ECE 122: Introduction to Programming for ECE- Spring 2020

Project 3: My Big Number Calculator

Due Date: **Deadline: see website, class policy and moodle for submission**
This is an individual project (discussions are encouraged but no sharing of code)

Description

In few applications in numerical approximations, or in cryptography with the computation of large prime numbers, it is essential to handle numbers with 'infinite' precision. In almost all programming languages, the standard native types such as int and float are limited in size. For example, an int variable in Java, C, (etc.) has a maximum absolute value of 2,147,483,647=$2^{31}-1$ (largest known prime until 1855), and primes generated for cryptography purposes can easily be several hundred digits. The situation is similar for floating point numbers (stored in double precision with 'only' 16 digits digital accuracy). To overcome these limitations, a common approach is to implement a custom number class, that supports addition, subtraction, multiplication and division of big numbers. These big numbers are commonly stored using lists (where each item of the list stores a particular digit). For example the following integer:

```
i=123456789123456789123456789
```

can be written as:

```
ilist=[1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9]
```

In fact, Python already provides an arbitrary precision arithmetic for int. The list implementation of the integer is implicit, as soon as the number becomes too big (greater than $2^{31}-1$), python switches automatically its internal representation to a list (without you, seeing it). So the number can expand indefinitely. For example, you could easily compute `i**2` with the number above and python will return a bigger number (this is not the case with most other languages). Python does not provide this feature with floating numbers where the number of representative decimal digits stays stuck at 16. To simplify a bit this project, however, we are not going to consider floating big numbers but integer big numbers. It means that we are going to develop our own big number class for integer regardless of the fact that it is redundant for Python.

The project must include two files:

1. `big_calculator.py` - This is the main code which is provided to you. It is then **not allowed** to modify it.
2. `BigNumber.py` - This is a module that contains the class BigNumber to represent the big numbers, and few instance methods that operate on big numbers. This is the file that you must change, complete and submit. You are not allowed to change the names of the methods, but you can add any additional internal methods if needed.

Using this new BigNumber class, you will be able to consider arbitrary size addition, subtraction, and multiplication. The class should easily be able to handle numbers that are hundreds of digits long. In order to implement the required mathematical operations, we will obviously need to traverse/scan and manipulate the list associated with an arbitrary size number. The project is then testing your algorithmic skill (basic mathematical algorithms). As a reminder, one of the main goal of this class is to learn how to use programming as a tool for solving Engineering/Scientific/Mathematical problems (and not only programming for programming). As always, try to design solutions using paper and pencil first and work on a 'code skeleton' (a.k.a. pseudocode) before any attempts to program anything. When asking questions to TA, you must be able to show and discuss your pseudocode solution.

The project is designed to be incremental, you can then debug, test and run your code after each new task/option is implemented. However, they are few special (corner) cases to account for along the way. It is not always essential to complete them in order to move to the next tasks, but they would need to be completed for full credit.

Submission/Grading Proposal

Since the main program must stay unchanged, you will only need to submit your completed `BigNumber.py` file on Moodle. This project will be graded out of 100 points:

1. Your program should implement all basic functionality/Tasks and run correctly (100 points).
2. Overall programming style: program should have proper identification, and comments. (-5 points if not)

How to Start

Although the file `BigNumber.py` is incomplete, the program is initially functional (like project1). It means that you should be able to run the main program `big_calculator.py` without expecting a crash. You will then get the following:

```
Welcome to BigNumber Calculator
================================

Choose Operation (none,==,+,-,scale,*) or q for quit:
```

This program asks the user to choose an operation. Depending on the operation selected, it will ask the user to enter one or two numbers, and it will then output the result. The user can select 'q' to quit. We are going to detail all the tasks associated with the various operations that must be performed incrementally.

Note that to simplify the project, we will consider only **positive number** as inputs.

Task-1 Operation `none` [20pts]

If the user enters **none**, you should get:

```
Choose Operation (none,==,+,-,scale,*) or q for quit: none
Enter first number: 123456
123,456
```

This option is just here to test your big number representation. The code will read a string, create an object of type BigNumber (which is mainly a list including all digits), and print this big number.

We note that the input String number may contain zeros on the left, random comas or random spaces. when creating the list, you will have to somehow filtered the string. Here more conversion examples:

String input	BigNumber output
12	12
1234	1,234
12345678912	12,345,678,912
1 23	123
12, 344 5,5 5	12,344,555
00004321	4,321

How to proceed?

1. you need to complete the `__init__` constructor method and initialize the `list_big` attribute, where each item will contain a single digit. For example for the number 58912 the digit 5 should be stored at index 0 of the list, 8 at index 1, and so on.

2. you need to complete the `__str__` method that returns a string. As you can see in the examples, this string will be formatted to include comma at the right places. Hint: you could scan the list from right to left, keep track of the number of digits, and check the remainder while dividing by 3.

Task-2 Operation == [15pts]

When this option selected, the program should ask for 2 big numbers, print them and compare them. It will also return some info. Here are some examples:

```
Choose Operation (none,==,+,-,scale,*) or q for quit: ==
Enter first number: 3223325532
3,223,325,532
Enter second number: 55532522
55,532,522
first number > second number
```

```
Choose Operation (none,==,+,-,scale,*) or q for quit: ==
Enter first number: 343432
343,432
Enter second number: 52525235235
52,525,235,235
first number < second number

Choose Operation (none,==,+,-,scale,*) or q for quit: ==
Enter first number: 123456
123,456
Enter second number: 123456
123,456
first number == second number
```

How to proceed?

1. You need to complete the method `compare`. Hint: since both input numbers are considered positive, you could compare the length of the lists to get the smaller or greater flag correct. If the lists are the same size, you need to scan from left to right and compare digit by digit.

Task-3 Operation + [25pts]

As you may have guessed, we need to perform the addition of two big numbers. Here five tests examples by increasing level of complexity:

```
Choose Operation (none,==,+,-,scale,*) or q for quit: +
Enter first number: 123456
123,456
Enter second number: 654321
654,321
Result: 777,777

Choose Operation (none,==,+,-,scale,*) or q for quit: +
Enter first number: 12345678
12,345,678
Enter second number: 111
111
Result: 12,345,789

Choose Operation (none,==,+,-,scale,*) or q for quit: +
Enter first number: 111
111
Enter second number: 12345678
12,345,678
Result: 12,345,789

Choose Operation (none,==,+,-,scale,*) or q for quit: +
Enter first number: 345666
345,666
Enter second number: 4299
4,299
Result: 349,965
```

```
Choose Operation (none,==,+,-,scale,*) or q for quit: +
Enter first number: 123456789123456789
123,456,789,123,456,789
Enter second number: 987654321987654321
987,654,321,987,654,321
Result: 1,111,111,111,111,111,110
```

The first test uses two numbers of the same size with 'no carry' involved. The second and third tests use numbers of different sizes (still 'no carry'). The fourth test has 'a carry' that is involved it means that if you start the addition from the right 6+9=15, you get 5 and carry the one to the next digit on the left, and so on. In the fifth test, the 2 numbers have the same size but the output is larger in size by one (because of the 'carry').

How to proceed?

1. You need to complete the method `add`. Make sure you get the algorithm correct on paper before programming. You can scan the 2 lists from right to left, add the digits and take care of the 'carry'. Hint: 15%10 will give you 5 (we are working in base 10) and 15//10 will give you 1 (the carry).
2. Implement your method step by step following the tests example above. There will be partial credit for each successful tests.
3. Hint: As a reminder the method `insert` can be used with a list to insert an item at a particular position. It is useful to always insert at index 0 (insert at the beginning of the list). As an example if you do `list.insert(0,1)` follows by `list.insert(0,2)`, list will contain `[2,1]`.

Task-4 Operation - [20pts]

You need to perform the subtraction of two big numbers. We will consider that number 1 is greater than number 2. Here are three test examples by increasing level of difficulty.

```
Choose Operation (none,==,+,-,scale,*) or q for quit: -
Enter first number: 123456
123,456
Enter second number: 111
111
Result: 123,345

Choose Operation (none,==,+,-,scale,*) or q for quit: -
Enter first number: 123456
123,456
Enter second number: 888
888
Result: 122,568

Choose Operation (none,==,+,-,scale,*) or q for quit: -
```

```
Enter first number: 123456
123,456
Enter second number: 123111
123,111
Result: 345
```

The first test does not have a 'carry', the second test has a 'carry', and in the third test the output is smaller in size than big number 1.

How to proceed?:

1. You need to complete the method `subtract`. Make sure you get the algorithm correct on paper before programming. You can scan the 2 lists from right to left, subtract the digits and take care of the 'carry'. Hint: let us compute (36-17) by steps: (i) perform (6-7)=-1; (ii) since negative add 10 (-1+10)=9 (9 will be your first utmost right digit) and carry 1 to the next digit; (iii) perform (3-(1+carry))=1 (1 will be your next digit on the left); (iv) result is 19.

2. Implement your method step by step following the example tests above. There will be partial credit for each successful tests.

3. You also need to consider the subtraction between two numbers where the first number is smaller. The result will then be negative. Hints: At the beginning of the subtract method, the 'compare' method could come handy here, if the big1 number is smaller than big2, just call the method `subtract` 'in reverse', and return. You may want add an attribute `is_positive` set to `True` by default in the constructor to keep track of the sign. You need to modify `__str__` accordingly. Example of execution:

```
Choose Operation (none,==,+,-,scale,*) or q for quit: -
Enter first number: 12345
12,345
Enter second number: 35235262462
35,235,262,462
Result: -35,235,250,117
```

Task-5 Operation `scale` [10pts]

The program is expecting one big number and one factor (number form 0 to 9). It will perform the multiplication, return and print the new big number. Some examples:

```
Choose Operation (none,==,+,-,scale,*) or q for quit: scale
Enter first number: 123
123
Enter the scaling factor: 9
Result: 1,107
```

```
Choose Operation (none,==,+,-,scale,*) or q for quit: scale
Enter first number: 719170080737319175
719,170,080,737,319,175
Enter the scaling factor: 3
Result: 2,157,510,242,211,957,525

Choose Operation (none,==,+,-,scale,*) or q for quit: scale
Enter first number: 97979
97,979
Enter the scaling factor: 0
Result: 0
```

How to proceed?:

1. You need to complete the method `scale`. I let you figure this one out.

Task-6 Operation $*$ [10pts]

Here it is:

```
Choose Operation (none,==,+,-,scale,*) or q for quit: *
Enter first number: 121234
121,234
Enter second number: 1212
1,212
Result: 146,935,608

Choose Operation (none,==,+,-,scale,*) or q for quit: *
Enter first number: 1414194714017401720747401740
1,414,194,714,017,401,720,747,401,740
Enter second number: 123456789123456789
123,456,789,123,456,789
Result: 174,591,938,587,953,644,979,910,605,479,961,758,913,412,860
```

How to proceed?

1. You need to complete the method `multiply`. You can use the method `scale` and `add` to perform successive additions.

Task-7- Bonus [5pts]

Your program should work with negative numbers. You can handle the general cases in methods `add`, `subtract` and `multiply` taking advantage of this table:

sign of $b1$	+	+	-	-
sign of $b2$	+	-	+	-
$b1 - b2$	$\|b1\| - \|b2\|$	$\|b1\| + \|b2\|$	$-(\|b1\| + \|b2\|)$	$\|b2\| - \|b1\|$
$b1 + b2$	$\|b1\| + \|b2\|$	$\|b1\| - \|b2\|$	$\|b2\| - \|b1\|$	$-(\|b2\| + \|b1\|)$
$b1 * b2$	$\|b1\| * \|b2\|$	$-\|b1\| * \|b2\|$	$-\|b1\| * \|b2\|$	$\|b1\| * \|b2\|$