Topic	HOW DO COMPUTERS SEE THE WORLD?	
Class Description	**The student will be introduced to the concept of Computer Vision. The student will learn how to read, process, and save images using OpenCV.**	
Class	**PRO C104**	
Class time	**50 mins**	
Goal	• Introducing Computer Vision. • Read, view and save images using OpenCV. • Modify images using OpenCV and NumPy arrays.	

Resources Required	• Teacher Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Visual Studio Code
	• Student Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Visual Studio Code	

Class structure	**Warm-Up** **Teacher-Led Activity 1** **Student-Led Activity 1** **Wrap-Up**	**10 mins** **20 mins** **15 mins** **05 mins**
Credit & Permissions:	OpenCV was started at Intel in 1999 by **Gary Bradsky**. Images used in the activities are created in-house or taken from **Pixabay.**	

WARM-UP SESSION - 05 mins	
Teacher Starts Slideshow  **Slide 1 to 4** Refer to speaker notes and follow the instructions on each slide.	
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide 5 to 23**	
Following are the session deliverables: • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
Teacher Ends Slideshow 	
TEACHER-LED ACTIVITY - 20 mins	
Teacher Initiates Screen Share	
ACTIVITY	

- **Introduction to Computer Vision.**
- **Explain different types of images.**
- **Image Manipulation using NumPy arrays.**

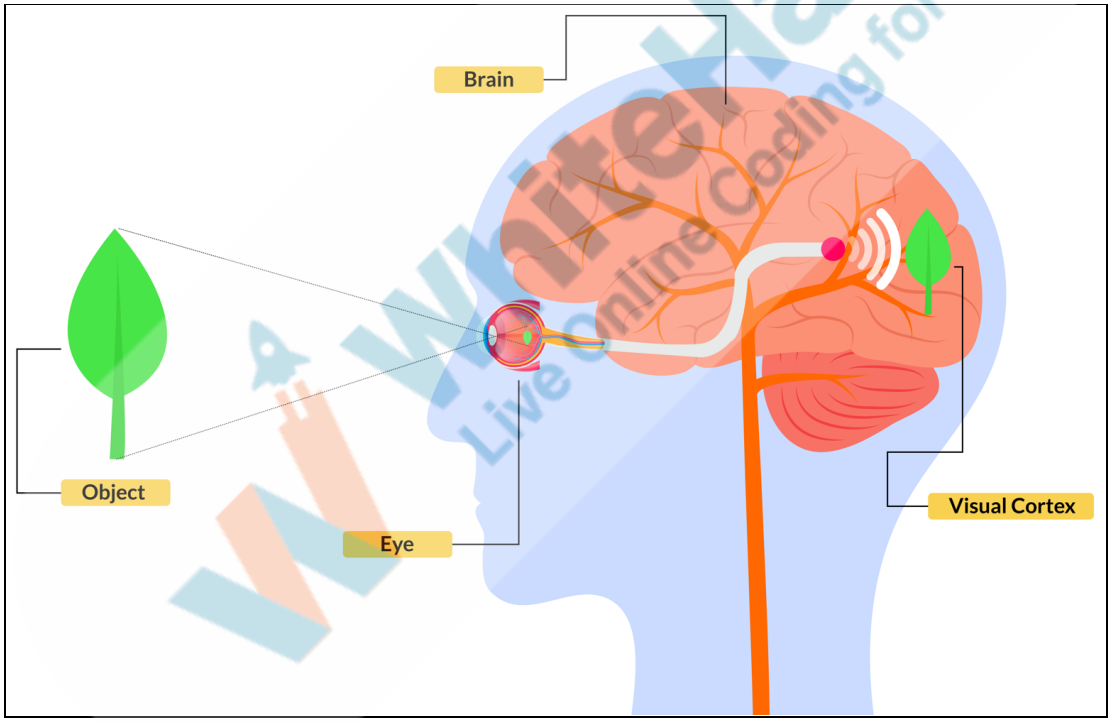
Teacher Action	Student Action
Well done! You have answered the questions well. Over the next few classes, we are going to explore how computers see the world. Before we do that, let's understand how humans see anything in the world. How do humans see things around them? Yes, correct! We can see things using our eyes. **Human Vision:** Humans can see beautiful things with their eyes. To understand more about how we can see things around us let's first understand how human vision works. Human eye is a very complex body part but to understand how the human vision works we can roughly classify the process into three main components: ***Note**: We will not deep dive into the biological technicality of the working of human eyes and the human brain.* A **Light** source which gets reflected from an **object** and reaches our eyes.	**ESR:** We see things with our eyes.

A **Receptor** in the **eyes** receives this light through the eye lenses and sends signals to the brain.

A **Visual Cortex** that is part of the **brain** processes the information sent through the eyes to make us see things. This information is processed using thousands of the brain cells.

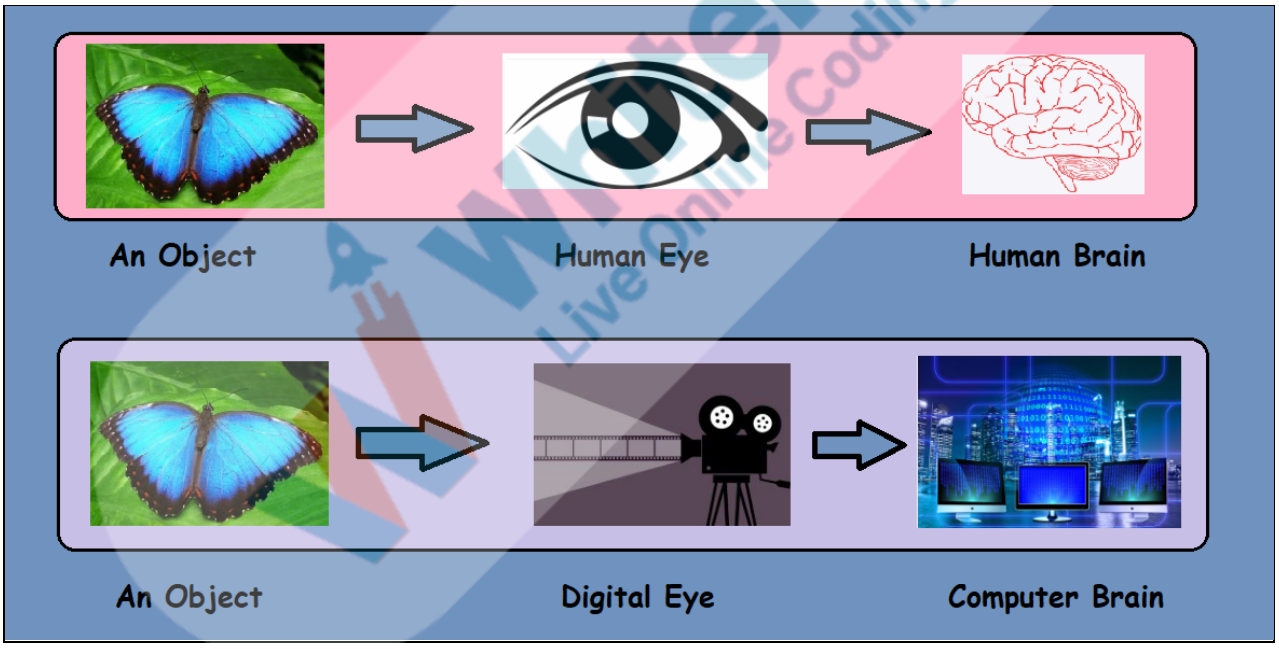
With this we can understand the human eyes and brain work together to help us see things around us.

***Note**: The below image is only for illustration purposes. This illustration can be performed using Visual Aids Slides.*



Now that we know we need **human eyes to capture the information,** and the **brain to process the information**, can you tell me how we can mimic the same functionality with computers(machines/systems)?

ESR: Varied.

***Note**: Encourage the student to give answers and be more involved in the discussion.* Well we need something to mimic the human eye that can capture information, can you tell me what can we use for that? Great! And to process the captured information like our brain, we will need computers. So we can say that a camera and a computer can work together to see things.	**ESR:** A camera.
	
A **Camera** is a device that is used to only capture the information in the form of images or videos.	

Computer Vision helps the computer to identify or understand the content of the digitally captured information.

The study of computer vision began around the 1960s to mimic the human visual system. Since then the field has expanded immensely in various areas of human life.

Let's understand through a few of the examples of computer vision tasks:

- **Image & Video Pre-Processing**: High and complex images can be processed to train machines better in identifying objects.

  Example: Medical image scanning uses high level image processing techniques to get accurate results.

- **Recognition**: This task of computer vision that helps to identify if an image contains specific objects or features of an object.

  Example: Face recognition to unlock smartphones.

- **Scene Segmentation**: This task splits the scene into multiple segments.

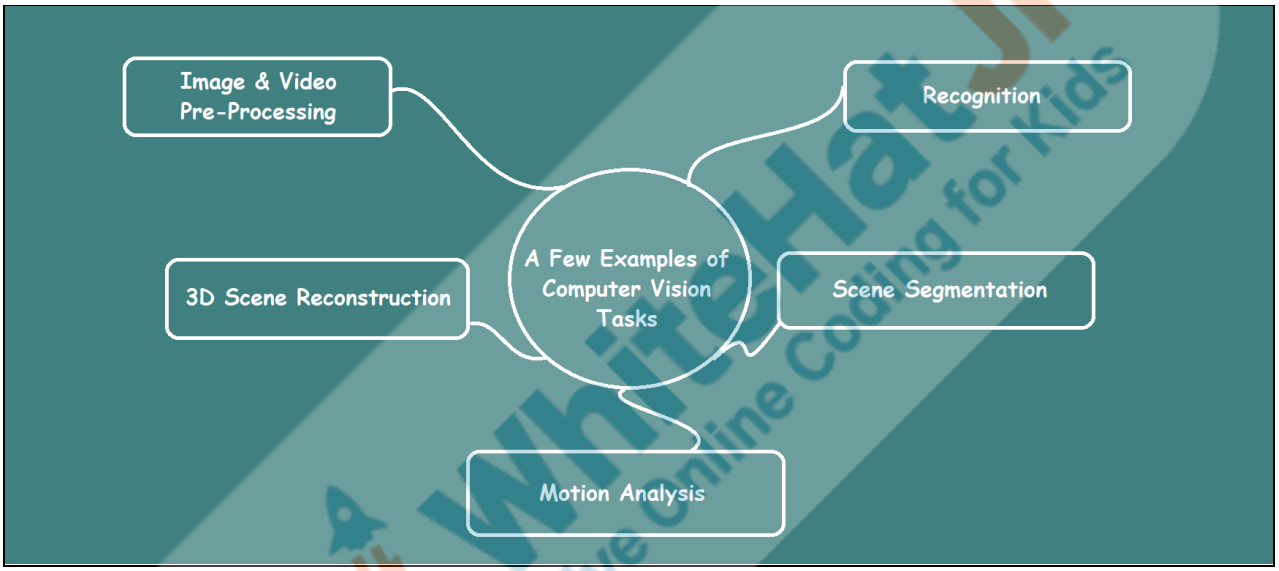
  Example: Road scene segmentation can be used in self-driving cars.

- **3D Scene Reconstruction:** Constructing realistic 3D images by taking multiple 2D images.

Example: Modern interior designers use this technique.

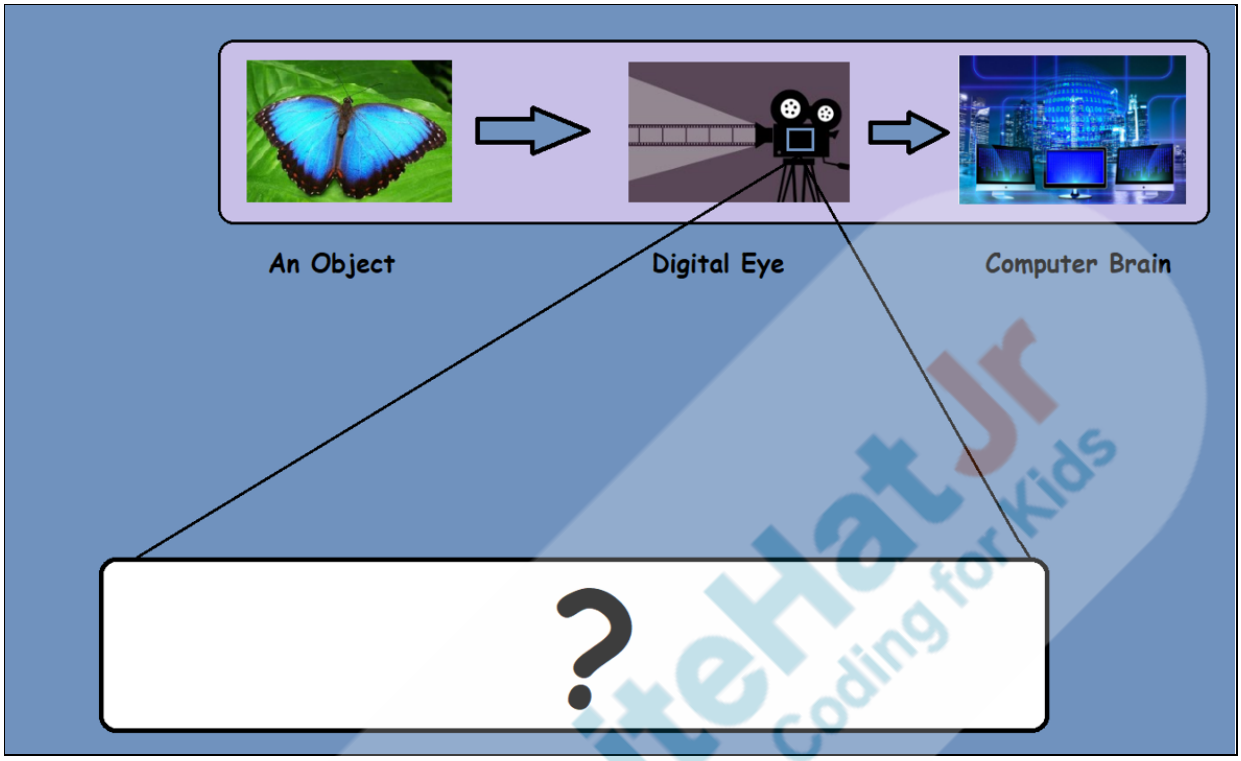
- **Motion Analysis**:This task helps to process moving objects.

  Example: Road Traffic can be monitored by tracking moving vehicles.



We will learn to work around a few of the computer vision tasks, but before we can begin applying these computer vision techniques, we need to understand how information is captured digitally.

In today's class we will learn how a computer views a digitally captured image.

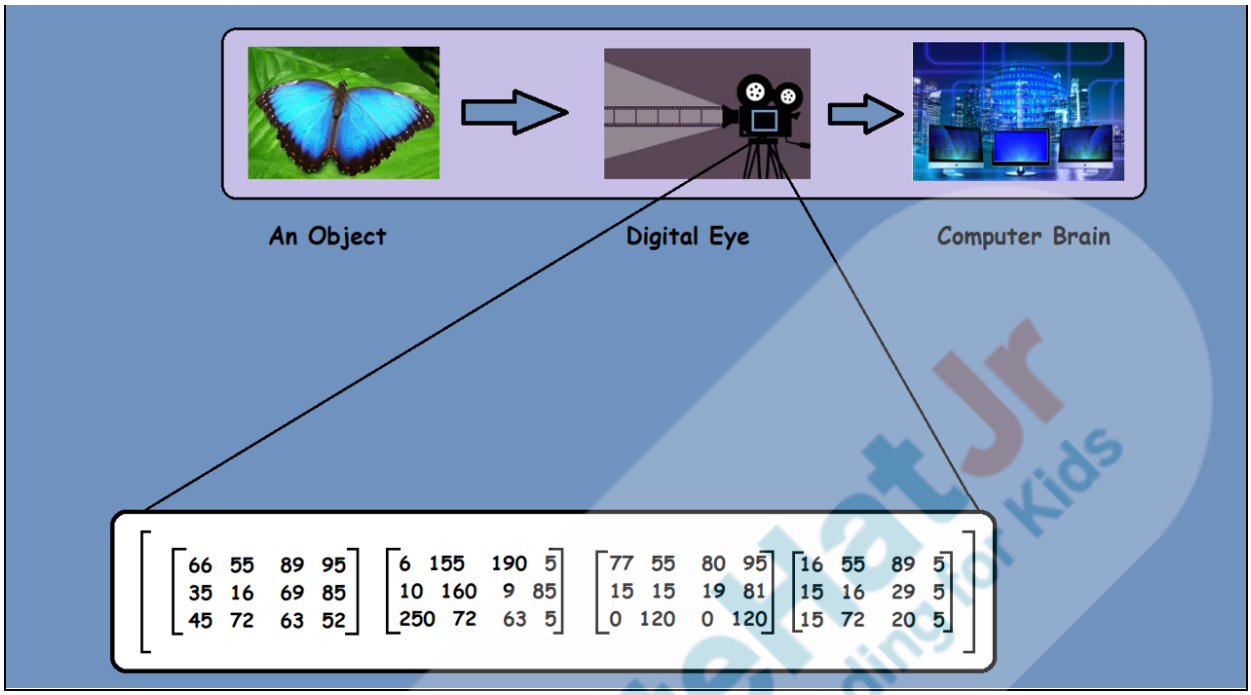


The image is stored as a set of numbers called **pixels**. **Pixels** are arranged in rows and columns to form a 2D array which can also be called a **matrix**.

***Note**: The numbers in the image below are only for representation purposes.*

As we proceed with the class we will learn to interact with the pixel values and understand how:

- *Pixels values are stored.*
- *To access a set of pixel values.*
- *To alter a set of pixel values.*



Social Media is very popular these days. Everyday millions of pictures are captured. Most of the time those pictures are first edited before those get uploaded, right?

ESR: Yes.

Have you ever tried cropping an image, changing the size of a photograph to frame it, or using filters on social media apps?

ESR: Varied.

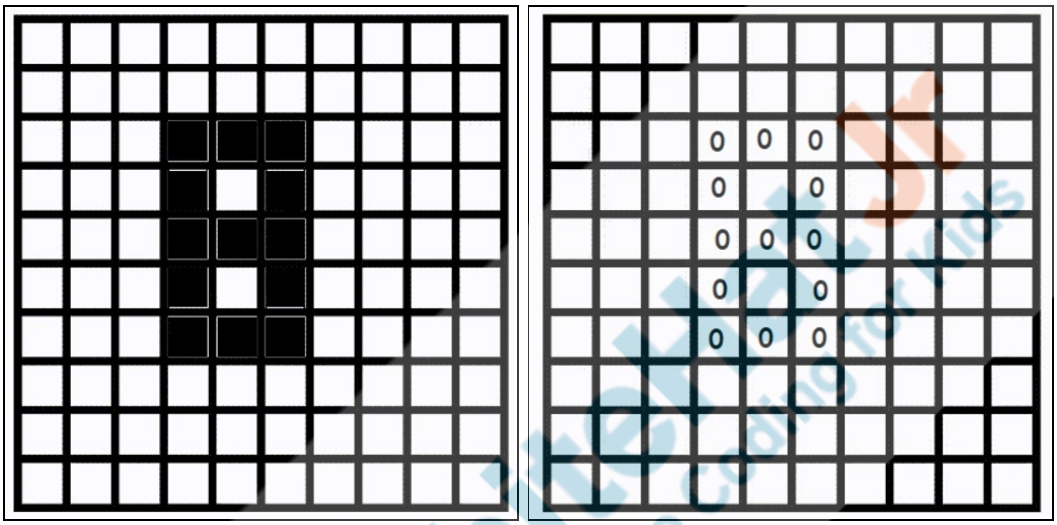
When these digitally created images are either altered or used for comparison, it is called **Image Processing**.

There are many software applications available for image processing. Image processing is very helpful in fields like biometrics, traffic movement, face/object detection, and many more.

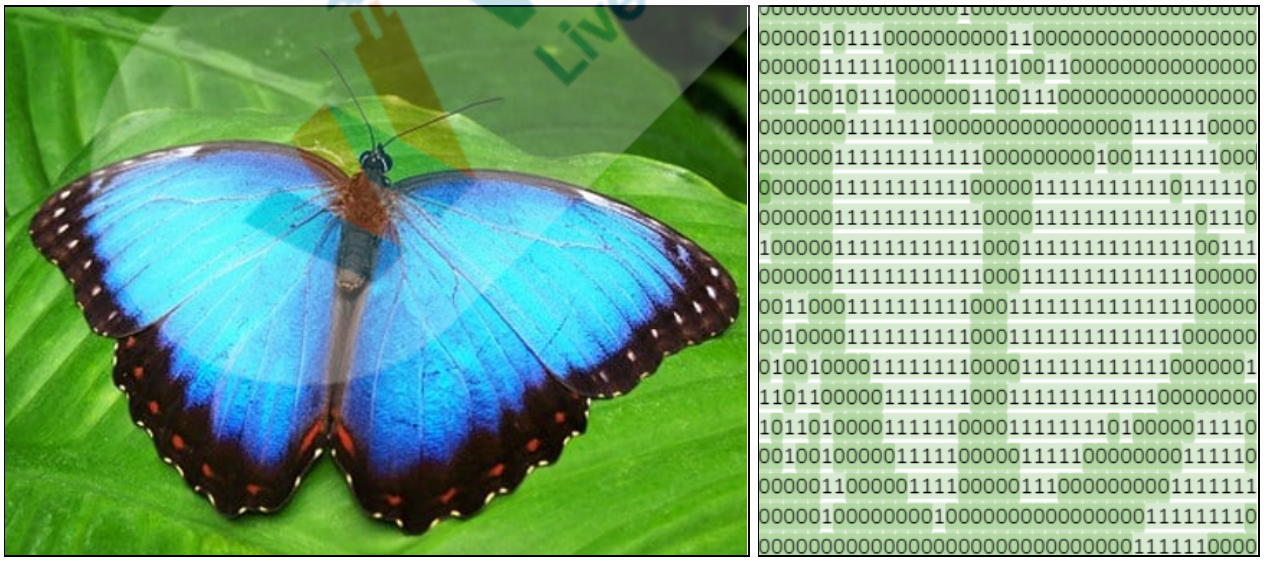
In today's class, we will learn how to access these 2D arrays of images and alter them to perform image processing using Python. To do so we will have to first understand the types of digital images. **Types of Digital Images**: There are three types of digital images: • Binary Images • Grayscale Images • Colored Images **Binary Images:** Binary images are black and white images. Each pixel value is represented by a single value, out of 2 values, either **0(black)** or **1(white)**. In binary images there is no gray level. Only two colors that are black and white are present in it. *The teacher opens Teacher Activity 1.* ***Note**: This activity is only for illustration purposes. This activity can be performed using Visual Aids Slides by clicking on the question mark to find out 0s.* Let's consider a 10x10 grid(10 rows and 10 columns). Each box represents a **pixel** value. For a binary image we can have only 0 or 1 as pixel value.	**ESR:** Varied

Now if we want to show this digital number 8, where should we fill 0s?

***Note**: Encourage the student to give answers and be more involved in the discussion.*



***Note**: A colored image in binary format with just 0s and 1s would roughly have these values.*



Gray-scale Images:

Gray-scale images are called **monochrome**, that is **one-colored** images. That one, as the name suggests gray. Each pixel will be represented by a single value ranging between 0-255 values.

In gray-scale images there are 256 gray levels representing 256 shades of one gray color with **0(black)**, and **255(white)**.

Colored Images:

Colored images have **three-band monochrome** data. The band is also called **channels**.

Red band: Values range 0-255

Green band: Values range 0-255

Blue band:Values range 0-255

Each pixel value is represented by 3 values as the combined value of three **bands**(**channels**), (R,G,B).

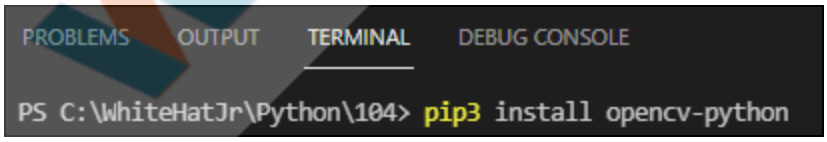
Did you know older movies were black and white and as technology improved, and colored images were introduced, this brought a major change in the film industry?

***Note**: Encourage the student to give answers and be more involved in the discussion.*

ESR: Varied.

Now that we understand different types of digital images, let us write a program to load and display these images in different color scales.

We can perform many computer vision tasks using a Python library called, **OpenCV,** which stands for **Open**

Source Computer Vision library, and it was developed by Intel in 1999. To use the functionalities of this **OpenCV** library we have to install the **opencv-python** module. ***Note**: The name of the library and module is different for installation.* What was the name of the package manager we used to install libraries? Yes, Correct! To install **opencv-python** module: 1. Create a **read_image.py** file in VSC Editor. 2. Open Command Prompt(Windows)/Terminal(Mac) and navigate to the current working directory(or Open the VSC Terminal) 3. Run the below command to install: **pip3 install opencv-python**	**ESR:** We used the package manager **pip3.**
	
In order to import the module, instead of using the name of the module, we have to use **cv2**. Open the **read_image.py** file and:	

1. Import open-python module:
   import cv2

2. Create a variable **img** to load the image using the **imread()** method**:**

   Syntax:

   ```
   cv2.imread("<filename>")
   ```

   Note: *The value inside <> is user defined.*
   Note*: Open Teacher Activity 2 to download image assets.*

   The **imread()** helps to load the image from a given path. The name of the file is passed as the argument to the **imread("<filename>")** method.

3. Display image using the **imshow()** method:
   Syntax:

   ```
   cv2.imshow("<display_window_name>", img)
   ```

   The **imshow()** method displays the image.
   The **display_window_name** is the name of the output window which automatically opens to display the image.
   Finally, we have the **img** to display.

```
import cv2

img = cv2.imread("butterfly.jpg")

cv2.imshow("Display Image",img)
```

4. Run the code using **python read_image.py** / **py read_image.py** command in the Command Prompt/Terminal.

```
>python read_image.py
```

***Note**: The output window will open for just a second and will close again and it will be difficult to see the output.*

Did you notice that the output appeared for just a glimpse and closed again?

To fix that, we will add a **waitKey()** method of **cv2** and run the code again.

This method takes time in milliseconds(1sec = 1000 milliseconds) as the parameter.

The **waitKey()** will keep the image open for:

- **Infinite time**: 0 milliseconds, the image window will be infinitely open until we close it manually.
- **Definite time**: Greater than 0 milliseconds, the image window will only be open for the given time.

ESR: Yes

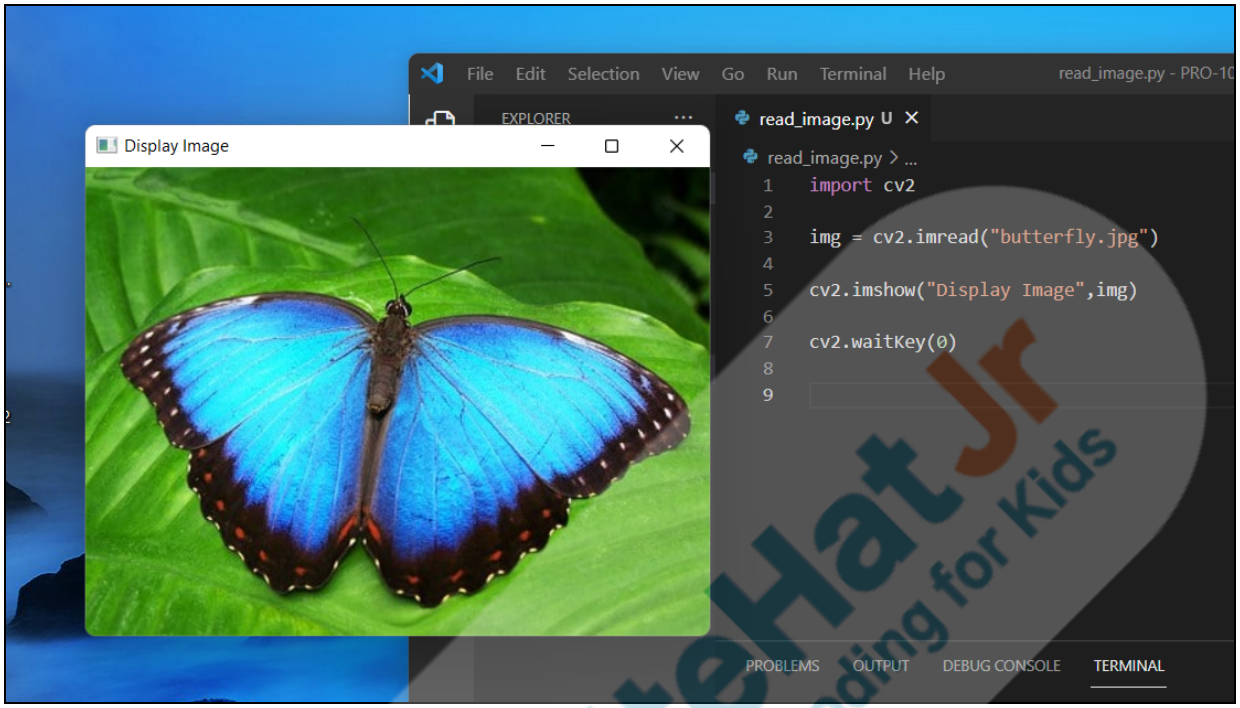
The window automatically closes after the given time.

***Note:** The **waitKey()** method returns the keycode values when a key is pressed on the keyboard which we will see in the next class.*

```
import cv2

img = cv2.imread("butterfly.jpg")

cv2.imshow("Display Image",img)

cv2.waitKey(0)
```

Output:

***Note**: The new window will pop-up. Notice the window name at top says "Display Image".*



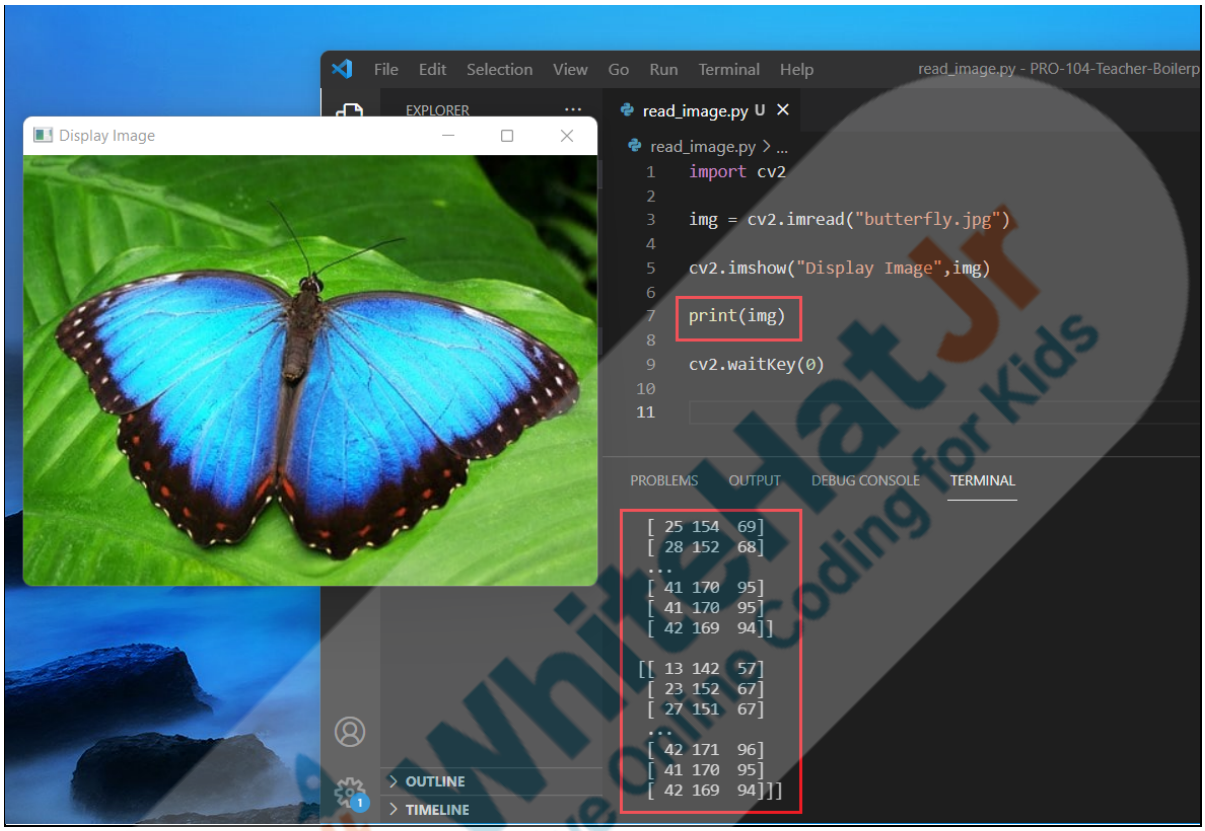
Now let's print the **img** variable using the **print()** method and see what we get:

***Note 1**: Remember to close the previous output window before running the code again.*

***Note 2**: The array data shown in the output result will be longer than what is shown here.*

```
import cv2

img = cv2.imread("butterfly.jpg")

cv2.imshow("Display Image",img)

print(img)

cv2.waitKey(0)
```

OUTPUT(In Terminal and In New Window)



Let's closely understand the output we got:

```
[[[ 40 148  75]
  [ 41 149  76]
  [ 25 154  69]
  [ 28 152  68]
  ...
  [ 41 170  95]
  [ 41 170  95]
  [ 42 169  94]]

 [[ 13 142  57]
  [ 23 152  67]
  [ 27 151  67]
  ...
  [ 42 171  96]
  [ 41 170  95]
  [ 42 169  94]]]
```

The output we received is a multi-dimensional array.

Let's first figure out how many dimensions are there in the array by looking at the output and then we will verify it through the program.

The array starts with 3 square brackets, this shows that the array is 3-dimensional.

Let's try to understand how 3D arrays are written.

***Note**: D stands for dimension(s).*

Till now you have worked with 1D arrays and 2D arrays.

Can you tell me how we write 1D and 2D arrays ?

- A **1D array** is a collection of data enclosed inside single square brackets. The elements are separated by comma.
  Python list can be used to form a 1D array.

- A **2D array** is a collection of data in the form of rows and columns.

ESR: Varied.

Python lists having lists as its element can be used from 2D array structure.

- A **3D array** is a collection of 2D arrays as data.

  Python lists having 2D lists as its element can be used from 3D array structure.

***Note**: Open [this] Google Colab link to explain the syntax of writing 1D, 2D and 3D arrays using Python list and run all cells.*

```
arr1_1d = [ 12, 77, 5]
arr2_1d = [ 52, 1, 99]
arr3_1d = [ 12, 77, 5]
arr4_1d = [ 52, 1, 99]
```

```
arr5_2d = [ [ 12, 77, 5],
            [ 52, 1, 99] ]

arr6_2d = [ [ 12, 77, 5],
            [ 52, 1, 99] ]
```

```
arr7_3d = [ [ [ 12, 77, 5],
              [ 52, 1, 99] ],
            [ [ 12, 77, 5],
              [ 52, 1, 99] ] ]
```

Now suppose we want to access some particular element of a 3D array, for this we start indexing the same way we have been doing 1D and 2D arrays before.

The indexing starts from the outermost to the innermost array.

For example if we want to access the first element of the 1st 2D array, then we need to specify the indexing as [0][0][0].

We will look into accessing elements in 2D arrays/lists as proceed in the class.

Now that we know how 3D arrays are written, why was the output of printing the image displayed 3D array?

Yes. Amazing!

ESR: Because we printed the colored image data.

The colored image is made up of 3 bands, Red, Green and Blue. The Red band, the Blue band and Green band is a 2D array. All put together form a 3D array of colored image data.

Generally,

```
colored_image_array = [
                        [ Red_2D_Array ],
                        [ Green_2D_Array ],
                        [ Blue_2D_Array ]
                      ]
```

It is important to remember that in general, the colored images are represented with **RGB** bands sequence, but in **OpenCV** it is represented as **BGR** bands sequence. Although it does not affect the output overall as only the sequence is changed, not the numbers which form the colors all together!

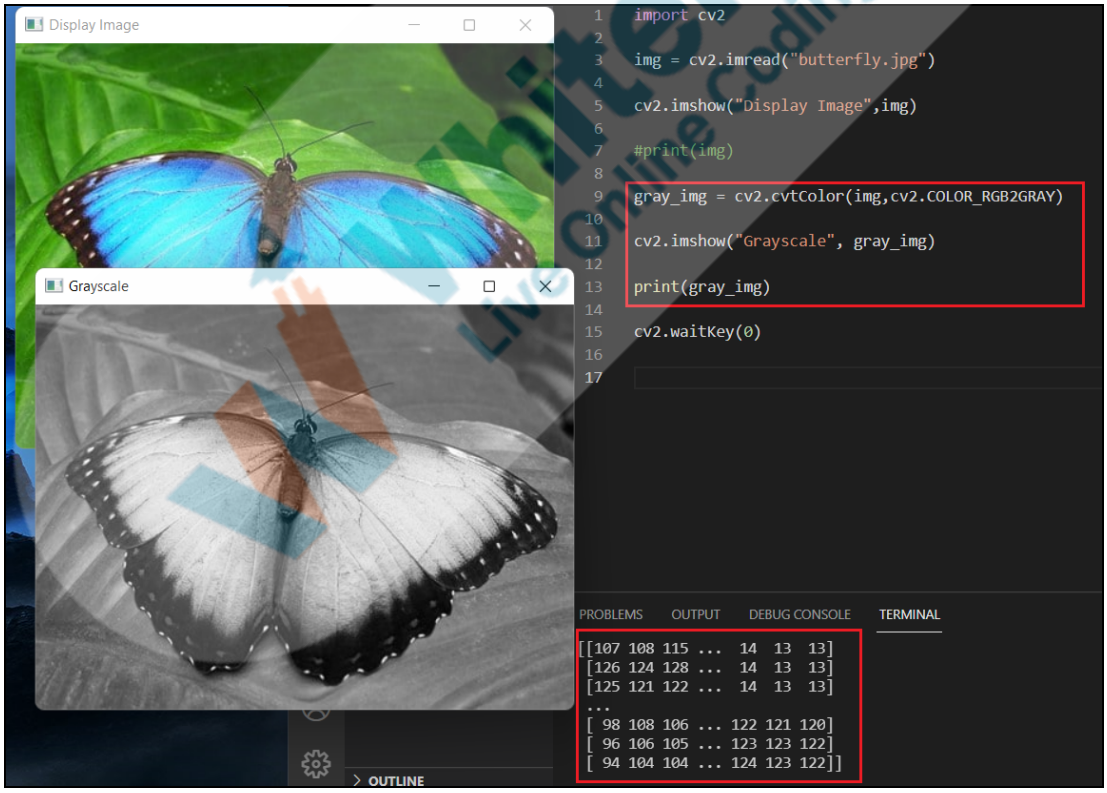
In **OpenCV**,

```
colored_image_array = [
                        [ Blue_2D_Array ],
                        [ Green_2D_Array ],
                        [ Red_2D_Array ]
                      ]
```

What do you think about how gray-scale image array data can be represented? Great! **grayscale_image_array** = [GraySacle_2D_Array] OpenCV can help us convert our colored images to grayscale easily. Since grayscale images are made up of only one color shade, these images are lighter and hence faster to process. We can use the **cvtColor(image, cv2.COLOR_RBG2GRAY)** method and pass **image** and **COLOR_RBG2GRAY** property to convert colored images into grayscale: 1. Take a variable **gray_img** 2. Use **cvtColor(image, cv2.COLOR_RBG2GRAY)** to assign the converted image to the **gray_img** variable. 3. Display image using **imshow()** method. 4. Print the **gray_img** variable to see the output as a 2D array.	**ESR:** It will be represented as a 2D array.

```
import cv2

img = cv2.imread("butterfly.jpg")

cv2.imshow("Display Image",img)

#print(img)

gray_img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)

cv2.imshow("Grayscale", gray_img)

print(gray_img)

cv2.waitKey(0)
```

OUTPUT(In Terminal and In New Window)



Now that we know how image data is represented as an array, we are going to work like a programmer working with the core concepts.

We will write a program to:

1. Create an image using arrays.
2. Modify the image using arrays.

We will create an image with a black background using arrays and then modify it to add a white block in the center of the image.

To create an image with a black background using arrays we will use the **NumPy** library.
NumPy stands for '**Numerical Python**'. Using NumPy, mathematical and logical operations on arrays can be performed.

Numpy has many methods, for now, we will use its method **.zeros()** to create an array filled with all **0.**

What kind of image can we create using 0s?

Correct!

ESR: A complete black image.

Let me start with installing the **NumPy** library using package installer **pip3**

1. Install **> pip3 install numpy**

   ***Note**: If your system has only one version of Python installed, the only pip would work.*

```
> pip3 install numpy
```

2. Create **image_with_arrays.py** and import **numpy** and **cv2.**

```
import numpy as np
import cv2
```

3. Create a variable **black.**

4. Assign with **numpy.zeros([600,600]).** The size of the array will be 600 width and 600 height which we are passing as a **list. This is also represented as 600x600 pixel values.**

5. Print the image data using **print()** method and display the image data using **imshow()** method.
6. Add **cv2.waitKey(0)** to add time delay.

7. Run the code using:

```
python image_with_arrays.py
```

```
# Create a black image
black = np.zeros([600,600])
print(black)


cv2.imshow("black",black)
cv2.waitKey(0)
```

OUTPUT(In New Window)

OUTPUT(In the Terminal)

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

We could successfully create an image with a black background using arrays.

For the time being to understand arrays better let me change the size of the array so that we can read it completely.

8. Change the size to **6 x 6**

9. Comment the **cv2** statements as currently, we won't be seeing the image.

10. Run the code.

```
import numpy as np
import cv2

# Create a black image
black = np.zeros([6,6])
print(black)

# cv2.imshow("black",black)
# cv2.waitKey(0)
```

OUTPUT(In the Terminal)

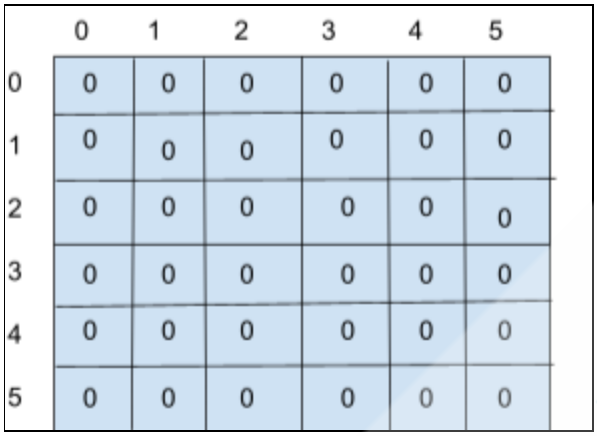
```
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
```

We can see an array or list of 6 columns and 6 rows with all values as 0.

Let's understand how we can access elements from 2D

Python lists.

***Note**: The teacher can open a link while explaining the concept to the student.*



	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

We can access these numbers using a range of rows and columns.

Let me show you few examples:

The elements of the matrix (also called 2D Python lists or 2D NumPy arrays) can be accessed by giving a range of:

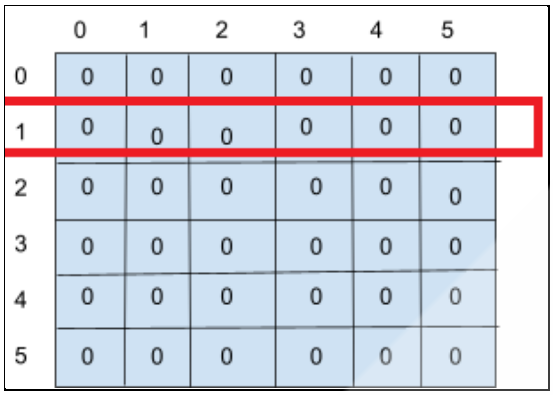
[start_row : end_row, start_column : end_column]

- **start_row**: Starting index of the row from which we want to access the data.
- **end_row**: Index of the next row before which we want to access the data.
- **start_column**: Starting index of the column from which we want to access the data.
- **end_column**: Index of the next column before which we want to access the data.

***Note**: Help the student to recollect accessing the elements*

of a Python list using the slice operator discussed in class 98.

Let us consider that we want to access the elements from the **second row** of the matrix.



In that case, which one is the start row?

ESR: 1st

Which will be the end row?

ESR: 2nd

Correct!

To access elements of the **second row**, we need to tell the computer to **start from the 1st row** and **stop right before the 3rd row**, that is, [1:2]. Columns can be kept blank if we want all the columns:

1. Create a variable **f_row**.
2. Assign the row and column range of the **black** array variable to access the particular range.
3. Run the code to see the output in the terminal.

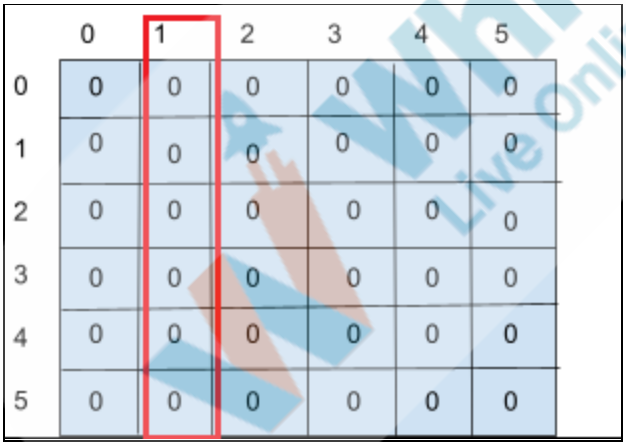
```
# Create a black image
black = np.zeros([6,6])
#print(black)

f_row = black[1:2]
print(f_row)

# cv2.imshow("black",black)
# cv2.waitKey(0)
```

OUTPUT(In the Terminal)

```
[0. 0. 0. 0. 0. 0.]
```

Similarly, let us now access the **second column**. We need to keep all rows, hence we will not give any numbers there.

Can you help me get the numbers for the second column?



ESR: We need columns from 1:2

```
# Create a black image
black = np.zeros([6,6])
#print(black)

# f_row = black[1:2]
# print(f_row)

f_col = black[:,1:2]
print(f_col)

# cv2.imshow("black",black)
# cv2.waitKey(0)
```

OUTPUT(In the Terminal)

```
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

Cool, isn't it? Now, can you tell me if I want to access 3rd and 4th rows and 3rd and 4th column, what should I write?	**ESR:** Yes. **ESR:** Varied

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

The range will be **[2:4 , 2:4]**

1. Set row range: Start from the 3rd row, stop before the 5th row.
2. Set column range: Start from the 3rd row, stop before the 5th column.
3. At the same time we will assign 255 values to this range.
4. Print the entire array using **print(black).**

```
black = np.zeros([6,6])

# # f_row = black[1:2]
# # print(f_row)

# # f_col = black[:,1:2]
# # print(f_col)

black[2:4,2:4] = 255
print(black)

cv2.imshow("black",black)
cv2.waitKey(0)
```

OUTPUT(In the Terminal)

```
[[  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0. 255. 255.   0.   0.]
 [  0.   0. 255. 255.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.]]
```

Let us check the image output by removing the comments from **imshow().**

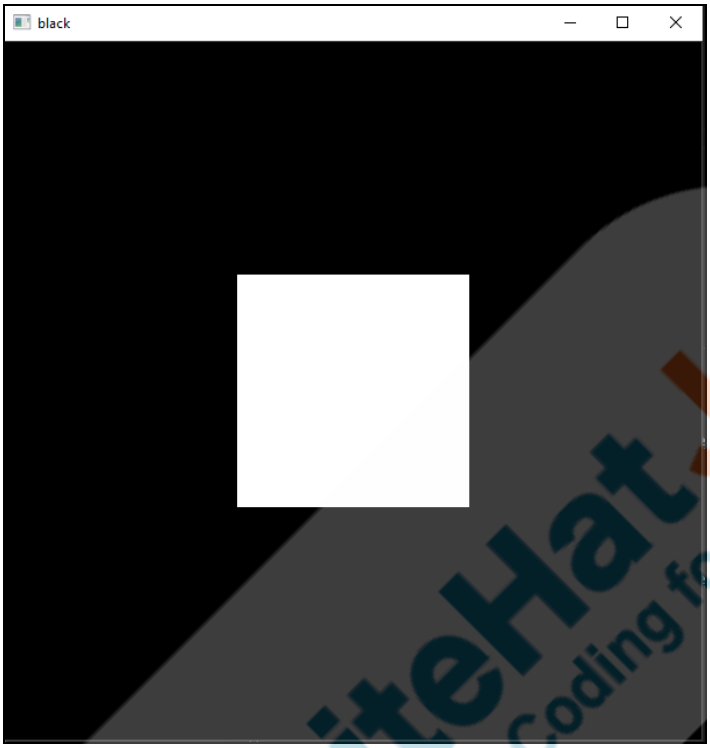
The image output is very small. Let us add 0s to make the size bigger:

1. Update size to **600 x 600**.
2. Update arrays range **[200:400, 200:400].**
3. Run the output again.

***Note**: Make sure to close the previous image window.*

```
#black = cv2.imread("white.png")
# Create a black image
black = np.zeros([600,600])

# # f_row = black[1:2]
# # print(f_row)

# # f_col = black[:,1:2]
# # print(f_col)

black[200:400,200:400] = 255
print(black)

cv2.imshow("black",black)
cv2.waitKey(0)
```

OUTPUT(In the New Window)



In this manner, all the image processing applications are altering the numbers in an image array (matrix) to modify them.

In this activity, we learned about different types of images and how they are created and read by the computer.

Now, let us have some fun with the images, we will make a beautiful "**personalized image**" by modifying the image using an array of data.

Are you excited?

Share your screen, and I will guide you through it.

ESR: Yes

Teacher Stops Screen Share

Teacher Starts Slideshow

Slide 24 to 25

Refer to speaker notes and follow the instructions on each slide.

We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	

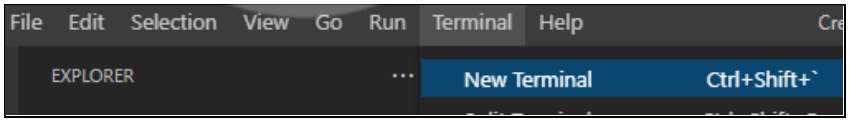
Teacher Ends Slideshow

STUDENT-LED ACTIVITY - 15 mins

- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Fullscreen.**

ACTIVITY

- **Read and view the given image**
- **Extract and paste the image data.**

Teacher Action	Student Action
What should we do first to access the image? Let us first install **OpenCV:** 1. Open the VSC editor terminal.  2. Run the Code to install OpenCV.	**ESR:** Install OpenCV

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\WhiteHatJr\Python\104> pip3 install opencv-python
```

3. Import **cV2** in **create_card.py.**

```
import cv2
```

*Guide to student to display **poster.jpg** image:*

***Note**: Open Student Activity 1 to download image assets.*

1. Create a variable **img = cv2.imread("poster.jpg").**
2. Use **cv2.imshow("output",img).**

   ***Note**: The **"output"** can be any word, this word will appear on a new pop up screen to display the image.*

3. Add a **waitKey()** method to set the delay in milliseconds(0 is infinite time).

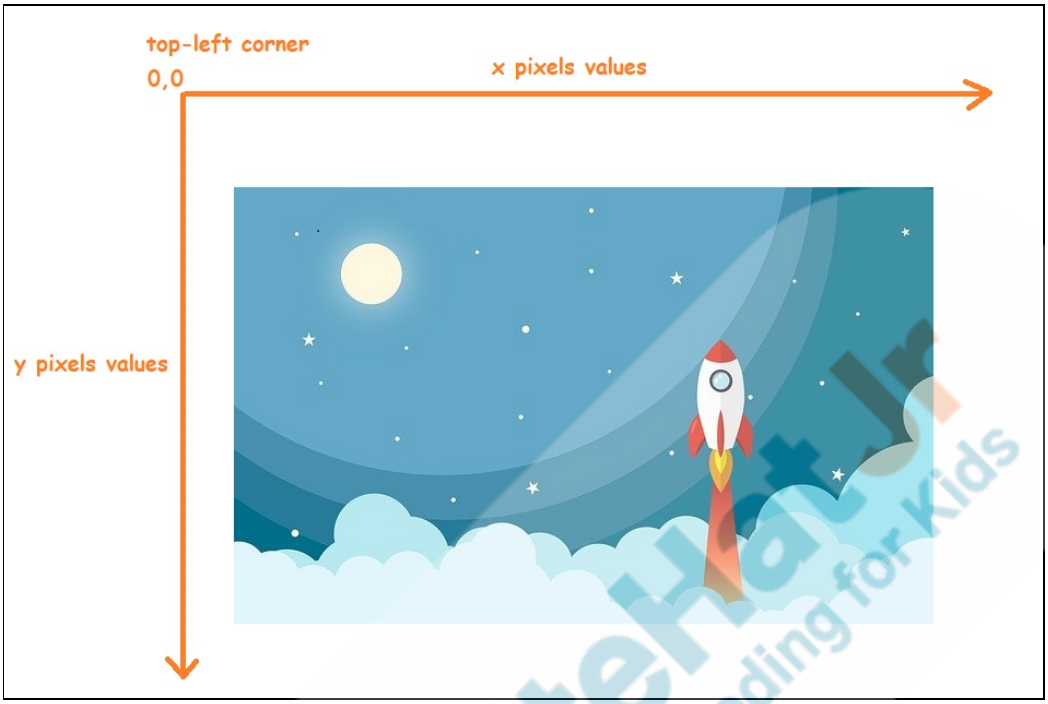
```
import cv2

img = cv2.imread("poster.jpg")

cv2.imshow("output",img)

cv2.waitKey(0)
```

OUTPUT(In New Window)

Superb!

Now, let us modify the image using **NumPy** arrays.

***Note**: The image pixel values start from the **top-left corner** of the output window. Help the student recollect the same concept they used in game development.*



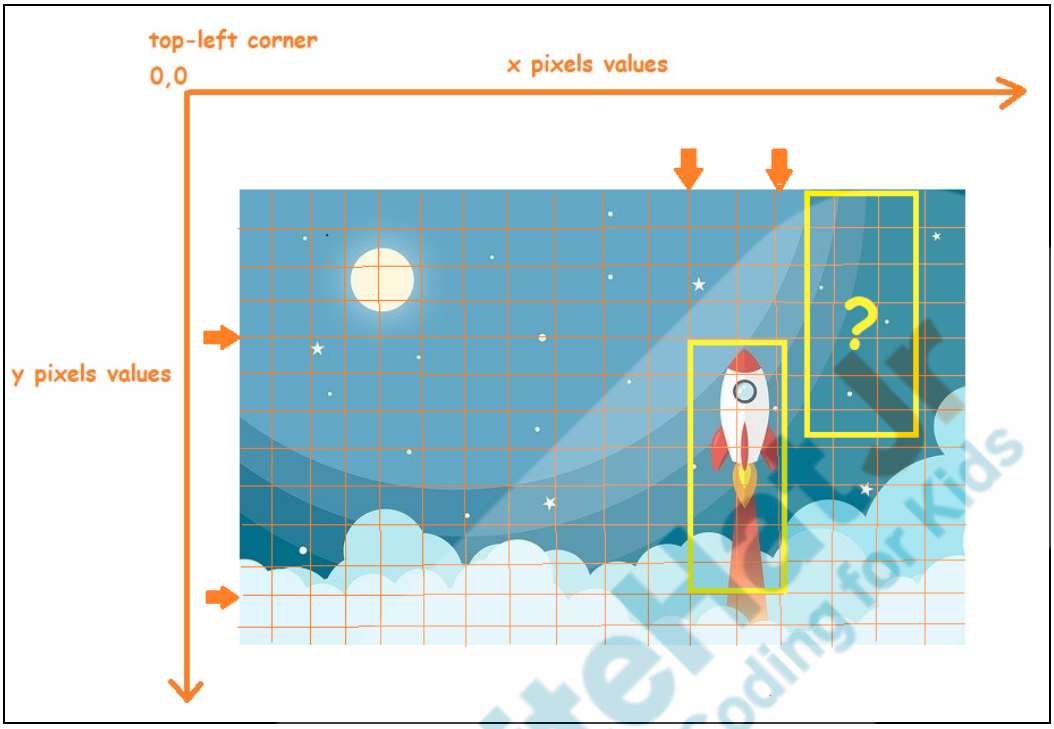
We will crop the rocket part with the help of array pixel values and repaste it on the image, which will give us **two rockets in one image**. To do so:

1. Create a variable **rocket** to store a part of a 2D NumPy array.

2. Assign a range **of pixel values in x and y direction.**

   The first pair of numbers represents **columns (y pixels values)** & the second pair of numbers represents **rows(x pixel values)**.

***Note**: Below image is only for representational purposes. This image shows the starting point of the image pixels and direction x and y pixels values. Also it is a rough representation of how to specify the start x & y and end x & y.*



We have done trial and error to get this value, you can try changing these numbers later to see the difference.

Now assign the array captured in the rocket to a different place in the image. Make sure the range is equal to what we have captured.

- From **120:360** there is a difference of **240 px.**
- From **400:500** there is a difference of **100 px.**

We need to give the numbers with exact difference of pixel values to paste the image.
***Note**: The teacher can allow students to provide the numbers.*

Remember, the first pair of numbers represents columns & the second pair of numbers represents rows.

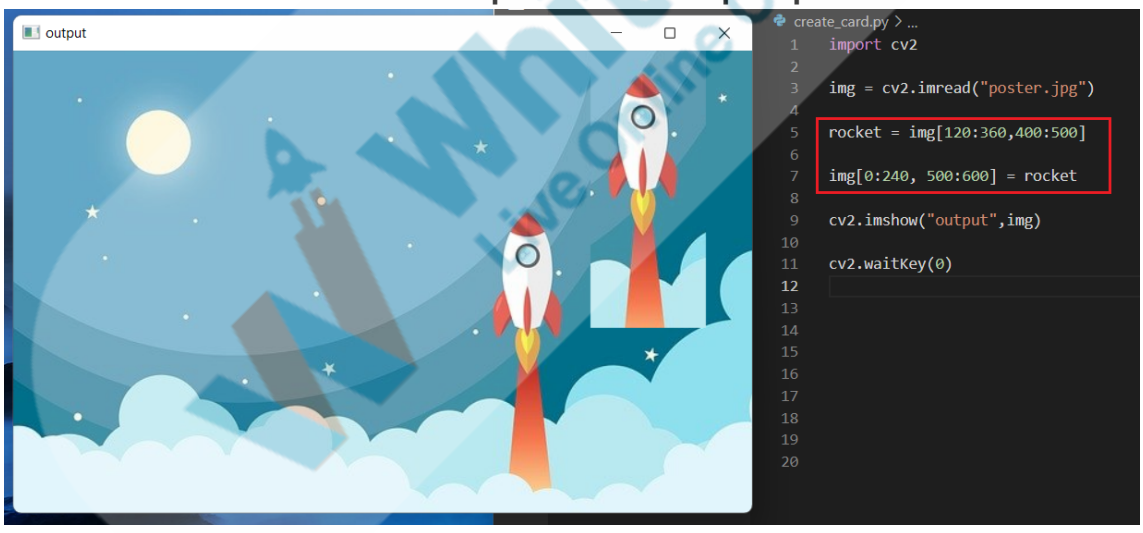
3. Select a new range and assign **rocket** to it.

img[30:190,480:580] = rocket.

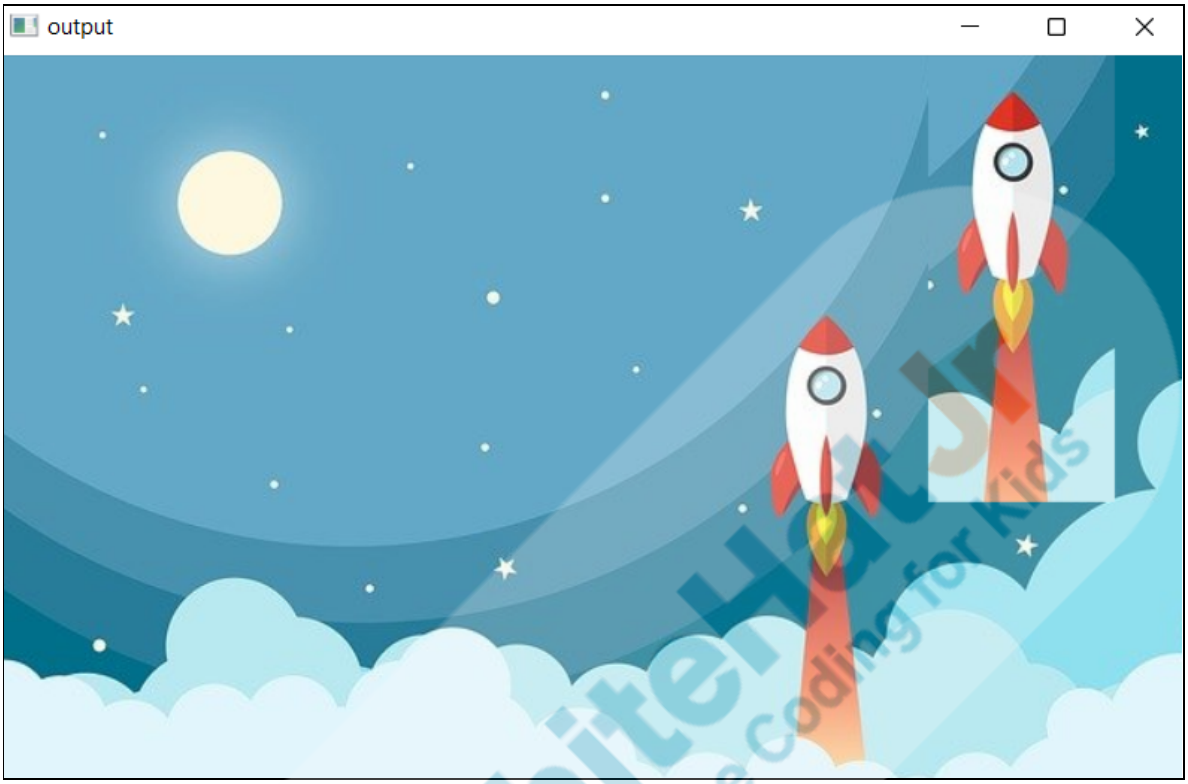
4. Run the code.

```
import cv2

img = cv2.imread("poster.jpg")

rocket = img[120:360,400:500]

img[0:240, 500:600] = rocket

cv2.imshow("output",img)

cv2.waitKey(0)
```

A New Output Window Pops-Up



OUTPUT(In New Window)



***Note**: If time permits allow the student to experiment with different numbers to assign different positions which they feel would be best.*

Let us perform some more actions on the image.

Let us put some text on the image.

***Note**: A sample text is given here, but the teacher can allow the student to add text of their own choice as well.*

1. Create a variable **text_to_show** to assign a string to it.

   text_to_show = "I love coding at WhiteHatJr"

2. Use the **putText()** method to add text over the image**.** To add text over the image with the OpenCV library of Python, use the **putText()** method.

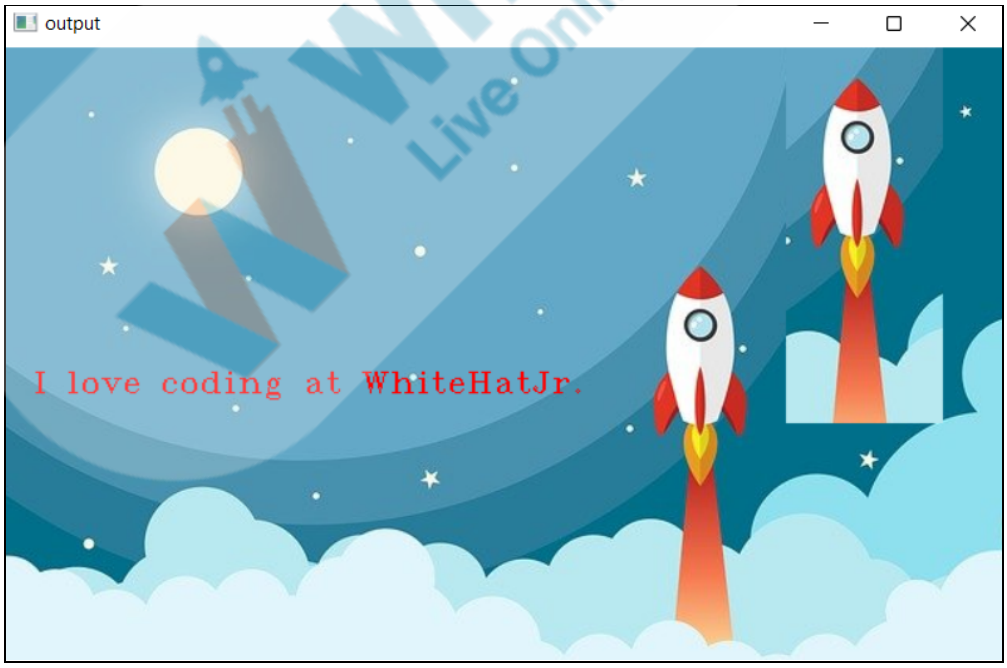
The **putText()** method accepts the following parameters:

- **image** to write the text on.
- **text** to write on the image.
- **position:** Distance along the horizontal (x) and vertical (y) axis from the top left corner of the image.
- **font family:** OpenCV supports many font families, you can choose one from the drop-down.
- **fontScale:** Size of the fonts
- **font color:** We can give (B, G, R) values to set the color. To represent the Red color, we can set (0, 0, 255). The convention for color band sequence is RGB but in OpenCV the sequence is BGR.

***Note**: If the **putText()** is used after **.imshow()** the text will not be visible as the image will get displayed before adding the text since the code always gets executed in sequential order.*

```
import cv2

img = cv2.imread("poster.jpg")

rocket = img[120:360,400:500]

img[0:240, 500:600] = rocket

text_to_show = "I love coding at WhiteHatJr."


cv2.putText(img,
            text_to_show,
            (20, 220),
            fontFace=cv2.FONT_HERSHEY_COMPLEX_SMALL,
            fontScale=1,
            color=(0,0,255)
            )

cv2.imshow("output",img)

cv2.waitKey(0)
```

OUTPUT(In New Window)

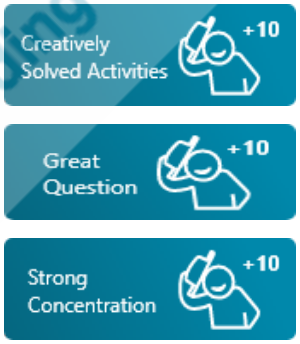


Let us stop here today, you can keep exploring OpenCV after class too. In upcoming classes, we will learn a lot more about OpenCV. Did you enjoy the class? Let me ask you a few questions to know if your concepts are clear.	**ESR:** Yes
Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 05 mins	
Teacher Starts Slideshow **Slide 26 to 32** 	
Activity details **Following are the WRAP-UP session deliverables:** ● Appreciate the student. ● Revise the current class activities. ● Discuss the quizzes.	
WRAP-UP QUIZ Click on In-Class Quiz	
Continue WRAP-UP Session **Slide 33 to 38** 	
Activity Details: **Following are the session deliverables:** ● Explain the facts and trivia	

- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate student's efforts in the class.**
- **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**

Teacher Action	Student Action
You get "hats-off" for your excellent work! In the next class, we will learn how to process a web camera feed and a video feed using OpenCV.	*Make sure you have given at least 2 hats-off during the class for:* 

PROJECT OVERVIEW DISCUSSION

Refer the document below in Activity Links Sections

Teacher Clicks ✖ End Class

ADDITIONAL ACTIVITIES

Teacher Action	Student Action
Save the image:	

If you want to save the image and pass it onto your friends and family, how will you pass it?

We can save the image in the local machine and share it with others. For that we have a simple method **cv2.imwrite().**

It is as simple as **cv2.imshow().**

The **cv2.imwrite()** method accepts two parameters:

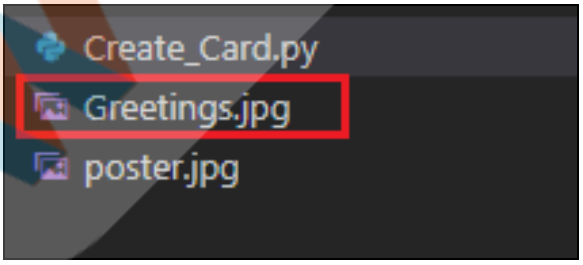
1. Name of the image with extension.
2. Image which is to be saved.

You can write: **cv2.imwrite("Greetings.jpg",img)**

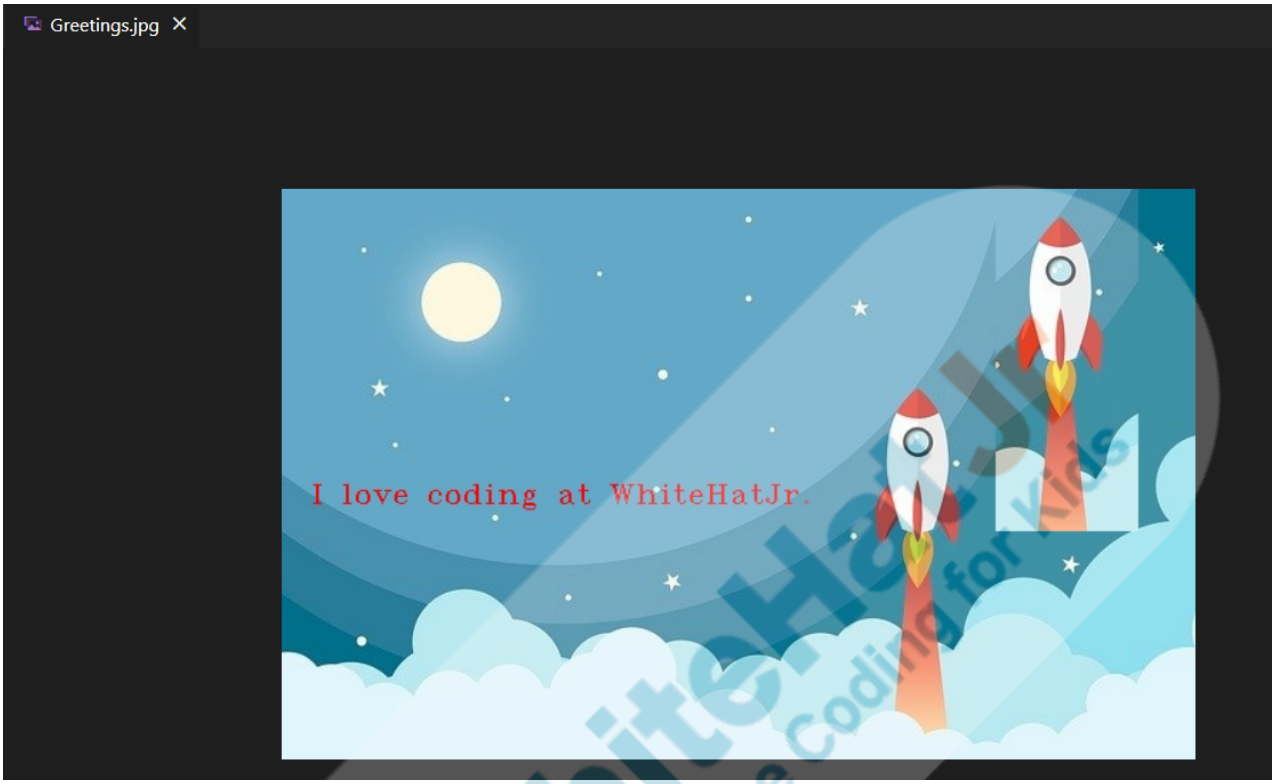
***Note**: Instead of **Greetings.jpg** any name can be used with extension .png & .jpg.*

```
cv2.imshow("output",img)
cv2.imwrite("Greetings.jpg",img)

cv2.waitKey(0)
```

OUTPUT



Click on "**Greetings.jpg**" to see the saved image.



If you see on the left side where files are kept, you will be able to see a new file called **Greetings.jpg** is created.

That's amazing!
We altered the image by adding a text and saving it with new changes.

We can do a lot of cool things using OpenCV.

Keep exploring!

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Binary Image Illustration	https://s3-whjr-curriculum-uploads.

		whjr.online/e27e2af9-25c6-4812-80bc-88b6c40f24ad.gif
Teacher Activity 2	Image Assets	https://github.com/procodingclass/PRO-104-OpenCV-Image-Assets
Teacher Activity 3	Reference Code	https://github.com/procodingclass/PRO-104-Reference-Code
Student Activity 1	Image Assets	https://github.com/procodingclass/PRO-104-OpenCV-Image-Assets
Teacher Reference 1	Access Elements of 2D Python Lists	https://s3-whjr-curriculum-uploads.whjr.online/4424f727-c370-41a7-b12c-b70f4ef94c52.pdf
Teacher Reference 2	1D, 2D, 3D Array Syntax	https://colab.research.google.com/drive/1CzbzD2vhvlsIGcCYOtiE_NENnT-Cp4pe?usp=sharing
Teacher Reference 3	Project Document	https://s3-whjr-curriculum-uploads.whjr.online/0b98173a-5234-4469-9f57-5818a4c3e60d.pdf
Teacher Reference 4	Project Solution	https://github.com/procodingclass/PRO-104-Project-Solution
Teacher Reference 5	Visual-Aid	https://s3-whjr-curriculum-uploads.whjr.online/dacbd8a0-d902-409b-b76a-a458fc4228e3.html
Teacher Reference 6	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/7e1e8103-2178-4243-89fa-54c474a64eb3.pdf