Python's Magic Methods: Two's Complement Binary Numbers

ISTA 350 Hw4, Due 3/5/2020 at 11:59 pm

Introduction. This homework is intended to acquaint you with Python's operator overloading functionality (magic methods) and the basics of binary integer arithmetic as it is done inside in computers. You have already been using operator overloading, possibly without knowing what it's called. For example, the plus sign is commonly and in Python used to add numbers. But it is also used for string concatenation. That is an example of operator overloading. **Special note: many of the test methods will not work until your eq method is correct.**

Instructions. Create a module named `hw4.py`. Below is the spec for nine methods. Implement them and upload your module to the D2L Assignments folder.

Testing. Download `hw4_test.py` and put it in the same folder as your `hw4.py` module. Run it from the command line to see your current correctness score. Each of the 10 methods s worth 10% of your correctness score. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program will be different and may uncover failings in your work not evident upon testing with the provided file. Add any necessary tests to make sure your code works in all cases.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, `ISTA 350 Hw4`, and a brief summary of the module. Each method/function must contain a docstring. Each docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code (not necessary on this assignment).

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

Resources.
https://docs.python.org/3/tutorial/index.html
https://docs.python.org/3/tutorial/classes.html
https://docs.scipy.org/doc/numpy/reference/
https://scipy.github.io/old-wiki/pages/Numpy_Example_List

Tkinter:
http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm
http://www.tutorialspoint.com/python/python_gui_programming.htm
https://wiki.python.org/moin/TkInter

`class Binary:` If you use any method you have defined in another method definition, use the overloaded operator, if there is one. For instance, you will lose points if we see anything like `bin1.__add__(bin2)`. This should be `bin1 + bin2`. Decorate your class with the `functools.total_ordering` decorator.

`init`: `init` has one string parameter with a default argument of `'0'`. This string can be the empty string (treat the same as `'0'`). Otherwise, it should consist only of `0`'s and `1`'s and should be 16 or less characters long. If the argument does not meet these requirements, raise a `RuntimeError`. Each `Binary` object has one instance variable, an integer array of length 16 called `bit_array`. `bit_array` has integers `0` or `1` in the same order as the corresponding characters in the argument. If the string is less than 16 characters long, `bit_array` should be padded on the left with the leftmost digit in the string (`int`ified, of course). **You may not use lists in this method.**

`eq`: Takes a `Binary` object as an argument. Return `True` if `self == ` the argument, `False` otherwise. **You may not use `int` in this method. Do this one after init – many of the tests won't work until this one passes.**

`repr`: Return a 16-character string representing the fixed-width binary number, such as: `'0000000000000000'`.

`add`: Takes a `Binary` object as an argument. Return a new `Binary` instance that represents the sum of `self` and the argument. If the sum requires more than 16 digits, raise a `RuntimeError`. **You may not use `int` in this method.**

`neg`: Return a new `Binary` instance that equals `-self`. **You may not use `int` in this method.**

`sub`: Takes a `Binary` object as an argument. Return a new `Binary` instance that represents `self` – the argument. **You may not use `int` in this method.**

`int`: Return the decimal value of the `Binary` object. This method should never raise a `RuntimeError` due to overflow. **Do not use `int` on `Binary` objects in any other method.** You may use it to convert strings to integers, but not on `Binary` objects. You will probably need to use the `item` method on the integer that you calculate to extract the regular Python `int` from the `np.int32` instance that you have created.

`lt`: Takes a `Binary` object as an argument. Return `True` if `self < ` the argument, `False` otherwise. This method should never raise a `RuntimeError` due to overflow. **You may not use `int` in this method.**

`abs`: Return a new instance that is the absolute value of `self`. **You may not use `int` in this method.**