Documented code of automated google calendar

First we need to understand basic code structure of google calendar API that we used in our code:

Prerequisites:

1.Credentials File:

- Ensure you have a "credentials.json" file. If not, follow the instructions in the provided link (watch till 00:00 to 05:20(Link)) to generate this file. Save it in the same directory as your script.

2.Token File:

- The script uses a "token.json" file to store user access and refresh tokens. This file is automatically created during the authorization flow for first-time use.

Script Walkthrough

1.Import Statements

```
from __future__ import print_function
import datetime
import os.path
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
```

These statements import necessary modules for working with Google Calendar API and handling authentication.

2.Constants

```
SCOPES = ['https://www.googleapis.com/auth/calendar']
```

Defines the required API scopes for accessing Google Calendar.

3. Check and Load Credentials

```
    creds = None
    if os.path.exists('token.json'):
        creds =
Credentials.from_authorized_user_file('token.json', SCOPES)
    # ...
```

Checks if a "token.json" file exists and loads credentials if available.

4. User Authentication

```
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow =
InstalledAppFlow.from_client_secrets_file('credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())
```

If valid credentials are not available, it initiates the user authentication flow using the "credentials.json" file.

5. Google Calendar API Connection

```
    try:
        service = build('calendar', 'v3', credentials=creds)
        # ...
```

Builds a connection to the Google Calendar API using the authenticated credentials.

6. Event Creation

```
        # Call the Calendar API
        event = {
            'summary': 'Appointment',
            'location': 'Somewhere',
            'description': 'somewhere online',
            'start': {
                'dateTime': '2023-10-15T10:00:00+05:30',
                'timeZone': 'Asia/Kolkata'
            },
            'end': {
                'dateTime': '2023-10-15T10:25:00+05:30',
                'timeZone': 'Asia/Kolkata'
            },
            'recurrence': [
                'RRULE:FREQ=WEEKLY;UNTIL=20231212;BYDAY=MO,WE',
                'EXDATE:20231115T043000Z',
                'RDATE:20231117T043000Z',
            ],
            'attendees': [
                {'email': 'sample_1@iiitd.ac.in'},
                {'email': 'sample_2@iiitd.ac.in'}
            ],
        }

        # Insert the event
        event = service.events().insert(calendarId='primary',
body=event).execute()
        print(f'Recurring event created: {event["htmlLink"]}')
    except HttpError as error:
        print('An error occurred: %s' % error)


if __name__ == '__main__':
    lst = []
    main()
```

Defines a sample event and inserts it into the user's primary calendar. Handles exceptions, printing an error message if an HTTP error occurs.

RRULE (Recurrence Rule):

- The RRULE property defines the recurring rule for the event. It specifies how often the event should repeat and additional parameters like the end date of the recurrence.
        - FREQ=WEEKLY: Specifies that the event should occur weekly.
        - UNTIL=20231212: Sets the end date for the recurrence, in this case, December 12, 2023.
    - BYDAY=MO,WE: Specifies that the event should occur on Mondays and Wednesdays.

EXDATE (Exception Date):

- The EXDATE property is used to exclude specific instances from the recurring rule. It allows you to specify dates on which the recurring event should not occur.
- EXDATE:20231115T043000Z: Excludes the occurrence on November 15, 2023, at 04:30 AM.

RDATE (Recurrence Date):

- The RDATE property is used to specify additional dates for recurrence. It allows you to add specific dates to the recurring rule.
- RDATE:20231117T043000Z: Adds an additional recurrence on November 17, 2023, at 04:30 AM.

<u>About Major Data Structures/Data Sets used in our Algorithm:</u>

1. Slot: It is a dictionary which contains slot no. as key and its details as values. Each value pair has one or more time slots specified as a list of lists. Each inner list contains four elements:

1. Start time in the format "HH:MM:SS" (hours, minutes, seconds).
2. End time in the format "HH:MM:SS" (hours, minutes, seconds).
3. Duration of the time slot in minutes.
4. Day of the week for the class.

2. Main_list: It also contains the date of each slot but on a temporary basis.

3. first_year_data: It is a dictionary which has the name of the 1st year course as key and all its details are stored as a list.

4. slot_data: It is a list of lists which contains a list of the details of all the courses whose timings are same on both the days and falls in a regular slot.
For example:`['CSE', 'CSE506', 'Rest', 'Data Mining', 'DMG', '40', 'C215', 'Vikram Goyal', '2', '2', '', '', '', '','sample@iiitd.ac.in']`

5. not_in_slot: It is also a list of lists which contains the details of all the courses who does not falls under any regular slots(i.e Nil slot courses).

6. substitute: It is a dictionary which contains all details regarding adjusted time table in an academic calendar as it maps a date to a day means that on the date '20231120' (interpreted as 'YYYYMMDD' format where '2023' is the year, '11' is the month, and '20' is the day), the timetable of that date will be of day 'FRIDAY'.
Example: {'20231120': 'FRIDAY'}

7. holiday: This dictionary represents the list of all holidays in an academic calendar along with its dates by a mapping between dates and their corresponding days of the week. Each key in the dictionary is a date in the format 'YYYYMMDD' (Year, Month, Day), and the corresponding value is the day of the week for that date.
Example:
{'20231120': 'Monday', '20230729': ' Saturday', '20230830': ' Wednesday', '20230928': ' Thursday', '20230815': ' Tuesday', '20230907': ' Thursday', '20231002': ' Monday', '20231024': ' Tuesday', '20231112': ' Sunday', '20231127': ' Monday', '20231225': ' Monday'}

Now Here is explanation of our code:

About find_day_of_week()

```python
from datetime import datetime

def find_day_of_week(date_string):
    """
    Finds the day of the week for a given date string.

    Parameters:
    - date_string (str): A string representing a date in the
format "%Y%m%d".

    Returns:
    - day_name (str): The name of the day of the week for the
given date.
      Returns "Invalid date format" if the input date string is
not in the correct format.

    The function parses the input date string, calculates the
day of the week as an integer
    (where 0 represents Monday and 6 represents Sunday), and
returns the corresponding day name.

    Usage:
    day_name = find_day_of_week('20231117')
    if day_name != "Invalid date format":
        # Process the day_name variable
    else:
        # Handle the case where the date format is invalid

    """
    try:
        # Parse the input date string
        date = datetime.strptime(date_string, "%Y%m%d")
        # Get the day of the week as an integer (0 = Monday, 6 =
Sunday)
        day_of_week = date.weekday()

        # Define a list of weekday names
        weekdays = ["Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday"]

        # Get the weekday name for the given date
        day_name = weekdays[day_of_week]

        return day_name
    except ValueError:
        return "Invalid date format"
```

Finds the day of the week for a given date string.

Parameters:
- date_string (str): A string representing a date in the format "%Y%m%d".

Returns:
- day_name (str): The name of the day of the week for the given date.
  Returns "Invalid date format" if the input date string is not in the correct format.

The function parses the input date string, calculates the day of the week as an integer
(where 0 represents Monday and 6 represents Sunday), and returns the corresponding day name.

Usage:
day_name = find_day_of_week('20231117')
if day_name != "Invalid date format":

```
        # Process the day_name variable
    else:
        # Handle the case where the date format is invalid
```

About def read_calendar()

```
import csv

def read_calendar():
    """
    Reads calendar information from a CSV file based on specific
criteria.

    Returns:
    - read_calendar_error (int): An error indicator (0 if
successful, 1 if there is an error).

    The function reads the content of a CSV file, specifically
looking for two lines:
    1. "TT: Adjusted Days": Indicates a flag for adjusted days.
    2. "H:This includes Saturdays/Sundays and GH": Indicates a
flag for special inclusions.

    If both flags are found, the function extracts and returns
the relevant information.

    If the flags are not found or not in the correct form, an
error message is printed,
    and read_calendar_error is set to 1.

    Usage:
    read_calendar_error =
read_calendar('your_calendar_file.csv')
    if read_calendar_error == 0:
        # Process the calendar information
    else:
        # Handle the error

    """
    read_calendar_error = 0

    # Open the CSV file for the first pass to identify flags
    with open(filename, 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        tt_add_flag = 0
        this_include_flag = 0

        # Iterate through each line in the CSV file
        for i in csvreader:
            if i[0] == "TT: Adjusted Days":
                tt_add_flag = 1
            if i[0] == "H:This includes Saturdays/Sundays and
GH":
                this_include_flag = 1

    # Open the CSV file for the second pass to extract relevant
information
    with open(filename, 'r') as csvfile:
        line_number = 1
        reach = 0
        csvreader = csv.reader(csvfile)

        # If both flags are found, extract relevant information
        if tt_add_flag == 1 and this_include_flag == 1:
            for i in csvreader:
                if i[0] == "TT: Adjusted Days" and reach == 0:
                    reach = 1
                    continue
                elif i[0] == "H:This includes Saturdays/Sundays
and GH":
                    break
                if reach == 1:
```

```python
                    # Append relevant information to the 'all' list (assuming 'all' is defined globally)
                    all.append(i)
        else:
            print("Error: 'TT: Adjusted Days' or 'H:This includes Saturdays/Sundays and GH' not found or not in correct form.")
            read_calendar_error = 1

    return read_calendar_error
```

Reads calendar information from a CSV file based on specific criteria.

Returns:
- read_calendar_error (int): An error indicator (0 if successful, 1 if there is an error).

The function reads the content of a CSV file, specifically looking for two lines:
1. "TT: Adjusted Days": Indicates a flag for adjusted days.
2. "H:This includes Saturdays/Sundays and GH": Indicates a flag for special inclusions.

If both flags are found, the function extracts and returns the relevant information.

If the flags are not found or not in the correct form, an error message is printed,
and read_calendar_error is set to 1.

Usage:
```
read_calendar_error = read_calendar('your_calendar_file.csv')
if read_calendar_error == 0:
    # Process the calendar information
else:
    # Handle the error
```

About work_all_calandar_date()

```python
import re
from datetime import datetime

def work_all_calendar_data():
    """
    Process calendar data stored in the 'all' list and update 
the 'substitute' and 'holiday' dictionaries.

    Returns:
    - error_flag (int): An error indicator (0 if successful, 1 
if there is an error).

    The function iterates through the 'all' list, extracting 
information about adjusted days and holidays.
    It updates the 'substitute' dictionary with information 
about adjusted days
    and the 'holiday' dictionary with information about 
holidays.

    Usage:
    error_flag = work_all_calendar_data()
    if error_flag == 0:
        # Process the 'substitute' and 'holiday' dictionaries
    else:
        # Handle the error

    """
    error_flag = 0

    for i in all:
        if i[0] != "":
            a = i[0]
            if "-" not in a:
                error_flag = 1
                print("Error: Missing '-' in the input", a)
                print("Please correct it.")
                continue

            s = a.split("-")
            day = s[0].split()
            on_which = s[1].split()

            temp1 = day[0]
            match = re.match(r'(\d+)([a-zA-Z]+)', temp1)
            number = match.group(1)
            characters = match.group(2)
            final_number = ""

            if len(number) == 1:
                final_number = "0" + number
            else:
                final_number = number

            final_month = ""

            if day[1].strip() not in check_month:
                print(s)
                print("Error: Incorrect spelling of the month.
Please check.")
                error_flag = 1
                continue

            if day[1].strip() in month_captial_written.keys():
                final_month = month_captial_written[day[1]]
            elif day[1].strip() in month_small_written.keys():
                final_month = month_small_written[day[1]]

            create_day = str(year) + final_month +
str(final_number)

            if on_which[0].strip() not in check_days:
                print(s)
                print("Error: Incorrect spelling of the day.
Please check.")
                error_flag = 1
                continue

            on_what = ""

            if on_which[0].strip() in days.keys():
                on_what = days[on_which[0]]
            else:
                on_what = on_which[0]

            substitute[create_day] = on_what
            on_what1 = find_day_of_week(create_day)
            holiday[create_day] = on_what1

        length = len(i) - 1
        start = 1

        while start <= length:
            if i[start] == "":
                start += 1
                continue
            else:
                if "-" not in i[start] or "," not in i[start +
1]:
                    print("Error: Missing '-' or ',' in the
input", i)
                    print("Please correct it.")
                    error_flag = 1
                    start += 2
                    continue

                x = i[start].split("-")
                y = i[start + 1].split(",")
                if x[1].strip() not in check_month:
                    print(x, y)
                    print("Error: Incorrect spelling of the
month. Please check.")
                    error_flag = 1
                    start += 2
                    continue

                temp1 = str(x[0])
                final_number = ""

                if len(temp1) == 1:
                    final_number = "0" + temp1
                else:
                    final_number = temp1

                hol_day_number = ""

                if x[1].strip() in month_captial_written.keys():
                    hol_day_number = month_captial_written[x[1]]
                elif x[1].strip() in month_small_written.keys():
                    hol_day_number = month_small_written[x[1]]

                holiday_date = str(year) + hol_day_number +
final_number

                if y[1].strip() not in check_days:
                    print(y[1])
                    print(x, y)
                    print("Error: Incorrect spelling of the day.
Please check.")
                    error_flag = 1
                    start += 2
                    continue
                holiday_date = holiday_date
                holiday_day = ""

                if y[1].strip() in days.keys():
                    holiday_day = days[y[1]]
                else:
                    holiday_day = y[1]

                holiday[holiday_date] = holiday_day
                start += 2

    return error_flag
```

Process calendar data stored in the 'all' list and update the 'substitute' and 'holiday' dictionaries.

Returns:
- error_flag (int): An error indicator (0 if successful, 1 if there is an error).

The function iterates through the 'all' list, extracting information about adjusted days and holidays.
It updates the 'substitute' dictionary with information about adjusted days and the 'holiday' dictionary with information about holidays.

Usage:
error_flag = work_all_calendar_data()
if error_flag == 0:
    # Process the 'substitute' and 'holiday' dictionaries
else:
    # Handle the error

About takingInput()

```
import tkinter as tk
from tkcalendar import DateEntry
```

```
def taking_input(t):
    """
    Displays a date selector window using Tkinter and returns
the selected date.

    Parameters:
    - t (str): The message to be displayed above the date
selector.

    Returns:
    - selected_date (str): The selected date in the format
"%d/%m/%Y".

    The function creates a Tkinter window with a message, a
calendar widget, and a submit button.
    After the user selects a date and clicks the submit button,
the window is closed, and the selected date is returned.

    Usage:
    selected_date = taking_input('Select a Date')
    print(f"Selected Date: {selected_date}")

    """
    def get_selected_date():
        selected_date = cal.get_date()
        formatted_date = selected_date.strftime("%d/%m/%Y")
        result_var.set(formatted_date)
        window.destroy()  # Close the input window

    # Create the main window
    window = tk.Tk()
    window.title("Date Selector")

    # Create and place the label
    label_message = tk.Label(window, text=t)
    label_message.grid(row=0, column=0, padx=10, pady=10)
    # Create and place the calendar widget
    cal = DateEntry(window, width=12, background="darkblue",
foreground="white", borderwidth=2, year=2023, month=11, day=23)
    cal.grid(row=1, column=0, padx=10, pady=10)

    # Create a StringVar to store the result
    result_var = tk.StringVar()

    # Create and place the button
    button_submit = tk.Button(window, text="Submit",
command=get_selected_date)
    button_submit.grid(row=2, column=0, pady=10)

    # Run the main loop
    window.mainloop()

    # Return the result after the window is destroyed
    return result_var.get()
```

Displays a date selector window using Tkinter and returns the selected date.

Parameters:
- t (str): The message to be displayed above the date selector.

Returns:
- selected_date (str): The selected date in the format "%d/%m/%Y".

The function creates a Tkinter window with a message, a calendar widget, and a submit button.
After the user selects a date and clicks the submit button, the window is closed, and the selected date is returned.

Usage:
selected_date = taking_input('Select a Date')
print(f"Selected Date: {selected_date}")

About read_slotData_file()

```
import csv

def read_slotdata_file():
    """
    Reads slot data from a CSV file and populates the 'slot'
dictionary.

    Returns:
    - read_slot_data_error (int): An error indicator (0 if
successful, 1 if there is an error).

    The function prompts the user for the slot data file name,
reads the file,
    and populates the 'slot' dictionary based on the specified
structure.

    Usage:
    read_slot_data_error = read_slotdata_file()
    if read_slot_data_error == 0:
        # Process the 'slot' dictionary
    else:
        # Handle the error

    """
    read_slot_data_error = 0
    SlotFileName = input("Please provide the file name: ")

    with open(SlotFileName, 'r') as read_slot:
        slot_reader = csv.reader(read_slot)
        flag = 0

        for i in slot_reader:
            if i[0] != 'Slot No.' and flag == 0:
                print("Error: 'Slot No.' is missing in row 1,
column 1.")
                flag = 1
                read_slot_data_error = 1
                break

            if i[0] == 'Slot No.':
                flag = 1
                continue

            if len(i[0]) == 0:
                continue

            temp_list = []  # Create a new list for each row of
data

            if len(i[0]) == 0 or len(i[1]) == 0 or len(i[2]) ==
0 or len(i[3]) == 0:
                print("Error: Empty column in the slot data.
Data: ", i)
                read_slot_data_error = 1
                continue

            temp_list.append(i[1])
            temp_list.append(i[2])
            temp_list.append(90)
            temp_list.append(i[3])

            if i[3] not in capital_days:
                print("Error in the slotdata file. Data: ", i)
                read_slot_data_error = 1
                continue

            main_list.append(temp_list.copy())  # Use copy() to
avoid modifying the same list
            temp_list.clear()
            if len(i[4]) == 0 or len(i[5]) == 0 or len(i[6]) == 0:
                print("Error: Empty column in the slot data. Data: ", i)
                read_slot_data_error = 1
                continue

            temp_list.append(i[4])
            temp_list.append(i[5])
            temp_list.append(90)

            if i[6] not in capital_days:
                print("Error in the slotdata file. Data: ", i)
                read_slot_data_error = 1
                continue

            temp_list.append(i[6])
            main_list.append(temp_list.copy())  # Use copy() to avoid modifying the same list

            slot[i[0]] = main_list.copy()  # Use copy() to avoid modifying the same list
            temp_list.clear()
            main_list.clear()

    return read_slot_data_error
```

Reads slot data from a CSV file and populates the 'slot' dictionary.

Returns:
- read_slot_data_error (int): An error indicator (0 if successful, 1 if there is an error).

The function prompts the user for the slot data file name, reads the file, and populates the 'slot' dictionary based on the specified structure.

Usage:
read_slot_data_error = read_slotdata_file()
if read_slot_data_error == 0:
    # Process the 'slot' dictionary
else:
    # Handle the error

About reading_rest_file(filename)

```
import csv

def reading_rest_file(rest_file):
    """
    Reads rest data from a CSV file and categorizes it into
'not_in_slot' and 'slot_data' lists.

    Parameters:
    - rest_file (str): The name of the CSV file containing rest
data.

    Returns:
    - rest_file_error (int): An error indicator (0 if
successful, 1 if there is an error).

    The function reads the rest data from the specified CSV file
and categorizes it into two lists:
    1. 'not_in_slot': Contains rows with "Nil" in the eighth
column.
    2. 'slot_data': Contains rows with valid slot data.

    Usage:
    rest_file_error =
reading_rest_file('your_rest_data_file.csv')
    if rest_file_error == 0:
        # Process the 'not_in_slot' and 'slot_data' lists
    else:
        # Handle the error

    """
    filename = rest_file
    rest_file_error = 0

    with open(filename, 'r') as read_rest_file:
        rest_reader = csv.reader(read_rest_file)

        for i in rest_reader:
            if i[2] != "Rest" and i[2] != "II Year":
                continue
            else:
                if i[8] == "Nil":
                    if i[13] not in check_days:
                        print("Error: Incorrect spelling of the 
day in the rest file. Row with the error:", i)
                        rest_file_error = 1
                    not_in_slot.append(i)
                else:
                    slot_data.append(i)

    return rest_file_error
```

Reads rest data from a CSV file and categorizes it into 'not_in_slot' and 'slot_data' lists.

Parameters:
- rest_file (str): The name of the CSV file containing rest data.

Returns:
- rest_file_error (int): An error indicator (0 if successful, 1 if there is an error).

The function reads the rest data from the specified CSV file and categorizes it into two lists:

1. 'not_in_slot': Contains rows with "Nil" in the eighth column.
2. 'slot_data': Contains rows with valid slot data.

Usage:
rest_file_error = reading_rest_file('your_rest_data_file.csv')
if rest_file_error == 0:
    # Process the 'not_in_slot' and 'slot_data' lists
else:
    # Handle the error

About get_next_day_occurrence(`input_date, target_day`)

```
from datetime import datetime, timedelta

def get_next_day_occurrence(input_date, target_day):
    """
    Calculates the next occurrence date of a specified day of
the week after a given input date.

    Parameters:
    - input_date (str): Input date in the format "%d/%m/%Y".
    - target_day (str): Target day of the week (e.g., "Monday",
"Tuesday").

    Returns:
    - next_occurrence_date (str): The next occurrence date in
the format "%d/%m/%Y".

    The function takes an input date and a target day of the
week, calculates the difference
    between the target day and the current day, and determines
the next occurrence date.

    Usage:
    next_occurrence = get_next_day_occurrence('01/01/2023',
'Wednesday')
    print(f"The next Wednesday after 01/01/2023 is on:
{next_occurrence}")

    """
    # Convert the input date string to a datetime object
    date_obj = datetime.strptime(input_date, "%d/%m/%Y")

    # Define a mapping for days of the week
    days_mapping = {
        "Monday": 0,
        "Tuesday": 1,
        "Wednesday": 2,
        "Thursday": 3,
        "Friday": 4,
        "Saturday": 5,
        "Sunday": 6
    }

    # Get the numerical representation of the target day
    target_day_num = days_mapping.get(target_day.capitalize())
    if target_day_num is None:
        raise ValueError("Invalid day of the week")

    # Calculate the difference between the target day and the
current day
    day_difference = (target_day_num - date_obj.weekday() + 7) %
7

    # Calculate the next occurrence date
    next_occurrence_date = date_obj +
timedelta(days=day_difference)

    # Format and return the result
    return next_occurrence_date.strftime("%d/%m/%Y")
```

    Calculates the next occurrence date of a specified day of the week after a given input date.

    Parameters:
    - input_date (str): Input date in the format "%d/%m/%Y".
    - target_day (str): Target day of the week (e.g., "Monday", "Tuesday").

    Returns:
    - next_occurrence_date (str): The next occurrence date in the format "%d/%m/%Y".

    The function takes an input date and a target day of the week, calculates the difference
    between the target day and the current day, and determines the next occurrence date.

    Usage:
    next_occurrence = get_next_day_occurrence('01/01/2023', 'Wednesday')
    print(f"The next Wednesday after 01/01/2023 is on: {next_occurrence}")

About do_mail()

```
def do_mail(s_date, e_date):
    """
    Sends emails based on the course slots, not_in_slot, and
first-year data within the specified date range.

    Parameters:
    - s_date (str): Start date in the format "%d/%m/%Y".
    - e_date (str): End date in the format "%d/%m/%Y".

    The function iterates through the slotdata, not_in_slot, and
first_year_data, and sends emails accordingly.
    It uses the mail, mail_2, mail_not_in_slot, and
mail_first_year functions to handle the email sending logic.
```

```python
    Usage:
    do_mail('dd/mm/yyyy', 'dd/mm/yyyy')

    """
    for allCourses in range(len(slotdata)):
        course_list = slotdata[allCourses]
        course_slot = course_list[8]

        if course_slot == "" or course_slot == " ":
            continue

        slot_list = slot[course_slot]

        # time starting time slot_d1 and slot_d2
        slot_d1 = slot_list[0][0]
        slot_d2 = slot_list[1][0]

        if slot_d1 == slot_d2:
            mail(s_date, e_date, allCourses)
        else:
            mail_2(s_date, e_date, allCourses)

    for allCourses in range(len(not_in_slot)):
        mail_not_in_slot(s_date, e_date, allCourses)

    mail_first_year(s_date, e_date)
```

Sends emails based on the course slots, not_in_slot, and first-year data within the specified date range.

Parameters:
- s_date (str): Start date in the format "%d/%m/%Y".
- e_date (str): End date in the format "%d/%m/%Y".

The function iterates through the slotdata, not_in_slot, and first_year_data, and sends emails accordingly.

It uses the mail, mail_2, mail_not_in_slot, and mail_first_year functions to handle the email sending logic.

Usage:
do_mail('dd/mm/yyyy', 'dd/mm/yyyy')

About mail()

```
def mail(s_date,e_date,allCourses):
    """
    Sends a recurring event invitation for a specific course during the
specified date range.

    Parameters:
    - s_date (datetime.date): Start date of the recurring event.
    - e_date (datetime.date): End date of the recurring event.
    - allCourses (int): Index of the course in the slot_data list.

    The function uses the Google Calendar API to create a recurring event for
the specified course during the given date range.
    It extracts necessary information from the slot_data and slot dictionaries,
such as course details, location, time slots,
    and recurrence rules.

    Usage:
    mail(datetime.date(dd/mm/yyyy), datetime.date(dd/mm/yyyy))
    """

    #print("In mail")
    course_list = slot_data[allCourses]#['CSE', 'CSE506', 'Rest', 'Data
Mining', 'DMG', '40', 'C215', 'Vikram Goyal', '2', '2', '', '', '', '']
    # print(course_list)
    location=course_list[6]
    course = course_list[3]#Data Mining
    course_slot = course_list[8]#2
    print(course_slot)
    slot_list = slot[course_slot]#[['11:00:00', '12:30:00', 90, 'MONDAY'],
['11:00:00', '12:30:00', 90, 'THURSDAY']]
    #slot_d1/d2 day
    slot_d1= slot_list[0][3] #Monday
    slot_d2  = slot_list[1][3] #Thursday
    # t1/t2 time
    t1 =  slot_list[0][0]
    t2=  slot_list[1][0]
    tt1 = [] #Stores starting time
    tt2 = [] #Stores Ending Time
    if ":" in  t1:
      tt1 = t1.split(":")
    if "." in t1:
      tt1 = t1.split(".")

    if ":" in  t2:
      tt2 = t2.split(":")
    if "." in t2:
      tt2 = t2.split(".")

    if(len(tt1[0]) == 1):
        tt1[0] = "0"+tt1[0]
    if(len(tt1) == 1):
      tt1.append("00")
    if(len(tt1) == 2):
      tt1.append("00")
    if(len(tt2) == 1):
      tt2.append("00")
    if(len(tt2) == 2):
      tt2.append("00")
    time_d1 = [tt1[0],tt1[1],tt1[2]]
    x = ["05","30","00"]
    x_date = timedelta(hours=int(x[0]),minutes=int(x[1]),seconds=int(x[2]))

    timee_d1
=timedelta(hours=int(time_d1[0]),minutes=int(time_d1[1]),seconds=int(time_d1[2]
))


    remaing_time =timee_d1 - x_date  #Stores time difference between starting
time and 05:30:00(GMT-INDIA)
    remaing_time = str(remaing_time)
    remaing_time = remaing_time.split(":")
    # actual holiday
    ee_date = []
    hol_size = len(hol)
    # print(remaing_time)
    # print(hol)
    # add 0 if need
    for i,j in hol.items():
        # print(i) #20231120
        # print(j) #Friday
```

```python
    j = j.lower()
    b = j.strip()
    if(b== slot_d1.lower() or b== slot_d2.lower()):
        ee_date.append(i)
# print("First Here:")
# print(ee_date)


rr_date = [] #Actual subsitute day
sub_size = len(sub)

for i,j in sub.items():
    j = j.lower()
    b = j.strip()
    # print("DBZ")
    # print(j)
    # print(b)
    # print(slot_d1)
    # print(slot_d2.lower())
    if(b == slot_d1.lower() or b == slot_d2.lower()):
        rr_date.append(i)

# print(rr_date)
start_t = slot_list[0][0]
start_t = tt1
end_t = slot_list[0][1]
if ":" in  end_t:
  end_t = end_t.split(":")
if "." in end_t:
  end_t = end_t.split(".")

if(len(end_t[0]) == 1):
    end_t[0] = "0"+end_t[0]
if(len(end_t) == 1):
  end_t.append("00")
if(len(end_t) == 2):
  end_t.append("00")

#-----(2)
# if(int(start_t[0]) > 0 and int(start_t[0]) <7):
#     a = int(start_t[0])+12
#     start_t[0] = str(a)
# if(int(end_t[0]) > 0 and int(end_t[0]) <7):
#     a = int(end_t[0])+12
#     end_t[0] = str(a)
```

```python
#------
startt_time = start_t[0]+":"+start_t[1]+":"+start_t[2]
endd_time = end_t[0]+":"+ end_t[1]+":"+end_t[2]

datee = s_date.strftime("%d")
mont = s_date.strftime("%m")
yea = s_date.strftime("%Y")

s_date= datee+"/"+mont+"/"+yea
next_occurrence = get_next_day_occurrence(s_date, slot_list[0][3])
n_o = next_occurrence.split("/")
s_date = date(int(n_o[2]),int(n_o[1]),int(n_o[0]))

# s_date = s_date + timedelta(days=day_number[slot_list[0][3]])
datee = s_date.strftime("%d")
mont = s_date.strftime("%m")
yea = s_date.strftime("%Y")

start_datetime = yea+"-"+mont+"-"+datee+"T"+startt_time+"+05:30"
end_datetime = yea+"-"+mont+"-"+datee+"T"+endd_time+"+05:30"

datee1 = e_date.strftime("%d")
mont1 = e_date.strftime("%m")
yea1 = e_date.strftime("%Y")
s_d1 = short_term[slot_d1]
s_d2 = short_term[slot_d2]
aa = s_d1+","+s_d2
rrule = 'RRULE:FREQ=WEEKLY;UNTIL='+yea1+mont1+datee1+';BYDAY='+aa #RR rule

add_remaing = remaing_time[0]+remaing_time[1]+remaing_time[2]
# print("Hello-1")
# print(add_remaing)
if(len(add_remaing) == 5):
    add_remaing = "0"+add_remaing+"Z"
    # 040000Z
else:
    add_remaing=add_remaing+"Z"

exdate =""  #EXDATE Rule
if(len(ee_date) != 0):
    exdate = "EXDATE:"
len_eedate =len(ee_date)
ac = 0
for i in ee_date:
    len_eedate -= 1

    exdate = exdate+i+"T"+add_remaing

    if(len_eedate != 0):
        exdate += ","
    #print(exdate)

rdate = ""   #Rdate Rule
if(len(rr_date) != 0):
    rdate = "RDATE:"
print("JJK")
print(rdate)
len_rdate = len(rr_date)
for i in rr_date:
    len_rdate -= 1
    rdate = rdate+i+"T"+add_remaing
    if(len_rdate != 0):
        rdate += ","
# print("Here-1")
# print(rdate)
# print(start_datetime)
# print(end_datetime)
# print(rrule)
#print(exdate)
"""Shows basic usage of the Google Calendar API.
Prints the start and name of the next 10 events on the user's calendar.
"""
creds = None
# The file token.json stores the user's access and refresh tokens, and is
# created automatically when the authorization flow completes for the first
# time.
if os.path.exists('token.json'):
    creds = Credentials.from_authorized_user_file('token.json', SCOPES)
# If there are no (valid) credentials available, let the user log in.
if not creds or not creds.valid:
    if creds and creds.expired and creds.refresh_token:
        creds.refresh(Request())
    else:
        flow = InstalledAppFlow.from_client_secrets_file(
            'credentials.json', SCOPES)
        creds = flow.run_local_server(port=0)
    # Save the credentials for the next run
    with open('token.json', 'w') as token:
        token.write(creds.to_json())
try:
    service = build('calendar', 'v3', credentials=creds)
    event = {
            'summary': course,
            'location': location,
            'description': 'Happy Learning...:)',
            'start': {
                'dateTime': start_datetime,
                'timeZone': 'Asia/Kolkata'
            },
            'end': {
                'dateTime': end_datetime,
                'timeZone': 'Asia/Kolkata'
            },
            'recurrence': [
                rrule,
                rdate,
                exdate,
            ],
            'attendees': [

            {'email':'sample@iiitd.ac.in'}


            ],
        }

    # Insert the event
    event = service.events().insert(calendarId='primary',
body=event).execute()
    print(f'Recurring event created: {event["htmlLink"]}')
except HttpError as error:
    print('An error occurred: %s' % error)
```

Sends a recurring event invitation for a specific course during the specified date range.

Parameters:
- s_date (datetime.date): Start date of the recurring event.
- e_date (datetime.date): End date of the recurring event.
- allCourses (int): Index of the course in the slot_data list.

The function uses the Google Calendar API to create a recurring event for the specified course during the given date range.
It extracts necessary information from the slot_data and slot dictionaries, such as course details, location, time slots,
and recurrence rules.

Usage:
For those courses which have same time on both the days
mail(datetime.date(dd/mm/yyyy), datetime.date(dd/mm/yyyy))

About Mail_2()

```
def mail_2(s_date,e_date,allCourses):
    """
    Sends a recurring event invitation for a specific course
during the specified date range.

    Parameters:
    - s_date (datetime.date): Start date of the recurring event.
    - e_date (datetime.date): End date of the recurring event.
    - allCourses (int): Index of the course in the slot_data
list.

    The function uses the Google Calendar API to create a
recurring event for the specified course during the given date
range.
    It extracts necessary information from the slot_data and
slot dictionaries, such as course details, location, time slots,
    and recurrence rules.

    Usage:
    For those courses which have different timings of their
classes on both days
    mail(datetime.date(2023, 1, 1), datetime.date(2023, 1, 15),
0)
    """
```

```
course_list = slot_data[allCourses]
course = course_list[3]
location=course_list[6]
course_slot = course_list[8]
slot_list = slot[course_slot]
slot_d1= slot_list[0][3]
slot_d2  = slot_list[1][3]
t1 =  slot_list[0][0]
t2=  slot_list[1][0]
tt1 = []
tt2 = []
if ":" in  t1:
  tt1 = t1.split(":")
if "." in t1:
  tt1 = t1.split(".")

if ":" in  t2:
  tt2 = t2.split(":")
if "." in t2:
  tt2 = t2.split(".")



if(len(tt1) == 1):
  tt1.append("00")
if(len(tt2) == 1):
  tt2.append("00")
if(len(tt1) == 2):
  tt1.append("00")
if(len(tt2) == 2):
  tt2.append("00")

if(len(tt1[0]) == 1):
    tt1[0] = "0"+tt1[0]
if(len(tt2[0]) == 1):
        tt2[0] = "0"+tt2[0]



    time_d1 = [tt1[0],tt1[1],tt1[2]]
    time_d2 = [tt2[0] ,tt2[1],tt2[0]]
    x = ["05","30","00"]
    x_date = 
timedelta(hours=int(x[0]),minutes=int(x[1]),seconds=int(x[2]))

    timee_d1 = 
timedelta(hours=int(time_d1[0]),minutes=int(time_d1[1]),seconds=
int(time_d1[2]))
    timee_d2 = 
timedelta(hours=int(time_d2[0]),minutes=int(time_d2[1]),seconds=
int(time_d2[2]))

    remaing_time1 =timee_d1 - x_date
    remaing_time1 = str(remaing_time1)
    remaing_time1 = remaing_time1.split(":")

    remaing_time2 =timee_d2 - x_date
    remaing_time2 = str(remaing_time2)
    remaing_time2 = remaing_time2.split(":")
    #print(remaing_time1)
    ee_date_1 = []
    ee_date_2 = []
    hol_size = len(hol)
    for i,j in hol.items():
        j = j.lower()
        b = j.strip()
        if(b== slot_d1.lower()):
            ee_date_1.append(i)
        elif(b == slot_d2.lower()):
            ee_date_2.append(i)
    rr_date_1 = []
    rr_date_2 = []
    sub_size = len(sub)
    for i,j in sub.items():
        j = j.lower()
        b = j.strip()
        if(b == slot_d1.lower() ):
            rr_date_1.append(i)

        elif(b == slot_d2.lower()):
            rr_date_2.append(i)

    # mail for 1

    datee = s_date.strftime("%d")
    mont = s_date.strftime("%m")
    yea = s_date.strftime("%Y")

    s_date_1 = datee+"/"+mont+"/"+yea
    next_occurrence = get_next_day_occurrence(s_date_1,
slot_list[0][3])
    n_o = next_occurrence.split("/")
    s_date_1 = date(int(n_o[2]),int(n_o[1]),int(n_o[0]))

    # s_date_1 = s_date +
timedelta(days=day_number[slot_list[0][3]])
    datee = s_date_1.strftime("%d")
    mont = s_date_1.strftime("%m")
    yea = s_date_1.strftime("%Y")




    start_t1 = slot_list[0][0]
    start_t1 = tt1
    end_t1 = slot_list[0][1]
    if ":" in  end_t1:
      end_t1 = end_t1.split(":")
    if "." in end_t1:
      end_t1 = end_t1.split(".")

    if(len(end_t1[0]) == 1):
        end_t1[0] = "0"+end_t1[0]
    if(len(end_t1) == 1):
      end_t1.append("00")
    if(len(end_t1) == 2):
      end_t1.append("00")

    # if(int(start_t1[0]) > 0 and int(start_t1[0]) <7):
    #     a = int(start_t1[0])+12
    #     start_t1[0] = str(a)
    # if(int(end_t1[0]) > 0 and int(end_t1[0]) <7):
    #     a = int(end_t1[0])+12
    #     end_t1[0] = str(a)
    startt_time1 = start_t1[0]+":"+start_t1[1]+":"+start_t1[2]
    endd_time1 = end_t1[0]+":"+ end_t1[1]+":"+end_t1[2]
    start_datetime_1 =
yea+"-"+mont+"-"+datee+"T"+startt_time1+"+05:30"
    end_datetime_1 =
yea+"-"+mont+"-"+datee+"T"+endd_time1+"+05:30"
    datee_1 = e_date.strftime("%d")
    mont_1 = e_date.strftime("%m")
    yea_1 = e_date.strftime("%Y")
    s_d1 = short_term[slot_d1]

    rrule1 =
'RRULE:FREQ=WEEKLY;UNTIL='+yea_1+mont_1+datee_1+';BYDAY='+s_d1

    add_remaing1 =
remaing_time1[0]+remaing_time1[1]+remaing_time1[2]

    if(len(add_remaing1) == 5):
        add_remaing1 = "0"+add_remaing1+"Z"
    else:
        add_remaing1=add_remaing1+"Z"
    exdate1 =""
    if(len(ee_date_1) != 0):
        exdate1 = "EXDATE:"
    len_eedate1 = len(ee_date_1)
    ac =0
    for i in ee_date_1:
        len_eedate1 -= 1
        exdate1 = exdate1+i+"T"+add_remaing1
        if(len_eedate1 != 0):
            exdate1 += ","

    rdate1 = ""
    if(len(rr_date_1) != 0):
        rdate1 = "RDATE:"
    len_rdate1 = len(rr_date_1)
    for i in rr_date_1:
        len_rdate1 -= 1
        rdate1 = rdate1+i+"T"+add_remaing1
        if(len_rdate1 != 0):
            rdate1 += ","

    """Shows basic usage of the Google Calendar API.
    Prints the start and name of the next 10 events on the
user's calendar.
    """
    creds = None
    # The file token.json stores the user's access and refresh
tokens, and is
    # created automatically when the authorization flow
completes for the first
    # time.
    if os.path.exists('token.json'):
        creds =
Credentials.from_authorized_user_file('token.json', SCOPES)
    # If there are no (valid) credentials available, let the 
user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    try:
        service = build('calendar', 'v3', credentials=creds)
        event = {
                'summary': course,
                'location': location,
                'description': 'Happy Learning...:)',
                'start': {
                    'dateTime': start_datetime_1,
                    'timeZone': 'Asia/Kolkata'
                },
                'end': {
                    'dateTime': end_datetime_1,
                    'timeZone': 'Asia/Kolkata'
                },
                'recurrence': [
                    rrule1,
                    exdate1,
                    rdate1,
                ],
                'attendees': [

                {'email':'sample@iiitd.ac.in'}
                ],
            }

        # Insert the event
        event = service.events().insert(calendarId='primary',
body=event).execute()
        print(f'Recurring event created: {event["htmlLink"]}')
    except HttpError as error:
        print('An error occurred: %s' % error)


    # mail for 2
    datee = s_date.strftime("%d")
    mont = s_date.strftime("%m")
    yea = s_date.strftime("%Y")

    s_date_2 = datee+"/"+mont+"/"+yea
    next_occurrence = get_next_day_occurrence(s_date_2,
slot_list[1][3])
    n_o = next_occurrence.split("/")
    s_date_2 = date(int(n_o[2]),int(n_o[1]),int(n_o[0]))


    # s_date_2 = s_date +
timedelta(days=day_number[slot_list[1][3]])
    datee = s_date_2.strftime("%d")
    mont = s_date_2.strftime("%m")
    yea = s_date_2.strftime("%Y")

    start_t2 = slot_list[1][0]
    start_t2 = tt1
    end_t2 = slot_list[1][1]
    if ":" in  end_t2:
      end_t2 = end_t2.split(":")
    if "." in end_t2:
      end_t2 = end_t2.split(".")

    if(len(end_t2[0]) == 1):
        end_t2[0] = "0"+end_t2[0]
    if(len(end_t2) == 1):
      end_t2.append("00")
    if(len(end_t2) == 2):
      end_t2.append("00")
    # if(int(start_t2[0]) > 0 and int(start_t2[0]) <7):
    #     a = int(start_t2[0])+12
    #     start_t2[0] = str(a)
    # if(int(end_t2[0]) > 0 and int(end_t2[0]) <7):
    #     a = int(end_t2[0])+12
    #     end_t2[0] = str(a)

    startt_time2 = start_t2[0]+":"+start_t2[1]+":"+ start_t2[1]
    endd_time2 = end_t2[0]+":"+ end_t2[1]+":"+ end_t2[2]
    start_datetime_2 =
yea+"-"+mont+"-"+datee+"T"+startt_time2+"+05:30"
    end_datetime_2 =
yea+"-"+mont+"-"+datee+"T"+endd_time2+"+05:30"
    datee_2 = e_date.strftime("%d")
    mont_2 = e_date.strftime("%m")
    yea_2 = e_date.strftime("%Y")
    s_d2 = short_term[slot_d2]
    rrule2 =
'RRULE:FREQ=WEEKLY;UNTIL='+yea_2+mont_2+datee_2+';BYDAY='+s_d2
    add_remaing2 =
remaing_time2[0]+remaing_time2[1]+remaing_time2[2]
    if(len(add_remaing2) == 5):
        add_remaing2 = "0"+add_remaing2+"Z"

    exdate2 =""
    if(len(ee_date_2) != 0):
        exdate2 = "EXDATE:"
    len_eedate2 = len(ee_date_2)
    ac =0
    for i in ee_date_2:
        len_eedate2 -= 1
        exdate2 = exdate2+i+"T"+add_remaing2
        if(len_eedate2 != 0):
            exdate2 += ","
    #print(exdate2)
    rdate2 = ""
    if(len(rr_date_2) != 0):
        rdate2 = "RDATE:"
    len_rdate2 = len(rr_date_2)
    for i in rr_date_2:
        len_rdate2 -= 1
        rdate2 = rdate2+i+"T"+add_remaing2
        if(len_rdate2 != 0):
            rdate2 += ","
    """Shows basic usage of the Google Calendar API.
    Prints the start and name of the next 10 events on the
user's calendar.
    """
    creds = None
    # The file token.json stores the user's access and refresh
tokens, and is
    # created automatically when the authorization flow
completes for the first
    # time.
    if os.path.exists('token.json'):
        creds =
Credentials.from_authorized_user_file('token.json', SCOPES)
    # If there are no (valid) credentials available, let the
user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
```

```python
        flow = InstalledAppFlow.from_client_secrets_file(
            'credentials.json', SCOPES)
        creds = flow.run_local_server(port=0)
    # Save the credentials for the next run
    with open('token.json', 'w') as token:
        token.write(creds.to_json())

try:
    service = build('calendar', 'v3', credentials=creds)
    event = {
            'summary': course,
            'location': location,
            'description': 'Happy Learning...:)',
            'start': {
                'dateTime': start_datetime_2,
                'timeZone': 'Asia/Kolkata'
            },
            'end': {
                'dateTime': end_datetime_2,
                'timeZone': 'Asia/Kolkata'
            },
            'recurrence': [
                rrule2,
                exdate2,
                rdate2,
            ],
            'attendees': [

            {'email':'sample@iiitd.ac.in'}


            ],
        }

    # Insert the event
    event = service.events().insert(calendarId='primary',
body=event).execute()
        print(f'Recurring event created: {event["htmlLink"]}')
    except HttpError as error:
        print('An error occurred: %s' % error)
```

   Sends a recurring event invitation for a specific course during the specified date range.

   Parameters:
   - s_date (datetime.date): Start date of the recurring event.
   - e_date (datetime.date): End date of the recurring event.
   - allCourses (int): Index of the course in the slot_data list.

   The function uses the Google Calendar API to create a recurring event for the specified course during the given date range.
   It extracts necessary information from the slot_data and slot dictionaries, such as course details, location, time slots,
   and recurrence rules.

   Usage:
   For those courses which have different timings of their classes on both days
   mail(datetime.date(2023, 1, 1), datetime.date(2023, 1, 15), 0)

About mail_not_in_slot()

```
def mail_not_in_slot(s_date,e_date,index):
    """
    Sends recurring event invitations for a course not in a regular time slot
during the specified date range.

    Parameters:
    - s_date (datetime.date): Start date of the recurring event.
    - e_date (datetime.date): End date of the recurring event.
    - index (int): Index of the course in the not_in_slot list.

    The function creates a recurring event for a course not in a regular time
slot, considering holidays and substitute dates.
```

```
    It uses the Google Calendar API to send invitations for the specified time 
slot.

    Usage:
    For the courses that doesnt fall in any slot(i.e Slot=Nil)
    mail_not_in_slot(datetime.date(yyyy,mm,dd), datetime.date(yyyy,mm, dd))
    """

    course_list = not_in_slot[index] #list of the course which contains all its 
details
    course = course_list[3] #Name of the course
    print(course_list)
    course_day = course_list[13] #Day on which Course lies
    s_time = course_list[11] #Starting Time
    e_time = course_list[12] #Ending time

    if "." in s_time:
      s_time = s_time.split(".")
    if ":" in s_time:
      s_time = s_time.split(".")

    if "." in e_time:
      e_time = e_time.split(".")
    if ":" in e_time:
      e_time = e_time.split(".")



    if(len(s_time[0]) == 1):
        s_time[0] = "0"+s_time[0]
    if(len(e_time[0]) == 1):
        e_time[0] = "0" + e_time[0]

    if(len(s_time) == 1):
      s_time.append("00")
    if(len(s_time) == 2):
      s_time.append("00")

    if(len(e_time) == 1):
      e_time.append("00")
    if(len(e_time) == 2):
      e_time.append("00")

    start_time = [s_time[0],s_time[1],s_time[2]] #Storing starting time in list
    end_time = [e_time[0],e_time[1],e_time[2]]   #Storing ending time in list
    startt_time = start_time[0]+":"+start_time[1]+":"+start_time[2]
    endd_time = end_time[0]+":"+ end_time[1]+":"+end_time[2]
    x = ["05","30","00"]

    x_date = timedelta(hours=int(x[0]),minutes=int(x[1]),seconds=int(x[2]))

    time_d = 
timedelta(hours=int(start_time[0]),minutes=int(start_time[1]),seconds=int(start
_time[2]))
    remaing_time =time_d - x_date  #Difference of the time between starting 
time and 05:30:00(GMT-INDIA)
    remaing_time = str(remaing_time)
    remaing_time = remaing_time.split(":")

    #Actual Holiday
    ee_date = []
    hol_size = len(hol)

    for i,j in hol.items():
        j = j.lower()
        b = j.strip()
        if(b== course_day.lower()):
            ee_date.append(i)


    #Actual subsitute Day
    rr_date = []
    sub_size = len(sub)

    for i,j in sub.items():
        j = j.lower()
        b = b.strip()
        if(b == course_day.lower()):
            rr_date.append(i)
    datee = s_date.strftime("%d")
    mont = s_date.strftime("%m")
    yea = s_date.strftime("%Y")

    ss_date = datee+"/"+mont+"/"+yea
    next_occurrence = get_next_day_occurrence(ss_date, course_day)
    n_o = next_occurrence.split("/")
    s_date = date(int(n_o[2]),int(n_o[1]),int(n_o[0]))
    datee = s_date.strftime("%d")
    mont = s_date.strftime("%m")
    yea = s_date.strftime("%Y")
    start_datetime = yea+"-"+mont+"-"+datee+"T"+startt_time+"+05:30"
    end_datetime = yea+"-"+mont+"-"+datee+"T"+endd_time+"+05:30"

    datee1 = e_date.strftime("%d")
    mont1 = e_date.strftime("%m")
    yea1 = e_date.strftime("%Y")
    rrule = 
'RRULE:FREQ=WEEKLY;UNTIL='+yea1+mont1+datee1+';BYDAY='+short_term[course_day.up
per()] #RRule

    add_remaing = remaing_time[0]+remaing_time[1]+remaing_time[2]
    if(len(add_remaing) == 5):
        add_remaing = "0"+add_remaing+"Z"
        # 040000Z

    exdate ="" #EXDATE
    if(len(ee_date) != 0):
        exdate = "EXDATE:"
    len_eedate =len(ee_date)
    ac = 0
    for i in ee_date:

        len_eedate -= 1

        exdate = exdate+i+"T"+add_remaing
        if(len_eedate != 0):
            exdate += ","

    rdate = "" #RDATE
    if(len(rr_date) != 0):
        rdate = "RDATE:"
    len_rdate = len(rr_date)
    for i in rr_date:
        len_rdate -= 1
        rdate = rdate+i+"T"+add_remaing
        if(len_rdate != 0):
            rdate += ","

    """Shows basic usage of the Google Calendar API.
    Prints the start and name of the next 10 events on the user's calendar.
    """
    creds = None
```

```python
# The file token.json stores the user's access and refresh tokens, and is
# created automatically when the authorization flow completes for the first
# time.
if os.path.exists('token.json'):
    creds = Credentials.from_authorized_user_file('token.json', SCOPES)
# If there are no (valid) credentials available, let the user log in.
if not creds or not creds.valid:
    if creds and creds.expired and creds.refresh_token:
        creds.refresh(Request())
    else:
        flow = InstalledAppFlow.from_client_secrets_file(
            'credentials.json', SCOPES)
        creds = flow.run_local_server(port=0)
    # Save the credentials for the next run
    with open('token.json', 'w') as token:
        token.write(creds.to_json())

try:
    service = build('calendar', 'v3', credentials=creds)
    event = {
            'summary': course,
            'location': 'Somewhere',
            'description': 'somewhere online',
            'start': {
                'dateTime': start_datetime,
                'timeZone': 'Asia/Kolkata'
            },
            'end': {
                'dateTime': end_datetime,
                'timeZone': 'Asia/Kolkata'
            },
            'recurrence': [
                rrule,exdate,rdate


            ],
            'attendees': [

                {'email':'sample_1@iiitd.ac.in'},
                {'email':'sample_279@iiitd.ac.in'}


            ],
        }
        # Insert the event
        event = service.events().insert(calendarId='primary',
body=event).execute()
        print(f'Recurring event created: {event["htmlLink"]}')
    except HttpError as error:
        print('An error occurred: %s' % error)
```

Sends recurring event invitations for a course not in a regular time slot during the specified date range.

Parameters:
- s_date (datetime.date): Start date of the recurring event.
- e_date (datetime.date): End date of the recurring event.
- index (int): Index of the course in the not_in_slot list.

The function creates a recurring event for a course not in a regular time slot, considering holidays and substitute dates.
It uses the Google Calendar API to send invitations for the specified time slot.

Usage:
For the courses that doesnt fall in any slot(i.e Slot=Nil)
mail_not_in_slot(datetime.date(yyyy,mm,dd), datetime.date(yyyy,mm, dd))

About holiday_add(ms_date,holys_date,mid_date,holy_date)

```
def holiday_add(ms_date,holys_date,mid_date,holy_date):
    """
    Adds holidays to the 'hol' dictionary in the specified date range.

    Parameters:
    - ms_date (list): List representing the current date [day, month, year].
    - holys_date (date): Start date of the holiday period.
    - mid_date (date): Current date to start adding holidays.
    - holy_date (date): End date of the holiday period.

    The function iterates through the date range from mid_date to holy_date,
adds each date as a key to the 'hol' dictionary,
    and assigns the corresponding day of the week as the value.

    Usage:
    holiday_add([dd, mm, yyyy], date(dd, mm, yyyy), date(dd, mm, yyyy),
date(dd, mm, yyyy))
    """
    # print("CheckPoint-1")
    # print(hol)
    while(mid_date <= holy_date):
        # print(mid_date)     #2023-10-28
        # print(ms_date)      #[28,10,2023]
        a = ms_date[2]+ms_date[1]+ms_date[0]#Check-4
        #print(a)             #20231130
        mid_date = date(int(ms_date[2]),int(ms_date[1]),int(ms_date[0]))

        dayy = mid_date.strftime('%A')
        #print(dayy)        #Saturday
        hol[a] = dayy
        # sss = int(ms_date[0])+1
        # print(sss)      #29
        #ms_date[0] = str(sss)
        incremented_date = mid_date + timedelta(days=1)
        mid_date=incremented_date
        year = str(mid_date.year)
        month = str(mid_date.month)
        day = str(mid_date.day)
        final_day=""
        if(len(day)==1):
            final_day="0"+day
        else:
            final_day=day
        final_month=""
        if(len(month)==1):
            final_month="0"+month
        else:
            final_month=month
        ms_date = [final_day, final_month, year]
        #mid_date = date(int(ms_date[2]),int(ms_date[1]),int(ms_date[0]))
```

Adds holidays to the 'hol' dictionary in the specified date range.

Parameters:
- ms_date (list): List representing the current date [day, month, year].
- holys_date (date): Start date of the holiday period.

- mid_date (date): Current date to start adding holidays.
- holy_date (date): End date of the holiday period.

The function iterates through the date range from mid_date to holy_date, adds each date as a key to the 'hol' dictionary,
and assigns the corresponding day of the week as the value.

Usage:
holiday_add([dd, mm, yyyy], date(dd, mm, yyyy), date(dd, mm, yyyy), date(dd, mm, yyyy))

About main()

```
def main():
    """
    The main function orchestrating the execution of various tasks.

    The function first attempts to read the calendar and work with its data. If successful, it proceeds to read the slot data,
    first-year data, and additional file data. If any of these steps encounters an error, an appropriate flag is set.

    After handling the initial setup, the user is prompted to input the semester start date, semester end date, mid-semester start date,
    and mid-semester holy date. The dates are processed, and holidays are added to the 'hol' dictionary using the 'holiday_add' function.
    Finally, the 'do_mail' function is called with the processed dates to execute the necessary actions related to Google Calendar.
    """
    read_calender_error = read_calender()
    error_flag = 0
    if(read_calender_error == 0):
       error_flag = work_all_calender_data()

    print(read_calender_error)
    print(error_flag)

    read_slot_data_error = read_slotdata_file()
    read_first_year_error_flag = read_first_year()

    print(read_slot_data_error)
    print(read_first_year_error_flag)
    rest_file  = input("Please provide the file name: ")

    rest_file_error =  reading_rest_file(rest_file)
    print(rest_file_error)

    if (error_flag == 0 or read_first_year_error_flag == 0 or
read_slot_data_error == 0 or rest_file_error == 0):

      s_date = input("please provide the semester start date (format
dd/mm/yyyy):")
      e_date = input("please provide the semester end date(format
dd/mm/yyyy):")
      mid_date = input("please provide the semester starting
mid-sem-date(format dd/mm/yyyy) :")
      holy_date = input("please provide the semester last
mid_sem-holy-date(format dd/mm/yyyy):")
      # s_date = takingInput("please provide the semester start date:")
      # e_date = takingInput("please provide the semester end date:")
      # mid_date = takingInput("please provide the semester starting
mid-sem-date :")
      # holy_date = takingInput("please provide the semester last
mid_sem-holy-date:")
      start_date = s_date.split("/")
      end_date = e_date.split("/")
      ms_date = mid_date.split("/")
      holys_date = holy_date.split("/")

      s_date     =
date(int(start_date[2]),int(start_date[1]),int(start_date[0]))
      e_date     =  date(int(end_date[2]),int(end_date[1]),int(end_date[0]))
      mid_date   =  date(int(ms_date[2]),int(ms_date[1]),int(ms_date[0]))
      holy_date  =
date(int(holys_date[2]),int(holys_date[1]),int(holys_date[0]))


      holiday_add(ms_date,holys_date,mid_date,holy_date)
      do_mail(s_date,e_date)
```

The function first attempts to read the calendar and work with its data. If successful, it proceeds to read the slot data,
first-year data, and additional file data. If any of these steps encounters an error, an appropriate flag is set.

After handling the initial setup, the user is prompted to input the semester start date, semester end date, mid-semester start date,
and mid-semester holy date. The dates are processed, and holidays are added to the 'hol' dictionary using the 'holiday_add' function.
Finally, the 'do_mail' function is called with the processed dates to execute the necessary actions related to Google Calendar.

About correct_order(s_date, e_date, mid_date, holy_date):

```
def correct_order(s_date, e_date, mid_date, holy_date):
    """
    Check the chronological order of dates in a semester-related
sequence.

    Parameters:
    - s_date (str): The starting date of the semester.
    - e_date (str): The ending date of the semester.
    - mid_date (str): The date of the midsem break.
    - holy_date (str): The holy date.

    Returns:
    - int: Returns 0 if the dates are in the correct order,
           1 if the last date of the midsem break is wrong,
           2 if the starting date of the midsem exam is wrong,
           3 if the ending date of the semester is wrong.

    Note: Assumes that the compare_dates function is available
to compare date strings.
    """
    if compare_dates(s_date, e_date):
        if compare_dates(s_date, mid_date) and
compare_dates(mid_date, e_date):
            if compare_dates(mid_date, holy_date) and
compare_dates(holy_date, e_date):
                return 0  # Every date is fine
            else:
                return 1  # Last date of midsem break is wrong
        else:
                return 2  # Starting date of midsem exam is wrong
    else:
        return 3  # Ending date of semester is wrong
```

Check the chronological order of dates in a semester-related sequence.

Parameters:
- s_date (str): The starting date of the semester.
- e_date (str): The ending date of the semester.
- mid_date (str): The date of the midsem break.
- holy_date (str): The holy date.

Returns:
- int: Returns 0 if the dates are in the correct order,
  1 if the last date of the midsem break is wrong,
  2 if the starting date of the midsem exam is wrong,
  3 if the ending date of the semester is wrong.

Note: Assumes that the compare_dates function is available to compare date strings.

About compare_dates(date_str1, date_str2):

```
from datetime import datetime

def compare_dates(date_str1, date_str2):
    """
    Compare two date strings in the format "DD/MM/YYYY".

    Parameters:
    - date_str1 (str): The first date string.
    - date_str2 (str): The second date string.
    Returns:
    - int: Returns 1 if date_str2 is after date_str1, 0 if they
are equal, and raises a ValueError if the date strings are
invalid.
    """
    # Define the date format
    format_str = "%d/%m/%Y"

    try:
        # Parse the date strings into datetime objects
        date1 = datetime.strptime(date_str1, format_str)
        date2 = datetime.strptime(date_str2, format_str)

        # Compare the datetime objects
        if date1 < date2:
            return 1
        elif date1 > date2:
            return 0
        else:
            return 0

    except ValueError as e:
        # Handle invalid date strings
        return f"Error: {e}. Please provide valid date strings
in the format DD/MM/YYYY."
```

Compare two date strings in the format "DD/MM/YYYY".

Parameters:
- date_str1 (str): The first date string.
- date_str2 (str): The second date string.

Returns:
- int: Returns 1 if date_str2 is after date_str1, 0 if they are equal, and raises a ValueError if the date strings are invalid.