Computer Science 111

Lab 2: Doing Arithmetic

This lab lets you practice using Python statements for doing input, output, and arithmetic.

1 Python Syntax Errors!

Sit in front of a computer, log in, and start up IDLE in interactive mode.

Python is perfect at finding syntax errors, but not great in diagnosing what the problem is. Therefore the error message can be hard for beginners to interpret.

Type the statements below, one by one. *You should get a syntax error after each one!* I have numbered them with comments for reference. Read the syntax error messages you get for each line and compare them to the notes below.

```
>>> class = "Computer Science 11"          #1
>>> first-name = "Monty"                   #2
>>> primt ()                               #3
>>> print (first-name)                     #4
```

1. The word `class` is a keyword, so is not allowed as a variable name. Python thinks you are misusing the `class` keyword and says *invalid syntax*.

2. You can't have hyphens in variable names. Python thinks you are trying to assign a value to an expression with a minus sign (an operator), and complains that the operator is being misused.

3. A misspelling of `print`. Python doesn't recognize the word `primt`.

4. The syntax is ok, but the part inside the print statement is interpreted as an expression (`first` minus `name`), with two variables. These two variables haven't previously been given values. Python balks at the first one, saying it is "not defined." If there are multiple problems with one statement you will

only see an error message for one problem. And, as in this case, there may be no error on the line it is complaining about.

In this and future labs, you should practice reading and trying to understand the syntax error messages. It is also important to practice ignoring the exact message and tracking down the error using your own understanding of Python syntax rules.

2 More about assignment statements.

Here are a couple additonal features of Python that were not mentioned in lectures this week.

Augmented assignment operators. You can combine any arithmetic operator with the assignment operator to get an "update" operation.

For example, `x = 33` transfers the value 33 to `x`, and it doesn't matter what the old value of `x` was. But `x += 33` modifies `x` by adding 33 to its current value. So `x += 33` has *exactly the same meaning* as `x = x + 33`.

Try typing these statements into the intepreter. After each one, type `x` to see what its value is, and check that you understand why.

```
>>> x = 3
>>> x     # see what its value is
>>> x += 2
>>> x     # now what is it
>>> x  *=  10
>>> x     #  and now?
>>> x  +=  x + 1
>>> x     # how do you explain this?
```

Some programmers who learned other languages prefer to use the augmented assignment operator. You don't have to if you don't want to.

String operators There are two operators, used on strings, as shown in the table below. There are also lots of built in functions. We will talk about programs that compute on strings later in the semester.

	Example	Meaning
+	`greeting = "Hi " + "there"`	Glue two strings together
*	`laugh = "ha " * 5`	Repeat the string some number of times

Output via print statements. Here are some examples of ways to use print statements.

```
>>>  m = 9
>>>  d = 13
>>>  y = 2013
>>>  print (m, d, y)   #print three values separated by space
>>>  print (m, d, y, sep='/', end=" ")   #separated by slash,  no newline
```

Formatting your output You can use a `format` function to specify how you want a number to look when printed. The function is of the form `format ( number, string)`, where *number* is some type of numeric (integer or float) value, and the *string* specifies how it should look. Here is a simplified table of format components (you can look up the fancier things on the web if you like).

	values	meaning
align	`< > =`	justify left, right, center
width	positive int	min field width
comma	,	use commas
precision	. int	number of decimal places
type	`f e d`	float, scientific, decimal integer

```
x = 12345.6789
print ("x= ",  format ( x ,  "3.2e")  )
print ('The integer part is',  format (int (x) , "10d"))
print ("Total price: ",  format (x,  "=10,.2f"))
```

Formats are easier to understand if you read them right-to-left:

1. The first format says prints the value of x using scientific notation (e for exponential); use 2 decimal places of precision; and use a minimum field width of 3. If the number won't fit into the minimum field width, the field grows so the whole number can be printed.

2. Print it as an integer (decimal number), using 10 spaces for the field. The number will be right-justified within that field. Note that `x`, which is a `float` has to be converted to an int to avoid a syntax error. Try it without the conversion and see what happens.

3. Print it as a float with two decimal places, and insert commas as appropriate. The number is centered in a field of total width 10.

3 Lab 2 Assignment

Enough playing around. Now it is time to write some real code!

- Open a browser and point it to `www.amherst.edu/ccm/cs111/lab2.py`.
- Right-click and choose Save As to download `lab2.py`. Ok to save it on your Desktop, but you probably want to move it to your U drive for permanent storage. Remember that items on the desktops get erased overnight.
- Open the file in IDLE: select **File: Open** and browse to find it. Now you are working in compile mode in the File window.
- To run the program, select **Run: Run Module** from the top menu. You could also type FN-(F5) by holding down the Function key and the F5 key at the same time. When the program runs, you are the user and must respond to all prompts. See how that works?
- You can save your program by typing COMMAND-S on a Mac or CTRL-S on a Windows platform. **You have to save it before running it. Every time you make a change.**
- Now go back to the File window and edit the file, following the instructions in comments. ***Get in the habit of re-saving and re-running after typing just a few lines (like 4 or less)!*** Seriously. Otherwise you you will spend all your time tracking down a ton of syntax errors. It's faster to work incrementally and fix them as you go.
- When you have each part finished, run the program to check that it works.
- When you are finished, point your browser to `http://www.cs.amherst.edu/submit`. Use the Submit program to turn it your working Python program by 9:00 MONDAY MORNING.
- Quit IDLE and Log Out before leaving the lab!