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

Homework:	**3**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**3/7/22**	**E-mail:**	`suneeta.ramaswami@rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 3

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 only be sought or accepted from the course instructor only. Any violation of this rule will be dealt with harshly.

This assignment contains one problem on recursion and one problem on a `Date` class implementation. (*Note:* You will continue to build on the `Date` class implementation in Homework 4.) 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 recursive function implementations (Problem 1), the class design (Problem 2), poor indentation, poor choice of object names, and lack of documentation. For documentation, use docstring documentation for the entire module, as well as each function, class, and method.

Download the homework document (`hw3.pdf`) and the test file for Problem 2 (`testdate.py`) prior to starting your work.

Important note: When writing each of the following programs, it is important that you name all functions, classes, and methods exactly as described because I will assume you are doing so when testing your programs. If your program produces errors because the functions do not satisfy the stated prototype, points will be deducted.

Problem 1 [30 points] Recursive Functions. In this problem, you are asked to write three recursive functions. Implement all functions in a module called `problem1.py`.

1. *(10 points)* Write a recursive function called `replace_char` with two parameters: a string `astr`, a character `old_char`, and another character `new_char`. The function should return a string in which every occurrence of `old_char` in `astr` is replaced with `new_char`. For example, `replace_char("how now brown cow", 'w', 'o')` should return the string `"hoo noo broon coo"`. Once again, your implementation *should not* contain any loops, and may use only the index and slice operators for strings. No other built-in functions may be used.
2. *(10 points)* Write a recursive function called `occurrences` with two parameters: a string `astr` and another (nonempty) string `substr`. The function returns the number of times the substring `substr` appears in the string `astr`. For example, `occurrences("how now brown cow", "ow")` should return 4, `occurrences("house mouse louse", "ow")` should return 0, and `occurrences("green eggs and ham", "egg")` should return 1. Your implementation should not contain any loops and may use only the index `[]` and slice operators `[:]` for strings. No other built-in functions may be used. Please note that `in` is a built-in function, and you may not use it to implement your function.

3. *(10 points)* Write a recursive function called `inverse_pair` with a single parameter `L`, which is a list of integers. The function returns `True` if `L` contains a pair of integers whose sum is zero and `False` otherwise. The base case occurs when the list has exactly two integers (since it doesn't make sense to talk about a "pair" of integers for lists with fewer than two elements). For example, `inverse_pair` should return `False` for the list `[12, 8, 10, -5]` and `True` for the list `[12, 5, 10, -5, -9]`. Your function should consist **only** of the base case and the recursive calls. As above, all usual restrictions apply for the recursive implementation.

Problem 2 [30 points] Calendar dates. In this problem, you are asked to implement a class called `Date` for calendar dates occurring on or after January 1, 1800. FYI, January 1, 1800 was a Wednesday. Instances of the `Date` class have a month, day, and year value representing valid calendar dates on or after January 1, 1800. Create a module called `problem2.py` to contain your `Date` class implementation. Details about the class and its methods are provided below. (Python has its own `datetime` module, but, needless to say, you **should not** use that when implementing this class.)

1. Assign a class attribute called `min_year` the value 1800. This represents the smallest year value allowed for instances of the `Date` class. The benefit of using a name to refer to the minimum year is that if we change our mind about the smallest allowable year value, we need to change it in only one place. Use `min_year` everywhere to refer to this value, rather than hard-coding 1800 in your code. In addition, assign another class attribute called `dow_jan1` the value `''Wednesday''`. This represents the day of week on January 1 of the year `min_year`.
2. `__init__`: The constructor sets the values of the month, day, and year attributes of the date. The default values for these should be 1, 1, and `min_year`, respectively. **The constructor must check for the validity of the date**, and if the date is invalid, should raise an exception. This means that the month should lie between 1 and 12 (inclusive), the day should be valid *for that month*, and the year should be greater than or equal to `min_year`. For example, the dates 2/30/2008, 2/29/2009, and 9/31/2010 are all invalid. Make sure that you take leap years into account when determining the validity of the date. The method `year_is_leap` will come in handy here.
3. `month`: Returns the month of the date.
4. `day`: Returns the day of the date.
5. `year`: Returns the year of the date.
6. `year_is_leap`: Returns `True` if the year of the date is a leap year and `False` otherwise. A year is said to be a leap year if it is a multiple of 4. However, if it is also a multiple of 100, then it is a leap year *only* if it is a multiple of 400. So, for example, 1948 was a leap year, as will be 2124. However, 1800 and 1900 were not leap years, but 2000 was.
7. `daycount`: Returns the total number of days from January 1, 1800 to the date. For example, if `d` is the date 2/14/1801, then `d.daycount()` should return 410 (there are 410 days from January 1, 1800 to February 14, 1801). Make sure that you take leap years into account.
8. `day_of_week`: Returns the day of week of the date. Hence, the return value is one of `"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", or "Sunday"`. Please make sure that the day of week returned by your method is exactly one of the

seven strings shown here; this will make it easier for me to test your method. Important fact: January 1, 1800 was a Wednesday. You will find the `daycount` method useful here.

9. `__str__`: This method returns a printable (i.e., string) representation of the date. For example, if `d` is the date 4/25/2015, then `str(d)` should return the string `"April 25, 2015"`.
10. `__repr__`: This method also returns a string representation of the date. You may choose the representation.

You are provided a test file called `testdate.py` to test your implementation of the `Date` class. Once you have completed the implementation, test it by running the `testdate.py` module at the command line.

Submission Guidelines

Implement the first problem in `problem1.py` and the second problem in `problem2.py`. *Your name and RUID should appear as a comment at the very top of each file.* 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 #3" under Homework Assignments.
3. Upload your homework files (`problem1.py` and `problem2.py`) when you are ready to submit.

You must submit your assignment at or before 11:59PM on March 7, 2022.