CS 150 Programming Assignment 5

Due Tuesday 10/11/11 @ Midnight

There are 5 questions, worth a total of 50 points.

All of your files must have exactly the filename listed in the assignment. Each file should have a comment at the beginning with your name and the date. Classes should have precisely the name specifed in the assignment, should include a docstring with a description of their intended behavior, and should not contain any code that executes when the class is imported.

Buzz.py

10 points

Create a class which mimics the operation of a Buzz Lightyear toy doll. The Buzz toy has two buttons, one which causes it to shoot laser beams, and the other which causes it to say one of five phrases. Buzz says the same five phrases in order forever, starting over again with the first when finished. Also, after two of the phrases, Buzz shoots his lasers.

You need to implement the Buzz class and the methods: __init__(self), shoot(self), and speak(self). Here is a sample usage of the class:

```
>>> from buzz import *
>>> bl = Buzz()
>>> bl.shoot()
Pow pow pow pow pow!
>>> bl.speak()
I come in peace
>>> bl.speak()
This is an intergalactic emergency!
>>> bl.speak()
Buzz Lightyear to the resecue
Pow pow pow pow pow!
>>> bl.speak()
To infinity and beyond
Pow pow pow pow pow!
>>> bl.speak()
I am Buzz Lightyear
>>> bl.speak()
I come in peace
```

Pencil.py 10 points

Create a class which emulates a pencil. A pencil can write things, but gets dull, making the writing hard to read.

You should implement a Pencil class. The class has two methods, write, which takes a string argument and writes it, and sharpen which sharpens the pencil. The constructor should take an optional argument which specifies the hardness of the pencil (harder pencils have higher numbers and write longer before needing sharpening). When writing a letter, the pencil either writes the letter correctly or else writes '#'. The choice is made randomly, and is more likely to be '#' the duller the pencil gets. A pencil with hardness 2 should be completely dull (all #'s) after 200 characters have been written.

```
>>> from Pencil import *
>>> p = Pencil()
>>> p.write('It is a far, far better thing that I do, than I have ever done; it is a far, far better rest that I go to than I have ever known.')
It is a f#r, #ar bette# th#ng tha# I do, t#a# I##ave #ver #on#; i# is a ##r, f## #ett#r#re#t#that#I#go #####an#I ##v###ver kno###
>>> p.write('All animals are equal but some animals are more equal than others.')
##l ###m##############but##########a####################a####
>>> p.sharpen()
>>> p.write('There go my people: I have to go and run and catch up because I am their leader.')
Th#re go my people: I#have to go an# r#n #n###atch ## bec#use I##m the#r#leader#
>>> p2 = Pencil(4)
>>> p2.write('Vanity was stronger than love at sixteen and there was no room in her hot heart now for anything but hate.')
Vanity was stronger than love at s#xteen an##there#was no room i# he##ho# heart now #or anything#b#t hate.
>>> p2.write("I'm glad I did it, partly because it was worth it, but mostly because I shall never have to do it again")
#'m glad I#di# i#,#par##y#beca#se #t #as##orth#### ##t#mos##y#beca#s######all##ev#r
it#ag###
```

Fraction.py 10 points

Use the Fraction class defined in the book, and add negation and inversion to it, as described in Exercises 6.10 and 6.11. You can get a copy of the Fraction.py file from the book's webpage (along with all examples from the book), or by copying it from the ~bryan/cs150/classes directory.

Account.py 10 points

Write a class Account that acts as a person's bank account. The class has a constructor, which (optionally) takes a starting balance and interest rate. Other methods are balance, which returns the account balance; deposit, which makes a deposit; withdraw, which makes a withdrawal of the specified amount if the balance is large enough; payInterest, which adds interest to the account at the current rate; and setInterestRate which sets the account's interest rate.

Here is an example usage:

```
>>> from Account import *
>>> a = Account(90)
>>> a.deposit(10)
>>> a.balance()
100
>>> a.setInterestRate(0.03)
>>> a.payInterest()
>>> a.balance()
103.0
>>> a.withdraw(75)
>>> a.balance()
28.0
>>> a.withdraw(40)
Traceback ...
ValueError: Balance too low
```

Blood.py 10 points

People have two blood alleles, each of which can be 'A', 'B', or 'O'. From the alleles, the person has a blood type, which is A, B, or O. Type A corresponds to alleles A and A, or A and O. Type B corresponds to alleles BB or BO. Type O corresponds to alleles OO, and type AB corresponds to alleles AB. When a child is born, the child gets one allele from the father and one allele from the mother, and these determine the child's blood type.

Write a class Blood that models a person's blood. The constructor for the class should take two strings as arguments, which may be 'A', 'B', or 'O'. There should be a type() method that returns a string ('A', 'B', 'AB', or 'O') representing the person's blood type. There should also be a method childwith which takes a second Blood object as an argument (the person's spouse) and creates a new Blood object for the child, where the child inherits a single allele from each parent, chosen ramdomly.

Here is a sample:

```
>>> from Blood import *
>>> me = Blood('A','B')
>>> her = Blood('B','O')
>>> me.type()
'AB'
>>> her.type()
'B'
>>> c = me.childwith(her)
>>> c.type()
'AB'
>>> c = me.childwith(her)
>>> c.type()
'B'
>>> c = me.childwith(her)
>>> c.type()
'B'
```