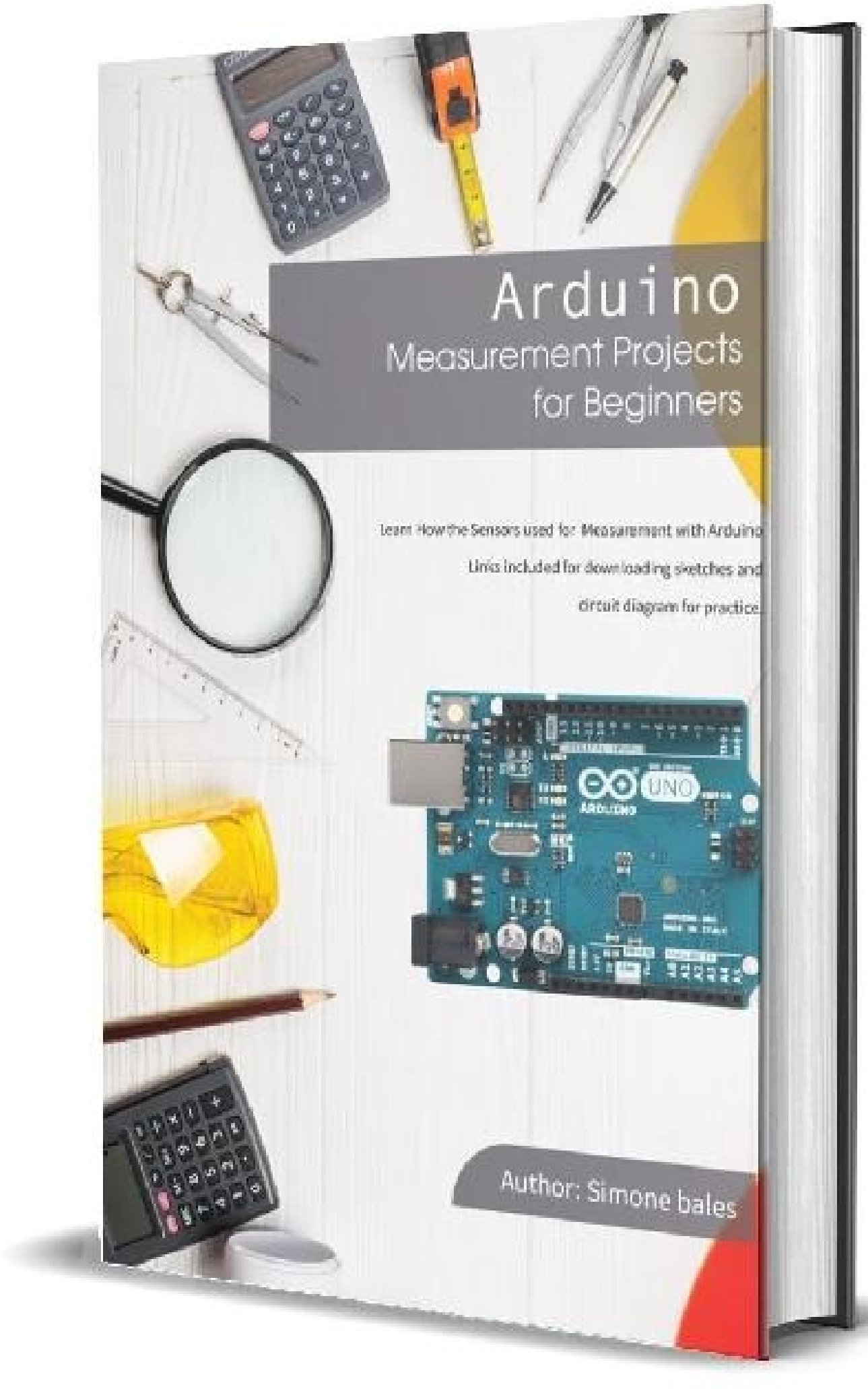

Arduino
Measurement Projects
for Beginners
Learn How the Sensors used for Measurement with Arduino
Unos included for downloading sketches and
circuit diagram for practice
UNO
ARDUINO
Author: Simone bales

Contents

Basics of Electronics

Electronics Primer

Before jumping into the Arduino Completely, we present the foundations of electricity and electronics and introduce critical concepts used. Here provides a broad overview of electronics to readers who have little to no experience with electronics and electricity. If you're already comfortable with some of the topics in this chapter, you can skip these and go to the programming page.

If you're new to electronics or just want a refresher, we suggest reading this webpage in full. While it's not a complete guide to electronics (there are whole books, classes, and degrees on the subject), this chapter is a handy reference designed to arm you with basic concepts and vocabulary.

What Is Electricity?

To understand electricity, you first need to understand the structure of an atom. Atoms are the building blocks of everything around you. An atom is made up of protons, neutrons, and electrons. The electrons have a negative charge, and the protons have a positive charge. A typical atom has the same number of electrons as protons and therefore is neutral in charge. Electricity is a form of energy that involves the movement or storage of charges; it is the phenomenon that occurs when we push or force charges to move in a prescribed manner or a defined path.

In simple terms, electricity is a form of energy that we can harness and convert into heat, light, movement, and power. Electricity has three main properties that will be important to us as we build projects: current, voltage, and power.

Current

The flow of electrical energy through a circuit is called the current. Electrical current flows through a circuit from the positive side of a power source, such as a battery, to the negative side of the power source. This is known as direct current (DC).In some circuits; the negative side is called ground (GND). Current is measured in amperes or “amps” (A). Small amounts of current are measured in milliamps (mA), where 1,000 milliamps equal 1 amp.

Voltage

Voltage is a measure of the difference in potential energy between a circuit's positive and negative ends. This is measured in volts (V). The higher the voltage, the faster the current moves through a circuit.

Power

Power is a measurement of the rate at which an electrical device converts energy from one form to another. Power is measured in watts (W). For example, a 100 W light bulb is much brighter than a 60 W bulb because the higher-wattage bulb converts more electrical energy into light. A simple mathematical relationship exists among voltage, current, and power:

Power (W) = Voltage (V) × Current (A)

What Is a Circuit?

Even with the electrical forces pushing them, charges need a path to follow from a point of higher potential to the point of lower potential. The path by which charges move from the positive (+) side of a battery (high potential) to the negative (–) side of the battery (low potential) is called a circuit. A circuit consists of a closed path from the positive terminal to the negative terminal through a device such as a light-emitting diode (LED), resistor, light, or motor.

In order for charges to move, the path must be made out of a material that is conductive. Conductivity is not an absolute measure but more of a continuum. In other words, some materials allow charges to move more freely than others.

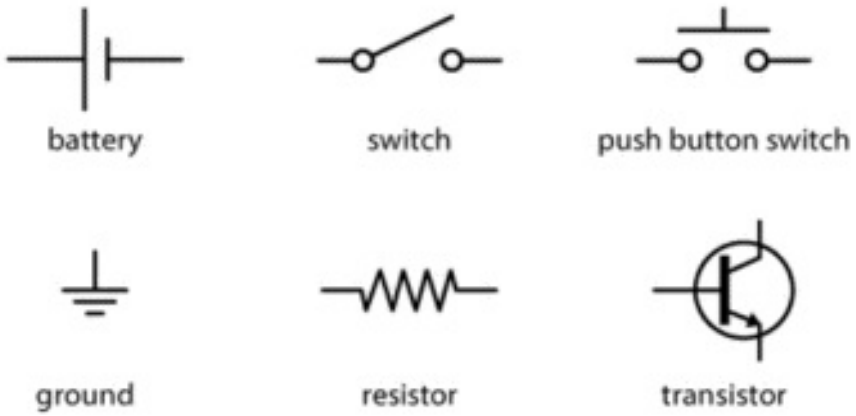
Ohm's Law

As you may already have guessed, there is a relationship between current, voltage, and resistance. This relationship is commonly called Ohm's Law, and it is represented mathematically as follows:

$$\mathbf{V = I \times R}$$

In this equation, V represents the voltage; I represent the current, and R is the resistance.

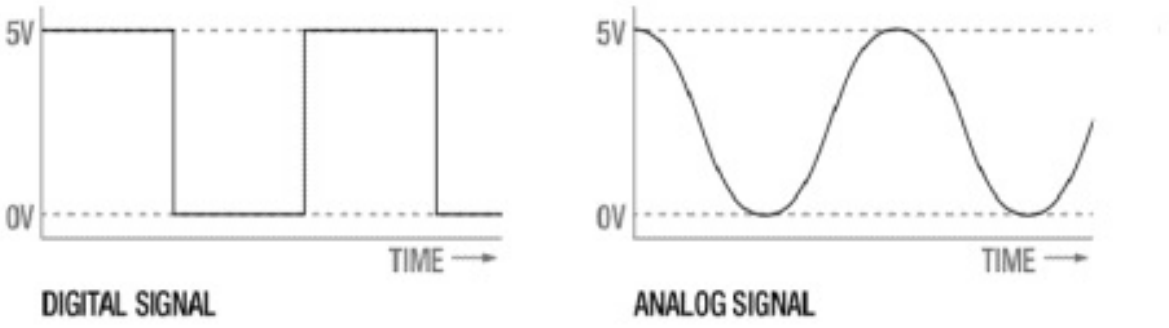
Schematics



Schematics are simplified drawings of circuits. We sometimes also call these wiring diagrams or circuit blueprints. A schematic shows what is connected to what and which components to use in building the circuit.

Analog VS Digital

With the concepts of circuits, components, voltage, current, and resistance defined, we can now talk about two different approaches to electronics: analog and digital. These approaches are not mutually exclusive, and you can't really understand the circuits you build without understanding both concepts.



Digital versus analog signals

Analog deals with values that vary within a set range. Think of the dimmer switch in some dining rooms; that is analog. Analog values can be on, off, and anything in between. Digital values, on the other hand, have only two. States: on or off.

Digital electronics tend to include a microcontroller or microprocessor that is programmed to turn things on and off in response to conditions, whereas analog circuits tend to use components to vary the current, voltage, and resistance of a circuit to achieve the same result. There are advantages and disadvantages to both ways of thinking, but you can't solely use one and not the other. For example, you couldn't read the temperature using a microcontroller without using a number of analog components as well.

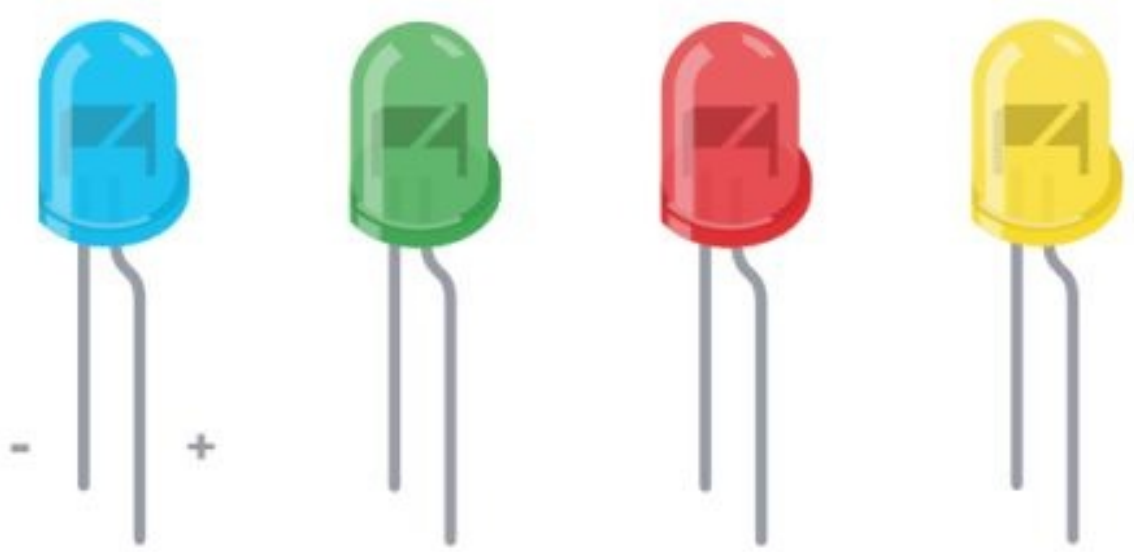
What is Microcontroller?

A microcontroller is a small computer that you can program by uploading a program or set of instructions. Microcontrollers are used to automate simple tasks, like controlling the temperature of your house or watering your lawn when it's dry.

LED

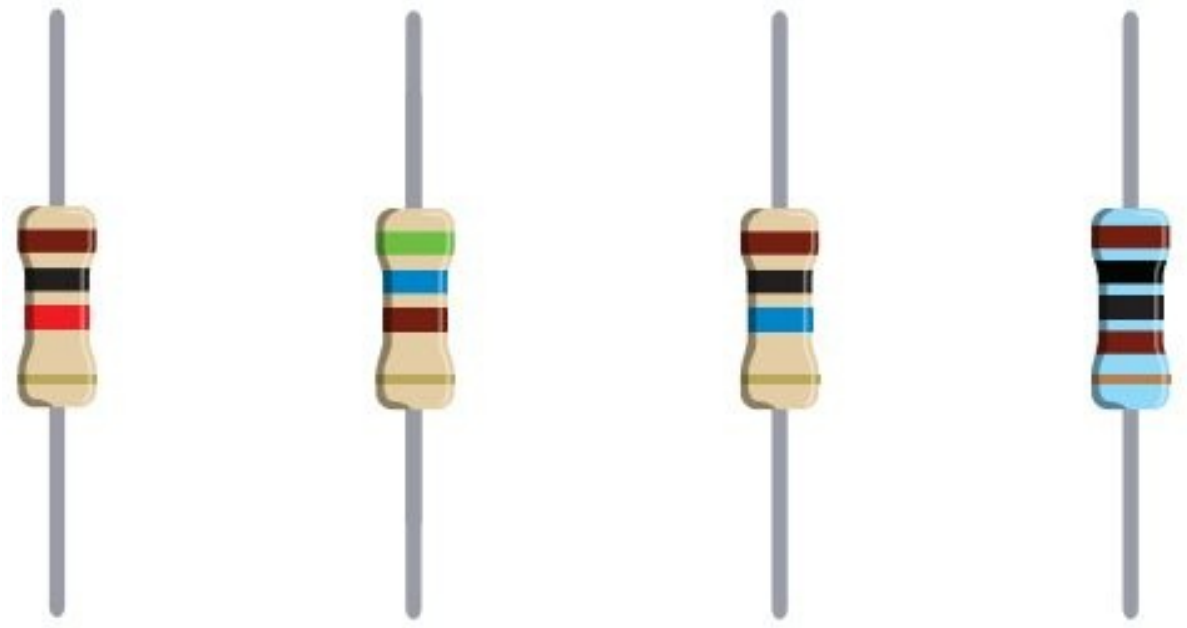
An LED is an acronym for Light Emitting Diode; it is a type of diode that illuminates when electricity passes through it. Their voltage drop is higher than a standard diode, from1.2v for an infrared LED, up to 3.5v for blue and ultraviolet LED. These LED's comes in different size and different colors. If you take a closer look to that LED picture, one lead (leg) of the LED's shorter and another one is longer. The shorter one is known as a cathode (negative (-) symbol), and the longer one is known as an anode (positive (+) symbol). If the current passing through the LED is too high, you will lose LED. A LED can withstand up to 3.5v, the output voltage from the Arduino pins are 5V to step down this voltage,

a resistor in series with LED should use to prevent the LED from hurt.



Resistors

Resist the flow of electrical current in the circuit, in other words, they limit the flow of electrons. The SI unit of resistance is Ohms, written as the Greek letter omega (Ω). They are often used with the unit prefixes kilo (k) and mega (M). The value of resistor indicated by 4 or 5 color band bands, using the resistor color code: The first 2 bands are the first digits of the value, and the 3rd band is the power of ten that comes after those 2 digits. This is also called the multiplier and is just the number of zeros you have to add. The last band is the tolerance and mostly it will be silver or gold color.



"BB ROY Great Britain Very Good Wife," try to memorize this sentence which allows you to remember 9 color bands in the resistor. First b represents black, the second b represents brown, R represents Red, O represents Orange, Y represents yellow, first word in Great G represents Green, first word in Britain B represents Blue, first word in Very V represents violet, the first letter in G represents Grey and, the first letter in Wife W represents White.

Color	1st Band (1st digit)	2nd Band (2nd digit)	3rd Band (Multiplier)	4th Band (tolerance)
Black			1Ω	

Brown	**1**	**1**	10Ω	±1%
Red	**2**	**2**	100Ω	±2%
Orange	**3**	**3**	1000 or 1kΩ	
Yellow	**4**	**4**	10kΩ	
Green	**5**	**5**	100kΩ	±0.5%
Blue	**6**	**6**	1MΩ	±0.25%
Violet	**7**	**7**	10MΩ	±0.1%
Grey	**8**	**8**		±0.05%
White	**9**	**9**		
Gold			0.1Ω	±5%
Silver			0.01Ω	±10%

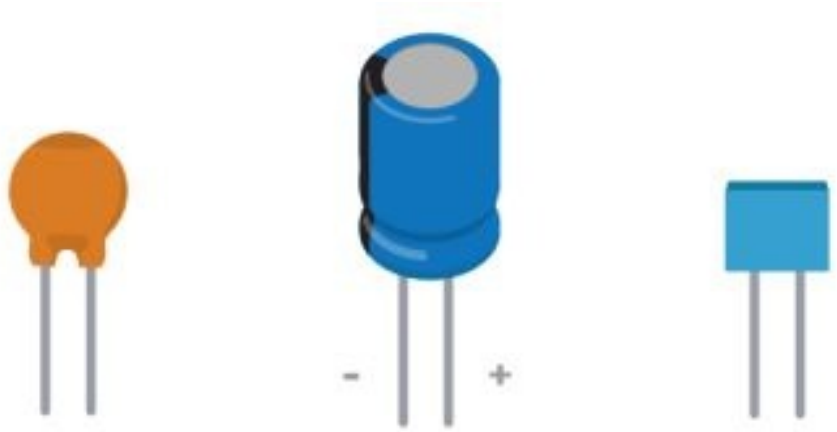
Eg: The first resistor toward right, in the above picture, shows Brown RED RED color, First band is Brown look up the table to find 1st digital value for this color,

value for the brown color in the first digit is ‘1’, second color is Red, if you look up the table for the 2nd digit value color for the Red is 2 and the third color is also red, if you look up the table for 3rd Band it is multiplier and the value is 100 ohms If we join all the value together, 1 2 * 100

And the final value will be 1200; the resistance value is 1200 Ω.

Capacitors

These components store and release electrical energy in a circuit. It behaves like a small rechargeable battery. Capacitors are used in filters. The SI unit of capacitance is farad, of F. This is a huge unit, and most often, you will see prefixes like pico(p), nano (n) or micro (µ). Often placed across power and ground close to a sensor or motor to help fluctuation in voltage.



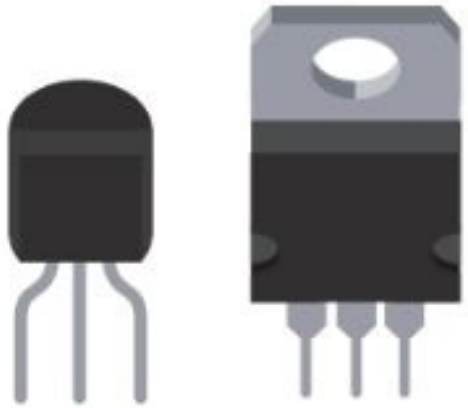
Transistor

A transistor is a semiconductor device, which is used to switch or amplify a signal. You can think of it as a switch that can be operated by using a feeble signal, a current controlled switch.

A transistor has three terminals: they are called the base (B), the emitter (E) and the collector (C).

The emitter 'emits' electrons, and they are 'collected' by the collector. The base is used to control this flow of electrons. If a small current flows from the base to the emitter, a much larger current will flow from the collector to the emitter. How much larger this C-E current is, depends on a constant, specific to the type of transistor. This constant is called the DC current gain and has the symbol of the Greek letter bèta (β) or Hfe.

E.g., if you have a transistor with $\beta = 100$, and your B-E current = 10mA, your C-E current will be 1A. This principle is used in amplifiers. However, a transistor cannot keep on amplifying forever: at a certain point, the transistor will just act like a switch.



Diode

Just like a transistor, a diode is a semiconductor device. One of the unusual properties of a diode is that they only conduct electricity in one direction.

For example, Arduino boards have a diode in series with their power input jack, to prevent you from reversing the power, and damaging the chip.

Diodes have a forward voltage drop ranging from 0.5v to 0.7v. This means that if you measure the voltage before the diode, it will be about 600mV higher than after the Diode, a diode has its limits: if the reverse voltage is too high, it will break, causing it to let current pass in the wrong direction. In some diodes, this is done in a controlled way. These diodes are called zener diodes. They will only conduct if the voltage is higher than an absolute value, specific to the zener. This value is constant, so zener diodes are used as a reference in voltage regulators.

Variable Resistor

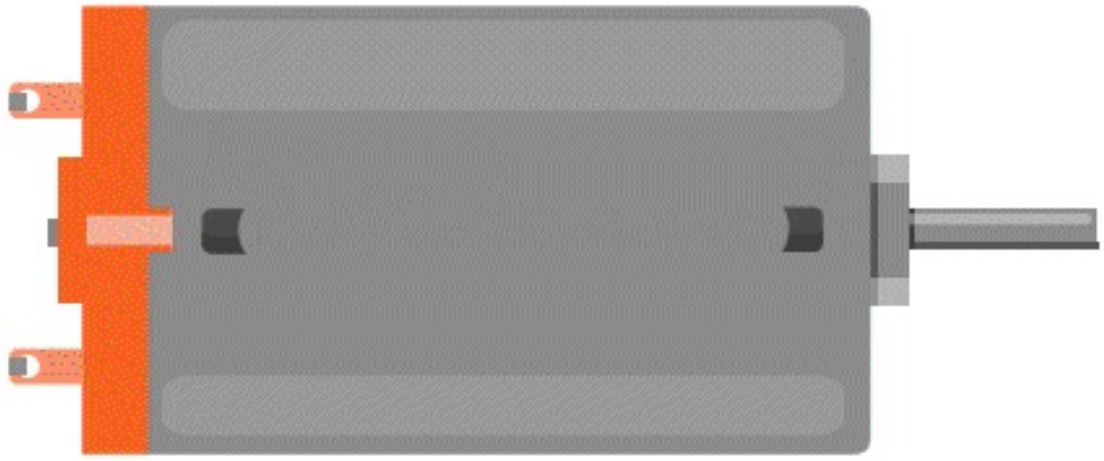
We have seen about Resistors previously, and the resistor value in those resistors cannot be changed, but in this resistor, it is possible to change the resistance value. These resistors come with three pins. Two pins are connected to the ends of a fixed resistor. The middle pin or wiper moves across the resistor dividing it into two halves. When the external sides of the potentiometer are connected to the voltage and ground the middle leg will give the difference in voltage as you turn the knob. This type of resistor also called potentiometer or Pot. For example, if you have a 10k ohms variable resistor, you can vary the resistance from 0 to 10k by rotating the knob. Mostly these type of resistors used with Arduino to control PWM pins available in the Arduino.

Light Dependent Resistor

As the name suggests, these are the type of resistor where the resistance change depends on the amount of light falls on the resistor head. This is also known as a photo sensor or light sensor, most of the yesteryears smart products use this simple sensor to do some magic. If you want to make an automatic light control system, it is possible with this sensor. Resistance increase or decrease depends on the type of sensor.

DC Motor

A simple motor allows us to experiment with Arduino and work with fun projects. This simple DC motor which cannot be used for making movable objects, this motor cannot produce necessary torque to move objects that’s the reason these motor coupled with gears to increase torque and do some useful work. A motor converts electrical energy into mechanical energy when electricity is applied to its leads.



Push Button

Push button or momentary switch used for giving inputs to Arduino. You can use this device to know how signal goes high or low based on the switch state.

Temperature Sensor

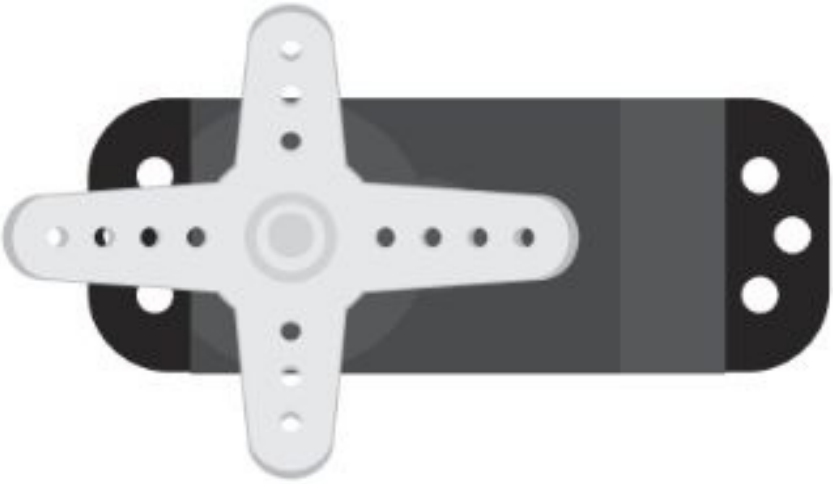
Lm35 is an Analog temperature sensor. This sensor can be used with any Arduino analog pins A0 to A5. You have to do an analog read and do some mathematical calculation to change the analog value to Temperature in Celsius or Fahrenheit. You can use any analog type sensor to get a temperature value.

Piezo Buzzer

Piezo buzzer is an electronic device commonly used to produce sound.

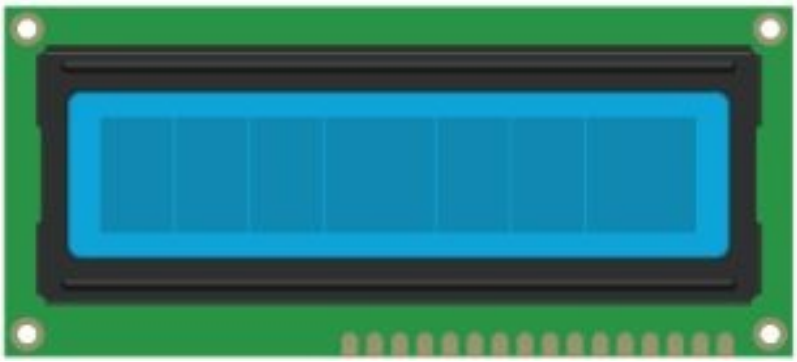
Servo Motor

These are particular type of motor, these motor are capable of rotating to a precise degree, but these motor can move only from 0 to 180 degree. It is controlled by sending electrical pulses from Arduino.



16x2 LCD display

Liquid display crystals are a type of alphanumeric or graphic display based on liquid crystals. These displays are available in many shapes and sizes, the most popular are the 16x2 type displays which have 2 rows and capable of display 16 characters on each row.

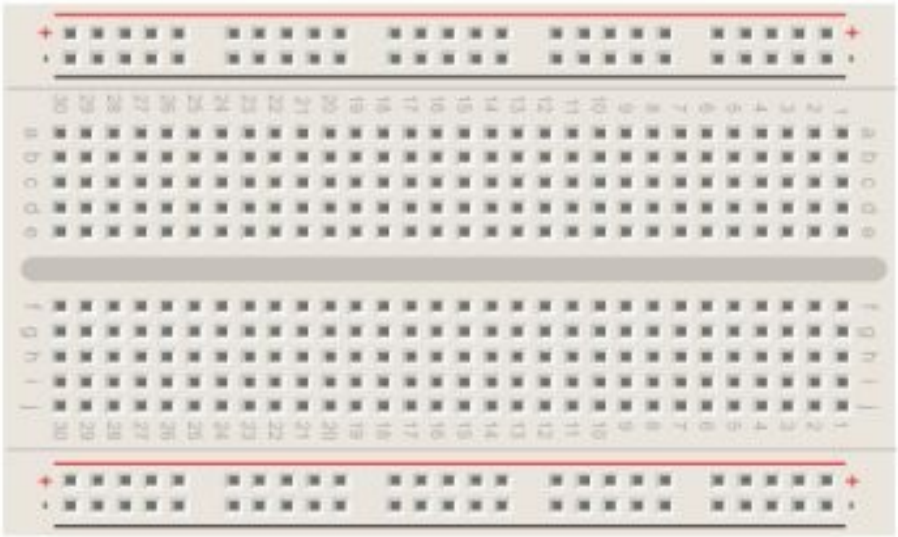


Connecting wires

These wires are very useful for connecting different components with Arduino. There are different types of wires available male to male, male to female, female to female. You should have 5 wires from each type for ease of connecting different components with Arduino.

Breadboard

The breadboard is the most critical component in any project. This board allows the user to build circuits. It’s like a patch panel, with rows of holes that allow you to connect wires and components together. This eliminates the need for soldering of components.



Arduino

An Arduino is an entirely open source DIY microcontroller become very famous among the maker community, Arduino is a programmable device that can add brain to non-intelligent things like a Robot. You can do wonders with this tiny device once you get your hands dirty with it. You can use an Arduino to blink some LED's to run robots around you, and even you use it to create handheld gaming devices for you. We delve more into Arduino details in this chapter.

There are loads of Arduino variants available in the market and Arduino Makers are catering to this board in many sizes and shapes that fit into their products or projects.

Some of the Arduino that mainly used are listed Arduino Uno, Arduino Mega, Arduino Nano. Throughout the book, Arduino Uno used for doing projects and explaining with examples. These boards come from authorized manufactures as well as others who copy the same design and sell in their brand name. You can buy a clone version of an Arduino for very less price compare to the official Arduino boards.

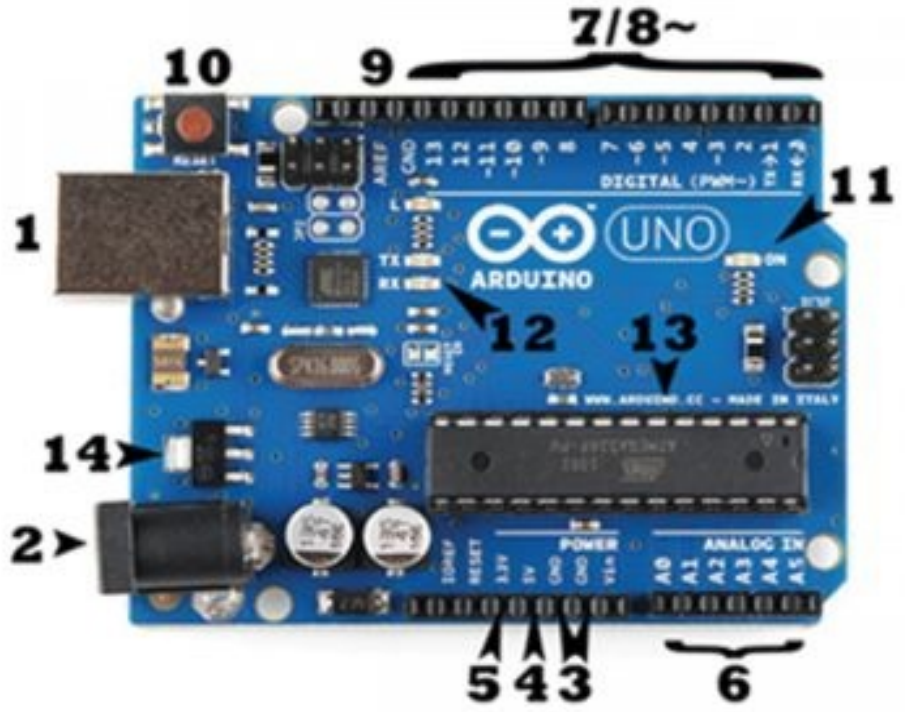
Arduino Uno

Arduino Uno is a famous and most used Variant of Arduino, this board based on the ATmega328p microcontroller. It has 14 digital input/output pins (Some cases when needed analog pins can also use as digital pins) Out of the leading 14 digital pins 6 can use as PWM outputs, and this board has 6 analog inputs, a 16 MHZ crystal oscillator, a power jack, a reset button, an ICSP header, and a USB connection. This board contains everything needed to support the microcontroller; This comes as a ready to use a device where it can be directly connected to a computer with a USB cable and start to program it and use with the project we have it in mind. This board can be powered with an external battery or power it with an AC-to-DC adapter. This development board is different from all the previous boards in that it does not use the FTDI USB to Serial driver chip.

The Arduino UNO can be driven from an AC to DC adapter or from a USB cable, AC to DC adapter which is terminated in a barrel jack. In the picture

below the USB connection is labeled (1) and the barrel jack is labeled (2).

Note: Recommended Voltage supplied to Arduino should not exceed greater than 12v, the voltage should be supplied between 6v to 12v



Pins (Analog, Digital, PWM, AREF)

These are the pins responsible for connecting wires and complete the circuit and make your dream project come true.

Digital Pins (labeled as 7): There are 14 digital pins in an Arduino Uno (0 to 13). These pins can be used to turn on/off a digital circuit and also used to take digital input from external devices, i.e., these pins can be configured and used as digital input as well as digital output.

These pins can produce Two states: HIGH and LOW, i.e., the HIGH state produces 5v and LOW states produces 0v. Each of these digital pins can provide or receive a maximum of 40 mA and has an inbuilt internal pull-up resistor (by default these are disconnected from the pins internally) of 20-50k Ohms; Also, some pins have specialized functions that can programmatically be done as follows:

- PWM (8): PWM expanded as (Pulse width Modulation) if you look closely in your Arduino Uno board you may notice a

symbol (~) next to some of the digital pins (11, 10, 9, 6, 5, 3), These pins act as a standard digital pins and also used for PWM functions, Think of these pins as an analog signal producer

- AREF (9): AREF stands for Analog Reference pin. These pins are mostly not used, and it is sometimes used for setting a Voltage reference externally between 0v to 5v as the upper limit for the analog input pins.
- Serial (0 –RX) and (1 – TX). These are the essential pin and responsible for serial communication between the Arduino and a PC for transferring of programs from the Arduino IDE to the Arduino board. Used to transmit and receive serial data.
- Analog Pins (Marked as 6 in the above picture) These pins are designed to receive analog signals and are labeled as Analog In in the Arduino Uno board. These pins receive an analog signal from sensors and convert it into a digital value that we can read programmatically.

Power (5v, 3.3v, GND)

- GND (Marked as 3 in the above Arduino Uno picture) There are 3 ground pins on the Arduino, any of the pins can be used to ground your circuit.
- 5V (Marked as 4 in the above Arduino Uno Picture) The pin 5v supplies 5v of power, and this can be used for powering an external circuit.
- 3.3V (Marked as 5 in the above Arduino Uno Picture) The pin 3.3v supplies 3.3v of power, and this can be used for powering an external circuit.

Reset Button

The Arduino has a reset button (Marked as 10 in the picture). The reset button in the Arduino does pretty much the same as switching off the board and switching it back in. It restarts your program from the beginning. When you Press the reset button on the Arduino board, the LED on pin 13 must flash some time while it's in the bootloader before it runs whatever program you have programmed in. If that LED doesn't flash when you press the reset button, then there is a serious fault with your board which will take further diagnostic.

Power LED Indicator: (Marked as 11 in the picture) This LED is located

near the ICSP pins. This LED should turn on when you plug the Arduino board to a power source. If this light does not turn on, there is something wrong with your Arduino board.

TX, RX LEDs

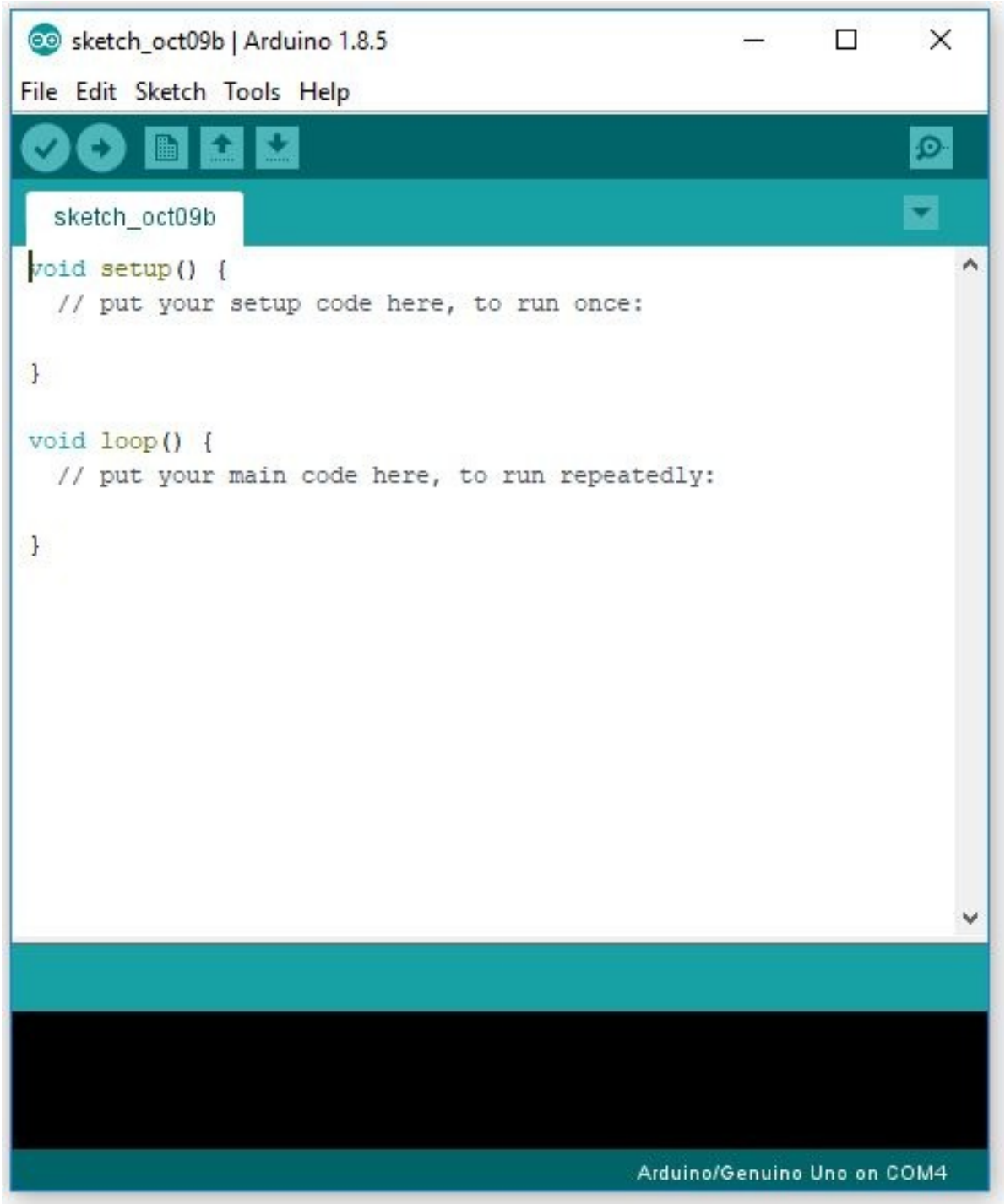
TX is short for the transmitter, RX is short for the receiver. These pins are responsible for serial communication. These LEDs (11, 12) will give us some excellent visual indications whenever our Arduino is receiving or transmitting data **ATmega328p IC**

Each Arduino board has its IC, or Integrated Circuit (13) or the microcontroller is the main component or can be said as the brain of the Arduino. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.

Voltage Regulator

The voltage regulator (14) controls the amount of voltage that is allowed into the Arduino board. Voltage regulators are acting as a gatekeeper; it will turn away an extra voltage that might harm the circuit. These voltage regulators have its limit, and it cannot control the voltage beyond its capacity, so don’t hook up your Arduino to anything higher than 20 volts.

Introduction to the Arduino IDE



In the last chapter, it’s full of Arduino hardware, and it's detailed, in this

chapter, we will look into the Arduino IDE (Integrated Development Environment) that is used for Programming the Arduino Hardware.

Note: This chapter deals with installing Arduino IDE in windows, another operating system may differ.

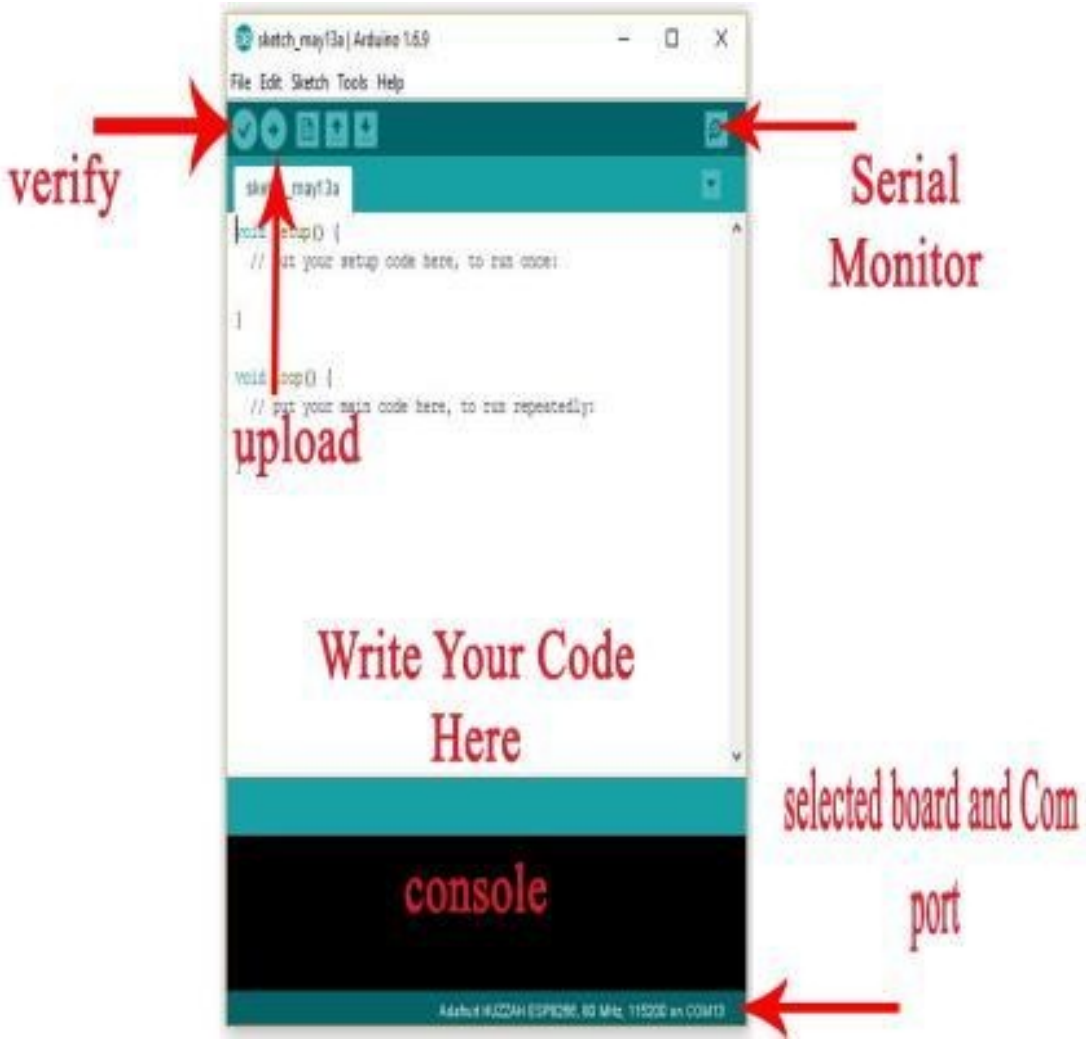
It's pretty straightforward to install Arduino IDE on a windows machine.

Firstly Download the IDE, start by visiting the Arduino's software page. https://www.arduino.cc/en/Main/Software) The Arduino IDE is available almost all the operating system including Linux, Mac OS, and window, make sure to download the right flavor of IDE that suitable to your operating system. If you are using an earlier version like windows 7 or older operating system, do not download the windows app version, as this needs Windows 8.1 or later.

Once you downloaded the Arduino IDE, go ahead and install the IDE, enable all the options during installation, including libraries and any USB drivers and make sure to read the EULA.

The Arduino IDE

The Arduino IDE is a great tool provides a complete environment for all the Arduino-based projects. The IDE looks very minimalistic, but it provides all the necessary library and function to make any complicated Arduino projects. The top menu bar has standard options like File, Edit, Tools, Sketch, Help.



The middle section of the Arduino IDE is like a text editor where all the programming coding is done here. The bottom section of the IDE called as Output console, that is used to see the status of the compilation, any errors in the program, how much memory has been used, and various useful messages are displayed in the console.

The Arduino IDE in its default state

Projects made using Arduino IDE are called as sketches, and these sketches are usually written in a short version of C++ (many features of C++ is not included in the Arduino IDE). Because programming a microcontroller is different from programming a computer, there are some libraries included in for Arduino hardware like – reading a pin, writing to a pin, reading analog values and other hardware specific functions, this often confuses many people, and they think Arduino is programmed in an “Arduino language.”

However, the Arduino is programmed in C++ and uses some unique Arduino

Hardware specific libraries.

The SIX buttons

Most projects in Arduino relay on these 6 buttons found below the menu bar.

1. The Tick mark icon is used to verify your Arduino Sketch. When you are done with your Arduino Program, you can click this once to verify the code you have written.
2. The Right arrow transfers the code to the Arduino
3. The dotted paper icon will create a new file.
4. The upward arrow icon is used to open an existing Arduino sketch.
5. The downward arrow icon is used to save the current Arduino Sketch.
6. The far right button looks like a zoom glass is a serial monitor, which is very handy for sending and receiving data from the Arduino to the PC for debugging purposes.

How to Install Arduino Libraries

The Arduino IDE comes with several libraries inbuilt during the time of installation that can be used to do many projects, but, sometimes, specialized or new type of sensors requires special library functions that needed to be installed on the Arduino IDE, we will learn how to install extra libraries for our project in this chapter.

There are three different ways available in practical to install an Arduino library 1) Manually installing the files 2) Installing using the library manager 3) Importing to Arduino using the zip file.

The second and third option can be done using Arduino menu options. However, for manual installation, you have to locate the library's files and place them inside the Arduino "libraries" folder.

The manual way

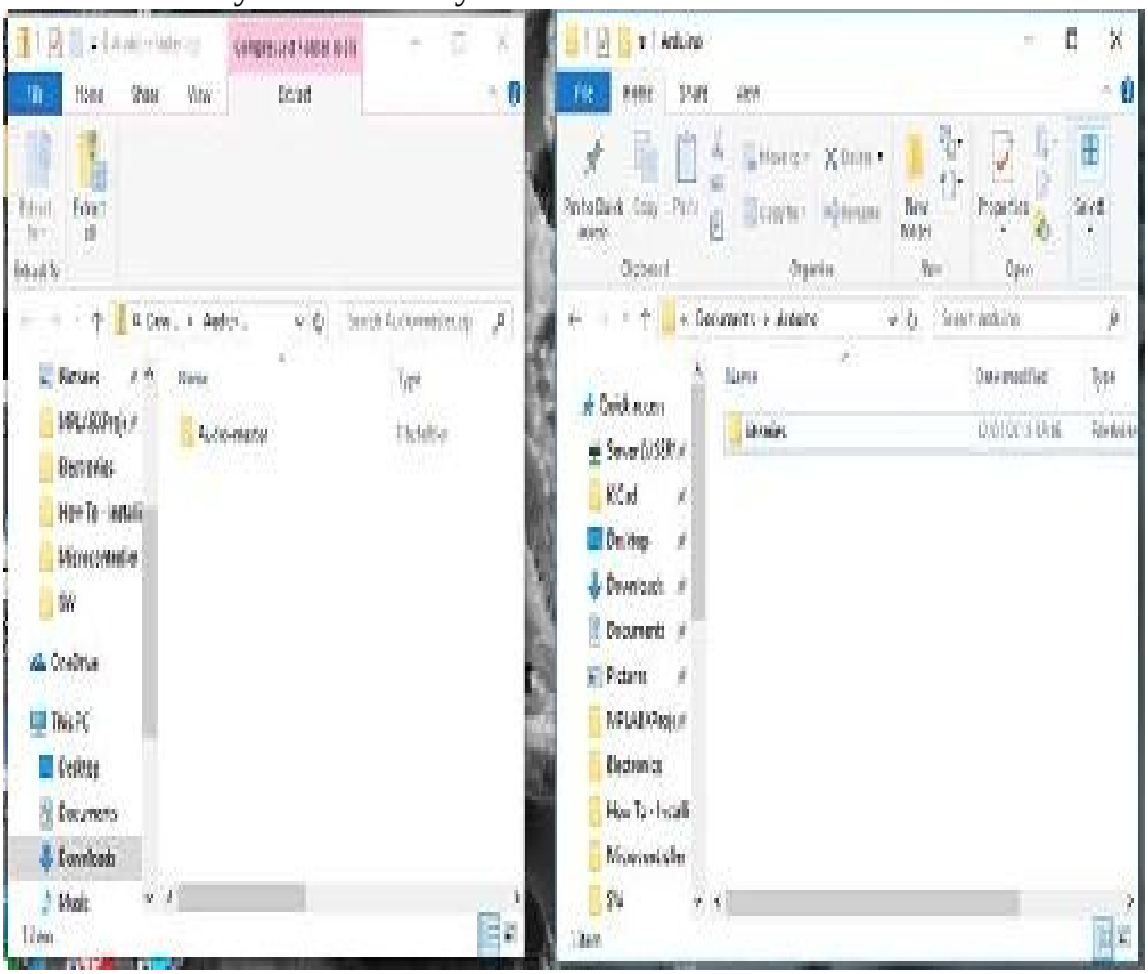
To install this way, we first need to download or find a specific library that we want to install manually. For this example, we will download a sound library for Arduino from Github.

https://github.com/arduino-libraries/Audio

the initial step is to download the required Arduino library as a zip file, click the green tab "clone or download" button on the GitHub page and then click "Download ZIP."

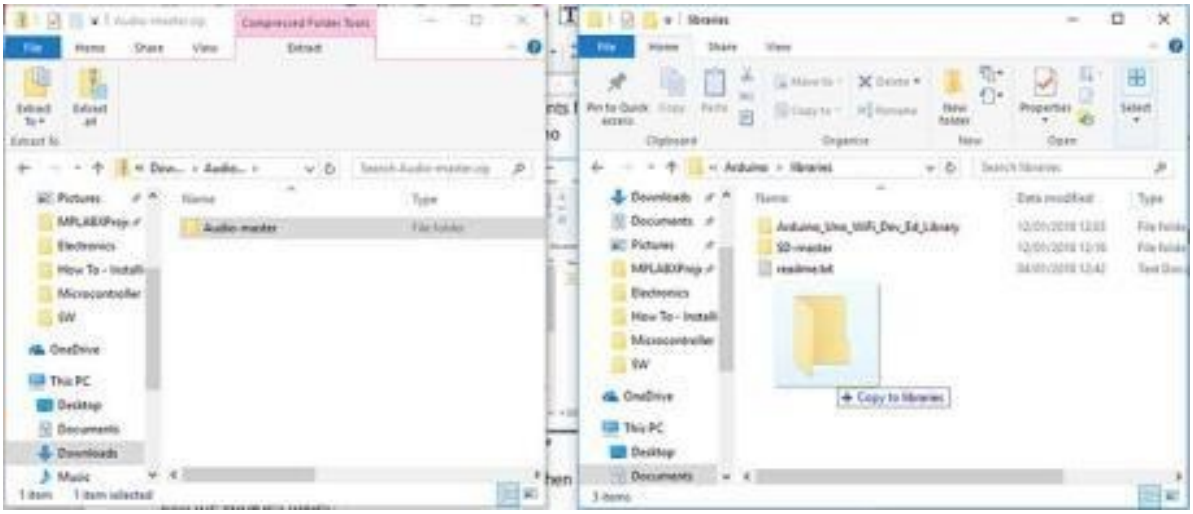
Once the zip file downloaded, open the zip file and unzip the libraries folder from the file, open the extracted file in a separate window and navigate to your Arduino folder. This is typically found in your “Documents,” folder and my case it is located at C:\Users\SimoneLaptop\Documents\Arduino.

Open both the windows, the Arduino library you have downloaded and the Arduino main library folder side by side.

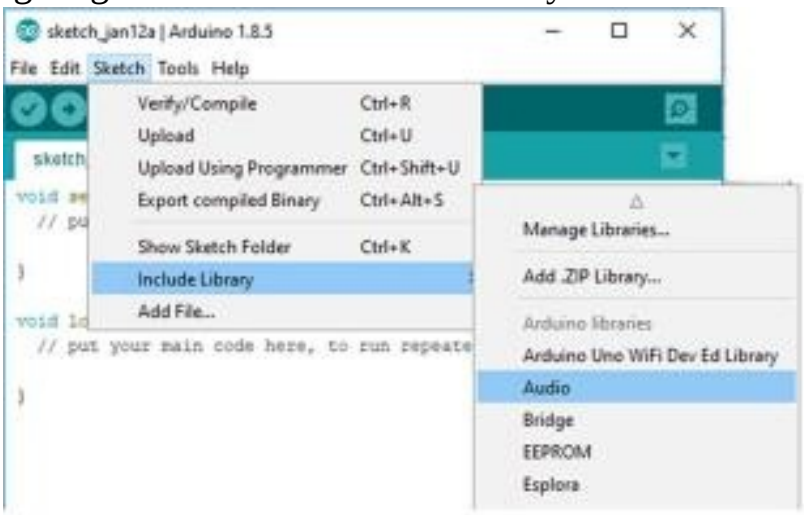


In the Arduino folder that located under the My Documents, open the “libraries” folder and then drag and drop the “sensor library” folder into the Arduino library folder.

Drag and drop library folder into the Arduino library folder

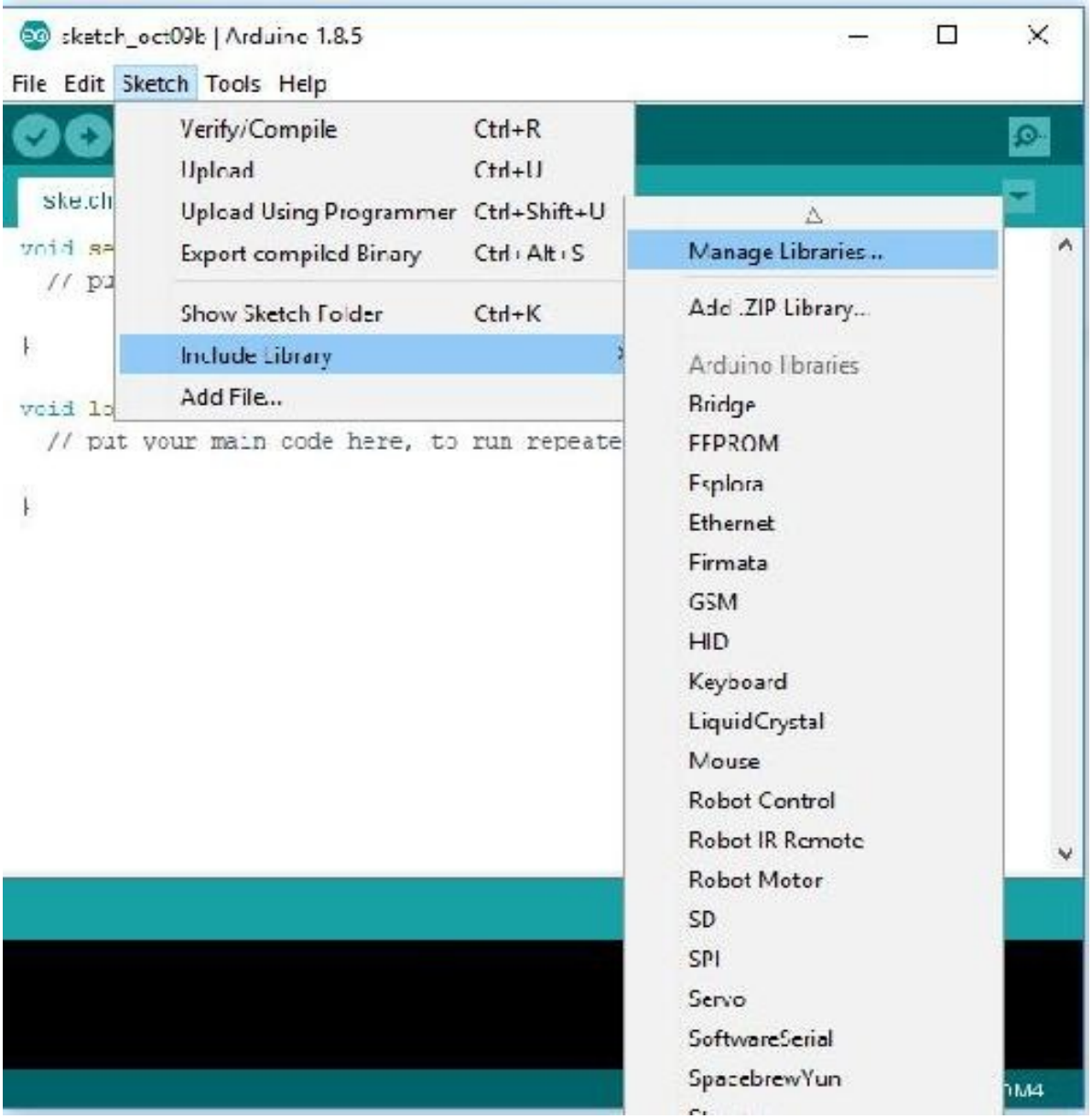


Restart the Arduino IDE and check the library and see if the library has detected by navigating to Sketch => Include Library.

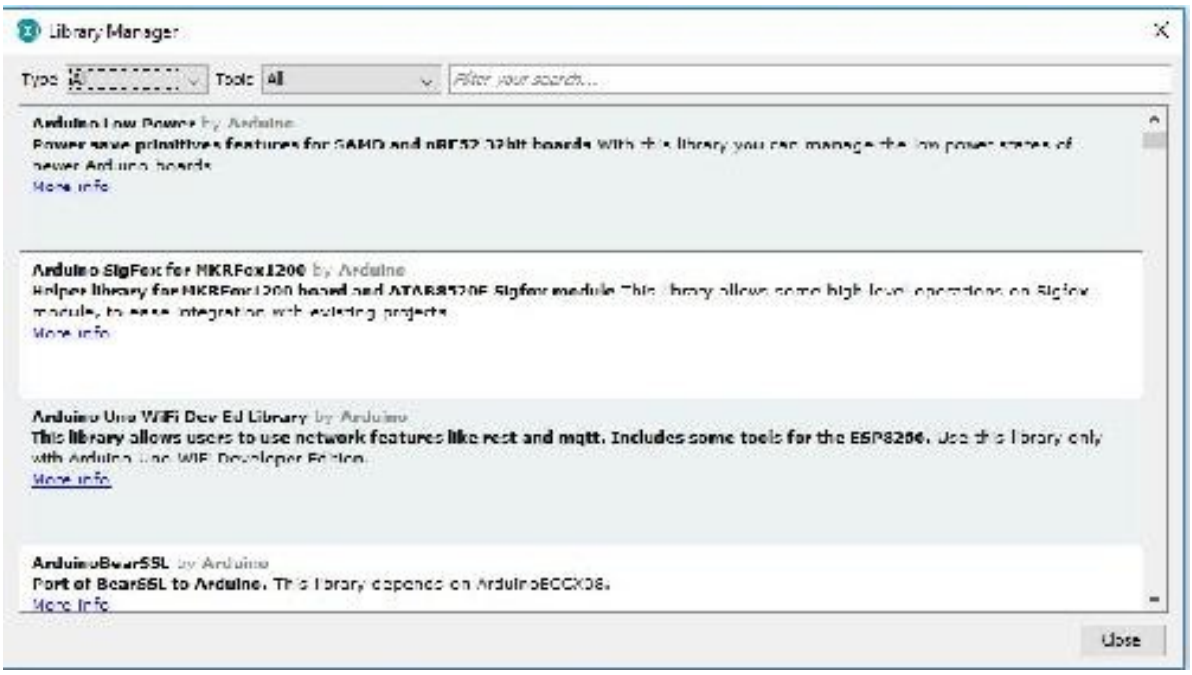


Installing using Library Manager

This method is straightforward, unlike the previous method you don't have to move away from your Arduino IDE to complete the task. The library manager can be loaded by navigating to Sketch => Include Library => Manage Libraries.



Once the library manager loaded, you can see a window similar to the one looks below:



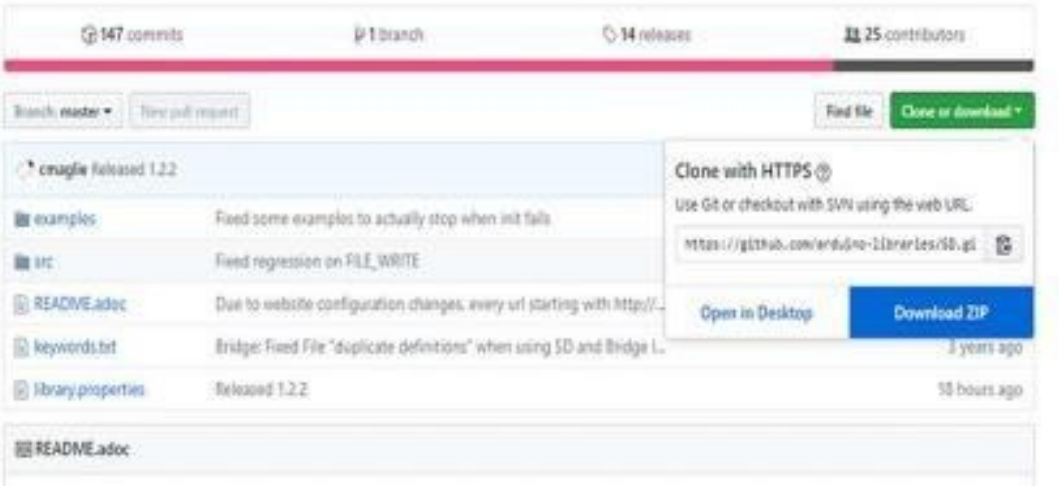
From here, you can search for many commonly used libraries and have the Arduino IDE automatically download and install them.

Importing Zip libraries

Libraries can be imported into Arduino library as a zip file, instead of the manual method, Importing zip libraries can be used if you are not finding a particular library in the Library Manager. To install the library, first, find the library that you want to include to the Arduino library. In this case, we are going to add the same library that we downloaded for adding the manual way.

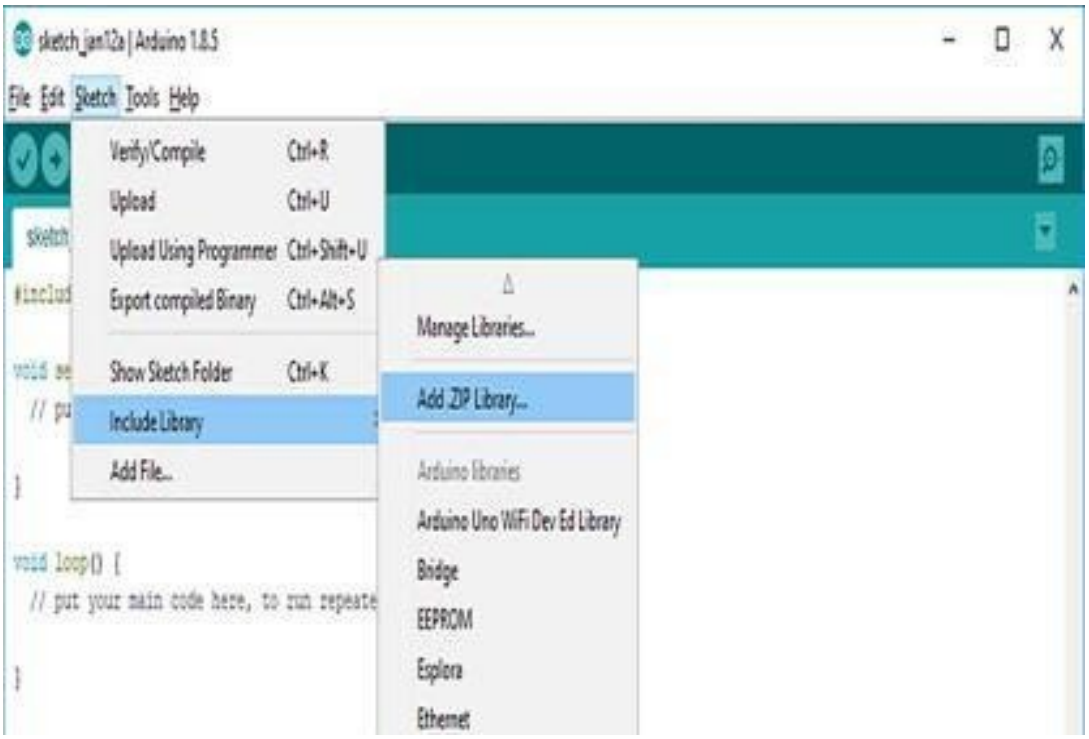
If you don't have any library to add, follow the following steps, download the SD card library from Github from the following link https://github.com/arduino-libraries/SD

first, download the library as a ZIP file, which can be done by clicking the green button "download or clone" button and then clicking "download ZIP."



Once downloaded the zip library, Open the Arduino IDE, and go to the Sketch => Include Library => Add .zip Library.

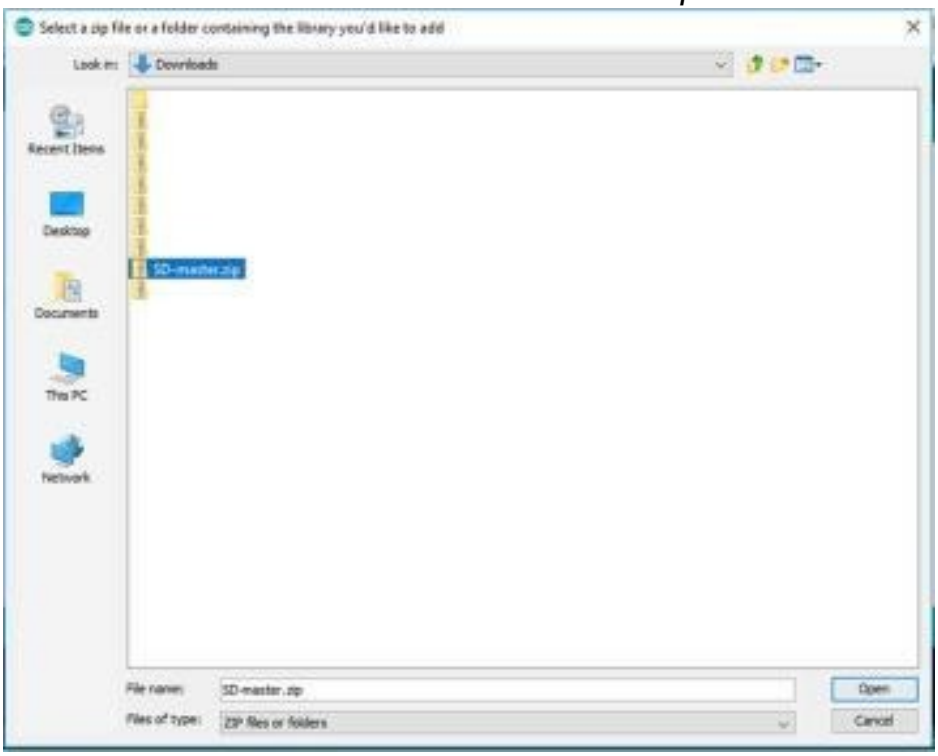
Adding a ZIP library to Arduino Library



When you click, the add ZIP Library… new dialogue windows that open find your downloaded Arduino library ZIP file. For me, this file was downloaded to

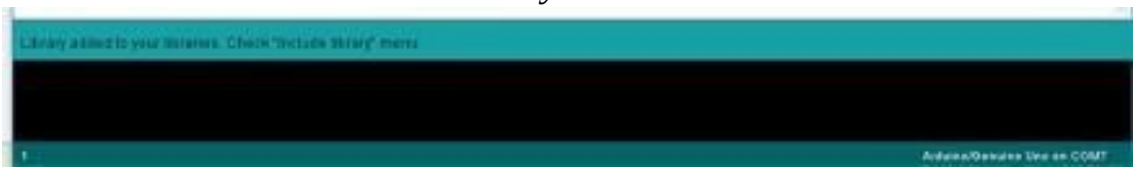
… you guessed it . “Downloads.”

The downloads folder



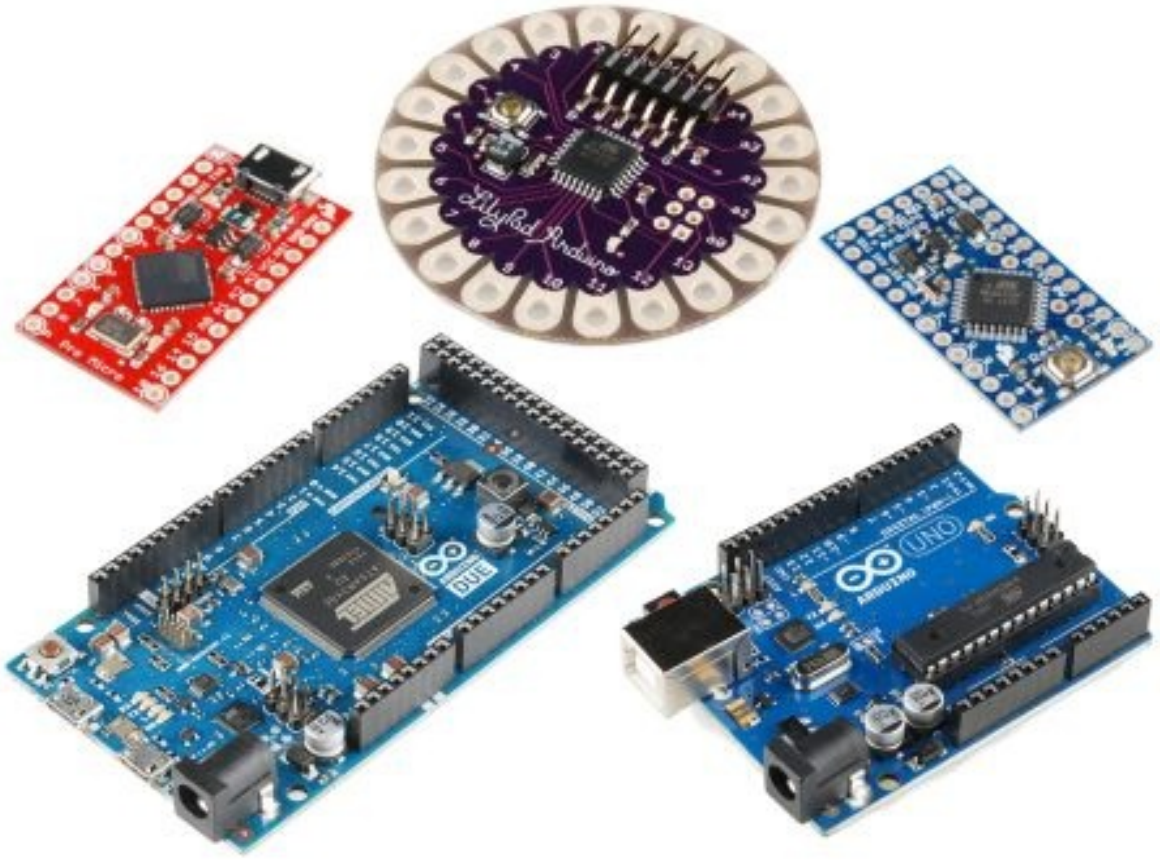
Once you clicked the zip file and opened, the IDE will halt for a second or two. Later, you should receive a “Library Added” note, just above the debug window.

The library has been added!



Lastly, check that the installation was successful. To do this, click on Sketch => Include Library and look for the installed library.

Which Arduino is Best for Your project



Many variants of Arduino

The word Arduino states to complete family of single board microcontroller development boards, which are used for building projects and prototypes. Almost all the Arduino boards carry ATMEL chips as their brain, and each IC has some different capabilities. Some microcontrollers have more input and output pins, some are faster, and some have more peripherals, and some are cheaper.

When selecting a development board which one should you select for your project?

Featured Devices

- Arduino Uno
- Arduino Mega
- Arduino Nano
- Arduino Due

- Arduino Yun
- Arduino MKR Zero
- Arduino Ethernet Rev 3

If you are a newbie to the Arduino world, you might have the confusion on which board to use for your projects. Then there are two devices which are suggested to everyone who is starting to the Arduino world: Arduino Uno and Arduino Nano The Arduino Uno is the most commonly available Arduino development board at the moment, and it has some accessible features, including Jumper pins for secure connectivity between other sensor and the Arduino board, onboard LEDs, USB connector for connecting to computers and laptop for programming and debugging, it has inbuilt Power handling which is very useful.

On the other hand, Arduino Nano has most of the same features like Arduino Uno, (except it has a smaller USB port and has no DC power jack), it is better suited for breadboard based projects. This is an optimal board who has already played with electronic circuits using a breadboard.

When to use the other boards?

While the nano and Arduino Uno are more enough to complete many projects, in some cases you require more functionality from your Arduino, However merely choosing the more powerful Arduino board from the family is not the solution to your problem, so which board should you pick from the ocean of Arduino development boards.

Input/Output

If your project requires many I/O pins, then you should opt for Arduino Due or the Arduino Mega, which both the board has I/O pins up to 54. But not every project requires these many I/Os. You should not select a board based only on the Digital pins; you should also give same important to Analog pins also, Arduino Mega has 16 analog pins whereas the Arduino Due has only 12.

Memory

If you are planning to build complex and advanced projects using Arduino, it may require large program memory as these advance projects will have large programs that use a lot of memory. We have to consider both ROM and RAM, in the world of microcontrollers whenever the ROM size increases, the RAM size automatically increases as well. The Arduino Due has 512KB of ROM and 96KB of RAM, and the Arduino Mega has a 256KB of ROM and 8KB of SRAM, but these two boards alone not rich in memory the Arduino M0, which is based on ARM Cortex-Mo+ has 256KB of ROM and 32KB of RAM.

The Internet of Things Nowadays many projects require internet connectivity for remote monitoring and sending of necessary data to the internet for further analysis. If you are looking for adding WiFi capabilities to your Arduino board, there are few boards which have inbuilt wifi capability which can enable you to send and receive data over the Internet. The Arduino Yun has the WiFi chip in it, and Arduino Ethernet REV 3 has an Ethernet connection and a micro SD card reader, which makes this board ideal for IoT projects.

Programming Structure

Overview

The Arduino software is open-source. The source code for the Java environment is released under the GPL, and the C/C++ microcontroller libraries are under the LGPL.

Sketch − The first new terminology is the Arduino program called "sketch."

Structure

The primary structure of the Arduino Programming language is relatively simple and runs in at least two parts. These two required parts, of functions, enclose blocks of statements.

```
Void setup()
                    {
Program:
                    }

Void loop()
                    {
Program;
                    }
```

Let's discuss the Arduino Structure.

The first line Void setup() is the preparation in the Arduino program.

Loop() is the execution part. These are the two primary functions required for the Arduino program to work.

The setup () function run only once when the Arduino powered. Any variable declaration should be declared at the very beginning of the program.

Arduino development board has many I/O pins based on the model, these I/O pins can be used as Input as well as output, we have to declare at the beginning of the program in the Setup() loop.

If you plan to use Serial communication in the Arduino Program that needs to be declared in the Setup() loop.

Setup( )

The setup() function in the Arduino IDE called only once during the execution of the program or when the Arduino board powered. Setup function used for initializing pin modes and serial communications to the Arduino board.

```
Void Setup() {
pinMode(pin,OUTPUT); }
```

pinMode – is the syntax for setting a pin to input or output at the beginning of the program.

Loop()

The loop function followed by the setup() function. Whatever the code we write inside the loop will continuously run until disturbed by an external interrupt.

```
Void loop() {
digitalWrite(pin,HIGH); delay(1000); digitalWrite(pin,LOW); delay(1000); }
```

Functions

A function is a block of code with a name and a block with instructions that are performed when the function is called.

Custom functions can be written to execute repetitive tasks and reduce clutter in a program. Functions are indicated by first declaring the function type. This is the type of value that must be returned by the function, such as "int" for a function with an integer. If no value is to be returned, the function type is invalid. After type, declare the name given to the function, and in brackets, any parameters passed to the function.

```
type functionname (parameters) {
Program;

                                        }
```

The following integer type function delayVal () is used to set a delay value in a program by reading the value of the potentiometer. First, it declares a local variable v, sets v to the value of the potentiometer that gives a number between 0-1023, then divides that value by 4 for a final value between 0-255 and finally returns that value back to the main program.

```
Int delayVal()

                                        {

Int v; // v-temporary variable V = analogRead(pot); // read the pot value v/= 4;
//converts0-1023 to 0-255
return v;

                                        }
```

{} Curly Braces

Curly braces (also called just "curly braces" or braces) define the start and end of function blocks and instruction blocks, such as the void loop () function and the for and IF instructions.

```
Type function()

                {

Program;}
```

An open brace {must always be followed by a closing accolade}. This is often called the alignment of the bracket. Unbalanced braces can often lead to cryptic, impenetrable compiler errors that are sometimes difficult to detect in an extensive program.

The Arduino environment contains a useful function to check the balance of braces. Merely select a brace or even click on the insertion point immediately after a brace, and the corresponding logic is highlighted.

; Semicolon

A semicolon must be used to end a statement and individual elements of the program.

A semicolon is also used to separate elements in a “for loop.”

```
Int x =13;
```

Note: If you forget to end a line in a semicolon, this results in a compiler error. The error text may be distinct and refer to a missing semicolon, or maybe not. If an impenetrable or seemingly illogical compilation error occurs, one of the first things to check is a missing semicolon, near the line where the compiler complained.

/*…….*/ block Comments

block comments or multi-line comments are text areas that are ignored by the program and used for extensive text descriptions of code or comments that help others understand parts of the program. They start with / *and end with* / and can span multiple lines

/* here we can type anything And the compiler will not consider these lines

Bypass the code

Don’t forget to close the comment.

*/

comments are ignored by the program and do not occupy memory space, they must be used generously and can also be used to "omit" code blocks for debugging purposes.

Note: although it is possible to place single line comments within a block comment, it is not allowed to add a second block comment.

// LINE comments

Single line comments start with // and end with the next line of code. Just like block notes, they are ignored by the program and do not take up memory space.

// whatever comes after this line will be ignored Single line comments are often used after a valid statement to provide more information about what the statement reaches or to give a future reminder.

Variable

A variable is a way to give a numerical value a name and save it for later use by the program. As their name giver suggests, variables are numbers that can be changed continuously as opposed to constants whose value never changes. A variable must be declared and optionally assigned to the value to be stored for later use. The code shown below declares a variable called intVariable and then assigns the value obtained on analog input pin 2:

```
Int intVariable = 0;
intVariable = analogRead(2);
```

'IntVariable' is the variable itself. The first line states that it will contain an integer. The 2nd line in the program sets the variable to the value on analogpin 2. This makes the value of pin 2 available elsewhere in the code.

Once a variable has been allocated or reassigned, you can check its value to see if it meets certain conditions, or you can use its value directly. As an example for illustrating three useful operations with variables, the following code tests whether the inputVariable is less than 100 if true assigns the value 100 to inputVariable and then sets a delay based on inputVariable that is now at least 100:

```
If (intVariable < 100)  // tests variable if less than 100

{

intVariable = 100; // if true assigns value of 100

}

Delay (intVariable); // uses variables as delay .
```

Note: Variables must be given descriptive names to make the code more readable. Variables name such as a tilt sensor or pushButton help the programmer and anyone who reads the code to realize what the variable represents. Variable names such as var or value, on the other hand, do little to make the code clear and are only used here as an example. A variable can be called any word that is not yet one of the keywords in the Arduino language.

Variable Declaration

All variables need to be declared before they can be used. Declaring a variable means defining the value type as in int, long, float, etc., setting a specified name and optionally assigning it as an initial value. This only requires to be done once in a program, but the value can be altered at any time using arithmetic and different assignments.

The following example explains that inputVariable is an int or integer type and that the initial value is zero. This is called a simple assignment.

Int inputVariable = 0; A variable can be declared at a number of locations in the program and where this identification takes place determines which parts of the program the variable can use.

Variable Scope

A variable can be stated at the beginning of the program before void setup (), locally within functions and sometimes within an instruction block, such as for loops, is declared. Where the variable is declared, the variable range or power of certain parts of a program determines the use of the variable.

A global variable is one that can be seen and used by any function and statement in a program. This variable is declared at the beginning of the program before the setup () function.

A local variable is one that is defined within a function or as part of a for loop. It is only visible and can only be used within the function in which it has been declared. It is, therefore, possible to have two or more variables with the same name in different parts of the same program that contain different values. By ensuring that only one function has access to the variables, the program is simplified and the chance of programming errors decreases.

The following example explains how to declare a number of different types of variables and show the visibility of each variable.

```
int value; // value is visible void setup()

                                        {

// setup is needed for this program }

Void loop()

                                        {

For (int i=0; i<20;) //

                                        {

i++;

                                        }
```

Float f; // ‘f’ is only visible } //inside loop

Data types in C denotes to a widespread system used for declaring variables or functions of different types. The type of a variable decides how much space it occupies in the storage and how the bit pattern stored is interpreted.

Data Types

Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one: Following are the examples of some widespread data types used in Arduino programming: **Byte**

Byte stores a –bit numerical value without decimal points. They have a range of 0-255.

byte someVariable = 180; //

int

Integers are the primary datatype for storage of numbers without decimal points and store a 16-bit value with a range of 32,767 to -32,768.

Int someVariable = 1500; // declares ‘somevariable’ as an integer type Note: Integer variables will roll over if forced past their maximum or minimum values by an assignment or comparison. For example, if x = 2767 and a following statement adds 1 to x, x = x+1 or x++, x will then roll over and equal to -32768

Long

Extended size datatype for long integers, short of decimal points, stored in a 32-bit value with a range of 2,147,483,647 to -2,147,483,648.

```
Long Variable = 80000; //.
```

Float

A datatype for floating-point numbers, or numbers that have a decimal point. Floating-point numbers have greater resolution than integers and are stored as a 32-bit value with a range of 3.4028235E+38 to -3.4028235E+38

Float someVariable =3.14; //declares ‘someVariable’ as a floating-point type Note: Floating-point numbers are not exact, and may yield strange results when compared. Floating point math is also much slower than integer math in performing calculations, so should be avoided if possible.

Arrays

An array is a collection of values that are accessed with an index number. Any value in the array may be called upon by calling the name of the array and the index number of the value. Arrays are zero indexed, with the first value in the array beginning at index number 0. An array needs to be declared and optionally assigned values before they can be used.

```
int myArray[] = {value0, value1, value2...}
int myArray[5]; // declares integer array w/ 6 positions myArray[3] = 10;  // assigns the 4th index the value 10
```

char

A data type that takes up 1 byte of memory that stores a character value. Character literals are written in single quotes, like this: 'A' (for multiple characters - strings - use double quotes: "ABC").

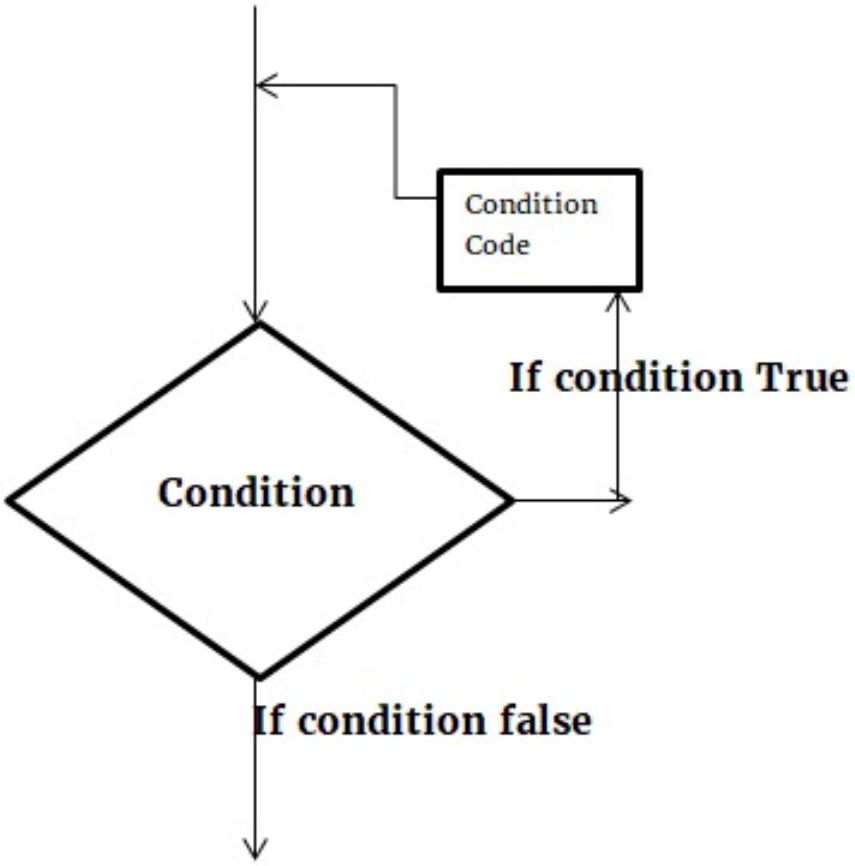
```
char myChar = 'A';
char myChar = 65;      // both are equivalent
```

Note: The char datatype is a signed type, meaning that it encodes numbers from -128 to 127. For an unsigned, one-byte (8 bit) data type, use the byte data type.

Loops

Overview

Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages



Loops and Description

for Loop

A for loop executes statements a predetermined number of times. The control expression for the loop is initialized, tested and manipulated entirely within the for loop parentheses.

```
// syntax for for loop

for (initialization; condition; expression)  {
   dosomething;

                                                }
```

while Loop

while loops will loop continuously, and infinitely, until the expression inside the parenthesis, becomes false. Something must change the tested variable, or the while loop will never exit.

```
// syntax for while loop

while ( somevariable ?? value)          /

                                        {

   dosomething;

                                        }
```

do while Loop

The do…while loop is similar to the while loop. In the while loop, the loop-continuation condition is tested at the beginning of the loop before performed the body of the loop.

```
The// syntax for doing while loop do

                                        {

  dosomething;
} while ( some variable ?? value)
```

Arithmetic

Arithmetic operators include addition, subtraction, multiplication, and division. They return the sum, difference, product, or quotient (respectively) of two operands.

```
z = z + 3;
y = y - 7;
u = u * 6;
r = r / 5;
```

The operation is conducted using the data type of the operands, so, for example, 9 / 4 results in 2 instead of 2.25 since 9 and 4 are ints and are incapable of using decimal points. This also means that the operation can overflow if the result is larger than what can be stored in the data type.

If the operands are of different types, the larger type is used for the calculation. For example, if one of the numbers (operands) are of the type float and the other of type integer, floating point math will be used for the calculation. Choose variable sizes that are large enough to hold the most significant results from your calculations. Know at what point your variable will roll over and also what happens in the other direction, *e.g.* (0 - 1) OR (0 - - 32768). For math that requires fractions, use float variables, but be aware of their drawbacks: large size and slow computation speeds.

Comparison operators

Comparisons of one variable or constant against another are often used in if a statement to test if a specified condition is true. Different types of condition used are:

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

Logical operators

Logical operators are usually a way to compare two expressions and return a TRUE or FALSE depending on the operator. There are three logical operators, AND, OR, and NOT, that are often used in if statements:

Logical AND:

if (x > 0 && x < 5) // true only if both expressions are true Logical OR:

if (x > 0 || y > 0) // true if either expression is true Logical NOT:

if (!x > 0) // true only if expression is false

Constants

The Arduino language has a few predefined values, which are called constants. They are used to make the programs more comfortable to read. Constants are classified in groups.

True/False These are Boolean constants that define logic levels. FALSE is easily defined as 0 (zero) while TRUE is often defined as 1, but can also be anything else except zero. So in a Boolean sense, -1, 2, and -200 are all also defined as TRUE.

```
if (b == TRUE);
{ doSomething; }
```

Control Statements

Overview

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program. It should be along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Control Statement & Description If statement

It takes an expression in parenthesis and a statement or block of statements. If the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

```
// syntax for if statement if (condition) {
do something; }
```

If …else statement

An if statement can be followed by an optional else statement, which executes when the expression is false.

```
// syntax for If …else statement if (condition) {
do something; }  else

                                {


do something; }
```

switch case statement Similar to the if statements, switch...case controls the flow of programs by allowing the programmers to specify different codes that should be executed in various conditions // syntax for switch case statementswitch (variable) { case label: { // statements break; }

case labe2: {

// statements

break; }

}

Digital Input -Output Functions

The digital inputs and outputs (digital I/O) on the Arduino are what allow you to connect the Arduino sensors, actuators, and other ICs. Learning how to use them will allow you to use the Arduino to do some beneficial things, such as reading switch inputs, lighting indicators, and controlling relay outputs. Etc **Digital Signal** Digital signals may not take any values within the range. Digital signals have two distinct values HIGH or 1 and LOW or 0. You use digital signals in situations where the input or output will have one of those two values.

Function

The Arduino functions associated with digital signals that we will be using in this tutorial are

1. pinMode (pin_number, mode)
2. digitalWrite(pin_number,value)
3. digitalRead(pin_number)

pinMode (pin_number, mode)

Because the Arduino digital I/O pins can be used for either input or output, you should first configure the pins you intend to use for digital I/O with this function. the pin is the number of the pin you wish to configure. mode must be one of three values: INPUT, OUTPUT, our INPUT_PULLUP. When the mode is set to INPUT_PULLUP, a 20 k ohm pull-up resistor is internally connected to the pin to force the input HIGH if there is nothing connected to the pin.

```
pinMode(pin,OUTPUT);  // set pin as  OUTPUT
pinMode(pin,INPUT);  // set pin as  INPUT
```

Note: INPUT/OUTPUT Constants used with pinMode() function to define the model of a digital pin as INPUT or OUTPUT.

digitalWrite(pin_number,value)

This function writes a digital value to a pin. pin specifies which Arduino pin the digital value will be written to, and value is the digital value to which the pin is set. The value must be either HIGH or LOW.

```
digitalWrite(pin,HIGH);   //set pin as HIGH
```

digitalRead(pin_number)

```
int reads = digitalRead(pin); //read the digital value on pin This function reads a digital value from a pin. the pin is the number of the digital I/O pin you want to read. This function returns one of two values: HIGH or LOW.
```

Note: HIGH/LOW These constants defined pin level as HIGH or LOW and used when reading or writing to digital pins.HIGH is defined as logic level 1, ON or 5 volts while LOW is logic level 0,OFF or 0 volts **Analog Input-Output Functions**

An analog signal is one that can take on any number of values, unlike a digital signal which has only two values: HIGH and LOW. To measure the value of analog signals, the Arduino has a built-in analog-to-digital converter (ADC). The ADC turns the analog voltage into a digital value. The function that you use to obtain the value of an analog signal is analogRead(pin). This function converts the value of the voltage on an analog input pin and returns a digital value from 0 to 1023, relative to the reference value. The reference is 5V on most Arduino **Function**

The Arduino functions associated with Analog signals that we will be using in this tutorial are

1. analogWrite(pin_number,value)
2. anlogRead(pin_number)

analogWrite(pin_number,value)

This function writes a analog value to a pin. Here in Arduino there are 6 Analog pins so use the same to write Value. Value must be range of 0 to 1023

```
analogWrite(Analogpin,255);   //set Analogpin as  255
analogRead(pin_number)
int reads = analogRead(Analogpin); //read the analog value on pin
```

This function reads a analog value from a pin. Analogpin is the number of the analog I/O pin you want to read. This function returns values from a range of 0 to 1023.

Additonal Notes

The Arduino does not have a built-in digital-to-analog converter (DAC), but it can pulse-width modulate (PWM) a digital signal to achieve some of the functions of an analog output. The function used to output a PWM signal is analogWrite(pin, value). pin is the pin number used for the PWM output. Value is a number proportional to the duty cycle of the signal. When value = 0, the signal is always off. When value = 255, the signal is always on. On most Arduino boards, the PWM function is available on pins 3, 5, 6, 9, 10, and 11. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards.

To map an analog input value, which ranges from 0 to 1023 to a PWM output signal, which ranges from 0 - 255, you can use the map(value, fromLow, fromHigh, toLow, toHigh) function. This function has five parameters, one is the variable in which the analog value is stored, while the others are 0, 1023, 0 and 255 respectively.

delay(ms)

Pauses a program for the amount of time as specified in milliseconds, where 1000 equals 1 second.

```
delay(1000); // waits for one second
```

millis() Returns the number of milliseconds since the Arduino board began running the current program as an unsigned long value.

```
value = millis();  // sets 'value' equal to millis()
```

Note: This number will overflow (reset back to zero), after approximately 9 hours.

map()

This function is used to change covert a value in a particular range to other range. for example, if we read a value from analog port A0 is 25, its between a range of 0-1023 in order to cover the value of analog port A0 to a range of 0-255 we use the map function

```
map(value, fromLow, fromHigh, toLow, toHigh) // example
analogValue = map(analogValue, 0, 1023, 0, 255);
```

random()

The random() function returns a semi-random number up to the parameters

specified. If no parameters are specified, it will return a value in the signed long data type, with a range of -2,147,483,648 to 2,147,483,647. Its syntax follows:

```
random(min, max)    // "min" minimum possible value expected from the random() //  "max" maximum value expected from the random() function
```

Getting Started with Arduino

So, you bought yourself an Arduino; you understood you felt where to start with it. Do not fear, for help is at hand! In this chapter, we will look at how to get in progress with Arduino development boards. This chapter covers software installation once more if you have not followed the software installing from the previous chapter, as well as connecting and setting the Arduino IDE.

You Will Need

- Arduino Uno or any other Arduino
- USB Cable and connecting wires
- Windows or any other operating system

Step 1: Download Arduino IDE and install it.

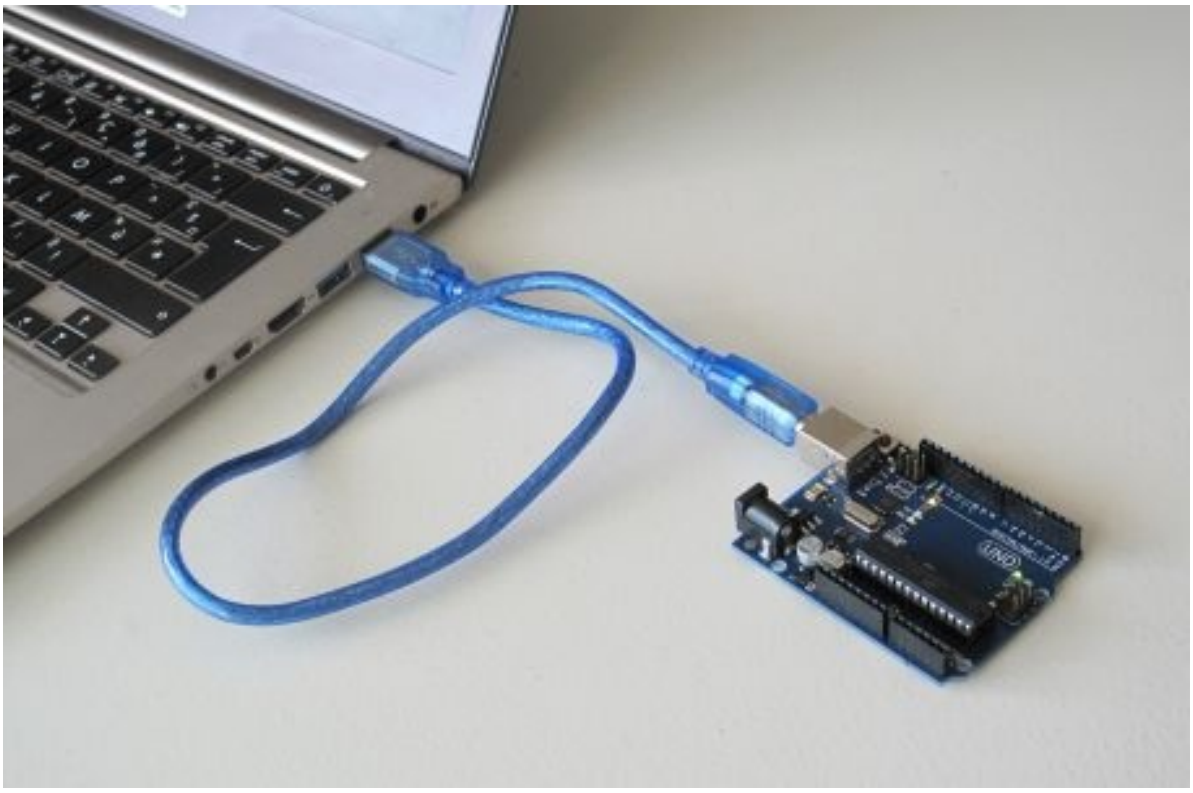
The Arduino IDE can download the IDE from the official Arduino website. (https://www.arduino.cc/en/Main/Software) Since the Arduino development board uses a USB to serial converter, the Arduino board is well-matched with most PCs that have a USB port. Luckily, the Arduino inventors have released multiple versions of the IDE for different operating systems, including, Linux, Mac, and Windows, In this tutorial, we will use Window 10, to confirm that you download the correct type of IDE if you do not have Windows 10.

Once downloaded the Arduino IDE, install the Arduino IDE and confirm that you enable most (if not all) of the options, WITH the drivers during the installation process.

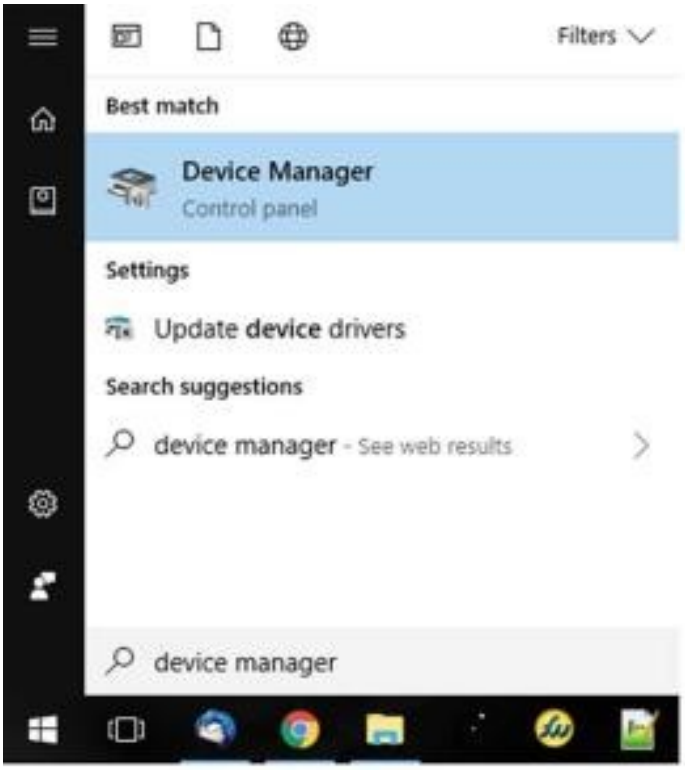
Step 2: Check for the Arduino COM Port Number (You can find it in your Arduino COM ports)

Next, you'll require to connect the Arduino Uno or any other variant of Arduino

board to the PC or to your laptop. This is done via a USB connection. Thanks to the beautiful world of USB, we do not need to provide power supply to the Arduino, as the USB provides 5V up to 2A. When the Arduino is connected, the operating system should identify the board as a generic COM port (for example, my Arduino Uno uses a CH340G,(clone version) which is an RS-232 serial to USB converter). Once it's accepted, we will need to find out what port number it has been allocated. The easiest way to do this is to type "device manager" into Windows Search and select Device Manager when it shows.

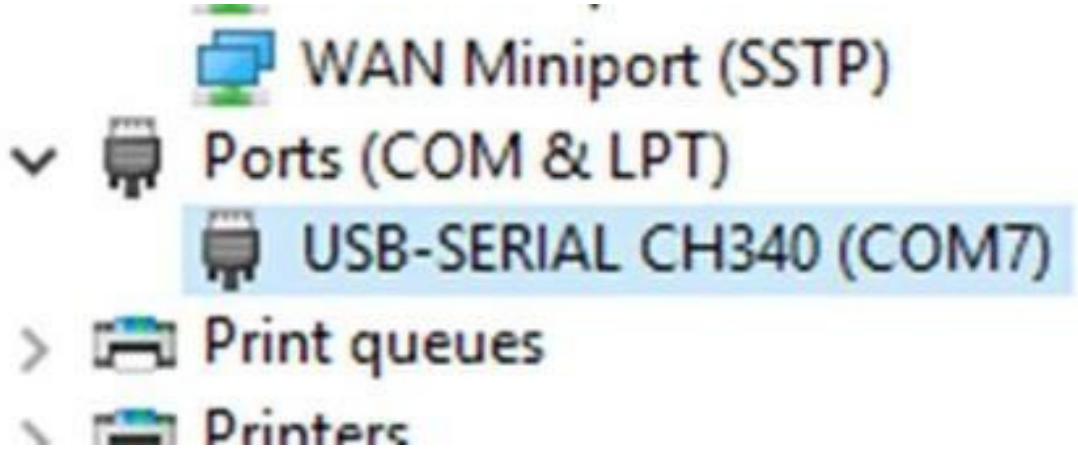


How to find Device manager in windows 10



In the Device Manager Window, check for a device under “Ports (COM & LPT) and chances are the Arduino will be the only device on the list. In my Device Manager, the Arduino shows up as COM7 (I know this because CH340 is in the device name).

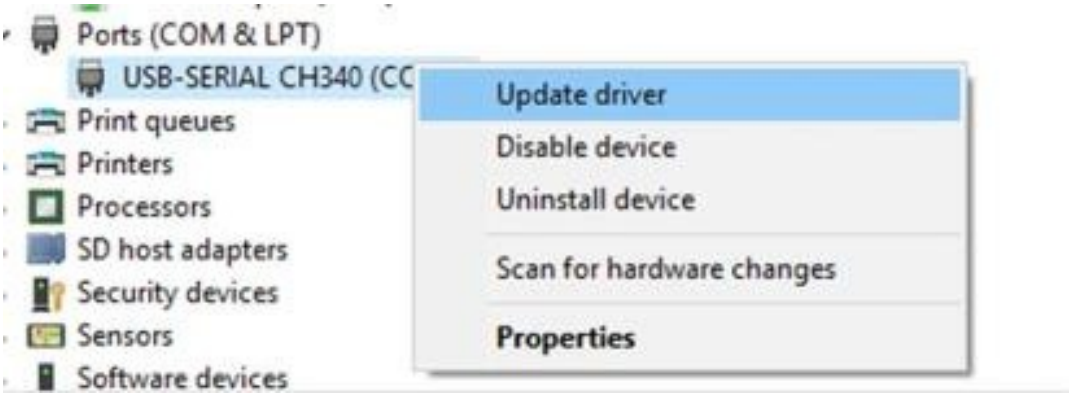
The Arduino in my case is a CH340, and it shows on COM7 (port 7).



Be warned; the Arduino won’t constantly be recognized automatically. If your Arduino is not identified by the windows. Then uninstall the driver, remove the

Arduino, reinsert the Arduino, find the unrecognized device, right click “Update driver,” and then click “Search automatically.” This should fix 99 out of 100 problems.

If the Arduino is not recognized, update the driver.



In the window that appears, click “Search automatically.”

Windows can be a real discomfort sometimes with COM ports, as it can magically change their numbers between connections. In other words, one day, your Arduino may be on port 7 (as shown here), but then on other days, Windows may shift it to a different port number.

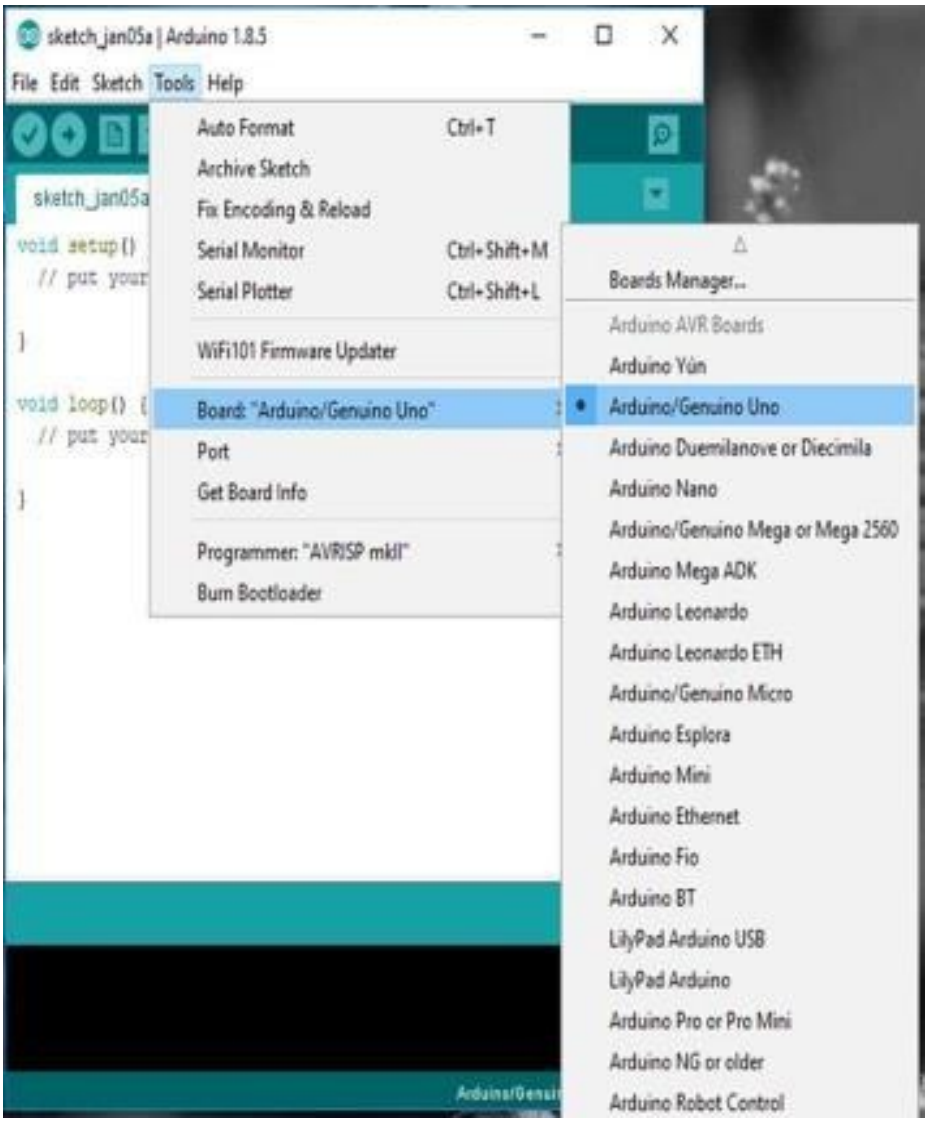
As I understand it, this happens when you connect other COM ports to your system (which I frequently do).

So, if you can't find your Arduino on the port that you usually use, just go to your Device Manager and check what port it's actually on and, if necessary, update your driver.

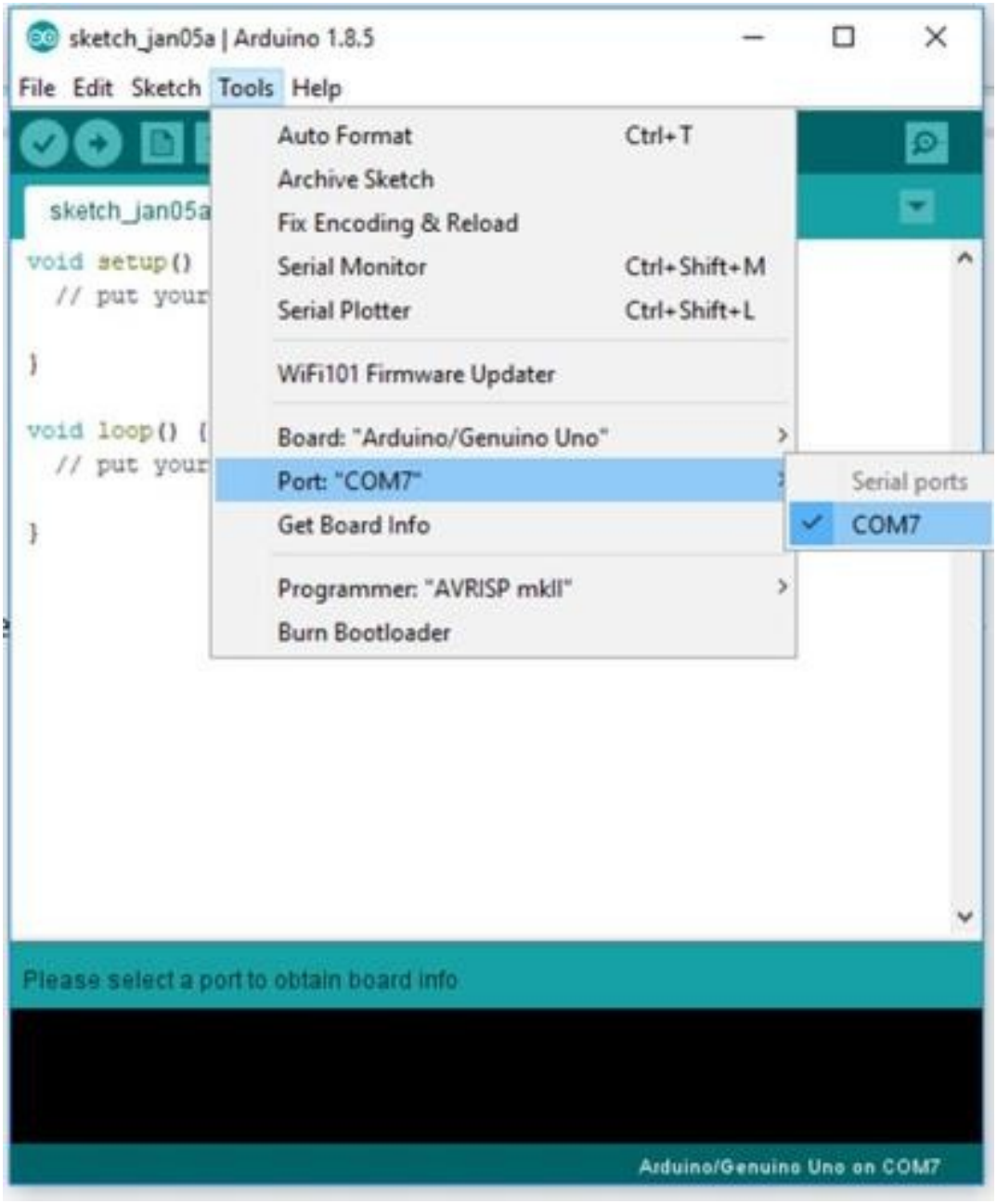
Step 3: Configure the IDE

Now that we have determined the COM port that the Arduino is on, it's time to load the Arduino IDE and configure it to use the same device and port. Start by loading the IDE. Once it's loaded, navigate to Tools > Board > Arduino Uno. However, if you are using a different board (i.e., not the Arduino Uno), you must select the proper board!

Tell the IDE which board you are using.



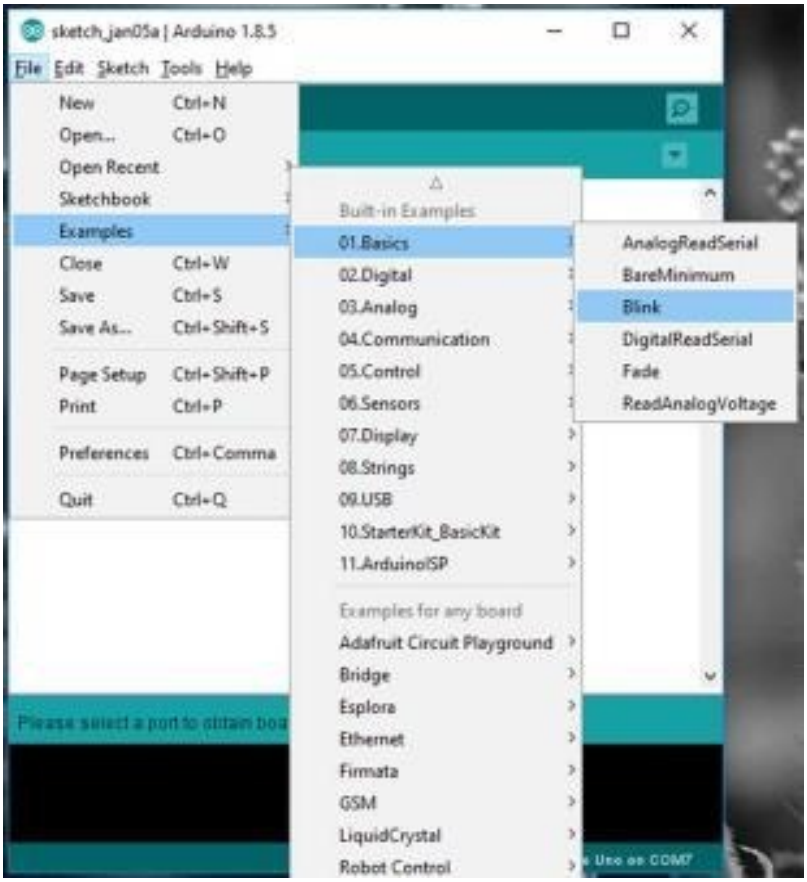
Next, you must tell the IDE which COM port the Arduino is on. To do this, navigate to Tools > Port > COM7. Obviously, if your Arduino is on a different port, select that port instead.



Step 4: Loading a Basic Example

For the sake of simplicity, we will load an example project that the Arduino IDE comes with. This example will make the onboard LED blink for a second continuously. To load this example, click File > Examples > 01.Basics > Blink.

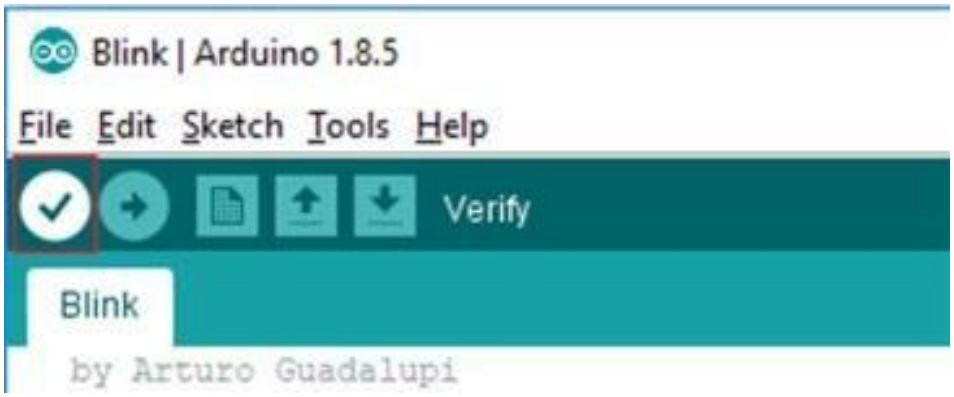
Load the blink example.



With the example loaded, it's time to verify and upload the code. The verify stage checks the code for errors, then compiles the ready-for-uploading code to the Arduino. The upload stage actually takes the binary data, which was created from the code, and uploads it to the Arduino via the serial port.

To verify and compile the code, press the check mark button in the upper left window.

The "Verify" button will compile the Arduino code.



If the compilation stage was successful, you should see the following message in the output window at the bottom of the IDE. You might also see a similar message—just it's one that does not have words like "ERROR" and "WARNING."

This is a successful compilation.

With the code compiled, you must now upload it the Arduino Uno. To do this, click the arrow next to the check mark.

The "Upload" button will program the Arduino with your code.

Conclusion

The Arduino is a powerful prototyping tool for many reasons, including its lack of a dedicated programmer, its wide range of available libraries, and the simplicity of its IDE. While we only got a light to blink in this project, you can expect much more in the future. Try your hand at interfacing with displays, taking measurements, talking over the internet, and possibly even working with AI!

Now try few more examples from next chapters before you begin with real projects.

Basic Projects with Arduino

Now we have little bit idea about Arduino, how to program using C *etc*. Here we are talking about practical sections of basic programming with examples. Does it in a continuous way it will help you learn basic programming of Arduino.

Connect your Arduino to the computer with the USB cable. You do not need the battery for now. The green PWR LED will light. If there was already a program burned into the Arduino, it would run.

Warning: Do not put your board down on a conductive surface; you will short out the pins on the back!

Serial Monitor

The serial monitor is the 'tether' between the computer and your Arduino - it lets you send and receive text messages, handy for debugging and also controlling the Arduino from a keyboard!

Serial.begin(rate)

Opens serial port and set baud rate for serial data transmission. Typically baud rate for communication with the computer is 96000 although other speeds are supported.

Note:when using serial communication digital pins 0(RX) and 1(TX) cannot be used at the same time

Serial.println(data)

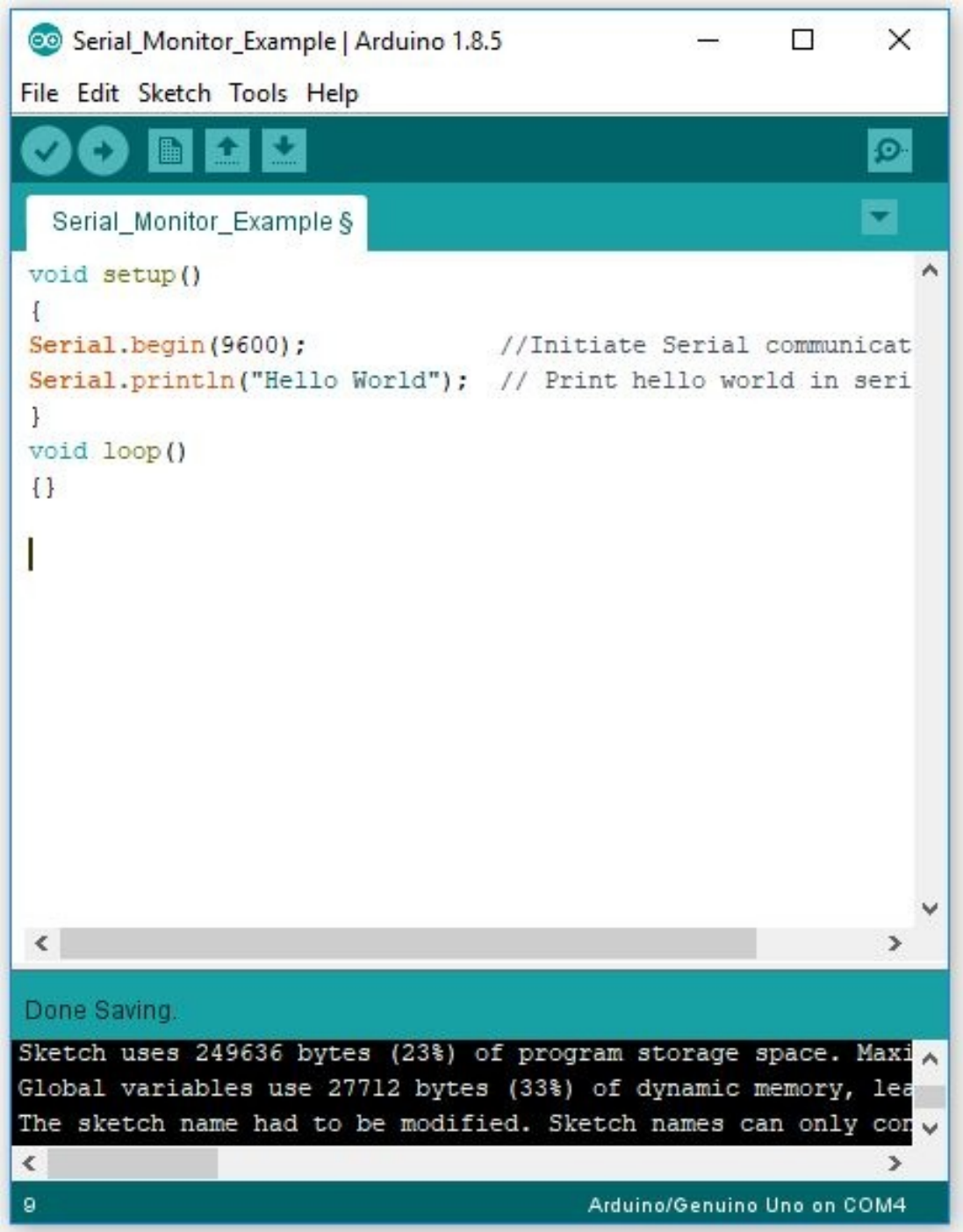
Print data to the serial port followed by an automatic carriage return and line feed .This command takes the same form as Serial.print() but is easier for reading data on the Serial monitor.

Serial.println(Anything);

Arduino Code

Open the Arduino IDE and Copy paste the Arduino Code below (You can find all the code is source file you can find at the end of this book)

```
void setup()

                                        {

Serial.begin(9600);            //Initiate Serial communication

Serial.println("Hello World");  // Print hello world in serial monitor

                                        }

void loop()

                                        {}
```



Open the Serial Monitor Icon (Zoom Glass on the right side of the Arduino IDE) you can find the result for the program.

LED-Digital Write

Overview

LEDs make great indicator lights. They use very little electricity, and they pretty much last forever. The most common of all LEDs a 5mm red LED. 5mm refers to the diameter of the LED and as well as 5mm, other standard sizes are 3mm and the large fun 10mm LEDs. You cannot directly connect an LED to a battery or voltage source. Firstly, because the LED has a positive and a negative lead and will not light if they are the wrong way around and secondly, an LED must be used with a resistor to limit the amount of current flowing through the LED - otherwise the LED could burn out

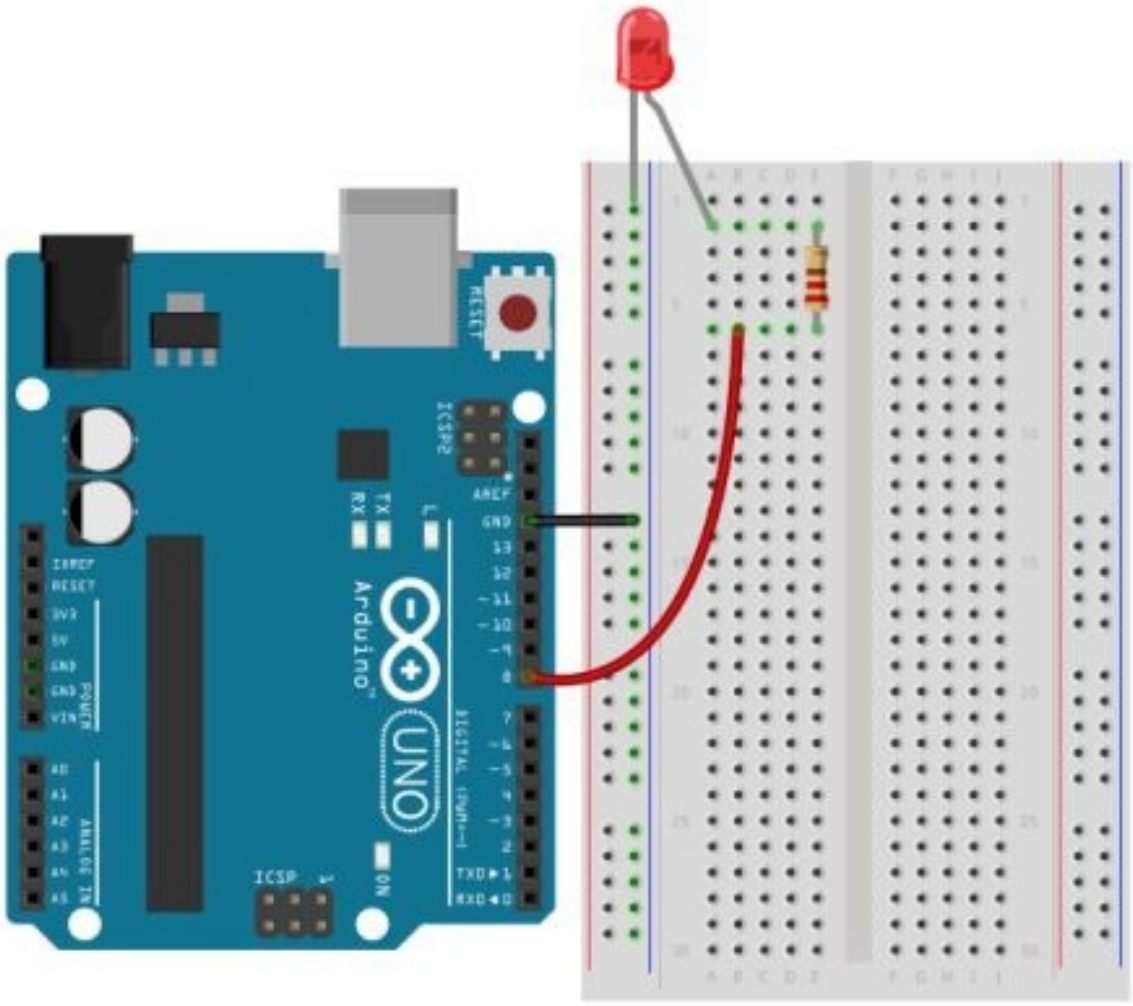
If you do not use a resistor with an LED, then it may well be destroyed almost immediately, as too much current will flow through the LED, heating it and destroying the 'junction' where the light is produced.

There are two ways to tell which is the positive lead of the LED and which the negative.

- Firstly, the positive lead is longer.
- Secondly, where the negative lead enters the body of the LED, there is a flat edge to the case of the LED.

How to Connect with Arduino

Bread Board Layout



Components Required

1) Arduino with cable 2) Breadboard 3) LED

4) 220 Ω or 1K Ω

Arduino Code-Blink LED

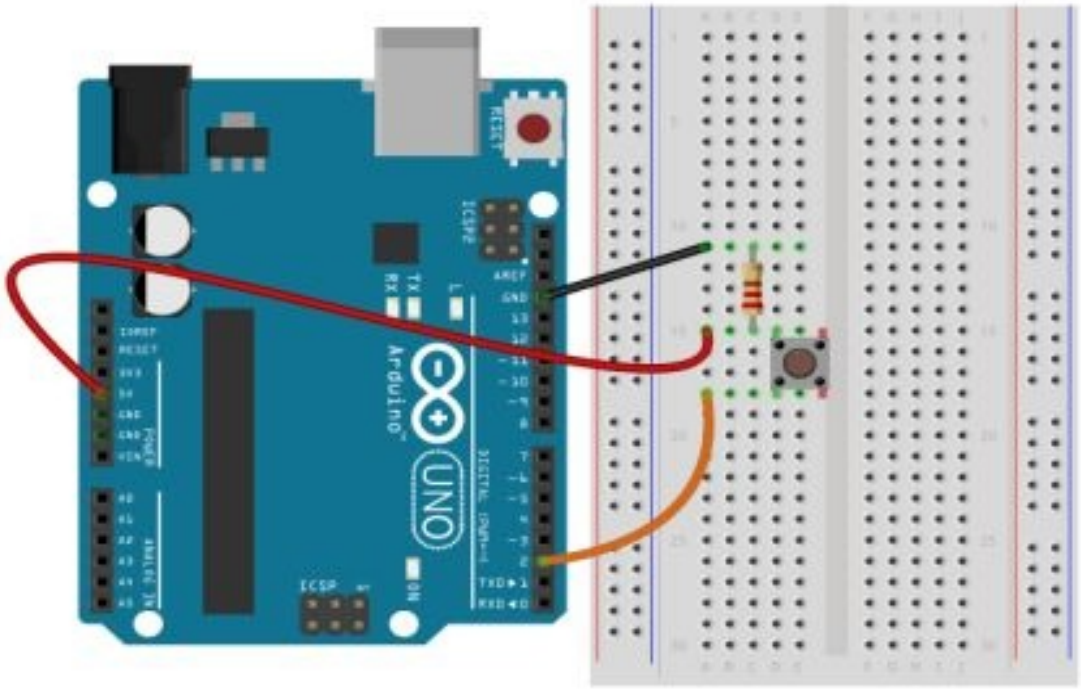
```
/*Blink,Turns on an LED on for one second, then off for one second, repeatedly.*/

// Pin 8 has an LED connected on most Arduino boards.

int led = 8;   // the setup routine runs once when you press reset: void setup() {

pinMode(led, OUTPUT);   // initialize the digital pin as an output.

                                                    }

// the loop routine runs over and over again forever: void loop() {

digitalWrite(led, HIGH);       // turn the LED on (HIGH is the voltage level) delay(1000);              // wait for a second digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW

delay(1000); // wait for a second }
```

Push Button Switch

Overview

Pushbuttons or switches connect two points in a circuit when you press them. This example turns on the built-in LED on pin 13 when you press the button.

How to Connect with Arduino Bread Board Layout



Components Required

1) Arduino with cable 2) Breadboard 3)push button 4) 220 Ω or 1K Ω

Connect three wires to the board. The first two, green and red, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10K ohm) to ground. The other leg of the button connects to the 5 volt supply.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor), and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that we read a HIGH.

You can also wire this circuit the opposite way, with a pull-up resistor keeping the input HIGH, and going LOW when the button is pressed. If so, the behavior of the sketch will be reversed, with the LED usually on and turned off when you press the button.

If you disconnect the digital I/O pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it will randomly return either HIGH or LOW. That's why you need a pull-up or pull-down resistor in the circuit.

Arduino Code-Button Read

```
 const int buttonPin = 2;     // the number of the pushbutton pin const int
ledPin =  13;      // the number of the LED pin int buttonState = 0;         //
variable for reading the pushbutton status void setup() {

   pinMode(ledPin, OUTPUT);  // initialize the LED pin as an output:
pinMode(buttonPin, INPUT);   // initialize the pushbutton pin as an input: }

 void loop() {
    buttonState = digitalRead(buttonPin);  // read the state of the pushbutton
value: if (buttonState == HIGH)   // check if the pushbutton is pressed. If it is,
the buttonState is HIGH: {
      digitalWrite(ledPin, HIGH);   // turn LED on: } else

                                                    {
```

```
digitalWrite(ledPin, LOW);     // turn LED off: }

                                }
```

POT-Analog Read

Overview

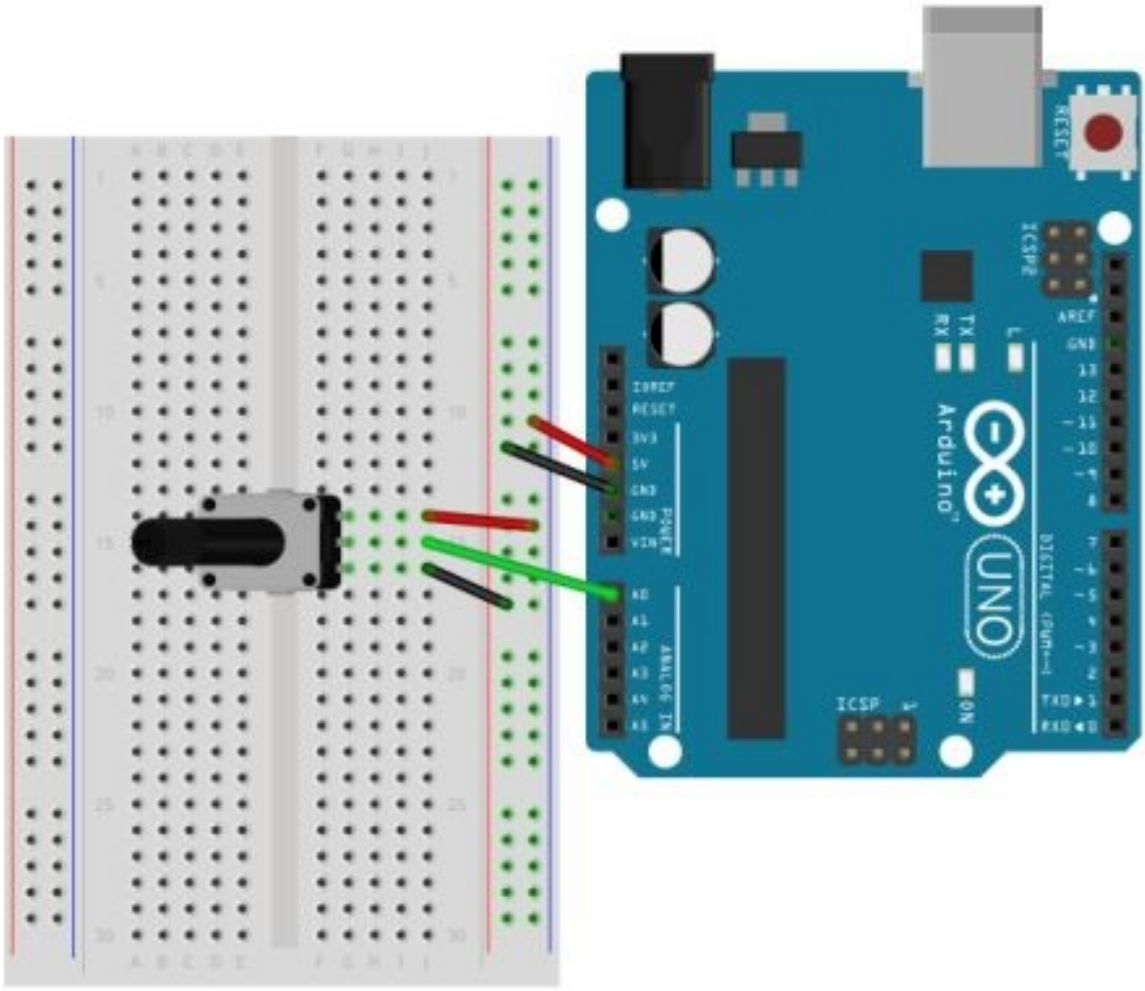
A potentiometer is a simple knob that provides a variable resistance, which we can read into the Arduino board as an analog value. In this example, we read the value of Potentiometer and display it in serial monitor.

We connect three wires to the Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, we change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the relative "closeness" of that pin to 5 volts and ground, giving us a different analog input. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and we read 0. When the shaft is turned all the way in the other direction, there are 5 volts going to the pin and we read 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

How to Connect with Arduino

Bread Board Layout



Components Required

1) Arduino with cable 2) Breadboard 3) Pot

4) 220 Ω or 1K Ω

Arduino Code-Analog Read

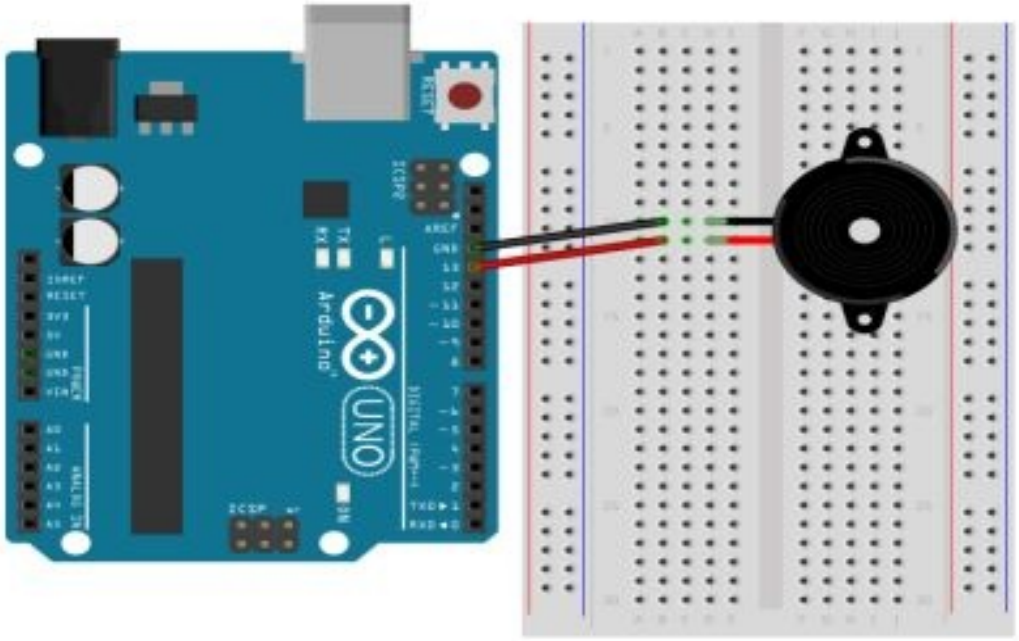
```
/*
Analog Input
*/
int potPin = 0;
void setup()
{
Serial.begin(9600);
}
void loop()
{
int reading = analogRead(potPin);
Serial.println(reading);
delay(500);
}
```

Buzzer

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric. Typical uses of buzzers and beepers include alarm devices, timers, and confirmation of user input such as a mouse click or keystroke.

Connecting a Buzzer to an Arduino Uno If a buzzer operates from a low enough voltage and draws low enough current, it can be interfaced directly to an Arduino Uno pin. The buzzer used in this example can operate from a voltage between 3 to 28V and draws the only 4mA of current at 12V. When the current drawn by the buzzer was measured at 5V, it was found that it only drew about 1.1mA which is well within the drive capability of an Arduino Uno pin.

Circuit Diagram



Components Required

1) Arduino with cable 2) Breadboard 3) buzzer 4) 220 Ω or 1K Ω

In this circuit, the positive lead of the buzzer is connected to pin 13 of the Arduino, the negative lead of the buzzer is connected to GND.

Arduino Sketch

```
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(LED_BUILTIN, OUTPUT); }
  // the loop function runs over and over again forever void loop() {
        digitalWrite(13, HIGH); // turn the buzzer on (HIGH is the voltage level) delay(1000); // wait for a second digitalWrite(13, LOW); // turn the buzzer off by making the voltage LOW
  delay(1000); // wait for a second
```

Arduino Measurement Projects

Arduino Capacitance Measurement Project

Capacitance is an object's capability to store an electric charge. Practically, this object is referred to as a capacitor. A capacitor that stores this charge in an electric field between two conductive plates is known as a parallel plate capacitor. The non-conductive material that is between these two plates is known as a dielectric. Dielectrics alter the amount of charge a capacitor can hold and, in practice, what the particular capacitor would be used for (e.g., high-frequency circuits, high voltage circuits, etc.).

The calculation for the capacitance of a parallel plate capacitor is: **C = (ε A) / d** Where ***ε*** is the permittivity of free space or dielectric, ***A*** is the surface area of overlap between the plates, and ***d*** is the distance between the plates.

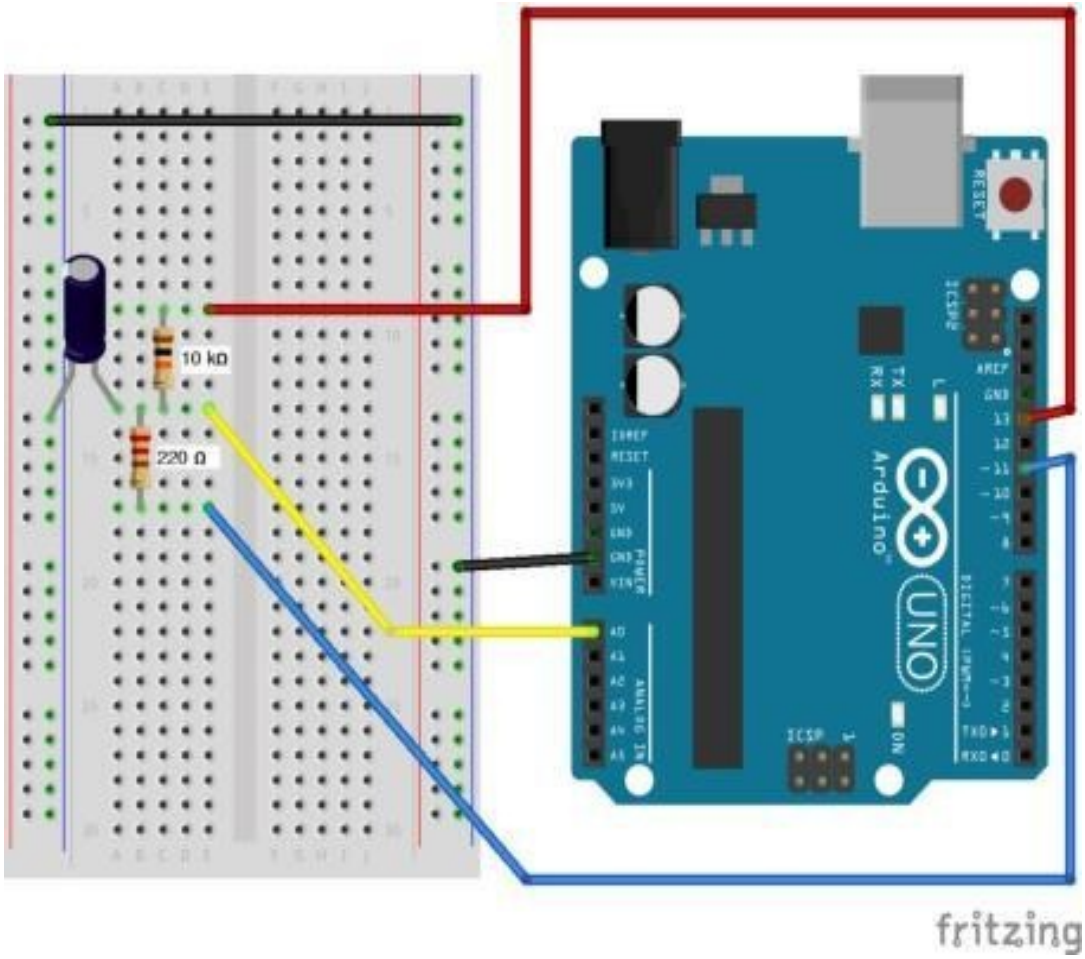
An RC (Resistor-Capacitor) circuit has a property known as an "RC Time Constant" or T (Tau). The equation for which is given below: **T = RC**

Tau can be shortened from a more complex equation (shown above) to signify the time it takes a capacitor to be charged, through a resistor, to reach 63.2% of its total voltage. This can also be calculated by the time it takes the capacitor to reach 36.8% of its total voltage upon discharging.

The Arduino will be programmed to calculate the time it takes for a capacitor to reach 63.2% of its total charge. It will then use the calculation for Tau to calculate the capacitance since the value of the resistor is already known.

Components Required: 1) Arduino Uno or any other board 2) Breadboard 3) Connecting wires 4) 220 Ohm resistor 5) 10k ohm resistor 6) Capacitor (an unknown value) and a known value capacitor for calibration.

Circuit Connection



Hardware Connection:

The wiring of this project is pretty straightforward. Just follow the diagrams provided.

NOTE: Make sure the silver stripe on the capacitor (if using a bipolar capacitor) is connected to Ground.

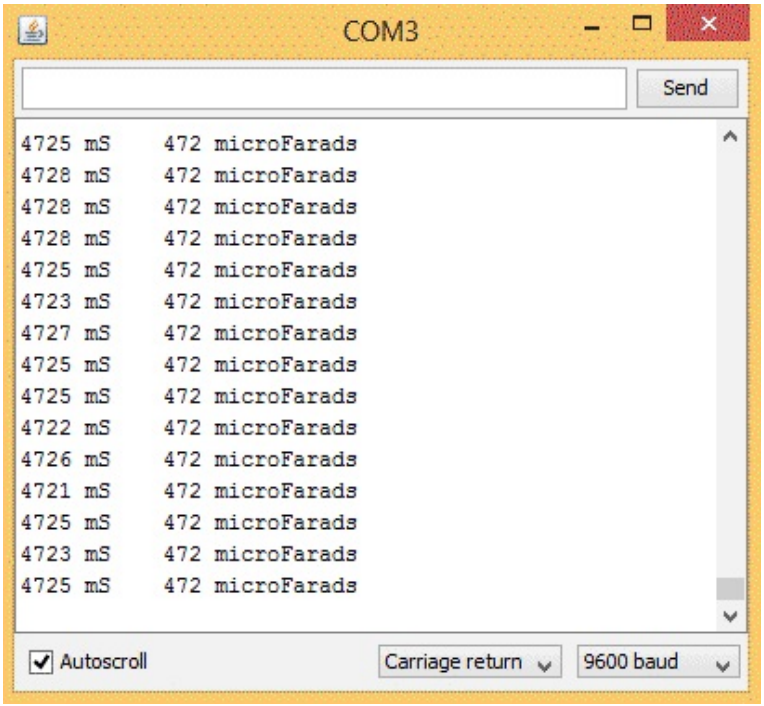
NOTE 2: the 220 Ω resistors and the wire connected to pin 11 are not necessary, but recommended as it speeds up the discharging time.

Arduino Program

```
int analogPin = 0;
int chargePin = 13;
int dischargePin = 11;

int resistorValue = 10000; unsigned long startTime; unsigned long elapsedTime; float microFarads;
float nanoFarads;

void setup()

                                        {

      pinMode(chargePin, OUTPUT); digitalWrite(chargePin, LOW); Serial.begin(9600);

                                        }


void loop()

                                        {

  digitalWrite(chargePin, HIGH); startTime = millis();

  while(analogRead(analogPin) < 648) {
    // Does nothing until capacitor reaches 63.2% of total voltage }

   elapsedTime= millis() - startTime; microFarads = ((float)elapsedTime / resistorValue) * 1000; Serial.print(elapsedTime); Serial.print(" mS    "); if (microFarads > 1) {
    Serial.print((long)microFarads); Serial.println(" microFarads"); }

  else

                                        {
```

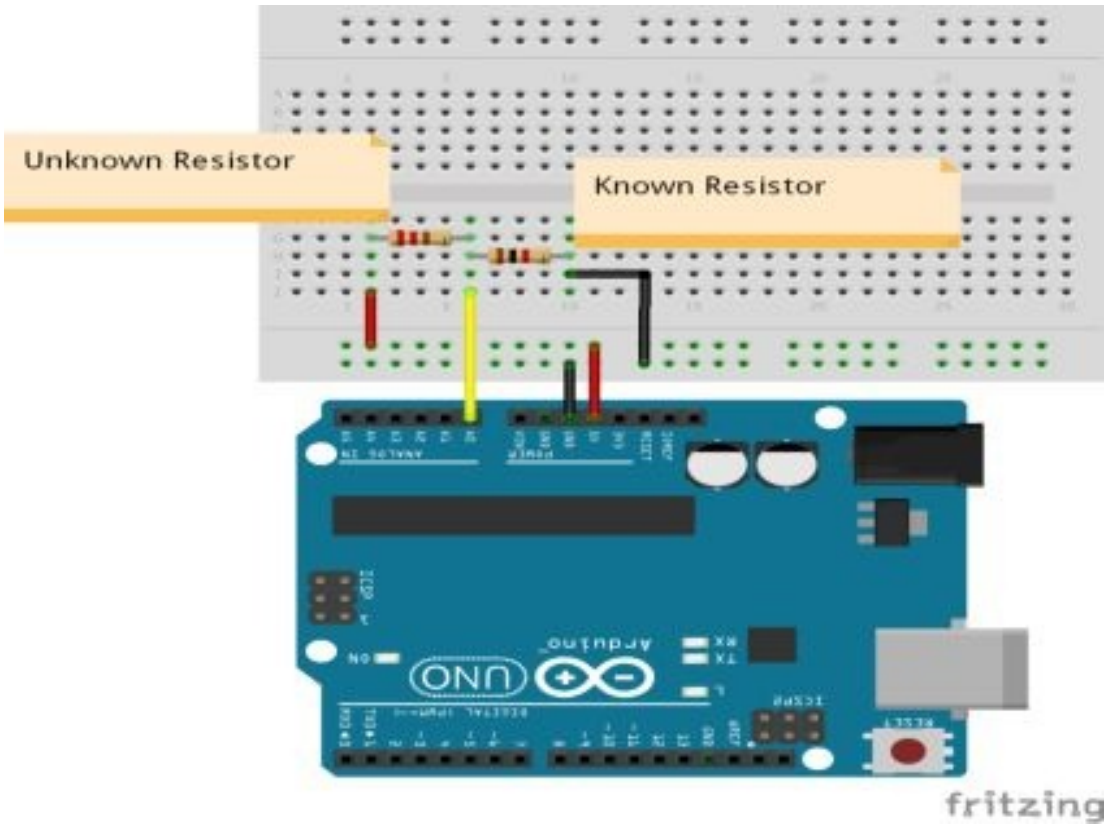
```
        nanoFarads = microFarads * 1000.0; Serial.print((long)nanoFarads);
Serial.println(" nanoFarads"); delay(500);

                                                }


        digitalWrite(chargePin, LOW); pinMode(dischargePin, OUTPUT);
digitalWrite(dischargePin, LOW); while(analogRead(analogPin) > 0) {
      // Do nothing until capacitor is discharged }

    pinMode(dischargePin, INPUT); }
```

OutPut:



Arduino Resistance (ohm) Measurement Project

If you have a tough time reading the color bands on resistors, this project is perfect for you. Instead of stressed every time you need to find the resistance of a resistor, just build an Ohm meter and measure all of your resistors. If you come up with a good way to label and organize them, you'll never need to spend time guessing the variance between those little grey and purple bands again.



Hardware for the project: The circuit is straightforward. All you need is an Arduino, the resistor you want to measure, and another resistor with a known value. We'll set up a voltage divider with the known and unknown resistors, and calculate the voltage between resistor with the Arduino. Then we'll run a program that will calculate the resistance from Ohm's Law.

Arduino Program:

```
int analogPin= 0;
int raw= 0;
```

```
int Vin= 5;
float Vout= 0;
float R1= 1000;
float R2= 0;
float buffer= 0;

void setup()

                                        {

Serial.begin(9600); }

void loop()

                                        {

raw= analogRead(analogPin); if(raw)

                                        {

buffer= raw * Vin; Vout= (buffer)/1024.0; buffer= (Vin/Vout) -1; R2= R1 *
buffer;
Serial.print("Vout: "); Serial.println(Vout); Serial.print("R2: ");
Serial.println(R2); delay(1000);

                                        }

                                        }
```

Enter the value of your known resistor (in Ohms) on line 5 of the code above. In my case, I'm using a known resistor with a value of 1K Ohms (1000 Ohms). Therefore, my line 5 should look like this: float R1 = 1000.

The program sets up analog pin A0 to read the voltage between the known resistor and the unknown resistor. You can use any other analog pin though, just change the pin number in line 1, and wire the circuit accordingly.

When you open up the serial monitor, you'll see the resistance values printed once per second. There will be two values, R2 and Vout.

R2: is the resistance of your unknown resistor in Ohms.

Vout: is the voltage drop across your unknown resistor.

Measurement of Temperature using Thermistor

Thermistors are simple, low-cost, and accurate components that make it easy to get temperature data for your projects. Remote weather stations, home automation systems, and equipment control and protection circuits are some applications where thermistors would be ideal. They're analog sensors, so the code is relatively simple compared to digital temperature sensors that require special libraries and lots of code.

In this project, you will learn how to set up a primary thermistor circuit with an Arduino that will output temperature readings to the serial monitor.

How a thermistor works

Thermistors are variable resistors that change their resistance with temperature. They are classified by the way their resistance responds to temperature changes. In the Negative Temperature Coefficient (NTC) thermistors, resistance decreases with an increase in temperature. In Positive Temperature Coefficient (PTC) thermistors, resistance increases with an increase in temperature.

NTC thermistors are the most common, and that's the type we'll be using in this tutorial. NTC thermistors are made from a semiconducting material (such as a metal oxide or ceramic) that's been heated and compressed to form a temperature sensitive conducting material.

The conducting material contains charge carriers that allow current to flow through it. High temperatures cause the semiconducting material to release more charge carriers. In NTC thermistors made from ferric oxide, electrons are the charge carriers. In nickel oxide NTC thermistors, the charge carriers are electron holes.

A basic Thermistor Circuit

Let's build a primary thermistor circuit to see how it works, so you can apply it to other projects later.

Since the thermistor is a variable resistor, we'll need to measure the resistance

before we can calculate the temperature. However, the Arduino can't measure resistance directly; it can only measure voltage.

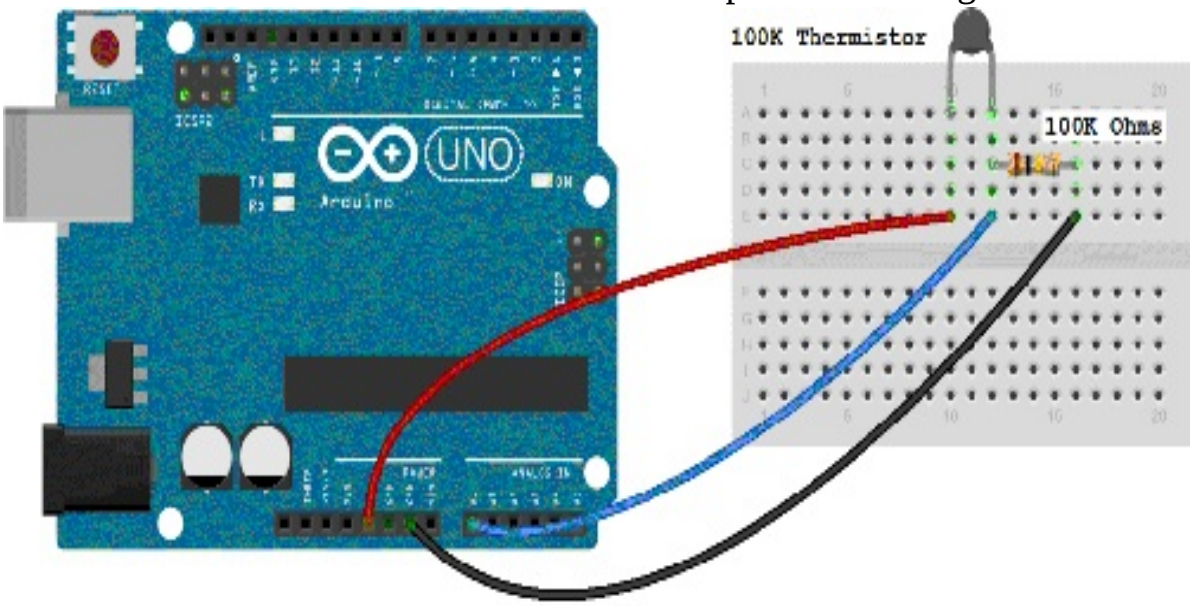
The Arduino will measure the voltage at a point between the thermistor and a known resistor. This is known as a voltage divider. The equation for a voltage divider is: $V_{out} = V_{in} \times \left(\frac{R2}{R1 + R2}\right)$ In terms of the voltage divider in a thermistor circuit, the variables in the equation above are:

$V_{out:}$ Voltage between the thermistor and a known resistor V_{in} : V_{cc}
R1: known Resistor Value

R2: Resistance of thermistor

This equation can be rearranged and simplified to solve for R2, the resistance of the thermistor:

$R2 = R1 \times \left(\frac{Vin}{Vout} - 1\right)$ Finally, the Steinhart-Hart equation is used to convert the resistance of the thermistor to a temperature reading.



The value of the resistor should be roughly equal to the resistance of your thermistor. In this case, the resistance of my thermistor is 100K Ohms, so my resistor is also 100K Ohms.

The manufacturer of the thermistor might tell you it’s resistance, but if not, you can use a multimeter to find out. If you don’t have a multimeter, you can make an Ohm meter with your Arduino by following our Arduino Ohm Meter tutorial. You only need to know the magnitude of your thermistor. For example, if your thermistor resistance is 34,000 Ohms, it is a 10K thermistor. If it’s 340,000 Ohms, it’s a 100K thermistor.

Arduino Program

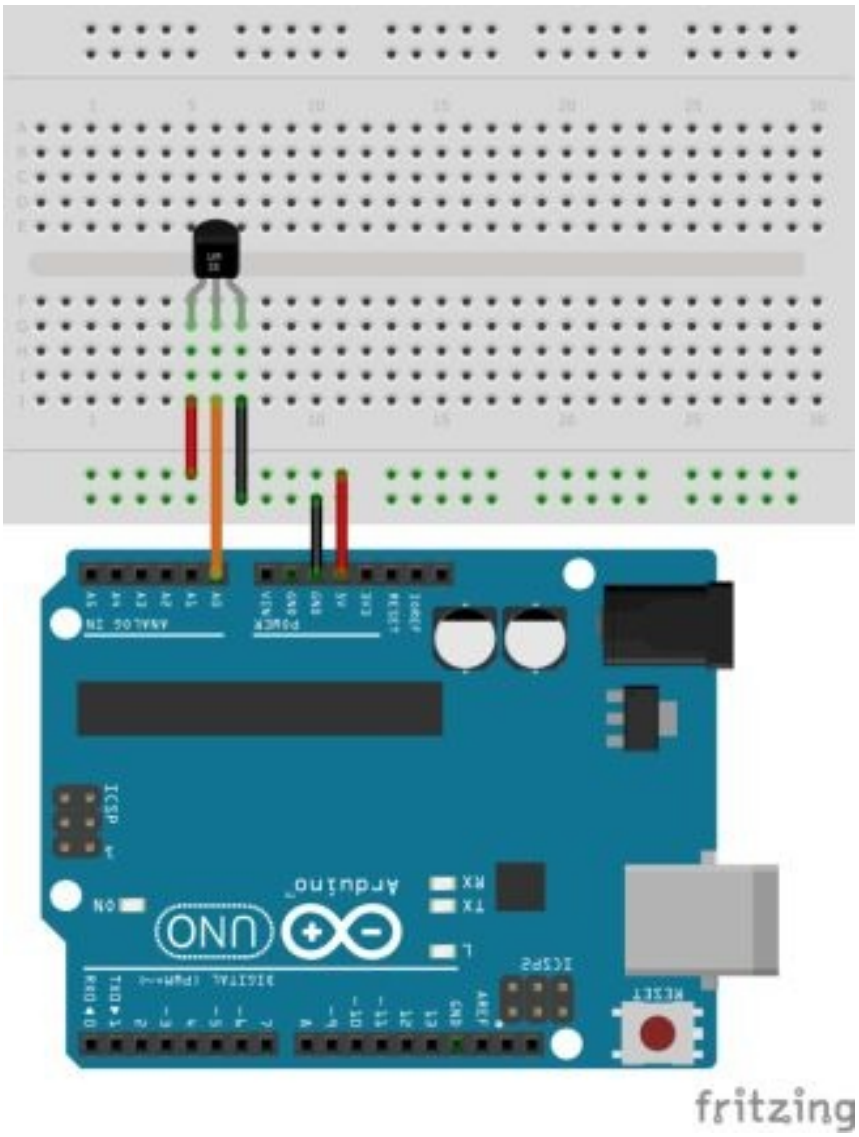
```
    int ThermistorPin = 0;
    int Vo;
    float R1 = 10000;
    float logR2, R2, T;
    float c1 = 1.009249522e-03, c2 = 2.378405444e-04, c3 = 2.019202697e-07;
void setup() {
    Serial.begin(9600);

                                                    }

    void loop() {
      Vo = analogRead(ThermistorPin); R2 = R1 * (1023.0 / (float)Vo - 1.0);
logR2 = log(R2);
      T = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2)); T = T - 273.15;
      T = (T * 9.0)/ 5.0 + 32.0; Serial.print("Temperature: "); Serial.print(T);
      Serial.println(" F"); delay(500);

                                                    }
```

After connecting the circuit above, upload this code to your Arduino to output the temperature readings to the serial monitor in Fahrenheit: To display the temperature in degrees Celsius, just comment out line 18 by inserting two forward slashes (“//”) at the beginning of the line.

Measurement of Temperature using LM35 Sensor

The LM35 is an integrated circuit sensor that can be used to measure temperature with an electrical output proportional to the temperature (in ° C).It can measure temperature more accurately than a using a thermistor. The sensor circuitry is sealed and not subject to oxidation. The LM35 generates a higher output voltage than thermocouples and may not require that the output voltage is amplified. The LM35 has an output voltage that is proportional to the Celsius temperature. The scale factor is .01V/ ° C.

The LM35 does not require any external calibration or trimming and maintains accuracy of +/-0.4 ° C at room temperature and +/-0.8 ° C over a range of 0 ° C to +100 ° C.Another essential characteristic of the LM35 is that it draws only 60 microamps from its supply and possesses a low self-heating capability. The LM35 comes in many different packages such as a TO-92 plastic transistor-like package,T0-46 metal can transistor-like package,8-lead surface mount SO-8 small outline package



Circuit Diagram

The circuit connections are made as follows:

- Pin 1 of the LM35 goes into +5V of the Arduino
- Pin 2 of the LM35 goes into analog pin A0 of the Arduino
- Pin 3 of the LM35 goes into the ground (GND) of the Arduino

Before getting a Celsius reading of the temperature, the analog output voltage must first be read. This will be the raw value divided by 1024 times 5000. It is divided by 1024 because a span of 1024 occupies 5V. Here we get the ratio of the raw value to the full span of 1024 and then multiply it by 5000 to get the millivolt value. Since the output pin can give out a maximum of 5 volts (1024), 1024 represents the possible range it can give out. The raw voltage over this 1024 (value) therefore represents the ratio of how much power the output pin is outputting

against this full range. Once we have this ratio, we then multiply it by 5000 to give the millivolt value. This is because there are 5000 millivolts in 5 volts. Once this analog voltage in millivolts is calculated, we then can find the temperature in Fahrenheit by the equation: ((Celsius * 9)/5 + 32).At the end of this program, a delay of 5000ms is included to take the temperature reading every 5 seconds.

Arduino Program

```
int outputpin= A0;
void setup()
{
Serial.begin(9600);
}
//main loop
void loop()
{
int rawvoltage= analogRead(outputpin);
float millivolts= (rawvoltage/1024.0) * 5000;
float celsius= millivolts/10;
Serial.print(celsius);
Serial.print(" degrees Celsius, ");
Serial.print((celsius * 9)/5 + 32);
Serial.println(" degrees Fahrenheit");
delay(1000);
}
```

Temperature and Humidity Measurement using DHT11 Sensor

The DHT11 humidity and temperature sensor make it really easy to add humidity and temperature data to your DIY electronics projects. It's perfect for remote weather stations, home environmental control systems, and farm or garden monitoring systems.

Here are the ranges and accuracy of the DHT11: Humidity Range: 20-90% RH

Humidity Accuracy: ±5% RH
Temperature Range: 0-50 °C
Temperature Accuracy: ±2% °C
Operating Voltage: 3V to 5.5V

The DHT11 measures relative humidity. Relative humidity is the amount of water vapor in air vs. the saturation point of water vapor in the air. At the saturation point, water vapor starts to condense and accumulate on surfaces forming dew.

The saturation point changes with air temperature. Cold air can hold less water vapor before it becomes saturated, and hot air can hold more water vapor before it becomes saturated.

The formula to calculate relative humidity is: $RH = \left(\frac{\rho w}{\rho s}\right) \times 100\%$

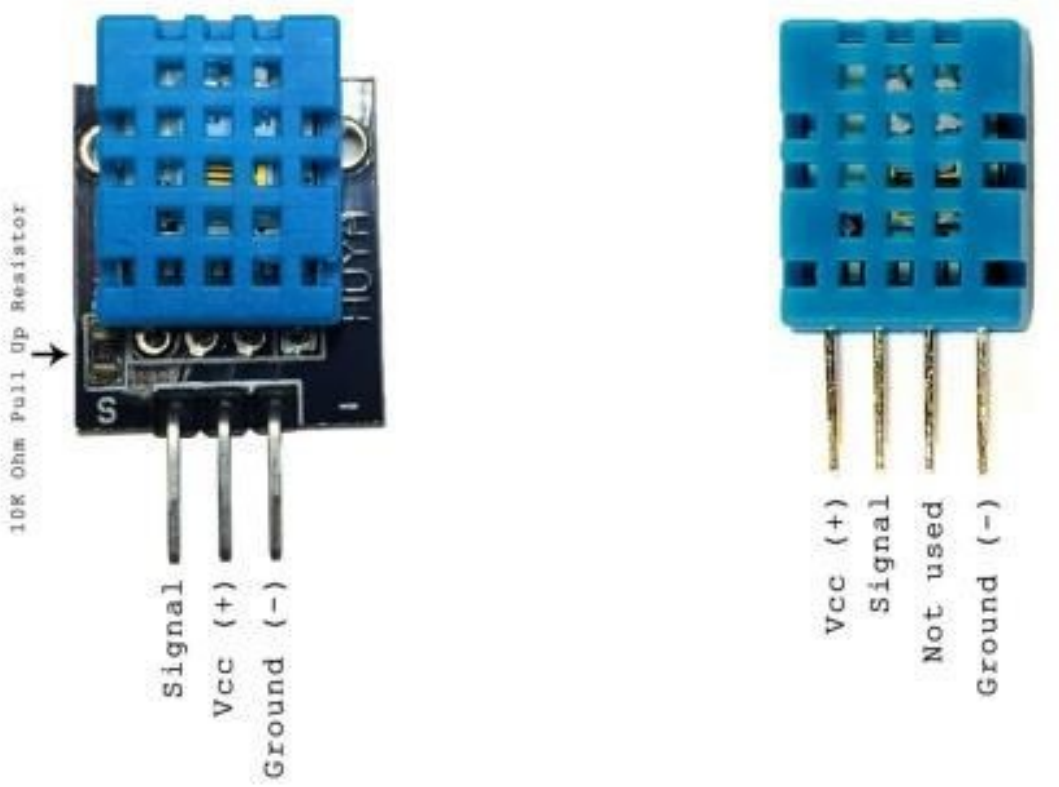
RH : Relative Humidity
Pw: Density of water vapor
Ps: Density of water at saturation

Relative humidity is expressed as a percentage. At 100% RH, condensation occurs, and at 0% RH, the air is arid.

The DHT11 detects water vapor by measuring the electrical resistance between two electrodes. The humidity sensing component is a moisture holding

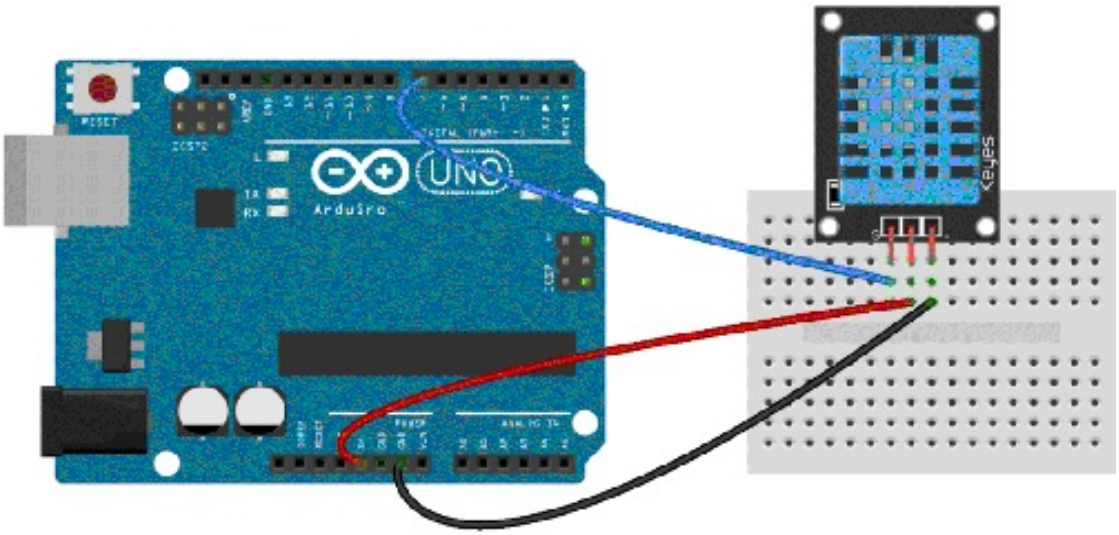
substrate with electrodes applied to the surface. When water vapor is absorbed by the substrate, ions are released by the substrate which increases the conductivity between the electrodes. The change in resistance between the two electrodes is proportional to the relative humidity. Higher relative humidity decreases the resistance between the electrodes, while lower relative humidity increases the resistance between the electrodes.

The DHT11 measures temperature with a surface mounted NTC temperature sensor (thermistor) built into the unit.

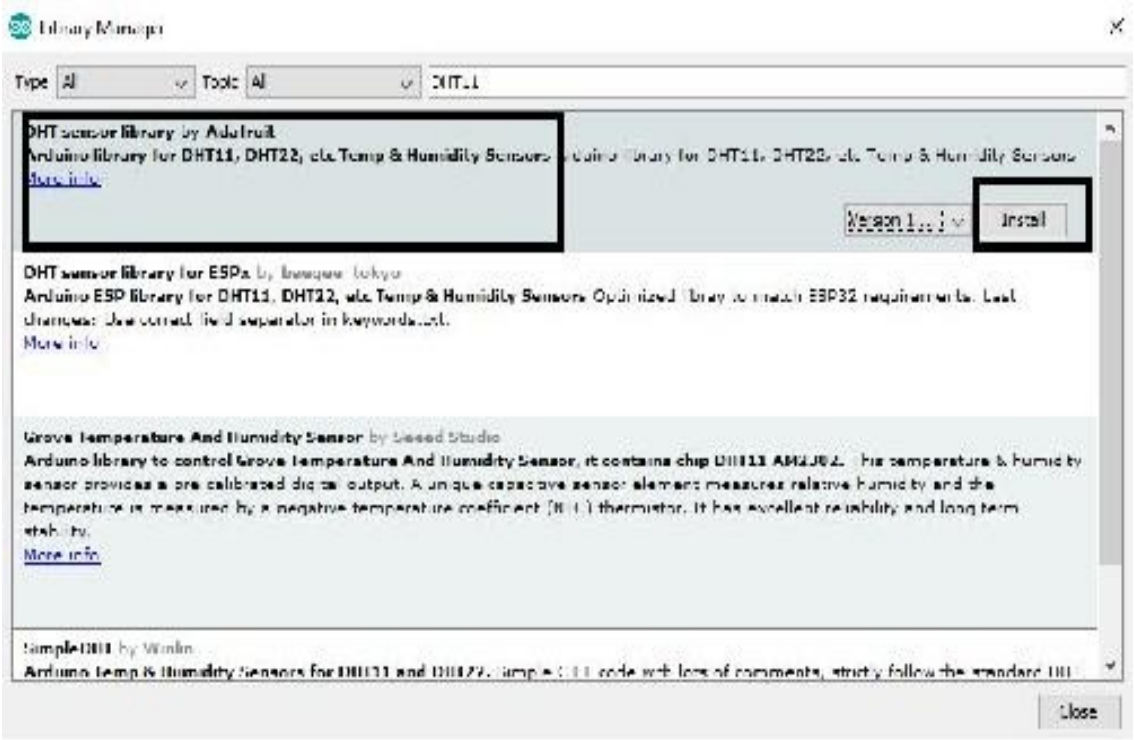


HOW TO SET UP THE DHT11 ON AN ARDUINO

Wiring the DHT11 to the Arduino is really easy, but the connections are different depending on which type you have.



Before typing your Arduino program, you need to install DHT11 from the Arduino Library Manager.

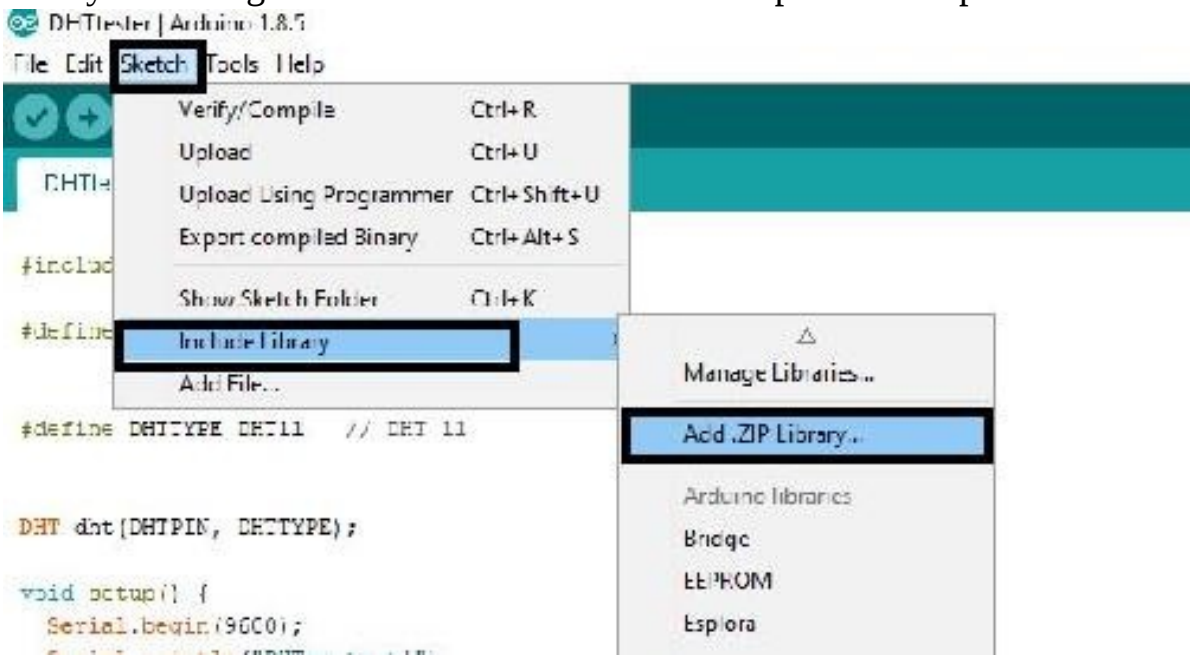


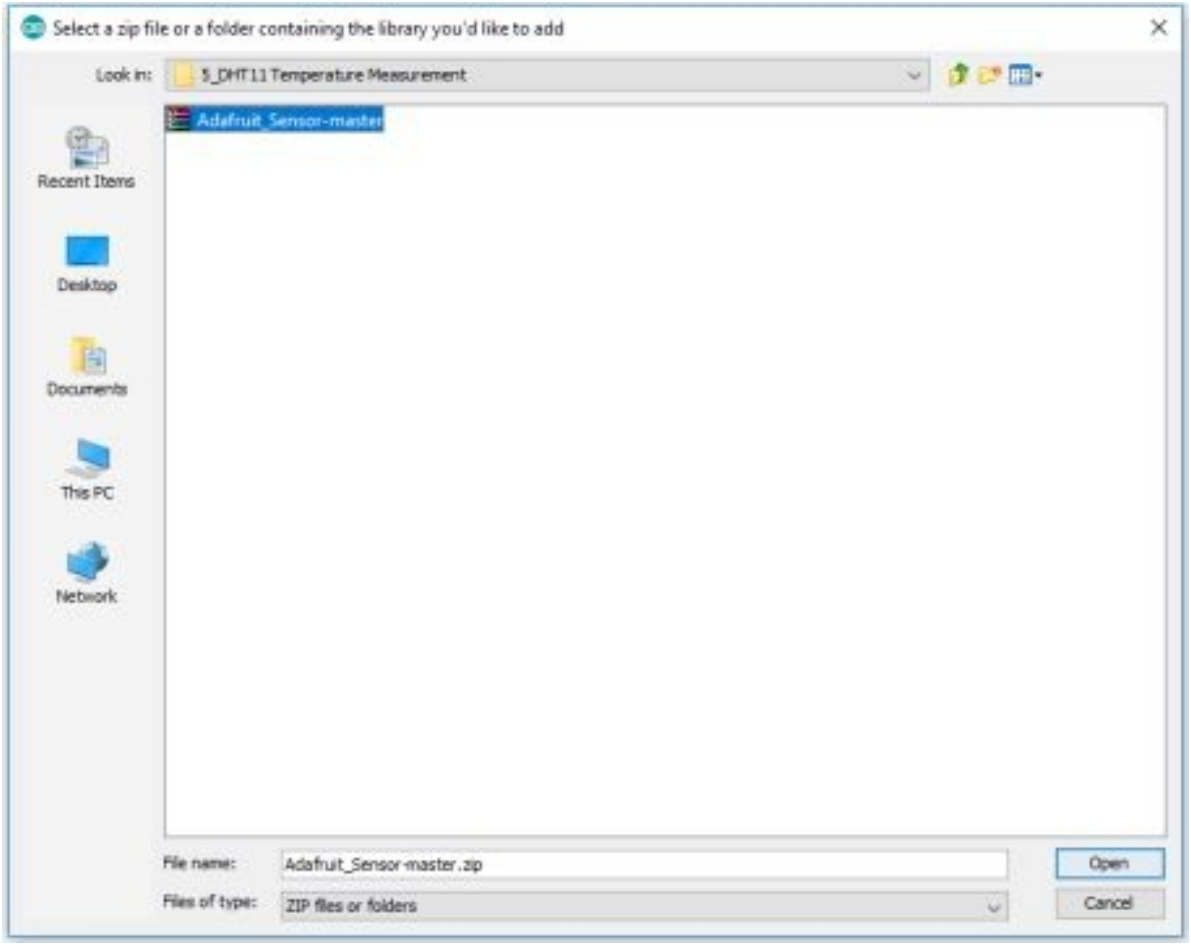
One more library file that needs to download from the external website to make this temperature sensor program to execute correctly in Arduino IDE.

Goto https://github.com/adafruit/Adafruit_Sensor

Download the zip file and add the zip file to Arduino library.

Open Arduino IDE > select sketch > click include a library and select add a .zip library and navigate the file downloaded from the previous step.




Select a zip file or a folder containing the library you'd like to add
Look in:
5_DHT11 Temperature Measurement
Adafruit_Sensor-master
Recent Items
Desktop
Documents
This PC
Network
File name:
Adafruit_Sensor-master.zip
Open
Files of type:
ZIP files or folders
Cancel

Arduino Program

```
#include "DHT.h"
#define DHTPIN 7
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  Serial.println("DHTxx test!"); dht.begin();

                                                    }


void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
   // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor) float h = dht.readHumidity(); // Read temperature as Celsius (the default) float t = dht.readTemperature(); // Read temperature as Fahrenheit (isFahrenheit = true) float f = dht.readTemperature(true); // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!"); return;

                                                    }


     // Compute heat index in Fahrenheit (the default) float hif = dht.computeHeatIndex(f, h); // Compute heat index in Celsius (isFahreheit = false) float hic = dht.computeHeatIndex(t, h, false); Serial.print("Humidity: "); Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: "); Serial.print(t);
  Serial.print(" *C ");
  Serial.print(f);
  Serial.print(" *F\t");
  Serial.print("Heat index: "); Serial.print(hic);
```

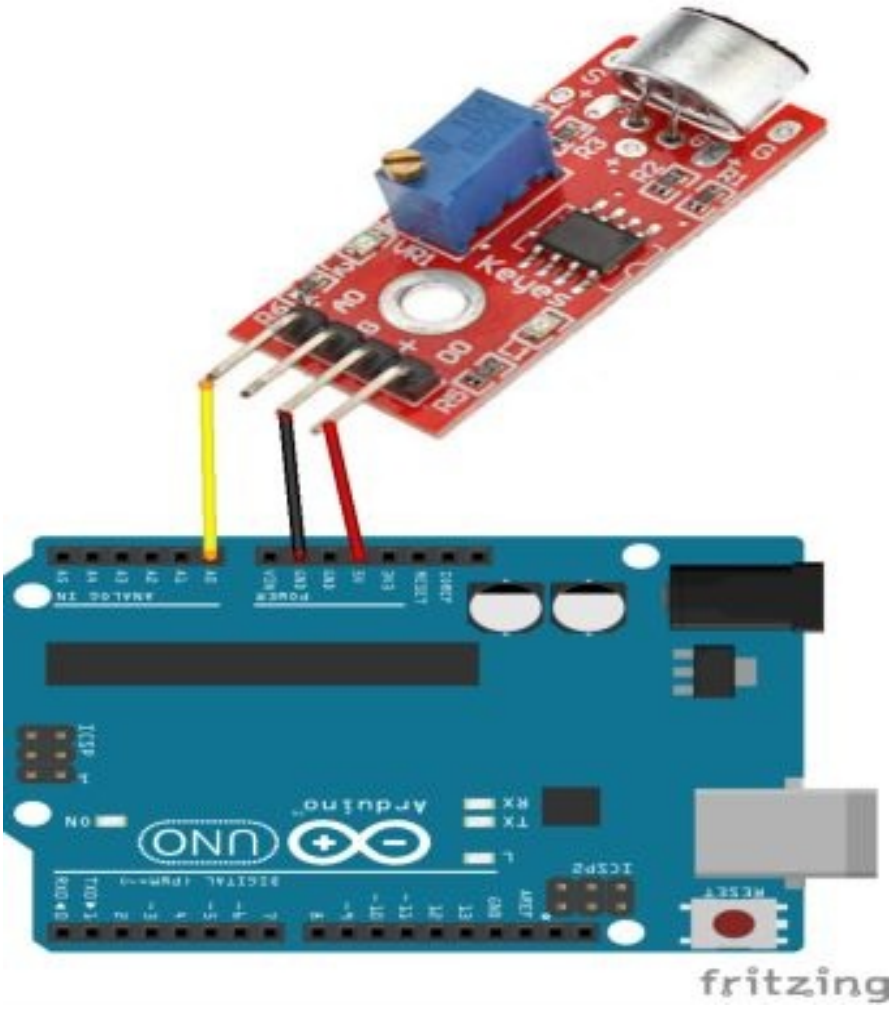
```
Serial.print(" *C ");
Serial.print(hif);
Serial.println(" *F");

                                                    }
```

Measurement of Sound/Noise using Sound sensor and Arduino

Noise pollution has really started to gain importance due to high population density. A normal human ear could hear sound levels from 0dB to 140dB in which sound levels from 120dB to 140dB are considered to be noise. Loudness or sound levels are commonly measured in decibel(dB), we have some instruments which could measure the sound signals in dB but these meters are slightly expensive, and sadly we do not have an out of box sensor module to measure sound levels in decibels. And it is not economical to purchase expensive microphones for a small **Arduino project which should measure the sound level** in a small classroom or living room.

So in this project, we will use a conventional **Sound sensor** and try measuring the sound or noise pollution level in dB as close as possible to the actual value. We will use a standard sound sensor, and the sound signals feed it to Arduino in which we will use a regression method to calculate the sound signals in dB. To check if the values obtained are correct, we can use the **"Sound Meter" android application** if you have a better meter you can use that for calibration. Do note that this project does not aim to measure dB accurately and will just give values as close as possible to the actual value.

Components Required: 1) Arduino Uno 2) Sound sensor 3) Connecting



Wires

Measuring Sound Levels

The Audio signal from the output of the amplifier is a varying voltage. To measure the sound level, we need to take multiple measurements to find the minimum and maximum extents or "peak to peak amplitude" of the signal.

In the example below, we choose a sample window of 50 milliseconds. That is sufficient to measure sound levels of frequencies as low as 20 Hz - the lower limit of human hearing.

After finding the minimum and maximum samples, we compute the difference and convert it to volts, and the output is printed to the serial monitor.

Arduino Program

```
const int sampleWindow = 50; // Sample window width in mS (50 mS = 20Hz) unsigned int sample;
void setup()

                                        {

  Serial.begin(9600);

                                        }


void loop()

                                        {

   unsigned long startMillis= millis();  // Start of sample window unsigned int peakToPeak = 0;    // peak-to-peak level unsigned int signalMax = 0; unsigned int signalMin = 1024; // collect data for 50 mS
  while (millis() - startMillis < sampleWindow) {
      sample = analogRead(0); if (sample < 1024)  // toss out spurious readings {
     if (sample > signalMax) {
       signalMax = sample;  // save just the max levels }
     else if (sample < signalMin) {
       signalMin = sample;  // save just the min levels }

                                        }

                                        }

    peakToPeak = signalMax - signalMin;  // max - min = peak-peak amplitude double volts = (peakToPeak * 5.0) 1024;  / convert to volts Serial.println(volts);

                                        }
```

After uploading the program to your Arduino board open the Serial monitor in your PC or Laptop and check the value the sensor displaying on your monitor.

Measurement of Distance using ultrasonic Sensor

Ultrasonic rangefinders are fun little modules that measure distance. You can use them to find the distance to an object or to detect when something is near the sensor like a motion detector. They're ideal for projects involving navigation, object avoidance, and home security. Because they use sound to measure distance, they work just as well in the dark as they do in the light.

THE SPEED OF SOUND

Ultrasonic range finders measure distance by emitting a pulse of ultrasonic sound that travels through the air until it hits an object. When that pulse of sound hits an object, it's reflected off the object and travels back to the ultrasonic range finder. The ultrasonic range finder measures how long it takes the sound pulse to travel in its round-trip journey from the sensor and back. It then sends a signal to the Arduino with information about how long it took for the sonic pulse to travel.

Knowing the time it takes the ultrasonic pulse to travel back and forth to the object, and also knowing the speed of sound, the Arduino can calculate the distance to the object. The formula relating the speed of sound, distance, and time traveled is: $Speed = \frac{distance}{Time}$

$$distance = speed \; x \; time$$

The time variable is the time it takes for the ultrasonic pulse to leave the sensor, bounce off the object, and return to the sensor. We actually divide this time in half since we only need to measure the distance to the object, not the distance to the object *and* back to the sensor. The speed variable is the speed at which sound travels through the air.

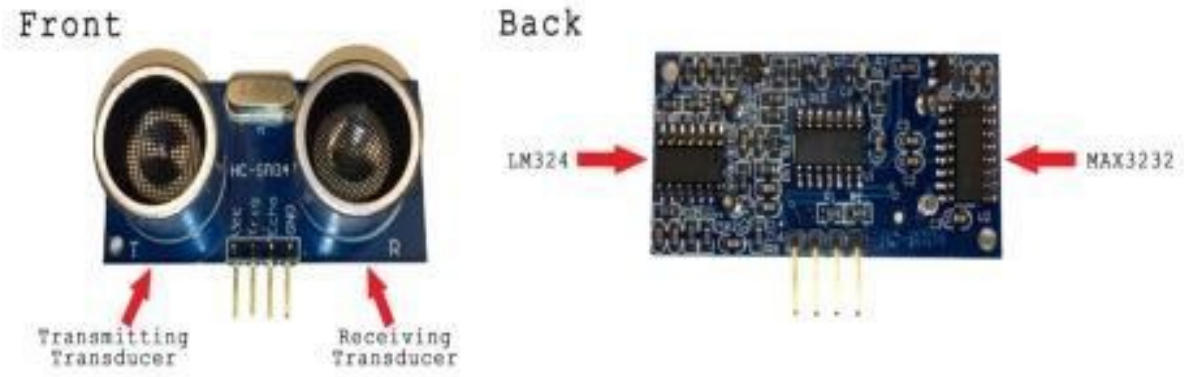
The speed of sound in air changes with temperature and humidity. Therefore, in order to accurately calculate distance, we'll need to consider the ambient temperature and humidity. The formula for the speed of sound in air

with temperature and humidity accounted for is: C= 331.4+ (0.606 x T)+(0.0124 x H) C: speed of sound in meters per second (m/s) 331.4: speed of sound (in m/s) at 0˚ C and 0% humidity T: Temperature H: % Humidity (Relative Humidity) For example, at 20 C and 50% humidity, sound travels at a speed of: C = 331.4 + (0.606 x 20) + (0.0124 x 50) C = 344.02 m/s

In the equation above, it's clear that temperature has the most significant effect on the speed of sound. Humidity does have some influence, but it's much less than the effect of temperature.

HOW THE ULTRASONIC RANGE FINDER MEASURES DISTANCE

On the front of the ultrasonic range, finder are two metal cylinders. These are transducers. Transducers convert mechanical forces into electrical signals. In the ultrasonic rangefinder, there is a transmitting transducer and receiving transducer. The transmitting transducer converts an electrical signal into the ultrasonic pulse, and the receiving transducer converts the reflected ultrasonic pulse back into an electrical signal. If you look at the back of the rangefinder, you will see an IC behind the transmitting transducer labeled MAX3232. This is the IC that controls the transmitting transducer. Behind the receiving transducer is an IC labeled LM324. This is a quad Op-Amp that amplifies the signal generated by the receiving transducer into a signal that's strong enough to transmit to the Arduino.



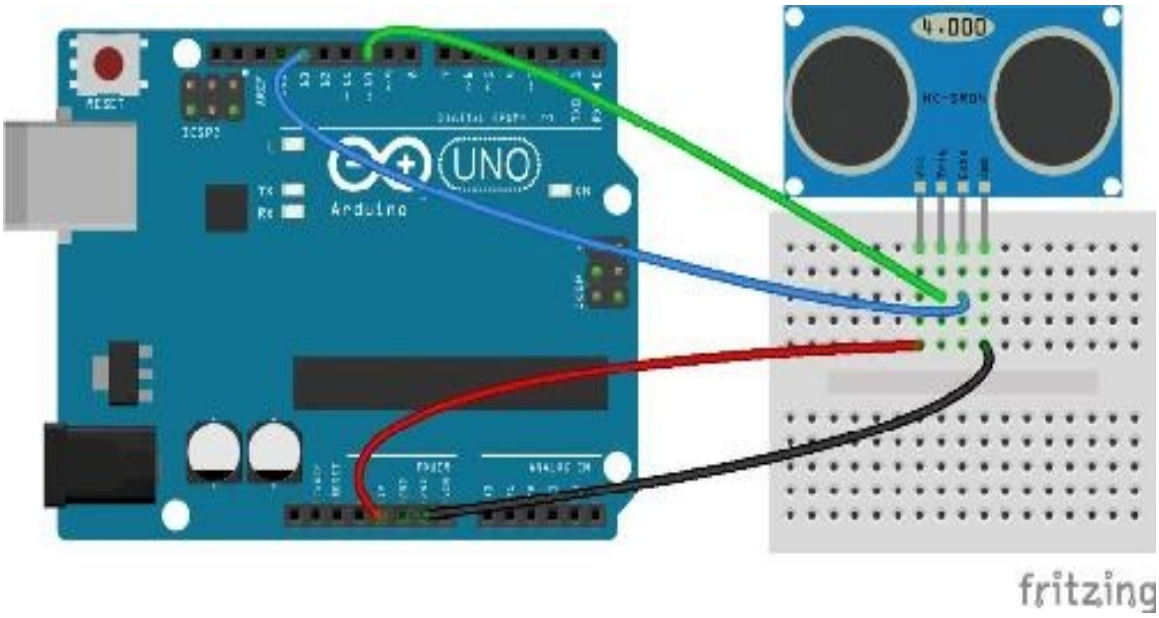
The HC-SR04 ultrasonic range finder has four pins: Vcc, Trig, Echo, and GND. The Vcc pin supplies the power to generate the ultrasonic pulses. The GND pin is connected to ground. The Trig pin is where the Arduino sends

the signal to start the ultrasonic pulse. The Echo pin is where the ultrasonic range finder sends the information about the duration of the trip taken by the ultrasonic pulse to the Arduino.

To initiate a distance measurement, we need to send a 5V high signal to the Trig pin for at least 10 µs. When the module receives this signal, it will emit 8 pulses of ultrasonic sound at a frequency of 40 KHz from the transmitting transducer. Then it waits and listens at the receiving transducer for the reflected signal. If an object is within range, the 8 pulses will be reflected back to the sensor. When the pulse hits the receiving transducer, the Echo pin outputs a high voltage signal.

The length of this high voltage signal is equal to the *total time* the 8 pulses take to travel from the transmitting transducer and back to the receiving transducer. However, we only want to measure the distance to the object, and not the distance of the path the sound pulse took. Therefore, we divide that time in half to get the time variable in the d = s x t equation above. Since we already know the speed of sound (s), we can solve the equation for distance.

Circuit Diagram



Arduino Program:

```
#define trigPin 10
```

```
#define echoPin 13

void setup() {
  Serial.begin (9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

                                        }


void loop() {
  float duration, distance;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

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

   duration = pulseIn(echoPin, HIGH); distance = (duration / 2) * 0.0344; if
(distance >= 400 || distance <= 2){
    Serial.print("Distance = "); Serial.println("Out of range"); }
  else {
    Serial.print("Distance = "); Serial.print(distance);
    Serial.println(" cm");
    delay(500);

                                        }

  delay(500);

                                        }
```

After uploading the program to your Arduino board open the Serial monitor in your PC or Laptop and check the value the sensor displaying on your monitor. This will show the distance between your ultrasonic sensor and the object you measure the distance.

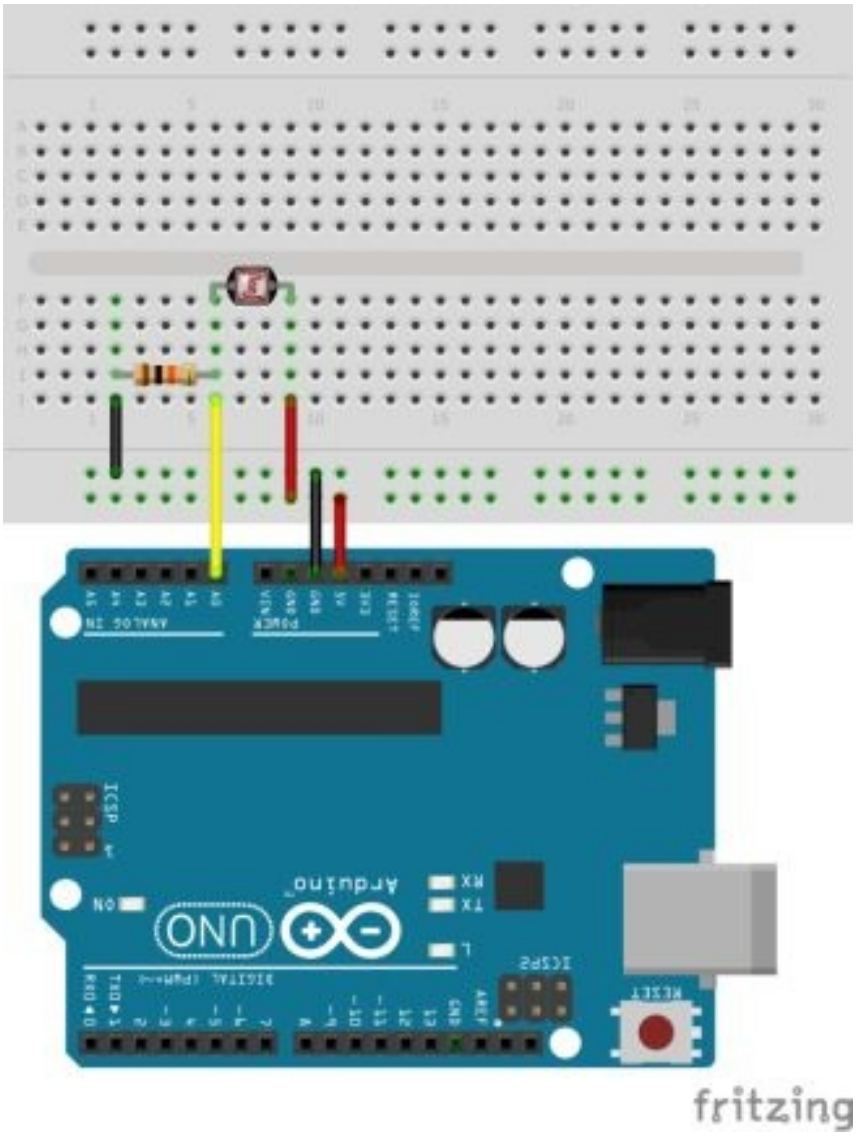
Measurement of Light using LDR (LUX meter)

A simple way to define a lux meter is to say that it measures the brightness of the light falling on the sensor. Commercially accessible lux meters can vary in price from around $10 up to thousands of dollars, but it’s inexpensive and way more fun to build one yourself. For this project, we’ll use an LDR, write software to calculate illuminance, and build an Arduino and LDR based lux meter.

Components Required:

1) LDR
2) Arduino Uno 3) Breadboard 4) 2.2k ohm resistor

Circuit Diagram



Building the lux meter is simple. You only need an Arduino, an LDR, and a 2.2 k ohm resistor. In theory, you can use any resistance value, but I chose 2.2 k ohm because the LDR's resistance was in the order of a few kilo-ohms under typical room lighting conditions.

Plug the shield into the Arduino, and build a simple voltage divider circuit with the LDR and resistor. The voltage divider circuit is the crux of our sensor circuit. The 5 volt supply is split between the LDR and the 2.2 k ohm resistor. As the LDR's resistance changes, the fraction of the voltage across the two resistors changes as well. If the voltage across the 2.2 k ohm resistor is measured by the Arduino, it's easy to add some code to determine the resistance that the

LDR is exhibiting. The LDR connects to 5V; the resistor connects to ground and the point in between connects to analog input 0.

Arduino Program

```
void setup() {

Serial.begin(9600);

                                        }

void loop()

                                        {

int vout1 = analogRead(A0); // Read the analogue pin float vout = vout1/204.6;

Serial.print(vout1);

Serial.print("DU");

Serial.print(vout);

Serial.println(" vout");

float R = (11000-vout*2200)/vout; // calculate the resistance //float R = pow( X, -1);

Serial.print(R); // light dependant resistance Serial.println(" Resistance."); float lux= (pow( R, (1/-0.8616)))/(pow( 10, (5.118/-0.8616))); //lux calculation Serial.print(lux);

Serial.print(" Lux.");

Serial.println("");
```

```
    delay(3000); //delay for a second //lux2

    float lux2 = 65.9 * (pow( vout1, 0.352)); Serial.print(lux2);

    Serial.print(" lux form2\n"); }
```

Depending on your type of LDR you might want to change some aspects of the code, but this code will work for most types.

After uploading the program to your Arduino board open the Serial monitor in your PC or Laptop and check the value the sensor displaying on your monitor.

Measurement of distance using sharp Sensor

An alternative to Ultrasonic sensors rangefinders, this GP2Y0A41SK0F Sharp sensor will provide accurate distance measurements. These Sharp sensors will detect distances in a range of 1.5-12 in (4-30 cm).

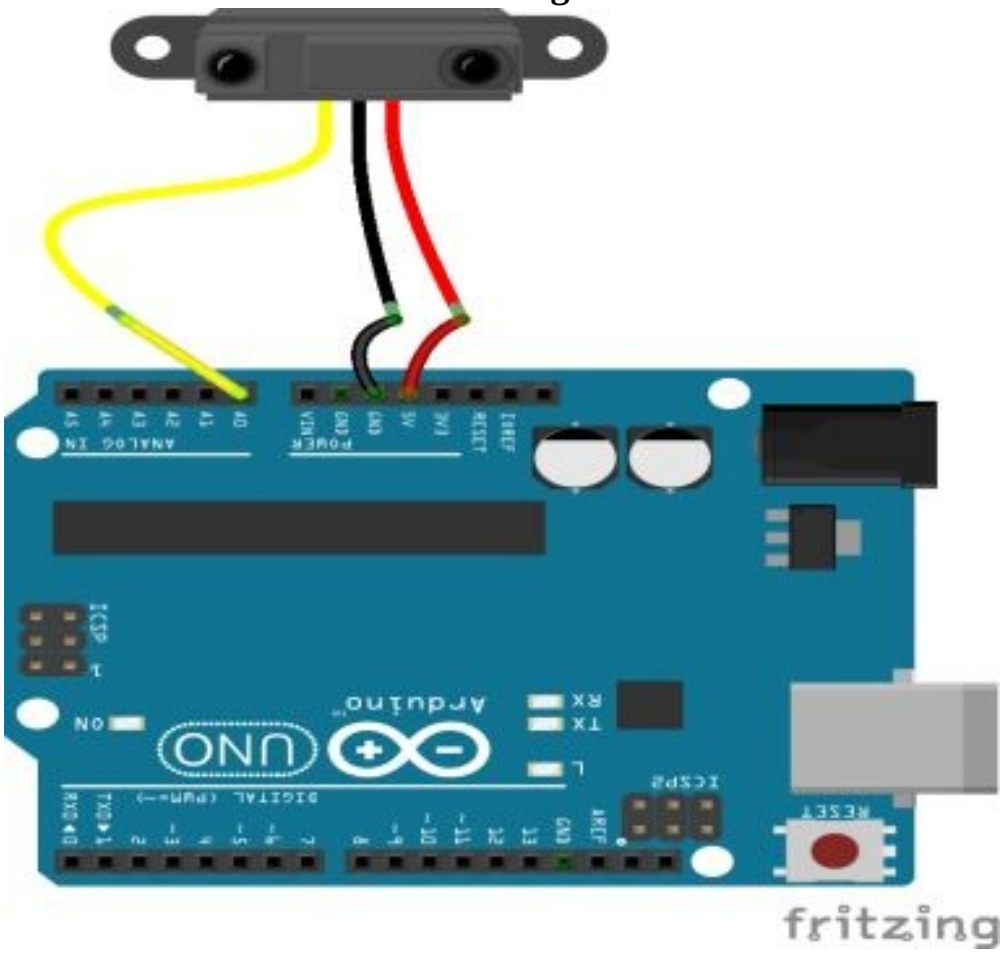
Infrared is sent out by the sensor which bounces off objects. The analog voltage that is returned determines how close the nearest object is. The closer it is, the higher voltage is returned.

Parts Required

1) Arduino 2) Sharp IR sensor 3) Connecting Wires 4) Breadboard (optional) Generally, when the detector area is exposed to a light spot, the device will convert light into several outputs of electrical currents in each of the sensor's electrodes. Using the distribution of the outputs currents, you can then find the light position When you add an IR emitting diode, you can make a distance sensor.

The diode will produce light with a specific wavelength (IR), and the light sensor will only detect this wavelength. When an object is close to the device, the light will bounce back on it, and the intensity of this light will be measured by the sensor (the voltage in our case). The sensor will then use this data to determine the distance of the object.

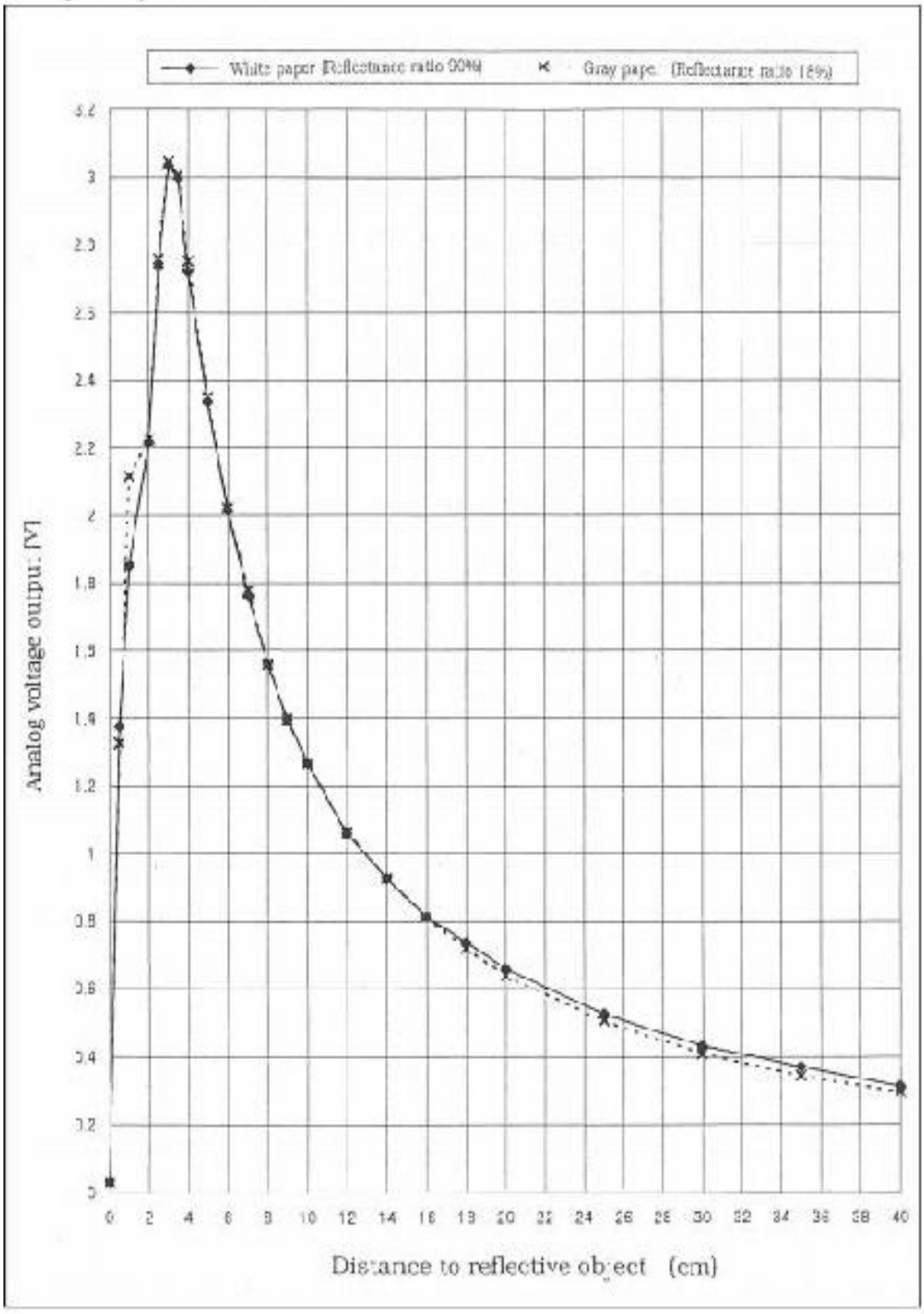
Circuit Diagram



Arduino Program

```
#define sensor A0

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

                                        }


void loop() {


                                        //

            //Serial.println(analogRead(sensor)); float volts = analogRead(sensor)*0.0048828125; Serial.println( volts); int distance = 13*pow(volts, -1); delay(1000);

  if (distance <= 30){
    Serial.println(distance); }

                                        }
```

•Example of output distance characteristics



Arduino Program Explanation Why do we multiply value from the sensor by (5/1024)?

We convert the analogRead() value to a voltage. The analog pins convert the value of the sensor to a byte value which is between 0 and 1023. But we need the real analog value. To do that, we divide the voltage rating, 5V, (maximum voltage that can safely be applied to an electric device) by 1024. Then we multiply the result of this operation by the value from the sensor to get our voltage.

What is pow(volts, -1)?

In this line ‘volts’ is the base and -1 is the exponent (power to which the base is raised).

Based on the Sharp datasheet graph we can calculate the function (For distance > 3cm, its exponential).

The distance between the sensor and the object will be printed on this monitor (9600 baud for this code).

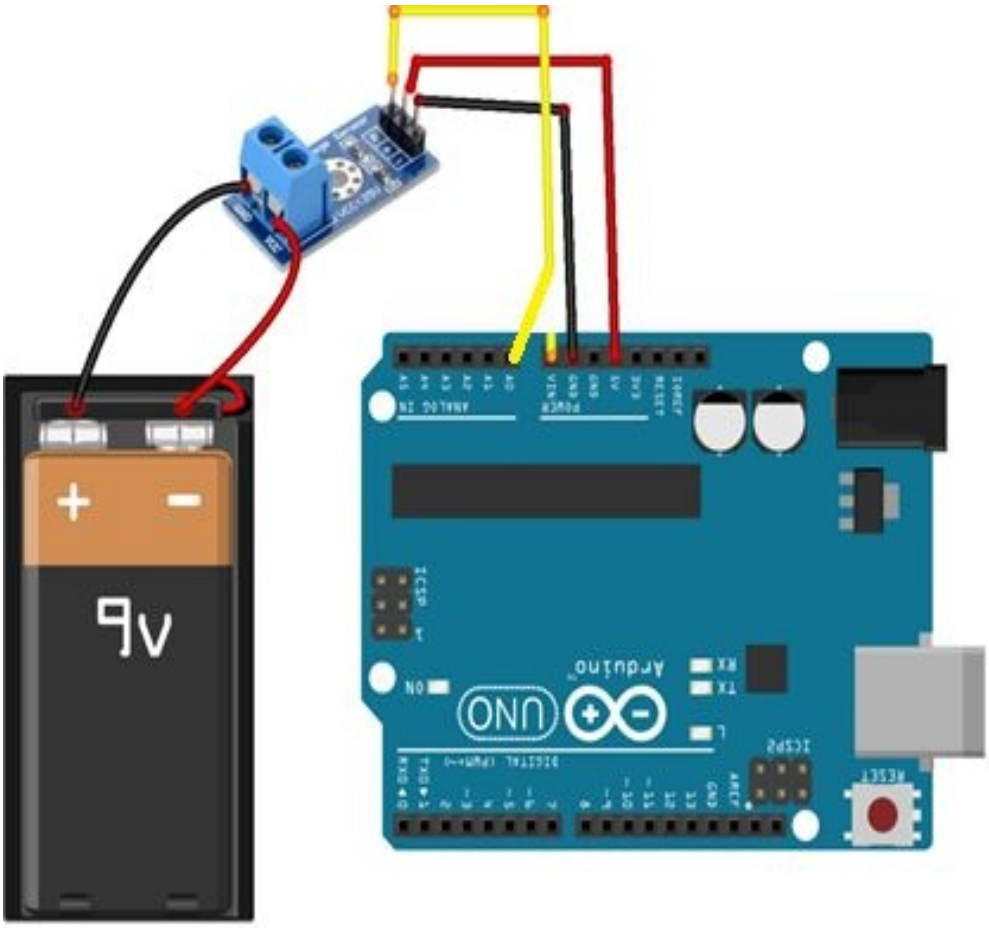
Voltage Measurement using Voltage sensor

Arduinos have built-in voltage measuring device. Unluckily, they only support voltages of 0-5V. This module (Voltage Sensor) allows you to measure voltages of 0-25V by giving a lower voltage to the Arduino for measuring. After you have this value, you merely feed it through some math, and you get your actual voltage.

It is basically a 5:1 voltage divider using a 7.5K and a 30K Ohm resistor.

Keep in mind; you are limited to voltages that are less than 25 volts. More than that and you will surpass the voltage limit of your Arduino input.

Components Required 1) Arduino 2) Voltage Sensor 3) Battery(less than 25v) for testing 4) Connecting wires 5) Breadboard (optional)


+
-
9v
UNO
Arduino

Inputs

- **GND –** This is where you connect the low side of the voltage you are measuring. Caution! : This is the same electrical point as your Arduino ground.
- **VCC:** The is where you connect the high side of the voltage you are measuring

Outputs

- **S:** This connects to your Arduino analog input.
- **– (or minus):** connect this to Arduino GND
- **+:** Connect this to 5v of Arduino board

Arduino Program

```
#include
int val11;
float val2;
void setup()

                                        {

Serial.begin(9600);  Serial.println("Emartee.Com");  Serial.println("Voltage:
"); Serial.print("V"); }
void loop()

                                        {

float temp;
val11=analogRead(1); temp=val11/4.092; val2=(temp/10);
Serial.println(val2); delay(1000);

                                        }
```

Measurement of current using current sensor

A Current Sensor is an essential device in power calculation and administration applications. It measures the current through a device or a circuit and produces an appropriate signal that is relative to current measured. Usually, the output signal is an analog voltage.

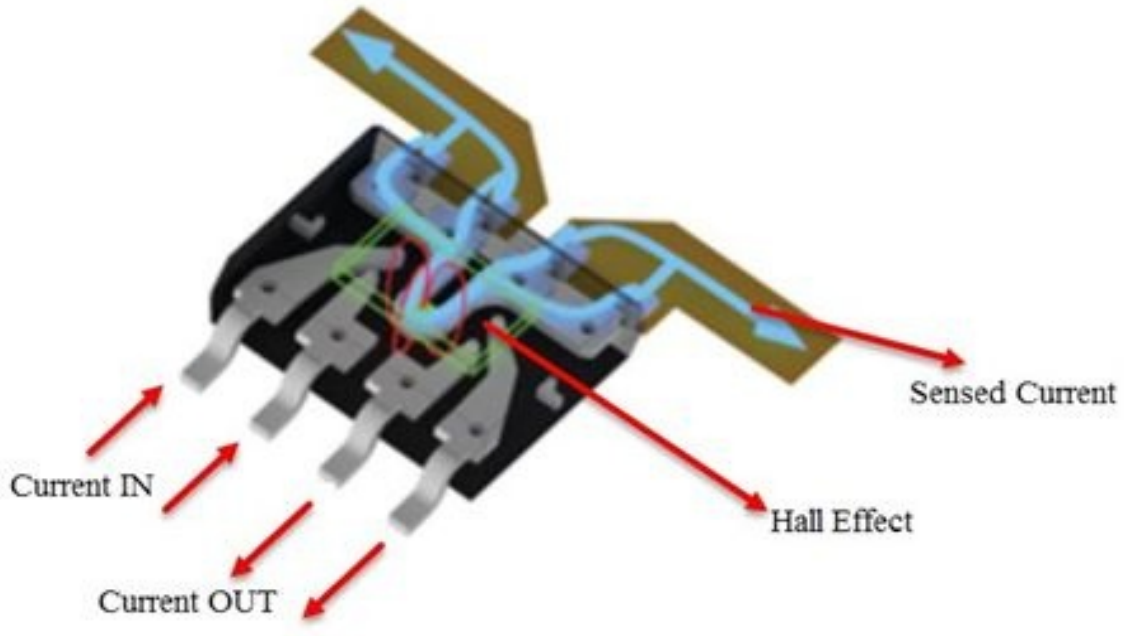
For this project ACS712 based current measuring device is used which is extensively available and cheap price.

The ACS712 Current Sensor is a product of Allegro MicroSystems that can be used for precise measurement of both AC and DC currents. This sensor is based on Hall Effect, and the IC has an integrated Hall Effect device.

Coming to the output of the ACS712 Current Sensor, it produces an analog voltage that is proportional to AC or DC currents (whichever is being sensed).

How Current Sensor Works?

As stated earlier, the ASC712 is based on Hall Effect. There is a copper strip linking the IP+ and IP-pins internally. When some current flows through this copper conductor, a magnetic field is generated. This is sensed by the Hall Effect sensor.

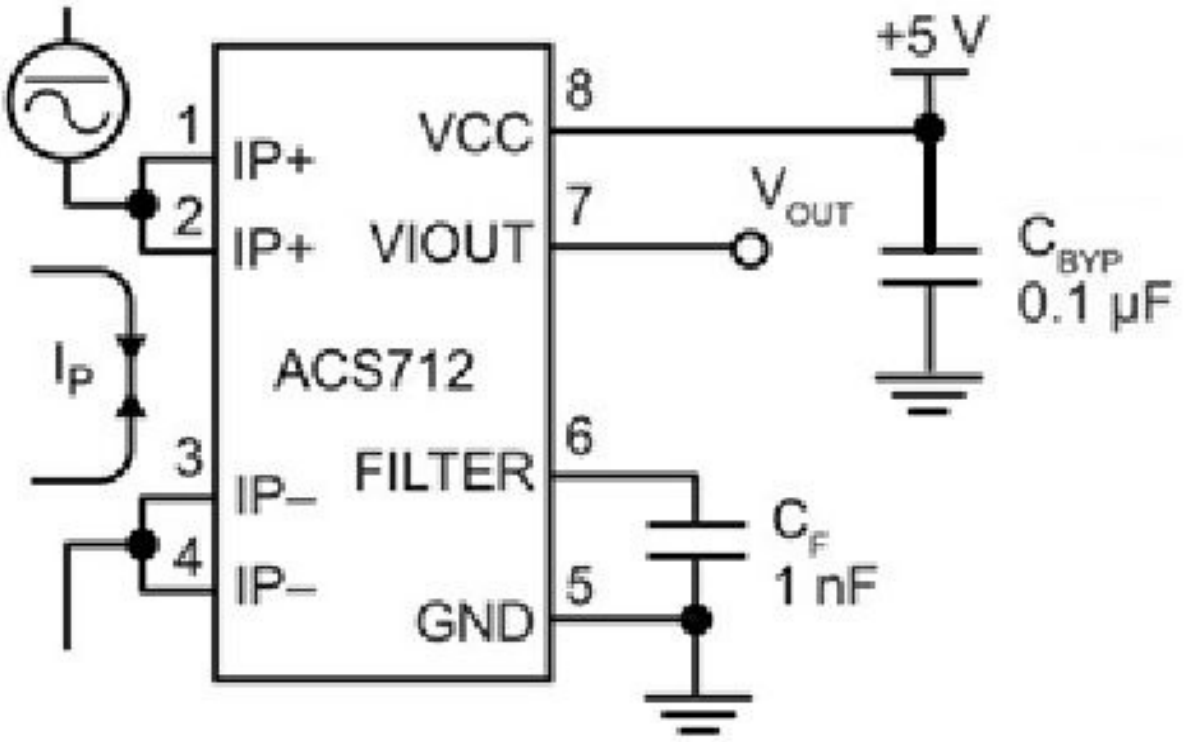


The Hall Effect sensor then transforms this magnetic field into the suitable voltage. In this method, the input and output are completely isolated.

ASC712 Current Sensor Application Circuit The distinctive application circuit using the ASC712 Current Sensor is given in its datasheet, and the succeeding images show the same.

ACS712 Current Sensor Module

Using one of the alternatives of the ACS712 IC (5A, 20A or 30A), several manufacturers developed ASC712 Current



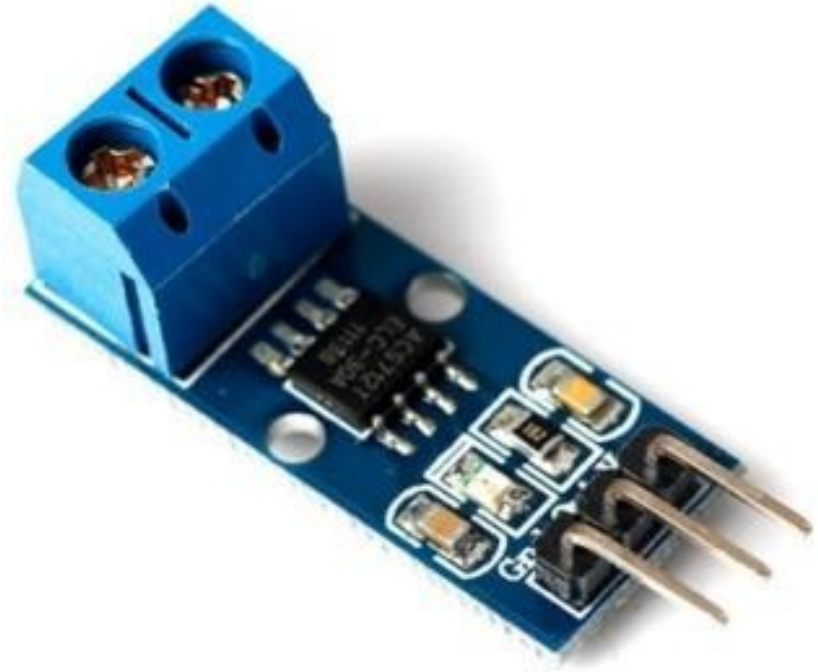
Sensor Module boards that can be easily interfaced to a microcontroller like Arduino.

The next image shows the ASC712 Current Sensor board used in this project.

As you can see, it is somewhat a simple board with only a few components

including the ASC712 IC, few passive components and connectors.

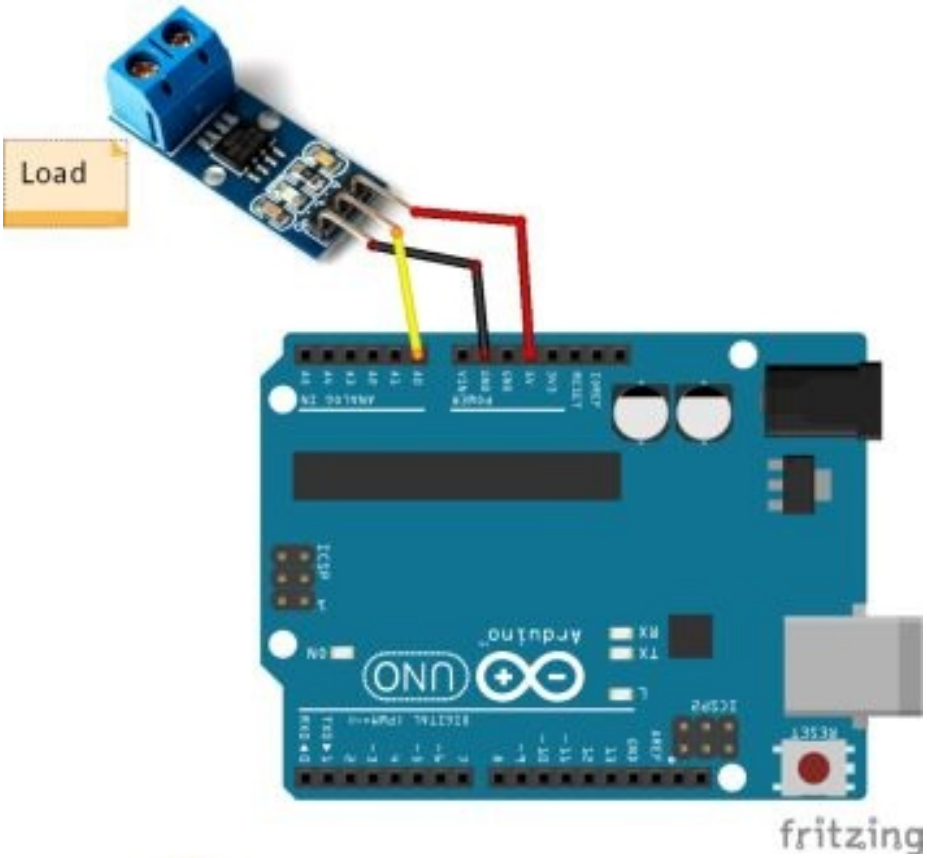
This particular board consists of ASC712 ELC-30, *i.e.* the range of this board is +/- 30A. The following image shows the components and pins on the board.

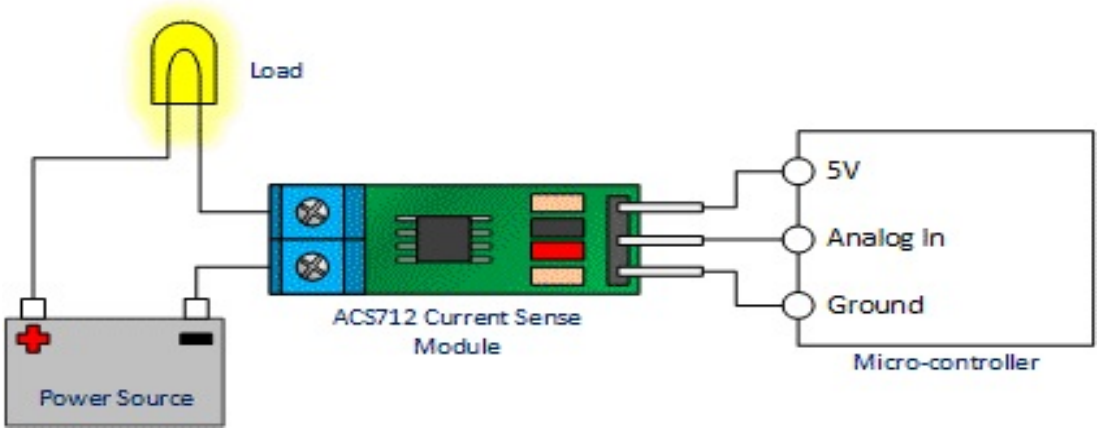


Interfacing ASC712 Current Sensor with Arduino Measuring voltages (DC Voltages) with Arduino is pretty simple. If your necessity is to measure less than or equal to 5V, then you can straightly measure using the Arduino Analog Pins. If you need to measure more than 5V, then you can use a simple voltage divider network or a voltage sensor module.

When it comes to measuring current, Arduino (or any other microcontroller) requires assistance from a dedicated Current Sensor. So, Interfacing an ACS712 Current Sensor with Arduino helps us in measuring current with the help of Arduino.

As ASC712 can be used for calculating either AC or DC currents, Arduino can be implemented to measure the same.

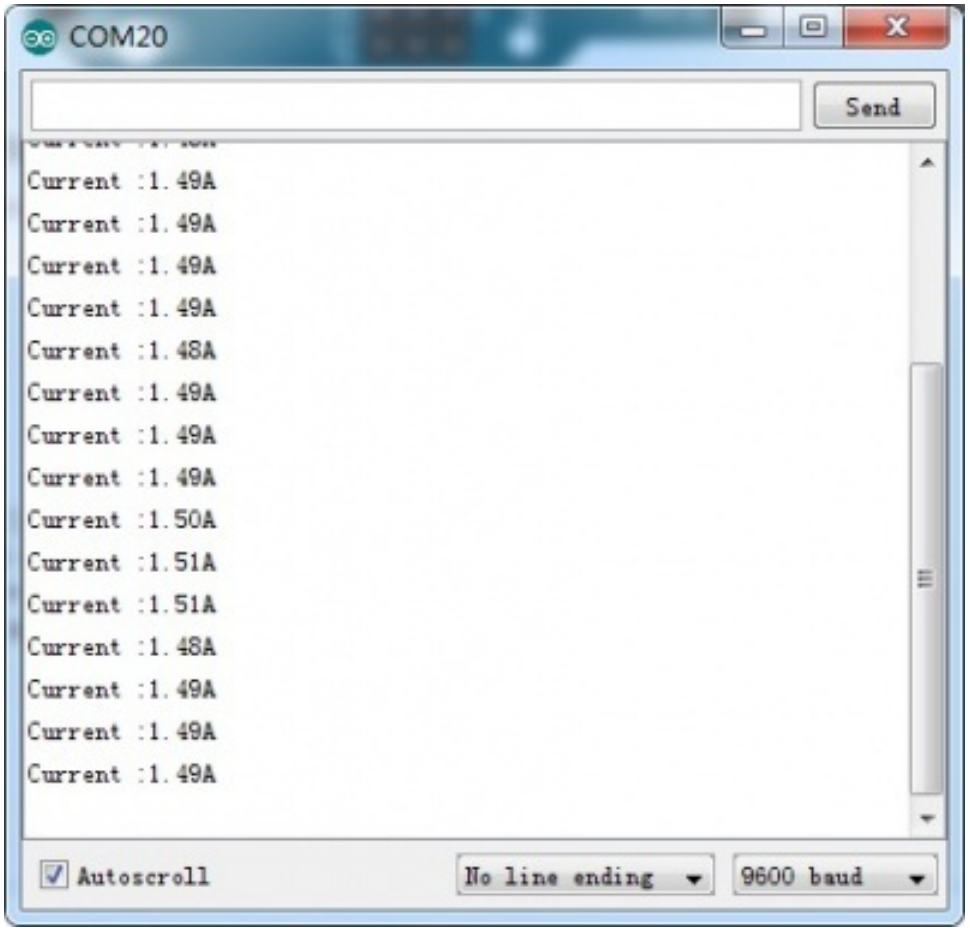

Load
fritzing


Load
5V
Analog In
Ground
ACS712 Current Sense
Module
Micro-controller
Power Source

Arduino Program

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

                                                }


    void loop() {
    float average = 0;
    for(int i = 0; i < 1000; i++) {
         average = average + (.0264 * analogRead(A0) -13.51);//for the 5A mode,
//     average = average + (.049 * analogRead(A0) -25);// for 20A mode //
average = average + (.742 * analogRead(A0) -37.8);// for 30A mode delay(1);

                                                }

    Serial.print("Current :");
    Serial.print(average/1000); Serial.println("A");

                                                }
```

Output:


COM20
Send
Current :1.49A
Current :1.49A
Current :1.49A
Current :1.49A
Current :1.48A
Current :1.49A
Current :1.49A
Current :1.49A
Current :1.50A
Current :1.51A
Current :1.51A
Current :1.48A
Current :1.49A
Current :1.49A
Current :1.49A
Autoscroll
No line ending
9600 baud

Measurement of Pressure using GY-68 BMP 180

The BMP180 barometric pressure measuring device is a great sensor that can be used to forecast the weather, sense altitude, and measure vertical velocity. It's perfect for weather stations, remote controlled vehicles, weather balloons, and lots of other projects. It's a sensitive sensor too.

WHAT IS BAROMETRIC PRESSURE?

Barometric pressure (also known as atmospheric pressure), is the pressure caused by the weight of air pressing down on the Earth. Imagine a column of air rising from the Earth's surface to the top of the atmosphere. The air in the atmosphere has mass, so gravity causes the weight of that column to exert pressure on the surface.

The pressure created by a 1×1 inch column of air reaching to the top of the atmosphere is defined as one atmosphere (atm) of pressure. This column of air weighs 14.7 pounds, which is why one atm equals 14.7 pounds per square inch (psi).

The SI unit for pressure is the Pascal (Pa). One Pascal is defined as one Newton of force per square meter. The BMP180 outputs pressure readings in Pascals, but they are converted to hectoPascals (hPa) by the software library we're going to use.

HOW THE BMP180 WORKS

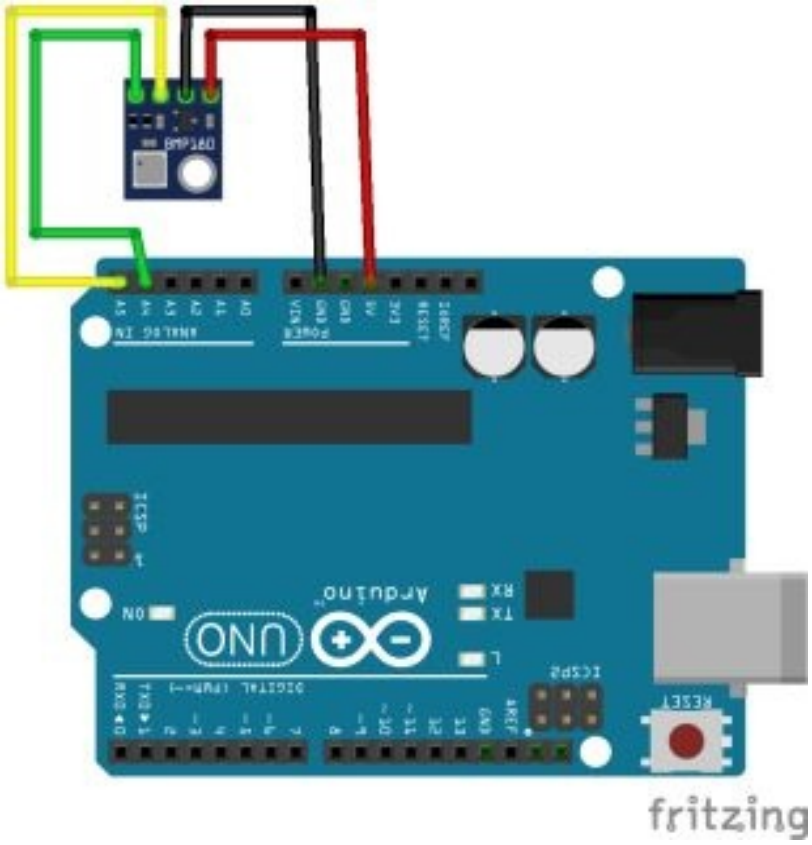
The BMP180 is a piezoresistive measuring device that senses pressure. The piezoresistive measuring device is made up of a semiconducting material (usually silicon) that changes resistance when a mechanical force like the atmospheric pressure is applied.

The BMP180 measures both pressure and temperature because temperature changes the density of gasses like air. At higher temperatures, air is not as dense and heavy, so it applies less pressure on the sensor. At lower temperatures, air is denser and weighs more, so it exerts more pressure on the sensor. The sensor uses real-time temperature measurements to compensate the pressure readings for changes in air density.

The BMP180 outputs an uncompensated temperature (UT) value and an uncompensated pressure (UP) value. The temperature measurement is taken first, followed by a pressure measurement.

CONNECTING THE BMP180 TO THE ARDUINO

The BMP180 communicates with the Arduino over I2C. The Arduino's I2C pins (SDA and SCL) are different depending on which Arduino you have.



Hardware Connections

The BMP180 measuring device has to be connected to Uno as follows: Vcc to 3.3V

Gnd to Gnd
SDA to A4
SCL to A5

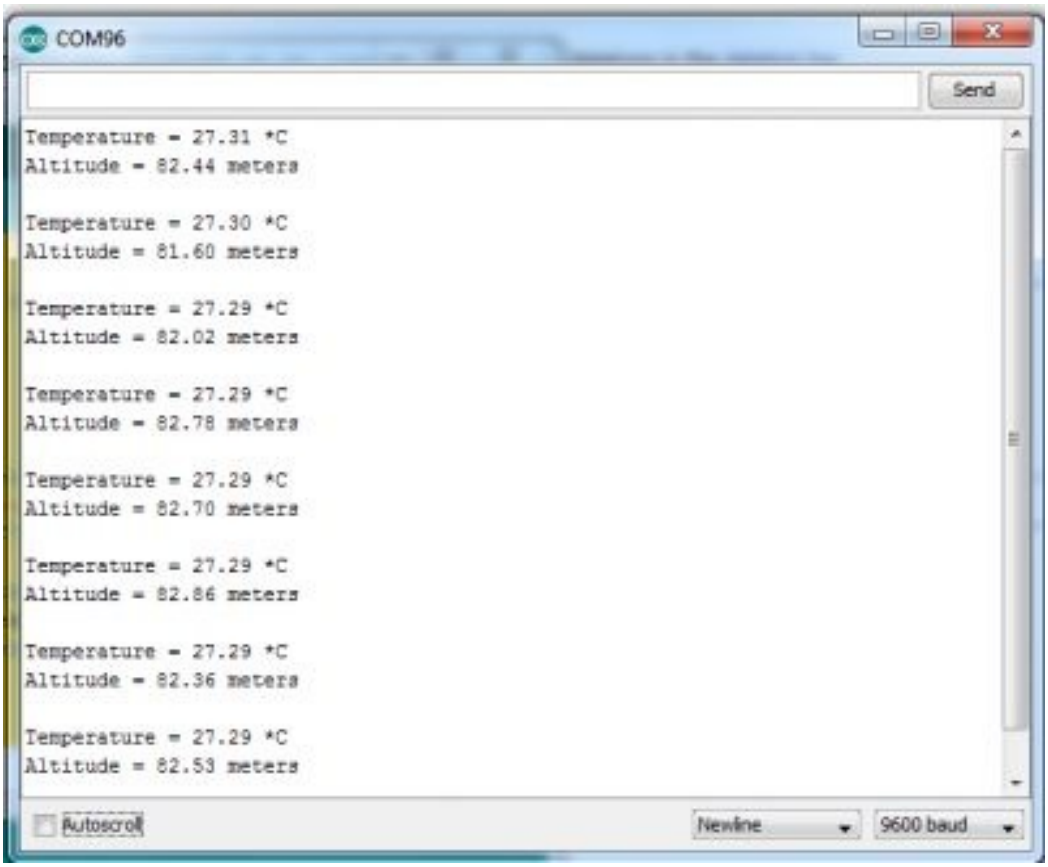
Arduino Program:

```
#include <Wire.h> #include <Adafruit_BMP085.h> Adafruit_BMP085 bmp; void setup()

{

Serial.begin(9600); if (!bmp.begin()) {
Serial.println("BMP180 sensor not found"); while (1) {}

}

}

void loop() {
    Serial.print("Temperature = "); Serial.print(bmp.readTemperature()); Serial.println(" *C"); Serial.print("Altitude = "); Serial.print(bmp.readAltitude(101500)); Serial.println(" meters"); Serial.println(); delay(1000);

}
```

This program requires Adafruit BMP180 library, you can add the library manually by adding the library from the source files, or you can download from the library manager.


Library Manager
Type All
Topic All
adafruit bmp
Adafruit BMP085 Library by Adafruit
A powerful but easy to use BMP085/BMP180 Library A powerful but easy to use BMP085/BMP180 Library
More info
Install
Adafruit BMP085 Unified by Adafruit
Unified sensor driver for Adafruit's BMP085 & BMP180 breakouts Unified sensor driver for Adafruit's BMP085 & BMP180 breakouts
More info
Adafruit BMP183 Library by Adafruit
Non Unified BMP library Non Unified BMP library
More info
Adafruit BMP183 Unified Library by Adafruit
Arduino library for the BMP183 sensors in the Adafruit shop Arduino library for the BMP183 sensors in the Adafruit shop
More info
Close

OutPut:


COM96
Send
Temperature = 27.31 *C
Altitude = 82.44 meters
Temperature = 27.30 *C
Altitude = 81.60 meters
Temperature = 27.29 *C
Altitude = 82.02 meters
Temperature = 27.29 *C
Altitude = 82.78 meters
Temperature = 27.29 *C
Altitude = 82.70 meters
Temperature = 27.29 *C
Altitude = 82.86 meters
Temperature = 27.29 *C
Altitude = 82.36 meters
Temperature = 27.29 *C
Altitude = 82.53 meters
Autoscroll
Newline
9600 baud

Measurement of GAS using MQ2 (Gas sensor Module)

The MQ series of gas measuring device uses a small heater inside with an electrochemical sensor. They are delicate for a range of gasses and are used in the house at room temperature. The output is an analog signal and can be read with an analog input of the Arduino.

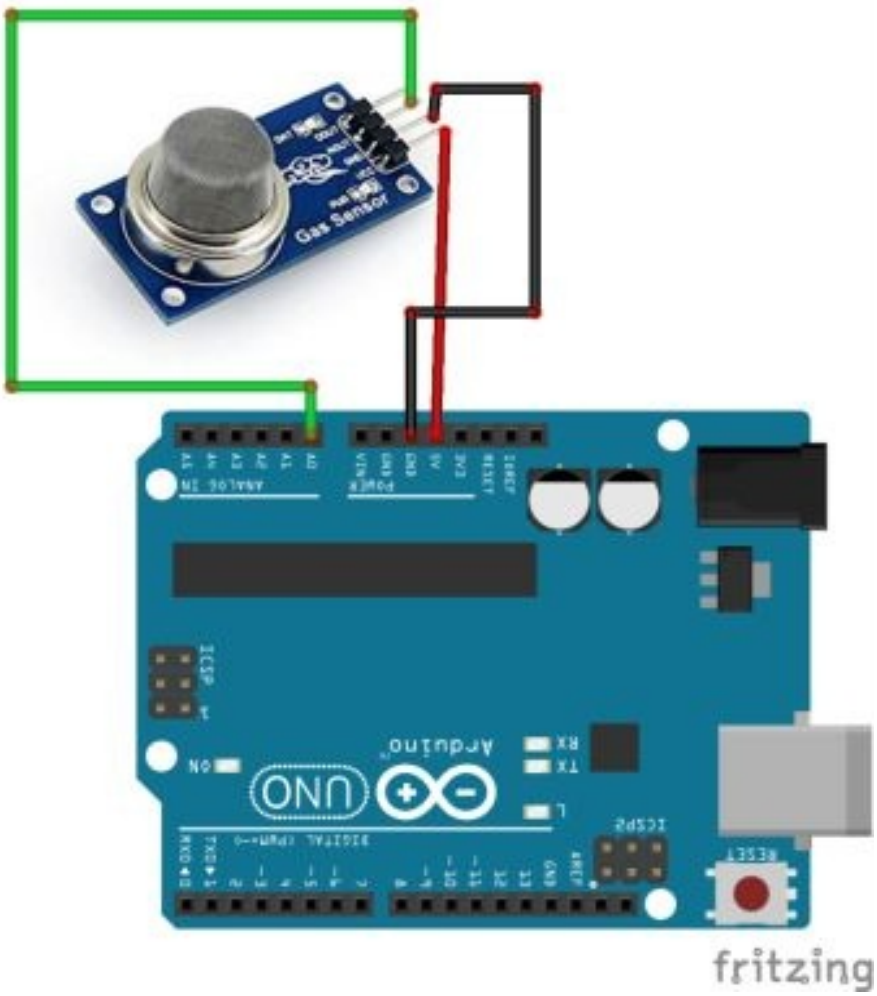
The MQ-2 Gas Sensor module is beneficial for gas leakage sensing in the home and industry. It can detect LPG, methane, alcohol, hydrogen, i-butane, propane, and smoke.

Some modules have a built-in variable resistor to regulate the sensitivity of the measuring device.

Note: The sensor becomes very hot after a while, don't touch it!

Components Required: 1) Arduino Uno 2) MQ-2 Gas sensor 3) Breadboard 4) Connecting Wires.

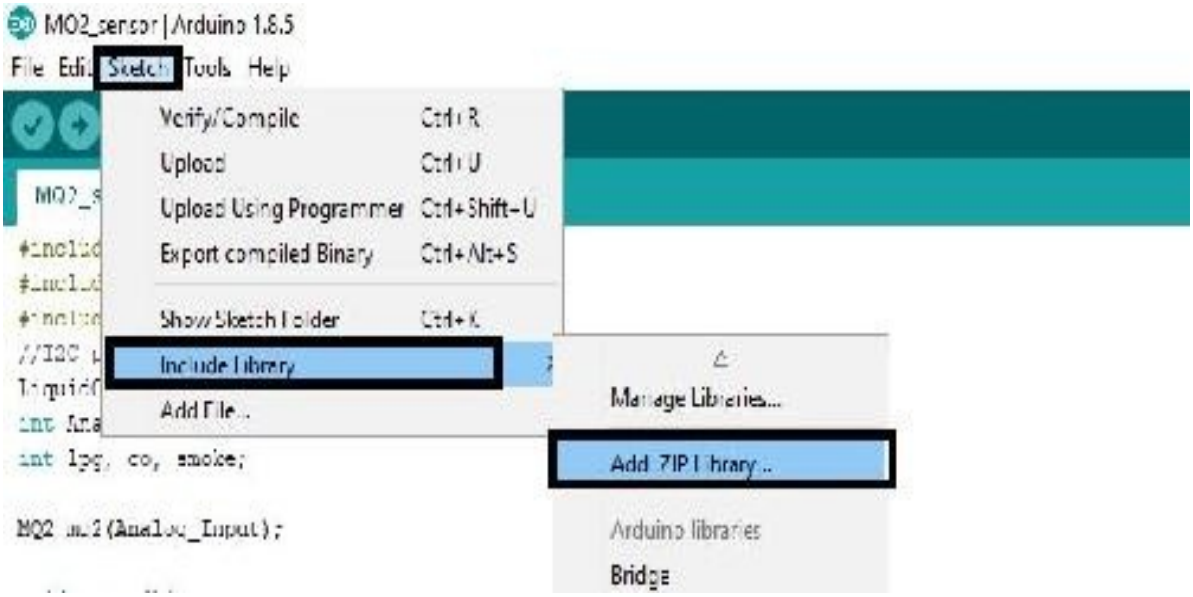
Circuit Diagram



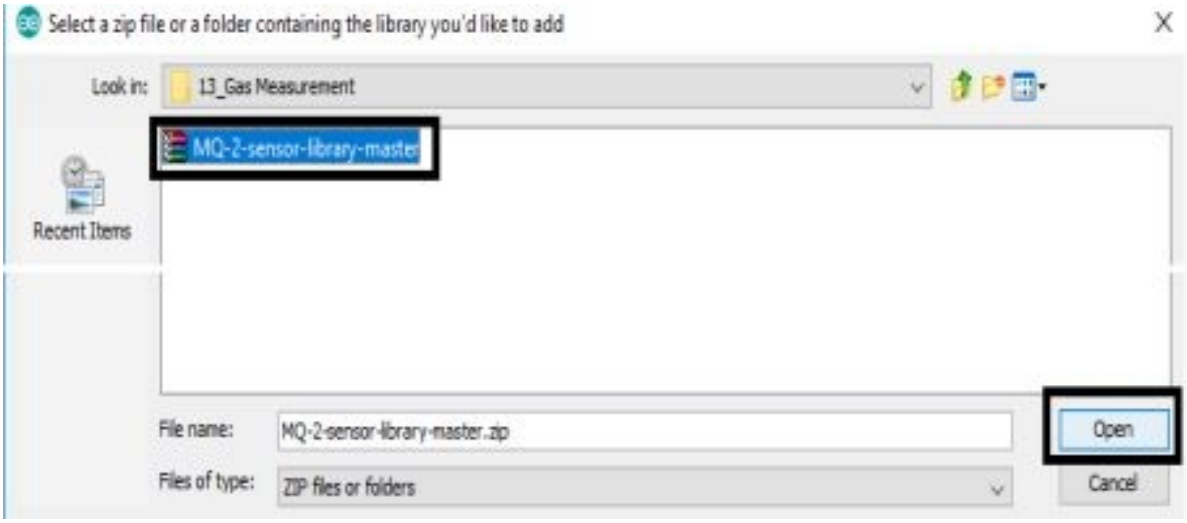
Circuit Connection

Arduino	**MQ2**
AO	A0
5V	VCC
GND	GND

Download the MQ-2 sensor library from the source file and add the zip file to your Arduino library.



Go to sketch and click include a library and select add a .zip library and navigate to MQ-2 sensor library and click open add it the Arduino library



Arduino Program

```
#include <MQ2.h> int Analog_Input = A0; int lpg, co, smoke;

MQ2 mq2(Analog_Input); void setup(){
  Serial.begin(9600); mq2.begin();

                                                    }

void loop(){
   float* values= mq2.read(true); //set it false if you don't want to print the values in the Serial //lpg = values[0]; lpg = mq2.readLPG(); //co = values[1];
    co = mq2.readCO(); //smoke = values[2]; smoke = mq2.readSmoke(); Serial.print("LPG:"); Serial.println(lpg); Serial.print(" CO:"); Serial.println(co); Serial.print("SMOKE:"); Serial.print(smoke); Serial.println(" PPM"); delay(1000);

                                                    }
```

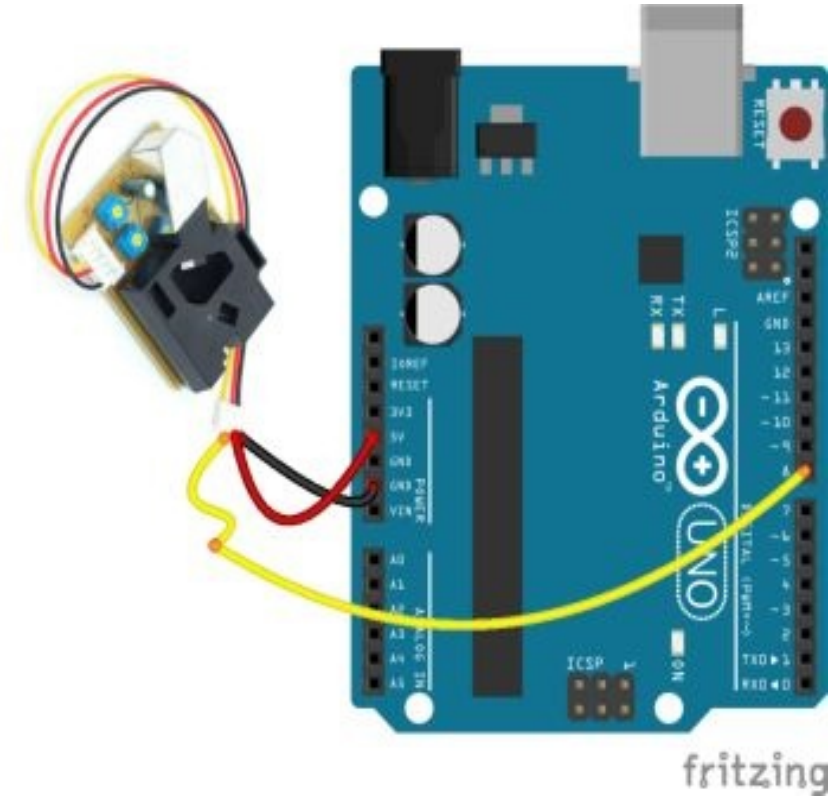
Dust Measurement using DSM501A Sensor

DSM501A is an optical air quality sensor, or may also know as dust sensor, is designed to detect dust particles. An infrared emitting diode and a phototransistor are diagonally arranged into this device, to allow it to detect the reflected light of dust in the air. It is particularly useful in detecting very fine particles like cigarette smoke and is commonly used in air purifier systems.

Components Required

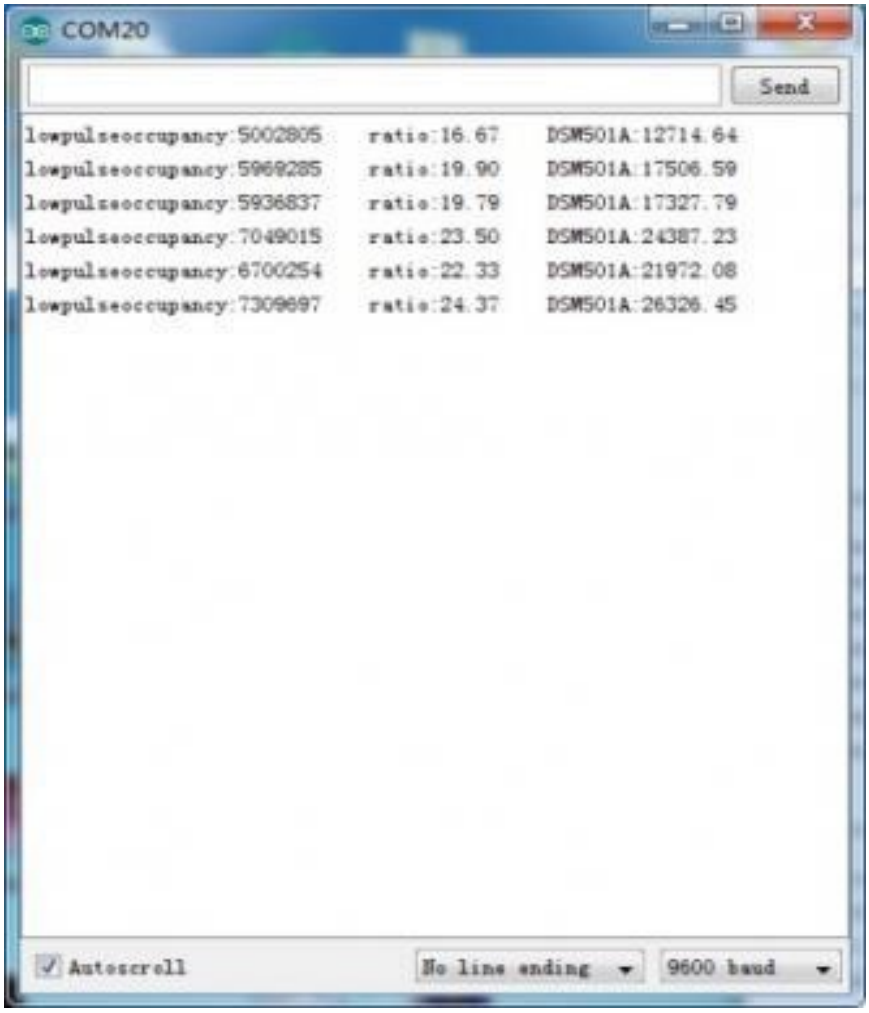
1) Arduino
2) Dust Sensor
3) Connecting Wires
4) Breadboard (Optional)

Circuit Connection



Arduino Program

```
#include<string.h> byte buff[2];
int pin = 8;//DSM501A input D8
unsigned long duration;
unsigned long starttime; unsigned long endtime;
unsigned long sampletime_ms = 30000; unsigned long lowpulseoccupancy = 0; float ratio = 0;
float concentration = 0; int i=0;
void setup()

                                        {

  Serial.begin(9600);
  pinMode(8,INPUT);
  starttime = millis();

                                        }

void loop()

                                        {

    duration = pulseIn(pin, LOW); lowpulseoccupancy += duration; endtime = millis();
  if ((endtime-starttime) > sampletime_ms) {
                ratio = (lowpulseoccupancy-endtime+starttime + sampletime_ms)/(sampletime_ms*10.0);  // Integer percentage 0=>100
    concentration = 1.1*pow(ratio,3)-3.8*pow(ratio,2)+520*ratio+0.62; // using spec sheet curve Serial.print("lowpulseoccupancy:"); Serial.print(lowpulseoccupancy); Serial.print(" ratio:"); Serial.print(ratio); Serial.print(" DSM501A:"); Serial.println(concentration); lowpulseoccupancy = 0; starttime = millis(); }

                                        }
```


COM20
Send
lowpulseoccupancy:5002805 ratio:16.67 DSM501A:12714.64
lowpulseoccupancy:5969285 ratio:19.90 DSM501A:17506.59
lowpulseoccupancy:5936837 ratio:19.79 DSM501A:17327.79
lowpulseoccupancy:7049015 ratio:23.50 DSM501A:24387.23
lowpulseoccupancy:6700254 ratio:22.33 DSM501A:21972.08
lowpulseoccupancy:7309697 ratio:24.37 DSM501A:26326.45
Autoscroll
No line ending
9600 baud

PDF link and Source Files of this book

http://bit.ly/ArduinoMeasurement

Google Drive Link:
https://drive.google.com/drive/folders/1EsDAE4-0Gpf_N2PzvwHCw1k5807ltdEA?usp=sharing

If you have any doubts regarding this book, please contact:

devtocontact@gmail.com

Send a mail to the above email and get a Raspberry pi Q&A for Free.