Pre-requisites and references

1. Minimum software installations: Python 3.6, numpy, matplotlib, cv2 (open computer vision library), Jupyter.

Image Processing Tools

1. OpenCV, **2. Python image library,** **3. Scikit-image library**

Python IDE

1. Jupyter, </n> **2. Pycharm,** **3. Spyder,** **4. Visual Studio code**

Introduction to Numpy.

1. NumPy is a Python open source package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

2. NumPy is often used along with packages like SciPy (Scientific Python) and Mat−plotlib (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing.

3. Numpy package can be imported as " import numpy as np".

List vs Numpy

- List

1. The list can be homogeneous or heterogeneous.
2. Element wise operation is not possible on the list.
3. Python list is by default 1 dimensional. But we can create an N-Dimensional list. But then too it will be 1 D list
   storing another 1D list
4. Elements of a list need not be contiguous in memory.

- Numpy

1. We can create a N-dimensional array in python using numpy.array().
2. Array are by default Homogeneous, which means data inside an array must be of the same Datatype. (Note you can also create a structured array in python).
3. Element wise operation is possible.
4. Numpy array has the various function, methods, and variables, to ease our task of matrix computation.
5. Elements of an array are stored contiguously in memory. For example, all rows of a two dimensioned array must have

the same number of columns. Or a three dimensioned array must have the same number of rows and columns on each card.

Example 1: Memory consumption between Numpy array and lists.

In [5]:

```
# importing numpy package
import numpy as np

# importing system module
import sys

# declaring a list of 1000 elements
S= range(1000)

# printing size of each element of the list
print("Size of each element of list in bytes: ",sys.getsizeof(S))

# printing size of the whole list
print("Size of the whole list in bytes: ",sys.getsizeof(S)*len(S))

# declaring a Numpy array of 1000 elements
D= np.arange(1000)

# printing size of each element of the Numpy array
print("Size of each element of the Numpy array in bytes: ",D.itemsize)

# printing size of the whole Numpy array
print("Size of the whole Numpy array in bytes: ",D.size*D.itemsize)
```

```
Size of each element of list in bytes:  48
Size of the whole list in bytes:  48000
Size of each element of the Numpy array in bytes:  4
Size of the whole Numpy array in bytes:  4000
```

Example 2: Time comparison between Numpy array and Python lists.

In [2]:

```
# importing required packages
import numpy
import time

# size of arrays and lists
size = 10000000  # 1 crore elements

# declaring lists
list1 = range(size)
list2 = range(size)

# declaring arrays
array1 = numpy.arange(size)
array2 = numpy.arange(size)

# capturing time before the multiplication of Python lists
initialTime = time.time()

# multiplying  elements of both the lists and stored in another list
resultantList = [(a * b) for a, b in zip(list1, list2)]

# calculating execution time
print("Time taken by Lists to perform multiplication:",(time.time() - initialTime),"
```

```
# capturing time before the multiplication of Numpy arrays
initialTime = time.time()

# multiplying  elements of both the Numpy arrays and stored in another Numpy array
resultantArray = array1 * array2

# calculating execution time
print("Time taken by NumPy Arrays to perform multiplication:",(time.time() - initial
```

```
Time taken by Lists to perform multiplication: 1.0900835990905762 seconds
Time taken by NumPy Arrays to perform multiplication: 0.01795029640197754 seconds
```

Example 3: Effect of operations on Numpy array and Python Lists.

In [3]:

```
# importing Numpy package
import numpy as np

# declaring a list
ls =[1, 2, 3]

# converting the list into a Numpy array
arr = np.array(ls)

try:
    # adding 4 to each element of list
    ls = ls + 4

except(TypeError):
    print("Lists don't support list + int")

# now on array
try:
    # adding 4 to each element of Numpy array
    arr = arr + 4

    # printing the Numpy array
    print("Modified Numpy array: ",arr)

except(TypeError):
    print("Numpy arrays don't support list + int")
```

```
Lists don't support list + int
Modified Numpy array:  [5 6 7]
```

Write a NumPy program to compute the inverse of a given matrix.

In [4]:

```
import numpy as np
m = np.array([[1,2],[3,4]])
print("Original matrix:")
print(m)
result =  np.linalg.inv(m)
print("Inverse of the said matrix:")
print(result)
```

```
Original matrix:
[[1 2]
 [3 4]]
Inverse of the said matrix:
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Write a NumPy program to get the dates of yesterday, today and tomorrow.

In [1]:
```
import numpy as np
yesterday = np.datetime64('today', 'D') - np.timedelta64(1, 'D')
print("Yestraday: ",yesterday)
today     = np.datetime64('today', 'D')
print("Today: ",today)
tomorrow  = np.datetime64('today', 'D') + np.timedelta64(1, 'D')
print("Tomorrow: ",tomorrow)
```

```
Yestraday:  2022-08-23
Today:  2022-08-24
Tomorrow:  2022-08-25
```

- Creating Numpy array

There are a number of ways to initialize new numpy arrays, for example from:

- A Python list or tuples.

- Using functions that are dedicated to generating numpy arrays, like arange, linspace, zeros, ones etc.

- Reading data from files.

- "rank" is defined as the size of the array.

In [6]:
```
# Creating an array from list
import numpy as np

lst = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]  # create a list

lst_arr = np.array(lst)  # convert a list to an array

print(type(lst_arr))
print("\nArray created using passed list:\n", lst_arr)
```

```
<class 'numpy.ndarray'>

Array created using passed list:
 [[1 2 3]
 [3 6 9]
 [2 4 6]]
```

In [61]:
```
# Creating an array from tuple
tup_arr = np.array((1, 3, 2))
print("\nArray created using passed tuple:\n", tup_arr)
```

```
Array created using passed tuple:
 [1 3 2]
```

In [8]:
```
# Creating an array using built-in functions
# np.range
```

```
a0=np.arange(1,100,2)# start , end , step

print(a0)
print(type(a0[0]))
print("\n")
```

```
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95
 97 99]
<class 'numpy.int32'>
```

In [3]:

```
# np.linespace

import numpy as np
a1=np.linspace(1,10,10) # i to 10 with    elements

print(a1)
print(type(a1[0]))
print("\n")

a2=np.linspace(1,10,20, dtype='float16')
print(a2)
print(type(a2[0]))
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
<class 'numpy.float64'>


[ 1.     1.474  1.947  2.422  2.895  3.37   3.842  4.316  4.79   5.26
  5.74   6.21   6.684  7.156  7.633  8.1    8.58   9.055  9.52  10.   ]
<class 'numpy.float16'>
```

In [10]:

```
# Example

a3 = np.zeros((3,4,2), dtype='uint8') # (no. of planes , rows, columns )

print ("\nAn array initialized with all zeros:\n", a3)
```

```
An array initialized with all zeros:
 [[[0 0]
  [0 0]
  [0 0]
  [0 0]]

 [[0 0]
  [0 0]
  [0 0]
  [0 0]]

 [[0 0]
  [0 0]
  [0 0]
  [0 0]]]
```

In [11]:

```
# Example
a4 = np.zeros((3,3), dtype='uint8')
print ("\nAn array initialized with all zeros:\n", a4)
```

```
An array initialized with all zeros:
 [[0 0 0]
```

```
 [0 0 0]
 [0 0 0]]
```

In [12]:
```
# Example
a5 = np.ones((3, 4))
print ("\nAn array initialized with all zeros:\n", a5)

a5copy=np.ones((3, 4), dtype='uint8')
print ("\nAn array initialized with all zeros in unsigned int datatype:\n", a5copy)
```

```
An array initialized with all zeros:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

An array initialized with all zeros in unsigned int datatype:
 [[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

In [7]:
```
# Example
# Create a constant value array of complex type

d = np.full((3, 4), 10, dtype = 'uint8')

print ("\nAn array initialized with all 6s. Array type is uint8:\n", d)
```

```
An array initialized with all 6s. Array type is uint8:
 [[10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]]
```

In [14]:
```
# Example
I = np.eye(3,dtype='uint8')
print ("\nAn identity matrix :\n", I)

print(type(I[0][0]))
print("\n")
```

```
An identity matrix :
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
<class 'numpy.uint8'>
```

In [15]:
```
# Example
diaga=np.diag([1,2,3,4])
print(diaga)
```

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

- Accessing array elements

In [4]:
```
# 1D array
import numpy as np

arr=np.arange(1,10,2)
```

```
print(arr)

print(arr[0]) # from first to last

print(arr[-1]) # from last to first

arr[2]=50
print(arr)
```

```
[1 3 5 7 9]
1
9
[ 1  3 50  7  9]
```

In [18]:

```
# 2D array
import numpy as np
arr3 = np.array( [[ 1, 2, 3],
                  [ 4, 2, 5],
                  [ 4, 2, 6]] )
print(arr3[1][2])
print(arr3[2,2])
print(arr3[-1,-1])
```

```
5
6
6
```

In [19]:

```
# 3D array
import numpy as np
arr4 = np.array([
                  [
                      [ 1, 2, 3],
                      [ 4, 2, 5],
                      [0,0,0]
                  ],
                  [
                      [ 41, 21, 51],
                      [121, 222,125],
                      [1,1,1]
                  ],
                  [
                      [ 9, 10, 11],
                      [12, 13, 15],
                      [2,2,2]
                  ]
                ] )
print(arr4.shape) # (dimention, rows, coloumn)
print("\n")
print(arr4)
print("\n")
print(arr4[1][1][1])
print(arr4[2][2][0])
print("\n")
arr4[2][2][2]=500
print("Updated array:\n\n",arr4)
```

```
(3, 3, 3)


[[[  1   2   3]
  [  4   2   5]
  [  0   0   0]]

 [[ 41  21  51]
```

```
 [121 222 125]
 [  1   1   1]]

[[  9  10  11]
 [ 12  13  15]
 [  2   2   2]]]


222
2


Updated array:

[[[  1   2   3]
 [  4   2   5]
 [  0   0   0]]

[[ 41  21  51]
 [121 222 125]
 [  1   1   1]]

[[  9  10  11]
 [ 12  13  15]
 [  2   2 500]]]
```

In [8]:

```
# few Built-in functions in numpy

arr1=np.array([[1,2,3,4,5],[6,5,7,8,9]])

print(arr1)
print("\n")

rowsum=arr1.sum(axis=1) # row wise

print("Row sum:",rowsum)
print("\n")

colsum=arr1.sum(axis=0)
print("Col sum:",colsum)

print()
print("Matrix dimension:",arr1.shape)
print("max value:",arr1.max())
print("min value:",arr1.min())
print("max value at index:",arr1.argmax())
print("min value at index:",arr1.argmin())
print("mean value:",arr1.mean())
print("median value:",np.median(arr1[0]))
```

```
[[1 2 3 4 5]
 [6 5 7 8 9]]


Row sum: [15 35]


Col sum: [ 7  7 10 12 14]

Matrix dimension: (2, 5)
max value: 9
min value: 1
max value at index: 9
min value at index: 0
mean value: 5.0
median value: 3.0
```

- Slicing numpy array elements

In [29]:
```
a=np.arange(10)
print(a)
a[1:8:2]
```

```
[0 1 2 3 4 5 6 7 8 9]
```

Out[29]:
```
array([1, 3, 5, 7])
```

In [30]:
```
# We can also combine assignment and slicing
a=np.arange(10)

a[5:]=10

print(a)
```

```
[ 0  1  2  3  4 10 10 10 10 10]
```

In [32]:
```
b=np.arange(5)
a[5:]=b[::-1] # start : end : -1 indicates reverse order  :4 (strat:end) ( start: en
print(a)
```

```
[0 1 2 3 4 4 3 2 1 0]
```

In [5]:
```
# crop sub array
# Initial Array
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])

print("Initial Array: ")
print(arr)

# Printing a range of Array
# with the use of slicing method

sliced_arr = arr[:2, ::2] # if single : (start : end) , if double :: (strat:end:step

print ("Array with first 2 rows and"
    " alternate columns(0 and 2):\n", sliced_arr)
```

```
Initial Array:
[[-1.   2.   0.   4. ]
 [ 4.  -0.5  6.   0. ]
 [ 2.6  0.   7.   8. ]
 [ 3.  -7.   4.   2. ]]
Array with first 2 rows and alternate columns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]
```

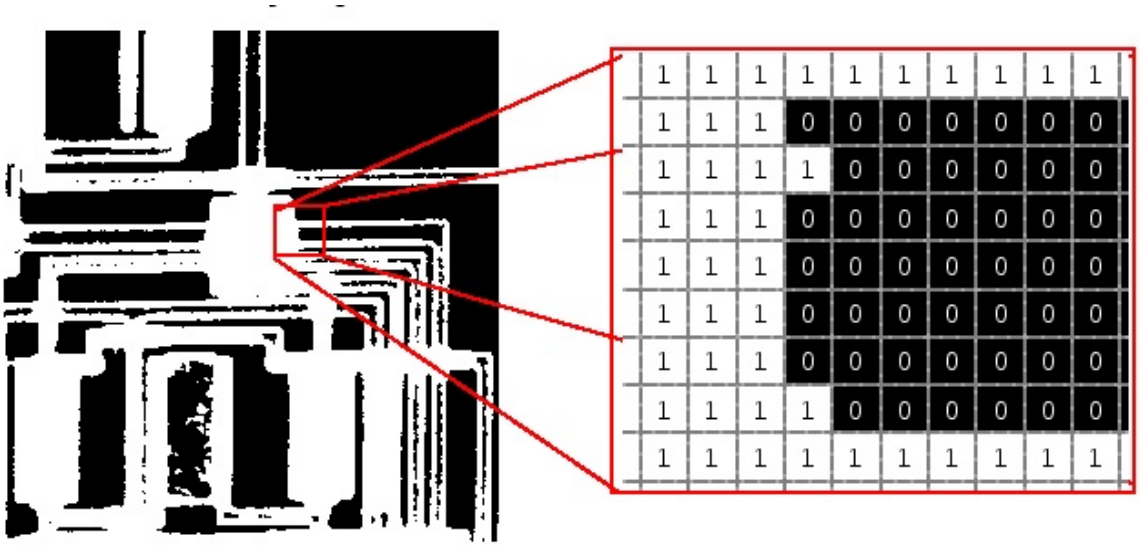
In [37]:
```
# reverse using Colon

arr=np.array([[1,2,3,4],[5,6,7,8],[10,11,12,13]])

print("Actual array:\n",arr)

print("\nDisplay using Colon:\n",arr[:,:])

print("\n Row wise reverse:\n",arr[::-1,:]) #arr[row,col]
```

```
print("\n column wise reverse:\n",arr[:,::-1]) #arr[row,col]

print("\n Row and column wise reverse:\n",arr[::-1,::-1]) #arr[row,col]
```

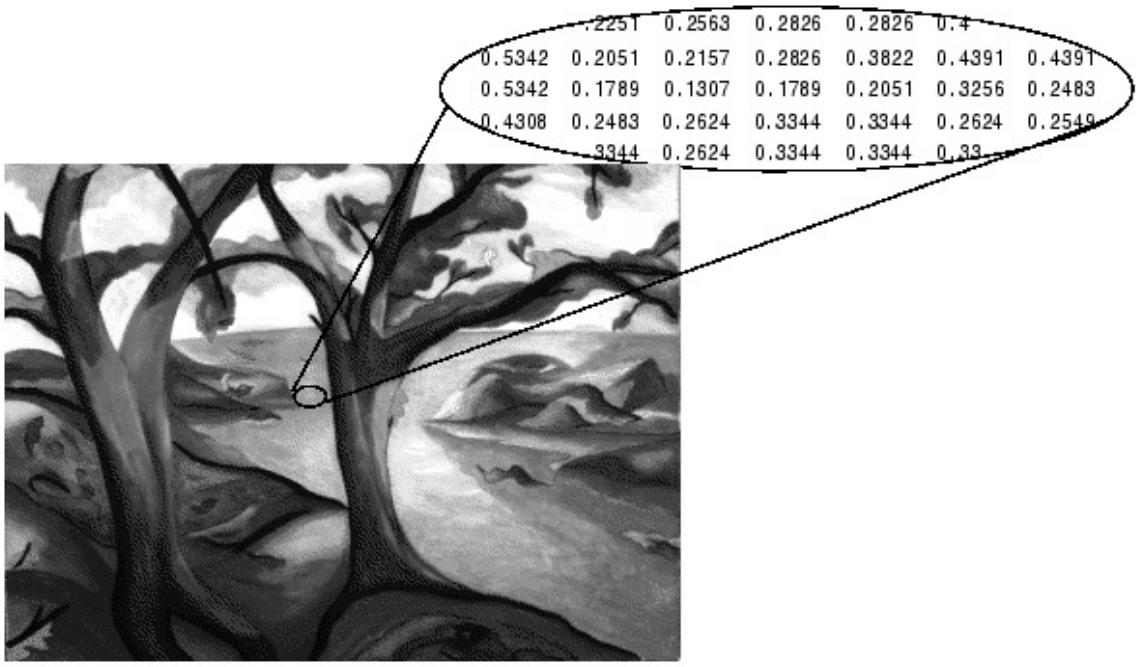
```
Actual array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [10 11 12 13]]

Display using Colon:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [10 11 12 13]]

 Row wise reverse:
 [[10 11 12 13]
 [ 5  6  7  8]
 [ 1  2  3  4]]

 column wise reverse:
 [[ 4  3  2  1]
 [ 8  7  6  5]
 [13 12 11 10]]

 Row and column wise reverse:
 [[13 12 11 10]
 [ 8  7  6  5]
 [ 4  3  2  1]]
```

Introduction to image processing

1. Black&White Image



2. Gray Image

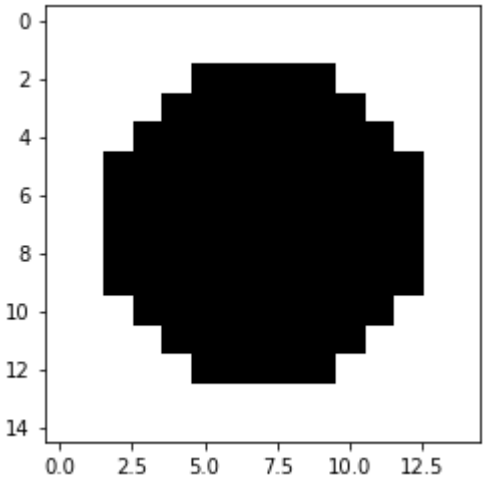


3. Color Image

Read, display and write an image.

In [5]:
```
import cv2
import matplotlib.pyplot as plt

image3=cv2.imread("smallimage.jpg",0)

cv2.imshow("small image",image3)

print(type(image3))

print()

print(image3)

img3=cv2.cvtColor(image3, cv2.COLOR_BGR2RGB)

plt.imshow(img3)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

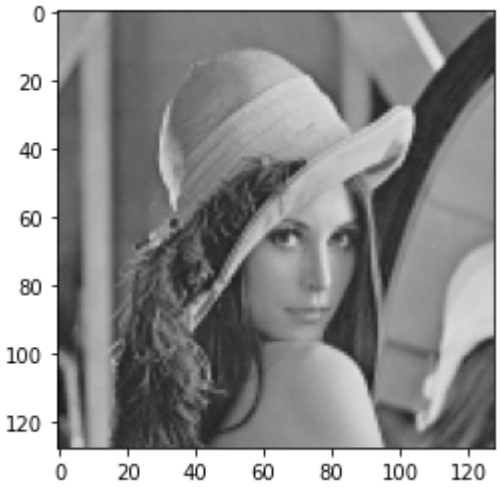
```
<class 'numpy.ndarray'>

[[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255]
 [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255]
 [255 255 255 255 255   0   0   0   0   0 255 255 255 255 255 255]
 [255 255 255 255   0   0   0   0   0   0   0 255 255 255 255 255]
 [255 255 255   0   0   0   0   0   0   0   0   0 255 255 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255 255]
 [255 255 255   0   0   0   0   0   0   0   0   0 255 255 255 255]
 [255 255 255 255   0   0   0   0   0   0   0 255 255 255 255 255]
 [255 255 255 255 255   0   0   0   0   0 255 255 255 255 255 255]
 [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255]
 [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255]]
```



In [7]:
```
import cv2
import matplotlib.pyplot as plt

img=cv2.imread("lena_gray_256.jpg",0)

image3 = cv2.resize(img, (128,128))

cv2.imshow("small image",image3)

print(type(image3))
```

```
print()

cv2.imwrite("image.jpg", image3)

print(image3)
image3=cv2.cvtColor(image3,cv2.COLOR_BGR2RGB)
plt.imshow(image3)

cv2.destroyAllWindows()
```

```
<class 'numpy.ndarray'>

[[162 163 164 ... 119 153 163]
 [162 161 161 ... 124 139 121]
 [156 157 159 ... 114  68  49]
 ...
 [ 53  55  62 ...  60  55  57]
 [ 52  52  58 ...  60  63  83]
 [ 49  52  55 ...  66  89 101]]
```



In [8]:

```
import cv2
image = cv2.imread("Veggi.png")

img_gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

print("width: {} pixels".format(image.shape[1]))

print("height: {} pixels".format(image.shape[0]))

print("channels: {}".format(image.shape[2]))

cv2.imwrite("Veggi_gray.png",img_gray)

cv2.imshow("Grayimg",img_gray)
cv2.imshow("Image", image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
width: 400 pixels
height: 302 pixels
channels: 3
```

Create synthesized white and black image filters and ramp image

In [9]:

```
import numpy as np
```

```
black_mask=np.zeros([512,512],dtype='uint8')

white_mask=np.ones([512,512,1],dtype='uint8')*255

ramp=np.zeros([256,256,1],dtype='uint8')


for i in range(1,256):
    for j in range(1,256):
        ramp[i,j]=j-1;

cv2.imshow("whiteimage",white_mask)
cv2.waitKey(0)

cv2.imshow("blackimage",black_mask)
cv2.waitKey(0)

cv2.imshow("Rampimage",ramp)
cv2.waitKey(0)



cv2.imwrite('whiteimage.jpg',white_mask)
cv2.imwrite('blackimage.jpg',black_mask)
cv2.imwrite('rampimage.jpg',ramp)


cv2.destroyAllWindows()
```

Display multiple Images together

In [10]:

```
# Display multiple Images as stack
img1=cv2.imread("blackimage.jpg")
img2=cv2.imread("whiteimage.jpg")
img3=cv2.imread("lena_gray_256.jpg")
img4=cv2.imread("rampimage.jpg")

print(img1.shape)
print(img2.shape)

#img1=cv2.resize(img1,(0,0),None,0.5,0.5)
#img2=cv2.resize(img2,(0,0),None,0.5,0.5)

ver=np.vstack((img3,img4))

hor=np.hstack((img1,img2))

cv2.imshow('Verticle',ver)
cv2.waitKey(0)

cv2.imshow('Horizontal',hor)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

```
(512, 512, 3)
(512, 512, 3)
```

Crop sub portion of image.

In [11]:

```
image1=cv2.imread('cameraman.tif')

image2=image1

subimage=image1[50:300,0:450] #; [rowstart:rowend,colstart:colend]


cv2.imshow("main image",image1)
cv2.waitKey(0)

cv2.imshow("sub image",subimage)
cv2.waitKey(0)


cv2.destroyAllWindows()
```

Image flip

In [12]:

```
image1=cv2.imread('lena_gray_256.jpg')

cv2.imshow("original image",image1)
cv2.waitKey(0)

i2=image1[::-1,:]
cv2.imshow("Vertical filp image",image1[::-1,:]) # reverse the row in this image
cv2.waitKey(0)

i3=image1[:,::-1]
cv2.imshow("Horizontal filp image",image1[:,::-1]) # reverse the coloumns in this im
cv2.waitKey(0)

i4=image1[::-1,::-1]
cv2.imshow("Horizontal & Vertical filp image",image1[::-1,::-1]) # reverse the rows
cv2.waitKey(0)

hor1=np.hstack((image1,i2))
hor2=np.hstack((i3,i4))

ver=np.vstack((hor1,hor2))

cv2.imshow('Verticle',ver)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

In [19]:

```
image1=cv2.imread('lena_gray_256.jpg')

cv2.imshow("original image",image1)
cv2.waitKey(0)

v_flip=np.flipud(image1)
cv2.imshow("Vertical filp image",v_flip) # reverse the row in this image
cv2.waitKey(0)


h_flip=np.fliplr(image1)
cv2.imshow("Horizonta filp image",h_flip) # reverse the row in this image
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Display image properties

In [13]:
```
image2=cv2.imread('lena_color_256.tif')

image3=cv2.imread('lena_gray_256.jpg',0)

print("Gray Image")
print(type(image3))
print(image3.shape)
print(image3.size)
print(image3.dtype)
height, width = image3.shape[:2]
print(height,width)

print("------------------------------")
print("Color Image")
print(type(image2))
print(image2.shape)
print(image2.size)
print(image2.dtype)
height, width = image2.shape[:2]
print(height,width)
```

```
Gray Image
<class 'numpy.ndarray'>
(256, 256)
65536
uint8
256 256
------------------------------
Color Image
<class 'numpy.ndarray'>
(256, 256, 3)
196608
uint8
256 256
```

Image color channels and conversion

In [14]:
```
import cv2
import numpy as np

image2=cv2.imread('lena_color_256.tif')
print(image2.shape)

zeros=np.zeros(image2.shape[:2],dtype="uint8")

# direct method in OpenCV is b,g,r=cv2.spilt()
blue = image2[:,:,0]

green = image2[:,:,1]

red = image2[:,:,2]

cv2.imshow("Oringinal image",image2)
cv2.waitKey(0)

cv2.imshow("Blue image",cv2.merge([blue,zeros,zeros]))
cv2.waitKey(0)

cv2.imshow("Green image",cv2.merge([zeros,green,zeros]))
cv2.waitKey(0)
```

```
cv2.imshow("Red image",cv2.merge([zeros,zeros,red]))
cv2.waitKey(0)

cv2.destroyAllWindows()
```

```
(256, 256, 3)
```

Negative image

In [15]:

```
image3=cv2.imread('lena_gray_256.jpg')
image5=image3.copy()

image5[::1]=abs(255-image3[::1])

cv2.imshow("gray image",image3)
cv2.waitKey(0)
cv2.imshow("gray image nagative",image5)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Image padding

In [16]:

```
# Image padding
import cv2
import numpy as np

img1=cv2.imread('lena_color_256.tif')

constant= cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER_CONSTANT,value=0)

cv2.imshow(" Pad image",constant)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Image Scalar Opertions

In [17]:

```
s1_img=cv2.imread('lena_gray_256.jpg')
img1=cv2.cvtColor(s1_img,cv2.COLOR_BGR2GRAY)

scalaradd=img1+25;

scalarsub=img1-25;

scalarmul=img1*0;

scalardiv=img1//2;

cv2.imshow("Original image",img1)
cv2.waitKey(0)
cv2.imshow("scalar image addition",scalaradd)
cv2.waitKey(0)
cv2.imshow("scalar image sub",scalarsub)
cv2.waitKey(0)
cv2.imshow("scalar image mul",scalarmul)
cv2.waitKey(0)
cv2.imshow("scalar image division",scalardiv)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

Image Opertions

In [18]:

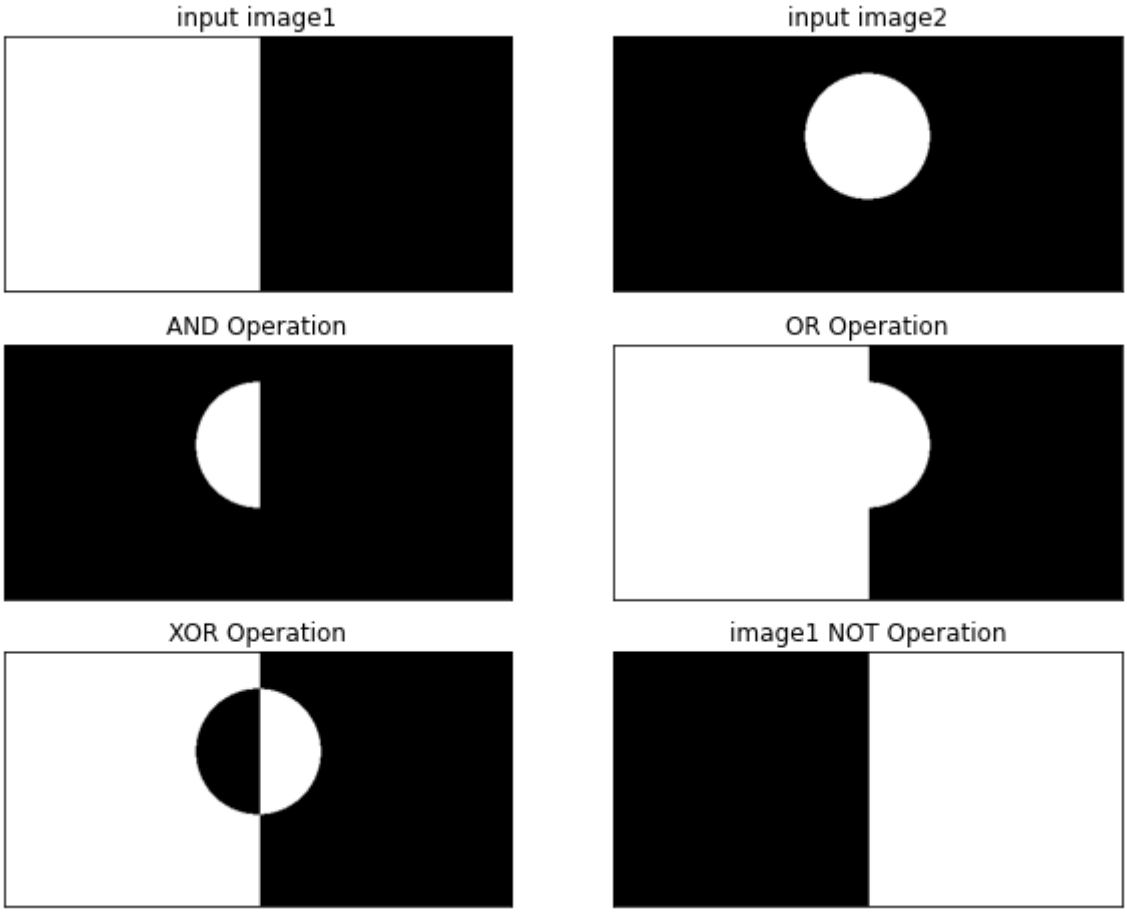
```
import cv2

s1_img=cv2.imread('lena_color_512.tif',0)

s2_img=cv2.imread('Myimage1111.jpg',0)

#image addition
addimg=cv2.add(s1_img,s2_img)

#image Substration
imagesubtract=cv2.subtract(s1_img,s2_img)

#image Multiplication
imagemulti=cv2.multiply(s1_img,s2_img)

#image Division
imagediv=cv2.divide(s1_img,s2_img)

cv2.imshow("image1",s1_img)
cv2.waitKey(0)
cv2.imshow("image2",s2_img)
cv2.waitKey(0)
cv2.imshow("added image",addimg)
cv2.waitKey(0)

cv2.imshow("cv2 subtration image",imagesubtract)
cv2.waitKey(0)

cv2.imshow("cv2 multiplication image",imagemulti)
cv2.waitKey(0)

cv2.imshow("cv2 Division image",imagediv)
cv2.waitKey(0)


cv2.destroyAllWindows()
```

Bitwise Operations on images

In [19]:

```
import matplotlib.pyplot as plt


def convert_rgb(img):
    imgg=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    return imgg

img1=cv2.imread("1bit1.png",0)
ret,img1 = cv2.threshold(img1,127,255,cv2.THRESH_BINARY)

img2=cv2.imread("2bit2.png",0)
ret,img2 = cv2.threshold(img2,127,255,cv2.THRESH_BINARY)

and_img= cv2.bitwise_and(img2, img1, mask = None)

or_img= cv2.bitwise_or(img2, img1, mask = None)
```

```
xor_img=cv2.bitwise_xor(img1, img2, mask = None)

not_img1=cv2.bitwise_not(img1, mask = None)

and_img=convert_rgb(and_img)
or_img=convert_rgb(or_img)
xor_img=convert_rgb(xor_img)
not_img1=convert_rgb(not_img1)

plt.figure(figsize=(10,8))
plt.subplot(3,2,1)
img1=convert_rgb(img1)
plt.imshow(img1)
plt.title('input image1')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,2)
img2=convert_rgb(img2)
plt.imshow(img2)
plt.title('input image2')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,3)
plt.imshow(and_img)
plt.title('AND Operation')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,4)
plt.imshow(or_img)
plt.title('OR Operation')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,5)
plt.imshow(xor_img)
plt.title('XOR Operation')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,6)
plt.imshow(not_img1)
plt.title('image1 NOT Operation')
plt.xticks([])
plt.yticks([])
plt.show()
```



- Image Geometric transformations

Translation

In [6]:

```
# Image Geometrical Operations # Traslate

image=cv2.imread('jeep.jpg')

def translate(image,x,y): # shifting

    trans_matrix=np.float32([[1,0,x],[0,1,y]])

    shifted=cv2.warpAffine(image,trans_matrix,(image.shape[1],image.shape[0]))

    return shifted;

#Image Translate
trans1=translate(image,0,50)
trans2=translate(image,0,-50)
trans3=translate(image,50,0)
trans4=translate(image,-50,0)

cv2.imshow("original image",image)
cv2.waitKey(0)
cv2.imshow("Down shift image",trans1)
cv2.waitKey(0)
cv2.imshow("Top shift image",trans2)
cv2.waitKey(0)
cv2.imshow("right shift image",trans3)
cv2.waitKey(0)
cv2.imshow("left shift image",trans4)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

Resize

In [7]:

```
# Image Geometrical Operations # resize

image=cv2.imread('jeep.jpg')


def resize(image,width=None,height=None):
    (h, w) = image.shape[:2]
    if width is None and height is None:
        return image
    if width is None:
        r = height / float(h)
        dim = (int(w * r), height)
    else:
        r = width / float(w)
        dim = (width, int(h * r))

    resized=cv2.resize(image,dim,interpolation=cv2.INTER_AREA)

    # interpolation method, which is the algorithm working behind the scenes to hand
    #   cv2.INTER_LINEAR, cv2.INTER_CUBIC, and cv2.INTER_NEAREST.
    return resized;

# Image resize
resizeimg1=resize(image,width=250,height=250)
resizeimg2=resize(image,width=800,height=800)
cv2.imshow("Original Image",image)
cv2.waitKey(0)
cv2.imshow("Resized Image1:",resizeimg1)
cv2.waitKey(0)
cv2.imshow("Resized Image2:",resizeimg2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Rotation

In [14]:

```
# Image Geometrical Operations rotation
import cv2

image=cv2.imread('lena_color_512.tif')

def rotate(image,angle,scale):
    (w,h)=image.shape[:2]

    rotate_m=cv2.getRotationMatrix2D((w//2,h//2),angle,scale) # (ratation point,angl

    rotated=cv2.warpAffine(image,rotate_m,(w,h))

    return rotated;

# Image Rotation
j=0;
for i in range(0,365,30):
    rotate1=rotate(image,-i,1) # +ve i Anti-clockwise
    cv2.imshow("Rotated images",rotate1)
    cv2.waitKey(0)
    if j==12:
```

```
        break;
    j+=1;

cv2.destroyAllWindows()
```

Drawing Commands

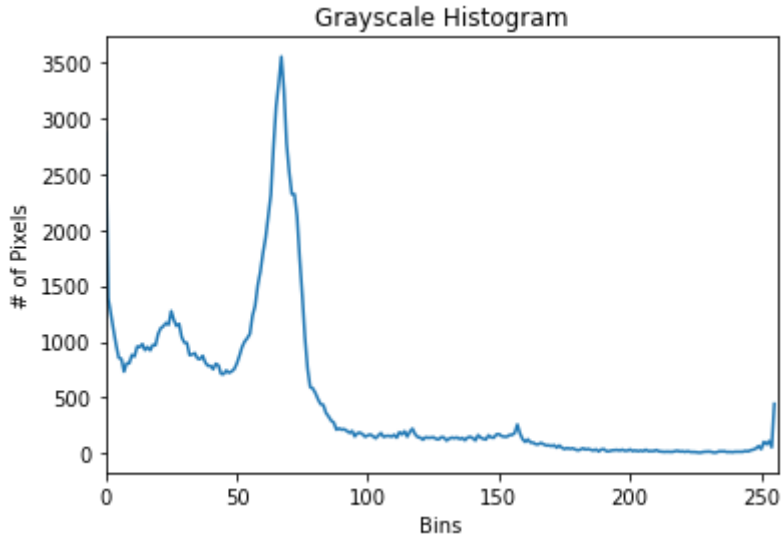
In [5]:

```
# Draw Shapes on image
import cv2
import numpy as np

sys_img=np.ones([512,512,3],dtype='uint8')*255

cv2.line(sys_img,(0, 0), (512, 512), (0,255,0),5)  #color channel (b,g,r)

#cv2.line(sys_img,(512, 0), (0, 512), (255,0,0),5)
cv2.rectangle(sys_img,(0, 0),(512, 512),(0,0,255),5)

for r in range(0,100,10):
    cv2.circle(sys_img, (256, 100), r, (0,0,0),2)

cv2.circle(sys_img, (256, 400),80,(0,255,0),-1)
cv2.rectangle(sys_img,(30,200),(150,300),(0,0,255),-1)
cv2.rectangle(sys_img,(330,200 ),(450, 300),(255,0,0),-1)

cv2.imwrite("draw_output.png",sys_img)

cv2.imshow("fig",sys_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

image histogram

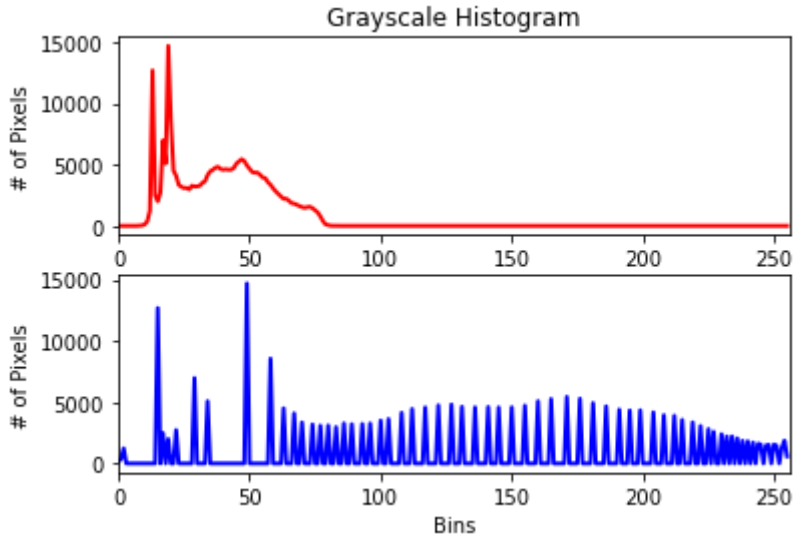
In [7]:

```
import cv2
from matplotlib import pyplot as plt

image1 = cv2.imread('123456.tiff')

cv2.imshow("Original", image)


hist = cv2.calcHist([image], [0], None, [256], [0, 256])


plt.figure()
plt.title("Grayscale Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.plot(hist)
plt.xlim([0, 256])
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```



In [15]:

```
import cv2
from matplotlib import pyplot as plt


image = cv2.imread('123456.tiff',0)

#histimg= cv2.imread('gimg1.jpg',0)
histimg=cv2.equalizeHist(image)

#cv2.calcHist(images,channels,mask,histSize,ranges)

hist1 = cv2.calcHist([image], [0], None, [256], [0, 256])

hist2 = cv2.calcHist([histimg], [0], None, [256], [0, 256])

cv2.imshow("Original image", image)
cv2.waitKey(0)
cv2.imshow("Histrogram equalized image ", histimg)
cv2.waitKey(0)

plt.figure(1)
plt.subplot(211)
plt.title("Grayscale Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.plot(hist1,'r',linewidth=2)
plt.xlim([0, 256])

plt.subplot(212)
#plt.title("Grayscale Histogram Equalize")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.plot(hist2,'b',linewidth=2)
plt.xlim([0, 256])
plt.show()

cv2.waitKey(0)
cv2.destroyAllWindows()
```



1. Write two functions for converting given gray image into black and white image. "blackandwhite_loop" function do the operation through standard for loops. "blackandwhite_compr" function do the process using "np.where" filter. Finally, compare its execution times.

In [2]:

```
# Example program: convert image to black and white
import cv2
import numpy as np
import time
import matplotlib.pyplot as plt

def blackandwhite_loop(img):
    width=img.shape[0]
    height=img.shape[1]
    image1=np.zeros([width,height], dtype="uint8")
    mean=np.mean(img)

    start=time.time()
    ## YOUR CODE STARTS HERE
    for i in np.arange(width):
        for j in np.arange(height):
            if image[i,j] > mean:
                image1[i,j]=255
    ## YOUR CODE ENDS HERE

    end=time.time()
    runtime=(end-start)
    print("Time taken for execution in loops :",runtime,"Sec")
    return image1

def blackandwhite_compr(img1):
    ## YOUR CODE STARTS HERE
    width=img1.shape[0]
    height=img1.shape[1]
    imge1=np.zeros([width,height], dtype="uint8")
    mean1=np.mean(img1)

    start1=time.time()

    imge1=np.where(img1>int(mean1),255,0)

    end1=time.time()
    runtime1=(end1-start1)
```

```
    ## YOUR CODE STARTS HERE
    print("Time taken for execution list comprehension :",runtime1,"Sec")
    return img1

image=cv2.imread('cameraman.tif',0)
#image1=cv2.imread('cameraman.tif')


# calling normal function
res1=blackandwhite_loop(image)
h_res1=np.hstack((image,res1))

#call list comprehension with lamda
res2=blackandwhite_compr(image)
h_res2=np.hstack((image,res1))

cv2.imshow("Result image using loop operation:",h_res1)
cv2.waitKey(0)
cv2.imshow("Result image using list comprehension:",h_res2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Time taken for execution in loops : 0.5255954265594482 Sec
Time taken for execution list comprehension : 0.0009951591491699219 Sec
```

2Q. Write a program to create below display image. Use lena_color_256.tif

In [3]:

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

im=cv2.imread('lena_color_256.tif')
width,hieght,ch=im.shape

## YOUR CODE STARTS HERE
filter_img=np.zeros([width,hieght,ch], dtype='uint8')
cv2.circle(filter_img, (width//2,hieght//2),120,(255,255,255),-1)
final_img=cv2.bitwise_and(im,filter_img)
## YOUR CODE ENDS HERE

final_img=cv2.cvtColor(final_img,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8,8))
plt.axis('off')
plt.imshow(final_img)
```

Out[3]: <matplotlib.image.AxesImage at 0x1dbc29a6ba8>

2. Write a python program to play a video file in mirror image using opencv.

In [22]:

```
# importing libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from IPython.core.display import Image
%matplotlib inline

# Create a VideoCapture object and read from input file
cap = cv2.VideoCapture(0)  # 0 for Webcam or 1 for external cam or give path of a vi

# Check if camera opened successfully
if (cap.isOpened()== False):
    print("Error opening video  file")

print("Press Q for stop")
# Read until video is completed

while(cap.isOpened()):
    # Capture frame-by-frame
    ret, frame = cap.read()

    if ret == True:

        mirrorframe=frame[:,::-1]

        # Display the resulting frame
        cv2.imshow('Frame', frame)

        cv2.imshow('Mirror Frame', mirrorframe)

        # Press Q on keyboard to  exit
```

```
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Break the loop
    else:
        break

# When everything done, release
# the video capture object
cap.release()

# Closes all the frames
cv2.destroyAllWindows()
```

```
Press Q for stop
```

3Q.Real-Time Edge Detection using OpenCV in Python | Canny edge detection method.

In [23]:

```
# OpenCV program to perform Edge detection in real time
# import libraries of python OpenCV

# where its functionality resides
import cv2

# np is an alias pointing to numpy library
import numpy as np

 #capture frames from a camera
cap = cv2.VideoCapture(0)

print("Press q for exit")

 # loop runs if capturing has been initialized
while(1):

    # reads frames from a camera
    ret, frame = cap.read()

    # converting BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define range of red color in HSV
    lower_red = np.array([30,150,50])
    upper_red = np.array([255,255,180])

    # create a red HSV colour boundary and
    # threshold HSV image
    mask = cv2.inRange(hsv, lower_red, upper_red)

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame,frame, mask= mask)

    # Display an original image
    cv2.imshow('Original',frame)

    # finds edges in the input image image and
    # marks them in the output map edges
    edges = cv2.Canny(frame,100,200)

    # Display edges in a frame
    cv2.imshow('Edges',edges)
```

```
    # Wait for Esc key to stop
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break


# Close the window
cap.release()

# De-allocate any associated memory usage
cv2.destroyAllWindows()
```

```
Press q for exit
```

In [ ]: