Goals and Topics

The assignment problem is straightforward. All necessary details have been supplied. The solution to the problem will use the programming concepts and strategies covered in workshops 1-10 delivered in the course. The subgoals are:

- Obtaining an advanced understanding of values, variables, and lists.
- Understanding program input and output, functions, and expressions.
- Understanding simple strategies like iteration, validation, sum, and count.
- Understanding of advanced strategies like swapping, sorting, tallying, and searching.
- Translating simple design into Python code.
- The mechanics of editing, interpreting, building, and running a program.
- Testing a program.
- Commenting source code, especially Docstring on functions.
- Becoming confident and comfortable with programming small problems.

The Task

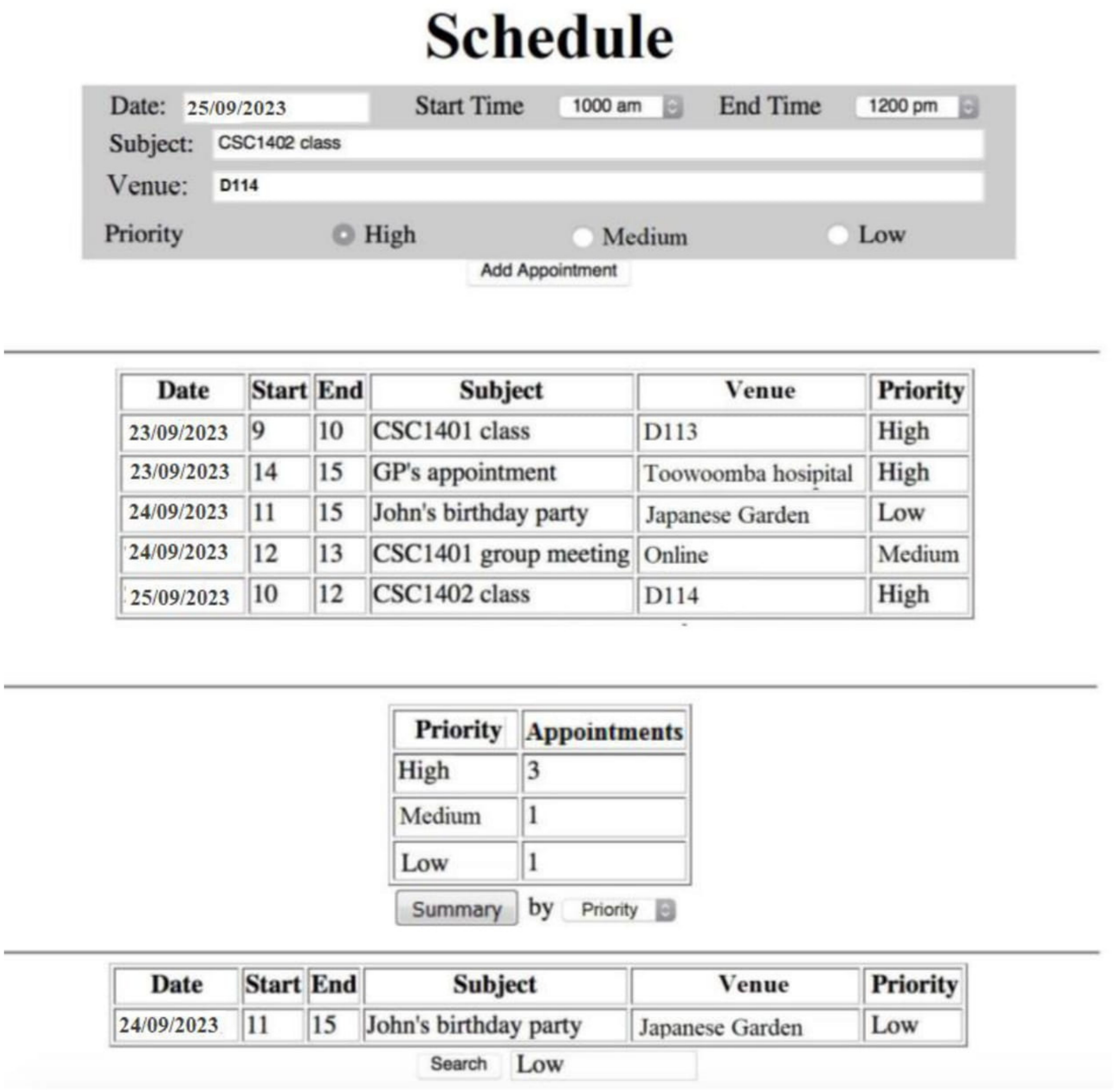
In this assignment, you are required to design, implement, and test a program that can be used to manage a simple personal Schedule with appointment records, which are stored in a list. Your program must provide an interactive design that allows the user to:

- create new appointment records and add them to the schedule.
- display all appointment records with details.
- tally all appointments for a summary of the schedule.
- search for specific appointment records in all records.

The program is to be implemented by Python and as a .ipynb file running on Jupyter notebook.

Schedule

Figure 1 illustrates the Schedule with an interactive editing environment on a webpage. Please use this illustration as the reference for the following descriptions. It is not necessary for you to present the interactive environment exactly as the Figure 1. The first section of input information can be implemented by 6 input functions and the outputs can be similar to Figure 2 (a “table” including headers, “--” and records in each row) and Figure 3.



Schedule

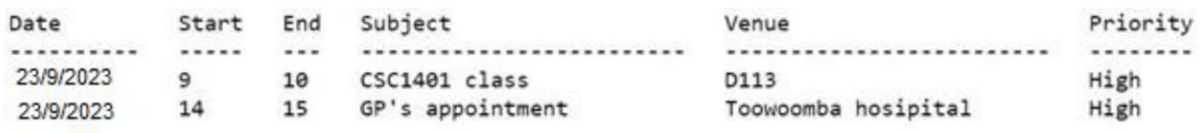
Date:	25/09/2023	Start Time	1000 am	End Time	1200 pm
Subject:	CSC1402 class				
Venue:	D114				
Priority	[✓ High]	Medium	Low		

Date	Start	End	Subject	Venue	Priority
23/09/2023	9	10	CSC1401 class	D113	High
23/09/2023	14	15	GP's appointment	Toowoomba hosipital	High
24/09/2023	11	15	John's birthday party	Japanese Garden	Low
24/09/2023	12	13	CSC1401 group meeting	Online	Medium
25/09/2023	10	12	CSC1402 class	D114	High

Priority	Appointments
High	3
Medium	1
Low	1

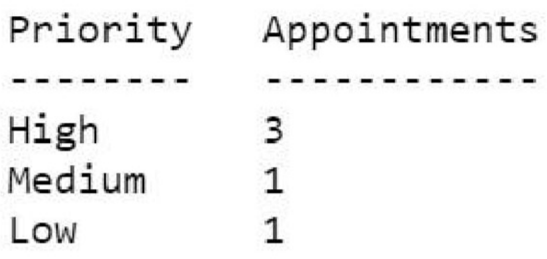
Date	Start	End	Subject	Venue	Priority
24/09/2023	11	15	John's birthday party	Japanese Garden	Low

Figure 1: The Interactive Editing Environment of Schedule



```
Date          Start   End   Subject                     Venue                       Priority
----------    -----   ---   -------------------------   -------------------------   --------
 23/9/2023    9       10    CSC1401 class               D113                        High
 23/9/2023    14      15    GP's appointment            Toowoomba hosipital         High
```

Figure 2: The example of the appointment records



```
Priority   Appointments
--------   ------------
High       3
Medium     1
Low        1
```

Figure 3: The example of the tallying results

Appointment records

An appointment record in this program will hold or encapsulate data for the date, start time, end time, subject, venue, and priority of the appointment. With respect to each of the attributes, an appointment record can be represented like the following sample:
"23/9/2023; 9; 10; CSC1401 class; D114; High"

This can be deciphered as the appointment with "High" priority, on the date of "23/9/2023", starting at "9:00 am" and finishing at "10:00 am", about the subject of "CSC1401 class" in "D114". You should create a global variable "list" to store all such appointments. For the sake of easy explanation, we refer to this variable by appointmentList in the rest of this document.

Some limitations must be imposed on the appointment records:
1. The primary unit of time in an appointment is the hour.
2. An appointment starts and finishes on the same day.
3. Only appointments occurring at a future time can be added to the Schedule.
4. No concurrent appointments are to be added to the Schedule.

Date

A date is valid if and only if:

- 9999>year> 2023
- 1 <= month <= 12
- 1 <= day <= daysInMonth(month, year)
- daysInMonth = 30 if the month is April, June, September, or November
- daysInMonth = 31 if the month is January, March, May, July, August, October, or December
- daysInMonth = 28 if the month is February and is not a leapYear(year)
- daysInMonth = 29 if the month is February and is a leap Year(year)

A year is a leap year if

- The year is divisible by 400
- The year is divisible by 4 and if the year is not divisible by 100

So, the year 2000 was a leap year, 2004 was a leap year but 2100 will not be a leap year.

Time

The time (Start time and End time) of an appointment is valid if and only if:

- 8 <= Start/End Time <= 18;
- Start<End
- ignore minutes and seconds, only integral point input.

Taking both date and time into account, **an appointment needs to start later than the time when the appointment is being added** to the Schedule.

Your Tasks

It is strongly suggested that the following approach is to be taken to design and implement the program.

Input Functions
You should first design, implement and test the input functions of the Schedule. Refer to Fig. 1 for the details you need to create 6 input functions. For example, the first input function is applied to enter a date with the hint “Please enter the date of your new appointment, e.g., 25/9/2023”. You are welcome to have your design of the 6 input functions, so as they input the date, start time, end time, subject, venue, and priority of the new appointments as the sample IDE illustrated in Fig. 1. The 6 inputs for one appointment will be a stored as a string (record) in the appointmentList.

All the functions except built-in functions should be presented with proper Docstrings.

The addRecord() Function
You should design, implement and test a function to add an appointment record to the Schedule. An appointment record will be added to the Schedule each time when the 6 inputs of the appointment record are all valid. If any input is invalid, alert an error message. The function handles the following tasks:

- Collect all data (date, start time, end time, subject, venue, and priority) for the appointment record (assigned them to a string as "23/9/2023; 9; 10; CSC1401 class; D114; High", other formats for the string are not acceptable for this assignment).
- Validate if the input for "Date" is correct regarding the specification in the Date section by using the function **isValidDate()** described below.
- Validate if the input for "Time" is correct regarding the specification in the Time section by using the **isValidTime()** described below.
- A non-empty string within 25 characters (including space between words) for the subject or venue
- “Low”, “Medium” and “High” are the valid inputs of priority, case insensitive.
- Call isConcurrentAppointment() first then add the valid appointment record into the appointmentList list if all data are valid.
- The program will repeatedly ask users to input the record until "END" is input for the date. Call showRecords() to present the table of all the records.

The showRecords() Function
You should design, implement and test a function to print all existing records in a “table” as the Figure 2 after the task of **Input Functions** is completed. The function should access the appointmentList and print all existing appointment records (no requirement about the order) in the table.

- a “table” including headers, “--” which separates the header from the content of each record.
- The number of “-” is equal to the maximum number of text or the length of the header for each column.
- Indentation is appropriate as shown in Figure 2.

<u>You could manually create some dummy appointment records and store them in the appointmentList to test this function, while other functions remain incomplete.</u>

The isValidDate() Function
You should design, implement, and test a function to validate the data input for the Date attribute of an appointment record. The function should alert an error message and return false if the input is invalid, otherwise, return true.

- Refer to the Date section for what the function needs to check for validation.

- To evaluate students' string handling capabilities, only the "25/9/2023" date format is a valid input format for 25th September 2023. The use of built-in functions or some libraries to input valid date is not acceptable for this assignment.
- You can revise this function's date validation code in your assignment1.

The isValidTime() Function
You should design, implement and test a function to validate the time input for an appointment record's start time and end time attributes. The function should alert an error message and return false if the input is invalid, otherwise, return true. Refer to the Time section for what the function needs to check for validation.

The isConcurrentAppointment() Function (Challenge task)
You should design, implement and test a function to validate if the input data for the Date, Start Time, and End Time of an appointment is concurrent to any existing appointments in the appointmentList. The time of two appointments is considered concurrent if the time span for the two appointments overlaps with each other in full or in part. Note that an appointment can start at a time when another finishes, or vice versa. An appointment starts and finishes on the same day.
The function should alert an error message and return true if the input appointment is concurrent with any existing appointments in appointmentList, otherwise, return false.

The tallyAppointments() Function
You should design, implement and test a function to calculate the number of appointments based on the Priority, and display the total number of appointment records of Schedule when the user enters the corresponding keyword. The program will continuously ask users whether to tally the records (e.g., the hints "Do you want to tally the appointments by priority". Only "YES" and "NO" are valid inputs, case insensitive) until "YES" is entered to print the tally results or "NO" is entered to stop the iteration.
You should use the following suggestion as a guideline:

Summary by priority:
- Find the appointments with different priorities by using a string search plan.
- Count how many records with "High", "Medium" and "Low" priorities and show the number by printing a table as in Figure 3.

You have to list the results following **a specific order of date (from High, Medium to Low).**

The searchRecord() Function
You should design, implement and test a function to search the appointment records using the keywords given by the user. The program will continuously ask users to input the search keywords (e.g., the hint "Please enter the keyword for searching".) until "END" is entered to stop the iteration.
- The keywords for searchRecord() Function are case insensitive. That means no matter whether the users search "Class" or "class", the program will show all the records including "class".

- Two examples:
  1. Once the users search "a", any record including the character "a" will be displayed no matter if the "a" is in "subject" or "venue". For example, both "Toowoomba" and "class" include character "a", so their relevant records will be displayed.

2. Once the users search "1", any record including the number "1" will be displayed. No matter the number "1" is for the day, month, year, start time, end time, subject, or venue.

You have to list the search results following a specific order of date (from an earlier date to later date) if the finding records are more than one as Figure 2.

Program Integration Test

You need to test the program for all functionality thoroughly before delivering the program to clients. The program should be running appropriately without any syntax or logic errors.

Submission

What You Need to Submit - Three Files.
For a complete submission, you need to submit three files as specified below. The assignment submission system will accept only the files with extensions specified in this section.

1. ***Statement of Completeness*** in a file saved in *.doc, .docx, .odt* format in 300-400 of your own words describes the following issues. You should first specify the registered team ID of the team (the teaching assistant will send you the ID after the team registration) and all members' names and student IDs on the top of the statement.
    - **The state of your assignment**, such as, any known functionality that has not been implemented, etc. (It is expected that most teams will implement all of the functionality of this assignment).
    - **Meeting attendance and code contribution summary** see Statement of Completeness (team-based) document
    - **Problems encountered**, such as, any problems that you encountered during the assignment work and how you dealt with them. This may include technical problems in programming and people-soft problems in team working;
    - **Reflection**, such as any lessons learnt in doing the assignment and handling team working and suggestions to future programming projects.
2. ***The program*** in a file saved with an *.ipynb* extension contains the source code implemented following the functional and non-functional requirements.
3. ***The program*** in a file saved with a *.doc/.docx/.odt* extension contains the exact same source code in the *.ipynb* file. You can just download the *.ipynb* file as *.py* file and paste code in it into the *.doc/.docx/.odt* file. If you don't submit your code in **both** *.doc/.docx/.odt* and *.ipynb* files, *or the codes* in *.doc/.docx/.odt* and *.ipynb* files are **different**. You will get a penalty (up to 10 marks).

Suggested Strategy

Plan to complete the assignment on time. Do not write all of the code in one sitting and expect that everything will be working smoothly like magic.

First step Form and register a team or check the registration status of your team if you have completed the registration process. Read the assignment specification carefully. Clarify anything unclear by putting a post on the Assignment 3 Public Forum. Have your team meeting regularly (face-to-face or online zoom meeting) for discussions on assignment requirements and team working style.
Have a discussion on the options of either meeting more challenges to designing the program by yourself or simply implementing the game based on the design provided. Make a decision on the options and then stick with it. Think about how to do it, how to test it, and devise high-level algorithms for each independent part of the assignment. Begin to type the program (with pseudocode), in incremental stages, and help each other in a team.

Second step Keep working on a team basis and have your team meeting regularly. Re-read the specification, continue to refine the design of various sections to code, and bring up any problems to the team for advice or to the public forum if necessary. By the end of the term, you should have had a working program and some integration tests done.

Third step Fully test the program; have another review of the source code; re-read the specification (especially marking criteria) to make sure you have done exactly what is required. Have a team meeting to discuss the experience. Write the "Statement of Completeness" and make the final submission.

Plagiarism and Academic Misconduct

USQ has zero tolerance for academic misconduct including plagiarism and collusion. Plagiarism refers to the activity of presenting someone else's work as if you wrote it yourself. Collusion is a specific type of cheating that occurs when two or more students exceed a permitted level of collaboration on a piece of assessment. Identical layouts, identical mistakes, identical arguments, and identical presentations in students' assignments are evidence of plagiarism and collusion. Such academic misconduct may lead to serious consequences.

Marking Criteria

You should use Table 1 as the checklist for implementation and to guide the testing of your program.

Table 1: Marking Criteria

ID	REQUIREMENTS If you don't submit your code in both .doc/.docx/.odt and .ipynb files, or the code in .doc/.docx/.odt and .ipynb files are different. You will get a penalty (up to 10 marks).	Mark
	Statement of Completeness (3 marks)	
1	The statement is in appropriate length of 300-400 of own words covering issues on both programming and team working. The "State of assignment", "Problems encountered" and "Reflection" are meaningful.	1
2	The "Meeting attendance" is presented clearly as required.	1

3	The "Code contribution summary" is presented clearly as required.	1
	Functional Requirements (16 marks)	
4	Input Functions are designed and implemented appropriately.	1
5	All the requirements for 6 Input Functions are met.	1
6	The addRecord() Function is designed and implemented appropriately.	1
7	All the requirements about addRecord() Function are met.	1
8	The showRecords() Function is designed and implemented appropriately.	1
9	All the requirements of the showRecords() Function are met.	1
10	The isValidDate() Functions are designed and implemented appropriately.	1
11	All the requirements for isValidDate() Functions are met.	1
12	The isValidTime() Functions are designed and implemented appropriately.	1
13	All the requirements for isValidTime() Functions are met.	1
14	The isConcurrentAppointment() Function is designed and implemented appropriately.	1
15	All the requirements of isConcurrentAppointment() Function are met.	1
16	The tallyAppointments() Function is designed and implemented appropriately.	1
17	All the requirements for tallyAppointments() Function are met	1
18	The searchRecords() Function is designed and implemented appropriately.	1
19	All the requirements of searchRecords() Function are met.	1
	Non-functional Requirements (1 mark)	
20	The script is running free from any syntax errors. Code has been cleaned, all testing statements (e.g., print-lining) are removed. Code in the script is indented correctly for all loops, if-else statements, and functions. Code is grouped based on common tasks. Each block of code is commented briefly. ALL functions should be formally commented appropriately (both style and content).	1
	Total	20