Digital Imaging Systems Project 01

Q1

Q1 a)

In [1]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image

%matplotlib inline
```

In [2]:

```
image = cv2.imread("lena.png")
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray_image.shape
```

Out[2]:

```
(440, 440)
```

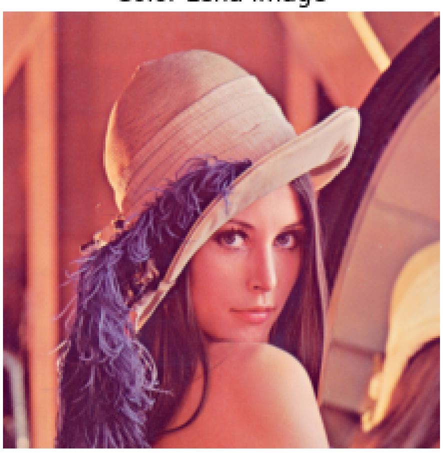
In [3]:

```
image_wolves = cv2.imread("wolves.png")
```

In [4]:

```
image1 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image1)
plt.title('Color Lena Image')
plt.axis('off')
plt.show()
```

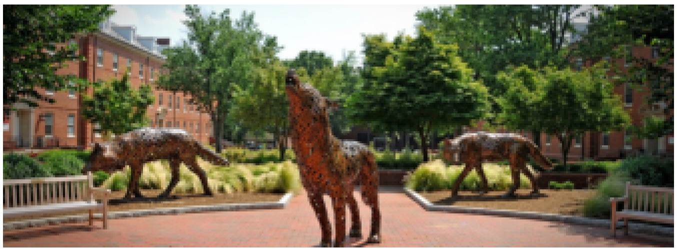
Color Lena Image



In [5]:

```
image_wolves = cv2.cvtColor(image_wolves, cv2.COLOR_BGR2RGB)
plt.imshow(image_wolves)
plt.title('Wolves color')
plt.axis('off')
plt.show()
```

Wolves color



In [6]:

```
plt.imshow(gray_image, cmap = "gray")
plt.title('Gray Lena Image')
plt.axis('off')
plt.show()
```

Gray Lena Image

Padding function

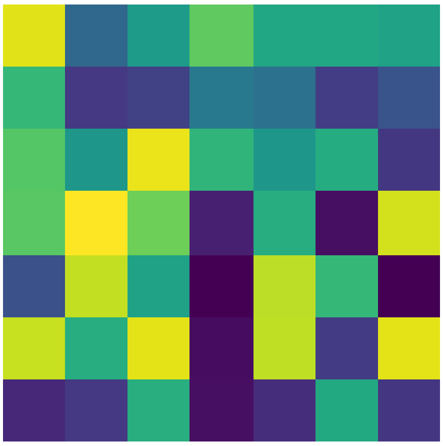
In [7]:

```
#pad is value of padding and is initialized to zero
#kernel_rows
#kernel_cols

def padding(input_image, pad = 0, kernel_rows = 0, kernel_cols = 0):

    #pad = 0 for zero padding
    #pad = 1 for wrap around
    #pad = 2 for copy edge
    #pad = 3 for reflect across edge

    input_dims = (input_image.shape[0], input_image.shape[1])

    #individual padding values for rows
    top_row_pad_value = int(np.ceil((kernel_rows-1)/2))
    bottom_row_pad_value = int(np.floor((kernel_rows-1)/2))

    #individual padding values for cols
    left_col_pad_value = int(np.ceil((kernel_cols-1)/2))
    right_col_pad_value = int(np.floor((kernel_cols-1)/2))

    if pad == 0:
        #zero padding
        padded_dims = (input_image.shape[0] + top_row_pad_value + bottom_row_pad_value
                       input_image.shape[1] + left_col_pad_value + right_col_pad_value

        padded_image = np.zeros(padded_dims)

        #inserting the original image to the zeros image
        padded_image[top_row_pad_value : top_row_pad_value + input_image.shape[0],
                     left_col_pad_value : left_col_pad_value + input_image.shape[1]] =

    if pad == 1:

        #wrap around
        padded_dims = (input_image.shape[0] + top_row_pad_value + bottom_row_pad_value
                       input_image.shape[1] + left_col_pad_value + right_col_pad_value

        padded_image = np.zeros(padded_dims)

        #inserting the original image to the zeros image
        padded_image[top_row_pad_value : top_row_pad_value + input_image.shape[0],
                     left_col_pad_value : left_col_pad_value + input_image.shape[1]] =


        #upper most row padding
        if top_row_pad_value != 0:
            padded_image[0 : top_row_pad_value, : ] = padded_image[-1*(top_row_pad_val
        #lowest row padding
        if bottom_row_pad_value != 0:
            padded_image[-1*(bottom_row_pad_value) : , : ] = padded_image[top_row_pad_
        #right most column padding
        if right_col_pad_value != 0:
            padded_image[ : ,-1*(right_col_pad_value) : ] = padded_image[ : ,left_col_
        #left most column padding
        if left_col_pad_value != 0:
            padded_image[ : ,0 : left_col_pad_value] = padded_image[ : ,-1*(left_col_p

    if pad == 2:
```

```

        #copy edge
        padded_dims = (input_image.shape[0] + top_row_pad_value + bottom_row_pad_value
                       input_image.shape[1] + left_col_pad_value + right_col_pad_value

        #print(padded_dims)
        padded_image = np.zeros(padded_dims)#.astype(int)
        #print("Dimensions of the padded image:", padded_image.shape)

        #inserting the original image to the zeros image
        padded_image[top_row_pad_value : top_row_pad_value + input_image.shape[0],
                     left_col_pad_value : left_col_pad_value + input_image.shape[1]] =


        #upper most row padding
        if top_row_pad_value != 0:
            padded_image[0 : top_row_pad_value, : ] = padded_image[[top_row_pad_value]
        #lowest row padding
        if bottom_row_pad_value != 0:
            padded_image[-1*(bottom_row_pad_value) : , : ] = padded_image[[-1*bottom_r
        #right most column padding
        if right_col_pad_value != 0:
            padded_image[ : ,-1*(right_col_pad_value) : ] = padded_image[ : ,[-1*(righ
        #left most column padding
        if left_col_pad_value != 0:
            padded_image[ : ,0 : left_col_pad_value] = padded_image[ : ,[left_col_pad_

    if pad == 3:

        #reflect across edge
        padded_dims = (input_image.shape[0] + top_row_pad_value + bottom_row_pad_value
                       input_image.shape[1] + left_col_pad_value + right_col_pad_value

        #print(padded_dims)
        padded_image = np.zeros(padded_dims)#.astype(int)
        #print("Dimensions of the padded image:", padded_image.shape)

        #inserting the original image to the zeros image
        padded_image[top_row_pad_value : top_row_pad_value + input_image.shape[0],
                     left_col_pad_value : left_col_pad_value + input_image.shape[1]] =


        #upper most row padding
        if top_row_pad_value != 0:
            padded_image[0 : top_row_pad_value, : ] = np.flip(padded_image[top_row_pad
        #lowest row padding
        if bottom_row_pad_value != 0:
            padded_image[-1*(bottom_row_pad_value) : , : ] = np.flip(padded_image[-2*(
        #right most column padding
        if right_col_pad_value != 0:
            padded_image[ : ,-1*(right_col_pad_value) : ] = np.flip(padded_image[ : ,-
        #left most column padding
        if left_col_pad_value != 0:
            padded_image[ : ,0 : left_col_pad_value] = np.flip(padded_image[ : ,left_c

    return padded_image
```

In [8]:

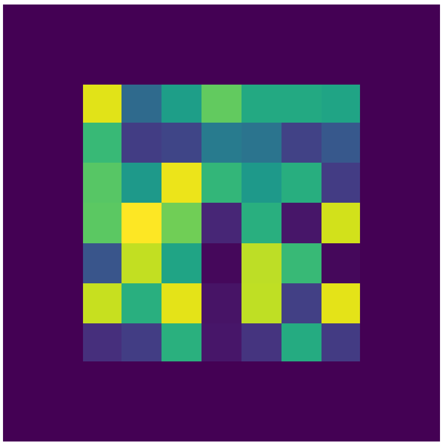
```
#random image to visualize padding
test_image_1 = np.random.randint(0,255,(7,7), dtype = np.uint8)
plt.imshow(test_image_1)
plt.axis('off')
plt.show()
```



Zero Padding

In [9]:

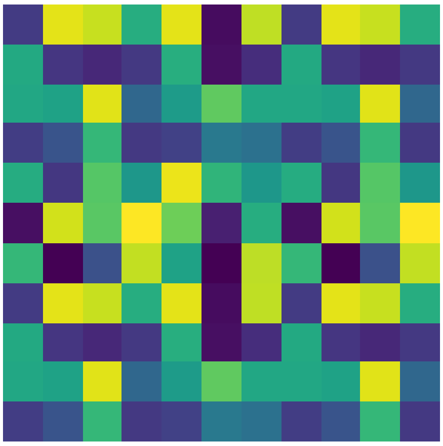
```
Image = padding(test_image_1, 0, 5, 5)
plt.imshow(Image)
plt.axis('off')
plt.show()
```



Wrap around Padding

In [10]:

```
Image = padding(test_image_1, 1, 5, 5)
plt.imshow(Image)
plt.axis('off')
plt.show()
```



Copy Edge Padding

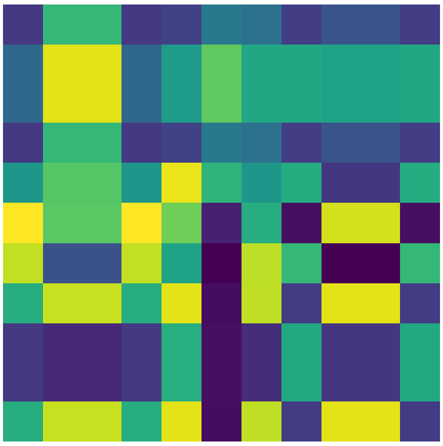
In [11]:

```
Image = padding(test_image_1, 2, 5, 5)
plt.imshow(Image)
plt.axis('off')
plt.show()
```

Reflect Edge Padding

In [12]:

```
#reflect edge padding
Image = padding(test_image_1, 3, 5, 5)
plt.imshow(Image)
plt.axis('off')
plt.show()
```



Function that multiplies the sub-image of the image(f) with the kernel

In [13]:

```
def convolution(f,w):

    return np.sum(f*w)
```

Conv2 Function

In [14]:

```
def conv2(f,w,pad = 0):

    if len(f.shape)< 3:

        image_padded = padding(f, pad, w.shape[0], w.shape[1])

        conv_image = np.zeros((f.shape[0], f.shape[1]))

        for row in range(conv_image.shape[0]):

            for col in range(conv_image.shape[1]):

                conv_image[row][col] = convolution(image_padded[row:row+w.shape[0],col:

        return conv_image


    elif len(f.shape) == 3:

        b,g,r = cv2.split(f)

        image_padded_b = padding(b, pad, w.shape[0], w.shape[1])
        image_padded_g = padding(g, pad, w.shape[0], w.shape[1])
        image_padded_r = padding(r, pad, w.shape[0], w.shape[1])

        conv_image_b = np.zeros((b.shape[0],b.shape[1]))
        conv_image_g = np.zeros((g.shape[0],g.shape[1]))
        conv_image_r = np.zeros((r.shape[0],r.shape[1]))

        for row in range(conv_image_b.shape[0]):

            for col in range(conv_image_b.shape[1]):

                conv_image_b[row][col] = convolution(image_padded_b[row:row + w.shape[(
                conv_image_g[row][col] = convolution(image_padded_g[row:row + w.shape[(
                conv_image_r[row][col] = convolution(image_padded_r[row:row + w.shape[(

        conv_image = cv2.merge((conv_image_b, conv_image_g, conv_image_r)).astype(np.u:


        return conv_image
```

Testing on a simple image

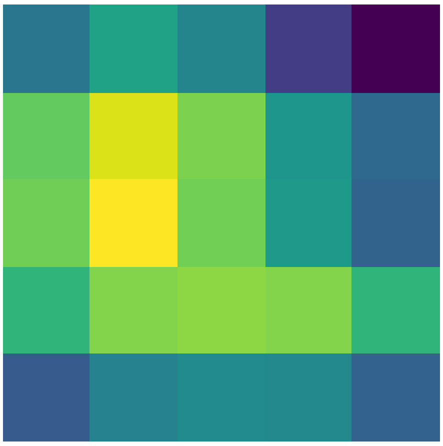
In [15]:

```
test_image_1 = np.random.randint(0,255,(5,5), dtype = np.uint8)
plt.imshow(test_image_1)
plt.axis('off')
plt.show()
```

In [16]:

```
# box filter on color image
w = 1/9*np.ones((3,3))
g = conv2(test_image_1,w,0)
plt.imshow(g)
plt.title('Convolution with 3*3 Box Filter on test image')
plt.axis('off')
plt.show()
```

Convolution with 3*3 Box Filter on test image



Box Filter on the lena image

In [17]:

```
# box filter on color image
w = 1/9*np.ones((3,3))
g = conv2(image1,w,0)
plt.imshow(g)
plt.title('3*3 Box Filter on gray image')
plt.axis('off')
plt.show()
```

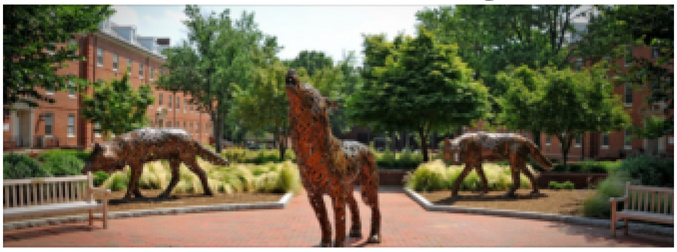
3*3 Box Filter on gray image

Box Filter on wolves image

In [18]:

```
w = 1/9*np.ones((3,3))
g = conv2(image_wolves,w,0)
plt.imshow(g)
plt.title('3*3 Box Filter on color image')
plt.axis('off')
plt.show()
```

3*3 Box Filter on color image

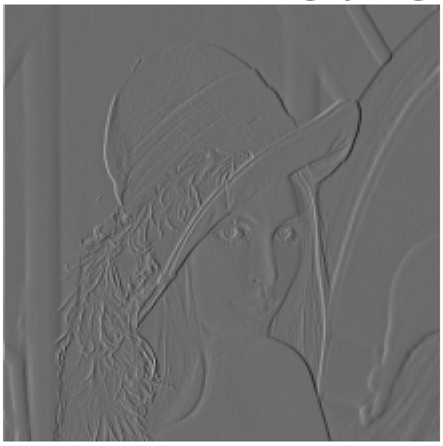


First Order Derivative Filter

In [19]:

```
w = np.array([[-1,1]])
g = conv2(gray_image,w,0)
plt.imshow(g, cmap = 'gray')
plt.title('First order derivative Filter on gray image (Mx)')
plt.axis('off')
plt.show()
```

First order derivative Filter on gray image (Mx)



In [20]:

```
w = np.array([[-1], [1]])
g = conv2(gray_image,w,0)
plt.imshow(g, cmap = 'gray')
plt.title('First order derivative Filter on gray image (My)')
plt.axis('off')
plt.show()
```

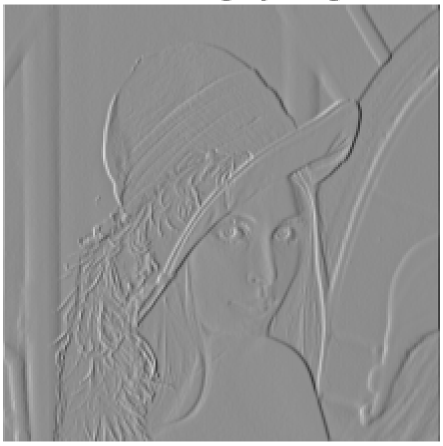
First order derivative Filter on gray image (My)

Prewitt Filter

In [21]:

```
#prewitt filter on gray image
w = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
g = conv2(gray_image,w,0)
plt.imshow(g, cmap = 'gray')
plt.title('Prewitt Filter on gray image (Mx)')
plt.axis('off')
plt.show()
```

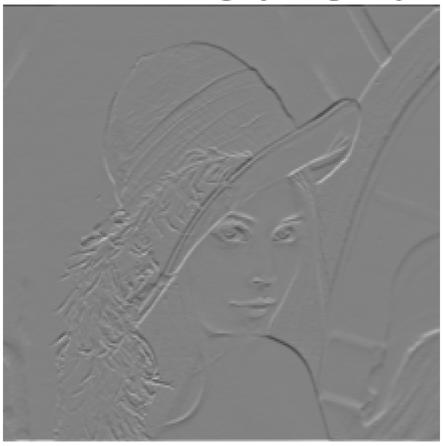
Prewitt Filter on gray image (Mx)



In [22]:

```
w = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
g = conv2(gray_image,w,0)
plt.imshow(g, cmap = 'gray')
plt.title('Prewitt Filter on gray image (My)')
plt.axis('off')
plt.show()
```

Prewitt Filter on gray image (My)



Roberts Filter

In [23]:

```
w = np.array([[0,1],[-1,0]])
g = conv2(gray_image,w,0)
plt.imshow(g, cmap = 'gray')
plt.title('Roberts Filter on gray image (Mx)')
plt.axis('off')
plt.show()
```

Roberts Filter on gray image (Mx)

In [24]:

```
w = np.array([[0,1],[-1,0]])
g = conv2(image1,w,0)
plt.imshow(g)
plt.title('Roberts Filter on color image (Mx)')
plt.axis('off')
plt.show()
```

Roberts Filter on color image (Mx)

In [25]:

```
w = np.array([[1,0],[0,-1]])
g = conv2(image1,w,0)
plt.imshow(g)
plt.title('Roberts Filter on color image (My)')
plt.axis('off')
plt.show()
```

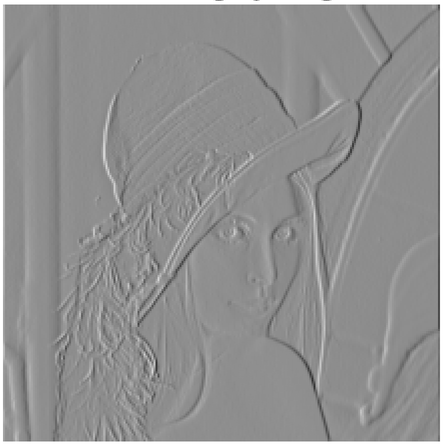
Roberts Filter on color image (My)

Sobel Filter

In [26]:

```
w = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
g = conv2(gray_image,w,0)
plt.imshow(g, cmap = 'gray')
plt.title('Sobel Filter on gray image (Mx)')
plt.axis('off')
plt.show()
```

Sobel Filter on gray image (Mx)



In [27]:

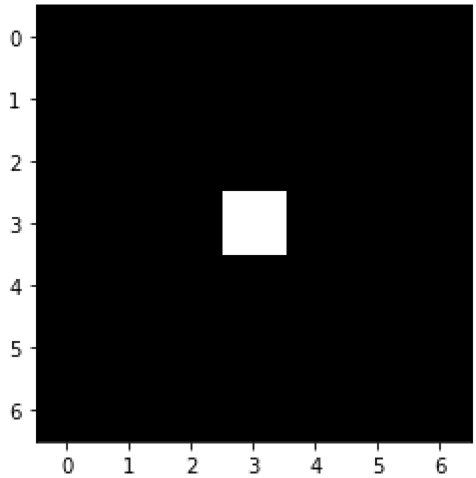
```
w = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
g = conv2(gray_image,w,1)
plt.imshow(g, cmap = 'gray')
plt.title('Sobel Filter on gray image (My)')
plt.axis('off')
plt.show()
```

Q.1 b)

Visualizing the slice of the impulse image

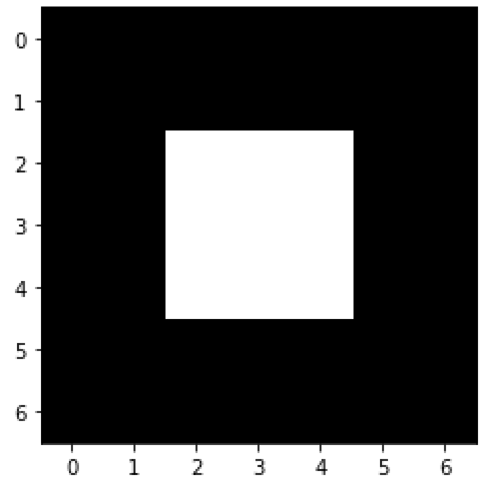
In [28]:

```
impulse_image = np.zeros((1024,1024))
impulse_image[512,512] = 1
plt.imshow(impulse_image[509:516,509:516], cmap = 'gray')
plt.show()
```



In [29]:

```
kernel = 1/9*np.ones((3,3))
impulse_conv = conv2(impulse_image, kernel,0)
impulse_conv[509:516,509:516]
plt.imshow(impulse_conv[509:516,509:516], cmap = 'gray')
plt.show()
```



In [30]:

```
print(impulse_conv[510:515,510:515])
```

```
[[0.         0.         0.         0.         0.        ]
 [0.         0.11111111 0.11111111 0.11111111 0.        ]
 [0.         0.11111111 0.11111111 0.11111111 0.        ]
 [0.         0.11111111 0.11111111 0.11111111 0.        ]
 [0.         0.         0.         0.         0.        ]]
```

Answer:

The average filter will average out the neighbouring pixel values and the impulse response location and this shows the convolution being performed. Passing the impulse through any function should give the output of the function at the location of that impulse. Eg. The averaging filter in our case 1/9[[1,1,1],[1,1,1],[1,1,1]] will give 1/9 at the location of the impulse.

Q2

Q2 a)

Implementing and testing 2D FFT

In [1]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

%matplotlib inline
```

In [2]:

```
image = cv2.imread("lena.png")
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray_image.shape
```

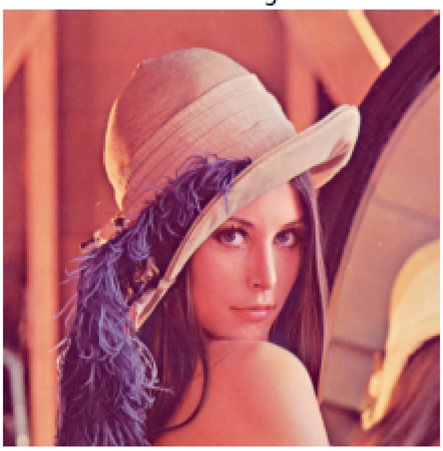
Out[2]:

```
(440, 440)
```

In [3]:

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
plt.axis('off')
plt.title('Color Image')
plt.show()
```

Color Image



In [4]:

```
plt.imshow(gray_image, cmap = "gray")
plt.axis('off')
plt.title('Gray Image')
plt.show()
```

Gray Image

scaling down the pixel values in the lena image to the range - [0,1]

In [5]:

```
def scaling_image(input_image):

    scaled_input_image = np.zeros((input_image.shape[0],input_image.shape[1]),dtype = 

    for row in range(input_image.shape[0]):

        for col in range(input_image.shape[1]):

            scaled_input_image[row,col] = input_image[row,col].astype('float32')/255.

    return scaled_input_image
```

In [6]:

```
scaled_image = scaling_image(gray_image)
scaled_image
```

Out[6]:

```
array([[0.63529414, 0.63529414, 0.6313726 , ..., 0.6745098 , 0.627451  ,
        0.5137255 ],
       [0.63529414, 0.63529414, 0.6313726 , ..., 0.6745098 , 0.627451  ,
        0.5137255 ],
       [0.63529414, 0.63529414, 0.6313726 , ..., 0.67058825, 0.61960787,
        0.50980395],
       ...,
       [0.16862746, 0.1764706 , 0.19215687, ..., 0.40784314, 0.39607844,
        0.38039216],
       [0.16862746, 0.18431373, 0.20392157, ..., 0.4       , 0.40784314,
        0.4117647 ],
       [0.17254902, 0.18431373, 0.21176471, ..., 0.39607844, 0.4117647 ,
        0.42352942]], dtype=float32)
```

In [7]:

```
gray_image
```

Out[7]:

```
array([[162, 162, 161, ..., 172, 160, 131],
       [162, 162, 161, ..., 172, 160, 131],
       [162, 162, 161, ..., 171, 158, 130],
       ...,
       [ 43,  45,  49, ..., 104, 101,  97],
       [ 43,  47,  52, ..., 102, 104, 105],
       [ 44,  47,  54, ..., 101, 105, 108]], dtype=uint8)
```

DFT2 function

In [8]:

```
def DFT2(input_image):
    fft1_image = np.zeros((input_image.shape[0], input_image.shape[1]), dtype = 'comple
    fft2_image = np.zeros((input_image.shape[0], input_image.shape[1]), dtype = 'comple
    for row in range(input_image.shape[0]):

        fft1_image[row, : ] = np.fft.fft(input_image[row ,: ])

    for col in range(fft1_image.shape[1]):

        fft2_image[:, col] = np.fft.fft(fft1_image[:, col])

    return fft2_image
```

In [9]:

```
# scaled_image(passed in the dft2 function below) is the scaled down image of the gray
f = DFT2(scaled_image)
f
```

Out[9]:

```
array([[ 9.41791595e+04   +0.j          , -1.32605217e+03+6928.26008459j,
         4.45443677e+03-3279.87446605j, ...,
        -6.91412729e+02-1838.48699722j,  4.45443677e+03+3279.87446605j,
        -1.32605217e+03-6928.26008459j],
       [-8.37846430e+01-3545.76435384j, -4.60583848e+03+4606.16030199j,
        -3.20398722e+03 +217.75357611j, ...,
         1.89208271e+03+2351.68299707j, -1.02786249e+03+1296.80102976j,
         1.02025131e+02+2741.4323825j ],
       [-1.30229192e+03 -475.37879051j, -7.95709409e+02 -387.10940328j,
         1.93767512e+03-1311.96666989j, ...,
         1.02391899e+03+2684.36743376j, -2.09555156e+03 -509.04877382j,
         2.37833145e+03 -182.71618199j],
       ...,
       [ 1.49437918e+03 -627.39367783j,  2.13376237e+03+2046.8179341j ,
        -1.52536299e+03 -580.14642037j, ...,
         2.87895939e+03 +316.70306245j,  3.76828573e+02-1965.96863635j,
        -1.56015196e+02+1749.10240541j],
       [-1.30229192e+03 +475.37879051j,  2.37833145e+03 +182.71618199j,
        -2.09555156e+03 +509.04877382j, ...,
        -1.50090182e+03-2181.31083998j,  1.93767512e+03+1311.96666989j,
        -7.95709409e+02 +387.10940328j],
       [-8.37846430e+01+3545.76435384j,  1.02025131e+02-2741.4323825j ,
        -1.02786249e+03-1296.80102976j, ...,
        -1.48190256e+03+5509.44338343j, -3.20398722e+03 -217.75357611j,
        -4.60583848e+03-4606.16030199j]])
```

Spectrum of the image

In [10]:

```
def spectrum(input_image):
    result = np.sqrt(np.real(input_image)**2+np.imag(input_image)**2)
    return result
```

In [11]:

```
spectrum_lena = spectrum(f)
spectrum_lena
```

Out[11]:

```
array([[94179.15949942,  7054.02028248,  5531.68902172, ...,
         1964.20111018,  5531.68902172,  7054.02028248],
       [ 3546.75411036,  6513.86681256,  3211.37832252, ...,
         3018.34224132,  1654.74898476,  2743.33020891],
       [ 1386.34384168,   884.87691467,  2340.05157947, ...,
         2873.01907695,  2156.4941446 ,  2385.33974633],
       ...,
       [ 1620.73809423,  2956.75590758,  1631.96266168, ...,
         2896.32663543,  2001.75734113,  1756.04668675],
       [ 1386.34384168,  2385.33974633,  2156.4941446 , ...,
         2647.7959258 ,  2340.05157947,   884.87691467],
       [ 3546.75411036,  2743.33020891,  1654.74898476, ...,
         5705.2608703 ,  3211.37832252,  6513.86681256]])
```

In [12]:

```
s = np.log(1+abs(f))
s
```

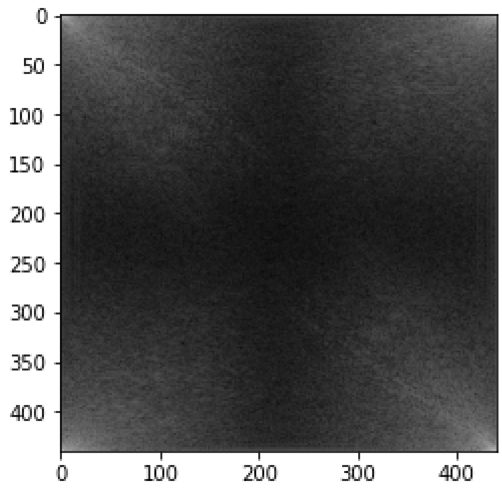
Out[12]:

```
array([[11.45296482,  8.86149474,  8.61842924, ...,  7.58334987,
         8.61842924,  8.86149474],
       [ 8.17407004,  8.78184205,  8.07476685, ...,  8.01279429,
         7.41200874,  7.91729232],
       [ 7.23514629,  6.78657802,  7.7583555 , ...,  7.9634667 ,
         7.67670271,  7.77751598],
       ...,
       [ 7.39125375,  7.99218612,  7.39815123, ...,  7.97154374,
         7.60228018,  7.47138966],
       [ 7.23514629,  7.77751598,  7.67670271, ...,  7.88186045,
         7.7583555 ,  6.78657802],
       [ 8.17407004,  7.91729232,  7.41200874, ...,  8.64931925,
         8.07476685,  8.78184205]])
```

Ploting the spectrum

In [13]:

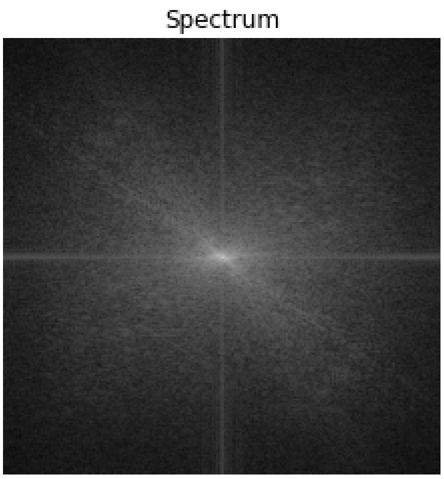
```
plt.imshow(s, cmap = 'gray')
plt.show()
```



Visualizing the shifted spectrum

In [14]:

```
s_shifted = np.fft.fftshift(f)
s_shifted = np.log(1+abs(s_shifted))

plt.imshow(s_shifted, cmap = 'gray')
plt.title('Spectrum')
plt.axis('off')
plt.show()
```



Calculating the phase angle

In [15]:

```
angle = np.arctan2(np.imag(f),np.real(f))
angle
```

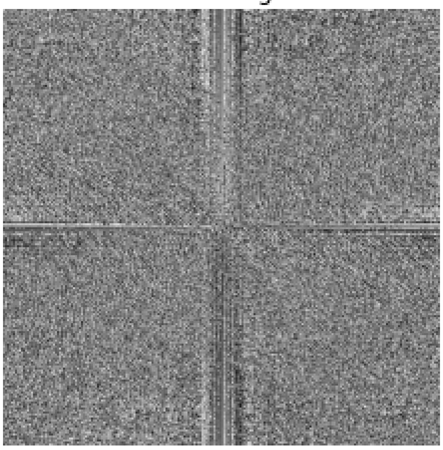
Out[15]:

```
array([[ 0.        ,  1.75990681, -0.63468588, ..., -1.93051091,
         0.63468588, -1.75990681],
       [-1.59442143,  2.35615956,  3.0737337 , ...,  0.89327781,
         2.24101728,  1.5335975 ],
       [-2.79158918, -2.68880653, -0.59517921, ...,  1.20639361,
        -2.90328972, -0.07667475],
       ...,
       [-0.39748829,  0.76460394, -2.77815439, ...,  0.10956554,
        -1.38141749,  1.65975818],
       [ 2.79158918,  0.07667475,  2.90328972, ..., -2.17347289,
         0.59517921,  2.68880653],
       [ 1.59442143, -1.5335975 , -2.24101728, ...,  1.83355257,
        -3.0737337 , -2.35615956]])
```

In [16]:

```
angle = np.fft.fftshift(angle)
plt.imshow(angle, cmap = 'gray')
plt.axis('off')
plt.title('Phase angle')
plt.show()
```

Phase angle



Q2 b)

IDFT2 function

In [17]:

```
def IDFT2(dft2_image):
    swapped = np.imag(dft2_image) + complex('j') * np.real(dft2_image)
    swapped_dft2_image = DFT2(swapped)
    idft2_image = np.imag(swapped_dft2_image) + complex('j') * np.real(swapped_dft2_ima

    return idft2_image/(idft2_image.shape[0]*idft2_image.shape[1])
```

In [18]:

```
g = IDFT2(f)
g
g_real = np.real(g)
```

In [19]:

```
np.imag(g)
```

Out[19]:

```
array([[-1.40468696e-17,  1.82816020e-17,  3.26409906e-17, ...,
        -1.96882383e-17, -1.66316024e-17, -6.37847972e-18],
       [-5.31418559e-18,  3.86877384e-18, -6.78568986e-18, ...,
        -3.37361671e-17, -4.02593538e-17, -3.19379713e-17],
       [-3.23976634e-17,  1.13817035e-17, -1.49609685e-17, ...,
        -8.48702578e-17, -2.75686755e-17,  2.25423713e-17],
       ...,
       [-3.00014228e-17, -1.65967360e-17, -4.16570256e-17, ...,
        -5.75406449e-17, -3.94259719e-17, -1.29446169e-17],
       [-7.50018658e-17, -3.13495481e-17, -4.42679008e-18, ...,
         1.14749351e-17, -3.14773350e-17, -1.75458419e-17],
       [-5.34013569e-17, -4.81203418e-17, -9.69945350e-18, ...,
         3.76586897e-17, -8.79412950e-18, -9.47196273e-18]])
```

In [20]:

```
d = scaled_image - g_real
d
```

Out[20]:

```
array([[ 0.00000000e+00, -1.11022302e-16, -1.11022302e-16, ...,
        -1.11022302e-16, -2.22044605e-16, -2.22044605e-16],
       [ 2.22044605e-16,  0.00000000e+00,  0.00000000e+00, ...,
         1.11022302e-16,  0.00000000e+00,  0.00000000e+00],
       [-1.11022302e-16,  0.00000000e+00,  0.00000000e+00, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       ...,
       [-5.55111512e-17, -1.66533454e-16, -2.77555756e-17, ...,
        -1.66533454e-16, -5.55111512e-17, -1.11022302e-16],
       [-5.55111512e-17, -5.55111512e-17, -2.77555756e-17, ...,
        -1.66533454e-16, -1.66533454e-16, -2.77555756e-16],
       [ 0.00000000e+00, -8.32667268e-17, -2.77555756e-17, ...,
        -1.66533454e-16, -5.55111512e-17, -5.55111512e-17]])
```

In [21]:

```
g_real
```

Out[21]:

```
array([[0.63529414, 0.63529414, 0.63137257, ..., 0.67450982, 0.627451  ,
        0.51372552],
       [0.63529414, 0.63529414, 0.63137257, ..., 0.67450982, 0.627451  ,
        0.51372552],
       [0.63529414, 0.63529414, 0.63137257, ..., 0.67058825, 0.61960787,
        0.50980395],
       ...,
       [0.16862746, 0.17647059, 0.19215687, ..., 0.40784314, 0.39607844,
        0.38039216],
       [0.16862746, 0.18431373, 0.20392157, ..., 0.40000001, 0.40784314,
        0.41176471],
       [0.17254902, 0.18431373, 0.21176471, ..., 0.39607844, 0.41176471,
        0.42352942]])
```

In [22]:

```
d2 = gray_image - (g_real * 255).astype(np.uint8)
```

In [23]:

```
d2
```

Out[23]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

Plot of the difference between the original image and the inverse fourier transformed image

In [24]:

```
plt.imshow(d2, cmap = "gray")
plt.axis('off')
plt.show()
```

In [25]:

```
# got a black image which means there is no difference
```