CSCI 1100 — Computer Science 1 Homework 1
Calculations and Strings

Overview

This homework is worth **75 points** total toward your overall homework grade (each part is 25 points), and is due Thursday, September 17, 2015 at 11:59:59 pm. There are three parts to the homework, each to 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. For this homework only, we will try to outline all relevant policies regarding homeworks below. We will not repeat these in future homeworks, so please read this carefully.

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 and will not be repeated in each homework handout. Generally, the homework submission server will 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, September 14, 2015, perhaps sooner.

Make a habit of testing your program by running it on Wing IDE. First, make sure that it can run. Then, go through the logic to make sure that it is correct. Submission server can get slow at busy times. Use it for submissions, not for testing your code. Learning to test your code is a big part of programming. However, you will not be penalized for submitting a program multiple times. You are graded on the active version, which is generally the latest version. The submission time is used to determine whether a homework is late or not.

How will you be graded?

In each part of this homework, you are asked to read user input and then produce some output. This will be typical of all homeworks. There are a few things that you will be graded on:

- First and foremost program correctness will determine your grade. But you will also get or lose points based on other criteria discussed below. For correctness, we will test your code with many different inputs upon submission, not just the ones we provide as sample test cases in homework description. Your output must match ours exactly to be considered identical. We will provide lots of help getting you there and there will be plenty of partial credit.
- Note: small differences in output spacing will cause you very small loss of points. What we care is correctness. But, matching output teaches you the use of print statements and strings correctly. What we care most is correct logic.

- 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.
- In addition to looking at output, we will also read your code. Make sure your program is well-written and has 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 organization of code, commenting the purpose of your code, names of variables, use of functions, etc. It is important to develop good habits even when writing relatively short programs. It will help you in the future.
- **Can I use a programming construct that we have not learnt yet?** You can and will not loose points for doing so, 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. But, 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 will be named:

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

  respectively. Each should be submitted separately.

Late homework policy

All homeworks are going to have multiple parts. If any part is late you will be charged late days according to the latest submitted part. Read the late homework policy in the syllabus. You have a total of three full 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.

If you want to ask for extra time on homeworks beyond this, you must have an exceptional exception. Please get an official excuse from Student Experience in such cases.

Wing IDE vs. Homework Submission server

Homework submission server runs your programs and those of many other classes at the same time. As a result, the method that it uses to run them is different in various ways. You may see a different output on the submission server than the one on Wing IDE or get a totally foreign error message on submission server.

The homework submission server is new and we are still perfecting it. We will do our best to find out what these errors mean and provide you with a key. We will post the key on Piazza and also code it into the submission server over time. In the meantime, please be patient with us.

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 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 will immediately print that input. For example, the following is the correct method:

```
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)

And, 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 your homework — 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 particular, in the above example, we are using an input a file (let's assume it is called `input.txt`) that contains two strings: `Rensselaer` and `rpiinfo@rpi.edu` in 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. You can do the same 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.

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

1 The Actual Homework Description

Part 1: Framed Box

In this part, we will solve a framing problem similar to Lab 2.

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, display a box of the given size, framed by the given character. Also, display the dimensions of the box inside the second line of the box (just after the left frame and a space).

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     #
#          #
#          #
#          #
#          #
#          #
############
```

This is in essence very similar to the lab, except that you will need to put the box dimensions in a string first, and then use its length to figure out how long the second line should be.

When you have tested your code, please submit it as `hw1_part1.py`.

Part 2: Madlibs

In this part you will write code to construct a madlib given below:

```
Look, <proper name> ...
  I can see you're really <emotion> about this ...
  I honestly think you ought to <verb> calmly ...
```

```
take a <adjective> <noun> and think things over ...
I know I've made some very <adjective> decisions recently,
but I can give you my complete <noun> that my work will be back
to <adjective>.
```

You will ask the missing pieces of the following madlib to the user using the (`raw_input`) function we learnt in class on thursday. For each input, you will ask the specific type of word required.

You will then take all the user specified inputs, and construct the above madlib. You can use any string method we have learnt. 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):

```
proper name ==> Brad
emotion ==> happy
verb ==> eat
adjective ==> silly
noun ==> mouse
adjective ==> hungry
noun ==> computer
adjective ==> cute

Here is your output:
Look, Brad ...
 I can see you're really happy about this ...
 I honestly think you ought to eat calmly ...
 take a silly mouse and think things over ...
 I know I've made some very hungry decisions recently,
 but I can give you my complete computer that my work will be back
 to cute.
```

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

Part 3: Name popularity

How popular were different names in different decades? Which names have increased or decreased in popularity? We can actually find out thanks to the various real data sets available these days. This homework will be based on real data, but we only ask you to read the numbers for one name using `raw_input()`.

Write a program that reads the number of babies born with a specific name in decades: 1970, 1980, 1990 and 2000. You must then print the information using a table like the one below. Then, compute the percent change in popularity between each consecutive decade. Print this information and the average percent difference.

You will need to use formatted strings for this to look correct. If you want to print percent sign % in a formatted string, you need to escape it by writing it twice. Here is an example:

```
>>> print "The decrease: %% %.2f" %(12.345678)
The decrease: % 12.35
```

Here is an example output of the program for name `Anya` (how it will look on the homework submission server):

```
Please enter a name ==> Anya
Count in 1970 => 520
Count in 1980 => 1326
Count in 1990 => 1495
Count in 2000 => 6875

Babies named Anya
*****************

Year / Total / % change from previous decade
1970 / 520
1980 / 1326 / % 155.00
1990 / 1495 / % 12.75
2000 / 6875 / % 359.87
Average change: % 175.87
```

How are the above percentage values computed? Let us look the first value: `155 = 100*(1326-520)/520`

Basically, in 1980s, there was an increase of `1326-520=806` for this name. How big is this value compared to the original `520`? We find this by computing `100*806/520`.

Note that for this part, we are not trying to produce a very polished output and we will not align columns in the table for decades. We simply have a single space and slash (`/`) after each value. Assume no zero values for any decade (so you will not get division by zero errors).

This is a relatively long program, but it is also repetitive. Let us try to employ the principle of iterative programming to solve it.

- First, write a program to read one input and print it. What is a good name for your variable?
- Now, add another line to read the next input, print it. Also, compute the percentage change from the previous one. Print this as well.
- Look at formatting. How do you print only two digits after comma? How do you print the percent sign? Fix it.
- Now, worry about reading the name and print it. Also figure out how many stars to print in the next line. How many are needed, given the length of the name?
- Now, add all the percentage values and divide by three to find the average. Display this value.
- You are not done yet! Don't forget an important part. TESTING! Test with some simple values: 1,2,4,8 and 1,2,6,24 (and decreasing values as well). What do you expect to see? Do you match? Now, try some more challenging values.

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

Finished and still want extra challenges to sharpen your programming skills?

Here are some ideas to try on your own (do not submit these): In part 1, try centering the box dimensions in the center of box and modify your program to work for any number of characters for the frame. In parts 2 and 3, try to write the program with and without formatted strings (even floats can be formatted without formatted strings, though takes some effort). As a final challenge, align the table for different decades by making sure each column is exactly 12 characters wide (do not use a specialized function).