Drawing on Images:

```
cv2.line(img,(0,0),(200,300),(255,0,0),5) #Draw a blue line with thickness of 5
cv2.re cta ngl e(i mg, (50 ,50 ),( 200 ,30 0), (0, 255 ,0),3) #Draw a green rectangle with thickness of 3
cv2.ci rcl e(i mg, (20 0,200), 50, (0,0,255), -1) #Draw a filled red circle
cv2.pu tTe xt( img ,'O pen CV' ,(1 0,500), cv2.FO NT_ HER SHE Y_S IMPLEX, 4,(255 ,25 5,2 55) ,2, cv2.LI -
NE_AA) #Add text
```

Feature Detection:

```
# identify distinct regions of an image that contain important information. Used for object
# recogn ition, image regist ration, and 3D recons tru ction. Common methods: Harris corner
# detection, Shi-Tomasi corner detection, FAST, SURF, SIFT.
img = cv2.im rea d(' ima ge.j pg', 0) #Read image in grayscale
fast = cv2.Fa stF eat ure Det ect or_ cre ate() ##FAST feature detection
keypoi nts _fast = fast.d ete ct(img)
orb = cv2.OR B_c reate() ##ORB feature detection
keypoi nts _orb, descri ptors = orb.de tec tAn dCo mpu te(img, None)
sift = cv2.xf eat ure s2d.SI FT_ cre ate() ##SIFT feature detection
keypoi nts _sift, descri ptors = sift.d ete ctA ndC omp ute (img, None)
surf = cv2.xf eat ure s2d.SU RF_ cre ate() ##SURF feature detection
keypoi nts _surf, descri ptors = surf.d ete ctA ndC omp ute (img, None)
cv2.dr awK eyp oin ts(img, keypoi nts _fast, img, color= (255, 0, 0))
cv2.dr awK eyp oin ts(img, keypoi nts _orb, img, color=(0, 255, 0))
cv2.dr awK eyp oin ts(img, keypoi nts _sift, img, color=(0, 0, 255))
cv2.dr awK eyp oin ts(img, keypoi nts _surf, img, color= (255, 255, 0))
```

Perspective Transformation:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
pts1 = np.flo at3 2([ [56 ,65 ],[ 368 ,52 ],[ 28, 387 ],[ 389 ,390]]) #Four points on original image
pts2 = np.flo at3 2([ [0, 0], [30 0,0 ],[ 0,3 00] ,[3 00, 300]]) #Four points on destin ation image
M = cv2.ge tPe rsp ect ive Tra nsf orm (pt s1, pts2) #Persp ective transf orm ation matrix
dst = cv2.wa rpP ers pec tiv e(i mg, M,( 300 ,300)) #Apply perspe ctive transf orm ation
cv2.im sho w(' ima ge' ,dst)
cv2.wa itK ey(0)
cv2.de str oyA llW ind ows()
```

Feature Description:

```
# extract numerical representation of detected feature in image. Important for object
# recogn ition and image matching. Common methods: BRIEF, ORB, SIFT, SURF.
img = cv2.im rea d(' ima ge.j pg', 0) # Read image in grayscale
fast = cv2.Fa stF eat ure Det ect or_ cre ate() #FAST feature detection
brief = cv2.xf eat ure s2d.Br ief Des cri pto rEx tra cto r_c reate() #BRIEF descriptor
keypoi nts _fast = fast.d ete ct(img)
keypoi nts _fast, descri pto rs_fast = brief.c om put e(img, keypoi nts _fast)
orb = cv2.OR B_c reate() #ORB feature detection and descriptor
keypoi nts _orb, descri pto rs_orb = orb.de tec tAn dCo mpu te(img, None)
```

Feature Description: (cont)

> sift = cv2.xfeatures2d.SIFT_create() #SIFT feature detection and descriptor
keypoints_sift, descriptors_sift = sift.detectAndCompute(img, None)
surf = cv2.xfeatures2d.SURF_create() #SURF feature detection and descriptor
keypoints_surf, descriptors_surf = surf.detectAndCompute(img, None)

Haar Cascade Classifiers (Object Detection):

```
# object detection using image patterns. Trains on features of positive and negative samples.
# Identifies object in new images by matching pattern. Can detect faces, eyes, and other objects.
face_c ascade = cv2.Ca sca deC las sif ier ('h aar cas cad e_f ron tal fac e_d efa ult.xml') #Load face
cascade classifier
img = cv2.im rea d(' ima ge.j pg', cv2.IM REA D_C OLOR)
gray = cv2.cv tCo lor (img, cv2.CO LOR _BG R2GRAY) #Convert to grayscale
faces = face_c asc ade.de tec tMu lti Sca le( gray, 1.3, 5) #Detect faces
for (x,y,w,h) in faces:
        cv2.re cta ngl e(i mg, (x, y), (x+ w,y +h) ,(2 55, 0,0),2) #Draw rectangle around face
```

Contour Detection:

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Convert to grayscale
ret, thresh = cv2.th res hol d(gray, 127, 255, cv2.TH RES H_B INARY) #Threshold
contours, hierarchy = cv2.fi ndC ont our s(t hresh, cv2.RE TR_ TREE, cv2.CH AIN _AP PRO X_S IMPLE) #Find
contours
cv2.dr awC ont our s(img, contours, -1, (0,255,0), 3) #Draw contours
```

Loading and Saving Images:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR) #Load image
cv2.im sho w(' ima ge' ,img) #Show image
cv2.im wri te( 'ou tpu t.j pg' ,img) #Save image
cv2.wa itK ey(0) #Wait for a keyboard event
cv2.de str oyA llW ind ows() #Close all windows
```

Image Properties:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
print( 'Image Shape:', img.shape) #Prints (height, width, channels)
print( 'Image Size:', img.size) #Prints the total number of pixels
print( 'Image Dataty pe:', img.dtype) #Prints the datatype of the pixels
```

Accessing a Camera/ Video Capture:

```
cap = cv2.VideoCapture(0) #Open default camera
while( True):
       ret, frame = cap.read() #Read frame from camera
        cv2.im sho w(' fra me' ,frame) #Show frame
       if cv2.wa itK ey(1) & 0xFF == ord('q'): #Quit when 'q' is pressed
                break
```

Accessing a Video File:

```
cap = cv2.VideoCapture('video.mp4') #Open video file
while( cap.is Ope ned()):
       ret, frame = cap.read() #Read frame from video
        cv2.im sho w(' fra me' ,frame) #Show frame
       if cv2.wa itK ey(25) & 0xFF == ord('q'): #Quit when 'q' is pressed
                break
```

Video Writer:

```
cap = cv2.VideoCapture(0)
fourcc = cv2.Vi deo Wri ter _fo urc c(* 'XVID') #Codec
out = cv2.Vi deo Wri ter ('o utp ut.a vi ',f ourcc, 20.0, (640,480)) #Create video writer
while( cap.is Ope ned()):
       ret, frame = cap.read() #Capture frame- by- frame
       if ret==True:
                out.wr ite (frame) #Write frame to output video
                cv2.im sho w(' fra me' ,frame) #Display frame
               if cv2.wa itK ey(1) & 0xFF == ord('q'): #Exit loop on 'q' key press
                        break
        else:
                break
cap.re lease() #Release camera
out.re lease() #Release video writer
```

Video Writer: (cont)

> cv2.destroyAllWindows()

Colorspace Conversion:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
gray_img = cv2.cv tCo lor (img, cv2.CO LOR _BG R2GRAY) #Convert to grayscale
hsv_img = cv2.cv tCo lor (img, cv2.CO LOR _BG R2HSV) #Convert to HSV colorspace
```

Blurring:

```
# reduces image noise and sharpness by averaging pixel values in a neighborhood.
gaussi an_blur = cv2.Ga uss ian Blu r(img, (5, 5), 0)
median _blur = cv2.me dia nBl ur(img, 5)
bilate ral _filter = cv2.bi lat era lFi lte r(img, 9, 75, 75)
```

Image Filtering:

```
# modifies pixel values to improve image quality. Common methods include 2D convolution,
#blurring, and edge detection.
img = cv2.im rea d(' ima ge.j pg')
kernel = np.one s(( 5,5 ),n p.f loa t32)/25
convol ution = cv2.fi lte r2D (im g,- 1,k ernel) #2D Convol ution
blur = cv2.bl ur( img ,(5,5)) # Image Blurring (avera ging)
gaussi an_blur = cv2.Ga uss ian Blu r(img, (5,5),0) # Gaussian Blurring
median _blur = cv2.me dia nBl ur(img, 5) # Median Blurring
bilate ral _filter = cv2.bi lat era lFi lte r(img, 9, 75, 75) #Bilateral Filtering
```

Edge Detection:

```
img = cv2.imread('image.jpg', 0) # Read image in grayscale
# Canny edge detection
edges_ canny = cv2.Ca nny (img, 100, 200)
# Sobel operator
edges_ sobel = cv2.So bel (img, cv2.CV _64F, 1, 0) + cv2.So bel (img, cv2.CV _64F, 0, 1)
# Laplacian of Gaussian (LoG)
edges_log = cv2.La pla cia n(c v2.G au ssi anB lur (img, (3, 3), 0), cv2.CV _64F)
cv2.im sho w(' Ori ginal', img)
cv2.im sho w(' Canny Edges', edges_ canny)
cv2.im sho w(' Sobel Edges', edges_ sobel)
cv2.im sho w('LoG Edges', edges_log)
cv2.wa itK ey(0)
cv2.de str oyA llW ind ows()
```

Morphological Operations:

```
kernel = np.ones((5,5),np.uint8) #5x5 kernel
erosion = cv2.er ode (im g,k ern el, ite rations = 1) #Erosion
dilation = cv2.di lat e(i mg, ker nel ,it era tions = 1) #Dilation
opening = cv2.mo rph olo gyE x(img, cv2.MO RPH _OPEN, kernel) #Opening
closing = cv2.mo rph olo gyE x(img, cv2.MO RPH _CLOSE, kernel) #Closing
```

Histogram Equalization:

```
# improves contrast by redistributing pixel values in an image.
# It's useful for image prepro cessing and analysis
eq_img = cv2.eq ual ize His t(img) #Apply histogram equali zation
```

Hough Transform:

```
# detect simple shapes such as lines, circles, and ellipses in an image
img = cv2.im rea d(' ima ge.j pg', cv2.IM REA D_G RAY SCALE)
edges = cv2.Ca nny (im g,5 0,1 50, ape rtu reSize = 3) #Canny edge detection
lines = cv2.Ho ugh Lin es( edg es, 1,n p.p i/1 80,200) #Hough transform
for rho,theta in lines[0]:
       a = np.cos (theta)
       b = np.sin (theta)
       x0 = a*rho
       y0 = b*rho
       x1 = int(x0 + 1000*(-b))
       y1 = int(y0 + 1000*(a))
       x2 = int(x0 - 1000*(-b))
       y2 = int
```

Thresholding:

```
# Thresholding separates object from background by setting pixel values above or below a
# threshold. It creates a binary image. Helps with object detection, segmen tation, and feature
# extrac tion. Methods: global, adaptive, and Otsu's.
img = cv2.im rea d(' ima ge.j pg', cv2.IM REA D_G RAY SCALE)
ret,th resh1 = cv2.th res hol d(i mg, 127 ,25 5,c v2.T HR ESH _BI NARY) #Binary thresh olding
ret,th resh2 = cv2.th res hol d(i mg, 127 ,25 5,c v2.T HR ESH _BI NAR Y_INV) #Inverse binary thresh olding
ret,th resh3 = cv2.th res hol d(i mg, 127 ,25 5,c v2.T HR ESH _TRUNC) #Truncated thresh olding
ret,th resh4 = cv2.th res hol d(i mg, 127 ,25 5,c v2.T HR ESH _TO ZERO) #Threshold to zero
ret,th resh5 = cv2.th res hol d(i mg, 127 ,25 5,c v2.T HR ESH _TO ZER O_INV) #Inverse threshold to zero
adapti ve_ thresh = cv2.ad apt ive Thr esh old (im g,2 55, cv2.AD APT IVE _TH RES H_G AUS SIA N_C ,cv 2.T -
HRE SH_ BIN ARY ,11,2) #Adaptive thresh olding
```

Image Pyramids:

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
lower_reso = cv2.py rDo wn(img) #Reduce resolution
higher _reso = cv2.py rUp (img) #Increase resolution
```