1. Install open CV in our Colab

In [ ]:
```
! pip install opencv-python
! pip install opencv-python-headless
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dis
t-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dis
t-packages (from opencv-python) (1.26.4)
Requirement already satisfied: opencv-python-headless in /usr/local/lib/pytho
n3.10/dist-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dis
t-packages (from opencv-python-headless) (1.26.4)
```

2. Upload the image to Google colab My Computer.

In [ ]:
```
from google.colab import files
# Upload an image from local system
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Lion.jfif to Lion (1).jfif
```

3. Load and Display the image using openCV and Matplotlib Now that you have an image, you can load it using OpenCV and display it using Matplotlib

In [ ]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# load the iamge (adjuist the file name/ppath)
img = cv2.imread('/content/Lion.jfif')

# Convert BGR to RGb for for displaying with matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Corrected attribute name

# display  the image
plt.imshow(img_rgb)
plt.axis('off')
plt.show()
```

4. Basics image operation Before displaying you can applying, we can apply some processing opperation such as converting to grayscale or resizing

In [ ]:
```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(gray_img, cmap = 'gray')
plt.axis('off')
plt.show()
```

Resize the image

In [ ]:
```
resize_img = cv2.resize(img,(400, 400))
plt.imshow(cv2.cvtColor(resize_img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

In [ ]:

In [ ]:

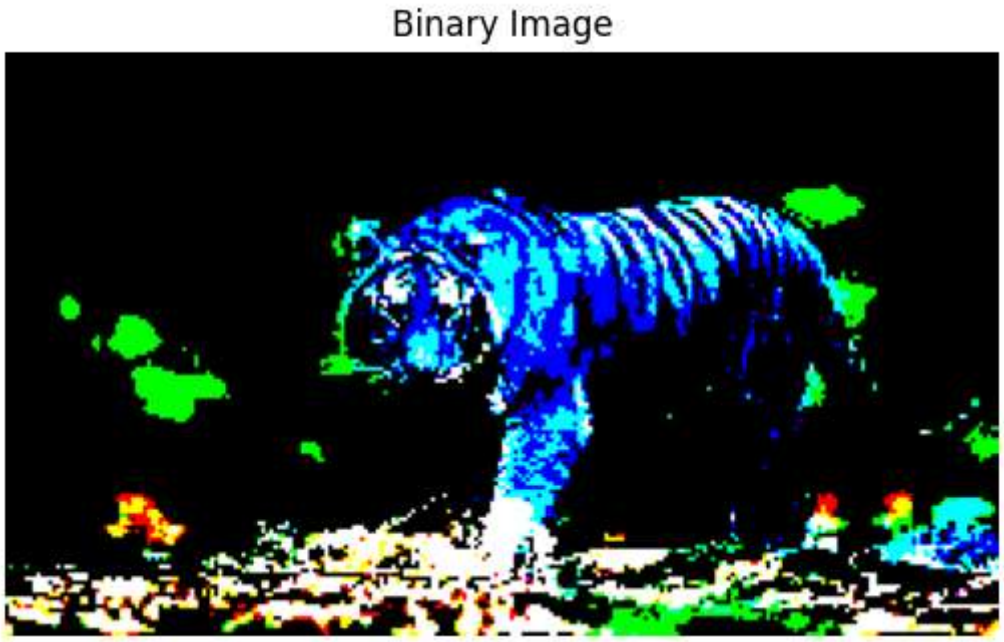
Tutorial: Negative of an iamge(Binary and Gray)

In [ ]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# load the iamge (adjuist the file name/ppath)
img = cv2.imread('/content/Lion.jfif')

# Display the orignal Grayscale image
plt.imshow(img, cmap = 'gray')
plt.title('Original Grayscale Image')
plt.axis('off')
plt.show()
```

Type *Markdown* and LaTeX: α^2

In [ ]:
```
# Create negative of the grayscale image
negative_img = 255 - img

# Display the negative iamge
plt.imshow(img, cmap = 'gray')
plt.title('Negative Grayscale Image')
plt.axis('off')
plt.show()
```

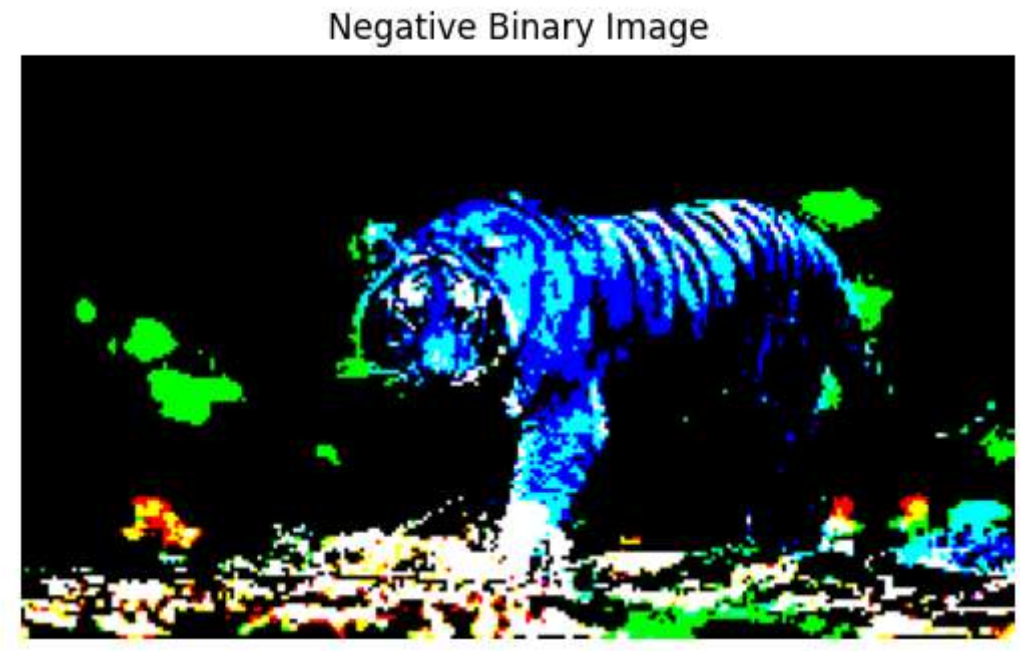
In [ ]:
```
# Apply binary thresholding
_, binary_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY) # Assign the se

# Display the binary iamge
plt.imshow(binary_img, cmap = 'gray')
plt.title('Binary Image')
plt.axis('off')
plt.show()
```



Create the Negative of the Binary Image

To get the negative of the binary image, you simply invert the pixel values (0 becomes 255, and 255 becomes 0):

In [ ]:
```
# Apply binary thresholding
_, negative_binary_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY) # Asst

# Display the binary image
plt.imshow(negative_binary_img, cmap = 'gray')
plt.title('Negative Binary Image')
plt.axis('off')
plt.show()
```



Task to Practice

Task 1: Load an image, convert it to grayscale, and display it.

Task 2. Create the negative of the grayscale image and display the result.

Task 3: Convert the grayscale image into a binary image using thresholding and display the binary image.

Task 4: Create and display the negative of the binary image. Task Extension: Play with Threshold Values in the binary thresholding step, experiment with different threshold values (eg, 100, 150, 200) and observe how the binary image changes:

Type *Markdown* and LaTeX: α^2

#to do tasks

##Lab 3

Contrast Switching of Low Contrast Image, Histogram and Histogram Equivalization

Lab3 CVJoynb-Colab

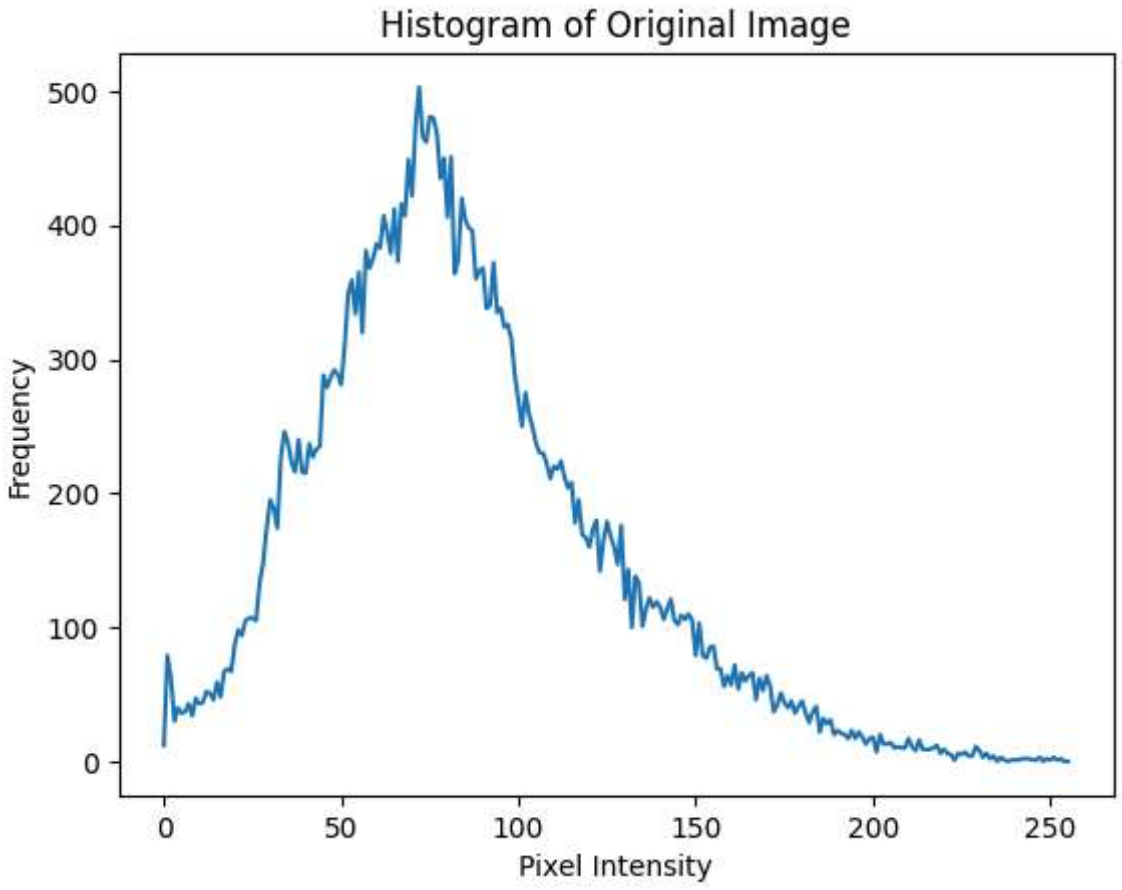
the details of the image more visible by spreading out the intensity values. *Preprocessing in Computer Vision: "Histogram equalization is often used as a preprocessing step in image recognition tasks to normalize brightness and contrast.

Load and Display the Image from an External Link:

In [ ]:
```
from google.colab import files
import cv2

import numpy as np
import matplotlib.pyplot as plt #Fixed typo here

from PIL import Image

#Load the uploaded inage file inage path=""

img = cv2.imread('/content/Lion.jfif')

#Convert the image to grayscale for histognan oqualization.
gray_Image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Fixed typo here and uncomme

# Display the original grayscale image

plt.imshow(gray_Image, cmap='gray')
plt.title('Original Grayscale Image')

plt.axis('off')

plt.show() #Fixed typo here
```

In [ ]:
```
#Display the Histogram of the Original Image

#Calculate and plot the histogram of the original image
hist = cv2.calcHist([gray_Image], [0], None, [256], [0, 256]) #Corrected typo f

#Plot the histogram
plt.plot(hist)

plt.title('Histogram of Original Image')

plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
```

Out[24]:
```
Text(0, 0.5, 'Frequency')
```

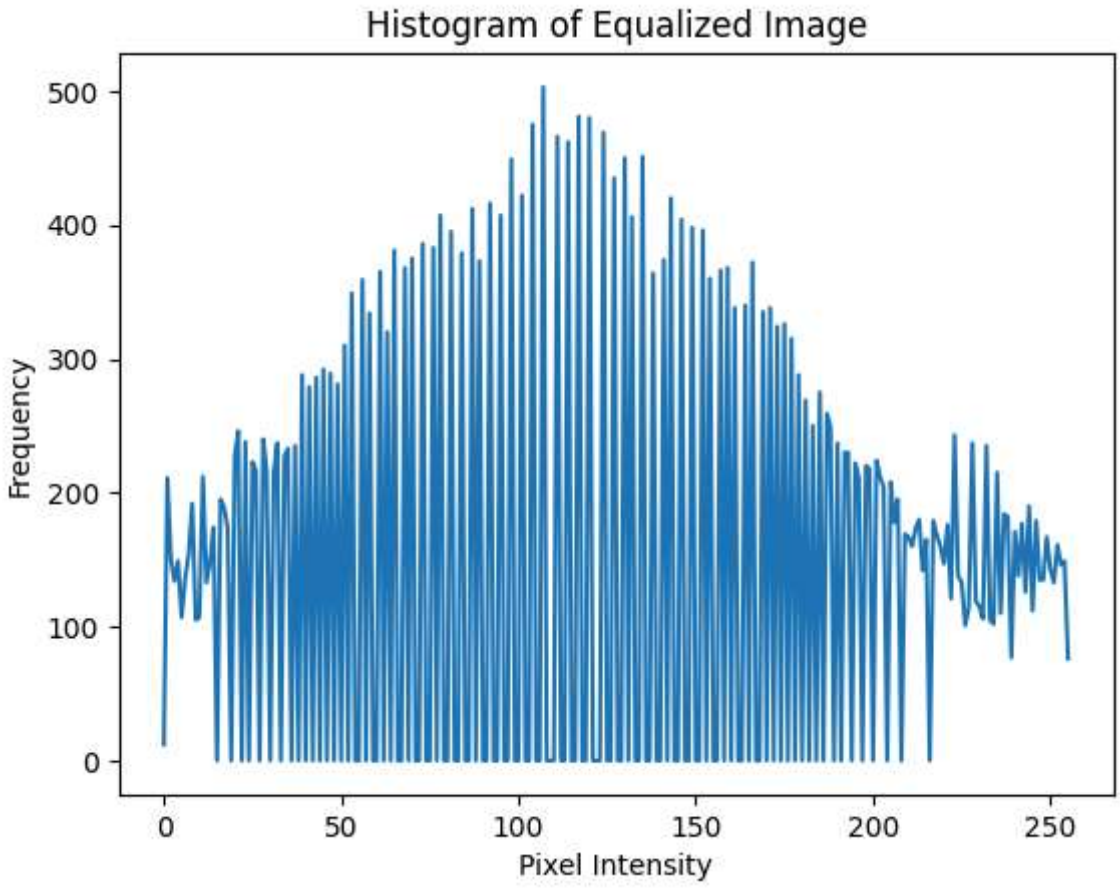


Apply Histogram Equalization

In [ ]:
```
#Histogram equalization improves the contrast by redistributing pixel intensiti

#Apply histogram equalization
equalized_image = cv2.equalizeHist(gray_Image) #Fixed typo gray_image to gray_I

#Display the equalized image
plt.imshow(equalized_image, cmap='gray')
plt.title('Equalized Image')

plt.axis('off') #Fixed typo pit to plt
plt.show()
```

Display the Histogram of the Equalized image

In [ ]:
```
#Calculate and plot the histogram of the equalized image
equalized_hist = cv2.calcHist([equalized_image], [0], None, [256], [0, 256])

#Plot the histogran of the equalized image
plt.plot(equalized_hist)

plt.title('Histogram of Equalized Image')

plt.xlabel("Pixel Intensity")

plt.ylabel('Frequency')

pit.show()
```



###Lab 4

Practice Tasks:

Uploading an image from your local machine.

Converting the image to grayscale.

Displaying the original image and its histogram.

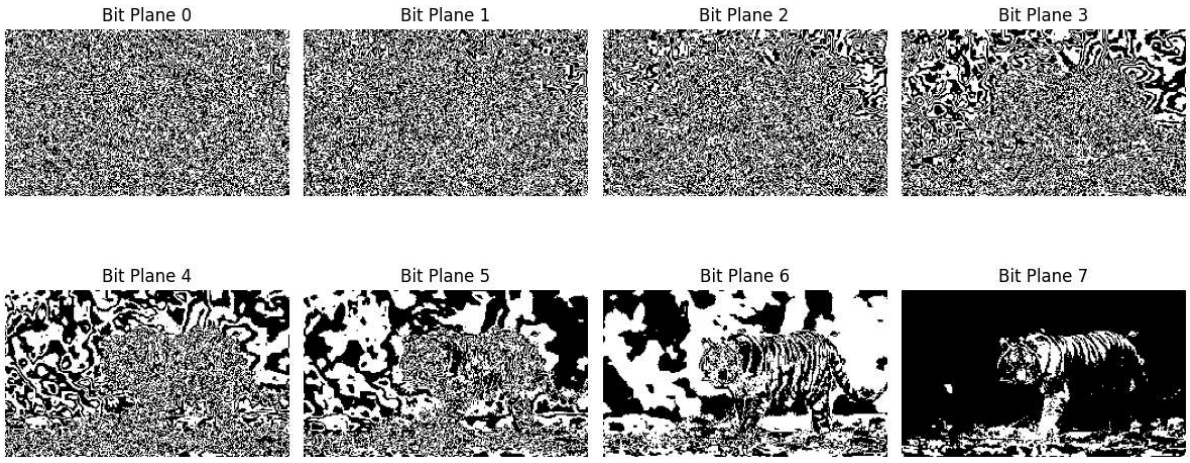
Applying histogram equalization to enhance the contrast. Displaying the enhanced image and its new histogram.

In [ ]:
```
from google.colab import files
import cv2

import numpy as np
import matplotlib.pyplot as plt #Fixed typo here

from PIL import Image

#Load the uploaded inage file inage path=""

img = cv2.imread('/content/Lion.jfif')

#Convert the image to grayscale for histognan oqualization.
gray_Image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Fixed typo here and uncomme

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

Extract Bit Planes

You can extract each bit plane using bitwise operations. Each bit of the pixel values can be isolated using a bit mask

In [42]:
```
# Function to extract and display bit planes

def show_bit_planes (image): #Removed extra space between ) and :
  bit_planes = [] #Indented this line

  for i in range(8): # Extract each bit plane using bitwise AND
    bit_plane = (image & (1<< i)) >> i # Isolate the i-th bit #Indented this li
    bit_planes.append(bit_plane * 255) # Scale to 255 for visualization #Indent


  #Plot all bit planes

  fig, axes = plt.subplots (2, 4, figsize=(12, 6)) #Indented this line and adde
  axes = axes.ravel() #Indented this line

  for i in range(8): #Indented this line

    axes[i].imshow(bit_planes[i], cmap='gray') #Indented this line

    axes[i].set_title(f'Bit Plane {i}') #Indented this line and fixed f-string

    axes[i].axis('off') #Indented this line

  plt.tight_layout() #Indented this line

  plt.show() #Indented this line

# Display bit planes
show_bit_planes(gray_Image) #Corrected variable name from gray_image to gray_Im
```



Explanation of Each Bit Plane

Bit Plane 0 (Least Significant Bit): This plane will typically show the least important details and noise.

Bit Plane 1: This plane may reveal additional small details, but it will still contain a lot of noise.

Bit Plane 2. Contains mid-range values that contribute slightly more to the image.

Bit Plane 3: Starts showing some structural information.

Bit Plane 4, 5: Represents more important information about the image.

Bit Plane 6: This starts showing significant structure in the image.

Bit Plane 7 (Most Significant Bit):

This plane holds the most important information and structure of the image,representing the general shape and primary features.

Bit Plane Tasks for Scene Images

Upload a Scene Image: Load a relevant scene image to work with.

Convert to Grayscale: Convert the color image to grayscale for easier analysis.

Extract Bit Planes: Isolate each bit plane (0 to 7) and analyze their contributions.

Visualize Each Bit Plane: Display each bit plane to see how they contribute to the image.

Reconstruct Image from Selected Bit Planes: Combine selected bit planes to form a new image.

Analyze the impact of Removing Bit Planes: Remove certain bit planes and observe the effects on image quality,

#to do

Lab 5

Affine transformations are a set of image processing techniques that alter the geometric properties of an image. They preserve points, straight lines, and planes. Common affine transformations include:

Translation: Moving the image along the x and/or y-axis.

Rotation: Rotating the image around a specified point.

Scaling. Resizing the image by a certain factor.

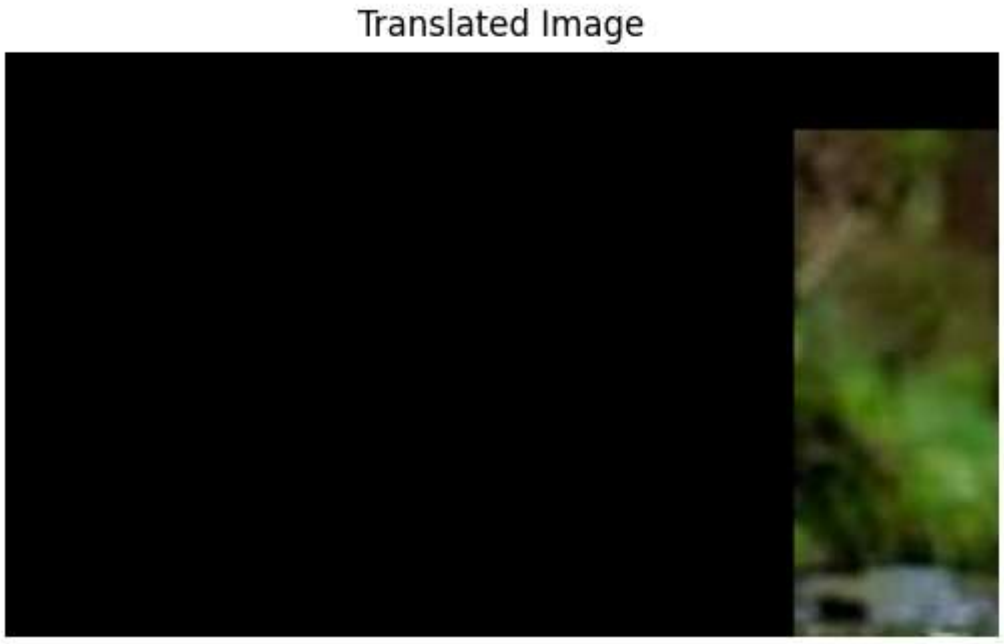
Shearing: Skewing the image in a specified direction (optional).

In [48]:

```python
from google.colab import files
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the uploaded image file

image_path = "/content/Lion.jfif"
Image = cv2.imread(image_path)

#Display the original
plt.imshow(cv2.cvtColor(Image, cv2.COLOR_BGR2RGB)) #Corrected variable name fro
plt.title('Original Image')
plt.axis('off')
plt.show()
```

Translation

In [52]:
```
def translate_image(image,tx,ty):
  translation_matrix=np.float32([[1,0,tx],[0,1,ty]])
  translated_image=cv2.warpAffine(image, translation_matrix, (image.shape[1], i
  return translated_image
translated_image=translate_image(Image,200,20) # Changed image to Image
plt.imshow(cv2.cvtColor(translated_image, cv2.COLOR_BGR2RGB))
plt.title('Translated Image')
plt.axis('off')
plt.show()
```



Rotation

In [53]:
```
def rotate_image(image,angle,centre=None):
  (h,w)=image.shape[:2]
  if centre is None:
    centre=(w//2, h//2)
  rotation_matrix=cv2.getRotationMatrix2D(centre, angle, 1.0)
  rotated_image=cv2.warpAffine(image, rotation_matrix, (w,h))
  return rotated_image
rotated_image=rotate_image(Image,45) # Changed image to Image to match the vari
plt.imshow(cv2.cvtColor(Image, cv2.COLOR_BGR2RGB)) # Changed image to Image
plt.title('Original Image')
plt.axis('off')
plt.show()
plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
plt.title('Rotated Image')
plt.axis('off')
plt.show()
```

Rotated Image

Scaling

In [56]:
```
def scale_image(image,fx,fy):
  scaled_image=cv2.resize(image, None, fx=fx, fy=fy, interpolation=cv2.INTER_LI
  return scaled_image
scaled_image = scale_image(Image,1.5,1.5) # Changed image to Image to match the
plt.imshow(cv2.cvtColor(Image, cv2.COLOR_BGR2RGB)) # Changed image to Image
plt.title('Original Image')
plt.axis('off')
plt.show()
plt.imshow(cv2.cvtColor(scaled_image, cv2.COLOR_BGR2RGB))
plt.title('Scaled Image (150%)')
plt.axis('off')
plt.show()
```

Shearing

In [58]:
```
def shear_image(image,shear_factor):
  shear_matrix=np.float32([[1,shear_factor,0],[0,1,0]]) # Changed to correctly
  sheared_image=cv2.warpAffine(image,shear_matrix,(int(image.shape[1])+int(shea
  return sheared_image
sheared_image=shear_image(Image,0.2) # Changed image to Image to match the vari
plt.imshow(cv2.cvtColor(Image, cv2.COLOR_BGR2RGB)) # Changed image to Image
plt.title("Sheared Image (shear factor: 0.5)") # Changed title to reflect actua
plt.axis('off')
plt.show()
```

Reflection

In [59]:
```
def reflect_image(image,axis):
  if axis=='horizontal':
    reflected_image=cv2.flip(image,0)
  elif axis=='vertical':
    reflected_image=cv2.flip(image,1)
  else:
    reflected_image=cv2.flip(image,-1)
  return reflected_image
reflected_image=reflect_image(Image,'horizontal') # Changed image to Image
plt.imshow(cv2.cvtColor(reflected_image, cv2.COLOR_BGR2RGB)) # Pass reflected_i
plt.title('Reflected Image (Horizontal)')
plt.axis('off')
plt.show() # Added parenth
```

Combined Transformation

In [65]:
```
def combined_transform(image, tx, ty, angle, scale_x, scale_y, shear_factor):
  (h, w) = image.shape[:2]
#Create individual transformation matrices.

  translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]]) # Changed Float32 t
  rotation_matrix = cv2.getRotationMatrix2D((w//2, h//2), angle, 1.0)
```