In [2]:
```
pip install opencv-contrib-python # to install OpenCV
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Invalid requirement: '#': Expected package name at the start of depen
dency specifier
    #
    ^
```

In [3]:
```
pip install cvlib # to work with yolov4 for object detection
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Invalid requirement: '#': Expected package name at the start of depen
dency specifier
    #
    ^
```

In [4]:
```
pip install --upgrade setuptools pip # to include version with GUI
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Invalid requirement: '#': Expected package name at the start of depen
dency specifier
    #
    ^
```

In [5]:
```
pip install --upgrade cvlib opencv-contrib-python opencv-python-headless # to
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Invalid requirement: '#': Expected package name at the start of depen
dency specifier
    #
    ^
```

In [6]:
```
pip cache purge # to remove files that conflict with allowing GUI
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Too many arguments
```

In [7]:
```
pip uninstall -y opencv-python opencv-python-headless opencv-contrib-python o
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Invalid requirement: '#': Expected package name at the start of depen
dency specifier
    #
    ^
```

In [8]:
```
pip install --upgrade opencv-python==4.5.3.56 # to include version with GUI
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Invalid requirement: '#': Expected package name at the start of depen
dency specifier
    #
    ^
```

In [9]:
```
pip install --upgrade opencv-contrib-python==4.5.3.56 # to include version wi
```

```
Note: you may need to restart the kernel to use updated packages.

ERROR: Invalid requirement: '#': Expected package name at the start of depen
dency specifier
    #
    ^
```

In [10]:
```
import cv2 # run to test if OpenCV is working
print(cv2.__version__)
print(cv2.INTER_AREA)
cv2.namedWindow("test window")
cv2.destroyAllWindows()
```

```
4.10.0
3
```

In [11]:
```
pip install wget # for yolov4 file access
```

In [12]:
```
# run only if the yolov4 library is not in the same directory
import wget

# URL for the YOLOv4 configuration file
url = 'https://github.com/AlexeyAB/darknet/blob/master/cfg/yolov4.cfg?raw=tru

# Download the file
wget.download(url, 'yolov4.cfg')
```

In [13]:

```
import cv2
import numpy as np

# Load YOLOv4 configuration and weights
net = cv2.dnn.readNet('yolov4.weights', 'yolov4.cfg')

# Load class labels
with open('coco.names', 'r') as f:
    classes = f.read().strip().split('\n')

# Initialize the webcam
stream = cv2.VideoCapture(0)
if not stream.isOpened():
    print("Error: Could not access the webcam.")
    exit()

print("Webcam accessed successfully.")

# Infinitely loop through each captured frame from the webcam (unless we quit
while True:
    ret, frame = stream.read()

    # Ensure frame is usable in the program
    if not ret:
        print("Failed to capture frame.")
        break

    if frame is None:
        print("Error: Frame is None")
        continue

    height, width = frame.shape[:2]

    # Create a blob from the frame & populate neural network
    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, cro
    net.setInput(blob)

    # Get the output layer names
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers(

    # Conduct a forward pass on neural network
    detections = net.forward(output_layers)

    # Loop through object detections to extract key information needed to dra
    boxes, confidences, class_ids = [], [], []

    for output in detections:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if (confidence > 0.25) and (class_id):
                box = detection[0:4] * np.array([width, height, width, height
                (centerX, centerY, w, h) = box.astype("int")

                x = int(centerX - (w / 2))
```

```
                y = int(centerY - (h / 2))

                boxes.append([x, y, int(w), int(h)])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    # Apply non-maxima suppression (NMS) i.e. filter to remove overlapping & 
    indices = cv2.dnn.NMSBoxes(boxes, confidences, 0.25, 0.4)

    # Draw detections as rectangles on the screen using box, confidence, and 
    if (len(indices) > 0):
        for i in indices.flatten():
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])
            class_id = class_ids[i]
            class_name = classes[class_id]

            if class_name in ['cell phone', 'laptop', 'remote', 'toothbrush']
                color = [int(c) for c in np.random.randint(0, 255, size=(3,))
                cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
                text = f"{class_name}: {confidences[i]:.2f}"
                cv2.putText(frame, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX

    cv2.imshow("ADHD Object Finder", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

stream.release()
cv2.destroyAllWindows()
```

```
Webcam accessed successfully.
```