# Task 1.1 - Problem Statement

In this task, you will be required to construct 3 functions detect_markers(), drawCube() and drawCylinder().

Go to *Problem Statement* folder in Task 1.1 and open the *detect.py* python file.
We will now explain the parts of the code to help you modify the functions.

**MAIN CODE**

```
if __name__== "__main__":
    cam, dist = getCameraMatrix()
    img = cv2.imread("..\\TestCases\\image_1.jpg")
    aruco_list = detect_markers(img, cam, dist)
    for i in aruco_list:
        img = drawAxis(img, aruco_list, i[0], cam, dist)
        ##  img = drawCube(img, aruco_list, i[0], cam, dist)
        ##  img = drawCylinder(img, aruco_list, i[0], cam, dist)
    cv2.imshow("img", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

The first line of the code calls the getCameraMatrix() function and returns the Camera matrix and Distortion matrix in two variables **cam** and **dist**.

The second line reads an image file from the TestCases folder using cv2.imread() function. You can specify the image file to be used by changing the parameter specified in cv2.imread().

The third line calls the detect_markers() function which takes **img, cam** and **dist** as input parameters. The output **aruco_list** is a list data structure which we will explain later.

In lines 4-7, a for-loop is called which iterates through **aruco_list**. For each loop call, it calls the drawAxis function which will draw 3 perpendicular axes on the arUco image.

2 more functions drawCube() and drawCylinder() are commented out. They can be uncommented to test the functions when you have written them.

The first function you need to modify is the detect_markers() function.

**FUNCTION**

```
def detect_markers(img, camera_matrix, dist_coeff):
    markerLength = 100
    aruco_list = []
    ################### INSERT CODE HERE ###################

    ########################################################

    return aruco_list
```

**Parameters**
*img*: Image expressed as a numpy array.

*camera_matrix*: Camera matrix extracted from .npz file

*dist_coeff*: Distortion matrix extracted from .npz file

**Return**

*aruco_list*: detect_markers() returns information about ArUco markers detected in image in form of a list of tuples.

Each tuple in aruco_list denotes information for a separate ArUco marker.

Each of the tuple has 4 elements. They are explained as follows:

[ (aruco_id, aruco_centre, rvec, tvec) ]

**aruco_id:** ID of the ArUco marker.
**aruco_centre:** pixel coordinates of the centre of the ArUco marker.
**rvec:** Rotation Vector of ArUco Marker
**tvec:** Translation Vector of ArUco Marker

Consider the input image to detect_markers() is given in Figure 1.

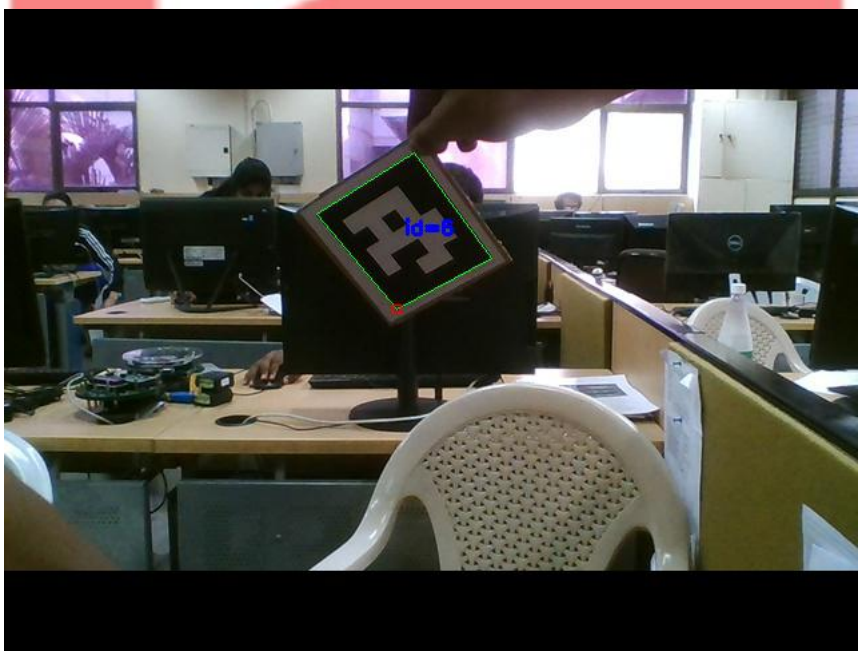


Figure 1: Aruco_Image

The corresponding output of detect_markers() should resemble Figure 2

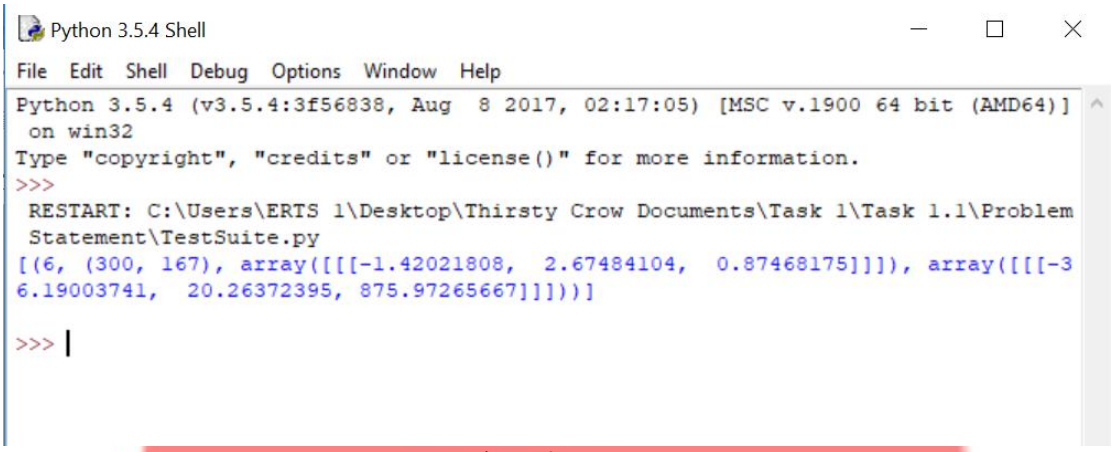


Figure 2: Output

If there are 2 or more ArUco markers in the image the format of aruco_list should be:

```
[ (aruco_id_1, aruco_centre_1, rvec_1, tvec_1) ,
  (aruco_id_2, aruco_centre_2, rvec_2, tvec_2) ]
```

Consider the input image to detect_markers() is given in Figure 3.

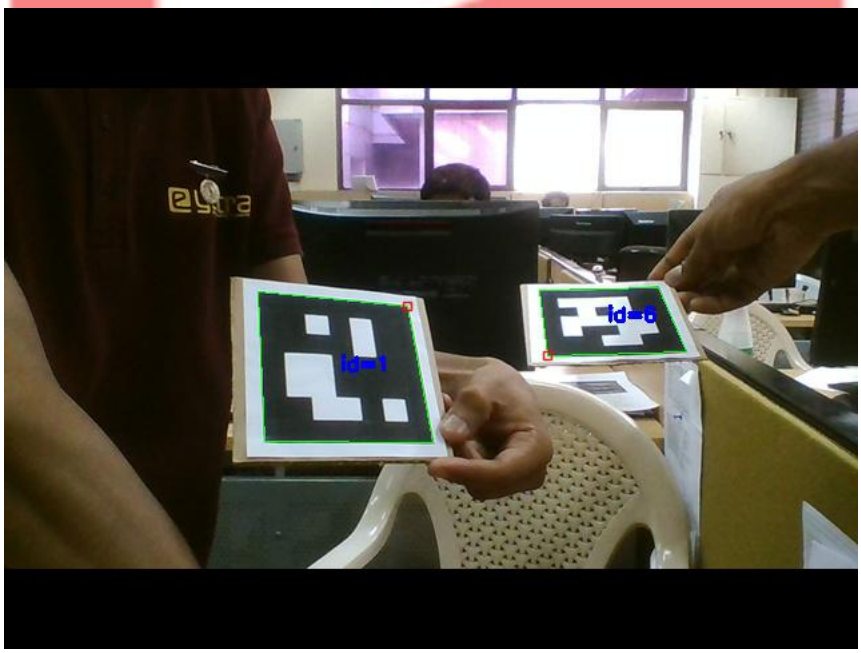


Figure 3: Aruco Image

The corresponding output of detect_markers() should resemble Figure 4

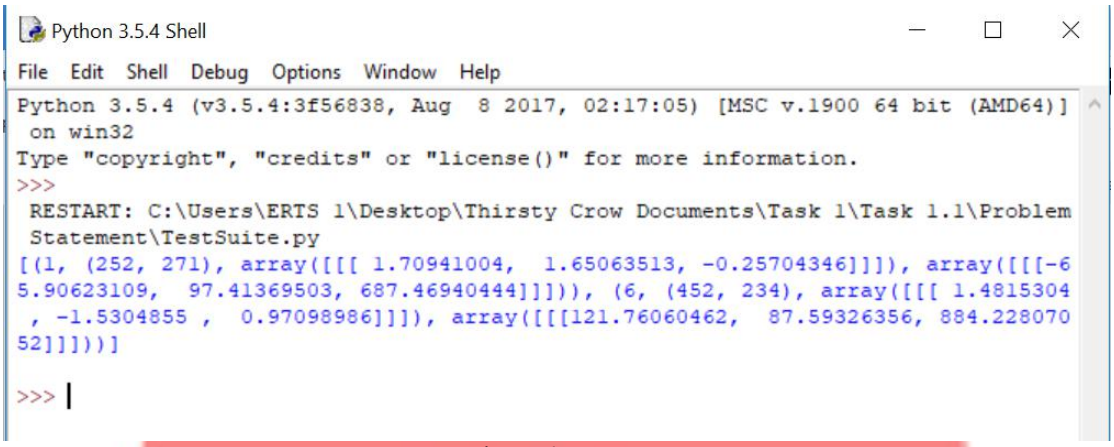


Figure 4: Output

Kindly go through all the tutorials in the *Tutorials* folder to get a clear understanding of ArUco marker detection.

## Drawing Functions

There are 3 drawing functions in *detect.py,* drawAxis(), drawCube(), and drawCylinder().

### 1. drawAxis()

**FUNCTION**

```
def drawAxis(img, aruco_list, aruco_id, camera_matrix, dist_coeff):
    for x in aruco_list:
        if aruco_id == x[0]:
            rvec, tvec = x[2], x[3]
    markerLength = 100
    m = markerLength/2
    pts = np.float32([[-m,m,0],[m,m,0],[-m,-m,0],[-m,m,m]])
    pt_dict = {}
    imgpts, _ = cv2.projectPoints(pts, rvec, tvec, camera_matrix, dist_coeff)
    for i in range(len(pts)):
        pt_dict[tuple(pts[i])] = tuple(imgpts[i].ravel())
    src = pt_dict[tuple(pts[0])];  dst1 = pt_dict[tuple(pts[1])];
    dst2 = pt_dict[tuple(pts[2])];  dst3 = pt_dict[tuple(pts[3])];

    img = cv2.line(img, src, dst1, (0,255,0), 4)
    img = cv2.line(img, src, dst2, (255,0,0), 4)
    img = cv2.line(img, src, dst3, (0,0,255), 4)
    return img
```

**drawAxis()** function draws 3 perpendicular axes on one corner of the ArUco marker.

**Parameters**
*img*: Image expressed as a numpy array.

*aruco_list:* aruco_list as returned by detect_markers()

aruco_id: ID of the specific aruco marker on which axis

*camera_matrix*: Camera matrix extracted from .npz file

*dist_coeff*: Distortion matrix extracted from .npz file

**Return**

img: Output image

Consider the following image as input to drawAxis() as shown in Figure 5

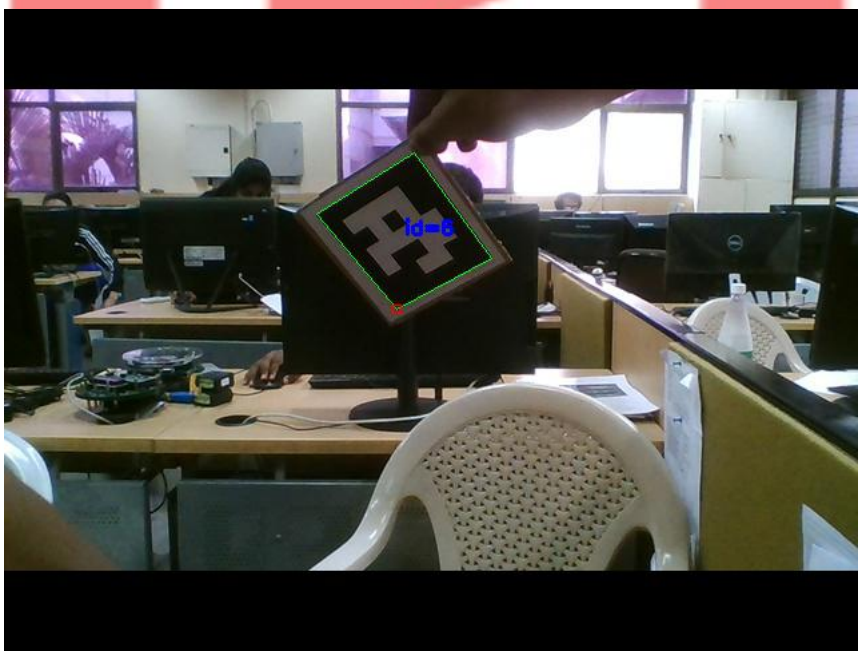


Figure 5: Aruco Image

In the function call to drawAxis(), aruco_id is specified as 6.

The corresponding output image is shown in Figure 6.

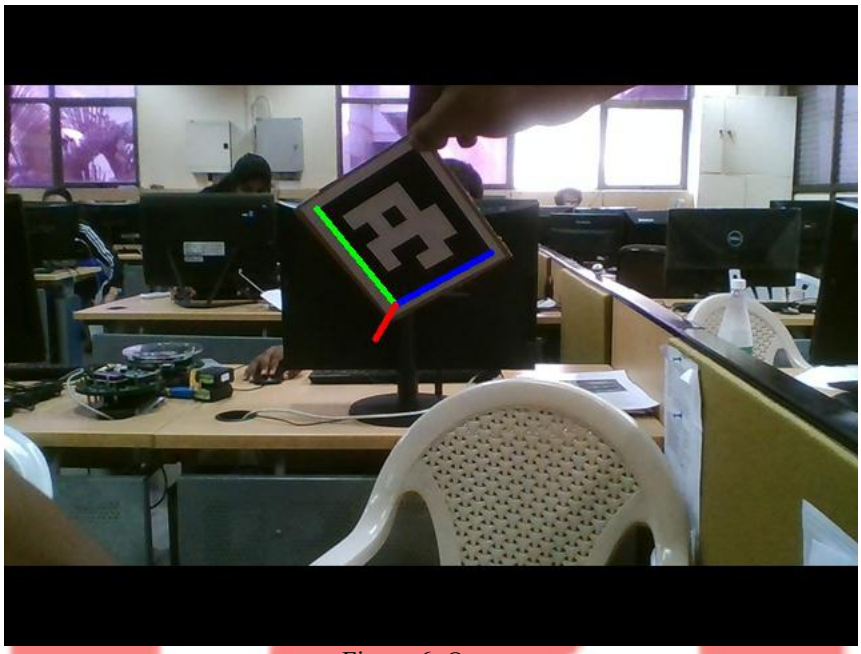

Figure 6: Output

Output Images for all other test cases have been already been generated. Navigate to SavedResults > drawAxis folder to view them.

You do not have to modify this function. Just go through the code written in the function and try to understand each and every line of code and why it is written. Once you understand the code, you should be able to complete the second function drawCube().

## 2. drawCube()

The drawCube function is used to draw a cube overlaying onto the ArUco marker.

**FUNCTION**

```
def drawCube(img , aruco_list, aruco_id, camera_matrix, dist_coeff):
    for x in aruco_list:
        if aruco_id == x[0]:
            rvec, tvec = x[2], x[3]
    markerLength = 100
    m = markerLength/2
    ################ INSERT CODE HERE ############################

    ##############################################################

    return img
```

**Parameters**

*img*: Image expressed as a numpy array.

*aruco_list:* aruco_list as returned by detect_markers()

aruco_id: ID of the specific aruco marker on which axis

*camera_matrix*: Camera matrix extracted from .npz file

*dist_coeff*: Distortion matrix extracted from .npz file

**Return**

img: Output image

Consider the following image as input to drawCube() as shown in Figure 7

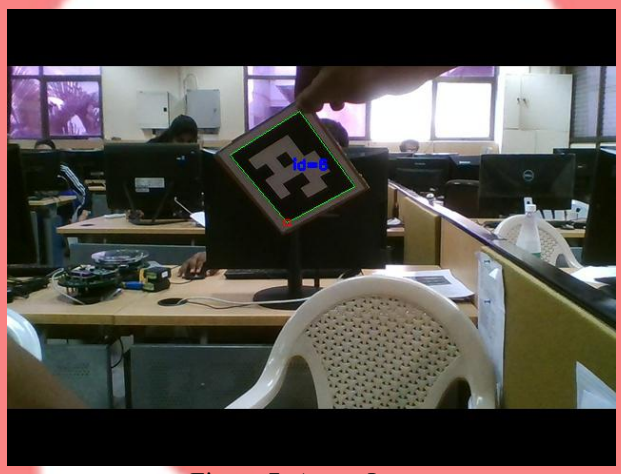


Figure 7: Aruco Image

The corresponding output image is shown in Figure 8.

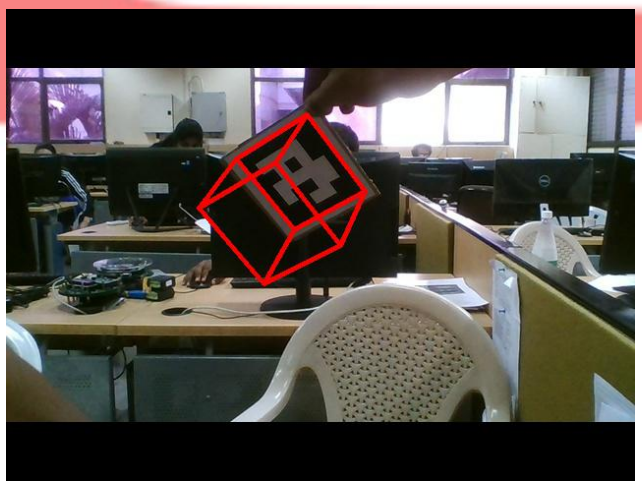

Figure 8: Output

You have to modify this function and write code to generate the output as shown in Fig 8 for all other images.

Use the drawAxis() function as a starting point, try to understand what it does and apply that knowledge in overlaying a cube on the ArUco marker as shown in Fig 8.

## 3. drawCylinder()

The drawCylinder function is used to draw a cylinder overlaying onto the ArUco marker.

**FUNCTION**

```
def drawCylinder(img    ,  aruco_list,  aruco_id,  camera_matrix,
dist_coeff):
    for x in aruco_list:
        if aruco_id == x[0]:
            rvec, tvec = x[2], x[3]
    markerLength = 100
    radius = markerLength/2; height = markerLength*1.5
    ################ INSERT CODE HERE ##########################

    ############################################################

    return img
```

**Parameters**
*img*: Image expressed as a numpy array.

*aruco_list:* aruco_list as returned by detect_markers()

aruco_id: ID of the specific aruco marker on which axis

*camera_matrix*: Camera matrix extracted from .npz file

*dist_coeff*: Distortion matrix extracted from .npz file

**Return**

img: Output image

Consider the following image as input to drawCylinder() as shown in Figure 9

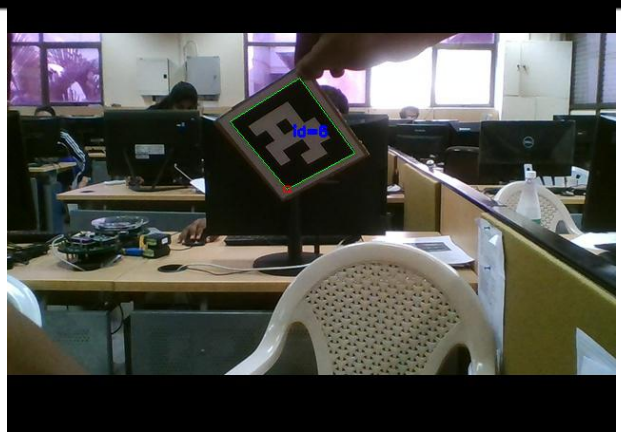


Figure 9: Aruco Image

The corresponding output image is shown in Figure 10.

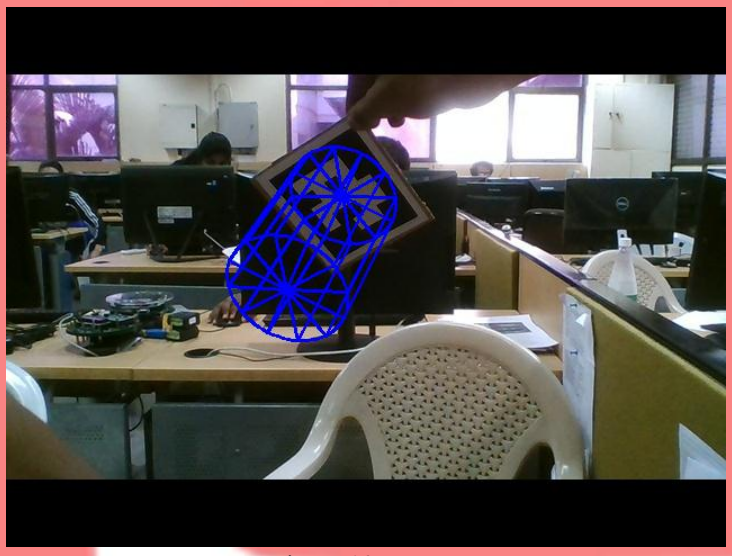

Figure 10: Output

You have to modify this function and write code to generate the output as shown in Fig 10 for all other images.

As mentioned earlier, use drawAxis() as a starting point and build up from there.

## Testing your Solution

After you have completed all three functions as instructed above, you will need to test your solution. Do the following steps to test your solution.

1. Open the TestSuite.py python file using IDLE. **Do not make any changes to the file.**
2. Go to Run > Run Module or Hit F5. If your code is correct, TestSuite.py will execute without errors and you should see python console similar to Figure 11.

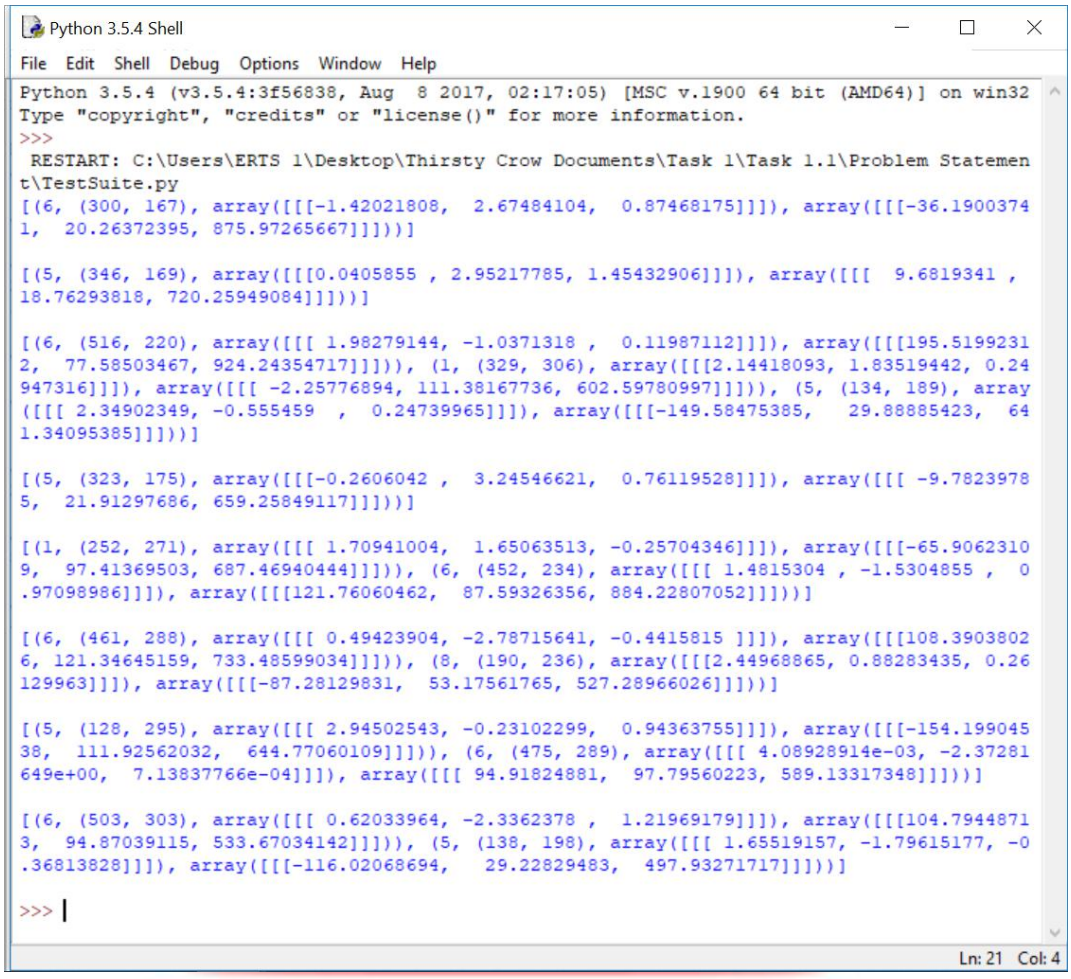


Figure 11: Python Console

3. Navigate to SavedResults> drawCube folder and SavedResults> drawCylinder folder. If TestSuite has executed correctly, output images would have been generated in these 2 folders. Check all the output images.
4. In addition to this a new file called Results.npz would have been generated SavedResults folder

**Congratulations! You have just finished Task 1.1 !**

## Submission Instructions

- ✓ If you have run the *TestSuite.py* function without errors, go to SavedResults folder inside the Task 1.1 folder
- ✓ Rename the SavedResults folder as **eYRC#TC#<Team_id>#Task1.1** . For example if your Team_id is 2343 then the folder name should be **eYRC#TC#2343#Task1.1**
- ✓ Convert your folder into a .zip archive using WinZip or any other software.
- ✓ Check the *Submission Instructions.pdf* for further instructions on how to upload your Task 1.