SCHEDULE MAKER

NAMES:

- Reem Hussin Mostafa Ibrahim(221000241)
- Mohanned Mahmoud (221000703)
- Mahmoud Essa (221001916)

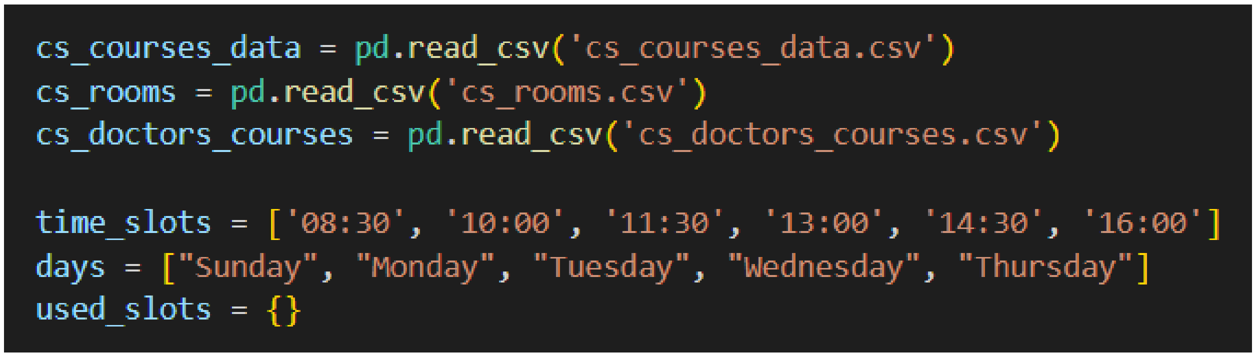
Project idea:

The project aims to develop an automated course scheduling system for students. This system will allow students to sign up for courses, choose from available course titles, and generate a personalized schedule. The schedule will consider constraints such as course capacities and avoid overlapping time slots, ensuring a feasible and optimized timetable for each student. By employing a Constraint Satisfaction Problem (CSP) backtracking algorithm, the system guarantees an efficient assignment of courses to available time slots while adhering to all defined constraints.

The code comprises several functions that work together to create and manage student course schedules. The main components are student registration, course selection, schedule generation using CSP backtracking, schedule formatting, and saving the schedule to a CSV file. Each function is designed to handle specific tasks, ensuring a modular and maintainable structure.

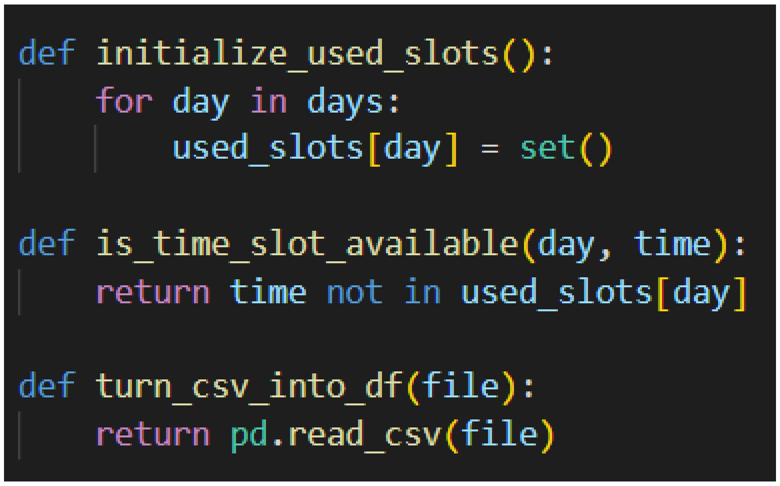
Code explanation:

(test.py)



```
cs_courses_data = pd.read_csv('cs_courses_data.csv')
cs_rooms = pd.read_csv('cs_rooms.csv')
cs_doctors_courses = pd.read_csv('cs_doctors_courses.csv')

time_slots = ['08:30', '10:00', '11:30', '13:00', '14:30', '16:00']
days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday"]
used_slots = {}
```

Load the data and define the available time slots and days for scheduling courses. The used_slots dictionary will keep track of the used time slots for each day.

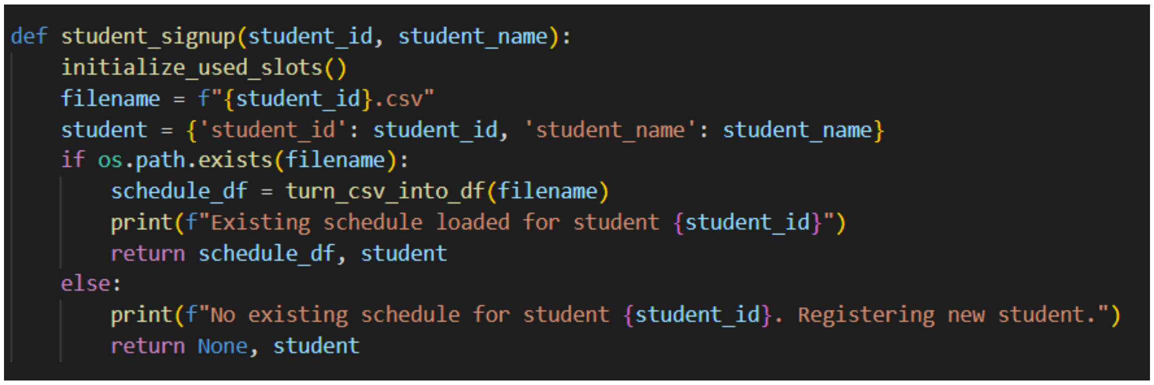


```
def initialize_used_slots():
    for day in days:
        used_slots[day] = set()

def is_time_slot_available(day, time):
    return time not in used_slots[day]

def turn_csv_into_df(file):
    return pd.read_csv(file)
```

The first function initializes the used_slots dictionary to keep track of which time slots have been used for each day, ensuring no conflicts in scheduling.

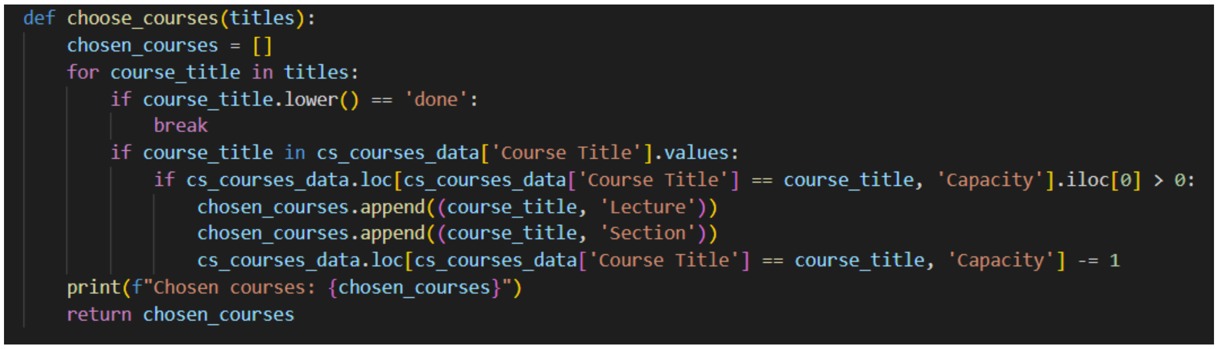
The second function checks if a specific time slot on a given day is available for scheduling.

The third function reads a CSV file and converts it into a Pandas Data Frame for easier manipulation and analysis.



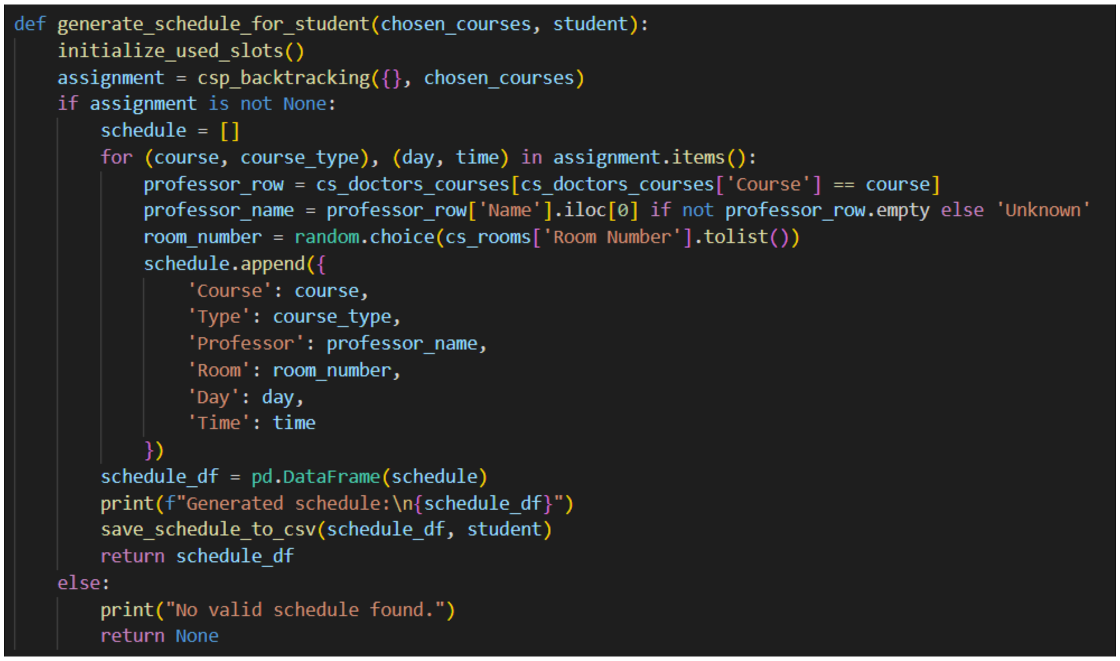
```python
def student_signup(student_id, student_name):
    initialize_used_slots()
    filename = f"{student_id}.csv"
    student = {'student_id': student_id, 'student_name': student_name}
    if os.path.exists(filename):
        schedule_df = turn_csv_into_df(filename)
        print(f"Existing schedule loaded for student {student_id}")
        return schedule_df, student
    else:
        print(f"No existing schedule for student {student_id}. Registering new student.")
        return None, student
```

The student_signup function handles student registration. If the student id already existing so the student already has a schedule, so the code loads it. otherwise, it initializes a new registration and a new schedule.



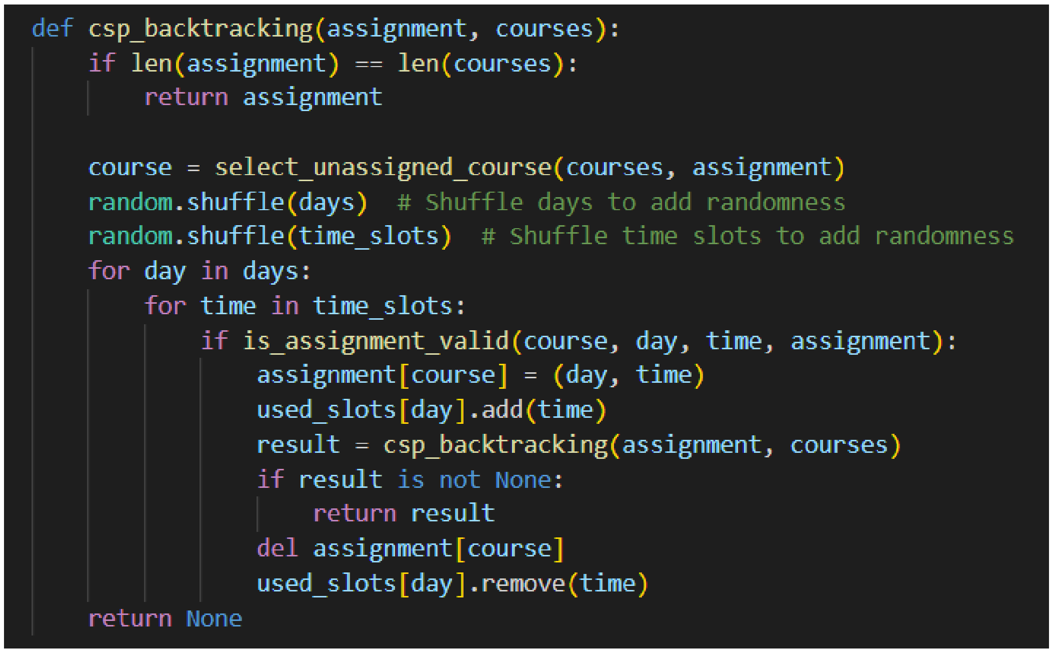
```python
def choose_courses(titles):
    chosen_courses = []
    for course_title in titles:
        if course_title.lower() == 'done':
            break
        if course_title in cs_courses_data['Course Title'].values:
            if cs_courses_data.loc[cs_courses_data['Course Title'] == course_title, 'Capacity'].iloc[0] > 0:
                chosen_courses.append((course_title, 'Lecture'))
                chosen_courses.append((course_title, 'Section'))
                cs_courses_data.loc[cs_courses_data['Course Title'] == course_title, 'Capacity'] -= 1
    print(f"Chosen courses: {chosen_courses}")
    return chosen_courses
```

This function allows students to choose their desired courses, ensuring that the courses are available and not exceeding capacity.



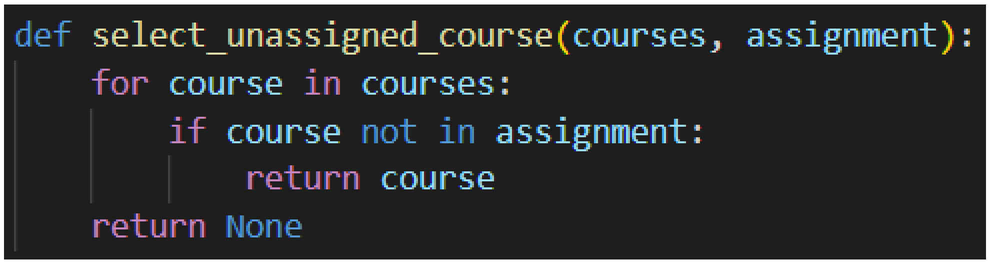
```python
def generate_schedule_for_student(chosen_courses, student):
    initialize_used_slots()
    assignment = csp_backtracking({}, chosen_courses)
    if assignment is not None:
        schedule = []
        for (course, course_type), (day, time) in assignment.items():
            professor_row = cs_doctors_courses[cs_doctors_courses['Course'] == course]
            professor_name = professor_row['Name'].iloc[0] if not professor_row.empty else 'Unknown'
            room_number = random.choice(cs_rooms['Room Number'].tolist())
            schedule.append({
                'Course': course,
                'Type': course_type,
                'Professor': professor_name,
                'Room': room_number,
                'Day': day,
                'Time': time
            })
        schedule_df = pd.DataFrame(schedule)
        print(f"Generated schedule:\n{schedule_df}")
        save_schedule_to_csv(schedule_df, student)
        return schedule_df
    else:
        print("No valid schedule found.")
        return None
```

Generates a schedule for the student by finding valid time slots using the CSP backtracking algorithm. It assigns the chosen courses, available professors, rooms, days, and times to create a complete timetable.



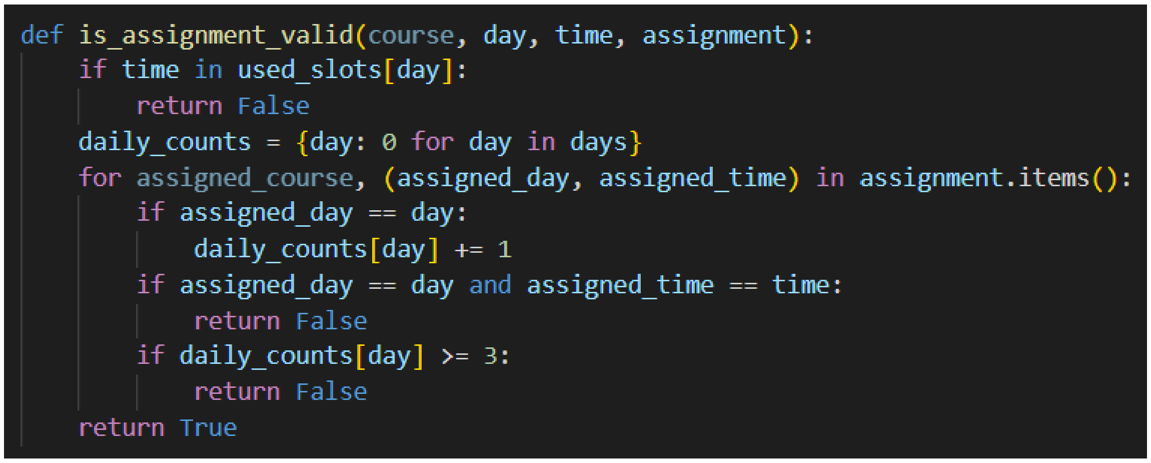
```python
def csp_backtracking(assignment, courses):
    if len(assignment) == len(courses):
        return assignment

    course = select_unassigned_course(courses, assignment)
    random.shuffle(days)  # Shuffle days to add randomness
    random.shuffle(time_slots)  # Shuffle time slots to add randomness
    for day in days:
        for time in time_slots:
            if is_assignment_valid(course, day, time, assignment):
                assignment[course] = (day, time)
                used_slots[day].add(time)
                result = csp_backtracking(assignment, courses)
                if result is not None:
                    return result
                del assignment[course]
                used_slots[day].remove(time)
    return None
```

The core of the scheduling algorithm, this function uses CSP backtracking to find a valid assignment of courses to time slots and days.



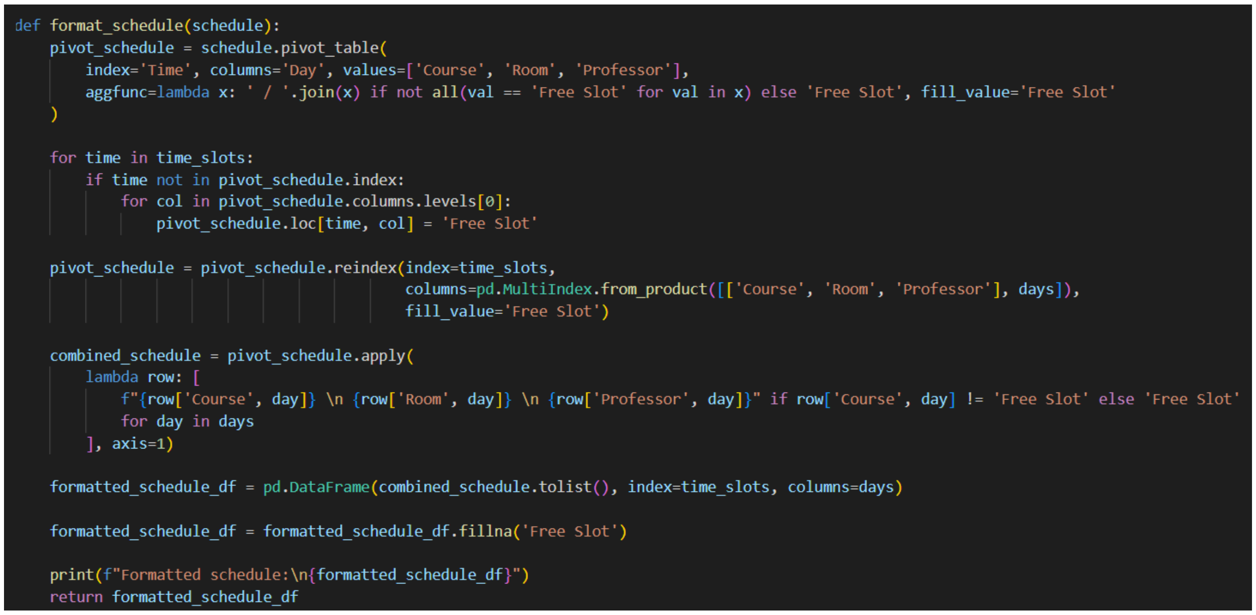
```python
def select_unassigned_course(courses, assignment):
    for course in courses:
        if course not in assignment:
            return course
    return None
```

This function selects the next unassigned course to be scheduled.



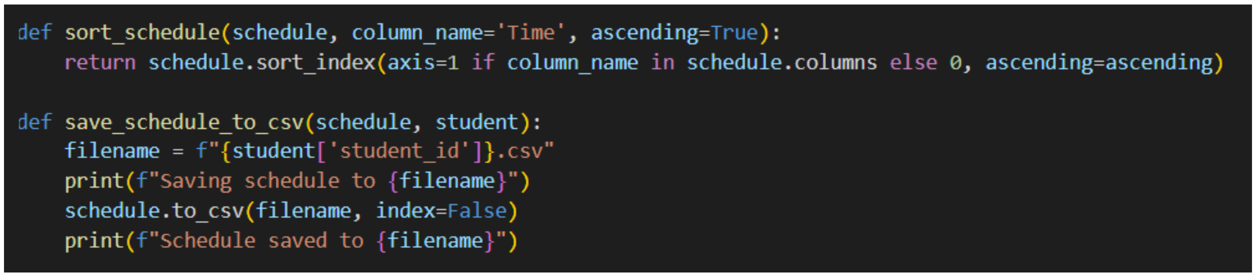
```python
def is_assignment_valid(course, day, time, assignment):
    if time in used_slots[day]:
        return False
    daily_counts = {day: 0 for day in days}
    for assigned_course, (assigned_day, assigned_time) in assignment.items():
        if assigned_day == day:
            daily_counts[day] += 1
        if assigned_day == day and assigned_time == time:
            return False
        if daily_counts[day] >= 3:
            return False
    return True
```

Ensures that a proposed assignment does not violate any constraints (e.g., overlapping times, maximum courses per day).



```python
def format_schedule(schedule):
    pivot_schedule = schedule.pivot_table(
        index='Time', columns='Day', values=['Course', 'Room', 'Professor'],
        aggfunc=lambda x: ' / '.join(x) if not all(val == 'Free Slot' for val in x) else 'Free Slot', fill_value='Free Slot'
    )

    for time in time_slots:
        if time not in pivot_schedule.index:
            for col in pivot_schedule.columns.levels[0]:
                pivot_schedule.loc[time, col] = 'Free Slot'

    pivot_schedule = pivot_schedule.reindex(index=time_slots,
                                            columns=pd.MultiIndex.from_product([['Course', 'Room', 'Professor'], days]),
                                            fill_value='Free Slot')

    combined_schedule = pivot_schedule.apply(
        lambda row: [
            f"{row['Course', day]} \n {row['Room', day]} \n {row['Professor', day]}" if row['Course', day] != 'Free Slot' else 'Free Slot'
            for day in days
        ], axis=1)

    formatted_schedule_df = pd.DataFrame(combined_schedule.tolist(), index=time_slots, columns=days)

    formatted_schedule_df = formatted_schedule_df.fillna('Free Slot')

    print(f"Formatted schedule:\n{formatted_schedule_df}")
    return formatted_schedule_df
```

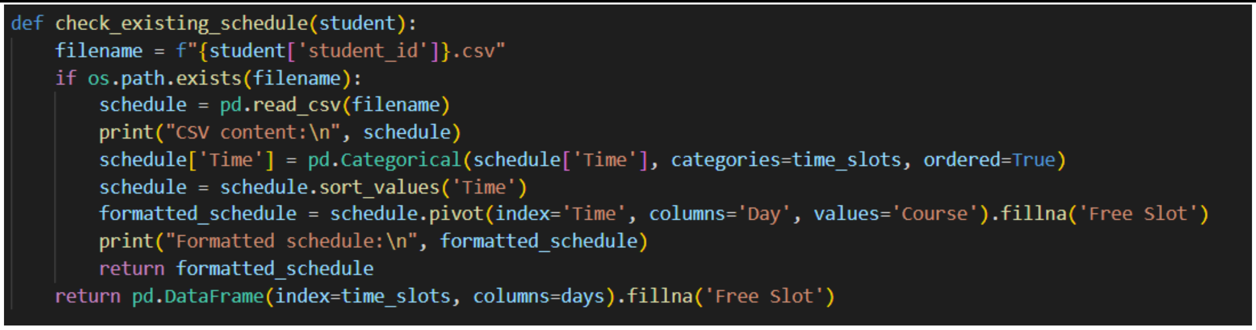
Formats the generated schedule into a more readable and structured table, showing the courses, rooms, and professors assigned to each time slot and day.



```python
def sort_schedule(schedule, column_name='Time', ascending=True):
    return schedule.sort_index(axis=1 if column_name in schedule.columns else 0, ascending=ascending)

def save_schedule_to_csv(schedule, student):
    filename = f"{student['student_id']}.csv"
    print(f"Saving schedule to {filename}")
    schedule.to_csv(filename, index=False)
    print(f"Schedule saved to {filename}")
```

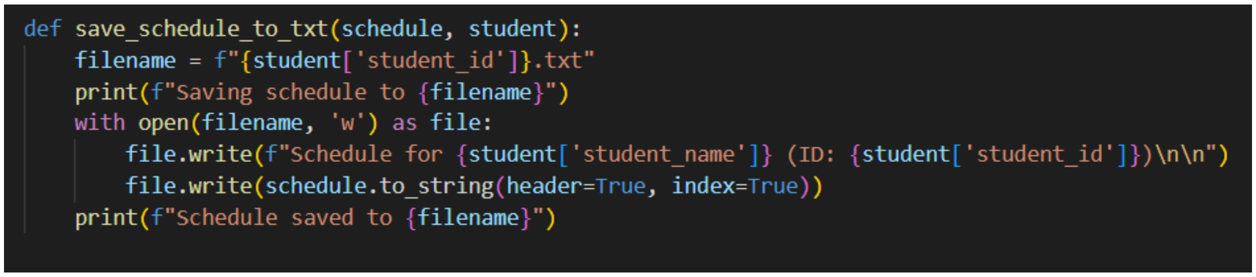
The first function sorts the schedule based on the specified column, typically by time.

The second function saves the generated schedule to a CSV file for future reference.



```python
def check_existing_schedule(student):
    filename = f"{student['student_id']}.csv"
    if os.path.exists(filename):
        schedule = pd.read_csv(filename)
        print("CSV content:\n", schedule)
        schedule['Time'] = pd.Categorical(schedule['Time'], categories=time_slots, ordered=True)
        schedule = schedule.sort_values('Time')
        formatted_schedule = schedule.pivot(index='Time', columns='Day', values='Course').fillna('Free Slot')
        print("Formatted schedule:\n", formatted_schedule)
        return formatted_schedule
    return pd.DataFrame(index=time_slots, columns=days).fillna('Free Slot')
```

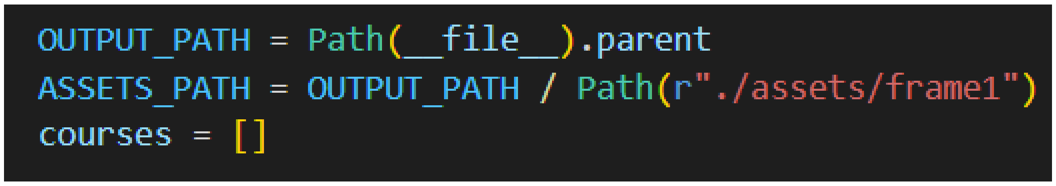
Checks if an existing schedule file exists for the student and loads it if available. Formats and sorts the schedule for display.



```python
def save_schedule_to_txt(schedule, student):
    filename = f"{student['student_id']}.txt"
    print(f"Saving schedule to {filename}")
    with open(filename, 'w') as file:
        file.write(f"Schedule for {student['student_name']} (ID: {student['student_id']})\n\n")
        file.write(schedule.to_string(header=True, index=True))
    print(f"Schedule saved to {filename}")
```

Saves the schedule to a text file for easy sharing and printing.

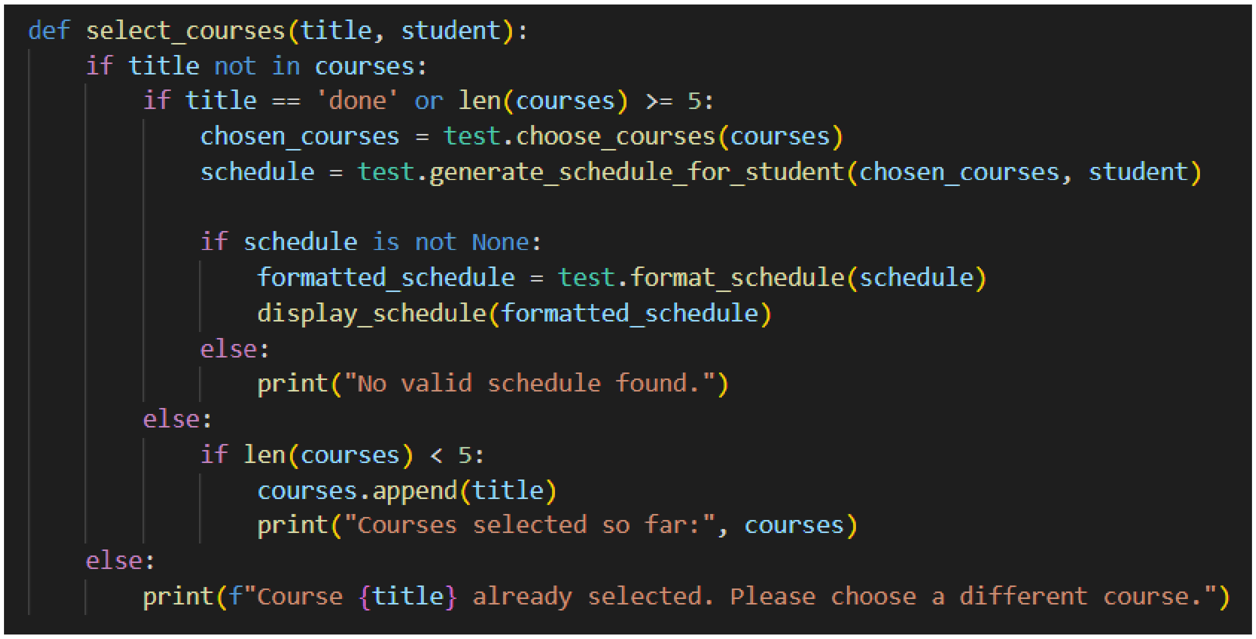
(gui1.py)



```python
OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH / Path(r"./assets/frame1")
courses = []
```

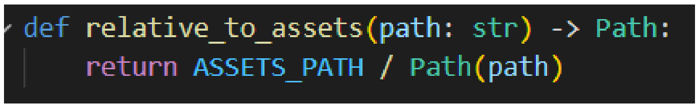
File Paths: Define paths to the output directory and assets directory.

Course List: Initialize an empty list courses to store selected course titles.



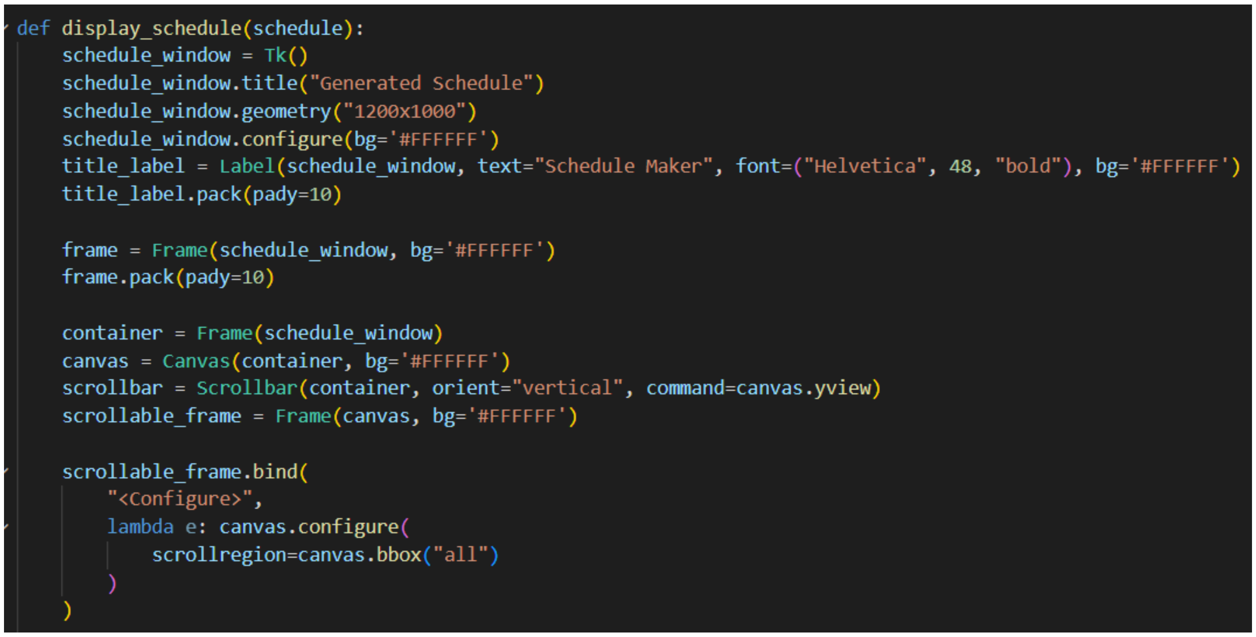
```python
def select_courses(title, student):
    if title not in courses:
        if title == 'done' or len(courses) >= 5:
            chosen_courses = test.choose_courses(courses)
            schedule = test.generate_schedule_for_student(chosen_courses, student)

            if schedule is not None:
                formatted_schedule = test.format_schedule(schedule)
                display_schedule(formatted_schedule)
            else:
                print("No valid schedule found.")
        else:
            if len(courses) < 5:
                courses.append(title)
                print("Courses selected so far:", courses)
    else:
        print(f"Course {title} already selected. Please choose a different course.")
```

Manages course selection by adding unique courses to the list until 'done' is selected or five courses are chosen. Calls scheduling functions from the test module and displays the schedule if valid.

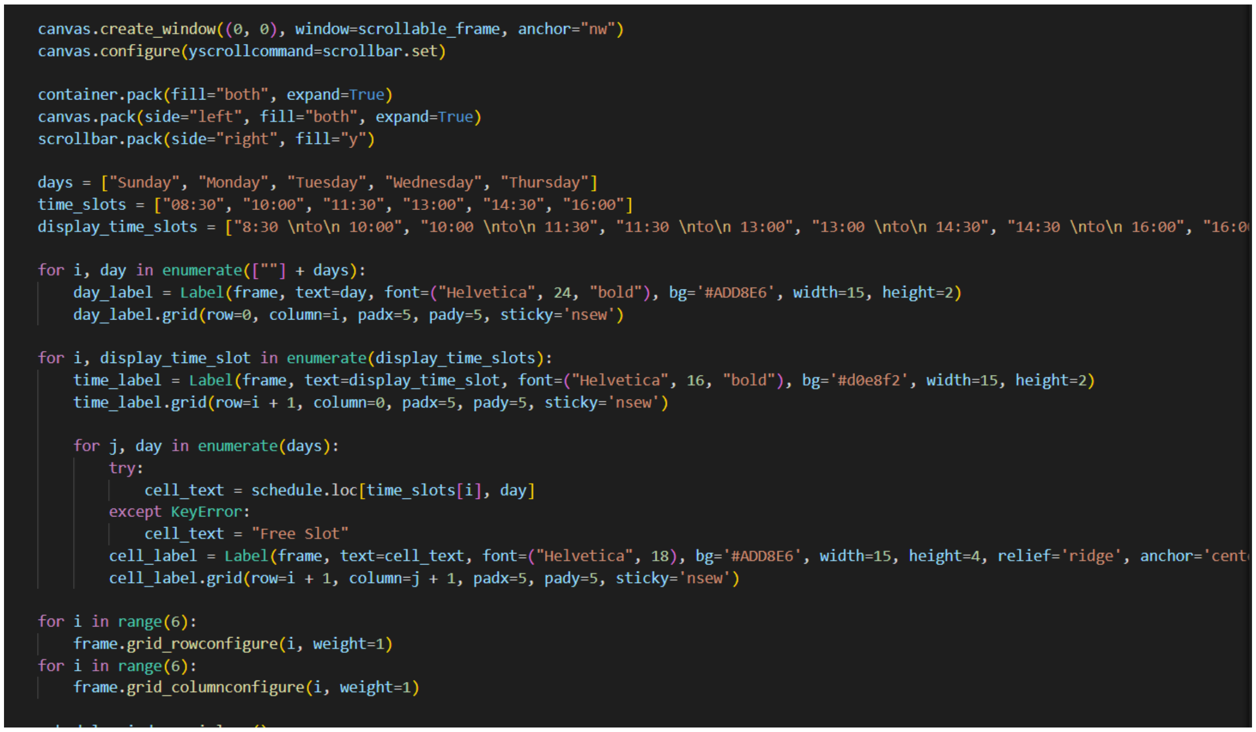


```python
def relative_to_assets(path: str) -> Path:
    return ASSETS_PATH / Path(path)
```

Helper function to get the relative path to assets, simplifying asset management.

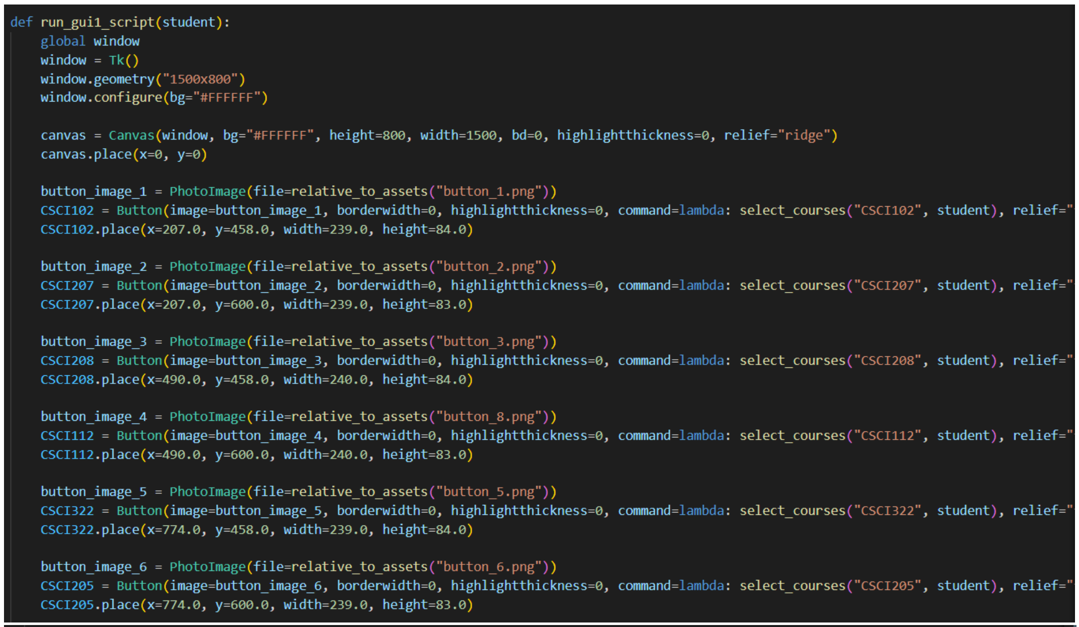


```python
def display_schedule(schedule):
    schedule_window = Tk()
    schedule_window.title("Generated Schedule")
    schedule_window.geometry("1200x1000")
    schedule_window.configure(bg='#FFFFFF')
    title_label = Label(schedule_window, text="Schedule Maker", font=("Helvetica", 48, "bold"), bg='#FFFFFF')
    title_label.pack(pady=10)

    frame = Frame(schedule_window, bg='#FFFFFF')
    frame.pack(pady=10)

    container = Frame(schedule_window)
    canvas = Canvas(container, bg='#FFFFFF')
    scrollbar = Scrollbar(container, orient="vertical", command=canvas.yview)
    scrollable_frame = Frame(canvas, bg='#FFFFFF')

    scrollable_frame.bind(
        "<Configure>",
        lambda e: canvas.configure(
            scrollregion=canvas.bbox("all")
        )
    )
```

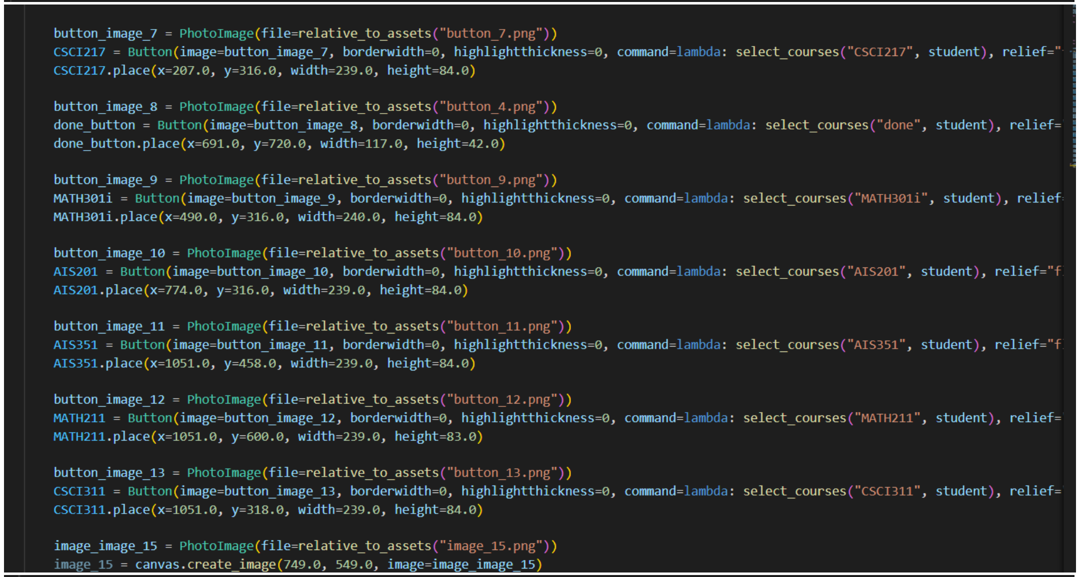


```python
    canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
    canvas.configure(yscrollcommand=scrollbar.set)

    container.pack(fill="both", expand=True)
    canvas.pack(side="left", fill="both", expand=True)
    scrollbar.pack(side="right", fill="y")

    days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday"]
    time_slots = ["08:30", "10:00", "11:30", "13:00", "14:30", "16:00"]
    display_time_slots = ["8:30 \nto\n 10:00", "10:00 \nto\n 11:30", "11:30 \nto\n 13:00", "13:00 \nto\n 14:30", "14:30 \nto\n 16:00", "16:0

    for i, day in enumerate([""] + days):
        day_label = Label(frame, text=day, font=("Helvetica", 24, "bold"), bg='#ADD8E6', width=15, height=2)
        day_label.grid(row=0, column=i, padx=5, pady=5, sticky='nsew')

    for i, display_time_slot in enumerate(display_time_slots):
        time_label = Label(frame, text=display_time_slot, font=("Helvetica", 16, "bold"), bg='#d0e8f2', width=15, height=2)
        time_label.grid(row=i + 1, column=0, padx=5, pady=5, sticky='nsew')

        for j, day in enumerate(days):
            try:
                cell_text = schedule.loc[time_slots[i], day]
            except KeyError:
                cell_text = "Free Slot"
            cell_label = Label(frame, text=cell_text, font=("Helvetica", 18), bg='#ADD8E6', width=15, height=4, relief='ridge', anchor='cent
            cell_label.grid(row=i + 1, column=j + 1, padx=5, pady=5, sticky='nsew')

    for i in range(6):
        frame.grid_rowconfigure(i, weight=1)
    for i in range(6):
        frame.grid_columnconfigure(i, weight=1)
```

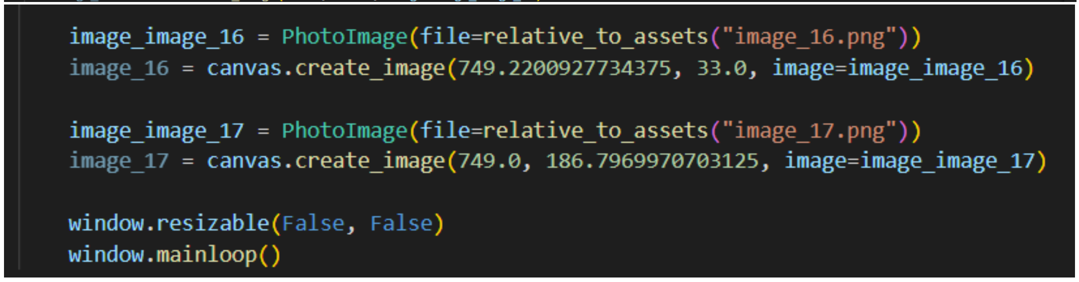
Creates a new window to display the generated schedule. Uses a Canvas and Scrollbar for scrolling capabilities. Populates a grid with course schedules for each day and time slot.



```
def run_gui1_script(student):
    global window
    window = Tk()
    window.geometry("1500x800")
    window.configure(bg="#FFFFFF")

    canvas = Canvas(window, bg="#FFFFFF", height=800, width=1500, bd=0, highlightthickness=0, relief="ridge")
    canvas.place(x=0, y=0)

    button_image_1 = PhotoImage(file=relative_to_assets("button_1.png"))
    CSCI102 = Button(image=button_image_1, borderwidth=0, highlightthickness=0, command=lambda: select_courses("CSCI102", student), relief="
    CSCI102.place(x=207.0, y=458.0, width=239.0, height=84.0)

    button_image_2 = PhotoImage(file=relative_to_assets("button_2.png"))
    CSCI207 = Button(image=button_image_2, borderwidth=0, highlightthickness=0, command=lambda: select_courses("CSCI207", student), relief="
    CSCI207.place(x=207.0, y=600.0, width=239.0, height=83.0)

    button_image_3 = PhotoImage(file=relative_to_assets("button_3.png"))
    CSCI208 = Button(image=button_image_3, borderwidth=0, highlightthickness=0, command=lambda: select_courses("CSCI208", student), relief="
    CSCI208.place(x=490.0, y=458.0, width=240.0, height=84.0)

    button_image_4 = PhotoImage(file=relative_to_assets("button_8.png"))
    CSCI112 = Button(image=button_image_4, borderwidth=0, highlightthickness=0, command=lambda: select_courses("CSCI112", student), relief="
    CSCI112.place(x=490.0, y=600.0, width=240.0, height=83.0)

    button_image_5 = PhotoImage(file=relative_to_assets("button_5.png"))
    CSCI322 = Button(image=button_image_5, borderwidth=0, highlightthickness=0, command=lambda: select_courses("CSCI322", student), relief="
    CSCI322.place(x=774.0, y=458.0, width=239.0, height=84.0)

    button_image_6 = PhotoImage(file=relative_to_assets("button_6.png"))
    CSCI205 = Button(image=button_image_6, borderwidth=0, highlightthickness=0, command=lambda: select_courses("CSCI205", student), relief="
    CSCI205.place(x=774.0, y=600.0, width=239.0, height=83.0)
```

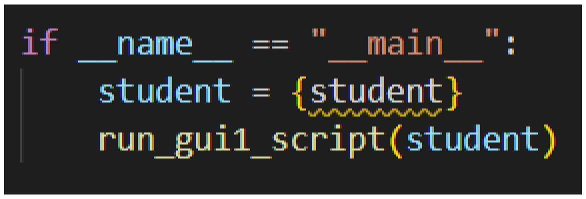


```
    button_image_7 = PhotoImage(file=relative_to_assets("button_7.png"))
    CSCI217 = Button(image=button_image_7, borderwidth=0, highlightthickness=0, command=lambda: select_courses("CSCI217", student), relief="
    CSCI217.place(x=207.0, y=316.0, width=239.0, height=84.0)

    button_image_8 = PhotoImage(file=relative_to_assets("button_4.png"))
    done_button = Button(image=button_image_8, borderwidth=0, highlightthickness=0, command=lambda: select_courses("done", student), relief=
    done_button.place(x=691.0, y=720.0, width=117.0, height=42.0)

    button_image_9 = PhotoImage(file=relative_to_assets("button_9.png"))
    MATH301i = Button(image=button_image_9, borderwidth=0, highlightthickness=0, command=lambda: select_courses("MATH301i", student), relief
    MATH301i.place(x=490.0, y=316.0, width=240.0, height=84.0)

    button_image_10 = PhotoImage(file=relative_to_assets("button_10.png"))
    AIS201 = Button(image=button_image_10, borderwidth=0, highlightthickness=0, command=lambda: select_courses("AIS201", student), relief="f
    AIS201.place(x=774.0, y=316.0, width=239.0, height=84.0)

    button_image_11 = PhotoImage(file=relative_to_assets("button_11.png"))
    AIS351 = Button(image=button_image_11, borderwidth=0, highlightthickness=0, command=lambda: select_courses("AIS351", student), relief="f
    AIS351.place(x=1051.0, y=458.0, width=239.0, height=84.0)

    button_image_12 = PhotoImage(file=relative_to_assets("button_12.png"))
    MATH211 = Button(image=button_image_12, borderwidth=0, highlightthickness=0, command=lambda: select_courses("MATH211", student), relief=
    MATH211.place(x=1051.0, y=600.0, width=239.0, height=83.0)

    button_image_13 = PhotoImage(file=relative_to_assets("button_13.png"))
    CSCI311 = Button(image=button_image_13, borderwidth=0, highlightthickness=0, command=lambda: select_courses("CSCI311", student), relief=
    CSCI311.place(x=1051.0, y=318.0, width=239.0, height=84.0)

    image_image_15 = PhotoImage(file=relative_to_assets("image_15.png"))
    image_15 = canvas.create_image(749.0, 549.0, image=image_image_15)
```



```
    image_image_16 = PhotoImage(file=relative_to_assets("image_16.png"))
    image_16 = canvas.create_image(749.2200927734375, 33.0, image=image_image_16)

    image_image_17 = PhotoImage(file=relative_to_assets("image_17.png"))
    image_17 = canvas.create_image(749.0, 186.7969970703125, image=image_image_17)

    window.resizable(False, False)
    window.mainloop()
```

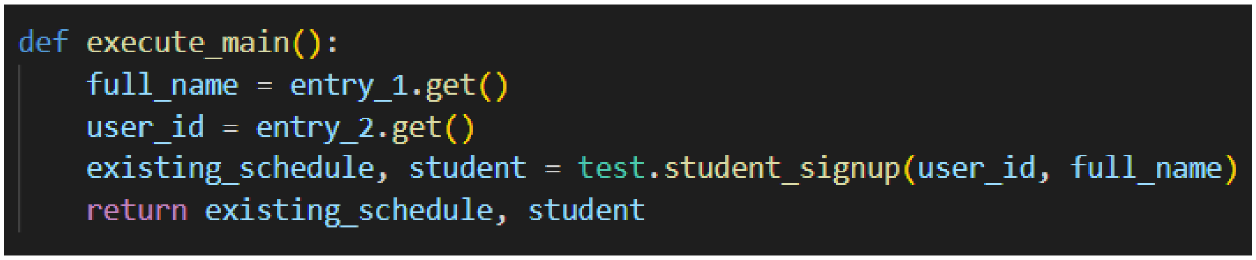
This function sets up the main GUI window. It defines a large canvas and places multiple buttons for course selection. Each button corresponds to a course and calls the select_courses function when clicked. An additional "done" button finalizes the course selection.



```python
if __name__ == "__main__":
    student = {student}
    run_gui1_script(student)
```

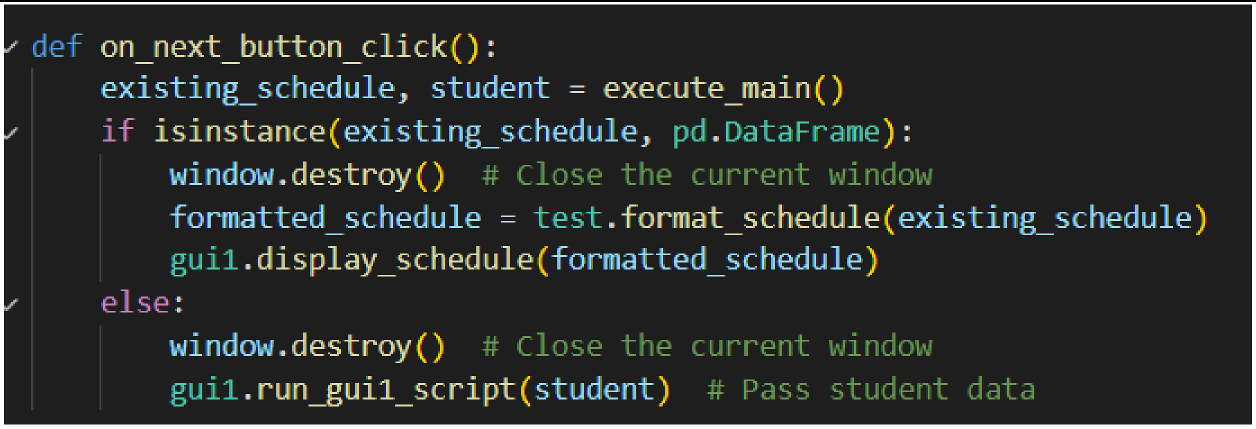
Defines the entry point of the script. It initializes the student dictionary and runs the run_gui1_script function to start the GUI application.

(gui.py)



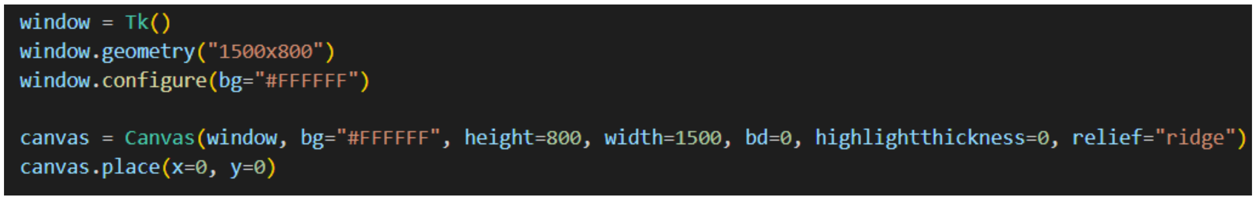
```python
def execute_main():
    full_name = entry_1.get()
    user_id = entry_2.get()
    existing_schedule, student = test.student_signup(user_id, full_name)
    return existing_schedule, student
```

Retrieves the full name and user ID from the entry fields, calls the student_signup function from the test module, and returns the existing schedule and student information.



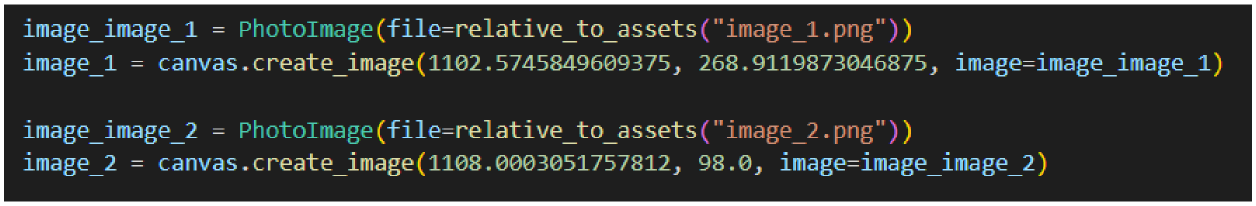
```python
def on_next_button_click():
    existing_schedule, student = execute_main()
    if isinstance(existing_schedule, pd.DataFrame):
        window.destroy()  # Close the current window
        formatted_schedule = test.format_schedule(existing_schedule)
        gui1.display_schedule(formatted_schedule)
    else:
        window.destroy()  # Close the current window
        gui1.run_gui1_script(student)  # Pass student data
```

Handles the click event for the "log in" button. It executes the execute_main function, checks if an existing schedule exists, and either displays the formatted schedule or runs the next part of the GUI to select courses.



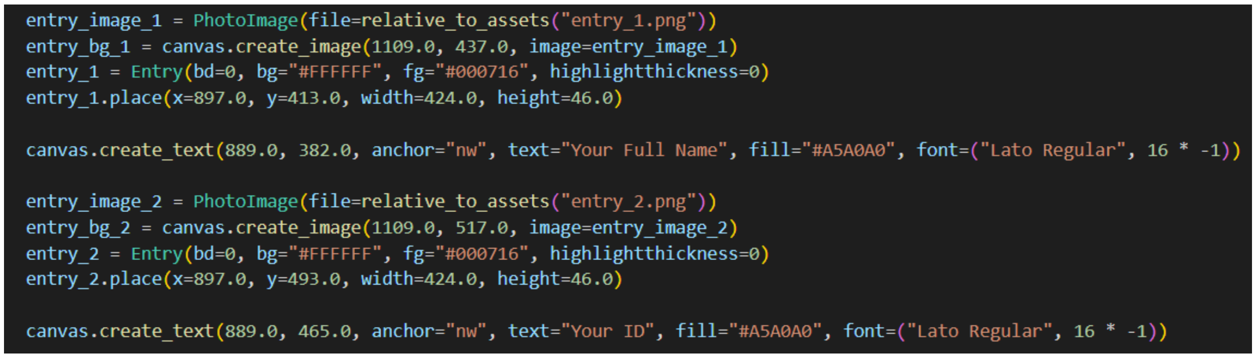
```python
window = Tk()
window.geometry("1500x800")
window.configure(bg="#FFFFFF")

canvas = Canvas(window, bg="#FFFFFF", height=800, width=1500, bd=0, highlightthickness=0, relief="ridge")
canvas.place(x=0, y=0)
```

Creates the main window and a canvas for placing GUI elements. The window is configured with a white background and a specific size.



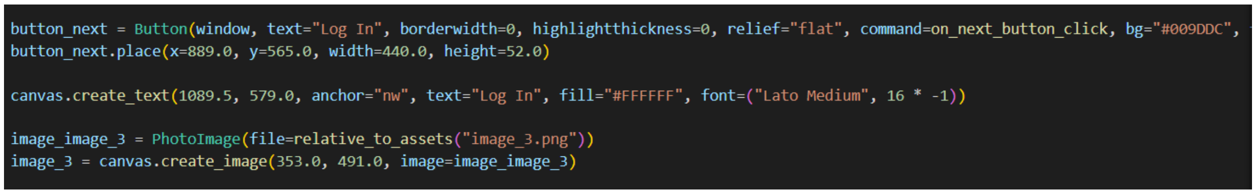
```
image_image_1 = PhotoImage(file=relative_to_assets("image_1.png"))
image_1 = canvas.create_image(1102.5745849609375, 268.9119873046875, image=image_image_1)

image_image_2 = PhotoImage(file=relative_to_assets("image_2.png"))
image_2 = canvas.create_image(1108.0003051757812, 98.0, image=image_image_2)
```

Loads and places images onto the canvas. These images are part of the GUI design and provide visual elements for the interface.



```
entry_image_1 = PhotoImage(file=relative_to_assets("entry_1.png"))
entry_bg_1 = canvas.create_image(1109.0, 437.0, image=entry_image_1)
entry_1 = Entry(bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
entry_1.place(x=897.0, y=413.0, width=424.0, height=46.0)

canvas.create_text(889.0, 382.0, anchor="nw", text="Your Full Name", fill="#A5A0A0", font=("Lato Regular", 16 * -1))

entry_image_2 = PhotoImage(file=relative_to_assets("entry_2.png"))
entry_bg_2 = canvas.create_image(1109.0, 517.0, image=entry_image_2)
entry_2 = Entry(bd=0, bg="#FFFFFF", fg="#000716", highlightthickness=0)
entry_2.place(x=897.0, y=493.0, width=424.0, height=46.0)

canvas.create_text(889.0, 465.0, anchor="nw", text="Your ID", fill="#A5A0A0", font=("Lato Regular", 16 * -1))
```

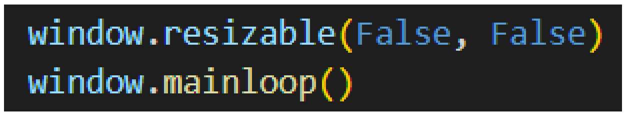
Creates entry fields for the user to input their full name and ID. These fields are placed on the canvas along with descriptive text.



```
button_next = Button(window, text="Log In", borderwidth=0, highlightthickness=0, relief="flat", command=on_next_button_click, bg="#009DDC",
button_next.place(x=889.0, y=565.0, width=440.0, height=52.0)

canvas.create_text(1089.5, 579.0, anchor="nw", text="Log In", fill="#FFFFFF", font=("Lato Medium", 16 * -1))

image_image_3 = PhotoImage(file=relative_to_assets("image_3.png"))
image_3 = canvas.create_image(353.0, 491.0, image=image_image_3)
```

-Creates a "Log In" button that triggers the on_next_button_click function when clicked. This button is styled and placed on the canvas.

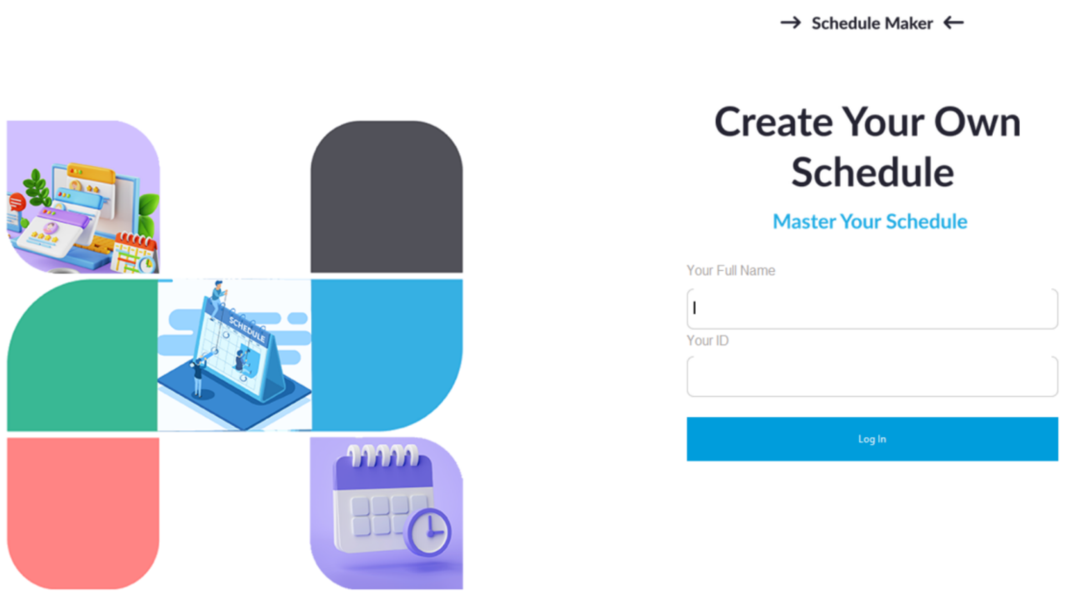
-Adds another image to the canvas as part of the GUI design.



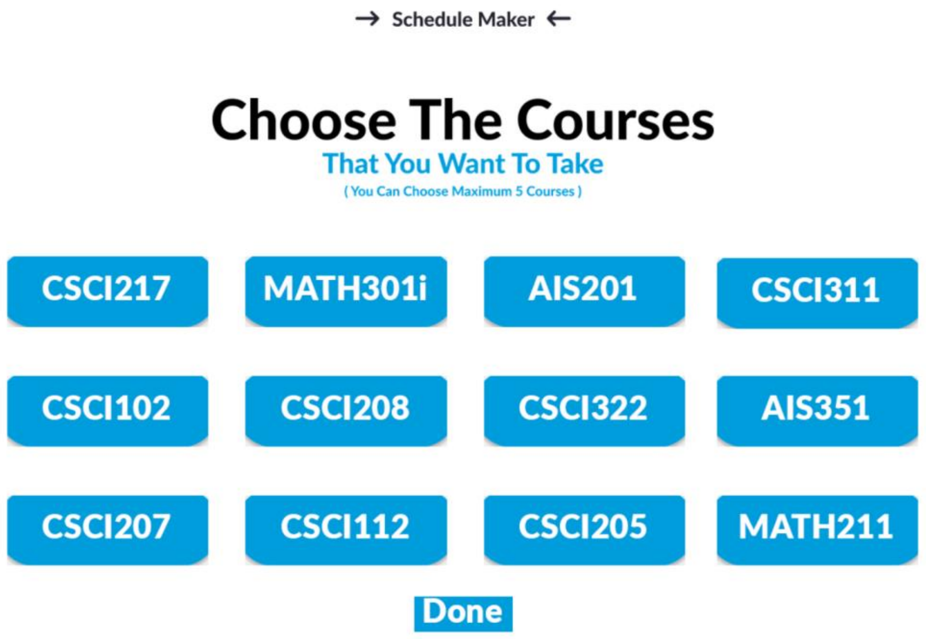
```
window.resizable(False, False)
window.mainloop()
```

Starts the Tkinter main loop, which keeps the window open and responsive to user interactions.

The Input:

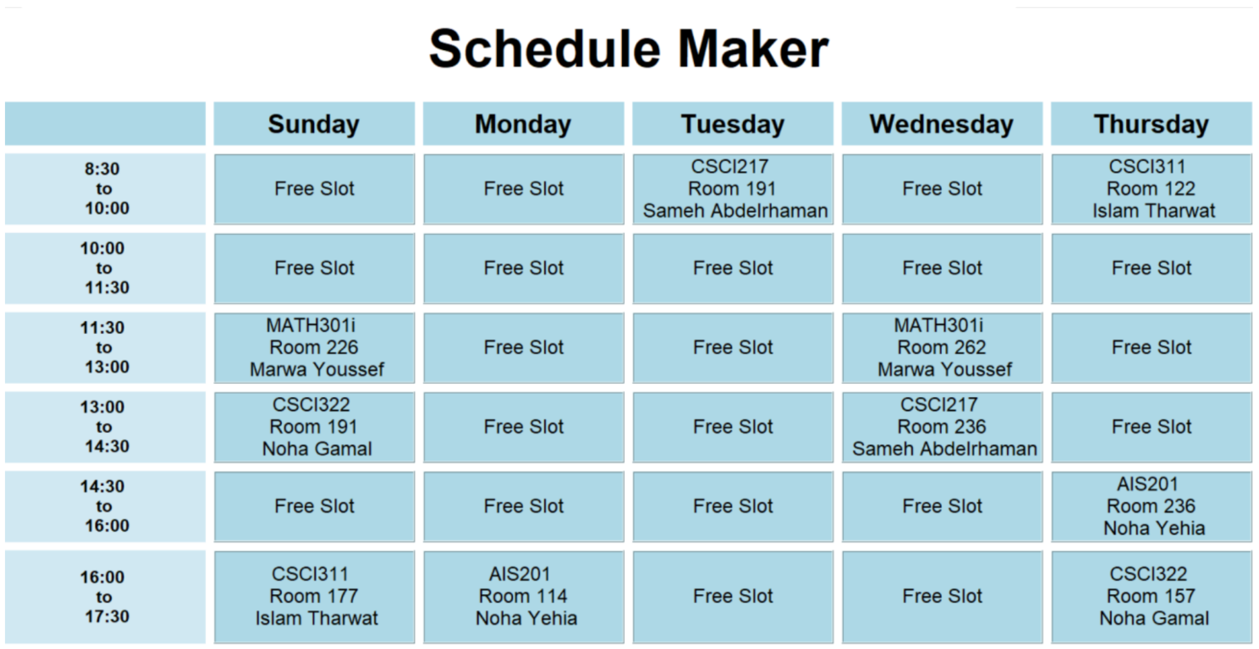


The input page to add the student name and id then press on Log in botton to go to the next page.



Choose the courses then click done or it will be clicked automatically after choosing more than 5 courses.

The Output:



Schedule Maker

	Sunday	Monday	Tuesday	Wednesday	Thursday
8:30 to 10:00	Free Slot	Free Slot	CSCI217 Room 191 Sameh Abdelrhaman	Free Slot	CSCI311 Room 122 Islam Tharwat
10:00 to 11:30	Free Slot	Free Slot	Free Slot	Free Slot	Free Slot
11:30 to 13:00	MATH301i Room 226 Marwa Youssef	Free Slot	Free Slot	MATH301i Room 262 Marwa Youssef	Free Slot
13:00 to 14:30	CSCI322 Room 191 Noha Gamal	Free Slot	Free Slot	CSCI217 Room 236 Sameh Abdelrhaman	Free Slot
14:30 to 16:00	Free Slot	Free Slot	Free Slot	Free Slot	AIS201 Room 236 Noha Yehia
16:00 to 17:30	CSCI311 Room 177 Islam Tharwat	AIS201 Room 114 Noha Yehia	Free Slot	Free Slot	CSCI322 Room 157 Noha Gamal

After the schedule is made it will be plotted in this page every course is plotted twice (lecture, sec) with the professor’s name and the room.

Literature Review for CSP Algorithms:

An essential subject in operations research and artificial intelligence is constraint satisfaction problems (CSPs). In CSPs, a problem is described by a set of variables, each of which has a particular domain of values, and a set of constraints that limit the values the variables may have simultaneously. The goal is to discover a solution that satisfies all the constraints.

Several algorithms have been developed to solve CSPs, each with unique strategies to handle constraints and optimize performance.

Backtracking Algorithm: This method is one of the simplest and most widely used for solving CSPs. It involves assigning values to variables sequentially and backtracking whenever a constraint is violated. Despite its simplicity, the basic backtracking algorithm can be inefficient due to its exhaustive nature. Heuristics and constraint propagation are two methods that have been added to the fundamental backtracking algorithm to improve it. These improvements aid in narrowing the search field and increasing efficiency by eliminating inconsistent values early in the search process [1].

Constraint Propagation: This technique reduces the range of values that may be assigned to each variable by spreading the restrictions throughout the variables' domains. Arc-consistency algorithms, such as AC-3 (Arc-consistency algorithm 3), ensure that all values within a variable's domain are compatible with the constraints [2].

Heuristics: Heuristics like the Degree Heuristic and Minimum Remaining Values (MRV) are frequently used with backtracking to increase performance. MRV selects the variable with the fewest remaining permissible values, which often leads to a faster solution [2].

Local Search Algorithms: These algorithms, such as Min-Conflicts, start with an arbitrary assignment and iteratively make local changes to reduce the number of constraint violations. They are particularly effective for large CSPs where complete search methods are impractical [4].

CSPs have numerous applications, including scheduling, timetabling, planning, and resource allocation. Classic problems like cryptarithmetic, Sudoku, and the N-Queens problem are commonly solved using CSP algorithms [3].

The evolution of CSP algorithms from basic backtracking to advanced techniques involving constraint propagation and heuristics has significantly improved the efficiency and applicability of solving constraint satisfaction problems. Ongoing research continues to enhance these methods, making them more robust and applicable to a broader range of real-world problems.

[1] https://www.researchgate.net/publication/271891537_An_Improved_Algorithm_of_CSP

[2] https://www.sciencedirect.com/science/article/abs/pii/S0377221798003646

[3] https://eprints.soton.ac.uk/36240/

[4] https://eprints.soton.ac.uk/36240/