CSCI 1100 — Computer Science 1
Spring 2016
Homework 1: Calculations and Strings

Overview

This homework, worth **75 points** total toward your overall homework grade, is due Thursday, February 11, 2016 at 11:59:59 pm. The three equally-weight parts should each be submitted separately. All parts should be submitted by the deadline or your program will be considered late.

General Comments: Read This Before Starting the Homework

Welcome to CS-1 homeworks. We are starting this assignment by outlining the course homework policies. These will not be repeated so please read this carefully and refer back to it for future assignments.

Submission of homeworks: test first, but can submit many times

As you learned in Lab 1, all homework will be submitted electronically through the Department of Computer Science server. Please use the link

```
https://submit.cs.rpi.edu/index.php?course=csci1100
```

we used in Lab 1 for all assignments. This link is also available on the Piazza site. The homework submission server will typically be available by the Monday of the week homework is due. So, you can expect that for this homework, you should be able to submit **by Monday, February 8, 2016**, perhaps sooner.

Make a habit of testing your program by running it using the Wing IDE. First, make sure that it can run. Then, go through the logic to make sure that it is correct. Learning to test your code is a big part of learning to program. Use the submission server, which can get slow near the deadline, for submitting and final changes rather than for testing. You will not be penalized for submitting a program multiple times, but you are graded on the last homework you submit. Note that the time you submit the last version determines whether a homework is late or not.

How will you be graded?

- Program correctness will be the most important determinant of your grade. In some cases, especially later in the semester, we will test your code with many different inputs, not just the ones we provide as sample test cases in the homework description.
- Small differences in output spacing will cause small losses of points. While we care mostly about correct logic, making sure the output matches the sample output exactly teaches you to use print statements and strings correctly.
- In addition to looking at output, we will also read your code. Make sure your program is well-written and has clear, correct program logic. Test it yourself with additional test cases, not only the ones we give you. You may lose points even if you match all test cases but your logic is incorrect.

- Starting with homework #2, we will also look at programming style, such as the organization of code, the written comments describing the purpose of your code, the names of variables, the use of functions, etc. It is important to develop good habits even when writing relatively short programs.
- Remember that the last version you submitted is the one that will be graded. You can make a past submission the active one if you want us to grade that one by the submission deadline. It is your responsibility to manage this.
- **Can I use a programming construct that we have not learned yet?** You can and will not lose points for doing so — unless we give you an explicit warning — but why do it? We design homeworks to specifically target the concepts we are learning right now. Also, learning to do things in multiple ways will make you more versatile. Sometimes, simpler solutions will also be faster and cleaner. Helping you with a concept not taught in class is not a priority for us. Try other challenges instead.
- For each part, you must submit a program with the correct name. Otherwise, the submission server will not be able to execute your program and we will not be able to give you a grade.

  Your programs for each part for this homework **must** be named:

  ```
  hw1_part1.py
  hw1_part2.py
  hw1_part3.py
  ```

  respectively. Each should be submitted separately.

Late homework policy

When a homework has multiple parts, the submission time is the time that the last part is submitted. For example, if you submit parts 1 and 2 three days before the due date and part 3 one hour after the submission deadline, you will have used up one late day. Reread the late homework policy in the syllabus. You have a total of three full or part days you can use for late homeworks in the whole semester. You can use them all on a single homework or distribute them to multiple homeworks. It is highly recommended that you save these for later homeworks that will be harder and longer.

Wing IDE vs. Homework Submission server

On rare occasions when the homework submission server runs your code it will produce different output than the Wing IDE does. In such cases pay careful attention to the server's output and try to figure out what happened. Usually you've done something wrong in the way you wrote, submitted or executed your program. If you can't fix the problem, check the Piazza site for a relevant discussion. Only after checking should you post a question (and remember do not post your code!). This is also a good topic for help during office hours.

Input format for homeworks in this class

Your program must read the same number of inputs as are required according to the given problem. For example, if we tell you to read a name first and an email address next, that means your program must have 2 `raw_input` statements. If it does not, you will get an error like:

```
EOFError:  EOF when reading a line
```

This means either you are trying to read too many or too few inputs. Read the problem specification carefully.

In all homeworks, we will use a convention specific to CS-1. If we ask you to read an input, you must immediately print that input. For example, the following is the correct method to input the name string and the email address string:

```
name = raw_input('Enter a name ==> ')
print name
email = raw_input('Enter an email ==> ')
print email
```

The above program will produce a different output in Wing IDE and the Homework submission server as shown below:

```
Enter a name ==> Rensselaer
Rensselaer
Enter an email ==> rpiinfo@rpi.edu
rpiinfo@rpi.edu
```

Wing IDE output
(what you see on your computer)

```
Enter a name ==> Rensselaer
Enter an email ==> rpiinfo@rpi.edu
```

Homework submission output
(what you see on submission server)

If you forgot to add the print statements, you would actually see something like this in the submission server which will be considered incorrect output:

```
Please enter a name==> Please enter your email==>
```

The difference — and this is not really important for actually completing Homework 1 — is that for each part of the homework, we place all the input into a file and do what's called "running from the command-line." In the above example, we are using an input a file (let's assume it is called `input.txt`) that contains the two strings `Rensselaer` and `rpiinfo@rpi.edu` on two lines.

When a program is run from a command shell, we use a command-line of the form:

```
python part1.py < input.txt
```

Unfortunately, the free version of the WingIDE that we have been using in class does not allow us to specify this input. But, you can do the command-line form on Mac/Linux with a Terminal window, and on Windows using Linux-derived Cygwin tools. If you want to learn how, you can ask us during office hours. Anyway, here is the bottom line:

Anytime you read input, just print it immediately afterwards to match the expected output.

The Actual Homework Description

Part 1: Speed Calculations

Many exercise apps record both the time and the distance a user covers while walking, running, biking or swimming. Some users of the apps want to know their average pace in minutes and seconds per mile, while others want to know their average speed in miles per hour. For example, if I run 6.3 miles in 53 minutes and 30 seconds, my average pace is 8 minutes and 29.5 seconds per mile — abbreviated as 8:29.5 — and my average speed is 7.07 miles per hour.

Your job in Part 1 of this homework is to write a program that asks the user for the minutes, seconds and miles from an exercise event and outputs both the average pace and the average speed, accurate to two decimal places. Here is an example showing the expected output of your program:

```
Minutes ==> 53
Seconds ==> 30
Miles ==> 6.3

Pace is 8:29.5
Speed is 7.07 miles per hour
```

You can expect minutes and seconds to both be integers, but miles will be a float. Note that our solution generates the blank line before the output of the calculations.

In solving this problem you'll need to think about handling the mixed integer and float calculations and generating formatted output. All of these are covered in the lecture notes.

When you have tested your code and are sure that it works, please submit it as `hw1_part1.py`.

Part 2: Madlibs

In this part you will write a Python program to construct the Madlib given below:

```
Good morning <proper name>,
  Welcome to what promises to be a/an <adjective> <noun>.
  If you submit your <noun> on-time you will not lose <noun>
  and you will be <emotion>.
  If you don't you will be <emotion> and <verb> a poor <noun>.
```

You will ask the user of the program for the missing words — those enclosed in `< >` — using the `raw_input` function. You will then take all the user specified inputs, and construct the above Madlib. You can use any string method we have learned. Make sure your output looks like the above paragraph, except that the missing information is filled in with the user input. Here is an example run of the program (how it will look at the homework submission server):

```
Let's play MadLibs, CS 1.
Type one word responses to the following:

proper_name ==> Alex
adjective ==> challenging
noun ==> semester
noun ==> code
noun ==> points
```

```
emotion ==> happy
emotion ==> sad
verb ==> earn
noun ==> grade

Here is your Mad Lib...

Good morning Alex,
  Welcome to what promises to be a/an challenging semester.
  If you submit your code on-time you will not lose points
  and you will be happy.
  If you don't you will be sad and earn a poor grade.
```

I've provided reasonable inputs, but the idea of MadLibs is to input random words and see how silly the result looks. Try it!

Of course, the program you write will only work for the specific MadLib we've written above. A more challenging problem, which you will be capable of solving by the end of the semester, is to write a program that reads in **any** MadLib, figures out what to ask the user, asks the user, reads the input, and generates the final MadLib. Hmmm....

When you have tested your code and you are sure that it works, please submit it as `hw1_part2.py`.

Part 3: Framed Box

(If you struggle with this part, please work on the questions for Lab 2 first.)

Write a program that asks the user for the width and height of a framed box, and the character to use in the frame. Then, output a box of the given size, framed by the given character. Also, output the dimensions of the box in the lower right corner of the box, as shown below.

Assume that the user inputs valid values for each input: width is a positive integer (7 or higher) and height is a positive integer (4 or higher), and a single character is given for the frame.

Here is an expected output of your program (how it will look on the homework submission server):

```
Width of box ==> 7
Height of box ==> 4
Enter frame character ==> @

Box:
@@@@@@@
@     @
@ 7x4 @
@@@@@@@
```

Here is another possible output:

```
Width of box ==> 12
Height of box ==> 8
Enter frame character ==> #

Box:
############
#          #
#          #
#          #
#          #
#          #
#     12x8 #
############
```

You will need to put the box dimensions in a string first, and then use its length to figure out how long the line containing the dimensions should be. If you have prior programming experience you might be tempted to look for how Python implements "loops" in order to generate full frame, but Python provides string manipulation tools (Lecture 3) that makes this unnecessary.

When you have tested your code, please submit it as `hw1_part3.py`.