1. **Write a program to read lenna.jpg. Convert it into png and tif. Display disk size of jpg, png, and tif. Display the actual size of jpg without using any library.**

   ➔ **Python Code**

```
pip install pillow
from PIL import Image
import os

# read the input image
img = Image.open('lenna.jpg')

# convert to png and save
img.save('lenna.png', 'png')

# convert to tif and save
img.save('lenna.tif', 'tif')

# get the file sizes
jpg_size = os.path.getsize('lenna.jpg')
png_size = os.path.getsize('lenna.png')
tif_size = os.path.getsize('lenna.tif')

# display the file sizes
print("JPG size: {} bytes".format(jpg_size))
print("PNG size: {} bytes".format(png_size))
print("TIF size: {} bytes".format(tif_size))

# display the actual size of jpg without using any library
with open('lenna.jpg', 'rb') as f:
  data = f.read()
  jpg_actual_size = len(data)

print("Actual JPG size: {} bytes".format(jpg_actual_size))
```

2. **Write a program to read lenna.jpg and crop width from 100 to 600 and height from 100 to 600 and display the image. Use numpy library for cropping.**

   ➔ **Python Code**

```
pip install numpy

from PIL import Image
import numpy as np

# read the input image
```

```
img = Image.open('lenna.jpg')

# convert the image to a NumPy array
img_array = np.array(img)

# crop the image to the desired size
cropped_array = img_array[100:600, 100:600]

# convert the cropped array back to an image
cropped_img = Image.fromarray(cropped_array)

# display the cropped image
cropped_img.show()
```

3. **Write a program to convert lenna.jpg to gray and binary image. Do not use any library for conversion.**

➔ **Python Code**

```
from PIL import Image

# read the input image
img = Image.open('lenna.jpg')

# convert the image to grayscale
grayscale_img = Image.new('L', img.size)
for x in range(img.width):
   for y in range(img.height):
      r, g, b = img.getpixel((x, y))
      grayscale = int(0.299 * r + 0.587 * g + 0.114 * b)
      grayscale_img.putpixel((x, y), grayscale)

# convert the grayscale image to binary
threshold = 128
binary_img = Image.new('1', img.size)
for x in range(grayscale_img.width):
   for y in range(grayscale_img.height):
      intensity = grayscale_img.getpixel((x, y))
      if intensity >= threshold:
         binary_img.putpixel((x, y), 255)

# display the binary image
binary_img.show()
```

4. Use simple dithering technique to convert lenna.jpg into binary image so that it looks gray image.

➔ **Python Code**

```
from PIL import Image

# read the input image
img = Image.open('lenna.jpg')

# define the dither matrix
dither_matrix = [[0, 128], [192, 64]]

# convert the image to binary using dithering
binary_img = Image.new('1', img.size)
for x in range(img.width):
    for y in range(img.height):
        intensity = sum(img.getpixel((x, y))) // 3
        threshold = dither_matrix[x % 2][y % 2]
        if intensity >= threshold:
            binary_img.putpixel((x, y), 255)

# display the binary image
binary_img.show()
```

5. Plot the histogram of lenna.jpg. You can use matplotlib library.

➔ **Python Code**

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

# read the input image
img = Image.open('lenna.jpg')

# get the pixel values as a flattened array
pixels = img.getdata()

# plot the histogram
plt.hist(pixels, bins=256, range=(0, 255), color='gray', alpha=0.8)
plt.xlabel('Pixel value')
plt.ylabel('Frequency')
plt.title('Histogram of Lenna.jpg')
plt.show()
```

6. **Generate random numpy array of size (400,400) in range [0,256) and save it as image.**

➔ **Python Code**

```
import numpy as np
from PIL import Image

# generate random numpy array
arr = np.random.randint(0, 256, size=(400, 400), dtype=np.uint8)

# create image from numpy array
img = Image.fromarray(arr)

# save image
img.save('random_array.png')
```

7. **Consider a sine wave with frequency 3hz, amplitude 1 and phase 0. Generate 10 different samples from the sine wave at interval of 0.3 starting from 0sec. Plot the samples against time using python matplotlib.**

➔ **Python Code**

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

# set parameters for sine wave
freq = 3  # frequency in Hz
amp = 1   # amplitude
phase = 0 # phase angle in radians

# generate time array
t = np.arange(0, 3.1, 0.3)

# generate samples of sine wave
y = amp * np.sin(2 * np.pi * freq * t + phase)

# plot the samples against time
plt.plot(t, y, 'bo-', linewidth=2, markersize=8)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('10 Samples of Sine Wave with f = 3 Hz')
plt.grid()
plt.show()
```

8. **Consider a sine wave with frequency 3hz, amplitude 1 and phase 0. Generate samples with sampling rate 2hz starting from 0sec to 10sec and plot the samples.**

➔ **Python Code**

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

# set parameters for sine wave
freq = 3  # frequency in Hz
amp = 1   # amplitude
phase = 0 # phase angle in radians

# generate time array with sampling rate of 2 Hz
t = np.arange(0, 10.5, 0.5)

# generate samples of sine wave
y = amp * np.sin(2 * np.pi * freq * t + phase)

# plot the samples
plt.plot(t, y, 'r-', linewidth=2)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Sine Wave with f = 3 Hz, 2 Hz Sampling Rate')
plt.grid()
plt.show()
```

9. **Consider a sine wave with frequency 3hz, amplitude 1 and phase 0. Plot the sine wave starting from 0sec to 10sec. Consider Nyquist theorem for sampling.**

➔ **Python Code**

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

# set parameters for sine wave
freq = 3  # frequency in Hz
amp = 1   # amplitude
phase = 0 # phase angle in radians

# define sampling rate and calculate Nyquist frequency
sampling_rate = 30 # in Hz
nyquist_freq = sampling_rate / 2

# generate time array with sampling rate
t = np.arange(0, 10, 1/sampling_rate)
```

```
# generate samples of sine wave
y = amp * np.sin(2 * np.pi * freq * t + phase)

# plot the sine wave
plt.plot(t, y, 'r-', linewidth=2)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Sine Wave with f = 3 Hz, Sampling Rate = 30 Hz')
plt.grid()
plt.show()
```

10. **Given the wave file, write python program to get different parameters of the wave file such as number of channels, sampling width (bit depth), sampling rate, number of samples. Use wave library.**

➔ **Python Code**

```
import wave

# open wave file for reading
wav_file = wave.open("example.wav", "r")

# get parameters of wave file
num_channels = wav_file.getnchannels()
sample_width = wav_file.getsampwidth()
sample_rate = wav_file.getframerate()
num_samples = wav_file.getnframes()

# close wave file
wav_file.close()

# print parameters
print("Number of channels: ", num_channels)
print("Sampling width (bit depth): ", sample_width)
print("Sampling rate: ", sample_rate)
print("Number of samples: ", num_samples)
```

11. **Let us consider a sine wave with frequency 4400hz, 400m amplitude and phase 0. Generate sample from this sine wave at the rate of 44100hz for 1sec. Save the samples in the form of wave file in python. Use wave library.**

➔ **Python Code**

```
import wave
import math
```

```
# Define the parameters
frequency = 4400  # Hz
amplitude = 400  # m
phase = 0  # radians
duration = 1  # seconds
sample_rate = 44100  # Hz

# Calculate the number of samples
num_samples = int(sample_rate * duration)

# Create a new wave file
with wave.open('sine_wave.wav', 'w') as file:
    # Set the wave file parameters
    file.setnchannels(1)  # mono
    file.setsampwidth(2)  # 2 bytes per sample
    file.setframerate(sample_rate)

    # Generate and write the samples
    for i in range(num_samples):
        # Calculate the sample value at the current time
        time = i / sample_rate
        sample = amplitude * math.sin(2 * math.pi * frequency * time + phase)

        # Convert the sample to a 2-byte integer
        sample = int(sample * 32767)
        sample_bytes = sample.to_bytes(2, byteorder='little', signed=True)

        # Write the sample to the wave file
        file.writeframes(sample_bytes)
```

12. Implement 1D DCT in python.

➔ **Python Code**

```
import numpy as np

def dct(signal):
    """Compute the 1D Discrete Cosine Transform of the input signal."""

    # Calculate the length of the input signal
    N = len(signal)

    # Initialize the DCT coefficients array
    dct_coef = np.zeros(N)

    # Compute the DCT coefficients
```

```
    for k in range(N):
        sum = 0
        for n in range(N):
            sum += signal[n] * np.cos(np.pi * k * (2*n + 1) / (2*N))
        dct_coef[k] = sum * np.sqrt(2/N)

        # Apply the scaling factor to the first coefficient
        if k == 0:
            dct_coef[k] *= np.sqrt(1/2)

    return dct_coef
```

13. Implement run length encoding.

➔ **Python Code**

```
def run_length_encoding(input_string):
    """Run-length encoding implementation that takes an input string and returns its RLE
representation"""

    # Initialize variables
    count = 1
    prev_char = input_string[0]
    output = ""

    # Iterate through the input string, starting from the second character
    for char in input_string[1:]:
        # If the current character is the same as the previous one, increment the count
        if char == prev_char:
            count += 1
        # If the current character is different from the previous one, add the previous character and
count to the output
        else:
            output += str(count) + prev_char
            count = 1
            prev_char = char

    # Add the last character and count to the output
    output += str(count) + prev_char

    return output
```

14. Implement LZW algorithm.

➔ **Python Code**

```
def compress(data):
    # Initialize the dictionary with all single-character strings
    dictionary = {chr(i): i for i in range(256)}
    next_code = 256

    # Initialize the output and the current string
    output = []
    current = ''

    # Process the input data one character at a time
    for char in data:
        # If the current string + the next character is in the dictionary, update the current string
        if current + char in dictionary:
            current = current + char
        # Otherwise, output the code for the current string and add the current string + the next
character to the dictionary
        else:
            output.append(dictionary[current])
            dictionary[current + char] = next_code
            next_code += 1
            current = char

    # Output the code for the final string
    output.append(dictionary[current])

    return output

def decompress(codes):
    # Initialize the dictionary with all single-character strings
    dictionary = {i: chr(i) for i in range(256)}
    next_code = 256

    # Initialize the output and the previous code
    output = ''
    previous = codes.pop(0)

    # Process the codes one at a time
    for code in codes:
        # If the code is in the dictionary, add it to the output and update the previous code
        if code in dictionary:
            current = dictionary[code]
            output += current
```

```
            dictionary[next_code] = dictionary[previous] + current[0]
            next_code += 1
        # Otherwise, the code must be the first character of the previous string plus the first character of the current string
        else:
            current = dictionary[previous] + dictionary[previous][0]
            output += current
            dictionary[next_code] = current
            next_code += 1
        previous = code

    return output
```