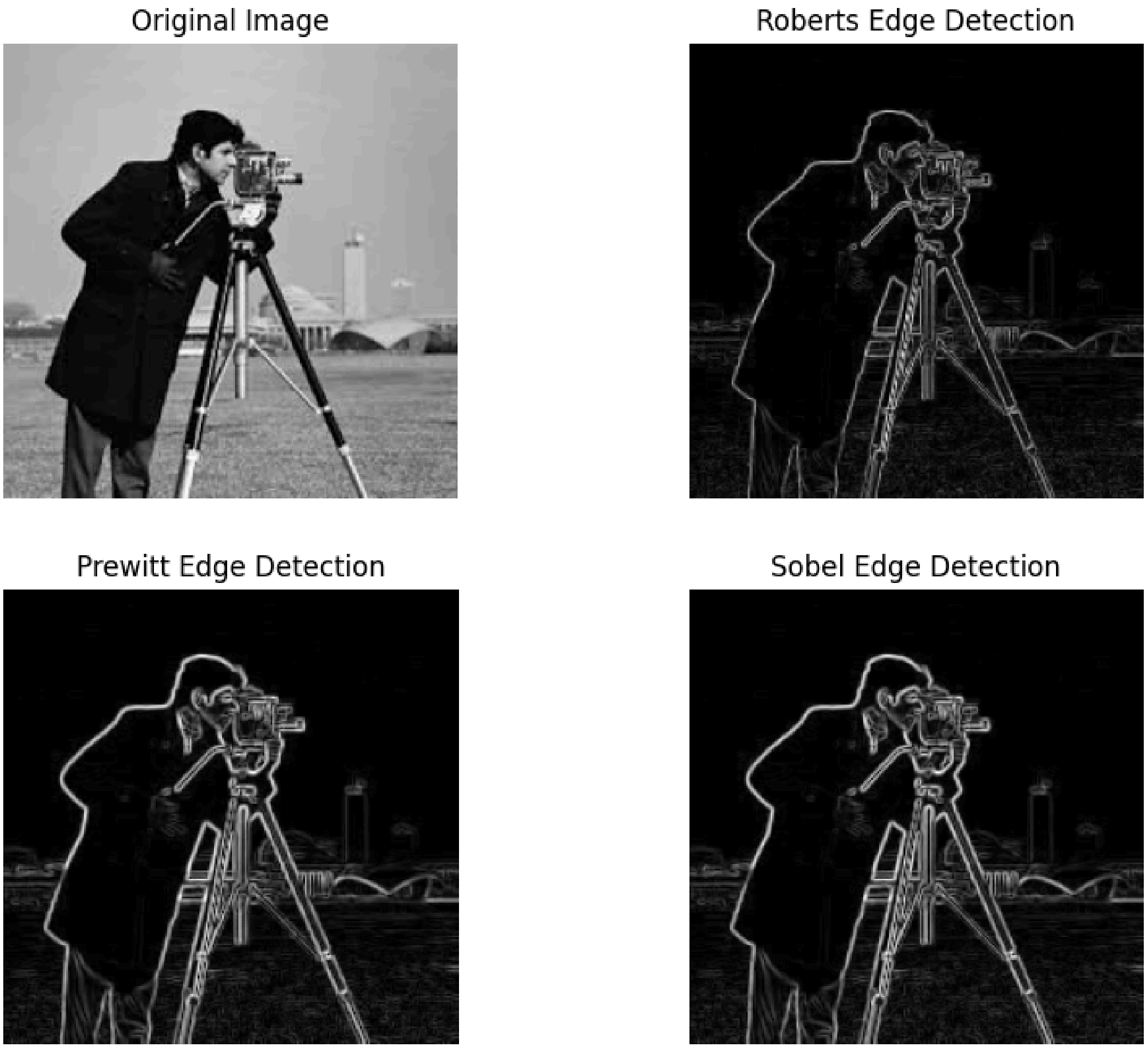
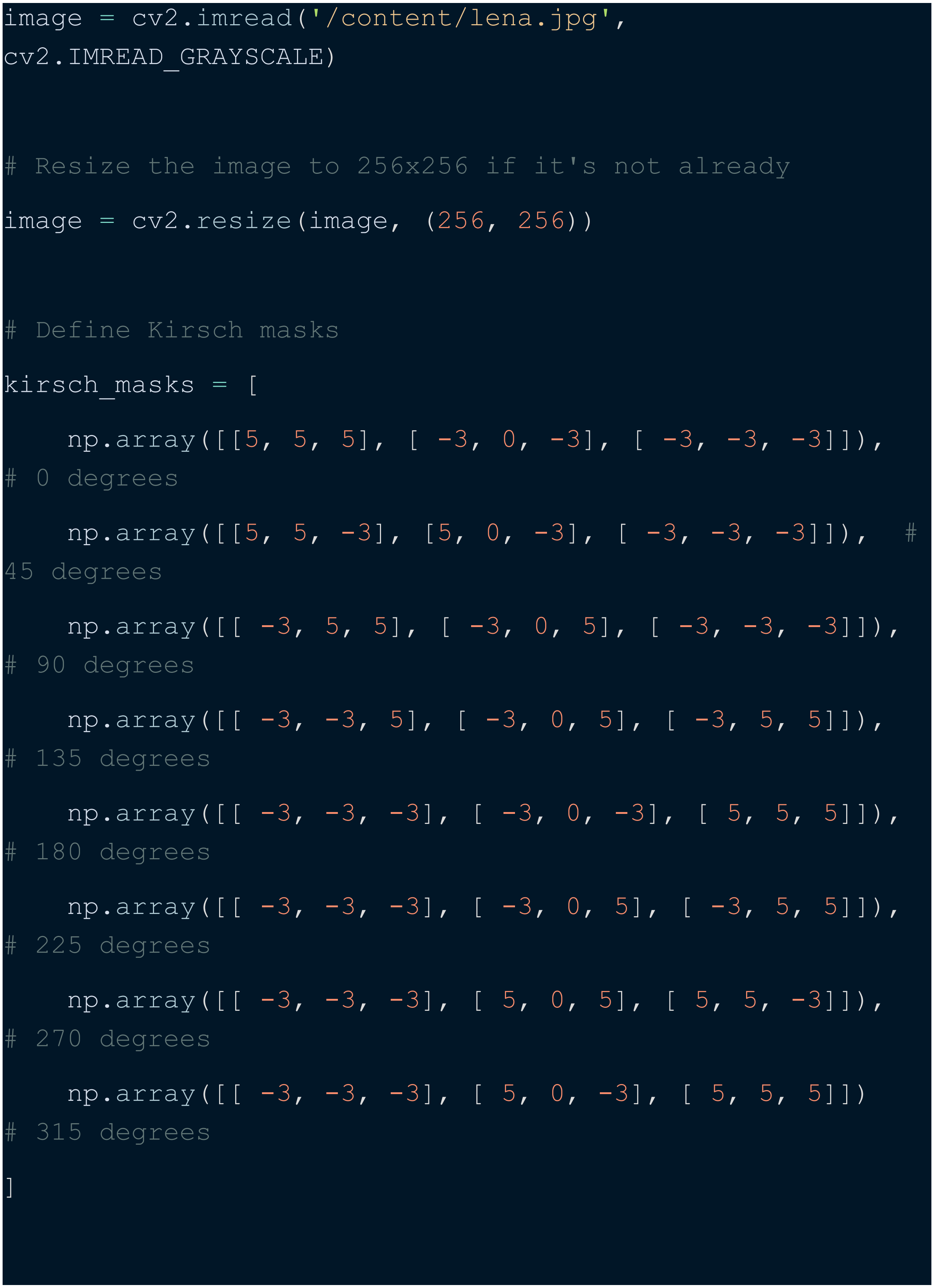
```python
#1

import cv2

import numpy as np

import matplotlib.pyplot as plt


# Read the image

image = cv2.imread('/content/download.jpg')

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


# Roberts Operator

roberts_kernel_x = np.array([[1, 0], [0, -1]], dtype=np.float32)

roberts_kernel_y = np.array([[0, 1], [-1, 0]], dtype=np.float32)

roberts_x = cv2.filter2D(gray_image, cv2.CV_32F, roberts_kernel_x)

roberts_y = cv2.filter2D(gray_image, cv2.CV_32F, roberts_kernel_y)

roberts_edge = cv2.magnitude(roberts_x, roberts_y)


# Prewitt Operator

prewitt_kernel_x = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]], dtype=np.float32)

prewitt_kernel_y = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]], dtype=np.float32)
```

```
prewitt_x = cv2.filter2D(gray_image, cv2.CV_32F, prewitt_kernel_x)
prewitt_y = cv2.filter2D(gray_image, cv2.CV_32F, prewitt_kernel_y)
prewitt_edge = cv2.magnitude(prewitt_x, prewitt_y)

# Sobel Operator
sobel_x = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=3)
sobel_edge = cv2.magnitude(sobel_x, sobel_y)

# Canny Edge Detector
canny_edge = cv2.Canny(gray_image, 100, 200)

# Display results
plt.figure(figsize=(10, 8))

plt.subplot(2, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title('Roberts Edge Detection')
plt.imshow(roberts_edge, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title('Prewitt Edge Detection')
plt.imshow(prewitt_edge, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.title('Sobel Edge Detection')
plt.imshow(sobel_edge, cmap='gray')
plt.axis('off')

plt.figure(figsize=(5, 5))
plt.title('Canny Edge Detection')
plt.imshow(canny_edge, cmap='gray')
plt.axis('off')

plt.show()
```

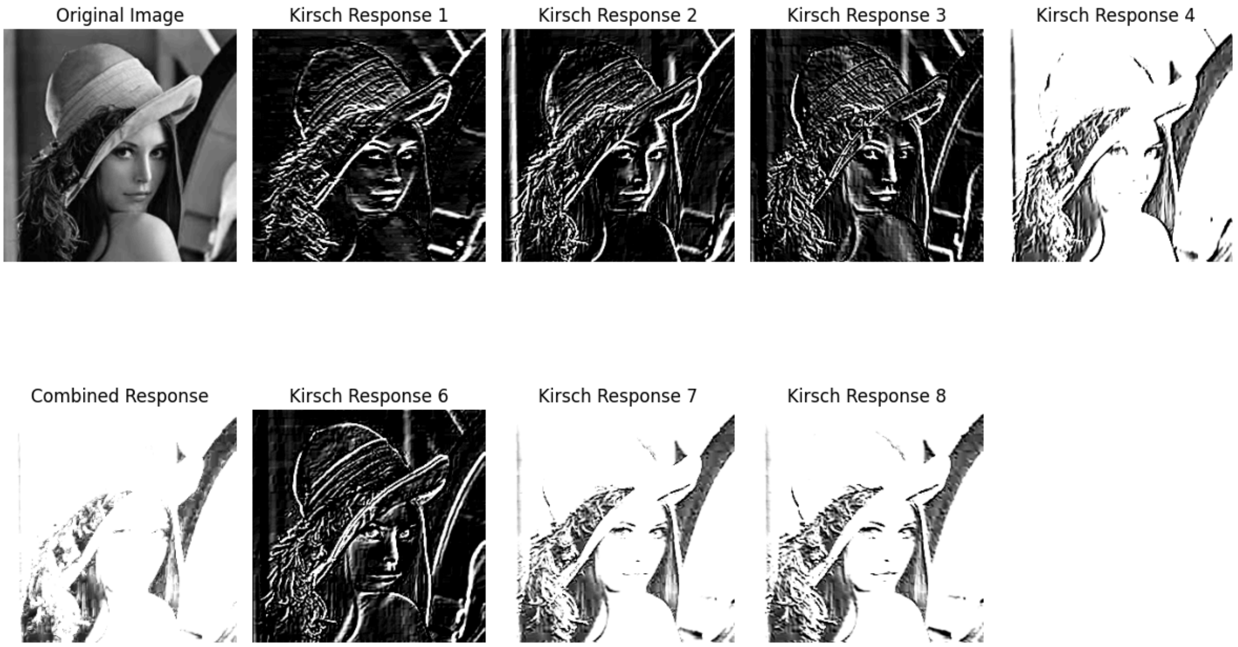


```
import cv2

import numpy as np

import matplotlib.pyplot as plt



# Load the Lena image (make sure it's 256x256 and
grayscale)
```



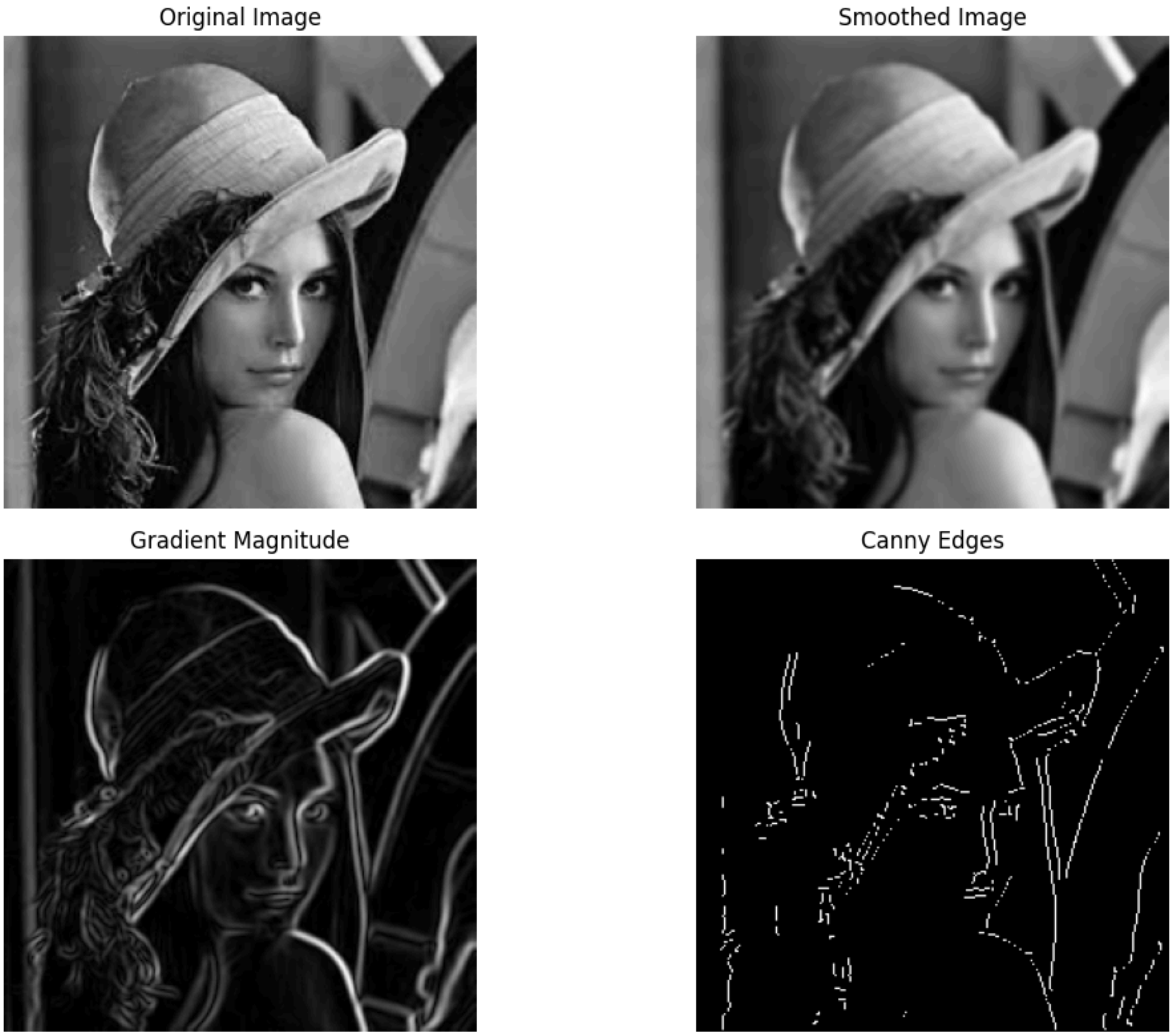
```
image = cv2.imread('/content/lena.jpg',
cv2.IMREAD_GRAYSCALE)


# Resize the image to 256x256 if it's not already

image = cv2.resize(image, (256, 256))


# Define Kirsch masks

kirsch_masks = [

    np.array([[5, 5, 5], [ -3, 0, -3], [ -3, -3, -3]]),
# 0 degrees

    np.array([[5, 5, -3], [5, 0, -3], [ -3, -3, -3]]),  #
45 degrees

    np.array([[ -3, 5, 5], [ -3, 0, 5], [ -3, -3, -3]]),
# 90 degrees

    np.array([[ -3, -3, 5], [ -3, 0, 5], [ -3, 5, 5]]),
# 135 degrees

    np.array([[ -3, -3, -3], [ -3, 0, -3], [ 5, 5, 5]]),
# 180 degrees

    np.array([[ -3, -3, -3], [ -3, 0, 5], [ -3, 5, 5]]),
# 225 degrees

    np.array([[ -3, -3, -3], [ 5, 0, 5], [ 5, 5, -3]]),
# 270 degrees

    np.array([[ -3, -3, -3], [ 5, 0, -3], [ 5, 5, 5]])
# 315 degrees

]
```

```
# Apply Kirsch masks
responses = []
for mask in kirsch_masks:
    response = cv2.filter2D(image, -1, mask)
    responses.append(response)

# Combine responses to get the maximum response
combined_response = np.maximum.reduce(responses)

# Display results
plt.figure(figsize=(12, 8))

plt.subplot(2, 5, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

for i, response in enumerate(responses):
    plt.subplot(2, 5, i + 2)
    plt.title(f'Kirsch Response {i + 1}')
    plt.imshow(response, cmap='gray')
    plt.axis('off')
plt.subplot(2, 5, 6)

plt.title('Combined Response')

plt.imshow(combined_response, cmap='gray')

plt.axis('off')


plt.tight_layout()

plt.show()
```



```
import cv2

import numpy as np

import matplotlib.pyplot as plt


def gaussian_smoothing(image, kernel_size=5, sigma=1.4):
    """Apply Gaussian smoothing to the image."""
    return cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)


def gradient_calculation(image):
    """Calculate gradients using Sobel operators."""
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    gradient_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
    gradient_direction = np.arctan2(sobel_y, sobel_x) * (180 / np.pi) % 180
    return gradient_magnitude, gradient_direction


def non_maximum_suppression(gradient_magnitude, gradient_direction):
    """Perform non-maximum suppression."""
    height, width = gradient_magnitude.shape
    suppressed = np.zeros_like(gradient_magnitude, dtype=np.float32)


    for i in range(1, height - 1):
        for j in range(1, width - 1):
            angle = gradient_direction[i, j]
            if (0 <= angle < 22.5) or (157.5 <= angle <= 180):
                neighbors = [gradient_magnitude[i, j + 1], gradient_magnitude[i, j - 1]]
            elif (22.5 <= angle < 67.5):
                neighbors = [gradient_magnitude[i + 1, j - 1], gradient_magnitude[i - 1, j + 1]]
            elif (67.5 <= angle < 112.5):
                neighbors = [gradient_magnitude[i + 1, j], gradient_magnitude[i - 1, j]]
            else:  # (112.5 <= angle < 157.5)
                neighbors = [gradient_magnitude[i - 1, j - 1], gradient_magnitude[i + 1, j + 1]]

            if gradient_magnitude[i, j] >= max(neighbors):
                suppressed[i, j] = gradient_magnitude[i, j]

    return suppressed

def hysteresis_thresholding(suppressed, low_threshold, high_threshold):
    """Apply hysteresis thresholding."""
    strong_edges = (suppressed > high_threshold)
    weak_edges = ((suppressed >= low_threshold) & (suppressed <= high_threshold))
    # Create an output image
    output = np.zeros_like(suppressed, dtype=np.uint8)
    output[strong_edges] = 255

    # Link weak edges to strong edges
    for i in range(1, suppressed.shape[0] - 1):
        for j in range(1, suppressed.shape[1] - 1):
            if weak_edges[i, j]:
                if (strong_edges[i + 1, j] or strong_edges[i - 1, j] or
                    strong_edges[i, j + 1] or strong_edges[i, j - 1] or
                    strong_edges[i + 1, j + 1] or strong_edges[i - 1, j - 1] or
                    strong_edges[i + 1, j - 1] or strong_edges[i - 1, j + 1]):
                    output[i, j] = 255

    return output

# Load the Lena image (make sure it's 256x256 and grayscale)
image = cv2.imread('/content/lena.jpg', cv2.IMREAD_GRAYSCALE)
image = cv2.resize(image, (256, 256))
```

```
# Step 1: Gaussian Smoothing

smoothed_image = gaussian_smoothing(image)


# Step 2: Gradient Calculation

gradient_magnitude, gradient_direction = gradient_calculation(smoothed_image)


# Step 3: Non-Maximum Suppression

nms_image = non_maximum_suppression(gradient_magnitude, gradient_direction)


# Step 4: Hysteresis Thresholding

low_threshold = 50

high_threshold = 150

edges = hysteresis_thresholding(nms_image, low_threshold, high_threshold)


# Display results

plt.figure(figsize=(12, 8))


plt.subplot(2, 2, 1)

plt.title('Original Image')

plt.imshow(image, cmap='gray')
plt.axis('off')


plt.subplot(2, 2, 2)

plt.title('Smoothed Image')

plt.imshow(smoothed_image, cmap='gray')

plt.axis('off')


plt.subplot(2, 2, 3)

plt.title('Gradient Magnitude')

plt.imshow(gradient_magnitude, cmap='gray')

plt.axis('off')


plt.subplot(2, 2, 4)

plt.title('Canny Edges')

plt.imshow(edges, cmap='gray')

plt.axis('off')


plt.tight_layout()

plt.show()
```



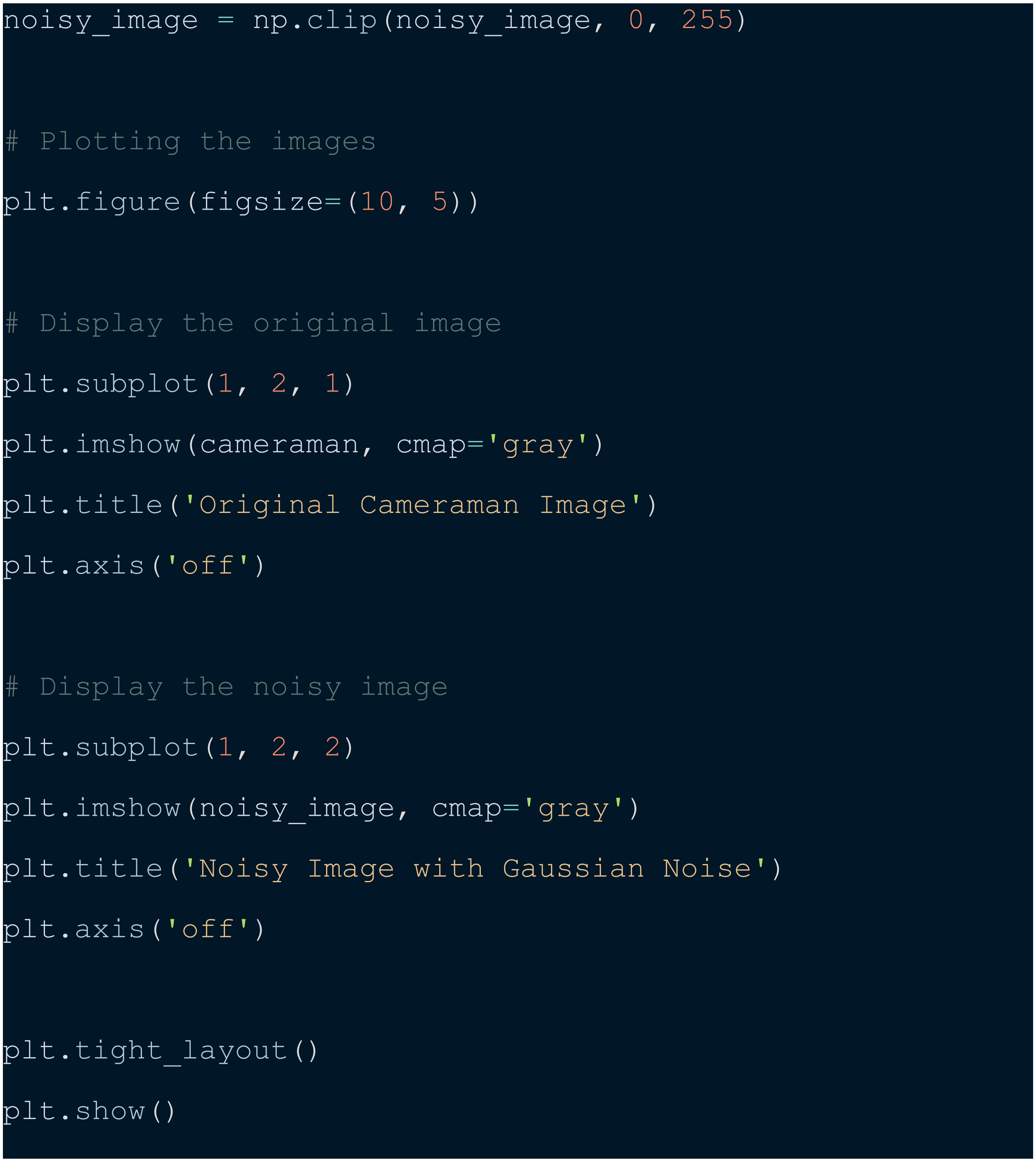
Task

Read 256x256 cameraman test image

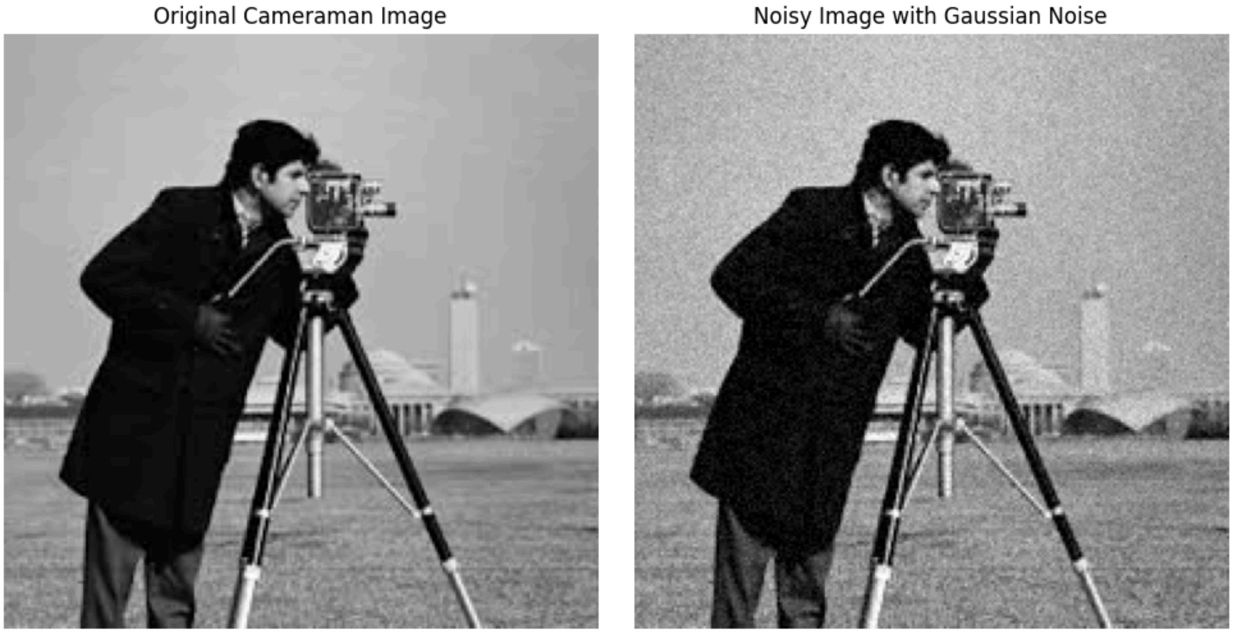
Add random noise which follows normal distribution with 0 mean and variance = 100 . display the noise free and noisy image and comment on the observed results

```
# Import required libraries

import numpy as np

import matplotlib.pyplot as plt
```

```
from skimage import data


# Load the cameraman image

# cv2.imread('/content/download.jpg')

cameraman = cv2.imread('/content/download.jpg',
cv2.IMREAD_GRAYSCALE)

# Ensure the image is 256x256

cameraman = cameraman[0:256, 0:256]


# Set the random seed for reproducibility

np.random.seed(0)


# Define noise parameters

mean = 0

variance = 100

sigma = np.sqrt(variance)


# Generate Gaussian noise

noise = np.random.normal(mean, sigma, cameraman.shape)


# Create the noisy image

noisy_image = cameraman + noise


# Clip the values to be in the valid range [0, 255]
```



```
noisy_image = np.clip(noisy_image, 0, 255)

# Plotting the images
plt.figure(figsize=(10, 5))

# Display the original image
plt.subplot(1, 2, 1)
plt.imshow(cameraman, cmap='gray')
plt.title('Original Cameraman Image')
plt.axis('off')

# Display the noisy image
plt.subplot(1, 2, 2)
plt.imshow(noisy_image, cmap='gray')
plt.title('Noisy Image with Gaussian Noise')
plt.axis('off')

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



Original Cameraman Image

Noisy Image with Gaussian Noise