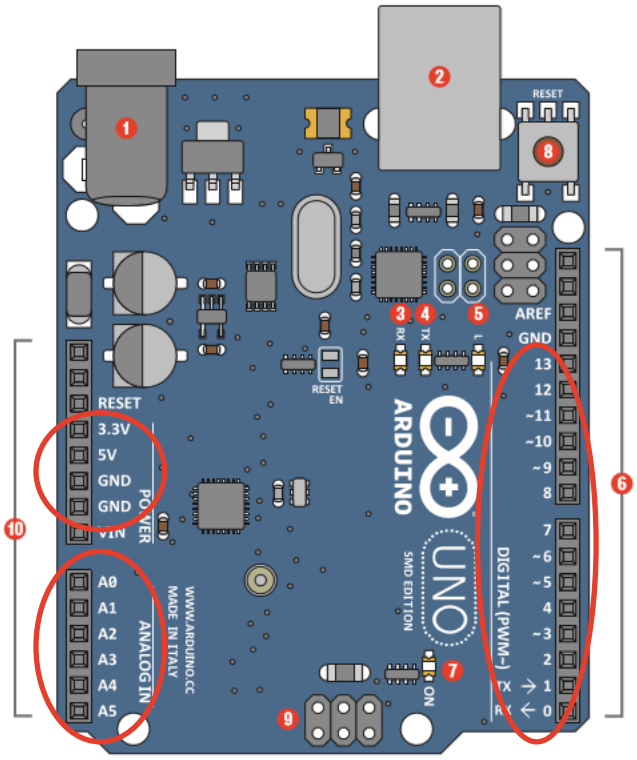
The Arduino Board



Arduino Uno

1	**Power In (Barrel Jack)** - Can be used with either a 9V or 12V battery pack.
2	**Power In (USB Port)** - Provides power and communicates with your board when plugged into a computer via USB
3	**LED (RX: Receiving)** - This shows when the Arduino is receiving data (such a being programmed)
4	**LED (TX: Transmitting)** - This shows when the Arduino is transmitting data.
5	**LED (Pin 13 Troubleshooting)** - This LED is connected to Pin 13 and can be used to troubleshoot code
6	**Pins** - Used for various inputs, outputs, power and ground. The ones circled are digital pins.
7	**LED (power)** - indicates if the Arduino is on
8	**Reset Button** - Can be used to manually reset the Arduino, making the code on it restart
9	**ICSP Pins** - used if you want to bypass the boot loader to upload code
10	**More Pins** - The upper circle are power pins, the lower are analog input pins.

An Arduino is essentially a small computer, often used for prototyping. It has a number of ports /pins that can be used as inputs - to take reading from sensors and switches - or as outputs - to drive lights and motors.

The Arduino is all open-source, meaning that all the hardware and software is freely available online.

Signals

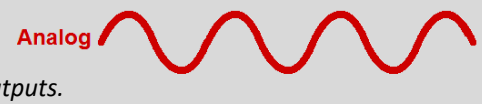
There are two kinds of signal used by the Arduino - Digital and Analog

Digital

Digital signals have two states - HIGH (1) or LOW (0). These are used to send signals in binary - 1's and 0's . This is called a square wave

Analog

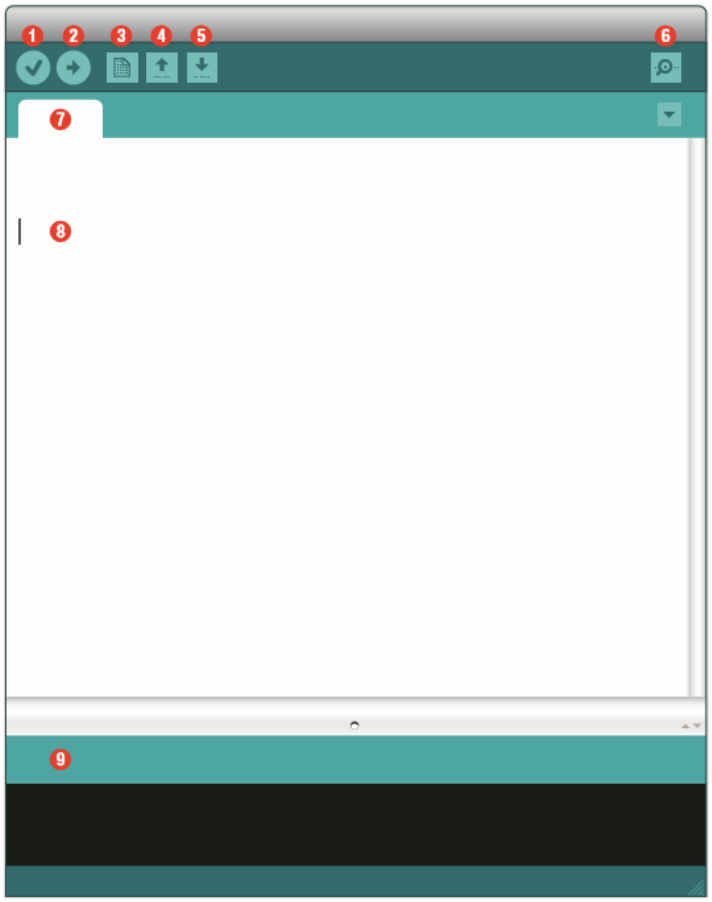
Analog signals cover a range of values to create waves of many shapes.



The Arduino has both digital and analog pins - useful for different sensors or outputs.

Figures and diagrams are sourced from the **SIK Experiment Guide for Arduino** and **Sparkfun Tutorials**

Programming an Arduino



Arduino IDE

(Integrated Development Environment)

1. **Verify**: Compiles the code and detects and errors in it. Will display messages in the message area.
2. **Upload**: Will upload your code to the Arduino. You should see lights flashing on the board. It will first compile the code.
3. **New**: Opens up a new code window.
4. **Open**: Will let you open another sketch.
5. **Save**: Lets you save the current sketch.
6. **Serial Monitor**: Will open a new window that shows and readout from the Arduino, useful for debugging.
7. **Sketch Name**: The name of the current sketch.
8. **Code Area**: This is the area where you compose code for your sketch.
9. **Message Area**: This is where any error messages or other messages are printed.

```
//General Outline of a Sketch

// variable names are set out here before the main functions

void setup(){

    // the setup function runs once when you press reset or power the board
    // it is used to initialize
}

void loop() {

    // the loop function runs over and over again forever
}
```

Arduino Reference Library: https://www.arduino.cc/en/Reference/HomePage

Syntax - The Grammar of Programming

"Let's eat Grandma!"

; (semicolon)	Every line must end with a semicolon.
{} (curly braces)	These are used in functions, loops and conditional statements. They are a way of containing everything within these structures.
// (single line comment)	Comments are used to document what code does. The syntax is used to tell the program that the line is not a part of the code.
/* */ (multi-line comment)	

The Arduino Process

1. Figure out what you want to do! What sensors / outputs do you need? How to you need to wire them? Can you adapt example code to do what you want? Design your project.
2. Set out the wiring. Figure out what each of your components need (Power? Ground? Signal?) and which pins you are going to use for this. Set it out on your breadboard.
3. Write the Program. In the Arduino IDE write your program - most of the time you'll be copy-pasting and changing values from example code. If you want to give yourself a challenge see if you can write it from scratch. Verify the program using the verify button. This checks that the program has correct syntax so the Arduino can understand it. If this isn't working see debugging.
4. Load your program onto the Arduino using the upload button. Now the computer is only powering the Arduino - the program is being run onboard. You can unplug the Arduino from your computer and plug in a battery pack.

Circuitry

Safety

The voltages and currents on the Arduino are not large enough to harm humans, but it is quite easy to blow up components or even the board itself.

The board is sensitive to static - handle it carefully and ground yourself before touching it.

The main reason for blowing up components is short circuits, over current or putting in components the wrong way round. Resistors can be used to limit current.

MORE INFO: *ww.rugged-circuits.com/10-ways-to-destroy-an-arduino/*

1. Units

Current	***Amps (A)***	Current is the flow of electrons (1 Amp is a certain number of electrons passing a point in a second).
Resistance	***Ohms (Ω)***	Resistance resists the flow of current (resistors are useful for limiting the current through a device such as a LED that will blow at too high a current).
Voltage	***Volts (V)***	Voltage is like water pressure - it is the force that "pushes" the electrons round a circuit. Without a voltage there can be no current.

2. Ohm's Law

Ohm's law gives the relationship between current, resistance and voltage

By covering the letter in the triangle you want, you are given the equation to find it:

$V = I \times R$ $I = V / R$ $R = V / I$

3. Conventions

Current flows from a high voltage to a low voltage

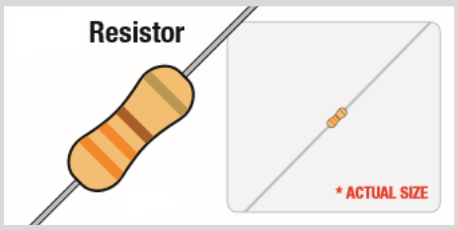
There are many different ways of saying this - it is also expressed as flowing from positive to negative. On the Arduino most pins have 3.3V, and Ground is 0V.

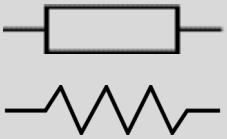
Electricity will always prefer the route of least resistance

If there is a choice between a path with a small resistance and a path with a higher resistance, more current will

Electrical Components

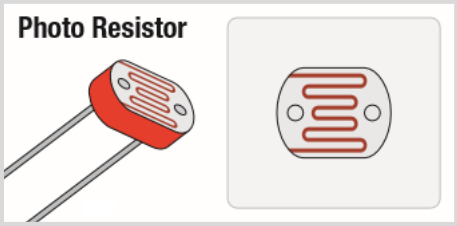
Resistors
(300Ω and 10kΩ)

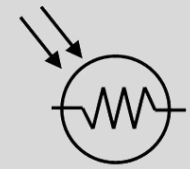




Resistors resist the flow of current in a circuit. There are two different commonly used symbols for resistors. Most resistors are made of a thin wire coil.

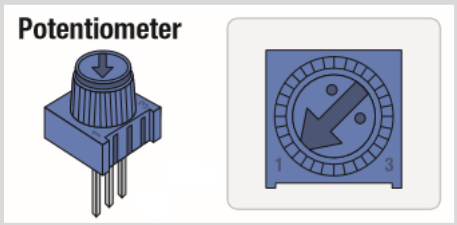
Light Dependent Resistor

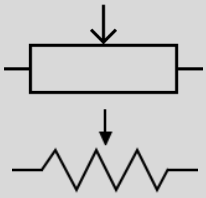




The resistance of a photo resistor depends on the light conditions. Some have high resistance when in bright light and low in low light, some are the other way round.

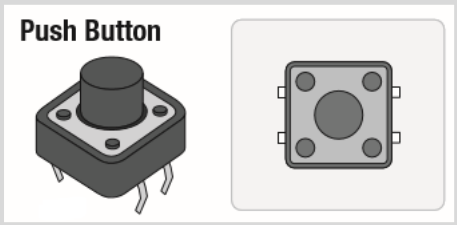
Potentiometer
(100kΩ)

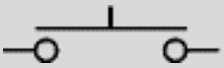




A potentiometer is a variable resistor. As you turn the dial the middle "sweeper" pin moves along the resistor, changing the resistance between it and the pins on either end.

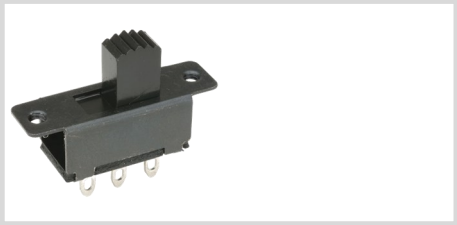
Tactile Button

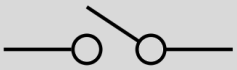




The parallel legs of a push button become connected when the button is pushed down.

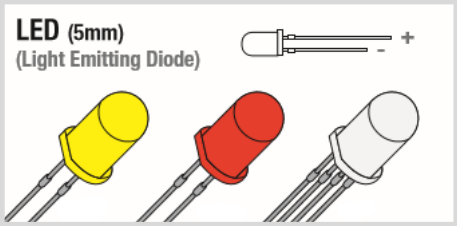
Small Slide Switch



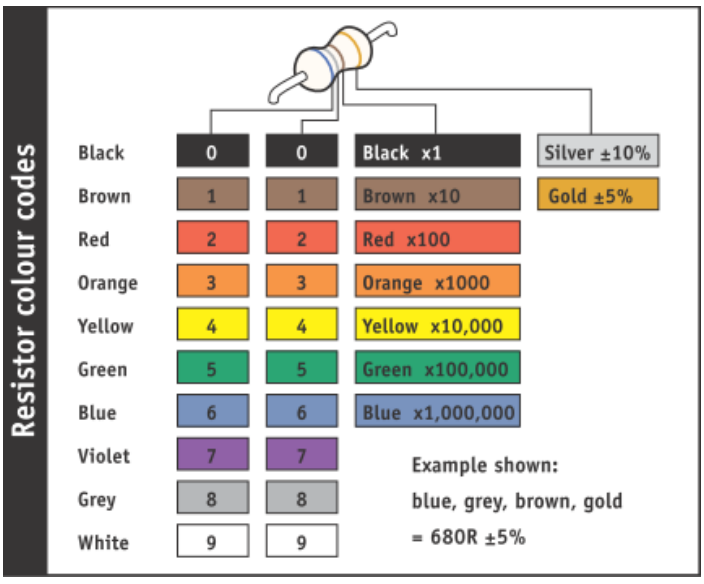


A slide switch has three pins - two of which are connected to each other at any time.

LED
(Light Emitting Diode)



A diode can only have current running through it from positive to negative, if current tried to go the other way it will be blocked.
An LED is a diode that emits light. Some have coloured plastic.



Resistors are labeled with a colour code that gives the value of the resistor - a key is shown on the left.

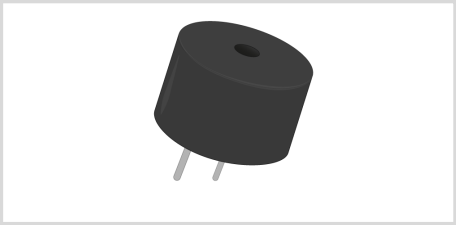
Reading from the left the first two digits are given and then a multiplier.

Eg) Red - Red - Brown
2 2 x10 = 22 x 10 = 220Ω

In reality resistors are not perfect. They are labeled with a tolerance - this is how much they can vary from the value given.

Electrical Components Cont.

Piezo Buzzer



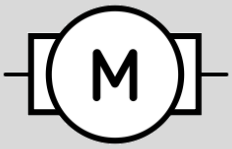
This can both detect vibrations and make sound.

This is because a piezo crystal coverts between electricity and physical vibrations.

Servo

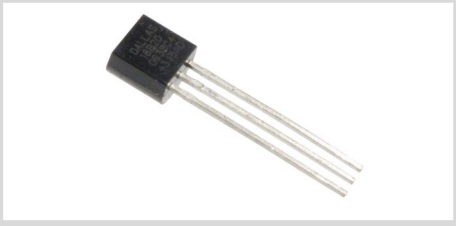
A position controlled electric motor. Most can only rotate 180° (they have a potentiometer inside that limits rotation but allows for position control).

DC Motor



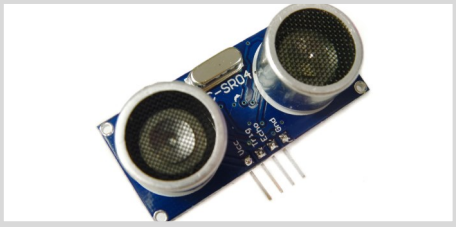
This is a simple motor. DC stands for Direct Current - it flows in a certain direction. Reversing the direction will change the direction the motor rotates.

Temperature Sensor



As the name would suggest, the signal from a temperature sensor varies as the temperature changes.

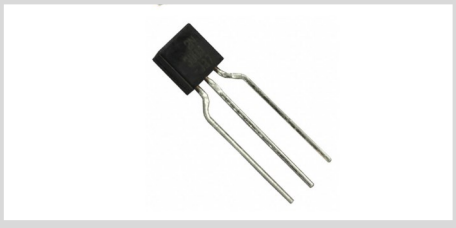
Ultrasonic Sensor

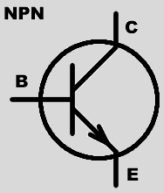


A ultrasonic sensor detects the distance to objects using sound.

It sends out a “ping” and then counts the time it takes to bounce back.

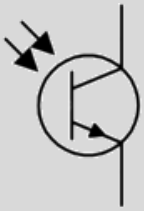
NPN Transistor





Transistors are used as switches and amplifiers. They are made of semiconducting material.

IR Phototransistor

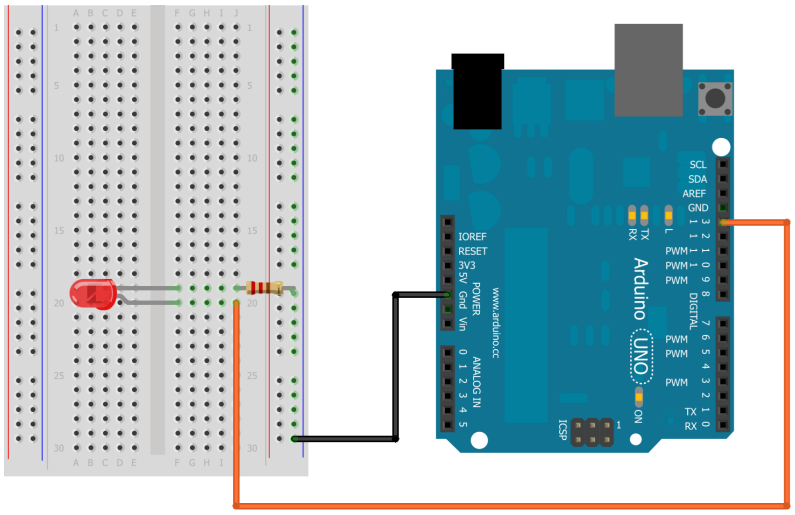


This is used together to detect Infra Red (IR) light - such as what is sent from a TV remote

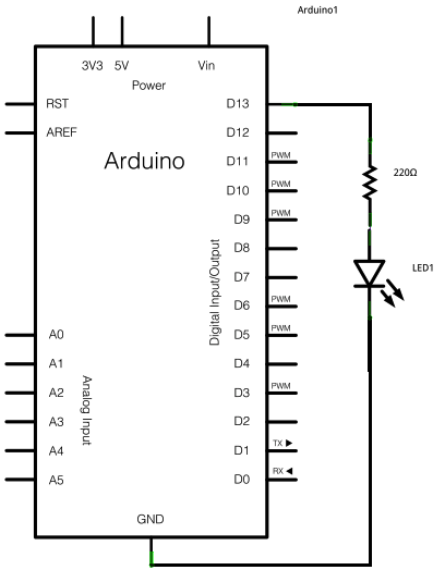
Blink an LED - “Hello World!”

Circuitry

Physical Layout



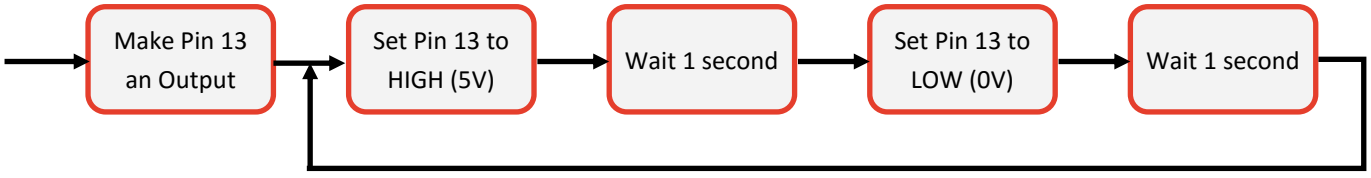
Schematic



NOTE: The way you control pins on an Arduino is by changing the voltage, HIGH is 5V, LOW is 0V

Programming

In the Arduino IDE Open: File > Examples > 01. Basics > Blink



New Functions

`pinMode(pin, mode);`	Used within `void setup()`. Sets the mode of a certain pin to **INPUT** or **OUTPUT** All pins must be initialized before use
`digitalWrite(pin, value);` *(Value is HIGH or LOW)*	Used to set the value (voltage) of a pin to **HIGH** (5V) or **LOW** (0V). In this case it is used to turn on or off an LED.
`delay( time in milliseconds)`	Basically a Pause function - the program will wait a set amount of time before moving onto the next command.

Variables

Programming

New Concepts: Variables

All variables MUST be declared.
This is so the program knows have much memory to put aside for variables before the program runs.

```
int ledPin = 13; // this initializes a variable and then sets its value.

pinMode( ledPin, OUTPUT );     // Using LedPin is exactly the same as using "13"
digitalWrite( ledPin, HIGH);

int catsAreTheBest; // a valid variable name can have no spaces ( useCamelCase
                    // or_underscores ), can't start with a number and can't be an
                    // already used word such as "int" or "for" etc…

catsAreTheBest = 2;       // I can change the value stored in the int
catsAreTheBest = ledPin; // Or assign the  value of another int to it

const int thisPin; // the const in front of this variable means it can't be changed
```

Maths!!!

The way Maths is written in computing is a little confusing.

```
x = x + 2; // this adds 2 to the variable x. The value "x + 2" is calculated and then
           // this is assigned to the variable x
```

Remember: "=" is always used to assign values, not to compare two sides of an equation or show that they are "equal"

In the same way we have:

```
x = x - 2; // minus 2 from the value "x"
x = x * 2; // multiply the value "x" by 2
x = x / 2; // divide the value "x" by 2
```

Since we often want to add or subtract one from a variable we have a shorthand for this:

```
x++; // this is the same as x = x + 1;
x--; // this is the same as x = x - 1;
```

```
int x = 0;

while( x < 5 ){
   //do something
   x++;

}
```

Counters

Say you want to do something 5, 10, 100 times in a program and then pause. Surely there must be a better way than writing it out that many times? YUP! Use a while loop (see next page) and a counter.

Conditional Statements

Programming

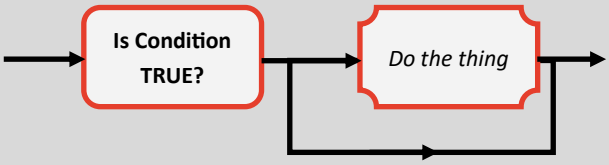
New Concepts: Conditional Statements (Control Structures)

Conditional Statements - Decision Making

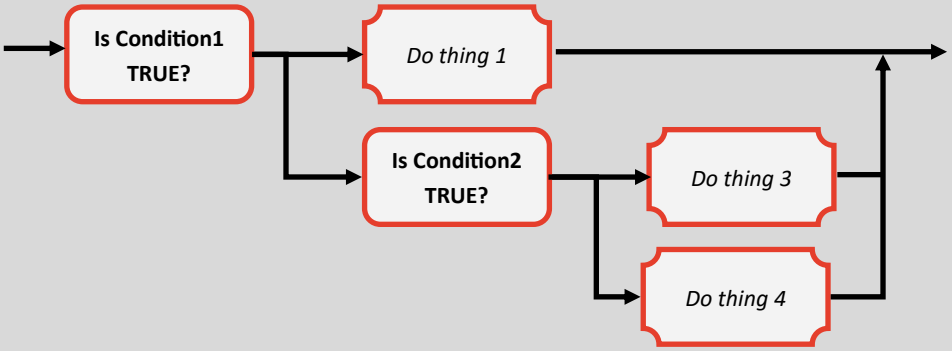
NOTE: "==" is used to compare values. When I write x == 2 will return TRUE if the value assigned t x is 2 and FALSE if it is not.

Control structures are how we make decisions in programs. There are three main ones you need to know - see the reference library for more.

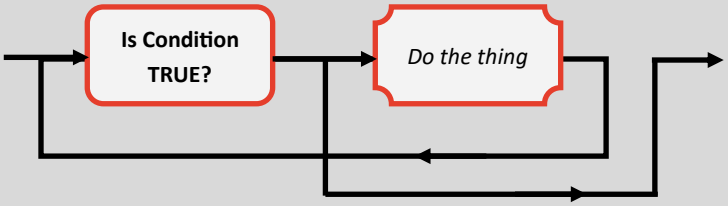
```
if( condition1 ){
  // do something
}
```




```
if( condition1 ){
  // do something1
} else if(condition2){
  // do something2
} else {
  // do something3
}
```




```
while( condition1 ){
  // do something
}
```



Using Conditions

In programming we often use "*Boolean Logic*" - which simply means that the conditions always evaluate to TRUE or FALSE. Conditions are often comparing two values, using the following comparators:

Boolean Condition (will be either TRUE or FALSE)

==	"is equal to" - (2 == 2) -> TRUE, (2 == 3) -> FALSE
< and >	"less than" and "greater than" ((2 < 3) -> TRUE, (2 > 3) -> FALSE, (2 < 2) -> FALSE)
<= and >=	"less than or equal to" and "greater than or equal to" ((2 <= 3) -> TRUE, (2 >= 3) -> FALSE, (2 <= 2) -> TRUE)
!=	"is not equal to" - (2 != 2) -> FALSE, (2 != 3) -> TRUE

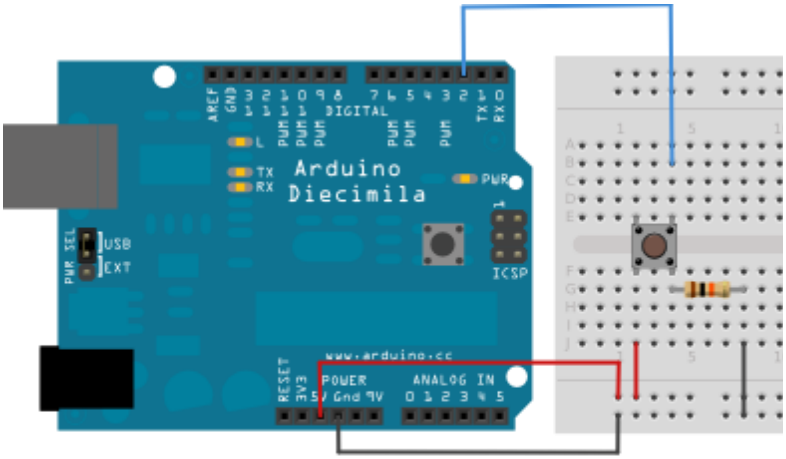
Multiple conditions can be used with && (and) as well as || (or)

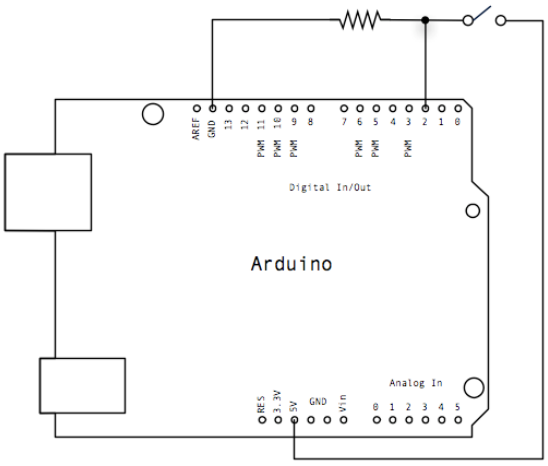
ie) if(x == 3 && y == 2) // will evaluate TRUE if x equals 3 AND y equals 2
if(x == 3 || y == 2) // will evaluate TRUE if x equals 3 OR y equals 2

Turn an LED on with a Button

Circuitry

New Concepts: Using a switch/button to control a Pin's Voltage





Why the Connection to Ground? (fun electrical trivia)

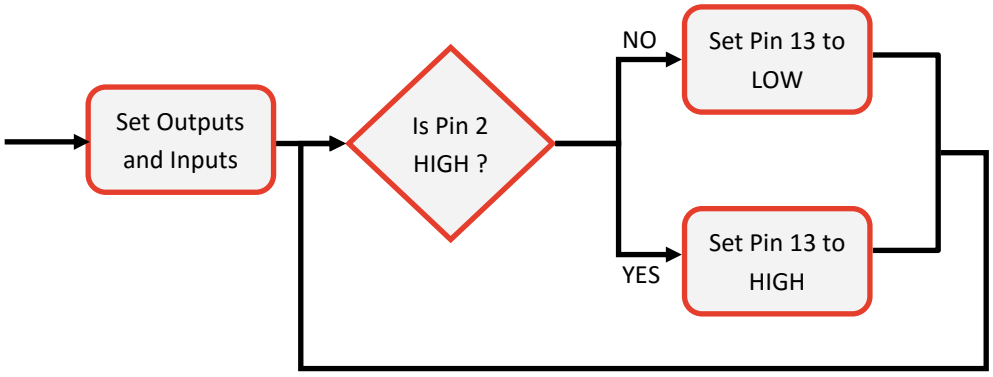
It seems like the circuit should work without the connection to ground - why do we need it?

If we didn't have it, when the switch is open pin 2 would just have a bit of useless wire coming out of it with no connection to power. This acts as a antenna - the radio waves in the air around us can actually create a current in this wire, making it occasionally read as "HIGH" even though the switch is open. The connection to ground keeps the state "LOW" when the switch is open, and the resistor means that when the switch is closed pin 2 will read as HIGH.

Programming

Using: Variables and Conditional Statements

In the Arduino IDE Open: File > Examples > 02. Digital > Button

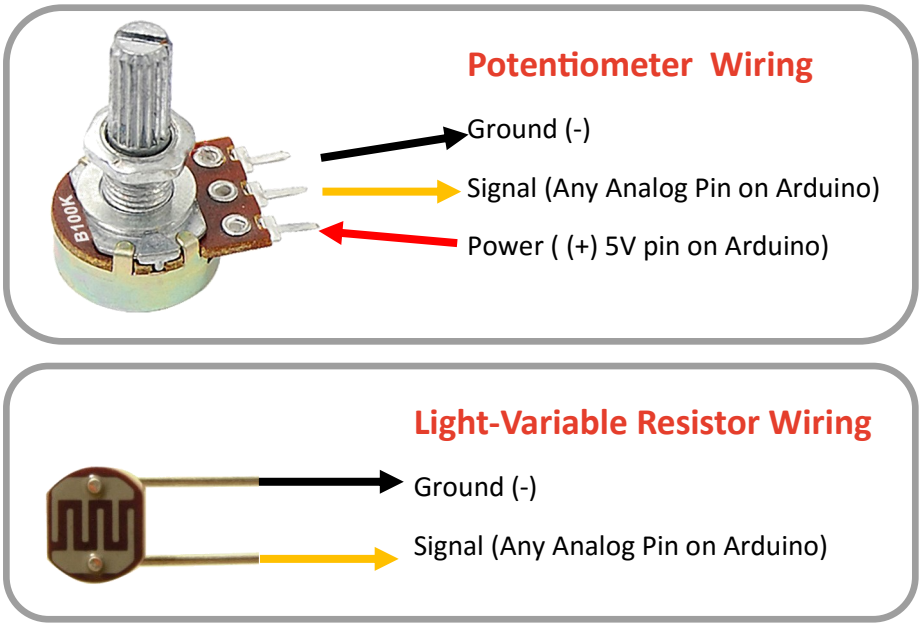


For this program we need to make a simple decision - if the switch is open (it's pin will read LOW) turn the LED off. If the switch is closed (it's pin will read HIGH) turn the LED on.

This can be done using an `if(){} else(){}` loop or two `if(){}` loops.

Analog Inputs

For this project we're going to read an analog input and print it on a screen (serial monitor). We're then going to use this input to change the brightness of an LED.



Programming

In the Arduino IDE Open: File > Examples > 03. Analog > AnalogInOutSerial

New Concepts: Analog Inputs

New Functions

`analogRead( [pin] );`	Will "read" the value of the pin and return a value between 0 and 1023
`annalogWrite(pin, value);` *(value is between 0 and 255)*	Used to set the value (voltage) of a pin to anything between 0 (0V) and 255 (5V).
`Serial.begin(9600);`	This begins communication. The number is the rate of info exchange (bits per second)
`Serial.print();` `Serial.println();`	This prints to the Serial Monitor. Anything in quotation marks will print exactly ie) "Hello World" Anything not quotation marks will be assumed to be a variable and the value of the variable printed. Ie) int x = 2; Serial.print(x); will print 2. /n creates a new line, /t is a tab (several spaces) Serial.println automatically prints a /n (newline) at the end

Turn a Servo

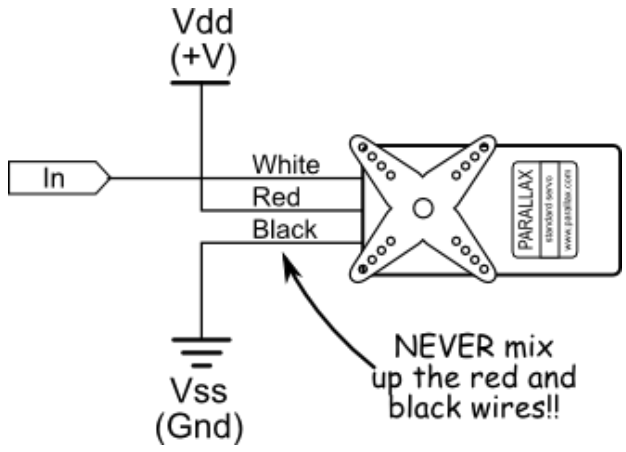
In the Arduino IDE Open: File > Examples > (scroll down) **> Servo > Sweep**
In the Arduino IDE Open: File > Examples > (scroll down) **> Servo > Knob**

New Concepts: Servos

The Servo Library

The functions for servos are stored in a separate library - this is why there is a `#include <Servo.h>` - this tells the program that it will need to go to that library to find some of the function you will use.

The Arduino reference page for this is: **www.arduino.cc/en/Reference/Servo**



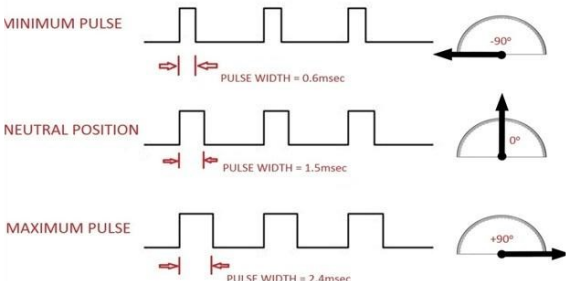
Wiring

RED - Power ((+) 5V pin on Arduino)

BLACK - Ground (-)

YELLOW - Signal (Any

DATASHEET



PWM - Pulse Width Modulation

The way the Arduino send the Servo an angle is through PWM.

The signal is encoded into the length of a pulse—for example, if the Arduino is sending 0.6msec pulses of 5V the servo will turn to –90° and if the Arduino is sending 2.4msec pulses the servo will turn to 90°.

MORE INFO: www.embedded.com/electronics-blogs/beginner-s-corner/4023833/Introduction-to-Pulse-Width-Modulation

Servo Functions *more at www.arduino.cc/en/Reference/Servo*

`Servo [name];`	Creates a servo which you can then attach to a pin, move to a degree etc…
`[name].attach( pin no.);`	Attaches the servo to a certain pin (require to then move the servo)
`[name].write( angle );`	Moves the servo to a certain angle (between 0 and 180°)
`[name].read();`	Read the current angle of the servo (the value passed to the last call to write() - you can't actually read the position back).
`map(low1, high1, low2, high2);`	This "maps" a value in a certain range to a value in a different range. (Not a specific servo function but useful for servos)

Move a Motor

Motor_On_Off.ino (source code provided)

New Concepts: Using a Transistor as an Amplifyer

Transistors

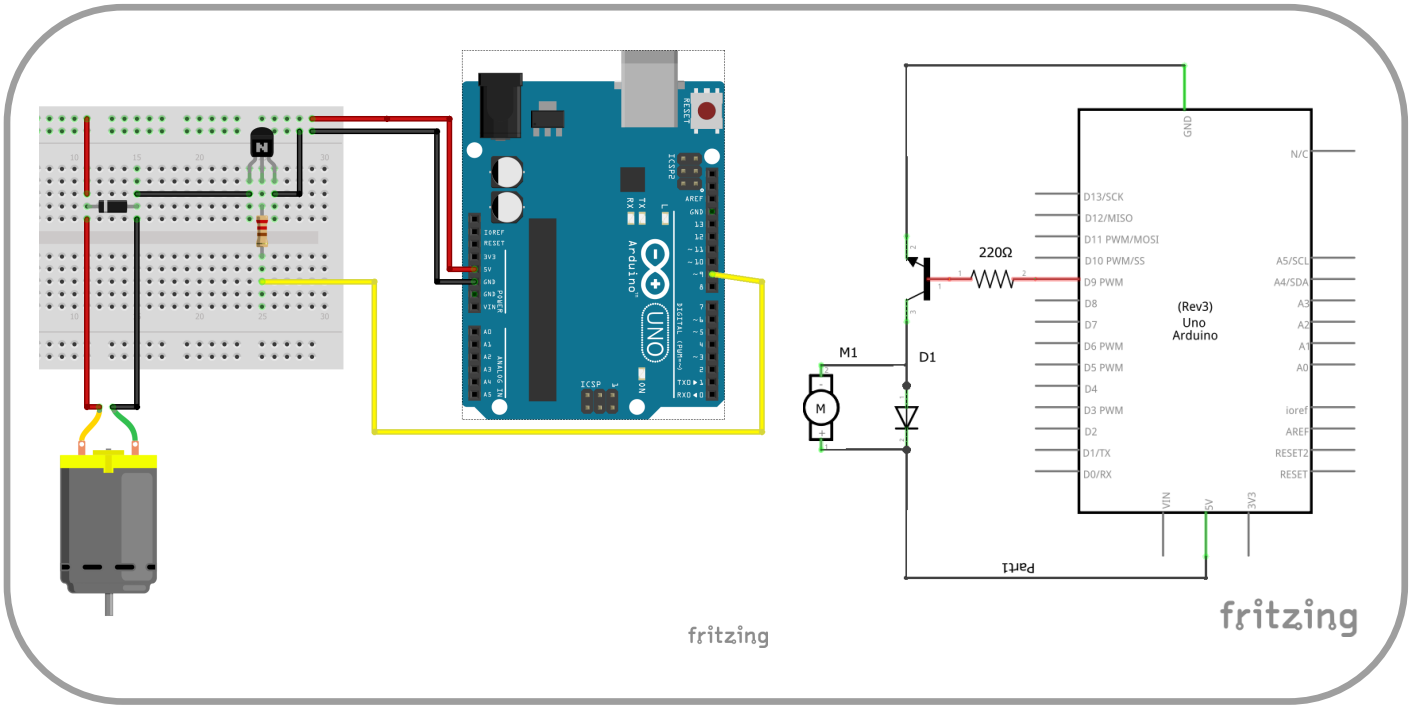
DON'T MIX UP THE TRANSISTOR AND TEMPERTURE SENSOR!!
The transistor should have CTBC 547B JS on it.

Transistors can be used as switch or amplifiers. In this case we're actually using it as an amplifier.

Non on the programmable pins on the Arduino have enough current to driver a motor, so we can't just plug the motor into ground and a pin and make it spin. However, the 5V pin on the Arduino does have enough current.

We put the transistor inn like we might put a switch in, but instead of having to physically press the switch to let current though we can use a signal to the middle pin of the transistor to control haw much current gets through. We are amplifying the low-current from a signal pin to a higher current that can drive the motor.

This setup does mean you can only rotate the motor in one direction.



Why the Diode?

All motors are also generators - if you spin a motor by hand it will actually create a small current. This also happens when the motor starts or stops suddenly. The diode stops current flowing backwards into the 5V pin , which can damage it

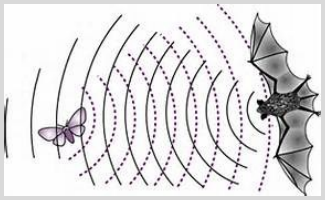
PROJECT SIMPLIFIED FROM:
learn.sparkfun.com/tutorials/sik-experiment-guide-for-arduino---v32/experiment-12-driving-a-motor

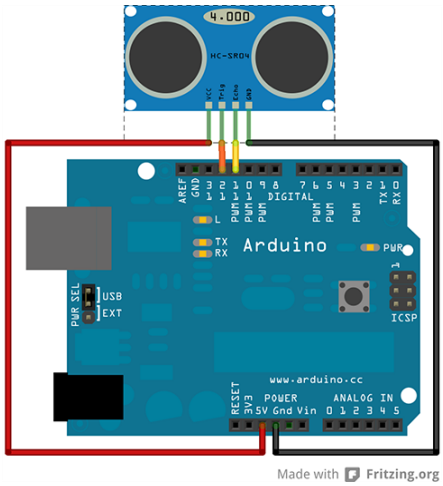
Measure Distance with an Ultrasonic Sensor

http://arduino.sundh.com/2014/03/ultrasonic-sensor/ (edited source code also provided)

Ultrasonic Sensors

Ultrasonic sensors use the same principle as bats use to hunt. They send out a out a pulse as a very high frequency (higher than humans can hear), and then time how long it takes the echo to get back. Using the speed of sound through air the distance to an object can then be found.





Wiring

RED - Power ((+) 5V pin on Arduino)

BLACK - Ground (-)

YELLOW - Signal (Any Digital Pin on Arduino)

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Temperature Sensor

Temp_Sensor.ino

New Concepts: Temperature Sensor

Temperature Sensor

Be careful not to get the temperature sensor mixed up with the Transistor - sensor should have **MCP 9700E** on it

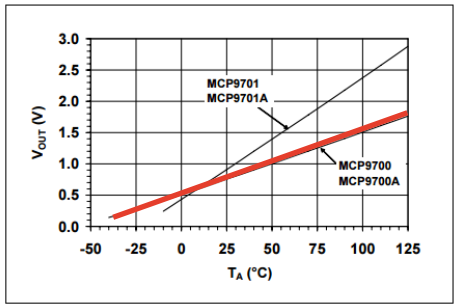
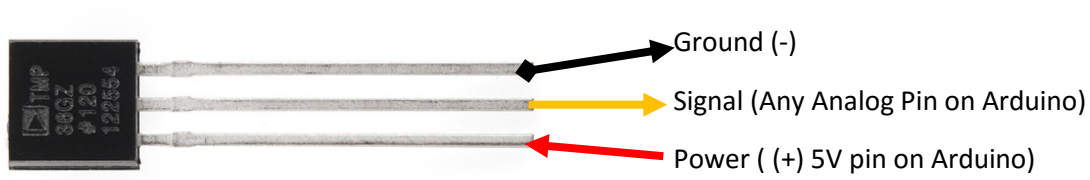


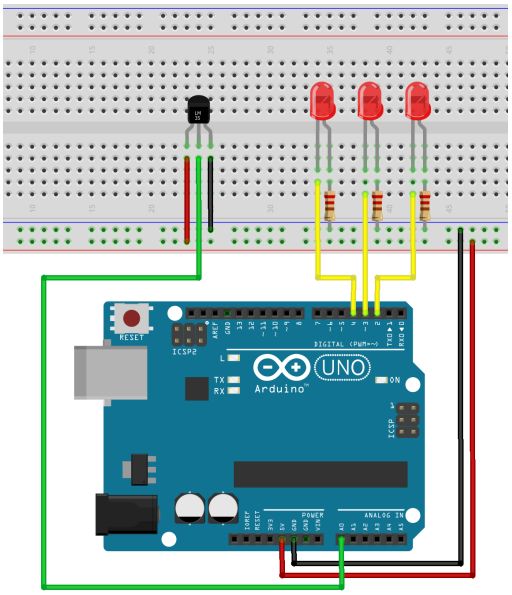
FIGURE 2-16: *Output Voltage vs. Ambient Temperature.*

Temperature Sensor Datasheet: www.sparkfun.com/datasheets/DevTools/LilyPad/MCP9700.pdf

On this datasheet we find a graph of the signal voltage against the temperature (I've coloured the correct one).

Wiring





Write an interesting program using this wiring

(if you need some help look at : File > Examples > 10. Starter-Kit_BasicKit > p03_LoveOMeter.

It uses this wiring but is also quite complex.

Buzzer

In the Arduino IDE Open: File > Examples > 02. Digital > toneMelody

Wiring

Ground (-)

Signal (Any Analog Pin on Arduino)

Music Functions

`tone(pin, frequency, duration);`	Plays a tone for a certain length of time
`noTone( pin );`	Stops a pin playing a tone

```
#include "pitches.h" //like servo.h -> this  means you can use NOTE_G3 etc...


// int name[] is called an array -> it  is a collection of integers. If I want
// the first element in the array I say name[0] (couting starts at zero in code),
// for the second name[1] etc...

// notes in the melody:
int melody[] = {
NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};


// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = { 4, 8, 8, 4, 4, 4, 4, 4 };
```

For Loops

(Another Conditional Statement)

For loops are just like the "counting" while loops mentioned earlier, just made more compact by bringing the initialization (x = 0) and the iteration (x++) inside the loop parameters.

Watch out for the semicolon (;) separating the parameters.

```
int x = 0 ;
while ( x < 3 ){
  //do something
  x++
}
```

```
int x;
for( x = 0 ; x < 3; x++ ){
  //do something
}
```

Programming Cheat - Sheet

https://www.arduino.cc/en/Reference/HomePage

```
//General Outline of a Sketch

// variable names are set out here before the main functions

void setup(){

    // the setup function runs once when you press reset or power the board
    // it is used to initialize
}

void loop() {

    // the loop function runs over and over again forever
}
```

Syntax

; (semicolon)	Every line / command must end with a semicolon.
{} (curly braces)	These are used in functions, loops and conditional statements. They are a way of containing everything within these structures.
// (single line comment) **/* */ (multi-line comment)**	Comments are used to document what code does. The syntax is used to tell the program that the line is not a part of the code.

Conditional Statements (Control Structures)

If( ){}	Will only execute the code in the curly brackets if the condition is TRUE
If( ) {} else if( ) {} else {}	A way of chaining if statements to be mutually exclusive
while(){}	While the condition is true the code will loop. Be careful of infinite loops.

Boolean Condition (will be either TRUE or FALSE)

==	"is equal to" - (2 == 2) -> TRUE, (2 == 3) -> FALSE
< and >	"less than" and "greater than" ((2 < 3) -> TRUE, (2 > 3) -> FALSE, (2 < 2) -> FALSE)
<= and >=	"less than or equal to" and "greater than or equal to" ((2 <= 3) -> TRUE, (2 >= 3) -> FALSE, (2 <= 2) -> TRUE)
!=	"is not equal to" - (2 != 2) -> FALSE, (2 != 3) -> TRUE
&&	"and" - used when multiple conditions are wanted
||	"or" - used when multiple conditions are wanted

Programming Cheat - Sheet

https://www.arduino.cc/en/Reference/HomePage

General Functions

`pinMode(pin, mode);`	Used within `void setup()`. Sets the mode of a certain pin to **INPUT** or **OUTPUT** All pins must be initialized before use
`digitalWrite(pin, value);` *(value is HIGH or LOW)*	Used to set the value (voltage) of a pin to **HIGH** (5V) or **LOW** (0V). Can be used to turn on or off an LED on a digital pin.
`annalogWrite(pin, value);` *(value is between 0 and 255)*	Used to set the value (voltage) of a pin to anything between 0 (0V) and 255 (5V).
`digitalRead( pin );`	Will "read" the value of the pin and return either **HIGH** or **LOW**
`analogRead( pin );`	Will "read" the value of the pin and return a value between 0 and 1023.
`delay( time in milliseconds)`	Basically a Pause function - the program will wait a set amount of time before moving onto the next command.
`map(low1, high1, low2, high2);`	This "maps" a value in a certain range to a value in a different range.

Servo Library

`Servo [name];`	Creates a servo which you can then attach to a pin, move to a degree etc…
`[name].attach( pin no.);`	Attaches the servo to a certain pin (require to then move the servo)
`[name].write( angle );`	Moves the servo to a certain angle (between 0 and 180°)
`[name].read();`	Read the current angle of the servo (the value passed to the last call to write() - you can't actually read the position back).

Music Functions Music Functions

`tone(pin, frequency, duration);`	Plays a tone for a certain length of time
`noTone( pin );`	Stops a pin playing a tone