CSC 242 Section 504 Winter 2017
Homework Assignment 3

Due date as specified on D2L

This assignment is worth 4% of your overall grade, and therefore will be graded on a scale of 0-4. Please upload a file containing your competed code onto D2L by the due date. While you may discuss the assignment with others, **you must write your code by yourself,** without assistance from other students or anyone else. Please see the course syllabus for details, and for my late homework submission policy.

Your assignment is to write a class called **`Date`**. A template for this class can be found in the accompanying `hw3.py` file. A `Date` object is intended to represent a particular date's month, day and year. You should represent each as an `int`. The `Date` class should have the 5 methods specified below. You can see the point value of each method. No partial credit will be given for incorrect implementations of the constructor, **__str__**, or **__repr__** methods, but partial credit may be given for the **next_date** and **earlier_date** methods if your method does not have any syntax errors and works on at least some examples.

1. (.5 points) A **constructor**. You should assume that your constructor will be passed 3 parameters, representing the month, day, and year to be stored in the object. Each is an integer. For example:

```
>>> mlk_day = Date(1, 16, 2017)
>>> hw3_due = Date(1, 29, 2017)
>>> revolution = Date(7, 4, 1776)
>>> feb_29 = Date(2, 29, 2016)
>>> ny_eve = Date(12, 31, 2016)
```

   You may assume that the constructor is passed numbers which specify a valid date; that is, constructor calls such as the following will not be made:

```
>>> june_31 = Date(6, 31, 2017)           # only 30 days in June
>>> this_feb_29 = Date(2, 29, 2017)       # 2017 is not a leap year.
```

   Remember that when *defining* a constructor, we will always call its first parameter `self`, which refers to the object which is created. However, when *calling* a constructor, `self` is not explicitly passed as a parameter. Thus, in the constructor's definition, it will appear that there is one additional parameter than in a call to the constructor. Any instance variables in the constructor (and other methods in the class) start with **`self.`** followed by the variable name.

2. (.5 points) A **`__str__` method**. This method is passed no parameters when called, and returns a string containing the month, day, and year, with '/' between each. Continuing with the examples above:

```
>>> str(mlk_day)
'1/16/2017'
>>> hw3_due.__str__()
'1/29/2017'
>>> print(str(revolution))
7/4/1776
>>> print(feb_29)
2/29/2016
```

Note that in the first two examples, calling the `str` constructor returns the same string as calling its parameter's `__str__` method. The third and fourth examples illustrate that the `print` function implicitly calls the `str` constructor, if it is not explicitly called. In either case, the string is printed without the ' '.

3. (1 point) An appropriate **__repr__** method. Please see my lecture notes or Chapter 8 of the text to see what **__repr__** must do.

4. (1 point) A **`next_date`** method. It should create and return a new `Date` object which represents the next day. You should write this method to be *non-destructive*, which means that the method should create a new Date object rather than modifying `self`. Again, continuing with the examples above:

```
>>> tuesday = mlk_day.next_date()
>>> print(tuesday)
1/17/2017
>>> str(hw3_due.next_date())
'1/30/2017'
>>> after_revolution = revolution.next_date()
>>> print(after_revolution)
7/5/1776
>>> march_1 = feb_29.next_date()
>>> print(march_1)
3/1/2016
>>> march_1 = Date(2,28,2017).next_date();
>>> print(march_1)
3/1/2017
>>> y2k = Date(12,31,1999).next_date();
>>> print(y2k)
1/1/2000
```

Keep in mind the rules for leap years: There is a leap year every year whose number is perfectly divisible by four - except for years which are both divisible by 100 and not divisible by 400. For example, 2017 is not a leap year, 2016 was a leap year, 2000 was a leap year, but 2100 will not be a leap year.

5. (1 point) A method called **earlier_date**. The method should return True or False, depending on whether or not one date is earlier than another. Keep in mind that a method is called using the "dot" syntax. Therefore, assuming that d1 and d2 are Date objects, a valid method call to **earlier_date** would be

>>> d1.earlier_date(d2)

and the method should return True if d1 is earlier than d2, or False otherwise. Here are some examples:

```
>>> today = Date(1,17,2017)
>>> y2k.earlier_date(today)
True
>>> today.earlier_date(mlk_day)
False
>>> feb_29.earlier_date(march_1)
True
```