Homework 6

Due 3:59pm, Wednesday, March 13, 2018

Installation and Handin

Homework Setup. For each homework assignment, there may be support files that you will need to complete the assignment. These can be copied to your home directory by using the `cs4_install` command in a Brown CS Terminal window. For this homework type

```
cs4_install hw06
```

There should now be a `hw06` folder within your homeworks directory. Using Terminal, you can move into the `hw06` folder with the cd command:

```
cd ~/course/cs0040/homeworks/hw06
```

Homework hand-in. Be sure to turn in all the files requested and that they are named exactly as specified, including spelling and case. When you're ready to submit the files, run:

```
cs4_handin hw06
```

from a Brown CS Terminal window from your `~/course/cs0040/homeworks/hw06` directory. The entire contents of your `~/course/cs0040/homeworks/hw06` directory will be handed in. Check for a confirmation email to ensure that your assignment was correctly submitted using the `cs4_handin` command. You can resubmit this assignment using the `cs4_handin` command at any time, but be careful, as only your most recent submission with be graded.

Part I: First **`Date`** with OOP (60)

Put your answers for this problem in a plain-text file named `hw06_1.py`.

In their age long struggle against the babies, the puppies have learned something vital--some babies aren't actually babies: they're toddlers! The puppies have tasked you to with using python to create revolutionary new date-finding technology to figure out how old some of these "babies" are to find which ones are the fakers.

In this problem, you will create a `Date` class, from which you will be able to create `Date` objects that represent a day, month, and year. You will add functionality to this class that will enable `Date` objects to find the day of the week to which they correspond.

Support Code

Take a moment to look over the `hw06_1.py` file as it stands so far. We have given you the following methods to start:

- The `__init__(self, month, day, year)` method, which is the constructor for `Date` objects. In other words, this is the method that Python uses when making a new `Date` object. It defines the attributes that compose a `Date` object (`month`, `day`, and `year`) and accepts parameters to set an object's attributes to some initial values.
- The `__repr__(self)` method, which returns a string representation of a `Date` object. This method will be called when an object of type `Date` is printed. It can also be tested by simply evaluating an object from the Shell. This method formats the month, day, and year that represent a `Date` object into a string of the form `'mm/dd/yyyy'` and returns it.
- The `isLeapYear(self)` method, which returns `True` if the called object is in a leap year, and `False` otherwise. In other words, when we create a `Date` object and call its `isLeapYear` method, the method will return whether that specific `Date` object falls in a leap year. There are no double-underscores here, because Python doesn't "expect" this method, but it certainly doesn't "object" to it either. (Clearly our puns have no class!)
- The `copy(self)` method, which returns a newly-constructed object of type `Date` with the same month, day, and year that the called object has. This allows us to create *deep copies* of `Date` objects.

The `Date` class provided also has three data members:

- A data member holding the month (this is `self.month`)
- A data member holding the day of the month (this is `self.day`)
- A data member holding the year (this is `self.year`)

Read over the starter code that we've given you. Make sure that you understand how the various methods work.
Then, try the following interactions in the Python Shell to experiment with the `__init__`, `__repr__`, and `isLeapYear` methods:

```
# Create a Date object named d1 using the constructor.
>>> d1 = Date(4, 10, 2016)

# An example of using the __repr__ method. Note that no quotes
# are displayed, even though the function returns a string.
>>> d1
04/10/2016

# Check if d1 is in a leap year -- it is!
>>> d1.isLeapYear()
True

# Create another object named d2
>>> d2 = Date(1, 1, 2017)

# Check if d2 is in a leap year.
>>> d2.isLeapYear()
False
```

Next, try the following examples in the Python Shell (by running python3 in your Terminal in the directory that has all of your files) to illustrate why we will need to override the `__eq__` method to change the meaning of the == operator:

```
>>> d1 = Date(1, 1, 2016)
>>> d2 = d1
>>> d3 = d1.copy()

# Determine the memory addresses to which the variables refer.
>>> id(d1)
430542          # Your memory address may differ.
>>> id(d2)
430542          # d2 is a reference to the same Date that d1 references.
>>> id(d3)
413488          # d3 is a reference to a different Date in memory.
```

```
# The == operator tests whether memory addresses are equal.
>>> d1 == d2
True            # Shallow copy -- d1 and d2 have the same memory address.
>>> d1 == d3
False           # Deep copy -- d1 and d3 have different memory address
```

If having trouble accessing Date from the terminal, try running `from hw06_1 import Date` from the python terminal, which should then give you access to the module.

Problem 6.1a) A Better Date

Add the following methods to the date class:

Method `equals(self, d2)`
This method should return `True` if the calling object (named `self`) and the argument (named `d2`) represent the same calendar date. If they do not represent the same calendar date, this method should return `False`. The examples above show that the same calendar date may be represented at multiple locations in memory—in that case the `==` operator returns `False`. `equals(self, d2)` can be used to see if two objects represent the same calendar date, regardless of whether they are at the same location in memory.

Example output:

```
>>> d1 = Date(1, 1, 2016)
>>> d1
01/01/2016
>>> d2 = Date(1, 1, 2016)
>>> d2
01/01/2016
>>> d1 == d2
False
>>> d1.equals(d2)
True
>>> d1.equals(Date(1, 1, 2016))  # this is OK, too!
True
>>> d1 == d1.copy()
False
```

Method `tomorrow(self)`

This method should return a new date object corresponding to a day after the calling object. This means that `day` will definitely be incremented. What's more, `month` and `year` might be incremented as well.

Hint: To tell how many days there are in February, explore the numeric properties of boolean values. For example, `True + 1 = 2` and `False + 1 = 1`.

Hint: Make a list consisting of the days in each month of the year. Have the element at index `0` be `0` for ease of indexing.

Example output:

```
>>> d = Date(2, 28, 2016)
>>> new_date = d.tomorrow()
>>> new_date
02/29/2016
>>> (d.tomorrow()).tomorrow()
03/01/2016
>>> d
02/28/2016
```

Method `yesterday(self)`
This method should return a new date object corresponding to a day before the calling object. This means that `day` will definitely be decremented. What's more, `month` and `year` might be decremented as well.

Example output:

```
>>> d = Date(1, 1, 2016)
>>> d.yesterday()
12/31/2015
>>> d
01/01/2016
```

Method `addNDays(self, N)`
Alters the invoking `Date` object so that it is `N` calendar days in the future. This method only needs to handle nonnegative integer arguments `N`.

Example outputs:

```
>>> d = Date(11, 11, 2015)
>>> d.addNDays(3)
>>> d
11/14/2015
```

You can check your own date arithmetic with this website.

Method `subNDays(self, N)`
Mutates the invoking `Date` object so that it is `N` calendar days before the starting date. This method only needs to handle nonnegative integer arguments `N`.

Example output:

```
>>> d = Date(11, 04, 2015)
>>> d.subNDays(5)
>>> d
10/30/2015
```

Try reversing the test cases from `addNDays`!

Method `isBefore(self, d2)`
This method should return `True` if the calling object is a calendar date before the argument named `d2` (which will always be an object of type `Date`). If `self` and `d2` represent the same day, this method should return `False`. Similarly, if self is after `d2`, this should return `False`.

Hint: Try comparing the list `Lself = [self.year, self.month, self.day]` with the list `L2 = [d2.year, d2.month, d2.day]`.

Example output:

```
>>> ny = Date(1,1,2016)    # New Year's
>>> d2 = Date(11,11,2015)
>>> ny.isBefore(d2)
False
>>> d2.isBefore(ny)
True
>>> d2.isBefore(d2)
False
```

Method `isAfter(self, d2)`
This method should return `True` if the calling object is a calendar date after the argument named `d2` (which will always be an object of type `Date`). If `self` and `d2` represent the same day, this method should return `False`. Similarly, if `self` is before `d2`, this should return `False`.

You can emulate your `isBefore` code here OR you might consider how to use the `isBefore` and or `equals` method to write `isAfter`.

Example output:

Try reversing the test cases of your method `isBefore`!

Method `diff(self, d2)`
This method should return an integer representing the number of days between `self` and `d2`. You can think of it as returning the integer representing `self - d2`. The method can return any integer. Construct your code accordingly.

NOTE: This method should **NOT** change `self` **NOR** should it change `d2`!

Hint: Make copies of `self` and `d2`. For example:

- `self_copy = self.copy()`
- `d2_copy = d2.copy()`

Example output:

```
>>> d1 = Date(11,11,2015)
>>> d2 = Date(12,18,2015)
>>> d2.diff(d1)
37
>>> d1.diff(d2)
-37
>>> d1
11/11/2015
>>> d2
12/18/2015
```

Use your `diff` method to compute your own age - or someone else's age - in days!
You can check other differences at www.timeanddate.com/date/duration.html .

Method `dow(self)`

This method should return a string that indicates the day of the week (dow) of the object (of type `Date`) that calls it. That is, this method returns one of the following strings: `"Monday"`, `"Tuesday"`, `"Wednesday"`, `"Thursday"`, `"Friday"`, `"Saturday"`, or `"Sunday"`.

Hint: Consider making a dictionary out of the days of the week.

Hint: How might it help to find the `diff` from a known date, like Wednesday, November 11, 2015? How might the mod (`%`) operator help?

Example output:

```
>>> d = Date(12, 7, 1941)
>>> d.dow()
'Sunday'

>>> d = Date(1, 1, 2100)
>>> d.dow()
'Friday'
```

You can views days of the week for an entire year at www.timeanddate.com/calendar/

NOTE: Before moving on, make sure you've thoroughly tested all of the methods you just wrote!

Problem 6.1b) Custom Operators

In Python you can also define operators for your own classes. For example, since the above `equals` method is how we want to express equality,i.e., `==`, you can override (i.e. redefine) the **==** operator by adding a method named `__eq__` (note that that name has two underscores on each side of the `eq`).

Method `__eq__(self, other)`
Write a method `__eq__(self, other)` that returns `True` if the called object (`self`) and the argument (`other`) represent the same calendar date (i.e., if the have the same values for their `day`, `month`, and `year` attributes). Otherwise, this method should return `False`.

Recall from lecture that the name `__eq__` is a special method name that allows us to *override* the `==` operator–replacing the default version of the operator with our own version. In other words, when the `==` operator is used with `Date` objects, our new `__eq__`method will be invoked!

This method will allow us to use the == operator to see if two `Date` objects actually represent the same date by testing whether their days, months, and years are the same, instead of testing whether their memory addresses are the same.

After implementing your `__eq__` method, try re-executing the following sequence of statements:

```
>>> d1 = Date(1, 1, 2016)
>>> d2 = d1
>>> d3 = d1.copy()

# Determine the memory addresses to which the variables refer.
>>> id(d1)
430542              # Your memory address may differ.
>>> id(d2)
430542              # d2 is a reference to the same Date that d1 references.
>>> id(d3)
413488              # d3 is a reference to a different Date in memory.

# The new == operator tests whether the internal date is the same.
>>> d1 == d2
True                # Both refer to the same object, so their internal
                    # data is also the same.
>>> d1 == d3
True                # These variables refer to different objects, but
                    # their internal data is the same!
```

Notice that we now get `True` when we evaluate `d1 == d3`. That's because the new `__eq__` method compares the internals of the objects to which `d1` and `d3` refer, rather than comparing the memory addresses of the objects, whereas the default `__eq__` method only tests "shallow equality" meaning it will only test whether or not the memory addresses of two objects are the same.

As part of the problem, please also include methods for the following operators:

- Implement `__lt__` to define the < operator.
- Implement `__gt__` to define the > operator.
- Implement `__sub__` to define the (-) operator.

Be sure to reuse method(s) you wrote previously!

Part II: Using Your Date Class (40)

Put your answers for this problem in a plain-text file named `hw06_2.py`.

Now that you have written a functional `Date` class, we will put it to use to expose the fake babies for what they really are! Remember that the `Date` class is only a blueprint, or template, for how `Date` objects should behave. We can now create `Date` objects according to that template and use them in *client* code.

Import the Class

IMPORTANT: Since your clients will need to construct `Date` objects, you need to import the `Date` class. Therefore, make sure that `hw06_2.py` is in the same directory as `hw06_1.py`, and include the following statement at the top of `hw06_2.py`:

```
from hw06_1 import Date
```

We haven't done this for you because the import command is such a common way of accessing additional python functionality, so you need to get used to using it yourself!

Problem 6.2a) get_age_on(birthday, other)

Write a function named get_age_on(birthday, other)that accepts two Date objects as parameters: one to represent a person's birthday, and one to represent an arbitrary date. The function should then return the person's age on that date as an integer.

Notes: You can assume that the other parameter will represent a date on or after the birthday date. Hint: It may be helpful to construct a new Date object that represents the person's birthday in the year of other. That way, you can determine whether the person's birthday has already passed in the year of other, and use that information to calculate the age.

Example output:

```
>>> birthday = Date(6, 29, 1994)
>>> d1 = Date(2, 10, 2014)
>>> get_age_on(birthday, d1)
19
>>> d2 = Date(11, 10, 2014)
>>> get_age_on(birthday, d2)
20
```

Problem 6.2b) print_birthdays(filename)

Write a function `print_birthdays(filename)` that accepts a string filename as a parameter. The function should then open the file that corresponds to that filename, read through the file, and print some information derived from that file (you do not need to write a test case for this function).

More specifically, the function should assume that the file in question contains information about birthdays in lines of the following format:

```
Name,month,day,year
```

In other words, each line of the file contains comma-separated birthday data.

The function should read this file line-by-line, and print the person's name, birthday, and the day of the week on which the person was born in the following format: name (mm/dd/yyyy) (day)

For example, the file birthdays.txt contains the following data:

```
George Washington,2,22,1732
Abraham Lincoln,2,12,1809
Susan B. Anthony,2,15,1820
Franklin D. Roosevelt,1,30,1882
Eleanor Roosevelt,10,11,1884
```

Therefore, calling `print_birthdays` with this filename should print the following information:

```
>>> print_birthdays('birthdays.txt')
George Washington (02/22/1732) (Friday)
Abraham Lincoln (02/12/1809) (Sunday)
Susan B. Anthony (02/15/1820) (Tuesday)
Franklin D. Roosevelt (01/30/1882) (Monday)
Eleanor Roosevelt (10/11/1884) (Saturday)
```

Notes: For every line of the file, you will need to create a Date object and invoke the appropriate methods on the object to get the information needed.

Hint: You can get a string representation of a Date object named d using the expression str(d).

Testing

You are required to write test cases for every function you implement in this homework that does not explicitly tell you to not include test cases. Be sure to develop (or include) appropriate a set of test cases for each problem you solve. **Note that if for any function your code does not compile, you can at most receive half credit on that problem.** You have to not only write your tests, but run them as well!

Please let us know if you find any mistakes, inconsistencies, or confusing language in this document or have any concerns about this and any other CS4 document by posting on Piazza or filling out our anonymous feedback form.