```python
# Prepared by Izam Mohammed

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


def basic():
    global img
    img = cv2.imread("data/girl.jpg")
    cv2.imshow("original image", img)
    cv2.imwrite("artifacts/girl.jpg", img)


def with_plt():
    rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.figure()
    plt.imshow(rgb_img)
    plt.show()


def channels():
    b, g, r = cv2.split(img)
    cv2.imshow("Blue", cv2.merge([b, np.zeros_like(r), np.zeros_like(r)]))
    cv2.imshow("green", cv2.merge([np.zeros_like(r), g, np.zeros_like(r)]))
    cv2.imshow("Red", cv2.merge([np.zeros_like(r), np.zeros_like(r), r]))


def resize():
    resized_img = cv2.resize(img, (200, 200))
    cv2.imshow("resized img", resized_img)


def crop():
    cropped_img = img[100:300, 100:450]
    cv2.imshow("croped img", cropped_img)


def video_read():
    global video
    video = cv2.VideoCapture("data/sample.mp4")
    while True:
        success, frame = video.read()
        cv2.imshow("video", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
    video.release()
    cv2.destroyAllWindows()


def webcam_video():
    global video
    video = cv2.VideoCapture(0)
    while True:
        success, frame = video.read()
        cv2.imshow("video", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
    video.release()
    cv2.destroyAllWindows()


def shapes():
    global blank
    blank = np.zeros((500, 500, 3), dtype="uint8")
    cv2.imshow("blank image", blank)

    rectangle = cv2.rectangle(blank, (10, 10), (100, 150), (0, 255, 0), 2)
    line = cv2.line(blank, (150, 150), (150, 350), (255, 255, 0), 1)
    circle = cv2.circle(blank, (350, 200), 100, (0, 0, 255), 2)
    cv2.imshow("shapes", blank)


def color_spaces():
    global gray, hsv, lap
```

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    lap = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    cv2.imshow("gray", gray)
    cv2.imshow("hsv", hsv)
    cv2.imshow("lap", lap)


def threshold():
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, binary_image = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
    cv2.imshow("simple binary thresholding", binary_image)

    _, binary_image_inv = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)
    cv2.imshow("Inverse binary thresholding", binary_image_inv)

    adp_img = cv2.adaptiveThreshold(
        gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2
    )
    cv2.imshow("adaptive thresholding", adp_img)


def track_bars():
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    def update(value):
        _, processed = cv2.threshold(gray, value, 255, cv2.THRESH_BINARY)
        cv2.imshow("track bar", processed)

    cv2.namedWindow("track bar")
    cv2.createTrackbar("Thresh val", "track bar", 127, 255, update)


def mouse_events():
    def draw_fn(event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            cv2.circle(img, (x, y), 20, (255, 255, 255), -1)

    cv2.namedWindow("events")
    cv2.setMouseCallback("events", draw_fn)
    while True:
        cv2.imshow("events", img)
        key = cv2.waitKey(1)
        if key == 27:  # esc key
            break


def blur():
    average = cv2.blur(img, (5, 5))
    median = cv2.medianBlur(img, 7)
    gaussian = cv2.GaussianBlur(img, (11, 11), 0)

    cv2.imshow("average blurring", average)
    cv2.imshow("median blurring", median)
    cv2.imshow("Gaussian blurring", gaussian)


def edge_detection():
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
    soble_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
    gradient_magnitude = np.sqrt(sobel_x**2 + soble_y**2)

    blurred = cv2.GaussianBlur(gray, (3, 3), 0)
    laplacian = cv2.Laplacian(blurred, cv2.CV_64F)
    laplacian = np.uint8(np.absolute(laplacian))

    canny = cv2.Canny(gray, 50, 150)

    cv2.imshow("sobel (gradient magnitude) edge detectin", gradient_magnitude)
    cv2.imshow("laplacian edge detection", laplacian)
    cv2.imshow("canny edge detection", canny)
```

```python
def morphological_transformations():
    binary_image = cv2.imread("data/num.jpg", cv2.IMREAD_GRAYSCALE)
    binary_image = cv2.resize(binary_image, (300, 500))
    kernel = np.ones((10, 10), np.uint8)

    dialated_img = cv2.dilate(binary_image, kernel, iterations=1)
    eroded_img = cv2.erode(binary_image, kernel, iterations=1)
    opening_img = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)
    closing_img = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE, kernel)

    cv2.imshow("original image", binary_image)
    cv2.imshow("dialated image", dialated_img)
    cv2.imshow("eroded image", eroded_img)
    cv2.imshow("opening image", opening_img)
    cv2.imshow("closing image", closing_img)


def image_pyramid():
    image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(8, 8))
    plt.subplot(3, 3, 1), plt.imshow(image_rgb), plt.title("original image")

    for i in range(2, 10):
        resized_image = cv2.resize(img, (img.shape[1] // i, img.shape[0] // i))
        resized_image_rgb = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)
        plt.subplot(3, 3, i), plt.imshow(resized_image_rgb), plt.title(f"Scale {1/i}")

    plt.tight_layout()
    plt.show()


def bitwise_operators():
    blank = np.zeros((400, 400), dtype="uint8")
    rectangle = cv2.rectangle(blank.copy(), (30, 30), (370, 370), 255, -1)
    circle = cv2.circle(blank.copy(), (200, 200), 200, 255, -1)

    bitwise_and = cv2.bitwise_and(rectangle, circle)
    bitwise_or = cv2.bitwise_or(rectangle, circle)
    bitwise_xor = cv2.bitwise_xor(rectangle, circle)
    bitwise_not = cv2.bitwise_not(circle)

    cv2.imshow("blank image", blank)
    cv2.imshow("rectangle image", rectangle)
    cv2.imshow("circle image", circle)

    cv2.imshow("bitwise_and image", bitwise_and)
    cv2.imshow("bitwise_or image", bitwise_or)
    cv2.imshow("bitwise xor image", bitwise_xor)
    cv2.imshow("bitwise not image", bitwise_not)


def masking():
    blank = np.zeros(img.shape[:2], dtype="uint8")
    mask = cv2.circle(
        blank.copy(), (img.shape[1] // 2, img.shape[0] // 2), 100, 255, -1
    )
    masked_img = cv2.bitwise_and(img, img, mask=mask)

    cv2.imshow("mask image", mask)
    cv2.imshow("masked image", masked_img)


def histogram():
    plt.figure()
    plt.title("colored histogram")
    plt.xlabel("bins")
    plt.ylabel("# of pixels")
    colors = ("b", "g", "r")
    for i, col in enumerate(colors):
        hist = cv2.calcHist([img], [i], None, [256], [0, 256])
        plt.plot(hist, color=col)
        plt.xlim([0, 255])

    plt.show(), plt.show()
```

```
def haarcasecade():
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # detect face
    haar_casecade_face = cv2.CascadeClassifier("xml_files/haar_face.xml")
    faces_rect = haar_casecade_face.detectMultiScale(
        gray, scaleFactor=1.1, minNeighbors=6
    )
    for x, y, w, h in faces_rect:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # detect eyes
    haar_casecade_eyes = cv2.CascadeClassifier("xml_files/haar_eye.xml")
    eyes_rect = haar_casecade_eyes.detectMultiScale(
        gray, scaleFactor=1.1, minNeighbors=6
    )
    for x, y, w, h in eyes_rect:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)

    cv2.imshow("image with faces", img)


basic()
haarcasecade()

cv2.waitKey(0)
```