BRONX COMMUNITY COLLEGE
of the City University of New York
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

CSI32 Section E01 **Project 2**
Spring 2018 **March 5, 2018**
Due: Monday, March 19, 2018

MODIFYING THE BINARY INTEGER CLASS

Signed Integers

This project is to modify the class `BinInteger`. The original version of the class used eight bits (values of zero or one) to represent non-negative integers whose values ranged between 0 and 255 according to **binary** (base-two) notation. An algorithm for adding two non-negative integers has been included with the original version, developed in class. In this project, integers can be negative. The range of values represented by eight bits will be between -128 and 127.

Representing Signed Integers

Positive numbers have a most significant—highest—bit of zero, negative numbers are represented by lists of bits whose highest digit is 1. The negation of a positive integer is represented by a sequence of bits found by taking its original representation as a binary number, then inverting each bit—this is called the **one's complement** of the absolute value—and finally, adding 1 to the resulting binary number, (this is called the **two's complement** of the original representation).

Methods

`__init__`

This constructor takes a Python integer as input, but it should raise a ValueError exception if that integer is out of range (over 127 or less than -128). If the integer is non-negative the normal base-two list of zeros and ones is used as the representation. If the integer is negative, the two's complement of the absolute value is the representation.

`__add__(self, other)`

This is implemented using the usual addition digit-by-digit in columns. A helping function, `addBinary(bitList1, bitList2)`, should be called to do this to add the representatives of each binary integer object.

`__neg__(self)`

This creates an object whose representative is the two's complement of self's representative.

`__sub__(self, other)`

Implemented by reusing `__neg__(self)` and `__add__(self, other)`. (Just return the sum of `self` and the opposite of `other`.)

`decimal(self)`

This will take the base-2 attribute of a `BinInteger` object and use it to return a python integer, which can then be displayed by python in usual base-10 format.

`__str__(self)`

This is used to print an object's representation: the string of 0's and 1's which represents the integer.

Helping Functions

These are not class methods, but simply functions which take Python "'bit lists"' as parameters. These lists should have size 8, and contain integer values which must be 0 or 1. These functions will be provided, pre-coded, to allow you to implement the actual methods of the BinInteger class by performing these operations on the representatives

`addBinary(bitList1, bitList2)`

Returns a new list of 8 bits by adding two bit lists according to base-two arithmetic.

`onesComplement(bitList)`

Returns a new list of bits which has the bits of an existing bit list exactly reversed.

`makeBinary(n)`

This returns a new bit list which is the base-two representation of the integer `n`.

`makeDecimal(b)`

This returns a normal Python integer which is represented by the bit list b.

Unit Testing

You can write a unit test, added at the end of the file `BinInteger.py`, which will call the methods you will be writing for this project, using the overloaded arithmetic operator binary symbols `+` and `-`, the unary binary symbol `-`, and calling all other defined methods.

Alternatively, you can use IDLE to import (or run) the file `BinInteger.py`, then enter expressions using the BinInteger class to test all methods and oveloaded symbols with various test values.

In either case, copy and paste your interactive testing sessions to the end of the file `BinInteger.py` as a mult-line comment.