VIDHI ROHIRA

S.Y BTECH

COMPUTER ENGINEERING

PD LAB 6

231071052

LABORATORY 6

AIM:- To create an image editor using open cv and tkinter.

THEORY:-

Q] WHAT IS OPEN CV?

OpenCV (Open Source Computer Vision Library) is an open-source software library focused on computer vision and image processing tasks. It provides a wide range of tools and algorithms for real-time image analysis, object detection, face recognition, and video processing. Written in C++, it also has bindings for languages like Python, Java, and MATLAB. OpenCV is widely used in applications such as robotics, machine learning, augmented reality, and surveillance systems. It is highly efficient and optimized for performance on various platforms, including desktop, mobile, and embedded systems.

Q] WHAT IS TKINTER?

Tkinter is the standard Python library for creating graphical user interfaces (GUIs). It provides a simple and intuitive way to design windows, buttons, menus, and other graphical elements. Tkinter is built on top of the Tcl/Tk GUI toolkit, making it lightweight and platform-independent. It allows developers to build desktop applications with event-driven programming, where user actions like clicks or key presses trigger specific functions. Tkinter is widely used for small to medium-sized GUI applications due to its ease of use and integration with Python.

Q] WHAT IS IMAGE EDITOR?

An image editor is a software application designed for modifying and enhancing digital images. It provides tools for tasks like cropping, resizing, adjusting brightness/contrast, and applying filters or effects. More advanced features may include layers, masking, color correction, and retouching. Image editors are widely used in graphic design, photography, and digital art creation. Popular examples include Adobe Photoshop, GIMP, and Corel PaintShop Pro. Users can manipulate image properties or create visual content from scratch, making image editors versatile tools for both professionals and hobbyists.

Q] HOW TO IMPLENT AN IMAGE EDITOR USING OPEN CV AND TKINTER?

Implementing a simple image editor using **OpenCV** and **Tkinter** combines the powerful image processing features of OpenCV with Tkinter's graphical user interface. Below is a step-by-step guide to create a basic image editor that allows users to open, edit, and save images with features like brightness/contrast adjustment, resizing, and applying filters.

Steps:

1. **Install Required Libraries:** Install OpenCV and Tkinter (which comes with Python by default):

```
pip install opencv-python
```

1. **Basic Structure:**
    - Use Tkinter for the GUI (buttons, sliders, file dialogs).
    - Use OpenCV for image processing tasks (loading, editing, and saving images).

CODE

```
import os
import tkinter as tk
from tkinter import filedialog, simpledialog, colorchooser, messagebox, ttk
from PIL import Image, ImageTk
import cv2
import numpy as np

class EnhancedEditor:
    def __init__(self, root):
        self.root = root
        self.root.title("Enhanced Image Editor with Drawing, Undo/Redo,
Eraser, Text, and Dark Mode")
        self.root.geometry("800x600")

        # Modes and states
        self.mode = "image"
        self.drawing_mode = False
        self.erase_mode = False
        self.draw_color = "black"
        self.erase_color = "white"
        self.line_width = 3
        self.shape = "line"  # Default shape
        self.filled_shape = False  # Whether to fill shapes
        self.is_dark_mode = False  # Light/Dark mode state

        # Actions stack
        self.actions = []
        self.redo_stack = []

        # Canvas for drawing and image display
        self.canvas = tk.Canvas(self.root, bg="white")
        self.canvas.pack(fill=tk.BOTH, expand=True)
        self.canvas_image = None

        # Status bar setup
        self.status_bar = tk.Label(self.root, text="Status: Ready", bd=1,
relief=tk.SUNKEN, anchor=tk.W)
        self.status_bar.pack(side=tk.BOTTOM, fill=tk.X)

        # Menu setup
        self.menu = tk.Menu(self.root)
        self.root.config(menu=self.menu)

        self.file_menu = tk.Menu(self.menu, tearoff=0)
        self.menu.add_cascade(label="File", menu=self.file_menu)
        self.file_menu.add_command(label="New", command=self.new_file)
```

```
        self.file_menu.add_command(label="Open", command=self.open_file)
        self.file_menu.add_command(label="Save", command=self.save_file)
        self.file_menu.add_command(label="Save As", command=self.save_as_file)
        self.file_menu.add_separator()
        self.file_menu.add_command(label="Clear Canvas",
command=self.clear_canvas)
        self.file_menu.add_command(label="Exit", command=self.root.quit)

        self.mode_menu = tk.Menu(self.menu, tearoff=0)
        self.menu.add_cascade(label="Mode", menu=self.mode_menu)
        self.mode_menu.add_command(label="Image Mode",
command=self.switch_to_image_mode)
        self.mode_menu.add_command(label="Draw Mode",
command=self.switch_to_draw_mode)

        self.dark_mode_menu = tk.Menu(self.menu, tearoff=0)
        self.menu.add_cascade(label="Appearance", menu=self.dark_mode_menu)
        self.dark_mode_menu.add_command(label="Toggle Dark Mode",
command=self.toggle_dark_mode)

        # Toolbar setup
        self.toolbar = tk.Frame(self.root, bd=1, relief=tk.RAISED)
        self.toolbar.pack(side=tk.TOP, fill=tk.X)

        self.load_icons()
        self.create_toolbar_buttons()

        # Bind mouse events
        self.last_x, self.last_y = None, None
        self.canvas.bind("<Button-1>", self.on_button_press)
        self.canvas.bind("<B1-Motion>", self.on_mouse_drag)
        self.canvas.bind("<ButtonRelease-1>", self.on_button_release)

        # Color palette
        self.create_color_palette()

    def create_toolbar_buttons(self):
        """Create toolbar buttons."""
        buttons = [
            (None, "Resize", self.resize_image),
            (None, "Crop", self.crop_image),
            (None, "Draw", self.toggle_drawing_mode),
            (None, "Color", self.choose_draw_color),
            (None, "Erase", self.toggle_erase_mode),
            (None, "Undo", self.undo),
            (None, "Redo", self.redo),
            (None, "Circle", self.toggle_circle_mode),
            (None, "Rectangle", self.toggle_rectangle_mode),
```

```
            (None, "Freehand", self.toggle_freehand_mode),
            (None, "Text", self.add_text),
            (None, "Clear", self.clear_canvas),
            (None, "Fill Shape", self.toggle_fill_shape)
        ]

        for icon, text, command in buttons:
            button = tk.Button(self.toolbar, text=text, command=command)
            button.pack(side=tk.LEFT, padx=2, pady=2)

        self.stroke_width_label = tk.Label(self.toolbar, text="Stroke Width:")
        self.stroke_width_label.pack(side=tk.LEFT, padx=2, pady=2)

        self.stroke_width_var = tk.StringVar()
        self.stroke_width_var.set(str(self.line_width))
        self.stroke_width_menu = ttk.Combobox(self.toolbar,
textvariable=self.stroke_width_var, values=[str(i) for i in range(1, 11)],
state="readonly")
        self.stroke_width_menu.pack(side=tk.LEFT, padx=2, pady=2)
        self.stroke_width_menu.bind("<<ComboboxSelected>>",
self.change_stroke_width)

    def create_color_palette(self):
        """Create a simple color palette for quick selection."""
        self.color_palette_frame = tk.Frame(self.root)
        self.color_palette_frame.pack(side=tk.TOP, fill=tk.X)

        colors = ['#000000', '#FF0000', '#00FF00', '#0000FF', '#FFFF00',
'#FF00FF', '#00FFFF', '#FFFFFF']
        for color in colors:
            button = tk.Button(self.color_palette_frame, bg=color,
command=lambda c=color: self.set_draw_color(c), width=2)
            button.pack(side=tk.LEFT)

    def set_draw_color(self, color):
        """Set the drawing color based on palette selection."""
        self.draw_color = color
        self.status_bar.config(text=f"Status: Color changed to {color}.")

    def load_icons(self):
        """Load icons for toolbar buttons, with fallback to text if
missing."""
        pass  # Placeholder for future icon loading

    def new_file(self):
        self.image = None
        self.canvas.delete("all")
        self.actions.clear()
```

```
        self.redo_stack.clear()
        self.status_bar.config(text="Status: New file created.")

    def open_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Image files",
"*.jpg *.jpeg *.png *.bmp *.tiff")])
        if file_path:
            self.image = cv2.imread(file_path)
            self.original_image = self.image.copy()
            self.display_image()
            self.status_bar.config(text=f"Status: Opened
{os.path.basename(file_path)}.")

    def save_file(self):
        file_path = filedialog.asksaveasfilename(defaultextension=".png",
filetypes=[("PNG files", "*.png")])
        if file_path:
            cv2.imwrite(file_path, self.image)
            self.status_bar.config(text=f"Status: Saved as
{os.path.basename(file_path)}.")

    def save_as_file(self):
        self.save_file()

    def clear_canvas(self):
        self.canvas.delete("all")
        self.actions.clear()
        self.redo_stack.clear()
        self.status_bar.config(text="Status: Canvas cleared.")

    def switch_to_image_mode(self):
        self.mode = "image"
        self.drawing_mode = False
        self.erase_mode = False
        self.canvas.config(bg="white")
        self.status_bar.config(text="Status: Switched to Image Mode.")

    def switch_to_draw_mode(self):
        self.mode = "draw"
        self.drawing_mode = True
        self.erase_mode = False
        self.canvas.config(bg="white")
        self.status_bar.config(text="Status: Switched to Draw Mode.")

    def display_image(self):
        """Display the image on the canvas."""
        image_rgb = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
        image_tk = ImageTk.PhotoImage(image=Image.fromarray(image_rgb))
```

```
        if self.canvas_image:
            self.canvas.itemconfig(self.canvas_image, image=image_tk)
        else:
            self.canvas_image = self.canvas.create_image(0, 0, anchor=tk.NW,
image=image_tk)

        self.canvas.config(scrollregion=self.canvas.bbox(self.canvas_image))
        self.canvas.image = image_tk

    def resize_image(self):
        if self.image is not None:
            width = simpledialog.askinteger("Resize", "Enter new width:")
            height = simpledialog.askinteger("Resize", "Enter new height:")
            if width and height:
                self.image = cv2.resize(self.image, (width, height))
                self.display_image()
                self.status_bar.config(text=f"Status: Resized image to
{width}x{height}.")

    def crop_image(self):
        if self.image is not None:
            self.canvas.bind("<ButtonPress-1>", self.on_crop_start)
            self.canvas.bind("<B1-Motion>", self.on_crop_drag)
            self.canvas.bind("<ButtonRelease-1>", self.on_crop_end)

    def on_crop_start(self, event):
        self.start_x = event.x
        self.start_y = event.y
        self.rect = self.canvas.create_rectangle(self.start_x, self.start_y,
self.start_x, self.start_y, outline="red")

    def on_crop_drag(self, event):
        self.canvas.coords(self.rect, self.start_x, self.start_y, event.x,
event.y)

    def on_crop_end(self, event):
        self.end_x = event.x
        self.end_y = event.y
        self.image = self.original_image[self.start_y:self.end_y,
self.start_x:self.end_x]
        self.display_image()
        self.canvas.delete(self.rect)
        self.canvas.unbind("<ButtonPress-1>")
        self.canvas.unbind("<B1-Motion>")
        self.canvas.unbind("<ButtonRelease-1>")
        self.status_bar.config(text="Status: Image cropped.")
```

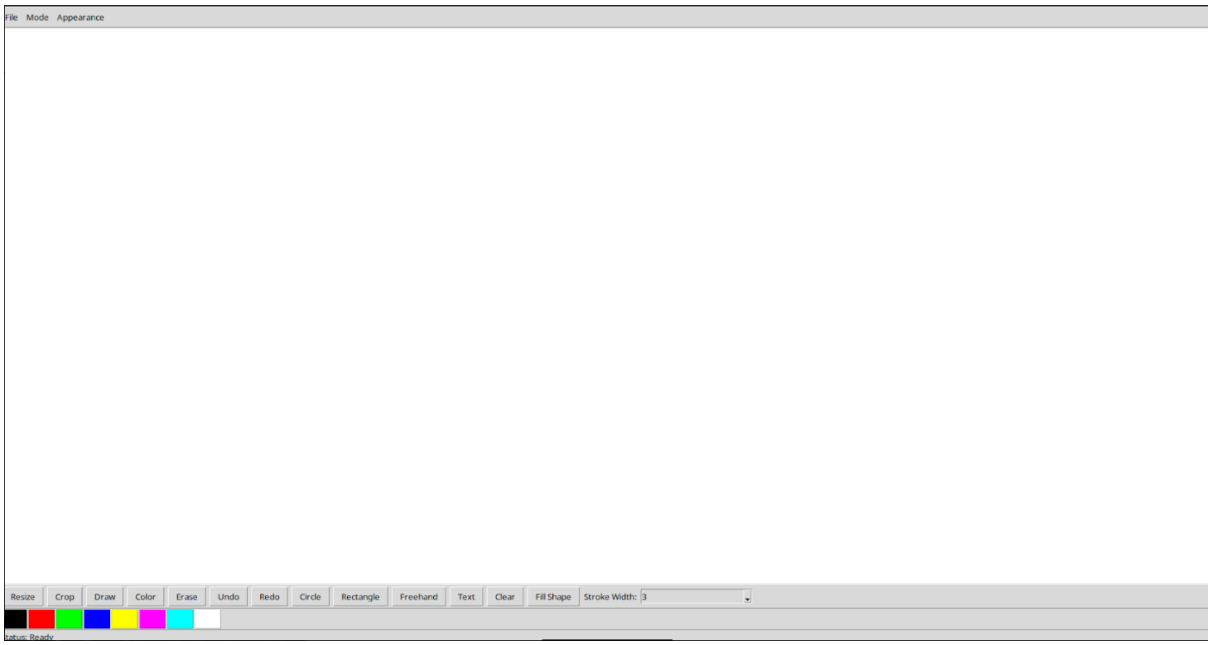
```
    def add_text(self):
        text = simpledialog.askstring("Text Input", "Enter text to add:")
        if text:
            x = simpledialog.askinteger("Text Position", "Enter X position:")
            y = simpledialog.askinteger("Text Position", "Enter Y position:")
            if x is not None and y is not None:
                self.canvas.create_text(x, y, text=text, fill=self.draw_color,
font=("Arial", 16))
                self.actions.append(("add_text", x, y, text, self.draw_color))
                self.status_bar.config(text="Status: Text added to canvas.")

    def toggle_dark_mode(self):
        """Toggle between dark and light mode."""
        self.is_dark_mode = not self.is_dark_mode
        if self.is_dark_mode:
            self.root.config(bg="black")
            self.canvas.config(bg="gray")
            self.status_bar.config(bg="black", fg="white")
            self.toolbar.config(bg="black")
            for button in self.toolbar.winfo_children():
                button.config(bg="gray", fg="white")
            self.status_bar.config(text="Status: Dark mode activated.")
        else:
            self.root.config(bg="white")
            self.canvas.config(bg="white")
            self.status_bar.config(bg="white", fg="black")
            self.toolbar.config(bg="white")
            for button in self.toolbar.winfo_children():
                button.config(bg="white", fg="black")
            self.status_bar.config(text="Status: Light mode activated.")

    def on_button_press(self, event):
        if self.drawing_mode or self.erase_mode:
            self.last_x, self.last_y = event.x, event.y
            if self.drawing_mode:
                if self.shape == "line":
                    self.canvas.create_line(self.last_x, self.last_y,
self.last_x, self.last_y, width=self.line_width, fill=self.draw_color)
                    self.actions.append(("draw_line", self.last_x,
self.last_y, self.last_x, self.last_y, self.draw_color, self.line_width))
                elif self.shape in ["circle", "rectangle"]:
                    self.start_x, self.start_y = event.x, event.y

    def on_mouse_drag(self, event):
        if self.drawing_mode:
            x, y = event.x, event.y
            if self.shape == "line":
```

```
                self.canvas.create_line(self.last_x, self.last_y, x, y,
width=self.line_width, fill=self.draw_color, capstyle=tk.ROUND)
                self.actions[-1] = ("draw_line", self.last_x, self.last_y, x,
y, self.draw_color, self.line_width)
            elif self.shape == "freehand":
                self.canvas.create_line(self.last_x, self.last_y, x, y,
width=self.line_width, fill=self.draw_color, capstyle=tk.ROUND)
                self.actions.append(("draw_line", self.last_x, self.last_y, x,
y, self.draw_color, self.line_width))
            self.last_x, self.last_y = x, y
        elif self.erase_mode:
            self.canvas.create_line(self.last_x, self.last_y, event.x,
event.y, width=self.line_width, fill=self.erase_color, capstyle=tk.ROUND)
            self.last_x, self.last_y = event.x, event.y

    def on_button_release(self, event):
        if self.drawing_mode:
            if self.shape == "circle":
                x0, y0 = self.start_x, self.start_y
                x1, y1 = event.x, event.y
                if self.filled_shape:
                    self.canvas.create_oval(x0, y0, x1, y1,
outline=self.draw_color, width=self.line_width, fill=self.draw_color)
                    self.actions.append(("draw_circle", x0, y0, x1, y1,
self.draw_color, self.line_width, True))
                else:
                    self.canvas.create_oval(x0, y0, x1, y1,
outline=self.draw_color, width=self.line_width)
                    self.actions.append(("draw_circle", x0, y0, x1, y1,
self.draw_color, self.line_width, False))
            elif self.shape == "rectangle":
                x0, y0 = self.start_x, self.start_y
                x1, y1 = event.x, event.y
                if self.filled_shape:
                    self.canvas.create_rectangle(x0, y0, x1, y1,
outline=self.draw_color, width=self.line_width, fill=self.draw_color)
                    self.actions.append(("draw_rectangle", x0, y0, x1, y1,
self.draw_color, self.line_width, True))
                else:
                    self.canvas.create_rectangle(x0, y0, x1, y1,
outline=self.draw_color, width=self.line_width)
                    self.actions.append(("draw_rectangle", x0, y0, x1, y1,
self.draw_color, self.line_width, False))
            self.last_x, self.last_y = None, None
            self.status_bar.config(text="Status: Shape drawn.")

    def toggle_drawing_mode(self):
        self.drawing_mode = not self.drawing_mode
```

```
        self.erase_mode = False
        self.status_bar.config(text="Status: Drawing mode toggled.")

    def toggle_circle_mode(self):
        self.shape = "circle"
        self.drawing_mode = True
        self.erase_mode = False
        self.status_bar.config(text="Status: Circle mode activated.")

    def toggle_rectangle_mode(self):
        self.shape = "rectangle"
        self.drawing_mode = True
        self.erase_mode = False
        self.status_bar.config(text="Status: Rectangle mode activated.")

    def toggle_freehand_mode(self):
        self.shape = "freehand"
        self.drawing_mode = True
        self.erase_mode = False
        self.status_bar.config(text="Status: Freehand mode activated.")

    def toggle_erase_mode(self):
        self.erase_mode = not self.erase_mode
        self.drawing_mode = False
        self.status_bar.config(text="Status: Erase mode toggled.")

    def toggle_fill_shape(self):
        self.filled_shape = not self.filled_shape
        self.status_bar.config(text="Status: Fill shape toggled.")

    def choose_draw_color(self):
        color = colorchooser.askcolor()[1]
        if color:
            self.draw_color = color
            self.status_bar.config(text=f"Status: Color changed to {color}.")

    def change_stroke_width(self, event):
        self.line_width = int(self.stroke_width_var.get())
        self.status_bar.config(text=f"Status: Stroke width changed to
{self.line_width}.")

    def undo(self):
        if self.actions:
            last_action = self.actions.pop()
            self.redo_stack.append(last_action)
            self.redraw()
            self.status_bar.config(text="Status: Undo performed.")
```

```
    def redo(self):
        if self.redo_stack:
            redo_action = self.redo_stack.pop()
            self.actions.append(redo_action)
            self.redraw()
            self.status_bar.config(text="Status: Redo performed.")

    def redraw(self):
        """Redraw all actions."""
        self.canvas.delete("all")
        for action in self.actions:
            if action[0] == "draw_line":
                self.canvas.create_line(action[1], action[2], action[3],
action[4], width=action[6], fill=action[5], capstyle=tk.ROUND)
            elif action[0] == "draw_circle":
                if action[6]:  # Filled
                    self.canvas.create_oval(action[1], action[2], action[3],
action[4], outline=action[5], width=action[6], fill=action[5])
                else:
                    self.canvas.create_oval(action[1], action[2], action[3],
action[4], outline=action[5], width=action[6])
            elif action[0] == "draw_rectangle":
                if action[6]:  # Filled
                    self.canvas.create_rectangle(action[1], action[2],
action[3], action[4], outline=action[5], width=action[6], fill=action[5])
                else:
                    self.canvas.create_rectangle(action[1], action[2],
action[3], action[4], outline=action[5], width=action[6])
            elif action[0] == "add_text":
                self.canvas.create_text(action[1], action[2], text=action[3],
fill=action[4], font=("Arial", 16))

if __name__ == "__main__":
    root = tk.Tk()
    editor = EnhancedEditor(root)
    root.mainloop()
```

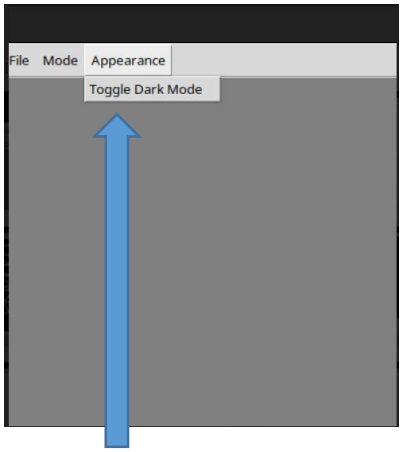
OUTPUT AND SCREENSHOTS POINTING OUT TO ITS FEATURES



The above image shows how my created image editor looks like. It has a bar at the top providing File (open new file, save file, open previously saved file), Mode (Image mode and draw mode) and Appearance (dark theme and light theme). At the bottom there are features like resize, crop, draw, colour, circle, rectangle, erase, undo, redo, freehand, text, clear, fill shape, stroke width and some standard colours.

One of the most special or highlighting feature is the status: where the status of our image editor will be denoted from.

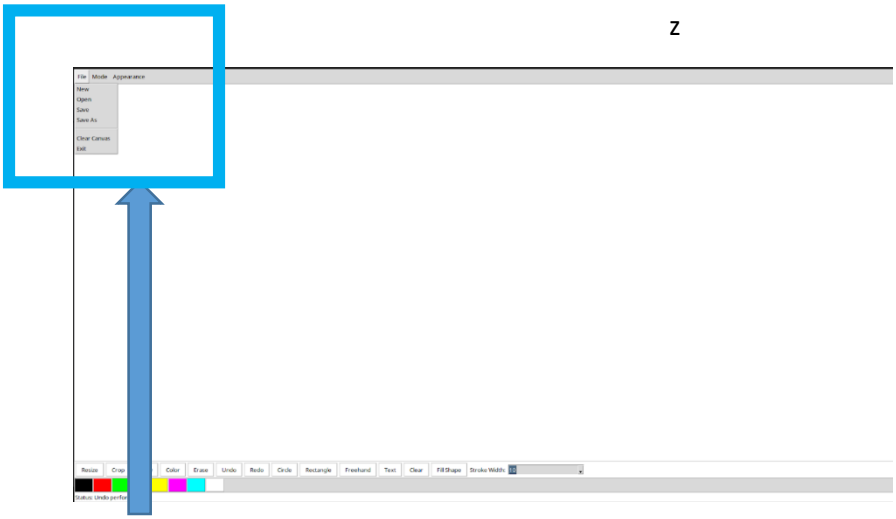
Currently our image editor is ready to use and is operated in light theme.



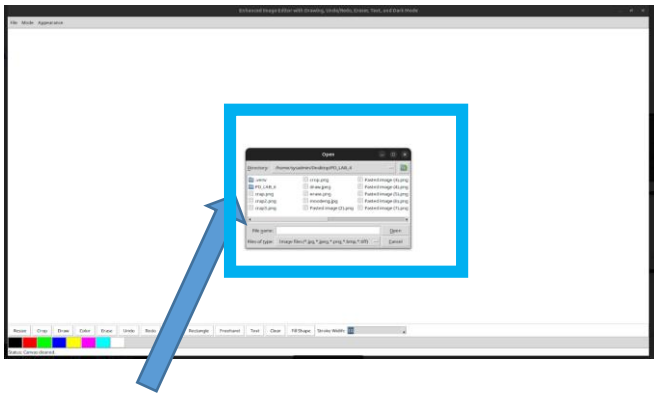
Option for dark mode

In dark mode

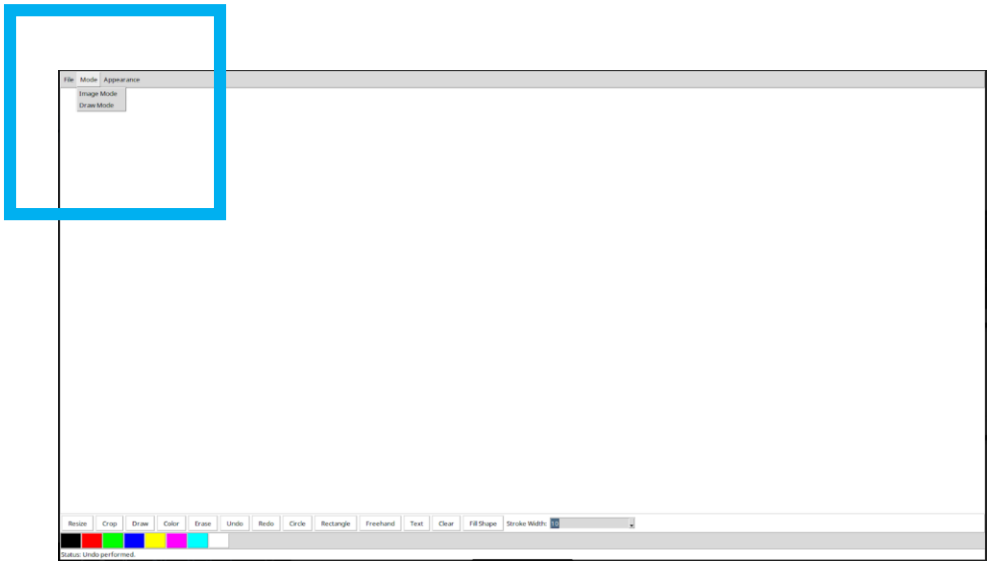
z



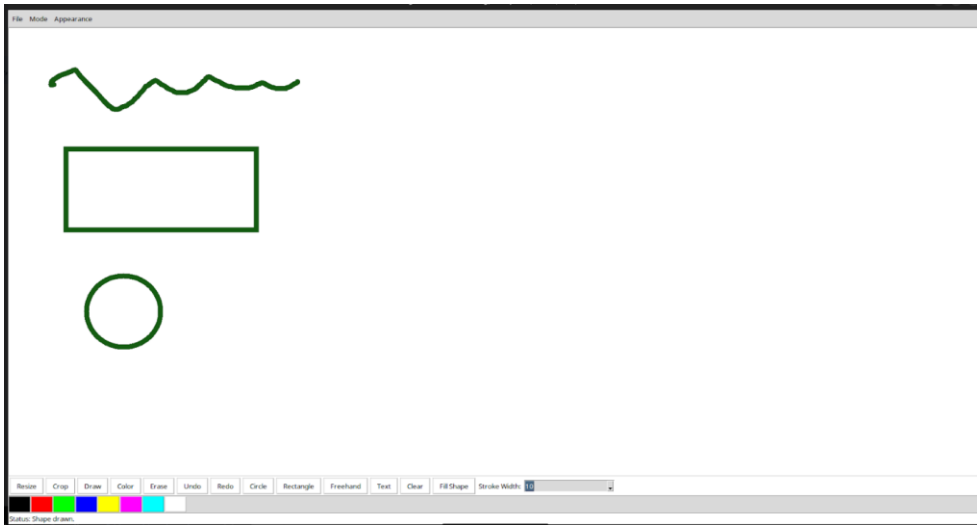
Shows features like open new file, save file, open file (open previously saved file)



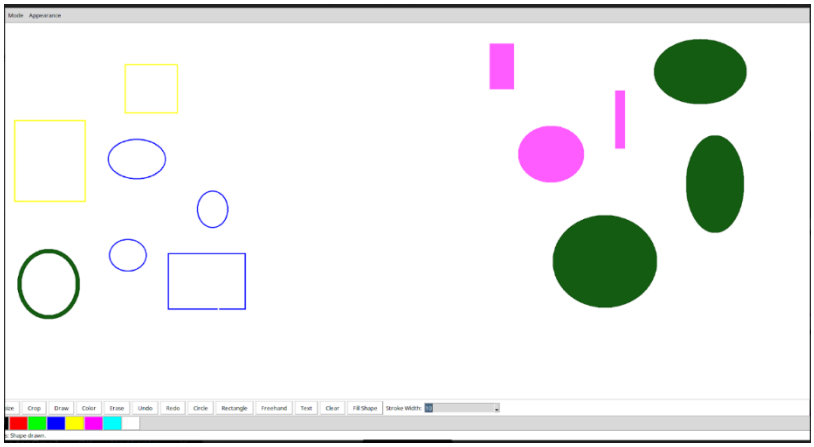
Features to open a saved file



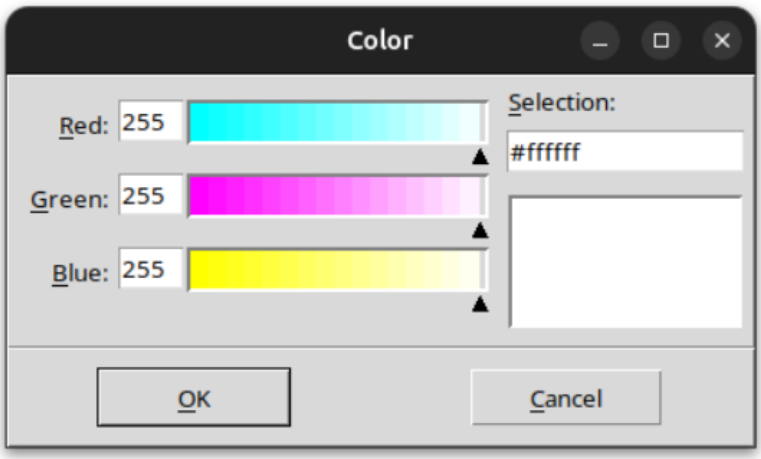
To enable image mode or drawing mode.



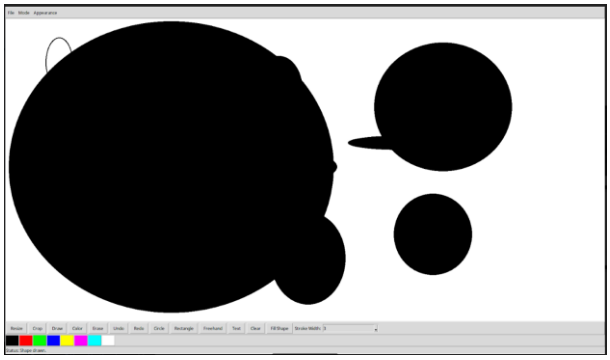
This shows freehand, rectangle and circle.



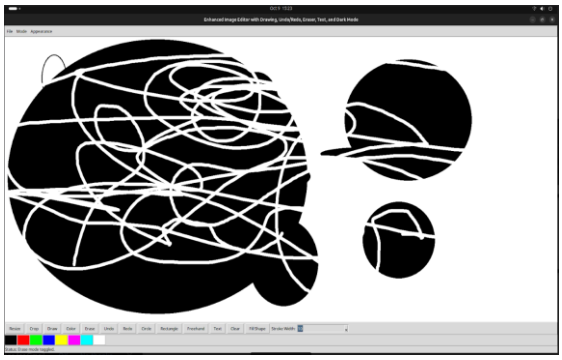
The left side is draw shape, right side is draw shape and select region



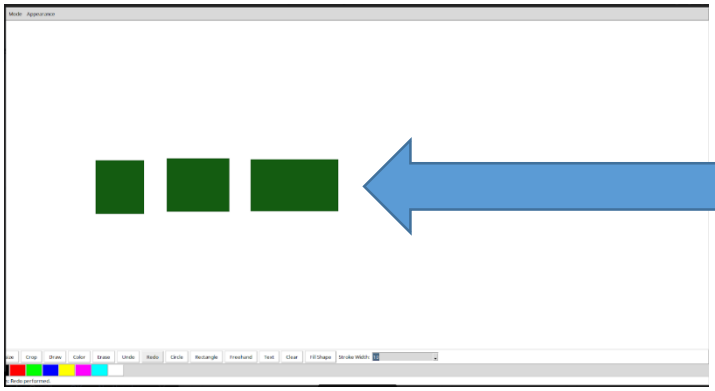
Option to choose customised colour.



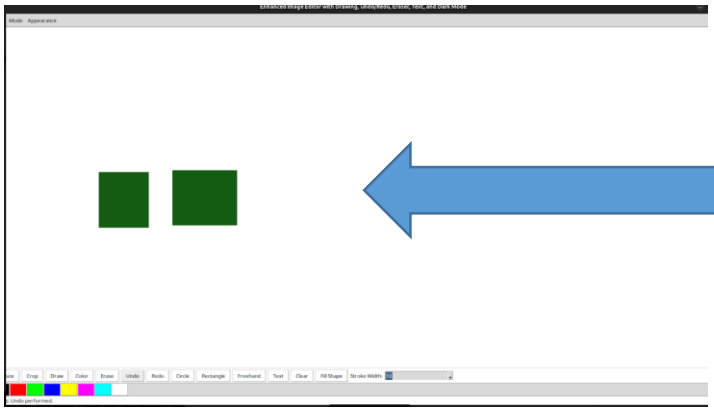
Drawing before eraser



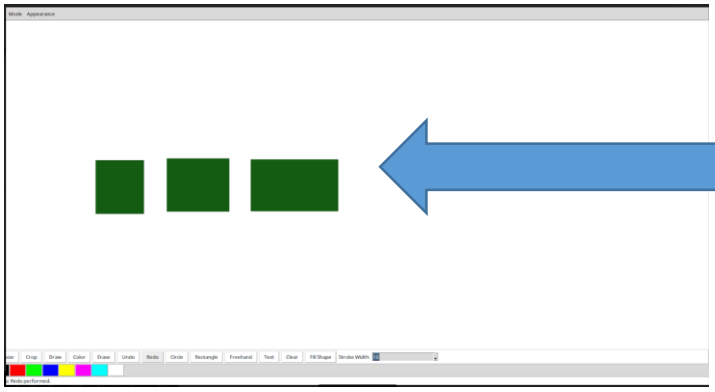
After eraser



DRAW

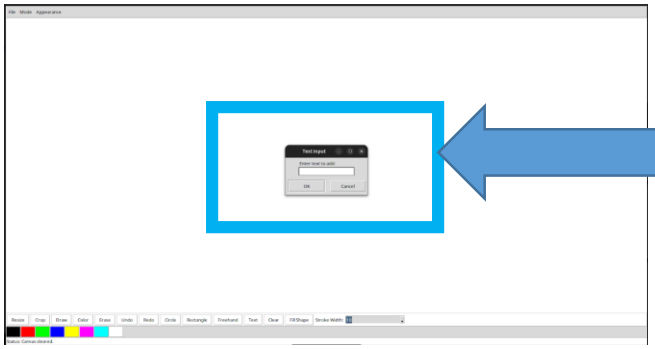


UNDO

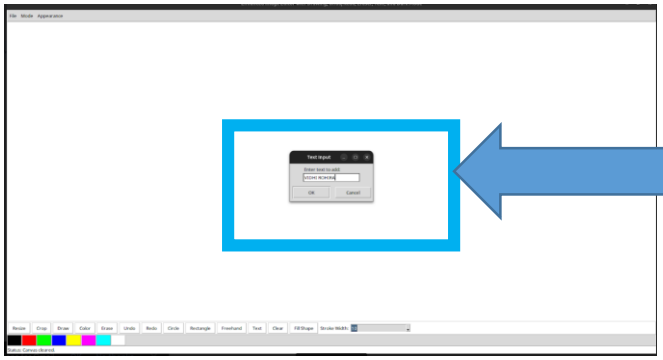


REDO

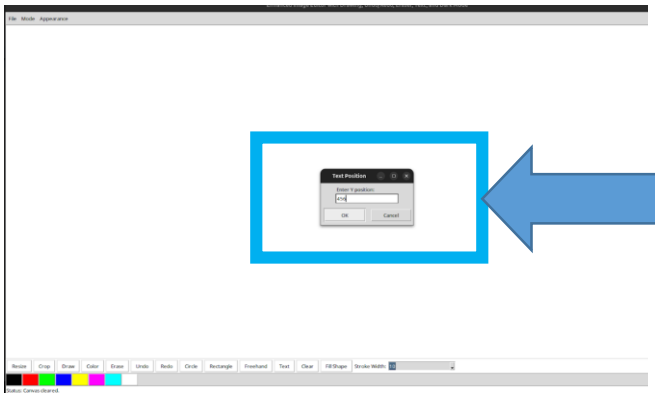
TEXT FEATURES



ADD TEXT BOX



ADD TEXT



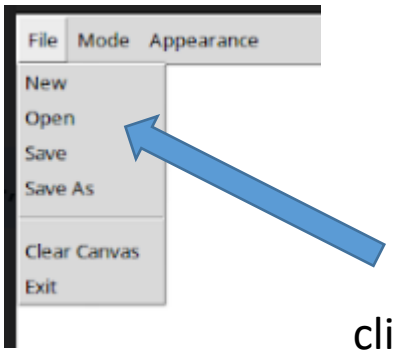
POSITIONING THE TEXT

OUTPUT

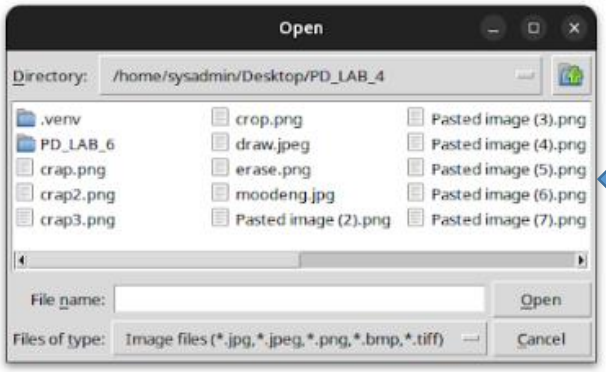
IMAGE FEATURES

1] First switch to image mode

2] Upload image from file option



click on open



select an image

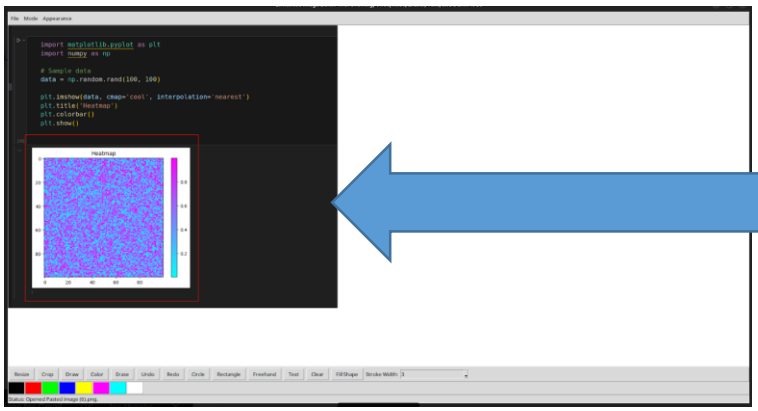
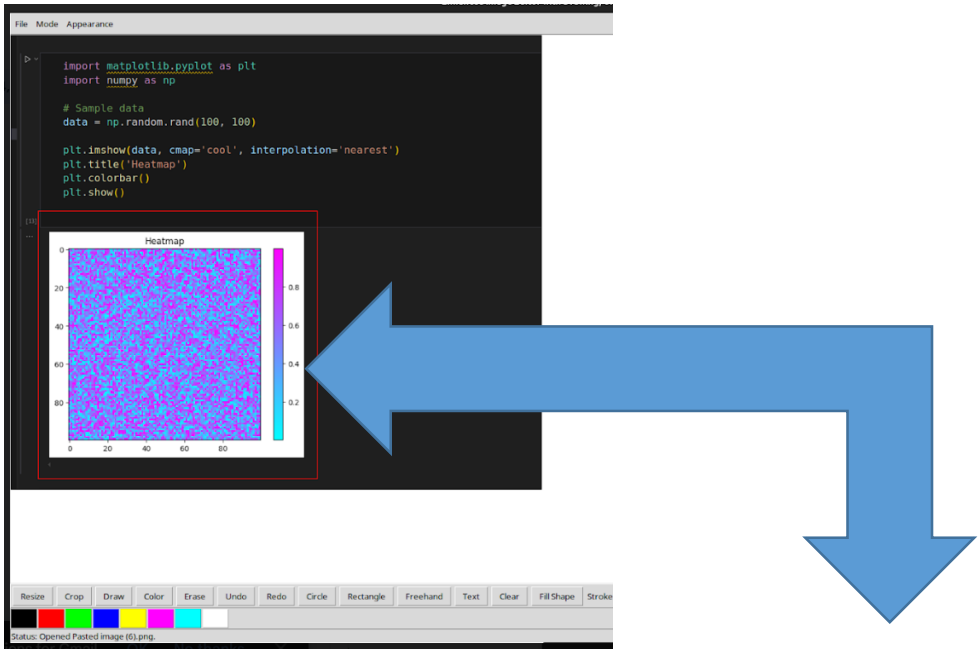
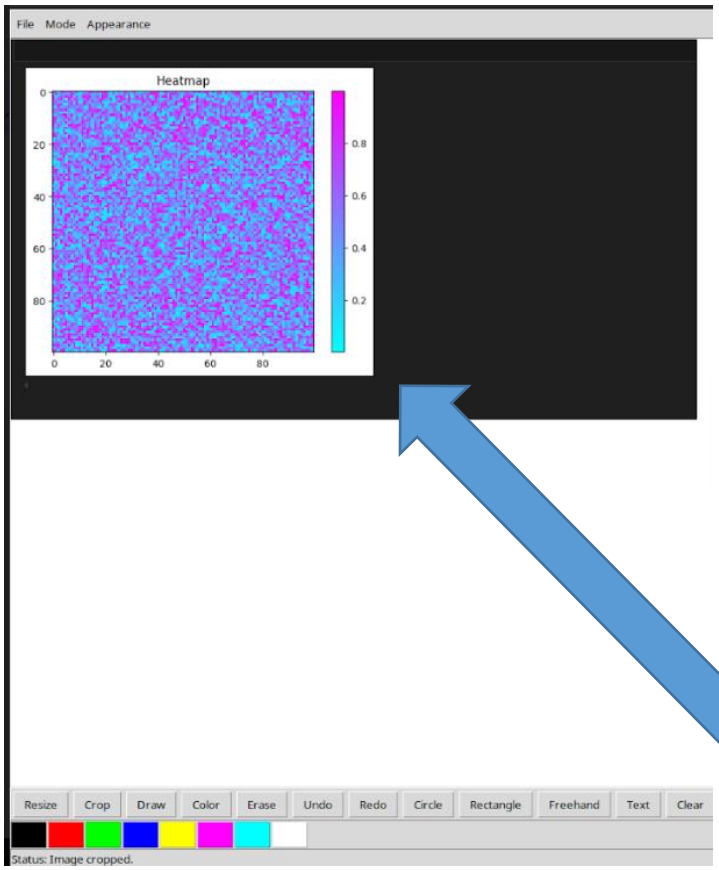


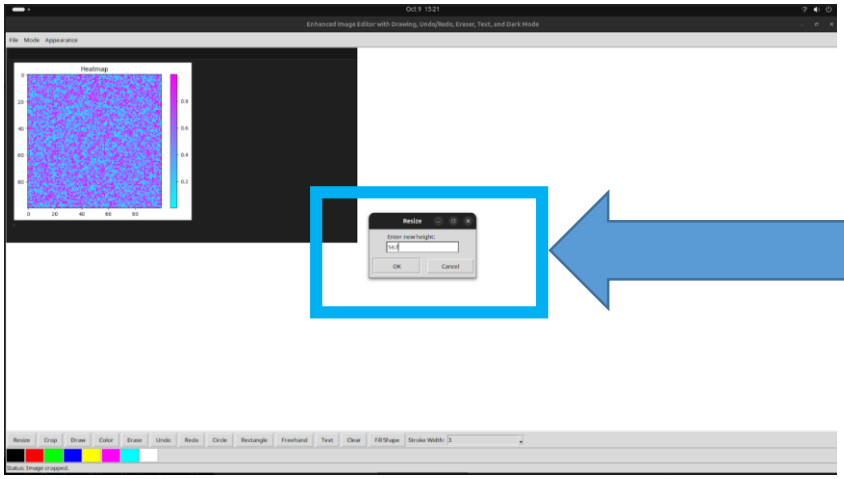
image chosen



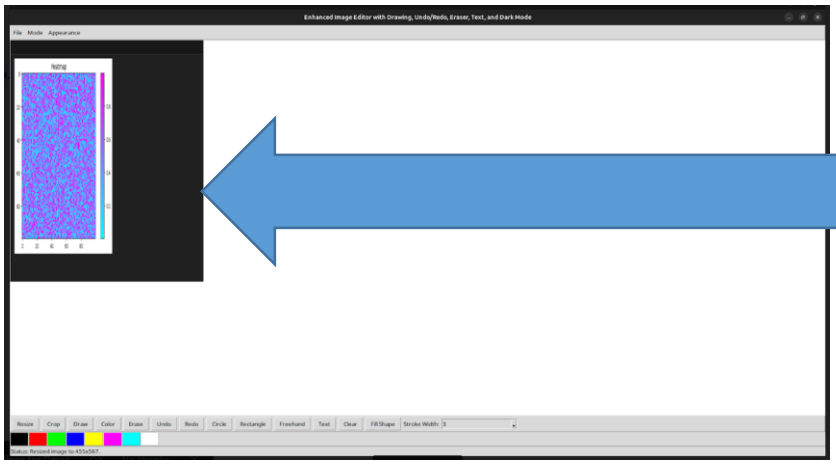
Red boundary is the region you wish to crop out the image to



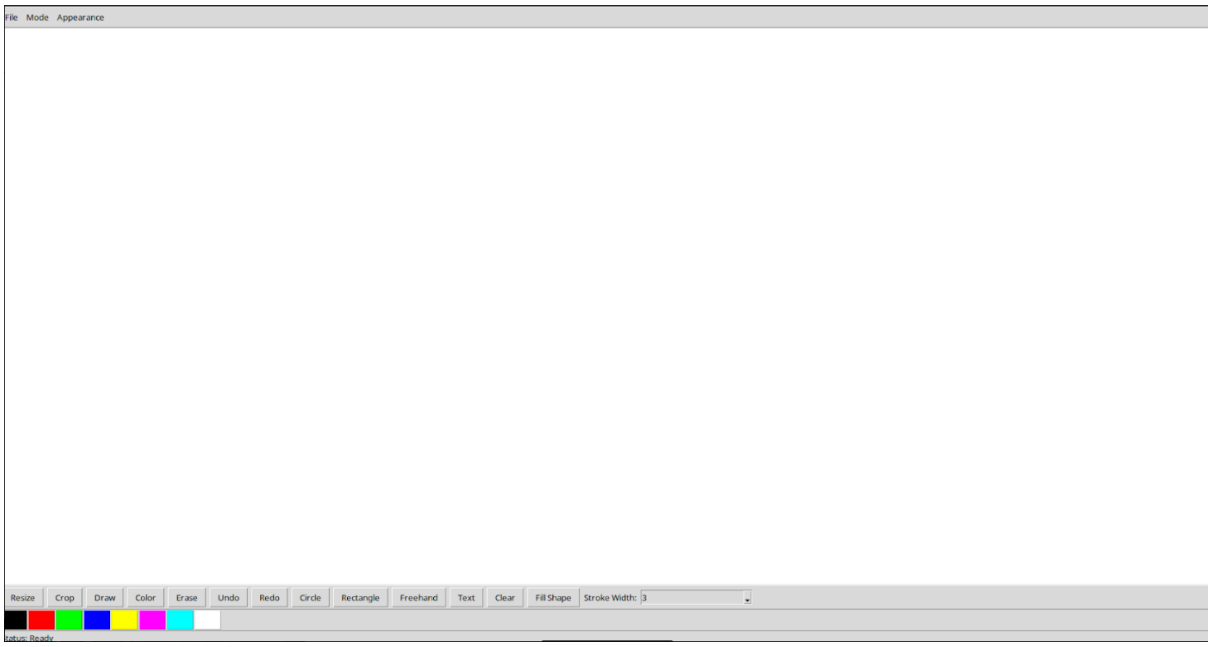
cropped image



option to resize



final resized image



Final after clear screen.

CONCLUSION:- Hence, in this lab session, we have learned about **Tkinter**, **OpenCV**, and how to create a basic **image editor**. We explored the implementation of an image editor by integrating OpenCV for image processing and Tkinter for building the graphical user interface. Additionally, we created an image editor and demonstrated its features, such as cropping and resizing images, drawing with freehand or predefined shapes, adding colors, inserting text, and filling regions. The code for these features has been provided and output has been explained.