ISC Arduino Tutorial 1

by **kenyesh** on May 11, 2015

Table of Contents

ISC Arduino Tutorial 1 1

Intro: ISC Arduino Tutorial 1 2

Step 1: Understanding the Arduino IDE 2

Step 2: Introduction to Basic Serial Communication 3

Step 3: Wiring an LED 4

Step 4: Make Arduino Blink It 6

Step 5: Inputs, Variables, and Switches 6

Step 6: Wiring an H-bridge and Driving Motors 8

Step 7: Analog Input 9

Step 8: PWM signals and controlling motor speed 10

Step 9: Writing Functions 10

Step 10: Writing Classes For the Motors 11

Advertisements 11

Intro: ISC Arduino Tutorial 1

This is a tutorial that strives to teach people the basics of Arduino. It was specifically developed to help train our new club members in basics of programming and robotics but we also plan to distribute to those we work with during our K-12 outreach giving them a means to continue their interests in robotics and programming on their own.

Note: Not complete currently published so I can get feedback from other people in my club

Parts:

For you convenience I have made a kit containing all of the parts in partnership with ClubJameco

http://www.jameco.com/webapp/wcs/stores/servlet/Pr...

- 1 Arduino
- 2 GM8 Motors
- 1 SN7544 H-bridge
- 10 LEDs
- 10 220 Ohm Resistors
- 10 10k Ohm Resistors
- 1 Half-size solderless breadboard
- 2 limit switches
- 1 Potentiometer
- Spool of wire
- 9 Volt Battery Snap

- 9 Volt Battery, Coming Soon To Kit
- USB Type A to Type B cable, Coming Soon To Kit

Example Code (Please Download For Later):

https://github.com/kyesh/ArduinoTutorial

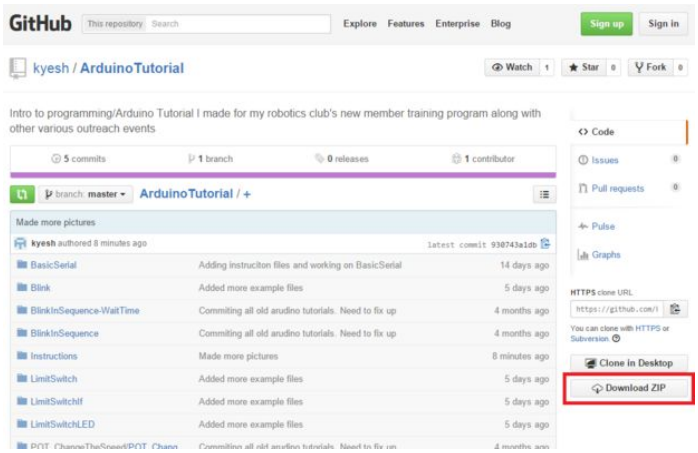
Tools:

Laptop

Imagination, creativity, and a desire to learn!

Arduino IDE (Please Install)

http://www.arduino.cc/en/Main/Software



Step 1: Understanding the Arduino IDE

After installing Arduino please open it up and refer to the photos above as I show you several key parts of the IDE (Integrated Development Environment).

1. The circle labeled with a 1 is circling the upload button. When your Arduino is plugged into the computer click this button to upload your code to the Arduino.

2. void setup is the section of your code that will run once and only once. When your Arduino is powered on or reboots it will run all the code you have placed in your void setup section

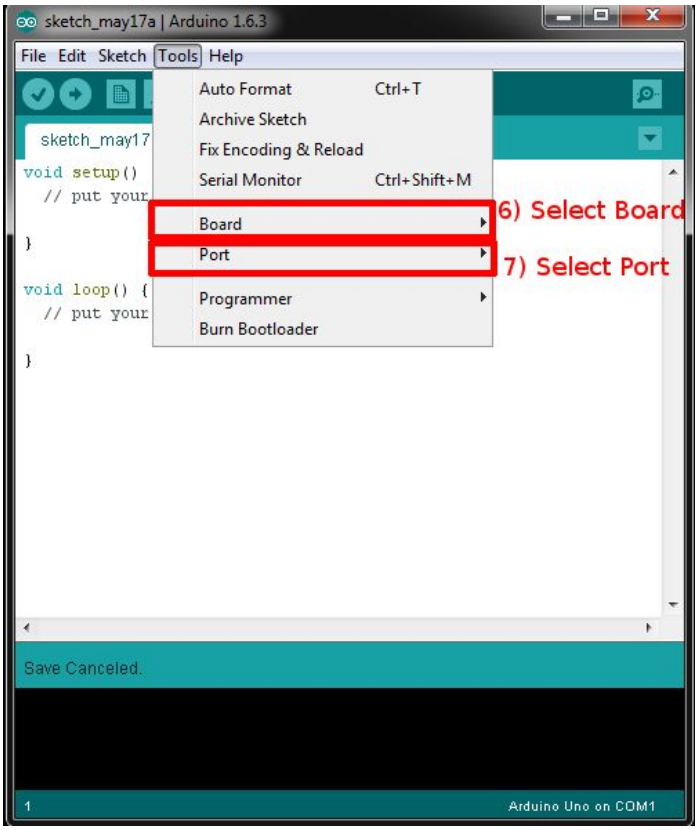
3. void loop is the section of your code that will run over and over again. After void setup is run, All your code in void loop will run. The difference is that after all your code in void loop is run, the Arduino will go back to start of your void loop section and run it all again. This repeats over and over again and thus is called a loop.

4. The serial Monitor button. This button opens up the serial monitor. The serial monitor allows you to send commands to the Arduino and receive messages back from the Arduino. More on how to do this later

5. This is the Arduino Console. Arduino will print out error messages here if it has issues uploading to the board or compiling your code.

6. In the Tools menu at the top there is a Board option that you can select. Use this to select the correct type of board. Usually the type of Arduino board is written on the device somewhere.

7. This is the option for selecting which com port connects your Arduino to the computer. If you have trouble figuring it out. Unplug your Arduino, look at the list, plug your Arduino back in. The new com port is the one for your Arduino.



Step 2: Introduction to Basic Serial Communication

Please open the "BasicSerial" example you downloaded from GitHub https://github.com/kyesh/ArduinoTutorial

This is the code I'll be explaining. There are only a few minor differences from this and the empty code editor.

Lines beginning with a double forward slash "//" are comments. Arduino will ignore them when compiling/running your code. This allows programmers to give information about how their program works to help themselves or other programmers figure out how the code works.

Next I would like you to read over the following links on the Arduino page. If you're new to programming don't be discouraged if they don't make any sense to you. I will re-explain them in simpler terms. You will need to get used to reading and understanding documentation like that as your programming skills develop.

http://www.arduino.cc/en/reference/serial

http://www.arduino.cc/en/Serial/Begin

http://www.arduino.cc/en/Serial/Println

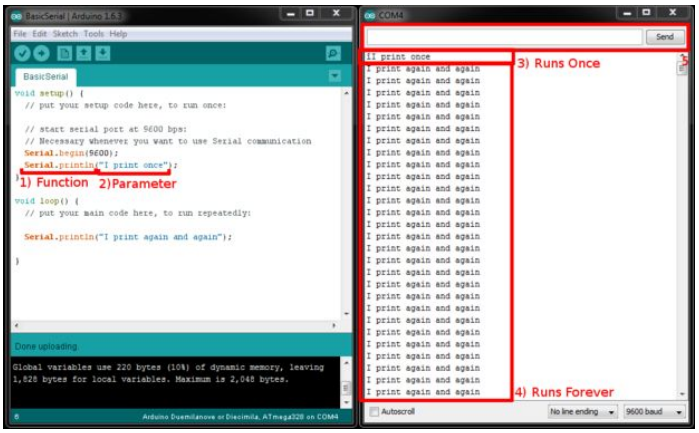
1. Serial.println() is called a function. Most things in programming involve using functions. Later in this tutorial you will learn how to write your very own functions! Serial.println() is a function that will print over the Arduino's serial line. When connected to your computer you can view what is printed by clicking on the serial monitor.

2. "I print once" is a parameter of the Serial.println() function. The parameter is your input to the function and in this case tells the function what to print. Try changing this to something else such as "I love hotdogs" and running the code again. Make sure to surround your words with quotations marks (“ “).

3. If you look at the first line displayed on the serial monitor it says "I print once". I on occasion you will get a funny character or two before the first line to print. Do not worry about this. Also notice how this is only printed once because it happens in the void setup portion of the code. Note: to see the first line you must un-check the box labeled Autoscroll and then scroll back to the top (Or have really fast eyes).

4. Notice how "I print again and again" continues to print over and over again. This is because it is in the void loop portion of the code. This code will continue to run over and over again until the end of the universe! Or until you disconnect/reprogram your Arduino.

5. This is the box allows you to send commands to your Arduino over serial. Examples could be if you had a robot to send command for drive straight, turn right, turn left. We will not be working with this but this is something cool you can explore in the future.



Step 3: Wiring an LED

So now let's control some electricity! To demonstrate our magnificent power to do this we're going to make an LED (a light) blink.

Let's start off simple by getting an LED to turn on.

To do this you will need 1 LED, 1 220 ohm Resistor, 1 bread board and some wire.

1. LEDs are a Light Emitting Diode. Diodes are a device which electricity can only go through them one way. The shorter side of the LED is the negative side. You will be plugging this into the negative rail of the bread board.

2. We're going to use the Arduino as the power source for our LED. This means you need to have it plugged into your computer to work. Every Arduino is arranged differently but the pins should be labeled. I have marked where the 5 volt and GND (0 volts) are on my board.

2-1. 5V this is the five volt source on your Arduino. This needs to be wired to the Positive rail on your bread board.

2-2. GND this is where the electricity comes back to after it's done doing work also known as 0 Volts. This needs to be wired to the negative rail of your bread board.

3. This a bread board. It's very helpful for wiring up and prototyping circuits.

3-1. Negative rail -This is the rail with a Minus sign over it and a blue stripe next to it. This is typically used as a ground rail. Everything connected to one point on the rail is connected to every point on the rail. There are two Negative rails on the breadboard pictured. One on the left side of the board and one of the right side of the board.

3-2. Positive rail - This is the rail with a Plus sign over it and a red stripe next to it. This is typically used as a positive voltage rail, in our case 5 volts. Everything connected to one point on the rail is connected to every point on the rail. There are two Positive rails on the breadboard pictured. One on the left side of the board and one of the right side of the board.

3-3, Rows, are like mini rails. They connect going horizontally across the board. For example row 1 connects 1a through 1e together. The big gap in the board indicate the 1f through 1j are connected together but not to 1a through 1e.

Wiring this all together

1) Connect 5V to the Positive rail of the breadboard

2) Connect GND to the Negative rail of the breadboard

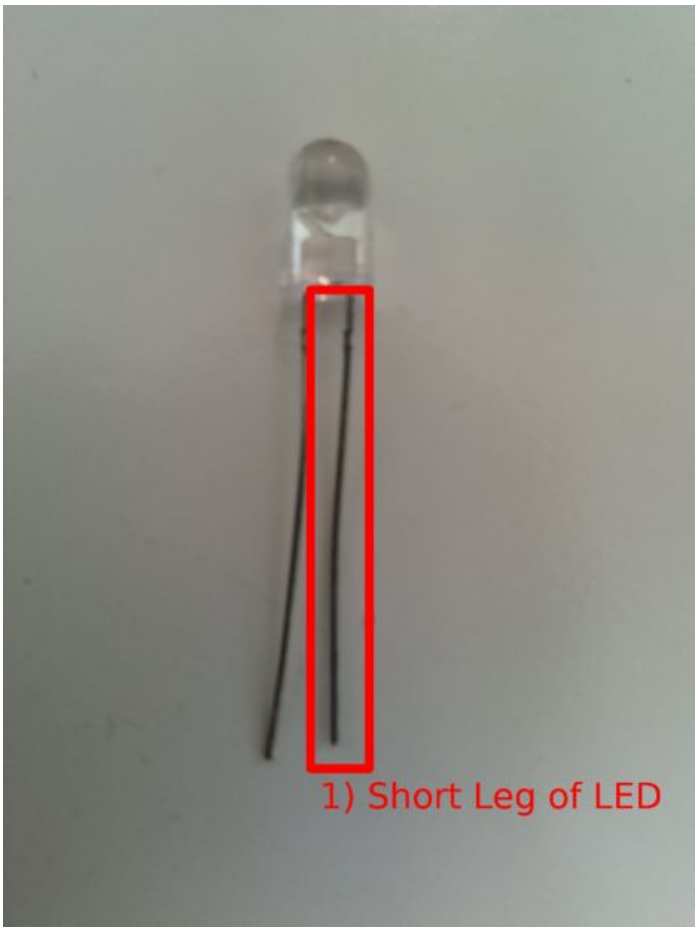
3) Connect the short end of the LED to the negative rail of the breadboard

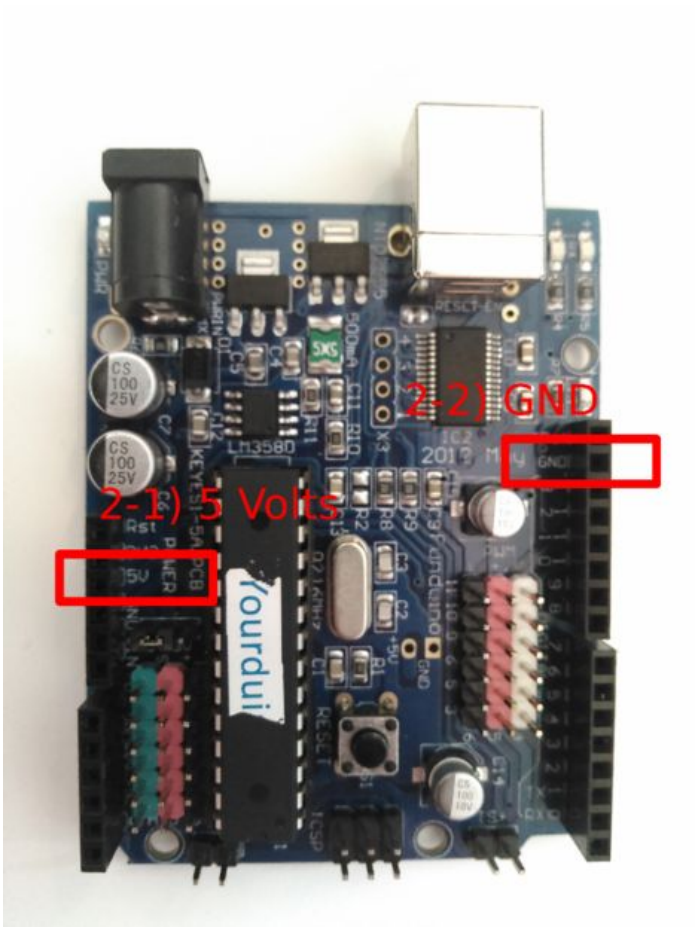
4) Connect the long end of the LED to 1a

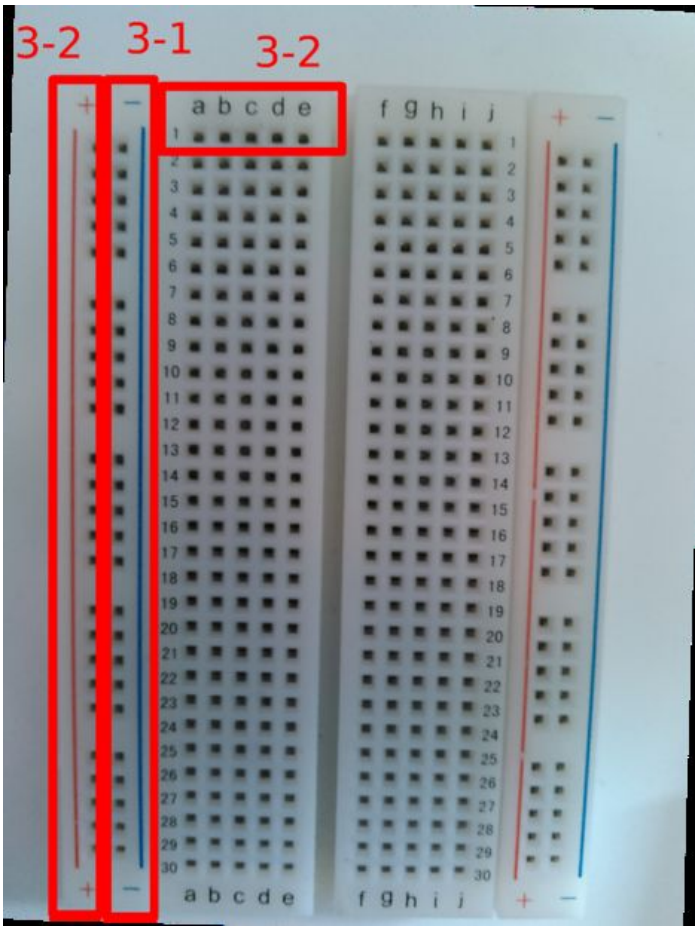
5) Connect the resistor from 1c to the positive rail

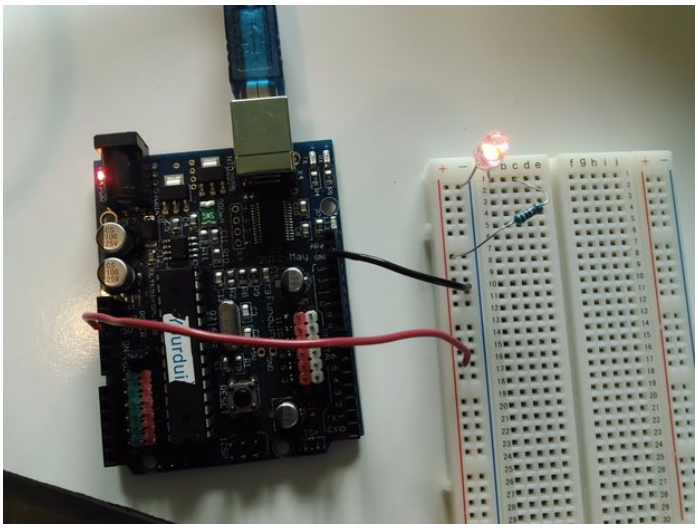
6) Appreciate the fine glow of your LED (If LED not glowing trying flipping it around. You probably got the ends mixed up)

Now to make it blink move the resistor from the Positive rail to the Negative rail and then back to the Positive rail once a second. I'm sure as you quickly realized this is extremely tedious and you'd much rather not have to manually move the resistor. Well your're in luck! In the next step we'll show you how to get your Arduino to do all this hard work for you!


1) Short Leg of LED


2-2) GND
2-1) 5 Volts


3-2
3-1
3-2



Step 4: Make Arduino Blink It

Next we'll have the Arduino do the blinking for us.

First you must move the resistor from the rail and connect it to pin 13 on the Arduino. It should be labeled with a 13 next to the pin.

As usual I highly recommend you try to read and understand the documentation for the functions being used before I give you the simplified explanation.

http://www.arduino.cc/en/Reference/PinMode

http://www.arduino.cc/en/Reference/DigitalWrite

http://www.arduino.cc/en/Reference/Delay

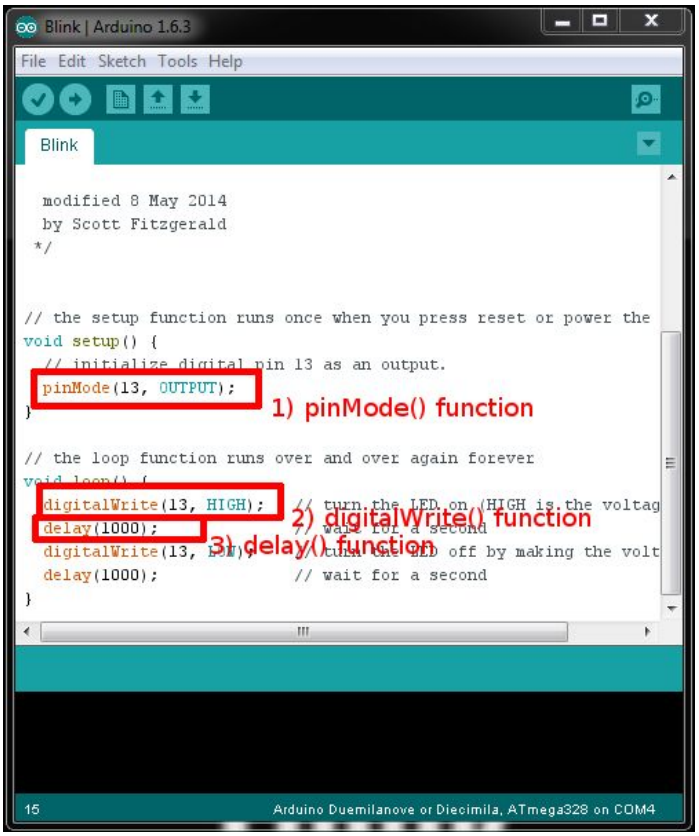
Please open the blink example

1. pinMode() is a function that says what you want to do with the pin. The first parameter is the pin you are talking about. In this case pin 13. The second parameter is OUTPUT this is saying you want to be able to set pin 13 between 0 Volts and 5 Volts in your program.

2. digitalWrite() is a function that sets the specified pin to LOW (0 Volts) or HIGH (5 Volts). The first parameter is 13 the pin we want to control. The second parameter is either HIGH or LOW. Setting it to HIGH is like when you were connecting the resistor to the positive rail. Setting it to LOW is like when you connected it to the negative rail.

3. delay() is a function the stops the Arduino for a set amount of time. Think of this as when you were waiting one second between going switching rails. This waits the amount of time specified in milliseconds or 1/1000 seconds

Challenge: Try making the light stay on for 2 seconds and off for .5 seconds



Step 5: Inputs, Variables, and Switches

Wiring the circuit.

Look at the fritzing diagram to wire the circuit.

The resistor used is a 10k resistor. This is called a pull down resistor because it pulls the voltage back down to 0 Volts when the switch is open (not pressed). When the switch is closed (pressed) the resistor is so large it has little effect on the voltage and so it is 5 volts.

Open up the file called LimitSwitch with arduino.

As usual read over the documentation before I give you the simplified explanation.

http://www.arduino.cc/en/Reference/DigitalRead

1. This command creates a new variable named pin12Value. The part in blue "int" specifies the type of variable. int is short for integer. Arduino needs to know what type of variables it's working with to efficiently store them in memory and do operations on them such as addition, subtraction, multiplication, and division. The second part is the name of the variable and how you refer to it later in your code.

2. digitalRead() this function returns a 1 (HIGH) or a 0 (LOW) depending on if there is 5 Volts applied to the pin or 0 Volts applied to the pin it is reading.

3. This next part is called an assignment. We are setting pin12Value to a 1 or a 0 depending on whether or not the limit switch is pressed and pin 12 is High.

4. Now we use Serial.println(pin12Value) to print the value of pin12Value. If you observer the serial monitor as you push the limit switch you should notice it go from 0 to 1.

So, this would look a lot nicer and easier to understand if HIGH and LOW were printed instead of just 1s and 0s. We can accomplish this using if statements!

Try to read over and understand the documentation on if statements below before preceding to my explanation.

http://www.arduino.cc/en/Reference/If

Open LimitSwitchIf in Arduino and run it

1. This is an if statement. If the condition is true then the code inside the curly brackets will run. If the condition is not true the code in the curly brackets will be skipped.

2. This is the condition of the if statement. This is asking a question, does pin12Value equal 1? If the answer is yes then the expression evaluates to true. Be careful and realize that the == is different than the = sign. == is asking a question. Are they equal? = is a command setting whats on the left side to be equal to what's on the right side.

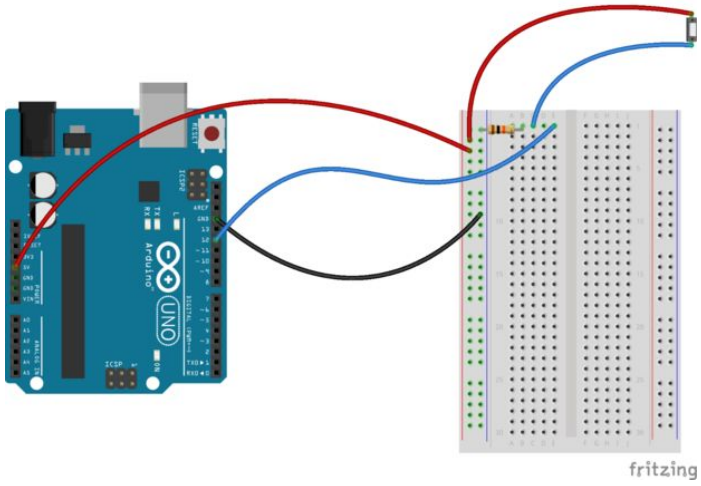
3. This is the code in the curly brackets that runs when the statement is true.

Challenge: Use what you've learned to have the LED turn on when the switch is pressed and off when it's released

To see how I accomplished the challenge open LimitSwitchLED

Challenge2: Use what you've learned to have 2 LEDs each turned on by a different switch.

To see how I accomplished the challenge open LimitSwitchLED2



Step 6: Wiring an H-bridge and Driving Motors

So for this next part were going to be using an H-bridge. Although I don't expect you to understand it, I'd like you to look over the documentation for the H-bridge so you know what spec sheets for electronic components look like. If you ever come across a component your unfamiliar with you should be able to google the spec sheet for it using then number inscribed on it.

http://www.ti.com/lit/ds/symlink/sn754410.pdf

The SN7544 H-Bridge has 16 pins and the pins are labeled 1 through 16 inside the box representing the chip. This chip is capable of powering two motors. One is wired to the left half of the chip the other on the right half of the chip. I will now describe what each pin does.

1.) This is the enable line for motor 1. When this is HIGH, 5 volts. The motor can be driven. When it is LOW, 0 Volts the motor will not operate. This can be used with a PWM signal (We will learn about later) to control the speed of the motor by turning it on and off really fast.

2.) This is a control line for motor 1. You use this with pin 7 to control which direction the motor 1 spins. If pin 2 is set HIGH and pin 6 is set LOW. The motor will spin one way. If you flip them so pin 6 is HIGH and pin 2 is LOW the motor will change direction. In all other states the motor does not spin.

3.) This pin gets wired to one side of Motor 1. The other side of Motor 1 gets wired to pin 6

4.) This goes to your Ground Rail, 0 Volts. This is where the power from the motor exits.

5.) Same as 4

6.) This pin gets wired to other one of side of Motor 1.

7.) This is the other control line for motor 1.

8.) This pin needs 5 Volts for the chip to operate properly. This powers the chip

9.) This is the enable line for motor 2.

10.) This is a control line for Motor 2.

11.) This is gets wired to one side of Motor 2. The other side of Motor 2 gets wired to pin 14

12.) Same as 4

13.) Same as 4

14.) This gets wired to the other side of motor 2.

15.) This is the other control line for motor 2.

16.) This is the power supply for motor 1 and motor 2. For this example I'd recommend a 9 volt battery.

Now for wiring it up!

Look at the fritzing diagram. Couple of key things to note. Make sure you connect the GND (0 Volts) of the Arduino with the - of the battery (0 Volts). And DO NOT connect the 5V of the Arduino with the + of the battery (9 Volts). This will hurt your Arduino.

After wiring it up open up _1Motor and run it.

Challenge: Wire the motor with the switch so that when pressed it spins one directing, and released it spins in the opposite direction.

Solution: _1MotorWithSwitch

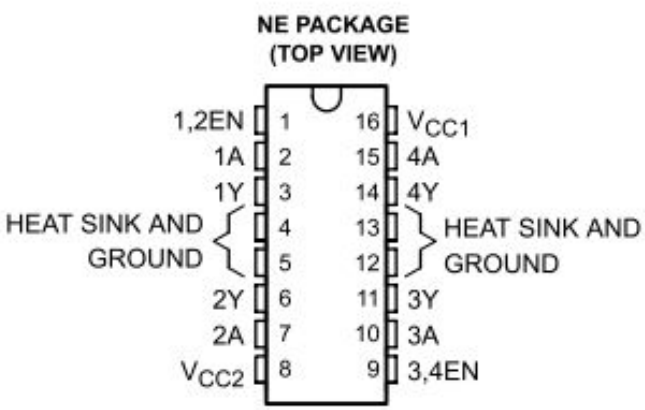
Challenge: Make it so that one switch makes the motor spin in one directing. The other switch makes it spin in the opposite directing. No switches make it stop.

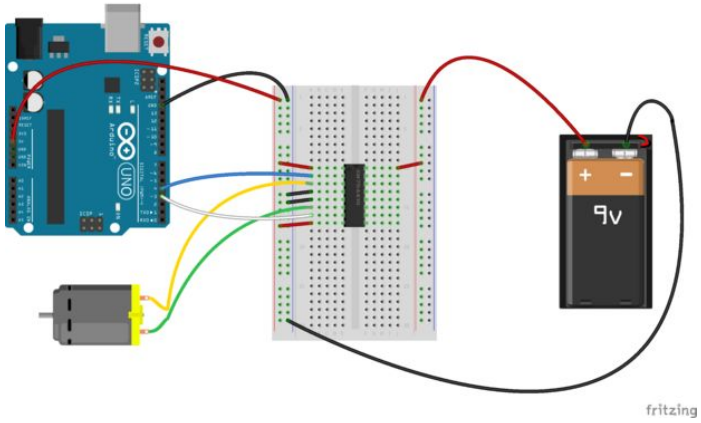
Hint: You may need to learn about the "and" operator for your if statement.

Solution: _1MotorTwoSwitches

Challenge: Hook Up the second Motor and do something interesting with it.

Note: If you want to start driving around all you need are some wheels which you can buy from here https://solarbotics.com/product/gmpw/ or make your own out of jar lids and rubber bands.





Step 7: Analog Input

Now we'll show you how to read in an analog input. There a many analog sensors out there you can use for your robot as a sensor.

Examples Below:

Potentiometer (What we will be using for this demo)

Range Finders https://www.pololu.com/product/2474

Temperature Sensors http://www.adafruit.com/products/165

And many more (Just google analog whatever type of sensor you want)

Potentiometers are one of the simplest analog inputs you can use. You turn the nob to change the value.

Wire it following the fritzing diagram.

Make sure to look over the documentation

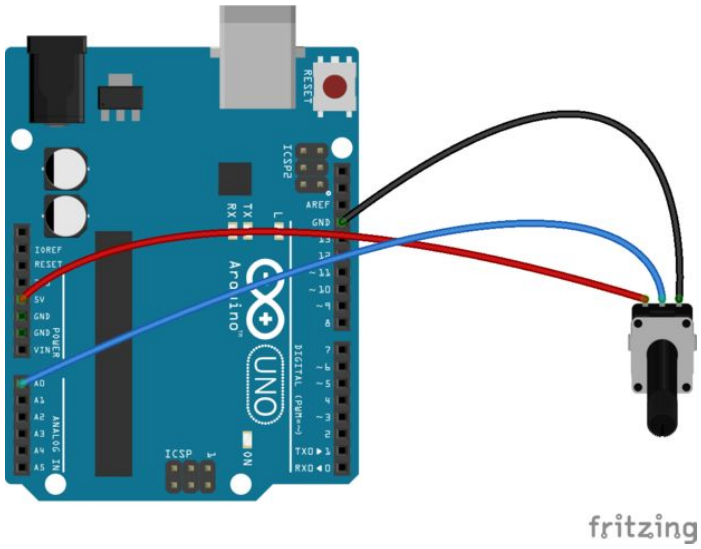
http://www.arduino.cc/en/Reference/AnalogRead

Open the program PrintAnalogValue and try it out with your potentiometer wired up.

If you noticed this is very similar to the limit switch example except we are using the "analogRead()" function and the pins labeled A0 through A5. A stands for analog and you must use these pins when trying to get an analog input. You also get a value from 0 to 1023 as you turn the potentiometer. This is what makes it an analog input. You don't get a 0 for LOW and a 1 for HIGH. Instead you get a 0 for Low, 1023 for HIGH. All the voltages in between are represented by numbers 1 to 1022.

Challenge:

See if you can figure out how POT_ChangeTheSpeed, POT_MoveTheLight, and POT_RaisingTheBar work and wire them up!



Step 8: PWM signals and controlling motor speed

Now we'll be using to the potentiometer to control the speed of the motors.

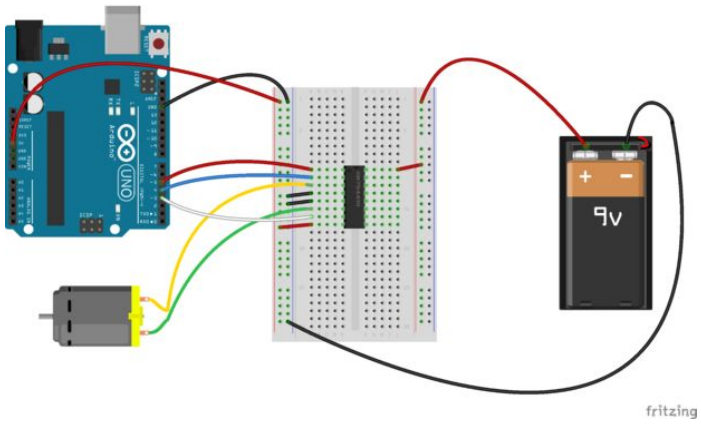
As I said earlier the enable lines on the H-bridge can be used to control Motor speed.

See Fritzing diagram for wiring.

The only major difference from last time is pin 1 on the H-bridge is now connected to pin 5 on the Arduino instead of the 5 Volt rail.

PWM signals are signals that turn on and off really fast. Typically they are given a value from 0 to 255. Where 0 means they are never on and 255 means they are on all the time. Value in between mean that they are on for only part of the time. Arduino denotes what pins support PWM putting at ~ next to the pin number.

Open



Step 9: Writing Functions

In this section we will go over writing your own functions.

Hint: Avoid naming a variable and function the same thing. This confuses the compiler.

As I said early everything in programming is based off functions in some way shape or form. Now you learn how to write your own. The idea behind functions is to take a complicated task and hide it away to make your code easier to use and understand. As I'm sure you've started to realize getting a motor to run requires you to be keeping track of multiple things, what speed you want it to run, what direction you want it to run, and what pins need to be HIGH and LOW to get this all to work.

In the file called MotorFunctions you can see some example functions we wrote for you and their explanation below.

1.) This part of the function declaration is the return type. We picked void because this function will not be returning anything. For example if you wanted to return a number you would use int. This might come in handy if you had a function that added two numbers together and returned their sum.

2.) This part of the function is the function name. This is the name you will use to call your function later in the code.

3.) These are the parameters of the function. Here you specify how many parameters you will take and what they will be called. For this one we have one parameter motorSpeed, and it is a number so we used the the type int. Specify parameters is just like declaring variables because they are essentially variables that are shared with your function from what called your function.

4.) This is the code that the function runs.

5.) Return statement, because we do not return anything we simply put this statement. If we were returning something. We would place that after the return statement.

Step 10: Writing Classes For the Motors

Yeah this sounds like a good idea but I'm probably not going to get to it any time soon. Just a mental place holder