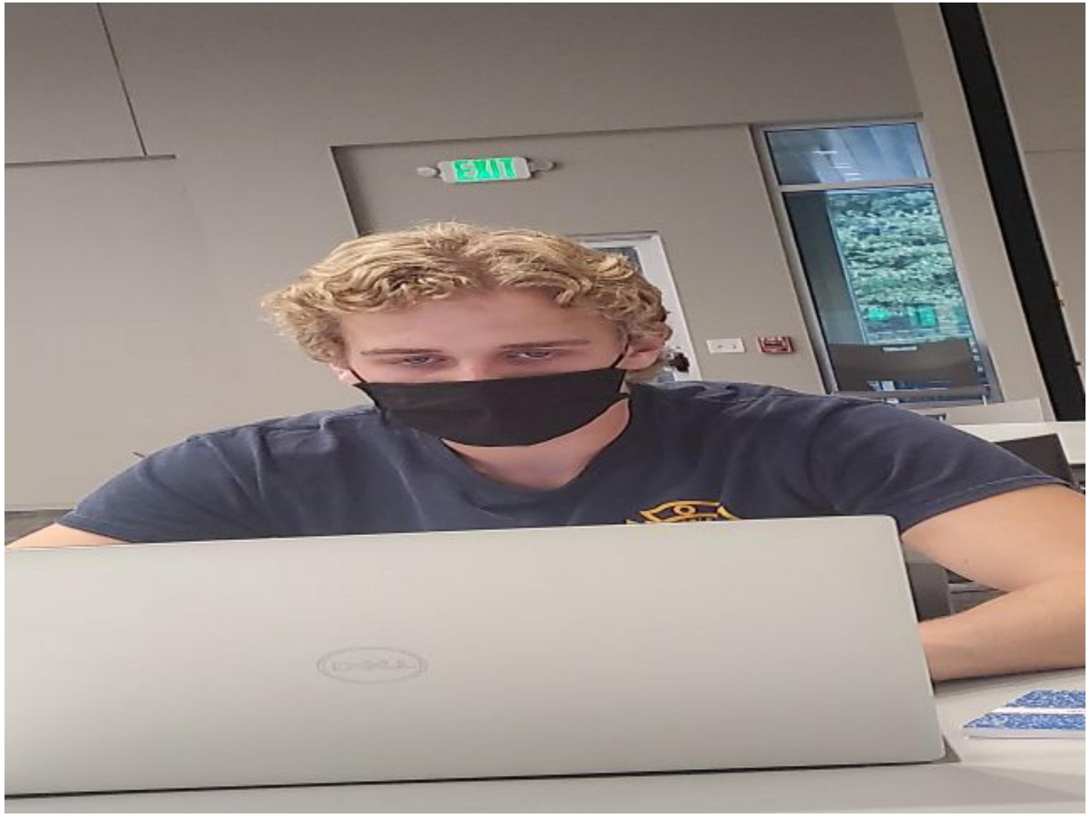
Graphics Programming

Assignment 01



Original Image (Spencer Lile, Hard at Work)

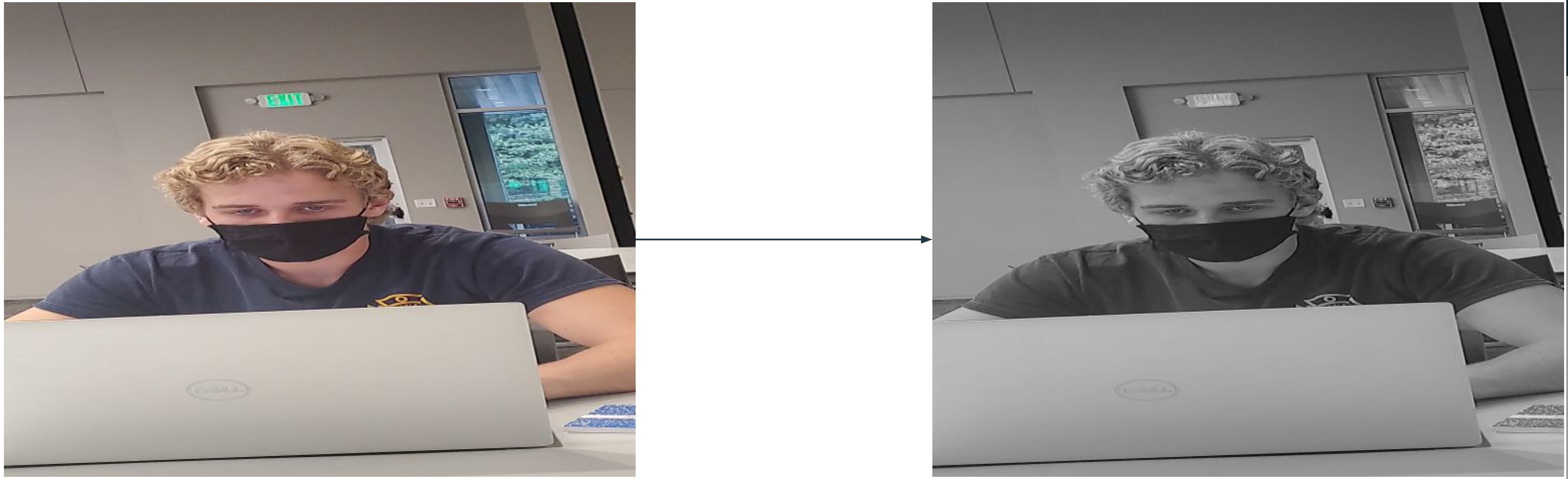
Greyscale

The Code:

```
out = img*1
out = img[:, :, 0]*0.1+img[:, :, 1]*0.7+img[:, :, 2]*0.2
return np.uint8(out)
```

First, a copy of the image is made with img*1. Then each color channel is given a certain weight by being multiplied by a certain weight, and those are averaged into a single color space. The number in that space represents a percentage of black & white mixed together.

Greyscale



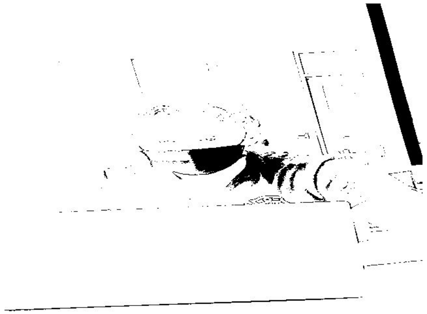
Black&White

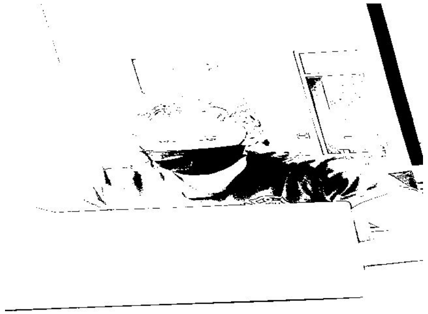
The Code:

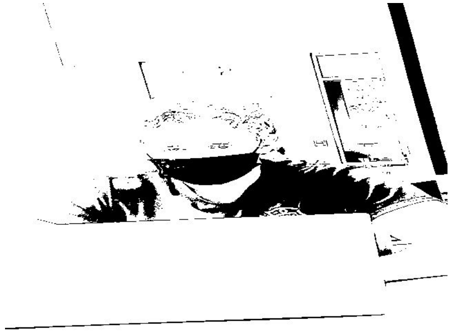
```
out = greyscale(img)
out[out > cutoff] = 255
out[out <= cutoff] = 0
Return out
```

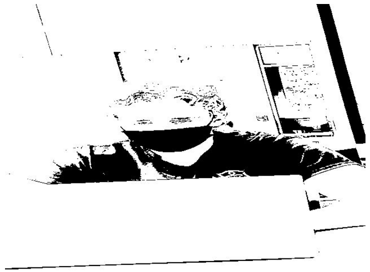
A black and white image is similar to a grayscale image; the only difference is that where greyscale contains a percentage of black and white, true black and white images have pixels either 255 or 0. The point of the cutoff variable in the code is to determine at what pixel value the pixels become either black or white.

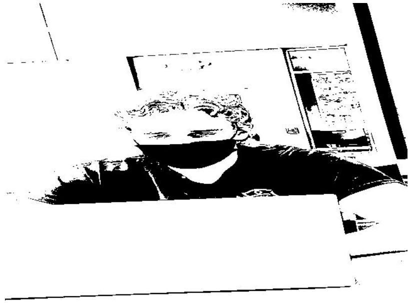
Black&White

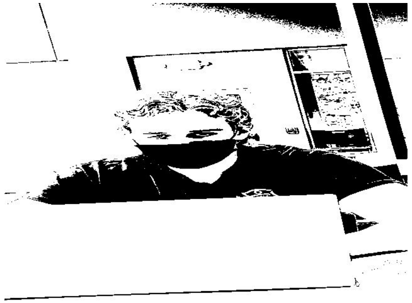

Cutoff = 48


Cutoff = 58

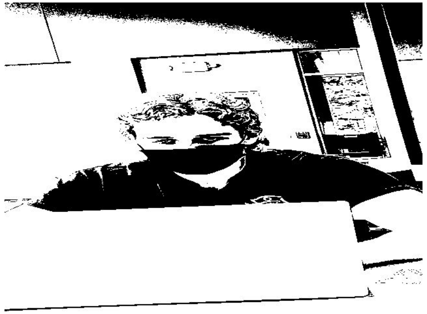

Cutoff = 68

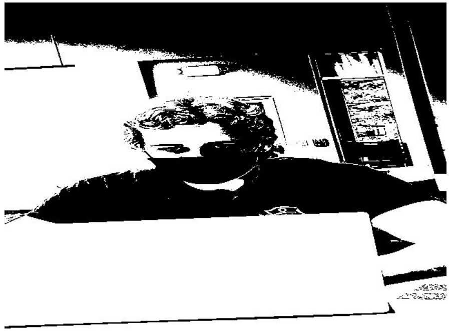

Cutoff = 78

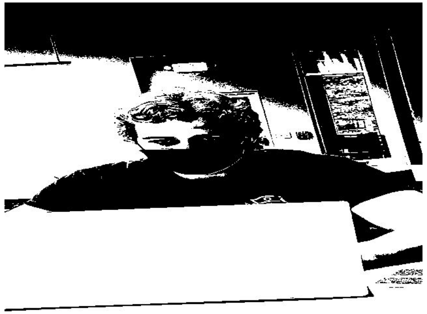

Cutoff = 88

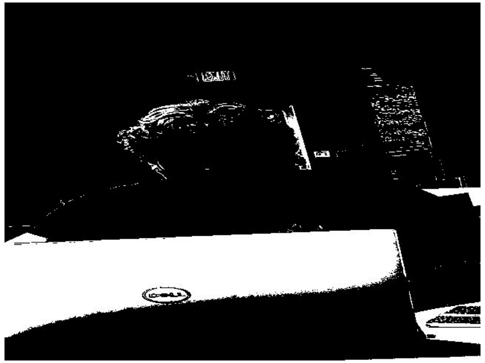

Cutoff = 98

Black&White


Cutoff = 108


Cutoff = 128


Cutoff =138


Cutoff = 188

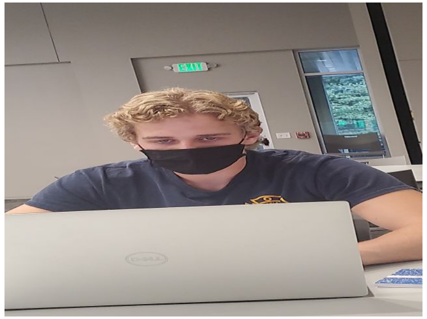
Desaturate

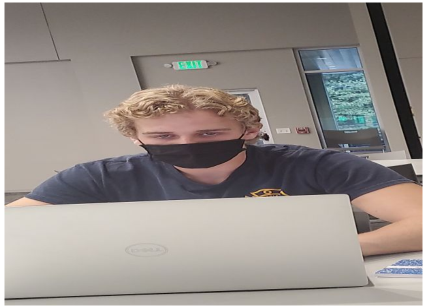
The Code:

```
out = (greyscale(img)[:, :, None]*percent)+img*(1-percent)
return out
```

A desaturated image is a certain percentage of a greyscale added to the remaining percentage of a color image, which is what the code above shows. The only quirk is that greyscale returns an image without a 3rd dimension, and to add the two images you need to add a third dimension by doing [:, :, None].

Desaturate


0% desaturation


10% desaturation

20% desaturation

30% desaturation

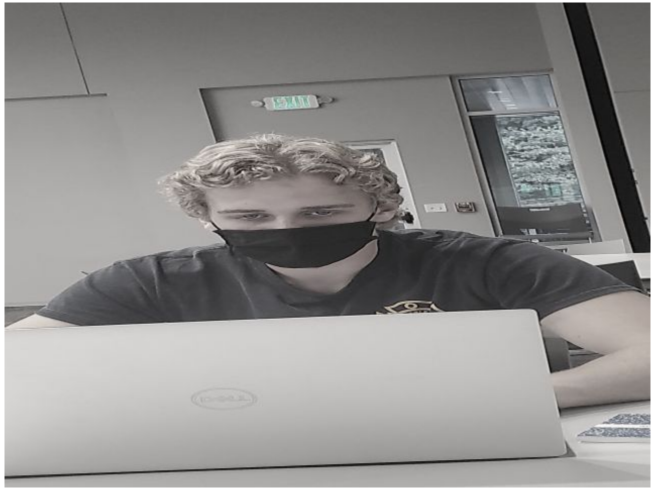
40% desaturation

50% desaturation

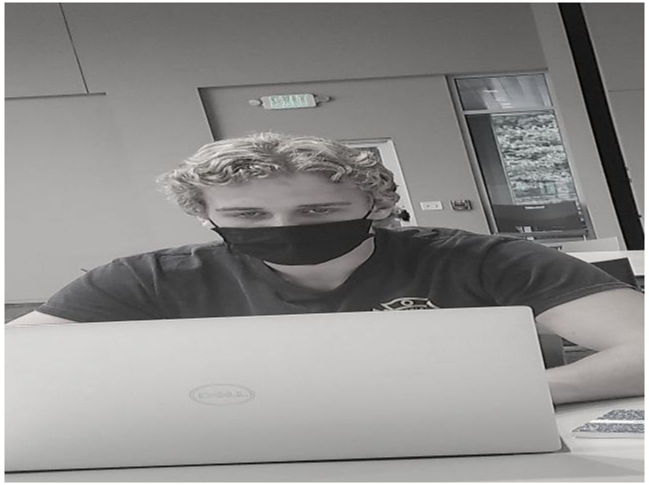
60% desaturation

70% desaturation

Desaturate



80% desaturation



90% desaturation

100% would just be a greyscale image

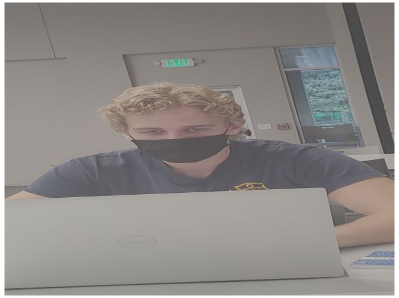
Contrast

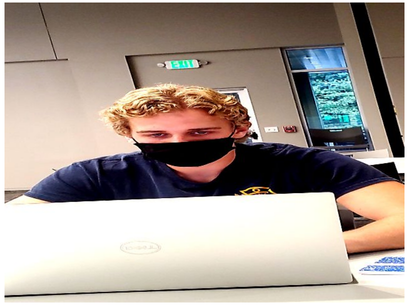
The Code:

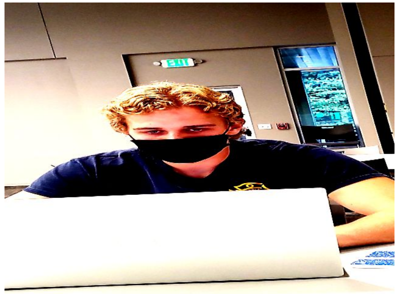
```
out = img*1.0
out = (out - 128)*factor + 128
out[out > 255] = 255
out[out < 0] = 0
```

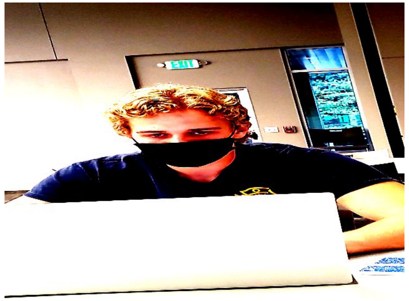
Contrasting an image means scaling pixel values by the factor away from 128. So pixel value 115, factored by 2, would become 102. The last two lines prevent any pixel from overflowing in case they are contrasted by some crazy silly factor. At 0, the image becomes completely grey; this is because every pixel becomes the same greyish neutral color. At high extreme values, I would describe the image as looking very ‘ warm’. Negative numbers would look very hellish, and result in the colors being kind of inverted.

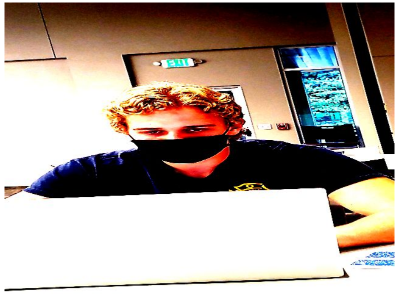
Contrast

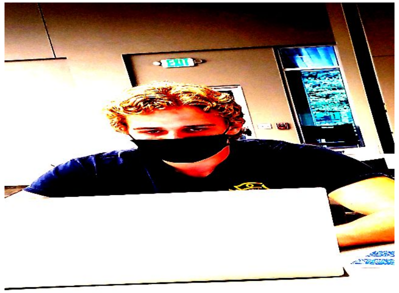

Factor of 0.5

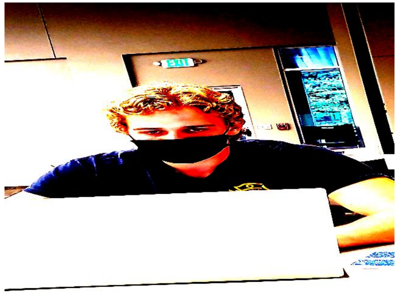

Factor of 2

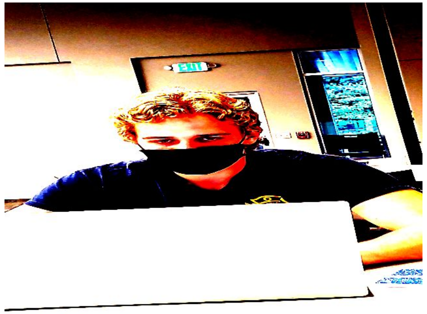

Factor of 3


Factor of 4

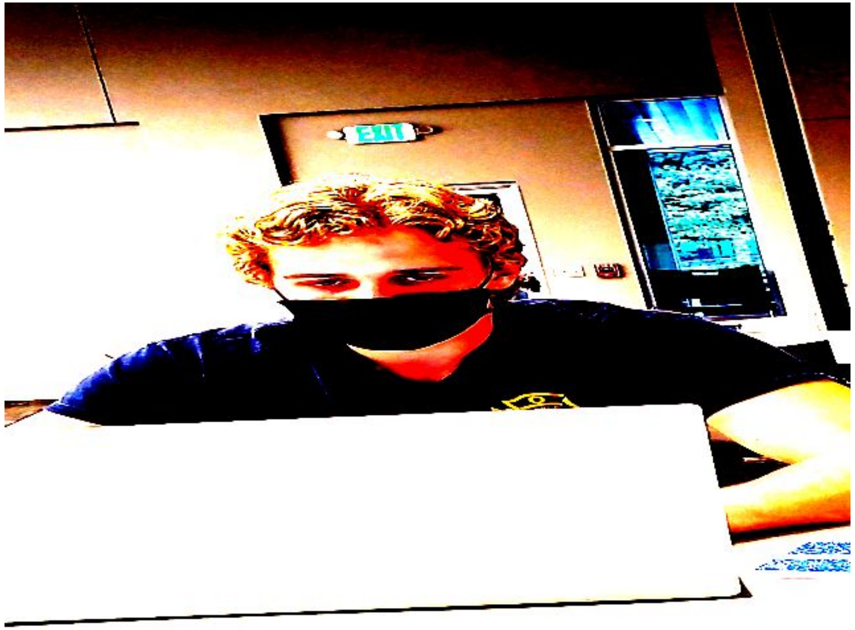

Factor of 5

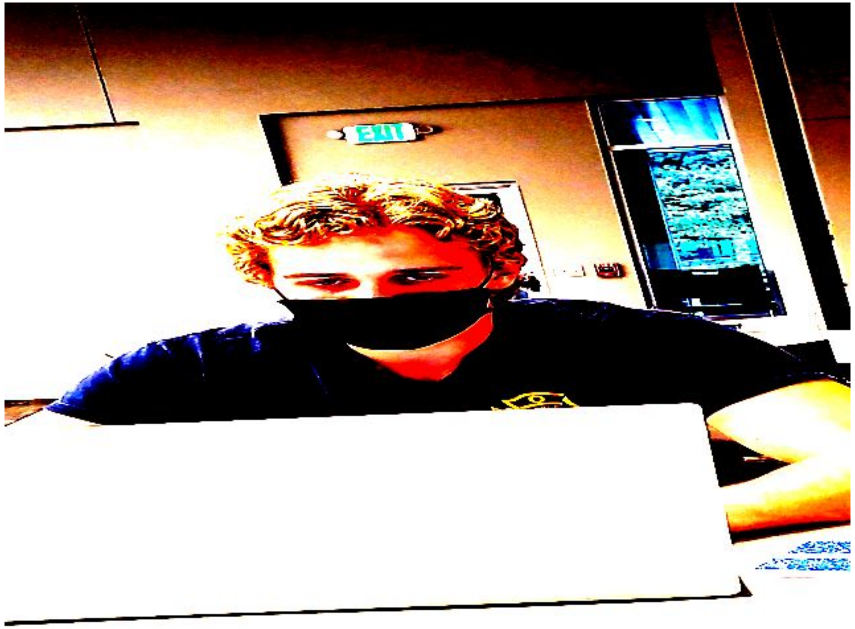

Factor of 6


Factor of 7


Factor of 8

Contrast





Factor of 9

Factor of 10

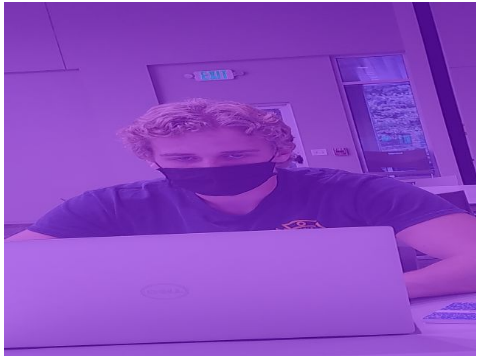
Tint

The Code:

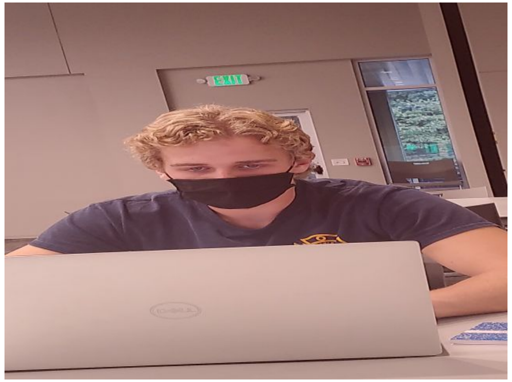
```
color_img = img*1
color_img[:, :] = color
out = np.uint8((color_img*percent)+img*(1-percent))
```

To tint, it is the same concept as desaturating except the grayscale image is replaced with a custom colored image. Color_img can be a scalar value, like 255, or something else such as (255, 128, 2) or something like this.

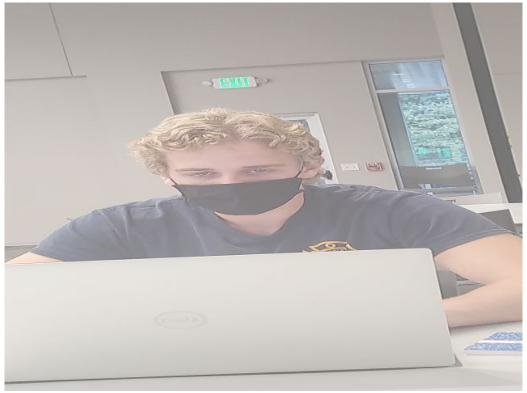
Tint



(255, 0, 128), 50%



(0, 0, 255), 10%



255, 40%