Lab – 9

Deadline – Friday, 10/28/2022, 1PM

Introduction

In this lab you will be learning about different types of colorblindness.

The three types of colorblindness that we are addressing in this lab are protanopia, deuteranopia, and tritanopia. People with protanopia are unable to perceive ‘red’ light. People with deuteranopia are unable to perceive ‘green’ light. People with tritanopia are unable to perceive ‘blue’ light. All of these are oversimplifications of how colorblindness really works. This lab may certainly suggest that a certain color palette may not work for a type of colorblindness when that is not really the case.

If you are curious about the different types of colorblindness you can click [**here**](https://www.colourblindawareness.org/colour-blindness/types-of-colour-blindness/#:~:text=People%20with%20protanopia%20are%20unable,blues%20and%20yellows%20stand%20out.)

The reason we are talking about colorblindness in this lab is because it is important to be aware of the colors you are using when developing something such as a website. This is when inclusive design comes into play. When developing a website, for example, you want to make sure to utilize specific colors in order to make sure the website’s visitors will be able to understand all the content.

If you are curious about hexadecimal, read this section. Otherwise, you can jump to step 1. If this is confusing, don’t worry too much about the values actually being stored by hexadecimal, but it can be helpful to have a grasp of alternate numbering schemes. In this lab, we will be utilizing hexadecimal to represent colors. A color can be represented using three values: red, green, and blue. Each of these are typically between 0 and 255, inclusive. A single hexadecimal character can represent a number 0-15, much like how a single decimal character can represent a number 0-9. Counting in hexadecimal looks like

0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 14 15 16 17 18 19 1A 1B … FD FE FF

FF is 255, the max value we will be using in this lab. Because each color (red, green, and blue) can be represented using two hexadecimal characters, a combination of these can be represented using six hexadecimal characters. For example, “FFFFFF” is white, as all three values are maxed at 255. You can experiment with creating colors using hexadecimal [**here**](https://www.google.com/search?q=rgb+color+picker&rlz=1C1CHBF_enUS756US756&oq=rgb+color+picker&aqs=chrome..69i57j0l7.5322j0j1&sourceid=chrome&ie=UTF-8).

Step 1 - Look Through Lab

This lab has a lot to it, but it is simpler than you think! Make sure to look through the entire lab before taking it step by step.

Step 2 - Reading get\_red

We have written getRed for you, so you have an example. Please read through it!

For this function you will extract the red component from a String. This is a helper function and will come in handy later.

To understand this function, we must first introduce slices. This is when we take a range of indices from a string (or list).

For example:

example **=** 'ABCDEF'

**print**(example[0]) *# prints 'A'*

**print**(example[0:2]) *# prints 'AB'*

**print**(example[3:6]) *# prints 'DEF'*

Testing

You should go ahead and write some tests in run\_tests() to make sure get\_red() is written correctly. This will give you practice before moving on.

Here’s an example for testing:

**def** **run\_tests**():

**print**(get\_red('FFAAAA'))

The output will be 255 because the first two indexes were extracted, FF and was converted from hex into decimal, which equals 255.

Step 3 - Writing get\_green

This function is very similar to the one you wrote above however it is extracting the green component from a String. This is a helper function and will come in handy later.

Inside the function body you must first get the 3rd and 4th character out of the parameter by using a slice (index positions 2 and 3). Then you need to cast the string to hex number. This can be done by using int(string\_to\_convert, 16).

Testing

You should write some tests in runTests for get\_green(). Maybe use the [**color picker**](https://www.google.com/search?q=rgb+color+picker&rlz=1C1CHBF_enUS756US756&oq=rgb+color+picker&aqs=chrome..69i57j0l7.5322j0j1&sourceid=chrome&ie=UTF-8). When you select a color, the second item in the RGB value is the green value! So using hex, ‘4d6285’ the getGreen should return 98.

Step 4 - Writing get\_blue

This function you will extract the blue component from a String. This is a helper function and will come in handy later.

This function is almost identical to the previous two functions. The difference is that you need to extract the last two characters in the input string (index positions 4 and 5).

Testing

You should write some tests in runTests for getBlue().

Step 5 - Writing id\_protanopia

This function returns true or false depending on whether or not if the given color is unaffected by Protanopia.

In the function body we are going to use one of the helper function we just wrote. First create a variable and assign it to the red value of the color passed in. Then return the result of checking if this is less than 64.

If the value of red is *less* than 64, return True, otherwise return False.

Example for creating a variable and assigning it to the return value from another function: (This has nothing to do with this program, it is just an example)

**def** **appleCount**(numApples): *# helper function*

total **+=** numApples

**return** total

**def** **findTotalApples**():

totalApples **=** appleCount(numApples) *# creating a variable and assigning it to total from the helper function*

In this case, and for the following functions, arbitrary values are selected for determining how these colors affect color blindness. This is an oversimplification for the sake of the lab.

Testing

When writing tests for this function, make sure you pass in values some where red is less than 64 and some where red is greater!

Step 6 - Writing id\_deuteranopia

This function returns true or false depending on whether or not if the given color is unaffected by Deuteranopia.

In the function body we are going to use one of the helper functions we just wrote. First create a variable and assign it to the green value of the color passed in. Then return the result of checking if this is less than 64. This will be the same as the function above, but you will check the value of green instead of red.

Testing

Always add tests for every function after you write them!

Step 7 - Writing id\_tritanopia

This function returns True or False depending on whether or not if the given color is unaffected by Tritanopia.

In the function body we are going to use all three of the helper functions we wrote. First, create a variable for each component of the color passed in and use the helper functions to assign them the appropriate values.

We will only return false from this function if several conditions are met. You can choose to write these as nested if statements or by combining conditions using the *and* operator. The three conditions we are looking to satisfy are:

* The blue component is greater than zero
* The red component is greater than 230
* The green component is greater than 220

If all of these conditions are met, return false. Otherwise, return true.

Here is an example of combining conditions using the *and* operator:

**if** apples**>**5 **and** oranges**==**10 **and** peaches**<**15:

**return** False

Here is an example of using nested if statements:

**if** apples**>**5:

**if** oranges**==**10:

**if** peaches**<**15:

**return** False

Both of these do the exact same thing.

Testing

Always add tests for every function after you write them! This one has a range of values to test, so you probably need at least three to five tests to make sure it is correct.