Note: For all programming questions, you must use Python 3.2.3 or higher. Consider using the design recipe and Python style guide available from the course website.

Programming Component
Approximately 80% of the marks of each question will be allocated to correctness, with the remaining 20% to style, including aspects of the design recipe.

1. (60 marks) An appointment book (for a single year) is used for booking appointments. Suppose appointments can only be made at full or half clock between 8AM and 5PM every day. When making appointments, we need to record date, time and purpose of the appointment. Given the following list of operations, implement AppointmentBook ADT.
(Note: To simplify the implementation, we represent dates using an integer number between 1 and 365; time using a float number, e.g. 8.0 for 8AM, 13.5 for 1:30PM. Assume that we always have the right TYPE of input. But you may need to check the VALUE of the inputs. You may have helper methods as needed. Use a1q1.py to submit your solution for this question.)

- AppointmentBook(): creates an empty appointment book.
- isAppointment(apptDate, apptTime): determines if an appointment exists for the date and time specified. Returns True if an appointment exists, False otherwise. (10 marks)
- makeAppointment(apptDate, apptTime, purpose): inserts the appointment for the date, time and purpose specified as long as it does not conflict with an existing appointment (a conflict occurs if two appointments are scheduled at the same date and time). If a conflict occurs, no changes to the ADT should occur. Returns True if successful, False otherwise. (10 marks)
- cancelAppointment(apptDate, apptTime): deletes the appointment for the date and time specified. Returns True if successful, False otherwise. To successfully cancel the appointment, there has to be an existing appointment at the date and time specified. (10 marks)
- checkAppointment(apptDate, apptTime): retrieves the purpose of the appointment at the given date and time, if one exists. Otherwise, returns a Null string. (10 marks)
- **changeAppointment(oldDate, oldTime, newDate, newTime):** change the date or time for an appointment. **Returns True if successful, False otherwise.** Print information on the screen such as: “You already have an appointment at ... on ...” or “Your appointment has been rescheduled to ... on ...” or “You do not have an appointment at ... on ...”. (10 marks)
- **getAppointmentsByDate(date):** retrieves all the appointments on the given date. 
  Returns a list of tuples containing time and purpose. This list should be ordered so that earlier appointments come before later appointments. An empty list returns if no appointment on the date. (10 marks)

Note: For Q2 and Q3, we are going to use our sample implementation of AppointmentBook ADT to test, since based on ADT definitions, they should be implementation independent. Use `a1q2.py` and `a1q3.py` to submit your solution. Assume all the appointment records in the text file are in correct format. They should be one appointment per line. Each record starts with a command within the command list: “Make”, “Cancel”, “Change”. A sample appointment record file is provided. When we test your solution, we will use a longer file.

2. (10 marks) Write a function `buildApptBook`, which reads in a text file `appointments.txt` and uses the AppointmentBook ADT. Eg.

```
Make 45 10.5 meeting
Cancel 80 14.5
Change 80 15 85 15
Make 120 12 meeting
......
```

3. (10 marks) Write a function `busiestDate`, using AppointmentBook ADT. It returns a list containing the date(s) that have the largest number of appointments in ascending order.

**Writing Component**

4. (20 marks) Suppose that we are given an array \( A \) of \( n > 0 \) integers such that each of them is in the range \( 0, 1, ..., k \) for some positive integer \( k \in O(n) \). We want to determine if there exist an integer that appears more than once in \( A \).
   - Design and analyze an algorithm that runs in \( O(n \log n) \) worst-case time. (10 marks)
   - Design and analyze a better algorithm that runs in \( O(n) \) worst-case time. (10 marks)

For each algorithm, you may use an \( O(n) \) space in addition to input array \( A \).