1. Black and White Binary Image

In [137]:
```
# Binary images are images whose pixels have only two possible intensity va
# They are normally displayed as black and white.
# Numerically, the two values are often 0 for black, and either 1 or 255 fo
import numpy as np
import matplotlib.pyplot as plt
```

In [138]:
```
M = np.array([[0,255,255,255,0],[0,0,255,0,0],[0,255,0,255,0],[0,255,255,25
T = np.array([[0,0,0,0,0],[1,1,0,1,1],[1,1,0,1,1],[1,1,0,1,1],[1,1,0,1,1],[
S = np.array([[0,0,0,0],[0,1, 1,1], [1,0, 1,1],[1,1, 0,1], [1,1, 1,0], [0,
U = np.array([[0,1,1,1,0],[0,1,1,1,0],[0,1,1,1,0],[0,1, 1,1, 0], [0,1, 1,1,
```

In [139]:
```
M.shape
```

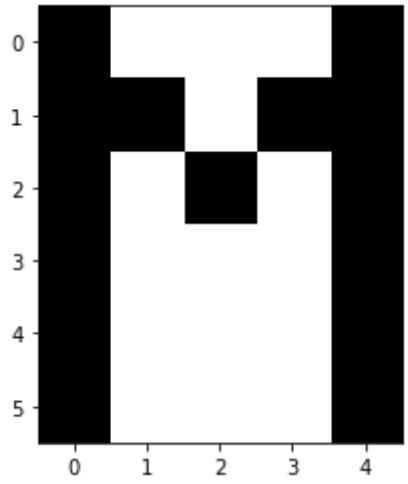
Out[139]:
```
(6, 5)
```

In [140]:
```
M
```

Out[140]:
```
array([[  0, 255, 255, 255,   0],
       [  0,   0, 255,   0,   0],
       [  0, 255,   0, 255,   0],
       [  0, 255, 255, 255,   0],
       [  0, 255, 255, 255,   0],
       [  0, 255, 255, 255,   0]])
```

In [141]:
```
plt.imshow(M,cmap="gray")
```

Out[141]:
```
<matplotlib.image.AxesImage at 0x7f7f86dfaaf0>
```



In [66]:
```
M[1,2]
```

Out[66]:
```
255
```

In [67]:
```
M[0,0]
```

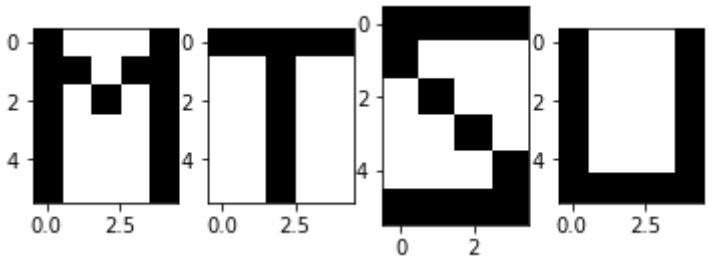
Out[67]: 0

In [68]:
```
M[5,4]
```

Out[68]: 0

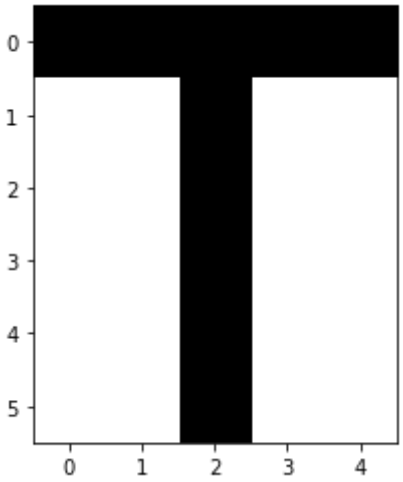
In [142]:
```
import matplotlib.pyplot as plt

ax1 = plt.subplot(141)
ax1.imshow(M,cmap="gray")

ax2 = plt.subplot(142)
ax2.imshow(T,cmap="gray")

ax3 = plt.subplot(143)
ax3.imshow(S,cmap="gray")

ax4 = plt.subplot(144)
ax4.imshow(U,cmap="gray")
```

Out[142]: <matplotlib.image.AxesImage at 0x7f7f86cae4c0>



In [70]:
```
plt.imshow(T,cmap="gray")
```

Out[70]: <matplotlib.image.AxesImage at 0x7f7f860bcdc0>



In [10]:
```
T[0,0]
```

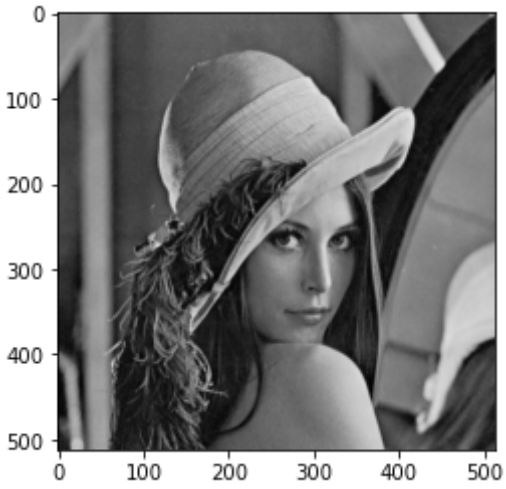
Out[10]: 0

In [11]:
```
T[1,3]
```

Out[11]: 1

2. Gray Scale (Intensity) Image

In [144]:
```
# a grayscale image is one in which the value of each pixel is a single sam
# that is, it carries only intensity information.
# Grayscale images, a kind of black-and-white or gray monochrome, are compo
# The contrast ranges from black at the weakest intensity to white at the s

# Grayscale images are distinct from one-bit bi-tonal black-and-white image
# in the context of computer imaging, are images with only two colors: blac
# Grayscale images have many shades of gray in between.

import matplotlib.pyplot as plt
import matplotlib.image as img

image = img.imread('lena.jpeg')

plt.imshow(image, cmap='gray', vmin = 0, vmax = 255)
plt.show()
```



In [145]:
```
image.shape
```

Out[145]: (512, 512)

In [146]:
```
image
```

Out[146]:
```
array([[135, 137, 138, ..., 148, 131,  92],
       [136, 137, 138, ..., 149, 134,  96],
       [137, 138, 138, ..., 149, 135,  96],
       ...,
       [ 20,  21,  24, ...,  71,  71,  70],
       [ 21,  22,  26, ...,  68,  70,  73],
       [ 23,  24,  28, ...,  67,  69,  75]], dtype=uint8)
```

In [147]:
```
image.max().max()
```

Out[147]:
```
253
```

3. RGB (Truecolor) Image

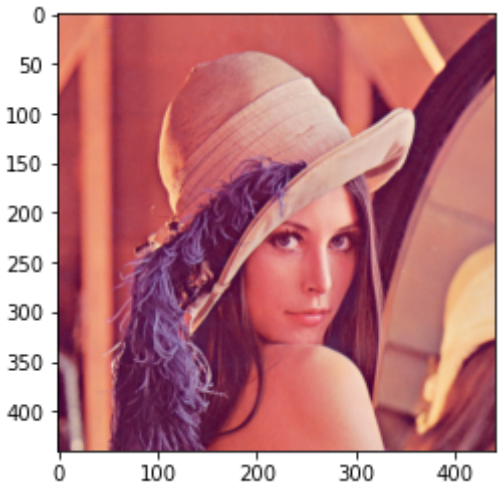
RGB Color Calculator: https://www.w3schools.com/colors/colors_rgb.asp (https://www.w3schools.com/colors/colors_rgb.asp)

In [148]:
```
# An RGB image, sometimes referred to as a truecolor image,
# is stored as an m-by-n-by-3 data array that defines red, green, and blue
# The color of each pixel is determined by the combination of the red, gree
import matplotlib.pyplot as plt
import matplotlib.image as img
image_color = img.imread('lena.png')
```

In [149]:
```
image_color.shape
```

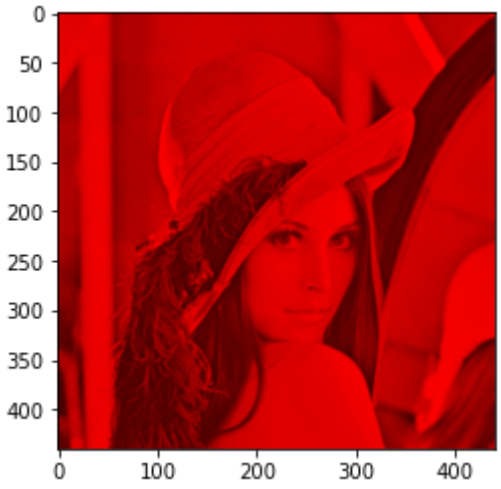
Out[149]:
```
(440, 440, 3)
```

In [156]:
```
plt.imshow(image_color, vmin = 0, vmax = 255)
plt.show()
```

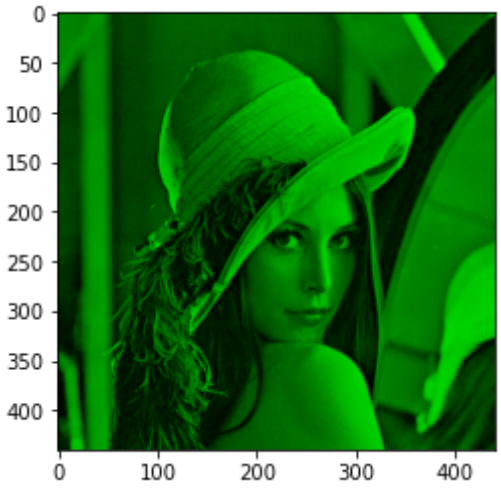


In [152]:
```
import numpy
import cv2
img = cv2.imread("lena.png")
blue,green,red = cv2.split(img)
zeros = numpy.zeros(blue.shape, numpy.uint8)
redScale = cv2.merge((red, zeros, zeros))
greenScale = cv2.merge((zeros, green, zeros))
blueScale = cv2.merge((zeros, zeros, blue))
```

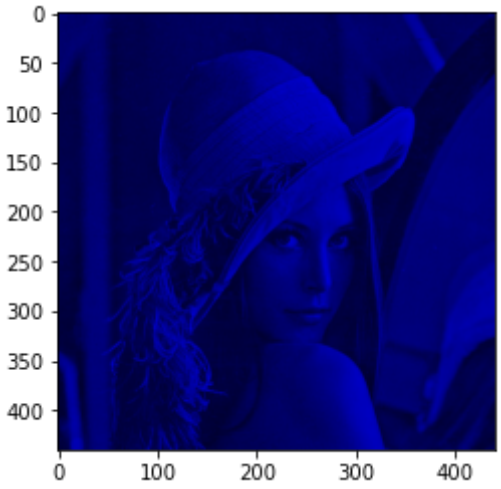
In [153]:
```
plt.imshow(redScale)
plt.show()
```



In [154]:
```
plt.imshow(greenScale)
plt.show()
```



In [155]:
```
plt.imshow(blueScale)
plt.show()
```



In [157]:
```
red
```

Out[157]:
```
array([[226, 225, 222, ..., 233, 224, 202],
       [226, 225, 222, ..., 233, 224, 202],
       [226, 225, 222, ..., 232, 223, 201],
       ...,
       [ 84,  86,  93, ..., 174, 169, 172],
       [ 82,  86,  95, ..., 177, 178, 183],
       [ 81,  86,  96, ..., 178, 181, 185]], dtype=uint8)
```

In [158]:
```
red.shape
```

Out[158]:
```
(440, 440)
```

In [159]:
```
green
```

Out[159]:
```
array([[137, 137, 136, ..., 150, 136, 102],
       [137, 137, 136, ..., 150, 136, 102],
       [137, 137, 136, ..., 149, 134, 101],
       ...,
       [ 18,  21,  25, ...,  72,  70,  62],
       [ 20,  24,  29, ...,  68,  71,  70],
       [ 22,  25,  31, ...,  67,  71,  74]], dtype=uint8)
```

In [160]:
```
green.shape
```

Out[160]:
```
(440, 440)
```

In [161]:
```
blue
```

Out[161]:
```
array([[124, 127, 132, ..., 123, 114,  92],
       [124, 127, 132, ..., 123, 114,  92],
       [124, 127, 132, ..., 122, 113,  92],
       ...,
       [ 60,  58,  58, ...,  84,  78,  80],
       [ 58,  59,  60, ...,  78,  80,  80],
       [ 56,  58,  62, ...,  76,  81,  81]], dtype=uint8)
```

In [162]:
```
# Graphics file formats store RGB images as 24-bit images, where the red, g
# This yields a potential of 16 million colors.
# The precision with which a real-life image can be replicated has led to t
2**24
```

Out[162]:
```
16777216
```

In [103]:
```
pip install pillow
```

```
Requirement already satisfied: pillow in /Users/xyang/opt/anaconda3/lib/p
ython3.8/site-packages (8.0.1)

[notice] A new release of pip is available: 23.1.2 -> 23.2.1
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

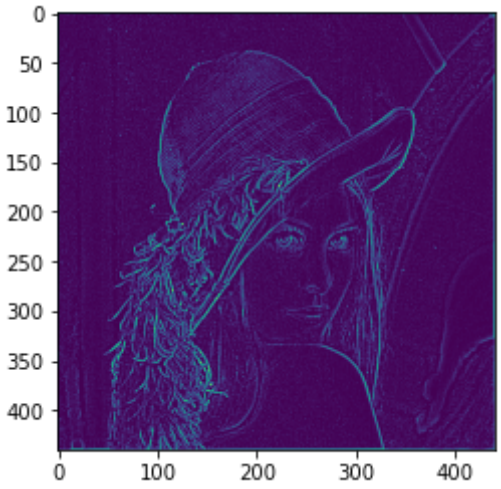
4. Edge Detection using Pillow

Reference: https://www.geeksforgeeks.org/python-edge-detection-using-pillow/ (https://www.geeksforgeeks.org/python-edge-detection-using-pillow/)

In [163]:
```
# Edge Detection, is an Image Processing discipline that incorporates mathe
# Edge Detection internally works by running a filter/Kernel over a Digital
# regions like stark changes in brightness/Intensity value of pixels.
from PIL import Image, ImageFilter
```

In [164]:
```
# Opening the image (R prefixed to string
# in order to deal with '\' in paths)
image = Image.open("lena.png")

# Converting the image to grayscale, as edge detection
# requires input image to be of mode = Grayscale (L)
image = image.convert("L")

# Detecting Edges on the Image using the argument ImageFilter.FIND_EDGES
image = image.filter(ImageFilter.FIND_EDGES)

# Saving the Image Under the name Edge_Sample.png
image.save("Edge_Sample.png")
```

In [165]:
```
plt.imshow(image)
plt.show()
```



5. Computer Vision projects for all experience levels

Reference: [https://neptune.ai/blog/15-computer-visions-projects (https://neptune.ai/blog/15-computer-visions-projects)](https://neptune.ai/blog/15-computer-visions-projects)

In [111]:
```
# Computer vision deals with how computers extract meaningful information f
# It has a wide range of applications, including reverse engineering, secur
# and processing, computer animation, autonomous navigation, and robotics.
```

In [112]:
```
# Beginner level Computer Vision projects

# 1. Edge & Contour Detection
# 2. Colour Detection & Invisibility Cloak
# 3. Text Recognition using OpenCV and Tesseract (OCR)
# 4. Face Recognition with Python and OpenCV
# 5. Object Detection
#

# Intermediate level Computer Vision projects
# 6. Hand Gesture Recognition
# 7. Human Pose Detection
# 8. Road Lane Detection in Autonomous Vehicles
# 9. Pathology Classification
# 10. Fashion MNIST for Image Classification

# Advanced level Computer Vision projects
# 11. Image Deblurring using Generative Adversarial Networks
# 12. Image Transformation
# 13. Automatic Colorization of Photos using Deep Neural Networks
# 14. Vehicle Counting and Classification
# 15. Vehicle license plate scanners


# Extra projects
# Photo Sketching
# Collage Mosaic Generator
# Blur the Face
# Image Segmentation
# Sudoku Solver
# Object Tracking
# Watermarking Images
# Image Reverse Search Engine
```

In [ ]:
```

```