PART 1 — REAL-TIME CARD DETECTION CODE (YOLO + OpenCV)

IMPORTS

- **ultralytics.YOLO** → loads your trained YOLO model (`playingCards.pt`), and runs detections.
- **cv2** → OpenCV, handles video capture and image processing.
- **cvzone** → a wrapper library that makes bounding boxes, text, rectangles look beautiful and easy.
- **math** → used for rounding confidence values.
- **HelperFunction** → your other Python file where `findPokerHand()` lives.

Video Capturing

- `VideoCapture(0)` → opens your laptop webcam.
- `.set(3, 1280)` → sets the frame width.
- `.set(4, 720)` → sets the frame height.

This ensures a clean HD feed to YOLO.

Load YOLO Model

```
model = YOLO("playingCards.pt")
```

This loads your custom-trained playing card detector.
It knows how to detect each card class (10C, AH, QS, …).

Class Names

You manually defined names for the 52 cards:

```
classNames = ['10C', '10D', '10H', '10S', ... 'QS']
```

Why?
YOLO gives you a class number (0–51).
You convert number → card name using this list.

Main Loop

```
while True:
    success, img = cap.read()
```

- Reads the next video frame.
- `success` tells if camera is working.
- `img` contains the frame.

Run YOLO

```
results = model(img, stream=True)
```

- Sends the frame into the model.
- `stream=True` lets you iterate detections one-by-one (efficient).

Container to store your detected hand

```
hand = []
```

We'll fill this with detected classes like `['KH', '10D', ...]`.

Loop Through Detections

```
for r in results:
    boxes = r.boxes
```

- YOLO returns “results”.
- `r.boxes` gives you all bounding boxes in the frame.

Extract position and draw rectangle

```
for box in boxes:
    x1, y1, x2, y2 = box.xyxy[0]
    x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
    w, h = x2 - x1, y2 - y1

    cvzone.cornerRect(img, (x1, y1, w, h))
```

- `xyxy` = top-left + bottom-right corners of box.
- Converted to int for OpenCV.
- `cornerRect()` draws a stylish bounding box.

Get confidence

```
conf = math.ceil(box.conf[0] * 100) / 100
```

YOLO returns a float (0.84657…).
You round it to 2 decimal places.

Get class index

```
cls = int(box.cls[0])
```

YOLO says something like:
"class = 13" → you convert it to an integer.

Then convert class → card label:

```
classNames[cls]
```

Put the text on the image

```
cvzone.putTextRect(img, f'{classNames[cls]} {conf}', ...)
```

Displays:

```
AH 0.86
```

Add to hand only if confidence > 0.5

```
if conf > 0.5:
    hand.append(classNames[cls])
```

This prevents false positives.

Remove duplicates

```
hand = list(set(hand))
```

If the card appears multiple times in frames, you only want it once.

When 5 cards are detected…

```
if len(hand) == 5:
    results = HelperFunction.findPokerHand(hand)
    cvzone.putTextRect(img, f'Your Hand: {results}', (300, 75), scale=3,
thickness=5)
```

- You compute poker ranking.
- Show beautiful label like:

```
Your Hand: Straight Flush
```

Show the frame

```
cv2.imshow("Image", img)
```

Quit on Q

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    cap.release()
    cv2.destroyAllWindows()
    break
```

PART 2 — Poker Hand Evaluation (HelperFunction.py)

Function signature

```
def findPokerHand(hand):
```

Receives a list like:

```
["KH", "AH", "QH", "JH", "10H"]
```

Initialize empty lists

```
ranks = []
```

```
suits = []
possibleRanks = []
```

- `ranks` → numeric values (A=14, K=13, …)
- `suits` → H/D/S/C
- `possibleRanks` → store possible hand types (1–10)

Extract rank and suit

```
for card in hand:
    if len(card) == 2:
        rank = card[0]
        suit = card[1]
    else:
        rank = card[0:2]
        suit = card[2]
```

Reason:

- `"9H"` → length 2
- `"10H"` → length 3

So some ranks are 1 char, some 2 chars.

Convert ranks to numbers

```
if rank == "A": rank = 14
elif rank == "K": rank = 13
elif rank == "Q": rank = 12
elif rank == "J": rank = 11
```

Numbers make straight-checking much easier.

Sort ranks

```
sortedRanks = sorted(ranks)
```

Example:

```
[10, 11, 12, 13, 14]
```

Checking Poker Hands

Flush Check

```
if suits.count(suits[0]) == 5:
```

All suits same → FLUSH TYPE.

Inside that:

Royal Flush

```
if 14,13,12,11,10 in sortedRanks:
    possibleRanks.append(1)
```

Straight Flush

```
elif all(sortedRanks[i] == sortedRanks[i - 1] + 1 ...):
    possibleRanks.append(2)
```

Normal Flush

```
else:
    possibleRanks.append(5)
```

Straight (no flush)

```
if all(sortedRanks[i] == sortedRanks[i - 1] + 1 ...):
    possibleRanks.append(6)
```

Four of a kind / Full House

```
handUniqueVals = list(set(sortedRanks))
```

If you have ONLY 2 unique ranks:

- 4 + 1 → Four of a kind
- 3 + 2 → Full house

Three of a kind / Two Pair

If you have 3 unique ranks:

- Some rank appears 3 times → Three of a kind
- Two ranks appear twice → Two Pair

Pair

If 4 unique ranks → exactly one pair.

High Card

No matches → high card.

Pick the BEST rank

```
pokerHandRanks = {1:"Royal Flush", ... 10:"High Card"}
output = pokerHandRanks[min(possibleRanks)]
```

Smaller number = better hand.

Finally

Print and return the result.