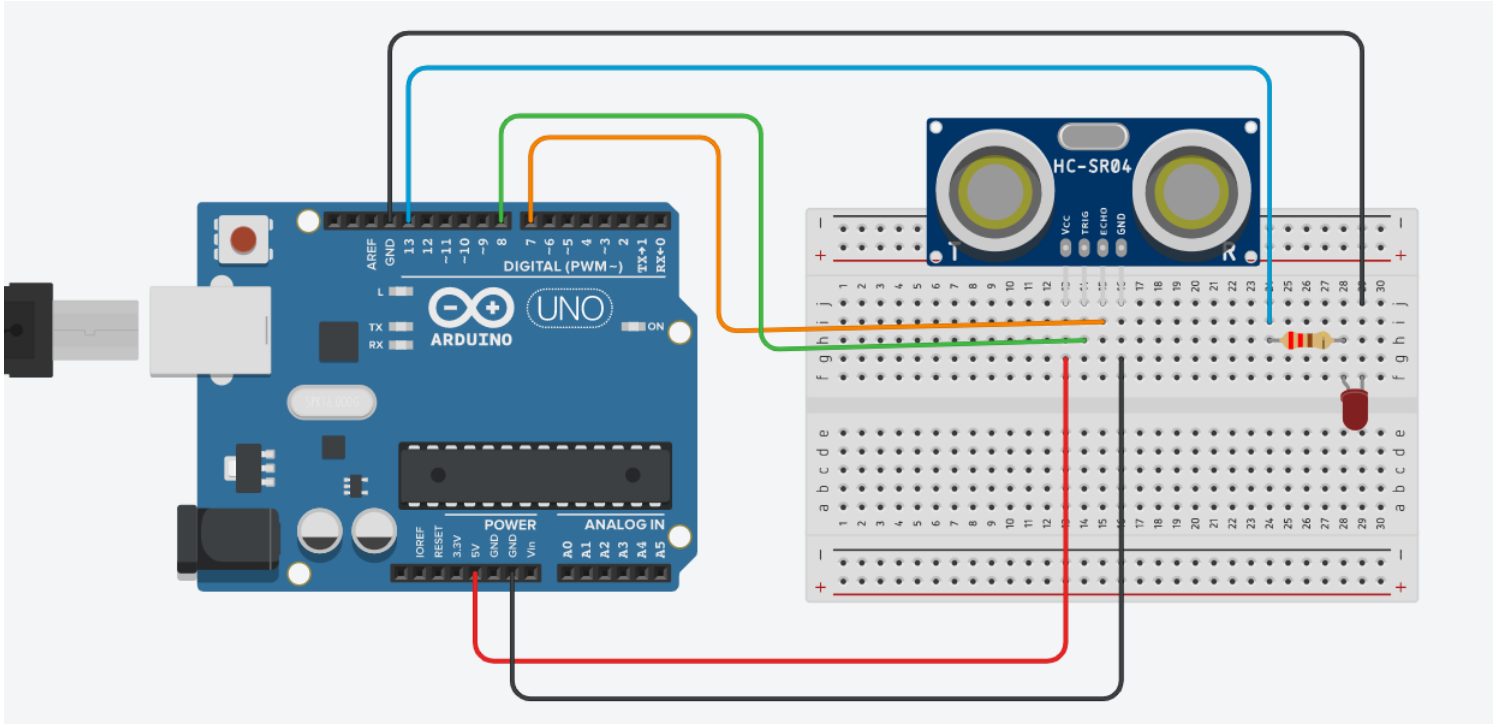
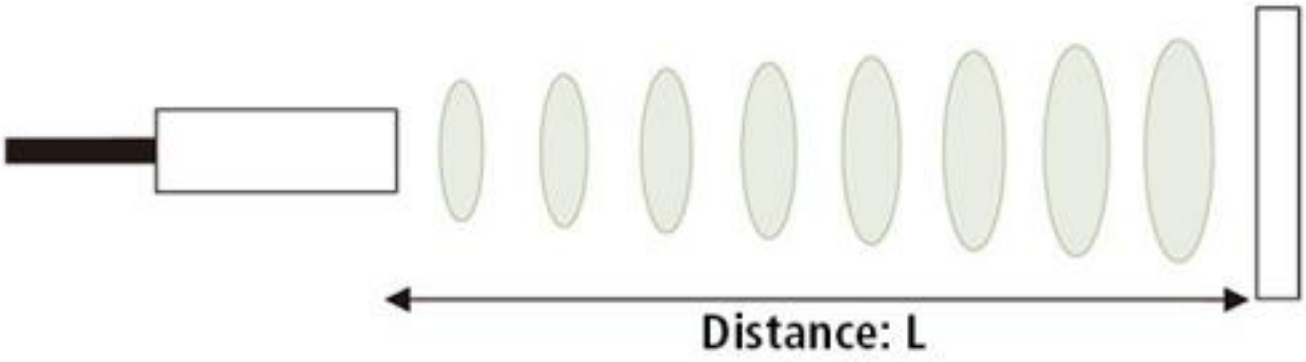
Program 4: Ultrasonic Sensor



As the name indicates, ultrasonic sensors measure distance by using ultrasonic waves. The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. Ultrasonic Sensors measure the distance to the target by measuring the time between the emission and reception.



An optical sensor has a transmitter and receiver, whereas an ultrasonic sensor uses a single ultrasonic element for both emission and reception. In a reflective model ultrasonic sensor, a single oscillator emits and receives ultrasonic waves alternately. This enables miniaturization of the sensor head.

Functions

Functions "Encapsulate" a task (they combine many instructions into a single line of code). Most programming languages provide many built in functions that would otherwise require many steps to accomplish, for example computing the square root of a number. In general, we don't care **how** a function does what it does, only that it "does it"!

When a function is "called" the program "leaves" the current section of code and begins to execute the first line inside the function. Thus the function "flow of control" is:

1. The program comes to a line of code containing a "function call".
2. The program enters the function (starts at the first line in the function code).
3. **All instructions** inside of the function are executed from top to bottom.
4. The program leaves the function **and goes back to where it started from.**
5. Any data computed and **RETURNED** by the function is used in place of the function in the original line of code.

Why do we Write Functions?

1. They allow us to conceive of our program as a bunch of sub-steps. (Each sub-step can be its own function. When any program seems too hard, just break the overall program into sub-steps!)

2. They allow us to reuse code instead of rewriting it.

3. Functions allow us to keep our variable namespace clean (local variables only "live" as long as the function does). In other words, function_1 can use a variable called i, and function_2 can also use a variable called i and there is no confusion. Each variable i only exists when the computer is executing the given function.

4. Functions allow us to test small parts of our program in isolation from the rest. This is especially true in interpreted langaues, such as Matlab, but can be useful in C, Java, ActionScript, etc.

Why do we Write Functions?

1. They allow us to conceive of our program as a bunch of sub-steps. (Each sub-step can be its own function. When any program seems too hard, just break the overall program into sub-steps!)

2. They allow us to reuse code instead of rewriting it.

3. Functions allow us to keep our variable namespace clean (local variables only "live" as long as the function does). In other words, function_1 can use a variable called i, and function_2 can also use a variable called i and there is no confusion. Each variable i only exists when the computer is executing the given function.

4. Functions allow us to test small parts of our program in isolation from the rest. This is especially true in interpreted langaues, such as Matlab, but can be useful in C, Java, ActionScript, etc.

Steps to Writing a Function

1. Understand the purpose of the function.
2. Define the data that comes into the function from the caller (in the form of parameters)!
3. Define what data variables are needed inside the function to accomplish its goal.
4. Decide on the set of steps that the program will use to accomplish this goal. (The Algorithm)

Parts of a "black box" (i.e., a function)

Functions can be called "black boxes" because we don't need to know **how** they work. Just what is supposed to *go* into them, and what is supposed to *come* out of them.

When defining a program as a black box, we must describe the following attributes of the function.

Note: *most documentation systems are just this, the attributes of a function with no code associated with it.*

1. The Name - describes the purpose of the function. Usually a verb or phrase, such as "compute_Average", or just "average".
2. The Inputs - called parameters. Describe what data is necessary for the function to work and gives each piece of data a **Symbolic Name** for use in the function.
3. The Calculation - varies for each function
4. The Output - Usually one (but sometimes zero or sometimes many) values that are calculated inside the function and "returned" via the output variables.

Function Workspace

Every function has its own **Workspace**. This means that every variable inside the function is only **usable** during the execution of the function (and then the variables go away).

Having a separate "workspace" for each function is critical to proper software engineering. If every function shared every variable in an entire program, it would be easy to inadvertently change the values of variables that you shouldn't. Further, it would be hard to remember what "names" have been used elsewhere, and coming up with new names to represent similar ideas would be challenging.

A side-effect of function variables not existing after the end of the function is that the only way to get information "out" of a function is by "returning" that information via the output of the function.

Additionally, the function can only "see" the information that is "passed" to it via parameters. Thus the only way information can get "in" to the function is by using parameters.

Note: *In certain object oriented languages (e.g., C++, Java, ActionScript), a function can also see all of the variables associated with its containing object.*

Sample Programing Calling a Function:

```
void setup() {

inches = ultrasonic (x);
}

long ultrasonic (long microseconds)
{
  Ultrasonic Sensor Code

  return microseconds
}
```

Variable x is passed to Function ultrasonic

Microseconds is returned back to Void Setup where the function was called and overwrite the variable x to what microseconds equals

Base Code

```
const int trigPin = 8; // Sends Sginal out from Ultrasonic Sensor
const int echoPin = 7; // Recieves Signal in for Ultrasonic Sensor

int led = 13;

long x, inches;         // long variable type holds more place holders for a real number than a double or float variable
                        // Variable x passes to the function ultrasonic to calculate distance
                        // Variable inches is set for distance when variable x is passed back from the function ultrasonic
void setup() {
  // put your setup code here, to run once:

Serial.begin (9600);
pinMode (led, OUTPUT);
pinMode (trigPin, OUTPUT);
pinMode (echoPin, INPUT);

}

void loop() {
  // put your main code here, to run repeatedly:

// Calls a Function to Use Ultrasonic Sensor
  inches = ultrasonic(x);   // Function to use ultrasonic sensor calcuations multiple
                            //times without having to write the code more than once
                            // Variable x is passed to function ultrasonic then
                            // becomes microseconds in the function

 Serial.print ("Inches=");  // Displays distance in inches from the function ultrasonic
 Serial.println (inches);
 delay (250);               // Slows the output of the ultrasonic sensor output.
                            // Change delay length to modify how fast the ultrasonic sensor reports distance value

 if (inches > 5)
 {
  digitalWrite (led, HIGH);
 }
 else
 {
  digitalWrite (led, LOW);
 }
}

long ultrasonic(long microseconds)    // Function to use the Ultrasonic sensor.
                                      // May use this code more than once without having to rewrite it,
                                      // by calling the functions above (setup or loop)
{
  // Following Code will turn on the sound from the trig side of the ultrasonic sensor
  // Sensor will pulsate the trig signal so when the sound wave hits an object
  // the sound can be recieved back in the echo side of ultrasonic sensor
  // trig signal will pass through the blank space between signals

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
```

```
  // pulseIn will receive the signal back as a unit of sound
  microseconds = pulseIn(echoPin, HIGH);

 // return will return a value that is converted sound wave to inches;
 // The return value will be set equal to variable x from the void loop when the function is called
 return microseconds / 74 / 2;
}
```

Assignment: Create the Following Colors Based on Distance of Ultrasonic

Wire a single RGB LED in place of 3 individual LEDS. See next page for wiring and code (NOTE: code uses a function called SetColor to send values 0-255 for each the Red, Green and Blue Spectrum)

Green: Object > 20

Blue: 5<= Object <=20

Red: 0 <= Object < 5

Optional: Add other color

Assignment: Add an RGB LED

Reference Tutorial RGB LED

Code:

```
int redPin = 11;
int greenPin = 10;
int bluePin = 9;

//uncomment this line if using a Common Anode LED
//#define COMMON_ANODE

void setup()
{
pinMode(redPin, OUTPUT);
pinMode(greenPin, OUTPUT);
pinMode(bluePin, OUTPUT);
}

void loop()
{
setColor(255, 0, 0); // red
delay(1000);
setColor(0, 255, 0); // green
delay(1000);
setColor(0, 0, 255); // blue
delay(1000);
setColor(255, 255, 0); // yellow
delay(1000);
setColor(80, 0, 80); // purple
delay(1000);
setColor(0, 255, 255); // aqua
delay(1000);
}

void setColor(int red, int green, int blue)
{
analogWrite(redPin, red);
analogWrite(greenPin, green);
analogWrite(bluePin, blue);
}
```

setColor is the name of the function, similar to void setup or void loop. Within the Void Loop setColor is called with 3 Values ranging from 0 to 255 that represent Red (R), Green (G) and Blue (B). These three values are sent to the void setColor Function below and become the values for the integers red, blue, green in that order. The values are then assigned to the specific pin location on the Arduino, mixing the values to create different colors.

Sample code show 6 different color combinations, but more can created by changing the RGB values between 0-255

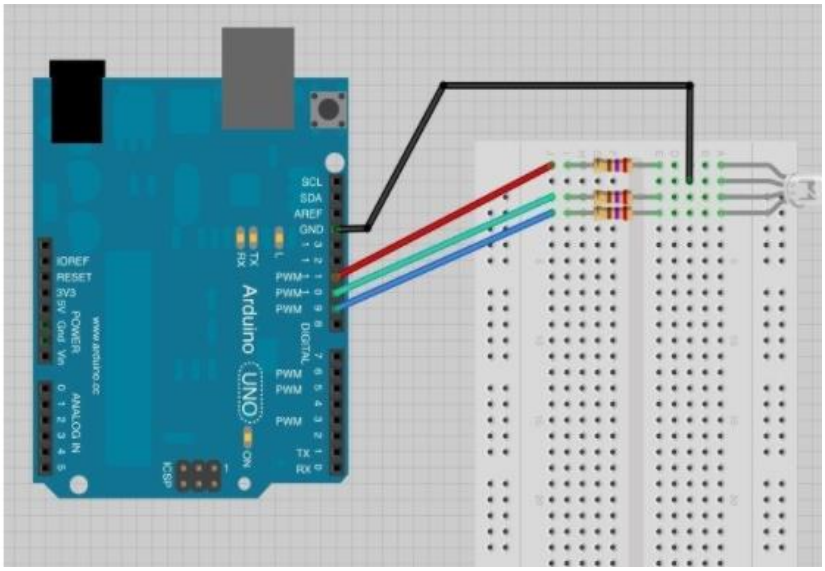
Sample Program to Call a Function

```
void setup() {

inches = ultrasonic (x);
}

long ultrasonic (long microseconds)
{
  Ultrasonic Sensor Code

  return microseconds
}
```

Variable x is passed to Function ultrasonic

Microseconds is returned back to Void Setup where the function was called and overwrite the variable x to what microseconds equals

Wiring Schematic



Note: Color Wires equal the pin color on the RGB LED