FINAL PROJECT

COMP202, Winter 2021

Due: Friday, April 30^{th}, 11:59pm

Please read the entire PDF before starting. You must do this final project individually.

Question 1:	100 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment through automated tests.

To get full marks, you must:

- Follow all directions below.
  - In particular, make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, you will not be awarded points for those modules/functions.
- Make sure that your code runs.
  - Code with errors will receive a very low mark.
- Write your name and student ID as a comment at the top of all `.py` files you hand in.
- Name your variables appropriately.
  - The purpose of each variable should be obvious from the name.
- Comment your work.
  - A comment every line is not needed, but there should be enough comments to fully understand your program.
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing. Vertical spacing is also important when writing code. Separate each block of code (also within a function) with an empty line.

Please note that the final project must be submitted by April 30th. There are NO late days available for the final project. If you do not submit the project by April 30th, you will be receiving a 0 on this assessment and your final course grade will be computed accordingly. Please also note that it is NOT necessary to submit the project in order to pass the course.

The Final Project

In this project you will be implementing several modules that will allow you to simulate a booking system for hotels.

This project will allow you to better understand how to read and write to files, how to write objected oriented programs, and how to plot data using Matplotlib.

For full marks

Note that the project is designed for you to be practicing what you have learned in the videos up to and including Week 13. For this reason, you are NOT allowed to use anything seen after Week 13 (e.g., NumPy) or anything not seen in class at all. You will be heavily penalized if you do so.

Examples

For each function/method, we provide **examples** of how your code should behave. All examples are given as if you were to use them within your docstring.

When you upload your code to codePost, some of these examples will be run automatically to check that your code outputs the same as given in the example. However, **it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your project, we will run additional, private tests that may use inputs not seen in the examples.

Furthermore, please note that your code files **should not contain any function calls in the main body of the program** (i.e., outside of any functions). Code that does not conform in this manner will automatically fail the tests on codePost and **be heavily penalized**. It is OK to place function calls in the main body of your code for testing purposes, but if you do so, make certain that you remove them before submitting. Please review what you have learned in video 5.2 if you'd like to add code to your modules which executes only when you run your files.

About the data

With this PDF you will be given a zip file containing a folder called `hotels`. Unzip this file so that the `hotels` folder is in the same folder as the Python code files that you will make for this project.

Inside the `hotels` folder, there will be one folder for each hotel that your booking system will manage. At the moment, there are two hotels, but you can add more for testing purposes. (Note that some examples for the methods below require that you test with the given `hotels` folder with no modifications.)

Inside each hotel folder, there will be a `hotel_info.txt` file. This file contains the name of the hotel on the first line, followed by one line for each room in the hotel. Each of these lines will have three pieces of data separated by commas: the room number, the room type, and the price per night.

Also inside each hotel folder will be a number of CSV files, one for each month in which the hotel has rooms to be reserved. The name of each file will have the year, followed by an underscore, followed by the month (e.g., `1975_Oct.csv`). Inside each CSV file will be one row for every room in the hotel. The first column will indicate the room name. Subsequent columns will indicate the reservation, if any, for the room on the day corresponding to the column (column 1 is day 1, column 30 is day 30, etc.). The reservation will be indicated by the booking number, followed by two hyphens, followed by the name of the person who reserved the room (e.g., `9998701091820--Jack`).

Note that CSV stands for comma-separated values file. What this means is that each line of the file contains what is called a data record. Each record consists of strings, separated by commas. The commas are what creates the columns. You need to keep this in mind when reading/writing a csv file. (UPDATED on Apr 17)

Safe Assumptions

You can assume the following:

- Every column of the data will be present, even though some data might be missing (i.e., some columns may simply contain an empty string).
- There are no spelling mistakes.
- The order in which the data appears follows the format described above.
- There are no entries (i.e., no rows) with the same room number.
- Your functions will always be tested with correct input, so no input validation is required, unless explicitly stated in the description of the functions/methods.
- If not otherwise specified, functions/methods working with strings should be case sensitive. (UPDATED on Apr 17)

Objects of type `date`

In this project you will be working with objects of type `date`. Python offers a module called `datetime` that can help you with this. Here are few things you can do with objects of type `date`:

- You can create and use `date` objects as follows:

```
>>> import datetime
>>> date1 = datetime.date(2021, 4, 16) # Year, month, day
>>> print(date1.year)
2021
>>> date2 = datetime.date(2021, 4, 30)
>>> print(date2.month)
4
```

- You can subtract two `date` objects. The result is a `timedelta` object, which has one attribute: `days`. This is how many days apart the two dates are. For example:

```
>>> diff = date2 - date1
>>> diff.days
14
```

- A `__str__` method has been implemented for objects of type `date`. The method returns the year, the month, and the day of the date joined together with a hyphen. For example:

```
>>> print(date1)
2021-04-16
```

- The operators `<`, `>` and `==` have been overloaded. A date is considered to be smaller than another date if it happens to represent an earlier date. It is considered to be larger if it represents a later date. For example:

```
>>> date1 < date2
True
>>> date2 > date1
True
>>> date1 > date2
False
>>> date3 = datetime.date(2021, 4, 16)
```

```
>>> date1 == date3
True
```

- A `ValueError` is raised whenever one tries to create an object of type `date` with inputs that are considered to be out of range. For example:

```
>>> datetime.date(2021, 1, 35)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: day is out of range for month

>>> datetime.date(2021, 0, 12)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: month must be in 1..12

>>> datetime.date(2021, 2, 29)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: day is out of range for month

>>> datetime.date(2020, 2, 29) # No error is raised because 2020 was a leap year!
datetime.date(2020, 2, 29)
```

- You can read more here: https://docs.python.org/3/library/datetime.html

Filesystem paths

In this project you will need to open files for reading and writing that are contained within sub-directories of the current folder.

So far, when we read and write to files, the files have always been in the same folder as our Python code files. If we want to open a file that is within a sub-directory (that is, within a folder that is in the current folder), then we must indicate the sub-directory's name in the filename that we give to the `open` function.

For example, if there is a file `x.txt` inside a folder called `files`, and the folder called `files` is in the same folder as our Python code file, then to open `x.txt`, we would use `files/x.txt` as the file path given to the `open` function.

If a file is contained within a sub-directory of a sub-directory, then we just extend the path by placing another forward slash and the other sub-directory name. For example, if we want to open the `hotel_info.txt` file, which is inside a folder called `overlook_hotel`, which itself is in a folder called `hotels`, and our Python code file is in the same folder as the `hotels` folder, then to open the file we would use `hotels/overlook_hotel/hotel_info.txt` as the file path given to the `open` function.

Question 1: Hotel reservation system (100 points)

Room

Create a module called `room.py` and place your name and student ID at the top. All the functions in this section will go inside this module. You may not import any modules other than `doctest` and `datetime`.

Inside this module define the two following global variables at the top of the file. Note that we define these global variables in all uppercase because they are 'constants': variables that will never change throughout the execution of the program (although the programmer may decide to alter them between executions).

- `MONTHS = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']`
- `DAYS_PER_MONTH = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]`

Then, create a class called `Room`. This class should have the following:

- Instance attributes: `room_type` (a string), `room_num` (an int), `price` (a float), `availability` (a dictionary mapping tuples of integers representing the year and the month (i.e., (year, month)) to list of booleans).
- Class attribute: `TYPES_OF_ROOMS_AVAILABLE` initialized with the following list

  `['twin', 'double', 'queen', 'king']`

  UPDATED on Apr 17: Fixed the last element in the list to be `'king'`
- A constructor that takes as input a string, an integer and a float representing the room type, number, and price per night respectively. The constructor uses these inputs to initialize all the instance attributes accordingly. The `availability` attribute should be initialized with an empty dictionary. Note that the constructor should raise an `AssertionError` if:
  - The type of any of the inputs does not match the expected one.
  - The room type does not match one of the available types (ignoring the case of the characters).
  - The room number is not positive.
  - The price is negative.
- A `__str__` method that returns a string representation of a room containing the room number, the type, and the price in the specified order. Each piece of information in the string should be separated by a comma.

  For example,

  ```
  >>> my_room = Room('Double', 237, 99.99)
  >>> str(my_room)
  'Room 237,Double,99.99'
  ```

- An instance method called `set_up_room_availability` which takes as input a list of strings representing months, and an integer representing the year. This method updates the `availability` attribute of the room. For each month, the corresponding tuple of integers is created and mapped to a list of booleans. This list will have `None` as a first element, followed by as many elements as the days in the specified month, all with value equal to `True`. Please remember that in leap years, February has 29 days instead of 28. You can assume that the inputs to this method are valid: i.e. the integer representing a year is a positive one, and the list contains strings in the same format as those used in the global variable `MONTHS`.

  Apr 22: Note that a leap year is one that is cleanly divisible by 4 (e.g., 2024, 2028, 2032, ...). However, a year that is divisible by 4 and 100 but also be divisible by 400 to be a leap year (so 1600 and 2000 were leap years, but 1700, 1800 and 1900 were not).

  You cannot assume that the month strings in the argument list are ordered.

  For example,

  ```
  >>> r = Room("Queen", 105, 80.0)
  >>> r.set_up_room_availability(['May', 'Jun'], 2021)
  >>> len(r.availability)
  2
  >>> len(r.availability[(2021, 6)])
  ```

```
31
>>> r.availability[(2021, 5)][5]
True
>>> print(r.availability[(2021, 5)][0])
None
```

- An instance method called `reserve_room` which takes as input an object of type `date`. The method does not return any value, but it updates the availability of the room accordingly. An `AssertionError` should be raised if the room is not available at the given date.

  For example,

```
>>> r = Room("Queen", 105, 80.0)
>>> r.set_up_room_availability(['May', 'Jun'], 2021)
>>> date1 = datetime.date(2021, 6, 20)
>>> r.reserve_room(date1)
>>> r.availability[(2021, 6)][20]
False
>>> r.availability[(2021, 5)][3] = False
>>> date2 = datetime.date(2021, 5, 3)
>>> r.reserve_room(date2)
Traceback (most recent call last):
AssertionError: The room is not available at the given date
```

- An instance method called `make_available` which takes as input an object of type `date`. The method does not return any value, but it updates the availability of the room at the given date to be `True`.

  For example,

```
>>> r = Room("Queen", 105, 80.0)
>>> r.set_up_room_availability(['May', 'Jun'], 2021)
>>> date1 = datetime.date(2021, 6, 20)
>>> r.make_available(date1)
>>> r.availability[(2021, 6)][20]
True
>>> r.availability[(2021, 5)][3] = False
>>> date2 = datetime.date(2021, 5, 3)
>>> r.make_available(date2)
>>> r.availability[(2021, 5)][3]
True
```

- An instance method called `is_available` which takes as input two objects of type `date`, one representing the check in date, and the other representing the check out date. The method returns `True` if the room is available every night from the first date (included), to the second date (excluded). It returns `False` otherwise. An `AssertionError` should be raised if the first date does not happen to be an earlier date than the second. If the availability of the room has not been set up for at least one of the dates that need to be checked, then the method returns `False`. Please note that you cannot assume that the two input dates belong to the same month, nor the same year.

  For example,

```
>>> r1 = Room("Queen", 105, 80.0)
>>> r1.set_up_room_availability(['May', 'Jun'], 2021)
>>> date1 = datetime.date(2021, 5, 25)
>>> date2 = datetime.date(2021, 6, 10)
```

```
>>> r1.is_available(date1, date2)
True
>>> r1.availability[(2021, 5)][28] = False
>>> r1.is_available(date1, date2)
False
```

- A static method called `find_available_room` which takes as input a list of objects of type `Room`, a string representing a room type, and two dates, the first representing the check in date and the second the check out date. An `AssertionError` should be raised if the check in date does not happen to be earlier than the check out date. Otherwise, the method returns the first `Room` from the input list which happens to be available for the specified dates and is of the correct type. If there is no such room, then the method returns `None`.

  For example,

```
>>> r1 = Room("Queen", 105, 80.0)
>>> r2 = Room("Twin", 101, 55.0)
>>> r3 = Room("Queen", 107, 80.0)
>>> r1.set_up_room_availability(['May'], 2021)
>>> r2.set_up_room_availability(['May'], 2021)
>>> r3.set_up_room_availability(['May'], 2021)
>>> r1.availability[(2021, 5)][8] = False
>>> r = [r1, r2, r3]
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 10)
>>> my_room = Room.find_available_room(r, 'Queen', date1, date2)
>>> my_room == r3
True
>>> r3.availability[(2021, 5)][3] = False
>>> my_room = Room.find_available_room(r, 'Queen', date1, date2)
>>> print(my_room)
None

>>> r = Room("King", 110, 120.0)
>>> r.set_up_room_availability(['Dec'], 2021)
>>> r.set_up_room_availability(['Jan'], 2022)
>>> date1 = datetime.date(2021, 12, 20)
>>> date2 = datetime.date(2022, 1, 8)
>>> my_room = Room.find_available_room([r], 'Queen', date1, date2)
>>> print(my_room)
None
>>> my_room = Room.find_available_room([r], 'King', date1, date2)
>>> my_room == r
True
```

Reservation

Create a module called `reservation.py` and place your name and student ID at the top. All the functions in this section will go inside this module. You may not import any modules other than `doctest`, `datetime`, `random`, and `room`.

Please note that all examples below assume the following import statement has been added inside this module:

```
from room import Room, MONTHS, DAYS_PER_MONTH
```

(UPDATED on Apr 17)

Inside this module create a class called `Reservation`. This class should contain the following:

- Instance attributes: `booking_number` (an integer), `name` (a string), `room_reserved` (a `Room`), `check_in` (a `date`), and `check_out` (a `date`)
- Class attribute: `booking_numbers` initialized with an empty list. This list will contain the booking numbers generated when reservations are created.
- A constructor that takes as input a string, a room, two dates (representing the check in and check out respectively), and an optional integer representing a booking number. If the last input is not provided, its default value should be `None`. An `AssertionError` should be raised if the input room is not available at the specified dates. Otherwise, the constructor uses the provided inputs to initialize all the instance attributes accordingly. If the booking number is not provided, then the constructor generates a *new* random 13 digit number for this reservation. This means that the number generated should not already be part of the list of booking numbers generated up to now. The constructor should also update the class attribute accordingly. Please note that booking numbers with 13 digits cannot have leading zeros (i.e., `0012345678912` is not a valid booking number). Finally, the constructor should make sure to reserve the specified room for all nights from the check in date (included) to the check out date (excluded).

  Note that if the booking number was provided, then the method raises an `AssertionError` if such number had already been used or if it is not a 13 digit number.

  UPDATED on Apr 17: The sentence above was rephrased.

  For example,

```
>>> random.seed(987)
>>> Reservation.booking_numbers = []
>>> r1 = Room("Queen", 105, 80.0)
>>> r1.set_up_room_availability(['May'], 2021)
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 10)
>>> my_reservation = Reservation('Mrs. Santos', r1, date1, date2)
>>> print(my_reservation.check_in)
2021-05-03
>>> print(my_reservation.check_out)
2021-05-10
>>> my_reservation.booking_number
1953400675629
>>> r1.availability[(2021, 5)][9]
False
```

- A `__str__` method that returns a string representation of a reservation containing the booking number, name on the reservation, room reserved and check-in and out dates. Each piece of information in the string should be on a new line.

  For example,

```
>>> random.seed(987)
>>> Reservation.booking_numbers = []
>>> r1 = Room("Queen", 105, 80.0)
>>> r1.set_up_room_availability(['May'], 2021)
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 10)
>>> my_reservation = Reservation('Mrs. Santos', r1, date1, date2)
```

```
>>> print(my_reservation)
Booking number: 1953400675629
Name: Mrs. Santos
Room reserved: Room 105,Queen,80.0
Check-in date: 2021-05-03
Check-out date: 2021-05-10
```

- An instance method called `to_short_string` which takes no input, and returns a string containing the booking number and name on the reservation separated by two hyphens.

  For example,

```
>>> random.seed(987)
>>> Reservation.booking_numbers = []
>>> r1 = Room("Queen", 105, 80.0)
>>> r1.set_up_room_availability(['May'], 2021)
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 10)
>>> my_reservation = Reservation('Mrs. Santos', r1, date1, date2)
>>> my_reservation.to_short_string()
'1953400675629--Mrs. Santos'
```

- A class method called `from_short_string` which takes as input a string of the same format as the one returned by the `to_short_string` method, two `date` objects representing the check-in and check-out dates, and a room object. The method creates and returns an object of type `Reservation` for a stay in the specified room.

  For example,

```
>>> Reservation.booking_numbers = []
>>> r1 = Room("Queen", 105, 80.0)
>>> r1.set_up_room_availability(['May'], 2021)
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 4)
>>> my_reservation = Reservation.from_short_string('1953400675629--Mrs. Santos', date1, date2, r1)
>>> print(my_reservation.check_in)
2021-05-03
>>> print(my_reservation.check_out)
2021-05-04
>>> my_reservation.booking_number
1953400675629
>>> r1.availability[(2021, 5)][3]
False
```

- A static method called `get_reservations_from_row` which takes as input a room object and a list of tuples. The tuples in the list will conform to the same format as those described in the `Hotel.load_reservation_strings_for_month` method below, that is, the first three elements of the tuple will be the year (integer), month (string) and day (integer), and the fourth element will be a string in the same format as that returned by the `to_short_string` method. The method should return a dictionary where each key is a booking number and each value is the reservation object for that booking number. Note that the check-in and check-out dates must be inferred by finding the minimum and maximum day/month over all tuples with the same booking number.

  A tuple in the list with an empty reservation string (fourth element) means there was no reservation on that day for the given room. Note that you can assume that the same reservation number has

not been used for disjoint dates.(UPDATED on Apr 17)

For example,

```
>>> random.seed(987)
>>> Reservation.booking_numbers = [] # needs to be reset for the test below to pass
>>> r1 = Room("Queen", 105, 80.0)
>>> r1.set_up_room_availability(MONTHS, 2021)
>>> rsv_strs = [(2021, 'May', 3, '1953400675629--Jack'), (2021, 'May', 4, '1953400675629--Jack')]
>>> rsv_dict = Reservation.get_reservations_from_row(r1, rsv_strs)
>>> print(rsv_dict[1953400675629])
Booking number: 1953400675629
Name: Jack
Room reserved: Room 105,Queen,80.0
Check-in date: 2021-05-03
Check-out date: 2021-05-05
```

UPDATED on Apr 17: Fixed a typo on the example above.

Hotel

Create a module called `hotel.py` and place your name and student ID at the top. All the functions in this section will go inside this module. You may not import any modules other than `doctest`, `datetime`, `random`, `copy`, `os`, `room` and `reservation`.

Please note that all examples below assume the following import statements has been added inside this module:

```
from room import Room, MONTHS, DAYS_PER_MONTH
from reservation import Reservation
```

(UPDATED on Apr 17)

Inside this module create a class called `Hotel`. This class should contain the following:

- Instance attributes: `name` (a string), `rooms` (a list of `Room` objects), `reservations` (a dictionary mapping integers, i.e. booking numbers, to `Reservation` objects).
- A constructor that takes as input a string, a list of `Room` objects, and a dictionary mapping integers to `Reservation` objects. The default values for the last two inputs should be an empty list and an empty dictionary respectively. The constructor uses the provided inputs to initialize all the instance attributes accordingly. The list of rooms and dictionary of reservations should be deep copied.
- An instance method called `make_reservation` which takes as input two strings representing the name of the person reserving and the type of room desired respectively, as well as two objects of type `date`, one representing the check in date, and the other representing the check out date. If a room of the specified type is available at the hotel for the dates indicated, then the method creates a reservation for the first available room of that type and returns the booking number of the reservation. If no room of the given type is available for the dates indicated, then the method raises an `AssertionErrror`. If needed, the method should also update the attribute storing all the hotel reservations accordingly.

  For example,

  ```
  >>> random.seed(987)
  >>> Reservation.booking_numbers = []
  >>> r1 = Room("Queen", 105, 80.0)
  >>> r1.set_up_room_availability(['May'], 2021)
  >>> h = Hotel("Secret Nugget Hotel", [r1])
  ```

```
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 10)
>>> h.make_reservation("Mrs. Santos", "Queen", date1, date2)
1953400675629
>>> print(h.reservations[1953400675629])
Booking number: 1953400675629
Name: Mrs. Santos
Room reserved: Room 105,Queen,80.0
Check-in date: 2021-05-03
Check-out date: 2021-05-10
```

UPDATED Apr 17: fixed the example above.

- An instance method called `get_receipt` which takes as input a list of integers representing booking numbers. The method returns a float indicating the amount of money a user should pay the hotel for those reservations. If the booking number refers to a reservation that was not made for this hotel, then the method simply ignores it. Apr 20: Do not round the float.

  For example,

```
>>> r1 = Room("Queen", 105, 80.0)
>>> r2 = Room("Twin", 101, 55.0)
>>> r3 = Room("Queen", 107, 80.0)
>>> r1.set_up_room_availability(['May', 'Jun'], 2021)
>>> r2.set_up_room_availability(['May', 'Jun'], 2021)
>>> r3.set_up_room_availability(['May', 'Jun'], 2021)
>>> h = Hotel("Secret Nugget Hotel", [r1, r2, r3])
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 10)
>>> num1 = h.make_reservation("Mrs. Santos", "Queen", date1, date2)
>>> h.get_receipt([num1])
560.0

>>> date3 = datetime.date(2021, 6, 5)
>>> num2 = h.make_reservation("Mrs. Santos", "Twin", date1, date3)
>>> h.get_receipt([num1, num2])
2375.0

>>> h.get_receipt([123])
0.0
```

- An instance method called `get_reservation_for_booking_number` which takes as input a booking number (integer), and returns the reservation object with the given number, or `None` if such a reservation cannot be found.

```
>>> random.seed(137)
>>> Reservation.booking_numbers = []
>>> r1 = Room("Queen", 105, 80.0)
>>> r1.set_up_room_availability(['May'], 2021)
>>> h = Hotel("Secret Nugget Hotel", [r1])
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 10)
>>> num1 = h.make_reservation("Mrs. Santos", "Queen", date1, date2)
>>> rsv = h.get_reservation_for_booking_number(num1)
>>> print(rsv)
```

```
Booking number: 4191471513010
Name: Mrs. Santos
Room reserved: Room 105,Queen,80.0
Check-in date: 2021-05-03
Check-out date: 2021-05-10
```

- An instance method called `cancel_reservation` which takes as input an integer representing a booking number. The method does not return any value, but cancels the reservation with the specified booking number. This means that such reservation should no longer appear in the reservations of the hotel, and the room originally reserved should be made available again. If the booking number does not refer to a reservation at the hotel, then the method does not do anything.

  For example,

```
>>> r1 = Room("Queen", 105, 80.0)
>>> r1.set_up_room_availability(['May'], 2021)
>>> h = Hotel("Secret Nugget Hotel", [r1])
>>> date1 = datetime.date(2021, 5, 3)
>>> date2 = datetime.date(2021, 5, 10)
>>> num1 = h.make_reservation("Mrs. Santos", "Queen", date1, date2)
>>> h.cancel_reservation(num1)
>>> num1 in h.reservations
False
>>> r1.availability[(2021, 5)][4]
True
```

- An instance method called `get_available_room_types` which takes no explicit inputs and returns a list of strings representing the room types available at the hotel.

  For example,

```
>>> r1 = Room("Queen", 105, 80.0)
>>> r2 = Room("Twin", 101, 55.0)
>>> r3 = Room("Queen", 107, 80.0)
>>> r1.set_up_room_availability(['May', 'Jun'], 2021)
>>> r2.set_up_room_availability(['May', 'Jun'], 2021)
>>> r3.set_up_room_availability(['May', 'Jun'], 2021)
>>> h = Hotel("Secret Nugget Hotel", [r1, r2, r3])
>>> types = h.get_available_room_types()
>>> types.sort()
>>> types
['Queen', 'Twin']
```

- A static method `load_hotel_info_file` which takes as input a path to a `hotel_info.txt` file for a given hotel. The method should read in the file at that path and return a 2-tuple of the hotel's name and a list of `Room` objects.

  For example,

```
>>> hotel_name, rooms = Hotel.load_hotel_info_file('hotels/overlook_hotel/hotel_info.txt')
>>> hotel_name
'Overlook Hotel'
>>> print(len(rooms))
500
>>> print(rooms[236])
```

```
Room 237,Twin,99.99
```

- An instance method called `save_hotel_info_file` which saves the hotel's name and room information into a file `hotel_info.txt` which should be located inside a folder named after the hotel (in all lowercase, with spaces replaced by underscores), and said folder should be in a folder called `hotels`. The first line of the file will contain the hotel's name, and each subsequent line will contain information for one of the hotel's rooms, in the format given by the Room `__str__` method.

  For example,

```
>>> r1 = Room("Double", 101, 99.99)
>>> r1.set_up_room_availability(['Oct', 'Nov', 'Dec'], 2021)
>>> h = Hotel("Queen Elizabeth Hotel", [r1], {})
>>> h.save_hotel_info_file()
>>> fobj = open('hotels/queen_elizabeth_hotel/hotel_info.txt', 'r')
>>> fobj.read()
'Queen Elizabeth Hotel\\nRoom 101,Double,99.99\\n'
>>> fobj.close()
```

  To test this method without getting errors, you can manually create an empty folder named 'queen_elizabeth_hotel' inside the folder named 'hotels'. We'll talk more about how to create directory using Python in one of the upcoming functions. (UPDATED on Apr 17)

- A static method called `load_reservation_strings_for_month` which takes a hotel folder name (string), a month (string, from the `MONTHS` list), and a year (integer). (UPDATED on Apr 17: removed one of the inputs) It should load the CSV file named after the given month and year (`YEAR_MONTH.csv` (UPDATED on Apr 17)) and located in the given folder name (which itself will be inside a folder called `hotels`). It should then return a dictionary, where each key is a room number (integer), and each corresponding value is a list of tuples corresponding to the reservation data for that room, one tuple for each day of the month. Each tuple should contain four elements: the year, month, day and the short reservation string for that day which corresponds to the text that can be found in the appropriate row and column of the csv file the method is loading. Note that you can assume that if a room x is reserved for day y, then the column y of the row corresponding to "Room x" will contain the reservation info following the same format used by the Reservation `to_short_string` method. If, on the other hand, the room x has not been reserved for day y, then an empty string will be found (UPDATED on Apr 17).

  For example,

```
>>> name, rooms = Hotel.load_hotel_info_file('hotels/overlook_hotel/hotel_info.txt')
>>> h = Hotel(name, rooms, {})
>>> rsvs = h.load_reservation_strings_for_month('overlook_hotel', 'Oct', 1975)
>>> print(rsvs[237])
[(1975, 'Oct', 1, ''), (1975, 'Oct', 2, ''), (1975, 'Oct', 3, ''), (1975, 'Oct', 4, ''), \
(1975, 'Oct', 5, ''), (1975, 'Oct', 6, ''), (1975, 'Oct', 7, ''), (1975, 'Oct', 8, ''), \
(1975, 'Oct', 9, ''), (1975, 'Oct', 10, ''), (1975, 'Oct', 11, ''), (1975, 'Oct', 12, ''), \
(1975, 'Oct', 13, ''), (1975, 'Oct', 14, ''), (1975, 'Oct', 15, ''), (1975, 'Oct', 16, ''), \
(1975, 'Oct', 17, ''), (1975, 'Oct', 18, ''), (1975, 'Oct', 19, ''), (1975, 'Oct', 20, ''), \
(1975, 'Oct', 21, ''), (1975, 'Oct', 22, ''), (1975, 'Oct', 23, ''), (1975, 'Oct', 24, ''), \
(1975, 'Oct', 25, ''), (1975, 'Oct', 26, ''), (1975, 'Oct', 27, ''), (1975, 'Oct', 28, ''), \
(1975, 'Oct', 29, ''), (1975, 'Oct', 30, '9998701091820--Jack'), \
(1975, 'Oct', 31, '9998701091820--Jack')]
```

  UPDATED on Apr 17: Fixed the example above.

- An instance method called `save_reservations_for_month` which takes as input a string corresponding to a month (from the `MONTHS` list) and a year (integer). The function should create a new CSV file named after the given month and inside a folder named after the hotel (all in lowercase, with spaces replaced by underscores), with that folder being in a folder called `hotels`. The CSV file should contain one row per room. The first column of each row will be the room number. There will be as many subsequent columns for days in the given month. In every row, after the room number, will be the reservation string as given by the Reservation `to_short_string` method, if a reservation occurs in that room on the given day. Or, if no reservation occurs in that room on that day, then the column should contain an empty string.

  For example,

```
>>> random.seed(987)
>>> r1 = Room("Double", 237, 99.99)
>>> r1.set_up_room_availability(['Oct', 'Nov', 'Dec'], 2021)
>>> Reservation.booking_numbers = []
>>> h = Hotel("Queen Elizabeth Hotel", [r1], {})
>>> date1 = datetime.date(2021, 10, 30)
>>> date2 = datetime.date(2021, 12, 23)
>>> num = h.make_reservation("Jack", "Double", date1, date2)
>>> h.save_reservations_for_month('Oct', 2021)
>>> fobj = open('hotels/queen_elizabeth_hotel/2021_Oct.csv', 'r')
>>> fobj.read()
'237,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,1953400675629--Jack,1953400675629--Jack\\n'
>>> fobj.close()
```

- An instance method called `save_hotel` which saves a file `hotel_info.txt` with the hotel's name and room information, and CSV files (one for each month in which there are rooms available) containing the reservation data, both as described above. If the folders in which the files should be saved do not exist, then the function should create them. Note that you can check if a folder exists using the `os.path.exists(path_str)` function, and you can create a folder using the `os.makedirs(path_of_folder_to_make)` function.

  Note that you can assume that if a room from the hotel is available in a certain month, then so are all the others. More over, if there are no rooms in a hotel, then no CSV files should be created. (UPDATED on Apr 17)

  For example,

```
>>> random.seed(987)
>>> Reservation.booking_numbers = []
>>> r1 = Room("Double", 237, 99.99)
>>> r1.set_up_room_availability(['Oct', 'Nov', 'Dec'], 2021)
>>> h = Hotel("Queen Elizabeth Hotel", [r1], {})
>>> date1 = datetime.date(2021, 10, 30)
>>> date2 = datetime.date(2021, 12, 23)
>>> h.make_reservation("Jack", "Double", date1, date2)
1953400675629
>>> h.save_hotel()

>>> fobj = open('hotels/queen_elizabeth_hotel/hotel_info.txt', 'r')
>>> fobj.read()
'Queen Elizabeth Hotel\\nRoom 237,Double,99.99\\n'
>>> fobj.close()

>>> fobj = open('hotels/queen_elizabeth_hotel/2021_Oct.csv', 'r')
```

```
>>> fobj.read()
'237,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,1953400675629--Jack,1953400675629--Jack\\n'
>>> fobj.close()
```

- A class method called `load_hotel` which takes a folder name as input, and loads the hotel info file and reservation CSV files (as described above) from inside that folder (itself being in a folder called `hotels`), then creates and returns an object of type `Hotel` with the loaded name, rooms and reservation information. Note that you will have to create `Reservation` objects in order to create the `Hotel` object.

  Also, each time you load a CSV file for a particular month and year, you should set up the availability for all the rooms listed in that CSV file for that month and year. Make sure to do this before attempting to create the reservations! (UPDATED on Apr 17)

  To get access to all the files/directories in a given folder you can use `os.listdir(path)`, where `path` is a string containing the path to the folder you are interested in. For example, `os.listdir('hotels/overlook_hotel')` will return a list of strings, each of which is the file name (extension included) of a file stored in the folder called `overlook_hotel` which is itself in a folder called `hotels`, which happens to be in the same directory as your Python code. If other folders happened to be inside `overlook_hotel`, then the name of those folders would have been part of such list.

  For example,

  ```
  >>> random.seed(137)
  >>> Reservation.booking_numbers = []
  >>> hotel = Hotel.load_hotel('overlook_hotel')
  >>> hotel.name
  'Overlook Hotel'
  >>> str(hotel.rooms[236])
  'Room 237,Twin,99.99'
  >>> print(hotel.reservations[9998701091820])
  Booking number: 9998701091820
  Name: Jack
  Room reserved: Room 237,Twin,99.99
  Check-in date: 1975-10-30
  Check-out date: 1975-12-24
  ```

  UPDATED on Apr 17: fixed a typo in the example.

Booking

Create a module called `booking.py` and place your name and student ID at the top. All the functions in this section will go inside this module. You may not import any modules other than `doctest`, `datetime`, `random`, `matplotlib`, `os` and `hotel`.

Inside this module create a class called `Booking`. This class should contain the following:

- A constructor that takes as input a list of hotels and initializes an instance attribute of the same name (`hotels`) accordingly.
- A class method called `load_system` that loads in all the hotels in the `hotels` folder and creates and returns an object of type `Booking` with said list of hotels.

  For example, if using the hotels folder given with the assignment, without any modifications made to it,

  ```
  >>> system = Booking.load_system()
  >>> len(system.hotels)
  ```

```
2
>>> system.hotels[0].name
'The Great Northern Hotel'
>>> print(system.hotels[0].rooms[314])
Room 315,Queen,129.99
```

Apr 22: The ordering of the `hotels` list does not matter for the above example or the examples below. (You may have to use index `0` or index `1` in the above example to get the output.) However, the ordering should be consistent throughout your methods. That is, if your hotels list has Hotel A followed by Hotel B, then further methods should also have output or return values consistent with the same order of A followed by B.

- An instance method called `menu` which takes no explicit inputs and does not return anything. The method welcomes the user to the booking system, and asks them to select which operation they would like to perform: create a new reservation, cancel a reservation, or look up a reservation. After getting the input from the user, the method should call the appropriate method (of the ones listed below), then save all the hotels back to disk.

  You can assume the user will enter a number between 1 and 3 corresponding to the three operations above. However, they may also enter the magic word, `xyzzy`. In that case, the system should go into the secret deletion mode (see the method below).

  For example,

```
>>> booking = Booking.load_system()
>>> booking.menu()
Welcome to Booking System
What would you like to do?
1        Make a reservation
2        Cancel a reservation
3        Look up a reservation
> 1
(See below for the output of the create_reservation method.)
```

- An instance method called `create_reservation` that takes no inputs and does not return a value. It should prompt the user for their name, then display a list of hotels for them to choose from, then a list of room types at that hotel. It should also prompt them for a check-in and check-out date. It should then make the reservation with the given information, and print out their booking number and total amount owing Apr 20: rounded to two decimal points..

  For example, if using the hotels folder given with the assignment, without any modifications made to it,

```
>>> random.seed(137)
>>> booking = Booking.load_system()
>>> booking.create_reservation()
Please enter your name: Judy
Hi Judy! Which hotel would you like to book?
1        The Great Northern Hotel
2        Overlook Hotel
> 1
Which type of room would you like?
1        Double
2        Twin
3        King
4        Queen
```

```
> 1
Enter check-in date (YYYY-MM-DD): 1989-01-01
Enter check-out date (YYYY-MM-DD): 1989-01-04
Ok. Making your reservation for a Double room.
Your reservation number is: 4191471513010
Your total amount due is: $459.84.
Thank you!
```

Apr 22: Added a random seed to the above example.

- An instance method called `cancel_reservation` that takes no inputs and does not return a value. It prompts the user to enter a booking number, and tries to cancel the reservation with that booking number (at any hotel). If the reservation could not be found, inform the user.

  For example, if using the hotels folder given with the assignment, without any modifications made to it,

```
>>> booking = Booking.load_system()
>>> booking.cancel_reservation()
Please enter your booking number: 9998701091820
Cancelled successfully.
>>> booking.cancel_reservation()
Please enter your booking number: 9998701091820
Could not find a reservation with that booking number.
```

- An instance method called `lookup_reservation` that takes no inputs and does not return a value. It asks the user if they have their booking number(s). If they do, then it asks them to enter them, one at a time, and then finds the associated reservation(s) (at any hotel) and prints the reservation to the screen (Apr 20: rounding the amount owing for each reservation to two decimal places). If any of the numbers are invalid, then it should inform the user as such.

  If they do not have their booking number(s), then it asks them to enter their name, hotel name, room number, and check-in and check-out dates, and tries to find a reservation matching the given information. If it finds such a reservation, then it prints it to the screen (Apr 20: rounding the amount owing for each reservation to two decimal places). Otherwise, it informs the user as such.

  For example, if using the hotels folder given with the assignment, without any modifications made to it,

```
>>> booking = Booking.load_system()
>>> booking.lookup_reservation()
Do you have your booking number(s)? yes
Please enter a booking number (or 'end'): 9998701091820
Please enter a booking number (or 'end'): end
Reservation found at hotel Overlook Hotel:
Booking number: 9998701091820
Name: Jack
Room reserved: Room 237,Twin,99.99
Check-in date: 1975-10-30
Check-out date: 1975-12-24
Total amount due: $5499.45

>>> booking.lookup_reservation()
Do you have your booking number(s)? no
Please enter your name: Judy
Please enter the hotel you are booked at: The Great Northern Hotel
```

```
Enter the reserved room number: 315
Enter the check-in date (YYYY-MM-DD): 1989-01-01
Enter the check-out date (YYYY-MM-DD): 1990-01-01
Reservation found under booking number 3020747285952.
Here are the details:
Booking number: 3020747285952
Name: Judy
Room reserved: Room 315,Queen,129.99
Check-in date: 1989-01-01
Check-out date: 1990-01-01
Total amount due: $47446.35
```

Apr 20: Fixed check-out date and amount owing in above examples.

- An instance method called `delete_reservations_at_random` that takes no inputs and does not return a value. The method should print `You said the magic word!`, then choose a hotel at random and delete all of its reservations. Choose the hotel by calling `random.randint` to find a random index into the `hotels` instance attribute list.

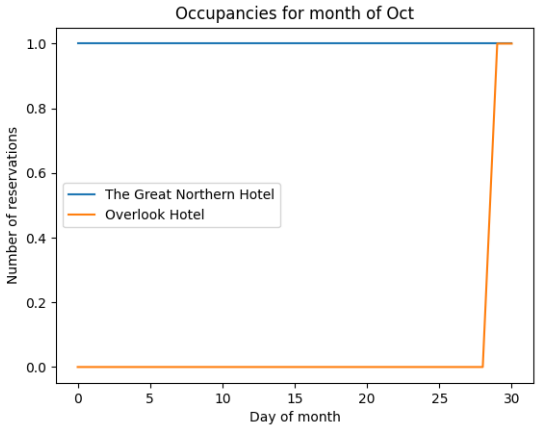
  For example, if using the hotels folder given with the assignment, without any modifications made to it,

  ```
  >>> random.seed(1338)
  >>> booking = Booking.load_system()
  >>> booking.delete_reservations_at_random()
  You said the magic word!
  >>> len(booking.hotels[1].reservations)
  0
  >>> len(booking.hotels[0].reservations)
  1
  ```

- **(Bonus +5 pts)** An instance method called `plot_occupancies` which takes one string as input, corresponding to a month from the `MONTHS` list. The function should create a line plot, with one line per hotel, showing the number of reservations each day of the month (over all years). That is, the x-axis should be the days of the month, and the y-axis should be the number of reservations at the hotel. Make sure to give titles for your axes and plot, and also display a legend containing a label with the hotel's name for each line. When finished plotting, save the figure to a file called `hotel_occupancies_MONTH.png` where `MONTH` is given by the argument to the method. It should then return a two-dimensional list of 2-tuples of the list of x-values and y-values for each hotel.

  For example, if using the hotels folder given with the assignment, without any modifications made to it, then running the following code should produce the figure below.

  ```
  >>> booking = Booking.load_system()
  >>> booking.plot_occupancies('Oct')
  [([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
  25, 26, 27, 28, 29, 30], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1]), ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
  18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1])]
  ```


Occupancies for month of Oct
Number of reservations
Day of month
The Great Northern Hotel
Overlook Hotel
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

What To Submit

You must submit all your files on codePost (https://codepost.io/). The file you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`room.py`
`reservation.py`
`hotel.py`
`booking.py`
`README.txt` In this file, you can tell the TA about any issues you ran into doing this project. Remember that this project like all other assignments is an `individual` assessment and must represent the entirety of your own work. You are permitted to verbally discuss it with your peers, as long as no written notes are taken. If you do discuss it with anyone, please make note of those people in this `README.txt` file. If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.