Computer Laboratory 2
CSci 1913: Introduction to Algorithms,
Data Structures, and Program Development
January 31/February 1, 2017

0. Introduction.

In this laboratory assignment, you will write a Python class called `Zillion`. The class `Zillion` implements a decimal counter that allows numbers with an effectively infinite number of digits. Of course the number of digits isn't really infinite, since it is bounded by the amount of memory in your computer, but it can be very large.

1. Examples.

Here are some examples of how your class `Zillion` must work. I'll first create an instance of `Zillion`. The string gives the initial value of the counter. Blanks and commas in the string are ignored.

```
z = Zillion('999 999 999 998')
```

This instance of `Zillion` contains the number nine hundred ninety nine billion, nine hundred ninety nine million, nine hundred and ninety nine thousand, nine hundred and ninety eight. This is much larger than the maximum number that can be represented in a C++ or Java `int` variable, which is only 2 147 483 647, or around two billion.

I'll add 1 to the counter, by calling the method `increment`, and I'll print the counter by calling the method `toString`. I should see `999999999999` (twelve nines) printed.

```
z.increment()
print(z.toString())
```

I'll add 1 to the counter again, and print its digits as before. I should see `1000000000000` (one with twelve zeroes) printed.

```
z.increment()
print z.toString()
```

Finally, I'll test if the counter contains zero by calling the method `isZero`. Of course `z.isZero()` will return `False`. But `Zillion('0').isZero()` will return `True`.

2. Theory.

Your class `Zillion` must represent a number internally as a list of one or more digits. Each digit d in the list is an integer $0 \le d \le 9$. For example, the number `1234` must be represented as the list `[1, 2, 3, 4]`. Although Python provides long integers that can have arbitrarily many digits, the class `Zillion` must not use long integers. *You will receive zero points for this assignment if you use Python's long integers in any way!*

The method `increment` must work like this. Starting at the right end of the list, and moving toward the left end, it must change `9`'s into `0`'s, until it finds a digit that is not `9`, or until there are no more digits left to be visited. If it stops because it has found a digit that is not `9`, then it must add `1` to that digit. If it stops because there are no more digits left, then it must add `1` to the front of the list. For example, if the list is `[1, 2, 9]`, then `increment` will first change the `9` at the end of the list to a `0`, then add `1` to the digit `2`, resulting in `[1, 3, 0]`. Similarly, if the list is `[9, 9]`, then `increment` will change both `9`'s to `0`'s, then add `1` to the front of the list, resulting in `[1, 0, 0]`.

Hint: unlike the previous lab, you are allowed to change list elements now. In fact, you *must* change list elements, or your program will not work correctly.

3. Implementation.

The class `Zillion` must define the following methods. To simplify grading, your methods must use the same names as the ones shown here. However, they need not use the same parameter names, except for `self`, which must be unchanged. To demonstrate your understanding of Python, some methods must be implemented in specific ways, as described below.

`__init__(self, digits)`

> The string `digits` must be a string containing nothing but digits (`0` through `9`), blanks, and commas. It must contain at least one digit. If `digits` contains no digits, or if it contains a character that is not a digit, a blank, or a comma, then raise a `RuntimeError` exception. Convert `digits` to a list of integer digits as described in section 2, and save it within the instance of `Zillion`. The list represents the number that is stored in the counter.
>
> Hints: it is not enough to test if `digits` is the empty string. For example, a string consisting of nothing but blanks and commas, like `' , ,,'` must raise an exception. Also, you may wish to call the built-in function `int`, which converts a string to an integer. For example, `int('0')` returns the integer `0`.

`increment(self)`

> Increment the counter, using the algorithm described in part 2. Hint: one way to do this is by using a `while` loop, and another way is to use recursion. There may be other ways.

`isZero(self)`

> Test if the counter contains zero. That is, test if your list of digits contains nothing but `0`'s.

`toString(self)`

> Convert the list of digits to a string, and return the string. Hint: you may wish to call the built-in function `str`, which converts an integer to a string. For example, `str(0)` returns the string `'0'`.

4. Tests.

The file `tests.py` on Moodle contains a series of tests. The tests create instances of the class `Zillion`, call their methods, and print what they return. Some of the tests raise exceptions instead of printing values. They are designed to show whether `Zillion`'s methods raise exceptions correctly.

To grade your work, the TA's will run the tests using your functions. If a test behaves exactly as it should, then you will receive all the points for that test. If a test does anything else, then you will receive no points for that test. Your score for this assignment is the sum of the points you receive for all the tests.

4. Deliverables.

Run the tests in the file `tests.py`. Then submit the Python code for your class `Zillion` and the results of the tests. Your lab TA will tell you how and where to turn them in. If your lab is on Tuesday, January 31, then your work must be submitted by Tuesday, February 7 at 11:55 PM. If your lab is on Wednesday, February 1, then your work must be submitted by Wednesday, February 8 at 11:55 PM. *To avoid late penalties, do not confuse these two dates!*