Computer Science 1 — CSci 1100
Lab 1 — Variables, Assignments and Expressions
Fall Semester 2014

Lab Overview

The goal of this lab is to start writing short programs involving variables, assignments and expresssions. This will be accomplished by three small checkpoints. We will practice finding and fixing syntax and semantic errors, we will study good program structure and also learn about hard-coding and how to avoid it.

Checkpoint 1: Syntax versus Semantic Errors

Download the program `lab1_area.py` from the Piazza resources pages under Labs. (Store this in a folder you create for Lab 1.) This simple Python program to compute the surface area of a rectangular solid (the sum of the areas of the six rectangular faces) contains **both** syntax errors (two) and a semantic error. If you don't remember the difference between a syntax error and a semantic error, look back at the Lecture 1 notes.

Working on your own machine and IDE, please find and fix the two syntax errors, making sure that the program runs. Remember that Wing IDE actually shows you syntax errors by underlining a part of the line in red. Often, this is the line containing the error.

When you have fixed the syntax errors, now fix the semantic error. You will see that fixing a semantic error is much harder, because you need to understand what the program is trying to do. This is where good programming practice comes in handy. Remember in your programs to:

- give meaningful (but short) names to your variables,
- write your program so that it has a clear logic flow: assign values to variables first, then use these variables in computation and then print the result of the computation,
- and finally write short comments to explain what your program is doing at pivotal points (we will talk about this later in class).

To complete Checkpoint 1: show a TA or a mentor that you have successfully resolved all the errors in this program and are able to run it.

Checkpoint 2: How big is your hard disk?

How much data can your hard drive take? This question is harder to answer than you think. The main measure of data in computers is a byte. For example, an integer in Python is 8 bytes.

In computer science, a gigabyte (GB) is a measure equivalent to $2^{10} * 2^{10} * 2^{10} = 2^{30}$ bytes. In fact, most operating systems use this convention. A terabyte (TB) is 1024 gigabytes, or 2^{40} bytes. We call this base 2 computation, everything is a power of 2.

However, computer manufacturers find this too complicated. So, instead they use base 10 computation. When they sell you a computer, a gigabyte is assumed to be $10^3 * 10^3 * 10^3 = 10^9$ bytes. A terabyte similarly is 10^3 gigabytes, i.e. $10^3 * 10^9 = 10^{12}$ bytes.

So, when you buy a 128 GB hard disk (manufacturer definition, base 10), put it in your computer, and your computer now thinks that it is actually 119 GB (computer definition, base 2). As a result, you have somehow lost 9 GB before even using your computer. Sad but true.

In this checkpoint, you will write a program that prints out this size difference. To accomplish this, you are going to write a new Python program. Start by clicking `File -> New` in the IDE to create a new program to type into. Type your solution for 128 GB first. Make sure you use at most three variables: one for base 10 hard disk size, one for base 2 hard disk size and one for the difference. Assign value 128 to the base 10 disk size, then write formula to compute the value of the other two variables. When you are done with the computation, print the result in the following format using your variables:

```
128 GB in base 10 is actually 119 GB in base 2, 9 GB less than advertised.
```

Now, save this (very short) program to a file called `lab1_part2.py` in your Lab 1 dropbox folder. Run it to make sure it is correct. Once it is complete, now reuse your code to print the same information for hard disk sizes of 256 GB, 512 GB and 1024 GB as well. If you did this right, all you have to do is copy all your code, and change only the value of the base 10 variable. It looks repetitive right? We will talk about that in class on thursday!

No hard-coding: You must use at most three variables for this solution and no hard coding of values. What is hard-coding? It is writing the solution directly into a program. In these early exercises and homeworks, the problems are easy so we know what the answer is before we start. This does not mean we should not write a program to learn the program syntax. For example, the following print statement is using hard-coding:

```
base10 = 128
print "128 GB hard disk"
```

The following print statement is not using hard-coding:

```
base10 = 128
print base10, "hard disk"
```

Of course, we need to hard-code the value for the initial variable for now until we learn how to read user input.

To complete Checkpoint 2: When you are convinced your program is correct, show it to a TA or a mentor. Your program should have only three variables and should not hard-code values.

Checkpoint 3: Jupiter and Earth

For the next checkpoint, you are going to write Python code that calculates and outputs some properties of Jupiter, the largest planet in our solar system. Jupiter has a diameter of 88,846 miles and its average distance from the Sun is about 483,632,000 miles. These look like big numbers, but they are a bit hard to understand intuitively. One way to appreciate them is to compare them to the values for the Earth, which has a diameter of 7,926 miles and a distance from the Sun of about 92,957,100 miles.

Write a Python program to calculate and output

1. the ratio of Jupiter's distance from the Sun to that of the Earth,

2. the ratio of Jupiter's volume to that of the Earth,

3. the time in minutes and seconds it takes light to travel from the Sun to Earth (use 186,000 miles per second as the speed of light), and

4. the time in minutes and seconds it takes light to travel from the Sun to Jupiter.

Similar to checkpoint 2, start by clicking `File -> New` in the IDE to create a new program to type into. Type your solution to just the first calculation, using variables for both the Earth's distance and Jupiter's distance, and printing a string along with the ratio to explain the output. Save this (very short) program to a file in your Lab 1 dropbox folder, and then run it. Your output should look something like

```
Jupiter is 5.202744061507943 times farther from the Sun than the Earth is.
```

Repeat this process for each of the required calculations, adding to your program. Run the program after adding each part of the solution. For the second calculation, you will need the volume code from Lecture 2. For the third and fourth calculations, start by calculating the number of seconds. Then convert this number to an integer (if it is not already an integer). Finally, convert it to minutes and seconds. This is a bit tricky, requiring the use of integer division and remainder discussed during lecture. Can you figure it out?

To complete Checkpoint 3: show a TA or a mentor the completed solution, including both the code and output. You are allowed to hard-code the initial values given to you (diameter and distance of planets), but not the results of the computations.

Learning to use the submission server

This part of the lab is not required for a grade, but everyone must try it before the end of this week to get ready for homework submissions. To complete this part, you will the original version of the program `lab1_area.py` with syntax errors and your corrected version.

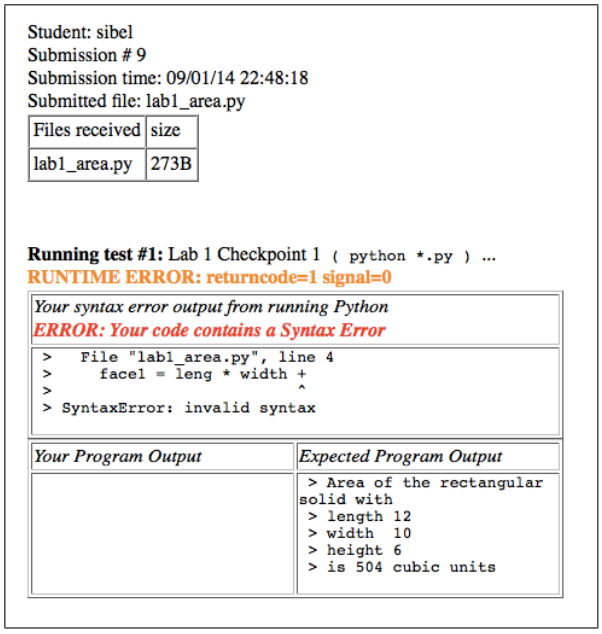
All submissions for the homework will be going through a Department of Computer Science web server. You will log into this server, upload your Python programs, run them, and view the results. You should be able to login using your RCS username and password. If this is not the case, you should let your TA know immediately and we will make sure that you have an account created on the submission server. Once the accounts are created, try this again.

1. First see what happens when you submit a homework that contains a syntax error. To do this, cut-and-paste the following link to the submission page into a browser:

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

   This submission is also linked from the Piazza site under Resources.

2. Now, submit a file with syntax errors such as the original `lab1_area.py`. Click on the radio button for Lab 1, then click on the `Choose File` button, browse to choose `lab1_area.py`, and click on `Send File`. The homework server will attempt to run this program using Python. It will fail, of course, because of the syntax errors, and the server will generate output that looks like this:



Student: sibel
Submission # 9
Submission time: 09/01/14 22:48:18
Submitted file: lab1_area.py

Files received	size
lab1_area.py	273B

Running test #1: Lab 1 Checkpoint 1 `( python *.py )` ...
RUNTIME ERROR: returncode=1 signal=0

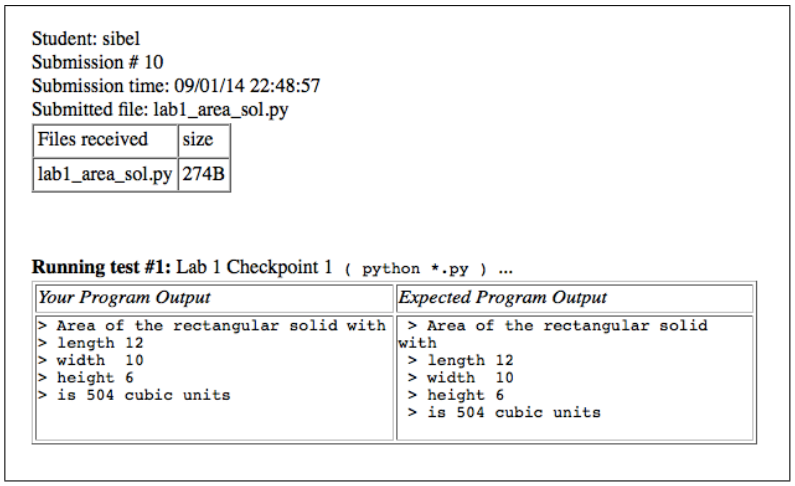
Your syntax error output from running Python
ERROR: Your code contains a Syntax Error

```
>   File "lab1_area.py", line 4
>     face1 = leng * width +
>                          ^
> SyntaxError: invalid syntax
```

Your Program Output	*Expected Program Output*
	> Area of the rectangular solid with > length 12 > width 10 > height 6 > is 504 cubic units

This is what the server shows you when your program has syntax errors. One thing to remember is that Python only shows you the first error!

3. Now, submit the syntactically-correct version of `lab1_area.py` to the homework server from checkpoint 1, again using the Lab 1 radio button. The server will now be able to run your program. It will display your results side-by-side with the expected results, and you will be able to see any discrepancies. Here is an example:



Student: sibel
Submission # 10
Submission time: 09/01/14 22:48:57
Submitted file: lab1_area_sol.py

Files received	size
lab1_area_sol.py	274B

Running test #1: Lab 1 Checkpoint 1 `( python *.py )` ...

Your Program Output	*Expected Program Output*
> Area of the rectangular solid with > length 12 > width 10 > height 6 > is 504 cubic units	> Area of the rectangular solid with > length 12 > width 10 > height 6 > is 504 cubic units

4. Repeat this as many times as you need to until your answer matches the expected output.

Final note on the homework server: You have seen at this point that multiple submissions of the same assignment are accepted on the homework server. There is no limit to the number of times you may submit and the server keeps track of all submissions. Please be aware, however, that the TAs will only look at the **last version you submit** when grading your homework.