Computer Science 1 — CSci 1100
Lab 10 — Date class

Lab Overview

This lab investigates Python classes and their use. The particular focus is a `Date` class. It builds heavily on the material covered in Lecture 18. Please continually refer to the `Point2d` class from Lecture 18 and the associated exercises when working on this lab. We have provided you with the starting material in `Date.py`, which you can download as part of the Lab 10 zip file from the Course Materials page on Submitty.

Start by looking at `Date.py`:

- You will notice two "global" variables at the start: one is a list of the number of days in each month, and the second is a list giving the names of each month. These are used here as constants that should not be changed. We will get to them in Checkpoint 2.
- The `Date` class does not have anything in it and is therefore non-functioning. Your job will be to replace the `pass` with methods and attributes.
- Some testing code is in the main code area. This will not work until you add methods.

Checkpoint 1

The dates will be stored with three attributes: the year, the month, and the day of the month, all as integers. These attributes will be created / assigned in the initializer methods. Therefore, please implement the following methods for the `Date` class

- `__init__`: This should take a year, a month and a day with default values of 1900, 1, 1.

```
>>> d1 = Date(1972, 3, 27)
>>> d2 = Date(1998)         # Will be January 1, 1998
```

- `__str__`: Format the data as a string with year/month/day. For example,

```
>>> d1 = Date(1972, 3, 27)
>>> s = str(d1)
>>> s
'1972/03/27'
>>> s1 = Date(1983, 11, 2)
>>> print(s1)
1983/11/02
```

  The example in `Point2d.py` is especially important here. As a hint for how to insert the `'0'` before the `'3'` in `'03'`, you may make use of the `rjust` method of strings. For example,

```
>>> s = '5'
>>> s1 = s.rjust(2,'0')
>>> print(s1)
05
>>> s = '21'
>>> s1 = s.rjust(2,'0')
>>> print(s1)
21
```

In the call to `rjust`, the 2 is the number of spaces, and the `'0'` is the character to inserted when there aren't enough characters in the string to be printed. You could also use the string `format` method to generate the string using the {:02d} format specifier.

- `same_day_in_year`: This should determine if two dates are on the same day within a year, even if they are not within the same year. For example, given

```
>>> d1 = Date(1972, 3, 27)
>>> d2 = Date(1998, 4, 13)
>>> d3 = Date(1996, 4, 13)
>>> d1.same_day_in_year(d2)
False
>>> d2.same_day_in_year(d3)
True
```

There is already code in the main area of `Date.py` to partially test these. The test code does not include code for testing the default assignments so you should add that. It also does not test all of the cases, so you will need to add some of your own.

To complete Checkpoint 1: show your code and the result of running it to a lab TA or a mentor.

Checkpoint 2 — Date Class Methods

Continuing with the `Date` class, please implement and test the following two methods, each of which involves significantly more logic that the first three methods.

- `is_leap_year`: This should return true if the year is a leap year. Leap years occur when the year is divisible by 4. The exception to this rule is years that are divisible by 100 but not 400. Got it? In other words, 2000, 2004, 2008 and 2012 were all leap years, but 1900, 2002, and 2011 were not. As examples, using the above values of `d1` and `d2`

```
>>> d1.is_leap_year()
True
>>> d2.is_leap_year()
False
```

- `__lt__`: This is the "less than" operator and it should return true if the first Date is earlier than the second.

```
>>> d1 = Date(1972, 3, 27)
>>> d2 = Date(1998, 4, 13)
>>> d3 = Date(1998, 5, 13)
>>> d4 = Date(1998, 4, 11)
>>> d1 < d2
True
>>> d2 < d3
True
>>> d3 < d4
False
```

Observe that there is a great deal more functionality that we could (and should) add to the `Date` class to make it fully functional, but these methods are enough for now.

To complete Checkpoint 2: Add code to test your new methods to the main code of `Date.py`. Show your code and the test results to a TA or a mentor. Your testing code should show `Date` objects and method call outputs that test the different conditions that each method has to handle. The example calls above cover some but not all of these conditions.

Please come to lab for the last checkpoint.