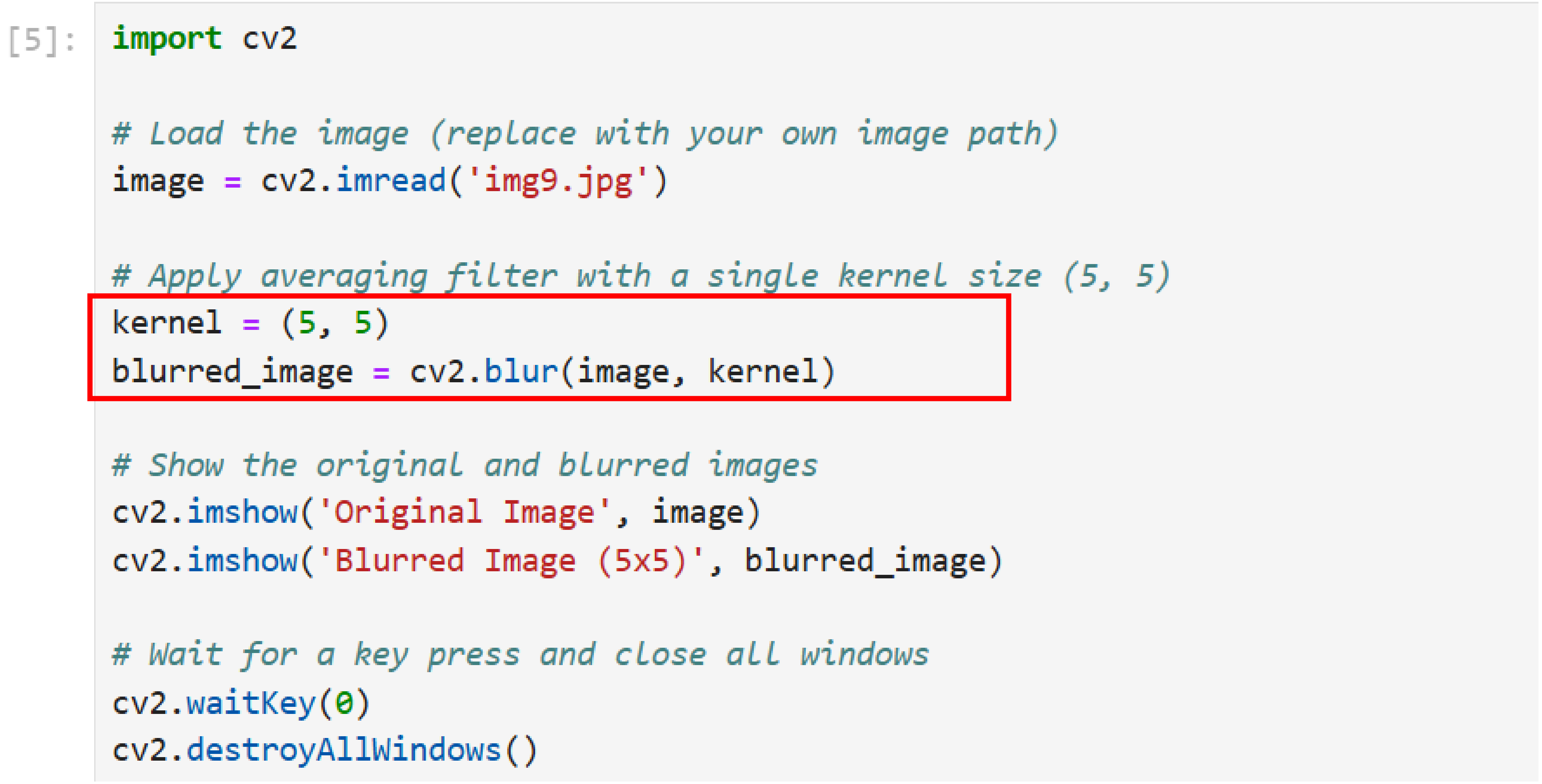
LAB: Digital Image Processing

Objective: To apply Spatial processing based filtering operations on digital image for blurring/sharpening using opencv library

Applying the Averaging Filter (Low-Pass Filter):

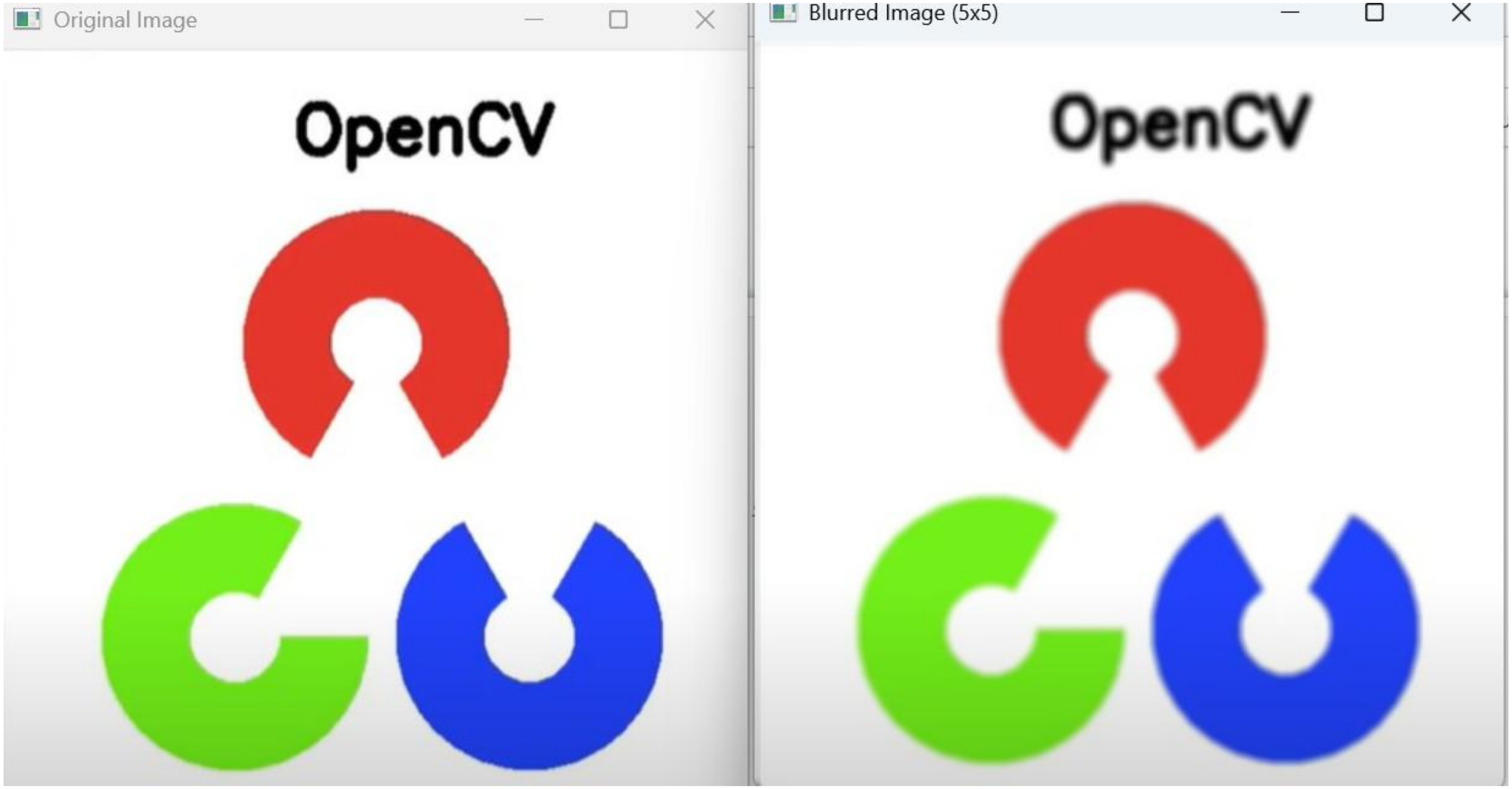


```
[5]: import cv2

     # Load the image (replace with your own image path)
     image = cv2.imread('img9.jpg')

     # Apply averaging filter with a single kernel size (5, 5)
     kernel = (5, 5)
     blurred_image = cv2.blur(image, kernel)

     # Show the original and blurred images
     cv2.imshow('Original Image', image)
     cv2.imshow('Blurred Image (5x5)', blurred_image)

     # Wait for a key press and close all windows
     cv2.waitKey(0)
     cv2.destroyAllWindows()
```

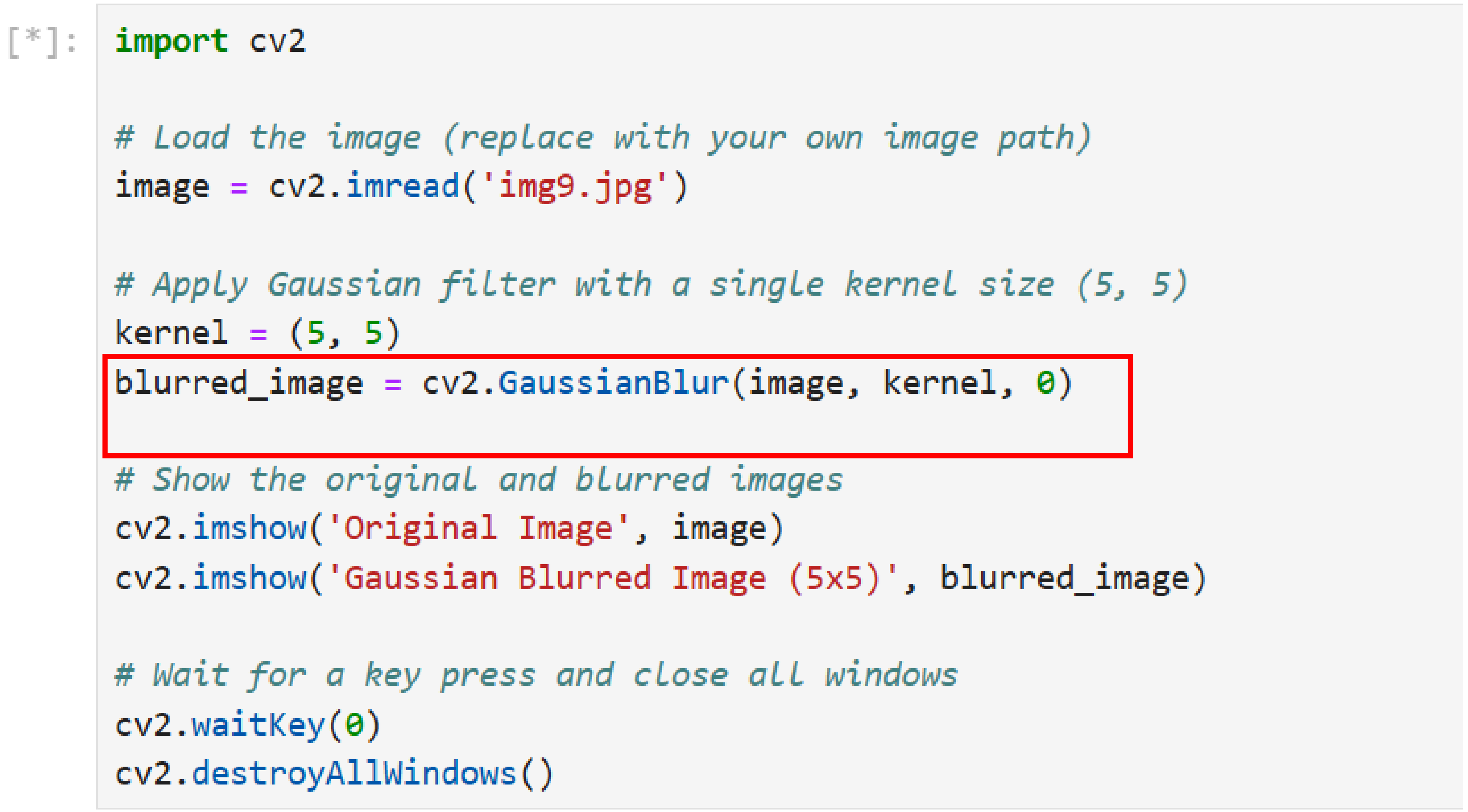
Applying the Averaging Filter (Low-Pass Filter):

- kernel = (5, 5)
- blurred_image = cv2.blur(image, kernel)

- **kernel = (5, 5)**: This defines the size of the kernel (a 5x5 matrix) used for the blurring operation. The kernel size determines how much of the image will be averaged over each pixel. A larger kernel size results in stronger blurring.
- **cv2.blur(image, kernel)**: This applies the **averaging filter** (box blur) to the image. It uses the kernel defined earlier to average the pixel values in a local neighborhood. The function produces a new image (blurred_image) where high-frequency details are reduced (the image becomes smoother).


Original Image
OpenCV
Blurred Image (5x5)
OpenCV

Applying the Gaussian Filter (Low-Pass Filter):



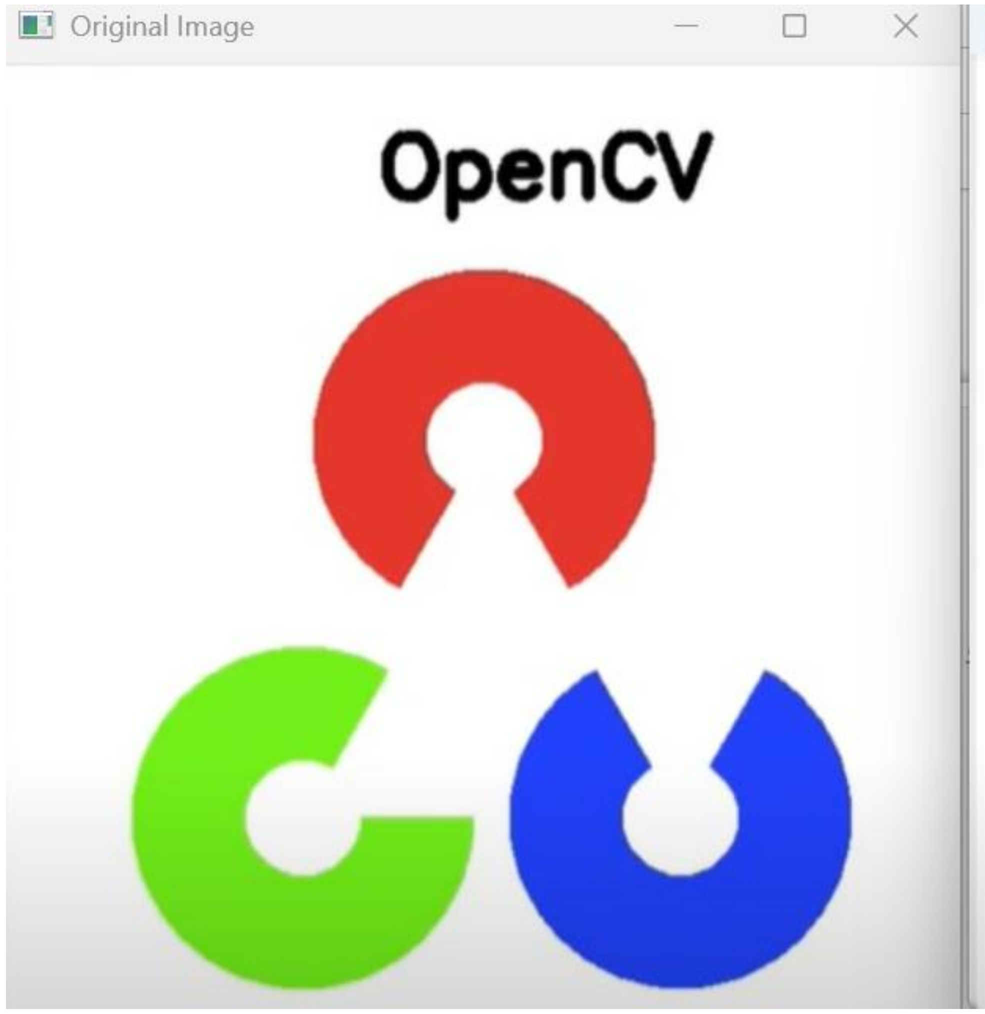
[*]:

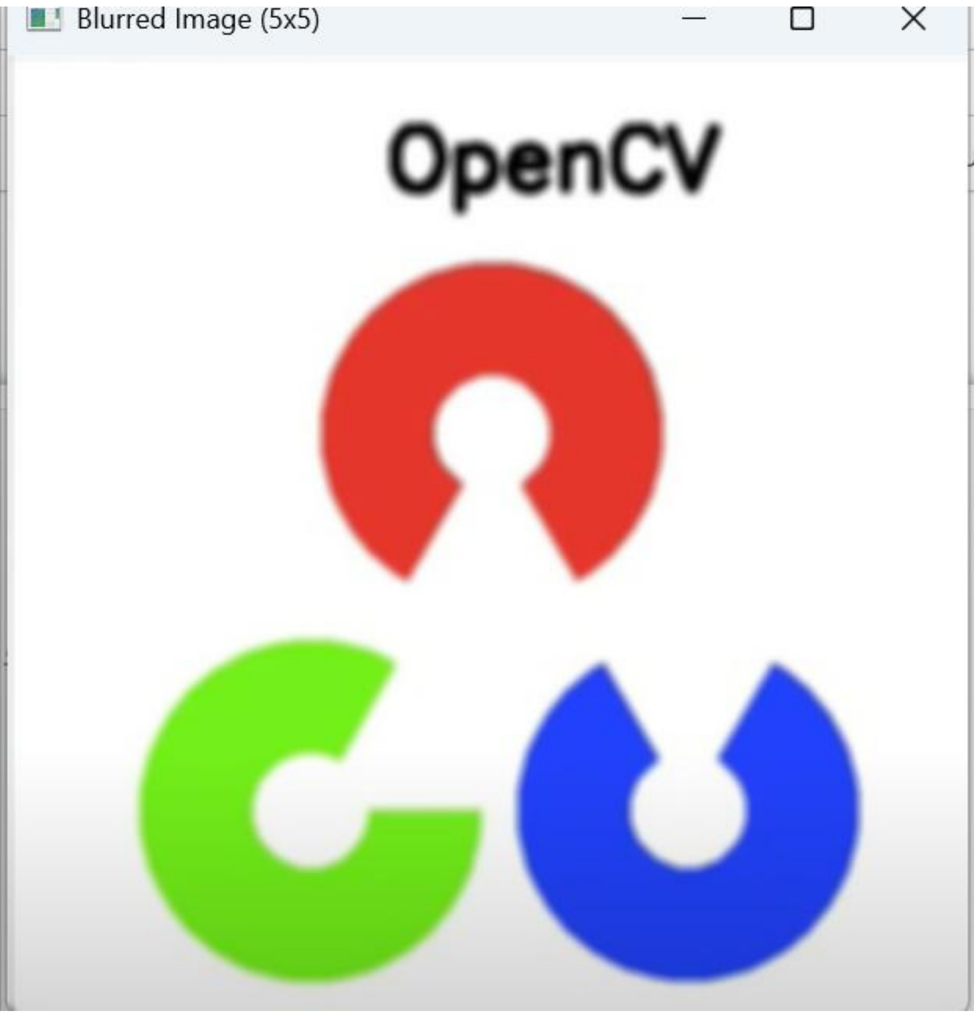
```
import cv2

# Load the image (replace with your own image path)
image = cv2.imread('img9.jpg')

# Apply Gaussian filter with a single kernel size (5, 5)
kernel = (5, 5)
blurred_image = cv2.GaussianBlur(image, kernel, 0)

# Show the original and blurred images
cv2.imshow('Original Image', image)
cv2.imshow('Gaussian Blurred Image (5x5)', blurred_image)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Applying the Gaussian Filter (Low-Pass Filter):

- Notes

- cv2.GaussianBlur(image, kernel, 0) → 0 is the standard deviation in X direction (auto-calculated if set to 0).
- You can change kernel size (3,3), (7,7), (9,9) to observe different levels of blurring.


Original Image
OpenCV


Blurred Image (5x5)
OpenCV

Task : Change the Kernel Size

Objective: Modify the kernel size from 5x5 to other values (e.g., 3x3, 7x7, 9x9).

Instructions:

- Adjust the size of the kernel and observe how the blurring effect changes.
- Compare the results with different kernel sizes (e.g., 3x3, 7x7, 9x9).

Task: Experiment with Different Image Types

Objective: Use different images (e.g., grayscale, color, or noisy images).

Instructions:

- Load different types of images, such as grayscale or images with noise, and apply the blurring effect.
- Observe how blurring affects images differently based on the type (grayscale vs color).

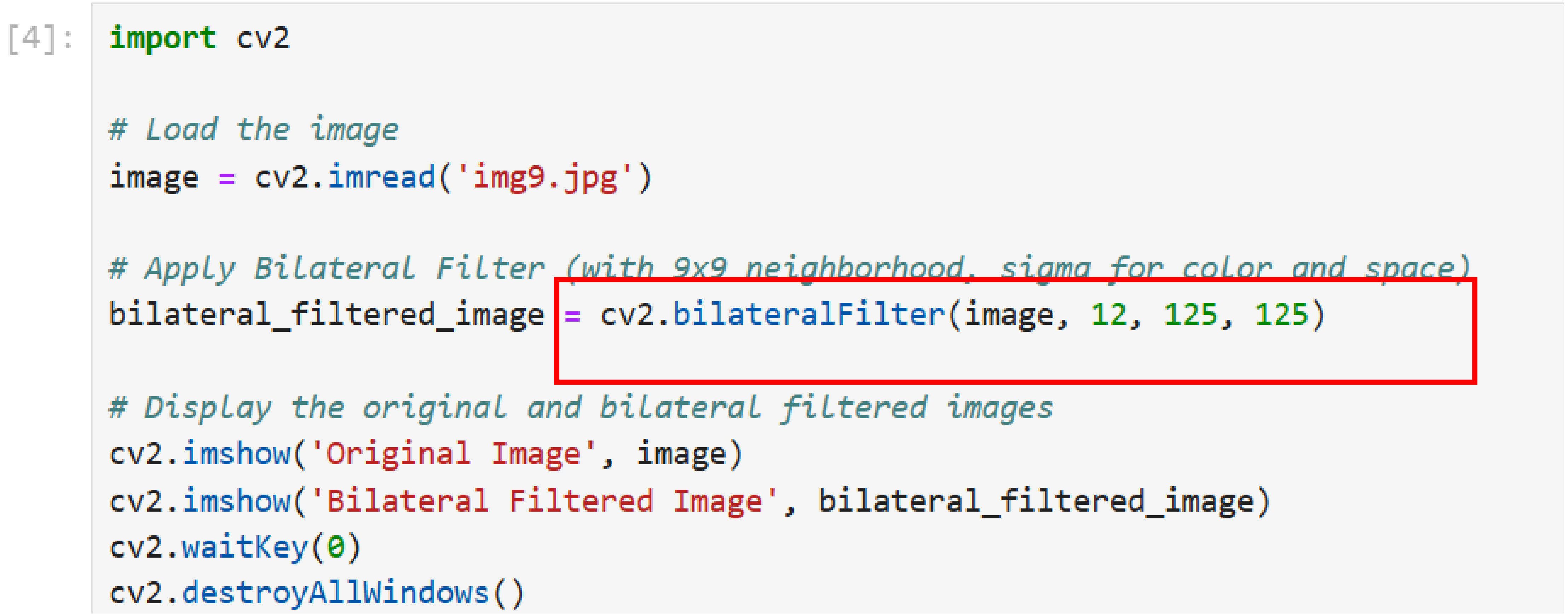
Task: Change the Image Format

- Objective: Convert an image from one format (e.g., JPG) into other formats such as PNG, BMP, and TIFF.

Instructions:

- Load an image in JPG format.
- Save and convert the image into PNG, BMP, and TIFF formats.
- Compare the results in terms of file size, compression, and image quality.

Applying the Bilateral Filter:



```
[4]: import cv2

# Load the image
image = cv2.imread('img9.jpg')

# Apply Bilateral Filter (with 9x9 neighborhood, sigma for color and space)
bilateral_filtered_image = cv2.bilateralFilter(image, 12, 125, 125)

# Display the original and bilateral filtered images
cv2.imshow('Original Image', image)
cv2.imshow('Bilateral Filtered Image', bilateral_filtered_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Task : use different neighborhood, sigma color, space value to compare results

Applying the Bilateral Filter:

- cv2.bilateralFilter() is an advanced image filtering function that applies the bilateral filter to the image.
- It preserves edges while smoothing the image (unlike Gaussian blur, which also smooths edges).The function takes four parameters.

image: The input image to which the filter will be applied.

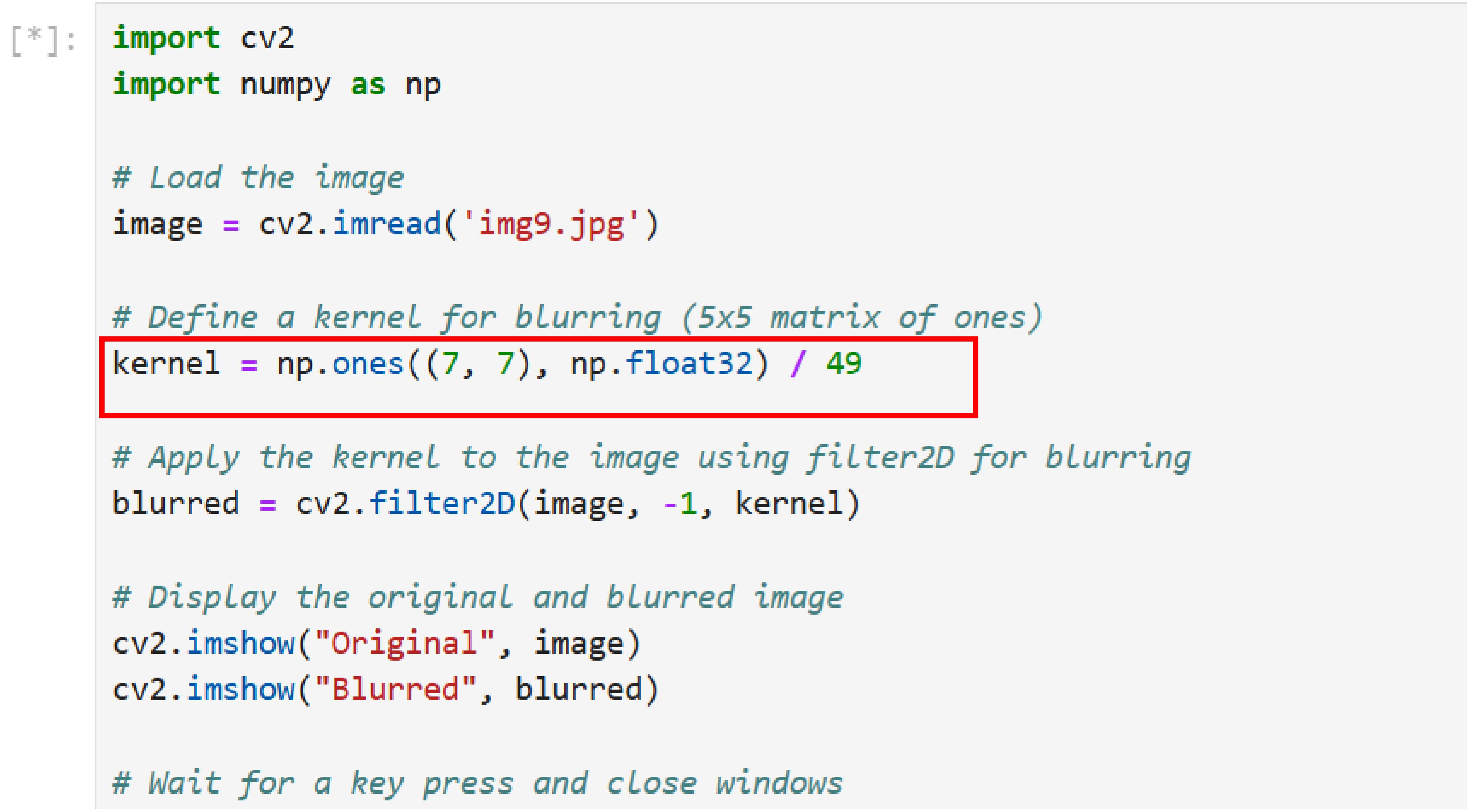
12: This is the diameter of the pixel neighborhood that will be considered for filtering. A larger value will result in more smoothing, while a smaller value keeps more details.

125 (sigmaColor): This controls how much the filter considers pixel color differences. Larger values allow for more color variations, which means more smoothing. Smaller values preserve more color details.

125 (sigmaSpace): This controls how much the filter considers the spatial distance between pixels. Larger values allow for more distant pixels to be used in the calculation, resulting in a more significant blur effect.).

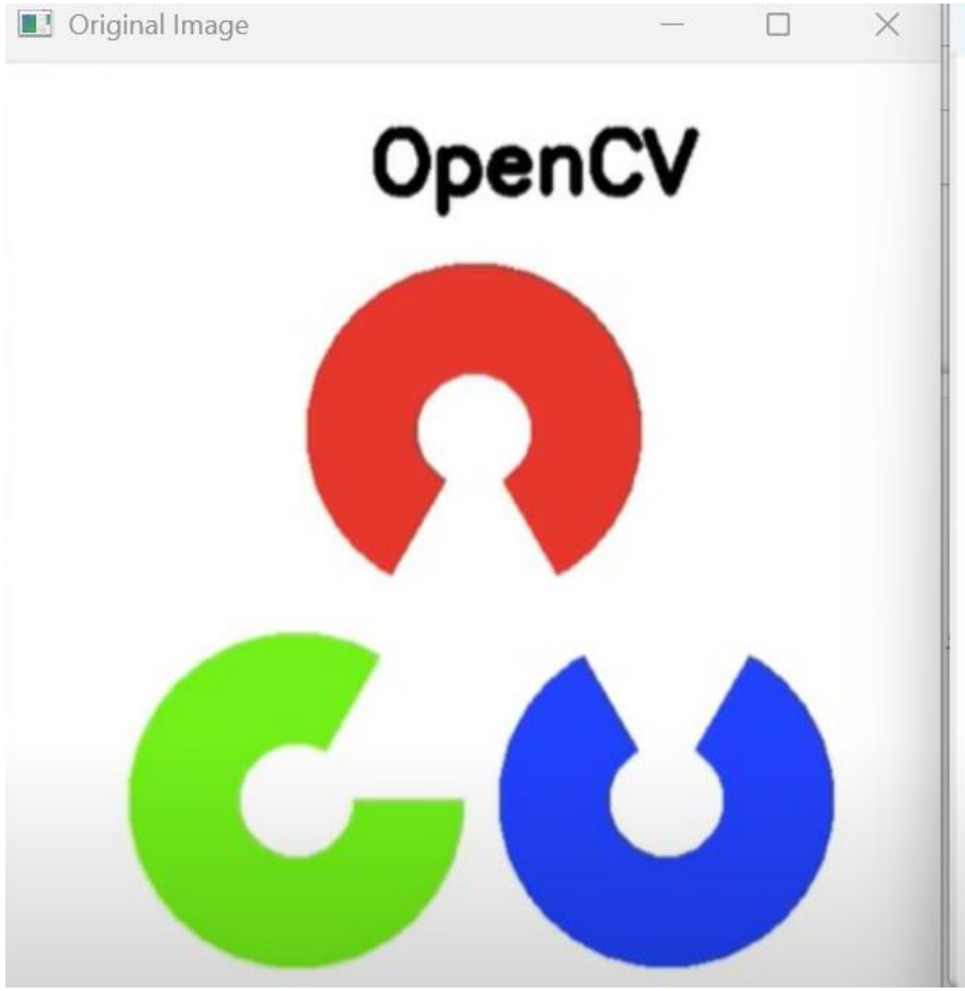
Task : change parameters and observe the results

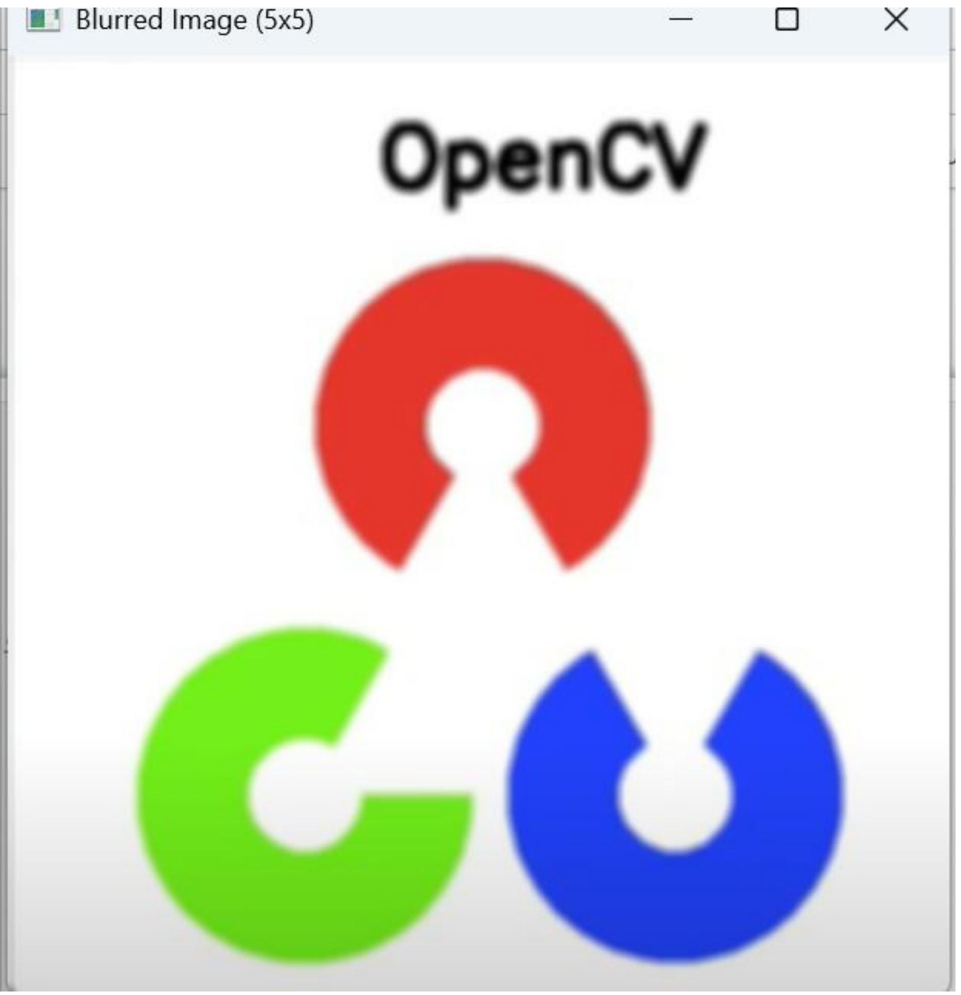
Apply low pass-filters using numpy:



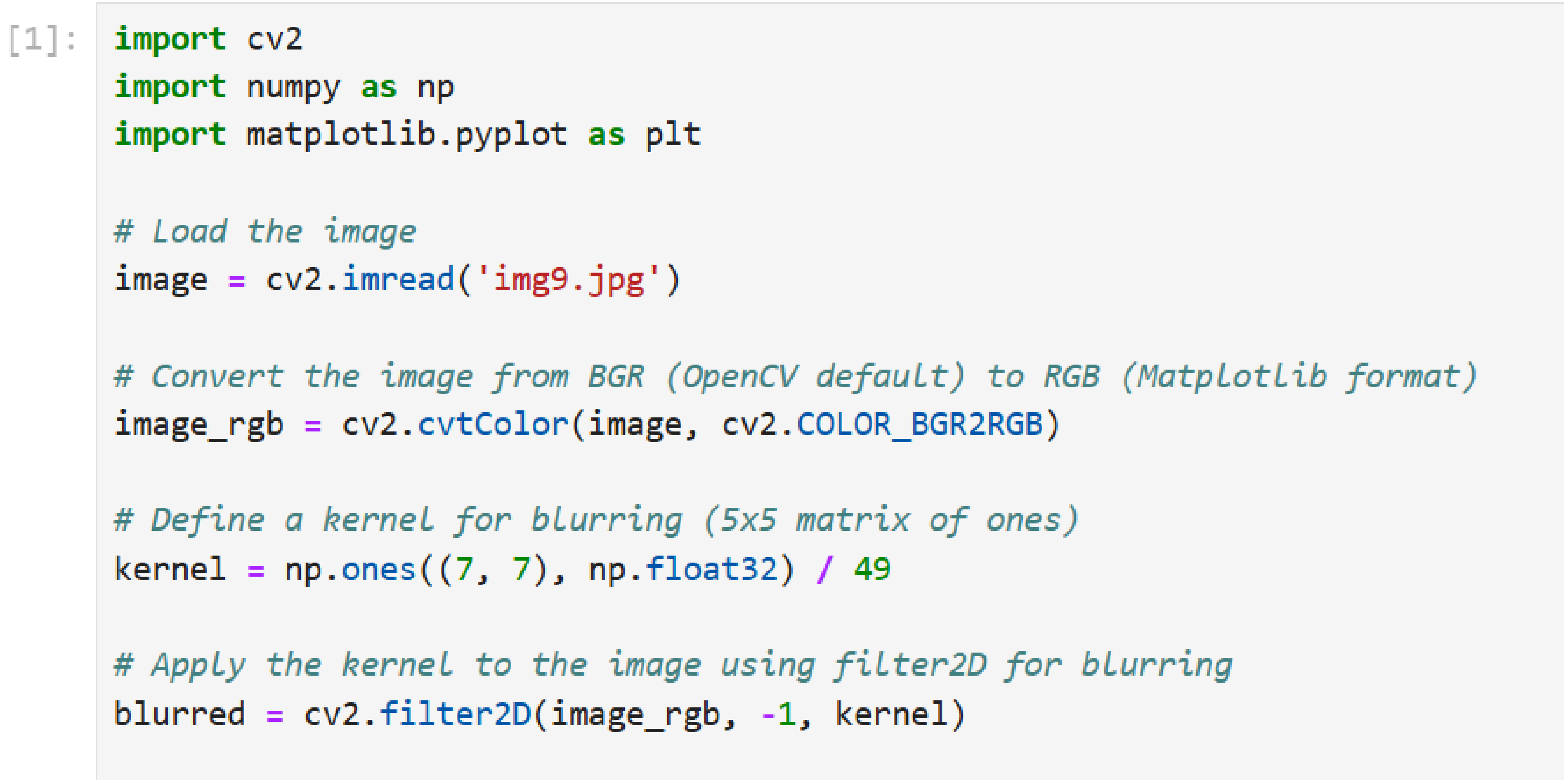
```
[*]: import cv2
     import numpy as np

     # Load the image
     image = cv2.imread('img9.jpg')

     # Define a kernel for blurring (5x5 matrix of ones)
     kernel = np.ones((7, 7), np.float32) / 49

     # Apply the kernel to the image using filter2D for blurring
     blurred = cv2.filter2D(image, -1, kernel)

     # Display the original and blurred image
     cv2.imshow("Original", image)
     cv2.imshow("Blurred", blurred)

     # Wait for a key press and close windows
```

- np.ones((7, 7)) creates a 7x7 matrix where each element is 1.
- This matrix is then divided by 49 (np.float32 / 49), which is the total sum of the kernel (7x7 matrix of 1s). This creates a kernel where each element is 1/49, which averages out pixel values in a 7x7 region, effectively blurring the image.

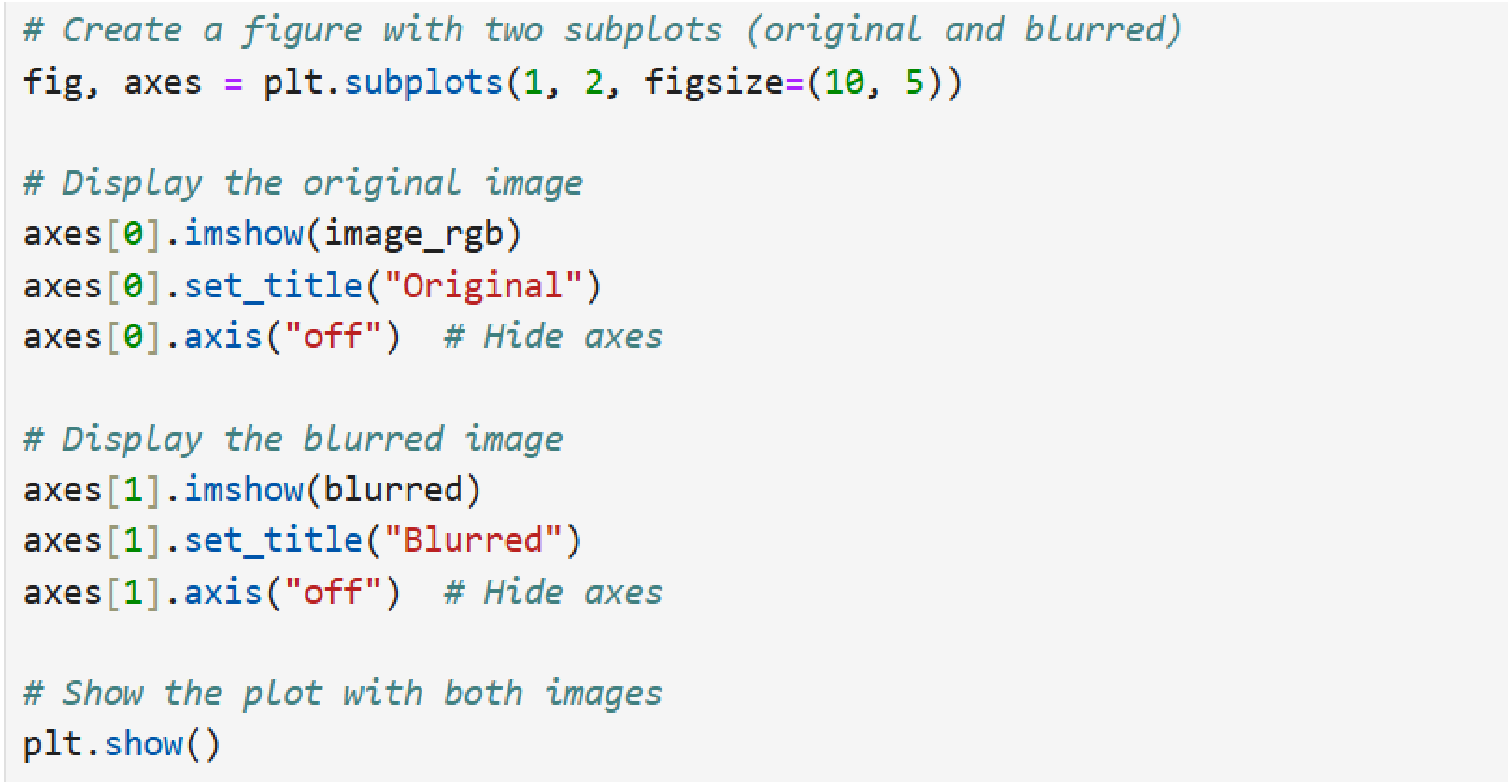

Original Image
OpenCV


Blurred Image (5x5)
OpenCV

Apply low pass-filters using matplotlib:



```
[1]:
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('img9.jpg')

# Convert the image from BGR (OpenCV default) to RGB (Matplotlib format)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Define a kernel for blurring (5x5 matrix of ones)
kernel = np.ones((7, 7), np.float32) / 49

# Apply the kernel to the image using filter2D for blurring
blurred = cv2.filter2D(image_rgb, -1, kernel)
```



```python
# Create a figure with two subplots (original and blurred)
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Display the original image
axes[0].imshow(image_rgb)
axes[0].set_title("Original")
axes[0].axis("off")  # Hide axes

# Display the blurred image
axes[1].imshow(blurred)
axes[1].set_title("Blurred")
axes[1].axis("off")  # Hide axes

# Show the plot with both images
plt.show()
```

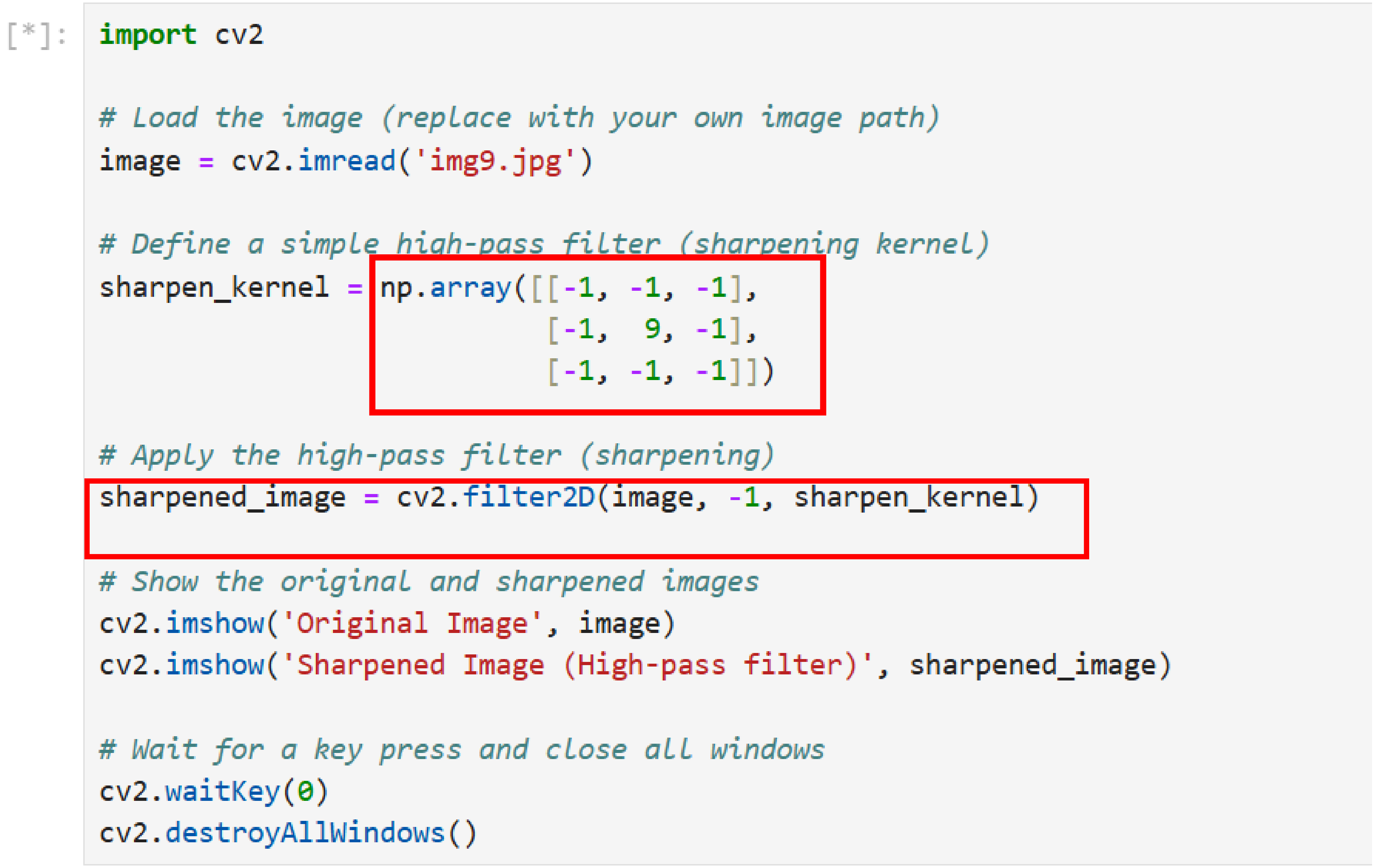
Task: Adjust the Normalization of the Kernel

Objective: Experiment with different normalization techniques, such as dividing by other values like 10, 100, or even 255.

Instructions:

- Change the kernel normalization value and observe how it affects the blurring strength.
- Try using larger values like 100 or smaller ones like 10.

Applying High-Pass Filter:



```
[*]: import cv2

     # Load the image (replace with your own image path)
     image = cv2.imread('img9.jpg')

     # Define a simple high-pass filter (sharpening kernel)
     sharpen_kernel = np.array([[-1, -1, -1],
                                [-1,  9, -1],
                                [-1, -1, -1]])

     # Apply the high-pass filter (sharpening)
     sharpened_image = cv2.filter2D(image, -1, sharpen_kernel)

     # Show the original and sharpened images
     cv2.imshow('Original Image', image)
     cv2.imshow('Sharpened Image (High-pass filter)', sharpened_image)

     # Wait for a key press and close all windows
     cv2.waitKey(0)
     cv2.destroyAllWindows()
```

Applying High-Pass Filter:

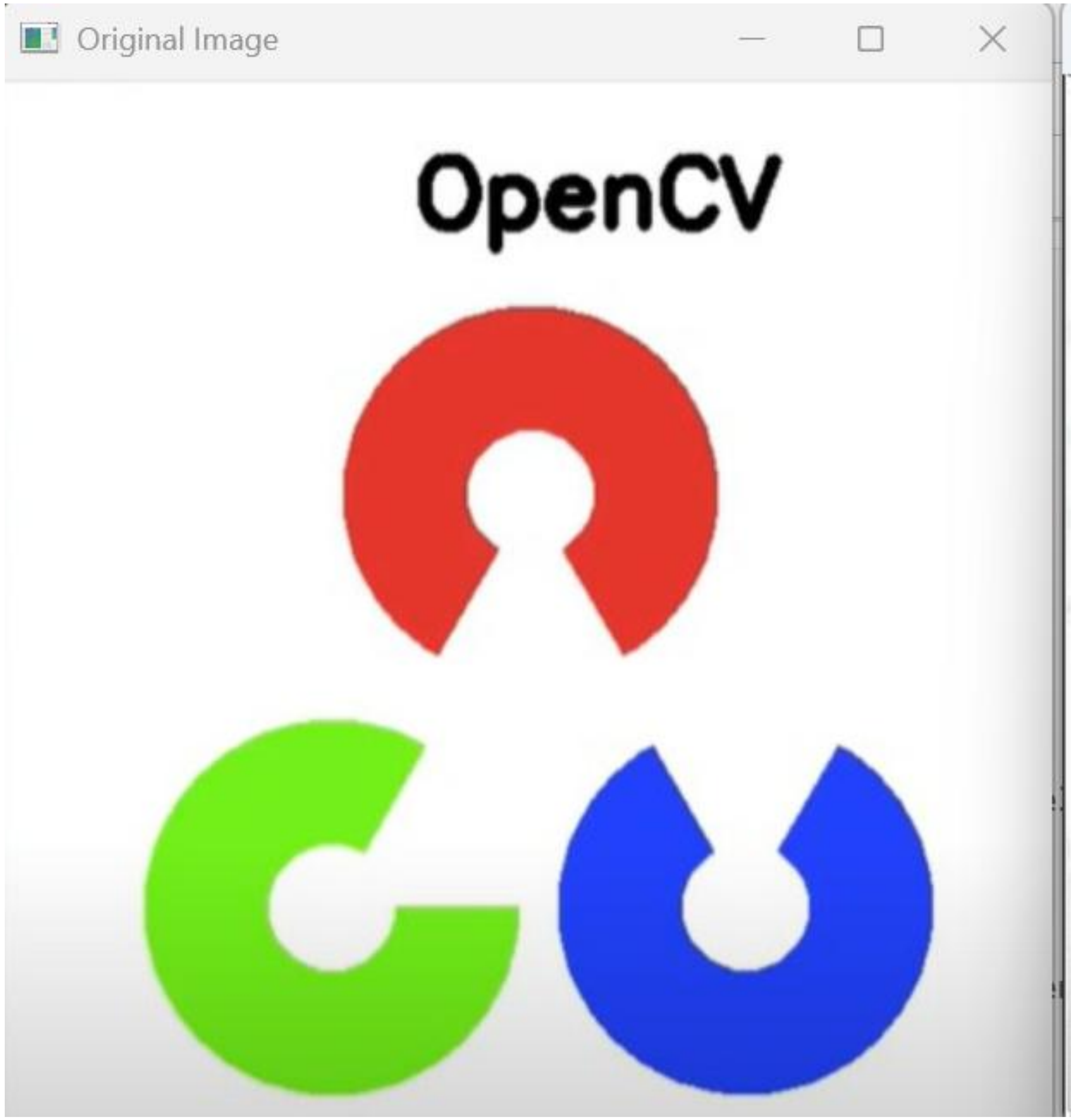
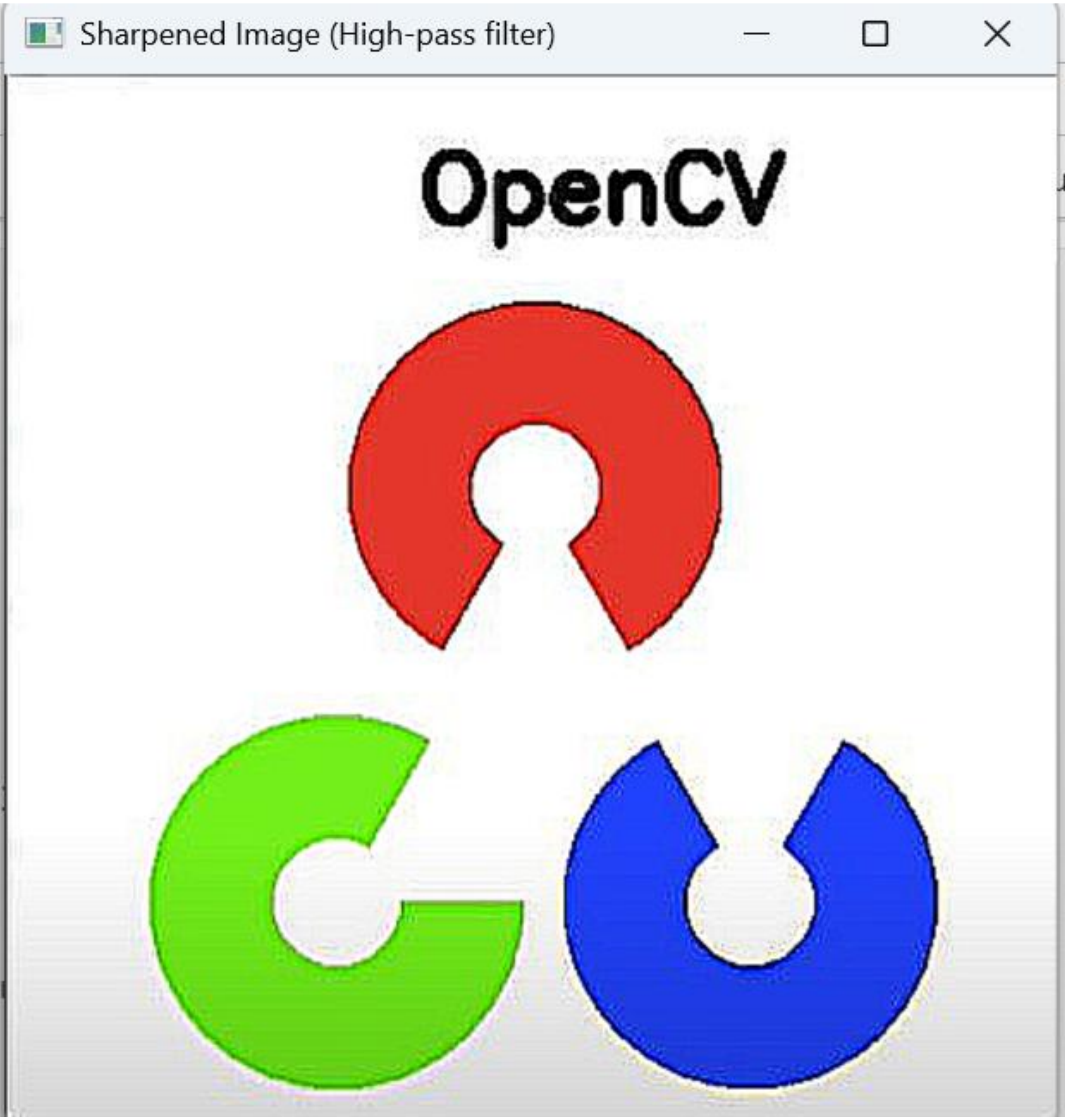
```
sharpen_kernel = np.array([[-1, -1, -1],
                           [-1,  9, -1],
                           [-1, -1, -1]])
```

np.array() creates a NumPy array that defines a sharpening kernel (a 3x3 matrix).

This kernel is a **high-pass filter** that is used to sharpen an image. The center value (9) amplifies the central pixel, and the surrounding values (-1) subtract from the neighboring pixels.

High-pass filters enhance the edges in the image, making them appear more defined by emphasizing sharp transitions

.


Original Image
OpenCV

Sharpened Image (High-pass filter)
OpenCV

Task: Apply different kernels matrix sizes/ parameters and observe the effects