Vision 2.0 Workshop Computer Vision

Python Revision

Q1: What is the difference between

```
a = 42

print (a/14)

print (a//14)
```

```
a = 42

print (a/13)   # Normal division always gives float
>>>  3.230769230769231

print (a//13)  # Integer division gives quotient only
>>> 3
```

Q2: What will happen

```
a = [1,2,46]
b = np.array( [1,2,46] )


print (a  + 32)


print (b  + 32)
```

```
a = [1,2,46]
b = np.array( [1,2,46] )

print (a  + 32)
>>> TypeError: can only concatenate list (not "int") to list

print (b  + 32)      # Vectorization
>>> np.array([33,34,78 ])
```

Q3: What will happen

```
a = ( ‘sad duck’, 77, False )
a[-1] = ‘pringles’
b = ‘candy’


print (a)
print (b[:-2] + ‘sister’[1:] )
```

```
a = ( ‘sad duck’, 77, False )
a[-1] = ‘pringles’
>>> TypeError: 'tuple' object does not support item assignment


print (b[:-2] + ‘sister’[1:] )
>>> canister
```

Q4: What will happen

```
a = np.arange(60).reshape(10,2,3)
b = a [ : , 1: -1,  1]


print (b.shape)
print (b.T.shape)
```

```
a = np.arange(60).reshape(5,4,3)
b = a [ : , 1: 3,  1]



print (b.shape)    >>> (5,2)
print (b.T.shape) >>> (2,5)
```

Q5: What will happen

```
a = np.eye(2)
b = np.ones((2,2))

print (a * b)
```

Q6: What will happen

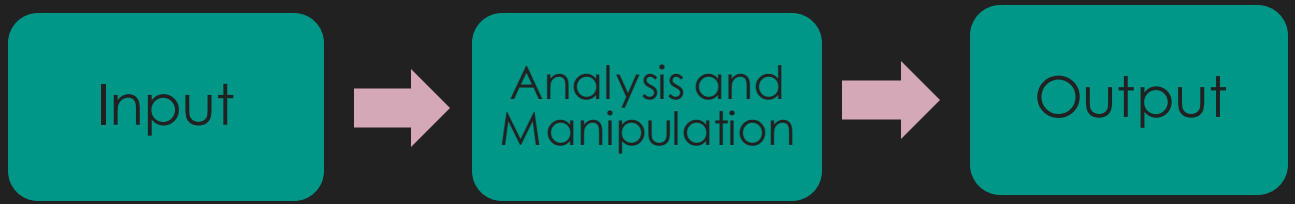
```
a = np.eye(2)
b = np.ones((2,2))


print (a * b)
>>> np.array([   [ 1 , 0 ],
                 [ 0 , 1 ]   ])


print (a @ b) # What is the Output
```

Image Processing:

- Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it.
- Image processing basically includes the following three steps:



OpenCV

- OpenCV is an Image Processing library created by Intel and maintained by Willow Garage.
- Available for C, C++, and Python.
- Open Source and free.
- Easy to use and install

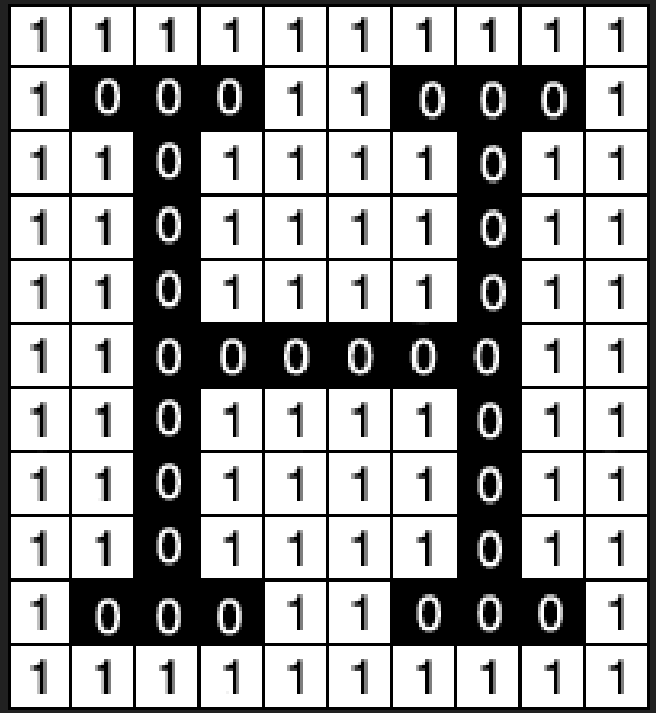
Images:

- Binary Image

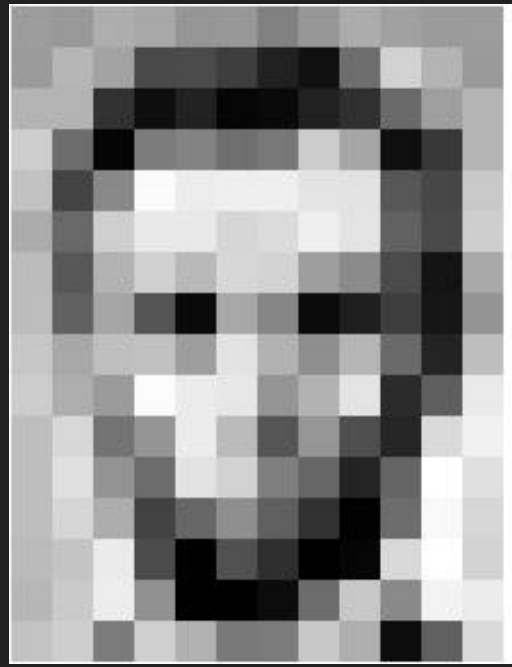
- Grayscale Image

- RGB Image

- Binary Image: (a x b) array of 0 or 1



- Grayscale Image: a x b image of values from **0-255**



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

RGB colour Format:

– Each pixel contains a vector representing red, green and blue components

RGB Components

10	10	16	28
9	65	70	56	43
15	32	99	70	56	78
32	21	60	90	96	67
	54	85	85	43	92
		32	65	87	99

Images in OpenCV:

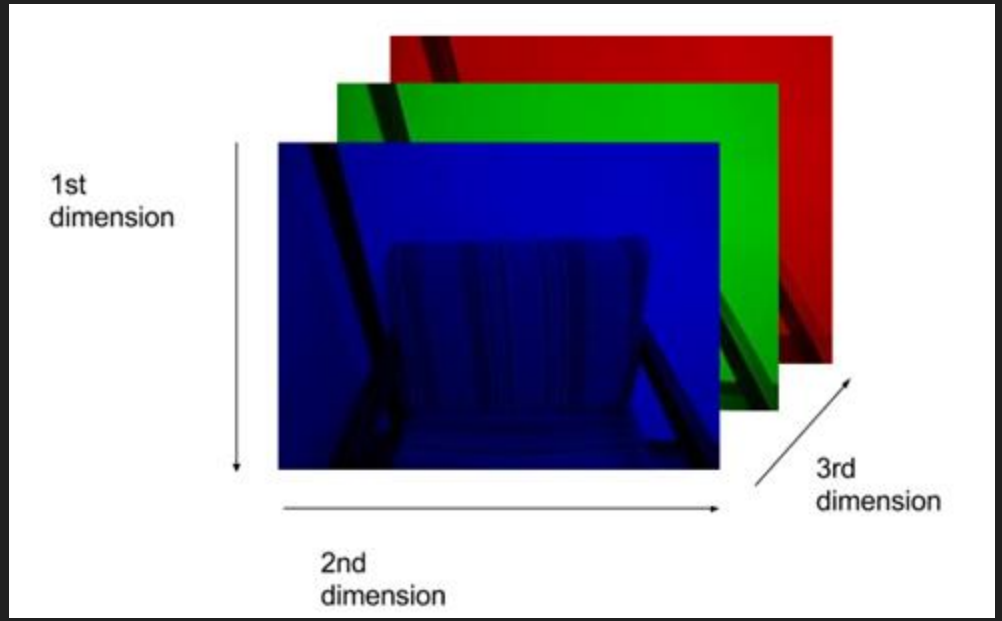
- **Data for each image is stored as a numpy array.**
- **Each image is seen as an array with dimensions equal to the** *number of pixel rows, number of pixel columns, number of channels.*
- **The indexing in an image is (y,x) or (row,column)**
- **Each pixel has a value of intensity.**
- Each element of the matrix contains the value of this intensity at the corresponding to the pixel it represents.

Images in OpenCV:

Binary Image	• All the elements of the matrix are either zero or one. • Zero represents black and 1 represents white.
Grayscale Image	• All the elements of the matrix lie between 0 and 255. • Zero represents Black, 255 represents White and the intermediate values represent shades of Gray.
RGB Image	• Each colour has a specific RGB value! • RGB Images are seen as 3D matrices with the 1st plane corresponding to R, 2nd to G and 3rd to B.

Images in OPENCV:

- BGR Image: a x b x 3 (0-255)



cv2.imread() : Read the image

- image = cv2.imread('C:/mypath/myimage.png',flag)
- First argument is the name of image. The full path of image should be given or relative path.
- Second argument is a flag which specifies the way image should be read.
  - cv2.IMREAD_COLOR Flag : 1
  - cv2.IMREAD_GRAYSCALE Flag : 0

cv2.imshow() : Display the image

- First argument is a window name which is a string.
- Second argument is our image.

```
import numpy as np
import cv2

# Load an color image in grayscale
img = cv2.imread('1.JPG',0)

cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The cv2.**waitKey**(n) function in **OpenCV** is **used** to introduce a delay of n milliseconds while rendering images to windows. cv2.Waitkey(0) shows image until you close it.

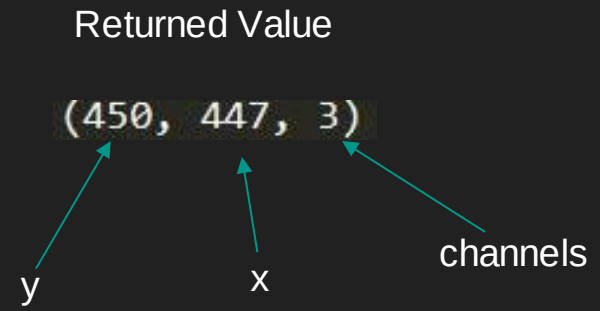
Numpy shape property:

- We can check the dimension of the image by the shape property in numpy.

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


img=cv2.imread("capture.jpg",1)

print(img.shape)
```



cv2.imwrite() : Saves the image

- First argument is file name which is a string.
- Second argument is the image you want to save.

```
cv2.imwrite('image.png',img)
```

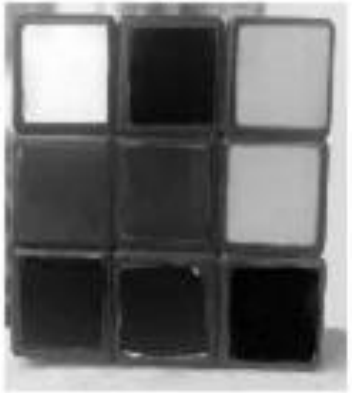
This will save the image in PNG format in the working directory.

What will be shown?

```
img = cv2.imread (‘cube.png’)
layer = img[ : , : ,2]
cv2.imshow(‘mywindow’, layer)
cv2.waitKey(0)
```

Normal

B



G

R

Resizing and Cropping

- Resizing is done by cv2.resize() function with arguments as image and shape of desired image.
- `new_img=cv2.resize(old_img,(new_x,new_y) )` #to a given size
- `new_img=cv2.resize(old_img, (0,0), fx=0.5, fy=0.5)` #to half size

↑ ----- it's a placeholder

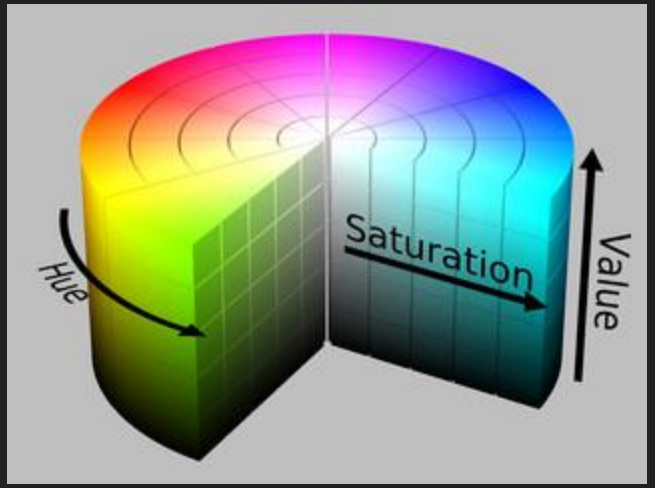
- Cropping is done using NumPy slicing.

```
crop_part = img[10:900, 120:700](First y then x)
```

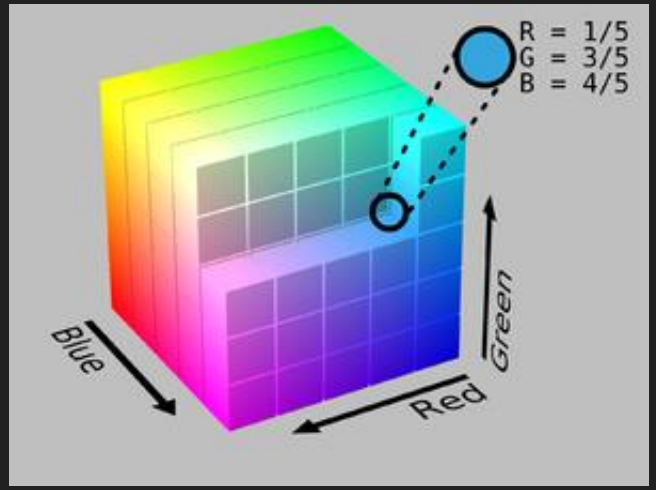
Changing Colorspaces

- `newimg=cv2.cvtColor( img, FLAG )` is used to convert image from one color space to other .

- Some Examples for FLAG:
  - cv2.COLOR_BGR2GRAY converts BGR image to Grayscale.
  - cv2.COLOR_BGR2RGB converts BGR to RGB image.
  - cv2.COLOR_BGR2HSV converts BGR to HSV.



HSV



RGB

- Hue: The dominant color perceived by the viewer.
- Saturation: Intensity of the color.
- Value: The chromatic notion of intensity.

Thresholding (grayscale):

- Thresholding is the simplest segmentation method.
- Binary thresholding is applied on grayscale images.
- The pixels are partitioned depending on their grayscale values.
- If the pixel value is greater than a particular value(T here) then it is set to 1 otherwise it is set to 0:

$$g(x, y) = \begin{cases} \text{True or 1,} & \text{if } f(x,y) > T \\ \text{False or 0,} & \text{if } f(x,y) <= T \end{cases}$$

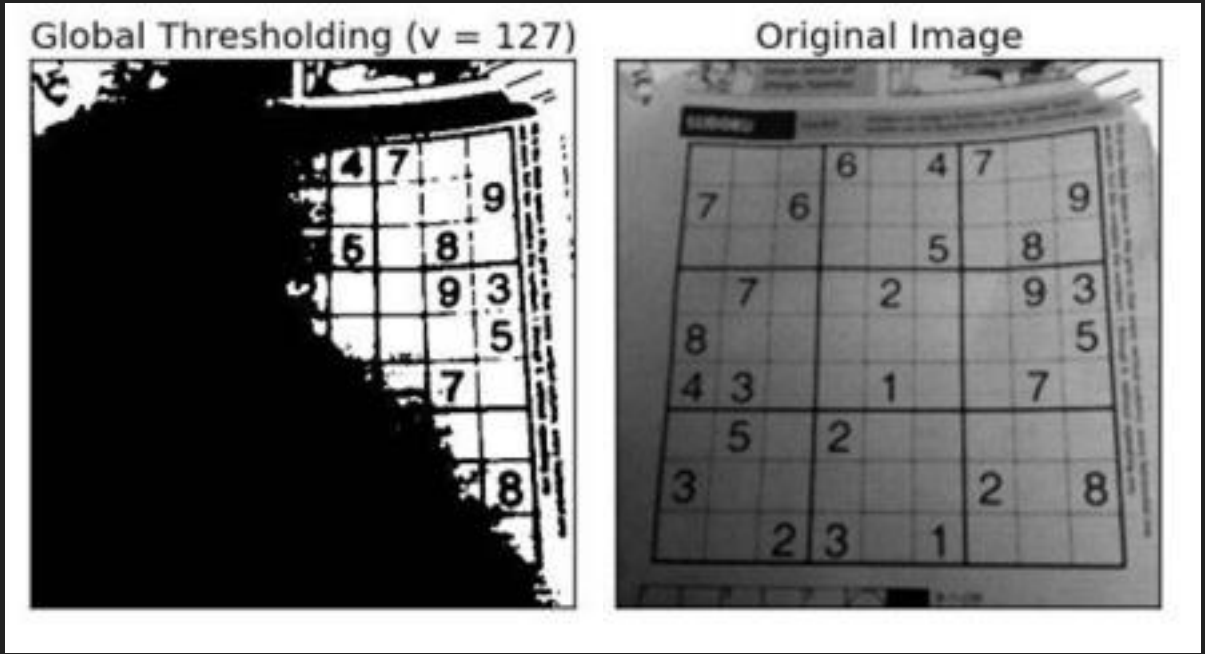
Thresholding(grayscale) code:

- In binary thresholding, basically a grayscale image is converted into a binary image.

```
_,threshold=cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

global_thresh = orig_imag > 127

Black and White Grayscale



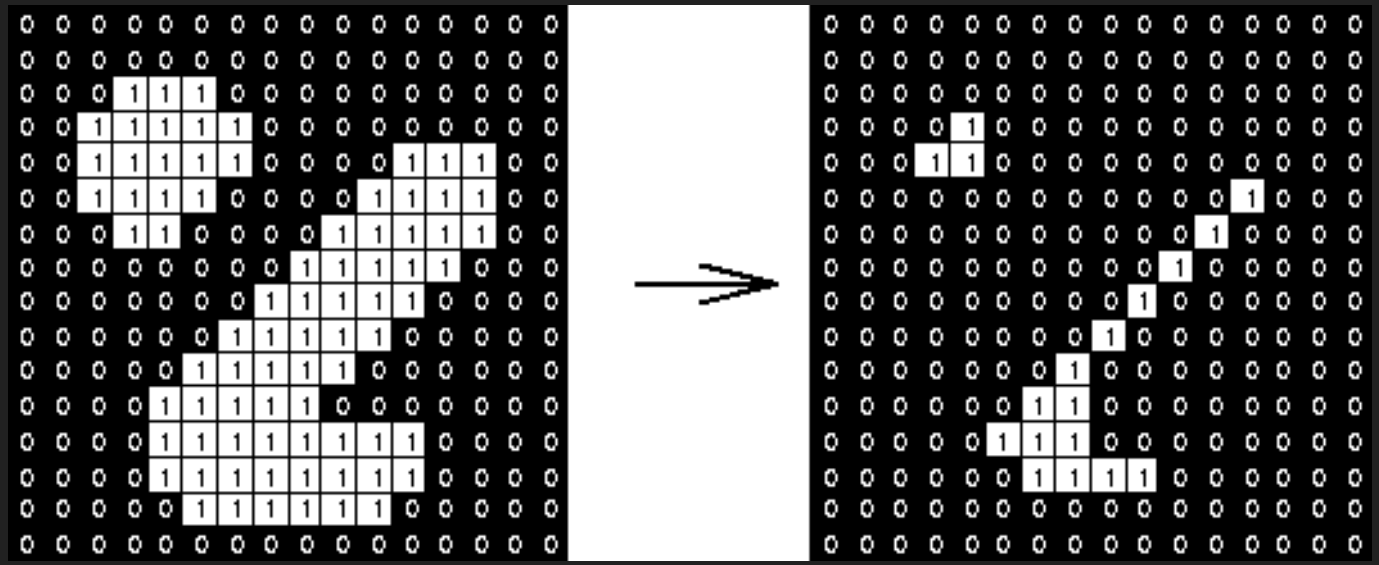
Masking:

- Masking means making a binary mask for a a desiered color.
- Masking is preferably done on HSV images because it is easy to choose color in HSV format.
- We have define a lower_value and an upper_value. All the colors which lie in the specified range are given value 1 in the mask and the rest are given 0.
- ```
  lower_blue = np.array([100,200,0])
  upper_blue = np.array([200,255,55]
  newblue =cv2.inRange(new,lower_blue,upper_blue)
  ```
  ```

Masking(code):

```
img=cv2.imread("capture.jpg",1)
hsv=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
lower_red=np.array([0,100,0])
upper_red=np.array([30,255,255])
mask=cv2.inRange(hsv,lower_red,upper_red)
```

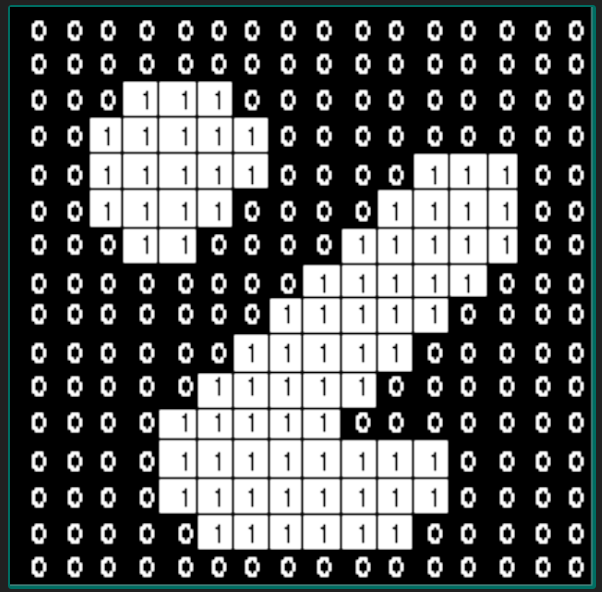
We chose this color

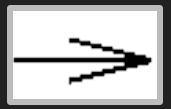


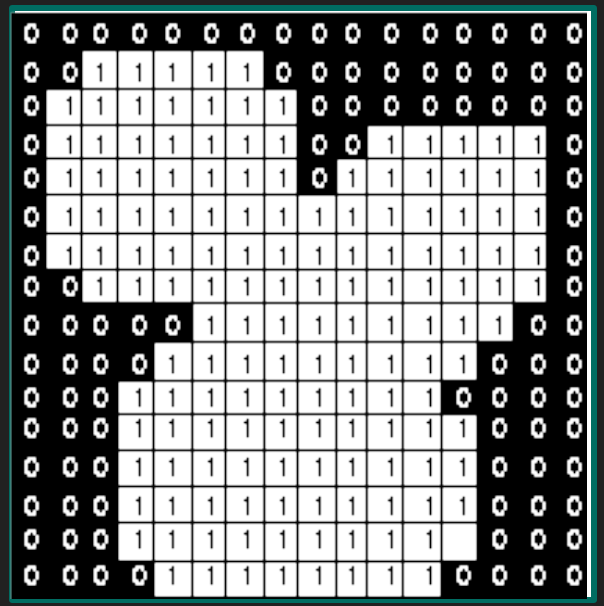
Morphological Operations

- Dilation:
  - Just opposite of erosion i.e. increases white area.

```
kernel = np.ones((5,5),np.uint8)
```



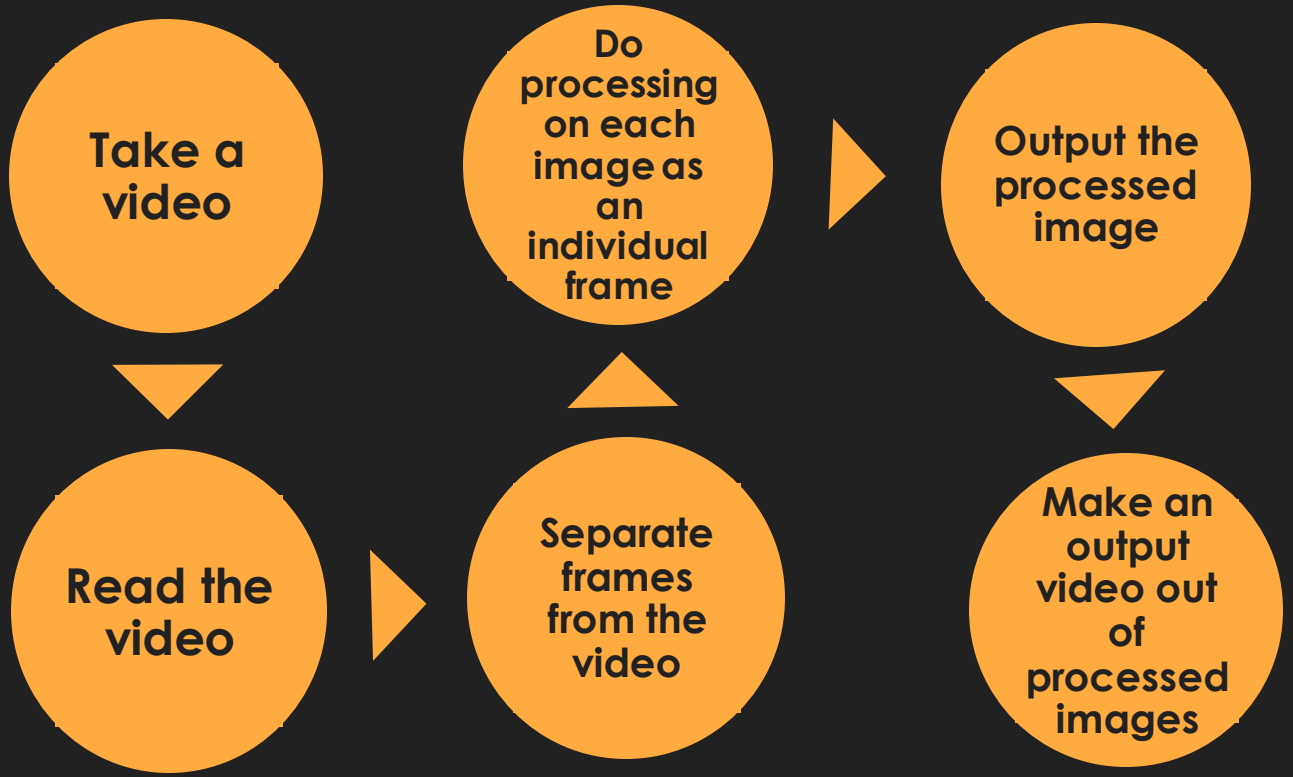




Video Processing:

- Videos are extension of images in 4th dimension.
- Frames can be BGR image or Grayscale or Binary image.

Video Processing:



Video Processing:

- To capture a video, you need to create a **VideoCapture** object.
- Its argument can be either the device index or the name of a video file.

```
cap = cv2.VideoCapture(0)
```

```
video = cv2.VideoCapture('begin.mkv')
while True:
    ret,img = video.read()
    if ret == False:
            break
    # Processing Here
    cv2.imshow('vid',img)
    k=cv2.waitKey(1)
    if(k==ord('q')):
      cap.release()
      cv2.destroyAllWindows
      break
```

cap.read() function returns 2 values, one of them is a Boolean which is stored in ret.

The cv2.waitKey(n) function in OpenCV is used to introduce a delay of n milliseconds while rendering images to windows. cv2.Waitkey(0) shows image until you close it.