```
from google.colab import files
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 family.jpg to family.jpg
{'family.jpg': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00\x00\x01\x00\x01\x00\x
```

```
import cv2
import numpy as np

image = cv2.imread('family.jpg')
```

```
# BGR Values for the first 0,0 pixel
B, G, R = image[10, 60]
print (B, G, R)
print (image.shape)
```

```
177 223 254
(2357, 1963, 3)
```

```
gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
print (gray_img.shape)
print (gray_img[10, 50])
gray_img[0, 0]
```

```
(830, 1245)
22
21
```

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
img = cv2.imread("family.jpg")
b = img[:,:,0]
g = img[:,:,1]
r = img[:,:,2]
cv2_imshow(b)
cv2_imshow(b)
cv2_imshow(b)
# Let's re-make the original image,
merged = cv2.merge([b,g,r])
cv2_imshow(merged)
# Let's amplify the red color
merged = cv2.merge([b, g, r+100])
cv2_imshow(merged)
```

```
# We need to import matplotlib to create our histogram plots
from matplotlib import pyplot as plt

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

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

# We plot a histogram, ravel() flatens our image array
plt.hist(image.ravel(), 256, [0, 256]); plt.show()

# Viewing Separate Color Channels
color = ('b', 'g', 'r')
```

```
# We now separate the colors and plot each in the Histogram
for i, col in enumerate(color):
    histogram2 = cv2.calcHist([image], [i], None, [256], [0, 256])
    plt.plot(histogram2, color = col)
    plt.xlim([0,256])

plt.show()


# Create a black image
image = np.zeros((512,512,3), np.uint8)

# Can we make this in black and white?
image_bw = np.zeros((512,512), np.uint8)

cv2_imshow(image)
cv2_imshow(image_bw)




# Draw a diagonal colour lines of thickness of n pixels
image = np.zeros((512,512,3), np.uint8)
cv2.line(image, (0,0), (200,200), (0,500,0), 50)
cv2.line(image, (200,200), (350,350), (500,0,0), 50)
cv2.line(image, (350,350), (511,511), (0,0,500), 50)
cv2.line(image, (0,511), (200,200), (0,500,0), 50)
cv2.line(image, (200,200), (350,350), (500,0,0), 50)
cv2.line(image, (350,350), (511,0), (0,500,0), 50)


cv2_imshow(image)



# Draw a Rectangle in
image = np.zeros((512,512,3), np.uint8)

cv2.rectangle(image, (0,0), (300,300), (0,0,500), -1)
cv2.rectangle(image, (300,300), (511,511), (127,50,127), 10)

cv2_imshow(image)


image = np.zeros((512,512,3), np.uint8)

cv2.circle(image, (250, 250), 100, (0,0,500), -1)
cv2.circle(image, (250, 250), 150, (0,0,500), 10)
cv2.circle(image, (250, 250), 200, (0,0,500), 10)
cv2.circle(image, (250, 250), 250, (0,0,500), 10)

cv2_imshow(image)


image = np.zeros((512,512,3), np.uint8)

# Let's define four points
pts = np.array( [[10,50], [400,50], [90,200], [50,500], [52,359],[368,511]], np.int32)

# Let's now reshape our points in form  required by polylines
pts = pts.reshape((-1,1,2))

cv2.polylines(image, [pts], True, (0,0,255), 3)
cv2_imshow(image)


#cv2.putText(image, 'Text to Display', bottom left starting point, Font, Font Size, Color,
image = np.zeros((512,512,3), np.uint8)

cv2.putText(image, 'Hello World!', (75,290), cv2.FONT_HERSHEY_COMPLEX, 2, (100,170,0), 5)

cv2_imshow(image)
```

```
# Store height and width of the image
image = cv2.imread('input.jpg')
height, width = image.shape[:2]

quarter_height, quarter_width = height/4, width/4

#       | 1 0 Tx |
#  T  = | 0 1 Ty |

# T is our translation matrix
T = np.float32([[1, 0, quarter_width], [0, 1,quarter_height]])

# We use warpAffine to transform the image using the matrix, T
img_translation = cv2.warpAffine(image, T, (width, height))
cv2_imshow(img_translation)
print(T)
```

```
image = cv2.imread('input.jpg')
height, width = image.shape[:2]

# Divide by two to rototate the image around its centre
rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), 90, .5)

rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))

cv2_imshow(rotated_image)
rotated_image = cv2.transpose(img)

cv2_imshow(rotated_image)
flipped = cv2.flip(image, 1)
cv2_imshow(flipped)
```

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
# load our input image
image = cv2.imread('input.jpg')

# Let's make our image 3/4 of it's original size
image_scaled = cv2.resize(image, None, fx=0.25, fy=0.25)
cv2_imshow(image_scaled)


# Let's double the size of our image
img_scaled = cv2.resize(image, None, fx=2, fy=2, interpolation = cv2.INTER_CUBIC)
cv2_imshow(img_scaled)


# Let's skew the re-sizing by setting exact dimensions
img_scaled = cv2.resize(image, (900, 400), interpolation = cv2.INTER_AREA)
cv2_imshow(img_scaled)


#image pyramiding
smaller = cv2.pyrDown(image)
larger = cv2.pyrUp(smaller)

cv2_imshow(image )

cv2_imshow(smaller )
cv2_imshow(larger )


#Croping the image
height, width = image.shape[:2]
```

```
# Let's get the starting pixel coordiantes (top  left of cropping rectangle)
start_row, start_col = int(height * .25), int(width * .25)

# Let's get the ending pixel coordinates (bottom right)
end_row, end_col = int(height * .75), int(width * .75)

# Simply use indexing to crop out the rectangle we desire
cropped = image[start_row:end_row , start_col:end_col]

cv2_imshow(image)
cv2_imshow(cropped)
```

```
#Arithmetic Operations
#These are simple operations that allow us to directly add or subract to the color intensi

#Calculates the per-element operation of two arrays. The overall effect is increasing or d
M = np.ones(image.shape, dtype = "uint8") * 175

# We use this to add this matrix M, to our image
# Notice the increase in brightness
added = cv2.add(image, M)
cv2_imshow(added)

# Likewise we can also subtract
# Notice the decrease in brightness
subtracted = cv2.subtract(image, M)
cv2_imshow(subtracted)
```

```
#Bitwise Operations and Masking
# Making a sqare
square = np.zeros((300, 300), np.uint8)
cv2.rectangle(square, (50, 50), (250, 250), 255, -2)
cv2_imshow(square)

# Making a ellipse
ellipse = np.zeros((300, 300), np.uint8)
cv2.ellipse(ellipse, (150, 150), (150, 150), 30, 0, 180, 255, -1)
cv2_imshow(ellipse)

And = cv2.bitwise_and(square, ellipse)
cv2_imshow(And)

Or = cv2.bitwise_or(square, ellipse)
cv2_imshow(Or)

Xor = cv2.bitwise_xor(square, ellipse)
cv2_imshow(Xor)

Not = cv2.bitwise_not(square)
cv2_imshow(Not)
```

```
# Creating our 3 x 3 kernel
kernel_3x3 = np.ones((3, 3), np.float32) / 9

# We use the cv2.fitler2D to conovlve the kernal with an image
blurred = cv2.filter2D(image, -1, kernel_3x3)
cv2_imshow(blurred)

# Creating our 7 x 7 kernel
kernel_7x7 = np.ones((7, 7), np.float32) / 49

blurred2 = cv2.filter2D(image, -1, kernel_7x7)
cv2_imshow(blurred2)

# Averaging done by convolving the image with a normalized box filter.
# This takes the pixels under the box and replaces the central element
# Box size needs to odd and positive
blur = cv2.blur(image, (3,3))
cv2_imshow(blur)

# Instead of box filter, gaussian kernel
```

```
Gaussian = cv2.GaussianBlur(image, (7,7), 0)
cv2_imshow(Gaussian)

# Takes median of all the pixels under kernel area and central
# element is replaced with this median value
median = cv2.medianBlur(image, 5)
cv2_imshow(median)

# Bilateral is very effective in noise removal while keeping edges sharp
bilateral = cv2.bilateralFilter(image, 9, 75, 75)
cv2_imshow(bilateral)

#Image De-noising - Non-Local Means Denoising
# Parameters, after None are - the filter strength 'h' (5-10 is a good range)
# Next is hForColorComponents, set as same value as h again
#
dst = cv2.fastNlMeansDenoisingColored(image, None, 6, 6, 7, 21)

cv2_imshow(dst)
```

```
#Sharpening
# Create our shapening kernel, we don't normalize since the
# the values in the matrix sum to 1
kernel_sharpening = np.array([[-1,-1,-1],
                              [-1,9,-1],
                              [-1,-1,-1]])

# applying different kernels to the input image
sharpened = cv2.filter2D(image, -1, kernel_sharpening)

cv2_imshow(sharpened)
```