Compact Course Python

Exercise 4

---

1 Fractions

Define a class for rational numbers (see lecture slides). The constructor shall take numerator and denominator as its arguments. The instance objects shall work with the standard Python operators `+`,`-`,`*` and `/`:

```
>>> Rat(2,5) + Rat(3,4)
23/20
```

To achieve this, you need to implement `__add__(self,b)`, `__sub__(self,b)`, `__mul__(self,b)` and `__truediv__(self,b)`, which all return new `Rat` objects. To make the pretty-print work, you need to implement the method `__repr__(self)` which returns a string.

Extra task: Implement the Euclidean Algorithm to compute the greatest common divisor and use it for a method that cancels down `Rat` objects. Like in the example above, all calculations (+, *, ...) shall return cancelled fractions.

2 Tic-tac-toe

Goal of this exercise is to implement Tic-tac-toe as a small Python game for two players. The interaction with the game is managed via the shell. Tic-tac-toe is placed on a game board with 3x3 squares. Two players using different symbols move in turns and put their symbol on a free square. The winner is the one that first fills a whole row, column or diagonal with his or her symbols.
(See also: `http://en.wikipedia.org/wiki/Tic-tac-toe`)

We provide you with a ready-to-use class named `GameBoard` that implements different functionalities of a game board: You can initialize its constructor with the edge length (= number of squares) of the board. For you own methods, you mainly need the method that put a symbol on a certain square and the one that checks which symbol is on a certain square. Squares are named like in a coordinate plane, starting at 0. (E.g. `(1 1)` is the field in the middle of the Tic-tac-toe board.)

We also wrote a game template for you (`TicTacToe`). You have to implement its (empty) methods:

- `__init__` shall initialize the board and every other helper structure that you'll need later. We already fixed the symbols for the two players here: `self.white` is the symbol of the player that opens the game, the other one is `self.black`.
- `isLegalMove(self ,x,y,s)` shall return `True` if the symbol s can be placed on square (`(x,y)`), otherwise `False` (in case the square is taken or does not exist).
- `move(self ,x,y,s)` puts a symbol `s` on the square `(x,y)`
- `evaluateBoard(self)` checks the current board. The method shall return -1 if the game is not over yet; 0 if the first player (white) has won the game; 1 for a draw; 2 if the second player has won.

We implemented the method `play()` for you. It processes the players' moves and prints the current board to the screen. It also announces the winner (or a draw) as soon as the game has finished. You can make moves by entering the coordinates of the next square you want to put you piece on. (The player may enter human-readable coordinates - the numbers start at 0, (2,2) is the middle of the board. This only concerns you in case you're playing the game – for implementing your algorithms, stick to the hints given before.)

```
+ ---- + ---- + ---- +
+      +      +  o   +
+ ---- + ---- + ---- +
+      +  x   +  x   +
+ ---- + ---- + ---- +
+  o   +      +  o   +
+ ---- + ---- + ---- +
x's move?
1 2
+ ---- + ---- + ---- +
+      +      +  o   +
+ ---- + ---- + ---- +
+  x   +  x   +  x   +
+ ---- + ---- + ---- +
+  o   +      +  o   +
+ ---- + ---- + ---- +
x wins!
```
