This exercise aims to make you comfortable with the basic image processing tools and libraries. This exercise will serve as a starting point before you dive deep into the course.

Let's first import basic image processing or related libraries.

In [59]:
```python
import numpy as np                    # numpy library useful for most of the mathematical operations
import matplotlib.pyplot as plt    # useful for data visualization/plotting purpose. Can also be used for image visualization.
from google.colab import files
# For this exercise, we will restrict ourselves to matplotlib only. Please note that other libraries such as PIL, OpenCV
# can also be used as image processing libraries.
```

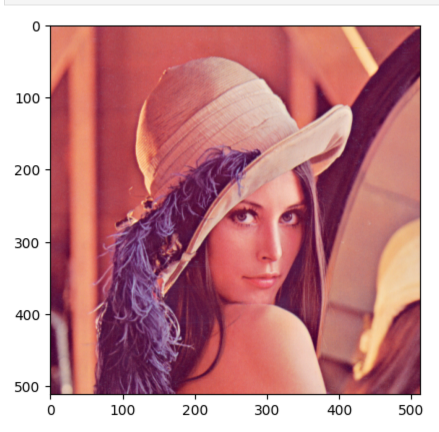
First load an image and visualize it.

In [60]:
```python
image= 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 lena.png to lena (1).png
```

In [61]:
```python
image = plt.imread("lena.png")
plt.imshow(image);
```



1. Image Information

It is always good to know basic image details, such as its dimensions, before one proceeds for the experiments.

Task1.1 : write code to find image dimension and print it

In [62]:
```python
w,h,c=image.shape
w,h,c
```

Out[62]:
```
(512, 512, 3)
```

Is this image RGB (no of channels?), gray or binary (intensity range?)? What can you say about aspect ratio (defined as width/height) of this image?

In [63]:
```python
np.max(image) #the image has been normalized
```

Out[63]:
```
1.0
```

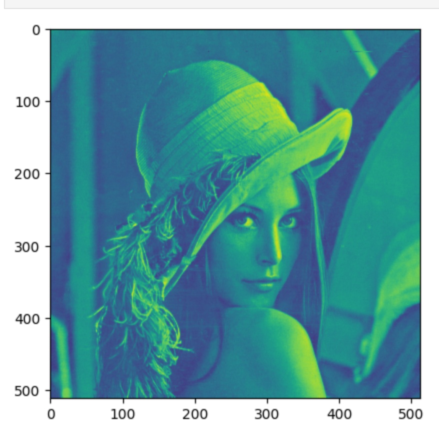
Task1.2: Visualization of each channel

An RGB image can be decomposed into three channels, Red(R), Green(G), Blue(B). In this subsection, let's visualize each channel separately.

In [64]:
```python
def VisualizeChannel(image,channel):
    '''
    This function is helpful to visualize a specific channel of an RGB image.
    image: RGB image
    channel: channel, one wish to visualize (can take value 0 (for red), 1(green), 2(blue))
    '''
    #write your code here
    output=image[:,:,channel]

    return output    # 'output' is image's particular channel values
```

Can you also comment on the maximum and minimum intensity values of each channel? What can you say about the range of intensity values?

In [65]:
```python
plt.imshow(VisualizeChannel(image, 2));
```



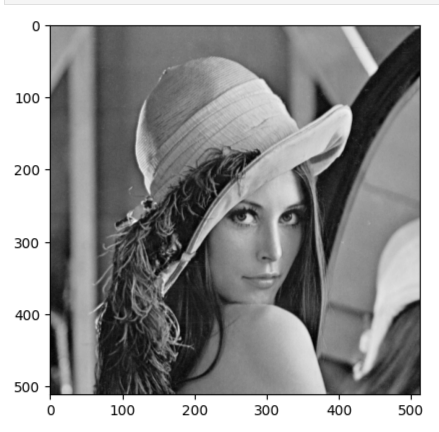
2. Intensity Manipulations

Task2.1: RGB to Gray

We may need a gray image for some of our applications. One can also convert RGB to gray to reduce computational complexity. For this part, we will convert an RGB image to grayscale. Refer this link for explanation: https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm

In [66]:
```python
def RGB2Gray(image):
    '''
    This function converts an RGB image to grayscale
    image: RGB image
    '''
    #write you code here and visualize the result
    r,g,b=image[:,:,0], image[:,:,1], image[:,:,2]
    gray=0.3*r+0.59*g+0.11*b
    return gray       #'gray' is grayscale image, converted from RGB image
```

In [67]:
```python
plt.imshow(RGB2Gray(image), cmap="gray"); #matplotlib by default tries to replicate the channels to 3 channels
```



In [68]:
```python
np.max(RGB2Gray(image))
```

Out[68]:
```
0.9614118
```

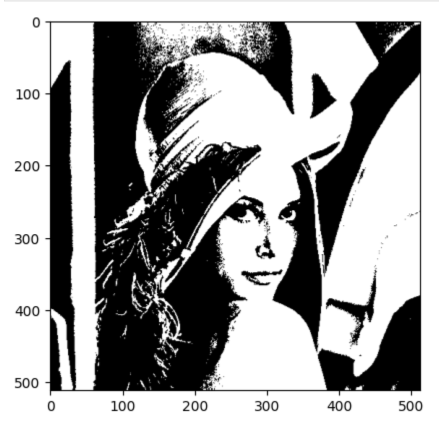
We can also convert a gray image to a binary image. For task2.2, consider a gray image as input (you may take the output from task2.1 as input). Write code to threshold a gray image such that I(x,y) = 1 if I(x,y) >= T = 0 if I(x,y) < T where T is threshold Though there are proper methods(such as the Otsu method) to find a suitable T, we will not go into details of those algorithms and randomly select T values and visualize the result.

Task2.2 : Gray to Binary

Before you proceed to code, Can you comment on the valid range of T? (Hint: --- Task1.2)

In [69]:
```python
def Gray2Binary(image,T):
    '''
    This function converts a gray image to binary based on the rule stated above.
    image: image (can be RGB or gray); if the image is RGB, convert it to gray first
    T: Threshold
    '''

    #check if image is RGB if yes, convert it to gray
    flag = len(image.shape)
    if flag == 3:         #i.e. RGB image, hence to be converted to gray
        # write code to convert it to gray or you can call function "RGB2Gray" defined in task2.1
      image=RGB2Gray(image)

    #Write code to threshold image based on the rule stated above and return this binarized image (say it 'bimage')
    image[image>T]=1
    image[image<=T]=0
    bimg=image

    #write code to visualize the resultant image


    return bimg
```

In [70]:
```python
plt.imshow(Gray2Binary(image, 0.5), cmap="gray");
```



An image is nothing but a matrix. Hence one can perform all kinds of mathematical operations on an image just like a matrix. To convince ourselves with the above statement, let's crop a section of a gray image, print its value, and perform some mathematical operations. For a better data display, we will cut only 5*5 areas of the gray image.

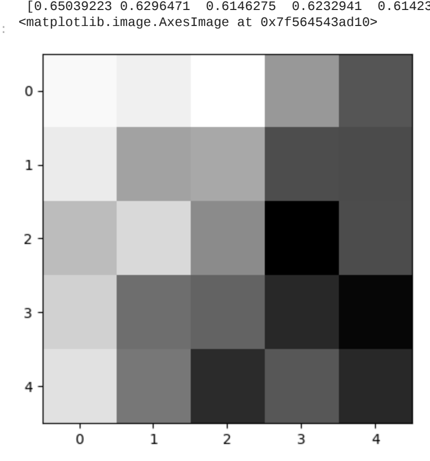
Task2.3: Crop a 5*5 section of a gray image

In [71]:
```python
def ImageCrop(image,r0,c0):
    '''
    This function crops 5*5 rectangular patch defined by image coordinates(r0,c0),(r0,c0+5),(r0+5,c0) and (r0+5,c0+5)
    of an image.
    image: Image can be RGB or gray
    r0: starting row index
    c0: starting column index
    '''
    # write code to check if input is RGB , if its RGB convert it to gray
    if image.shape[2]==3: image=RGB2Gray(image)


    # write code to select 5*5 rectangular patch defined as above (say it 'patch')
    patch=image[r0:r0+5, c0:c0+5]

    # visualize patch and print its value



    return patch
```

In [72]:
```python
r0, c0= 12, 50
patch = ImageCrop(image,r0,c0)
print(patch)
plt.imshow(patch, cmap="gray")
```

```
[[0.65513724 0.65345097 0.6564706  0.63619614 0.6229412 ]
 [0.6523529  0.6381177  0.6392549  0.62145096 0.6209412 ]
 [0.64313734 0.648902   0.6336471  0.6061569  0.621098  ]
 [0.64729416 0.62788236 0.6257647  0.61407846 0.6075294 ]
 [0.65039223 0.6296471  0.6146275  0.6232941  0.6142353 ]]
```

Out[72]:
```
<matplotlib.image.AxesImage at 0x7f564543ad10>
```



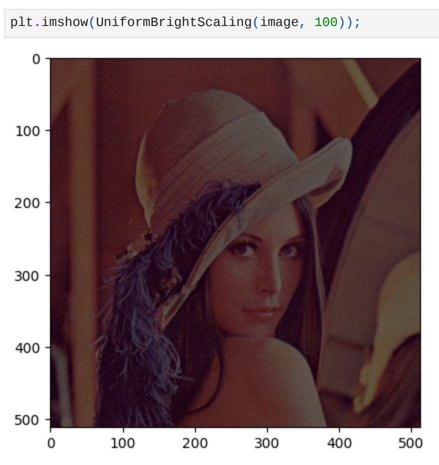
Now you have 5*5 patch and you know its values too. Can you try 1. multiplying patch by 0.5 2. multiplying patch by 2 3. create another random 5*5 patch (numpy array) and add/subtract it to the patch Does it follow matrix addition/subtraction and multiplication rules? You can also play around with other matrix operations.

Task2.4: Uniform Brightness Scaling

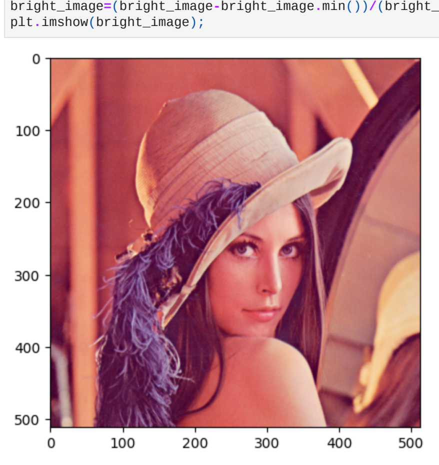
Hopefully, you are convinced that an image is a matrix. Hence we can perform multiplication/division or addition/subtraction operations. These operations will change the brightness value of the image; can make an image brighter or darker depending on the multiplying/scaling factor. For this task, let's change the image brightness uniformly. Consider scale to be 0.3,0.5,1,2 for four different cases. What is your observation?

In [73]:
```python
def UniformBrightScaling(image,scale):
    '''
    This function uniformly increases or decreases the pixel values (of all image locations) by a factor 'scale'.
    image: image (can be RGB or gray image)
    scale: A scalar by which pixels'svalues need to be multiplied
    '''
    #write your code here
    output=(image*scale).astype(np.uint8)

    #display the resultant image


    return output        #replace output with the variable name you used for final result
```

In [74]:
```python
plt.imshow(UniformBrightScaling(image, 100));
```



In [75]:
```python
## Image normalization
bright_image=UniformBrightScaling(image, 100)
bright_image=(bright_image-bright_image.min())/(bright_image.max()- bright_image.min())
plt.imshow(bright_image);
```

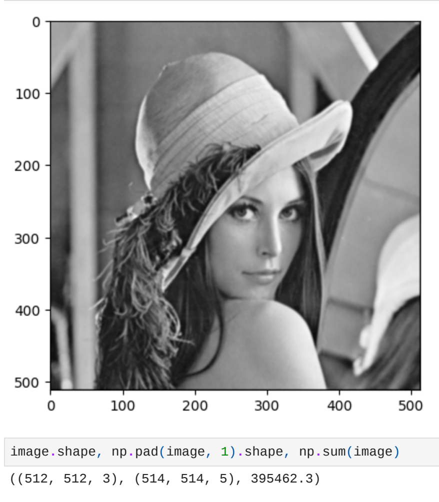


3. Image Filtering

In this section, you will perform some of the image filtering techniques. --- Convolution is one of the most widely used operations for images. Convolution can be used as a feature extractor; different kernel results in various types of features. Refer https://en.wikipedia.org/wiki/Kernel_(image_processing) to see few examples of kernel.

In [76]:
```python
def feature_extractor(image,kernel):
    '''
    This function performs convolution operation to a gray image. We will consider 3*3 kernel here.
    In general kernel can have shape (2n+1) * (2n+1)  where n>= 0
    image: image (can be RGB or gray); if RGB convert it to gray
    kernel: 3*3 convolution kernel
    '''
    # first convert RGB to gray if input is RGB image

    l = len(image.shape)

    if l == 3:
        #write code to convert it to gray scale
        image=RGB2Gray(image)

    # write code to create a zero array of size (r,c) which will store the resultant value at specific pixel locations (say it output)
    output=np.zeros(image.shape)

    #write code to create a zero array with size (r+2,c+2) if (r,c) is the gray image size.  (say it pad_img)
    pad_img= np.zeros((image.shape[0]+2, image.shape[1]+2))


    #now copy gray image to above created array at location starting from (1,1)
    pad_img[1:-1,1:-1]=image

    #write code to convolve the image
    for row in range(1, pad_img.shape[0]-1):     # use appropiate range values for row and col
        for col in range(1, pad_img.shape[1]-1):
            # select 3*3 patch with center at (row,col), flatten it. flatten the kernel and take dot product between both (or directly take element wise multiplication and su
            patch=pad_img[row-1:row+2, col-1:col+2]
            conv_val=np.sum(np.multiply(patch, kernel))
            # store this scalar value to output matrix with starting location (0,0)         (alternatively one could also create a list and reshape it to output size)
            output[row-1, col-1]= conv_val



    return output
```

In [77]:
```python
plt.imshow(feature_extractor(image, np.ones((3,3))), cmap="gray");
```



In [78]:
```python
image.shape, np.pad(image, 1).shape, np.sum(image)
```

Out[78]:
```
((512, 512, 3), (514, 514, 5), 395462.3)
```

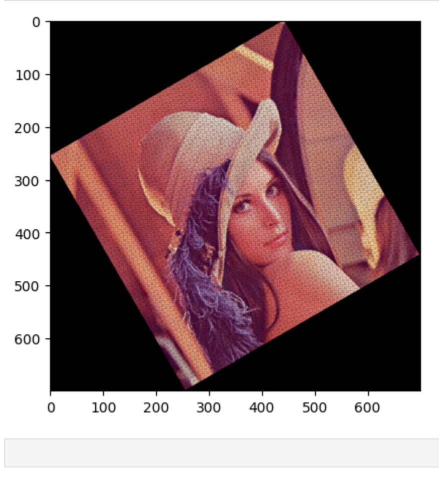
In [79]:
```python
## Note that the steps described above are to help you get started. You can follow other valid steps too. Result from all
 #of the method should be the same. Pseudocode is available at: https://en.wikipedia.org/wiki/Kernel_(image_processing)
```

4.Geometric Transformation

Task4.1: Image Rotation (In-plane)

In [80]:
```python
import math
def rotate(img, degree):
  degree=math.radians(degree)
  cos,sin=math.cos(degree), math.sin(degree)
  cx, cy= round(img.shape[0]/2), round(img.shape[1]/2) #centre of original image

  hn, wn= round(abs(img.shape[0]*cos)+ abs(img.shape[1]*sin)), round(abs(img.shape[0]*sin)+ abs(img.shape[1]*cos)) #new_height and new_width of the frame after transformation
  cxn, cyn= round(hn/2), round(wn/2) #centre of the frame and not of the rotated image
  output=np.zeros((hn, wn, img.shape[2]))

  rotate_m=np.array([[cos, -sin], [sin, cos]]) #rotation matrix

  for r in range(img.shape[0]):
    for c in range(img.shape[1]):
      coords=np.matmul(rotate_m, np.array([[cy-r], [cx-c]])).flatten().astype("int")
      output[cyn-coords[0]-1, cxn-coords[1]-1]=img[r,c]

  return output
```

In [81]:
```python
plt.imshow(rotate(image, 30));
```



In [81]: