# SESSION 6

## *Color Images and their Components*

AOLME Curriculum Level 1

## GOALS

1. Represent grayscale and color images using RGB.
2. Manipulate real images (using a digital camera).
3. Open image files in Python and familiarize with AOLME Python Library.
4. Link image creation with binary and hexadecimal numbers.

**Activities:**

**6.1. Creating grayscale and color images using the WebApp**

**6.2. Connecting color images with RGB Hex and decimal values**

**6.3. Processing Real images with Python**

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Each activity includes 1 card. One side of the card is in Spanish and on the back the same information is in English. Each card has 4 quadrants, each quadrant includes a task related to the main goal of the activity. The numbers in the square on the left describe the order to perform the tasks. The card must be at the center of the table. Students need to have access to it and take turns reading it. They can read it in the language they feel more comfortable.

# 6.1. CREATING GRAYSCALE AND COLOR IMAGES USING THE WEB APP

**Activity 1 Goal:**

Analyze and create grayscale and color images using the "Binary Image Generator" (binary colors, open play)

**Resources for the Activity**

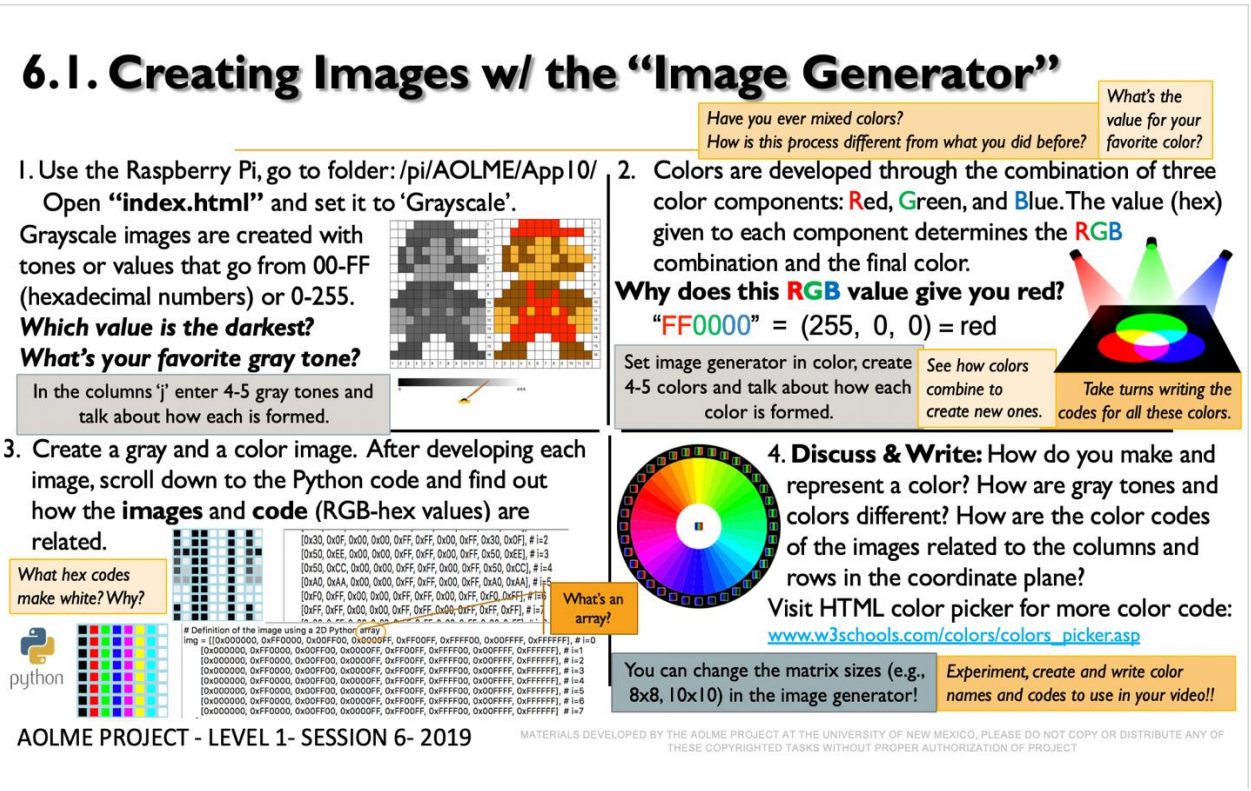
1. Activity Card 6.1
2. Folder: /home/pi/AOLME/Session 6/ "index.html"
3. The "WebApp.html" online
4. Raspberry Pi and Monitor
5. Student journal
6. HTML color picker for more color code:

   www.w3schools.com/colors/colors_picker.asp

**Interactions**

In this activity students are to think about the color components (RGB)of images. It is important that they recall what they remember about color creation and combination and use it as a way to transition into these ideas related to color and computers. For example, yellow, red, and blue are seen as primary colors. Or the basis to create other colors, here it is red, blue, and green instead. Always use their experiences as a starting point and then build on ideas. Throughout the activity provide a friendly environment, supporting the participation of everyone. Notice who participates more or less and pay attention to why it might be and act on it, so participation can be more even from everyone. Support at all times the use of the language (Spanish or English) that the students want to use.

## Activity Card 6.1:




## Recommended Steps for the Activity

1. Make sure the **"Image Generator" / Web App** online is set in grayscale or color and highlight the choice students have to set up the 'size' of the matrix
2. Have students think about what colors need to be combined to create a color; for example, white, magenta, or cyan
3. Motivate students to identify colors they like and come up with hex RGB codes for that color.
4. Let students have fun creating images.
5. Have students debrief what they learned by experimenting with the image generator and take notes in student journal.

## Content:

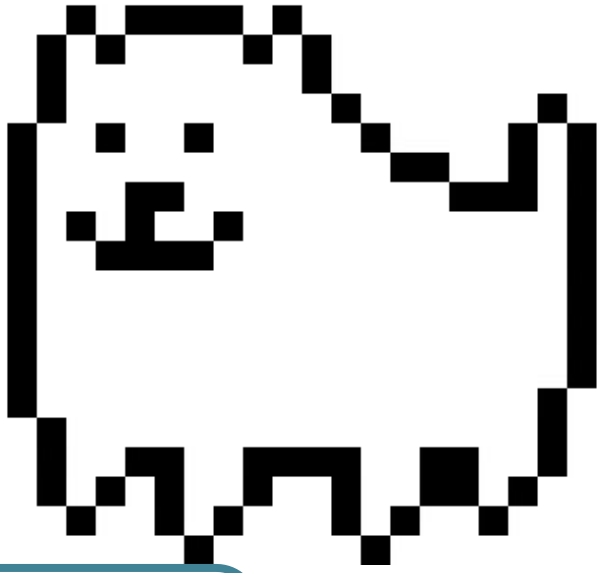
The picture you see on your computer screen is made up of tiny blocks. How can tiny blocks make so many colors?

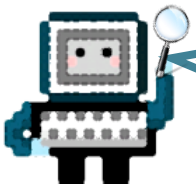
Have you ever mixed paint? Then you know you can get purple from adding red to blue. These colors can combine to make all the colors of the rainbow!

*Let's begin by looking at a black and white image.*

Undertale "Annoying Dog"
http://pixelartmaker.com/art/1409799f9511b88.png

*The image is made up of little squares. The squares are called **pixels**.*





*If we zoom way out, our eyes combine pixels into smooth lines!*

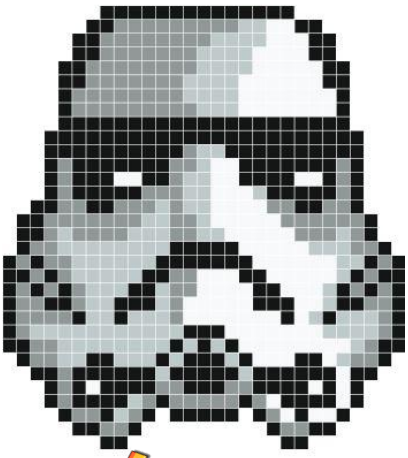
Let's see how computers do this!

## Grayscale Images

Imagine you have a light switch with a dial and you can turn that dial up to 255 and down to 0. When the dial is at 0, the light is off and it is black. When the dial is max, it is completely on and pure white.


This is what we call a **grayscale**. You can see here using what you learned about hex, the scale for grayscale can also be represented using hex. From 00 (zero) to – FF (255).

http://www.stickaz.com/fr/community/7287-casque.html



With grayscale we can make more detail to the pixels in our image by changing the **values of the pixels**.

*In computer programming, a positive integer number is called an **unsigned integer number**. Computers store an image's pixels as binary numbers.*

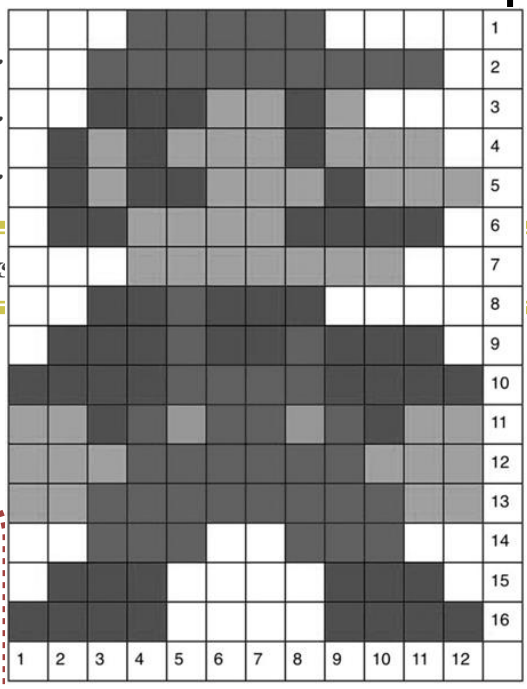
***A grayscale pixel, positive integer number from 0 to 255, is stored in the computer with how many bits?***

……

……

……

*Let's see another image and all the grays*




*When the computer reads the matrix of numbers below, we see Mario!*

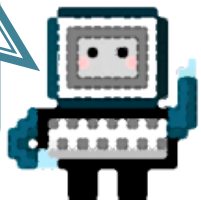
*Here, Row is y and Column is x to get (y,x)*

https://mspremiseconclusion.wordpress.com/2010/03/06/its-a-me-mario/

*The c… …any numbers in matrix.*

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 255 | 255 | 255 | 97 | 97 | 97 | 97 | 97 | 255 | 255 | 255 | 255 | 0 |
| 255 | 255 | 97 | 97 | 97 | 97 | 97 | 97 | 97 | 97 | 97 | 255 | 1 |
| 255 | 255 | 79 | 79 | 79 | 160 | 160 | 79 | 160 | 255 | 255 | 255 | 2 |
| 255 | 79 | 160 | 79 | 160 | 160 | 160 | 79 | 160 | 160 | 255 | 255 | 3 |
| 255 | 79 | 160 | 79 | 79 | 160 | 160 | 160 | 79 | 160 | 160 | 160 | 4 |
| 255 | 79 | 79 | 160 | 160 | 160 | 160 | 79 | 79 | 79 | 79 | 255 | 5 |
| 255 | 255 | 255 | 160 | 160 | 160 | 160 | 160 | 160 | 160 | 255 | 255 | 6 |
| 255 | 255 | 79 | 79 | 97 | 79 | 79 | 79 | 255 | 255 | 255 | 255 | 7 |
| 255 | 79 | 79 | 79 | 97 | 79 | 79 | 97 | 79 | 79 | 79 | 255 | 8 |
| 79 | 79 | 79 | 79 | 97 | 97 | 97 | 97 | 79 | 79 | 79 | 79 | 9 |
| 160 | 160 | 79 | 97 | 151 | 97 | 97 | 151 | 97 | 79 | 160 | 160 | 10 |
| 160 | 160 | 160 | 97 | 97 | 97 | 97 | 97 | 97 | 160 | 160 | 160 | 11 |
| 160 | 160 | 97 | 97 | 97 | 97 | 97 | 97 | 97 | 97 | 160 | 160 | 12 |
| 255 | 255 | 97 | 97 | 97 | 255 | 255 | 97 | 97 | 97 | 255 | 255 | 13 |
| 255 | 79 | 79 | 79 | 255 | 255 | 255 | 255 | 79 | 79 | 79 | 255 | 14 |
| 79 | 79 | 79 | 79 | 255 | 255 | 255 | 255 | 79 | 79 | 79 | 79 | 15 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |

*Computers convert a matrix of pixels to display images. Each pixel is defined using a hexadecimal number.*



# 6.2. CONNECTING COLOR IMAGES WITH RGB HEX AND DECIMAL VALUES

**Activity 2 Goals:**

Explore how Hex and decimal values are linked to pixel RGB values in color /grayscale images

**Resources for the Activity**

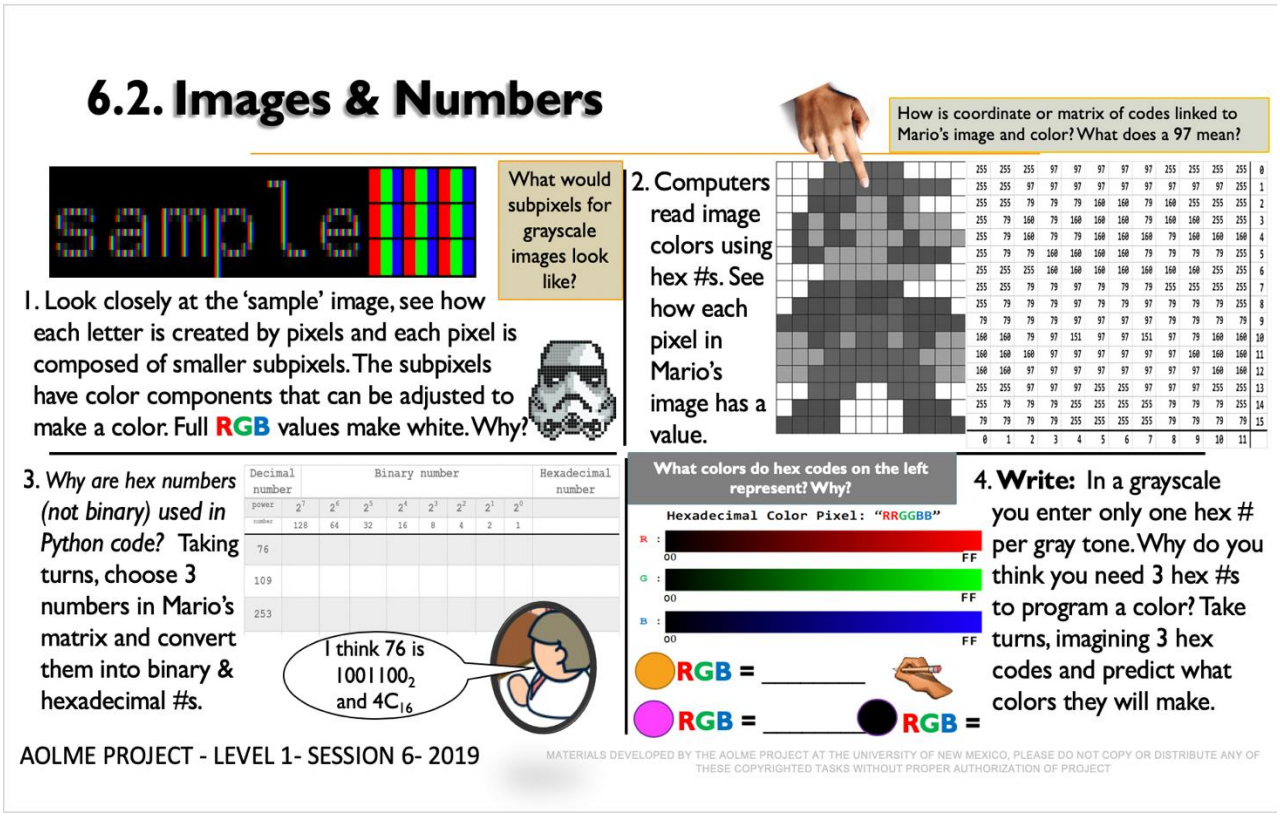
7. Activity Card 6.2
8. Folder: /home/pi/AOLME/Session 6/
9. The "WebApp.html" online
10. Raspberry Pi and Monitor
11. Student journal
12. HTML color picker for more color code:

    www.w3schools.com/colors/colors_picker.asp

**Interactions:**

Throughout the activity provide a friendly environment, supporting the participation of everyone. Notice who participates more or less and pay attention to why it might be and act on it, so participation can be more even from everyone. Support at all times the use of the language (Spanish or English) that the students want to use. Support recalling of hex and binary with the purpose of generating images. Highlight that the idea is not to convert to hex accurately but to understand base 16 and it differs from base 2 and 10 systems, but also that hex relate to the creation and naming of colors through the combination the basic digital color components.

**Activity Card 6.2:**




## Recommended Steps for the Activity

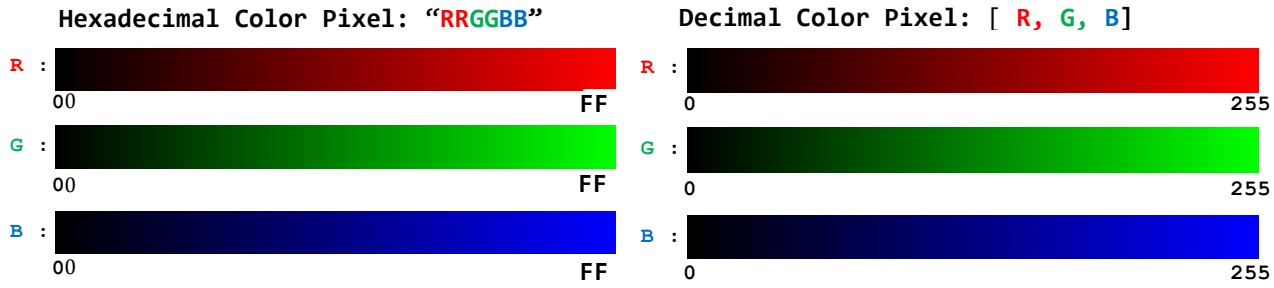
1. Identify that RGB works at the pixel level and it can be changed.
2. Highlight how a grayscale pic is only one frame, but color pics include three frames, RGB, in different tones each frame, like the Mario illustration.
3. Practice binary and hex numbers connected to colors. If needed, use materials from Session 5. Numbers could be corroborated through Python.
4. Have students debrief what they learned by experimenting and take notes of that on journal.
5. Activity 6.1 was to think about the components or colors to be combined to create another color, but this activity is to think about the RGB numerical values associated to that color combination to create new colors.

Content:

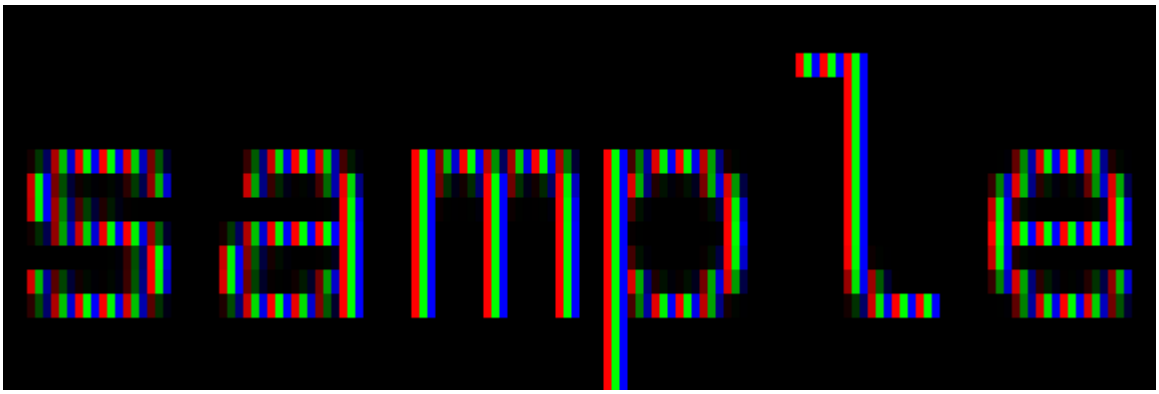
*Convert the following decimal grayscale values into binary and then into hexadecimal. How many digits do you need for the hexadecimal number?*

| Decimal number | Binary number | | | | | | | | Hexadecimal number |
|---|---|---|---|---|---|---|---|---|---|
| power | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
| number | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | |
| 76 | | | | | | | | | |
| 109 | | | | | | | | | |
| 253 | | | | | | | | | |
| 17 | | | | | | | | | |
| 94 | | | | | | | | | |

Remember the light switch example? Grayscale colors happen when we turn on the three colors Red, Blue, and Green and set them as the same value. If they are all 255 or FF we get white, if they are all 0 or 00 we get black!

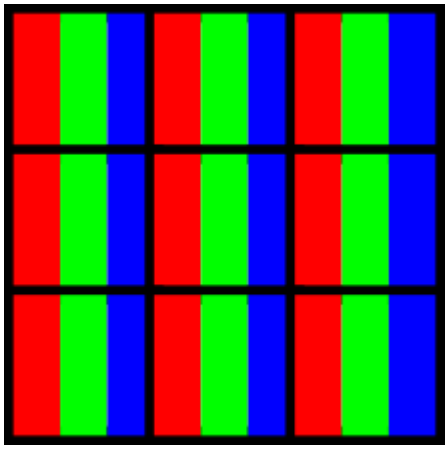


But why do you not see blocks of color on your monitor? It uses **subpixels**. Every pixel you see in the computer has a red, blue and green band of color.



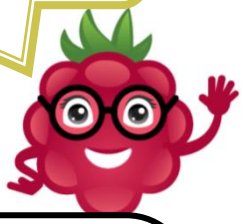
https://en.wikipedia.org/wiki/Subpixel_rendering#/media/File:Subpixel-rendering-RGB.png

Look closely, can you see the three bands of color? The zoomed out text  looks gray!



In this image, inside the black squares are pixels, and the stripes of R, G, and B are subpixels!

*Your monitor is made of little tiny subpixels just like these!*



*Grayscale pixels have values between 0 and 255 and are stored using 8 bits. Color pixels are 3 values, each between 0 and 255. How many bits do you think a computer needs to store one value?*

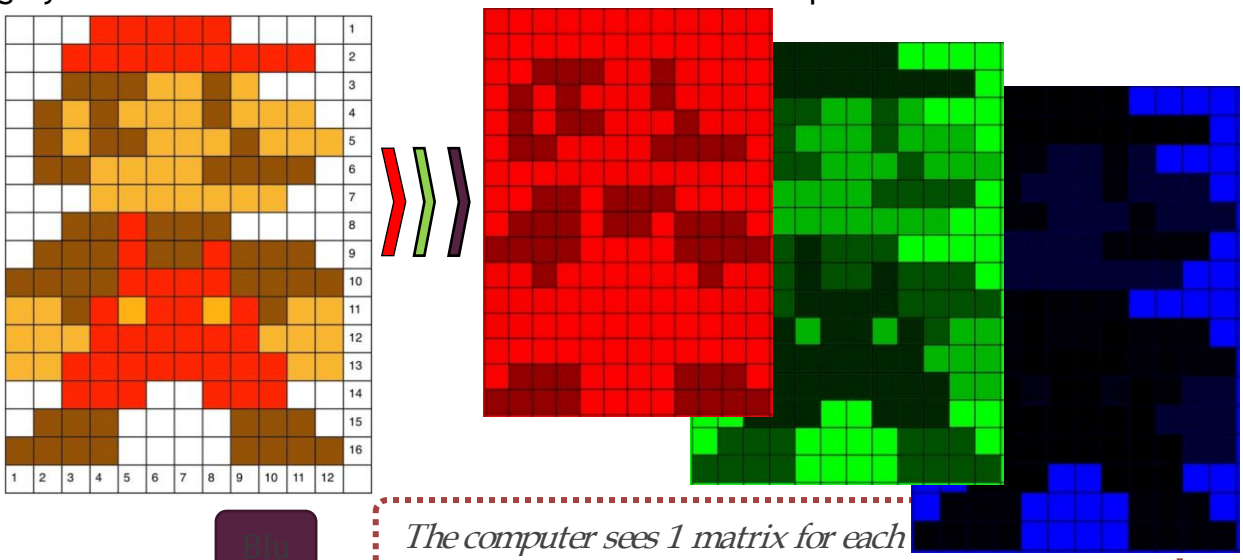
……

……

……

*Color images are called RGB images because the pixels are made from combinations of the three colors.*

Like grayscale, each color component has its own matrix. Remember the grayscale Mario? Now we will look at his three color components.



*The computer sees 1 matrix for each*

Blu

Gree

Re

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 0 |
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 1 |
| 255 | 255 | 148 | 148 | 148 | 248 | 248 | 148 | 248 | 255 | 255 | 255 | 2 |
| 255 | 148 | 248 | 148 | 248 | 248 | 248 | 148 | 248 | 248 | 255 | 255 | 3 |
| 255 | 148 | 248 | 148 | 148 | 248 | 248 | 248 | 148 | 248 | 248 | 248 | 4 |
| 255 | 148 | 148 | 248 | 248 | 248 | 248 | 148 | 148 | 148 | 148 | 255 | 5 |
| 255 | 255 | 255 | 248 | 248 | 248 | 248 | 248 | 248 | 248 | 255 | 255 | 6 |
| 255 | 255 | 148 | 148 | 255 | 148 | 148 | 148 | 255 | 255 | 255 | 255 | 7 |
| 255 | 148 | 148 | 148 | 255 | 148 | 148 | 255 | 148 | 148 | 148 | 255 | 8 |
| 148 | 148 | 148 | 148 | 255 | 255 | 255 | 255 | 148 | 148 | 148 | 148 | 9 |
| 248 | 248 | 148 | 255 | 255 | 255 | 255 | 255 | 255 | 148 | 248 | 248 | 10 |
| 248 | 248 | 248 | 255 | 255 | 255 | 255 | 255 | 255 | 248 | 248 | 248 | 11 |
| 248 | 248 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 248 | 248 | 12 |
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 13 |
| 255 | 148 | 148 | 148 | 255 | 255 | 255 | 255 | 148 | 148 | 148 | 255 | 14 |
| 148 | 148 | 148 | 148 | 255 | 255 | 255 | 255 | 148 | 148 | 148 | 148 | 15 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |

*Can you find the [R,G,B] value of a pixel that colors Mario's hat and suspenders? (Hint: Look in the first row!)*

*What is the (y,x) location of a hat pixel? What is the [R,G,B] value of the pixel? What is the hex value (RRGGBB)?*

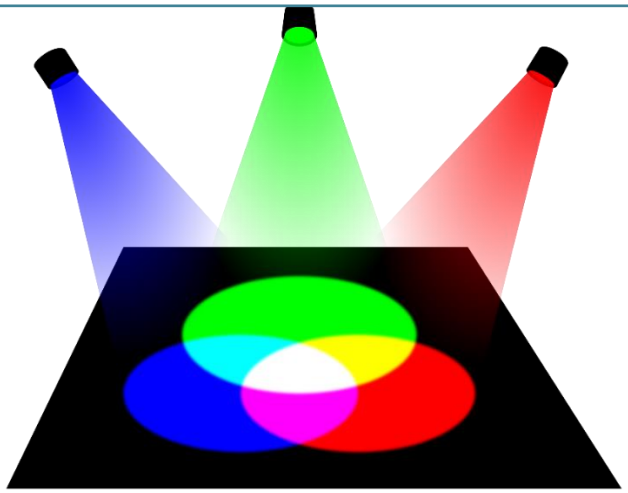
……

**Teamwork**

*Let's take a look at the pixels on your monitor! Get a camera and take a very close picture of the screen. Can you see the tiny squares (pixels) in your picture?*

Let's look at some colored beams of light.

https://openclipart.org/detail/246218/additive-color-



**Discuss with your classmates**

**Let's look at the primary colors:**

| Red | Green | Blue |
|---|---|---|
| "FF0000" = (255, 0, 0) | "00FF00" = (0, 255, 0) | "0000FF" = (0, 0, 255) |
| [ R , G , B ] | [ R , G , B ] | [ R , G , B ] |

When we mix Red and Blue circles together we get a new color. This is the same as combining Red (255,0,0) and Blue (0,0,255) to get (255, 0, 255) as the result, which is the same as "FF00FF" in hex! This color is called "Magenta".

*Now come up with the other two colors where red and green mix and where blue and green mix.*

| | **Red + Green** | **Green + Blue** |
|---|---|---|
| RGB | | |
| Hex | | |
| Name | | |

# 6.3. PROCESSING REAL IMAGES WITH PYTHON

**Activity 3 Goal:**

Process real-life images through the AOLME library by defining image Coordinates plane (coordinates, grouping, blocks, share and modify-'debug'), arrays, and ranges with Python (debug, program, share and modify-DMI)

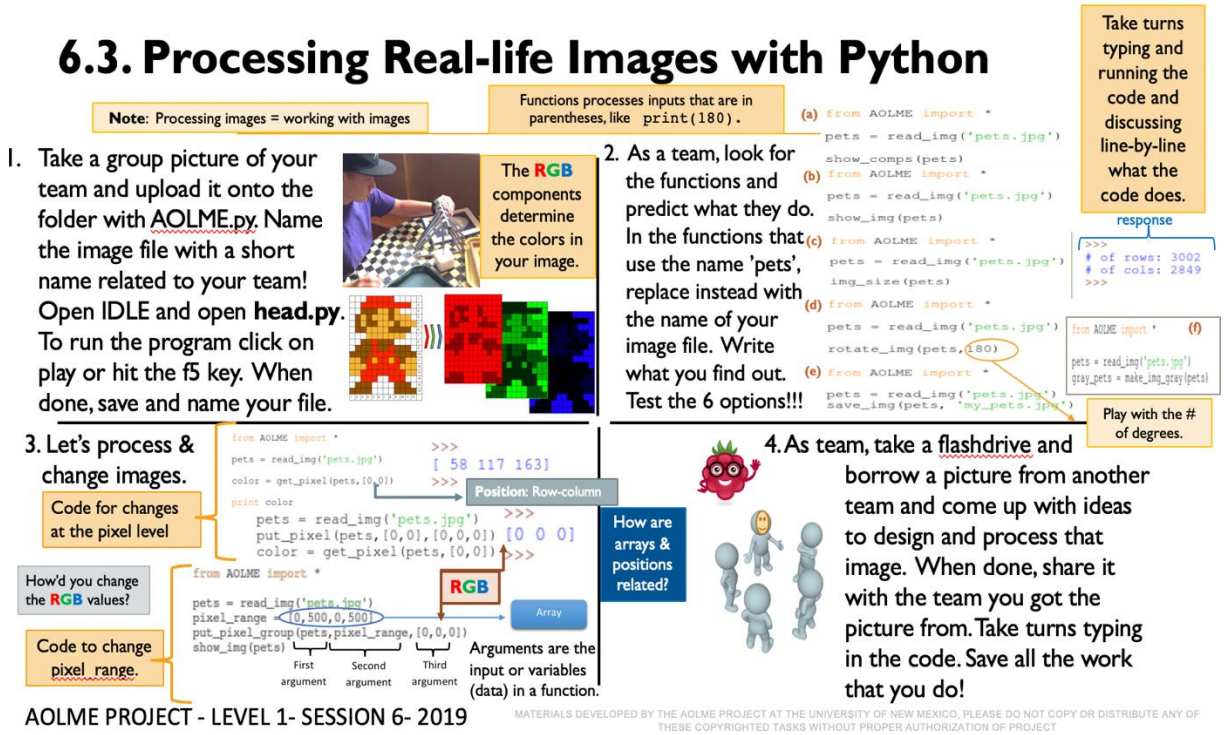
**Resources for the Activity**

1. Activity Card
2. Digital Camera per group
3. USB drive per group
4. AOLME.py folder (head.py file)
5. Raspberry Pi and Monitor
6. Student journal

**Interactions**:

Provide a problem-solving environment around the use of images so that the codes become more fun. Let students make decisions about what images to take so that they feel excited and motivated about making changes to the images. Perhaps they want to be in the image? Let them take the lead. If needed leave the room and go outside to take pics of building, nature or themselves.

**Activity Card 6.3:**




## Recommended Steps for the Activity

1. Let the team take initiative to decide where to take the picture.
2. Make sure name of the image is remembered by the group and saved onto both AOLME.py folder and onto the USB to share with another group.
3. In point 2, let students focus on the code syntax and meaning by experimenting and inferring what happens with each code.
4. Have them take notes on journal on what they discover.
5. In 2d emphasize the use of degrees, have them recall what they know and make connections.
6. In point 3, make sure that the teams change picture and are able to name and use arrays, ranges, and arguments. And that RGB components are changed.
7. Be creative and funny in point 4.

**Content:**

# Project: Our Team Picture!

**Teamwork**

*For this project you need to take a picture with your team. Go find a bright location around the school with lots of colors and take your picture. Do a silly pose or have some fun!*

## Using the AOLME Image Library

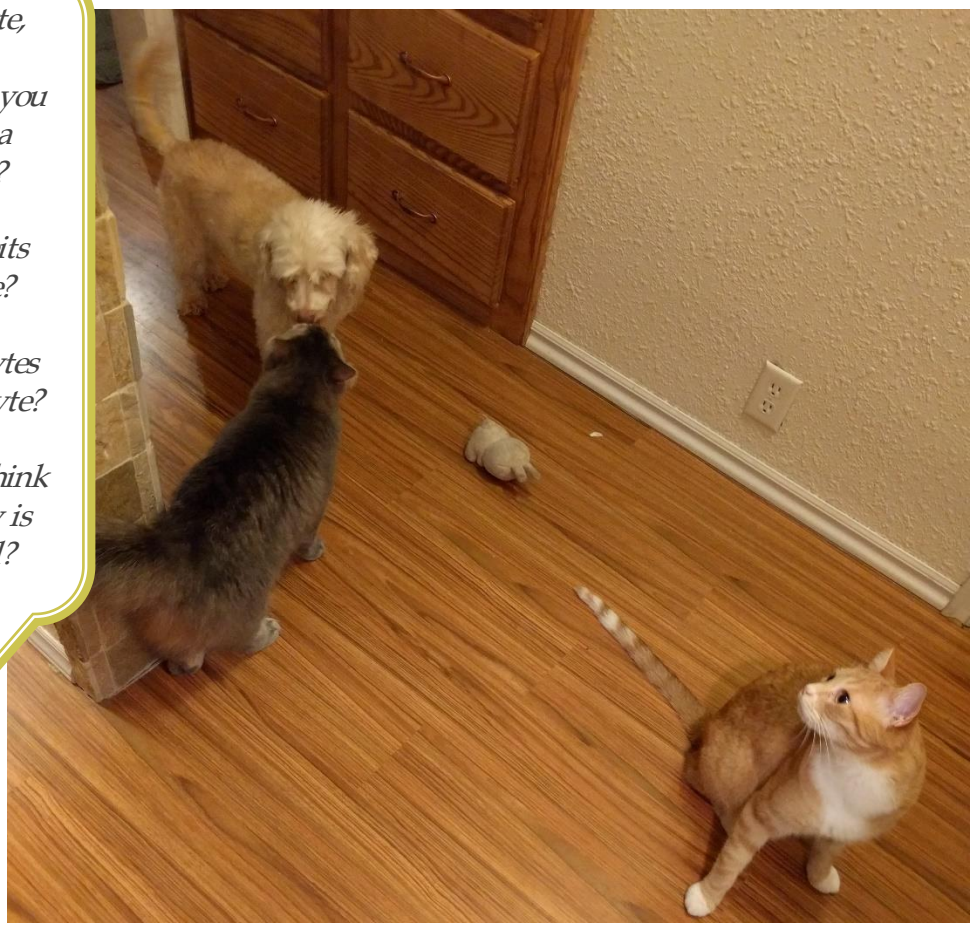
For this part we will use this example image, but you will use your team picture.

*Meet Terabyte, Pixel and Sopapilla! Do you know what a Terabyte is?*

*How many bits are in a byte?*

*How many bytes are in a Terabyte?*

*Why do you think the gray kitty is named Pixel?*



**Computer Time!**
Playing with colors.

1. Put your group picture in the same folder with AOLME.py and open a python session. Make sure you rename it to something easy!

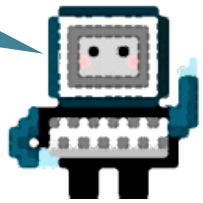
2. First we need to read the image so the computer can read and show the matrices.
To do so, open IDLE and make a new file.

3. We will type the `read_img()` function to do that, and give it the name of the picture you took. We need to set a variable, this is any name of the file you want to call (that is `'pets.jpg'`).

```
from AOLME import *

pets = read_img('pets.jpg')
```

*Let's look at the RGB components in your group picture!*



4. The next function in the file will be `show_comps()`. For this function to work we need to pass our pets image to it.

```
from AOLME import *

pets = read_img('pets.jpg')

show_comps(pets)
```

*When we put a variable between the parentheses in a function we call this '**passing** a variable'. Anything passed to a function is called a function's **arguments**.*

5 Now we are ready to run the program, save the file and name it, then go to the **Run** menu and pick 'Run Module'. You can also just hit the f5 key!

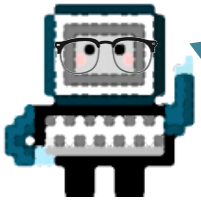
Here the results, what did you get?


**Discuss with your classmates.**

- *Do you see black areas in any of the component images? Why do you think this happened? Which one is the darkest component? Did you see something similar in group images?*


*Let's use more of the functions in the library on your image!*

There are lots of things you can do with your image. The AOLME library has easy functions you can use to change or view the picture.



***You know how to read the picture into a variable, so let's use this and try some challenges!***

How do you show the image that's stored in a variable?

```
from AOLME import *

pets = read_img('pets.jpg')

show_img(pets)
```

We use the function `show_img()` and pass our variable to it to view it.

**Teamwork**

*Save your team picture with the filename "secret_image_yourteamname.jpg" and put it on a thumbdrive. Go to another group and trade images with them. Take their secret image from the thumbdrive and use the AOLME library to see their picture!*

Now that you can read and view the images, let's do something fun.

```
from AOLME import *

pets = read_img('pets.jpg')

img_size(pets)
```

*The function img_size() gives the number of rows and columns in the image matrix.*

```
>>>
# of rows: 3002
# of cols: 2849
>>>
```

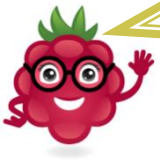
*Here's how many rows and columns are in the pets picture!*

Is your image a little crooked? There's a function for that too, it's rotate_img()!

```
from AOLME import *

pets = read_img('pets.jpg')

rotate_img(pets,180)
```



*If 180 degrees flipped the image, how many degrees do you need to rotate a sideways image?*

*Wait!* `rotate_img(pets,180)`

*The function rotate_img() has two* ***arguments****. The first one refers to the variable* `'pets'` *or the picture and second argument is an* ***integer number*** *which says how many degrees we want the function to rotate the image by.*

Say you really like a color and quickly want to know what the RGB value is of the color. You can get the value by using the get_pixel() function.

```
from AOLME import *

pets = read_img('pets.jpg')

color = get_pixel(pets,[0,0])

print color
```

```
>>>
[ 58 117 163]
>>>
```

*Pass the [y,x] location of the pixel to the function and print the*

*The function will print the [R,G,B] value of the pixel! Convert this to*

But if we don't like that color we can instead change it to something we do want using `put_pixel()`!

*Pass the [y,x] location of the pixel to the function and print the variable.*

*This time we check to make sure the pixel color changed, it did!*

```
from AOLME import *

pets = read_img('pets.jpg')
put_pixel(pets,[0,0],[0,0,0])
color = get_pixel(pets,[0,0])

print color
```

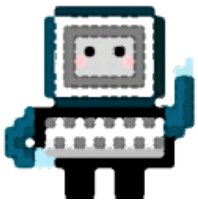
```
>>>
[0 0 0]
>>>
```

*Notice now we need three **arguments** to change the pixel. The first argument is the image, then the location we want to put the pixel, and last we need to give values [R,G,B] for a color.*

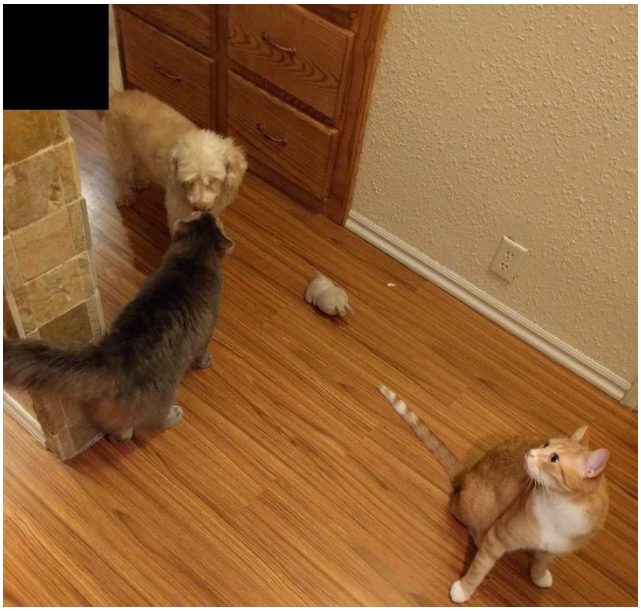
*Something's different!! We did not show the image here! Can you guess why we have to check the pixel with get_pixel()?*

Say you want to add an entire block of pixels to the image to cover up a part you don't like. There is a function for this called `put_pixel_group()`. Now we need to specify which range of pixels we want to put the block in instead of a single pixel like `put_pixel()`. We are going to use a second **variable** called "pixel_range" to do this.

***When we want to make a new variable we name it first then make it equal to something new. To set the range, we want to pick the first row pixel, the last row pixel, the first column pixel then the last column pixel and put them in a list called an* array.**



*Set up the pixel range and pass it to the function* `put_pixel_group()` *don't forget to add the color in [R,G,B] values.*



```
from AOLME import *

pets = read_img('pets.jpg')
pixel_range = [0,500,0,500]
put_pixel_group(pets,pixel_range,[0,0,0])
show_img(pets)
```

What happened when you used the function `put_pixel_group()`? The top 500x500 pixel square filled with black pixels!

*Remember, the second* ***argument*** *in put_pixel_group() is a range. It can be passed directly or through a* ***variable****. The range must be in the format of [y1,y2,x1,x2]. This format is called an* ***array****.*

```
from AOLME import *

pets = read_img('pets.jpg')
pixel_range = [0,500,0,500]
put_pixel_group(pets,pixel_range,[0,0,0])
show_img(pets)
```

Array

First argumen

Second argumen

Third argumen

To save the new image after making changes, store the changed image using the `save_img()` function. When the image changes, it must be saved into a variable first or the changes will be lost. This is done by setting the new variable equal to the function. A copy of the image can be saved with a different name using this function.

*Tell the function to save the variable pets as 'my_pets.jpg'. Don't forget the file type (.jpg)!*

```
from AOLME import *

pets = read_img('pets.jpg')
save_img(pets, 'my_pets.jpg')
```

*Check your folder! Now there is a file called pets.jpg and a file called my_pets.jpg!*

Now let's look at some advanced functions. First, recall the lights and how mixing only two of the three colors had a cool effect on the image. We can see this effect using the function `get_comps()`. This means "get all components". This function has a ***return value*** which means to see the results we need to save the output into a variable. The return value type is special because it is an **array** like we've seen before. Let's try it!

*Each variable inside of the return value is a different image, then we save every component as the name 'color.jpg'.*

```
from AOLME import *

pets = read_img('pets.jpg')
[r,g,b,y,c,m] = get_comps(pets)

save_img(r, 'red.jpg')
save_img(g, 'green.jpg')
save_img(b, 'blue.jpg')
save_img(y, 'yellow.jpg')
save_img(c, 'cyan.jpg')
save_img(m, 'magenta.jpg')
```

*Do you remember how to get cyan, yellow, and magenta channels using R,G,B?*

*How would you show the components instead of saving them?*

If you want to cut or crop the image or it has something extra that you don't want to see, you can use the `crop_img()` function.

*Give crop_img() return value a new variable to keep the changes. The ranges are defined the same*

```
from AOLME import *

pets = read_img('pets.jpg')
sopapilla = crop_img(pets, [1757,2847,1757,3001])

save_img(sopapilla,'sopapilla_only.jpg')
```

Now we have a good picture of Sopapilla to use with functions or share with friends!

*The range must still be in the array format of [y1,y2,x1,x2]. The best practice is to save this as a variable. Warning: Don't name your variable* ***range****, python has reserved that name for another purpose, name it ranges or pixel_range or something else.*


*Here are some optional functions to explore.*

Do you remember when you learned the difference between a grayscale image and an R,G,B image? Let's make the image gray and look at some of the above functions.

*Give the function a new variable and use the new variable.*

```
from AOLME import *

pets = read_img('pets.jpg')
gray_pets = make_img_gray(pets)
```

*Let's try some exercises! Repeat the functions from before with your grayscale image.*

**Discuss with your classmates**

- *Which functions had a different output?*
- *Why is the output different?*
- *Why do you think* `put_pixel()` *didn't work correctly?*