Lab1

1. Abstract

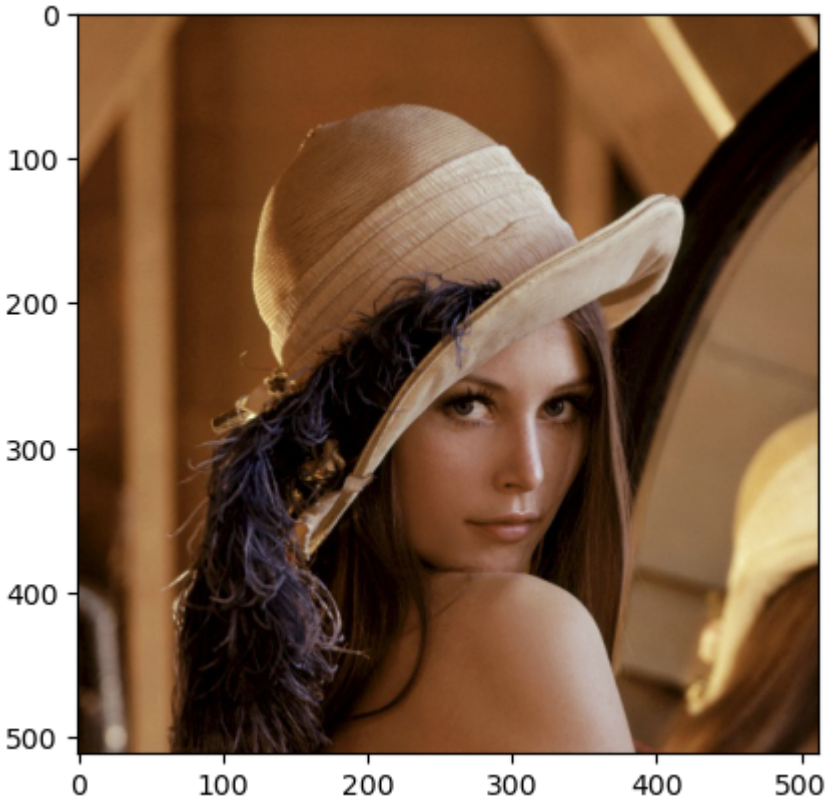
In this lab report, we learn how to import an image and analyze its features like shape, size, channels and how to manipulate them. We explored operations like creating and applying various masks to an image. Additionally, we also explore Fast Fourier Transform (fft) method, and how to apply it correctly when dealing with optics problems. This method was used to visualize the images and their phase in frequency domain.

2. Importing and analyzing an Image

In [1]:
```python
# Importing necessary packages
import numpy as np
import matplotlib.pyplot as plt
import imageio.v3 as io
```

In [2]:
```python
im = io.imread("Lena512-Copy1.jpg") # Load the image
print(f"Image shape : {im.shape}") # Prints the shape of the image

# Display the image
plt.figure()
plt.imshow(im)
plt.show()
```

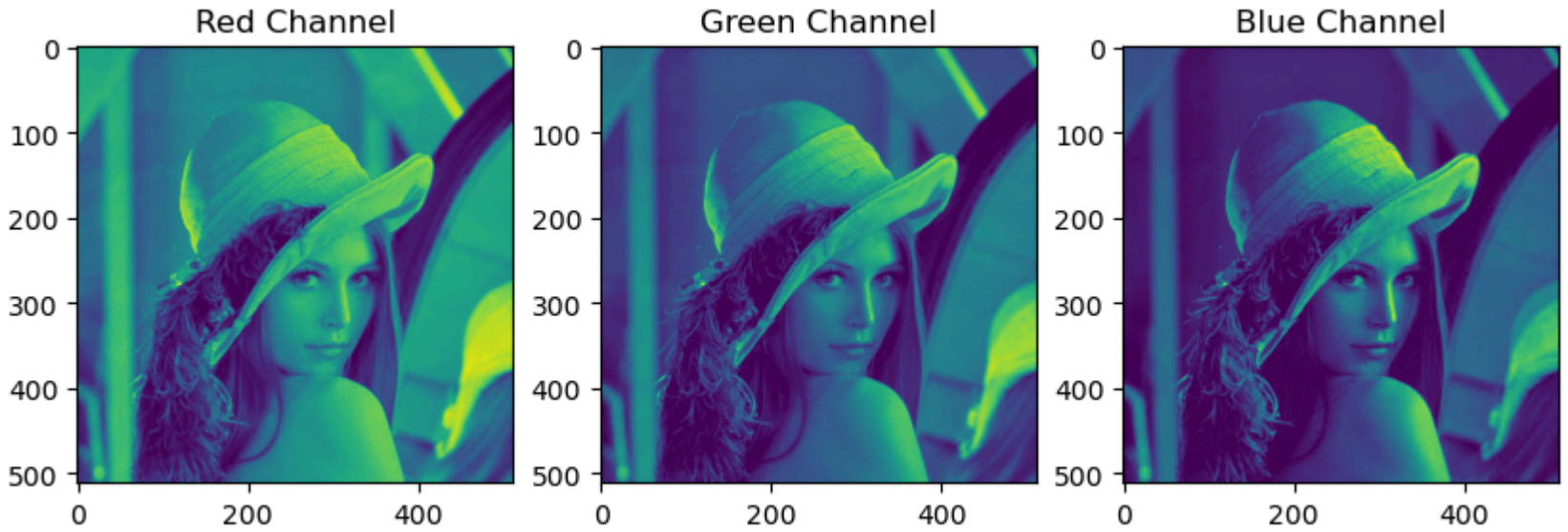
```
Image shape : (512, 512, 3)
```



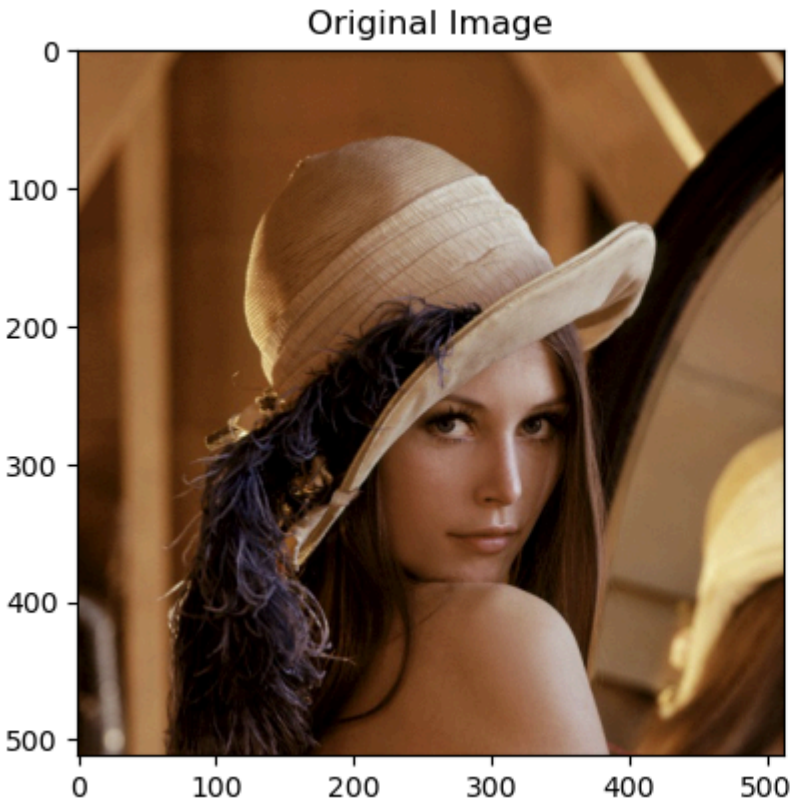
- The dimension of the images (512,512,3) is the height, width and the color channels of the image.
- The three color channels are Red, Green and Blue respectively.

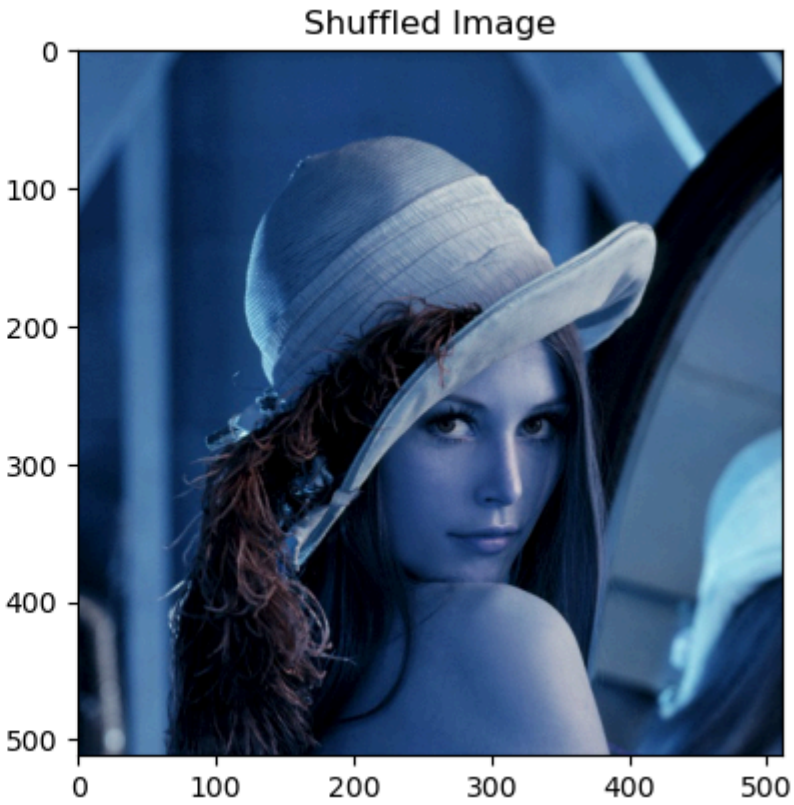
In [3]:
```
R = im[:,:,0] # Stores the red channel as R
G = im[:,:,1] # Stores the green channel as G
B = im[:,:,2] # Stores the blue channel as B

plt.figure(figsize = (10,5))
plt.subplot(1,3,1)
plt.imshow(R) # Shows the red channel from RGB
plt.title("Red Channel")
plt.subplot(1,3,2)
plt.imshow(G)  # Shows the green channel from RGB
```

```
plt.title("Green Channel")
plt.subplot(1,3,3)
plt.imshow(B) # Shows the blue channel from RGB
plt.title("Blue Channel")
plt.show()
```



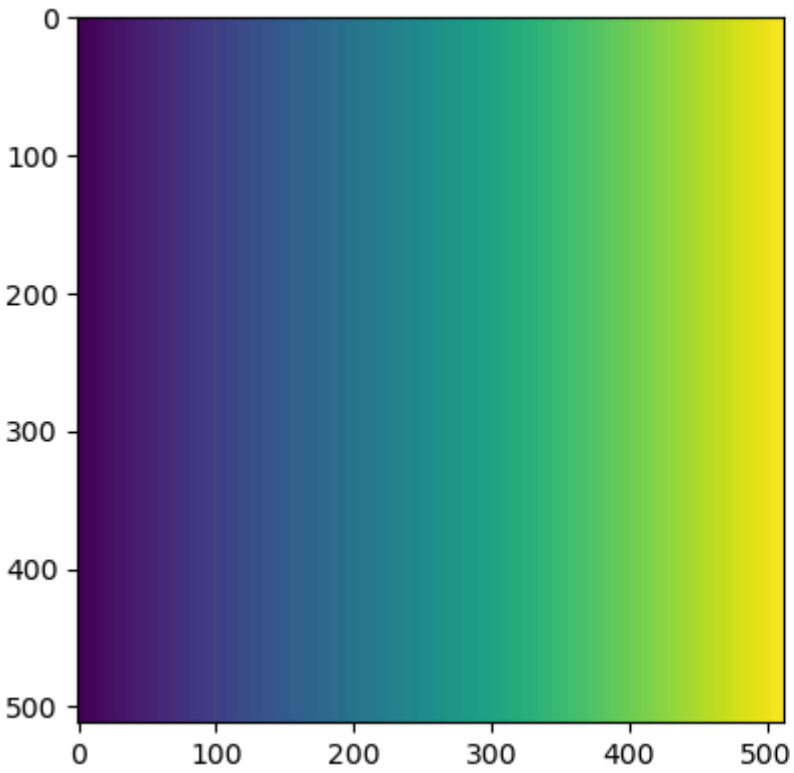
In [4]:
```
# Shuffling the channels of the original image
custom_im = np.zeros((512,512,3)) # Makes a tensor of zeros with shape (512,512,3)
custom_im[:,:,0] = B
custom_im[:,:,1] = G
custom_im[:,:,2] = R
custom_im = custom_im/np.max(custom_im) # Normalize the values from 0 to 1

# Display the original and shuffled image
plt.figure(figsize = (10,5))
plt.subplot(1,2,1)
plt.imshow(im)
plt.title("Original Image")
plt.subplot(1,2,2)
plt.imshow(custom_im)
plt.title("Shuffled Image")
plt.show()
```

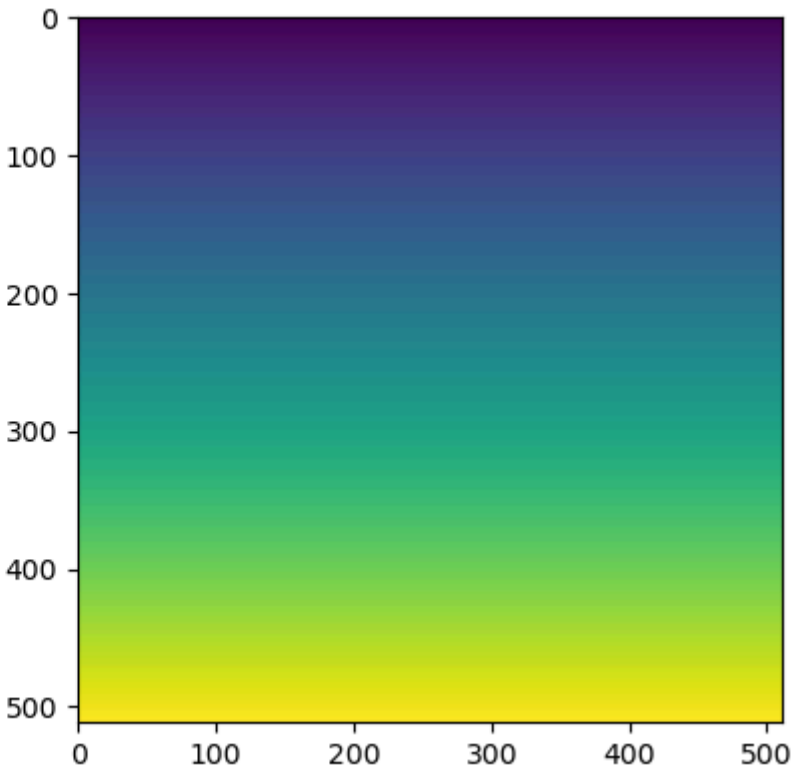




3. Creating a Meshgrid

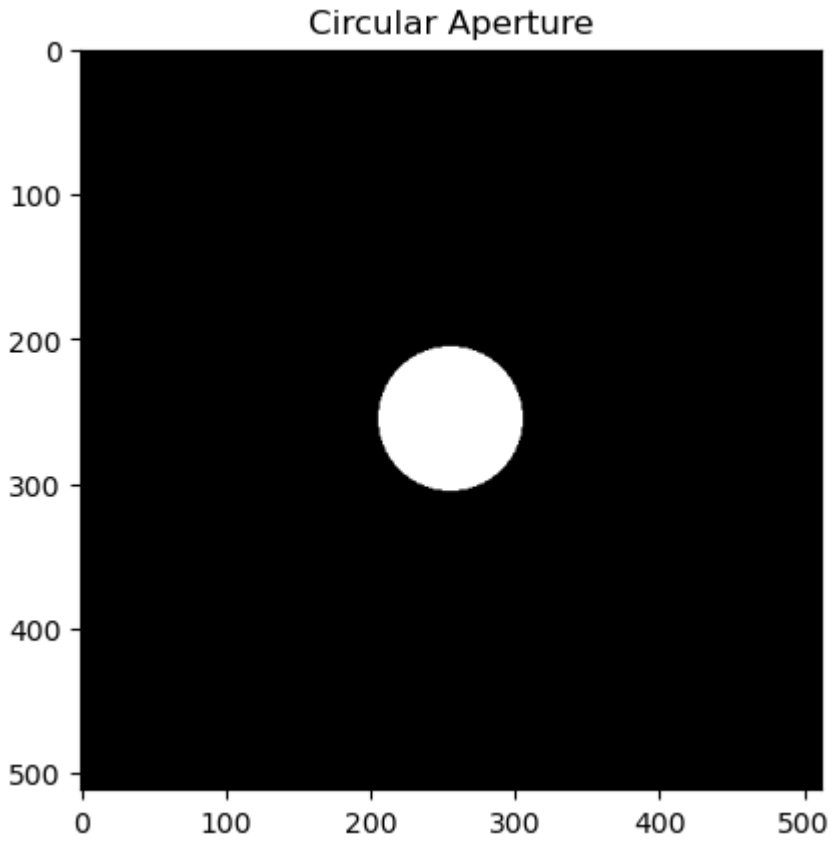
In [5]:
```
rows = np.linspace(-256, 256, 512, endpoint = True) # endpoint = True will include the endpoints (e.g. 256)
column = np.linspace(-256, 256, 512, endpoint = True)

x, y = np.meshgrid(rows, column) # Creates a meshgrid with vectors rows and column

plt.figure(figsize = (10, 5))
plt.subplot(1,2,1)
plt.imshow(x)
plt.subplot(1,2,2)
plt.imshow(y)
plt.show()
```





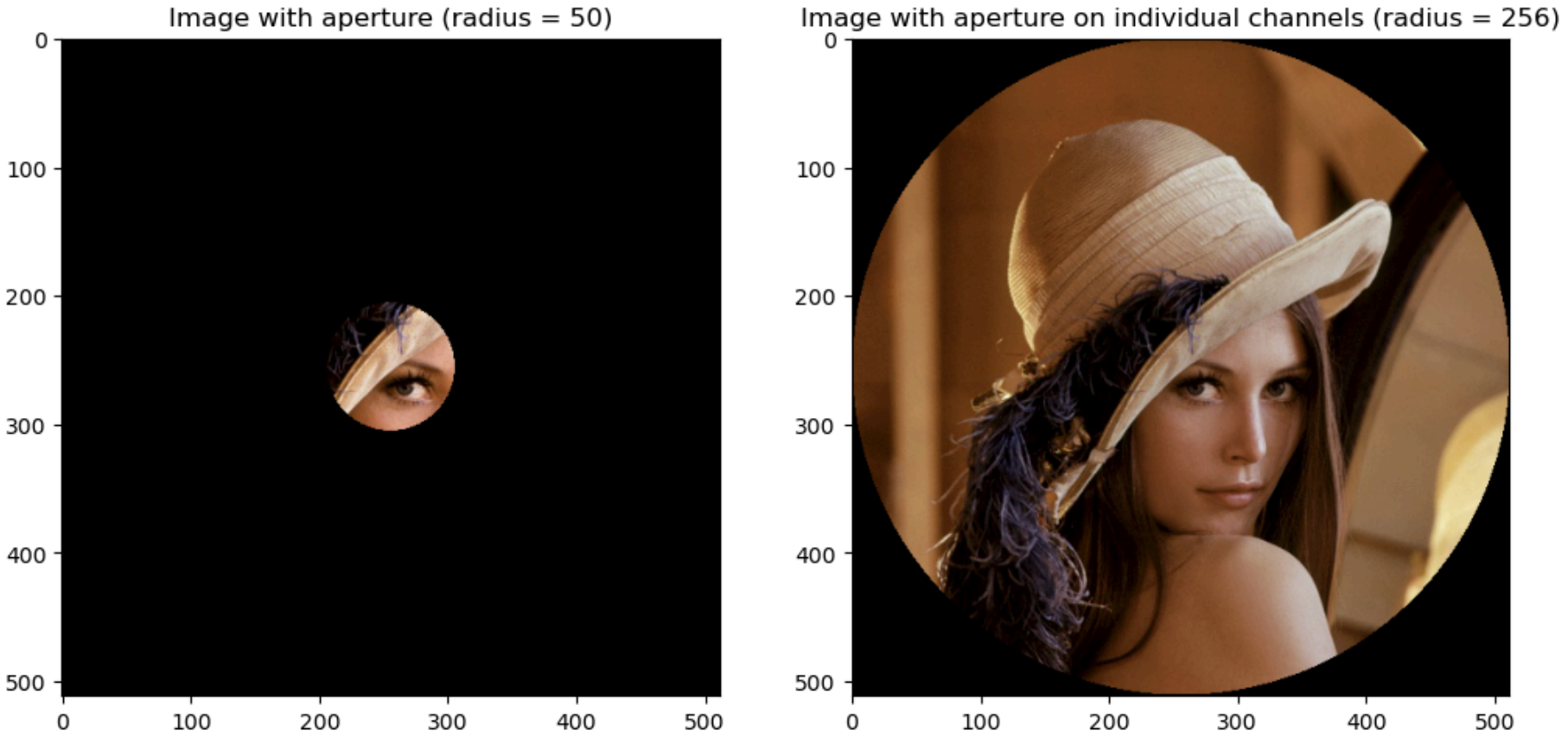
Creating an aperture

In [6]:
```
# Making a circular aperture
circ = np.zeros((512, 512, 3)) # Creates a square grid of all zero values
circ[x**2 + y**2 <= 2500] = 1 # Defines a circular region at the center with radius 50 pixels

# Display the circular aperture
plt.figure()
plt.imshow(circ)
plt.title("Circular Aperture")
plt.show()
```



In [7]:
```
# Applying the aperture to the image.

N = 512
im_apertured = im*circ # Applies circular aperture on the image
im_apertured = im_apertured/np.max(im_apertured)

# Applying the aperture to each channel
circ1 = np.zeros((512, 512))
circ1[x**2 + y**2 <= (N/2)**2] = 1 # Defines a circular region

im_ch = np.zeros((512,512,3))
im_ch[:,:,0] = R*circ1 # Applies circular aperture on Red channel
im_ch[:,:,1] = G*circ1 # Applies circular aperture on Blue channel
im_ch[:,:,2] = B*circ1 # Applies circular aperture on Green channel
```

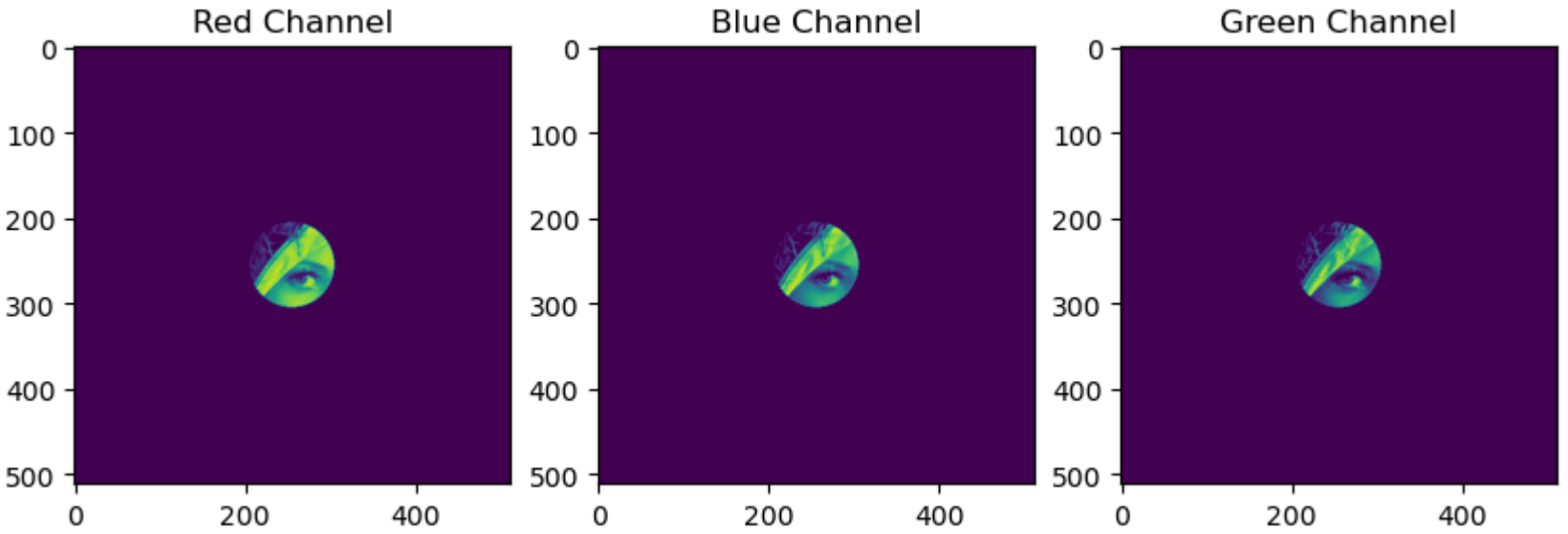
```
im_ch = im_ch/np.max(im_ch)

# Display the results
plt.figure(figsize = (12,6))
plt.subplot(1,2,1)
plt.imshow(im_apertured) # Image multiplied with the circular aperture (r = 50) directly
plt.title("Image with aperture (radius = 50)")
plt.subplot(1,2,2)
plt.imshow(im_ch) # Individual channels of the image multiplied with the circular aperture (r = 256)
plt.title("Image with aperture on individual channels (radius = 256)")
plt.show()
```



Note - A very common error when applying the aperture to the image is a shape error i.e. the shape of aperture and image does not match. Its because the * operator when used with matrices is an elementwise multiplication of two matrices. In such cases either make sure both the

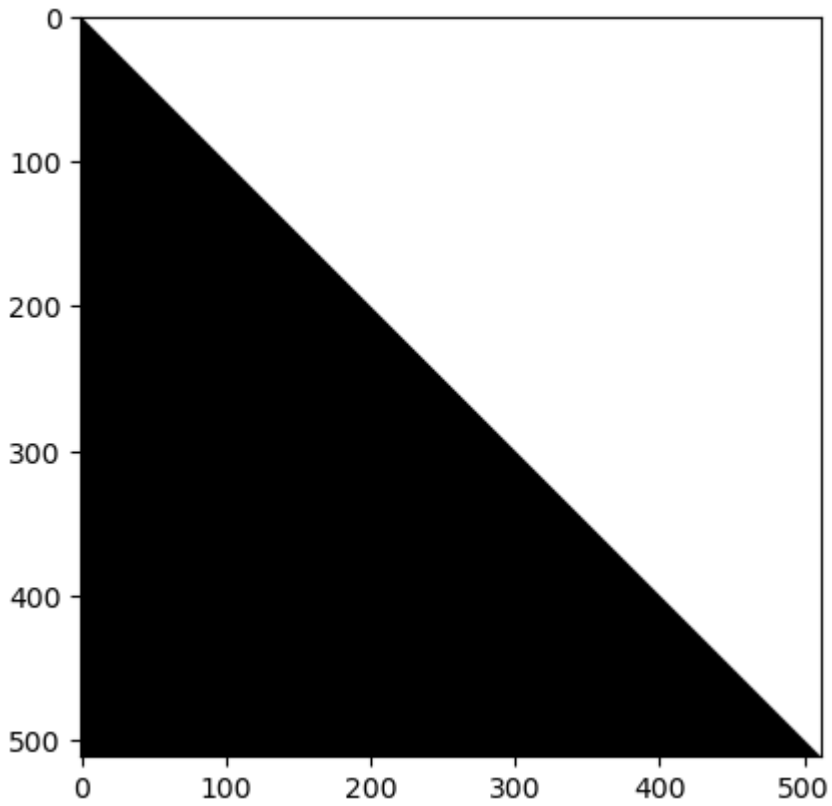
aperture and the image have the same shape or if the aperture is a 2d object you can apply this to individual channels of the image and then display the combined image.

In [8]:
```
# Display individual channels with the aperture
plt.figure(figsize=(10,5))
plt.subplot(1,3,1)
plt.imshow(im_apertured[:,:,0])
plt.title("Red Channel")
plt.subplot(1,3,2)
plt.imshow(im_apertured[:,:,1])
plt.title("Blue Channel")
plt.subplot(1,3,3)
plt.imshow(im_apertured[:,:,2])
plt.title("Green Channel")
plt.show()
```



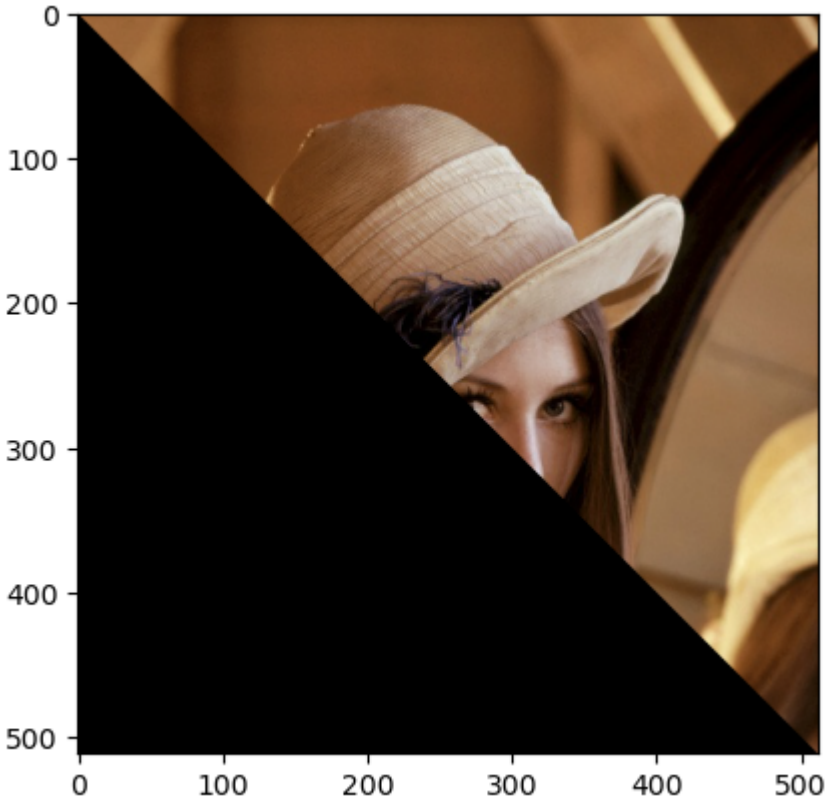
In [9]:
```
# Making a diagonal aperture
grid = np.zeros((512,512, 3))
grid[x >= y] = 1

plt.figure()
```

```
plt.imshow(grid)
plt.show()
```



In [10]:
```
# Applying the diagonal aperture to the image
im_diag = im*grid
im_diag = im_diag/np.max(im_diag)

plt.figure()
plt.imshow(im_diag)
plt.show
```

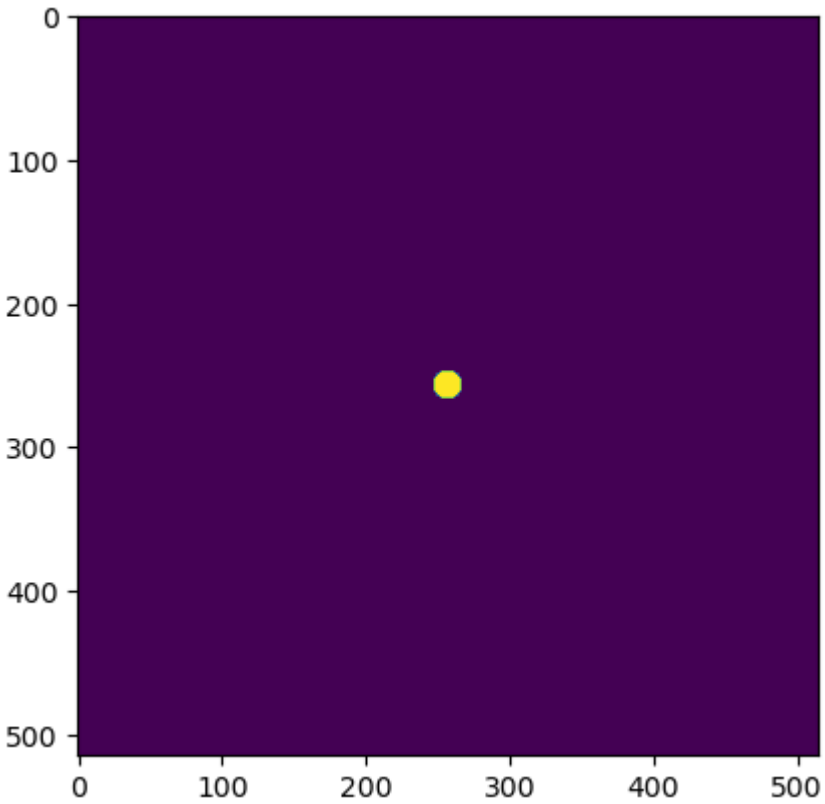
Out[10]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



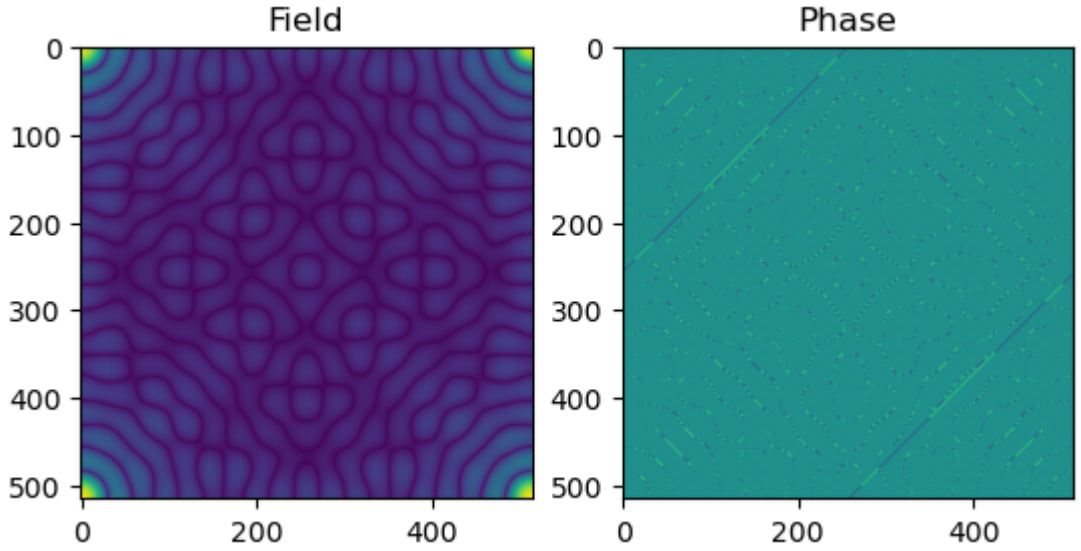
3. Fast Fourier Transform

In [11]:
```
# Defining an odd grid
N = 515

nrow = np.linspace(-N/2, N/2, N, endpoint = True)
ncol = np.linspace(-N/2, N/2, N, endpoint = True)

x1, y1 = np.meshgrid(nrow, ncol)

circ2 = np.zeros((N, N))
circ2[x1**2 + y1**2 <= 100] = 1 # Creates a circular aperture with radius 10 pixels
```

```
plt.figure()
plt.imshow(circ2)
plt.show()
```

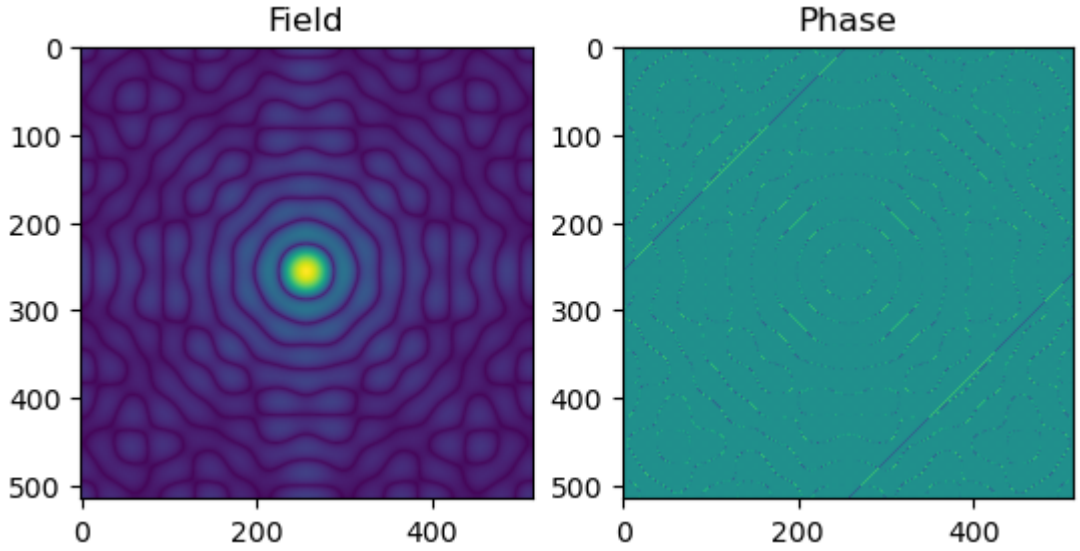


In [12]:
```
# Using only fft2 function
C1 = np.fft.fft2(circ2)
plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.sqrt(np.abs(C1)))
plt.title("Field")
plt.subplot(1,2,2)
plt.imshow(np.angle(np.real(C1)))
plt.title("Phase")
plt.show()
```



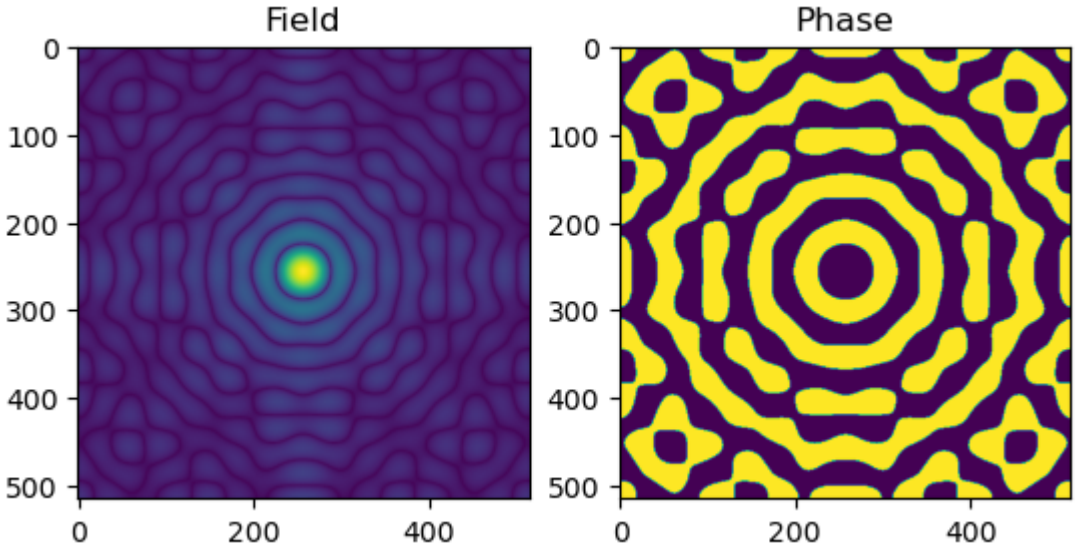
The images show that the fourier transformed field values are jumbled and not in center, and the phase is also not correct.

In [13]:
```
# Using fftshift with fft2
C2 = np.fft.fftshift(np.fft.fft2(circ2))
plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.sqrt(np.abs(C2)))
plt.title("Field")
plt.subplot(1,2,2)
plt.imshow(np.angle(np.real(C2)))
plt.title("Phase")
plt.show()
```



The fourier field seems correct but the phase values are still not as one would expect.

In [14]:
```
# Using fftshift and ifftshift with fft2
C3 = np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(circ2)))
plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.sqrt(np.abs(C3)))
plt.title("Field")
plt.subplot(1,2,2)
plt.imshow(np.angle(np.real(C3)))
plt.title('Phase')
plt.show()
```



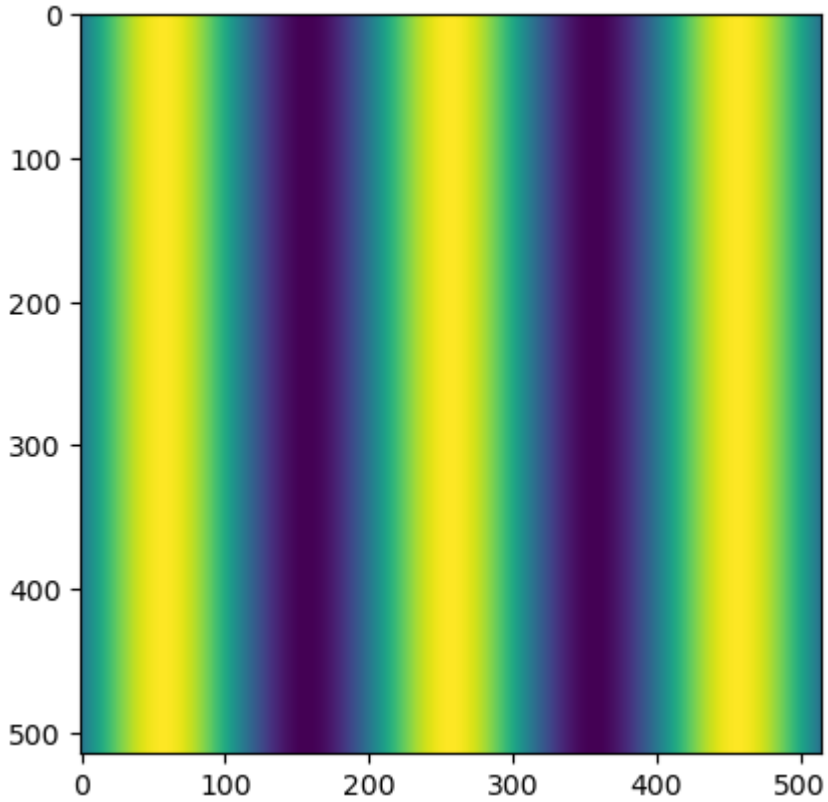
Both the field and phase values seem correct now.

Note - It is apparent from the above results that the correct method to use the fft2 algorithm is using it in the order np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(image))). Python uses the coordinates such that the 0 coordinate is at the top left side but in optics we need the 0 at the optic axis i.e. at the center of the image. This is why the values seem jumbled if we just use fft2 or only fftshift with fft2. The fftshift and ifftshift functions assign the 0 coordinate at the center of image and also distribute the phase in the same manner. This same order is used with inverse fourier transforms using ifft2.

In [15]:
```
print(f" Minimum phase : {np.min(np.angle(np.real(C3)))}, Maximum phase : {np.max(np.angle(np.real(C3)))}")
```

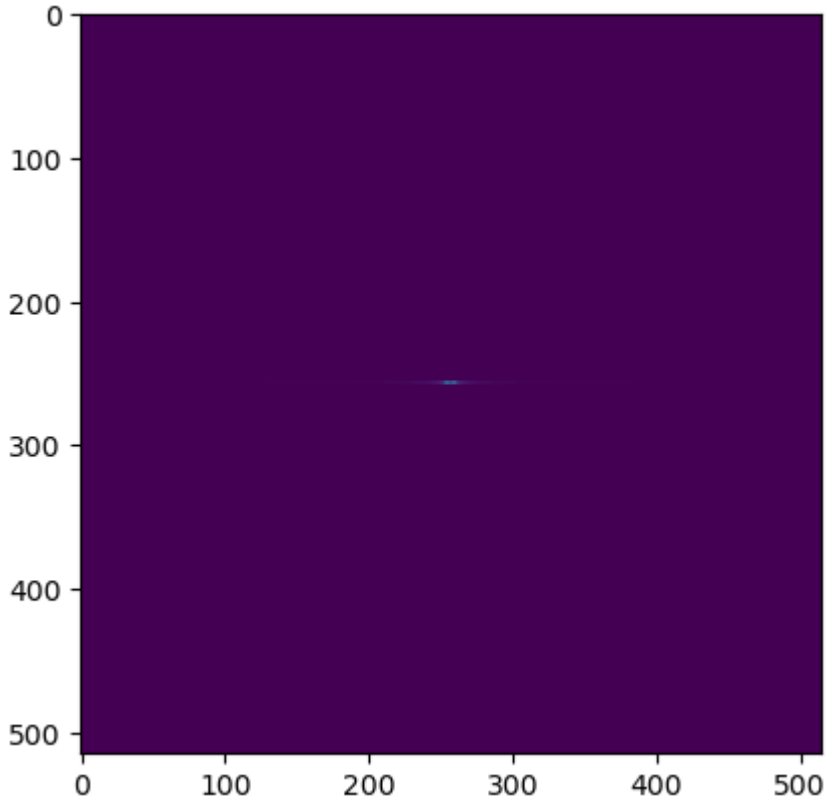
```
Minimum phase : 0.0, Maximum phase : 3.141592653589793
```

In [16]:
```
fx = 0.005
circ3 = np.zeros((515, 515))
circ3[x1**2 + y1**2 <= (N/2)**2] = 1
g = np.cos(2*np.pi*(fx)*x1) # returns a cosine wave with frequencey equals to fx

# Display the cosine function
plt.figure()
plt.imshow(g)
plt.show()
```



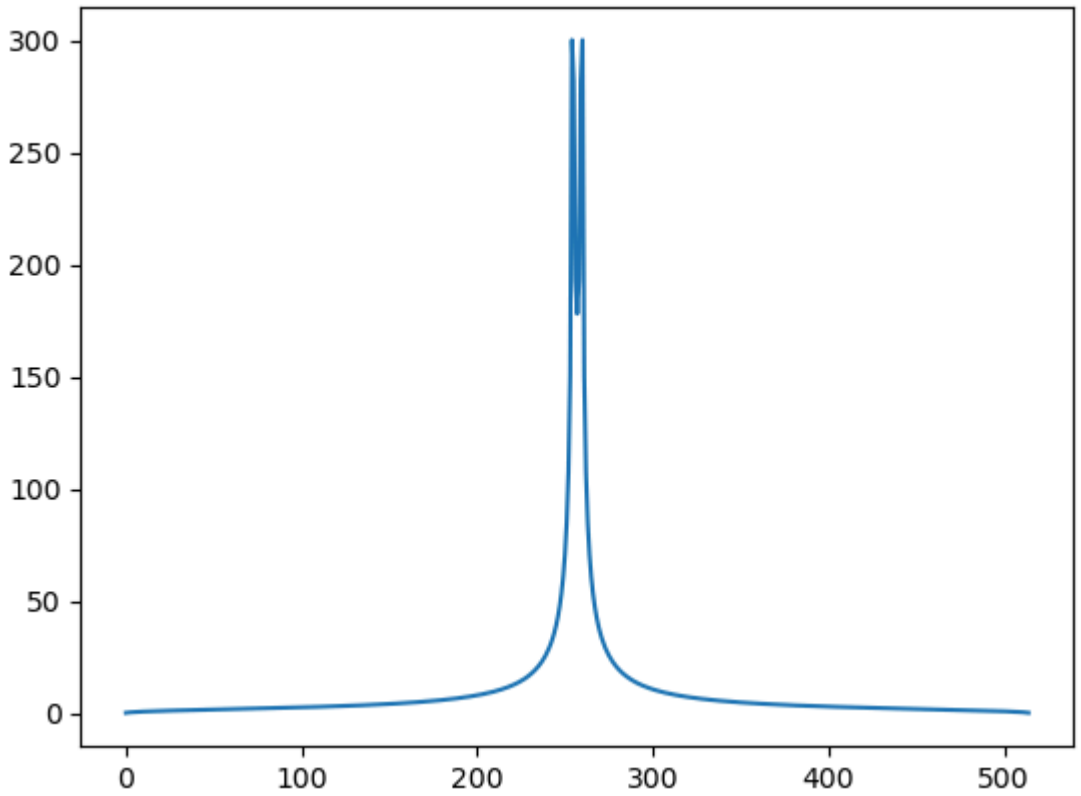
In [17]:
```python
# Defining custom functions for fft2 and ifft2
def ft2(x):
    return np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(x))) # Fourier transform

def ift2(x):
    return np.fft.fftshift(np.fft.ifft2(np.fft.ifftshift(x))) # Inverse Fourier transform
```

In [18]:
```python
G = ft2(g) # Fourier transform the cosine wave

# Display the fourier transform of the cosine
plt.figure()
plt.imshow(np.sqrt(np.abs(G)))
plt.show()
```



In [19]:
```
plt.plot(np.abs(np.sqrt(G[257,:]))) # Plotting the center row where the two peaks are located.
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x1fc91174750>]
```

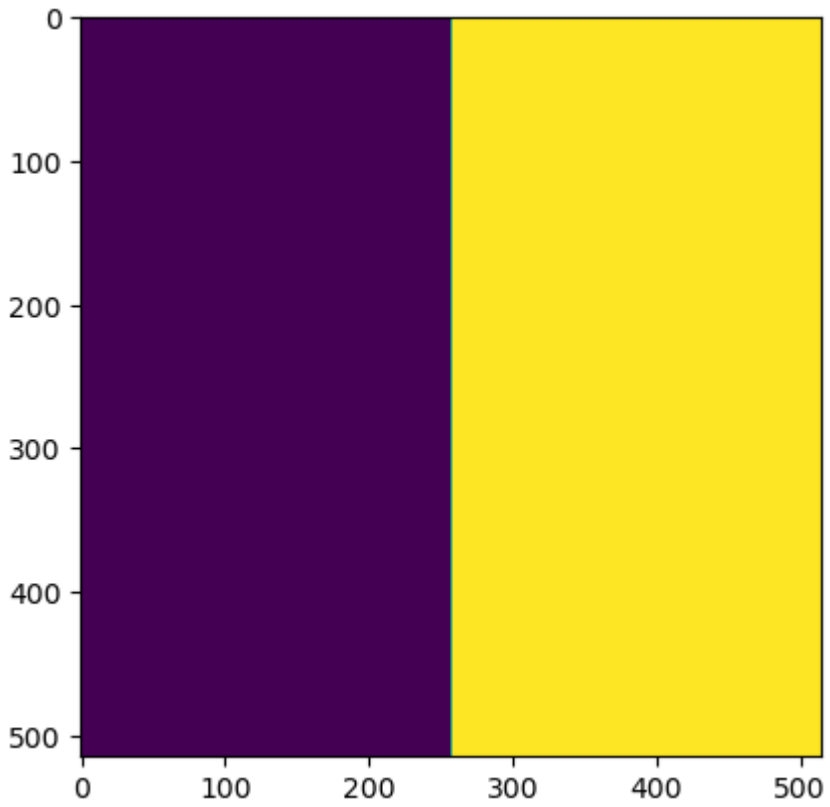


06 August, 2024

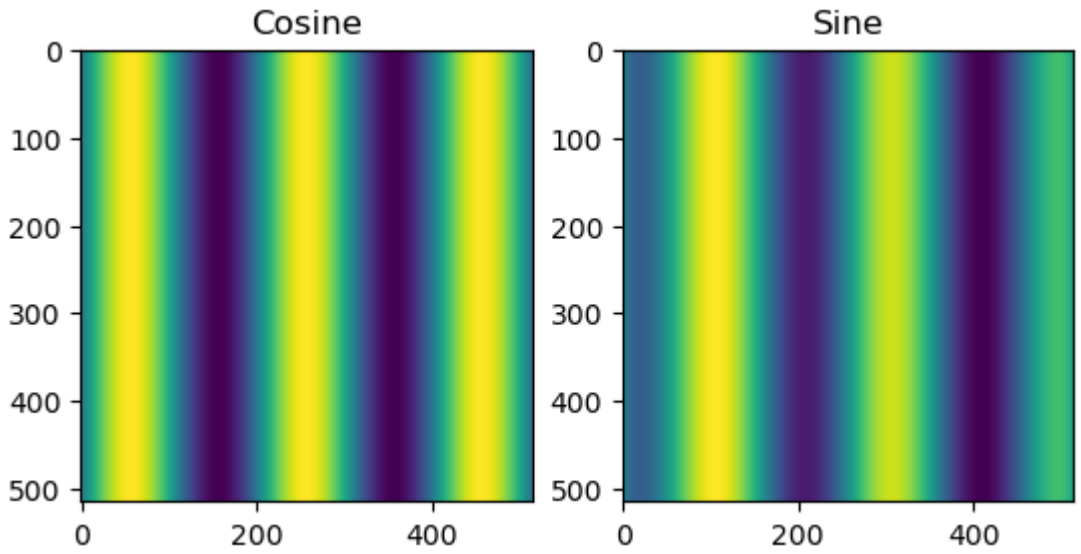
In [20]:
```
# Creating a meshgrid in fourier space
F = nrow/N
FX,FY = np.meshgrid(F,F)
```

In [21]:
```
sgn = np.zeros((515,515))
sgn[FX>0] = 1 # Defines a signum function

sgn_f = 2*sgn - 1

plt.imshow(sgn) # Show the signum grid
G_sgn = G*(-1j*sgn_f) # Apply the sgn on the fourier transform of cosine wave
```

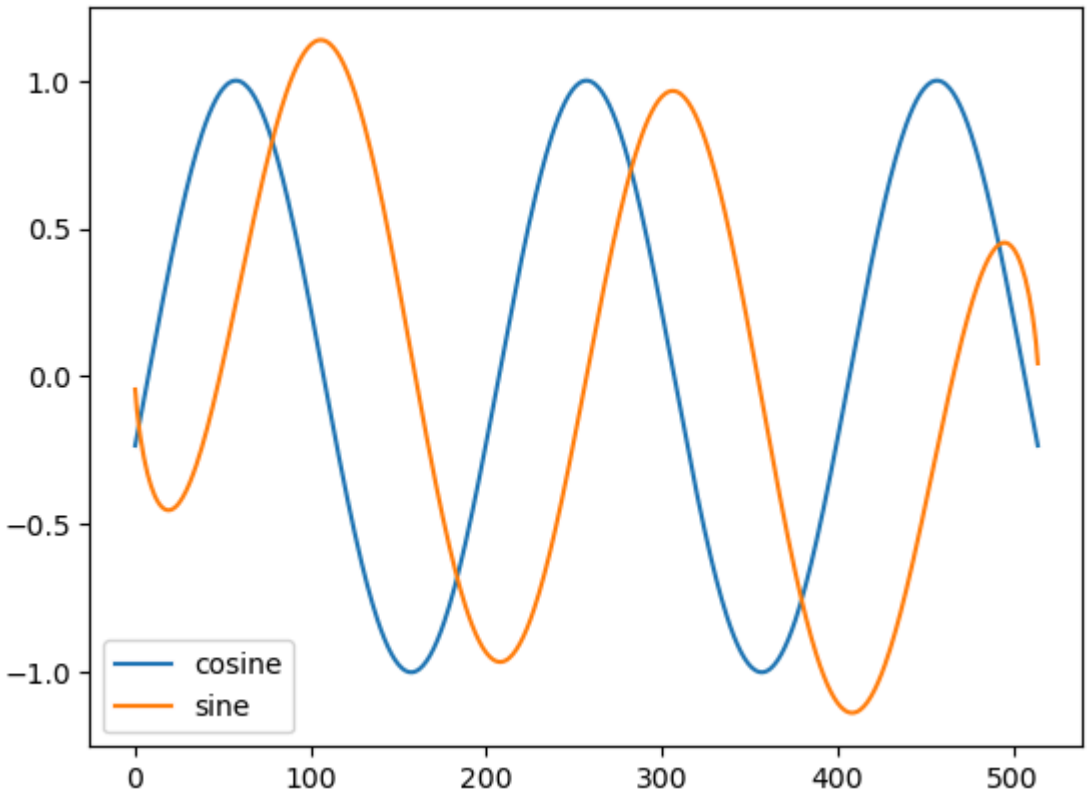
```
g_sgn = ift2(G_sgn)
```



In [22]:
```
# Display the cosine and the hilbert transformed wave.
plt.figure()
plt.subplot(1,2,1)
plt.imshow(g)
plt.title("Cosine")
plt.subplot(1,2,2)
plt.imshow(np.real(g_sgn))
plt.title("Sine")
plt.show()
```



In [23]:
```
# Plotting the line profiles for the two waves
plt.figure()
plt.plot(np.real(g[257,:]), label = "cosine")
plt.plot(np.real(g_sgn[257,:]), label = "sine")
plt.legend()
plt.show()
```

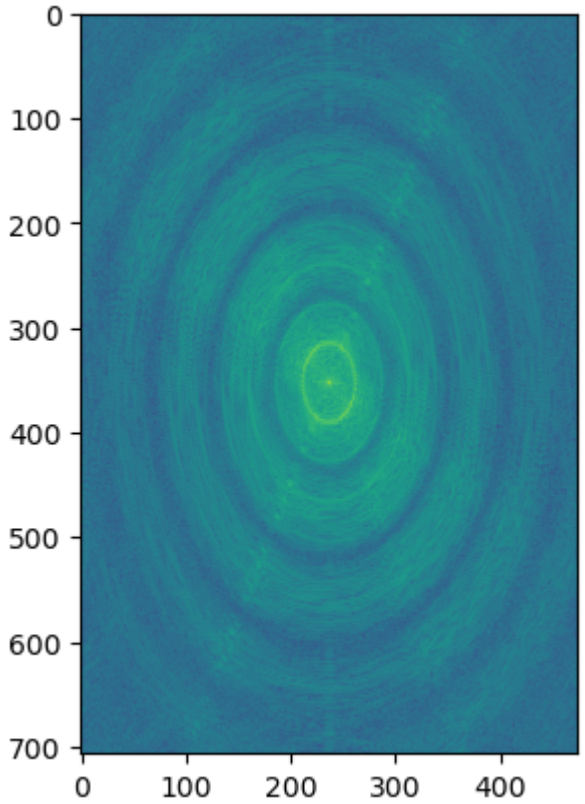


Fourier transforming images

In [24]:
```
from skimage import color
im = io.imread("OIP.jpeg") # Imports an image of a fingerprint
im_R = im[:,:,0]
im_R = im_R/np.max(im_R)
plt.imshow(im, cmap = "gray")
```

Out[24]:
```
<matplotlib.image.AxesImage at 0x1fc8fa5c890>
```

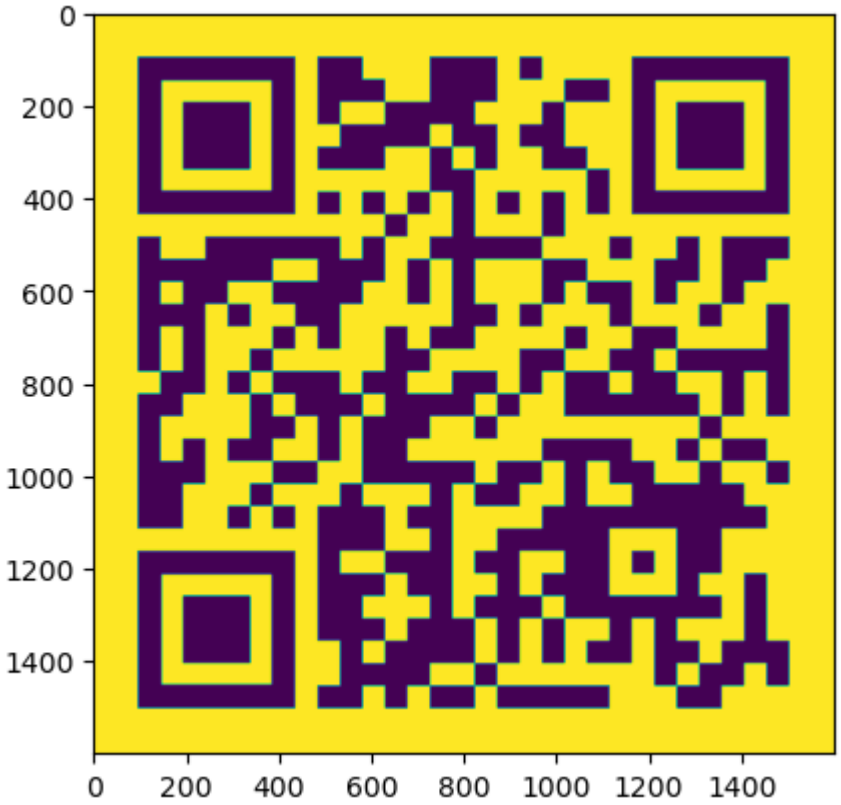
In [25]:
```
im_fft = ft2(im_R)

plt.figure()
plt.imshow(np.log(np.abs(im_fft)))
plt.show()
```



The circle like pattern is because of the repetitive nature of fingerprints.

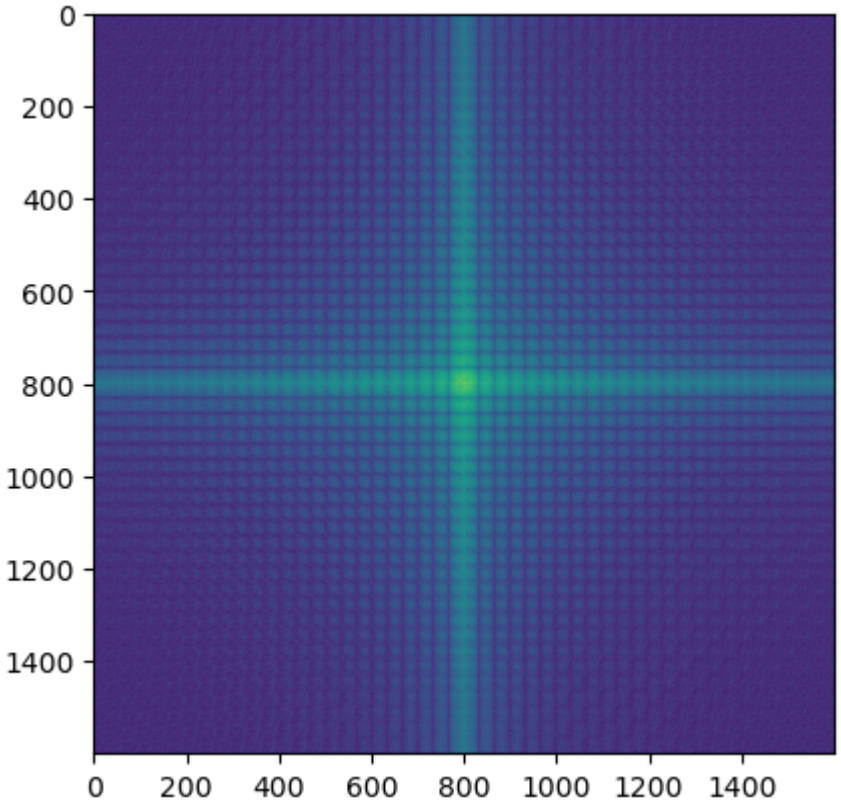
In [26]:
```
# Import an image of a QR code
qr = io.imread("qr-code.png")
qr = color.rgb2gray(qr)
plt.imshow(qr)
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x1fc93e09090>
```



In [27]:
```
qr_fft = ft2(qr)
plt.imshow(np.abs(np.log(qr_fft)))
```

Out[27]: <matplotlib.image.AxesImage at 0x1fc97cc5750>

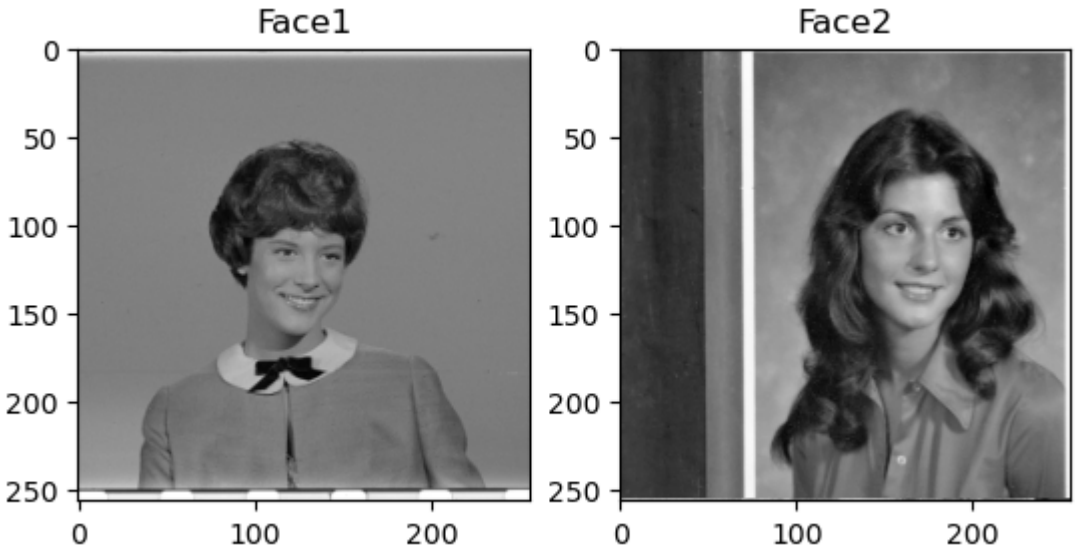


4. Exercise

- Take two grayscale images A and B and take fft of both. Take abs and angle of both in the fourier domain.
- Make two new matrics A1 = abs(fft(A)) * exp(ι * phase(B)), and B1 = abs(fft(B)) * exp(ι * phase(A)).
- Take inverse fft of both A1 and B1 and observe absolute values of both.

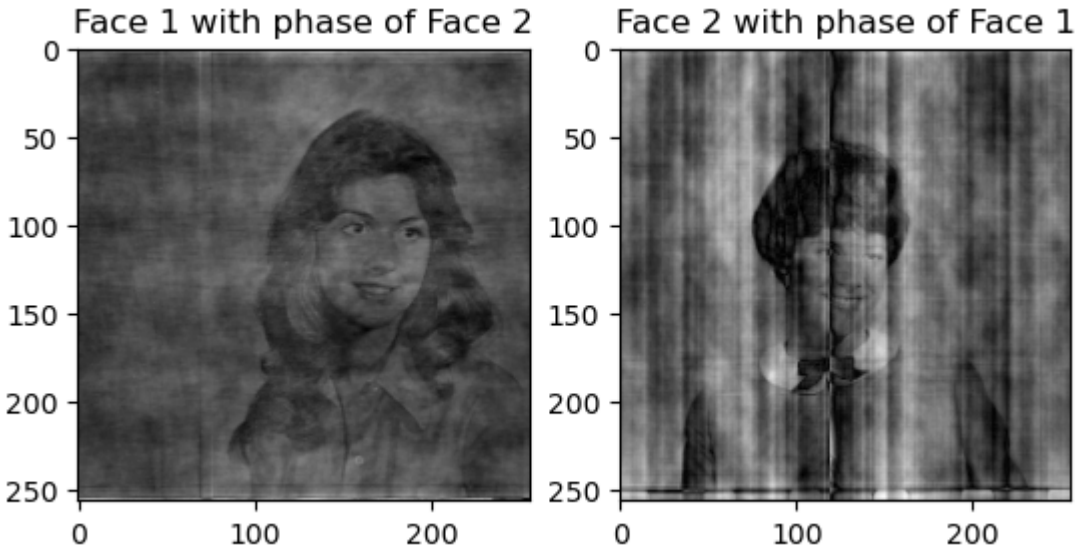
In [28]:
```
from skimage.transform import resize

# Import the face images
face1 = io.imread("4.1.03.tiff")
face2 = io.imread("4.1.04.tiff")
```

```
# Grayscale the images
face1 = color.rgb2gray(face1)
face2 = color.rgb2gray(face2)

# Display the images
plt.figure()
plt.subplot(1,2,1)
plt.imshow(face1, cmap ="gray")
plt.title("Face1")
plt.subplot(1,2,2)
plt.imshow(face2, cmap = "gray")
plt.title("Face2")
plt.show()
```



In [29]:
```
# Fourier transform the two face images
FACE1 = ft2(face1)
FACE2 = ft2(face2)

A1 = np.abs(FACE1)*np.exp(1j*np.angle(FACE2)) # Phase exchanged with the phase of Face 2
#A1 = np.exp(1j*np.angle(FACE2))
B1 = np.abs(FACE2)*np.exp(1j*np.angle(FACE1)) # Phase excahnged with the phase of Face 1
#B1 = np.exp(1j*np.angle(FACE1))
```

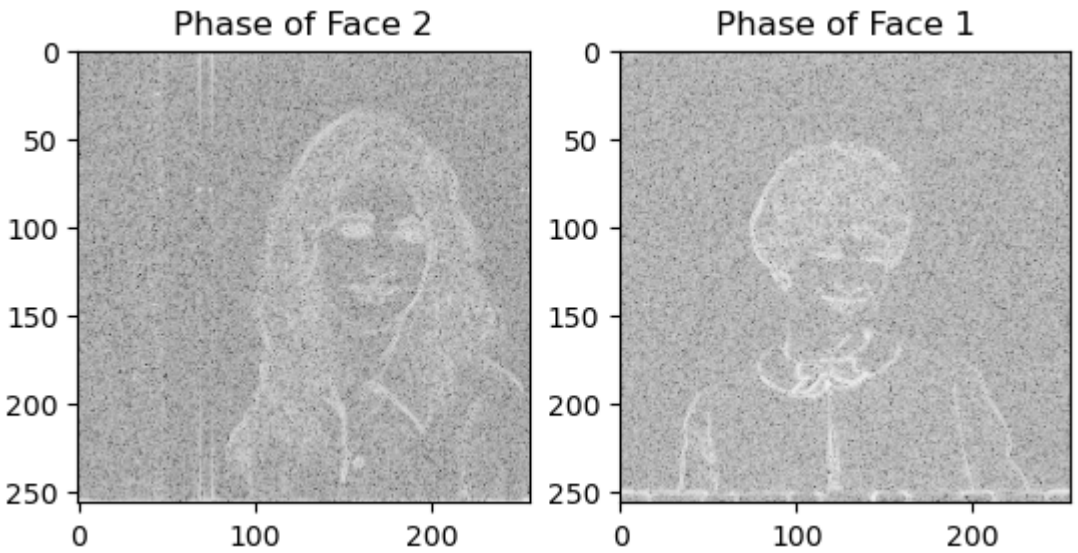
```
# Inverse transform the images with exchanged faces
a1 = ift2(A1)
b1 = ift2(B1)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.abs(a1), cmap ="gray")
plt.title("Face 1 with phase of Face 2")
plt.subplot(1,2,2)
plt.imshow(np.abs(b1), cmap = "gray")
plt.title("Face 2 with phase of Face 1")
plt.show()
```



The images seem swapped after swapping their phase. It seems most of the information about the object is in its phase.

In [30]:
```
A2 = np.exp(1j*np.angle(FACE2)) # Only the phase of face2
B2 = np.exp(1j*np.angle(FACE1)) # Only the phase of face1

# Inverse transform the images with just their phase
a2 = ift2(A2)
b2 = ift2(B2)
```

```
plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.log(np.abs(a2)), cmap ="gray")
plt.title("Phase of Face 2")
plt.subplot(1,2,2)
plt.imshow(np.log(np.abs(b2)), cmap = "gray")
plt.title("Phase of Face 1")
plt.show()
```



5. Conclusion

We successfully imported an image and analyzed its various features. We used image processing techniques to apply filters on images. Additionally, we implemented fast fourier transform and observed diffraction pattern for a circular aperture. The same fft algorithm applied on images gave some interesting results for images such as qr codes and fingerprints. Because of the continuous pattern in these images their fourier transform also has some symmetry and patterns. In the exercise we swapped phases of two face images and it resulted in the images themselves being swapped, indicating that most of the information is in the phase.

In [ ]: