Session 10

Explanation in detailed stepwise

Here is a detailed, step-by-step explanation of the document scanner code from the attached file, aimed at helping a student understand how it works and why each step is necessary:

Step 1: Installing Required Libraries

You first need to install all the necessary Python libraries — `OpenCV-Python`, `imutils`, `scikit-image`, and `NumPy`.

- **Purpose:** These libraries are essential for image processing, image manipulation, numerical computation, and thresholding, which are the core tasks of the document scanner.

Step 2: Loading and Resizing the Image

```
original_img = cv2.imread('images/sample.jpg')
copy = original_img.copy()
ratio = original_img.shape[^0] / 500.0
img_resize = imutils.resize(original_img, height=500)
cv2.imshow('Resized image', img_resize)
cv2.waitKey(0)
```

- **Purpose:** Read the input image and create a copy for later use.
- **Resize Goal:** The image is resized to a fixed height (500 pixels) to standardize processing speed and aspect ratio. Keeping a ratio allows proper scaling later during perspective transformation.

Step 3: Converting to Grayscale

```
gray_image = cv2.cvtColor(img_resize, cv2.COLOR_BGR2GRAY)
cv2.imshow('Grayed Image', gray_image)
cv2.waitKey(0)
```

- **Purpose:** Most image-processing tasks (like edge detection) are simpler and more effective in grayscale, where only intensity values are considered, not color.

Step 4: Edge Detection

```
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
edged_img = cv2.Canny(blurred_image, 75, 200)
cv2.imshow('Image edges', edged_img)
cv2.waitKey(0)
```

- **Purpose:**
  - **Gaussian Blur:** Removes noise and smooths the image.
  - **Canny Edge Detector:** Finds the outlines of objects (like the document edges) in the image.

Step 5: Finding Contours

```
cnts, _ = cv2.findContours(edged_img, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]
for c in cnts:
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.02 * peri, True)
    if len(approx) == 4:
        doc = approx
        break
```

- **Purpose:**
  - **Find Contours:** Detect all continuous lines or curves in the edged image.
  - **Sort and Approximate:** From the largest contours, approximate their shape. The first contour with four points (corners) probably outlines the document.

Step 6: Marking the Detected Corners

```
p = []
for d in doc:
    tuple_point = tuple(d[^0])
    cv2.circle(img_resize, tuple_point, 3, (0, 0, 255), 4)
    p.append(tuple_point)
cv2.imshow('Circled corner points', img_resize)
cv2.waitKey(0)
```

- **Purpose:**
    - Visually confirm by drawing circles at the detected document corners.
    - Prepares to extract these corner points for perspective correction.

Step 7: Perspective Transformation

```
warped_image = perspective_transform(copy, doc.reshape(4, 2) * ratio)
warped_image = cv2.cvtColor(warped_image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Warped Image", imutils.resize(warped_image, height=650))
cv2.waitKey(0)
```

In transform.py:

```
def order_points(pts):
    rect = np.zeros((4, 2), dtype="float32")
    s = pts.sum(axis=1)
    rect[^0] = pts[np.argmin(s)]  # Top-left
    rect[^2] = pts[np.argmax(s)]  # Bottom-right
    diff = np.diff(pts, axis=1)
    rect[^1] = pts[np.argmin(diff)]  # Top-right
    rect[^3] = pts[np.argmax(diff)]  # Bottom-left
    return rect

def perspective_transform(image, pts):
    rect = order_points(pts)
    (tl, tr, br, bl) = rect
    widthA = np.linalg.norm(br - bl)
    widthB = np.linalg.norm(tr - tl)
    maxWidth = max(int(widthA), int(widthB))

    heightA = np.linalg.norm(tr - br)
    heightB = np.linalg.norm(tl - bl)
    maxHeight = max(int(heightA), int(heightB))

    dst = np.array([
        [0, 0],
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight - 1]], dtype="float32")
    transform_matrix = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(image, transform_matrix, (maxWidth, maxHeight))
    return warped
```

- **Purpose:**
    - *Order the points* to consistently identify corners.
    - *Perspective warping* "flattens" the document as if viewed from directly above, correcting for camera angle distortion.

Step 8: Thresholding for a Scanned Look

```
T = threshold_local(warped_image, 11, offset=10, method="gaussian")
warped = (warped_image > T).astype("uint8") * 255
cv2.imwrite('./scan.png', warped)
```

- **Purpose:**
    - *Local Adaptive Thresholding* highlights text and removes background noise, imitating a high-quality scanned document, not just a photograph.
    - *Saving* as PNG preserves quality.

Step 9: Displaying and Cleaning Up

```
cv2.imshow("Final Scanned image", imutils.resize(warped, height=650))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Purpose:**
    - Show the final result: a clean, overhead "scan" of your document.
    - Clean up windows after all operations.

In summary:

The code takes a photo of a document, detects and isolates its edges, corrects any tilt or angle using perspective transformation, enhances clarity with adaptive thresholding, and finally outputs a high-contrast image resembling a scanned document—all using stepwise, logical processing.