

## CptS 111 — Homework #3

**Due Tuesday Jan. 31, 2012 midnight (24:00)**

In problems 3 through 5 we will use the `eval()` function. Recall that this function evaluates the string parameter passed to it as if the string had been entered directly into the interpreter. `eval()` is described in Sec. 3.3 of the textbook.

**1. Reading Assignment:** Read Chap. 5 of the textbook. **Seriously!** This is good for you.

**2.** There are a couple of important concepts you need to internalize right now. So, *repeat the following out loud three times*:

`return` doesn't print anything and `print()` doesn't return anything.

Also, *repeat the following out loud three times*:

Data enters a function via the parameters and exits via a `return` statement.

You also need to understand what these two sentences mean!

### 3. Programming Assignment A

As demonstrated in class, with `eval()` we can easily obtain multiple values from a single line of input. Knowing this, we want to reimplement the score-calculating program from the second lab. In Task 3 of the second lab you were asked to calculate an overall score using the formula

$$0.1(qz + lb) + 0.15(t_1 + t_2 + t_3 + hw) + 0.2t_4$$

where  $t_1$ ,  $t_2$ , and  $t_3$  are scores on the three midterms,  $t_4$  is the score on the final exam,  $qz$  is the quiz score,  $lb$  is the lab score, and  $hw$  is the homework score. Recall that there is also an attendance bonus of up to three points in this class. This bonus is added directly to the overall score.

Your program, which you should name `hw3a.py`, should prompt the user for the following input (each item below corresponds to one line of input):

- The three midterms scores.
- The quiz, lab, and homework scores.
- The final exam score.
- The attendance bonus.

All values are from 0 to 100 except the attendance bonus which is between 0 and 3.

The following demonstrates input and output when this program runs correctly (user input shown in bold).

```

1 Enter tests [t1, t2, t3]: 85, 91, 82
2 Enter quiz, lab, and homework [qz, lb, hw]: 93, 94, 95
3 Enter final exam: 87
4 Enter attendance bonus: 3
5 Final score without bonus: 89.05
6 Final score with bonus: 92.05

```

Your program should be organized in terms of functions. Actually, in this case, the program is simple enough that you may define a single function: `main()`. (But, you are free to use more functions if you want.) The last statement of your program should be a call to `main()`.

#### 4. Programming Assignment B

The following is a summary of what the next program does (you should name this `hw3b.py`):

- Prints a multi-line message that explains what the user can (and should) do. The specific text of this message is shown below. A *single print() statement should be used to generate this output*. (Thus, the argument of the `print()` statement must be a multi-line string.)
- Prompts for, and reads, an arithmetic expression. The expression can contain the variable  $x$ . This expression is read as a string (and stored to a variable).
- Prompts for, and reads, a value for the variable  $x$ .
- Evaluates the expression.
- Reports the result in accordance with the example shown below.

Here is an example of the input and output when this program runs correctly:

```

1 Enter an arithmetic expression in terms of the variable x.
2 In addition to x, your expression may contain numeric constants
3 and parentheses. The expression can use the following operators:
4
5 +, -  <=> addition, subtraction
6 /, *  <=> float division, multiplication
7 //    <=> integer division
8 %     <=> modulo (remainder)
9 **    <=> exponentiation
10
11 Enter expression: (x + 27) / x ** 2

```

```

12 Enter numeric value for x: -9.234
13
14 f(x) = (x + 27) / x ** 2
15 f(-9.234) = 0.208357873964

```

Here is another example:

```

1 Enter an arithmetic expression in terms of the variable x.
2 In addition to x, your expression may contain numeric constants
3 and parentheses. The expression can use the following operators:
4
5 +, -  <=> addition, subtraction
6 /, *  <=> float division, multiplication
7 //    <=> integer division
8 %     <=> modulo (remainder)
9 **    <=> exponentiation
10
11 Enter expression: (x + x ** 2 + x ** 3 + x ** 4) % (x + 7)
12 Enter numeric value for x: 13
13
14 f(x) = (x + x ** 2 + x ** 3 + x ** 4) % (x + 7)
15 f(13) = 0

```

Your program must generate the initial instructions as shown in lines 1 through 10 of these examples. The last two lines of output (lines 14 and 15) must also be consistent with these examples, i.e., the second to the last line of output writes “`f(x) =`” followed by the expression the user entered. The last line of output shows the function with the value of `x` inserted as the argument of the function and also shows the result of evaluating the expression for the given value of `x`. For generating the last line of output, don’t forget the `sep` optional parameter for `print()` (you should specify that there are no spaces between the output values).

Your program should again use a `main()` function and the last statement of the program should be a call to `main()`. (There is no need for other functions.)

## 5. Programming Assignment C

Later in the course we will discuss some of the formatting features that Python provides (i.e., features that allow us to format the output). But, for this problem, we want to write our own function to generate “cleaner” output of floats than we obtain simply by using the `print()` function.

This function, which should be called `fixed_precision()`, takes two parameters: a float and an integer. We want to display the value of the float parameter but only to as many digits beyond the decimal point as specified by the second parameter.

So, for example, if we simply write `print(1 / 7)`, the output is `0.142857142857`. However,

using this new function, if we write `fixed_precision(1 / 7, 3)`, the output is `0.142`. Note that the last digit of output is not rounded to the closest value. Instead the original value is simply truncated after the third digit.

There is a string-based way to solve this problem, but we don't want to use that here (since we haven't yet covered strings in sufficient depth). Instead, we will use a purely numeric approach.

Let us call the `fixed_precision()`'s first and second parameters `x` and `n`, respectively. Here is one way to implement this function.

1. Obtain (and store) the integer portion of `x`.
2. Store `x` minus its integer portion. This is the fractional part of `x`.
3. Multiply the fractional part by the value that will move `n` digits to the left of the decimal point. So, if `n` were 1, this value would be 10. If `n` were 2, this value would be 100. And so on. (Think exponentiation.)
4. Obtain the integer portion of this new number. These are actually the digits we want to display to the right of the decimal point.
5. Using a `print()` statement, print the integer obtained in step 1, then a dot, and then the integer obtained in step 4. Ensure there is no space between these values (using `print()`'s optional `sep` parameter).

In addition to the `fixed_precision()` function, your program (called `hw3c.py`), should have a `main()` function. This function should prompt the user for an arithmetic expression that uses only (literal) numeric values. This expression should evaluate to a `float`. A `print()` statement is then used to show the value to which this expression evaluates (this is merely for the sake of comparison with the subsequent output). The user is then prompted for the desired number of digits. Next a `print()` statement is used to write information about what is to follow (this output corresponds to the first part of line 4 in the examples shown below). This `print()` statement should use the optional `end` parameter to prevent the output from being terminated with a newline character. Finally, the `fixed_precision()` function should be called.

The last statement of your program should be a call to `main()`.

The follow demonstrates the correct behavior of this program where the user has entered the expression "1 / 7" and has requested 4 digits of precision.

```

1 Enter a numeric expression: 1 / 7
2 Using print() = 0.142857142857
3 Enter desired number of digits: 4
4 Using fixed_precision() = 0.1428

```

Here is another example, again using 4 digits of precision:

```
1 Enter a numeric expression: 1 / 10
2 Using print() = 0.1
3 Enter desired number of digits: 4
4 Using fixed_precision() = 0.1000
```

This final example shows the function will not “tack on” additional digits to a number that has a fractional part of zero:

```
1 Enter a numeric expression: 20 / 2
2 Using print() = 10.0
3 Enter desired number of digits: 4
4 Using fixed_precision() = 10.0
```

There are other scenarios in which trailing zeros will be discarded. This is considered correct behavior for the sake of this assignment.

### **Submitting Your Work**

Use a zip file to bundle together your programs. They should be called `hw3a.py`, `hw3b.py`, `hw3c.py`. Your submission should be in accordance with how you submitted your work in the first lab. Recall that you can find a link to the submission page at the course Web site.