שאלה 2:

1. **קריאת הנתונים והכנת הפאזל**:
אנחנו מפעילים את prepare_puzzle כדי לאתחל את התיקייה abs_pieces ולקרוא את קובץ matches.txt. הפונקציה מחזירה לנו את רשימת ההתאמות בין נקודות הפאזל, האם מדובר בטרנספורמציה אפינית או הומוגרפית, וכמה חלקים יש בפאזל.
2. **חישוב מטריצת הטרנספורמציה**:
בתוך get_transform, אנחנו מסתכלים על הנקודות שמתאימות בין החלק הראשון לחלק הנוכחי, ובודקים אם אנחנו מבצעים טרנספורמציה אפינית או הומוגרפית.

```
def get_transform(matches, is_affine):
    # use src/dst coordinates to estimate transformations
    src_cords=matches[:, 0]
    dst_cords=matches[:, 1]
    if not is_affine:
        result, _ = cv2.findHomography(src_cords, dst_cords)
    else:
        result, _ = cv2.estimateAffine2D(src_cords, dst_cords)

    return result
```

בהתאם, אנחנו מפעילים את cv2.estimateAffine2D או cv2.findHomography כדי לקבל את מטריצת הטרנספורמציה.

3. **היפוך הטרנספורמציה והחלתה על החלק**:

בפונקציה `inverse_transform_target_image`, מסתכלים על המטריצה שחישבנו ב־`get_transform` ובודקים אם היא 2×3 (אפין) או 3×3 (הומוגרפיה). אנחנו הופכים את המטריצה המתאימה ואז "מותחים" (warp) את התמונה החדשה לגודל הקנבס, כך שתתאים לתמונה הראשונה.

```
1 usage   Idan Morad
def inverse_transform_target_image(target_img, original_transform, output_size):
    # determine transform type by matrix shape, invert accordingly
    if not original_transform.shape == (2, 3):
        # homography case
        inverse_h = np.linalg.inv(original_transform)
        result = cv2.warpPerspective(target_img, inverse_h, output_size, flags=cv2.INTER_LINEAR)
    else:
        # affine case
        inverse_aff = cv2.invertAffineTransform(original_transform)
        result = cv2.warpAffine(target_img, inverse_aff, output_size, output_size, flags=cv2.INTER_LINEAR)
    return result
```

4. **הדבקה (stitch) של החלקים**:

בפונקציה `stitch`, משווים בין שני הדימויים שכבר מכוילים על אותו קנבס, ולכל פיקסל לוקחים את הערך המרבי (`cv2.max`).

אם אין חפיפה – ניקח את הפיקסל מהתמונה שמכילה אותו, ואם יש חפיפה, משתמשים בפיקסל התואם לכל חלק.

- לא צירפתי תמונה כי זו פונקציה של שורה אחת.

5. **הרכבת הפאזל השלם**:

מתחילים עם `piece_1.jpg` כתמונה ראשית (שנמצא כבר במקום הנכון), ושומרים את המימדים שלו.

```
#load piece1 which is already in correct position
piece1 = cv2.imread(os.path.join(pieces_path, 'piece_1.jpg'))
h, w, _ = piece1.shape
#put first piece in final puzzle and iterate over rest of image pieces
final_puzzle = piece1
```

לכל חלק נוסף, מחשבים את מטריצת הטרנספורמציה, מוצאים את ההופכית, ושומרים את התמונה המהופכת ב־`abs_pieces`. בשלב האחרון, "מדביקים" כל חלק על התמונה הסופית באופן איטרטיבי.

```python
for i, filename in enumerate(all_pieces):
    #if we're looking at the first peice (which we already used) just skip iteraton
    if filename == 'piece_1.jpg':
        continue
    # get transform to place piece onto the final puzzle
    transform = get_transform(matches[i - 1], is_affine)
    # load current piece and inverse transform it
    curr_image = cv2.imread(os.path.join(pieces_path, filename))
    inverse_piece = inverse_transform_target_image(curr_image, transform, output_size: (w, h))
    # save inverted image
    outpath = os.path.join(edited, f'{filename.split(".")[0]}_relative.jpg')
    cv2.imwrite(outpath, inverse_piece)
    # stitch it into the puzzle
    final_puzzle = stitch(final_puzzle, inverse_piece)

sol_file = f"solution.jpg"
cv2.imwrite(os.path.join(puzzle, sol_file), final_puzzle)
```

לבסוף, שומרים את התמונה השלמה כ־`solution.jpg`.