The **focus** of this homework assignment is implementing an *interpreter* for a tiny toy language, which we shall call *PMPV*, *from scratch*. From scratch means that the interpreter, which we shall call ***pmpvi***, must do all the core work explicitly itself, in the written and submitted code, without using features such as *eval*, for instance. *If this critical requirement is unclear, please ask for clarification.*

The **PMPV language**, named for Plus-Minus-Parentheses-and-Variables, is in effect a very simple calculator that supports integer expressions that use only the plus and minus operators, along with parentheses; it also allows the results of such expressions to be stored to and retrieved from variables. The sample below illustrates the key points. The class discussion forum should be used for further details or clarifications on the PMPV language.

Sample input to `pmpvi` (that is, a sample *PMPV program*) appears in the left column below. The corresponding **sample output** (that is, the result of interpreting, running, executing, or evaluating that program) appears in the right column. Lines of the input and output (including empty lines) are each terminated with a single newline character. The third column is not part of the input or output but is only for explanatory purposeses here.

Input	Output	Comment
`3 + 5 - -2 - 2`	`8`	a simple expression and its result
`x = 3 + 5 - -2 - 2`		result is stored instead of printed
`y = x - (x - 2)`		use of variables in expression
`y`	`2`	simple variable lookup
`ans = (17 - (5 - 20)) - (1 - 11)`		more complex parentheses use
`ans`	`42`	

Input-output: The `pmpvi` program should read its input from the *standard input* stream and write its output to the *standard output* stream. Optional diagnostics may be written to the *standard error* stream. It is very important that the program read its input only from the standard in put stream and that it write nothing except the specified output to the standard output stream. In particular, there should be no prompts or informational messages printed to standard output. In the sample above, the first column corresponds to standard input and the second corresponds to standard output. The interpreter should produce each line of output immediately after reading the corresponding line of input.

The **submission** consists of an single electronic package that contains the **source** code for your implementation of the `pmpvi` interpreter, following the submission procedure described in class and on the class discussion forum. *Using the* ***discussion forum*** *to clarify details of both the main program and the submission format and procedures is an important part of this homework.* Use the *gzipped tar* (strongly preferred) or *zip* formats to package your submission. Name the electronic submission using the template

`cos301-hw01-`*`lastname`*`-`*`firstname`*`-`*`pqrs`*`.tgz`

where *`lastname`* and *`firstname`* are replaced by the obvious and *`pqrs`* is replaced by a

4-digit string of your choosing. (Replace `.tgz` with `.zip` if you use zip instead of tar for packaging.) The submission should be designed so that the command

`tar zxf cos301-hw01-`*`lastname-firstname-pqrs`*`.tgz`

results in the creation of a directory `cos301-hw01-`*`lastname-firstname-pqrs`*. In that directory should be all the source code (organized in further sub-directories as needed) as well as a README file with the usual semantics. *Do not submit any kind of non-source files* (results of compilation, etc.).

You are welcome to use any inanimate **resources** (e.g., books, Web sites, publicly available code) to help you with your work. However, *all such help must be clearly noted* in your submissions. Further, no matter what you use, *you must be able to explain, in detail, how it works*. (You may be called upon to explain your homework individually.) Refer to the class policy for details, and ask for clarifications if you are unsure if something is allowed.

Your implementation must use **clean, portable Python 3** that minimizes dependencies on OS, version (beyond 3.x), etc. (If in doubt, please ask.) Packaging and documentation of code are worth a very significant portion of the grade.