# 

# 

# Introduction to Computer Vision Lab 02: Drawing Functions

# Instructor: Dr Tariq bashir

# Hamdan Sethi SP23-BAI-015

**Code Files:**

1- [Main](https://github.com/hvmdvvn/Intro-to-Computer-Vision/blob/main/Lab_02_Drawing_Functions.ipynb)  
2- [Optional/Extra](https://github.com/hvmdvvn/Intro-to-Computer-Vision/blob/main/Lab_02_Drawing_Functions_(Optional).ipynb)  
3- [Report](https://docs.google.com/document/d/1W0ChIBlFPcyuFXa10TvxNQCf10uTDqdGhbSmMHAJHyk/edit?usp=sharing)

## Task 01: Drawing Ellipses

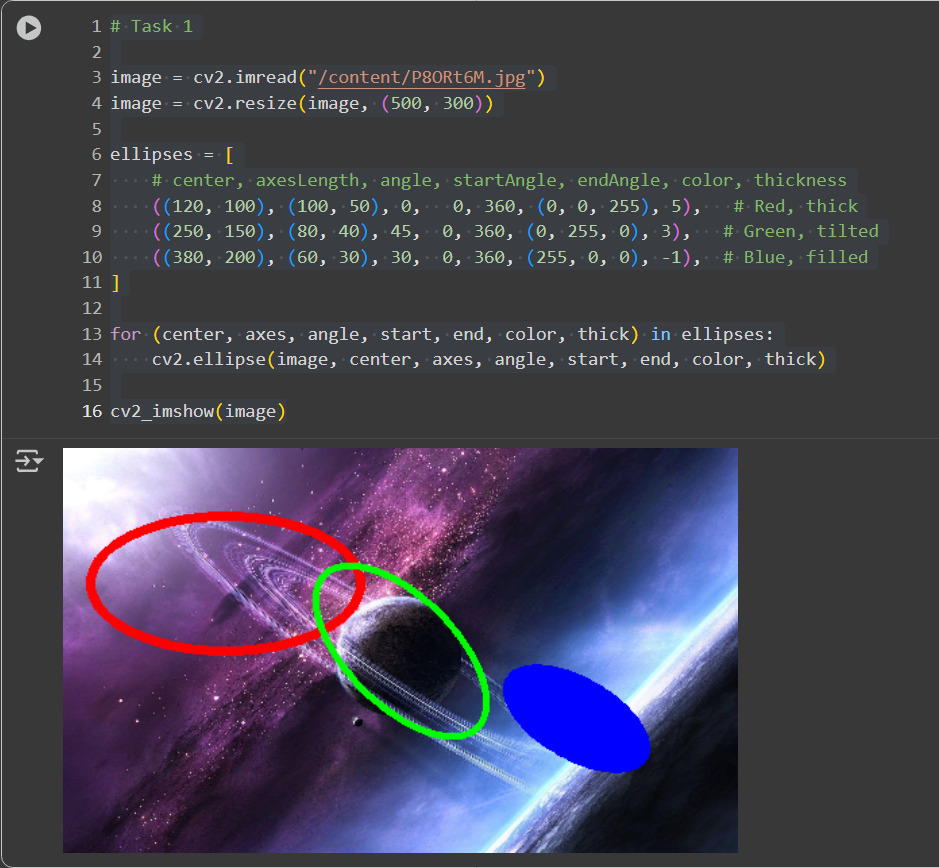
**Description:**

This code loads an image, resizes it to **500×300**, and then draws **three ellipses** with different styles:

* A **red ellipse** with a thick border.
* A **green ellipse** rotated at **45°** with a thinner border.
* A **blue ellipse** tilted at **30°** and **filled in completely**.

Finally, the modified image with all three ellipses is displayed.

#### Image:



**Task 02: Converting Ellipses to Polygon**

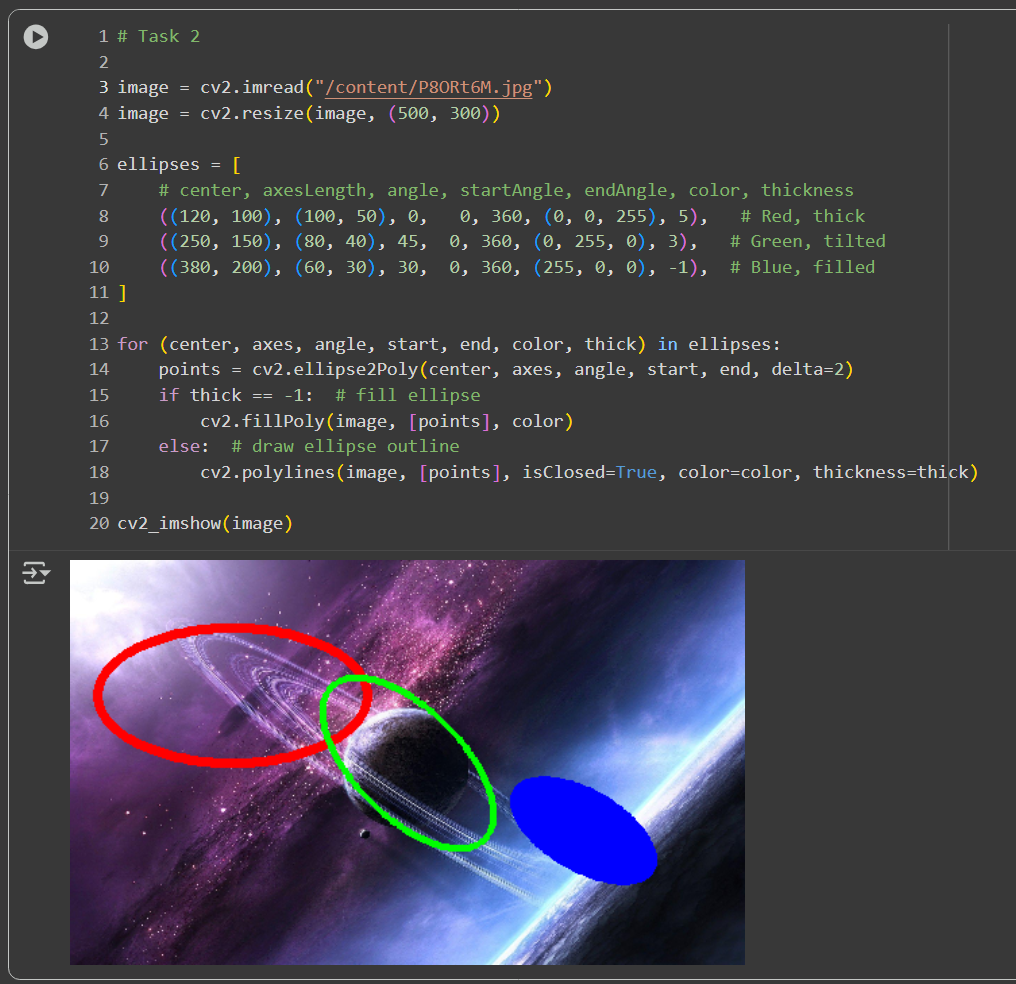
**Description:**

Here’s what happens step by step:

1. The image is loaded and resized to **500×300**.
2. Three ellipses are defined (red, green, and blue with different sizes and angles).
3. For each ellipse:
   * cv2.ellipse2Poly generates a set of polygon points approximating the ellipse.
   * If the thickness is -1, the ellipse is **filled** using cv2.fillPoly.
   * Otherwise, only the **outline** is drawn with cv2.polylines.
4. The result is displayed, showing the same three ellipses, but created from polygonal approximations.

**Task 1 draws ellipses directly, while Task 2 recreates them using polygonal approximations (points + polyline/polyfill).**

**Image:**



**Task 03: Filling Convex Polygons**

**Description:**

Here’s what it does:

1. Loads the input image and resizes it to **500×300**.
2. Defines three ellipses (red, green, blue) with different sizes, orientations, and thickness styles.
3. For each ellipse:
   * Uses cv2.ellipse2Poly to generate polygon points along the ellipse boundary.
   * If thickness == -1, it uses **cv2.fillConvexPoly** to fill the ellipse area (blue one).
   * Otherwise, it uses **cv2.polylines** to draw only the outline (red and green ellipses).
4. Finally, displays the result with the three ellipses.

**Difference from Task 2:**

* Task 2 used cv2.fillPoly for filling.
* Task 3 uses cv2.fillConvexPoly, which is optimized for convex shapes like ellipses and guarantees proper filling without gaps.

**Image:**



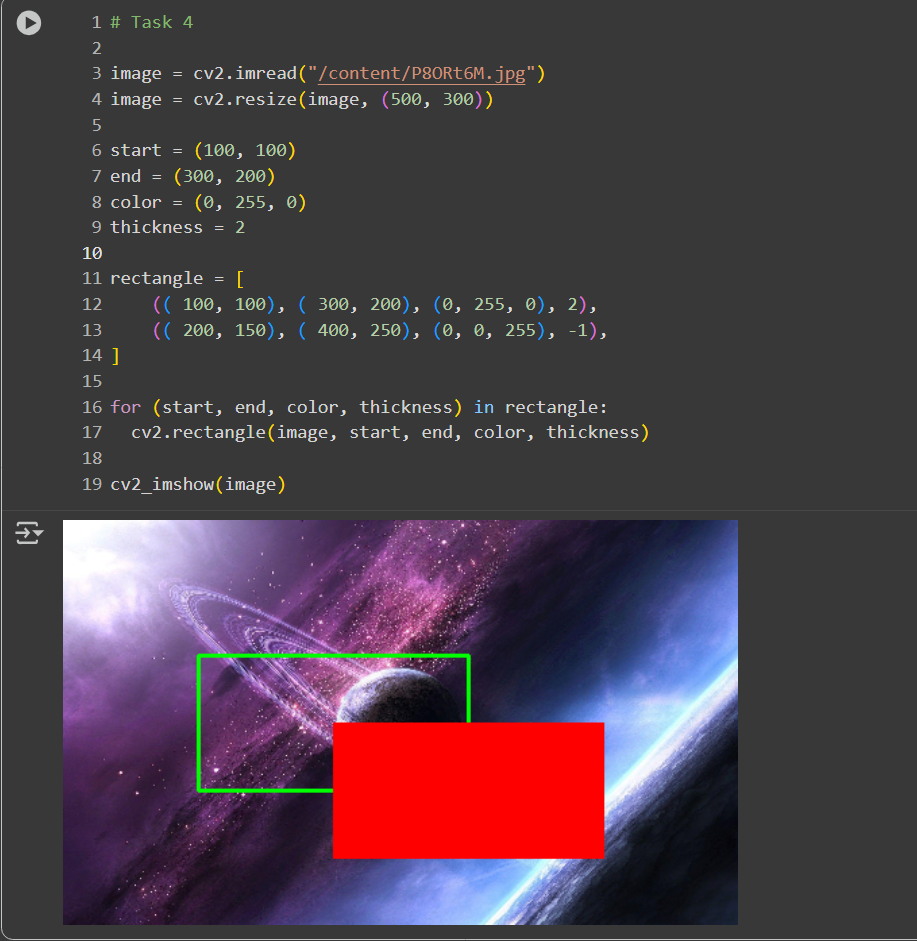
**Task 04: Drawing Rectangles**

**Description:**

Here’s what happens step by step:

1. The image is loaded and resized to **500×300**.
2. Two rectangles are defined in a list:
   * **Green rectangle** from (100, 100) to (300, 200) with outline thickness 2.
   * **Red rectangle** from (200, 150) to (400, 250) with -1 thickness, meaning it is **filled**.
3. A loop iterates through the rectangle definitions and uses cv2.rectangle() to draw each one.
4. The final image is displayed with both rectangles.

**Image:**



**Task 05(a): Draw Lines**

**Description:**

The image is loaded and resized to **500×300**.

A list of line definitions is created, where each entry contains:

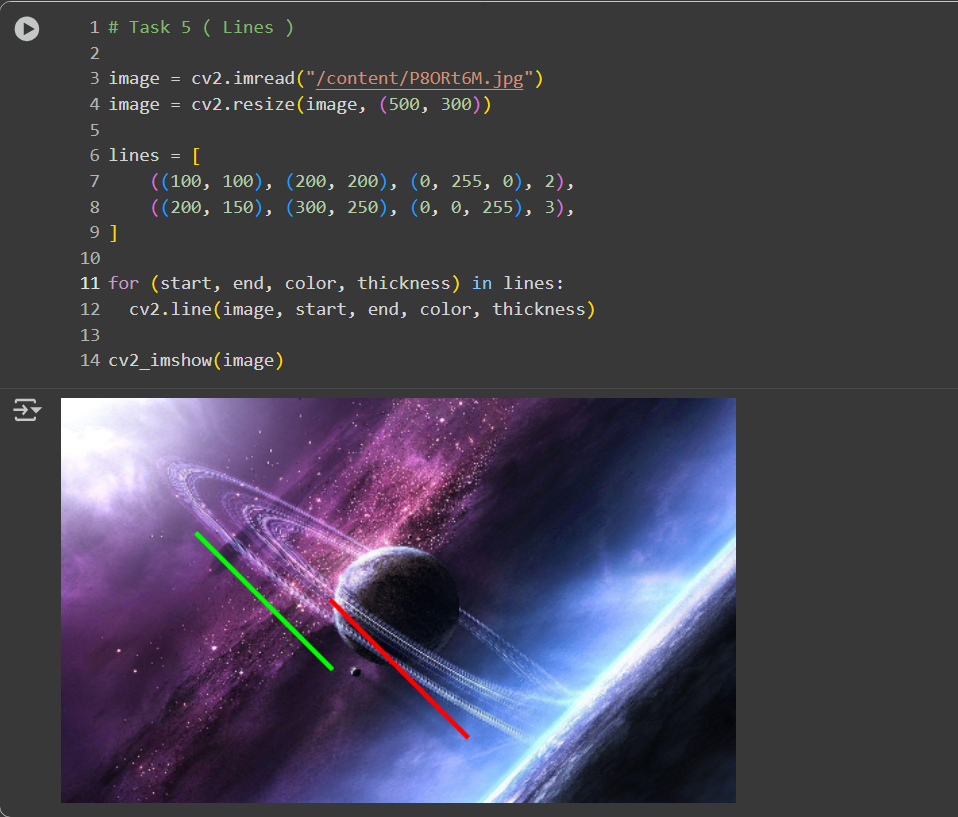
* start and end coordinates
* color (BGR format)
* thickness

Two lines are drawn:

* A **green line** from (100, 100) to (200, 200) with thickness 2.
* A **red line** from (200, 150) to (300, 250) with thickness 3.

The modified image with both lines is displayed.

**Image:**



**Task 05(b): Draw Arrows**

**Description:**

The image is loaded and resized to **500×300**.

A list of arrow definitions is created with:

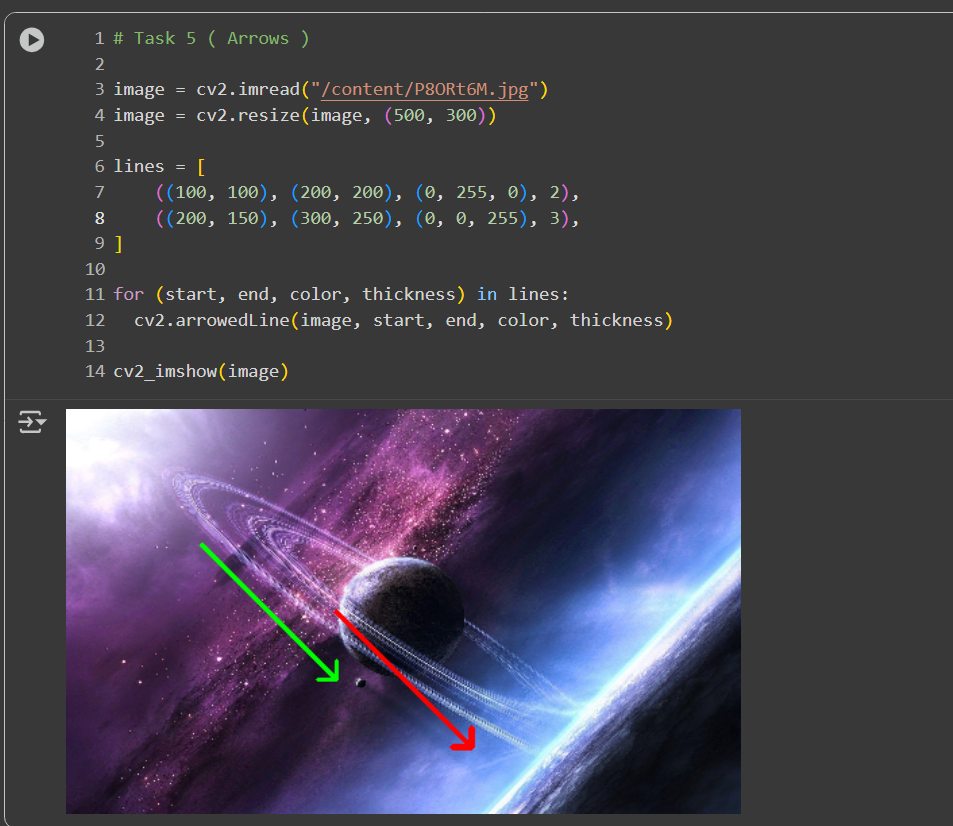
* start and end coordinates
* color (in BGR)
* thickness

Two arrowed lines are drawn:

* A **green arrow** from (100, 100) to (200, 200) with thickness 2.
* A **red arrow** from (200, 150) to (300, 250) with thickness 3.

The final image displays both arrows.

**Image:**



**Task 06: Draw Polygons**

**Description:**

The image is loaded and resized to **500×300**.

A list of polylines is defined, each with:

* A set of points (x, y) forming vertices
* A color in BGR format
* A thickness for the line width

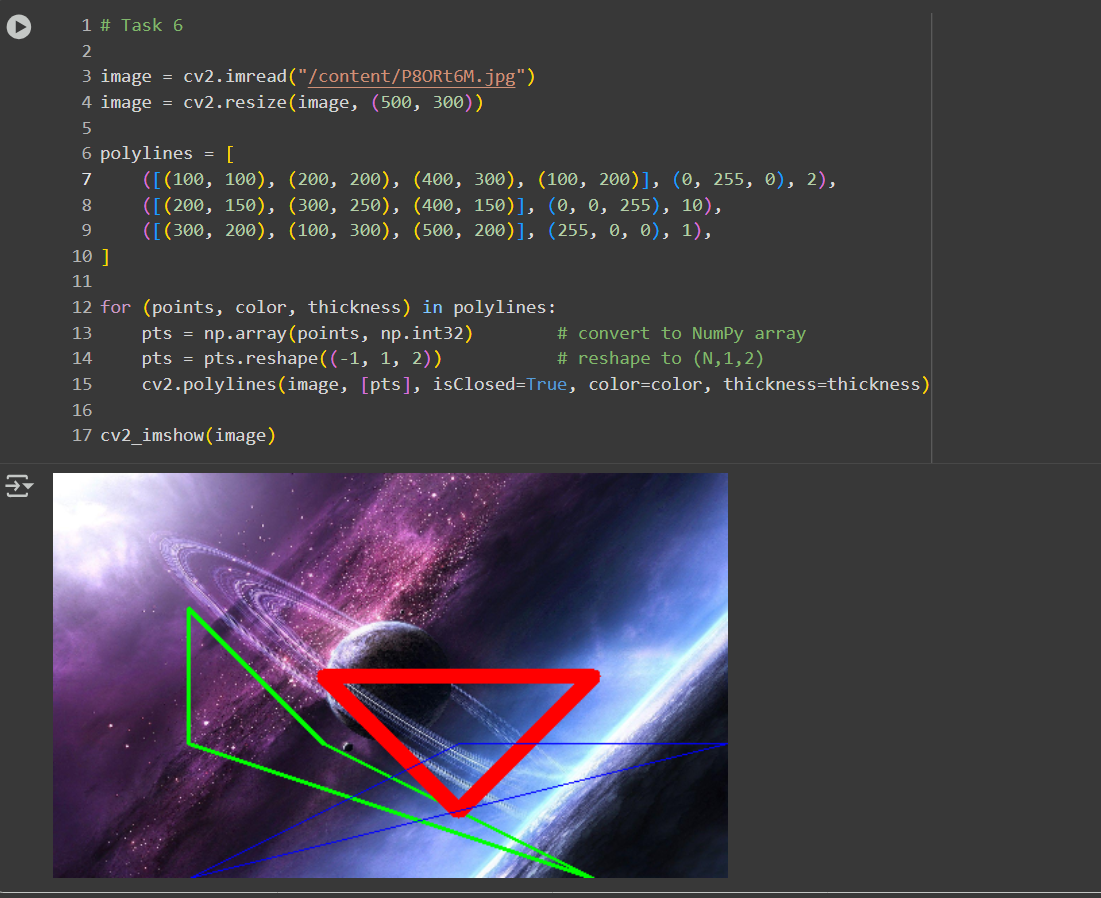
Each polyline is converted to a **NumPy array** and reshaped to the format required by OpenCV: (N, 1, 2).

cv2.polylines() is used to draw each polyline, with isClosed=True meaning the last point connects back to the first.

The final image shows:

* A **green quadrilateral** with thickness 2.
* A **red triangle** with thickness 10.
* A **blue triangle** with thin lines (1).

**Image:**



**Task 07: Write Text**

**Description:**

The image is loaded and resized to **500×300**.

The text "Hello, World!" is defined along with font settings:

* **Font** → cv2.FONT\_HERSHEY\_SIMPLEX
* **Font scale** → 1
* **Color** → White (255, 255, 255)
* **Thickness** → 2
* **Line type** → cv2.LINE\_AA for smooth edges

The size of the text is calculated using cv2.getTextSize(), and then (text\_x, text\_y) is computed to **center the text** in the image.

cv2.putText() is used to render the text on the image.

The final image shows the **centered white text** "Hello, World!".

**Image:**

