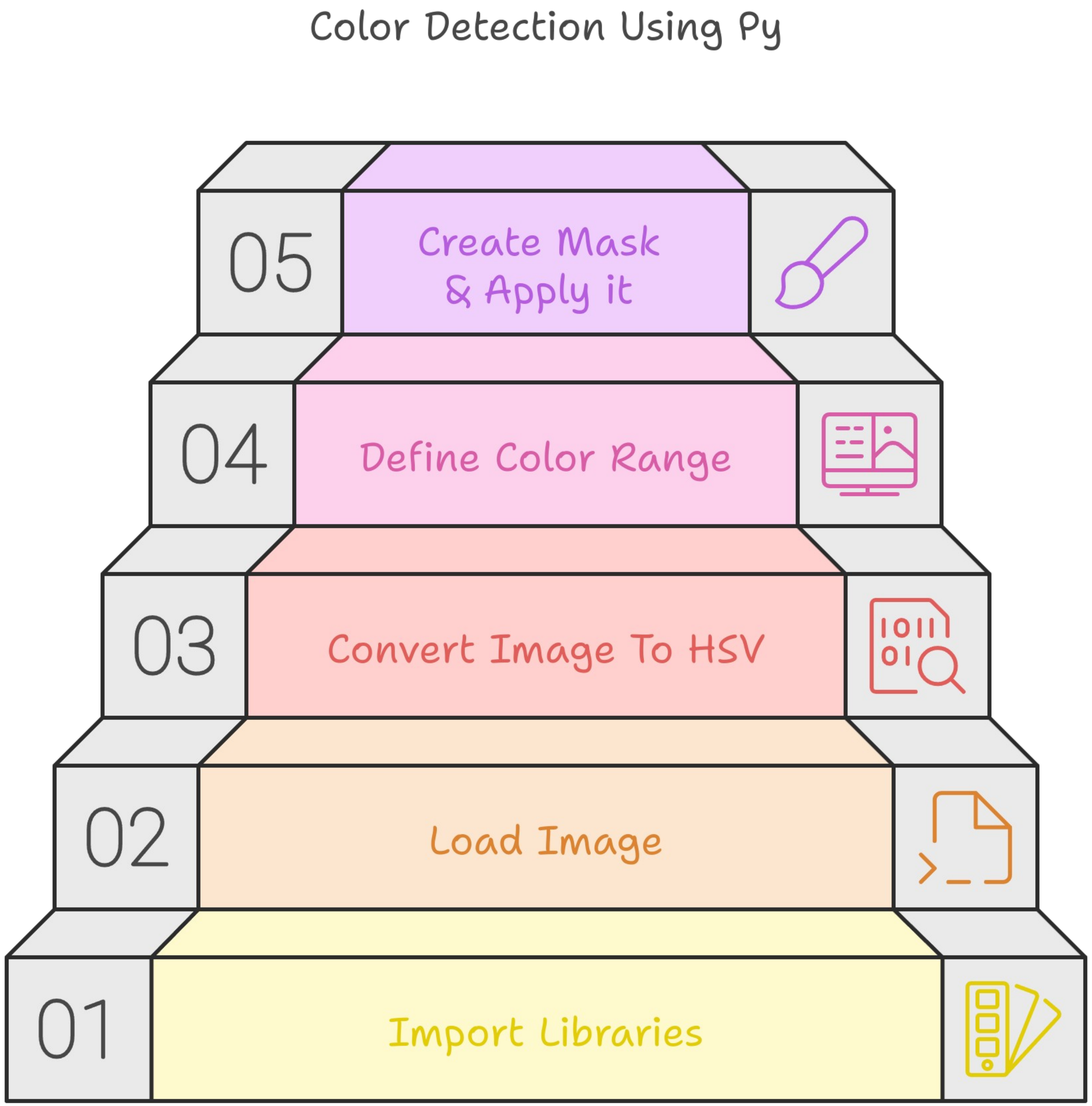
Color Detection in Python

- Color detection is a fundamental task in computer vision that involves identifying and classifying colors within an image. This document provides an overview of how to implement color detection using Python, focusing on libraries such as OpenCV and NumPy. By the end of this guide, you will have a clear understanding of how to detect specific colors in images and how to manipulate image data for various applications.


Color Detection Using Py

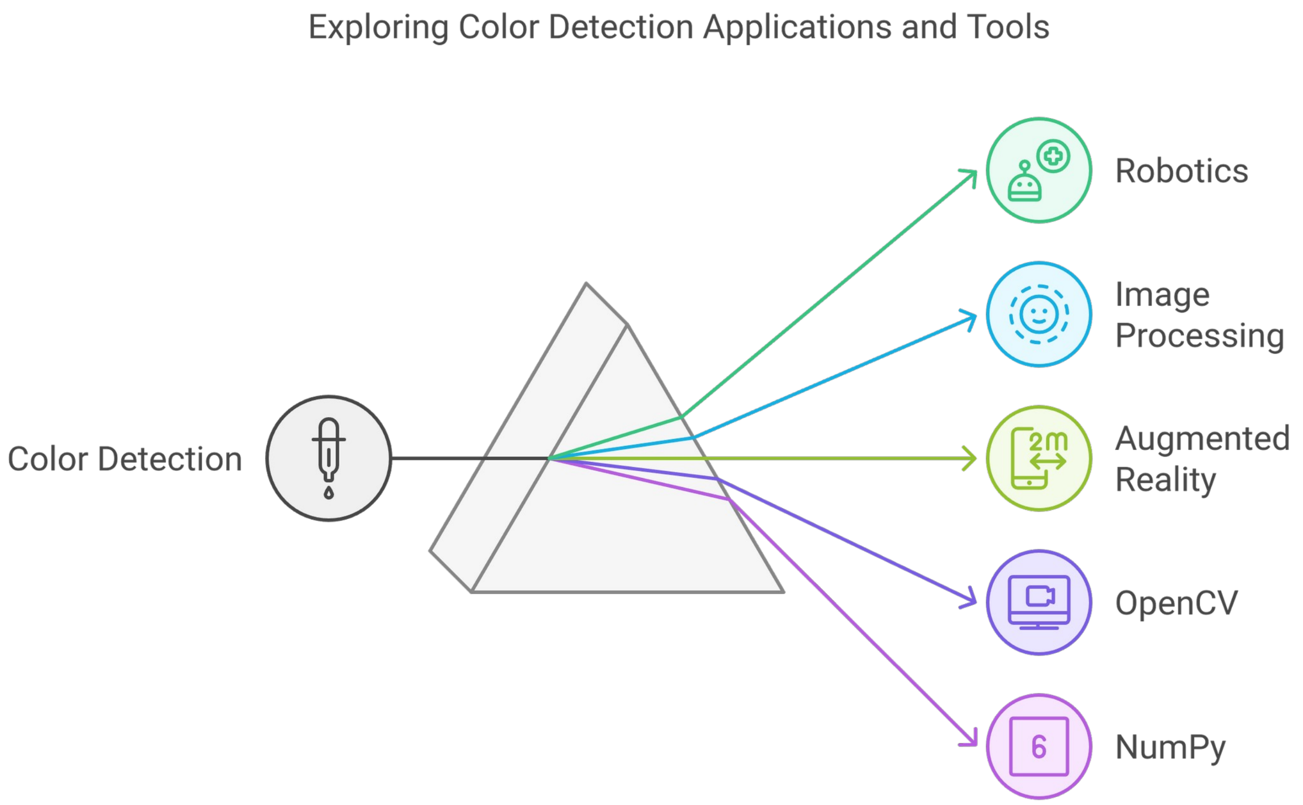
05 Create Mask & Apply it

04 Define Color Range

03 Convert Image To HSV

02 Load Image

01 Import Libraries

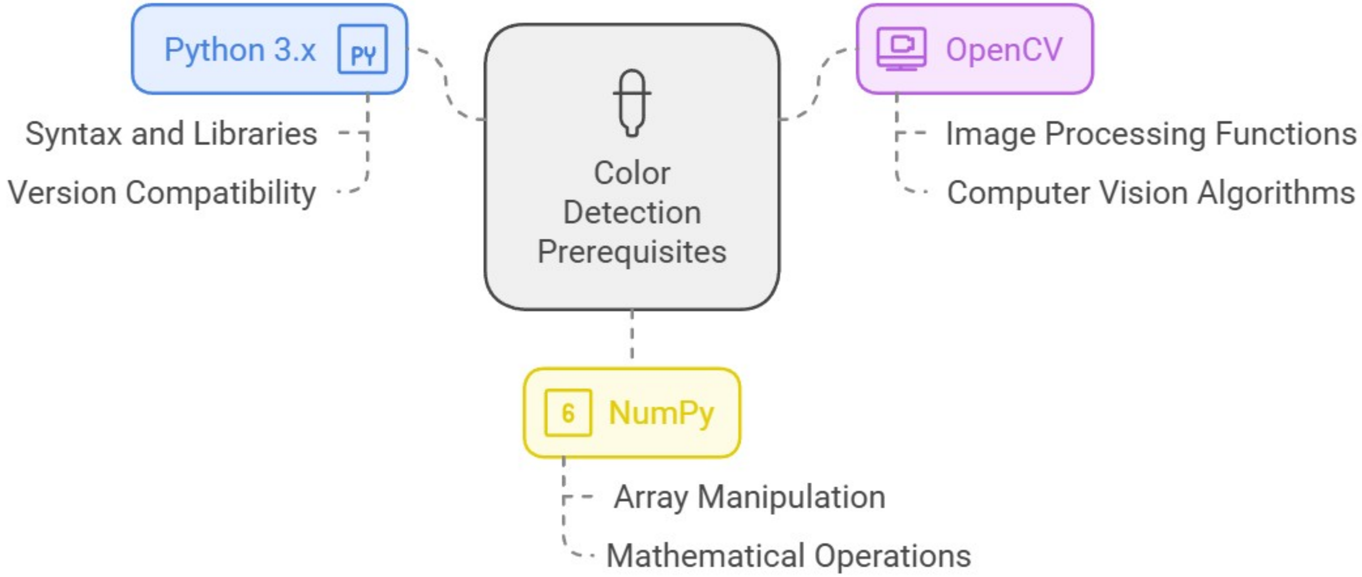


Introduction

- Color detection can be applied in various fields, including robotics, image processing, and augmented reality. Python, with its rich ecosystem of libraries, makes it easy to implement color detection algorithms. OpenCV, a powerful library for computer vision tasks, provides tools for image processing, while NumPy allows for efficient numerical operations.

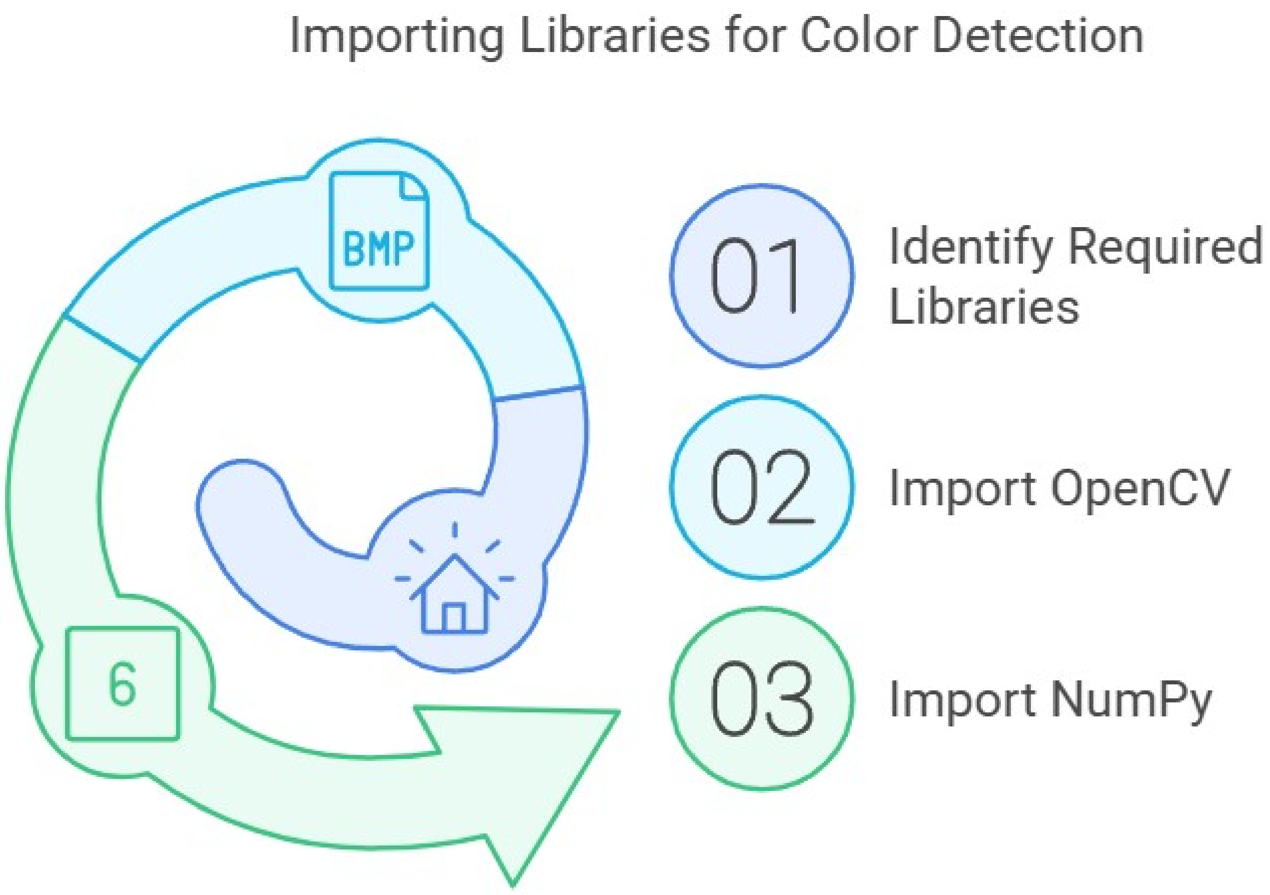
Prerequisites

- Before diving into color detection, ensure you have the following installed:
- Python 3.x
- OpenCV
- NumPy



Step 1

Start by importing the necessary libraries:



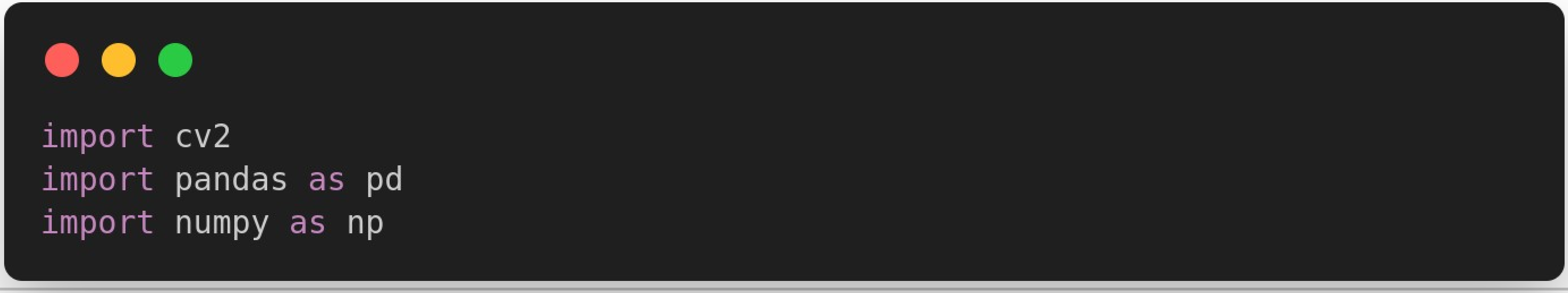
Importing Libraries for Color Detection

01 Identify Required Libraries

02 Import OpenCV

03 Import NumPy

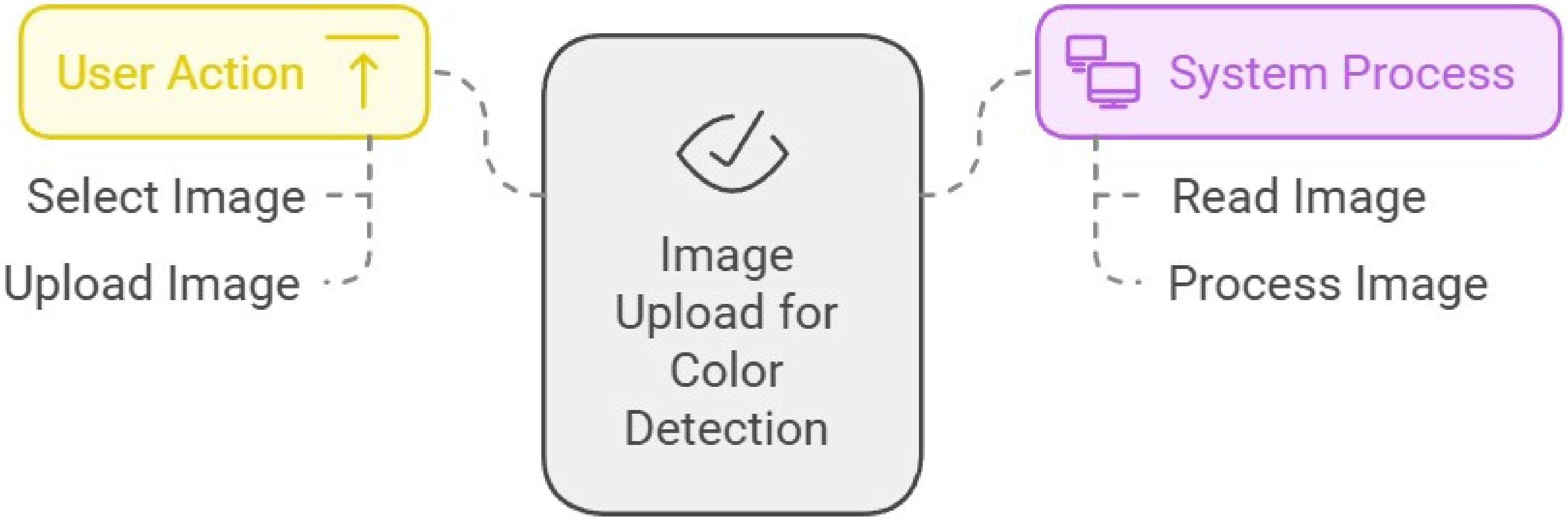
In Our Code :



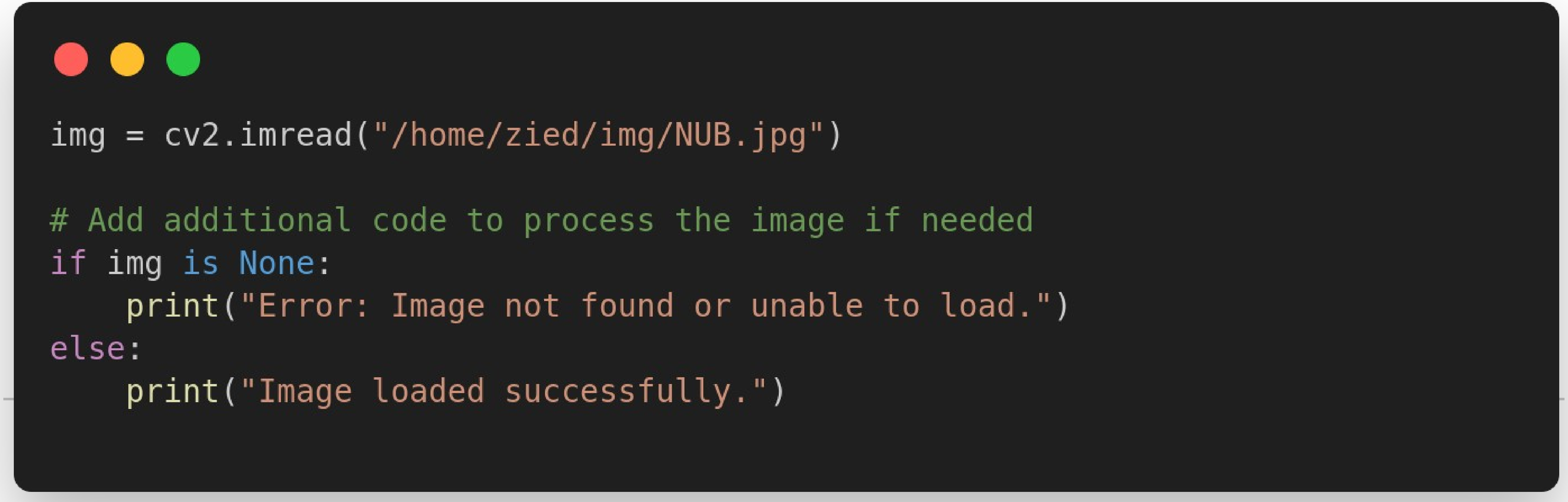
```python
import cv2
import pandas as pd
import numpy as np
```

Step 2: Load an Image

Load the image in which you want to detect colors:



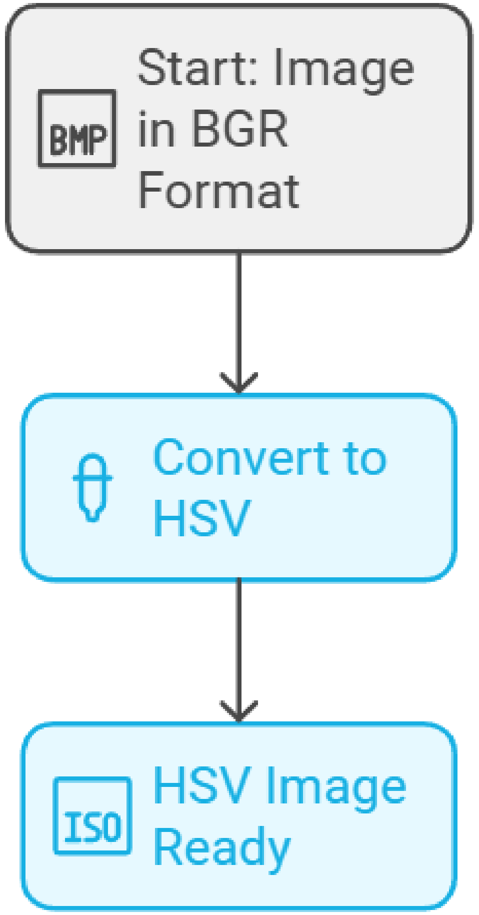
In Our Code :



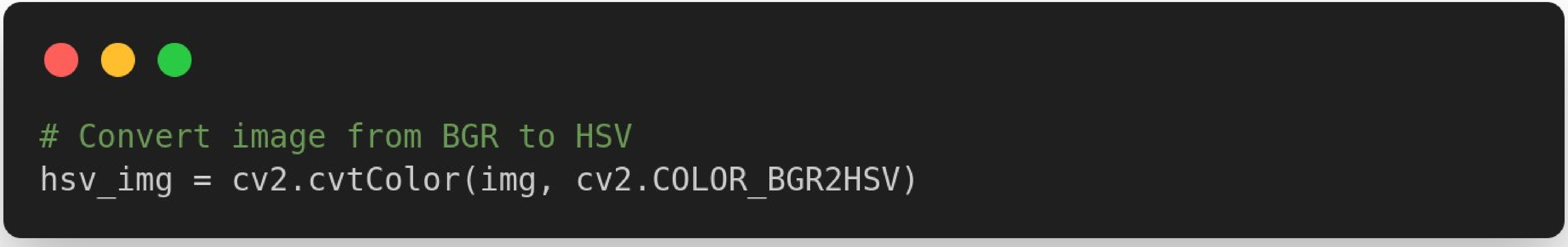
```
img = cv2.imread("/home/zied/img/NUB.jpg")

# Add additional code to process the image if needed
if img is None:
    print("Error: Image not found or unable to load.")
else:
    print("Image loaded successfully.")
```

Step 3: Convert the Image to HSV

- Color detection is often more effective in the HSV (Hue, Saturation, Value) color space. Convert the image from RGB (default in OpenCV) to HSV:



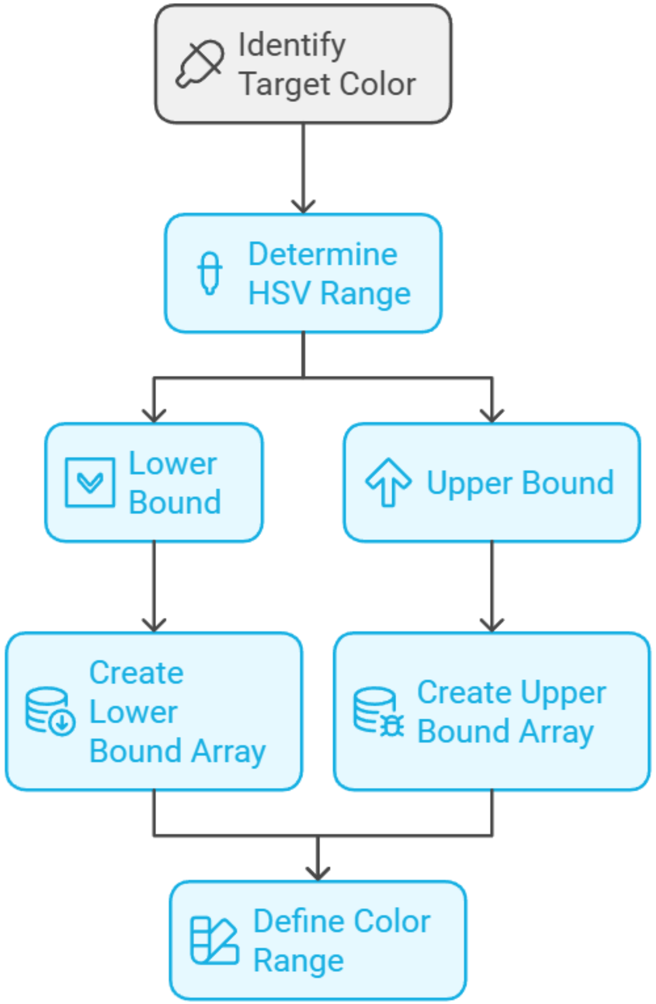
In Our Code :



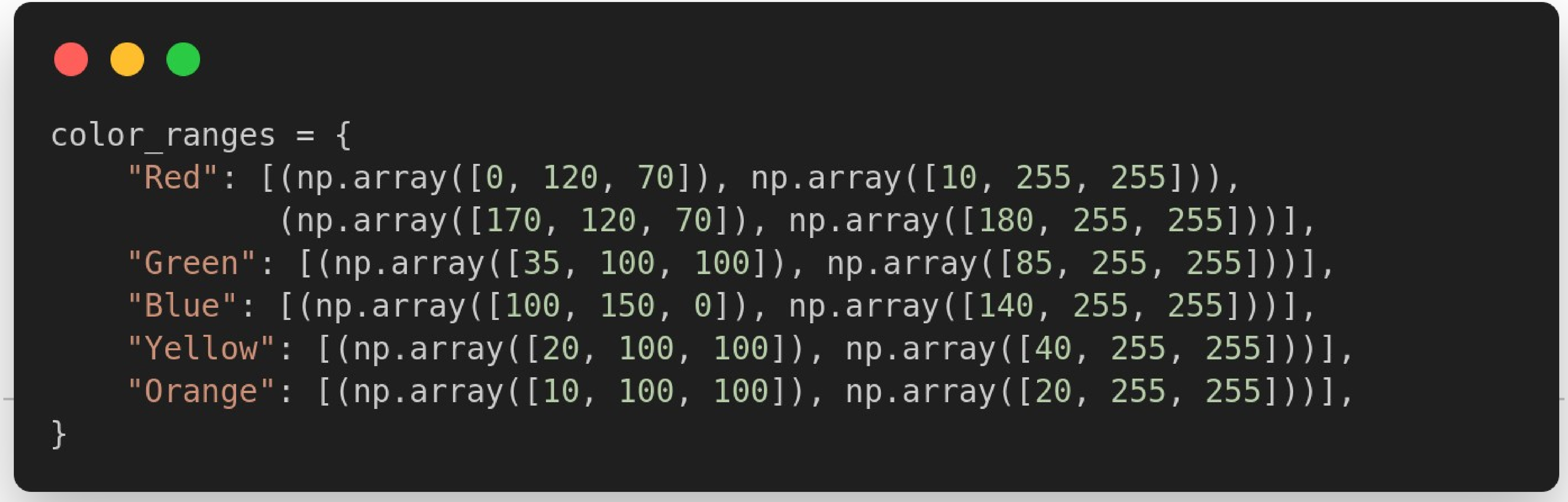
```python
# Convert image from BGR to HSV
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

Step 4: Define Color Range

- Define the color range you want to detect. For example, to detect red color:

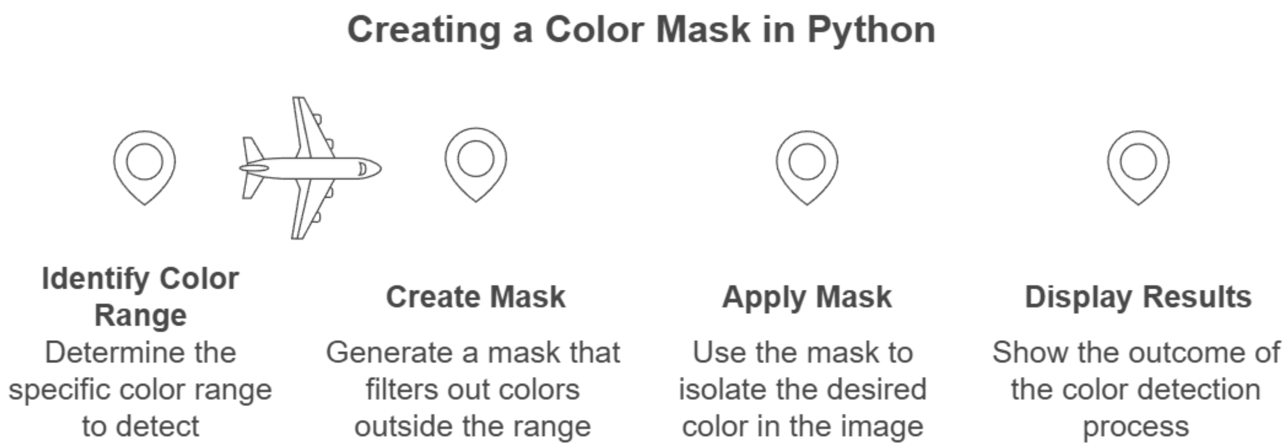


In Our Code :



```
color_ranges = {
    "Red": [(np.array([0, 120, 70]), np.array([10, 255, 255])),
            (np.array([170, 120, 70]), np.array([180, 255, 255]))],
    "Green": [(np.array([35, 100, 100]), np.array([85, 255, 255]))],
    "Blue": [(np.array([100, 150, 0]), np.array([140, 255, 255]))],
    "Yellow": [(np.array([20, 100, 100]), np.array([40, 255, 255]))],
    "Orange": [(np.array([10, 100, 100]), np.array([20, 255, 255]))],
}
```

Step 5: Create a Mask



Creating a Color Mask in Python

Identify Color Range
Determine the specific color range to detect

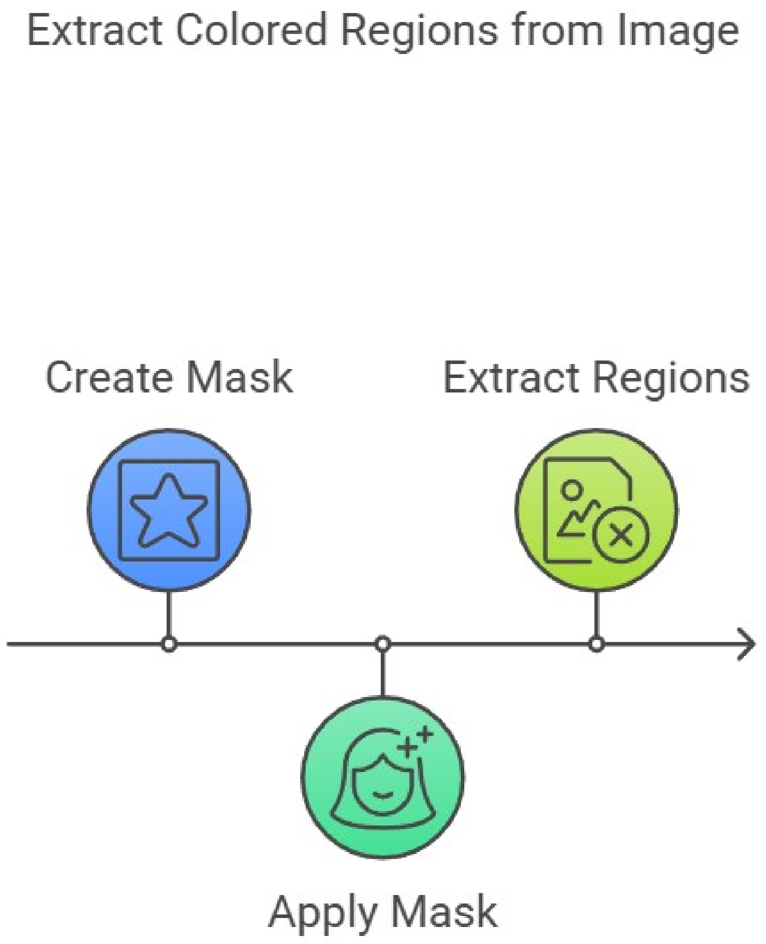
Create Mask
Generate a mask that filters out colors outside the range

Apply Mask
Use the mask to isolate the desired color in the image

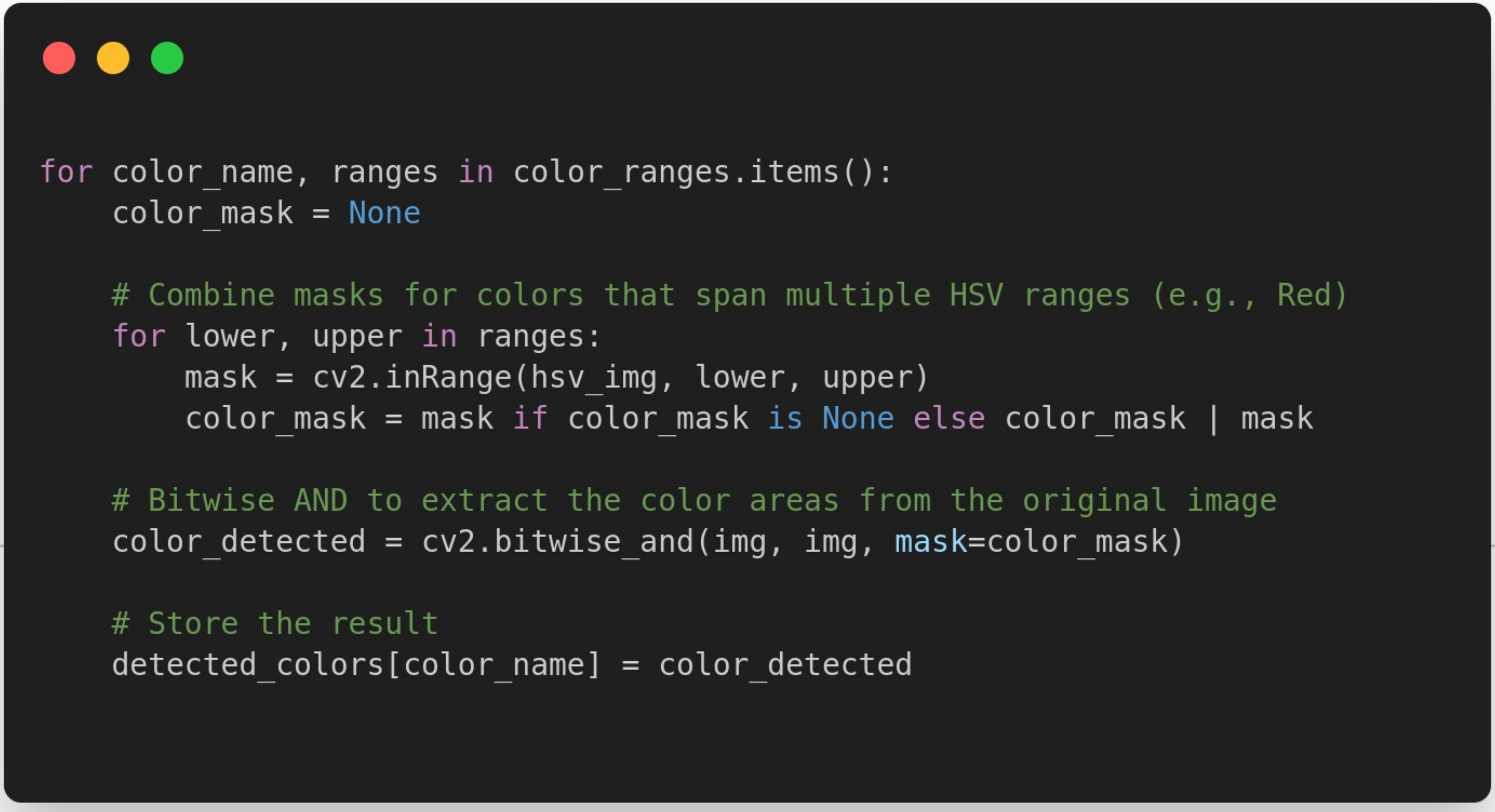
Display Results
Show the outcome of the color detection process

Step 6: Apply the Mask

- Use the mask to extract the colored regions from the original image:

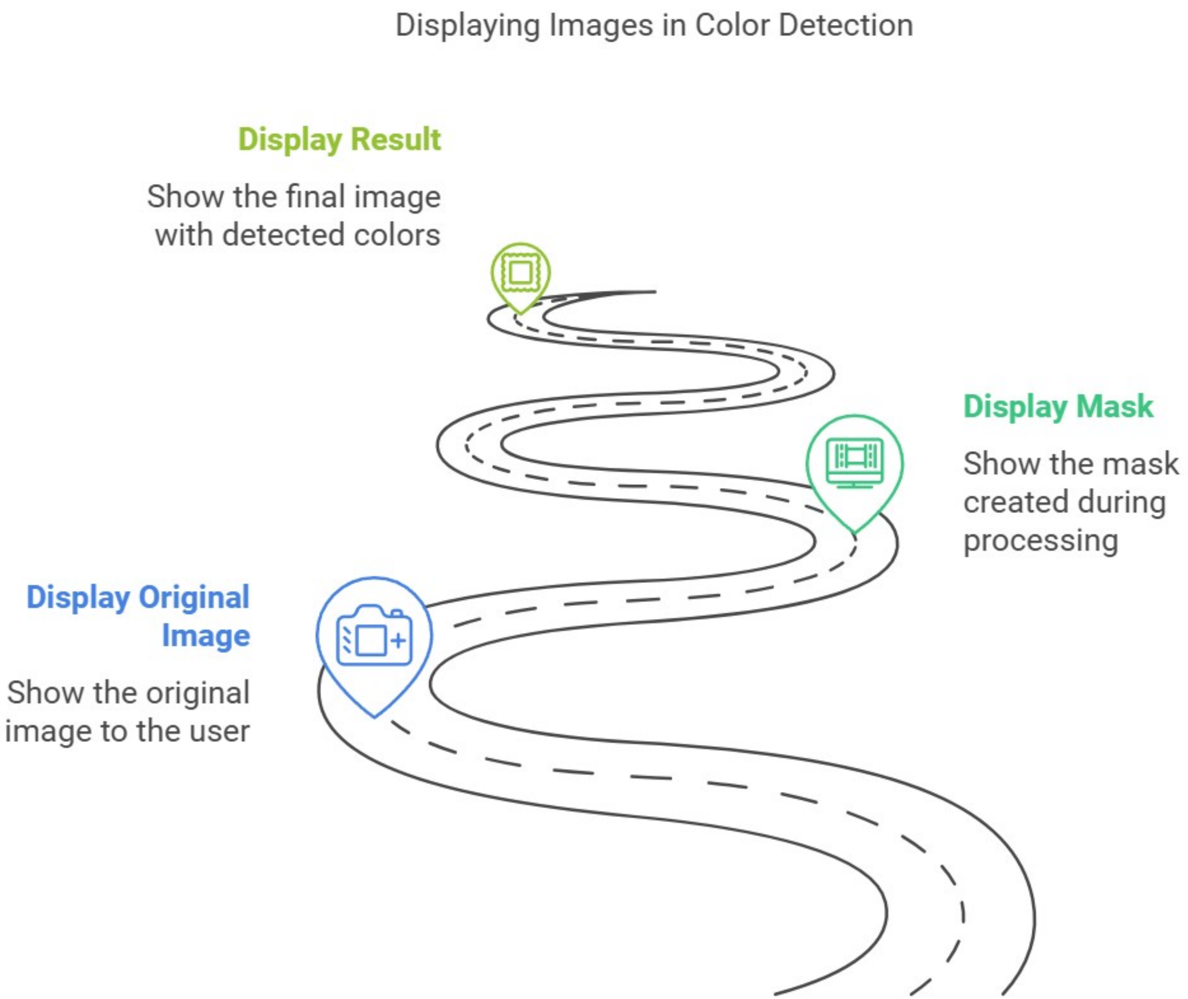


In Our Code :



```
for color_name, ranges in color_ranges.items():
    color_mask = None

    # Combine masks for colors that span multiple HSV ranges (e.g., Red)
    for lower, upper in ranges:
        mask = cv2.inRange(hsv_img, lower, upper)
        color_mask = mask if color_mask is None else color_mask | mask

    # Bitwise AND to extract the color areas from the original image
    color_detected = cv2.bitwise_and(img, img, mask=color_mask)

    # Store the result
    detected_colors[color_name] = color_detected
```

Step 7: Display the Results



Displaying Images in Color Detection

Display Result

Show the final image with detected colors

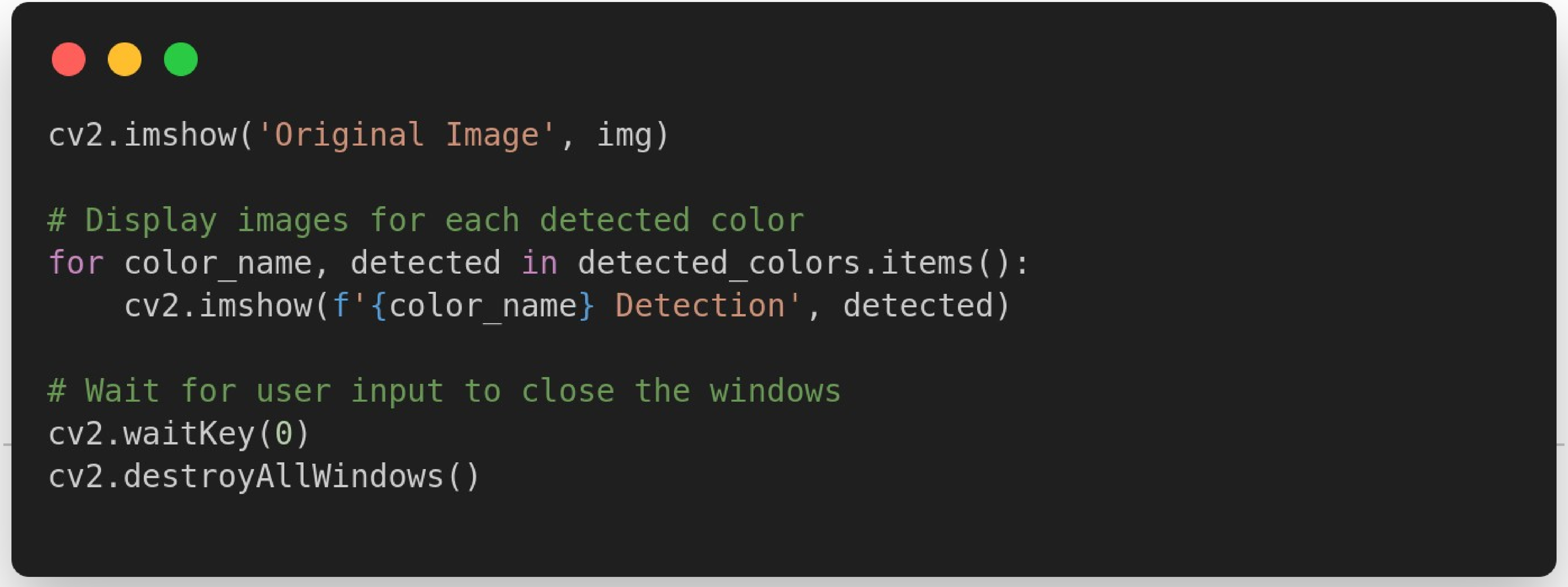
Display Mask

Show the mask created during processing

Display Original Image

Show the original image to the user

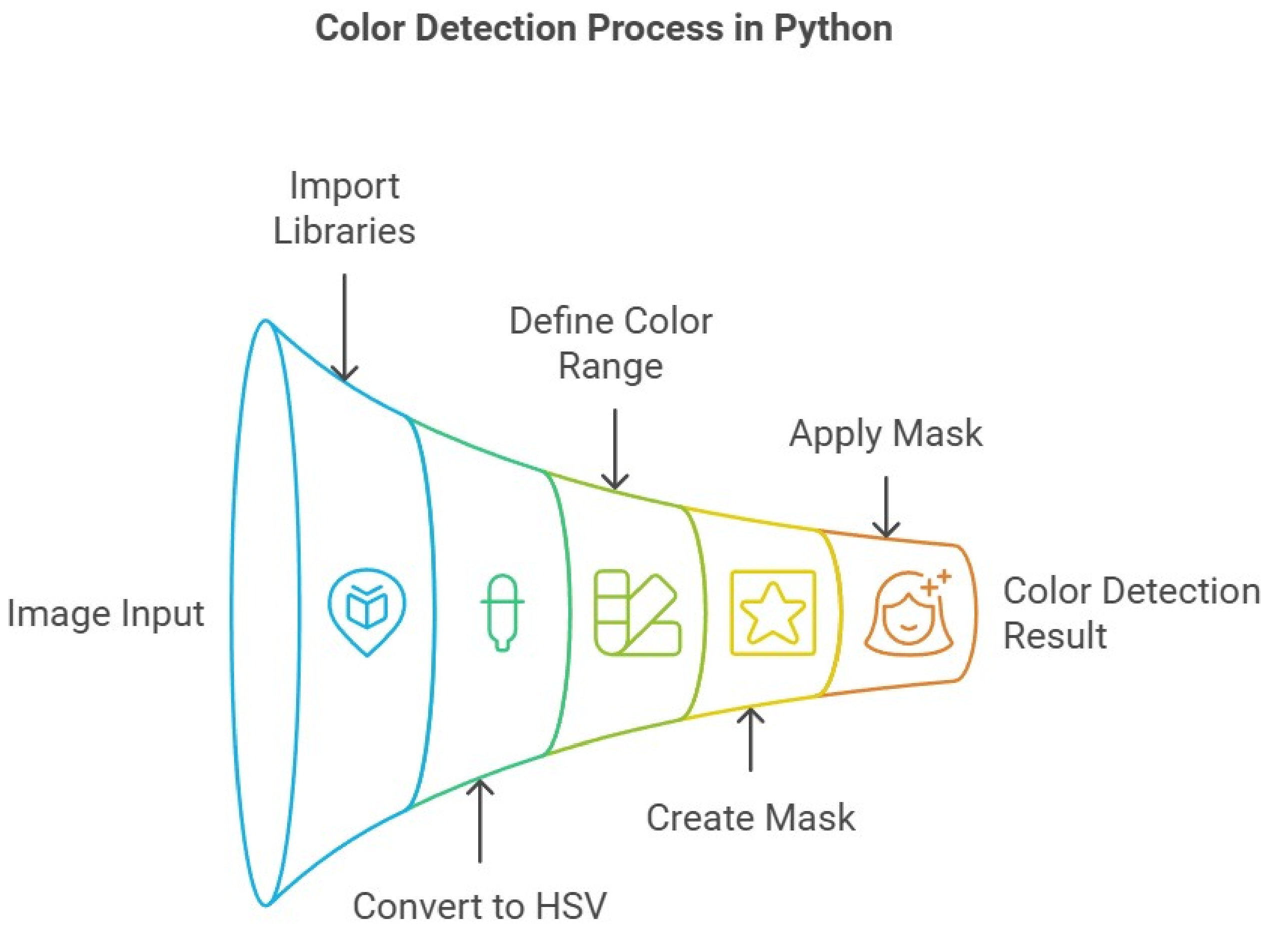
In Our Code :



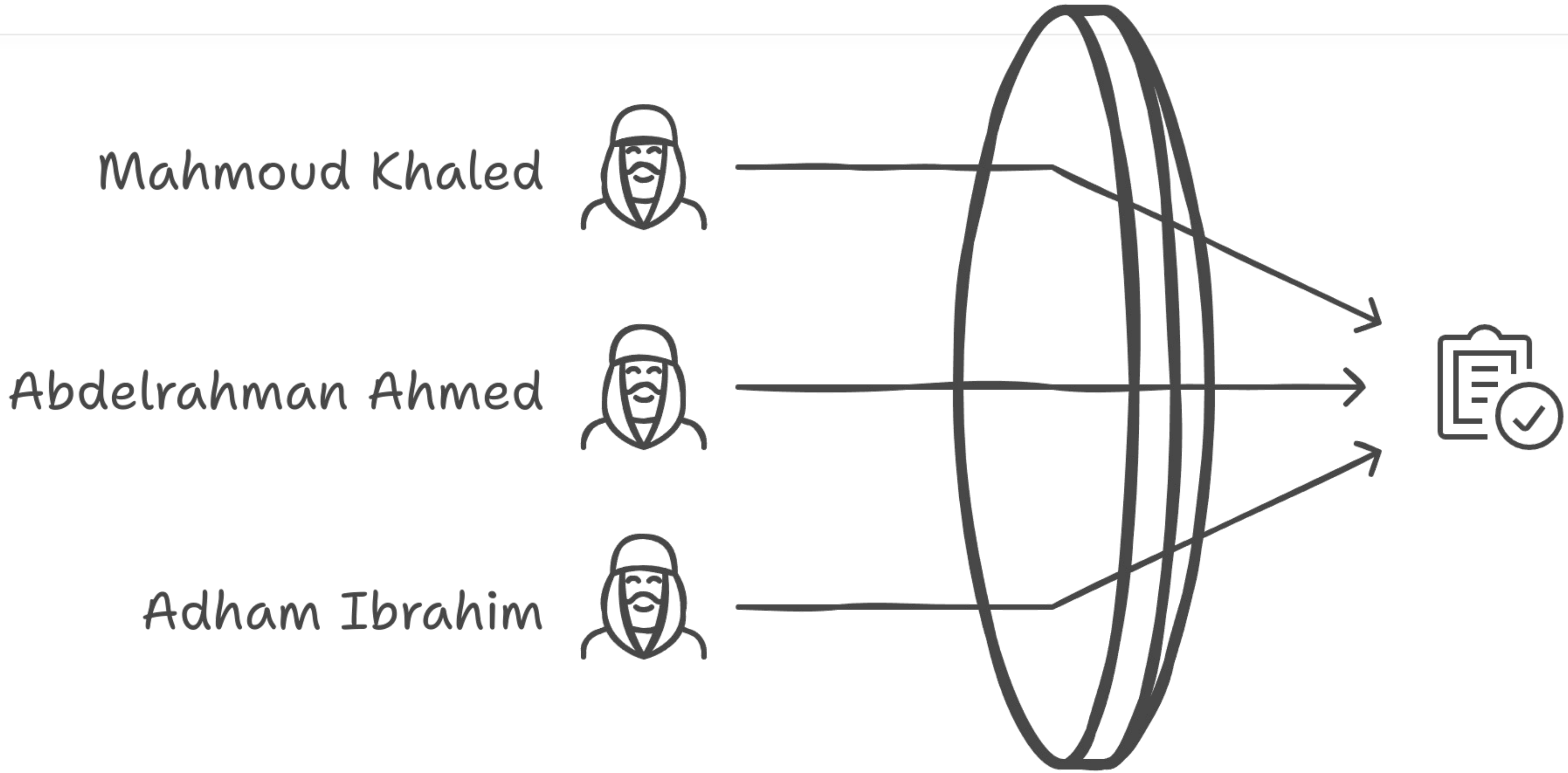
```python
cv2.imshow('Original Image', img)

# Display images for each detected color
for color_name, detected in detected_colors.items():
    cv2.imshow(f'{color_name} Detection', detected)

# Wait for user input to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Conclusion

- Color detection in Python using OpenCV is a straightforward process that can be adapted for various applications. By following the steps outlined in this document, you can easily detect specific colors in images and manipulate them for further analysis or processing. Experiment with different color ranges and images to enhance your understanding of color detection techniques.



Made By



COLOR DETECTION

THANKS