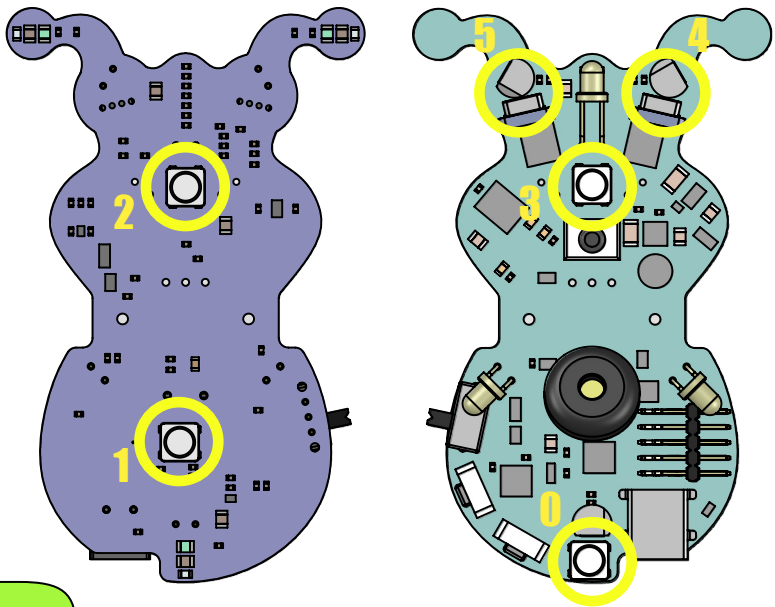
# Using Ringo's Color Lights

Ringo is fitted with six color lights. They are called "RGB" lights, which stands for "Red, Green, and Blue". Each light actually contains three small colored lights inside. You guess it - they are red, green, and blue. It turns out you can create any color you can think of by mixing different amounts of these three primary colors. Your TV screen creates colors in the same way.

The RGB Lights in Ringo were made popular by Adafruit Industries, coining the name "NeoPixels". We owe them a huge high-five for writing the code required to communicate with these awesome little lights.

Each light can have each of its red, green, and blue LEDs set to any level between 0 and 255. When set to 0, the given color is off, and when set to 255, the given color is on at max brightness. The NeoPixel lights are extremely bright, so a value from 100 to 150 is still plenty bright.

As stated, Ringo has six of these color lights. Computers like to number things beginning with 0. So the lights are actually numbered 0 through 5 (not 1 through 6 as you may expect). This address number of each light is printed next to it on the circuit board for easy reference. Check out the graphics below showing each light with its address.




1
SKILL LEVEL

# Using Ringo's Color Lights

### A few setup steps...

You need a few lines of code in your project to tell the Arduino IDE to compile in the super special Adafruit NeoPixel driver software. This code is already present in the example projects, so don't let this part scare you. The code belongs in your main project page above the line: `void setup(){`

Like this:

```
#include "RingoHardware.h"
```

This include line will insert the `RingoHardware.h` file, which itself includes a special line: `#include <Adafruit_NeoPixel.h>` By including `RingoHardware.h`, the IDE will automatically include the `Adafruit_NeoPixel.h` file which allows Ringo to talk to his lights.

```
void setup(){
  HardwareBegin();          //Sets all of Ringo's I/O pins correctly
  SwitchButtonToPixels();   //set shared line to control NeoPixel lights
  // (it's okay if there is more stuff below here)
}
```

### Time for fun! Let's Do This!

Now we're ready to make the lights go! You will control the lights with the function `SetPixelRGB();` Here is an example of how the function is formatted.

```
SetPixelRGB(address,red,green,blue);  //example formatting of function
```

You'll replace the word "address" with the address of the pixel you want to adjust. Remember this is a number between 0 and 5. Then you'll place a number between 0 and 255 in place of the words "red", "green", and "blue". The number you use will correspond to how bright or dim each of the three colors are.

# Using Ringo's Color Lights

If you have trouble remembering the address of each pixel, note that the address is printed on Ringo's circuit board right next to the light. You may also refer to each pixel by using key words as follows:

```
EYE_LEFT          //Pixel 5
EYE_RIGHT         //Pixel 4
BODY_TOP          //Pixel 3
BODY_BOTTOM       //Pixel 2
TAIL_TOP          //Pixel 0
TAIL_BOTTOM       //Pixel 1
```

You can use the key word interchanably with the pixel address when calling the SetPixelRGB() function. Like this:

```
SetPixelRGB( 3, 0, 0, 200);         //this code is the same
SetPixelRGB( BODY_TOP, 0, 0, 200);  //as this code
```

Lets see if we can make the light on the top of Ringo's body turn blue. You'll be writing most of your code inside the `void loop(){` function (at least until you start writing your own functions!). Make your `void loop()` function look like this:

```
void loop(){
  SetPixelRGB( 3, 0, 0, 200);  //set pixel 3 to 0 red, 0 green, and 200 blue
}
```

Now try to compile and load this code onto your Ringo. About a second after you turn on the power switch, the light on the front top of Ringo's body should be bright blue. Now let's talk about what happened, and why.

The `SetPixelRGB()` is taking four numbers, called "arguments" which control what the function does. The first number, in this case "3" tells Ringo's brain to target the pixel at address "3". The next three numbers tell the pixel at address 3 how bright to turn on the red, green, and blue lights inside the pixel. In this case, the red is set to 0 so it won't be on at all, as well as the green will be set to 0 so it won't be on either. The last number is 200, which corresponds to the blue led which will be turned on at a brightness of 200, which as you can see, is pretty bright!

# Using Ringo's Color Lights

Using the basic example above, we can start to play with the numbers in the function to change the behavior of the pixels. Let's say your favorite color is something different than red, or green, or blue. Like, PLUM! You can easily create different colors by mixing different amounts of the three primary colors Red, Green, and Blue. Try this example code. Compile it, load it, and observe what happens.

```
void loop(){
  SetPixelRGB( 3, 220, 30, 160);  //set pixel 3 to 220 R, 30 G, and 160 B
}
```

Sweet! Purple, right? I think you're getting the idea now. So what if you want to control more than one pixel at a time? What if we want to make both of Ringo's eyes turn purple? We just do the same thing, except we call the `SetPixelRGB` function more than once. We call it once for the right eye, and again for the left eye, then finally after both eyes have been commanded to a new value, we will latch the new colors by executing the `RefreshPixels();` function. Like this:

```
void loop(){
  SetPixelRGB( 4, 220, 30, 160);  //set pixel 4 (Right eye)
  SetPixelRGB( 5, 220, 30, 160);  //set pixel 5 (Left eye)
}
```

Purple eyes? Easy right? I think you've got the hang of it now. Now lets have some real fun. One more example then you can go off and play on your own.

```
void loop(){
  SetPixelRGB( 4, 220, 30, 160);  //set pixel 4 (Right eye)
  SetPixelRGB( 5, 220, 30, 160);  //set pixel 5 (Left eye)
  delay(100);
  SetPixelRGB( 4, 30, 220, 210);  //set pixel 4 (Right eye)
  SetPixelRGB( 5, 30, 220, 210);  //set pixel 5 (Left eye)
  delay(100);
}
```

*Example Sketch: Ringo_Guide_Pixels_01*

# Using Ringo's Color Lights

After you take that example for a spin, see if you can figure out what's going on. And what's up with `delay(100);` I added to the code? Any idea why the main function we're editing is called `loop()` ? Does anything in this example appear to be looping? Think it over and discuss it. Then see the "Discussion Answers" at the end of this section and see if you've got the right idea.

### Gotcha! A couple warnings to keep in mind...

In this section of each lesson, you'll find some helpful tips and warnings to keep in mind. Ringo is designed with robust components and he has some built-in protection systems to keep him safe. Here are some situations you may run into.

### Over Current Limit:

Each NeoPixel LED draws a current of 60 milliamps when all three of its colors (red, green, and blue) are turned on full brightness. If you did this to all six NeoPixels at once, then Ringo would need to draw about 360 milliamps from his battery. Battery current draw is beyond the scope of this lesson, but suffice to say, Ringo's battery can only safely supply about 200 milliamps. For this reason, all the parts on Ringo that are capable of drawing a lot of current (the NeoPixels, motors, sound chirp, underside edge sensors, and the three infrared emitters) are all powered through a limit switch that limits current to the safe 200 milliamps. Ringo's brain is however not powered through this switch - so his brain will usually remain active even during current limit. When the current limit is engaged, you will see the pixels and other current limited parts stop working or behave in strange ways. This is okay and does not cause them any harm. Just edit your code and reduce the brightness of your NeoPixels, speed of your motors, etc. In most cases you'll never hit this current limit as you're not usually running motors full speed while turning on lots of NeoPixels at high brightness. As you saw in this lesson, even at only 100 brightness, the light is still pretty bright, so going full blast to 255 is usually just wasting current.

### NeoPixel/Button Shared Line:

We based Ringo on the Arduino UNO as it is the most popular Arduino board. However, the processor on that board (and thus the processor in Ringo) is somewhat limited on how many signal lines it has. For this reason, we have doubled up on some signal lines.

# Using Ringo's Color Lights

The signal line controlling the NeoPixel LEDs is also used to sense if someone is pushing the User Button. When controlling the NeoPixels, this line is an output. When looking at the button, it is an input. By default it is set to control the NeoPixel lights.

Pressing the User Button will block communication to the NeoPixel lights while it is pressed. This does not harm the button, the NeoPixels, or Ringo's brain, but it is something you should be aware of. We've provided two simple functions that can be used to change the mode of this line as follows:

```
SwitchPixelsToButton(); //use this function before reading button
SwitchButtonToPixels(); //use this function to resume pixel operation
```

Note that SwitchButtonToPixels() is called automatically with all of the light functions, so it is not necessary to call it explicitly.

### It no workey. (Sad Ringo).

If you spend more than five minutes learning how to code, you will find your patience tested. Writing code is really pretty simple, but there is a small catch. Computers expect to see instructions formatted a very specific way. There are different rules in each computer language that must be followed. A piece of software that runs in the background on your computer (called a "compiler") reads your code and turns it into a bunch of 1's and 0's that make sense to Ringo's brain. The it is very easy to make the compiler grumpy, in which case it will throw a fit and refuse to do what it's told. The solution to keeping it happy is to pay close attention to how you format your code. Here are some common mistakes you may be making, and how to correct them.

```
SetPixelRGB( 1, 120, 50, 90);  //correct upper-lowercase function. This works.
setpixelRGB( 1, 120, 50, 90);  //no workey
setpixelrgb( 1, 120, 50, 90);  //no workey
SETPIXELRGB( 1, 120, 50, 90);  //no workey

delay(100);                    //correct delay. This works.
Delay(100);                    //no workey
```

Be sure you're using commas between the arguments.

```
SetPixelRGB( 1, 120, 50, 90);  //using commas correctly. This works.
SetPixelRGB( 1; 120; 50; 90);  //wrong. semi-colons between arguments. no workey
SetPixelRGB( 1 120 50 90);    //forgot commas between arguments. no workey
```

Be sure you're using normal parentheses around arguments.

```
SetPixelRGB( 1, 120, 50, 90);  //normal parenthesis. This works.
SetPixelRGB{ 1, 120, 50, 90};  //curley braces are wrong. no workey
SetPixelRGB[ 1, 120, 50, 90];  //brackets are wrong. no workey
```

Don't forget the semi-colon at the end of every line! (Yes, I realize there are some lines in the example code that don't have semi-colons at the end. The mean people who created the C language did this to confuse you. We'll discuss this later). For now, suffice to say that missing a semi-colon is the most sure way to upset the compiler.

```
SetPixelRGB( 1, 120, 50, 90);  //correct semi-colon. This works.
SetPixelRGB( 1, 120, 50, 90)   //oops. no semi-colon. upset compiler. no workey
SetPixelRGB( 1, 120, 50, 90),  //this is a comma, not a semi-colon. no workey
```

As pickey as the compiler can be, it is usually okay with extra blank spaces. Sometimes adding some extra space between arguments of a function can make the function easier to read.

```
SetPixelRGB(1,120,50,90);        //the compiler sees this the same as...
SetPixelRGB( 1, 120, 50, 90);    //this
```

**Discussion Answers:**

1) What's up with the `delay(100);` inserted into the code? Well, "delay" does just what you'd expect it to. It causes Ringo's brain go spin in a circle and do nothing for a certain amount of time. The argument passed is the number of milliseconds Ringo's brain should wait before proceeding. Keep in mind there are 1000 milliseconds in one second. So this line causes Ringo to pause for a tenth of a second before going ahead in the code.

2) Why is the main function we're editing called `loop()` ? As you probably noticed, Ringo keeps doing the same thing over and over. He sets his eyes to one color, then latches the color, then delays a short time, then sets his eyes to a different color, then latches the new color, then delays a short time... then he does it all again. The `void loop()` function runs all

the code contained within its curley braces. Once it reaches the end, it "loops" and does it all again. This loop function is where you'll be writing most of your code for a while (you'll eventually start writing your own stand-alone functions). Forget about the word "void" at the start of the line for now. Just know that it needs to be there and you don't need to know why at this point.

**Experiments & Challenges:**

Now that you've mastered controlling Ringo's color lights, go ahead and experiment. Here are some suggestions and challenges.

1) What happens if you play with the argument inside the `delay();` function? What happens if you make the delays different values?

2) What happens if you use the `SetPixelRGB()` function to control pixels at different addresses? Can you make Ringo's bottom lights work without any further hints from me?

3) Good job! You made the bottom lights work! What happens if you now take Ringo into a dark room and place him on different surfaces while the bottom lights are lit up? (Hint, look for something semi-opaque, a clear table top, and a white surface and see what happens).

4) What color do you get if you make all three Red, Green, and Blue values the same?

5) What happens if you run this code:

```
void loop(){
  SetPixelRGB( 4, 255, 255, 255);  //set pixel 4 (Right eye)
  SetPixelRGB( 5, 255, 255, 255);  //set pixel 5 (Left eye)
  RefreshPixels();
}
```

Woah! That's BRIGHT!!! Now take Ringo into a dark room. What do you see projected on the walls of the room? Why is this happening?

6) Remember the example where we made Ringo's eyes flash between two colors? Can you make his eyes flash between three different colors now? What about ten colors? Twenty? Try lots of different mixes of color and different delays.

7) Write the code on your own to turn on any NeoPixel then wait a short delay. How do you now make it turn back off? Can you make a certain NeoPixel blink on and off repeatedly?

8) Can you give Ringo oogley eyes? Try making a color alternate between the two eyes. (The right eye turns on one color while the left is not turned on, then you turn on the left eye to the same color while turning off the right eye - then loop).

9) Can you make Ringo look like his eyes are hypnotized?

10) Can you do something cool with Ringo's lights that we haven't even though of yet?

**Bonus Function:**

Now that you've familiar with the SetPixelRGB() function, we'll tell you about another function you can use to control the lights.

```
SetAllPixelsRGB(red,green,blue);  //example formatting of function
```

The `SetAllPixelsRGB()` function is useful for doing exactly what you would expect. This single function sets every light to the same value. It also automatically refreshes the pixels, so you don't need to call the `RefreshPixels()` after running it. This function is most useful for turning off all the NeoPixel lights at the same time. Occasionally, if Ringo is reset (or the User button is pressed) during a command to the lights, some of them may "stick" on with strange colors. Using `SetAllPixelsRGB()` with zeros is an easy way to make sure all lights are turned off, which clears any "stuck" pixels.

Like this:

```
SetAllPixelsRGB(0,0,0);  //turns off all NeoPixel lights
```

A few more examples...

```
SetAllPixelsRGB(0,0,50);  //all pixels to Blue
SetAllPixelsRGB(50,0,0);  //all pixels to Red
```

Use caution when using the SetAllPixelsRGB() function. As stated above, the pixels draw a bit of current, so trying to set them all to a high value at once will likely cause the over-current switch to trip. This isn't dangerous to Ringo, but it will make the lights go all wonkey on you until Ringo is reset and loaded with new code that doesn't draw so much current.

There are a handful of other similar functions as follows. Experiment with these on your own and you'll get the hang of using them.

```
OffPixels();                    //turns off all pixels
OffPixel(Address);              //turns off a specific pixel
OnEyes(Red, Green, Blue);       //makes eyes the given color
LeftEye(Red, Green, Blue);      //sets left eye to given color
RightEye(Red, Green, Blue);     //sets right eye to given color
OffEyes();                      //turns off eye lights
```