Advanced Python Programming

Assignment 6

'Computer Algebra Systems'

In this episode we are going to introduce a whole new world that exists almost apart from our common realities (as being, believe it or not, software engineers or "programmers"). Most of the times we just use a standard toolkit arming with imperative (and sometimes functional) programming languages that have no idea what is algebra mean even on the basic level. Such a sad story considering the fact that Computer Science back in time was founded by the needs of mathematicians. Who, needless to say, are fathers of nearly all of its downstream branches. The needs were (and are) to solve equations, balance them, factorize, put in standard form and 100+ other possible operations. For example, can we – in our beloved Python – write the following `expr = x + 2*y` (where `x` and `y` are undefined) and not to get an error? Definitely not. More than that, in mathematical sense, we don't really want to evaluate it *immediately* rather than just treat it as *algebraic* expression that *can be* evaluated later on (when the context will be fulfilled). Doing so is called *symbolic computation*. It means our compiler concerns not only with the values the symbols represent but rather with symbols themselves, finding patterns, reducing and *deferring* their evaluation (do not confuse with *lazy evaluation* which is a different beast). Symbolic computation is an inextricable part or even the *core* of the more general notion – Computer Algebra System (CAS). Probably where the most notable exemplars are Wolfram Mathematica and MATLAB.

So from now on our goal is to implement that core. The core that sheds a light on what the algebraic systems *are* in their essence (or could be, at least). But before we start let us refresh our formal arrangements:

- No `regexp` as usual
- Do not mess with `tokenizer` or `ast` modules (they are for *Python* source code which is not the case now).
- Only two arithmetic operations are allowed in your source code: plus and minus (nothing more)
- Undocumented code → degradation of points
- The assignment is due this Thursday 3:30pm (just before the lecture)

1. Just to warm up, let us start from the main concepts of every CAS. Give *clear and intuitive* explanation for each of the following (answers should not be too concise):
   (a) What is algebraic expression? How does it differs from mathematical formula?
   (b) What is term rewriting? Is it the same as symbolic computation?
   (c) What is symbolic computation comparing with numerical computation?
   (d) What is the difference between evaluation and interpretation (in math sense)?
   (e) What is lazy evaluation comparing with eager evaluation?
   (f) How functional programming related to algebra systems?

(g) Is lambda function in Python a *replacement* for symbolic computation? If yes, why? If not, why not? Provide *at least* two examples.

Give your answers as a neat `answers.pdf` file.

2. Implement an interactive shell (under CAS realm they may be also called *notebook*) with two basic operations (addition, subtraction) and brackets support:

```
$ ls ./
cas.py answers.pdf

$ python cas.py -h
(please describe how to use your program if there is something specific)

$ python cas.py
>>>                    (just empty line waiting for an expression)
>>> 2 + 2
0:  4                  (results are numbered and can be used later on)
>>> 6 + [0]
1:  10
>>> -1 - (2 + 4)       (support for nested brackets is not expected!)
2:  -7
>>> -1 - 2 + 4
3:  1
>>> 1 + -1
4:  0
>>> 2+2
err: invalid expression
```

3. The next task is to implement simple symbolic computations:

```
$ python cas.py
>>> x + x              (if easier, restrict unbound vars to 1 char)
0:  2x
>>> x * x
1:  x^2
>>> [0] - 1
2:  2x - 1
>>> x + x * 3
3:  4x
>>> [3] * [2]          ([2] is not monomial, embrace with () immediately)
4:  4x * (2x - 1)      (reduction is possible but brackets prevents that)
>>> expand [4]         (keyword that tries to do reduction despite () )
5:  8x^2 - 4x
>>> 2y
6:  2y                 (just does nothing)
```

(note: we haven't yet implemented multiplication on this stage! This is just substitution of operator symbols when several ones occur in a sequence)

4. Your task is to proof that once we've got 2 basic operations as of + and −, the only limit is space & time. Implement division and multiplication by means of addition and subtraction on integers. The trick is that the number can be bigger than any modern CPU can accept (under their registers of course).

```
$ python cas.py
>>> 4294967295 + 4294967295 * 4294967295
0:  18446744069414584320
>>> -4294967295 - 4294967295
1:  -8589934590
>>> [0] / [1]
2:  -2147483648
>>> 7 / 2
3:  3
>>> 7/2
err: invalid expression
```

(hint: recall your school level arithmetics using paper, pen and columns. Then do *exactly* the same thing but substitute paper by strings)

5. Implement fractions and add support for floating-point numbers.

```
$ python cas.py
>>> 1/3 + 1/7
0:  10/21
>>> 1/3 / 1/7
1:  7/3
>>> expand [1]
2:  2.33333333          (cut all decimal numbers to 8 places after dp)
>>> 9 / 4
3:  2.25
>>> 9/4
4:  9/4
```

That's certainly enough! Before the upload, please create `results.txt` where you put all of your terminal output once you evaluate the following expressions:

```
>>> 1/5 + 1/12
>>> 1/9 / 1/3           (expected 3/9, not 1/3)
>>> expand [last]       (now we can expect that)
>>> 9 / 4
>>> 5294967195 * 4294967299
>>> z + z - z * z
>>> 2x + x + y - x
>>> (x - y) * (y + x)
>>> (x - y) * (x + y)
>>> 2u + 3 + u * 2 - 1
```

```
>>> [last] / (1 + 1)   (last means a result from previous calculation,
>>> expand [last]       you can implement it, or just put an appr.number)
```

Again, *your* final behavior may slightly differ from examples provided above. That's ok! We will see how you have understood the assignment.

Division only by subtraction

Here is little hint how to do division in terms of subtraction.
(the example was given during the lab and can be useful for those who were absent)

```
[1] 9 / 4                   (original expression)
res = ""                    (resulting "string number")
[2] 9 - 4 - 4               (count how many 4s "fits" into 9)
        .   .
res += "2"                  (two 4s fits into 9)
[3] 1                       (the reminder from [2])
res += "." if reminder != 0 else ""
[4] 10 - 4 - 4              (multiply reminder by 10 and repeat [2])
         .   .
res += "2"                  (two 4s fits into 10)
[5] 2                       (the reminder from [4])
[4] 20 - 4 - 4 - 4 - 4 - 4 (multiply reminder by 10 and repeat [2])
         .   .   .   .   .
res += "5"                  (five 4s fits into 20)
[5] 0                       (no reminder, so we've done!)
---
res == "2.25"

What if there is no end in case of 10 / 3?
Limit your loop once you start multiplying reminder by 10 to 8 cycles.
You get the precision of your decimal to 8 characters after decimal point.
```

Resources

1. https://en.wikipedia.org/wiki/Computer_algebra
2. https://en.wikipedia.org/wiki/Computer_algebra_system
3. https://stackoverflow.com/questions/16395704/what-is-symbolic-computation
4. Watt, Stephen. (2019). What happened to languages for symbolic mathematical computation? (only for broadening the horizon)