Project: Classes

Weight: 40%

Type: Group Assignment

- Students should **ONLY USE** programming constructs covered in the course material.
- **Submissions using programming concepts that are not covered in the course material will be penalized.**
    - **Penalty with no limit could be applied**
- **Late submissions will not be accepted**

Scenario

Hair Salon needs a system to track customer appointments. The salon's owner has hired your team to create an appointment management system which is customized to her salon's business needs.

Equipment and Materials

For this assignment, you will need:

- Python IDE
- Git software
- GitHub repository

Instructions

This assignment consists of two parts, both completed outside of class time. See the course schedule and Brightspace for exact dates. Rubrics are available in Brightspace.

Project Part 1: (No Submission)

1. Create the Appointment class as outlined in the document and verify its correct operation by running the test program provided.
2. The appointment class source code (recommended filename: *appointment.py*)
3. Test the appointment class and create a sample run (as a .txt, doc, or pdf file) by running the supplied *test_appointment.py* program, which outputs a single line of data and reports errors in the implementation of the class.
4. Each student is encouraged to individually complete Part 1. At least, group members MUST equally participate in Part 1.

Project Part 2: Submission

1. Have one member of your group set up a GitHub repository for Part 2 of this project. Make sure that this GitHub repository is **private**. Add group members and your instructor as collaborators.
2. Create a separate branch in GitHub for each group member, containing the part they will work on. The branch name should include the task name and the student name.
3. Develop the code for the Appointment Management Module as specified in this document. As a team, you will also need to assess each member's Appointment class and then finalize one version for your group's solution.
4. GitHub must be effectively used for group collaboration.
5. Ensure all group members update their parts to their respective GitHub branches.
6. The final project code must be pushed to master/main on GitHub.
7. Check your solution against the detailed marking criteria available in Brightspace.
8. Create a project video.
   Each group will submit a **video** to Brightspace or their GitHub Repository (either as a URL or an uploaded video file). In the video:
   - EACH team member will introduce themselves – camera should be on. Please use laptop camera/screen share to create video (not a phone).
   - Provide a demonstration of the application covering all the functionality CORRESPONDING TO THE SAMPLE RUN PROVIDED.
   - Share the **group's final solution** (Python code) on screen and have EACH team member explain a **portion of the code** that they developed. A debugger may be used to step through sections of code (optional).
9. Submit the following files to Brightspace:
   - A ZIP file for the whole project folder that includes:
     - The code of the program that you implemented (.py files).
     - A copy of your sample runs (.txt files) that correspond to the sample runs provided.
     - All data files updated by program after the test runs have been completed.
     - GitHub training certificate.
   - A link to your group's GitHub project repository
     - The GitHub project repository should include:
       - The code of the program that you implemented (.py files).
       - A copy of your sample runs (.txt files) that correspond to the sample runs provided.
       - All data files updated by program after the test runs have been completed.

- **Presentation video.**
- Peer assessment

Appointment Management System Details

The Hair Salon requires that the appointment management system track the following information for each appointment:

- Client name
- Client phone number
- Type of appointment:
    0. Available
    1. Mens cut **$40**
    2. Ladies cut **$60**
    3. Mens Colouring **$40**
    4. Ladies Colouring **$80**
- Day of week: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
- Start time of appointment (e.g. **9 AM**)

Note: all appointments are one hour long.

The appointment management system needs to provide the following functionality:

- create a one-week calendar of available appointments for Monday - Saturday, 9 AM – 4 PM daily (last appointment is 4-5 PM)
- allow the user to optionally load previously booked appointments into the week's calendar
- allow the user to select and perform any of the following menu functions (continuously until the user chooses to exit the system):
    - 1) Schedule an appointment
    - 2) Find appointment by name
    - 3) Print calendar for a specific day
    - 4) Cancel an appointment
    - 5) Change an appointment
    - 6) Calculate total fees for a day
    - 7) Calculate total weekly fees
    - 9) Exit the system
- allow the user to optionally save all scheduled appointments to a file

A senior software developer has already done some design work on the *Appointment* class as well as the core functions required for the appointment management module. Please refer to the

detailed information in the following two sections as you complete both programming parts of this project.

Part 1 – Appointment Class

Properties/Attributes (hidden)

client_name, client_phone, appt_type, day_of_week, start_time_hour

Methods

Method Name	Description
Constructor	• __init__() initializes the Appointment object properties. • The constructor should assign empty/zero for the first three properties (i.e. representing an available appointment) and accept two parameters for assigning to day_of_week and start_time_hour
Getters	• Implement one getter method for each Appointment property. The getter method should return the value of the property
get_appt_type_desc()	• An additional "getter" method that translates and returns the object's appt_type as a text description, I.e.: **appt_type** **description** 0 Available 1 Mens Cut 2 Ladies Cut 3 Mens Colouring 4 Ladies Colouring
get_end_time_hour()	• An additional "getter" method that returns the value of the object's start_time_hour + 1
Setters	• Implement a setter method for the first three Appointment properties: client_name, client_phone, and appt_type. The setter method should set the property to the parameter value received
schedule()	• Sets client_name. client_phone, and appt_type properties to the parameter values received
cancel()	• Resets client_name. client_phone, and appt_type properties to empty/zero
format_record()	• Returns a string representation of an Appointment object • This representation should include all appointment properties separated by a comma • Same format as in "appointments.csv" file, e.g.: `Harvey,403-233-3944,1,Saturday,09`

__str__()	• Returns a string representation of an Appointment object • This representation is formatted for display, e.g.: `Harvey  403-233-3944  Saturday  09:00 - 10:00  Mens Cut`

Part 2 - Appointment Management Module

Create a **separate module** (**appt_manager.py** file) for your appointment management application code which will be comprised of various functions, including a *main()* entry function that controls the overall processing. The functions listed below are required but you may use additional functions too.

Function Name	Description
create_weekly_calendar()	• main() calls it. • Receives an empty list of the Appointment's objects. • Clears the received list. • Iterates through each day of the week (Monday to Saturday) • For each day, iterates through each available hour (9 to 16) • For each hour, creates new Appointment object and adds it to the appointment list (i.e. calendar)
load_scheduled_appointments()	• main() calls it if the user wants to initially load the scheduled appointments from a file. • Inputs appointment filename from user • Ensure that the file exists. • Iterates over each line (i.e. appointment) in the file, parsing the attribute values into separate variables • Calls find_appointment_by_time() to locate the corresponding appointment in the list and invoke the schedule() method to set the properties appropriately • Returns the number of scheduled appointments loaded
print_menu()	• main() calls it after the step of loading the scheduled appointments from a file. • Displays the menu of application options • Accepts menu selection from user until valid selection is entered • Returns user's **valid** selection

find_appointment_by_time()	• Receives the list of Appointment's objects, the day, and start hour of the appointment to find • Searches the list of Appointments for corresponding day and start hour • If the appointment is found, returns the Appointment object, otherwise returns nothing
show_appointments_by_name()	• Receives the list of Appointment's objects and the client name of the appointment(s) to show • Searches the list of Appointments for corresponding client name, allowing for partial & non-case sensitive matches • Displays all matching appointments in the format given in the Sample Run (hint: use the __str__() method implicitly)
show_appointments_by_day()	• Receives the list of Appointment's objects and the day of the appointments to show • Searches the list of Appointments for the corresponding day • Displays all matching appointments in the format given in the Sample Run (hint: use the __str__() method implicitly)

change_appointment_by_day_time()	• Receives the list of Appointment's objects
	• Asks the user to enter the day and the start hour for the appointment which will be changed
	• After ensuring that the appointment exists in the calendar:
	  • Asks the user for the new day and new start hour and ensure that this appointment is available
	  • If it is available, the calendar will be updated.
	  • For the new appointment, the client name, phone, and the appointment type should be set to the client name, phone, and appointment type associated with the old appointment.
	  • For the old appointment, the client name and phone will be eliminated and the appointment type will be set to available.
	• Finally displays a confirmation message which includes the client name and the new appointment day and time.

calculate_fees_per_day()	• Receives the list of Appointment's objects. • Asks the user to enter the day for which total fees will be calculated.

	• If the day is valid:
	  ◦ Iterates through the list of Appointment's objects and search for the entered day
	  ◦ Finds the fee for each appointment in this day and adds it to the total.
	  ◦ Finally, displays the total fees for this day.
	• Otherwise, display a message indicating either the day is invalid of the salon is closed.

calculate_weekly_fees()	• Receives the list of Appointment's objects. • Iterates through the list of Appointment's objects and • Finds the fee for each appointment adds it to the total. • Finally, displays the total fees for this day.
save_scheduled_appointments()	• Called when exiting the system. • Receives the list of Appointment's objects • Inputs appointment filename from user, checks if the file already exists and if so, allows user to proceed (i.e. overwrite the file) or repeat the filename input • Iterates over each appointment in the list and if scheduled (i.e. appt_type != 0), writes the appointment to the file in the proper CSV format (hint: use the format_record() method) • Returns the number of scheduled appointments saved

main()	• Entry point for the Appointment Management System
	• Coordinates the overall processing:
	  ◦ Set up a list of appointments for the week
	  ◦ Optionally load a previous appointment schedule
	  ◦ Present the menu, get and evaluate user's selection, repeating until user chooses to quit:
	  ▪ Get additional user inputs as needed
	  ▪ Call appropriate functions and/or Appointment class methods to help perform the actions for each menu option
	  ▪ Display relevant context/status messages
	  ◦ Optionally save scheduled appointments to a file before ending program

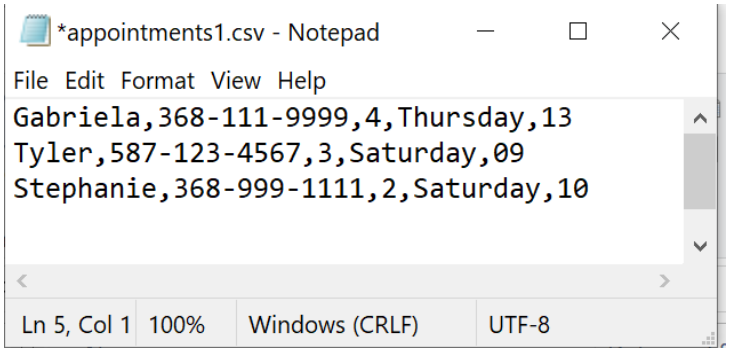
Note that except for *print_menu()* and *main()*, the appointment management module functions listed above should all be defined to receive a list of Appointment objects as a parameter. Of course, the *find_** and *show_** functions may require an additional parameter or two as indicated in the table above.

Sample Run: see *Project Sample Run F2024 document*.

Make sure your program output matches this sample run.

Starting Data File: see *appointments1.csv*.

Make sure that your application maintains the same comma-separated format established in this file.



Peer Assessment

Please accurately use and complete the following table. Each member should assign a mark out of 10 to other group members.

- The peer assessment should be completed accurately and honestly
  - Deduction will be applied if it is not
- All members should equally participate in the project.
  - The grade of each member will be based on her/his participation
- Each member should assess the contribution/participation of all other group members.
- Each member should assign a mark out of 10 to other members.
- Please use the following table to complete the peer assessment. Marks/tasks are just samples.
  - A deduction will be applied when using a different format

	Member 1 Name	Member 2 Name	Member 3 Name	Tasks Description

Member 1 Name	--	9	10	Task 1 Description
Member 2 Name	10	--	10	Task 2 Description
Member 3 Name	10	9	--	Task 3 Description
Average	10	9	10	

Grading

Item	Percentage
Working Code	**60**
Style	**10**
Documentation	**5**
Testing	**10**
Video	**5**
Version Control and GitHub training certificate	**7**
Pee Assessment	**3**
Total	**100**

Marking Criteria

Rubric available on Brightspace.