The PracticeFractions Project

Library modules used: `fractions.Fraction` *and* `operator`
Third party module used: `easygui`
File to be submitted: `practiceFractions.py`

The goal of this project is a program to be used to practice fraction arithmetic. It must have a graphical user interface. Once the program is started, there is to be no use of the command prompt. To keep things simple, you will use the `easygui` module whose model is not event-driven. You should become familiar with this module first.

Comment: when presenting a user with fewer than five choices, use a buttonbox rather than a choices window. I find the latter to be very inelegant in that the box containing the choices is the same width regardless of the length of the choice strings.

A *fraction string* is either a string representation of an integer or two integer strings joined by '/'. Examples: f1 = '25', f2 = '-5/6', etc. There should be no whitespace before or after the '/'. Also, the denominator part should represent a nonzero integer.

We will restrict our fraction operators to +, -, and `*`. To keep things simple, we will not consider fraction division. A fraction expression consists of two fraction strings separated by an operator character. The operator may be surrounded by spaces, but this is not required.

Your program should start with a main window with a brief message and three choices (buttons): Solver, Quizzer and Quit.

Clicking on the Solver button should bring up the Solver window where the user may enter an arbitrary fraction expression. If the entered string is a valid fraction expression, the value of the expression will be displayed with buttons to solve another or return to the main window; otherwise a window appears with an error message and buttons to try again or return to the main window.

Clicking on the Quizzer button of the main window will bring up the Quizzer window, where the user select one of the operators ('+','-','*'). This could be done with four buttons – one for each operator and one to get the expression, which will bring up the quiz question window.

The quiz question window will display a random fraction expression followed by '=' and space for the user to enter the answer. The fractional expression should consist of two randomly chosen fractions joined by the operator symbol. For a random fraction, you should choose a random integer between -15 and 15 for the numerator and a random integer in the range from 1 to 15 for the denominator. There are three possible outcomes: correct, correct value but not reduced and incorrect. After the user enters their answer, a message window appears with the results. There are three possible outcomes: completely correct, correct but not reduced and incorrect. If the user's answer is completely correct, a congratulations message will appear; it the answer is

not completely correct the message will indicate the outcome and show the correct answer in the form of the expression, '=' and the correct answer.

The fractions.Fraction module

```
>>> from fractions import Fraction
>>>
>>> Fraction(3,5)
Fraction(3, 5)
>>> print(Fraction(3,5))
3/5
>>> Fraction('3/5')
Fraction(3, 5)
>>> Fraction(6)
Fraction(6, 1)
>>> Fraction(24,36)
Fraction(2, 3)
>>> F = Fraction('3/7')
>>> F.numerator
3
>>> F.denominator
7
```

The `operator` module

The `operator` module provides functions corresponding to many of the defined operators. For example, `operators.add, operators.sub, operators.mul.` Their use is illustrated below.

```
>>> import operator
>>> operator.add(2,5)
7
>>> operator.mul(2,5)
10
>>> operator.add(Fraction(1,5),Fraction(3,2))
Fraction(17, 10)
```

Technique: using a dictionary to emulate the C++ switch statement.

Switch statements are used to consolidate long if-else if … constructs. While python has no switch statement, python dictionaries can be used instead.

```
>>> def apply_op(a,b,op):
...     if op == '+':
...             return a+b
...     if op == '-':
...             return a-b
...     if op == '*':
...             return a*b
...
>>> opfunc = {'+':operator.add,
              '-':operator.sub,
              '*':operator.mul
             }
>>> def apply_op2(a,b,op):
...     return opfunc[op](a,b)
...
>>>
```