In [2]:
```
### Import required libraries
import cv2
import numpy as np
import matplotlib as plt
```

Basic Operations (Grayscale)

In [3]:
```
### Upload the image
# uploaded = files.upload()
# image_path = list(uploaded.keys())[0]

image_path = (r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Geomet

### Read the image
img = cv2.imread(image_path)
```

In [4]:
```
import matplotlib.pyplot as plt
import cv2

# Assuming 'img' is your original image loaded using cv2.imread or similar method

# Convert the image to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Display the original and grayscale images
print("Original Image:")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for proper d
plt.axis('off')
plt.show()

print("Grayscale Image:")
plt.imshow(img_gray, cmap='gray')  # Grayscale image needs the 'gray' colormap
plt.axis('off')
plt.show()
```

```
Original Image:
```

Grayscale Image:

Arithmetic Operations

In [5]:
```
# Step 2: Perform arithmetic operations on the grayscale image

# Adding a scalar value (50) to the image
img_add = cv2.add(img_gray, 50)

# Subtracting a scalar value (50) from the image
img_subtract = cv2.subtract(img_gray, 50)

# Multiplying the image by a scalar (1.5) and clipping values to 255
img_multiply = cv2.multiply(img_gray, 1.5)  # Multiplies element-wise
```

```
# Dividing the image by a scalar (2)
img_divide = cv2.divide(img_gray, 2)

# Step 3: Display the results using matplotlib
def display_image(img, title=""):
    plt.imshow(img, cmap='gray')
    plt.title(title)
    plt.axis('off')
    plt.show()

# Display each image
display_image(img_gray, "Original Grayscale Image")
display_image(img_add, "Image after Addition (img + 50)")
display_image(img_subtract, "Image after Subtraction (img - 50)")
display_image(img_multiply, "Image after Multiplication (img * 1.5)")
display_image(img_divide, "Image after Division (img / 2)")
```

Image after Addition (img + 50)

Image after Subtraction (img - 50)

Image after Multiplication (img * 1.5)

Image after Division (img / 2)

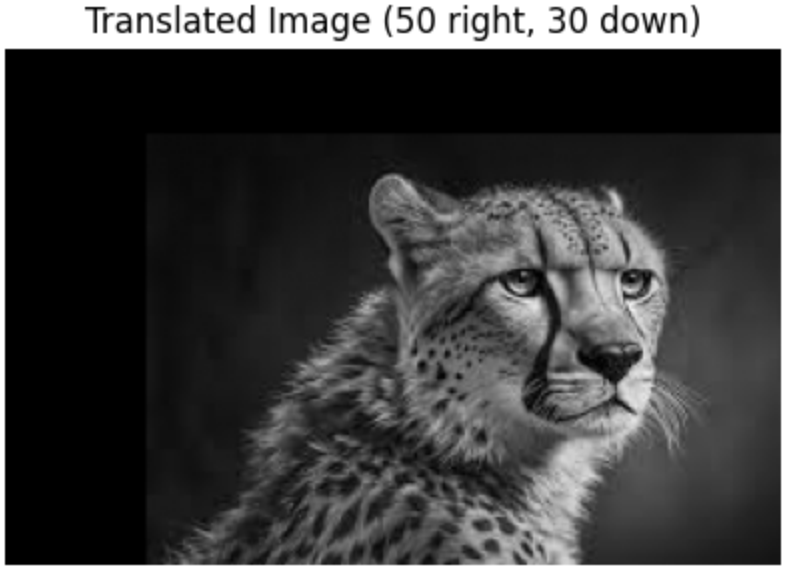
Geometric Transforms

In [6]:
```
# Get image dimensions
height, width = img_gray.shape
print("Original Image Dimensions:", width, height)  # Print original image size

# --- 1. Translation ---
tx, ty = 50, 30
translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
img_translated = cv2.warpAffine(img_gray, translation_matrix, (width, height))
```

```
# --- 2. Rotation ---
angle = 45
center = (width // 2, height // 2)
rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
img_rotated = cv2.warpAffine(img_gray, rotation_matrix, (width, height))

# --- 3. Scaling ---
scaling_factor = 1.5
new_width = int(width * scaling_factor)
new_height = int(height * scaling_factor)
print(f"New Scaled Dimensions: {new_width}, {new_height}")  # Check the new dimensi

# Resize image with new dimensions
img_scaled = cv2.resize(img_gray, (new_width, new_height))  # Resize to new dimensi

# --- 4. Shearing ---
shear_factor = 0.3
shearing_matrix = np.float32([[1, shear_factor, 0], [0, 1, 0]])
img_sheared = cv2.warpAffine(img_gray, shearing_matrix, (new_width, height))  # Adj

# Step 3: Display the results using Matplotlib with explicit image size
print("Original Grayscale Image:")
plt.figure(figsize=(5, 5))
plt.imshow(img_gray, cmap='gray')
plt.title("Original Grayscale Image")
plt.axis('off')
plt.show()

print("resized Image:")
plt.figure(figsize=(5, 5))
plt.imshow(img_resized, cmap='gray')
plt.title("Translated Image (50 right, 30 down)")
plt.axis('off')
plt.show()

print("Rotated Image (45 degrees):")
plt.figure(figsize=(5, 5))
plt.imshow(img_rotated, cmap='gray')
plt.title("Rotated Image (45 degrees)")
plt.axis('off')
plt.show()

print("Scaled Image (1.5x):")
plt.figure(figsize=(8, 8))  # Larger figure size for scaling
plt.imshow(img_scaled, cmap='gray')
plt.title(f"Scaled Image (1.5x) - Dimensions: {new_width}x{new_height}")
plt.axis('off')
plt.show()

print("Sheared Image (0.3 factor):")
plt.figure(figsize=(8, 5))  # Adjust based on shear factor
plt.imshow(img_sheared, cmap='gray')
plt.title("Sheared Image (0.3 factor)")
plt.axis('off')
plt.show()
```

```
Original Image Dimensions: 275 183
New Scaled Dimensions: 412, 274
Original Grayscale Image:
```


```
Translated Image:
```




```
Rotated Image (45 degrees):
```


```
Scaled Image (1.5x):
```


```
Sheared Image (0.3 factor):
```

Roll Number Problem

In [17]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Step 2: Load the two images corresponding to roll number 25
img1 = cv2.imread(r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Ge
img2 = cv2.imread(r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Ge

# Step 3: Resize both images to the same dimensions (use the smaller dimensions of
common_height = min(img1.shape[0], img2.shape[0])  # Use the minimum height
common_width = min(img1.shape[1], img2.shape[1])   # Use the minimum width

# Resize both images to match the smallest width and height
img1_resized = cv2.resize(img1, (common_width, common_height))
img2_resized = cv2.resize(img2, (common_width, common_height))

#todo: get rid of black borders

# Step 4: Apply horizontal translation to both images
translation_matrix = np.float32([[1, 0, 30], [0, 1, 0]])  # Translate left by 15 pi
img1_translated = cv2.warpAffine(img1_resized, translation_matrix, (common_width, c

translation_matrix = np.float32([[1, 0, -30], [0, 1, 0]])  # Translate right by 15
img2_translated = cv2.warpAffine(img2_resized, translation_matrix, (common_width, c

# Step 5: Perform addition and subtraction
img_add = cv2.add(img1_resized, img2_resized)  # Add the two images
img_subtract = cv2.subtract(img1_resized, img2_resized)  # Subtract the two images

# Step 6: Display the resulting images for addition and subtraction
print("Image after Addition (2.png + 5.png):")
img_rgb = cv2.cvtColor(img_add, cv2.COLOR_BGR2RGB)

# Display using matplotlib
plt.imshow(img_rgb)
```

```
plt.axis('off')
plt.show()

print("Image after Subtraction (2.png - 5.png):")
img_rgb = cv2.cvtColor(img_subtract, cv2.COLOR_BGR2RGB)

# Display using matplotlib
plt.imshow(img_rgb)
plt.axis('off')
plt.show()

# Step 7: Save the resulting images from addition and subtraction
cv2.imwrite(r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Geometri
cv2.imwrite(r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Geometri

# Step 8: Concatenate the images horizontally to form 25.jpg
combined_img = np.hstack((img1_translated, img2_translated))  # Horizontal stacking

# Step 9: Display and save the combined image
print("Combined Image (25.jpg):")

# Convert to RGB for proper color display in matplotlib
img_rgb = cv2.cvtColor(combined_img, cv2.COLOR_BGR2RGB)

# Display using matplotlib
plt.imshow(img_rgb)
plt.axis('off')
plt.show()

cv2.imwrite(r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Geometri
```

```
Image after Addition (2.png + 5.png):
```

```
Image after Subtraction (2.png - 5.png):
```

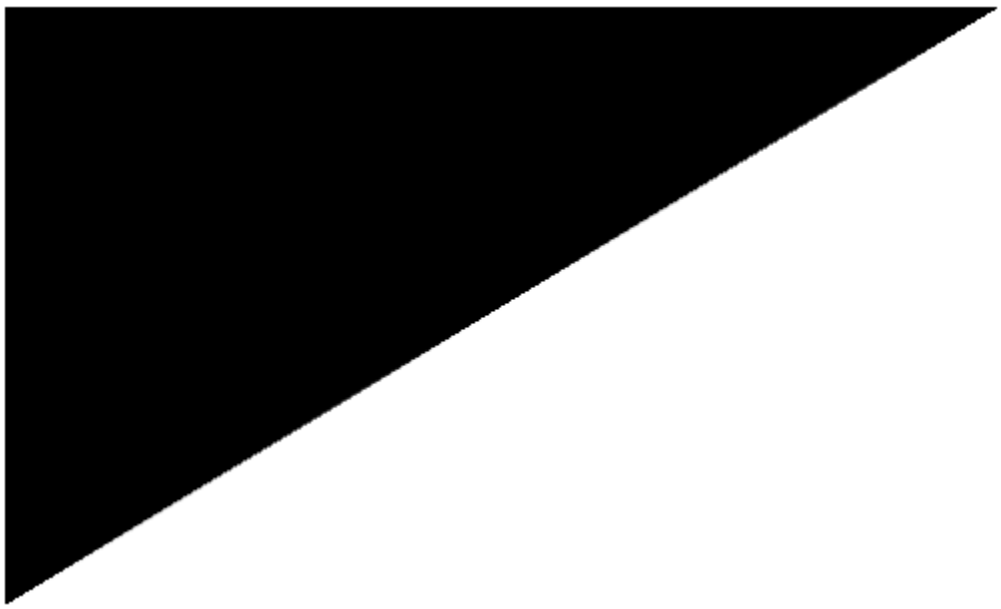
Combined Image (25.jpg):

Out[17]: True

Diagonal (Pattern 1)

In [36]:
```
# Step 1: Create a blank image (rectangle) with white background
height, width = 300, 500  # You can adjust the dimensions here
img = np.ones((height, width, 3), dtype=np.uint8) * 255  # Initialize with white (2

# Step 2: Fill the bottom-right triangle (diagonal split) with black
for y in range(height):
### uncomment based on "top-left diagonal" or "top-right diagonal"
    # for x in range(y * width // height, width):
    for x in range(0, width - (y * width // height)):
```

```
        img[y, x] = (0, 0, 0)  # Set the pixels to black

# Step 3: Show the image
# Display using matplotlib
plt.imshow(img)
plt.axis('off')
plt.show()

# Step 4: Save the image if needed
cv2.imwrite(r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Geometri
```

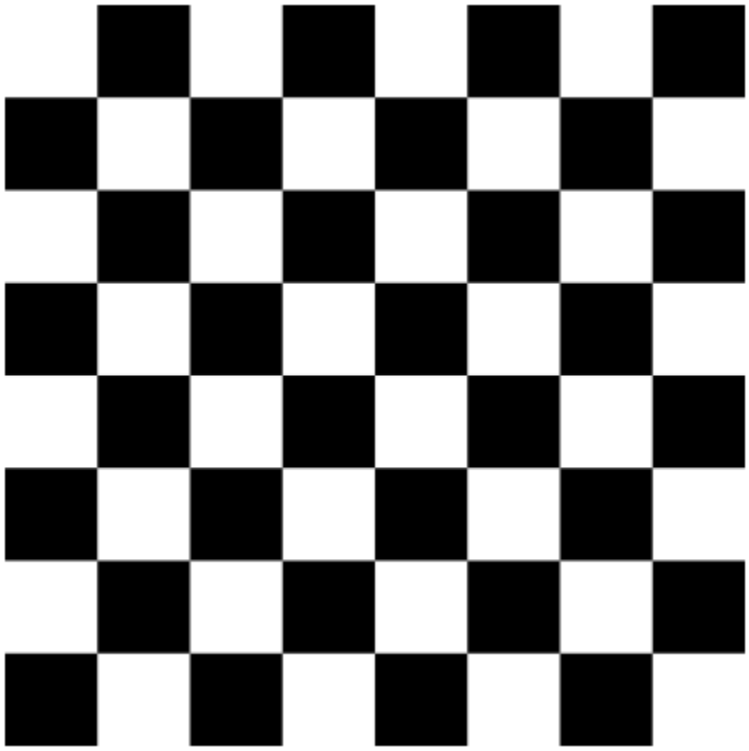


Out[36]: True

Chessboard (Pattern 2)

In [37]:
```
board_size = 8
square_size = 60
width = height = board_size * square_size

img = np.ones((height, width, 3), dtype=np.uint8) * 255

for row in range(board_size):
    for col in range(board_size):
        if (row + col) % 2 == 0:
            img[row * square_size: (row + 1) * square_size, col * square_size: (col
        else:
            img[row * square_size: (row + 1) * square_size, col * square_size: (col

# Convert to RGB for proper color display in matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display using matplotlib
plt.imshow(img_rgb)
plt.axis('off')
plt.show()

# Save the image
```

```
cv2.imwrite(r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Geometri
```



Out[37]: True

Distance Measures

In [39]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function to generate a solid Euclidean distance (circle)
def euclidean_distance(img, center, max_distance):
    height, width, _ = img.shape
    for y in range(height):
        for x in range(width):
            dist = np.sqrt((x - center[0])**2 + (y - center[1])**2)
            if dist <= max_distance:
                img[y, x] = (0, 0, 0)  # Inside the circle, set to black
            else:
                img[y, x] = (127, 127, 127)  # Outside the circle, set to white
    return img

# Function to generate a solid Manhattan distance (diamond)
def manhattan_distance(img, center, max_distance):
    height, width, _ = img.shape
    for y in range(height):
        for x in range(width):
            dist = abs(x - center[0]) + abs(y - center[1])
            if dist <= max_distance:
                img[y, x] = (0, 0, 0)  # Inside the diamond, set to black
            else:
                img[y, x] = (127, 127, 127)  # Outside the diamond, set to white
```

```
    return img

# Function to generate a solid Chessboard distance (square)
def chessboard_distance(img, center, max_distance):
    height, width, _ = img.shape
    for y in range(height):
        for x in range(width):
            dist = max(abs(x - center[0]), abs(y - center[1]))
            if dist <= max_distance:
                img[y, x] = (0, 0, 0)  # Inside the square, set to black
            else:
                img[y, x] = (127, 127, 127)  # Outside the square, set to white
    return img

# Set up parameters for the grid
height, width = 63, 63
center = (width // 2, height // 2)  # center of the image
max_distance = min(center) // 2  # Max distance will be half of the smaller dimensi

# Create blank images for each distance
euclidean_img = np.ones((height, width, 3), dtype=np.uint8) * 127  # white backgrou
manhattan_img = np.ones((height, width, 3), dtype=np.uint8) * 127  # white backgrou
chessboard_img = np.ones((height, width, 3), dtype=np.uint8) * 127  # white backgro

# Generate solid distance figures
euclidean_img = euclidean_distance(euclidean_img, center, max_distance)
manhattan_img = manhattan_distance(manhattan_img, center, max_distance)
chessboard_img = chessboard_distance(chessboard_img, center, max_distance)

# Plot using Matplotlib with horizontal alignment
fig, axes = plt.subplots(1, 3, figsize=(15, 5))  # 1 row, 3 columns for side-by-sid

# Show the images in the axes
axes[0].imshow(cv2.cvtColor(euclidean_img, cv2.COLOR_BGR2RGB))
axes[0].set_title('Euclidean Distance (Circle)')
axes[0].axis('off')

axes[1].imshow(cv2.cvtColor(manhattan_img, cv2.COLOR_BGR2RGB))
axes[1].set_title('Manhattan Distance (Diamond)')
axes[1].axis('off')

axes[2].imshow(cv2.cvtColor(chessboard_img, cv2.COLOR_BGR2RGB))
axes[2].set_title('Chessboard Distance (Square)')
axes[2].axis('off')

# Adjust layout and display
plt.tight_layout()
plt.show()

# Optionally save the image
fig.savefig(r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P1_Geometri
```


Euclidean Distance (Circle)
Manhattan Distance (Diamond)
Chessboard Distance (Square)