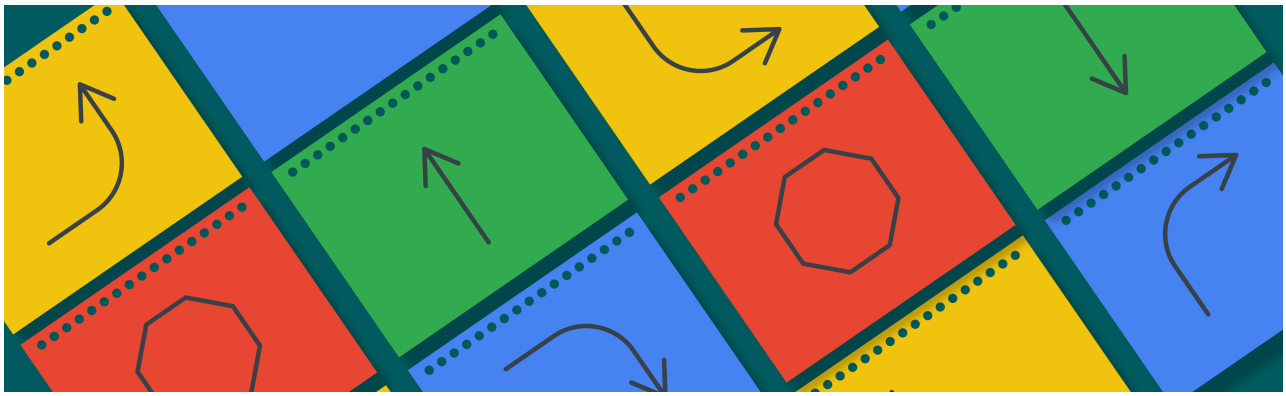
**3:0 HR**

# Colorful Reactions

Trigger actions using data from the RGB color sensor.



## Introduction

Remember when we experimented with the RGB LEDs to create various colors? On the underside of your robot, we actually have another RGB component, but this time it's not an output - it's an input called an **RGB Color Sensor** and we can use it to detect three colors. Just kidding, it can detect millions of color variations... at least in theory.

In this round, we build some very cool code. We'll teach Alvik how to scan and identify colors, all thanks to custom functions created by you! Once colors are understood, we can put them to use by changing your robots behavior based on the color detected.

To wrap up the lesson, you are challenged to complete the Rainbow Road, a fun course of colors that will test Alvik's ability to respond to its environment.

Piece of rainbow cake, right?

# Learning Objectives

- Understand how the RGB Color sensor works.
- Develop functions to scan, save, and check RGB values.
- Create and import a custom color module.
- Use defined functions to control robot outputs.

# Setting Up

Gather the materials below for this project:

- **Color Sheets of A4 or A5 Paper**
- **Notebook** (recommended)

You need a few sheets of color paper on your desk. We'll use these to test the color sensor and program smart reactions.

# RGB Color Sensor

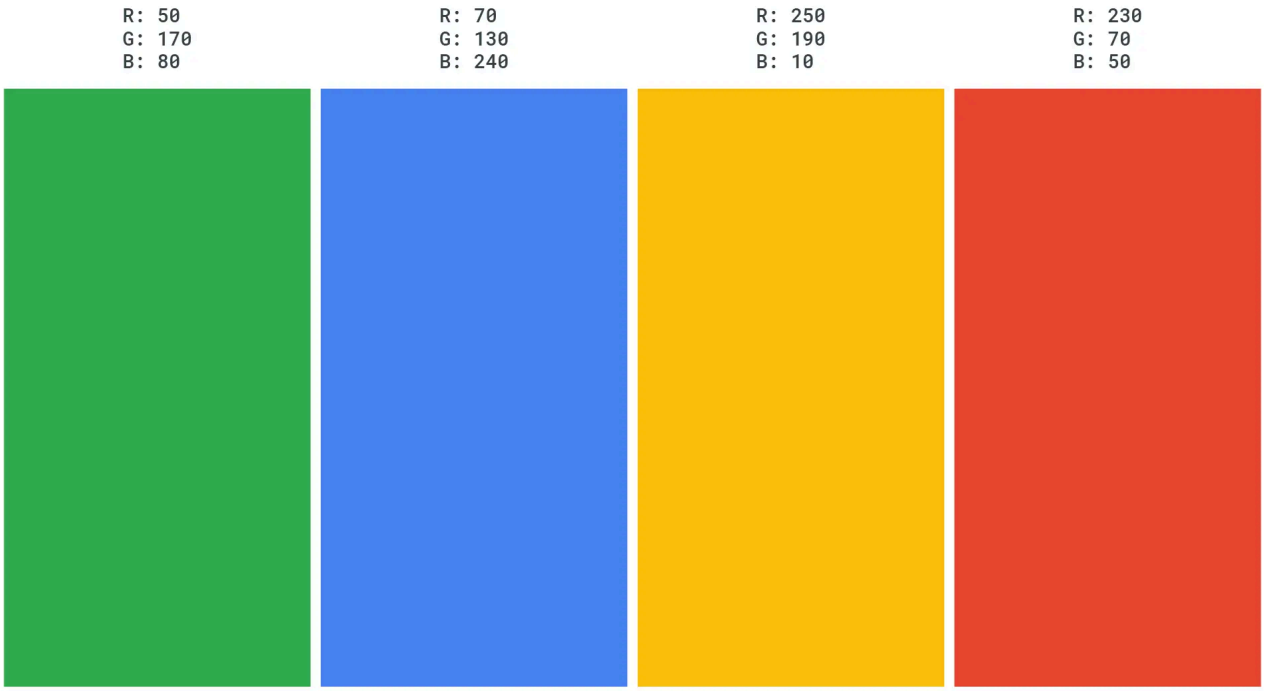

An RGB color sensor works by detecting intensities of red, green, and blue light. Here's a quick rundown:

**Light Detection:** The sensor contains small photodiode detectors for measuring red, green, and blue light. Recall that humans see colors reflected back to our eyes, meaning that something like a red apple is not technically red - rather, the apple absorbs all frequencies of light except the color red. Your robot's color sensor works in a similar way, measuring reflected light frequencies of the visible light spectrum.

If an object is the color green, for example, it means that it absorbs all non-green colors from the visible light spectrum and reflects the green frequency back to the color sensor.


**Color Depth:** Each color (red, green, and blue) is measured with a precision of 256 levels, thanks to the sensor's 8-bit resolution. That means colors can have intensity values from 0 (none of that color) to 255 (the maximum amount of that color).

**Color Combination:** The sensor combines these three colors in various amounts to detect a wide range of color shades. For example, if it senses 255 red, 100 green, and 100 blue, it interprets a shade of pink. Check out some more possible combinations below.



# Save Colors

## Sensor Test

Gather a few sheets of color paper on your desk, aiming to choose a good mix of primary and secondary colors like yellow, green, red, or blue. With the code below, we will use the `get_color_raw()` function to scan colors and print their values in the order (R, G, B).

As we have seen in previous lessons, there is more than one way to return data from a function. Uncomment the lines you prefer and test the script in Arduino Lab.

```
  # print(alvik.get_color_raw())

  # Option 2 - Assign values to a variable and print
  # colors = alvik.get_color_raw()
  # print(colors)

  # Option 3 - Unpack values and assign them to unique variables
  # r, g, b = alvik.get_color_raw()
  # Then, print extracted values separately or together
  # print(r)
  # print(g)
  # print(b)
  # print(r, g, b)

  delay(100)

def cleanup():
  alvik.stop()

start(setup, loop, cleanup)
```

Place Alvik on different sheets of paper, observing the changes in RGB values for each. On the color green, the values are likely around (200, 190, 130), while on the color red, the values may be (290, 140, 130). If you are reading different values, that is completely to be expected. Keep in mind that environmental lighting affects readings and not all color paper is exactly the same.

Whatever the values are in your case, know that they are perfect as they are. We understand what colors the values represent - all we need to do now is help Alvik understand as well.

## Building the "Save Color" Function

Imagine you could place Alvik on any color, push a button, and boom - the color is now saved. Now, any time the robot sees that color, it could do something special like spin around or flash its lights. Such a program is definitely possible and we are going to make it happen!

We first need to define a friendly function that will guide users through the color-scanning process. Let's start by giving the function a name and setting up a few important commands.

```
def save_color():
  print("Place Alvik on a color.")
  print("Push the OK checkmark button on Alvik to read color.")
  alvik.left_led.set_color(1, 1, 1)
  alvik.right_led.set_color(1, 1, 1)
  button_ok = alvik.get_touch_ok()
```

This function is called `save_color()` and it has no parameters within parentheses. Once called, the user needs clear instructions on next steps. The first line prints these instructions to the terminal, explaining that Alvik should be placed on a color and the OK checkmark button must be pushed to read that color.

Next, we turn on both RGB LEDs to signal the scanning process is ready. When running cable-free without access to the terminal, this provides a visual cue that Alvik is ready to scan and waiting for the user to push the OK button.

At last, we initialize the variable `button_ok` with the method `alvik.get_touch_ok()`, which we later use to check if the button is touched or not touched. When the checkmark button is touched, the button state changes from `False` to `True`, and vice versa.

```
def save_color():
  print("Place Alvik on a color.")
  print("Push the OK checkmark button on Alvik to read color.")
  alvik.left_led.set_color(1, 1, 1)
  alvik.right_led.set_color(1, 1, 1)
  button_ok = alvik.get_touch_ok()

  while not button_ok:
    button_ok = alvik.get_touch_ok()
```

```
    delay(100)
    if button_ok:
        r, g, b = alvik.get_color_raw()
        color = (r, g, b)
        alvik.left_led.set_color(0, 0, 0)
        alvik.right_led.set_color(0, 0, 0)
        print("R, G, B readout saved!")
        print(color)
        delay(500)
        return color
```

The next part of our function improves the experience by providing more user control, waiting for you to confirm with a button-press before the sensor takes the color reading. We achieve this by using the loop `while not button_ok:` which is the same as saying `while button_ok == False:` , or in pseudo-code language, "if the OK button is not pressed yet, keep repeating the loop".

Each time the loop repeats, notice how we update `button_ok` to check if the button has been pressed. If the button is pressed, `button_ok` changes from `False` to `True` and the condition `if button_ok` essentially converts to `if True` , thus executing the following:

**1.** Sensor reads RGB values with `alvik.get_color_raw()` and stores them in `r, g, b` .

**2.** Variables `r, g, b` are packed into a tuple list assigned to the `color` variable.

**3.** LEDs are turned off to confirm the button press.

**4.** Confirmation and readout values are printed to the terminal.

**5.** Delay for a moment to pace between new color scans.

**6.** The function returns the values stored in `color` .

## "Save Color" Function Testing

Now it's time to test the `save_color()` function to see how it works. If you haven't done so already, create a new file and save it to your projects folder.

```
    alvik.begin()
    delay(1000)

    # Save as many different colors as you like to unique variables
    global color_1
    print("Ready to scan Color 1")
    color_1 = save_color()

    global color_2
    print("Ready to scan Color 2")
    color_2 = save_color()

def loop():
    print("Results:", color_1, color_2)
    delay(15000)

def cleanup():
    alvik.stop()

start(setup, loop, cleanup)
```

We initialize variables for each color scanned and saved inside of `def setup():`. As you can see, the `save_color()` function is called and returns the `(r, g, b)` values of `color`, which are assigned to the variable `color_1`. Once completed, the process repeats again for a new color with the variable `color_2`.

> **NOTE:** You probably noticed that we declared color variables globally like this: `global color_1`. That's because variables only exist locally within the scope of the function they are inside of, which is `def setup():` in this example. In order to use these variables in other functions like `def loop():`, they must become global.

At last inside of our main loop, we have a quick print test to see if everything is working properly. You should see the two scanned results appear in the terminal.

## Check Colors

Alvik can now scan and save colors, but it would be so much cooler if it could react to different colors as well. In this section, we will develop one more function to help your robot do exactly that. Let's get to work on a new function called

`check_color()`, with the goal to look for a specific color and return `True` if that color is detected.

```
1   def check_color(color_name, color_read, tolerance):
2     r_cn, g_cn, b_cn = color_name
3     r_cr, g_cr, b_cr = color_read
4     r_match, g_match, b_match = False, False, False
```

The function has three main parts that we need to breakdown. In the first part above, take a look at the three parameters which require arguments when the function is called: `color_name`, `color_read`, and `tolerance`. The parameter `color_name` is where we place the saved color we are looking for - for example, the variables `color_1` or `color_2` that we saw in the last section.

`color_read` is the live reading from the color sensor, gathered by the object `alvik.get_color_raw()`.

In both cases, we unpack their values to become `r_cn, g_cn, b_cn` for `color_name` and `r_cr, g_cr, b_cr` for `color_read`. Later, you will see how we use `r_match, g_match, b_match` to decide if the live color reading matches our saved color values, but for the moment, we simply initialize these to `False`.

```
1   def check_color(color_name, color_read, tolerance):
2     r_cn, g_cn, b_cn = color_name
3     r_cr, g_cr, b_cr = color_read
4     r_match, g_match, b_match = False, False, False
5
6     if r_cr >= (r_cn - (r_cn * tolerance)) and r_cr <= (r_cn + (r_cn * tolera
7         r_match = True
8     if g_cr >= (g_cn - (g_cn * tolerance)) and g_cr <= (g_cn + (g_cn * tolera
9         g_match = True
10    if b_cr >= (b_cn - (b_cn * tolerance)) and b_cr <= (b_cn + (b_cn * tolera
11        b_match = True
```

In the next part of the function, we have three conditions that are virtually the same, except you will notice that each condition focuses on a specific RGB color value. The goal here is to take a color reading and compare it to a saved color's RGB values. If the RGB values of the color reading are the same or nearly the same as the saved color's RGB values, we can conclude that the color is a match.

In order to do this, we check if the color reading is within range of the saved color's values. Imagine, for example, an R value is read at 98 and the R value of a saved color is 100. Since readings fluctuate slightly, we can't exclusively use readings that perfectly match 100, so the solution is to add a small tolerance. Adding a tolerance of +/- 5% to the saved color value of 100 would mean that any scanned R value between 95 and 105 would be considered a color match. Thus, in this example, an R value of 98 would be considered a match at 5% tolerance.

Let's plug these example values into the our first condition to better understand the math.

```
if r_cr >= (r_cn - (r_cn * tolerance)) and r_cr <= (r_cn + (r_cn * toleranc
```

**Here is how it would look:** if 98 >= (100 - (100 * 0.05)) and 98 <= (100 + (100 * 0.05)).

- **Challenge:** Imagine you change the tolerance to 1% - would the scanned value of R = 98 still be in range?
- **Challenge:** What happens if you increase the tolerance too much, maybe to 15% or more? Will you have reliable color matching?

Take time to carefully reflect on the code so far. As soon as you feel fully comfortable with the logic, keep moving to see the final part of the function.

```
def check_color(color_name, color_read, tolerance):
  r_cn, g_cn, b_cn = color_name
  r_cr, g_cr, b_cr = color_read
  r_match, g_match, b_match = False, False, False

  if r_cr >= (r_cn - (r_cn * tolerance)) and r_cr <= (r_cn + (r_cn * tolera
    r_match = True
  if g_cr >= (g_cn - (g_cn * tolerance)) and g_cr <= (g_cn + (g_cn * tolera
    g_match = True
  if b_cr >= (b_cn - (b_cn * tolerance)) and b_cr <= (b_cn + (b_cn * tolera
    b_match = True

  if r_match and g_match and b_match:
    return True
  else:
    return False
```

We can only confirm a full color match if all three RGB values match a saved color within the defined tolerance, and that is where the final conditional statement comes into play. Previous functions check individual RGB values, but we now need to compare all three values to confirm the match.

Recall that `r_match`, `g_match`, and `b_match` were initialized as `False`, but as you'll see in the conditions checking R, G, and B values, these variables change to `True` when each respective condition is true. With our final condition `if r_match and g_match and b_match`, we check if all three variables are `True`, and if so, our function returns `True` to confirm that we have a full color match.

# Fun Reaction

## Testing Time

Alright, that was enough code to make our brains pop! Now for the fun part where we put it all together. Copy the code below and don't forget to save your script.

```
55
56 def loop():
57   delay(1000)
58   get_color = alvik.get_color_raw()
59
60   is_color_1 = check_color(color_1, get_color, 0.05)
61   is_color_2 = check_color(color_2, get_color, 0.05)
62
63   if is_color_1:
64     print("It is Color 1!")
65   elif is_color_2:
66     print("It is Color 2!")
67   else:
68     print("Unknown Color")
69
70 def cleanup():
71   alvik.stop()
72
73 start(setup, loop, cleanup)
```

Take time to review what is happening inside of the main loop. Reflect on the code inside of `check_color()` and look carefully at the three arguments in the

function. It is important to think through what's happening step-by-step until you feel that exciting sense of confidence gained from understanding the logic.

At last, of course, try it out! Run the code and save two colors - then, try placing Alvik on different color surfaces and observe activity in the terminal.

Have fun with it! Try scanning different surfaces like a banana peel or your teacher's head... just joking!

## Tidy Code

It is a common practice in coding to keep everything as tidy and organized as possible. Placing the newly defined functions directly in our script is fine, but it's an eyesore that makes our code feel busy.

For better organization, simply copy your two newly defined functions into a new script called *color_functions.py* and click save. Now, we have a special color module that you can import to future scripts with `import color_functions`. Once imported, the module brings all the same functionality of your color functions to the new script. Alternatively, you may prefer to add these functions to the file *my_functions.py* created in a previous lesson.

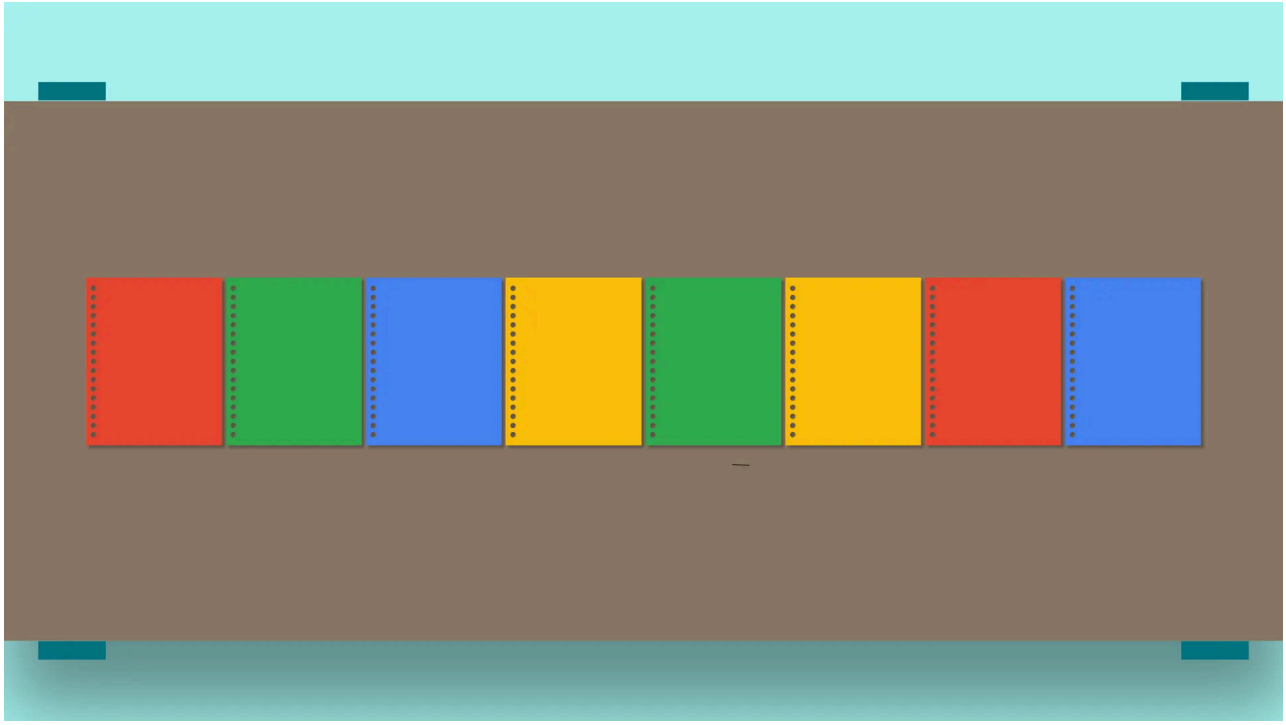
Try updating our last script with an imported module like the example below. Looks way better, right?

```
  color_2 = save_color()

def loop():
  delay(1000)
  get_color = alvik.get_color_raw()

  is_color_1 = check_color(color_1, get_color, 0.05)
  is_color_2 = check_color(color_2, get_color, 0.05)

  if is_color_1:
    print("It is Color 1!")
  elif is_color_2:
    print("It is Color 2!")
  else:
    print("Unknown Color")

def cleanup():
  alvik.stop()

start(setup, loop, cleanup)
```

# Rainbow Road

Now the most fun part! Alvik can respond to different colors, meaning we have the possibility to program movements based on colors detected. This is crucial to navigating the rainbow road from start to finish.

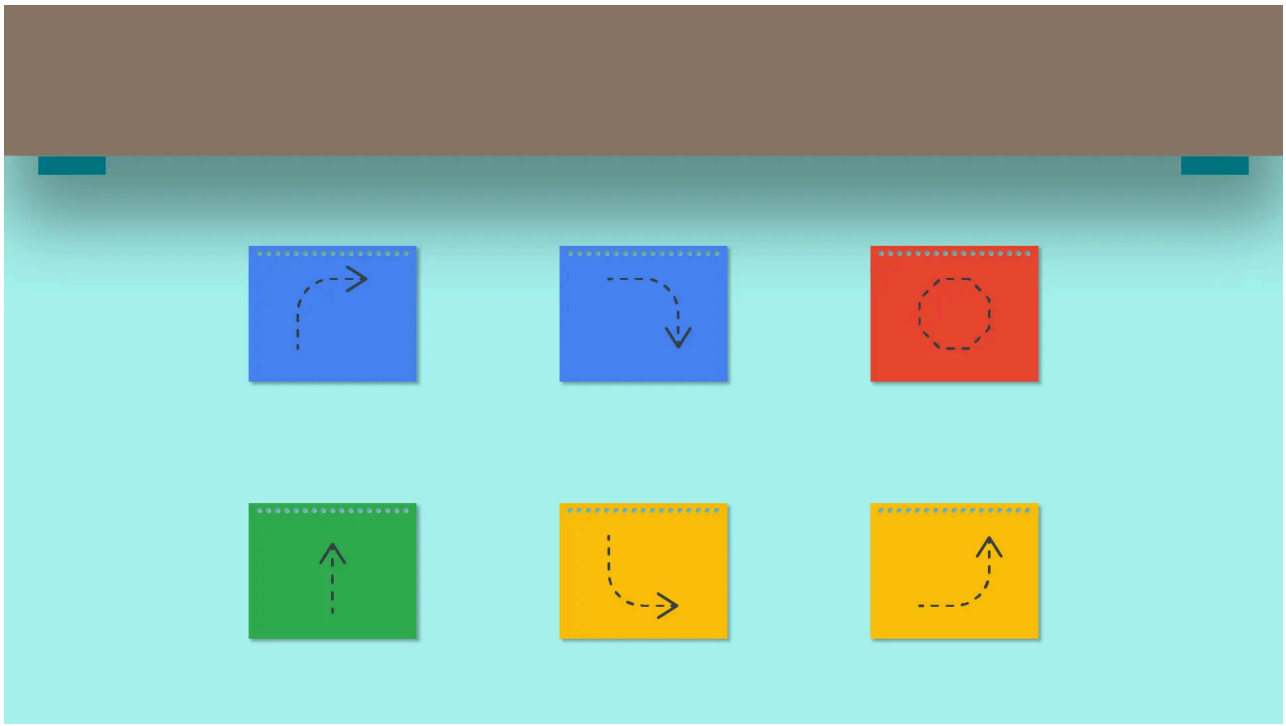
### Level 1



Place 10-20 random sheets of paper in a line, alternating different colors along the way. Using the data from its color sensor, you must program Alvik to stop on the final sheet of color paper (for example, the blue sheet) - no distance measurements or special motor functions allowed! Remember to use your custom color functions to scan and save color values before writing the main program.

**Tip:** Think about how you could use variables to count the colors detected along the path. Each time a color is detected, the count variable for that color would increase by one.

## Level 2



Arrange sheets of color paper on the floor in a way that creates an invisible path between each sheet. When Alvik passes over a certain color, it must complete a specific movement - for example, crossing green could mean "drive forward" while crossing yellow might mean "turn left". Your goal is to navigate this invisible route, touching each sheet of paper along the way until the finish line.

**Bonus:** Can you make your robots LEDs change to the same color of the paper detected?