Object-Oriented Programming
50:198:113 (Spring 2022)

Homework:	**4**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**3/26/22**	**E-mail:**	`suneeta.ramaswami@rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 4

The assignment is due by 11:59PM of the due date. The point value is indicated in square braces next to each problem. Each solution must be the student's own work. Assistance should be sought or accepted from the course instructor only. Any violation of this rule will be dealt with harshly.

The first problem of this assignment is a continuation of the `Date` class implementation from Homework 3, wherein you are asked to include the implementation of additional methods for the class, including overloaded arithmetic and comparison operators. The second problem asks you implement some functions that manipulate `Date` objects. The third problem requires you to use a `Bag` class implementation that is given to you to implement functions that us `Bag` objects as parameters and/or return `Bag` objects.

As usual, you are graded not only on the correctness of the code, but also on clarity and readability. I will deduct points for not following the guidelines for your class design, poor indentation, poor choice of object names, and lack of documentation. You are expected to provide docstring documentation for modules, classes and their methods, as well as functions.

Problem 1 [26 points] You are now asked to include new methods for the `Date` class from Homework 3. In particular, you will implement two new methods called `nextday` and `prevday`, as well as several overloaded operators. Please note that you are required to make all instance attributes private in this implementation.

If your implementation of the `Date` class in Homework 3 was completely correct, you may use your own implementation (but make sure you change your instance attributes to make them private). However, if it was not fully correct, please use my implementation provided under "Assignment #3 Solutions" on Canvas. This will allow me to grade your implementation of the newly inserted methods for this homework assignment, without penalizing you for carry-over errors from the previous assignment. You will also rename the module containing the `Date` class as `date.py`.

1. `nextday`: Returns the date of the following day. For example, if `d` is the date 3/31/1801, `d.nextday()` should return the date 4/1/1801. **Important:** This function returns a `Date` object.
2. `prevday`: Returns the date of the previous day. Note that January 1, 1800 does not have a previous day. This error should be caught by raising an `Exception`. **Important:** This function returns a `Date` object.
3. `__add__`: This method overloads the `+` operator. It has two parameters: `self` and an integer `n`. It returns the date that occurs `n` days `after` the date `self`. For example, if `d` is the date 4/25/2015, then after the expression `newd = d + 9`, `newd` is the date 5/4/2015. You will find the method `nextday` useful here.

4. `__sub__`: This method overloads the `-` operator. It has two parameters: `self` and an integer `n`. It returns the date that occurs `n` days *before* the date `self`. For example, if `d` is the date 4/25/2015, then after the expression `newd = d - 25`, `newd` is the date 3/31/2015. You will find the method `prevday` useful here.
5. `__lt__`: This method overloads the `<` operator. It has two parameters: `self` and `other`, which is a date. It returns True if `self` comes *before* `other`.
6. `__eq__`, `__le__`, `__gt__`, `__ge__`, and `__ne__`: Overload all remaining relational operators as well, using the obvious interpretation of these operators for dates.

You may once again test your implementation by using the test module `testdate.py` provided on Canvas.

Problem 2 [24 points] Functions that manipulate `Date` objects. In this problem, you are asked to implement three functions that manipulate `Date` objects in a module called `datefuns.py`. You must use Date class methods (as implemented in Problem 1 above) to implement all of the following functions. At the top of your `datefuns.py` module, import the `Date` class using `from date import Date`. **Important:** You may not manipulate instance attributes directly to implement these functions (which you cannot do anyway if you made all the instance attributes private). You must use `Date` methods.

1. Implement a function called `weekend_dates` with two parameters, a month `m` ($1 \leq$ `m` ≤ 12) and a year `y`. The function should *print* all the weekend (Saturday and Sunday) dates that occur in month `m` of year `y`. For example, `weekend_dates(4, 2016)` should print

```
April 2, 2016 (Saturday)
April 3, 2016 (Sunday)
April 9, 2016 (Saturday)
April 10, 2016 (Sunday)
April 16, 2016 (Saturday)
April 17, 2016 (Sunday)
April 23, 2016 (Saturday)
April 24, 2016 (Sunday)
April 30, 2016 (Saturday)
```

2. Implement a function called `first_mondays` with a single parameter, namely a year `y`. The function should print the dates of the first Monday of every month in that year. For example, `first_mondays(2016)` should print

```
First Mondays of 2016:

January 4, 2016
February 1, 2016
March 7, 2016
April 4, 2016
May 2, 2016
June 6, 2016
July 4, 2016
August 1, 2016
```

```
September 5, 2016
October 3, 2016
November 7, 2016
December 5, 2016
```

3. There are many situations in which one needs a schedule of dates occuring at regular intervals between a start date and an end date. For example, a course instructor may want to give an online class quiz every 12 days starting on September 6, 2016 and ending on or before October 31, 2016. Implement a function called `interval_schedule` with three parameters: `start_date` (a `Date` object), `end_date` (a `Date` object), and `interval` (a positive integer). The function should **return** the list of dates that occur every `interval` days, starting on `start_date` and ending on or before `end_date`. *Note that the function is returning a list of* `Date` *objects.* For example, `interval_schedule(Date(9, 6, 2016), Date(10, 31, 2016), 12)` should return a list of `Date` objects. If you print the elements of this list, you should see:

```
September 6, 2016
September 18, 2016
September 30, 2016
October 12, 2016
October 24, 2016
```

Problem 3 [30 points] Bags. You are given a class called `Bag` in a module called `bag.py`, which is available on the Canvas page for this homework assignment. Download this module before you start working on this problem. A `Bag` object is simply a container that stores an unordered collection of items. An item may occur several times in a bag. Read the implementation and the documentation carefully to understand all the methods for this class.

In this problem, you are asked to write some functions that have `Bag` objects as parameters or return values. The point of this exercise is to *use* a class that is given to you. Insert these functions in the `bagfunctions.py` file. **Make sure you import the `bag` module when implementing `bagfunctions.py`.** In each of the following functions, you are **required** to use the `Bag` class methods to implement the function. **You may not** manipulate `Bag` instance attributes directly (in any case, you will not be able to, as the attribute is private).

1. Implement a function called `remove_item` with two parameters: a `Bag` object `B` and an item `item`. The function removes *all* occurrences of `item` from `B`. The actual `Bag` parameter will be modified by this function.
2. Implement a function called `remove_repeats` with a single parameter, which is a `Bag` object `B`. The function removes all repeating items from `B` and retains exactly one copy of each item. The actual parameter will be modified by this function.
3. Implement a function called `mode` with a single parameter, which is a `Bag` object `B`. The function returns a list containing the most frequently occurring item(s) in the bag. The actual bag parameter should not be modified by this function. *Note:* The order of items in the returned list is not relevant.
4. Implement a function called `union` with two parameters, a `Bag` object `B1` and another `Bag` object `B2`. The function **returns** a `Bag` object containing the union of `B1` and `B2`. The *union* of two bags `B1` and `B2` is a bag containing all the items from `B1` and from `B2`. Note that since a bag may have repeating items, the count of an item in the union is

equal to the sum of the counts of that item in each of `B1` and `B2`. The actual parameters should not be modified by this function.

5. Implement a function called `intersection` with two parameters, a `Bag` object `B1` and another `Bag` object `B2`. The function **returns** a `Bag` object containing the intersection of `B1` and `B2`. The *intersection* of two bags `B1` and `B2` is bag containing items that are common to `B1` and `B2`. Note that the count of an item in the intersection is equal to the minimum of the counts of that item in each of `B1` and `B2`. The actual parameters should not be modified by this function.

For example, let `bone` be a `Bag` object containing the items `1, 1, 'hello', 'hello', 2, 2, 'there', 3` and let `btwo` be a `Bag` object containing the items `2, 2, 2, 'hello', 'there', 'there', 3`. Then,

- After the function call `remove_item(bone, 'hello')`, `bone` will contain the items `1, 1, 2, 2, 'there', 3`.
- Assuming the original contents of `bone`, after the function call `remove_repeats(bone)`, `bone` will contain the items `1, 'hello', 2, 'there', 3`.
- Assuming the original contents of `bone`, the function call `mode(bone)` **returns** the list `[1, 2, 'hello']` and `mode(btwo)` returns the list `[2]`.
- Assuming the original contents of `bone`, the function call `union(bone, btwo)` **returns** a `Bag` whose contents are `1, 1, 'hello', 'hello', 'hello', 'there', 'there', 'there', 2, 2, 2, 2, 2, 3, 3`, and
- the function call `intersection(bone, btwo)` **returns** a `Bag` whose contents are `'hello', 2, 2, 'there', 3`.

Submission Guidelines

Implement the first problem in a module called `date.py`, the second problem in a module called `datefuns.py`, and the third one in a module called `bagfunctions.py`. *Your name and RUID should appear as a comment at the very top of each module.* Points will be deducted if you do not follow the specified naming convention.

Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Canvas as follows:

1. Go to the “Assignments” tab of the Canvas site for this course.
2. Click on “Programming Assignment #4” under Homework Assignments.
3. Upload your homework files (`date.py`, `datefuns.py` and `bagfunctions.py`) when you are ready to submit.

You must submit your assignment at or before 11:59PM on March 26, 2022.