GESTURE CONTROLLED ROBOTICS

USED THIS?

Original
X-Pro II
Lomo-fi
Earlybird
Sutro
Lily
Earlybird
Sutro
Toaster
Brannan
Inkwell
Walden
Hefe
Apollo
Walden
Hefe
Poprocket
Nashville
Gotham
1977
Lord Kelvin
Lomo-fi
1977
Lord Kelvin

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.

What is an Image?

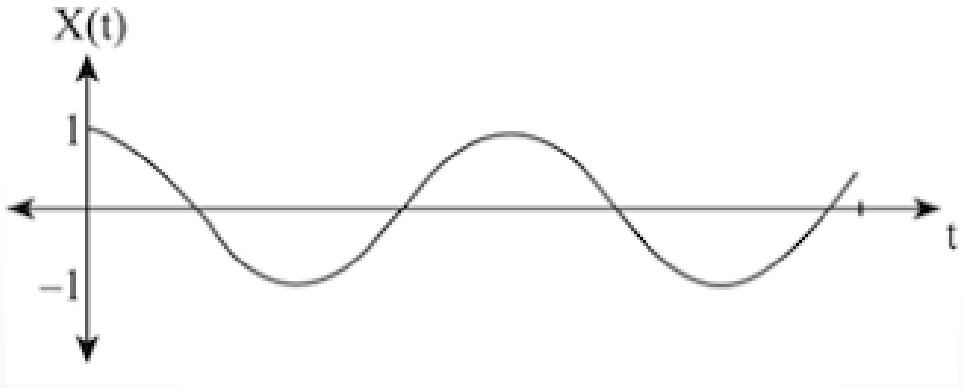
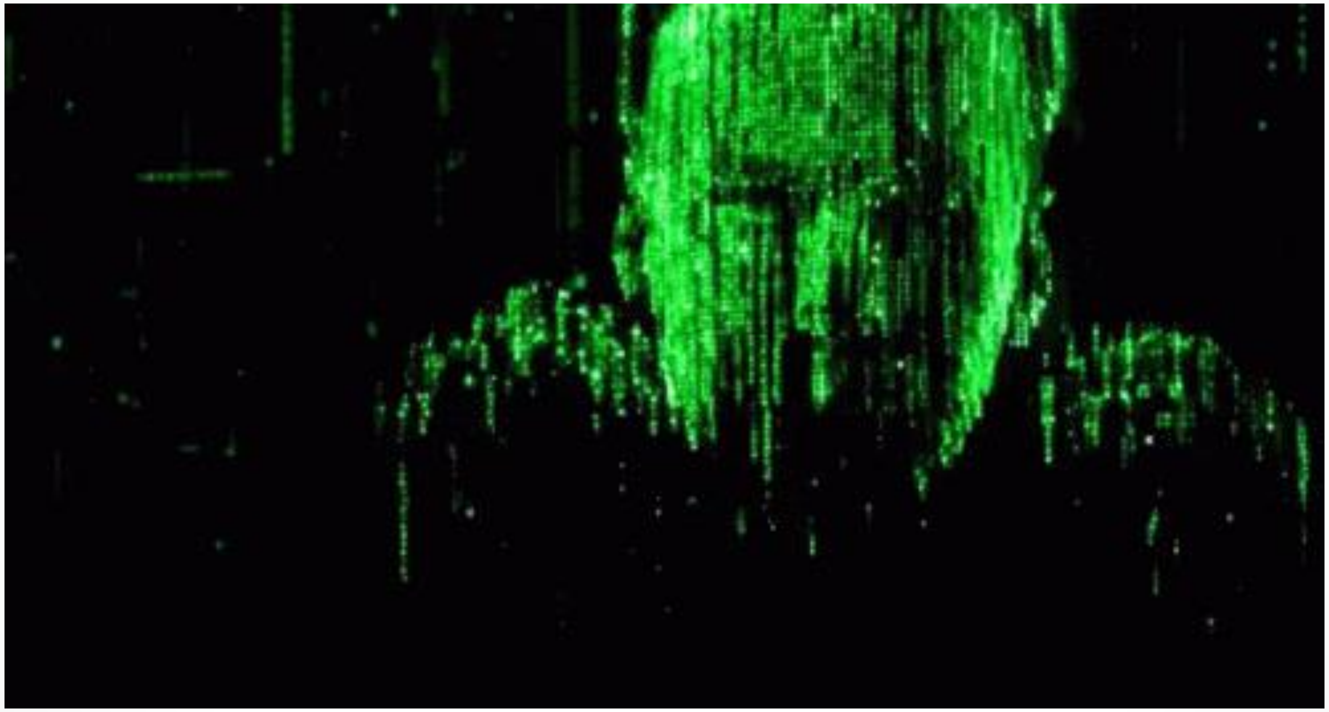
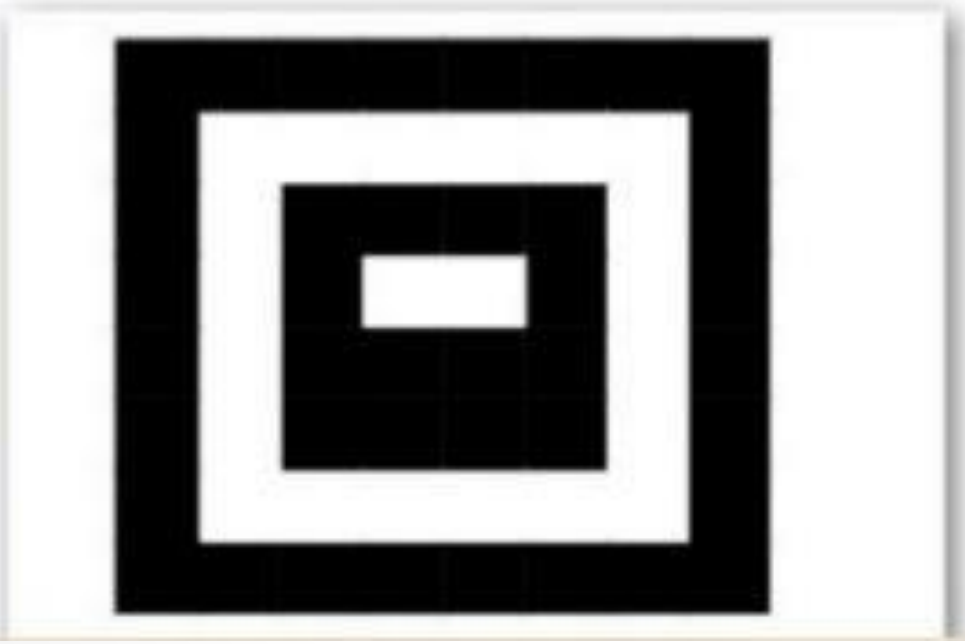


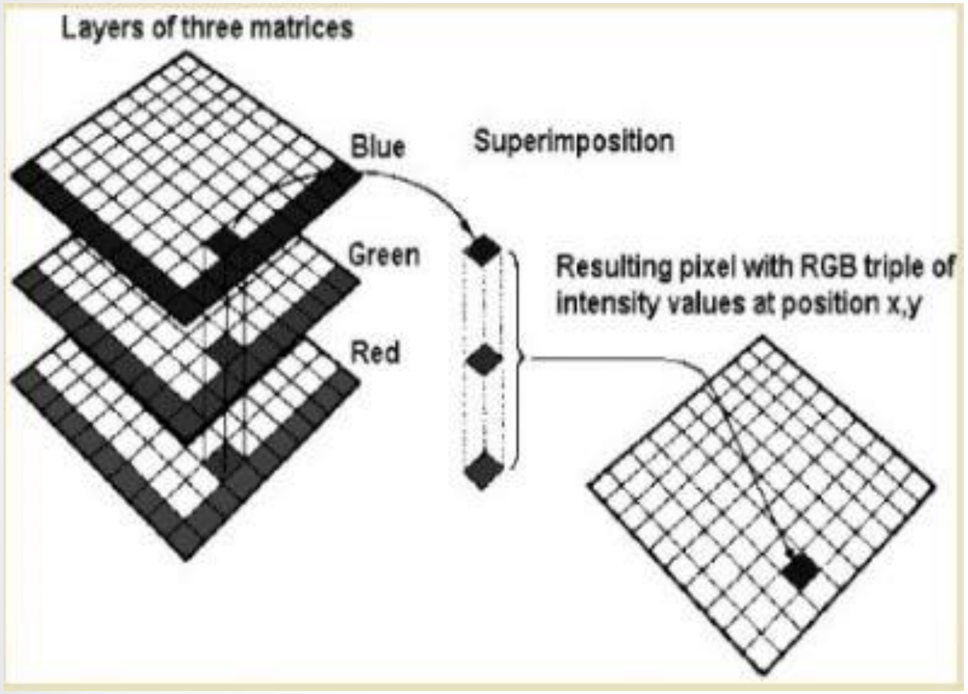
Image can be represented as a …

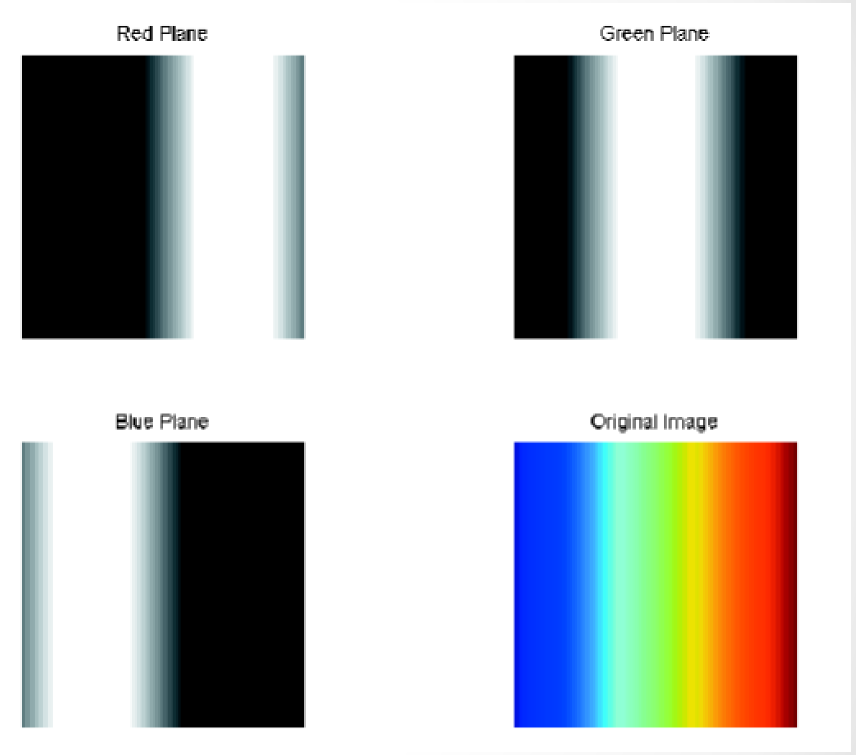


```
0, 0, 0, 0, 0, 0, 0, 0
0, 1, 1, 1, 1, 1, 1, 0
0, 1, 0, 0, 0, 0, 1, 0
0, 1, 0, 1, 1, 0, 1, 0
0, 1, 0, 0, 0, 0, 1, 0
0, 1, 0, 0, 0, 0, 1, 0
0, 1, 1, 1, 1, 1, 1, 0
0, 0, 0, 0, 0, 0, 0, 0
```

This matrix would correspond to the following image where 0=black and 1=white.




Layers of three matrices
Blue
Superimposition
Green
Resulting pixel with RGB triple of intensity values at position x,y
Red


Red Plane
Green Plane
Blue Plane
Original Image

OpenCV ?

- A Library
- Can be interfaced with C++, Java , Python

Basic Functions of Opencv

cv2.imread()

Use the function **cv2.imread()** to read an image. The image should be in the working directory or a full path of image should be given.

cv2.imshow

Use the function **cv2.imshow()** to display an image in a window. The window automatically fits to the image size.

cv2.waitKey()

The function waits for specified milliseconds for any keyboard event. If **0** is passed, it waits indefinitely for a key stroke

cv2.destroyAllWindows() simply destroys all the windows we created.

The function **cv2.imwrite()** to save an image.

First argument is the file name, second argument is the image you want to save.

Program to open an Image

```
#Prog_1
import cv2
img=cv2.imread('images.jpg')
cv2.imshow(‘Image',img)
cv2.waitKey(0)
```

`cv2.VideoCapture()`

- Pass 0 in the argument to use your laptop camera
- Pass 1 to use the second camera (in case you connect any other camera to the Laptop)
- Captures the video frame by frame

Video Capture

```
#Prog_2
import numpy as np
import cv2
cap = cv2.VideoCapture(0)

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

    # Display the resulting frame
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Drawing Functions

cv2.line(image , starting co-ord, final co-ord, bgr, thickness)

image : The image where you want to draw the shapes

bgr : Color of the shape, say (0,0,255)

thickness : Thickness of the line

Example : Draw a line

```
#Prog_3
import numpy as np
import cv2
# Create a black image
img = np.zeros((512,512,3), np.uint8)
# Draw a diagonal blue line with thickness of 5
px
img = cv2.line(img,(0,0),(511,511),(255,0,0),5)
cv2.imshow('Image',img)
cv2.waitKey(0)
```

Other functions

- `cv2.circle()`
- `cv2.rectangle()`
- `cv2.ellipse()`

Accessing and Modifying Pixel values

```
px = img[100,100]
print px [157 166 200]

 # accessing only blue pixel
 blue = img[100,100,0]
print blue
157
```

<u>You can modify the pixel values the same way.</u>

```
img[100,100] = [255,255,255]
print img[100,100]
[255 255 255]
```

Image Properties

Shape of image is accessed by img.shape. It returns a tuple of number of rows, columns and channels (if image is color):

```
print img.shape
(342, 548, 3)
```

Total number of pixels is accessed by img.size:

```
print img.size
562248
```

Image datatype is obtained by img.dtype:

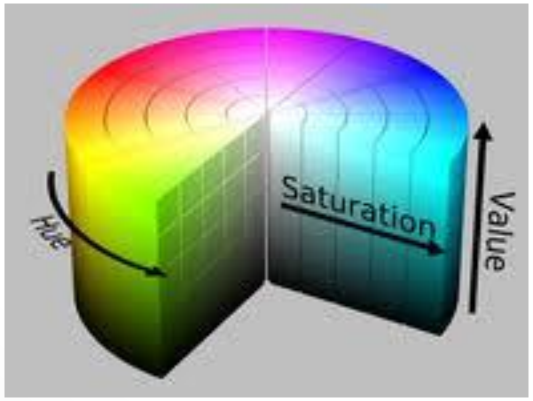
```
print img.dtype
uint8
```

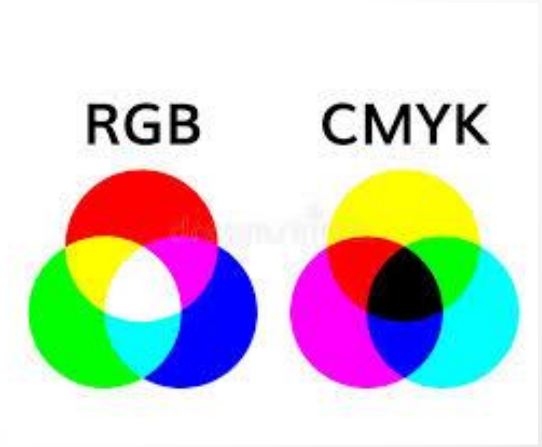
COLOR SPACES

A specific range of colours which can span the entire set of colours is called Colour space.

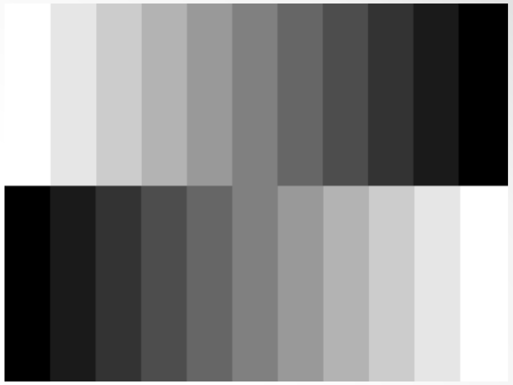
Some of the most used Colour Spaces are

- RGB
- sRGB
- CYMK
- HSV
- HSL





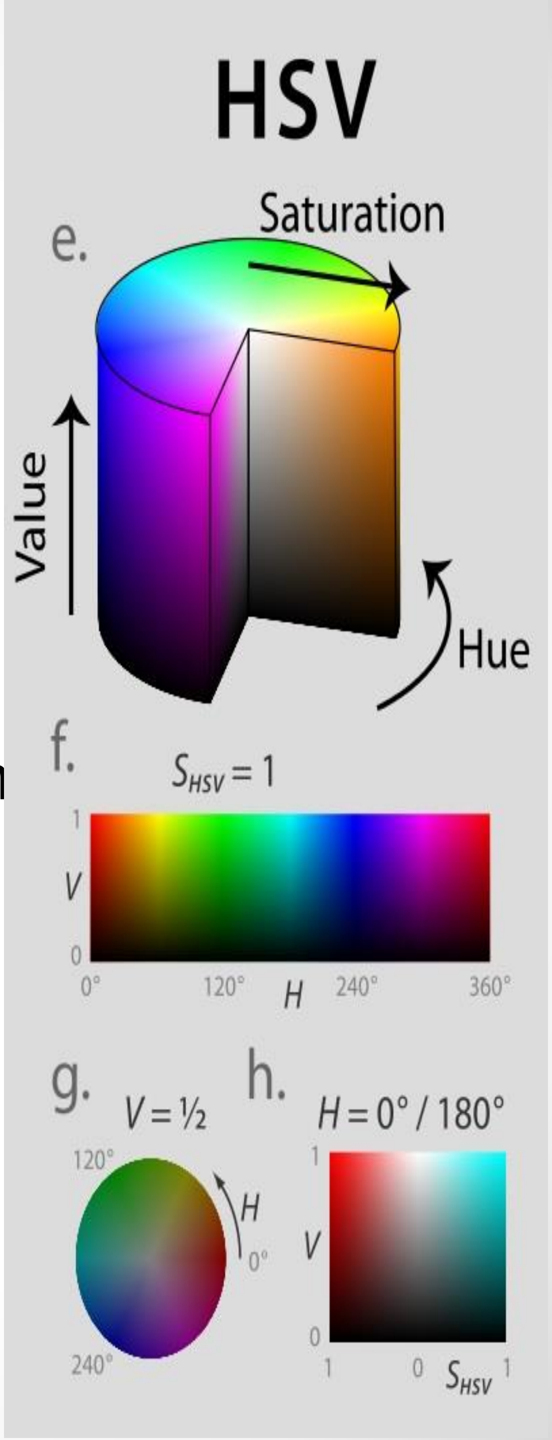
GRAYSCALE



- Another type of colour space is Grayscale
- Grayscale maps the intensity of each pixel from black to white
- It carries only intensity information and not colour information
- Thus 8 – Bit Grayscale has values from 0 – 255
- 0 corresponds to Black
- 255 corresponds to White
- Grayscale is generally used when colour information is not needed and where processing should be faster because we use only one channel now rather than 3 channels

HSV

- HSV is Hue Saturation Value
- **Hue** means the Colour content. It is measured in degrees
- **Saturation** is the colour concentration of the specific colour
- **Value** means the Brightness concentration of the colour specified
-



Program for Detecting Blue Color objects

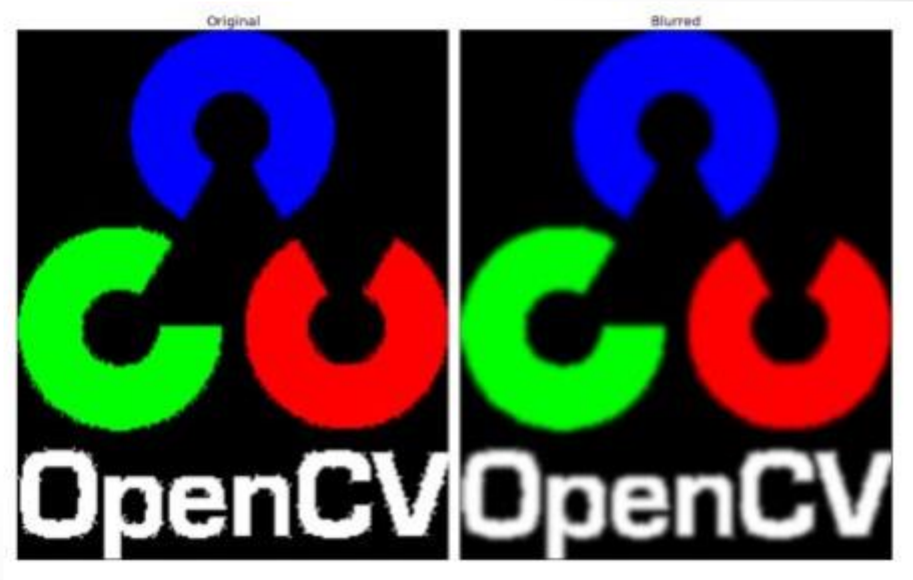
```
#Prog4
import cv2
import numpy as np
cam = cv2.VideoCapture(0)
while True:
        ret, frame = cam.read()
        hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
        lower_blue = np.array([100,60,50])
        upper_blue = np.array([130,255,255])

        #Creating a mask where the values in the range are made 255 others
are made 0
        mask = cv2.inRange(hsv , lower_blue, upper_blue)
        res = cv2.bitwise_and(frame, frame,mask = mask)
        cv2.imshow('Frame',frame)
        cv2.imshow('Smooth', mask)
        cv2.imshow('Filter', res)
        if cv2.waitKey(1) & 0xFF == 27:
                break
cam.release()
cv2.destroyAllWindows()
```

Image Blurring (Image Smoothing)

- Image blurring is achieved by convolving the image with a low-pass filter kernel.

- It is useful for removing noise. It actually removes high frequency content (e.g: noise, edges) from the image resulting in edges being blurred when this is filter is applied. (Well, there are blurring techniques which do not blur edges).

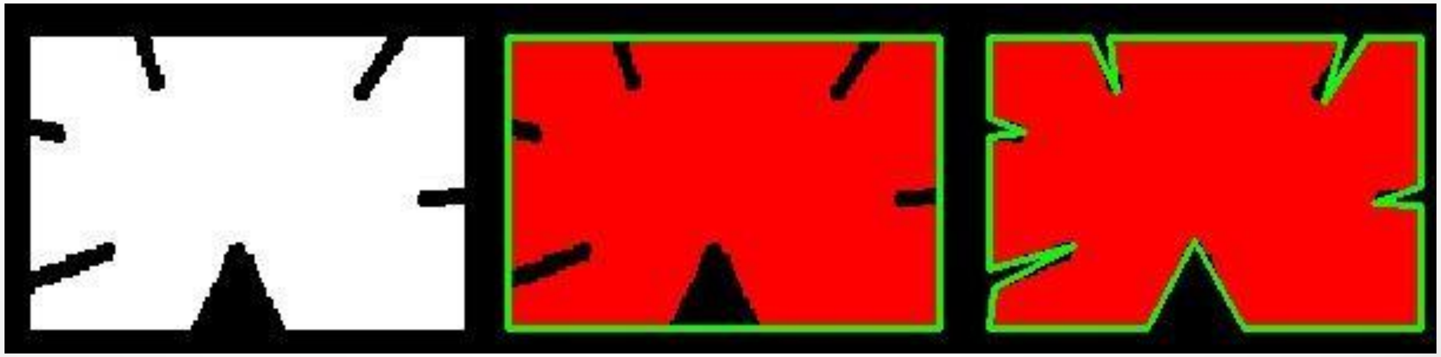
```
blur = cv2.GaussianBlur(img,(5,5),0)
```



CONTOURS

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.



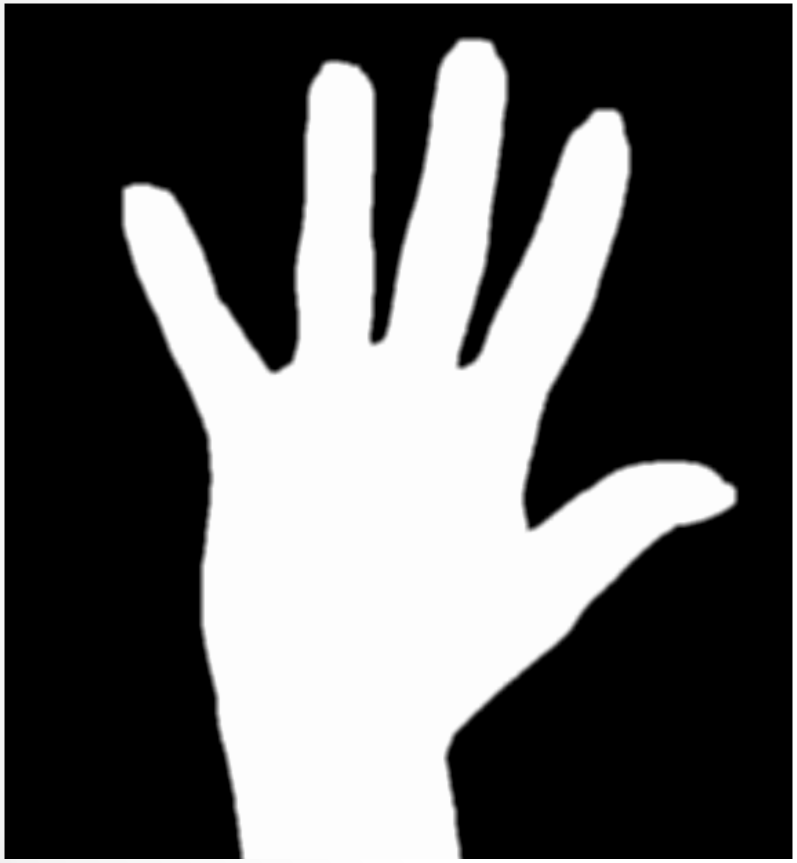
Program to Detect Contours

```
#Prog_5
import cv2
import numpy as np
im=cv2.imread('images.jpg')

imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(imgray,127,255,0)
image, contours, hierarchy =
   cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_A
   PPROX_SIMPLE)
im = cv2.drawContours(im, contours, -1,(0,255,0),
   3)
cv2.imshow('new',im)
cv2.waitKey(0)
```

Hand Detection

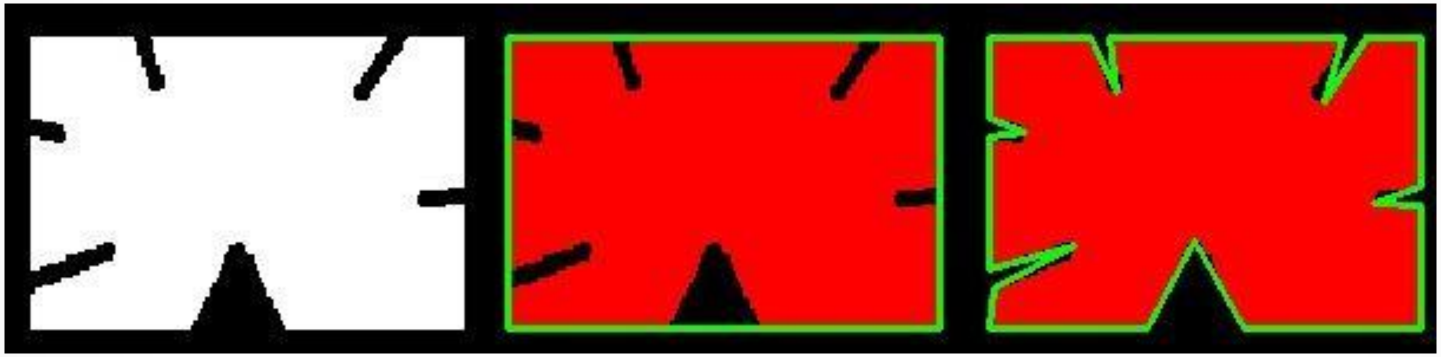
```
#Prog_6
import cv2
import numpy as np
frame= cv2.imread("hand.jpg")
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
kernel = np.ones((3,3),np.uint8)

# define range of skin color in HSV
lower_skin = np.array([0,30,70], dtype=np.uint8)
upper_skin = np.array([20,255,255], dtype=np.uint8)
```

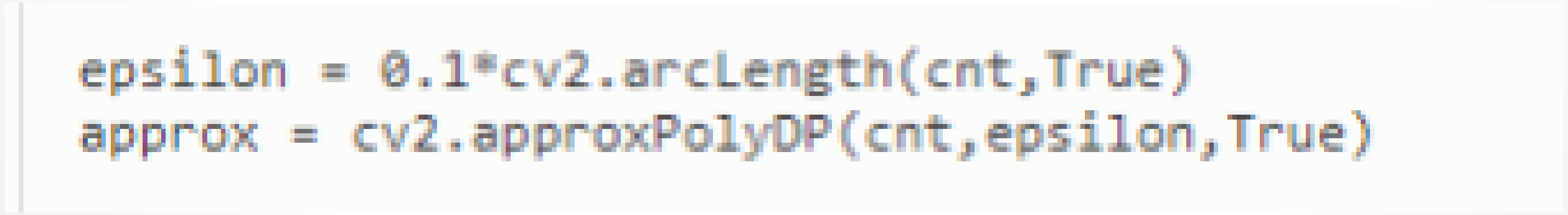
```
mask = cv2.inRange(hsv, lower_skin, upper_skin)
mask = cv2.dilate(mask,kernel,iterations = 4)
mask = cv2.GaussianBlur(mask,(5,5),100)
#find contours
_,contours,hierarchy=
cv2.findContours(mask,cv2.RETR_TREE,cv2.CHAIN_APPROX
_SIMPLE)
im = cv2.drawContours(frame, contours, -1,(0,255,0), 3)
cv2.imshow('Contour',im)
cv2.imshow('Mask',mask)
cv2.waitKey(0)
```



Contour Approximation

It approximates a contour shape to another shape with less number of vertices depending upon the precision we specify.



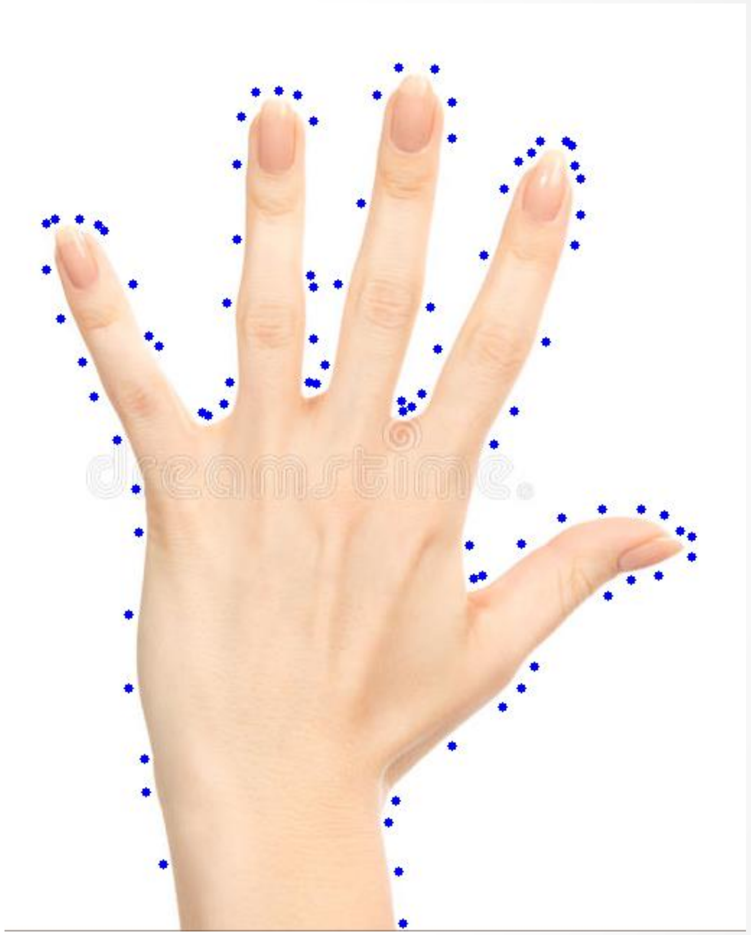


```
epsilon = 0.1*cv2.arcLength(cnt,True)
approx = cv2.approxPolyDP(cnt,epsilon,True)
```

Second image, green line shows the approximated curve for epsilon = 10% of arc length.

Third image shows the same for epsilon = 1% of the arc length

Actual Image



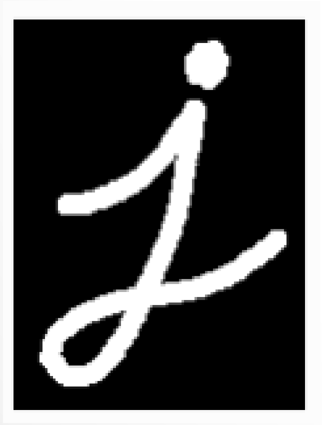
Approx contour points

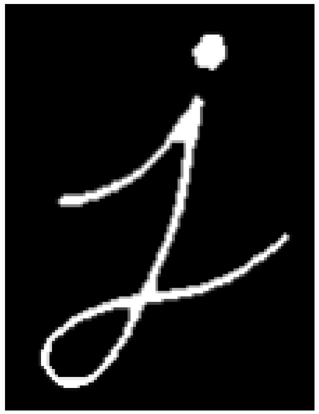
EROSION & DILATION

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images.

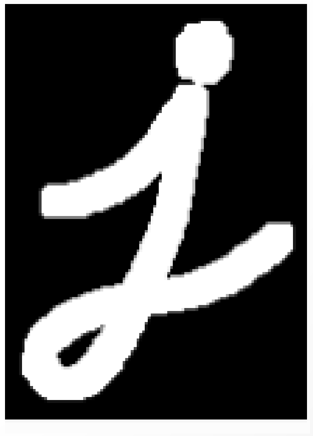
It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation.

Two basic morphological operators are Erosion and Dilation.





EROSION



DILATION

CONVEX HULL

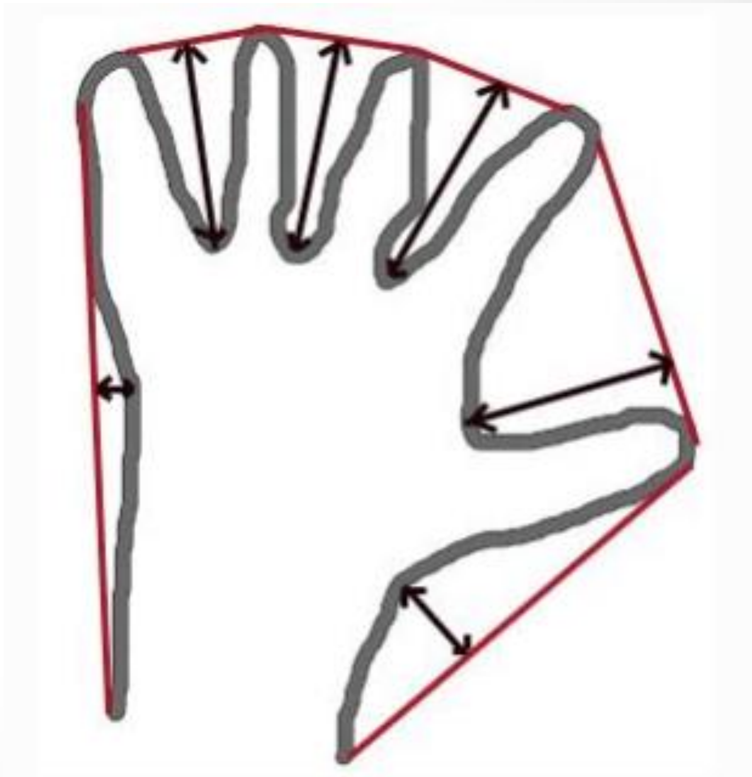
cv2.convexHull() function checks a curve for convexity defects and corrects it.

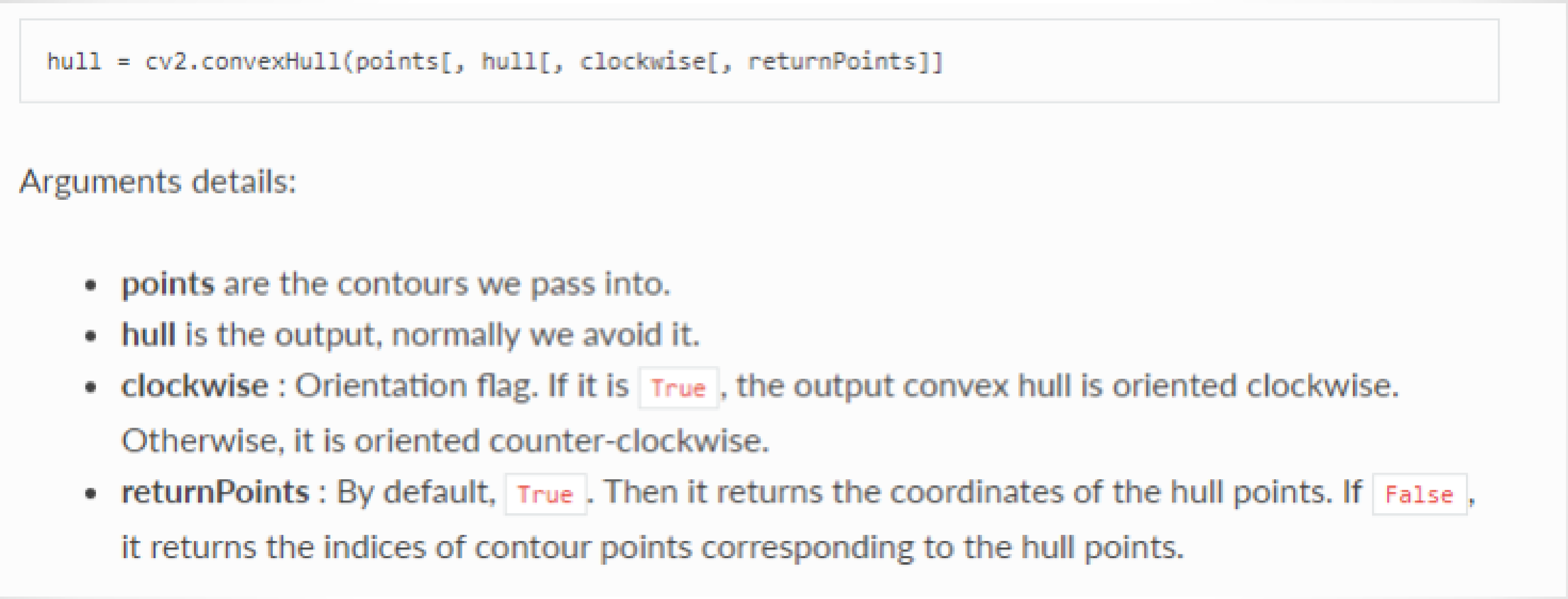
Generally speaking, convex curves are the curves which are always bulged out, or at-least flat. And if it is bulged inside, it is called convexity defects.

For example,.

Red line shows the convex hull of hand.

The double-sided arrow marks shows the convexity defects, which are the local maximum deviations of hull from contours.





```
hull = cv2.convexHull(points[, hull[, clockwise[, returnPoints]]
```

Arguments details:

- **points** are the contours we pass into.
- **hull** is the output, normally we avoid it.
- **clockwise** : Orientation flag. If it is `True`, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise.
- **returnPoints** : By default, `True`. Then it returns the coordinates of the hull points. If `False`, it returns the indices of contour points corresponding to the hull points.

Convexity Defects

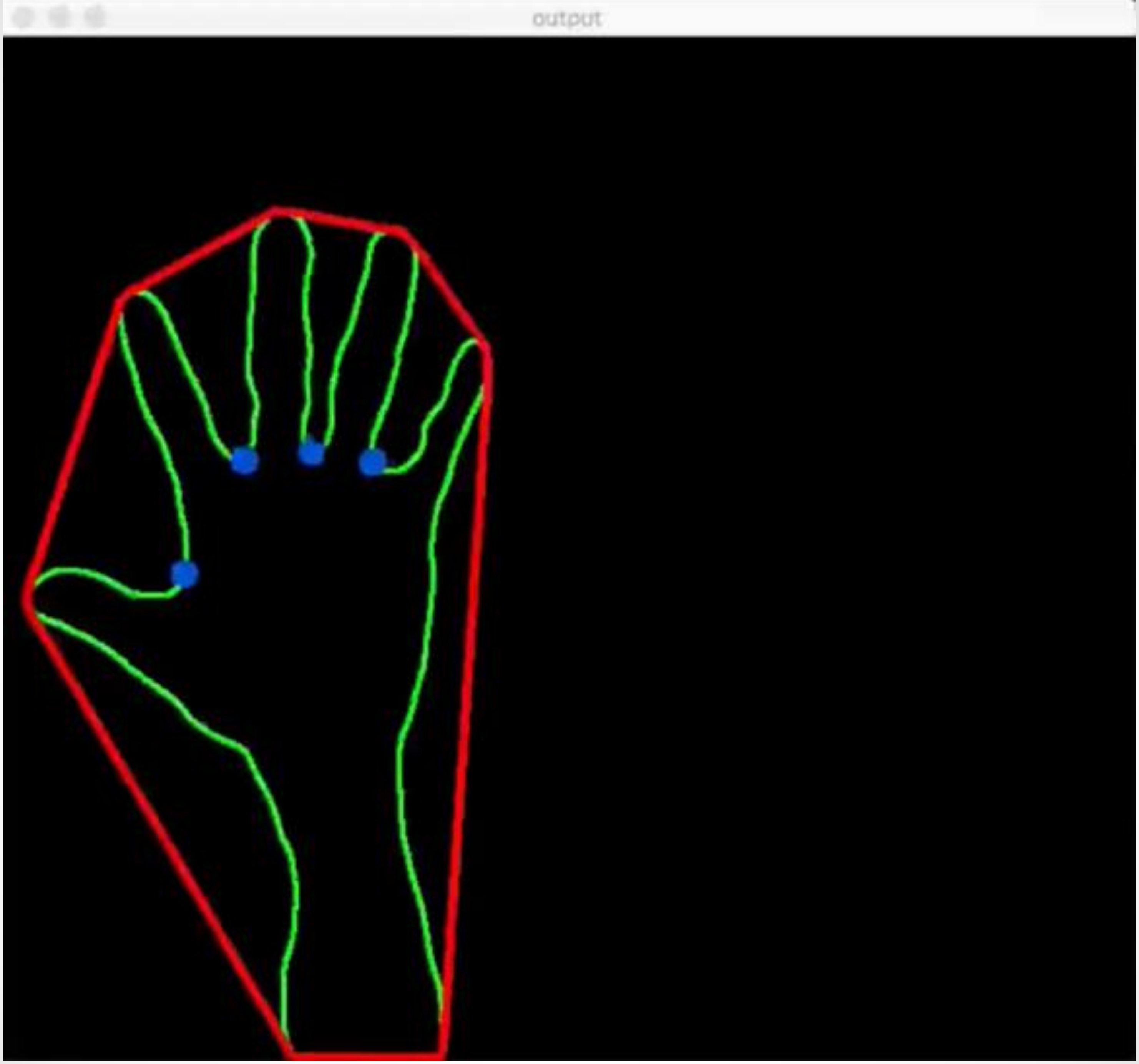
Any deviation of the object from this hull can be considered as convexity defect.

```
hull = cv2.convexHull(cnt,returnPoints = False)
defects = cv2.convexityDefects(cnt,hull)
```

It returns an array where each row contains these values - **[start point, end point, farthest point, approximate distance to farthest point]**.

We can visualize it using an image. We draw a line joining start point and end point, then draw a circle at the farthest point.

Remember first three values returned are indices of cnt. So we have to bring those values from cnt.


output