# Light it Up!light-bulb-clip-art-bulb-lamp-hi.png

## [Lesson 1]

To get you started with programming, we’re going to work with Light Emitting Diodes, called LEDs for short. Using your Arduino, we can control the circuit containing an LED to turn it on or off. Start out by reading through the comments in the template code (any lines beginning with //) to get a grasp of what programming languages look like!

Code runs from top to bottom; your Arduino will run through the code one line at a time, performing the actions that you tell it to do in order. There are two sections you should take note of: one that begins with void setup() and another that begins with void loop(). These are called **functions**, and you may notice that following each of these functions are curly brackets { }. All of the code that is written between these brackets is part of the function, and will execute line by line when the function is called.

The function void setup() does as the name implies; this is where you set up your Arduino by telling it what each pin is going to do. After the Arduino has an idea of what each pin is, we can begin programming! To turn the light on, we’re going to write the command digitalWrite(yellow, HIGH); inside of the curly brackets of void setup(), like so:

|  |
| --- |
| void setup() {  // This is all of the pin setup code  pinMode(red, OUTPUT);  pinMode(yellow, OUTPUT);  pinMode(green, OUTPUT);    pinMode(trainSensor, INPUT);  // Now we can start programming!  digitalWrite(yellow, HIGH);  } |

The command digitalWrite(yellow, HIGH); is telling the Arduino to turn on the pin connected to the yellow LED. Try turning on all of the LEDs!

# Blinking Forever

## [Lesson 2]

Next, we’re going to make an LED blink forever! In the last lesson, you learned that the command digitalWrite(yellow, HIGH); would turn an LED on. Here, we’re going to change that command a little bit to turn an LED off. You may have noticed that the words yellow and HIGH are inside of parentheses (). These words are called **arguments**, and by changing the arguments of a command, we can change what the command is doing! In this case, by replacing HIGH with LOW, the command digitalWrite will turn an LED off.

In order to make our lights blink forever, we have to make use of a logic structure called a **loop**. Loops are an important part of programming, and can be used to repeat a part of your program for a specific number of times, or repeat code while a certain condition is met. The loop that start with is a while-loop, which runs everything inside of the loop for as long as a condition is met. While-loops have the format while(condition){}, and will continue to run the contents of its curly brackets while the value condition evaluates to true. To make our while-loop run forever, we will simply write while(true){}, and fill in the commands inside of the curly bracket. For our light to blink, we’re going to also need the command delay(1000); to tell the Arduino to wait a little bit before turning the LED on or off, otherwise it would be blinking too fast! Try putting this code inside your void setup():

|  |
| --- |
| while(true) {  digitalWrite(yellow, HIGH);  delay(1000);  digitalWrite(yellow, LOW);  delay(1000);  } |

You can change the argument of the delay function to adjust how long the Arduino waits for in between commands. The number that goes into the function is the number of milliseconds (of a second) that the program will wait for; for instance, delay(1000); will wait for a second, while delay(500); will wait for half of a second. Try making your LEDs blink faster or slower!

# Traffic Lightsstop_light_v.png

## [Lesson 3]

Now that we know how to turn lights on and off forever, we can make a working traffic light! Traffic lights only have one light on at a time, and change colors in the order of green, yellow, then red. Using your knowledge of turning LEDs on and off and looping through commands, can you make your LEDs work like a traffic light? Once you get your traffic light working, try changing how long each light is on for! Remember, real lights only show yellow for a short time.

### Helpful Hints:

* Think about the steps that would go into the traffic light
  + The green light turns on
  + The green light turns off while the yellow light turns on
  + The yellow light turns off while the red light turns on
  + The red light turns off
* How can you turn these steps into a loop to make them run forever?

# Kill Switchkillswitchx-large.jpg

## [Lesson 4]

Our traffic light has helped many drivers navigate through busy intersections, but we need a way to turn it off to make repairs. We should change our program to turn the traffic light off when a button is pressed! Arduinos can receive inputs from their pins using the command digitalRead(pin), in which the argument is simply the pin that we are reading in a value from, and the function returns a value of either HIGH or LOW. HIGH indicates that the pin is receiving a signal, while LOW indicates that the pin is receiving nothing.

Earlier when while-loops were introduced, it was mentioned that a while-loop will only run while its condition is true. When the while-loop’s condition is no longer true, it stops repeating its contents. Before, we simply wrote true as the condition for the while-loop, but we can actually write statements that our program will determine as either true or false for the while-loop. Below are some examples of **comparison** statements we can use:

|  |
| --- |
| // Pretend that the button connected to inputPin is not being pressed  digitalRead(inputPin) == LOW // The symbols == ask if the two values are equal, or in this  // case if the value of digitalRead(inputPin) is LOW. This  // entire statement is true because the button is not pressed  // so digitalRead(inputPin) returns a value of LOW, making the  // comparison LOW == LOW, which is true  digitalRead(inputPin) != LOW // Here, they symbols != ask if the two values are NOT equal,  // or in this case if the value of digitalRead(inputPin) is  // NOT LOW. This entire statement is false because  // digitalRead(inputPin) returns a value of LOW, making the  // comparison LOW != LOW, which is false |

We can make use of both digitalRead and while-loops to create a while loop that runs until the Arduino detects a signal on its pin:

|  |
| --- |
| while(digitalRead(inputPin) == LOW){ // This while-loop will only run while there is no  // signal detected from inputPin, and will end when  // a signal is received  // do something  } |

An important thing to note is that the while-loop only checks to see if its condition is true after each repetition of its contents, so if the value of digitalRead(inputPin) were to change to HIGH while the code in the loop was still running, then switch back to LOW before the while-loop checked if digitalRead(inputPin) == LOW, the loop would still continue to run.

Can we edit our code to make the traffic light turn off when it detects that a button is pressed?

# The (Yellow) Flash

## [Lesson 5]

Drivers using your traffic light are having a hard time noticing when the light switches to yellow, and asked you to make it more obvious when they should slow down. Let’s help them out by making the yellow light flash! To do this, we will have to make use of a different condition in our while-loop. As mentioned earlier, a while-loop will run for as long as the condition within its parentheses is true. If we wanted our while loop to only run for a specific number of times, we have to make use of **variables** and **comparisons**. A variable is a value that is stored in our program and can be changed. We can define a variable with a statement such as:

|  |
| --- |
| int x = 5; |

This statement is creating an integer named “x” that has a value of 13. After “x” has been created, we can change the value of “x” by setting it equal to another number, or increasing or decreasing the value:

|  |
| --- |
| x = 10; // x now has a value of 10  x++; // x increases its value by 1, from 10 to 11  x--; // x decreases its value by 1, from 11 to 10 |

A comparison is essentially asking your program to evaluate whether a statement is true or false. Below are some examples of comparisons:

|  |
| --- |
| int x = 5;  x < 10 // This statement evaluates to true, because 5 is less than 10  x > 10 // This statement evaluates to false, because 5 is not greater than 10 |

Using a combination of variables and comparisons, we can make a while-loop only run for a specific number of times:

|  |
| --- |
| int x = 0;  while(x < 5){ // This while-loop will run its contents for a total of 5 times, increasing  x--; // the value of x by 1 each time until x is no longer less than 5, causing  } // the while-loop to stop running. |

We can use this sort of while-loop to control the number of times an LED will blink. With this knowledge, can we change our traffic light code to make the yellow light flash multiple times instead of simply turning on and off?

### Helpful Hints:

* We can place a while-loop inside of another while-loop. This is called a **nested loop**, and with each repetition of the outer loop, the inner loop will repeat all of its contents while its condition is true, then proceed to perform the rest of the commands of the outer loop