```
#**FUTURESKILLS AI BOOTCAMP ASSIGNMENT 4**

---



**Problem Statement:**
Perform basic image processing tasks such as loading, displaying,
resizing, converting color spaces, and applying filters using OpenCV.
```

FUTURESKILLS AI BOOTCAMP ASSIGNMENT 4

Problem Statement:
Perform basic image processing tasks such as loading, displaying, resizing, converting color spaces, and applying filters using OpenCV.

We need to install OpenCV (if not already installed) and import the required libraries.

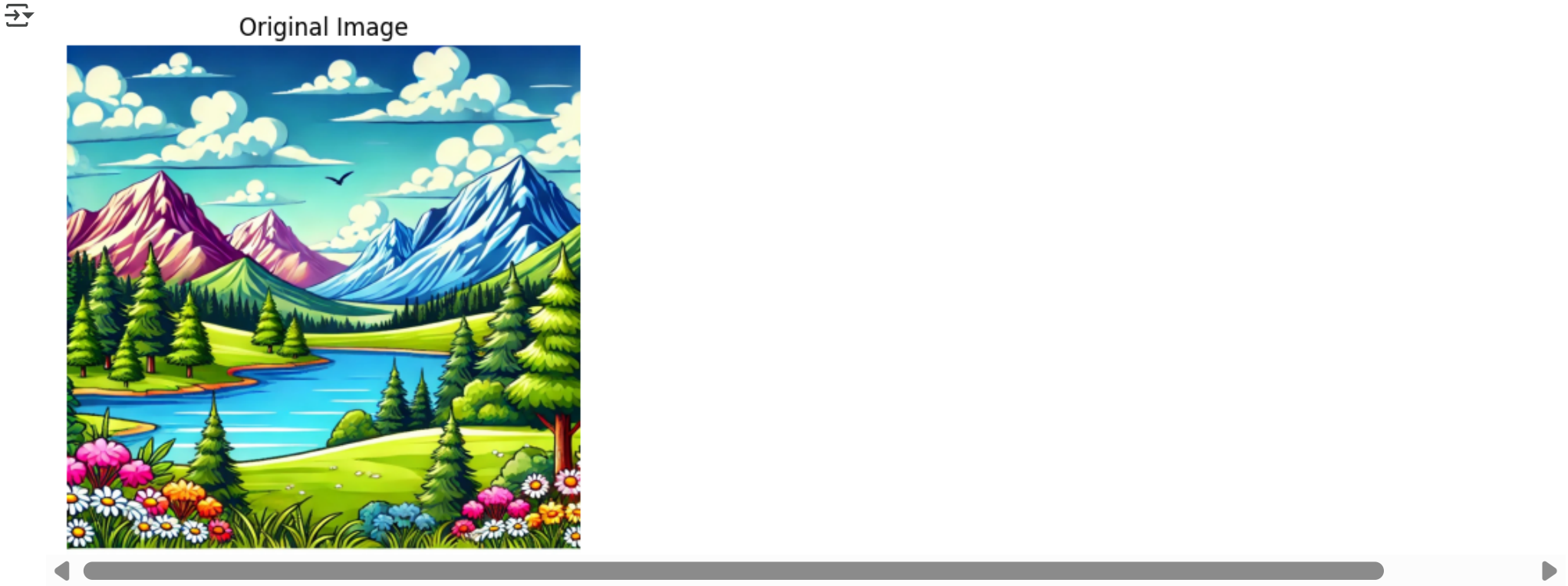
```
# Install OpenCV
!pip install opencv-python-headless

# Import necessary libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python-headless) (2.0.2)
```

We first load the image using OpenCV and display it.

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

# Convert BGR to RGB (as OpenCV loads in BGR format)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image
```

```
plt.imshow(image_rgb)
plt.axis('off')
plt.title("Original Image")
plt.show()
```

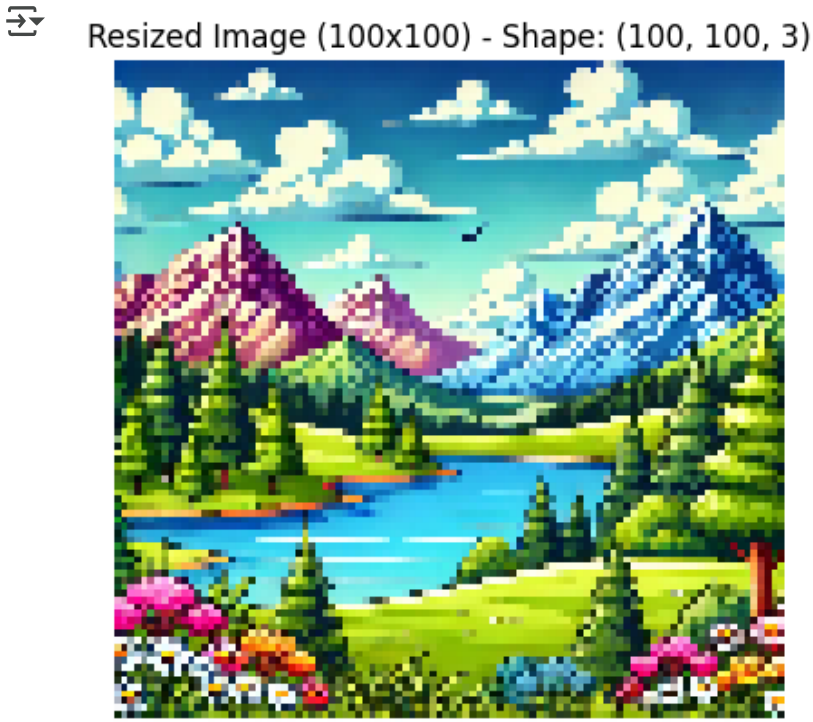


Grayscale images simplify processing by reducing color information.

```
# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display grayscale image
plt.imshow(gray_image, cmap='gray')
plt.axis('off')
plt.title("Grayscale Image")
plt.show()
```

Resizing helps adjust the image dimensions.

```
# Resize the image to 100x100
resized_image = cv2.resize(image, (100, 100))

# Convert BGR to RGB for displaying
resized_rgb = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)

# Display resized image with shape
plt.imshow(resized_rgb)
plt.axis('off')
plt.title(f"Resized Image (100x100) - Shape: {resized_image.shape}")
plt.show()
```

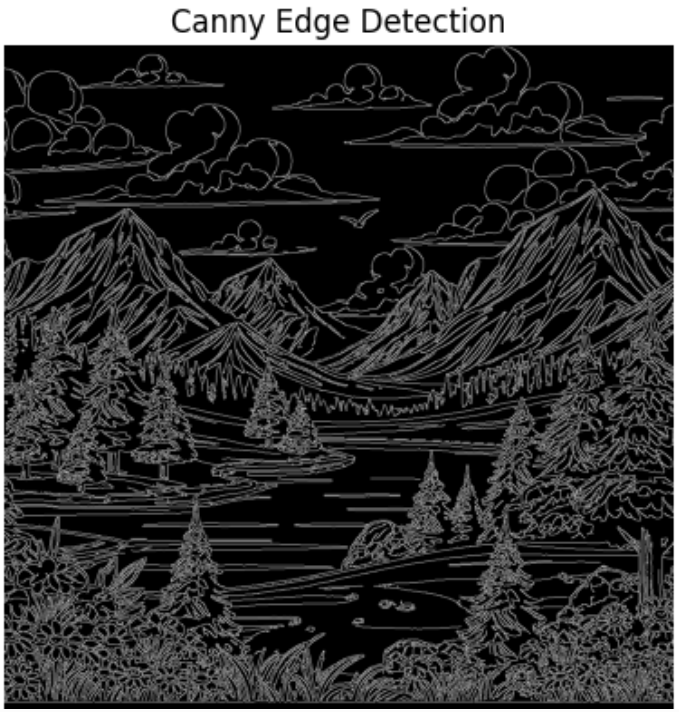


Gaussian Blur smooths the image and reduces noise.

```
# Apply strong Gaussian Blur with a 15x15 kernel
blurred_image = cv2.GaussianBlur(image, (15,15), 0)

# Convert to RGB for displaying
blurred_rgb = cv2.cvtColor(blurred_image, cv2.COLOR_BGR2RGB)

# Display blurred image
plt.imshow(blurred_rgb)
plt.axis('off')
plt.title("Gaussian Blurred Image (15x15 Kernel)")
plt.show()
```

Canny edge detection highlights the edges in the image.

```
# Apply Canny edge detection
edges = cv2.Canny(gray_image, 50, 150)

# Display edges
plt.imshow(edges, cmap='gray')
plt.axis('off')
plt.title("Canny Edge Detection")
plt.show()
```



We will draw a red rectangle and a green circle on the image.

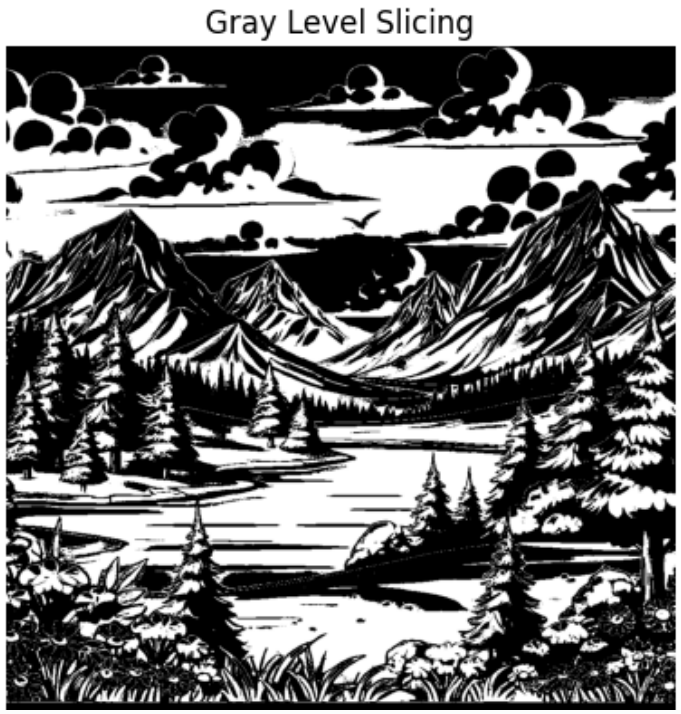
```
# Create a copy of the image to draw shapes on
image_shapes = image.copy()

# Draw a red rectangle (x1, y1, x2, y2)
cv2.rectangle(image_shapes, (50, 50), (250, 250), (0, 0, 255), 3)

# Draw a green circle (center_x, center_y, radius)
cv2.circle(image_shapes, (150, 150), 50, (0, 255, 0), -1)

# Convert to RGB for displaying
shapes_rgb = cv2.cvtColor(image_shapes, cv2.COLOR_BGR2RGB)

# Display the image with shapes
plt.imshow(shapes_rgb)
plt.axis('off')
```

```
plt.title("Image with Shapes")
plt.show()
```

Gray Level Slicing enhances specific intensity ranges.

```
# Define slicing range
min_gray, max_gray = 100, 200

# Create a mask and apply slicing
gray_sliced = np.where((gray_image > min_gray) & (gray_image < max_gray), 255, 0).astype(np.uint8)

# Display gray level slicing result
plt.imshow(gray_sliced, cmap='gray')
plt.axis('off')
plt.title("Gray Level Slicing")
```

```
plt.show()
```



We save the processed images for future use.

```
# Save the processed images
cv2.imwrite('Grayscale_Image.jpeg', gray_image)
cv2.imwrite('Resized_Image.jpeg', resized_image)
cv2.imwrite('Blurred_Image.jpeg', blurred_image)
cv2.imwrite('Canny_Edges.jpeg', edges)
cv2.imwrite('Image_with_Shapes.jpeg', image_shapes)
cv2.imwrite('Gray_Level_Slicing.jpeg', gray_sliced)

print("All processed images have been saved successfully.")
```

```
All processed images have been saved successfully.
```

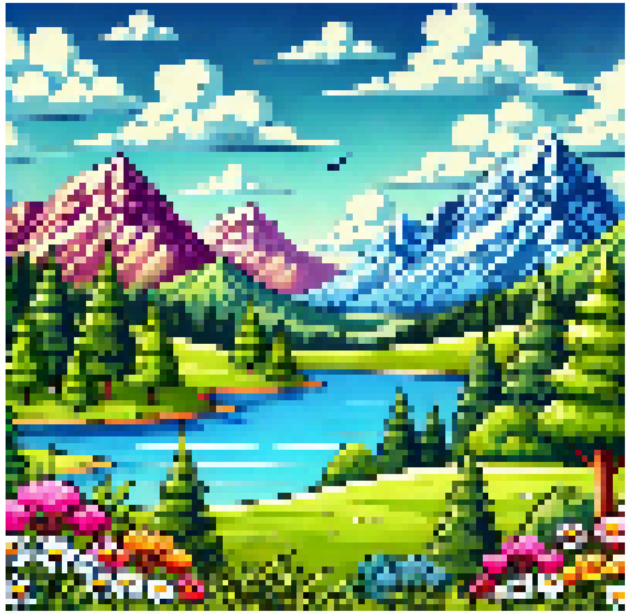
To visualize all processed images at once.

```
# Create a figure to display multiple images
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

# List of images and their titles
images = [image_rgb, gray_image, resized_rgb, blurred_rgb, edges, shapes_rgb]
titles = ["Original Image", "Grayscale", "Resized", "Blurred", "Canny Edges", "Shapes"]

# Loop through and display images
for ax, img, title in zip(axes.flatten(), images, titles):
    ax.imshow(img if len(img.shape) == 3 else img, cmap='gray' if len(img.shape) == 2 else None)
    ax.set_title(title)
    ax.axis('off')

plt.tight_layout()
plt.show()
```

Original Image

Grayscale

Resized



Blurred

Canny Edges

Shapes