

You should **submit** an single electronic package that contains the **source** files for your work on the programming questions, along with a **README** file and sample inputs and outputs, following a procedure similar to that used for the previous homework. The rules for using outside resources are also similar to those in that homework.

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. All submitted code must be formatted using the *Black*¹ formatter. This homework centers on **extending the calc.py** example (which is included with PLY and has been discussed extensively in class) as follows. (Reminder: The program should read its input from the standard input stream and write the specified output to the standard output stream.)

1. It should not print any prompts; rather, the only outputs are the results of evaluation of the input. (However, prompts, diagnostics, and similar messages may optionally be printed to the standard error stream.)
2. It should support real numbers, in addition to integers, represented in the conventional manner (e.g., 3.1415, 602.3) and in scientific notation (e.g., 6.022E24, 9.109E-31).
3. It should support *div* and *mod* operators with the usual semantics with operator tokens `//` and `%`, respectively.
4. It should allow integers (both literals and values of variables) to be converted to reals using the syntax `real(x)` (e.g., `real(42)` should yield a real value 42.0).
5. It should allow reals (both literals and values of variables) to be converted to integers using the syntax `floor(x)` (e.g., `floor(42.92)` should yield the integer 42, `-42.92` should yield the integer -43).
6. The arithmetic operators (+, *, etc.) must have operands of the same type. If they are used with mixed-type operands, an error should be flagged by printing a line containing the string `type error` to standard error (with optional additional details on the same line). The value 0 should be used instead of the value of the erroneous expression.

Input-output: The 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 input 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.

The **input** consists of the calculator language of `calc.py` as discussed in class, extended to support the above features. The **output** consists of (only) the values of stand-alone expressions (excluding expressions that are part of an assignment statement) in the input. The output for a statement must be produced as soon as the statement appears in the input stream (before waiting for or reading any further input that may appear). There is exactly one (i.e. a unique) output for any given input (but there may be multiple inputs resulting in

¹<https://github.com/psf/black>

the same output). If two outputs for the same input differ by even a single character/byte then at least one of them is incorrect. Sample inputs and outputs should be discussed on the class discussion forum.

Submission: The packaging and submission procedure is similar to that used for the previous homework, with `hw01` replaced by `hw02` in the file name. As before, use the class discussion forum (ask questions) for further details and clarifications.

Reminders: Be sure to follow the policies in the syllabus, especially items related to the use of generative AI and other resources. Also, readability and organization of code is very important. Ask for clarifications if needed.

Input Line	Corresp. Output Line	Comment
<code>2 + 3</code>	5	basic usage; same as PLY's <code>calc.py</code>
<code>2.2 + 3.3</code>	5.5	real numbers work as expected
<code>7 / 2</code>	3.5	real division
<code>7 // 2</code>	3	integer division (quotient)
<code>7 % 2</code>	1	mod (remainder)
<code>floor(3.141)</code>	3	simple use of floor
<code>floor(-3.141)</code>	-4	floor is different from truncate
<code>floor(7 / 2)</code>	3	floor's argument can be more complex
<code>real(42)</code>	42.0	simple use of real
<code>real(2 + 3 // 2)</code>	3.0	a bit more complex
<code>2 + floor(3.141)</code>	5	adding with strict types
<code>x = 2 + 3</code>		assignment of integer value (output: blank line)
<code>y = 4.2 + 5.3</code>		assignment of real value
<code>x</code>	5	can be printed as usual
<code>y</code>	9.5	ditto
<code>x + x</code>	10	integer addition from variable
<code>y + y</code>	19.0	similarly for real addition
<code>x + floor(y)</code>	14	
<code>real(x) + y</code>	14.5	
<code>7 + 11</code>	18	
<code>7 / 17</code>	0.4117647058823529	
<code>floor(1000.0 * (7 / 17))</code>	411	
<code>real(floor(1000.0 * (7 / 17))) / real(1000)</code>	0.411	
<code>floor = 10000</code>		variables named floor/real are OK
<code>floor + 42</code>	10042	and they work as usual
<code>floor(3.141)</code>	3	the function floor remains unchanged