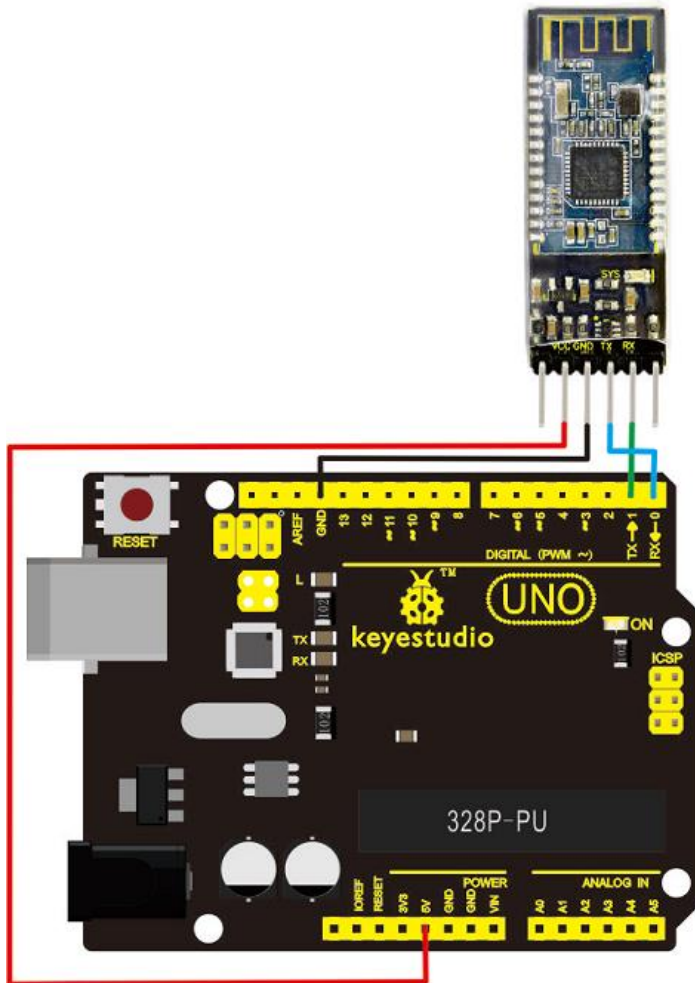
```

CHAPTER 8: WIRELESS COMMUNICATION

8.1 HM-10 Bluetooth Module



The HM-10 Bluetooth module is an Arduino compatible module, that allows users to add Bluetooth functionality to their Arduino projects. **NOTE:** you will need to download/use an external app that allows you to connect/send data to the module.



<i>HM-10</i>	<i>Arduino</i>
VCC	5V
GND	GND
TX	RX
RX	TX

HM-10 Code Example:

```
void setup(){
  Serial.begin(9600); //begin the serial monitor

  pinMode(13 ,OUTPUT); //set pin 13 as output
}
void loop(){

  int val = Serial.read(); //set the val variable to what was read in the serial
  monitor
  if(val=='a'){ //if the value read = a, run the code

    digitalWrite(13,HIGH); //turn on the on board LED

    delay(1000);

    digitalWrite(13,LOW); //turn off the on board LED

    delay(1000);

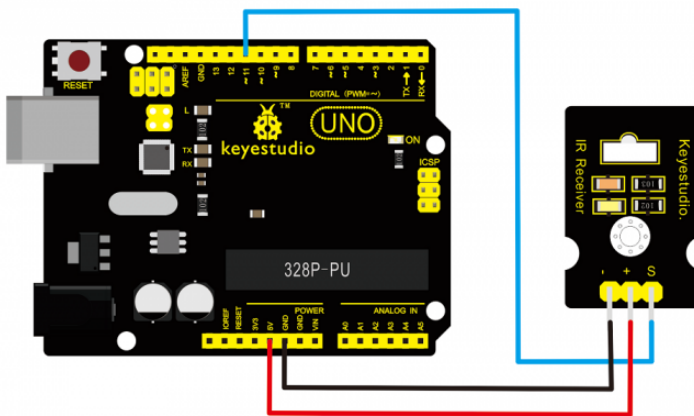
    Serial.println("Hello, World!"); //print "Hello, World!" to serial

  }
}
```

8.2 IR Remote and Receiver



The IR remote and receiver allows users to add infrared functionality to their Arduino projects. Infrared is a form of wireless communication that can be used to control/send signals to various electronic components. **Note:** the **IRremote.h** library will need to be downloaded and added to the Arduino IDE.



<i>IR Receiver</i>	<i>Arduino</i>
-	GND
+	5V
S	11

IR Remote and Receiver Code Example:

```
#include <IRremote.h>

int receiverPin = 11; //Input pin on Arduino Board
IRrecv irrecv(receiverPin); //Create a receiver object
decode_results results; //Create a decoded results object

void setup() {
  Serial.begin(9600); //Begin the Serial Monitor
  irrecv.enableIRIn(); // Start the receiver
}

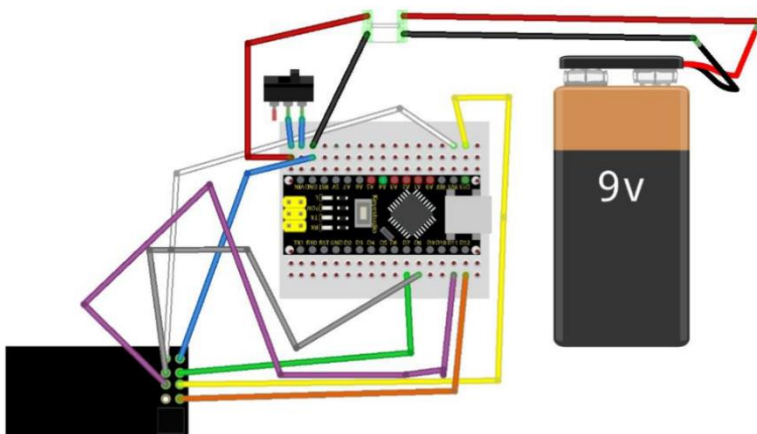
void loop() {
  if (irrecv.decode(&results)) { //Run if data was found through the receiver
    Serial.println(results.value); //Print the results value to serial
    irrecv.resume(); // Receive the next value found by the receiver
  }
}
```

8.3 RF Transceiver (NRF2L401):



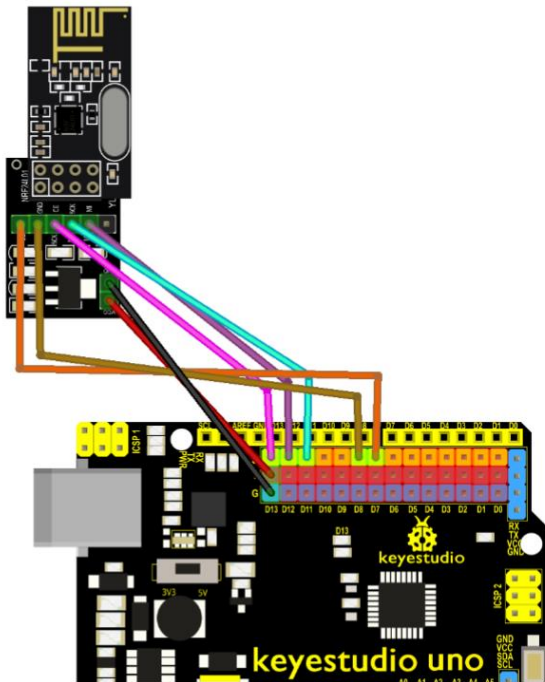
The NRF2L401 module is a wireless PC Data Transmission module, that allows users to implement bidirectional data transmission between two Arduino boards. One module will act as the transmitter and the other one will act as a receiver. When wiring, remember that the RF modules only take 3.3v, and if you have an adapter, it can take 5v. **Note** the **RF24.h** library will need to be added to the Arduino IDE.

Receiver Diagram:



<i>NRF2L401</i>	<i>Arduino</i>
GND	GND
3.3V	3.3V
CE	7
CSN	8
SCK	13
MOSI	11
MISO	12

Transmitter diagram:



<i>NRF24L01</i>	<i>Arduino</i>
GND	GND
5V	5V
CE	7
CSN	8
SCK	13
MOSI	11
MISO	12

Receiver Code Example:

```
//Sketch > Include Library > Add .ZIP Library
//Include SPI (Serial Peripheral Interface)
#include <SPI.h>
//Include Radio Frequency
#include <RF24.h>

//Include Servo
#include <Servo.h>

//Define DC motor ports --> these, you can calibrate at will if they are not going in the
right direction. If so, just switch the ports, and by doing so, it switches the
direction.
#define leftB A1 //left backward
#define leftF A2 //left forward
#define rightF 2 //right forward
```

```

#define rightB 4 //right backward

//Define DC PWM pins --> controls speed using pwm (digital pins with pwm capabilities)
#define ENL 5
#define ENR 6

//Define Button (could be digital pin but no digital pins remaining)
#define button A5

RF24 radio2(7, 8); //Define radio module. "radio2" is the name
byte addresses[][6] = {"0"}; //Set address [6]--> Must be same as transmitter

/* Define package data --> What will be received
   ML --> move left wheels with left joystick
   MR --> move right wheels with right joystic
   AM --> arm move with button
*/
struct package{
    int ML = 0;
    int MR = 0;
    int AM = 0;
};

//Define structure type
typedef struct package Package;
Package data;

//Define Servo
Servo servo;

void setup(){

```

```

//Set DC motor pin modes
pinMode(leftF, OUTPUT);
pinMode(leftB, OUTPUT);
pinMode(rightF, OUTPUT);
pinMode(rightB, OUTPUT);

pinMode(ENL, OUTPUT);
pinMode(ENR, OUTPUT);

//Set DC motor so it is stopped by default
analogWrite(ENL, 0);
analogWrite(ENR, 0);
digitalWrite(leftF, LOW);
digitalWrite(leftB, LOW);
digitalWrite(rightF, LOW);
digitalWrite(rightB, LOW);

//Set button input
pinMode(button, INPUT);

//Setup servo
servo.attach(3);
servo.write(0); //set default angle to 0

//Serial Monitor for debugging purposes
Serial.begin(115200);

radio2.begin(); //Start radio
//CHANGE VALUE BELOW FOR DIFFERENT ROBOTS, SAME VALUE BOTH RECEIVER AND TRANSMITTER

radio2.startListening(); //Start to listen for a package

```

```

Serial.println("Receiving packages"); //debug
delay(1000); //not necessary, but adds delay between setup and the loop
}

void loop(){
  //Only executes code if the radio is on
  if ( radio2.available()){
    while (radio2.available()){
      radio2.read( &data, sizeof(data) ); //Read the data that is send through the pipe.
    }

    //Prints received data in monitor
    Serial.println(data.ML);
    Serial.println(data.MR);
    Serial.println(data.AM);
    Serial.println();

    //Servo control
    servo.write(data.AM);

    //Controls for the left wheels
    if (data.ML >= 10){ //joystick pushed up
      digitalWrite(leftF, HIGH);
      digitalWrite(leftB, LOW);
      analogWrite(ENL, data.ML);
      Serial.println("Forward Left"); //debug (not necessary for final product)
    }
  }
}

```

```

else if (data.ML <= -10){ //joystick down
    digitalWrite(leftF, LOW);
    digitalWrite(leftB, HIGH);
    analogWrite(ENL, -data.ML);
    Serial.println("Backward Left"); //debug
}

else if (10 >= data.ML >= -10){ //deadzone
    analogWrite(ENL, 0);
}

//Controls for the right wheels
if (data.MR >= 10){ //right joystick up
    digitalWrite(rightF, HIGH);
    digitalWrite(rightB, LOW);
    analogWrite(ENR, data.MR);
    Serial.println("Forward Right"); //debug
}

else if (data.MR <= -10){ //joystick down
    digitalWrite(rightF, LOW);
    digitalWrite(rightB, HIGH);
    analogWrite(ENR, -data.MR);
    Serial.println("Backward Right"); //debug
}

else if (10 >= data.MR >= -10){ //deadzone
    analogWrite(ENR, 0);
}

```

```

if (digitalRead(button) == LOW){
  analogWrite(ENL, 0);
  analogWrite(ENR, 0);
  delay(2000);
  while (digitalRead(button) == HIGH){
    //While loop --> serial print for debug
    Serial.println("dead");
  }
  delay(500);
}
}
}

```

Transmitter Code Example:

```

#include <SPI.h>
#include <RF24.h>

RF24 radio1(7, 8); //Define radio module. "radio1" is the name
byte addresses[][6] = {"0"}; //Set address [0]--> Must be same as receiver

/*Define package data --> What will be sent
  ML --> move left
  MR --> move right
  AM --> arm move
*/

```



```

struct package{
    int ML = 0;
    int MR = 0;
    int AM = 0;
};

//Define structure type
typedef struct package Package;
Package data;

void setup(){
    //Serial Monitor for debugging purposes
    Serial.begin(115200);

    radio1.begin(); //Start radio

    //CHANGE VALUE BELOW FOR DIFFERENT ROBOTS, SAME VALUE BOTH RECEIVER AND
    TRANSMITTER

    radio1.setChannel(115); //Set Channel --> Must also be same as transmitter, and
    when having two different sets of modules, you must have these as different
    channels from 0-125

    //These settings are not as important as the others. It is recommended to have
    it the same as this for optimal use.

    //Reference manual can be found here:
    https://maniacbug.github.io/RF24/classRF24.html

    radio1.setPALevel(RF24_PA_MAX); //Set power amplifier (PA) level --> from
    levels 1-4, 4 being the max

    radio1.setDataRate(RF24_250KBPS); //Set data rate

    //These settings will be different from the bot, because this program will be
    SENDING the data.

```

```

    radio1.openWritingPipe(addresses[0]); //Open WRITING pipe on address [0] -->
    Must be same as transmitter

    radio1.stopListening(); //Doesn't need to listen, because it is the transmitter

    Serial.println("Sending packages"); //debug
    delay(1000); //not necessary, but adds delay between setup and the loop
}

void loop()
{
    //Write and send data
    radio1.write(&data, sizeof(data));

    //Print data in monitor
    Serial.println(data.ML);
    Serial.println(data.MR);
    Serial.println(data.AM);
    Serial.println();

    //Map joysticks between -255 to 255 --> the purpose of this is to determine
    whether the wheels go forward or backwards
    data.ML = map(analogRead(A1), 0, 1023, -255, 255);
    data.MR = map(analogRead(A2), 0, 1023, -255, 255);

    //Right joystick button --> reads as HIGH or LOW
    if (digitalRead(A3) == HIGH){
        data.AM = 125; //When button is pressed, send the angle of the servo to the
        bot
    }
}

```



```
else {  
    data.AM = 0; //else, set it to 0 degrees  
  
}  
delay(5); //also not necessary, but just makes serial monitor go a bit slower
```

CHAPTER 9: ADDITIONAL ARDUINO HARDWARE

9.1 Parts You Will Learn

Now that you have learned some of the basics about sensors, motors and shields, you can learn some of the additional hardware that Arduino has to offer that you can use to create projects with.

9.1.1 DS 1307 RTC Module Pinout

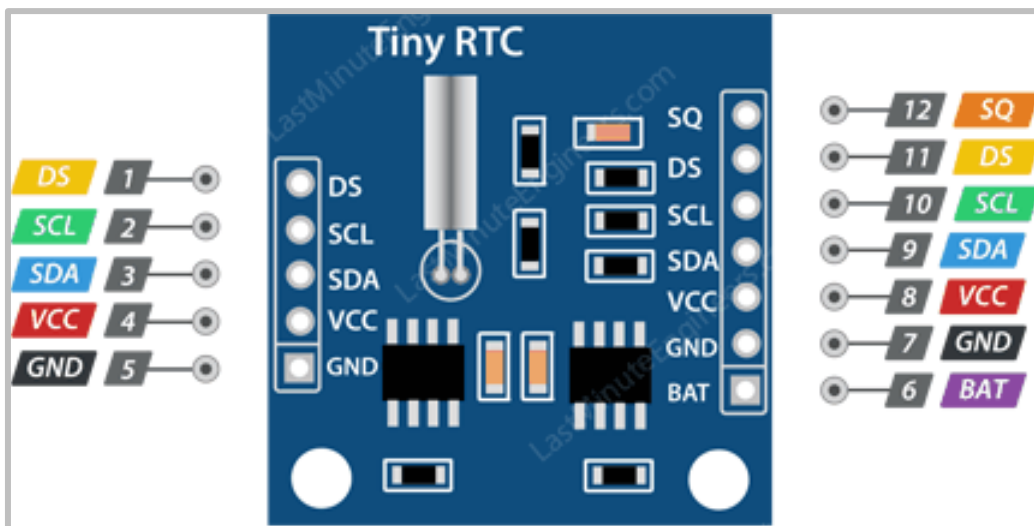
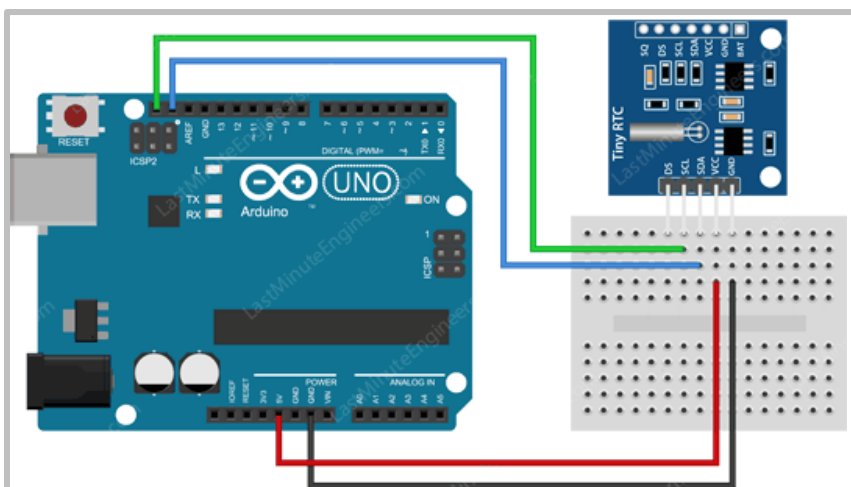


Figure 8-1. The RTC Module Pinout diagram.

RTC Module or Real Time clock, enables you to track time anywhere you go. The module is aware of the time around them. This module comes in handy when you come across the idea where you need to track time. It's perfect for projects containing data-logging, clock-building, time stamping, timers, alarms etc.



Pin Connections

[SCL] → [ICSP2]

[SDA] → [ICSP]

[VCC] → [5V]

[GND] → [GND]

RTC Module Sample Code:

```
#include <Wire.h>
#include <RTCLib.h>

RTC_DS1307 rtc;
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};

void setup ()
{
  Serial.begin(9600);
  delay(3000); // wait for console opening
  if (! rtc.begin())
  {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  if (!rtc.isrunning())
  {
    Serial.println("RTC lost power, lets set the time!");
    // Comment out below lines once you set the date & time.
    // Following line sets the RTC to the date & time this sketch was
    // compiled
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // Following line sets the RTC with an explicit date & time
    // for example, to set January 27 2017 at 12:56 you would call:
    // rtc.adjust(DateTime(2017, 1, 27, 12, 56, 0));
  }
}

void loop ()
{
  DateTime now = rtc.now();
  Serial.println("Current Date & Time: ");
  Serial.print(now.year(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print(" (");
  Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
  Serial.print(") ");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
```



```
Serial.print(now.minute(), DEC);
Serial.print(':');
Serial.print(now.second(), DEC);
Serial.println();
Serial.println("Unix Time: ");
Serial.print("elapsed ");
Serial.print(now.unixtime());
Serial.print(" seconds/");
Serial.print(now.unixtime() / 86400L);
Serial.println(" days since 1/1/1970");

// calculate a date which is 7 days & 30 seconds into the future
DateTime future (now + TimeSpan(7,0,0,30));
Serial.println("Future Date & Time (Now + 7days & 30s): ");
Serial.print(future.year(), DEC);
Serial.print('/');
Serial.print(future.month(), DEC);
Serial.print('/');
Serial.print(future.day(), DEC);
Serial.print(' ');
Serial.print(future.hour(), DEC);
Serial.print(':');
Serial.print(future.minute(), DEC);
Serial.print(':');
Serial.print(future.second(), DEC);
Serial.println();
Serial.println();
delay(1000);
}
```

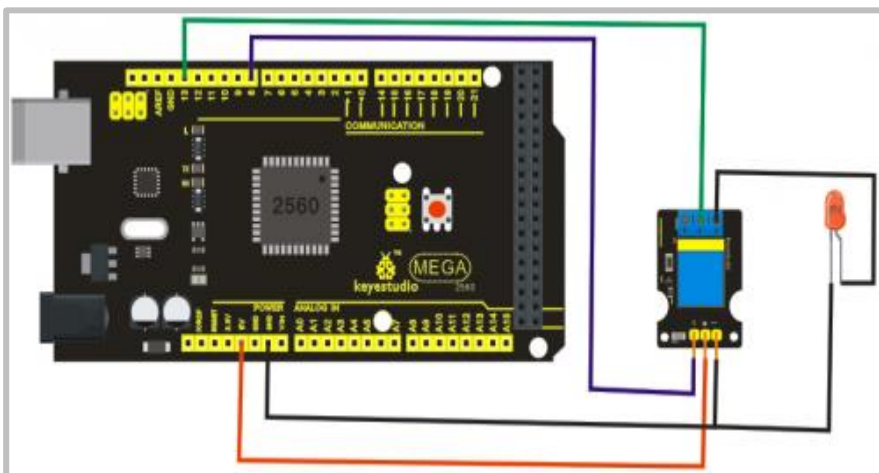
9.1.2 RELAY

DC Motor Specifications	
Type:	Digital
Rated current:	10A (NO) 5A (NC)
Maximum switching voltage:	150VAC 24VDC
Contact action time:	10ms
Size:	40*28mm
Weight:	15g



Figure 8-3. The Relay.

A **relay** is an electrically operated switch. Many relays use an electromagnet to manage a switch mechanically, but other operating principles are also applied, such as solid-state relays. Relays are used where it is necessary to control a circuit by a separate low-power signal, or where several circuits must be controlled by one signal.



Pin Connections Relay

[S] → [8]

[+] → [+5V]

[-] → [GND]

[+RE] → [+5V]

[-RE] → [+LED]

LED

[+LED] → [+RE]

[-LED] → [GND]

RELAY Example Code:

```
int relay_pin = 8;
int led_pin = 13;

void setup()
{
  pinMode(relay_pin, OUTPUT);
  pinMode(led_pin, OUTPUT);
  digitalWrite(led_pin, HIGH);
}

void loop()
{
  digitalWrite(relay_pin, HIGH);
  delay(5000);
  digitalWrite(relay_pin, LOW);
  delay(5000);
}
```

9.2 Practice Examples

1) Using a Meter to control an LED

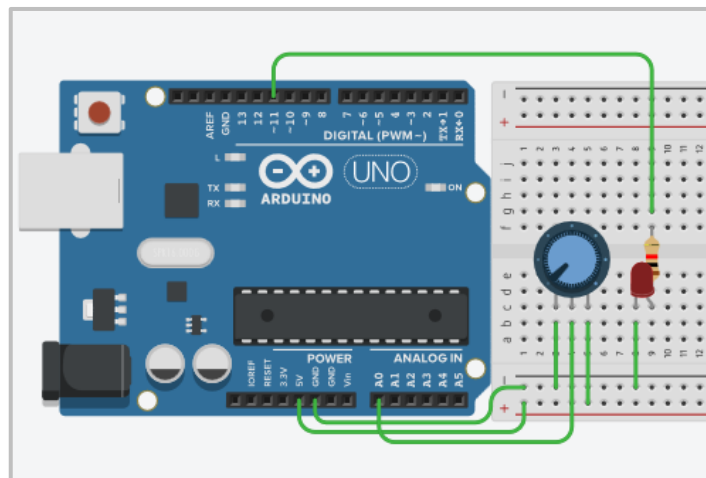


Figure 8-5. A circuit with an LED and meter attached to it.

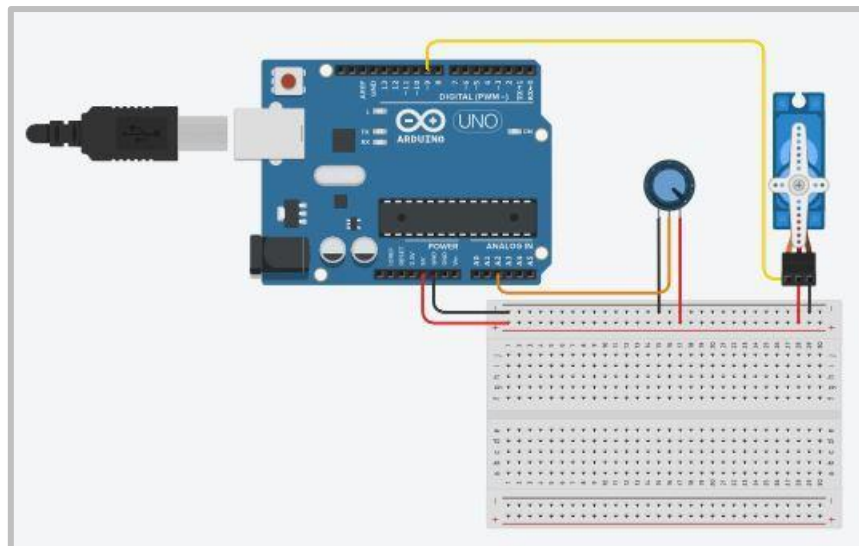
Sample Code:

```
int Meter = 0;
int LED = 11;

void setup()
{
  pinMode(Meter, INPUT);
  pinMode(LED, OUTPUT);
}

void loop()
{
  int brightness = analogRead(Meter);
  analogWrite(LED, brightness);
}
```

2) Moving a Servo Motor using only a Potentiometer



Sample Code:

```
#include <Servo.h>

Servo myservo;
```

```
int pot;
int angle;

void setup()
{
  myservo.attach(9);
  myservo.write(0);
}

void loop()
{
  pot = analogRead(A2);
  angle = map(pot, 0, 1023, 0, 180);
  myservo.write(angle);
}
```

Figure 8-6. A circuit with an LED and meter attached to it.

9.3 Homework Questions

Q. 1) Using a dial to position a motor.

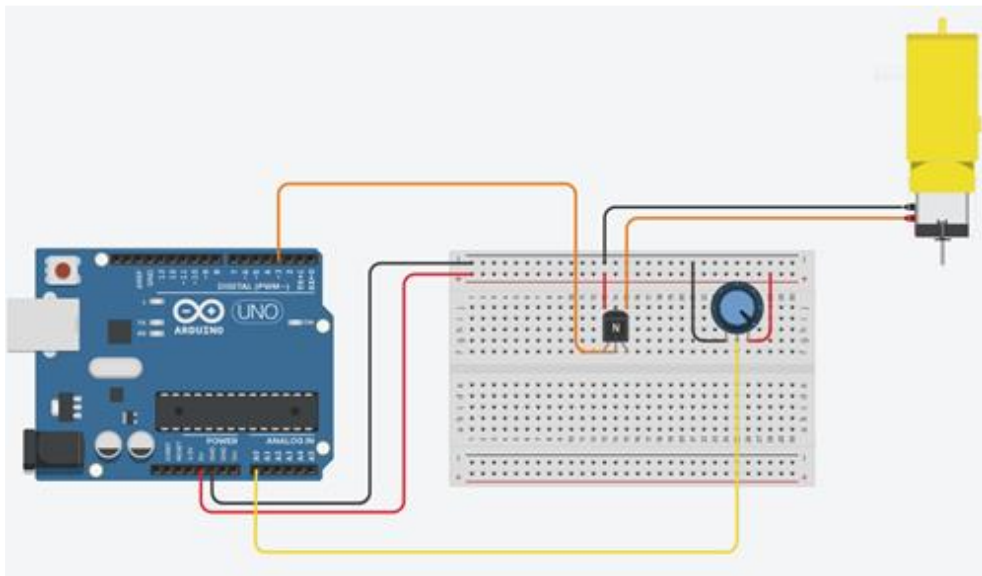


Figure 8-7. A circuit with a dial to control a motor.

- a. Build the following circuit above.
- b. Write a program to map the dial position to the speed of the motor
Remember: analog input range is: 0 - 1023

Analog output range is: 0 - 255

Simple map formula (assumes both ranges start at 0)

$Out = (In / Max_In) * Max_Out + Min_Out$



Q2. Create a digital safe! [CHALLENGE]

The goal will be to have three potentiometers to control the combination. When the safe is locked, display its status and the values of the three dials. Use a push button to confirm the combination entered and checked if it's correct. The suggested range for the combination is 60, like a real lock. However, you can decide how your lock works.

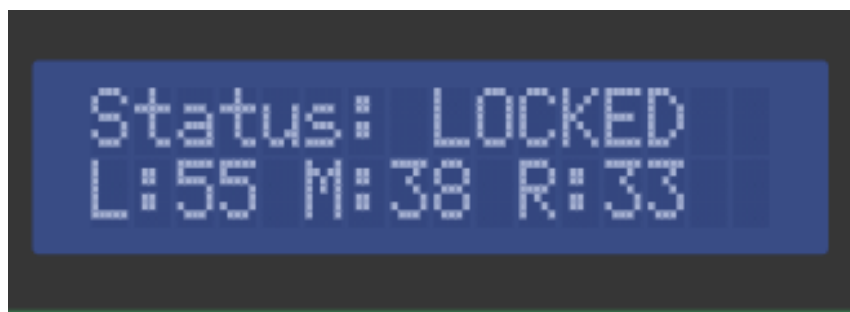


Figure 8-8. An LCD displays.

The project should include:

- Three potentiometers
- 1 LCD
- One push button
- LEDs (optional)

Extension: As a challenge, make this work with one potentiometer instead of 3. It should still require three numbers for the combination.

CHAPTER 10: HELPFUL BEGINNER TIPS

10.1 Common Issues and Troubleshooting

Now that you have learned a lot about hardware, software, shields and additional pieces Arduino has to offer, you are bound to meet some issues and challenges while building these complex circuits. This chapter is for you! Here, you can find some common issues. And solutions to those problems.

10.1.1 Problems in Circuits

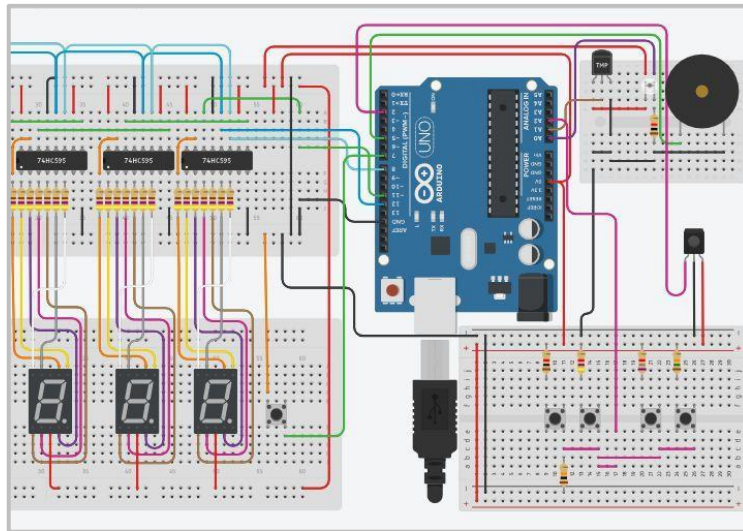


Figure 9-1. A complicated circuit

Tips to prevent problems in circuits:

Make sure that you have properly connected all the hardware parts in the circuit diagram
Connection is key, some wires may come loose over a certain period. Therefore, make sure that all the wires are in place and they are properly secured.

Some hardware parts may break down for some unexpected reason. Therefore, make sure that your parts are not damaged, as it can affect your whole project

Make sure that you are using the correct pins. With Arduino UNO having so many pins, it is easy to connect the wires in the incorrect pins.

Make sure that you are using the correct Arduino parts



Helpful
Tips
to know!

10.2 Problems in Programming

10.2.1 Variable

Tips to prevent errors in Programming:

Make sure that the name of the variable you are using is the same all the time. Sometimes you can misspell the variables and that creates an error.

Make sure that you don't repeat the same name for the variables. If you use the same name for different variables, then your code will be confused as to which value it needs to get.

Make sure that you name your variables with proper names so that they match the data inside of the variable. For example; if you have an integer, you may want to make a variable name `my_int` and assign an integer to this variable.



Helpful
Tips
to know!

```
int x = 27;
```

10.1.3 Equals, Assignments and Syntax:

When programming with Arduino, it is a very common mistake to confuse the assignment operator to the comparison operator. For example:

```
if (myInt = yourInt)
{
    //Statements
}.
```


Do you see the mistake here? Instead of using '==', the code is using '='. Therefore, the code is not comparing the variables, but it is assigning the variable yourInt to myInt. This is a very common mistake and the simple fix is to ensure you are using adding two '==' to the if statement.

Float and Integer Math. In Arduino, when doing floating in math, it is very important to know what kind of syntax to use. For example:

```
int num = 2;  
float num2 = 5 / num;  
Serial.println(num2);
```

This example will print out the number 2 instead of 2.5. The compiler is only looking at integers and it is looking at the integer math. Then it assigns the value of float to the num2. An easy way to fix this problem is to make the number a float beforehand. Just by adding .0 to the number. This makes the number a float and it does not change the value of the number. For example:

```
Int num = 2;  
Float num2 = 5.0 / num;  
Serial.println(num2)
```

This example will print out the number 2.5. The compiler now notices that the number is a float/decimal, therefore the answer will be in decimal as well.

Note: Another common mistake is not putting ';' at the end of each line. You must put ';' at the end of each line of code. If you don't, it will give out an error.



Figure 9-3. The semicolon

If, Else, Else if, Functions

You must remember to properly spell the If, Else, Else if or Function. If You misspell the functions, then they will not execute, meaning that you will not get a desired outcome.

If, Else, Else if

Figure 9-4. If else spelling.

Libraries

If you need to include a library in your code, then you need to go to the Sketch Folder in Arduino, go to include library and then include the library that you need.

If you don't see the library that you need, then you can download a library from Arduino. You can go to the Sketch Folder in Arduino, go to Include Library and then scroll up to see Manage Libraries. When you click on that, you can search up the library that you need and install it.

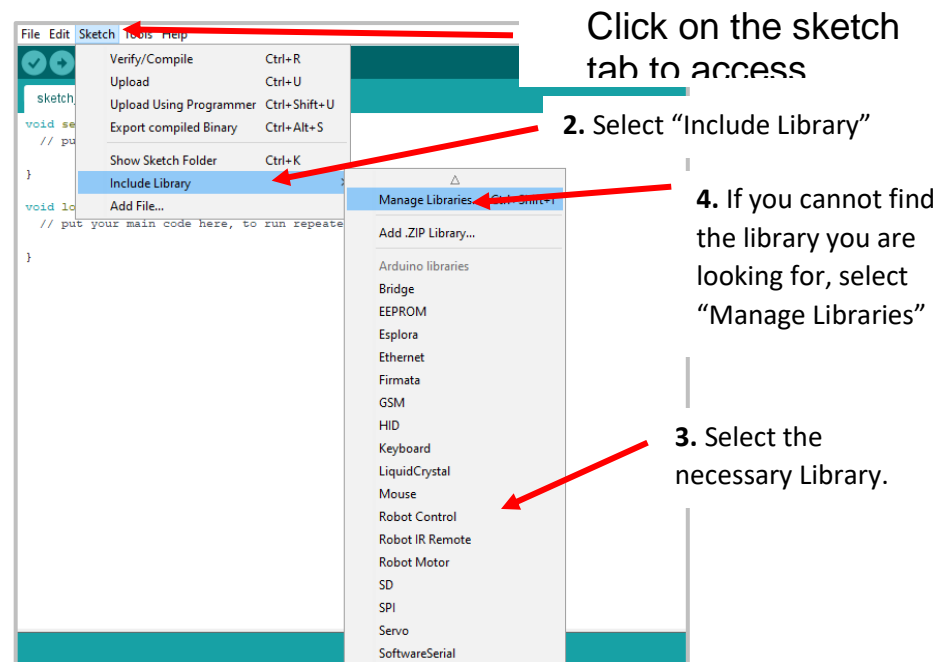
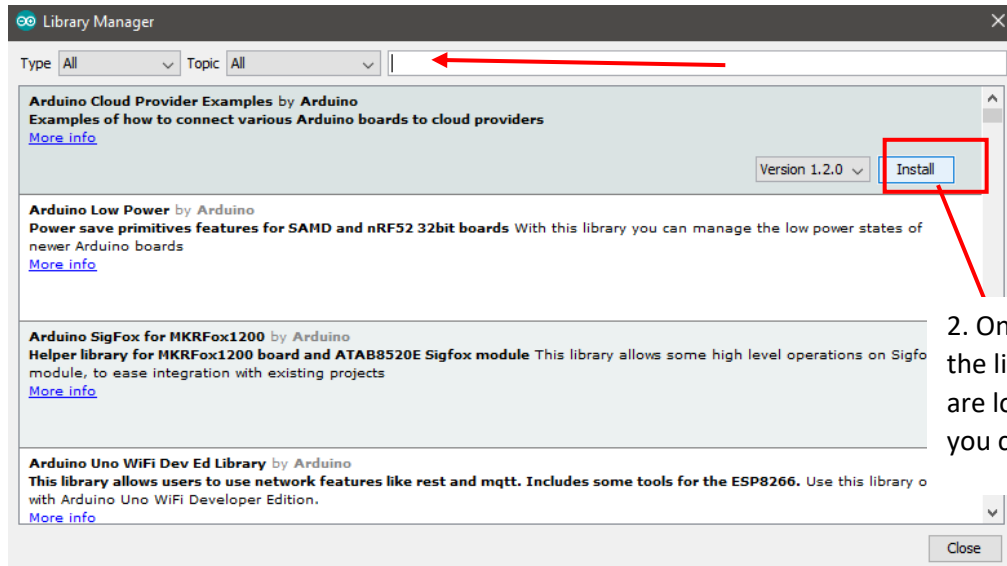


Figure 9-5. The Arduino IDE and how to access Arduino Libraries

Search for the specific library you need after selecting “Manage Libraries”



2. Once you find the library you are looking for, you can Install it

Figure 9-6. The Arduino Search Libraries Menu

10.3 Common Errors

DIY Robotics Lab provides an excellent summary of the common errors you might encounter:

```
/*- Blink an LED -//
```

```
Error: Uncaught exception type:class java.lang.RuntimeException
java.lang.RuntimeException: Missing the */ from the end of a /*
comment */
```

The error on line 1 is caused by mixing the comment styles. The comment begins with the “/*” characters but ends with “//” instead of the “*/” characters. The correct line is as follows:

```
int ledPin = 23; \\ We're using Digital Pin 23 on the Arduino.  
error: stray '\ ' in program
```

This is another problem with incorrect commenting technique. In this line the “\\” characters are used to begin a comment instead of the “//” characters. The correct line follows:

```
int ledPin = 3; // We're using Digital Pin 3 on the Arduino.
```

```
void setup();  
error: expected unqualified-id before '{' token
```

This is an easy mistake to make. The problem is the semicolon “;” at the end of a function declaration. The article “[Learning the C Language with Arduino](#)” contains a section about correct usage of semicolons. To correct this problem, remove the semicolon as shown below:

```
void setup()
```

```
error: too few arguments to function “void pinMode(uint8_t uint8_t)”  
At global scope: In function ‘void setup()’:  
error: expected `)’ before numeric constant/home/myDirectory/Desktop/myPr  
ograms/arduino-0015/hardware/cores/arduino/wiring.h:102:
```

To correct the error shown above, the corrections are shown below:

```
pinMode(ledPin OUTPUT); //Set up Arduino pin for output only.
```

The clue to this problem is found in the message “*error: too few arguments to function ‘void pinMode(uint8_t, uint8_t)’*”. The message includes a list of the function’s arguments and data types (uint8_t). The error is complaining that we have too few arguments. The problem with this line of code is the missing comma between ledPin, and OUTPUT. The corrected code is on the following line:

```
pinMode(ledPin, OUTPUT); //Set up Arduino pin for output only.
```

```
loop()
```

error: expected constructor, destructor, or type conversion before '(' token

In this line the type specifier for the function is missing.

To fix this problem place the data type for the function's return type. In this case we're not returning any value, so we need to add the keyword **void** in front of the **loop** function name. Make the following change:

```
void loop()
```

```
void loop () (
```

error: function 'void loop()' is initialized like a variable

The block of code that makes up the loop function should be contained within curly braces "{" and "}". In this line a left parenthesis character "(" is used instead of the beginning curly brace "{". Replace the left parenthesis with the left curly brace as shown below:

```
void loop ()
{
    //Statements
}
```

```
//The HIGH and LOW values set voltage to 5 volts when HIGH and 0 volts LOW.
error: expected primary-expression before '/' token At global scope:
```

This line is supposed to be a comment describing what the program is doing. The error is caused by having only one slash character “/” instead of two “//”. Add the extra slash character as shown below:

```
//The HIGH and LOW values set voltage to 5 volts when HIGH and 0 volts LOW.
```

```
digitalWrite(ledPin, high); //Setting a digital pin HIGH turns on the LED.  
error: ‘high’ was not declared in this scope at global scope:
```

This error message is complaining that the variable “high” was not declared. Programming in C is case sensitive, meaning that it makes a difference if you are using upper or lower case letters. To solve this program replace “high” with the constant value “HIGH” then recompile.

```
digitalWrite(ledPin, HIGH); //Setting a digital pin HIGH turns on the LED
```

```
delay(1000): //Get the microcontroller to wait for one second.  
error: expected `;’ before ‘:’ token At global scope:
```

```
delay(1000); //Get the microcontroller to wait for one second.
```

```
digitalWrite(ledPin. LOW); //Setting the pin to LOW turns the LED off.  
error: expected unqualified-id before numeric constant At global scope
```

This error can be particularly troublesome because the comma “,” after the variable ledPin is actually a period “.” making it harder to spot the problem.

The C programming language allows you to build user defined types. The dot operator (period “.”) is part of the syntax used to reference the user type’s value. Since the variable ledPin is defined as an integer variable, the error message is complaining about the unqualified-id.

```
digitalWrite(LedPin, LOW); //Setting the pin to LOW turns the LED off.
```

```
Delay(1000); //Wait another second with the LED turned off.  
error: 'Delay' was not declared in this scope At global scope:
```

In function 'void loop()':

This error was caused by the delay function name being spelled with an incorrect upper-case character for the letter "d". Correct the spelling using the lower case "d" as shown below:

```
delay(1000); //Wait another second with the LED turned off.
```

```
Void Loop()  
{  
  //Statements  
  
}  
  
}
```

```
error: expected declaration before '}' token
```

There is an extra curly brace at the end of this program. Delete the extra brace to correct this error.

The compiler completed without any more error messages so why doesn't the LED flash after loading the program on my Arduino? No error message was given by the compiler for this problem:

```
int ledPin = 23; \\We're using Digital Pin 23 on the Arduino.
```

```
int ledPin = 3; \\We're using Digital Pin 3 on the Arduino.
```

10.4 Good Programming Techniques

As a programmer, it is very important to learn good programming techniques that will help you in your future. Before you start making a project or your code. You first need to make a **PPD or Project Planning Document**. A PPD allows you to see everything that you will have in your project. PPD includes: **the description of the project** that you are doing, a **flowchart**, a **PERT** chart, and everything that you think is valuable or worth sharing in PPD. Let's first define all those terms and explain how to work with them.

Description of a project or a synopsis of a project is a brief discussion about your project. What will it do? What does it do? What components does this project require? These are all the questions that you might want to consider asking yourself when making a description of a project. You can add more information you need for the description of the project that you are doing.

Flowchart is self-explanatory. It is a chart that shows the flow of the project. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. It shows the logic of a code. You don't have to know how to make a proper flowchart using different rectangular shapes and arrows. You just need to visualize the flow of the code and how everything will be connected so that it makes a logical sense.

PERT chart or **Program Evaluation Review Technique** is a project management tool used to schedule, organize, and coordinate tasks within a project. It helps the user, in this case yourself, to organize and prioritize certain tasks and programs. In order to make a PERT chart, you need to create circles with different tasks. Then you can connect each circle with an arrow showing that from this task, your next task is going to be a new circle.

For example:

- A. Description
- B. Flowchart
- C. PERT chart
- D. Engineering Design Process
- E. Buying the supplies
- F. Creating a prototype
- G. Testing

- H. Making an updated version of the product
- I. Showcasing it to other people.

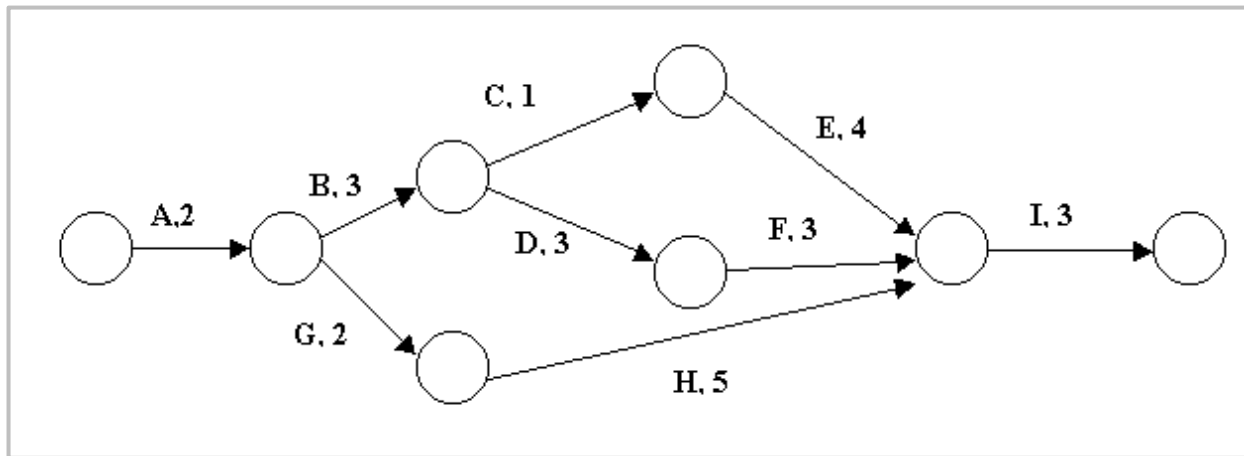


Figure 9-7. An example of a PERT Chart

The chart above shows you a distinct set of tasks that you need to finish over a certain period. The tasks all depend on how much you are working and how many tasks do you need to finish over what time. If you need to have more tasks, then you will add more circles to the chart and add numbers representing either the number of days or the number of weeks, it all depends on you. If you need less tasks, then you will subtract circles from the chart. In the end, PERT chart allows you to see on what tasks you will be working and how long each task will take in terms of time.

CHAPTER 11: ARDUINO RESEARCH SKILLS

11.1 Engineering Design Process

Before you begin designing everything that you do in Arduino, you must know how the Engineering Design Process works and how to make it while designing your own projects.

You may be asking, what is an Engineering Design Process? The **Engineering Design Process** is a series of steps that engineers use in creating functional products and processes. It is a guide that helps you convert resources optimally to meet a stated objective. An Engineering Design Process consists of **7 steps** as you can see in the flowchart below.

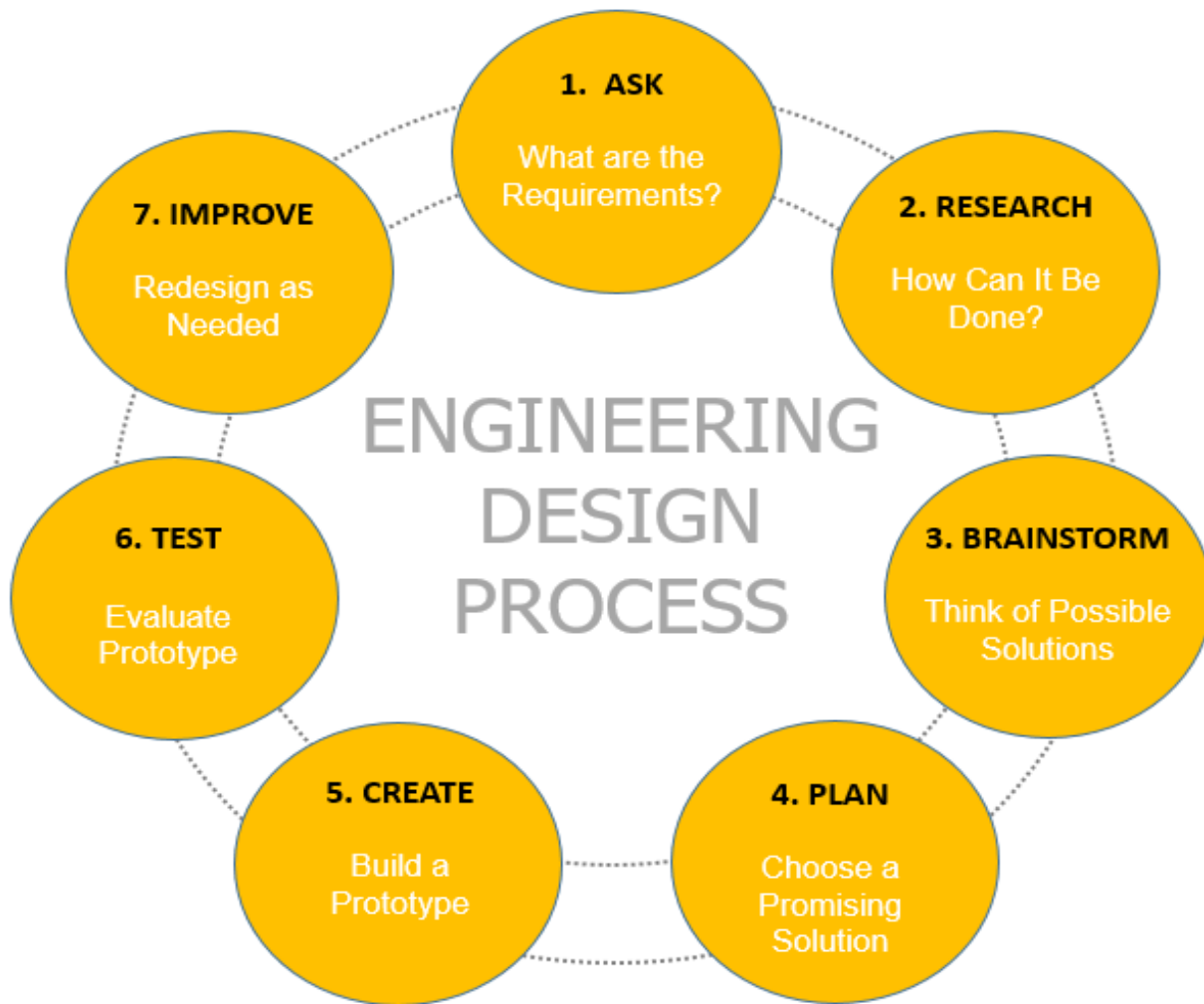


Figure 10-1. A flowchart of the Engineering Design Process

11.1.1 WHAT ARE THE REQUIREMENTS?

Before you start building and engineering products/processes, you first need to know what the **requirements** are for developing a project. The conditions may include cost, timing, manufacturing, assembly, features etc. If you look at the table below, this is an example of what your table of requirements should look like. You can have multiple identical categories if the conditions are not the same.

No.	Category	Requirement
1	function	To wirelessly activate a finger based on the position of your finger
2	feature	Detect the positioning of a person's finger
3	feature	Make a bionic finger mimic a person's finger
4	feature	wireless
5	cost	Up to \$25
6	timing	The prototype was done in 3 weeks (week of November 14)
7	manufacture	No extra manufacturing needed
8	assembly	No glue or soldering, ease of assembly for 9-year old
9	document	Assembly and programming slides

11.1.2 HOW CAN IT BE DONE?

Research the problem and see what the project requires from you. It would be best if you came up with questions to research on. Then, you can identify and solve the problem more in-depth and specific steps. Maybe you use a sensor to complete this project; then you must research on how to use this sensor. The table below, this is an example of what your table should look like.

No.	Category	Research Details
1	function	How do I use the flex sensor's data to accurately and precisely move the finger?
2	feature	What will I use for detecting the finger's position? How will I make a flex sensor?
3	feature	What will I put the sensor on? How will it fit with a hand comfortably?
4	feature	How do I move the finger and have it always in the right position? If I need to, how do I recalibrate it? What can I do to reduce drift as much as possible?
5	feature	What will I use to make the sensors around my hand compact and size efficient?
6	feature	What are technologies for wireless transmission feasible?
7	cost	What is the opportunity cost of using a flex sensor that we pay for versus making a flex sensor? The opportunity cost of different ways of making it wireless?

11.1.3 RESEARCH ON POSSIBLE SOLUTIONS

From the questions on the Research Table, you must research the items that you asked. Moreover, record the information gathered on a new table with citations from where you found the useful information. If you did not see the answers to your questions, then it is okay. You can try to find ways to solve the problem by thinking outside the box or implementing new ideas to solve this problem. Make sure to fill the whole table; never leave a cell without any research. If you look at the chart below, this is an example of what your Solutions Table should look like:

.

No	Category	Possible Solutions
1	function	<p>Flex sensor to servo finger positioning can be done by experimenting to find the highs and lows of the flex sensors values from straight to most bent wanted, as well as the range for the servo for the finger, and use map</p> <p>Link: https://codebender.cc/sketch:55013#GloveTX_ENG_v1.ino</p>
2	feature	<p>Flex sensor, can either buy or make own using paper, construction paper, aluminum foil</p> <p>Link: https://www.instructables.com/id/Arduino-Make-a-Flex-Sensor-for-Robotic-Hand-Cheap/</p>
3	feature	<p>To make a casing to put the flex sensor on that my hand will go into, I can design a series of joints and loops to fit my fingers through, ensuring that bends at joints are detected</p>
4	feature	<p>For wireless transmission:</p> <p>https://www.instructables.com/id/wireless-pan-and-tilt-camera-rig-with-arduino/</p> <p>https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/</p> <p>Using Nano or lily pad 328, uncertain of the power supply, investigate power supply options or type of “Arduino.”</p> <p>Nano – 6-20V, lily pad – 1.7-4.5V</p>
5	feature	<p>To make the sensors around my hand compact and efficient, I will spread the sensors around my hand as flat as possible, so it would not interfere with my hand movement. I will make sure the sensors are precisely measured so that sensors would not take extra space.</p>
6	feature	<p>Radio Frequency Transmission, Infrared transmission, WIFI transmission</p>
7	cost	<p>Flex sensor is \$12.50 vs making the flex sensor which does not cost efficient. Rather than making the flex sensor, which takes more time, I can buy it.</p>

11.1.4 CHOOSING A PROMISING SOLUTION

By selecting the most promising solution, you are increasing the success of the project significantly as out of all the possible solutions; you choose the one that looked the most promising. It would help if you decided which of the possible solutions that have the best chance of meeting the project requirements.

11.1.5 BUILD A PROTOTYPE

This is the most exciting part of the Engineering Design Process. As after all the research that you have put in for completing this project, you can finally build a prototype. If you have more questions and problems, then you can revert to your Research Table and do more research on how to solve specific problems that you encountered.

11.1.6 EVALUATE THE PROTOTYPE

At this step, you must evaluate the prototype. Does the prototype meet all my requirements in step #1? After you are done evaluating your prototype, then you must input the information of the prototype on a new table. Below is an example of how Evaluation Table looks:

No.	Category	Project Requirement	Meets Requirement?
1	function	To wirelessly activate a finger based on the position of your finger	Yes
2	feature	Detect the positioning of a person's finger	Yes
3	feature	Make a bionic finger mimic a person's finger	Yes
4	feature	wireless	Yes
5	cost	Up to \$25	No
6	timing	The prototype was done in 3 weeks (week of November 14)	No
7	manufacture	No extra manufacturing needed	Yes
8	assembly	No glue or soldering, ease of assembly for nine-year-olds	Yes
9	document	Assembly and programming slides	Yes

11.1.7 REDESIGN AS NEEDED

At this step, you can redesign the project/prototype as needed. If the prototype did not meet the Evaluation Requirements, then you need to redesign the project as needed so the final product will meet all the requirements needed. If your prototype did not function completely, then you must redesign, rebuild and retest until your product works.

11.2 How Do You Start?

After learning the Engineering Design Process, you must learn how to identify, break down and research a problem efficiently and effectively. You can see the cycle of researching a problem from the chart below:

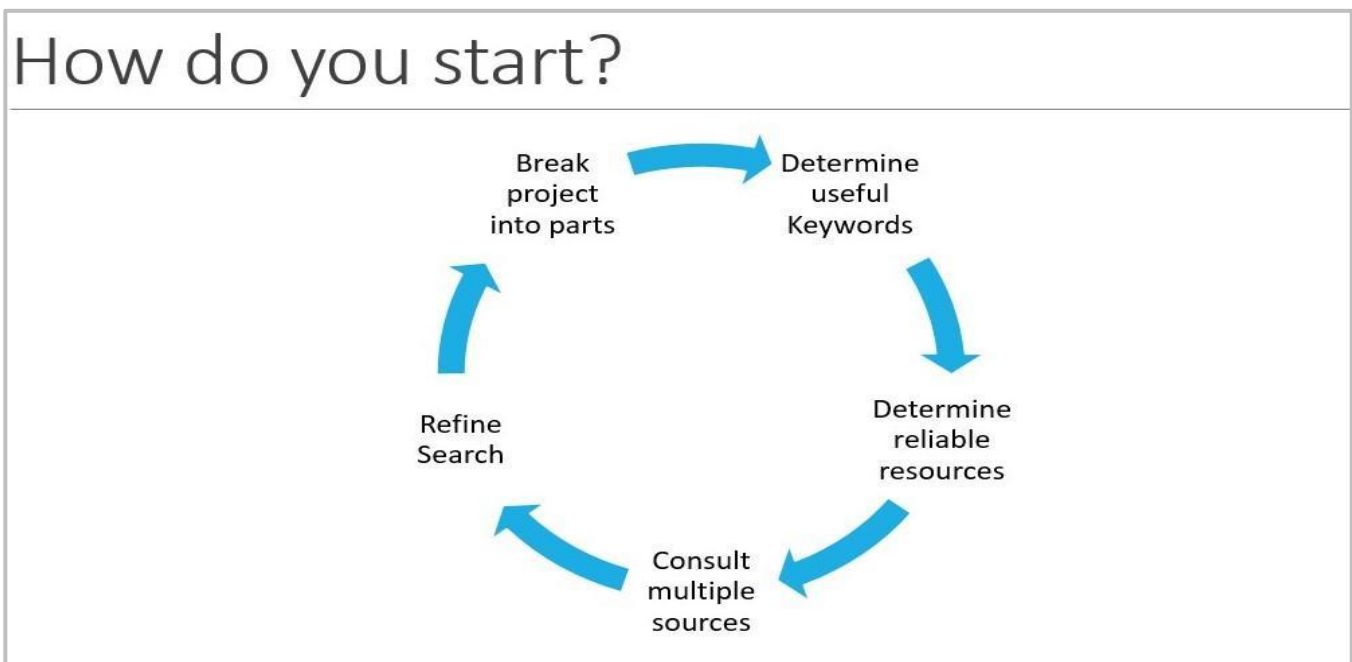


Figure 10-2. A chart of steps taken to break down a complex problem

Breaking Down the Problems

The first step, you need to break down the problem into many different parts. This process will help you to refine your research and help you to solve the problem more effectively.

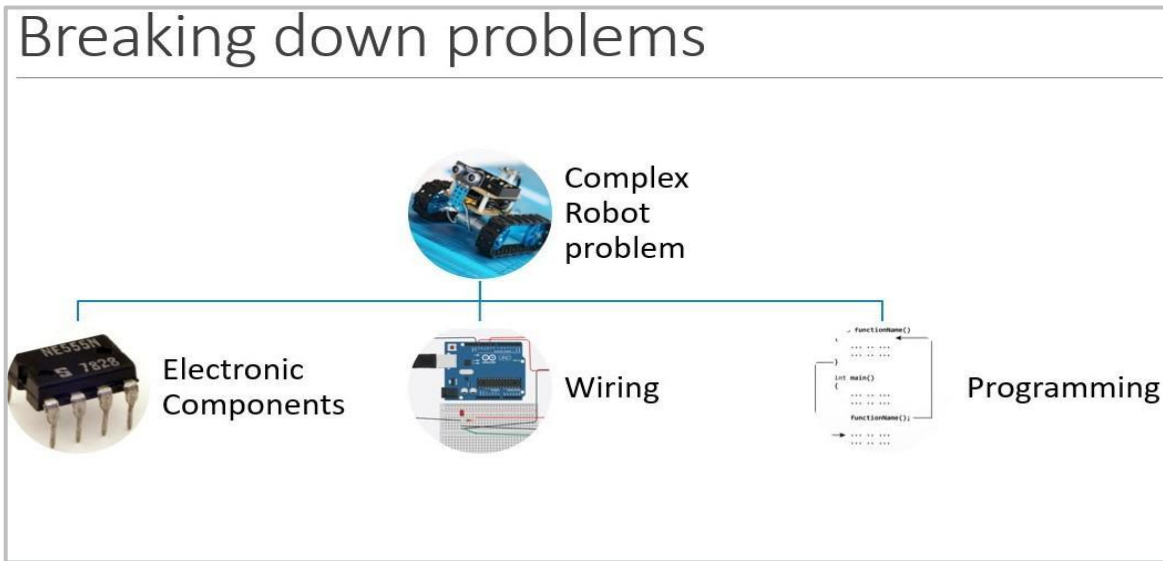


Figure 10-3. Breaking down a problem involves taking a complex problem and breaking it down into manageable, unique parts.

Determine Useful Keywords

The second step, you need to determine useful keywords to get the best research. The valuable keywords would be the Arduino parts that you are going to use or questions about the project that you do not understand. For example, let's say you are going to design a 2D platformer game with an LCD and an Arduino. Then you must search "Arduino LCD" to get the best research and answer this question.



Figure 10-4. Using specific keywords to get the best search results possible to research a project you want to build.

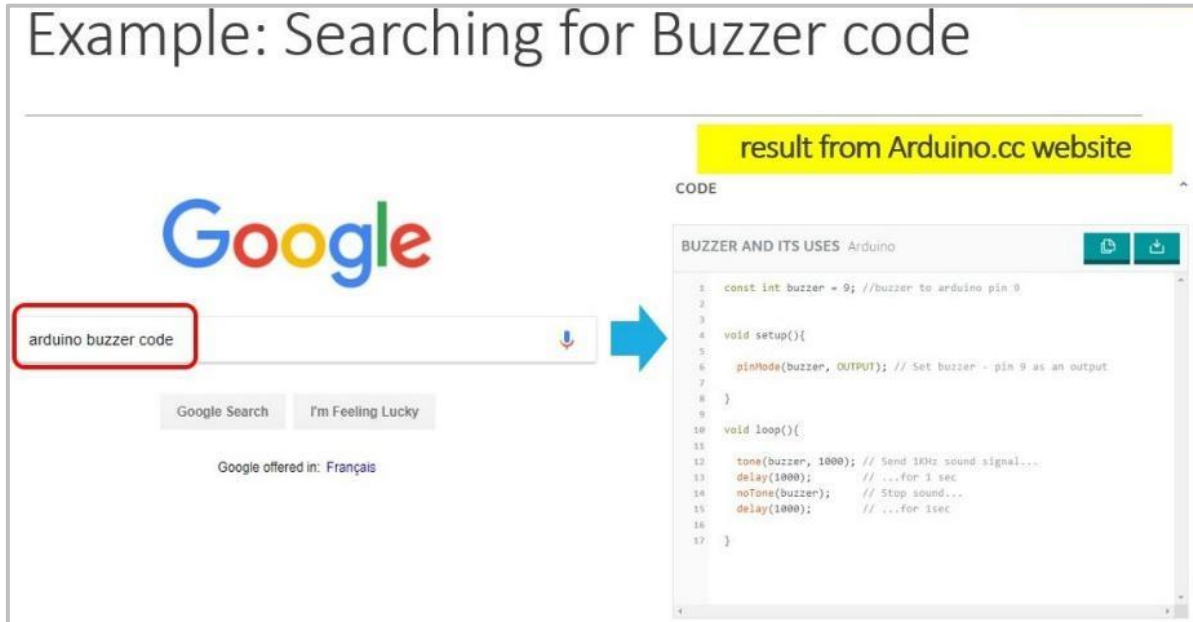


Figure 10-5. Using specific keywords to search for a specific result to research and learn.

Determine Reliable Resources/Consult Multiple Sources

The third step, you need reliable resources. There are many different types of reliable resources. However, this handbook will provide you with the four reliable resources for many separate occasions. The image below shows some reliable resources you can use and the type of information they provide:

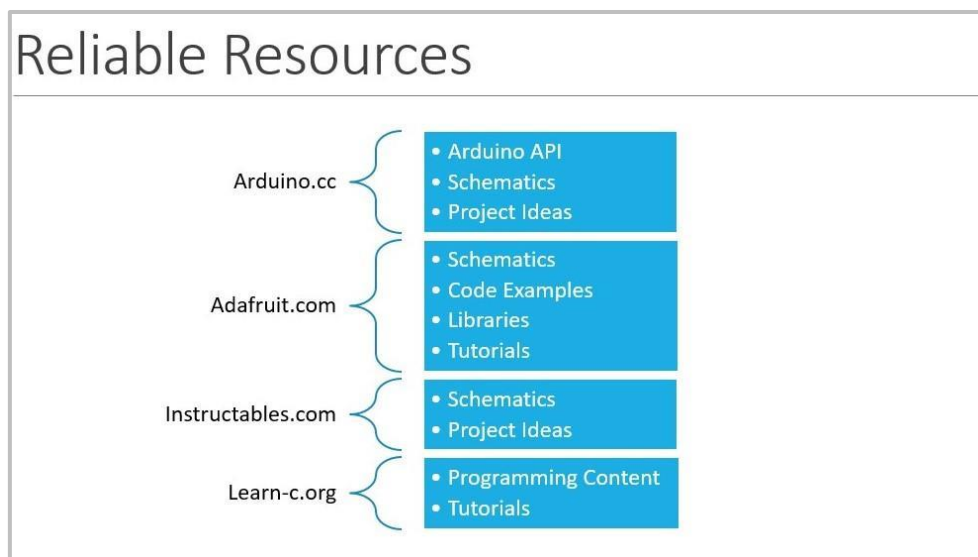


Figure 10-6. A list of useful resources and the useful information you can find with them while researching for Arduino based projects.

11.3 Types of Useful Resources

Four different types of resources will help you with circuits, mechanics or programming. These resources are **API's**, **Schematics**, **Data Sheets** and **Code Samples**.

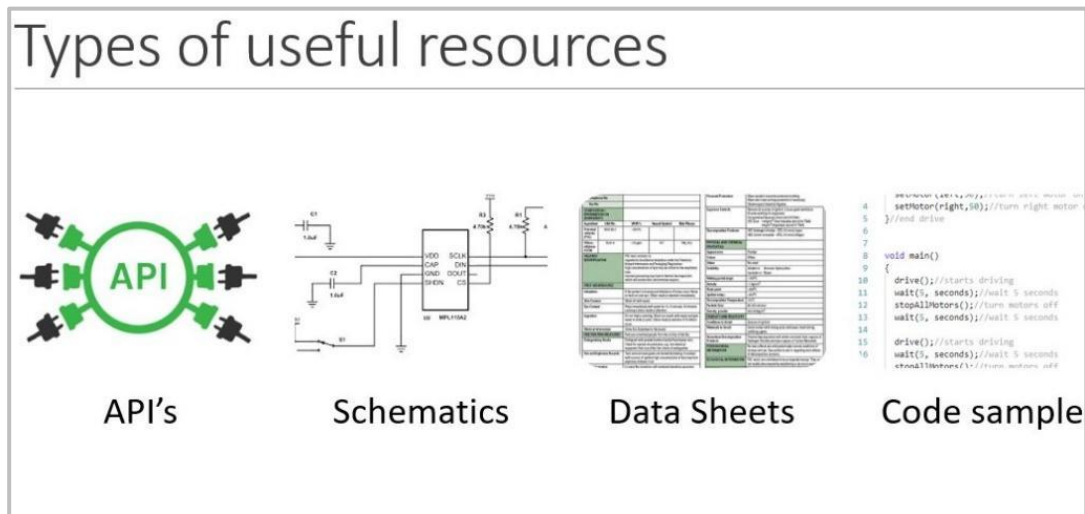


Figure 10-7. A list of useful resources that you can come across while designing circuits, projects, code etc. These include **API's**, **Schematics**, **Data Sheets** and **Code Samples**.

11.3.1 API's

API is an acronym that stands for “**A**pplication **P**rogramming **I**nterface.” An API lists operations and things that developers can use in the project they created, along with a description of what they do. It aims to cover everything a person would need to know for practical applications. API documentation is found in documentation files, but they can also be found in blogs, forums, and Q&A websites.

In the interest of clarity, API documentation may include a description of classes and methods in the API as well as "typical usage scenarios, code snippets, etc.".

How to read an API

Description
Explanation of the functions purpose

Arguments
Data (variables) required to complete the task

Return Type
The data type of data returned by the function

Description
Reads the value from a specified digital pin, either HIGH or LOW.

Syntax
`digitalRead(pin)`

Parameters
`pin`: the number of the digital pin you want to read

Returns
HIGH or LOW

Example Code

```

Sets pin 13 to the same value as pin 7, declared as an input.
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7; // pushbutton connected to digital pin 7
int val = 0; // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT); // sets the digital pin 13 as output
  pinMode(inPin, INPUT); // sets the digital pin 7 as input
}

void loop()
{
  val = digitalRead(inPin); // read the input pin
  digitalWrite(ledPin, val); // sets the LED to the button's value
}

```

Figure 10-8. This image shows you how to read an API. There are descriptions, as well as arguments and return types for an API.

11.3.2 Datasheets

A **datasheet** or **spec sheet** is a document that summarizes the performance or technical specifications/characteristics of a product in sufficient detail that allows a person to understand the physical and software aspects of the product. The datasheet includes information that can help in making a purchasing decision about a product by providing technical specifications about the product.

The image below shows you how to read Datasheets. A Datasheet contains PINOUTS where you can acquire a PIN and the purpose for each PIN. Also, there are Electrical Ratings on the Datasheets, which shows you the minimum and maximum voltages and current requirements for these voltages.

Reading Datasheets

PINOUT
Pin number and purpose for each pin

Electrical Rating
Min/ Max voltage and current requirements

INTERFACE PIN FUNCTIONS

Pin No.	Symbol Level	Description
1	VSS	Ground.
2	VDD	Power supply for logic operating.
3	V0	Adjusting supply voltage for LCD driving.
4	RS	Signal for selecting registers.
5	RW	Data Register (for read and write) Instruction Register (for writes, Busy flag, Address Counter (for read), RW = "H": Read mode, RW = "L": Write mode.
6	E	An enable signal for writing or reading data.
7	DB0	This is an 8-bit bi-directional data bus.
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	
15	VLED	Power supply for backlight.
16	LED	The backlight ground.

ABSOLUTE MAXIMUM RATINGS (Ta = 25°C)

Parameter	Symbol	Min	Max	Unit
Supply voltage for logic	Vcc	-0.3	+7.0	V
Supply voltage for LCD	V ₀	0	V _{cc} +0.3	V
Input voltage	V _i	-0.3	V _{cc} +0.3	V
Normal Operating temperature	T _{op}	-20	+70	°C
Normal Storage temperature	T _{stg}	-30	+80	°C

Notes: Stresses beyond those given in the Absolute Maximum Rating table may cause operational errors or damage to the device. For normal operational conditions see AC/DC Electrical Characteristics.

DC ELECTRICAL CHARACTERISTICS

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Supply voltage for logic	VDD	—	4.8	5.0	5.2	V
Supply current for logic	IDD	—	—	35	40	mA
Operating voltage for LCD	VLCD	-10°C	—	—	—	—
		25°C	4.8	5.0	5.2	V
		+60°C	—	—	—	—
Input voltage "H" level	V _{IH}	—	0.7 VDD	—	VDD+0.3	V
Input voltage "L" level	V _{IL}	—	0	—	0.2VDD	V

LED BACKLIGHT CHARACTERISTICS

COLOR	Wavelength λ (nm)	Operating Voltage(±0.15V)	Spectral line half width Δλ (nm)	Forward Current (mA)
white	—	3.2	—	30

NOTE: Do not connect +5V directly to the backlight terminals. This will ruin the backlight.

Figure 10-9. This image shows you how to read a datasheet.

11.3.3 Schematics

A **schematic**, **schematic diagram** or **wiring diagrams/circuit diagram** are associated with electrical circuits. These diagrams show how the different components of a circuit are connected. These diagrams have lines representing connecting wires, while other components such as lightbulbs and switches are represented by standardized symbols called **electrical schematic symbols**.

Having a schematic diagram on hand may help a user design an entire circuit before building it or troubleshoot a circuit that has stopped working. Schematic diagrams can also be used to examine how an electronic can function, without detailing the hardware or software used in the electronic.

The next image is how you read a schematic. A schematic allows you to visualize the circuit as well as the necessary components that are required.

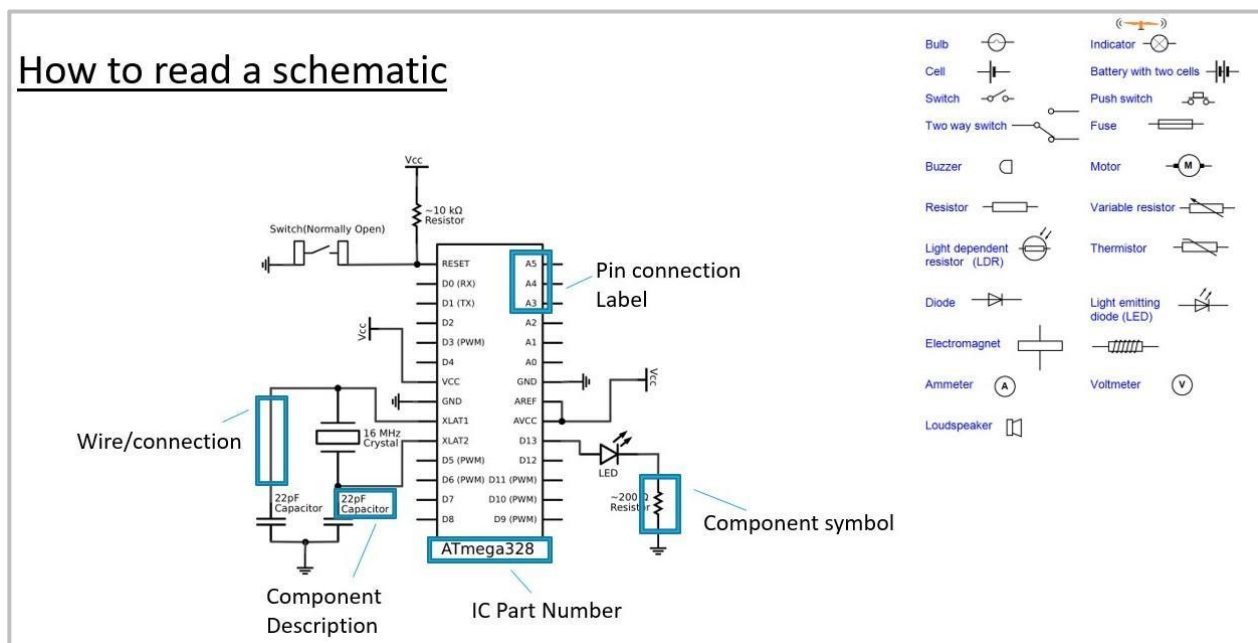


Figure 10-10. This image shows you how to read a schematic.

11.3.4 Code Sample

A code sample is used to demonstrate programming tasks that are not easily demonstrated in a full code. A code sample is a complete with references to all required source files in its description.

When you approach a code sample, try to put yourself in the readers' shoes. Ask yourself questions like these to enhance your samples and the content around them:

- “What do I want to learn from this sample?” (Not “What do I want to see this sample do?”)
- “How do I know that specific code blocks do what it looks like they are supposed to do?”
- “How can I map my own input and output requirements to those demonstrated in the sample code components?”
- “Where do I go to learn more? What OTHER resources are easily available?”

CHAPTER 12: ARDUINO CODING REFERENCE

12.1 Arduino Command Quick Reference

Arduino Command Quick Reference Table	
Command	Description
setup()	A function that runs once when the Arduino first starts. See also http://arduino.cc/en/Reference/Setup .
loop()	A function that is repeatedly run after the setup() is completed and until the Arduino is turned off. See also http://arduino.cc/en/Reference/Loop .
pinMode()	Sets the PIN entered as the argument to either output electricity. See also http://arduino.cc/en/Reference/PinMode .
OUTPUT	Keyword set in the second argument of pinMode() that says the pin will output electricity. See also http://arduino.cc/en/Reference/Constants .
digitalWrite()	Turns on or off the electricity at the specified pin. See also http://arduino.cc/en/Reference/DigitalWrite .
HIGH	Keyword used to turn on the electricity in digitalWrite(). See also http://arduino.cc/en/Reference/Constants .
LOW	Keyword used to turn on the electricity in digitalWrite(). See also http://arduino.cc/en/Reference/Constants .
delay()	Pauses the Arduino Uno for a specified number of milliseconds. See also http://arduino.cc/en/Reference/Delay .

12.2 Functions Reference

Functions: Digital I/O		
Command	Description	Example
pinMode(pin, mode)	Configures the specified pin to behave either as an input or an output. the pin is the pin number	pinMode (13, OUTPUT);
digitalWrite (pin, value)	Write a HIGH or a LOW value to a digital pin.	digitalWrite (2, LOW);

digitalRead (pin)	Reads the value from a specified digital pin. The result will be either HIGH or LOW	<code>digitalRead (4);</code>
tone(pin, frequency, duration)	Generates a sound	<code>tone (8, 1000, 2000);</code>

Functions: Analog I/O

Command	Description	Example
analogWrite(pin,value)	Writes an analog value (PWM wave) to a pin. Value is the duty cycle: between 0 (always off) and 255 (always on). Works on pins 3, 5, 6, 9, 10, and 9.	<code>analogWrite (9, 256);</code>
analogRead(pin)	Reads the value from the specified analog pin and returns a value between 0 and 1023 to represent a voltage between 0 and 5 volts (for default). It takes about 0.0001 seconds to read an analog pin.	<code>analogRead (A0);</code>

Functions: Serial Communication

Command	Description	Example
Serial.begin(9600)	Used to begin serial communications, typically at a 9600 baud rate (bits per second)	<code>Serial.begin (9600);</code>
Serial.print(Val,format) ;	Prints data to the serial port as human-readable ASCII text.	<code>Serial.print(78)</code> gives "78" <code>Serial.print(1.23456)</code> gives "1.23"

		<pre>Serial.println(1.23456, 4) gives "1.2346"</pre> <pre>Serial.print("Hello world.") gives "Hello world."</pre>
Serial.println(val);	Prints val followed by carriage return	<pre>Int i; i = analogRead (A0); Serial.println (i);</pre> <p>Will read the value of i and prints it</p>

Functions: Servo Motor Control

Command	Description	Example
#include <Servo.h>	Includes servo library in Arduino	<code>#include <Servo.h></code>
Servo motorname	Makes a name for the servo motor	<code>Servo armmotor;</code>
myservo.attach(pin)	Assigns Arduino pin to Servo	<code>armmotor.attach(9);</code>
myservo.write(Val)	Moves servo motor to the angle in Val	<code>armmotor.write (60);</code>
myservo.read()	Read the current angle of the servo (from 0 to 180 degrees)	<code>armmotor.read (armmotor);</code>

Functions: Time

Command	Description	Example
millis()	Returns the number of milliseconds since the Arduino board began running the current program	<pre>int tm; tm = millis();</pre>

delay()	Pauses the program for time (in milliseconds) specified as a parameter	digitalWrite (3, HIGH); delay (1000);
Variables		
Command	Description	Example
int i;	Will define the word i to be an integer variable	int i; i = 10;

12.3 Structures Reference

Structures: Comparison Operators
x == y (x is equal to y)
x != y (x is not equal to y)
x < y (x is less than y)
x > y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)
Structures: Boolean Operators
&& (and)
 (or)
! (not)

Structures: Control Structures	
Command	Description
if	Tests whether a certain condition has been reached. Used in conjunction with a comparison operator
Example:	

```

if (x > 120)
{
  digitalWrite(LEDpin, HIGH);
}
if (x > 120)
{
  digitalWrite(LEDpin, HIGH);
}
if (x > 120)
{
  digitalWrite(LEDpin, HIGH);
}
if (x > 120)
{
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}

```

Command	Description
if...else	Allows you to do multiple tests

Example

```

if (pinFiveInput < 500)
{
  // do Thing A
}
else if (pinFiveInput >= 1000)
{
  // do Thing B
}
else
{
  // do Thing C
}

```

Command	Description
while	while loops, will loop continuously, and infinitely until the expression inside the parenthesis, () becomes false

Example

```

var = 0;
while(var < 200){

```

<pre>// do something repetitive 200 times var++; }</pre>	
Command	Description
break	the break is used to exit from a do, for, or while loop, bypassing the normal loop condition
Example	
<pre>if (sens > threshold){ x = 0; break; }</pre>	

CHAPTER 13: TOOLS, COMPONENTS, AND ADDITIONAL INFORMATION

13.1 Starter Kits

Starter Kits are a useful set of essential items and instructions for taking up an activity or process for the first time. Starter Kits walk you through the basics of using Arduino in a hands-on way.

13.1.1 KeyeStudio Starter kit

KeyeStudio is a great place to buy Arduino hardware parts as well as Arduino Starter Kits. You get 80 components such as an Uno R3 board, a 5V Relay, a DHT11 Temperature and Humidity sensor, an ultrasonic sensor, a 9G Servo Motor, many LEDs, and more. This starter kit also comes with interactive courses that you can learn from and software, video, libraries and document tutorials to show how to use Arduino.



Figure 12-1. The components included in the KeyeStudio starter kit.

If you need more information you can visit:

https://www.amazon.ca/KEYESTUDIO-Starter-Arduino-Educational-Gifts/dp/B017D6TY14/ref=sr_1_3?keywords=keyestudio+uno+starter+kit&qid=1552058447&sr_gateway&sr=8-3

Alternatively, you can visit: <https://www.keyestudio.com/>

13.1.2 Arduino Starter Kit

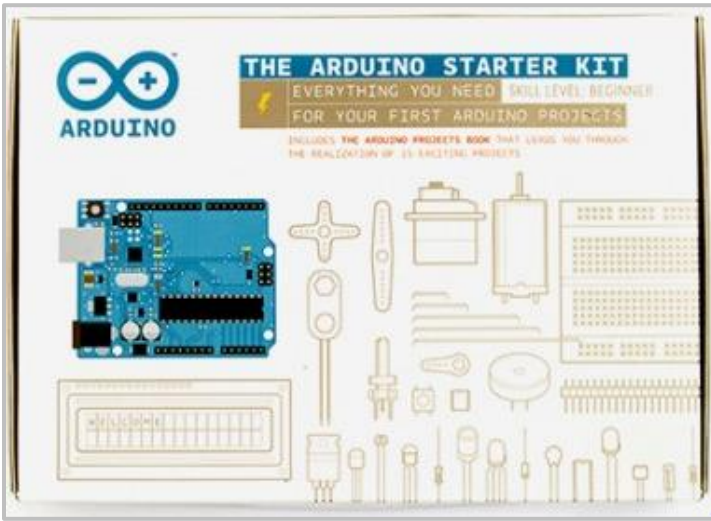


Figure 12-2. The Arduino starter kit.

The official Arduino website also contains a starter kit that you can purchase. The Starter Kit is a great way to get started with Arduino and coding. It has all the necessary components you need, to do fun projects following the step-by-step tutorials in the Project Book. Starting with the basics of electronics, to more complex projects, the kit will help you control the physical world with sensors and actuators.

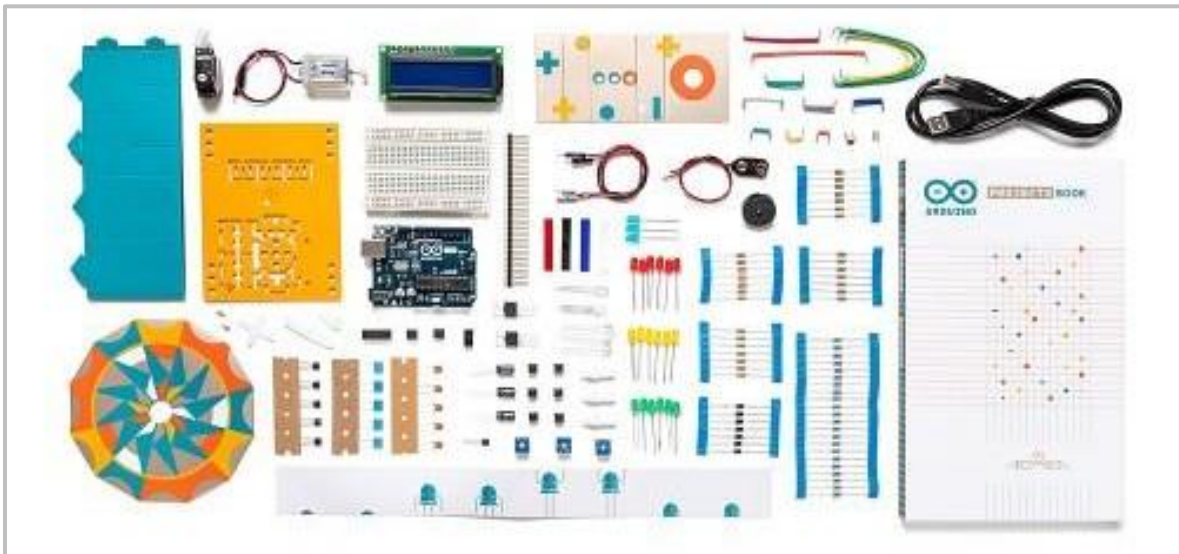


Figure 12-3. The components included in the Arduino's starter kit.

If you need more information or you still have questions, you can visit:

<https://store.arduino.cc/usa/arduino-starter-kit>

Alternatively, you can visit: <https://www.arduino.cc/>

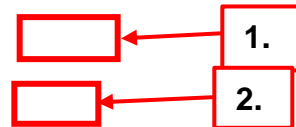
13.2 Additional Information on the Web

13.2.1 The Blog

A blog is an essential way to find, research and promote the projects that interest you. If you go to the official Arduino website and click on the Blogs in the Community section, to find different blogs that people have submitted with up-to-date projects and information you can use.

Got to the official Arduino website: <https://www.arduino.cc/>

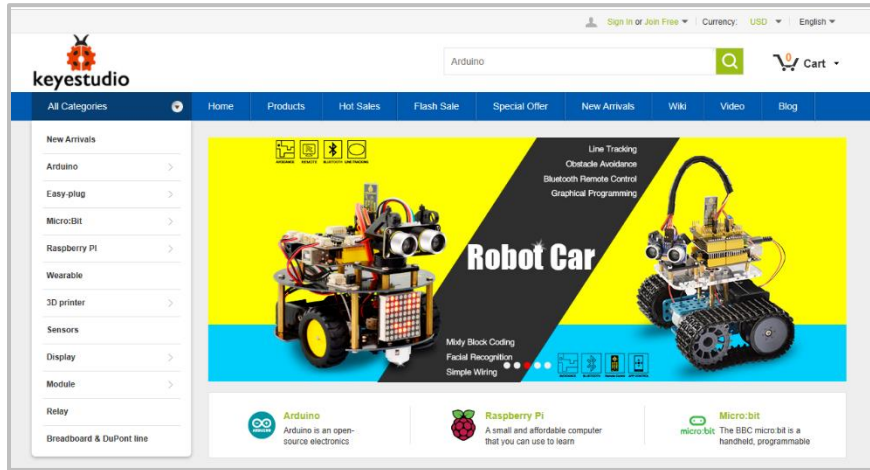
- 2) Under the community section, click on the BLOG on the top right corner and then you can scroll through the different projects people have made.



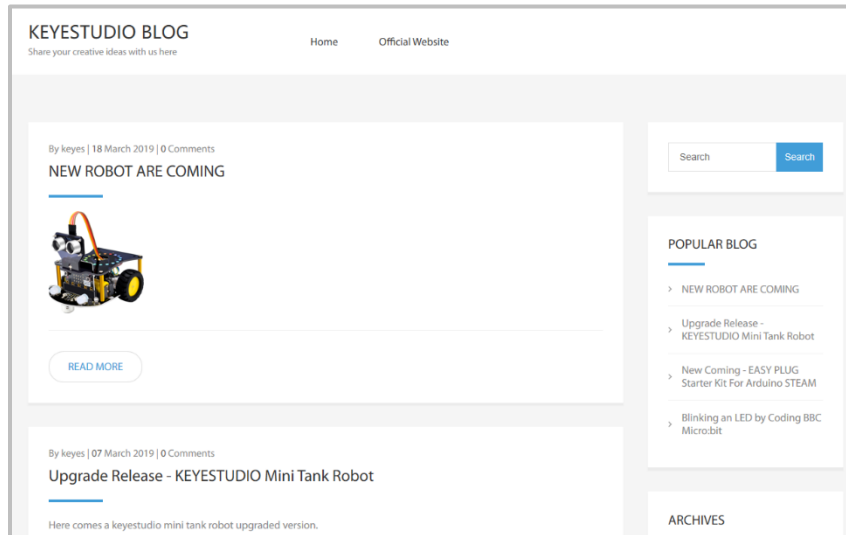
If you need more information or you still have questions, you can visit:

<https://blog.arduino.cc/>

1) Additionally another good blog resource is the official KeyeStudio website and blog for updates for their products: <https://www.keyestudio.com/>



Visit the blog here: <https://keyestudio.com/blog/>



13.2.2 Videos

Videos are a great way to learn and educate yourself on different topics and subjects. Videos are also beneficial when it comes to explaining the Arduino material. To better understand different topics in Arduino, such as sensors, motors, etc. You can visit the Exceed Robotics YouTube channel. On Exceed Robotics YouTube channel, you can find tutorials, explanations and features of many different Arduino hardware parts and projects. It is a great place to learn and understand the material, primarily since the professionals in this field teach it.

Other companies as well have useful videos that you can use to learn about Arduino projects in-depth such as:

- Adafruit: <https://www.youtube.com/user/adafruit/videos>
- SparkFun: <https://www.youtube.com/user/sparkfun/videos>
- Make Magazine: <https://www.youtube.com/user/makemagazine/videos>
- KeyStudio: <https://www.youtube.com/channel/UCS7bhtVrSE2Wpy2B12kIGdQ>

12.2.3 Websites

Here are websites where you can find additional information on the topic of Arduino:

- Reference: <http://exceedrobotics.com/>
- Reference: <https://keyestudio.com/>
- Reference: <https://www.arduino.cc/>
- Reference: <https://www.instructables.com/>
- Reference: <https://www.adafruit.com/>
- Reference: <https://www.learn-c.org>



CHAPTER 14: GLOSSARY

Actuator A device that translates an electrical signal into a real-world activities such as light, sound or movement. Examples include motors, lights and speakers.

Alligator clips Wires with spring-loaded clips that resemble the jaws of an alligator. They are useful for prototyping soft circuits or connecting components that don't use jumper wires.

Analogue A signal that varies between LOW and HIGH, as opposed to a digital signal. On the Arduino Uno, an analogue signal can be measured as a number between 0 for ground and 1023 for 5V. An analogue signal can be output as a value between 0 for 0V and 255 for 5V.

Anode The positive leg of a directional component, such as the long leg of an LED.

Argument A piece of information given to a function, which the function then uses to perform its task. The argument goes inside the brackets that follow the function name. For example, the function `delay(1000)` has the argument 1000, which is the number of milliseconds you want the Arduino to wait before executing the next line.

Array A list of the same type of thing in code. For example, an array can hold a list of ints.

Binary A number that uses only the digits 0 and 1, as opposed to decimal, which uses the digits 0 to 7. Binary is also known as base-1. Decimal is referred to as base-8.

Breadboard A reusable device that allows you to create circuits without needing to solder all the components. Breadboards have some holes into which you push wires and components to create circuits.

Capacitance The ability to store an electrical charge. Electrical components built especially to hold a charge are called capacitors, but other objects – even people – also have capacitance.

Cathode The negative leg of directional components, such as the short leg of an LED.

Comments Notes within your code that explain what a line or section of code is intended to do. Each comment line begins with `//` or, if you want to write a comment that spans multiple lines, it is placed between `/*` and `*`. These special characters tell the computer running the program to ignore that line or lines.

Compiling The process of taking code written by a human and turning it into instructions that can be understood by a machine.

Current The rate at which electrical energy flows past a point in a circuit. It is the electrical equivalent of the flow rate of water in pipes. Current is measured in amperes (A). Smaller currents are measured in milliamperes(mA).

Debugging The process of locating the cause of any errors in your computer program code and fixing them.

Declaring Where a new variable is created by giving it a name and a data type such as int. The variable does not hold value until it is given its first value.

Digital A signal that is only either on or off, or HIGH or LOW. On the Arduino Uno, a HIGH signal is 5V, and a LOW signal is ground.

Direct current (DC) The type of electricity used in Arduino circuits. It's the same kind that is generated by a battery and is the opposite of alternating current (AC), which is what comes out of mains plugs in the wall.

Driver A piece of software that lets your computer communicate with an external device, such as a printer or a keyboard.

Dual in-line package (DIP or DIL) One possible shape of an IC chip. It has two rows of legs that can fit into a breadboard.

Duty cycle The ratio of time a signal is HIGH versus LOW in a given cycle. In PWN, the higher the duty cycle, the higher the output voltage.

Float A data type for numbers that aren't whole numbers, but include a decimal place such as 1.3 or -54.087.

Floating input A pin that is not connected to anything. The pin reads in random values if it is not connected to a voltage source such as ground, 5V or a sensor.

for-loop A programming device that repeats a block of code for a predetermined number of times.

Function A set of lines of code that have a name. A function can be used over and over again. It may take some information as input and output more information when it is finished, but not all functions need to do that.

Instantiation Giving a variable value for the first time. Instantiation can happen at the same time you declare the variable, or you can do it later, but the declaration always needs to come first.

Integrated circuit (IC) Circuits contained within a single chip. The same circuit can be put into differently shaped chips, called packages. When working with a breadboard, you need what is known as a DIP or DIL package. That's the shape that has legs that fit into a breadboard.

Integrated development environment (IDE) A software application that is used to write computer code in a particular language; it's also referred to as a programming environment. The application can create and edit code, as well as run (or execute) the code. Many IDEs also provide features to help programmers debug their programs – in other words, check their programs for errors.



Light-emitting diode (LED) An electrical component that lights up when electrical current flows through it. A diode only lets electricity flow in one direction, so an LED lights up only when the long leg is connected to the positive side of a power source, and the short leg is connected to the negative side. If the legs are switched, the LED won't light up.

Library A collection of reusable functions in code that can be imported and used in multiple sketches.

Light-dependent resistor A resistor that changes its resistance according to how much light it is exposed to. It is also sometimes called a photoresistor.

Long A data type that can hold whole integer numbers from $-2,147,483,648$ to $2,147,483,647$.

Newline character A character that represents what happens when you press the Enter or Return key on your keyboard.

Ohm's Law The mathematical relationship between voltage, current and resistance. Voltage equals current multiplied by the resistance – or, put another way, $V = IR$.

Panel mount push button A push button that is designed to be mounted inside a case. It comes with a nut and washer to secure it to a panel.

Piezo A crystal that expands and shrinks when electricity is run through it. It also generates electricity when it is squeezed or bent.

Potentiometer A type of resistor with an adjustable knob to vary the resistance of current.

Pull-up resistor A resistor that is connected to the high voltage in a circuit, which sets the default state of the pin on that circuit to HIGH. The resistor is usually $10k\Omega$.

Pulse width modulation (PWM) How the Arduino board generates an output signal between 0V and 5V. The signal switched quickly between LOW and HIGH, and the resulting output voltage is between two voltages.

Red-green-blue light-emitting diode (RGB LED) A signal LED with four legs that contain three lights: one red, one green and one blue. The three lights share either a common anode or a common cathode.

Resistor An electrical component that resists current in a circuit. For example, LEDs can be damaged by too much current, but if you add a resistor with the correct value to the LED circuit to limit the amount of current, the LED is protected. Resistance is measured in ohms or Ω . You need to pick a resistor with the correct value to limit the current through a circuit; the value of a resistor is shown by coloured bands that are read from left to right.

Sensor A device that detects something in the real world such as light, sound or movement and translates it into an electrical signal. Examples include potentiometers and light-dependent resistors.

Serial communication A way that two devices, such as a computer and an Arduino board, can send and receive data to each other. One piece of data is sent at a time.

Servo A motor that can be controlled to rotate to a specific position. It usually can't rotate more than 180 degrees.

Shift register A device that can control multiple outputs with relatively few inputs. It is commonly used to control a large number of LEDs.

Sketches Arduino programs. The name comes from the quick drawing artists make.

Soft circuit Circuits built with flexible materials such as conductive thread and fabric. Soft circuits are often used in projects that are going to be worn.

Surface-mount device (SMD) One possible shape of an IC chip or another component such as a resistor. It is made for soldering onto a flat surface without any legs being inserted into holes on a circuit board.

Switch A component that either disrupts or redirects the flow of current in a circuit.

Tactile pushbutton A type of switch. A push to break pushbutton interrupts the flow of current in a circuit when it is pressed. A push-to-make pushbutton does the opposite and interrupts current only when it is not pressed.

Two-dimensional array Data stored in rows and columns, like in a spreadsheet.

Variable A code construct that holds a value that can be changed. For example, the variable green LED stores number 4.

Voltage The difference in electrical energy between two points in a circuit. It is the electrical equivalent of water pressure in pipes, and it is this pressure that causes a current to flow through a circuit. Voltage is measured in volts (V).

Voltage divider A circuit that outputs a fraction of the input voltage. It is a useful circuit for translating a change in resistance to a change in voltage.

Thanks to these resources used for information and images put together in this handbook:

<http://onlineresize.club/pictures-club.html>

http://www.resistorguide.com/resistor-color-code/#4_band_resistor

http://interactionstation.wdka.hro.nl/wiki/Arduino_basics_workshop

<https://store.arduino.cc/usa/>

https://wiki.keyestudio.com/Ks0084_keyestudio_New_sensor_kit_with_2560_R3

<https://components101.com/microcontrollers/arduino-uno>

<https://electronicsclub.info/circuitdiagrams.htm>

https://www.diffen.com/difference/Current_vs_Voltage

https://www.tutorialspoint.com/arduino/arduino_functions.htm

<https://startingelectronics.org/software/arduino/learn-to-program-course/15-functions/>

<https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all>

<http://www.learningaboutelectronics.com/Articles/What-is-negative-voltage>

<https://www.redbubble.com/people/lessonhacker/works/11635070-programming-symbols-coding-literacy?p=poster>

<https://en.wikipedia.org/wiki/Schematic>

<https://learn.sparkfun.com/tutorials/how-to-read-a-schematic/all>

<https://www.thoughtco.com/what-is-a-schematic-diagram-4584811>

