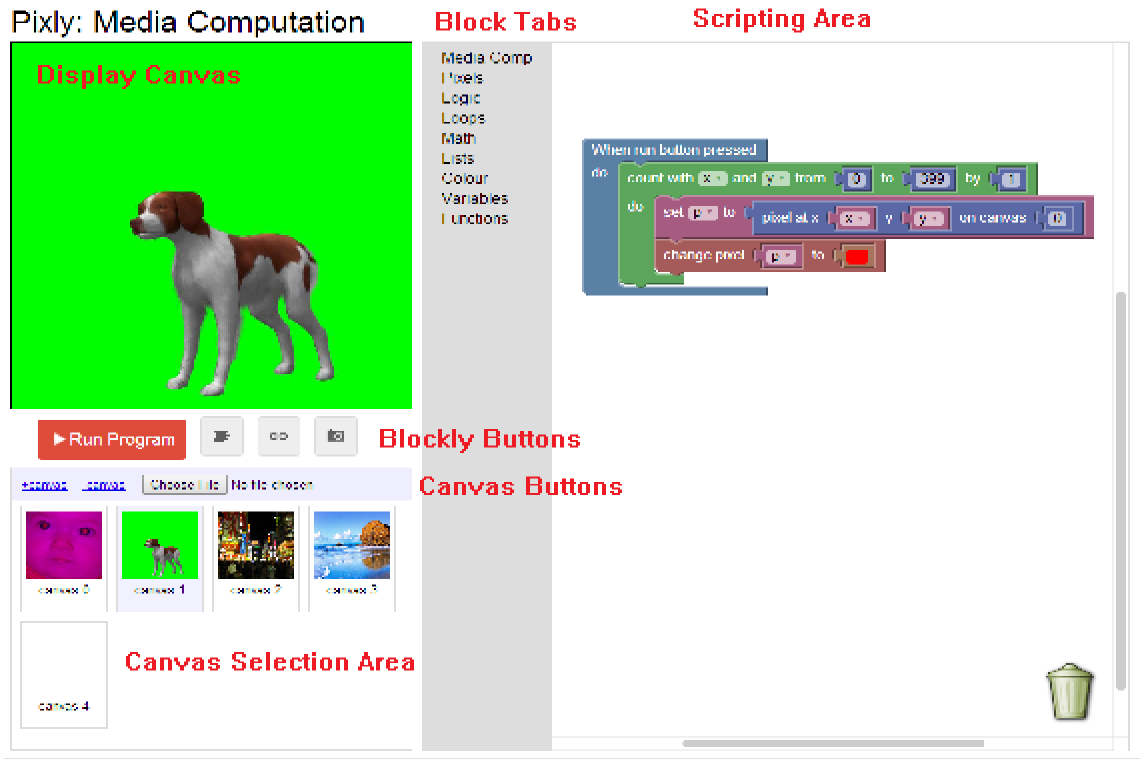
Pixly: User Manual

Pixly is a custom extension to the **Blockly** web-based, graphical programming editor. Introducing blocks specific to canvas image manipulation, Pixly allows people to create syntactically correct program with blocks to programmatically manipulate the pixels of an image. This allows for all sorts of media computation applications like red-eye removal, green-screening, negatives , and more.

I. Layout of the Pixly Application

When Pixly begins, you will see a large image in the **Display Canvas** on the left of the screen, with some buttons and smaller images below, and the **Scripting Area** on the right of the screen.



The *scripting area* on the right is where you will drag and drop your blocks to create your pixly program. The *display canvas* on the left is where you will see the result of your image manipulation program.

II. Selecting, Adding and Removing Canvases

Pixly has multiple canvases displayed in the **Canvas Selection Area.** You can click on any one of these to make that canvas displayed as the *display canvas*. Each canvas in the *canvas selection area* is numbered, and this number is the one you will reference in your blocks to specify which canvas(es) you are programmatically changing.

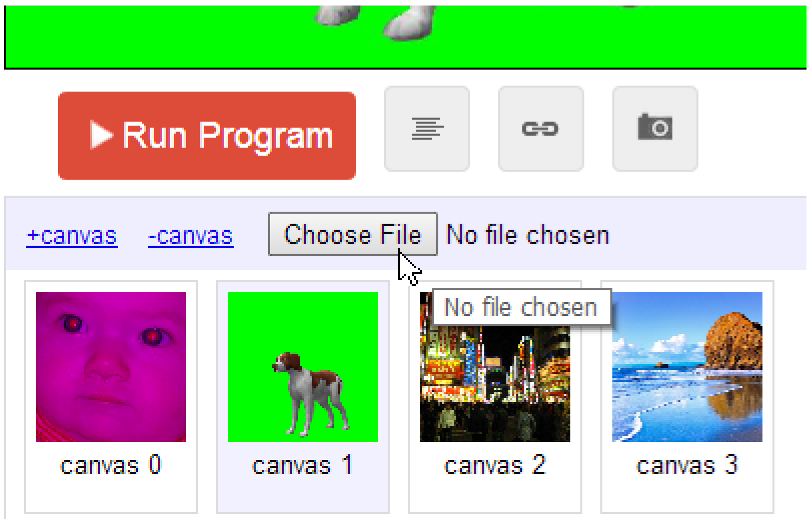
Blank canvases can be added with the **+canvas** button in the **Canvas Buttons** area, and the currently selected canvas can be hidden with the **-canvas** button.

II. Uploading User Images to the Canvas

It is possible to upload images from your own hard drive to edit.

In the *canvas buttons* area, there is a button labeled "**Choose File**". Clicking on this button will open up a file explorer dialogue that will allow you to select the image you wish to upload onto your Pixly Canvas.

Currently, because Pixly only supports 400 px by 400 px canvases, your image will be scaled to these dimensions upon uploading.



WARNING: Uploading an image will overwrite whichever current canvas you have selected. *(i.e. if the green dog canvas is selected (and is being displayed as the main canvas) when you upload an image, the green dog image will be drawn over with the new image, and will no longer exist!).* For this reason, it is suggested to add a new blank canvas (with the +canvas option) to upload your image onto.

III. Exporting/Importing Blocks and Saving the Canvas

In the **Blockly Buttons** area, there are four buttons: one large, red **Run Program** button, which you press to execute the blockly program you create in the *scripting area*, one blue **Export/Import Blocks** button and the **Code** and **Capture** buttons.

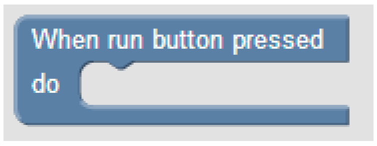
The Export/Import Blocks button will open a dialogue allowing you to copy the xml structure of the blocks in your scripting area for later use, or allowing you to paste in the xml structure of blocks you copied earlier to quickly Import block programs.

The leftmost grey button is the *code* button, and clicking it will show you the javascript code generated from the blocks you arranged in the *scripting area*. The rightmost button is the *capture* button, and will save the current display canvas as an image on your hard drive.

IV. Pixly Specific Blocks

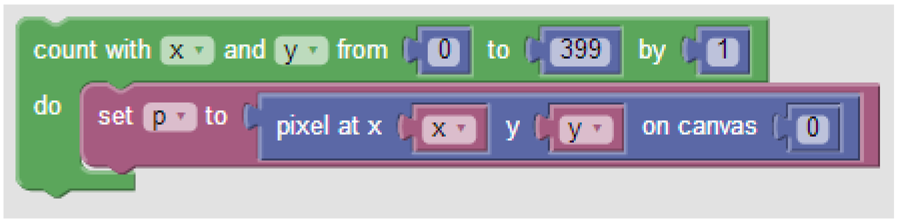
Now we'll go through all of the blocks custom created for the Pixly media computation application.

A. Blocks under the Media Comp tab



The first block under the media comp tab, and the only block that is placed on the scripting area whenever Pixly is started. This block is where any blocks that you want to be executed when you press the run button need to be placed.

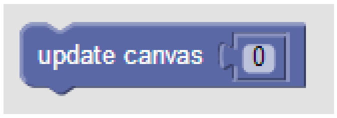
Any blocks that are outside of this run button block are considered scratch and will not be have any direct effect on the canvases. (**NOTE:** an exception is any **function** created outside of the *run* block. These will not be executed directly, but can be referenced from inside of the *run* block and will function properly).


count with x and y from 0 to 399 by 1
do
set p to pixel at x x y y on canvas 0

This is one of the most essential blocks for media computation: the **main media computation loop**. Generally, this block combination will be placed at the beginning of the run program, and any blocks that manipulate pixels will be placed inside of the "**count with x and y**" block under the "**set p to pixel**" block (*see example code at the end of this manual*).

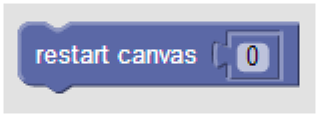
Altogether, this block combination is a nested for loop which will go through every single pixel of the specified canvas, reading from left to right, top to bottom. It then sets the variable **p** equal to the pixel at the current x and y coordinates in the loop. Most blocks that deal with pixel manipulation in pixly will refer to **p** by default, so it is generally important to put these blocks below the *set p to pixel* block but still inside of the *count with x and y* wrapping block, so that they can work on every pixel in the canvas.

In Pixly's current state, no modifications to the *count with x and y* wrapping block are necessary, as all canvases are 400 pixels by 400 pixels. However, the rightmost portion of the *set p to pixel* block combination, labeled "0" above, should be edited to specify which canvas you want to look at pixels from in the loop.



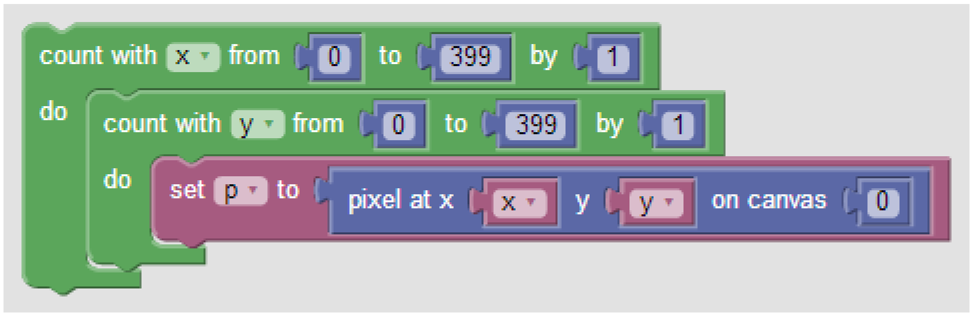
When you run your block program, the canvases will be updated at the end of execution without any further prompt. However, this **update canvas** block will update the display of the canvas so that you can see intermediary changes to the canvases. The number on the right should be changed to whatever canvas you want to update the display of.

(**NOTE:** currently, this block doesn't execute correctly, so it will have no effect on your program. This is being worked on and will likely be fixed in the next version of Pixly)



This block simply redraws the original image of the canvas. For instance, if you edit the green dog image on canvas 1, but wish to have the original image back again, you can use this block. The number on the right should be changed to whatever canvas you want to restore.

(**NOTE:** upon uploading an image to a canvas, the **restart canvas** block will use that new uploaded image as the original image of the canvas).

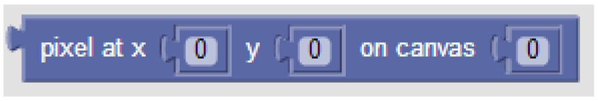


```
count with x from 0 to 399 by 1
do  count with y from 0 to 399 by 1
    do  set p to pixel at x x y y on canvas 0
```

This block combination functionally acts the same as the second block sequence in the Media Comp tab (*count with x and y -> set p to pixel*)

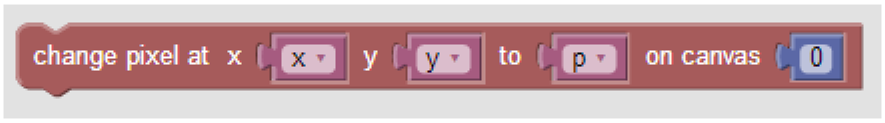
The difference between the two is that instead of combining x and y start and end coordinates of your media computation loop into one block wrapper, this separates them into two: this allows you to further specify which portions of the image you want to edit (*i.e. if you only want to perform a computation on the top half of the image, you could use this block combination and change the **second number** in the **count with y from 0 to 399** block wrapper to **199** instead of 399).*

B. Blocks Under the Pixels Tab



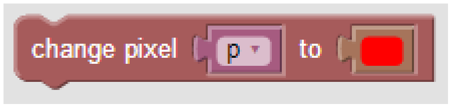
The **get pixel** block. This block is, by default, included in the *main media computation loop*.

This block will return a Pixel object from the specified x and y coordinates from the specified canvas. A pixel object is a representation of a pixel on a canvas image, and is composed of three values: **r**, **g**, and **b**, which correspond to the **red**, **green**, and **blue** values of the specific pixel. *(http://en.wikipedia.org/wiki/RGB_color_model for more information)*. These values can be retrieved from other blocks in the pixel tab.

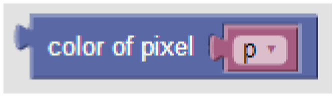


This block will change the pixel at the specified x, y coordinate on the specified canvas to specifed value.

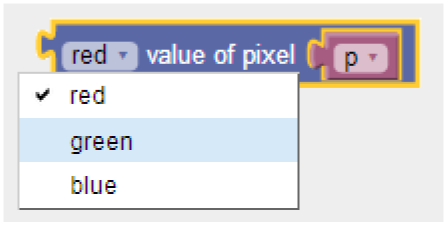
The x and y blocks can be replaced with number blocks from the Math tab to specify a specific x and y coordinate not inside of a loop. The pixel value can either be a pixel variable (for instance, **p**), or a colour block from the colour tab). This block can be used in conjunction with the main media computation loop to fully or partially copy one canvas onto another.



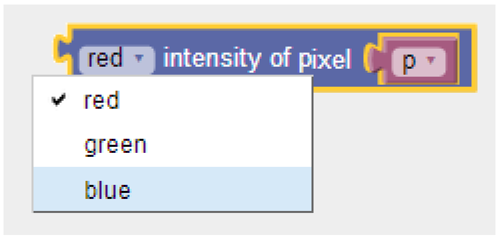
This block will change the specified pixel to the specified value. The pixel on the left needs to be a pixel variable created from the *get pixel* block (either directly, by inserting the **get pixel** block into the left slot, or indirectly, by first creating a variable (like in the *main media computation loop*). The value on the right can either be a colour block from the colour tab, or another pixel.



This block will return the colour value of the specified pixel. The color value acts the same as any colour block from Blockly's colour tab, and is represented in code as a hexadecimal string of the color.



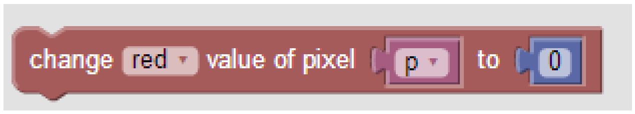
This block will return the **red**, **green**, or **blue** *value* of a pixel. RGB values range from 0 to 255.



This block will return the **red**, **green**, or **blue** *intensity* of a pixel. Color **intensity** is calculated as follows:

R Intensity: (red value) / ((blue value + green value) / 2)
G Intensity: (green value) / ((blue value + red value) / 2)
B Intensity: (blue value) / ((red value + green value) / 2)

This block is very handy for red-eye removal or green-screening algorithms.

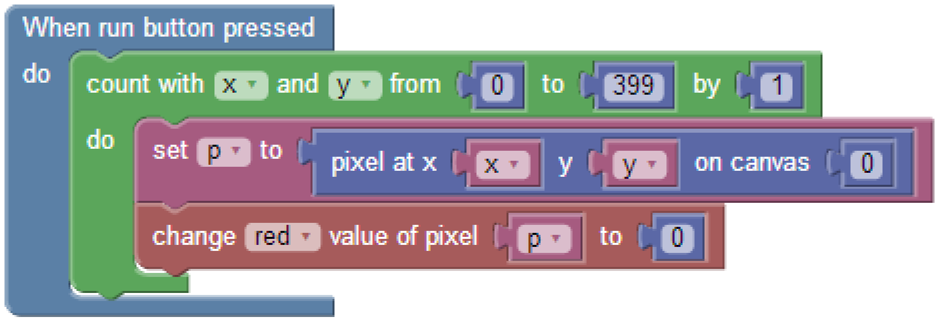


This block can be used to set the **red**, **green**, or **blue** *value* of a pixel. RGB values range from 0 to 255.

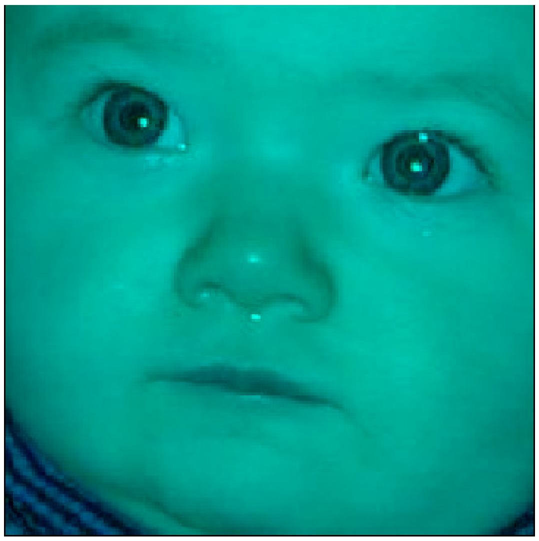
For instance, if you wanted to make a picture look cerulean (get rid of all the red in the picture), you would use this block to set the red value of every pixel to 0, which would still maintain their blue and green values.

V. Code Examples

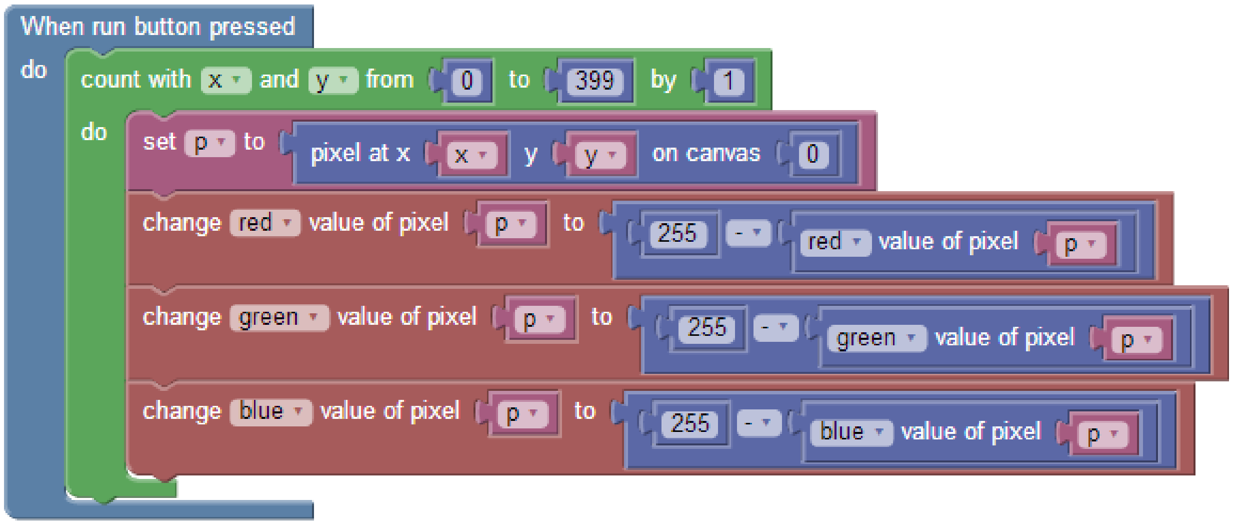
A. Chromakey Cerulean



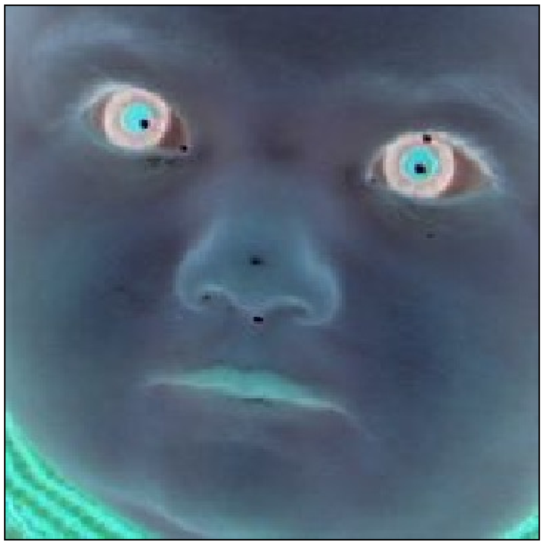
```
When run button pressed
do  count with x and y from 0 to 399 by 1
    do  set p to pixel at x x y y on canvas 0
        change red value of pixel p to 0
```



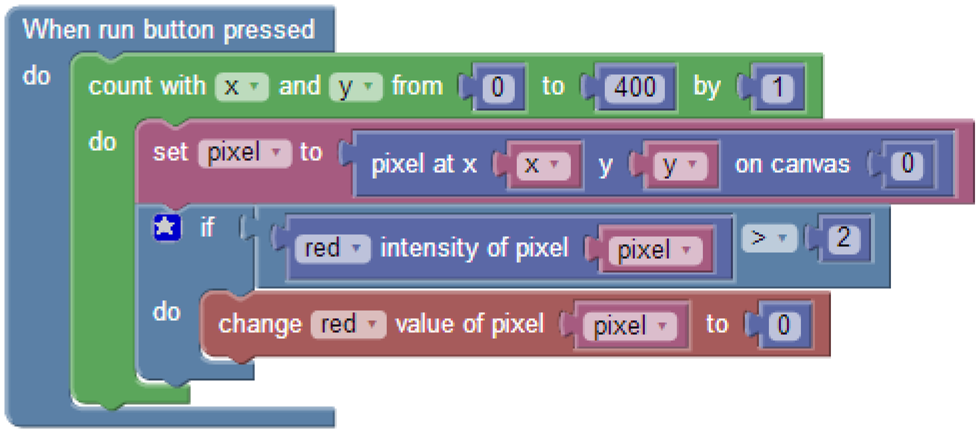
B. Negative



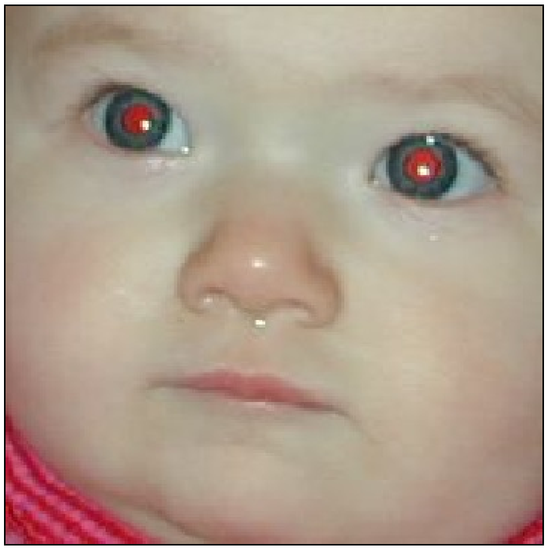
```
When run button pressed
do  count with x and y from 0 to 399 by 1
    do  set p to pixel at x x y y on canvas 0
        change red value of pixel p to 255 - red value of pixel p
        change green value of pixel p to 255 - green value of pixel p
        change blue value of pixel p to 255 - blue value of pixel p
```

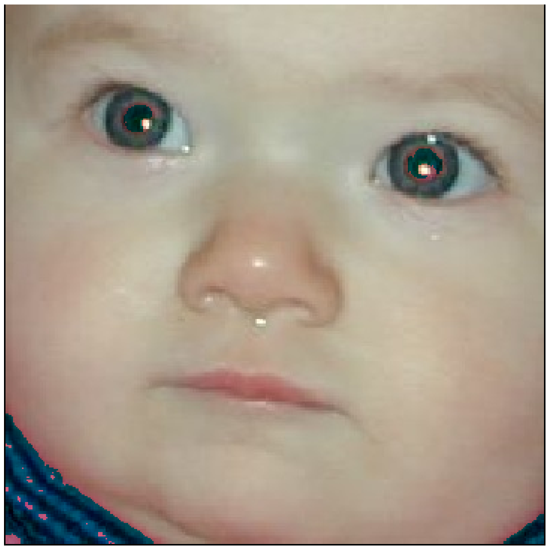


C. Red Eye Removal (Simple)

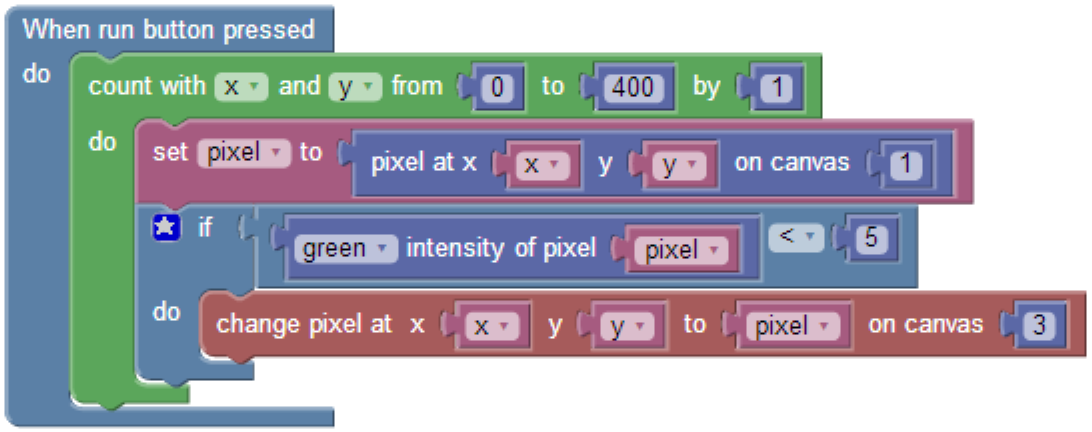


```
When run button pressed
do
    count with x and y from 0 to 400 by 1
    do
        set pixel to pixel at x x y y on canvas 0
        if red intensity of pixel pixel > 2
        do
            change red value of pixel pixel to 0
```





D. Green Screen (From Canvas 1 to Canvas 3)



```
When run button pressed
do
  count with x and y from 0 to 400 by 1
  do
    set pixel to pixel at x x y y on canvas 1
    if green intensity of pixel pixel < 5
    do
      change pixel at x x y y to pixel on canvas 3
```