CSC 110
Introduction to Computer Science

Homework Assignment 2
Input and Output with Functions

Due: Before Lecture 11

Note: Homework Assignment 2 should be completed individually.

For this homework you need to complete three tasks:

1) Make the `read_two_ints(...)` function and call it inside `main` where indicated.
2) Make the `compute_multadd(...)` function and call it inside `main` where indicated.
3) Make the `print_fancy(...)` function and call it inside `main` where indicated.

A Note on passing the tests

Any tests are very strict with respect to the format of requested prompts and printouts so pay attention to exactly what is requested and replicate it as exactly as possible.

Task 1: read_two_ints

Part 1.1: define the function

The objective of this function is to read two inputs from the user, cast them to integers, and return them. The steps are detailed below:

- Remove the `return 1, 2` line and instead:
- You should use the `input` function to read one number using the following prompt: `"give me x: "`; Note that 1) there is a space after the colon, and 2) whatever is read is a String.
- Cast the read variable into another by using the `int` function.
- You should use the `input` function to read another number using the following prompt: `"give me y: "`; Note that 1) there is a space after the colon, and 2) whatever is read is a String.
- Cast the read string variable into a new integer variable by using the `int` function.
- Return **both** integers at the same time. Example: if you want to return variables `a` and `b` at the same time, you use the statement:
  `return a,b`
- Note that the return statement has no parentheses!!

Part 1.2: call the function

- Add a call to `read_two_ints` where indicated, inside `main()`
- The call should invoke read_two_ints (with no arguments) and assign, as output, the two output values into two variables `x` and `y`.
- Note that, to "catch" multiple returned values, you need to use enough variables. Example: you know some function `foo()` returns two values, and you want to call them `n` and `m`, then, you use the statement:
  `n, m =foo()`

Task 2: compute_multadd

Part 2.1: define the function

The purpose of multadd is to calculate and print the parts of an operation on two input variables a and b, and then and return it. The whole operation is shown below:

$$\frac{a * b}{a + b}$$

In the following instructions, the notation `<something>` indicates a placeholder for something you need to calculate yourself.

You need to do the following:

1. locate the function with name `compute_multadd`;
2. verify it accepts two parameters, `a`, and `b`;
3. remove the `pass` keyword and replace it with the following steps;
4. inside, you should first calculate the numerator of the operation indicated above (`a*b`) and save it in a variable;
5. Then, you should print the variable holding the result of a*b with the following format:
   `"mult result: <result of a*b>"`
   Again, the `< ... >` notation is a placeholder for you to replace with the actual value or variable... for example, if the result of a*b is 15, the printout should be:
   `"mult result: 15"`
6. Then, you should calculate the denominator of the operation indicated above (`a+b`) and save it in a variable;
7. Then, you should print the variable holding the result of a+b with the following format:
   `"add result: <result of a+b>"`
8. you should return the result of the whole operation (if the first result was called `mult_result` and the second was called `add_result`, then you should return `mult_result/add_result` ... again, with no parentheses.)

Part 2.2: call the function

Add a call to `compute_multadd` where indicated, inside `main()`

The call should invoke compute_multadd with two arguments (`x` and `y`) and assign, as output, the output value into one variable called `xy_multadd`.

Task 3: print_fancy

Part 3.1: define the function

The purpose of print_fancy is to print the inputs obtained and the result of multadd in a fancy format.

You need to do the following:

1. locate the function with name `print_fancy`;
2. verify it accepts three parameters, `a`, `b`, and `ab_multadd`;
3. remove the `pass` keyword and instead;
4. It should print a row of 16 stars (`*`) equal to the example shown below
5. It prints the message: `RESULTS:`
6. Then `first number:` followed by the value inside the first number;
7. Then `second number:` followed by the value inside the second number;
8. Then `multadd result:` followed by the value inside the value of ab_multadd;
9. Finally, it should print a row of 16 equal signs (`=`) equal to the example shown below

Example: if the inputs of `a`, `b`, and `ab_multadd` are 10, 15, and 6.0, respectively, then the result of the print_fancy function should be the pringing of this:

```
****************
RESULTS:
first number: 10
second number: 15
```

```
multadd result: 6.0
================
```

Part 3.2: call the function

Add a call to `print_fancy` where indicated, inside `main()`

The call should invoke print_fancy with three arguments (`x`, `y`, and `xy_multadd`, which you defined as output of the compute_multadd call). This call has no output.

Output Examples

Below, we show three examples of printouts for three different runs of the program:

Example 1: if we give a 10 to the first number and a 15 to the second, we would get this whole printout:

```
give me x: 10
give me y: 15
mult result: 150
add result: 25

***************
RESULTS:
first number: 10
second number: 15
multadd result: 6.0
================

The End
```

Note that the prompts inside the `input` calls are printed just like print statements, and that whatever you write into the Console as input is also recorded (10 and 15 in this case).

Example 2: if we give a 8 to the first number and a 2 to the second, we would get this whole printout:

```
give me x: 8
give me y: 2
mult result: 16
add result: 10

***************
RESULTS:
first number: 8
second number: 2
multadd result: 1.6
================

The End
```

Example 3: if we give a 5 to the first number and a 20 to the second, we would get this whole printout:

```
give me x: 5
give me y: 20
mult result: 100
add result: 25

***************
RESULTS:
first number: 5
second number: 20
multadd result: 4.0
```

```
================

The End
```

Grading criteria:

General

The submission:

- runs without syntax errors (or -50%)
- adds a few small but informative comments (or -10%)

Operations

The program:

- Passes all 6 tests (or lose 15% per missed test).

Submitting

To submit, remember that you need to:

- Stage all changes for tracking (that is, run the command `git add .` on the repo directory)
- Commit your changes: run `git commit - m "< your message >"` NOTE: add a commit message that indicates what you've done so far (e.g. "finished part 1", or "completed the hw", etc)
- Push the changes by running: `git push`

Note that you can commit amany times nd push once o more times.