121COM: Introduction to Computing

Academic Year 2015/16

LabSheet 8

For use in labs the week beginning Mon 16th November 2015

Author: Dr Matthew England (Coventry University)

C Lab Exercises 1 - Fractions

Q1 Define a Python class called `Fraction`. Every Fraction object consists of two attributes `n` and `d` representing the numerator and denominator.

(a) Note that `n` and `d` should both be integers. Ensure this by having the instantiation method throw an appropriate exception otherwise.

(b) Further, `d` should always be positive. Extend the instantiation method so that if `d` is zero an exception is thrown; while if `d` is negative the equivalent object is created with attributes $-n, -d$.

(c) Write a `__str__` method that converts a `Fraction` instance to the string `"n/d"`.

(d) Recall the following rules for arithmetic with fractions.

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 * d_2 + n_2 * d_1}{d_1 * d_2}$$
$$\frac{n_1}{d_1} - \frac{n_2}{d_2} = \frac{n_1 * d_2 - n_2 * d_1}{d_1 * d_2}$$
$$\frac{n_1}{d_1} * \frac{n_2}{d_2} = \frac{n_1 * n_2}{d_1 * d_2}$$
$$\left(\frac{n_1}{d_1}\right) / \left(\frac{n_2}{d_2}\right) = \frac{n_1 * d_2}{d_1 * n_2}$$

Use these to implement the appropriate arithmetic special methods for the class `Fraction`. See the documentation at the link below to get the right names.

`https://docs.python.org/3.4/reference/datamodel.html#emulating-numeric-types`

I Lab Exercises 2 - Extensions

Q2 Consider again your `Fractions` class from Q1 of this labsheet. Two possible extensions:

(a) Note that different `Fraction` objects can be mathematically equal. For example, $2/4 = 1/2$. In general $n_1/d_1 = n_2/d_2$ if and only if $n_1 * d_2 = n_2 * d_1$. Write a method that takes two `Fraction` objects; and returns `True` is they are mathematically equal and `False` otherwise.

Give it the appropriate special method name so that Python will use it when checking equality of two `Fraction` objects with `==`. You could write similar methods for checking whether one fraction is bigger or smaller than another.

(b) All equivalent mathematical fractions can be simplified to a unique form by cancelling common factors in the numerator and detonator. For example,

$$\frac{30}{18} = \frac{2*15}{2*9} = \frac{15}{9} = \frac{3*5}{3*3} = \frac{5}{3}.$$

Write a function to simplify a `Fraction`. Make this a method of the class and run it as part of the instantiation of any `Fraction` object.

Q3 Consider again your `BankAccount` class from the Exercise at the end of Worksheet 8. Can you think of an appropriate meaning for the addition of a bank account to an integer? If so, implement the special method `__add__`. How about the addition of two bank accounts?

Extended Task

This task uses the API at the url below which provides data on the major European football leagues.

`url = http://api.football-data.org`

Select a league and season from those listed at `url/alpha/soccerseasons` and take a note of the `id` code (for example, the 2015-16 season of the English premier league has `id` 398). Do the following in Python:

- Define a class `Team` with attributes `fullName`, `abbreviation` and `squadValueEuros`. The instantiation should ensure the first two attributes are strings and the third an integer. Create instances of `Team` for all the teams in your chosen league-season using the data at `url/alpha/soccerseasons/<id>/teams`.
- In the same script define another class `Fixture` with attributes `homeTeam`, `awayTeam`, `homeTeamScore` and `awayTeamScore`. Ensure these are of appropriate types (note how the API records the score for unplayed fixtures) Create instances using the data at `url/alpha/soccerseasons/<id>/fixtures`.
- Write a function to analyse the completed fixtures and thus determine the current league table. Note that a win gives a team 3 points, a draw 1 point and a loss zero points. When two teams have the same points the one with the greater goal difference is ranked higher. Compare your table with the one on your favourite news website to check it is correct.

Ensure your objects can be converted to appropriate strings for printing; and that your classes have docstrings and comments. If you make a lot of requests get a key from `www.football-data.org/register`.

Extensions you could consider:

- Compare the `squadValueEuros` for a team with its performance. Is there a strong link between the market value of a team's players and its performance in your league?
- Before the 1990s many leagues awarded only 2 points for a win. How would your league table change?
- Generalise your code so a user can state the id of their preferred league-season. Create a third class `League` whose attributes are teams and fixtures; and which has a method to calculate the current league table.
- Note that each team has its own `teamID` and that the API provides details on a team's players at `url/alpha/teams/<teamID>/players`. Attributes include their `name`, `nationality` and `marketValue`. You could use this data to create a simple fantasy football game: each player chooses players up to a certain market value; they receive points for the goals scores by the teams of their attacking players and lose points for the goals conceded by the teams of their defending players.