App.py

import requests

import json

import mysql.connector

from flask import Flask, abort, request

from uuid import uuid4

import requests

import requests.auth

import urllib

# API credentials

api\_key = 'oVhfBgjRRe5QCjvFlB5Iw'

api\_secret = 'IYMLJO5o05E3LtJ3afdT9tPjgs4bROD3'

CLIENT\_ID = api\_key # Fill this in with your client ID

CLIENT\_SECRET = api\_secret # Fill this in with your client secret

REDIRECT\_URI = "http://localhost:65010/zoom\_callback"

app = Flask(\_\_name\_\_)

@app.route('/')

def homepage():

    text = '<a href="%s">Authenticate with Zoom</a>'

    return text % make\_authorization\_url()

def make\_authorization\_url():

    # Generate a random string for the state parameter

    # Save it for use later to prevent xsrf attacks

    params = {"client\_id": CLIENT\_ID,

              "response\_type": "code",

              "redirect\_uri": REDIRECT\_URI}

    url = "https://zoom.us/oauth/authorize?" + urllib.parse.urlencode(params)

    return url

@app.route('/zoom\_callback')

def zoom\_callback():

    error = request.args.get('error', '')

    if error:

        return "Error: " + error

    code = request.args.get('code')

    access\_token = get\_token(code)

    # Note: In most cases, you'll want to store the access token, in, say,

    # a session for use in other parts of your web app.

    return "Your user info is: %s" % get\_username(access\_token)

def get\_token(code):

    global oAuth

    client\_auth = requests.auth.HTTPBasicAuth(CLIENT\_ID, CLIENT\_SECRET)

    post\_data = {"grant\_type": "authorization\_code",

                 "code": code,

                 "redirect\_uri": REDIRECT\_URI}

    response = requests.post("https://zoom.us/oauth/token",

                             auth=client\_auth,

                             data=post\_data)

    token\_json = response.json()

    print(token\_json)

    oAuth = token\_json["access\_token"]

    return token\_json["access\_token"]

def get\_username(access\_token):

    headers= {"Authorization": "bearer " + access\_token}

    response = requests.get("https://api.zoom.us/v2/users/me", headers=headers)

    me\_json = response.json()

    # getMeetingInfo(access\_token)

    return me\_json

def getMeetingInfo(access\_token):

    # Meeting ID for which you want to get attendees

    meeting\_id = '88321453048'

    # API endpoint

    url = f'https://api.zoom.us/v2/report/meetings/{meeting\_id}/participants'

    # Headers

    headers = {

        'Authorization': f'Bearer {access\_token}',

        'Content-Type': 'application/json'

    }

    # Make GET request

    response = requests.get(url, headers=headers)

    # Check if request was successful

    if response.status\_code == 200:

        attendees = response.json()

        participant\_list = json.dumps(attendees, indent=4)

        #print(participant\_list)

        with open('participant\_data.json', 'r') as f:

            data = json.load(f)

        #print(data)

        try:

            con = mysql.connector.connect(

                user = 'root',

                password = 'root@123',

                host = 'localhost',

                port = 3306,

                database = 'test\_database'

            )

            if con.is\_connected():

                print("Connected!")

        except Exception as e:

            print("Cannot Connect! Try Again!")

        cursor = con.cursor()

        #cursor.execute("create table zoom\_participant\_list (Name varchar(255), User\_Email varchar(255), Join\_Time datetime, Leave\_Time datetime, Duration int(5))")

        for item in data['participants']:

            Name = item['name']

            User\_Email = item['user\_email']

            Join\_Time = item['join\_time']

            Leave\_Time = item['leave\_time']

            Duration = item['duration']

            cursor.execute("INSERT INTO zoom\_participant\_list (Name,User\_Email,Join\_Time,Leave\_Time,Duration)")

            con.commit()

        #print(type(Name))

    else:

        print(f'Failed to retrieve attendees. Status code:{response.status\_code}, Error: {response.text}')

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, port=65010)

# # Generate JWT token

# import jwt

# import datetime

# token\_exp = datetime.datetime.utcnow() + datetime.timedelta(minutes=60)  # Token expires in 1 hour

# token\_payload = {

#     'iss': api\_key,

#     'exp': token\_exp

# }

# # API endpoint

# url1 = f'https://zoom.us/oauth/token'

# headers = {

#     'Authorization': f'Basic 3Q\_qrPdJQ-GJBpIPbcZ6Yw',

#     'Content-Type': 'application/x-www-form-urlencoded'

# }

# # Make GET request

# response = requests.get(url1, headers=headers)

# jwt\_token = jwt.encode(token\_payload, api\_secret, algorithm='HS256')

# #jwt\_token = "eyJzdiI6IjAwMDAwMSIsImFsZyI6IkhTNTEyIiwidiI6IjIuMCIsImtpZCI6IjJhYzBkNmQyLTk0YTEtNGNlZi1hMmE3LWQ0NDMzNzI4ZjQ4MCJ9.eyJ2ZXIiOjksImF1aWQiOiI1ZmQ5YWU1YjNiMWRmNzc2OTY2Nzc5NDk4ZDVkYTY3MiIsImNvZGUiOiJGcm1VRzY1clRZdXdzUko0WFVCVG5tZ2ZTdzNxS1EzTVEiLCJpc3MiOiJ6bTpjaWQ6cHJobG5xdkNSQ0cyaGZIb3FvZWZRIiwiZ25vIjowLCJ0eXBlIjowLCJ0aWQiOjAsImF1ZCI6Imh0dHBzOi8vb2F1dGguem9vbS51cyIsInVpZCI6IkVZczVFZk1iUmptTzlzTVRobFJTYkEiLCJuYmYiOjE3MDkzMzc3ODgsImV4cCI6MTcwOTM0MTM4OCwiaWF0IjoxNzA5MzM3Nzg4LCJhaWQiOiJWX19QNXF3NlE5ZUVjbDJrZHljMmhRIn0.Rb9AsCuTD0FJyI3RBgvoCv6v018qYEVMcHglhmhdWuKTlSe9WSwW01Wv3VyfvL3fU5b2zfLe4NRgu1VIwp0ESQ"

Sql\_Backend

import mysql.connector

import pandas as pd

from tabulate import tabulate

import json

import tkinter as tk

from tkinter import ttk

def fetch\_and\_display\_data(host, user, password, database, query, json\_file):

    # Load participant data from JSON

    with open('participant\_data.json', 'r') as f:

        data = json.load(f)

    try:

        # Connect to MySQL database

        con = mysql.connector.connect(

            user=user,

            password=password,

            host=host,

            database=database

        )

        if con.is\_connected():

            print("Connected!")

    except Exception as e:

        print("Cannot Connect! Try Again!")

        return

    # Create a cursor

    cursor = con.cursor()

    # Iterate over the data and insert each participant into the database

    for item in data['participants']:

        Name = item['name']

        User\_Email = item['user\_email']

        Join\_Time = item['join\_time']

        Leave\_Time = item['leave\_time']

        Duration = item['duration']

        add\_user = """INSERT INTO test\_database.zoom\_participant\_list

                      (Name, User\_Email, Join\_Time, Leave\_Time, Duration)

                      VALUES (%s, %s, %s, %s, %s)"""

        data\_user = (Name, User\_Email, Join\_Time, Leave\_Time, Duration)

        cursor.execute(add\_user, data\_user)

    # Execute the query to fetch data

    cursor.execute(query)

    # Fetch the result

    result = cursor.fetchall()

    # Get column names

    columns = [desc[0] for desc in cursor.description]

    # Convert the result into a DataFrame

    result\_df = pd.DataFrame(result, columns=columns)

    # Close cursor and connection

    cursor.close()

    con.close()

    return result\_df

def sort\_table(df, column):

    return df.sort\_values(by=column)

def filter\_table(df, column, filter\_text):

    # Convert column to string type before applying .str accessor

    if df[column].dtype != 'object':

        df[column] = df[column].astype(str)

    return df[df[column].str.contains(filter\_text, case=False)]

def display\_table\_in\_window(df):

    def sort\_table\_handler():

        sorted\_df = sort\_table(df, combo\_sort.get())

        label.config(text=sorted\_df.to\_string())

    def filter\_table\_handler():

        filtered\_df = filter\_table(df, combo\_filter.get(), entry\_filter.get())

        label.config(text=filtered\_df.to\_string())

    root = tk.Tk()

    # Create a frame for sorting and filtering

    frame = tk.Frame(root)

    frame.pack()

    # Create a sorting dropdown

    combo\_sort = ttk.Combobox(frame, values=df.columns.tolist())

    combo\_sort.pack(side=tk.LEFT)

    combo\_sort.set("Sort by")

    # Create a button for sorting

    btn\_sort = tk.Button(frame, text="Sort", command=sort\_table\_handler)

    btn\_sort.pack(side=tk.LEFT)

    # Create a filter dropdown

    combo\_filter = ttk.Combobox(frame, values=df.columns.tolist())

    combo\_filter.pack(side=tk.LEFT)

    combo\_filter.set("Filter by")

    # Create an entry for filter criteria

    entry\_filter = tk.Entry(frame)

    entry\_filter.pack(side=tk.LEFT)

    # Create a button for filtering

    btn\_filter = tk.Button(frame, text="Filter", command=filter\_table\_handler)

    btn\_filter.pack(side=tk.LEFT)

    # Create a Tkinter label to display the table

    label = tk.Label(root, text=df.to\_string(), justify='left', font=('Courier', 10))

    label.pack()

    root.mainloop()

# Example usage:

host = 'localhost'

user = 'root'

password = 'root@123'

database = 'test\_database'

query = "SELECT \* FROM test\_database.zoom\_participant\_list"

json\_file = 'participant\_data.json'

result\_df = fetch\_and\_display\_data(host, user, password, database, query, json\_file)

display\_table\_in\_window(result\_df)

main.py

import tkinter

from tkinter import \*

from tkinter import ttk

import tkinter.messagebox as tkMessageBox

from tkinter.filedialog import askopenfilename

from tkinter import filedialog

from PIL import Image, ImageTk

import mysql.connector

import datetime

from tkcalendar import Calendar, DateEntry

import re

import io

from io import BytesIO

import tkinter.messagebox as mbox

from tkinter.font import Font

import pyttsx3

engine = pyttsx3.init()

def database():

    try:

        global con, cur

        con = mysql.connector.connect(

        host="localhost",

        user="root",

        password="root@123",

        database="python\_mini\_project"

        )

        print("Connection Established Successfully")

        cur = con.cursor()

    except:

        print("Database Status: There was an error")

def detailDisplay(event):

    database()

    detail\_display = Tk()

    window\_width = 700

    window\_height = 700

    screen\_width = detail\_display.winfo\_screenwidth()

    screen\_height = detail\_display.winfo\_screenheight()

    x = (screen\_width/2) - (window\_width/2)

    y = (screen\_height/2) - (window\_height/2)

    detail\_display.geometry('%dx%d+%d+%d' % (window\_width, window\_height, x, y-50))

    detail\_display.resizable(width=False, height=False)

    detail\_display.configure(bg='white')

    item = tree.selection()[0]

    sap\_no = tree.item(item, "values")[3]

    cur.execute("SELECT \* FROM student\_data WHERE sap\_no=%s", (sap\_no,))

    record = cur.fetchone()

    if record:

        detail\_display.title(f"{record[2]} Record Details")

        # Add labels to display the record details

        roll\_label = Label(detail\_display, text="Roll No", font=("Helvatica", 10, "bold"), bg="white")

        roll\_label.place(x=370, y=50)

        Label(detail\_display, text=record[2],  bg="white", font=("Helvatica", 10)).place(x=430,y=50)

        sap\_label = Label(detail\_display, text="Sap No", font=("Helvatica", 10, "bold"), bg="white")

        sap\_label.place(x=150, y=50)

        Label(detail\_display, text=record[1],font=("Helvatica", 10),  bg="white").place(x=210,y=50)

        year\_of\_admission\_label = Label(detail\_display, text="Year of Admission: ", bg="white", font=("Helvatica", 10, "bold"))

        year\_of\_admission\_label.place(x=20,y=120)

        Label(detail\_display, text=record[3],font=("Helvatica", 10),  bg="white").place(x=150,y=120)

        type\_of\_admission\_label = Label(detail\_display, text="Type of Admission: ", bg="white", font=("Helvatica", 10, "bold"))

        type\_of\_admission\_label.place(x=20,y=155)

        Label(detail\_display, text=record[4], font=("Helvatica", 10),  bg="white").place(x=150,y=155)

        branch\_label = Label(detail\_display, text="Branch: ", bg="white", font=("Helvatica", 10, "bold"))

        branch\_label.place(x=20,y=210)

        Label(detail\_display, text=record[5], font=("Helvatica", 10),  bg="white").place(x=150,y=210)

        name\_label = Label(detail\_display, text="Student Name: ", bg="white", font=("Helvatica", 10, "bold"))

        name\_label.place(x=20, y=245)

        Label(detail\_display, text=record[6], font=("Helvatica", 10),  bg="white").place(x=150, y=245)

        contact\_label = Label(detail\_display, text="Contact No.: ", bg="white", font=("Helvatica", 10, "bold"))

        contact\_label.place(x=20,y=290)

        Label(detail\_display, text=record[8], font=("Helvatica", 10),  bg="white").place(x=150,y=315)

        Label(detail\_display, text=record[7], font=("Helvatica", 10),  bg="white").place(x=150,y=290)

        email\_label = Label(detail\_display, text="Email: ", bg="white", font=("Helvatica", 10, "bold"))

        email\_label.place(x=20,y=360)

        Label(detail\_display, text=record[10], font=("Helvatica", 10),  bg="white").place(x=150,y=390)

        Label(detail\_display, text=record[9], font=("Helvatica", 10),  bg="white").place(x=150,y=360)

        date\_of\_birth\_label = Label(detail\_display, text="Date of Birth:", bg="white", font=("Helvatica", 10, "bold"))

        date\_of\_birth\_label.place(x=20,y=445)

        Label(detail\_display, text=record[11], font=("Helvatica", 10),  bg="white").place(x=150, y=445)

        address\_label = Label(detail\_display, text="Current Address: ", bg="white", font=("Helvatica", 10, "bold"))

        address\_label.place(x=20, y=505)

        addr = Label(detail\_display, text=record[12], font=("Helvatica", 10),  bg="white", width=20, height=2, wraplength=160, anchor="w", justify="left")

        addr.place(x=150, y=505)

        achedemic\_label = Label(detail\_display, text="Achedemic Record: ", bg="white", font=("Helvatica", 10, "bold"))

        achedemic\_label.place(x=20, y=560)

        l1 = Label(detail\_display, text="Exam ", bg="white", font=("Helvatica", 10)).place(x=20, y=580)

        l2 = Label(detail\_display, text="Year of Passing", bg="white", font=("Helvatica", 10)).place(x=80, y=580)

        l3 = Label(detail\_display, text="Percentage(%)", bg="white", font=("Helvatica", 10)).place(x=200,y=580)

        l4 = Label(detail\_display, text="SSC", bg="white", font=("Helvatica", 10)).place(x=20, y=620)

        l5 = Label(detail\_display, text="HSC", bg="white", font=("Helvatica", 10)).place(x=20, y=660)

        Label(detail\_display, text=record[13], font=("Helvatica", 10),  bg="white").place(x=80,y=620)

        Label(detail\_display, text=record[14], font=("Helvatica", 10),  bg="white").place(x=200,y=620)

        Label(detail\_display, text=record[15], font=("Helvatica", 10),  bg="white").place(x=80,y=660)

        Label(detail\_display, text=record[16], font=("Helvatica", 10),  bg="white").place(x=200,y=660)

        sem\_label = Label(detail\_display, text="Semesters", bg="white", font=("Helvatica", 10, "bold")).place(x=400, y=400)

        year\_label = Label(detail\_display, text="Year", bg="white", font=("Helvatica", 10, "bold")).place(x=500, y=400)

        percent\_label = Label(detail\_display, text="Percentages(%)", bg="white", font=("Helvatica", 10, "bold")).place(x=580, y=400)

        sem1\_label = Label(detail\_display, text="Semester 1", bg="white", font=("Helvatica", 10)).place(x=400, y=440)

        sem2\_label = Label(detail\_display, text="Semester 2", bg="white", font=("Helvatica", 10)).place(x=400, y=460)

        sem3\_label = Label(detail\_display, text="Semester 3", bg="white", font=("Helvatica", 10)).place(x=400, y=480)

        sem4\_label = Label(detail\_display, text="Semester 4", bg="white", font=("Helvatica", 10)).place(x=400, y=500)

        sem5\_label = Label(detail\_display, text="Semester 5", bg="white", font=("Helvatica", 10)).place(x=400, y=520)

        sem6\_label = Label(detail\_display, text="Semester 6", bg="white", font=("Helvatica", 10)).place(x=400, y=540)

        Label(detail\_display, text=record[18], font=("Helvatica", 10),  bg="white").place(x=500, y=440)

        Label(detail\_display, text=record[19], font=("Helvatica", 10),  bg="white").place(x=500, y=460)

        Label(detail\_display, text=record[20], font=("Helvatica", 10),  bg="white").place(x=500, y=480)

        Label(detail\_display, text=record[21], font=("Helvatica", 10),  bg="white").place(x=500, y=500)

        Label(detail\_display, text=record[22], font=("Helvatica", 10),  bg="white").place(x=500, y=520)

        Label(detail\_display, text=record[23], font=("Helvatica", 10),  bg="white").place(x=500, y=540)

        Label(detail\_display, text=record[24], font=("Helvatica", 10),  bg="white").place(x=580, y=440)

        Label(detail\_display, text=record[25], font=("Helvatica", 10),  bg="white").place(x=580, y=460)

        Label(detail\_display, text=record[26], font=("Helvatica", 10),  bg="white").place(x=580, y=480)

        Label(detail\_display, text=record[27], font=("Helvatica", 10),  bg="white").place(x=580, y=500)

        Label(detail\_display, text=record[28], font=("Helvatica", 10),  bg="white").place(x=580, y=520)

        Label(detail\_display, text=record[29], font=("Helvatica", 10),  bg="white").place(x=580, y=540)

        image\_data = record[30]

        # Create an image from the retrieved data

        image = Image.open(io.BytesIO(image\_data))

        # Calculate the aspect ratio of the image

        image\_width, image\_height = image.size

        aspect\_ratio = image\_width / image\_height

        # Determine the dimensions for cropping

        canvas\_width = 150

        canvas\_height = 150

        if aspect\_ratio > 1:

            # Landscape image, crop the width

            crop\_width = image\_height \* aspect\_ratio

            crop\_height = image\_height

        else:

            # Portrait image or square image, crop the height

            crop\_width = image\_width

            crop\_height = image\_width / aspect\_ratio

        crop\_x = (image\_width - crop\_width) / 2

        crop\_y = (image\_height - crop\_height) / 2

        # Crop the image to desired dimensions

        cropped\_image = image.crop((crop\_x, crop\_y, crop\_x + crop\_width, crop\_y + crop\_height))

        # Resize the cropped image to fit the canvas

        resized\_image = cropped\_image.resize((canvas\_width, canvas\_height), Image.ANTIALIAS)

        # Display the resized image in a canvas

        image\_upload = Canvas(detail\_display, width=canvas\_width, height=canvas\_height)

        image\_upload.place(x=500, y=120)

        image\_object = ImageTk.PhotoImage(resized\_image)

        image\_upload.image = image\_object  # Keep a reference to the image object

        image\_upload.create\_image(int(canvas\_width / 2), int(canvas\_height / 2), image=image\_object)

        image\_upload.update()

def display():

    database()

    count = 0

    tree.delete(\*tree.get\_children())

    cur.execute("ALTER TABLE student\_data AUTO\_INCREMENT = 1")

    cur.execute("select sr\_no, stu\_name, roll\_no, sap\_no, branch, present\_year from student\_data")

    fetch = cur.fetchall()

    for row in fetch:

        tree.insert("", "end", values=row)

        tree.bind("<Double-Button-1>", detailDisplay)

        count += 1

    tkMessageBox.showinfo("Total Records",f"Total Records Found: {count}")

    cur.close()

    con.close()

def insert():

    """

    This function creates a GUI window for inserting student biodata records with various input fields

    and options.

    """

    def clearData():

        image = canvas\_image.delete(0, ALL)

        sap\_no = sap\_text.delete(0,END)

        roll\_no = roll\_text.delete(0,END)

        yoa = year\_of\_admission\_option.delete(0,END)

        branch = branch\_text.delete(0,END)

        stu\_name = name\_text.delete(0,END)

        contact1 = contact\_text1.delete(0,END)

        contact2 = contact\_text2.delete(0,END)

        email1 = email\_text1.delete(0,END)

        email2 = email\_text2.delete(0,END)

        #dob= date\_of\_birth.delete(0,END)

        address = address\_text.delete("1.0",END)

        ssc\_pass\_year = tf1.delete(0,END)

        ssc\_marks = tf2.delete(0,END)

        hsc\_pass\_year = tf3.delete(0,END)

        hsc\_marks = tf4.delete(0,END)

        sem1 = year1\_entry.delete(0,END)

        sem2 = year2\_entry.delete(0,END)

        sem3 = year3\_entry.delete(0,END)

        sem4 = year4\_entry.delete(0,END)

        sem5 = year5\_entry.delete(0,END)

        sem6 = year6\_entry.delete(0,END)

        sem1\_percent = percentage1\_entry.delete(0,END)

        sem2\_percent = percentage2\_entry.delete(0,END)

        sem3\_percent = percentage3\_entry.delete(0,END)

        sem4\_percent = percentage4\_entry.delete(0,END)

        sem5\_percent = percentage5\_entry.delete(0,END)

        sem6\_percent = percentage6\_entry.delete(0,END)

    def storeData():

        """

        The function connects to a database, inserts a record, commits the changes, and closes the

        database.

        """

        try:

            database()

            sap\_no = sap\_text.get()

            if not sap\_no:

                tkMessageBox.showinfo("Alert", "Please Enter SAP No")

                return False

            cur.execute("SELECT COUNT(\*) FROM student\_data WHERE sap\_no = %s", (sap\_no,))

            count = cur.fetchone()[0]

            if count > 0:

                tkMessageBox.showinfo("Alert", "SAP No already exists")

                return False

            image\_binary = open(filename, 'rb').read()

            if not image\_binary:

                tkMessageBox.showinfo("Alert","Please Select a Image")

                return False

            sap\_no = sap\_text.get()

            if not sap\_no:

                tkMessageBox.showinfo("Alert", "Please Enter SAP No")

                return False

            roll\_no = roll\_text.get()

            if not roll\_no:

                tkMessageBox.showinfo("Alert", "Please Enter Roll No")

                return False

            yoa = year\_of\_admission\_option.get()

            if not yoa:

                tkMessageBox.showinfo("Alert", "Please Select Year of Admission")

                return False

            toa = selected\_value.get()

            branch = branch\_text.get()

            if not branch:

                tkMessageBox.showinfo("Alert", "Please Select A Branch")

                return False

            stu\_name = name\_text.get()

            if not stu\_name:

                tkMessageBox.showinfo("Alert", "Please Enter Stuent Name")

                return False

            contact1 = contact\_text1.get()

            contact2 = contact\_text2.get()

            if not contact1 and not contact2:

                tkMessageBox.showinfo("Alert", "Please Enter Atleast 1 Contact Info.")

                return False

            email1 = email\_text1.get()

            email2 = email\_text2.get()

            if not email1 and not email2:

                tkMessageBox.showinfo("Alert", "Please Enter Atleast 1 Email ID")

                if validate\_email\_input:

                    tkMessageBox.showinfo("Alert", "Please Enter Correct Email")

                    return False

            date = date\_of\_birth.get()

            dob = str(date)

            address = address\_text.get("1.0", "end-1c")

            if not address:

                tkMessageBox.showinfo("Alert", "Please Enter Your Address")

                return False

            ssc\_pass\_year = tf1.get()

            ssc\_marks = tf2.get()

            hsc\_pass\_year = tf3.get()

            hsc\_marks = tf4.get()

            sem1 = year1\_entry.get()

            sem2 = year2\_entry.get()

            sem3 = year3\_entry.get()

            sem4 = year4\_entry.get()

            sem5 = year5\_entry.get()

            sem6 = year6\_entry.get()

            sem1\_percent = percentage1\_entry.get()

            sem2\_percent = percentage2\_entry.get()

            sem3\_percent = percentage3\_entry.get()

            sem4\_percent = percentage4\_entry.get()

            sem5\_percent = percentage5\_entry.get()

            sem6\_percent = percentage6\_entry.get()

            calc = int(yoa)

            year = datetime.datetime.now().year

            year = year - calc

            present\_year = str(year)

            cur.execute("insert into student\_data (sap\_no, roll\_no, yoa, toa, branch, stu\_name, contact1, contact2, email1, email2, dob, address, ssc\_pass\_year, ssc\_marks, hsc\_pass\_year, hsc\_marks, present\_year, sem1, sem2, sem3, sem4, sem5, sem6, sem1\_percent, sem2\_percent, sem3\_percent, sem4\_percent, sem5\_percent, sem6\_percent, image\_binary) values (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)",

                        (sap\_no, roll\_no, yoa, toa, branch,stu\_name,contact1, contact2, email1, email2, dob, address, ssc\_pass\_year, ssc\_marks, hsc\_pass\_year, hsc\_marks, present\_year, sem1, sem2, sem3, sem4, sem5, sem6, sem1\_percent, sem2\_percent, sem3\_percent, sem4\_percent, sem5\_percent, sem6\_percent,image\_binary))

            con.commit()

            tkMessageBox.showinfo("Success", "Record Inserted")

            display()

        except Exception as e:

            print("Database Error => ", e)

        finally:

            if con:

                con.close()

                print("Database Closed Successfully")

                print()

    insert = Tk()

    insert.title("Add Student")

    # setting window on the center of the screen

    window\_width = 700

    window\_height = 700

    screen\_width = insert.winfo\_screenwidth()

    screen\_height = insert.winfo\_screenheight()

    x = (screen\_width/2) - (window\_width/2)

    y = (screen\_height/2) - (window\_height/2)

    insert.geometry('%dx%d+%d+%d' % (window\_width, window\_height, x, y-50))

    insert.resizable(width=False, height=False)

    insert.configure(bg='white')

    def choose\_image():

        global canvas\_image, image, filename

        canvas\_image = Canvas(width=150, height=150)

        canvas\_image.place(x=500,y=120)

        filename = filedialog.askopenfilename(title="Select file", filetypes=(("PNG files", "\*.png"), ("JPEG files", "\*.jpg"), ("JPG files", "\*.jpg")))

        print("Selected file:", filename)  # Add this line to print the filename

        # Load the image

        if filename:

            try:

                image = Image.open(filename)

            except Exception as e:

                print("Error loading image:", e)  # Add this line to print any errors

                return

            image\_width, image\_height = image.size

            canvas\_width, canvas\_height = canvas\_image.winfo\_width(), canvas\_image.winfo\_height()

            # Calculate the scaling factor

            scale = min(canvas\_width/image\_width, canvas\_height/image\_height)

            new\_width = int(image\_width\*scale)

            new\_height = int(image\_height\*scale)

            # Resize the image

            resized\_image = image.resize((new\_width, new\_height), Image.ANTIALIAS)

            image\_object = ImageTk.PhotoImage(resized\_image)

            # Add the image to the canvas

            canvas\_image.delete("all")  # Remove any previously added image

            canvas\_image.create\_image(canvas\_width/2, canvas\_height/2, image=image\_object)

            canvas\_image.image = image\_object

    # Creating a form

    canvas\_image\_upload = Canvas(insert, width=150, height=150)

    canvas\_image\_upload.place(x=500,y=120)

    img\_label = Label(insert, text="Upload Photo: ", bg="white", font=("Helvatica", 10, "bold"))

    img\_label.place(x=400, y=120)

    # Create a button for selecting the image

    upload\_button = Button(insert, text="Choose Photo", command=choose\_image)

    upload\_button.place(x=530, y=290)

    sap\_label = Label(insert, text="Sap No", font=("Helvatica", 10, "bold"), bg="white")

    sap\_label.place(x=50, y=50)

    sap\_text = Entry(insert, width=20, font=("Helvatica", 10), relief="groove", bd=2)

    sap\_text.place(x=110,y=50)

    roll\_label = Label(insert, text="Roll No", font=("Helvatica", 10, "bold"), bg="white")

    roll\_label.place(x=330, y=50)

    roll\_text = Entry(insert, width=20, font=("Helvatica", 10), relief="groove", bd=2)

    roll\_text.place(x=400,y=50)

    current\_year = datetime.datetime.now().year

    year = [i for i in range(2005,current\_year)]

    year\_of\_admission\_label = Label(insert, text="Year of Admission: ", bg="white", font=("Helvatica", 10, "bold"))

    year\_of\_admission\_option = ttk.Combobox(insert, width=20, font=("Helvatica", 10), values=year, state="readonly")

    year\_of\_admission\_option.place(x=150,y=120)

    year\_of\_admission\_label.place(x=20,y=120)

    selected\_value = StringVar(value="First Year")

    def update\_selection(selection):

        selected\_value.set(selection)

    type\_of\_admission\_label = Label(insert, text="Type of Admission: ", bg="white", font=("Helvatica", 10, "bold"))

    type\_of\_admission\_option1 = Radiobutton(insert, text="First Year", variable=selected\_value, value="First Year", bg="white", command=lambda: update\_selection("First Year"))

    type\_of\_admission\_option2 = Radiobutton(insert, text="Direct Second Year",  variable=selected\_value, value="Direct Second Year", bg="white", command=lambda: update\_selection("Direct Second Year"))

    type\_of\_admission\_option1.place(x=150,y=155)

    type\_of\_admission\_option2.place(x=150,y=176)

    type\_of\_admission\_label.place(x=20,y=155)

    branch\_options = ["Information Technology", "Computer Engineering", "Mechanical Engineering", "Civil Engineering", "Electrical Engineering", "Plastic Engineering", "Chemical Engineering", "Electronics & Telecommunicaltion Engineering"]

    branch\_label = Label(insert, text="Branch: ", bg="white", font=("Helvatica", 10, "bold"))

    branch\_text = ttk.Combobox(insert, width=20, font=("Helvatica", 10), values=branch\_options, state="readonly")

    branch\_text.place(x=150,y=210)

    branch\_label.place(x=20,y=210)

    def validate\_name\_input(name):

        if name == "":

            return True

        name\_regex = "^[A-Za-z\s]{1,}[\.]{0,1}[A-Za-z\s]{0,}$"

        if re.match(name\_regex, name):

            return True

        else:

            return False

    validate\_name = (insert.register(validate\_name\_input), '%P')

    name\_label = Label(insert, text="Student Name: ", bg="white", font=("Helvatica", 10, "bold"))

    name\_text = Entry(insert, width=23, font=("Helvatica", 10), relief="groove", bd=2, validate='all', validatecommand=validate\_name)

    name\_text.place(x=150, y=245)

    name\_label.place(x=20, y=245)

    def validate\_contact\_input(contact):

        contact\_regex = r"\d{10}"

        if re.match(contact\_regex, contact):

            return True

        else:

            return False

    validate\_contact = (insert.register(validate\_contact\_input), '%P')

    contact\_label = Label(insert, text="Contact No.: ", bg="white", font=("Helvatica", 10, "bold"))

    contact\_text1 = Entry(insert, width=23, font=("Helvatica", 10), relief="groove", bd=2, validate='focusout', validatecommand=validate\_contact)

    contact\_text1.place(x=150,y=290)

    contact\_text2 = Entry(insert, width=23, font=("Helvatica", 10), relief="groove", bd=2, validate='focusout', validatecommand=validate\_contact)

    contact\_text2.place(x=150,y=315)

    contact\_label.place(x=20,y=290)

    def validate\_email\_input(email):

        email\_regex = r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

        if re.match(email\_regex, email):

            return True

        else:

            return False

    validate\_email = (insert.register(validate\_email\_input), '%P')

    email\_label = Label(insert, text="Email: ", bg="white", font=("Helvatica", 10, "bold"))

    email\_text1 = Entry(insert, width=23, font=("Helvatica", 10), relief="groove", bd=2, validate='focusout', validatecommand=validate\_email)

    email\_text1.place(x=150,y=360)

    email\_text2 = Entry(insert, width=23, font=("Helvatica", 10), relief="groove", bd=2, validate='focusout', validatecommand=validate\_email)

    email\_text2.place(x=150,y=390)

    email\_label.place(x=20,y=360)

    date\_of\_birth\_label = Label(insert, text="Date of Birth:", bg="white", font=("Helvatica", 10, "bold"))

    date\_of\_birth\_label.place(x=20,y=445)

    date\_of\_birth = DateEntry(insert, width=12, background='darkblue', foreground='white', borderwidth=2, state="readonly", date\_pattern="dd-mm-yyyy")

    date\_of\_birth.place(x=150, y=445)

    address\_label = Label(insert, text="Current Address: ", bg="white", font=("Helvatica", 10, "bold"))

    address\_label.place(x=20, y=505)

    address\_text = Text(insert, width=23, font=("Helvatica", 10), relief="groove", bd=2, height=2, wrap="word")

    address\_text.place(x=150, y=505)

    achedemic\_label = Label(insert, text="Achedemic Record: ", bg="white", font=("Helvatica", 10, "bold"))

    achedemic\_label.place(x=20, y=560)

    l1 = Label(insert, text="Exam ", bg="white", font=("Helvatica", 10)).place(x=20, y=580)

    l2 = Label(insert, text="Year of Passing", bg="white", font=("Helvatica", 10)).place(x=80, y=580)

    l3 = Label(insert, text="Percentage(%)", bg="white", font=("Helvatica", 10)).place(x=200,y=580)

    l4 = Label(insert, text="SSC", bg="white", font=("Helvatica", 10)).place(x=20, y=620)

    l5 = Label(insert, text="HSC", bg="white", font=("Helvatica", 10)).place(x=20, y=660)

    tf1 = Entry(insert, width=10, font=("Helvatica", 10), relief="groove", bd=2)

    tf1.place(x=80,y=620)

    tf2 = Entry(insert, width=10, font=("Helvatica", 10), relief="groove", bd=2)

    tf2.place(x=200,y=620)

    tf3 = Entry(insert, width=10, font=("Helvatica", 10), relief="groove", bd=2)

    tf3.place(x=80,y=660)

    tf4 = Entry(insert, width=10, font=("Helvatica", 10), relief="groove", bd=2)

    tf4.place(x=200,y=660)

    sem\_label = Label(insert, text="Semesters", bg="white", font=("Helvatica", 10, "bold")).place(x=350, y=400)

    sem1\_label = Label(insert, text="Semester 1", bg="white", font=("Helvatica", 10)).place(x=350, y=440)

    sem2\_label = Label(insert, text="Semester 2", bg="white", font=("Helvatica", 10)).place(x=350, y=460)

    sem3\_label = Label(insert, text="Semester 3", bg="white", font=("Helvatica", 10)).place(x=350, y=480)

    sem4\_label = Label(insert, text="Semester 4", bg="white", font=("Helvatica", 10)).place(x=350, y=500)

    sem5\_label = Label(insert, text="Semester 5", bg="white", font=("Helvatica", 10)).place(x=350, y=520)

    sem6\_label = Label(insert, text="Semester 6", bg="white", font=("Helvatica", 10)).place(x=350, y=540)

    year\_label = Label(insert, text="Year, Month", bg="white", font=("Helvatica", 10, "bold")).place(x=460, y=400)

    year1\_entry = Entry(insert, text="M1", bg="white", font=("Helvatica", 10), width=9, relief="groove", bd=2)

    year1\_entry.place(x=470, y=440)

    year2\_entry = Entry(insert, text="M2", bg="white", font=("Helvatica", 10), width=9, relief="groove", bd=2)

    year2\_entry.place(x=470, y=460)

    year3\_entry = Entry(insert, text="M3", bg="white", font=("Helvatica", 10), width=9, relief="groove", bd=2)

    year3\_entry.place(x=470, y=480)

    year4\_entry = Entry(insert, text="M4", bg="white", font=("Helvatica", 10), width=9, relief="groove", bd=2)

    year4\_entry.place(x=470, y=500)

    year5\_entry = Entry(insert, text="M5", bg="white", font=("Helvatica", 10), width=9, relief="groove", bd=2)

    year5\_entry.place(x=470, y=520)

    year6\_entry = Entry(insert, text="M6", bg="white", font=("Helvatica", 10), width=9, relief="groove", bd=2)

    year6\_entry.place(x=470, y=540)

    percent\_label = Label(insert, text="%", bg="white", font=("Helvatica", 10, "bold")).place(x=570, y=400)

    percentage1\_entry = Entry(insert, text="Percentage 1", bg="white", font=("Helvatica", 10), width=7, relief="groove", bd=2)

    percentage1\_entry.place(x=560, y=440)

    percentage2\_entry = Entry(insert, text="Percentage 2", bg="white", font=("Helvatica", 10), width=7, relief="groove", bd=2)

    percentage2\_entry.place(x=560, y=460)

    percentage3\_entry = Entry(insert, text="Percentage 3", bg="white", font=("Helvatica", 10), width=7, relief="groove", bd=2)

    percentage3\_entry.place(x=560, y=480)

    percentage4\_entry = Entry(insert, text="Percentage 4", bg="white", font=("Helvatica", 10), width=7, relief="groove", bd=2)

    percentage4\_entry.place(x=560, y=500)

    percentage5\_entry = Entry(insert, text="Percentage 5", bg="white", font=("Helvatica", 10), width=7, relief="groove", bd=2)

    percentage5\_entry.place(x=560, y=520)

    percentage6\_entry = Entry(insert, text="Percentage 6", bg="white", font=("Helvatica", 10), width=7, relief="groove", bd=2)

    percentage6\_entry.place(x=560, y=540)

    submit\_record = Button(insert, text="Submit", command=storeData, width=10, bg="blue", fg="white", font=("Helvatica", 10, "bold"))

    submit\_record.place(x=400,y=600)

    clear\_record = Button(insert, text="Clear", command=clearData, width=10, bg="blue", fg="white", font=("Helvatica", 10, "bold"))

    clear\_record.place(x=500,y=600)

    insert.mainloop()

def delete():

    """

    This function creates a frame to delete a record from the database and asks for a reason.

    """

    def delete\_student():

        database()

        reason = delete\_text.get('1.0', 'end-1c')

        if not reason:

            tkMessageBox.showinfo("Alert", "Please fill the mandatory field")

            return False

        selected\_items = tree.selection()  # Get the selected items (rows)

        for item in selected\_items:

            sap\_id = tree.item(item)['values'][3]  # Retrieve the Sap Id (index 3) from the item

            cur.execute("DELETE FROM student\_data WHERE sap\_no=%s", (sap\_id,))

        con.commit()

        display()

        tkMessageBox.showinfo("", f"{sap\_id} Deleted")

    def delete\_reason():

        global delete\_text

        delete\_reason = Tk()

        delete\_reason.geometry("300x200")

        delete\_reason.resizable(False, False)

        delete\_reason.configure(background="#1338BE")

        delete\_label = Label(delete\_reason, text="Please Enter A Reason", font=("Helvatica", 11, "bold"), bg="#1338BE", fg="white").place(x=70, y=30)

        delete\_text = Text(delete\_reason, width=30, height=3, font=("Helvatica", 10), wrap="word",  relief="solid")

        delete\_text.place(x=50,y=70)

        delete\_msg = delete\_text.get("1.0", "end-1c")

        confirm\_button = Button(delete\_reason, text="Confirm", width=10, height=1, font=("Helvatica", 10), bg="white", fg="green", command=delete\_student).place(x=50, y=150)

        cancel\_button = Button(delete\_reason, text="Cancel", width=10, height=1, font=("Helvatica", 10), bg="white", fg="red", command=delete\_reason.destroy).place(x=160, y=150)

        delete\_reason.mainloop()

    if not tree.selection():

        tkMessageBox.showinfo("","Please Select a Student Record to Delete")

    else:

        delete\_reason()

def search():

    """

    This function creates a search window with a label, text box, search button, and close button, and

    performs a search query on a database based on the keywords entered by the user.

    """

    search\_window = Tk()

    global SEARCH

    SEARCH = StringVar()

    # create a new window

    search\_window.title("Search Data")

    search\_window.resizable(width=False, height=False)

    search\_window.configure(background="#1338BE")

    # create a label and text box for the user to enter search keywords

    search\_label = Label(search\_window, text="Enter keywords to search:", bg="#1338BE", fg="white")

    search\_label.pack(side=LEFT, padx=5, pady=5)

    search\_entry = Entry(search\_window, textvariable=SEARCH)

    search\_entry.pack(side=LEFT, padx=5, pady=5)

    def advanced\_search():

        """

        The function "advanced\_search" is used to retrive data such as

        Marks Range or semester or HSC or SSC results also used to retrive results of semesters

        eg:

        [] > [] or [] < []

        [] >= [] or [] <= []

        """

        tkMessageBox.showinfo("Alert", "This feature is under progress!")

    # create a function to search for the keywords entered by the user

    def search\_keyword():

        try:

            database()

            #checking search text is empty or not

            if SEARCH.get() != "":

                #clearing current display data

                tree.delete(\*tree.get\_children())

                #select query with where clause

                cur.execute("""SELECT sr\_no, stu\_name, roll\_no, sap\_no, branch, present\_year FROM student\_data WHERE

                                sr\_no = %s OR

                                stu\_name LIKE %s OR

                                roll\_no LIKE %s OR

                                sap\_no LIKE %s OR

                                branch LIKE %s OR

                                present\_year LIKE %s""",

                                (SEARCH.get(), '%' + SEARCH.get() + '%', '%' + SEARCH.get() + '%', '%' + SEARCH.get() + '%', '%' + SEARCH.get() + '%', '%' + SEARCH.get() + '%'))

                #fetch all matching records

                fetch = cur.fetchall()

                #loop for displaying all records into GUI

                if len(fetch) == 0 or SEARCH.get() == "" or SEARCH.get() is None:

                    tkMessageBox.showinfo("Alert","Record Not Found")

                else:

                    for data in fetch:

                        tree.insert('', 'end', values=data)

                        # set foreground color for inserted items

                        if SEARCH.get().casefold() in data[1].casefold() or data[2].casefold() or data[3].casefold() or data[4].casefold() or data[5].casefold():

                            tree.tag\_configure('found', foreground='red')

                            tree.item(tree.get\_children()[-1], tags=('found',))

        except Exception as e:

            print("Database error => ", e)

        finally:

            if con:

                con.close()

                print("Database closed successfully")

                print()

    # create a button to perform the search

    search\_button = Button(search\_window, text="Search", command=search\_keyword, bg="white")

    search\_button.pack(side=LEFT, padx=5, pady=5)

    # create a button to close the search window

    close\_button = Button(search\_window, text="Close", command=search\_window.destroy, bg="white")

    close\_button.pack(side=LEFT, padx=5, pady=5)

    adv\_button = Button(search\_window, text="Advanced Search", command=advanced\_search, bg="white")

    adv\_button.pack(side=LEFT, padx=5, pady=5)

    search\_window.mainloop()

def sort():

    '''

    In this function i have created a login window for admin

    which have input fields such as username and password as well as submit button to submit the form

    '''

    tkMessageBox.showinfo("Alert","This module is not been completed yet")

def updateDisplay(event):

    database()

    update = Tk()

    window\_width = 700

    window\_height = 700

    screen\_width = update.winfo\_screenwidth()

    screen\_height = update.winfo\_screenheight()

    x = (screen\_width/2) - (window\_width/2)

    y = (screen\_height/2) - (window\_height/2)

    update.geometry('%dx%d+%d+%d' % (window\_width, window\_height, x, y-50))

    update.resizable(width=False, height=False)

    update.configure(bg='white')

    item = tree.selection()[0]

    sap\_no = tree.item(item, "values")[3]

    cur.execute("SELECT \* FROM student\_data WHERE sap\_no=%s", (sap\_no,))

    record = cur.fetchone()

    def revert\_info():

        database()

        cur.execute("SELECT \* FROM student\_data WHERE sap\_no=%s", (sap\_no,))

        record = cur.fetchone()

        if record:

            # Clear the entry fields

            contact1\_entry.delete(0, END)

            contact2\_entry.delete(0, END)

            email1\_entry.delete(0, END)

            email2\_entry.delete(0, END)

            address\_entry.delete("1.0", END)

            y1.delete(0,END)

            y2.delete(0,END)

            y3.delete(0,END)

            y4.delete(0,END)

            y5.delete(0,END)

            y6.delete(0,END)

            p1.delete(0,END)

            p2.delete(0,END)

            p3.delete(0,END)

            p4.delete(0,END)

            p5.delete(0,END)

            p6.delete(0,END)

            # Restore the original values

            contact1\_entry.insert(END, record[7])

            contact2\_entry.insert(END, record[8])

            email1\_entry.insert(END, record[9])

            email2\_entry.insert(END, record[10])

            address\_entry.insert(END, record[12])

            y1.insert(END, record[18])

            y2.insert(END, record[19])

            y3.insert(END, record[20])

            y4.insert(END, record[21])

            y5.insert(END, record[22])

            y6.insert(END, record[23])

            p1.insert(END, record[24])

            p2.insert(END, record[25])

            p3.insert(END, record[26])

            p4.insert(END, record[27])

            p5.insert(END, record[28])

            p6.insert(END, record[29])

            tkMessageBox.showinfo("Success", "Reverted to original values")

        con.close()

    def update\_info():

        database()

        cur.execute("SELECT \* FROM student\_data WHERE sap\_no=%s", (sap\_no,))

        record = cur.fetchone()

        if record:

            new\_address = address\_entry.get("1.0", "end-1c")

            new\_contact1 = contact1\_entry.get()

            new\_contact2 = contact2\_entry.get()

            new\_email1 = email1\_entry.get()

            new\_email2 = email2\_entry.get()

            new\_y1 = y1.get()

            new\_y2 = y2.get()

            new\_y3 = y3.get()

            new\_y4 = y4.get()

            new\_y5 = y5.get()

            new\_y6 = y6.get()

            new\_p1 = p1.get()

            new\_p2 = p2.get()

            new\_p3 = p3.get()

            new\_p4 = p4.get()

            new\_p5 = p5.get()

            new\_p6 = p6.get()

            cur.execute("UPDATE student\_data SET address=%s, contact1=%s, contact2=%s, email1=%s, email2=%s, sem1=%s, sem2=%s, sem3=%s, sem4=%s, sem5=%s, sem6=%s, sem1\_percent=%s, sem2\_percent=%s, sem3\_percent=%s, sem4\_percent=%s, sem5\_percent=%s, sem6\_percent=%s WHERE sap\_no=%s",

                        (new\_address, new\_contact1, new\_contact2, new\_email1, new\_email2, new\_y1, new\_y2, new\_y3, new\_y4, new\_y5, new\_y6, new\_p1, new\_p2, new\_p3, new\_p4, new\_p5, new\_p6, sap\_no))

            con.commit()

            tkMessageBox.showinfo("Success", "Record Updated Successfully")

    if record:

        update.title(f"{record[2]} Record Details")

        # Add labels to display the record details

        roll\_label = Label(update, text="Roll No", font=("Helvatica", 10, "bold"), bg="white")

        roll\_label.place(x=370, y=50)

        Label(update, text=record[2],  bg="white", font=("Helvatica", 10)).place(x=430,y=50)

        sap\_label = Label(update, text="Sap No", font=("Helvatica", 10, "bold"), bg="white")

        sap\_label.place(x=150, y=50)

        Label(update, text=record[1],font=("Helvatica", 10),  bg="white").place(x=210,y=50)

        year\_of\_admission\_label = Label(update, text="Year of Admission: ", bg="white", font=("Helvatica", 10, "bold"))

        year\_of\_admission\_label.place(x=20,y=120)

        Label(update, text=record[3],font=("Helvatica", 10),  bg="white").place(x=150,y=120)

        type\_of\_admission\_label = Label(update, text="Type of Admission: ", bg="white", font=("Helvatica", 10, "bold"))

        type\_of\_admission\_label.place(x=20,y=155)

        Label(update, text=record[4], font=("Helvatica", 10),  bg="white").place(x=150,y=155)

        branch\_label = Label(update, text="Branch: ", bg="white", font=("Helvatica", 10, "bold"))

        branch\_label.place(x=20,y=210)

        Label(update, text=record[5], font=("Helvatica", 10),  bg="white").place(x=150,y=210)

        name\_label = Label(update, text="Student Name: ", bg="white", font=("Helvatica", 10, "bold"))

        name\_label.place(x=20, y=245)

        Label(update, text=record[6], font=("Helvatica", 10),  bg="white").place(x=150, y=245)

        contact1\_entry = Entry(update, width=20, font=("Helvetica", 10),relief="groove", bd=2)

        contact1\_entry.place(x=150, y=290)

        contact1\_entry.insert(END, record[7])

        contact2\_entry = Entry(update, width=20, font=("Helvetica", 10), relief="groove", bd=2)

        contact2\_entry.place(x=150, y=315)

        contact2\_entry.insert(END, record[8])

        email1\_entry = Entry(update, width=20, font=("Helvetica", 10), relief="groove", bd=2)

        email1\_entry.place(x=150, y=360)

        email1\_entry.insert(END, record[9])

        email2\_entry = Entry(update, width=20, font=("Helvetica", 10), relief="groove", bd=2)

        email2\_entry.place(x=150, y=390)

        email2\_entry.insert(END, record[10])

        date\_of\_birth\_label = Label(update, text="Date of Birth:", bg="white", font=("Helvatica", 10, "bold"))

        date\_of\_birth\_label.place(x=20,y=445)

        Label(update, text=record[11], font=("Helvatica", 10),  bg="white").place(x=150, y=445)

        address\_label = Label(update, text="Address:", bg="white", font=("Helvatica", 10, "bold"))

        address\_label.place(x=20,y=505)

        address\_entry = Text(update, width=20, font=("Helvetica", 10), height=2, wrap="word", relief="groove", bd=2)

        address\_entry.place(x=150, y=505)

        address\_entry.insert(END, record[12])

        achedemic\_label = Label(update, text="Achedemic Record: ", bg="white", font=("Helvatica", 10, "bold"))

        achedemic\_label.place(x=20, y=560)

        l1 = Label(update, text="Exam ", bg="white", font=("Helvatica", 10)).place(x=20, y=580)

        l2 = Label(update, text="Year of Passing", bg="white", font=("Helvatica", 10)).place(x=80, y=580)

        l3 = Label(update, text="Percentage(%)", bg="white", font=("Helvatica", 10)).place(x=200,y=580)

        l4 = Label(update, text="SSC", bg="white", font=("Helvatica", 10)).place(x=20, y=620)

        l5 = Label(update, text="HSC", bg="white", font=("Helvatica", 10)).place(x=20, y=660)

        Label(update, text=record[13], font=("Helvatica", 10),  bg="white").place(x=80,y=620)

        Label(update, text=record[14], font=("Helvatica", 10),  bg="white").place(x=200,y=620)

        Label(update, text=record[15], font=("Helvatica", 10),  bg="white").place(x=80,y=660)

        Label(update, text=record[16], font=("Helvatica", 10),  bg="white").place(x=200,y=660)

        sem\_label = Label(update, text="Semesters", bg="white", font=("Helvatica", 10, "bold")).place(x=400, y=400)

        year\_label = Label(update, text="Year, Month", bg="white", font=("Helvatica", 10, "bold")).place(x=500, y=400)

        percent\_label = Label(update, text="%", bg="white", font=("Helvatica", 10, "bold")).place(x=580, y=400)

        sem1\_label = Label(update, text="Semester 1", bg="white", font=("Helvatica", 10)).place(x=400, y=440)

        sem2\_label = Label(update, text="Semester 2", bg="white", font=("Helvatica", 10)).place(x=400, y=460)

        sem3\_label = Label(update, text="Semester 3", bg="white", font=("Helvatica", 10)).place(x=400, y=480)

        sem4\_label = Label(update, text="Semester 4", bg="white", font=("Helvatica", 10)).place(x=400, y=500)

        sem5\_label = Label(update, text="Semester 5", bg="white", font=("Helvatica", 10)).place(x=400, y=520)

        sem6\_label = Label(update, text="Semester 6", bg="white", font=("Helvatica", 10)).place(x=400, y=540)

        y1 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        y1.place(x=500, y=442)

        y1.insert(END, record[18])

        y2 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        y2.place(x=500, y=462)

        y2.insert(END, record[19])

        y3 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        y3.place(x=500, y=482)

        y3.insert(END, record[20])

        y4 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        y4.place(x=500, y=502)

        y4.insert(END, record[21])

        y5 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        y5.place(x=500, y=522)

        y5.insert(END, record[22])

        y6 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        y6.place(x=500, y=542)

        y6.insert(END, record[23])

        p1 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        p1.place(x=585, y=442)

        p1.insert(END, record[24])

        p2 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        p2.place(x=585, y=462)

        p2.insert(END, record[25])

        p3 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        p3.place(x=585, y=482)

        p3.insert(END, record[26])

        p4 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        p4.place(x=585, y=502)

        p4.insert(END, record[27])

        p5 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        p5.place(x=585, y=522)

        p5.insert(END, record[28])

        p6 = Entry(update, font=("Helvatica", 10),  bg="white", width=8, relief="groove", bd=2)

        p6.place(x=585, y=542)

        p6.insert(END, record[29])

        update\_record = Button(update, text="Update", width=10, bg="blue", fg="white", font=("Helvatica", 10, "bold"), command=update\_info)

        update\_record.place(x=400,y=600)

        revert\_record = Button(update, text="Revert", width=10, bg="blue", fg="white", font=("Helvatica", 10, "bold"), command=revert\_info)

        revert\_record.place(x=500,y=600)

        image\_data = record[30]

        # Create an image from the retrieved data

        image = Image.open(io.BytesIO(image\_data))

        # Calculate the aspect ratio of the image

        image\_width, image\_height = image.size

        aspect\_ratio = image\_width / image\_height

        # Determine the dimensions for cropping

        canvas\_width = 150

        canvas\_height = 150

        if aspect\_ratio > 1:

            # Landscape image, crop the width

            crop\_width = image\_height \* aspect\_ratio

            crop\_height = image\_height

        else:

            # Portrait image or square image, crop the height

            crop\_width = image\_width

            crop\_height = image\_width / aspect\_ratio

        crop\_x = (image\_width - crop\_width) / 2

        crop\_y = (image\_height - crop\_height) / 2

        # Crop the image to desired dimensions

        cropped\_image = image.crop((crop\_x, crop\_y, crop\_x + crop\_width, crop\_y + crop\_height))

        # Resize the cropped image to fit the canvas

        resized\_image = cropped\_image.resize((canvas\_width, canvas\_height), Image.ANTIALIAS)

        # Display the resized image in a canvas

        image\_upload = Canvas(update, width=canvas\_width, height=canvas\_height)

        image\_upload.place(x=500, y=120)

        image\_object = ImageTk.PhotoImage(resized\_image)

        image\_upload.image = image\_object  # Keep a reference to the image object

        image\_upload.create\_image(int(canvas\_width / 2), int(canvas\_height / 2), image=image\_object)

        image\_upload.update()

    update.mainloop()

def update():

    if not tree.selection():

        tkMessageBox.showinfo("","Please Select a Student Record to Update")

    else:

        selected\_item = tree.selection()

        sap\_no = tree.item(selected\_item)['values'][3]

        # Perform the detail display of the particular SAP number

        updateDisplay(sap\_no)

def help\_option():

    help\_window = Tk()

    help\_window.title('Help')

    help\_window.geometry("500x500")

    # Define the font style

    title\_font = Font(family="Helvetica", size=11, weight="bold")

    text\_font = Font(family="Helvetica", size=9)

    # Insert label and description

    insert\_label = Label(help\_window, text="Insert", font=title\_font, anchor="w", justify=LEFT)

    insert\_label.pack()

    insert\_description = Label(help\_window, text="The insert feature allows users to add new student data into the record management system. It prompts the user to input information such as the student's name, age, grade, and other relevant details. The system then stores this data, enabling easy retrieval and management of student records", font=text\_font, anchor="w", justify=LEFT, wraplength=480)

    insert\_description.pack()

    # Delete label and description

    delete\_label = Label(help\_window, text="Delete", font=title\_font, anchor="w", justify=LEFT)

    delete\_label.pack()

    delete\_description = Label(help\_window, text=" The delete feature enables users to remove specific student data from the record management system. Users can select a student's record to delete based on unique identifiers such as student ID or name. Deleting a record ensures that outdated or erroneous information is removed from the system, maintaining data accuracy.", font=text\_font, anchor="w", justify=LEFT, wraplength=480)

    delete\_description.pack()

    # Search label and description

    search\_label = Label(help\_window, text="Search", font=title\_font, anchor="w", justify=LEFT)

    search\_label.pack()

    search\_description = Label(help\_window, text="The search feature allows users to find specific student data within the record management system. Users can enter search criteria such as student name, ID, or any other relevant information. The system then retrieves and displays the corresponding student records, making it efficient to locate specific information quickly.", font=text\_font, anchor="w", justify=LEFT, wraplength=480)

    search\_description.pack()

    # Sort label and description

    sort\_label = Label(help\_window, text="Sort", font=title\_font, anchor="w", justify=LEFT)

    sort\_label.pack()

    sort\_description = Label(help\_window, text="The sort feature organizes the student records within the system based on a specific criterion. Users can choose to sort the records alphabetically, numerically, or based on other relevant attributes such as grades or enrollment dates. Sorting helps in presenting the student records in a structured and easily navigable manner.", font=text\_font, anchor="w", justify=LEFT, wraplength=480)

    sort\_description.pack()

    # Display label and description

    display\_label = Label(help\_window, text="Display", font=title\_font, anchor="w", justify=LEFT)

    display\_label.pack()

    display\_description = Label(help\_window, text="The display feature presents all the student records stored in the system. It provides an overview of all the information available, such as student names, ages, grades, and any other relevant data. Displaying the records allows users to review the complete dataset, facilitating analysis, decision-making, or generating reports as needed.", font=text\_font, anchor="w", justify=LEFT, wraplength=480)

    display\_description.pack()

    help\_window.mainloop()

def aboutUs():

    about\_window = Tk()

    about\_window.title('About Us')

    about\_window.geometry("500x300")

    name\_label = Label(about\_window, text="Jainam Barbhaya", font=("Varenda", 15, "bold"))

    name\_label.place(x=20, y=20)

    message = """

    Email       : jainambarbhaya1509@gmail.com \n

    Contact     : +91 9702288992\n

    Department  : Information Technology\n

    SAP No      : 57498210010\n

    Roll No     : T010\n

    Semester    : 4\n

    """

    info = Message(about\_window, text=message, width=400, font=("Varenda", 8, "bold"))

    info.place(x=0, y=50)

    # Canvas for image

    img\_canvas = Canvas(about\_window, height=150, width=150)

    img\_canvas.place(x=300, y=20)

    # Load and crop the image

    image = Image.open("C:\\Users\\AWI-Guest\\Desktop\\ActivateWork\\Projects\\Zoom\\Images\\image.jpeg")

    image = image.resize((150, 150), Image.ANTIALIAS)  # Resize the image to fit the canvas

    cropped\_image = ImageTk.PhotoImage(image)

    # Display the cropped image on the canvas

    img\_canvas.create\_image(0, 0, anchor="nw", image=cropped\_image)

    about\_window.mainloop()

def window():

    '''

    In this function i have created main window after admin logins

    this module contains clickable labels such as insert, update, delete, sort, search & display

    on clicking the label, it will be redirected to a new screen where the admin can perform the function

    This function uses TreeView to display the data in tabular format

    '''

    # creating a window

    global window

    window = Tk()

    window.title("Student Bio-Data RMS")

    # setting window on the center of the screen

    window\_width = 1000

    window\_height = 500

    screen\_width = window.winfo\_screenwidth()

    screen\_height = window.winfo\_screenheight()

    x = (screen\_width/2) - (window\_width/2)

    y = (screen\_height/2) - (window\_height/2)

    window.geometry('%dx%d+%d+%d' % (window\_width, window\_height, x, y))

    window.resizable(width=False, height=False)

    window.configure(bg='white')

    # setting canvas taskbar

    canvas = Canvas(window, width=1000, height=30, bg="blue")

    canvas.pack()

    #setting scrollbar

    global tree

    scrollbary = Scrollbar(window, orient=VERTICAL)

    tree = ttk.Treeview(window, columns=("Sr No.", "Full Name", "Roll No.", "Sap Id", "Department","Year"),selectmode="extended", height=21, yscrollcommand=scrollbary.set)

    scrollbary.config(command=tree.yview)

    scrollbary.pack(side=RIGHT, fill=Y)

    tree.heading('Sr No.', text="Sr No.", anchor=W)

    tree.heading('Full Name', text="Full Name", anchor=W)

    tree.heading('Roll No.', text="Roll No.", anchor=W)

    tree.heading('Sap Id', text="Sap Id", anchor=W)

    tree.heading('Department', text="Department", anchor=W)

    tree.heading('Year', text="Year", anchor=W)

    #setting width of the columns

    tree.column('#0', stretch=NO, minwidth=0, width=0)

    tree.column('#1', stretch=NO, minwidth=0, width=140)

    tree.column('#2', stretch=NO, minwidth=0, width=140)

    tree.column('#3', stretch=NO, minwidth=0, width=140)

    tree.column('#4', stretch=NO, minwidth=0, width=140)

    tree.place(x=10,y=40)

    # adding CRUD lables and their function in list and iterating them

    # display()

    add\_label = Label(window,text="Add", fg="white", bg="blue", cursor="hand2", font=("Helvatica", 8, "bold"))

    add\_label.place(x=20, y=7)

    add\_label.bind("<Button-1>", lambda event: insert())

    delete\_label = Label(window,text="Delete", fg="white", bg="blue", cursor="hand2", font=("Helvatica", 8, "bold"))

    delete\_label.place(x=70, y=7)

    delete\_label.bind("<Button-1>", lambda event: delete())

    search\_label = Label(window,text="Search", fg="white", bg="blue", cursor="hand2", font=("Helvatica", 8, "bold"))

    search\_label.place(x=130, y=7)

    search\_label.bind("<Button-1>", lambda event: search())

    display\_label = Label(window,text="Display", fg="white", bg="blue", cursor="hand2", font=("Helvatica", 8, "bold"))

    display\_label.place(x=190, y=7)

    display\_label.bind("<Button-1>", lambda event: display())

    update\_label = Label(window,text="Update", fg="white", bg="blue", cursor="hand2", font=("Helvatica", 8, "bold"))

    update\_label.place(x=250, y=7)

    update\_label.bind("<Button-1>", lambda event: update())

    sort\_label = Label(window,text="Sort", fg="white", bg="blue", cursor="hand2", font=("Helvatica", 8, "bold"))

    sort\_label.place(x=310, y=7)

    sort\_label.bind("<Button-1>", lambda event: sort())

    help\_label = Label(window,text="Help", fg="white", bg="blue", cursor="hand2", font=("Helvatica", 8, "bold"))

    help\_label.place(x=850, y=7)

    help\_label.bind("<Button-1>", lambda event: help\_option())

    about\_label = Label(window,text="About Us", fg="white", bg="blue", cursor="hand2", font=("Helvatica", 8, "bold"))

    about\_label.place(x=900, y=7)

    about\_label.bind("<Button-1>", lambda event: aboutUs())

    window.mainloop()

meeting\_info.py

import requests

import json

import mysql.connector

# API credentials

#api\_key = 'smdiXiQBQyGCi33lQj5Tg'

#api\_secret = 'GUb452sabnTh9BrFZlK4aiNHid6mZs3u'

# Generate JWT token

#import jwt

#import datetime

#token\_exp = datetime.datetime.utcnow() + datetime.timedelta(minutes=60)  # Token expires in 1 hour

#token\_payload = {

    #'iss': api\_key,

    #'exp': token\_exp

#}

#jwt\_token = jwt.encode(token\_payload, api\_secret, algorithm='HS256')

#jwt\_token = "eyJzdiI6IjAwMDAwMSIsImFsZyI6IkhTNTEyIiwidiI6IjIuMCIsImtpZCI6ImZlMTk5ZDRmLTYyZWUtNDYzOS05MmM5LTNmZmJhNjlkYzliZiJ9.eyJ2ZXIiOjksImF1aWQiOiIxOTg4N2QzNWJhMDQ3OThmZGJmOWM2OTBkZjJkZGQwMSIsImNvZGUiOiI0VVg1TFRTTTE2T3RaWGx4UTVYUmlPVHFfeU5FUExQQ3ciLCJpc3MiOiJ6bTpjaWQ6cnp5aHU0eFhRcmFwUDRubWtwa0xnIiwiZ25vIjowLCJ0eXBlIjowLCJ0aWQiOjAsImF1ZCI6Imh0dHBzOi8vb2F1dGguem9vbS51cyIsInVpZCI6IkVZczVFZk1iUmptTzlzTVRobFJTYkEiLCJuYmYiOjE3MDk2NjM2ODUsImV4cCI6MTcwOTY2NzI4NSwiaWF0IjoxNzA5NjYzNjg1LCJhaWQiOiJWX19QNXF3NlE5ZUVjbDJrZHljMmhRIn0.iFnyfLb31xJgESjMQPChtncFbLVI4I4bCtsxR19LdcfxEGySYRDZUmE5Pj\_4zdISavWYi-nqCdFINDAmXJRf2A"

# Meeting ID for which you want to get attendees

#my\_meeting\_id = '88321453048'

# API endpoint

#url = f'https://api.zoom.us/v2/report/users/banugus@gmail.com/meetings?from=2024-02-02&to=2024-03-03&page\_size=30'

# Headers

#headers = {

    #'Authorization': f'Bearer {jwt\_token}',

    #'Content-Type': 'application/json'

#}

# Make GET request

#response = requests.get(url, headers=headers)

# Check if request was successful

#if response.status\_code == 200:

    #meeting\_ids = response.json()

    #meeting\_id\_list = json.dumps(meeting\_ids, indent=4)

    #print(participant\_list)

with open('meeting\_id.json', 'r') as f:

    meeting\_id\_data = json.load(f)

    #print(data)

try:

    con = mysql.connector.connect(

        user = 'root',

        password = 'root@123',

        host = 'localhost',

        port = 3306,

        database = 'test\_database'

    )

    if con.is\_connected():

        print("Connected!")

except Exception as e:

    print("Cannot Connect! Try Again!")

cursor = con.cursor()

#cursor.execute("create table zoom\_participant\_list (Name varchar(255), User\_Email varchar(255), Join\_Time datetime, Leave\_Time datetime, Duration int(5))")

#print(meeting\_id\_data)

for item in meeting\_id\_data['meetings']:

    Meeting\_ID = item['id']

    Meeting\_Topic = item['topic']

    No\_of\_Participants = item['participants\_count']

    Total\_Minutes = item['total\_minutes']

    Duration = item['duration']

    print(Meeting\_ID)

    print(Meeting\_Topic)

    print(No\_of\_Participants)

    print(Total\_Minutes)

    print(Duration)

    #cursor.execute("CREATE TABLE test\_database.Meeting\_List (Meeting\_ID INT(255) PRIMARY KEY, Meeting\_Topic VARCHAR(255) NOT NULL, No\_of\_Participants INT(100), Total\_Minutes INT,Duration INT(100))")

    add\_meeting = """INSERT INTO test\_database.Meeting\_List

                      (Meeting\_ID,Meeting\_Topic,No\_of\_Participants,Total\_Minutes,Duration)

                      VALUES (%s, %s, %s, %s, %s)"""

    data\_meeting = (Meeting\_ID,Meeting\_Topic,No\_of\_Participants,Total\_Minutes,Duration)

    cursor.execute(add\_meeting, data\_meeting)

    con.commit()

    #print(type(Name))

#else:

    #print(f'Failed to retrieve attendees. Status code:{response.status\_code}, Error: {response.text}')

Meeting\_id.json

{

    "from": "2024-02-03",

    "to": "2024-03-03",

    "page\_count": 1,

    "page\_size": 30,

    "total\_records": 3,

    "next\_page\_token": "",

    "meetings": [

        {

            "uuid": "+ExHYbBuTbqlPNoHRri3jg==",

            "id": 88321453048,

            "host\_id": "EYs5EfMbRjmO9sMThlRSbA",

            "type": 2,

            "topic": "Banu Priya's Zoom Meeting",

            "user\_name": "Banu Priya",

            "user\_email": "banugus@gmail.com",

            "start\_time": "2024-03-01T01:18:41Z",

            "end\_time": "2024-03-01T02:28:58Z",

            "duration": 71,

            "total\_minutes": 145,

            "participants\_count": 4,

            "source": "Zoom"

        },

        {

            "uuid": "HJXCXBTCSYGUFKMABKLDoA==",

            "id": 87389331528,

            "host\_id": "EYs5EfMbRjmO9sMThlRSbA",

            "type": 2,

            "topic": "Banu Priya's Zoom Meeting",

            "user\_name": "Banu Priya",

            "user\_email": "banugus@gmail.com",

            "start\_time": "2024-03-01T02:44:54Z",

            "end\_time": "2024-03-01T03:37:55Z",

            "duration": 54,

            "total\_minutes": 70,

            "participants\_count": 3,

            "source": "Zoom"

        },

        {

            "uuid": "Zur4cAHiRn+h3WdALveLoQ==",

            "id": 81956206807,

            "host\_id": "EYs5EfMbRjmO9sMThlRSbA",

            "type": 2,

            "topic": "Banu Priya's Zoom Meeting",

            "user\_name": "Banu Priya",

            "user\_email": "banugus@gmail.com",

            "start\_time": "2024-03-01T04:42:27Z",

            "end\_time": "2024-03-01T05:27:58Z",

            "duration": 46,

            "total\_minutes": 46,

            "participants\_count": 2,

            "source": "Zoom"

        }

    ]

}

Admin.py

import tkinter as tk

from tkinter import \*

from tkinter import ttk

import mysql.connector

import pyttsx3

from PIL import Image, ImageTk

import main as m

from main import window

from tkinter import messagebox

engine = pyttsx3.init()

global admin

def verifyAdmin(username, password, is\_signup):

    '''

    In this function i have established a connection to my database "python\_mini\_project" and table "admin\_data"

    this function verifies weather the admin data is registered in database or not

    if data is there then success message is displayed else failure

    '''

    try:

        con = mysql.connector.connect(

        host="localhost",

        user="root",

        password="root@123",

        database="python\_mini\_project"

    )

        print("Database Status: Connection Established Successfully")

        if is\_signup:

            cursor = con.cursor()

            # Insert new admin data into the database

            #sql = ("INSERT INTO python\_mini\_project.admin\_data (username,password,admin\_name) VALUES (%s,%s,%s);", (username, password,"test"))

            sql1 = """INSERT INTO python\_mini\_project.admin\_data (username, password,admin\_name) VALUES (%s, %s,%s)"""

            val2 = (username, password,"Test")

            cursor.execute(sql1,val2)

            con.commit()

            #print("Signup Status: Signup Success")

            messagebox.showinfo("Signup Success", "Admin account created successfully!")

            #admin = tk.Tk()

            #admin.destroy()

            #admin.title("SignUp Panel")

            #status\_label = tk.Label(admin, text="Sign Up Successfull!", bg="green", fg="white")

            #status\_label.place(x=50,y=90)

            #admin.update()

            return True

        else:

            cursor = con.cursor()

            sql = "select \* from admin\_data where username=%s and password=%s"

            val = (username, password)

            cursor.execute(sql,val)

            result = cursor.fetchone()

            if result:

                print("Login Status: Login Success")

                return True

            else:

                print("Login Status: Login Failure")

                return False

    except mysql.connector.Error as err:

        print("Database Status: There was an error connecting to the database")

        return False

    finally:

        if con:

            con.close()

            print("Database Status: Connection Closed Successfully")

            #print()

def signup():

    username = signup\_username.get()

    password = signup\_password.get()

    if verifyAdmin(username, password, True):

        print("Signup Success", "Admin account created successfully!")

    else:

        print("Signup Failed", "Failed to create admin account.")

def open\_signup\_form():

    # Create a new Tkinter window for the sign-up form

    signup\_form = tk.Toplevel()

    signup\_form.title("Sign Up")

    signup\_form.geometry("300x200")

    # Create labels and entry widgets for username and password fields

    signup\_username\_label = tk.Label(signup\_form, text="Username1:")

    signup\_username\_label.pack()

    global signup\_username

    signup\_username = tk.Entry(signup\_form)

    signup\_username.pack()

    signup\_password\_label = tk.Label(signup\_form, text="Password1:")

    signup\_password\_label.pack()

    global signup\_password

    signup\_password = tk.Entry(signup\_form, show="\*")

    signup\_password.pack()

    # Create a button to trigger the sign-up process

    signup\_button = tk.Button(signup\_form, text="Sign Up", command=signup)

    signup\_button.pack()

def adminWindow():

    '''

    In this function i have created a login window for admin

    which have input fields such as username and password as well as submit button to submit the form

    '''

    # creating a new Tkinter window

    admin = tk.Tk()

    admin.title("Admin Panel")

    icon = PhotoImage(file = "C:\\Users\\AWI-Guest\\Desktop\\ActivateWork\\Projects\\Zoom\\Images\\Project-Icon.png")

    admin.iconphoto(False, icon)

    # setting window on the center of the screen

    window\_width = 1000

    window\_height = 500

    screen\_width = admin.winfo\_screenwidth()

    screen\_height = admin.winfo\_screenheight()

    x = (screen\_width/2) - (window\_width/2)

    y = (screen\_height/2) - (window\_height/2)

    admin.geometry('%dx%d+%d+%d' % (window\_width, window\_height, x, y))

    admin.resizable(width=False, height=False)

    admin.configure(bg='white')

    # setting canvas frame - start

    canvas = Canvas(admin, width=window\_width, height=40, bg="blue")

    canvas.create\_text(470, 20, text="Admin Login", fill="white", font=("Helvetica", 12, "bold"))

    canvas.pack()

    # setting canvas frame - end

    #canvas = Canvas(width=600, height=145)

    #canvas.configure(bg="white")

    #canvas.place(x=350,y=150)

    #image = PhotoImage(file="C:\\Users\\AWI-Guest\\Desktop\\ActivateWork\\Projects\\Zoom\\Images\\sbmp.png")

    #cropped\_image = image.subsample(2, 2)

    #canvas.create\_image(0, 0, anchor=NW, image=cropped\_image)

    global username\_text  # Declare as global to make it accessible to other functions

    global password\_text  # Declare as global to make it accessible to other functions

    # setting login fields --> Username & Password

    username\_label = tk.Label(admin, text="Username: ", bg="white",font=("Helvatica", 10, "bold"))

    username\_label.pack()

    username\_text = tk.Entry(admin, width=20, font=("Helvetica", 11), relief="groove", bd=2)

    username\_text.place(x=100,y=150)

    username\_label.place(x=20,y=150)

    password\_label = tk.Label(admin, text="Password: ", bg="white",font=("Helvatica", 10, "bold"))

    password\_label.pack()

    password\_text = tk.Entry(admin, show="\*", width=20, font=("Helvetica", 11), relief="groove", bd=2)

    password\_text.place(x=100,y=220)

    password\_label.place(x=20,y=220)

    def login():

        '''

        this function displays the label on screen to notify the user weather it is a success or a failure

        the result is retrned to "verifyAdmin()" and it checks wether the admin data exists or not

        '''

        result = verifyAdmin(username\_text.get(), password\_text.get(),False)

        if result:

            status\_label = tk.Label(admin, text="You have been successfully logged in!", bg="green", fg="white")

            status\_label.place(x=50,y=90)

            admin.update()

            engine.say("Login Successful")

            engine.runAndWait()

            admin.after(500, admin.destroy)

            m.window()

        else:

            status\_label = tk.Label(admin, text="There was an error logging you in!", bg="red", fg="white")

            status\_label.place(x=60,y=90)

            admin.update()

            engine.say("Login failed please enter correct id or password")

            engine.runAndWait()

    # adding a submit button

    submit\_button = tk.Button(admin, text="Submit", command=login, width=10, bg="blue", fg="white") # the command attribute will run the "login()" function

    submit\_button.place(x=140,y=285)

    #sign\_up\_button = tk.Button(admin, text="Sign Up",width=10, bg="blue", fg="white")

    #sign\_up\_button.place(x=40,y=350)

    signup\_button = tk.Button(admin, text="Sign Up", command=open\_signup\_form,width=10, bg="blue", fg="white")

    signup\_button.pack()

    signup\_button.place(x=40,y=285)

    admin.mainloop()

# Calling the function

adminWindow()

oauth.py

#!/usr/bin/env python

from flask import Flask, abort, request

from uuid import uuid4

import requests

import requests.auth

import urllib

CLIENT\_ID = "rzyhu4xXQrapP4nmkpkLg" # Fill this in with your client ID

CLIENT\_SECRET = "mJXor9TaWDYeBGEqcPMnm7hlpfbb6QMZ" # Fill this in with your client secret

REDIRECT\_URI = "http://127.0.0.1:65010/zoom\_callback"

app = Flask(\_\_name\_\_)

@app.route('/')

def homepage():

    text = '<a href="%s">Authenticate with Zoom</a>'

    return text % make\_authorization\_url()

def make\_authorization\_url():

    # Generate a random string for the state parameter

    # Save it for use later to prevent xsrf attacks

    params = {"client\_id": CLIENT\_ID,

              "response\_type": "code",

              "redirect\_uri": REDIRECT\_URI}

    url = "https://zoom.us/oauth/authorize?" + urllib.parse.urlencode(params)

    return url

@app.route('/zoom\_callback')

def zoom\_callback():

    error = request.args.get('error', '')

    if error:

        return "Error: " + error

    code = request.args.get('code')

    access\_token = get\_token(code)

    # Note: In most cases, you'll want to store the access token, in, say,

    # a session for use in other parts of your web app.

    return "Your user info is: %s" % get\_username(access\_token)

def get\_token(code):

    client\_auth = requests.auth.HTTPBasicAuth(CLIENT\_ID, CLIENT\_SECRET)

    post\_data = {"grant\_type": "authorization\_code",

                 "code": code,

                 "redirect\_uri": REDIRECT\_URI}

    response = requests.post("https://zoom.us/oauth/token",

                             auth=client\_auth,

                             data=post\_data)

    token\_json = response.json()

    print(token\_json)

    return token\_json["access\_token"]

def get\_username(access\_token):

    headers= {"Authorization": "bearer " + access\_token}

    response = requests.get("https://api.zoom.us/v2/users/me", headers=headers)

    me\_json = response.json()

    return me\_json

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, port=65010)