Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2012

Lab 11 - Introduction to Object-Oriented Programming (Implementing a Class)

Objective

To design, implement and test the methods of a simple Python class

Attendance/Demo

To receive credit for this lab, you must make an effort to complete the exercises, and demonstrate your work.

When you have finished all the exercises, call a TA, who will review your code. For those who don't finish early, a TA will ask you to demonstrate whatever exercises you've completed, starting about 30 minutes before the end of the lab period. Finish any exercises that you don't complete before the end of the lab on your own time.

Getting Started

Open fractions.py in Wing IDE 101. This module contains a (very) incomplete implementation of a class named Fraction. Don't delete any of the code or documentation strings (the comments enclosed in triple double-quotes: """).

Exercise 1

A fraction is a rational number expressed in the form *a*/*b*, where *a* (the numerator) and *b* (the denominator) are integers.

Read the documentation string for the constructor for class Fraction (the method named __init__) and write the method's code. Don't delete the documentation string! Every Fraction object should have two instance variables named num and den, which store the fraction's numerator and denominator, respectively. If the denominator is 0, the method should print 'Zero denominator' and return without initializing the Fraction object.

Don't delete the method call self.reduce(). This should be the last statement in your method. Currently, this method does nothing, but after you implement it, it will convert your fraction to reduced form (you'll do this in Exercise 3).

Interactively test your function by typing the following statements in the shell and verifying that you obtain the correct results. Recall that every object has a variable named __dict__, which is bound to the dictionary that stores the object's instance variables. To observe an object's instance variables, all we need to do is evaluate __dict__ (remember to type two leading and two trailing underscores).

```
>>> f = Fraction(1, 3)
>>> f.__dict__
{'num': 1, 'den': 3}
```

```
>>> f = Fraction(6, 8)
>>> f.__dict__
{'num': 6, 'den': 8}

>>> f = Fraction(-2, 3)
>>> f.__dict__
{'num': -2, 'den': 3}

>>> f = Fraction(4, -5)
>>> f.__dict__
{'num': 4, 'den': -5}

>>> f = Fraction(-6, -8)
>>> f.__dict__
{'num': -6, 'den': -8}

>>> f = Fraction(0, 7)
>>> f.__dict__
{'num': 0, 'den': 7}

>>> f = Fraction(2, 0)
Zero denominator
>>> f.__dict__
{}       # The object has no instance variables, so its dictionary is empty
```

Exercise 2

The *greatest common divisor* of two integers *a* and *b* is the largest positive integer that evenly divides both values. Here is *Euclid's algorithm* for calculating greatest common divisors, which uses iteration and calculation of remainders:

1. Assign the absolute value of *a* to *q* and the absolute value of *b* to *p*.
2. Assign the remainder of *q* divided by *p* to *r*.
3. while *r* is not 0:
   Assign *p* to *q* and *r* to *p*.
   Assign the remainder of *q* divided by *p* to *r*.
4. *p* is the greatest common divisor (return this value).

Module fraction.py contains an incomplete definition of a function named gcd. Read the documentation string for this function and implement it using Euclid's algorithm. You should delete the pass statement, but don't delete the documentation string. Recall that Python has a built-in function named abs, which returns the absolute value of its argument.

Interactively test your function by typing the following statements in the shell and verifying that the results returned by gcd are correct.

```
>>> gcd(42, 30)
```

```
6

>>> gcd(30, 42)
6

>>> gcd(-42, 30)
6

>>> gcd(42, -30)
6

>>> gcd(-42, -30)
6

>>> gcd(144, 55)
1
```

Exercise 3

A *reduced fraction* is a fraction *a*/*b* written in lowest terms, by dividing the numerator and denominator by their greatest common divisor. For example, 2/3 is the reduced fraction of 8/12.

For our purposes, we'll also include the following in our definition of a reduced fraction:

- if the numerator is equal to 0 the denominator is always 1;
- if the numerator is not equal to 0 the denominator is always positive, and the numerator can be positive or negative.

Module fraction.py contains an incomplete definition of a method named reduce. Read the documentation string for this method, carefully, and implement it. This method's algorithm is not complex, but note that there are several cases (all described in the method's documentation string) that you'll have to implement.

Check your __init__ method from Exercise 1 and verify that the last statement in __init__ calls reduce this way: self.reduce().

If your implementation of reduce is correct, every Fraction object will be reduced when it is initialized. You're now going to repeat the tests you performed in Exercise 1 and verify that Fraction objects now represent reduced fractions.

```
>>> f = Fraction(1, 3)
>>> f.__dict__
{'num': 1, 'den': 3}

>>> f = Fraction(6, 8)
>>> f.__dict__
{'num': 3, 'den': 4}

>>> f = Fraction(-2, 3)
```

```
>>> f.__dict__
{'num': -2, 'den': 3}

>>> f = Fraction(4, -5)
>>> f.__dict__
{'num': -4, 'den': 5}

>>> f = Fraction(-6, -8)
>>> f.__dict__
{'num': 3, 'den': 4}

>>> f = Fraction(0, 7)
>>> f.__dict__
{'num': 0, 'den': 1}
```

Exercise 4

Read the documentation string for the __str__ method and implement it. Note that this method returns a string - it does not print one.

Interactively test your method by typing the following statements in the shell and verifying that the results returned by __str__ are correct.

```
>>> f = Fraction(1, 3)
>>> s = f.__str__()
>>> type(s)        # Verify that the value returned by __str__ is a string
<type 'str'>

>>> s
'1/3'

>>> str(f)         # str calls __str__ on the object bound to f and returns that string
'1/3'

>>> print f        # print calls __str__ to get the string it displays
1/3

>>> f = Fraction(1, -3)
>>> str(f)
'-1/3'        # Note that the fraction is stored in reduced form (+ve denominator)

>>> print f
-1/3
```

Exercise 5

Read the documentation string for the __repr__ method and implement it. Note that this method returns a string - it does not print one.

Interactively test your method by typing the following statements in the shell and verifying that the results returned by __repr__ are correct.

```
>>> f = Fraction(1, 3)
>>> s = f.__repr__()
>>> type(s)          # Verify that the value returned by __repr__ is a string
<type 'str'>

>>> s
'Fraction(1, 3)'

>>> repr(f)          # repr calls __repr__ on the object bound to f and returns that string
'Fraction(1, 3)'

>>> f                  # The shell calls __repr__ to get the string representation of the
Fraction(1, 3)       # object bound to variable f

>>> f = Fraction(1, -3)
>>> repr(f)
'Fraction(-1, 3)'    # Note that the fraction is stored in reduced form (+ve denominator)

>>> f
Fraction(-1, 3)
```

Exercise 6

If a class defines a method named __add__ and variable x is bound to an instance of the class, Python calls that __add__ method when it executes an expression of the form x + y. In other words, Python converts x + y to x.__add__(y).

Read the documentation string for the __add__ method and implement it. The fraction returned by this function must be in reduced form (the constructor takes care of this when the Fraction object containing the sum is created).

Note that the sum of two fractions:

$$\frac{a}{b}+\frac{c}{d}$$

is not calculated as:

$$\frac{a+c}{b+d}$$

If you don't remember the formula for adding fractions, look at this page:

http://mathworld.wolfram.com/Fraction.html

Interactively test your method by typing the following statements in the shell and verifying that the results returned by __add__ are correct.

```
# Try 2/3 + 1/2
>>> f1 = Fraction(2, 3)
>>> f2 = Fraction(1, 2)

>>> f3 = f1 + f2
>>> type(f3)
<class '__main__.Fraction'>

>>> print f1
2/3             # Verify that addition didn't change f1
>>> print f2
1/2             # Verify that addition didn't change f2
>>> print f3
7/6

# Try 2 + 1/2
>>> f1 = Fraction(2, 1)
>>> f3 = f1 + f2
>>> print f3
5/2
```

Exercise 7

By default, the == operator determines object identity; that is, the expression x == y returns True only if x and y are bound to the same object. Usually, we want == to determine object equality; that is, the expression x == y should return True if x and y are bound to objects that have the same value.

If a class defines a method named __eq__ and variable x is bound to an instance of the class, Python calls that __eq__ method when it executes an expression of the form x == y. In other words, Python converts x == y to x.__eq__(y).

Read the documentation string for the __eq__ method and implement it.

Interactively test your __eq__ method.

Exercise 8 (Challenge)

Locate the Python 2.7 *Language Reference* at http://www.python.org. What are the names of the methods that must be implemented in class Fraction so that Fraction objects support the binary -, * and / operators? What are the names of the methods that must be implemented so that Fraction objects support the binary >, >=, < and <= operators? What method implements the unary - (negation) operator? What method would Python call if you passed a Fraction object to the built-in abs function? What method implements the ** operator?

Design, implement and test these methods.

Posted: Tuesday, November 27, 2012.