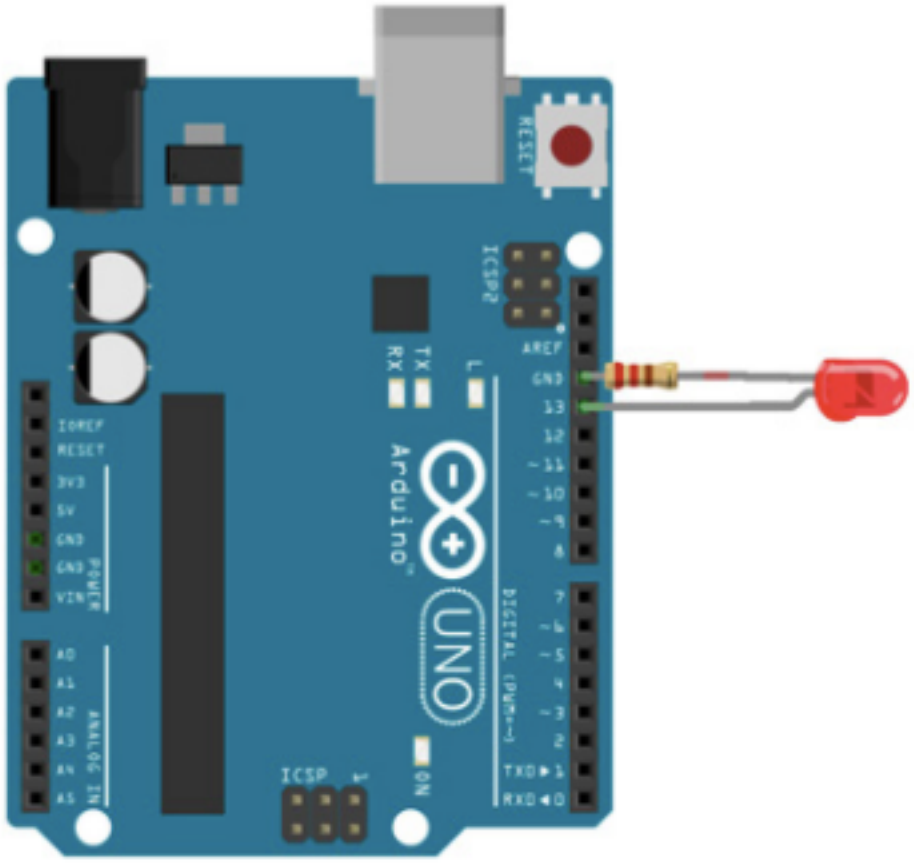
Advanced Arduino

#1 – Test Arduino
The first project is one of the most basic and simple circuits you can create with Arduino. This project will test your Arduino by blinking an LED that is connected directly to the board.

Parts Needed

(1) Arduino Uno
(1) USB A-to-B Cable
(1) LED 5mm
(1) 220 Ω Resistor

Project Diagram



Project Steps

Twist a 220 Ω resistor to the long leg (+) of the LED.
Push the short leg of the LED into the ground (GND) pin on the board.
Push the resistor leg that's connected to the LED into the #13 pin.

Project Code

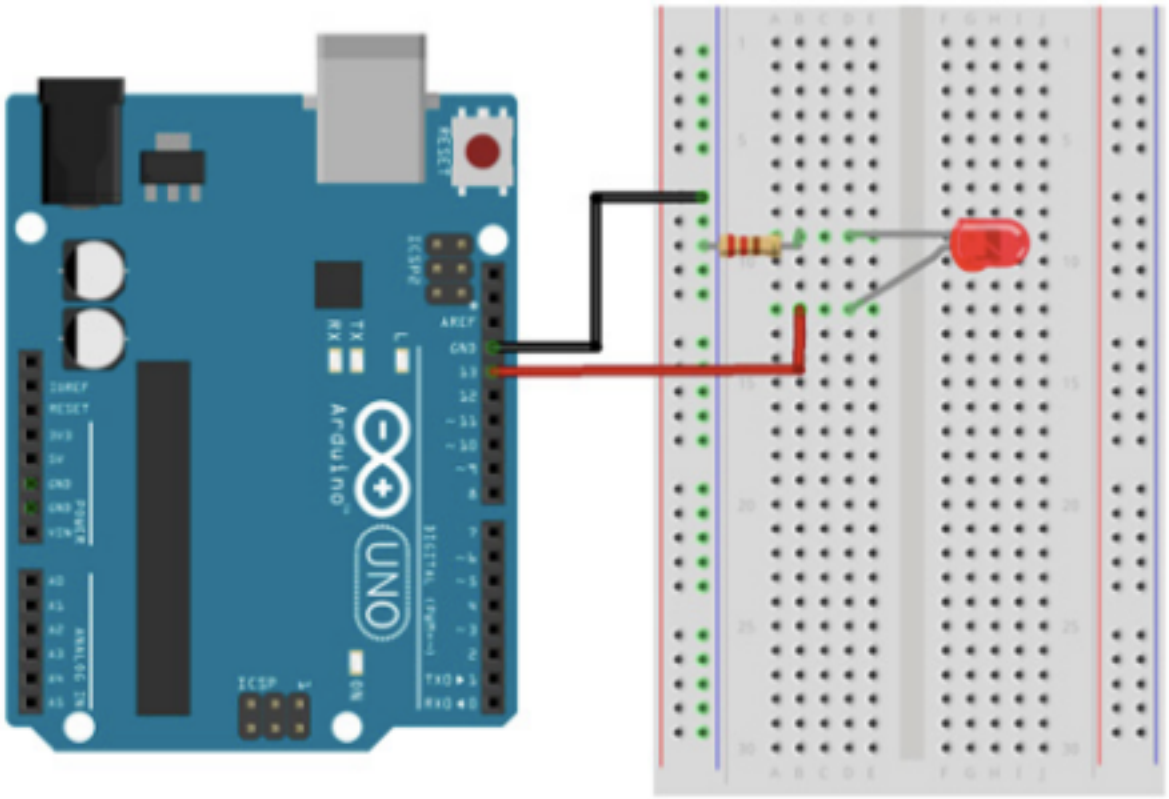
Connect the Arduino board to your computer using the USB cable.
Open project code – **Circuit_01_TestArduino**
Select the board and serial port as outlined in earlier section.
Click upload button to send sketch to the Arduino.

#2 – Blink an LED
This project is identical to project #1 except that we will be building it on a breadboard. Once complete, the LED should turn on for a second and then off for a second in a loop.

Parts Needed

(1) Arduino Uno
(1) USB A-to-B Cable
(1) Breadboard – Half Size
(1) LED 5mm
(1) 220 Ω Resistor
(2) Jumper Wires

Project Diagram



Project Code

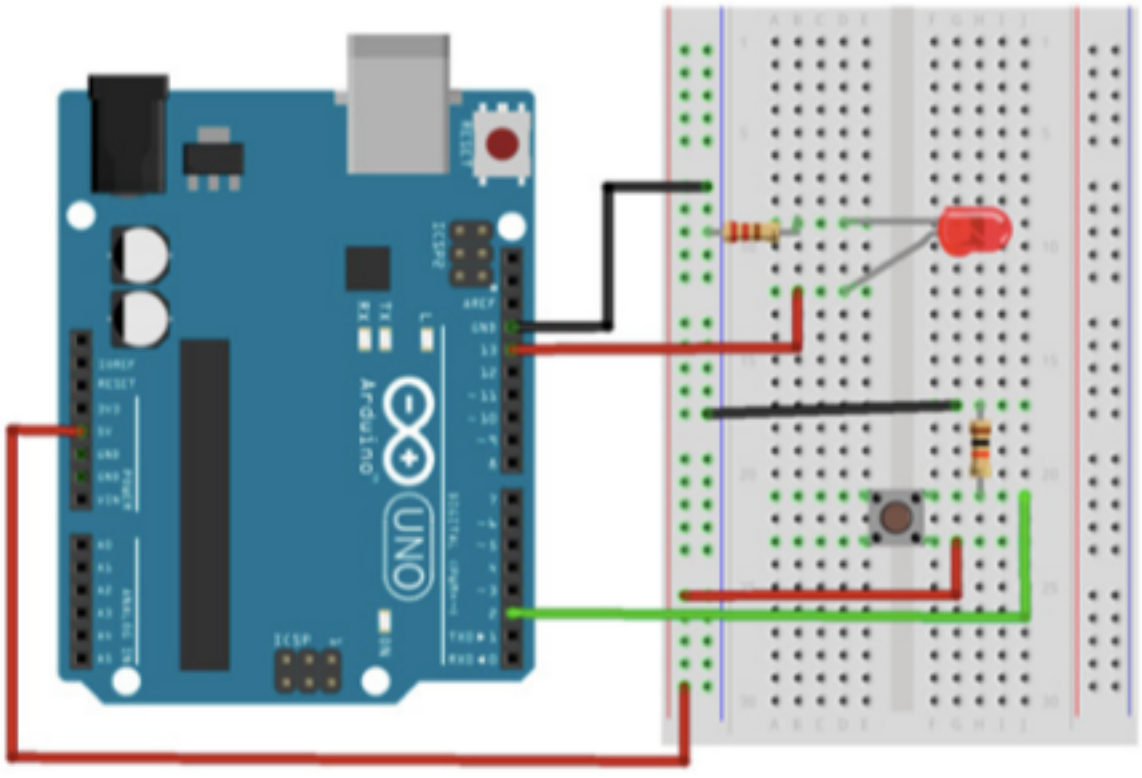
Connect the Arduino board to your computer using the USB cable.
Open project code – **Circuit_02_Blink**
Select the board and serial port as outlined in earlier section.
Click upload button to send sketch to the Arduino.

#3 – Push Button
Using a push button switch, you will be able to turn on and off an LED.

Parts Needed

(1) Arduino Uno
(1) USB A-to-B Cable
(1) Breadboard – Half Size
(1) LED 5mm
(1) 220 Ω Resistor
(1) 10K Ω Resistor
(1) Push Button Switch
(6) Jumper Wires

Project Diagram



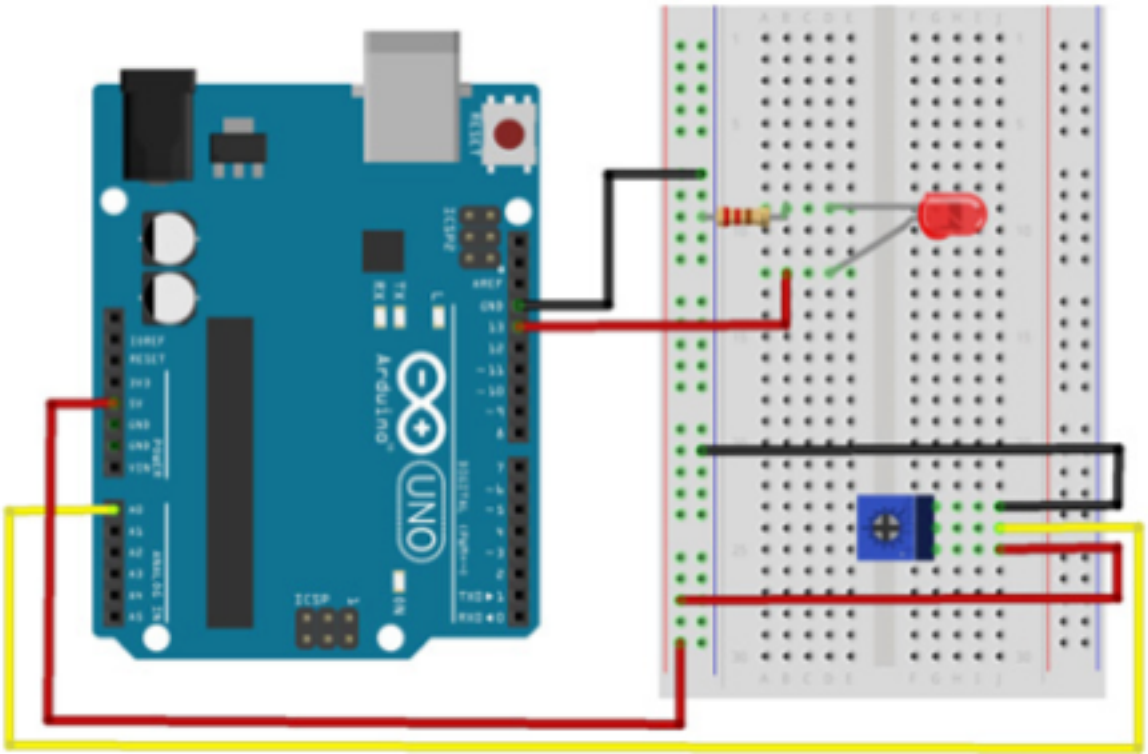
Project Code

Connect the Arduino board to your computer using the USB cable.
Open project code – **Circuit_03_Pushbutton**
Select the board and serial port as outlined in earlier section.
Click upload button to send sketch to the Arduino.

#4 – Potentiometer
Using a potentiometer, you will be able to control the resistance of an LED. Turning the knob will increase and decrease the frequency the LED blinks.

Parts Needed

(1) Arduino Uno
(1) USB A-to-B Cable
(1) Breadboard – Half Size
(1) LED 5mm
(1) 220 Ω Resistor
(1) Potentiometer (10k Trimpot)
(6) Jumper Wires
Project Diagram



Project Code

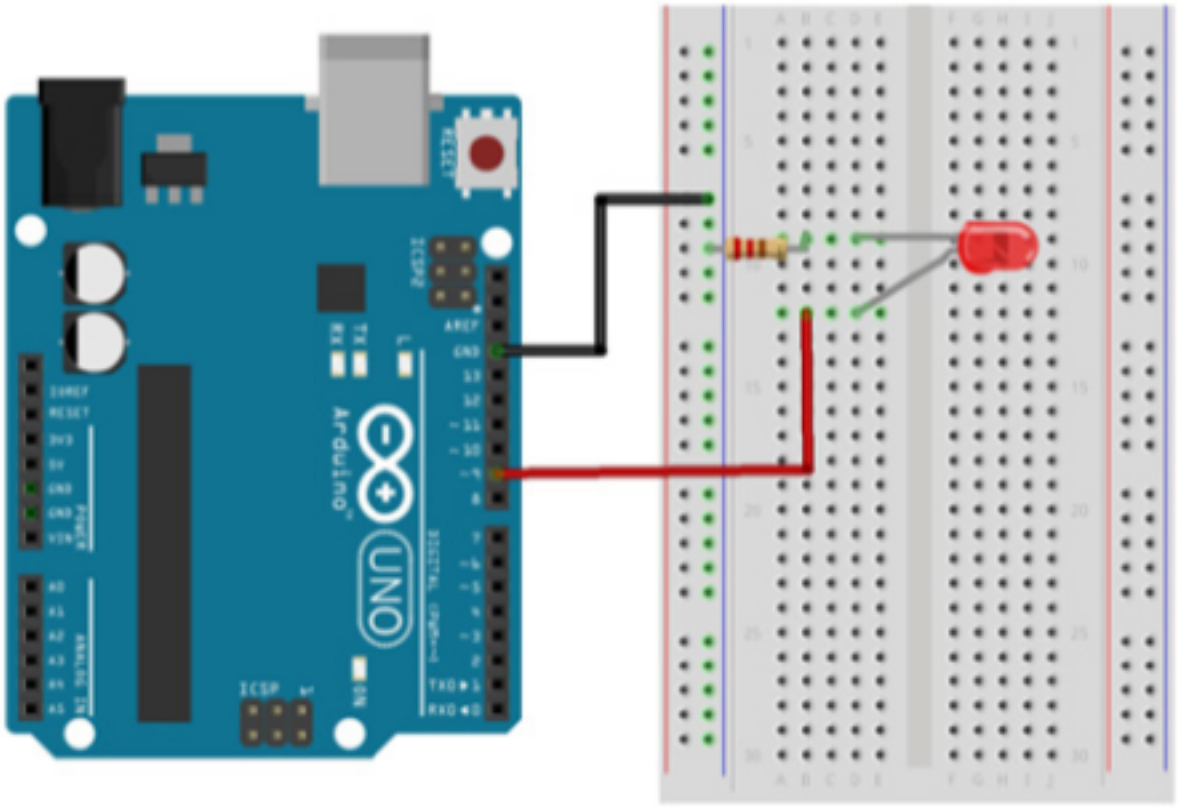
Connect the Arduino board to your computer using the USB cable.
Open project code – Circuit_04_Potentiometer
Select the board and serial port as outlined in earlier section.
Click upload button to send sketch to the Arduino.

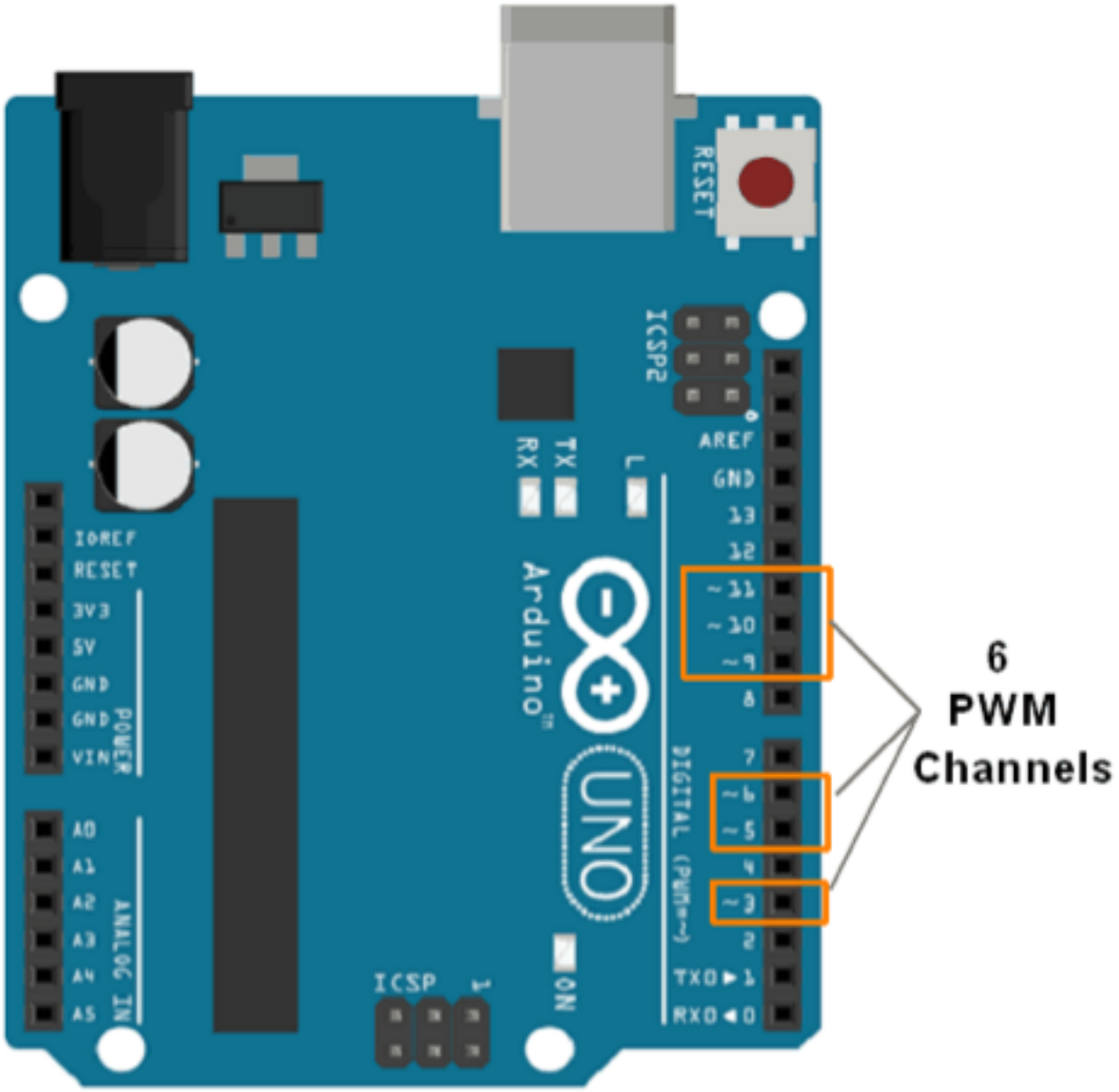
#5 – Fade an LED
By using a PWM pin on the Arduino, you will be able to increase and decrease the intensity of brightness of an LED.

Parts Needed

(1) Arduino Uno
(1) USB A-to-B Cable
(1) Breadboard – Half Size
(1) LED 5mm
(1) 220 Ω Resistor
(2) Jumper Wires

Project Diagram





PWM- pulse width modulation pins

Project Code

Connect the Arduino board to your computer using the USB cable.
Open project code – Circuit_05_Fade
Select the board and serial port as outlined in earlier section.
Click upload button to send sketch to the Arduino.

PWM stands for **Pulse Width Modulation** technique , It is used to convert the digital signal into an analog by varying the width of the Pulse . The **PWM pins** are used for giving the desired analog output . They are used to set the LED brightness or to run Stepper or Servo Motor or anything which require analog inputs .

OHM'S LAW VIDEO:

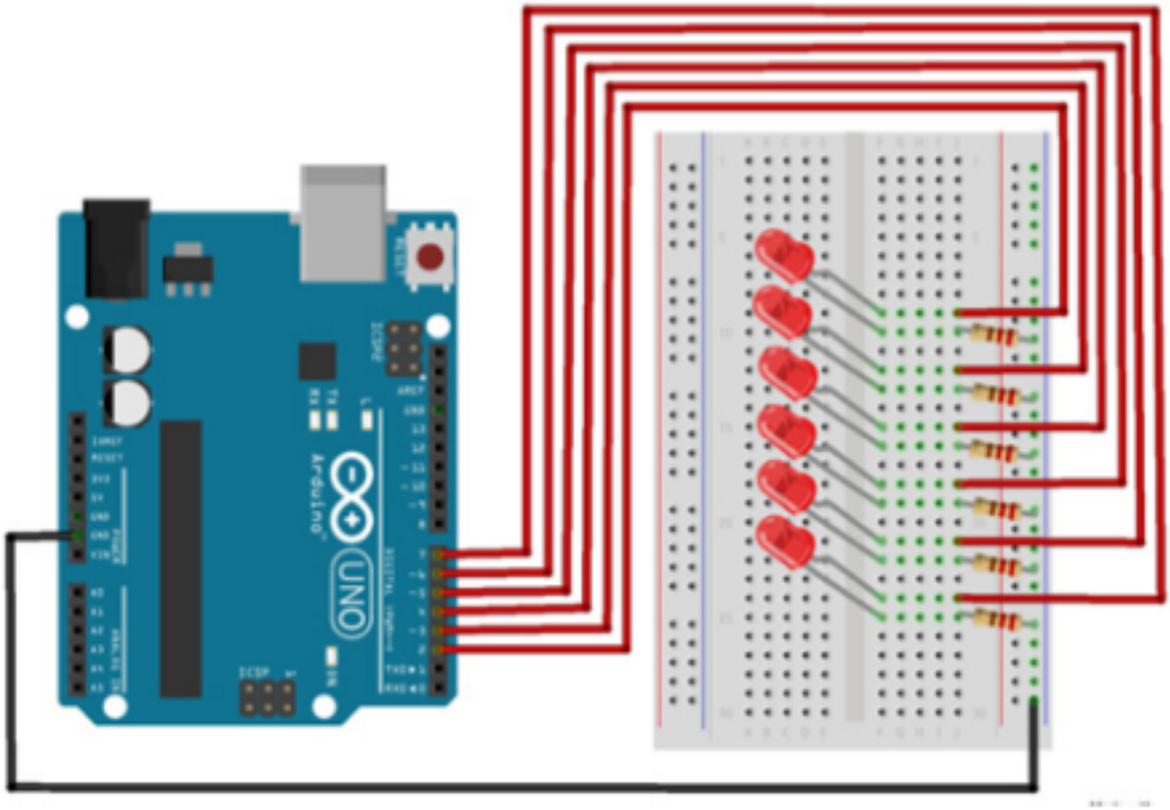
https://www.youtube.com/watch?v=HsLLq6Rm5tU&feature=emb_logo

#6 – Scrolling LED
This project will blink 6 LEDs, one at a time, in a back and forth formation. This type of circuit was made famous by the show Knight Rider which featured a car with looping LEDs.

Parts Needed

(1) Arduino Uno
(1) USB A-to-B Cable
(1) Breadboard – Half Size
(6) LED 5mm
(6) 220 Ω Resistor
(7) Jumper Wires

Project Diagram



Project Code

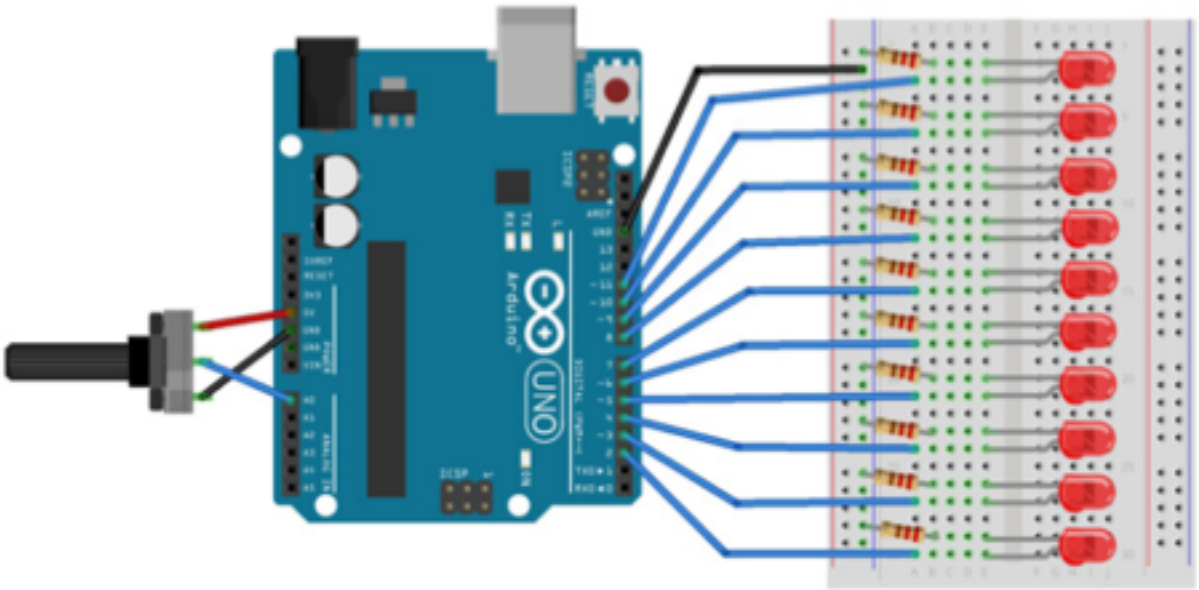
Connect the Arduino board to your computer using the USB cable.
Open project code – Circuit_06_Scrolling
Select the board and serial port as outlined in earlier section.
Click upload button to send sketch to the Arduino.

#7 – Bar Graph
Using a potentiometer, you can control a series of LEDs in a row. Turning the potentiometer knob will turn on or off more of the LEDs.

Parts Needed

(1) Arduino Uno
(1) USB A-to-B Cable
(1) Breadboard – Half Size
(1) Potentiometer – Rotary
(10) LED 5mm
(10) 220 Ω Resistor
(11) Jumper Wires

Project Diagram



Project Code

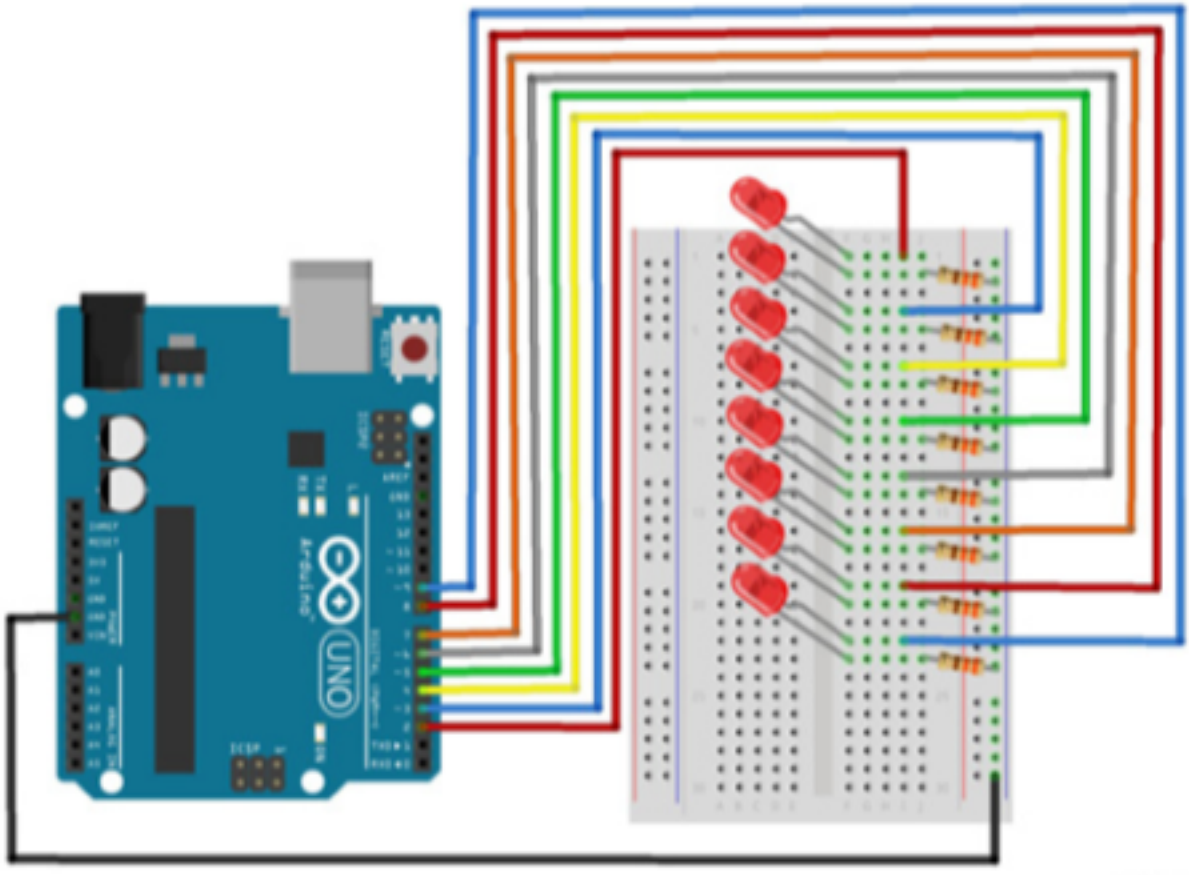
Connect the Arduino board to your computer using the USB cable.
Open project code – Circuit_07_BarGraph
Select the board and serial port as outlined in earlier section.
Click upload button to send sketch to the Arduino.

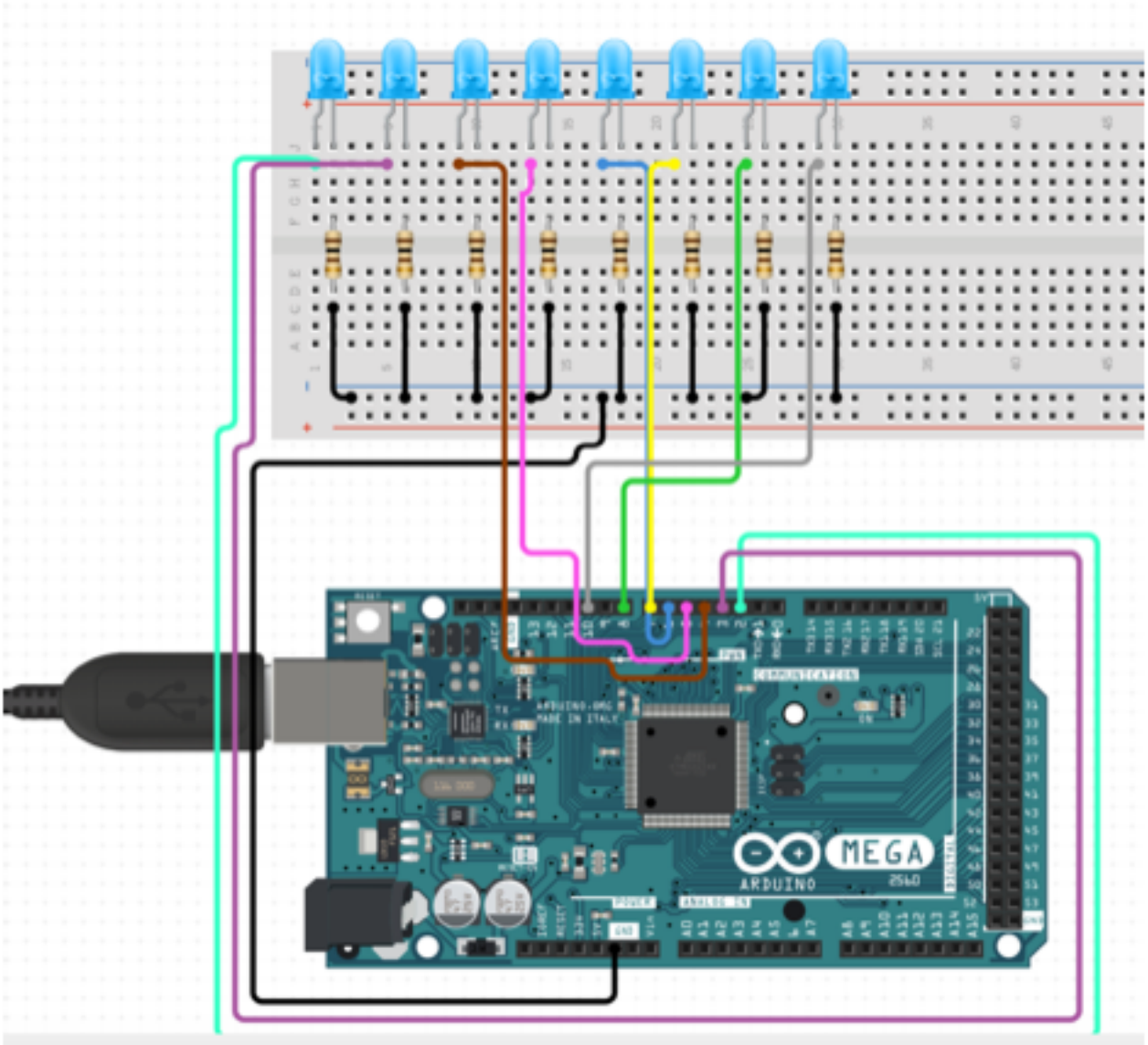
#8 – Multiple LEDs
This project will use 8 pins on the Arduino board to blink 8 LEDs at the same time.

Parts Needed

(1) Arduino Uno
(1) USB A-to-B Cable
(1) Breadboard – Half Size
(8) LED 5mm
(8) 330 Ω Resistor
(9) Jumper Wires

Project Diagram

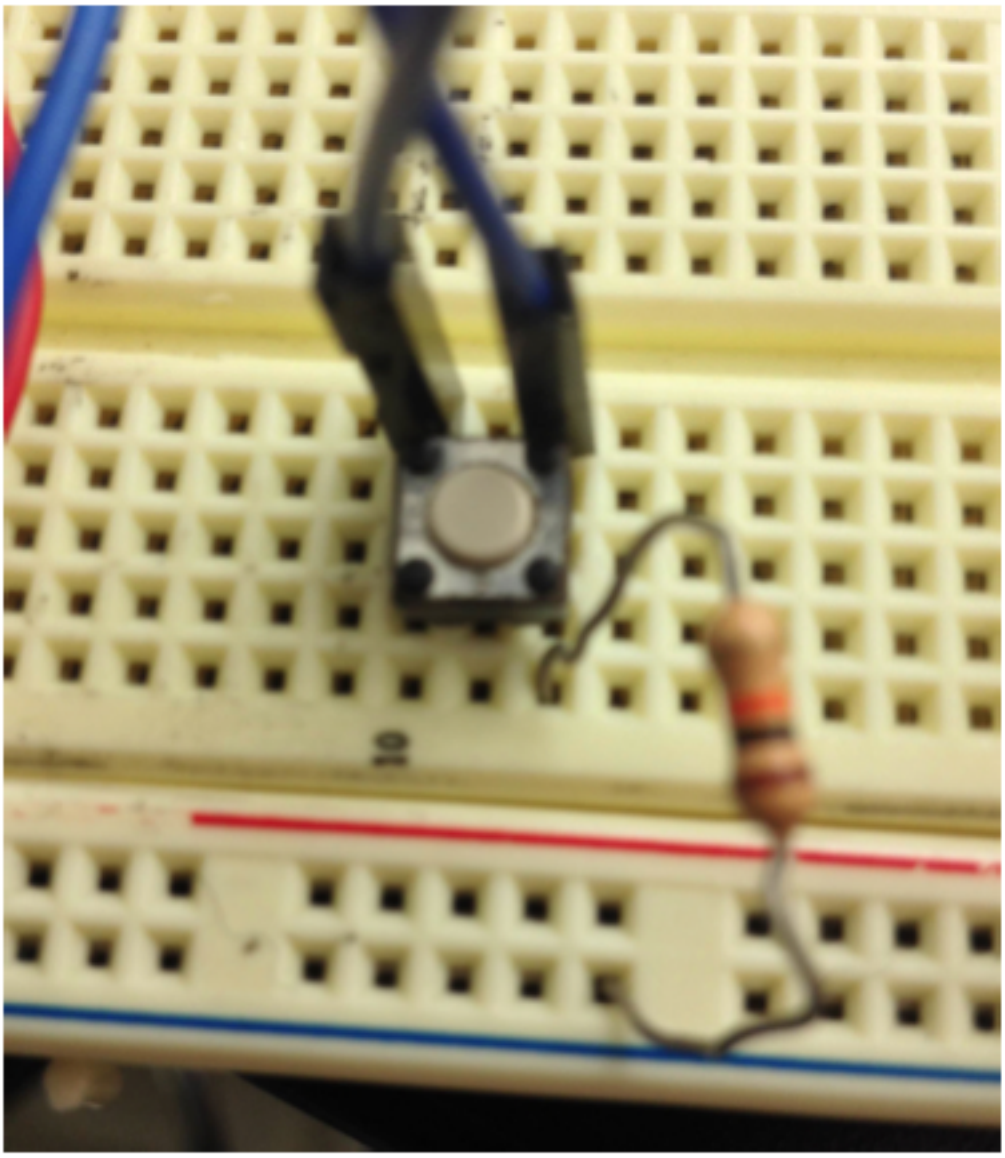


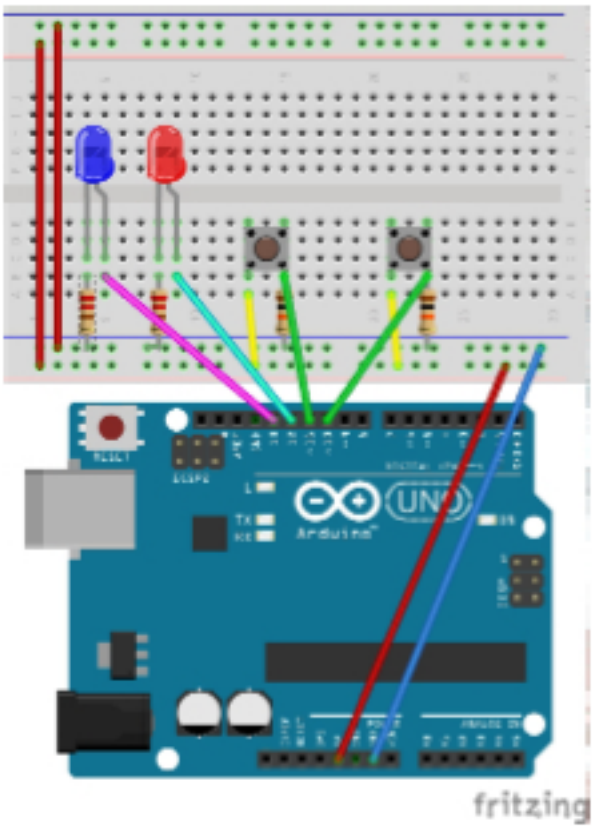


Project Code

Connect the Arduino board to your computer using the USB cable.
Open project code – Circuit_08_MultipleLEDs
Select the board and serial port as outlined in earlier section.
Click upload button to send sketch to the Arduino.

Button as Input

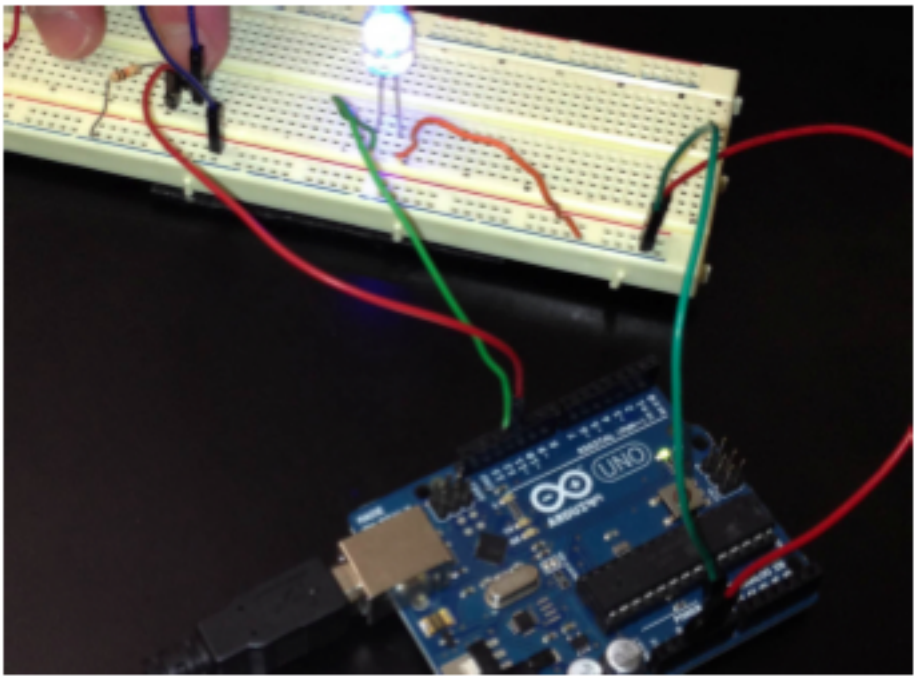




Now, we'll use the button as an input source. From the breadboard, take out the LED, the resistor, and the flex wire to ground, but leave in the button. Change the flex wire connecting to the Arduino to port 11. In the same row as the flex wire that connects to port 11, put in the 10,000 Ohm resistor. The other end of the 10,000 Ohm resistor should go in one of the blue ground rows on the side of the breadboard. With another flex wire, connect the row with the other button wire to one of the red power rows on the side of the breadboard.

Notice how the button is wired. One end of the button connects to power and the other end of the button connects to port 11. (The 10,000 Ohm resistor is only in the circuit to prevent bad stuff like short circuits from happening.) When the button is pushed it will connect the red power row to port 11; that is, when the button is pushed the voltage at port 11 will be high.

Great, now set up the second button exactly like the first, but have a flex wire connect to port 10 instead. Woohoo!



Let's add some LEDs as well. Place the high voltage wire of the LED in any empty row and put the low voltage wire into a row right next to it. Connect the low voltage wire row of the LED to one of the blue ground rows on the side of the breadboard using a 220 Ohm resistor. Take a flex wire and use it to connect the power side of the LED to port 12.

Take another LED, 220 Ohm resistor, and flex wire, and set them up just like the previous LED, except have the flex wire for the second LED go to port 13.

Cool. Everything is set up except for the red and blue columns. We need to finally put these to work. Connect a flex wire from the red power row on the breadboard to the **5 V** port on the Arduino, and connect a flex wire from the blue ground row on the breadboard to the GND port on the Arduino. It makes things easier if the red and blue column flex wires are right next to each other and on one far side of the breadboard. You know what else makes things easier... using a red wire for the red column and a blue or black wire for the ground column. Why, you ask? Because in circuits, every time there's a power connection it's labeled red, and every ground connection is either blue or black. It just makes things easier to keep track of. Also, make sure that the red flat

wires connecting the blue columns and the red columns are still in your breadboard. You may have accidentally removed them after the last unit.

One last helpful piece of advice... for virtually every unit and every project for the rest of this course, you will need a wire going from power to the red column and a wire going from ground to the blue column. **This is the #1 mistake that people make with their circuits.** They either...

1. forget to connect these wires
2. connect them backwards (which is bad for the Arduino)
3. connect power and ground to the same column by mistake (which is super bad for the Arduino)

Just be aware of this moving forward, and think of it first whenever things aren't working.

Take the following code and upload it to the Arduino:

```
void setup() {
pinMode(11,INPUT); // sets port 11 as an input port for the button
pinMode(12,OUTPUT); // sets port 12 as an output port for the first LED
pinMode(13,OUTPUT); // sets port 13 as an output port for the second LED
}

void loop() {
digitalWrite(12, LOW); // sets port 12 to low voltage; turn off first LED

if (digitalRead(11) == HIGH) { // if the button is pushed (and connects high voltage to port 11)
    • digitalWrite(12, HIGH); // sets port 12 to high voltage; turn on first LED
        ◦ delay(500); // wait 0.5 seconds
            ▪ }
            ▪ }
```

This code has 3 new features:

1) **digitalRead(11) == HIGH**: this function does the exact opposite of digitalWrite(). Rather than send high or low voltage to a port, the digitalRead()

function **checks** the voltage at a port and returns either HIGH or LOW. (Remember the 2 purposes of a port from the last unit?)

2) **if**: the if command performs a test in its parentheses. If the test is true, then it does the code in the curly braces. If it's false, it doesn't do anything. In the program above, the if statement checks if the voltage at port 11 is high (if the button is pushed).

3) **==**: the double equals sign is **NOT** the same as an equals sign from math class. The double equals means are these 2 things the same? So, the double equals asks a question; it compares the 2 things. That's why it's called the comparator command. (The single equals sign means set them equal. For instance, x = 3 means set x equal to 3.) In this program, the double equals asks if digitalRead(11) is the same as HIGH. That is, it asks if the value of the voltage at port 11 is high.

Input vs. Output

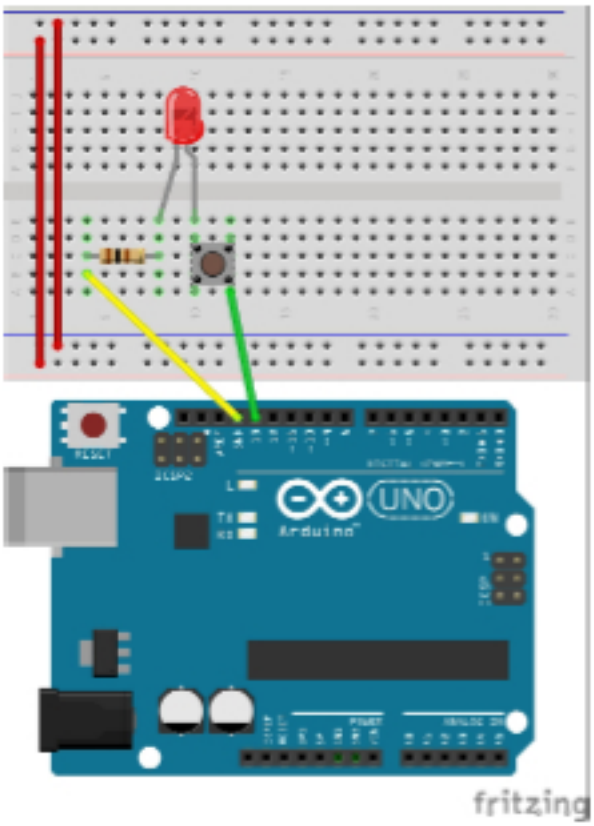
We have already seen ways that the Arduino can output information -- sending high and low voltages to ports. The high voltages turned on the LED and the number 8 block, and the low voltages turned them off.

The ports can also handle inputted information in the same form -- high and low voltages. When a port detects a voltage, your program can respond to this input and do something (turn on a light, make a sound, print something, etc.).

For each program, a port can only be **EITHER** an input or an output port, and it cannot change during a program. Thus, when you set up ports in the setup() function, it can be input or output and must remain in this state for the entire program. There is no switching from input to output with ports.

Using a Button

Remember how we talked about how every circuit needs to form a complete loop? If there is any break in the loop, then the entire circuit doesn't work. Buttons are designed to be breaks in the circuit that are controlled by a person. When you push a button, it completes the loop (and allows current to flow). When the button is not pushed, there is a break in the circuit and no current flows.



Now, let's add a button to the breadboard. If you still have your LED and resistor (100 Ohms or 220 Ohms are perfect) in the breadboard from the last unit, awesome. If not, you'll need to connect them again. If you forget how to connect your LED and resistor, go back to the 'The Best Thing Since Sliced Bread' section of the ports unit to read how.

The button's 2 wires should go in different rows. For instance, rows 20 and 22. The spacing for the wires on the button is designed to fit across 3 rows (that is, wire...empty row...wire); so, 3 rows works perfectly, but 2 rows right next to

each other for the wires makes for a tight fit. It doesn't matter which way you put in the button, it'll work either way (like a resistor, but unlike the LED).

Take the flex wire connecting to port 13 and move the end in the breadboard to a new row that is 2 rows away from the high voltage wire of the LED. For example, if your high voltage wire is in row 8, move it to row 6. Now, take your button and put one of the wires into the row with the flex wire going to port 13 and put the other wire in the row with the high voltage side of the LED. When you're all done, the circuit loop should be... port 13 to button, button to LED, LED to resistor, resistor to ground. OK?

Now, upload the following code to the Arduino and notice what happens:

```
void setup() {
pinMode(13,OUTPUT); // sets port 13 as an output port
}

void loop() {
digitalWrite(13, HIGH); // sets port 13 to high voltage
}
```

So, what happened? Nothing! Why? The circuit is broken (not complete) because of the button. To light up the LED, push the button. Woot!! When you push the button, the circuit is complete and the LED gets current running through it... aka electricity.

Try changing the code to make the LED blink when you push the button.

Hide Answer

Remember you need to turn on the LED, wait, turn it off, and then wait again.

digitalRead() does what?
Hide Answer
Checks the voltage (HIGH or LOW) at a port

What does the '= =' do?
Hide Answer
Checks if the 2 things are equal to one another

How would you make the other LED light up instead?
Hide Answer
Change the 2 lines with digitalWrite to:
digitalWrite(13, LOW);
digitalWrite(13, HIGH);

OK, now try to make the other LED light up. Once you get this working, try adding your own lines of code to make **both** LEDs light up when you push the button.

If...else

The if statement is one of the most useful and frequently used commands in computer programming. It allows the program to make a decision based on some input. It will branch the flow of the program, depending on the test it runs. Closely related to the if statement are the else statement and the else if statement. We'll take a look at all 3 here.

if: if the test statement in parentheses is true, run the code. If not, do nothing. The if statement (like all the commands that follow) takes 2 slightly different

forms depending on whether you want to run 1 line of code if true or run multiple lines of code if true. Here are the 2 forms of the if statement:

```
// the single line of code takes the form: if (test statement) code to run;

if (digitalRead(11) == HIGH) digitalWrite(12, HIGH); // run this one line of code (set port 12 to high) if the test is true


// the multiples lines of code takes the form: if (test statement) { lines of code to run }

if (digitalRead(11) == HIGH) {
  digitalWrite(12, HIGH); // run these multiple lines of code if the test is true
    delay(500);
      }
```

Remember what the == mean?
Show Answer

Notice that there is **NO** semicolon after the if parentheses in the second method because there are curly braces in this method. In general, all functions follow this format...

- A single line of code ends in a semicolon.
- Multiple lines of code are surrounded by curly braces {} and there is a semicolon at the end of each line of code within the curly braces. (You could always use this method if you want, even when you only have 1 line of code to run, if you want to make your code consistent.)

else: used only with the if statement; never by itself. If the test statement in parentheses from the if statement is false, run this code instead. Here is an example of the else statement:

```
if (digitalRead(11) == HIGH) {
    digitalWrite(12, HIGH); // run this one line of code (set port 12 to high) if the test is true
        }

        else {
            digitalWrite(12, LOW); // run this line of code (set port 12 to low) if the test is false
                }
```

Notice that there is no test statement for the else statement. Why? **Because the else statement code is run only if the if statement is false.** There's nothing to test because it was already tested in the if statement.

else if: used only with the if and else statements; never by itself. The else if statement offers multiple choices for the program. The if statement always goes first and the else statement always goes last. In between, you can put as many else if statements as you like. Each else if statement is tested in order. If the if statement or any of the else if statements are true, all of the remaining else if statements and the else statement are skipped.

Here's a real-world example of if, else if, and else: let's say you're the best quarterback who ever lived (like Tom Brady) and you're trying to decide what to do during a pass play. This pseudo-code might be Tom's thought process.

```
if (Edelman == open) pass to Edelman;
else if (Gronk == open) pass to Gronk;
else if (LaFell == open) pass to LaFell;
else if (Amendola == open) pass to Amendola;
else run with the ball;
```

First off, notice how it's OK to have multiple else if statements. So, see how these commands work? If Edelman is open, pass him the ball. If he's covered, go to the next option... Gronk. If Gronk is open, pass to him. You keep going through all of the else if statements to see if any are true. If one is true, then you run that code and skip everything else. So, if Gronk were open, Brady

would pass to him and not even check if LaFell or Amendola were open as well. This is how Brady thinks and also how a computer thinks. Finally, if nobody is open (we've gone through all the if and else if statements are they're all false), then do the else code, which in this case happens to be running with the ball. So, the else command is the last option. It's what to do when everything else fails.

If Edelman is open, what happens?
Show Answer

What happens if both Amendola and LaFell are open?
Show Answer

If you want Brady to throw to Amendola before Gronk, how would you change the code?
Show Answer

Can I have more than one else statement?
Show Answer

Can I have more than one else if statement?
Show Answer

Can you have more than one if statement?
Show Answer

The else if statement takes the same form as the if statement... else if (test statement) code to run if true. Here is an example of the else if statement with some real code that checks multiple buttons and then prints stuff to the serial monitor:

```
void setup() {
Serial.begin(9600);
pinMode(10, INPUT);
pinMode(11, INPUT);
}

void loop() {
if (digitalRead(10) == HIGH) {
    • Serial.println("Button 1 is being pushed"); // print this if the port 10 button is pushed
        ◦ }
        ◦
        ◦ else if (digitalRead(11) == HIGH) {
            ▪ Serial.println("Button 2 is being pushed now!"); // run this if the port 11 button is pushed
                ▪ }
                ▪
                ▪ else {
                    ▪ Serial.println("Neither button is being pushed... I'm bored."); // run this if neither button is pushed
                        ▪ }
                        ▪ }
```

Notice that the else statement has no test, because it is the fallback statement. That is, the else statement is run only when all of the above if and else if statements are false.

Try running the above code, look at the serial monitor, and see if it makes sense. If you want, you can change the print statements to whatever you'd like.

Try pushing both buttons at the same time. What happens and why?
Show Answer

Now, try changing the tests for both the if statement and the else if statement to "LOW". Upload this new code and see what happens. Does this make sense? Change the print statements to make them accurate.

With both ports checking for LOW voltage, how do you get the else statement to print?
Show Answer

In the future, if you ever get confused about the if, else if, or else statements and how to use them, you can always go to the Code page and review their definitions and usage. There is also a link for the Code page on the top of every web page.