```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 7 20:47:02 2023

@author: Divya
"""

import tkinter as tk
from tkinter import filedialog, colorchooser
from PIL import Image, ImageOps, ImageTk, ImageFilter
from tkinter import ttk
from PIL import ImageGrab



eraser_size = 5 # Default eraser size
drawing_mode = "draw"

# Add these variables at the beginning of your code
text_color = "#000000" # Default text color
text_size = 12 # Default text size
text_objects = [] # List to store text objects
drag_data = {"x": 0, "y": 0, "item": None} # Dictionary to track dragging data

import tkinter.simpledialog as sd
import tkinter.colorchooser as cc




def add_image():
    global file_path, background_image, image_added
    file_path = filedialog.askopenfilename(initialdir="D:/codefirst.io/Tkinter Image Editor/Pictures")
    if file_path:
        image = Image.open(file_path)
        width, height = int(image.width / 2), int(image.height / 2)
        image = image.resize((width, height), Image.LANCZOS)
        background_image = ImageTk.PhotoImage(image)

        canvas.delete("all") # Clear existing content on the canvas
        # Center the image on the canvas
        canvas_width = canvas.winfo_width()
        canvas_height = canvas.winfo_height()
        x_center = (canvas_width - width) / 2
        y_center = (canvas_height - height) / 2

        canvas.create_image(x_center, y_center, image=background_image, anchor="nw")

        image_added = True # Set the flag to indicate an image has been added


def rotate_image(angle):
    global total_rotation
    global file_path
    if file_path:
        total_rotation += angle
        total_rotation %= 360 # Ensure total rotation stays within 360 degrees

        image = Image.open(file_path)
        w, h = int(image.height/2), int(image.width/2)
        image = image.resize((w, h), Image.ANTIALIAS)
        image = image.rotate(total_rotation, resample=Image.ANTIALIAS, expand=True) # Use
ANTIALIAS for better quality

        # Calculate the new center position after rotation
        new_w, new_h = image.size
        x_center = (w - new_w) / 2
        y_center = (h - new_h) / 2

        canvas.config(width=w, height=h)
        canvas.delete("all")

        image = ImageTk.PhotoImage(image)
        canvas.image = image
        canvas.create_image(x_center, y_center, image=image, anchor="nw")
        file_paths.append(file_path)



def erase(event):
    global drawing_operations, image_added, drawing_mode
    if not image_added:
        return
    x, y = event.x, event.y
    x1, y1 = (x - eraser_size), (y - eraser_size)
    x2, y2 = (x + eraser_size), (y + eraser_size)
    canvas.create_oval(x1, y1, x2, y2, fill="#FFB6C1", outline='', width=eraser_size)
    drawing_operations.append((x1, y1, x2, y2, "#FFB6C1", eraser_size))

    drawing_mode = "erase" # Set drawing mode to erase




def draw(event):
    global drawing_mode
    x1, y1 = (event.x - pen_size), (event.y - pen_size)
    x2, y2 = (event.x + pen_size), (event.y + pen_size)
    canvas.create_oval(x1, y1, x2, y2, fill=pen_color, outline='')
    canvas.update()
    drawing_operations.append((x1, y1, x2, y2, pen_color, pen_size))
    drawing_mode = "draw" # Set drawing mode to draw



def switch_to_draw():
    global drawing_mode
    if drawing_mode == "erase":
        canvas.bind("<B1-Motion>", draw)



def switch_to_erase():
    global drawing_mode
    if drawing_mode == "draw":
        canvas.bind("<B1-Motion>", erase)


def select_eraser_size(size):
    global eraser_size
    eraser_size = size
```

```
def undo():
    global drawing_operations, undone_operations
    if drawing_operations:
        undone_operations.append(drawing_operations.pop())
        canvas.delete("all")
        if image_added:
            canvas.create_image(0, 0, image=background_image, anchor="nw")
        for operation in drawing_operations:
            x1, y1, x2, y2, color, size = operation
            canvas.create_oval(x1, y1, x2, y2, fill=color, outline='', width=size)




def change_color():
    global pen_color
    pen_color = colorchooser.askcolor(title="Select Pen Color")[1]




def change_size(size):
    global pen_size
    pen_size = size




""
def draw(event):
    x1, y1 = (event.x - pen_size), (event.y - pen_size)
    x2, y2 = (event.x + pen_size), (event.y + pen_size)
    canvas.create_oval(x1, y1, x2, y2, fill=pen_color, outline='')
    canvas.update()
    drawing_operations.append((x1, y1, x2, y2, pen_color, pen_size))
    drawing_operations.append((x1, y1, x2, y2, pen_color, pen_size))
```

```
'''
def rotate_image(angle):
    global total_rotation
    global file_path
    if file_path:
        total_rotation += angle
        total_rotation %= 360 # Ensure total rotation stays within 360 degrees

        image = Image.open(file_path)
        w=int(image.height/2)
        h=int(image.width/2)

        image=image.resize((w,h),Image.LANCZOS)
        image = image.rotate(total_rotation, resample=Image.BICUBIC, expand=True) # Use
BICUBIC for better quality

        w=int(image.height)
        h=int(image.width)


        #w, h = image.size
        canvas.config(width=w, height=h)
        image=image.resize((w,h),Image.LANCZOS)

        canvas.delete("all")


        image = ImageTk.PhotoImage(image)
        canvas.image = image
        canvas.create_image(0, 0, image=image, anchor="nw")
        file_paths.append(file_path)





def redo():
    if undone_operations:
        operation = undone_operations.pop()
        drawing_operations.append(operation)
        x1, y1, x2, y2, color, size = operation
        canvas.create_oval(x1, y1, x2, y2, fill=color, outline='', width=size)
```

```
def save_drawn_image():
    global file_path
    if file_path:
        x = root.winfo_rootx() + canvas.winfo_x()
        y = root.winfo_rooty() + canvas.winfo_y()
        x1 = x + canvas.winfo_width()
        y1 = y + canvas.winfo_height()

        # Capture the content of the canvas
        captured_image = ImageGrab.grab(bbox=(x, y, x1, y1))

        # Ask user for a file path to save the image
        output_path = filedialog.asksaveasfilename(
            initialdir="/", title="Save As", defaultextension=".png", filetypes=[("PNG files", "*.png")])

        if output_path:
            captured_image.save(output_path)
            file_path = output_path # Update file_path with the new saved image path

            # Display the newly saved image on the canvas
            canvas.delete("all")
            background_image = ImageTk.PhotoImage(file=output_path)
            canvas.create_image(0, 0, image=background_image, anchor="nw")




def apply_filter(filter):
    image = Image.open(file_path)
    width, height = int(image.width / 2), int(image.height / 2)
    image = image.resize((width, height), Image.LANCZOS)

    if filter == "Black and White":
        image = ImageOps.grayscale(image)
    elif filter == "Blur":
        image = image.filter(ImageFilter.BLUR)
    elif filter == "Sharpen":
        image = image.filter(ImageFilter.SHARPEN)
    elif filter == "Smooth":
        image = image.filter(ImageFilter.SMOOTH)
    elif filter == "Emboss":
        image = image.filter(ImageFilter.EMBOSS)
    elif filter == "Edge Enhance":
        image = image.filter(ImageFilter.EDGE_ENHANCE)
    elif filter == "Contour":
        image = image.filter(ImageFilter.CONTOUR)
    elif filter == "Detail":
        image = image.filter(ImageFilter.DETAIL)
    elif filter == "Pencil Sketch":
        image = image.convert("L")
        image = image.filter(ImageFilter.FIND_EDGES)
    elif filter == "Charcoal":
        image = image.filter(ImageFilter.CONTOUR)

    elif filter == "Invert Colors":
        image = ImageOps.invert(image.convert('RGB'))

    edited_window = tk.Toplevel(root)
    edited_window.title("Edited Image")
    edited_window.config(bg="#FFB6C1")

    edited_canvas = tk.Canvas(edited_window, width=image.width, height=image.height,
bg="white")
    edited_canvas.pack()

    edited_image = ImageTk.PhotoImage(image)
    edited_canvas.image = edited_image
    edited_canvas.create_image(0, 0, image=edited_image, anchor="nw")

    edited_canvas.create_rectangle(5, 5, image.width - 5, image.height - 5, outline="#FF69B4",
width=10)

    save_button = tk.Button(edited_window, text="Save Image", command=lambda:
save_image(image), bg="#FF91A4")
    save_button.pack(pady=10)
def save_image(image):
    output_path = filedialog.asksaveasfilename(
        initialdir="/", title="Save As", defaultextension=".png", filetypes=[("PNG files",
    "*.png")]) if output_path:
        image.save(output_path)



def clear_canvas():
    canvas.delete("all")
    drawing_operations.clear()
    undone_operations.clear()




def rotate_90_degrees():
    rotate_image(90)

def rotate_minus_90_degrees():
    rotate_image(-90)




root = tk.Tk()
root.geometry("1000x600")
root.title("Image Editing Tool")
root.config(bg="#FFB6C1") # Light Pink theme

pen_color = "#FF69B4" # Pink color
pen_size = 5
file_path = ""
file_paths = []

left_frame = tk.Frame(root, width=200, height=600, bg="#FFB6C1")
left_frame.pack(side="left", fill="y")
canvas = tk.Canvas(root, width=1000, height=1000, bg="#FFB6C1")
canvas.pack()





# Add Image Button
image_button = tk.Button(left_frame, text="Add Image", command=add_image,
bg="#FFB6C1",activebackground="#FB607F")
image_button.pack(pady=15)

#colour pallet
```

```
color_button = tk.Button(
   left_frame, text="Change Pen Color", command=change_color,
bg="Pink",activebackground="#FB607F")
color_button.pack(pady=10,padx=20)




#pen size
pen_size_frame = tk.Frame(left_frame, bg="#FF91A4")
pen_size_frame.pack(pady=5)

pen_size_1 = tk.Radiobutton(
   pen_size_frame, text="Small", value=3, command=lambda: change_size(3), bg="#FB607F")
pen_size_1.pack(side="left")

pen_size_2 = tk.Radiobutton(
   pen_size_frame, text="Medium", value=5, command=lambda: change_size(5),
bg="#FB607F")
pen_size_2.pack(side="left")
pen_size_2.select()

pen_size_3 = tk.Radiobutton(
   pen_size_frame, text="Large", value=7, command=lambda: change_size(7), bg="#FB607F")
pen_size_3.pack(side="left")


#FF91A4
#save button
save_drawn_button = tk.Button(left_frame, text="Save Drawn Image",
command=save_drawn_image, bg="#FFB6C1",activebackground="#FB607F")
save_drawn_button.pack(pady=5)

#undo button
undo_button = tk.Button(left_frame, text="Undo", command=undo,
bg="#FFB6C1",activebackground="#FB607F")
undo_button.pack(pady=5)

# Add Redo Button
redo_button = tk.Button(left_frame, text="Redo", command=redo,
bg="#FFB6C1",activebackground="#FB607F")
```

```
redo_button.pack(pady=5)


#clear button
clear_button = tk.Button(left_frame, text="Clear", command=clear_canvas,
bg="#FFB6C1",activebackground="#FB607F")
clear_button.pack(pady=5)


drawing_operations = [] # List to store drawing operations
undone_operations = []

total_rotation = 0 # Initialize total rotation angle



#rotate buttons
rotate_button = tk.Button(left_frame, text="Rotate 90°", command=rotate_90_degrees,
bg="#FFB6C1",activebackground="#FB607F")
rotate_button.pack(pady=5)

rotate_minus_button = tk.Button(left_frame, text="Rotate -90°",
command=rotate_minus_90_degrees, bg="#FFB6C1",activebackground="#FB607F")
rotate_minus_button.pack(pady=5)



# Apply Filter
filter_label = tk.Label(left_frame, text="Select Filter", bg="#FB607F", fg="BLACK")
filter_label.pack()
filter_combobox = ttk.Combobox(left_frame, values=["Black and White", "Blur", "Emboss",
"Sharpen", "Smooth", "Edge Enhance", "Contour", "Detail", "Enhance", "Charcoal","Pencil
Sketch", "Invert Colors"])
filter_combobox.pack()

filter_combobox.bind("<<ComboboxSelected>>", lambda event:
apply_filter(filter_combobox.get()))


# Add eraser size selection
eraser_size_frame = tk.Frame(left_frame, bg="white")
eraser_size_frame.pack(pady=5)
```

```
eraser_size_1 = tk.Radiobutton(
    eraser_size_frame, text="Small", value=3, command=lambda: select_eraser_size(3),
bg="#FB607F")
eraser_size_1.pack(side="left")

eraser_size_2 = tk.Radiobutton(
    eraser_size_frame, text="Medium", value=5, command=lambda: select_eraser_size(5),
bg="#FB607F")
eraser_size_2.pack(side="left")

eraser_size_3 = tk.Radiobutton(
    eraser_size_frame, text="Large", value=7, command=lambda: select_eraser_size(7),
bg="#FB607F")
eraser_size_3.pack(side="left")

# Add a "Draw" button to switch back to drawing mode
erase_button = tk.Button(left_frame, text="Erase", command=switch_to_erase,
bg="#FFB6C1",activebackground="#FB607F")
erase_button.pack(pady=5)

draw_button = tk.Button(left_frame, text="Draw", command=switch_to_draw,
bg="#FFB6C1",activebackground="#FB607F")
draw_button.pack(pady=5)


import tkinter.font as tkFont



def add_text():
    global text_objects, drag_data
    text = sd.askstring("Enter Text", "Enter the text:")
    if text is None or text == "":
        return

    font = tkFont.Font(family="Arial", size=12) # Default font
    font_name = sd.askstring("Enter Font", "Enter the font name:")
    if font_name is not None and font_name != "":
        font.configure(family=font_name)

    text_color = cc.askcolor(title="Select Text Color")[1]
    if text_color is None:
        return

    font_size = sd.askinteger("Enter Font Size", "Enter font size:")
    if font_size is None:
        return
    font.configure(size=font_size)


    x, y = 50, 50 # Default position
    text_object = canvas.create_text(x, y, text=text, font=font, fill=text_color, anchor="nw")

    text_objects.append(text_object)

    # Set up event handlers for dragging the text
    canvas.tag_bind(text_object, "<ButtonPress-1>", on_text_press)
    canvas.tag_bind(text_object, "<B1-Motion>", on_text_drag)
    canvas.tag_bind(text_object, "<ButtonRelease-1>", on_text_release)
    canvas.tag_bind(text_object, "<ButtonPress-1>", on_text_press)
    canvas.tag_bind(text_object, "<B1-Motion>", on_text_drag)
    canvas.tag_bind(text_object, "<ButtonRelease-1>", on_text_release)

def on_text_press(event):
    global drag_data
    drag_data["item"] = canvas.find_closest(event.x, event.y)[0]
    drag_data["x"] = event.x
    drag_data["y"] = event.y
    canvas.unbind("<B1-Motion>", draw)
    canvas.unbind("<B1-Motion>", erase)

    drag_data["item"] = canvas.find_closest(event.x, event.y)[0]
    drag_data["x"] = event.x
    drag_data["y"] = event.y
def on_text_drag(event):
    global drag_data
    canvas.move(drag_data["item"], event.x - drag_data["x"], event.y - drag_data["y"])
    drag_data["x"] = event.x
    drag_data["y"] = event.y
    canvas.move(drag_data["item"], event.x - drag_data["x"], event.y - drag_data["y"])
    drag_data["x"] = event.x
    drag_data["y"] = event.y
```

```
def on_text_release(event):
    global drag_data
    drag_data["item"] = None
    canvas.bind("<B1-Motion>", draw)
    canvas.bind("<B1-Motion>", erase)

    drag_data["item"] = None




# Add Text Button
text_button = tk.Button(left_frame, text="Add Text", command=add_text, bg="#FFB6C1",
activebackground="#FB607F")
text_button.pack(pady=15)


canvas.bind("<B1-Motion>", draw)


root.mainloop()
```