CS:1210 Practice Problem Set 13

Complete before Tuesday, 5-5-2015

1. The *greatest common divisor (gcd)* of two positive integers x and y can be computed using the very old *Euclid's algorithm*. Euclid's algorithm is based on the principle that the gcd of two numbers does not change if the larger number is replaced by the remainder obtained by dividing the larger number by the smaller number. In other words,

   $$gcd(21, 35) = gcd(21, 14) = gcd(7, 14) = 7.$$

   Implement this algorithm as a recursive function with the function header:

   ```
   def gcd(a, b):
   ```

   Pay close attention to the base case(s).

2. Implement a class called `rational` for representing rational numbers. This class should contain two functions: `__init__` and `__repr__`and I'd like to use this class as follows:

   ```
   >>> r = rational(10, 15)
   >>> r
   2/3
   >>> print(r)
   2/3
   >>> p = rational(11, 7)
   >>> p
   11/7
   p.denominator
   7
   p.numerator
   11
   ```

   As you can see from the above example, the numerator and denominator of the rational are stored in reduced form (i.e., after reduction by the gcd). Functions in your `rational` class can call the `gcd` function implemented in the previous problem.

3. We want to define a class called `employeeInfo` that a company uses to maintain a collection of employee records. Each employee record contains a *name*, a *social security number*, a *salary*, and an *employment start date*. In addition to an initialization method (`__init__`) and a representation method (`__repr__`), the class provides an `add` method for adding an employee record to the collection and a `remove` method for deleting an employee record. Here is an example of how a user might interact with the `employeeInfo` class:

   ```
   >>> emp = employeeInfo()
   >>> emp.add("Isaac Newton", 31415926, 1000000, "05152013")
   >>> emp.add("Robert Boyle", 15793861, 5000000, "11152010")
   >>> emp
   Robert Boyle 15793861 5000000 11152010
   Isaac Newton 31415926 1000000 05152013
   >>> emp.add("Robert Hooke", 53589793, 200000, "04152012")
   >>> emp
   Robert Boyle 15793861 5000000 11152010
   Isaac Newton 31415926 1000000 05152013
   Robert Hooke 53589793 200000 04152012
   >>> emp.remove(31415926)
   ```

```
>>> emp
Robert Boyle 15793861 5000000 11152010
Robert Hooke 53589793 200000 04152012
```

We assume that employees have distinct social security numbers and no employee appears twice in the collection of employee records. Therefore, we can use a dictionary-based implementation with the social security numbers acting as keys. Below we provide implementation of the initialization method and the `add` method:

```
class employeeInfo():

    def __init__(self):
        self.D = {}

    def add(self, name, ssn, salary, start):
        self.D[ssn] = [name, salary, start]
```

(a) Implement the `remove` method.
(**Hint:** This just takes two lines of code including the function header.)

(b) Implement the `__repr__` method. Recall that the `__repr__` method is required to return a string. The examples above of interacting with the `employeeInfo` class tell us that the string returned by `__repr__` contains information about each employee separated by the end-of-line character. Also, each employee's information contains the employee's name, employee's social security number, employee's salary, and employee's starting date, *in that order*, separated by a single blank character. Finally, note that the employee information appears in increasing order of social security numbers.

(c) Now suppose that we change the implementation of the `employeeInfo` class, without changing how it behaves to an outside user. Specifically, instead of using a dictionary to store the collection of employee records, we will use a list. Below we provide part of the new implementation, namely the initialization method and the `__repr__` method:

```
class employeeInfo():

    def __init__(self):
        self.L = []

    def __repr__(self):
        s = ""
        for x in self.L:
            s = s+str(x[0])+" "+str(x[1])+" "+str(x[2])+" "+str(x[3])+"\n"

        return s.strip()
```

Your task for this problem is to implement the `add` method.
(**Hint:** The implementation of the `__repr__` shown above contains many clues about how the list of employee records is organized.)
