Final Task

Welcome to the final task of OpenCV.

- ❖ The aim of this task is to familiarize you with important concepts of Image Processing, and Python Programming

- ❖ This task is based on Image Processing using OpenCV library of Python. The task here is to determine different shapes and colors in a given image.**)**

The objective of this task is to use OpenCV library in Python to write a function `detect_shapes()` which is able to:

- Detect shapes in a given image
- Detect color of shapes
- Detect centroid of shape

1. Final Task Folder Layout : Download the given task folder **Final_task.zip** for the files necessary to complete Task 1.

Task folder has the following layout:

- *Final_task folder*
  - Final_task.py
  - *test_images folder*
    - test_image_1.png to test_image_15.png

2. Problem Statement

Each image in *test_images* folder of Task_1A contains multiple shapes which are to be detected.

- The shape could be any of the following: Triangle/Square/Rectangle/Pentagon/Circle
- The color of each shape could be any of the following: Red/Blue/Green/Orange

A sample image can be found below for reference:

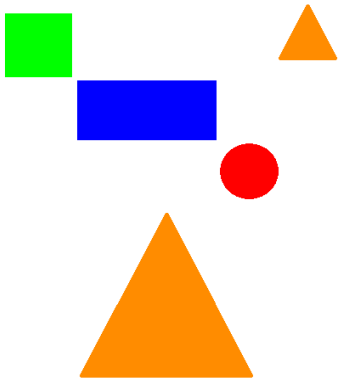

Figure 1 sample image

Task is to find the following details for each image in the manner shown below:

1. Detect all the non-white shapes in the images
2. Store the details of these detected shapes in a list in the same order as mentioned below:
   - ‘test_image_name’: [‘Color’, ‘Shape’, (cX, cY)]
     - Example => [‘Red’, ‘Circle’, (588, 370)]
3. All detected shapes in a single image should be stored as a nested list as shown below:

```
For test_image_14.png
[['Orange', 'Triangle', (269, 457)], ['Red', 'Circle', (367, 271)], ['Blue', 'Rectangle', (246, 195)], ['Green', 'Square', (11
9, 115)], ['Orange', 'Triangle', (436, 109)]]
```

Figure 2 : Sample output

4. It is mandatory to make sure that each of these details are of definite data type as listed below:
   - ‘Color’ => String in single quotation marks, with only the first letter in capital
   - ‘Shape’ => String in single quotation marks, with only the first letter in capital
   - cX => Int value (centroid coordinate of shape on horizontal X-axis direction)
   - cY => Int value (centroid coordinate of shape on vertical Y-axis direction)

Note:

1. This data type convention should be strictly followed. Failing to follow this data type convention will lead to failure of test cases and deduction of marks.
2. For cX and cY consider the top left point of the image as origin (0,0) reference.

---

3. Task Instructions

For this part of the task we have provided a “snippet” of outline code in the `Final_task.py` file:

- Teams are NOT allowed to import any other library/module, other than the ones already imported in the `Final_task.py`
- Teams are NOT allowed to edit the main() function.
- Teams should modify the detect_shapes() function to take the image as input and return a nested list with details of the non-white shapes as mentioned above.

Function name	**`detect_shapes()`**
Purpose	This function takes the image as an argument and returns a nested list containing details of colored (non-white) shapes in that imag
Input Argument	`img : [ numpy array ]` numpy array of image returned by cv2 library
Returns	`detected_shapes : [ list ]` nested list containing details of colored (non-white) shapes present in image
Example Call	`shapes = detect_shapes(img)`

---

4. Running your Solution

- Run the command: `python task_1a.py` to execute your solution.
- When you run the `task_1a.py`, as a default, the main() function will feed the file path of test_image_1.png file which is present in the *test_images* folder as shown above.
- It will print the output of the function i.e. the nested list is returned.
- It will then ask you if you want to run the same code for the rest of the images in the *test_images* folder. If you are satisfied with the output then you can press 'y' and press enter to proceed.
- The final output of `task_1a.py` should resemble Figure 3.

```
============================================

For test_image_1.png
[['Red', 'Circle', (588, 370)], ['Green', 'Square', (325, 145)]]

Do you want to run your script on all test images ? => "y" or "n": y

============================================

For test_image_1.png
[['Red', 'Circle', (588, 370)], ['Green', 'Square', (325, 145)]]

============================================

For test_image_2.png
[['Blue', 'Circle', (274, 348)], ['Blue', 'Pentagon', (648, 420)], ['Green', 'Triangle', (766, 165)], ['Red', 'Rectangle', (30
1, 118)]]

============================================

For test_image_3.png
[['Green', 'Rectangle', (467, 467)], ['Blue', 'Square', (425, 272)], ['Green', 'Circle', (290, 110)], ['Red', 'Triangle', (643,
194)]]

============================================

For test_image_4.png
[['Blue', 'Square', (325, 451)], ['Red', 'Rectangle', (249, 199)], ['Blue', 'Circle', (588, 226)]]
```

Figure 3: Overall output

- Once your code runs successfully, you can do a manual verification of the output to ensure you are getting proper output.
- *Your solution will be tested against blind test cases. Ensure your solution is robust and considers all the corner cases.*