```python
start = time.time()
result_matrix = [[sum(matrix1[i][k] * matrix2[k][j] for k in range(matrix_size)) for j in range(matrix_size)] for i in range(matrix_size)]
end = time.time()
print(f"Python list matrix multiplication time: {end - start:.5f} seconds")
```

Double-click (or enter) to edit

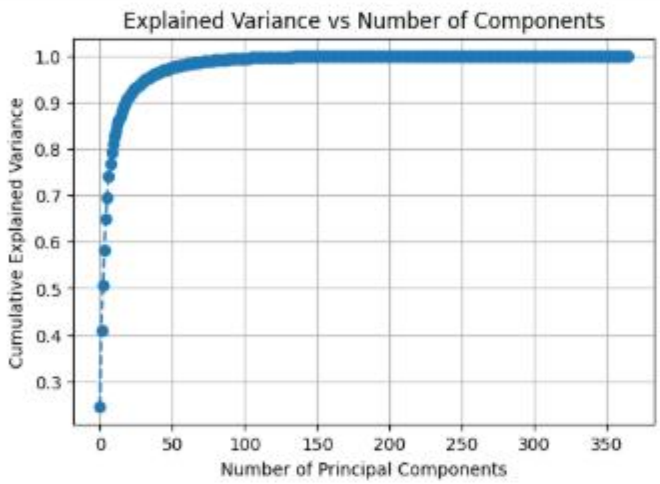
Using NumPy arrays, perform matrix multiplication of two matrices of size 1000x1000. Measure and print the time taken for this operation

Reconstructed Image with 200 Principal Components

Reconstructed Image with 50 Principal Components

Reconstructed Image with 100 Principal Components

```python
# Reconstruction and Experiments
for k in [10, 50, 100, 200]:  # Different combinations of PCs
    top_k_eigenvectors = eigenvectors[:, :k]
    reduced_data = np.dot(standardized_image, top_k_eigenvectors)
    reconstructed_image = np.dot(reduced_data, top_k_eigenvectors.T) + mean

    plt.figure(figsize=(6, 6))
    plt.imshow(reconstructed_image, cmap="gray")
    plt.axis("off")
    plt.title(f"Reconstructed Image with {k} Principal Components")
    plt.show()
```



Converted RGB Image

Binary Image (Threshold Applied)

Rotated Image (90 Degrees Clockwise)

```
plt.show()
```

Grayscale Image

Middle 150 Pixels Section

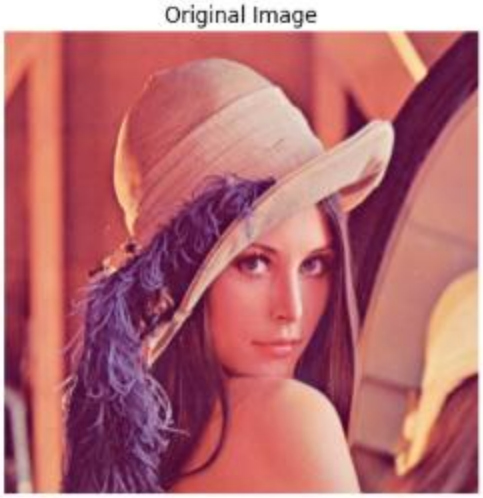
[40]

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# 1. Load and display a grayscale image
image_path = "/content/drive/MyDrive/Artificial Inte
gray_image = Image.open(image_path).convert("L")

plt.figure(figsize=(6, 6))
plt.imshow(gray_image, cmap="gray")
plt.axis("off")
plt.title("Grayscale Image")
plt.show()

# 2. Extract and display the middle section of the i
gray_array = np.array(gray_image)
height, width = gray_array.shape
middle_section = gray_array[:, width // 2 - 75: widt

plt.figure(figsize=(6, 6))
plt.imshow(middle_section, cmap="gray")
plt.axis("off")
plt.title("Middle 150 Pixels Section")
plt.show()

# 3. Apply a simple threshold to the image (binary c
binary_image = np.where(gray_array < 100, 0, 255).as
```

Top-Left 100x100 Pixels

```python
ax[1].imshow(G, cmap='Greens')
ax[1].set_title('Green Channel')
ax[1].axis('off')

ax[2].imshow(B, cmap='Blues')
ax[2].set_title('Blue Channel')
ax[2].axis('off')

plt.show()

# 4. Modify the top 100 × 100 pixels to a value of 210
modified_image = image_np.copy()
modified_image[:100, :100] = 210  # Set top-left 100x100 pixels to light gray

plt.figure(figsize=(6, 6))
plt.imshow(modified_image)
plt.axis("off")
plt.title("Modified Image (Top 100x100 Pixels = 210)")
plt.show()
```

Original Image



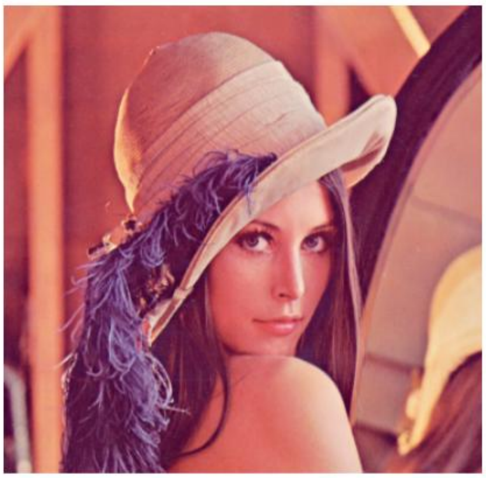
Top-Left 100x100 Pixels

```python
[39] from
     impo
```

```
[38] import numpy as np
     # Convert the Pillow image to a NumPy array
     image_array_colored = np . array ( image_colored )
     # Display the shape of the NumPy array (height , width , channels )
     print (" Shape of the image array :", image_array_colored .shape )
```

```
Shape of the image array : (357, 366, 4)
```

```
[37] from PIL import Image
     # display image in colab
     image_colored = Image . open ("/content/drive/MyDrive/Artificial Intelligence and Machine Learning/lenna_image.png")
     display ( image_colored )
```



```
[36] from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
pip install pillow
```

```
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (11.1.0)
```

+ Code + Text