```python
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import os
import math
import cv2
import pygame

# Global variables to store choices
photos = []
top_3_items = []
logo_styles = []
frame_shapes = []
frame_bgs = []
arrangement_styles = []
def play_sound(sound_file):
    pygame.mixer.init()
    pygame.mixer.music.load(sound_file)
    pygame.mixer.music.play()

def update_frame():
    ret, frame = cap.read()
    if ret:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(frame)
        img.thumbnail((video_width, video_height), Image.LANCZOS)
        imgtk = ImageTk.PhotoImage(image=img)
        lbl.imgtk = imgtk
        lbl.config(image=imgtk)
    lbl.after(10, update_frame)

def open_string_input_window():
    play_sound('button_click.wav')
    string_input_window = tk.Toplevel(root)
    string_input_window.title("String Input")
    string_input_window.geometry('800x600')

    # Set background image
    bg_image_path = "2nd bg.jpg"
    if os.path.isfile(bg_image_path):
        bg_image = Image.open(bg_image_path)
        bg_photo = ImageTk.PhotoImage(bg_image)
        canvas = tk.Canvas(string_input_window,
width=string_input_window.winfo_width(),
height=string_input_window.winfo_height())
        canvas.pack(fill=tk.BOTH, expand=True)
        canvas.create_image(0, 0, image=bg_photo, anchor=tk.NW)
        canvas.image = bg_photo  # Keep a reference to avoid garbage collection

        def resize_background(event):
            canvas_width = string_input_window.winfo_width()
            canvas_height = string_input_window.winfo_height()
            bg_resized = bg_image.resize((canvas_width, canvas_height),
Image.LANCZOS)
            bg_photo_resized = ImageTk.PhotoImage(bg_resized)
            canvas.create_image(0, 0, image=bg_photo_resized, anchor=tk.NW)
            canvas.image = bg_photo_resized  # Keep a reference to avoid garbage
collection

        string_input_window.bind("<Configure>", resize_background)

    # Input elements on canvas
    string_label = tk.Label(canvas, text="ENTER YOUR COMPANY NAME:",
font=("Arial", 24), bg='black', fg='white')
    string_entry = tk.Entry(canvas, font=("Arial", 24), width=20)
    analyze_button = tk.Button(canvas, text="Analyze", bg="#FF3399",
fg="#FFFFFF", font=("Arial", 24),
                               command=lambda: analyze_string(string_entry.get(),
string_input_window))

    def place_elements():
        canvas_width = canvas.winfo_width()
        canvas_height = canvas.winfo_height()
        label_x = canvas_width // 2
        label_y = canvas_height // 2 - 70
        entry_y = canvas_height // 2
        button_y = canvas_height // 2 + 70

        canvas.create_window(label_x, label_y, window=string_label)
        canvas.create_window(label_x, entry_y, window=string_entry)
        canvas.create_window(label_x, button_y, window=analyze_button)


    canvas.bind("<Configure>", lambda event: place_elements())

def open_second_window():
    play_sound('button_click.wav')
    second_window = tk.Toplevel(root)
    second_window.title("Design Preferences")
    second_window.geometry('800x600')
    second_window.state('zoomed')  # Open window in maximized state

    # Set background image
    bg_image_path = "stringinputwindow.jpg"
    if os.path.isfile(bg_image_path):
        bg_image = Image.open(bg_image_path)
        bg_photo = ImageTk.PhotoImage(bg_image)

        canvas = tk.Canvas(second_window, width=800, height=600)
        canvas.pack(fill=tk.BOTH, expand=True)
        canvas.create_image(0, 0, image=bg_photo, anchor=tk.NW)
        canvas.image = bg_photo  # Keep a reference to avoid garbage
collection

        def resize_background(event):
            canvas_width = event.width
            canvas_height = event.height
            bg_resized = bg_image.resize((canvas_width, canvas_height),
Image.LANCZOS)
            bg_photo_resized = ImageTk.PhotoImage(bg_resized)
            canvas.create_image(0, 0, image=bg_photo_resized, anchor=tk.NW)
            canvas.image = bg_photo_resized  # Keep a reference to avoid garbage
collection

        canvas.bind("<Configure>", resize_background)

    # Option 1: Logo Style
    style_label = tk.Label(second_window, text="Choose the style(s) of the
logo:", bg='#fbebda', fg='black', font=("Arial", 16))
    style_label.place(relx=0.5, rely=0.05, anchor='n')

    style_frame = tk.Frame(second_window, bg='#fbebda')
    style_frame.place(relx=0.5, rely=0.1, anchor='n')

    global logo_style_vars
    logo_style_vars = []

    styles = ["_alt", "_alt1", "_alt2", "_alt3"]
    for i, style in enumerate(styles):
        img_path = f"{style}.png"
        if os.path.isfile(img_path):
            image = Image.open(img_path)
```

```python
            image = image.resize((100, 100), Image.LANCZOS)
            photo = ImageTk.PhotoImage(image)
            photos.append(photo)
            var = tk.BooleanVar()  #tracking selection
            logo_style_vars.append(var)
            cb = tk.Checkbutton(style_frame, image=photo, variable=var,
bg='black', indicatoron=0)
            cb.grid(row=0, column=i, padx=10, pady=10)  # Adjust the padding
            cb.var = var
            cb.photo = photo
            cb.config(command=lambda cb=cb, color='black', option='logo_style',
vars=logo_style_vars: update_checkbutton_color(cb, color, option, vars))
        else:
            print(f"Image not found for {style}")

    # Option 2: Frame Shape
    shape_label = tk.Label(second_window, text="Choose the frame shape(s):",
bg='#fbebda', fg='black', font=("Arial", 16))
    shape_label.place(relx=0.5, rely=0.3, anchor='n')

    shape_frame = tk.Frame(second_window, bg='#fbebda')
    shape_frame.place(relx=0.5, rely=0.35, anchor='n')

    global frame_shape_vars
    frame_shape_vars = []

    shapes = ["none", "pentagonal", "rectangular", "circular"]
    for shape in shapes:
        var = tk.BooleanVar()
        frame_shape_vars.append(var)
        cb = tk.Checkbutton(shape_frame, text=shape.capitalize(), variable=var,
bg='black', fg='white', font=("Arial", 16))
        cb.pack(side='left', padx=5, pady=5)  # Adjust the padding
        cb.var = var
        cb.config(command=lambda cb=cb, color='black', option='frame_shape',
vars=frame_shape_vars: update_checkbutton_color(cb, color, option, vars))

    # Option 3: Background Color
    background_options = ["white", "black", "red", "green", "blue"]

    background_label = tk.Label(second_window, text="Choose the background
color(s):", fg='black', bg='#fbebda', font=("Arial", 16))
    background_label.place(relx=0.5, rely=0.45, anchor='n')
    background_frame = tk.Frame(second_window, bg='#fbebda')
    background_frame.place(relx=0.5, rely=0.5, anchor='n')

    global frame_bg_vars
    frame_bg_vars = []

    for color in background_options:
        var = tk.BooleanVar()
        frame_bg_vars.append(var)
        cb = tk.Checkbutton(background_frame, text='', variable=var, bg=color,
fg='white', font=("Arial", 16))
        cb.pack(side='left', padx=10, pady=5)  # Adjust the padding
        cb.var = var
        cb.config(command=lambda cb=cb, color='black', option='background_color',
vars=frame_bg_vars: update_checkbutton_color(cb, color, option, vars))
    # Option 4: Design Question (Arrangement Style)
    design_label = tk.Label(second_window, text="Choose the arrangement
style(s):", bg='#fbebda', fg='black', font=("Arial", 16))
    design_label.place(relx=0.5, rely=0.6, anchor='n')

    global arrangement_vars
    arrangement_vars = []

    arrangement_frame = tk.Frame(second_window, bg='#fbebda')
    arrangement_frame.place(relx=0.5, rely=0.65, anchor='n')

    arrangement_options = ["horizontal", "vertical", "triangular", "L shape"]
    for arrangement in arrangement_options:
        var = tk.BooleanVar()
        arrangement_vars.append(var)
        cb = tk.Checkbutton(arrangement_frame, text=arrangement.capitalize(),
variable=var, bg='black', fg='white', font=("Arial", 16))
        cb.pack(side='left', padx=10, pady=10)  # Adjust the padding
        cb.var = var
        cb.config(command=lambda cb=cb, color='black',
option='arrangement_style', vars=arrangement_vars: update_checkbutton_color(cb,
color, option, vars))

    confirm_button = tk.Button(second_window, text="Confirm Choices",
bg="#FF3399", fg="#FFFFFF", font=("Arial", 16), command=lambda:
confirm_choices(second_window))
    confirm_button.place(relx=0.5, rely=0.8, anchor='s')

def update_checkbutton_color(cb, color=None, option=None, vars=None):
    play_sound('button_click.wav')
```

```python
    for var in vars:
        if var != cb.var:
            var.set(False)
            for widget in cb.master.winfo_children():
                if isinstance(widget, tk.Checkbutton) and widget.var == var:
                    widget.config(bg=color if color else 'black', fg='white')
    if cb.var.get():
        cb.config(bg='yellow', fg='black')
    else:
        cb.config(bg=color if color else 'black', fg='white')

def confirm_choices(second_window):
    play_sound('button_click.wav')
    global logo_styles, frame_shapes, frame_bgs, arrangement_styles
    logo_styles = [style for var, style in zip(logo_style_vars, ["_alt", "_alt1",
"_alt2", "_alt3"]) if var.get()]
    frame_shapes = [shape for var, shape in zip(frame_shape_vars, ["none",
"pentagonal", "rectangular", "circular"]) if var.get()]
    frame_bgs = [bg for var, bg in zip(frame_bg_vars, ["white", "black", "red",
"green", "blue"]) if var.get()]
    arrangement_styles = [arrangement for var, arrangement in
zip(arrangement_vars, ["horizontal", "vertical", "triangular", "L shape"]) if
var.get()]
    second_window.destroy()
    open_third_window()


def open_third_window():
    play_sound('button_click.wav')
    third_window = tk.Toplevel(root)
    third_window.title("Final Logo")
    third_window.geometry('800x600')

    # Play video as the background
    final_logo_frame = tk.Frame(third_window)
    final_logo_frame.pack(fill=tk.BOTH, expand=True)

    cap_final = cv2.VideoCapture('congrats.mp4')  # Change path to your video
file
    video_width_final = int(cap_final.get(cv2.CAP_PROP_FRAME_WIDTH))
    video_height_final = int(cap_final.get(cv2.CAP_PROP_FRAME_HEIGHT))

    lbl_final = tk.Label(final_logo_frame)
    lbl_final.pack(fill=tk.BOTH, expand=True)
    def update_frame_final():
        ret, frame = cap_final.read()

        if ret:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(frame)
            img.thumbnail((video_width_final, video_height_final), Image.LANCZOS)
            imgtk = ImageTk.PhotoImage(image=img)
            lbl_final.imgtk = imgtk
            lbl_final.config(image=imgtk)

        lbl_final.after(10, update_frame_final)

    update_frame_final()

    # Display final logo
    display_final_logo(third_window)
    exit_button = tk.Button(third_window, text="    Exit    ", bg="#FF3399",
fg="#FFFFFF", font=("Arial", 16), command=root.destroy)
    exit_button.place(relx=0.4, rely=0.9, anchor='center')
    regenerate_button = tk.Button(third_window, text="Regenerate", bg="#FF3399",
fg="#FFFFFF", font=("Arial", 16), command=open_string_input_window)
    regenerate_button.place(relx=0.6, rely=0.9, anchor='center')


def create_frame(window, shape, relx, rely):
    play_sound('button_click.wav')
    frame_bg_color = frame_bgs[0] if frame_bgs else 'white'

    if shape == 'circular':
        return create_circular_frame(window, relx, rely, frame_bg_color)
    elif shape == 'rectangular':
        return create_rectangular_frame(window, relx, rely, frame_bg_color)
    elif shape == 'pentagonal':
        return create_pentagonal_frame(window, relx, rely, frame_bg_color)
    else:
        frame = tk.Frame(window, bg=frame_bg_color)
        frame.place(relx=relx, rely=rely, anchor='center')
        return frame

def create_circular_frame(window, relx, rely, bg_color):
    canvas = tk.Canvas(window, width=300, height=300, highlightthickness=0,
bg=bg_color)
    canvas.create_oval(10, 10, 290, 290, outline="white", width=2)
    canvas.place(relx=relx, rely=rely, anchor='center')
    frame = tk.Frame(canvas, bg=bg_color)
    frame.place(relx=0.5, rely=0.5, anchor='center')
    return frame

def create_rectangular_frame(window, relx, rely, bg_color):
    canvas = tk.Canvas(window, width=300, height=300, highlightthickness=0,
bg=bg_color)
    canvas.create_rectangle(10, 10, 290, 290, outline="white", width=2)
    canvas.place(relx=relx, rely=rely, anchor='center')

    frame = tk.Frame(canvas, bg=bg_color)
    frame.place(relx=0.5, rely=0.5, anchor='center')
    return frame

def create_pentagonal_frame(window, relx, rely, bg_color, size=190):  # Set size
to 190
    canvas = tk.Canvas(window, width=size*2, height=size*2, highlightthickness=0,
bg=bg_color)
    vertices = []
    for i in range(5):
        angle = math.radians(72 * i - 90)
        x = size + size * math.cos(angle)
        y = size + size * math.sin(angle)
        vertices.extend((x, y))
    canvas.create_polygon(vertices, outline="white", width=2, fill=bg_color)  #
Outline color set to white
    canvas.place(relx=relx, rely=rely, anchor='center')

    frame = tk.Frame(canvas, bg=bg_color)
    frame.place(relx=0.5, rely=0.5, anchor='center')
    return frame

def display_final_logo(window):
    play_sound('button_click.wav')
    for shape in frame_shapes:
        frame = create_frame(window, shape, 0.5, 0.5)
        display_images(frame, top_3_items, logo_styles[0] if logo_styles else "",
arrangement_styles[0] if arrangement_styles else "horizontal")

def display_images(frame, items, suffix, arrangement):
    play_sound('button_click.wav')
    global photos

    bg_color = frame_bgs[0] if frame_bgs else "white"
```

```python
    if arrangement == "horizontal":
        positions = [(0, i) for i in range(len(items))]
    elif arrangement == "vertical":
        positions = [(i, 0) for i in range(len(items))]
    elif arrangement == "triangular":
        positions = [(0, 1), (1, 0), (1, 2)]
    elif arrangement == "L shape":
        positions = [(0, 0), (1, 0), (1, 1)]

    for i, (char, count) in enumerate(items):
        path = f"{char}{suffix}.png"
        if os.path.isfile(path):
            image = Image.open(path)
            new_width, new_height = 80, 80
            image = image.resize((new_width, new_height), Image.LANCZOS)
            photo = ImageTk.PhotoImage(image)

            label = tk.Label(frame, image=photo, bg=bg_color)
            label.image = photo  # Keep a reference to avoid garbage collection
            position = positions[i]
            label.grid(row=position[0], column=position[1])
            photos.append(photo)
        else:
            print(f"No image found for character: {char}{suffix}")

def analyze_string(s, string_input_window):
    play_sound('button_click.wav')
    s = s.lower()
    s = ''.join(filter(lambda x: x.isalpha(), s))
    if distinct_chars(s) < 3:
        messagebox.showwarning("Warning", "Please enter a string with at least 3 
distinct characters.")
    else:
        d = {x: s.count(x) for x in s}
        sd = sorted(d.items(), key=lambda x: (-x[1], x[0]))
        global top_3_items
        top_3_items = sd[:3]
        result_text = "Top 3 characters and their counts:\n" + "\n".join(f"{l}: 
{c}" for l, c in top_3_items)
        messagebox.showinfo("Result", result_text)
        string_input_window.destroy()
        open_second_window()

def distinct_chars(s):
    return len(set(s))

# Create a Tkinter window
root = tk.Tk()
root.title("First Window")

# Open the video file
cap = cv2.VideoCapture('homepage.mp4')

# Get the original video dimensions
video_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
video_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Set the window size to the video dimensions
root.geometry(f"{video_width}x{video_height}")

# Create a label to hold the video frame
lbl = tk.Label(root)
lbl.pack(fill=tk.BOTH, expand=True)

# Start updating the frame
update_frame()

# Add "GET STARTED" button
get_started_button = tk.Button(root, text="GET STARTED", bg="#FF3399",
fg="#FFFFFF", font=("Arial", 16), command=open_string_input_window)
get_started_button.place(relx=0.5, rely=0.8, anchor='center')

# Run the Tkinter main loop
root.mainloop()

# Release the video capture object and close any OpenCV windows
cap.release()
cv2.destroyAllWindows()
```