Computer Science 1 — CSci 1100
Lab 9 — Date class
Spring Semester 2014

Lab Overview

This lab investigates Python classes and their use. The particular focus is a `Date` class. Please download `Date.py` from the Laboratories heading of the Piazza resource page. Please also download `Point2d.py` from the Lectures heading. This is the solution from Lecture 17, with additional explanatory comments. Please continually refer to this example when working on the lab.

Start by looking `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.
- The 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 must 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.

- `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 functions are enough for now.

To complete Checkpoint 2: Add code to test your new functions 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 function 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.

Checkpoint 3 — Birthdays

Create a new .py file for Checkpoint 3 that starts with the line

```
from Date import *
```

This will allow you to use the `Date` class without having to write something like

```
d = Date.Date()
```

which is what you would have to do if you just imported `Date`. Instead, you can simply write:

```
d = Date()
```

Start by writing a function that reads birthdays from a file (`birthdays.txt` is the test file we have provided) and returns a list of Date objects.

Then write main function code that scans through this list to find and output:

- the earliest birthday — corresponding to the person who would be the oldest person represented in the file,
- the latest birthday — corresponding to the youngest person represented in the file, and
- the name of the month that has the most birthdays. Use the `month_names` list in `Date.py`

To complete Checkpoint 3: Show a TA or Mentor your code and output.

Additional challenges (no extra credit)

Implement and test a function called `julian_day` that returns an integer giving the "Julian day" corresponding to the current date. The Julian day is the numbered day within the year, with January 1 being Julian day 1, February 1 being Julian day 32, February 28 being Julian day 59, and March 1 being Julian day 60 or 61, depending on whether or not it is a leap year. This method should make use of the `days_in_month` list. Here are examples of using the method

```
>>> d1 = Date(1972, 3, 27)
>>> d2 = Date(1998, 4, 13)
>>> d1.julian_day()
87
>>> d2.julian_day()
103
```

You must make use of `is_leap_year` in your `julian_day` function.