**Assignment Statement Continued**

In the previous example, we saw why it's important to order a set of statements properly. Here is a second example to illustrate that same idea:

>> x = 4

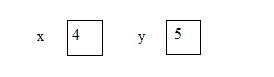
>> y = 5

>> x = y

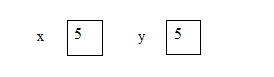
>> y = x

It is likely that the user wanted to exchange the values of x and y, so that x will be 5 and y will be 4 after this sequence of code is executed.

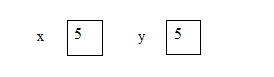
Unfortunately, this is not what happens. Let's take a look at what memory looks like after the second, third and fourth statements. After the second statement, we have:



In the third statement, we evaluate the right hand side (y) to get 5. Then we set the variable on the left, x, to that value:



Now, when we execute the last statement, we see that x is now 5, so y will get set to 5, NOT 4:



The key problem here is that we need the value 4, but we wrote over it.

To solve this problem, we create a third variable to store the value 4, so that we have an extra copy of it that doesn't get written over. Here is the correct version of the code that executes the exchange/swap:

>> x = 4

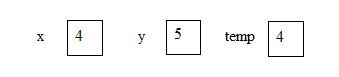
>> y = 5

>> temp = x

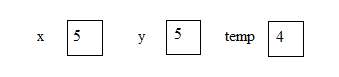
>> x = y

>> y = temp

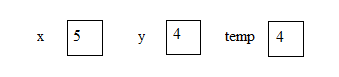
The key here is in the last statement, we must set y to what is stored in temp, not x. Our picture after the third statement is as follows:



After the fourth statement, we have:



Then, when we execute y = temp, we get:



**Writing a Program in a Separate File, using IDLE**

Using the interpreter in IDLE is nice, since you get to see the results of your code immediately. However, once you close the IDLE window, everything you have written vanishes. It would be nice to save some sets of commands so that you can execute them at any time, without rewriting them. We can do so by writing a python program in a separate file. To do so, in the IDLE editor, click on the File menu and choose "New Window". In this new window, you can type a sequence of statements that you would type in the IDLE editor. It's also customary to add what is known as a header comment in the file to indicate who wrote the program, when they wrote it, and a basic description of what the program does. The following is a program that calculates the area and perimeter of a rectangle. (Everything shown is what's in the separate file.)

# Arup Guha

# 1/11/2012

# Calculates the perimeter and area of a rectangle

length = 5

width = 8

perimeter = 2\*(length + width)

area = length\*width

print("The perimeter of your rectangle is", perimeter)

print("The area of your rectangle is", area)

A comment is denoted by the # sign. Everything after a pound sign on a line is ignored by the interpreter and is only there for other people reading the program.

After this program is typed in, it needs to be saved. This can be done by clicking on the File menu and choosing "Save As" and saving the program as .py file. (For example, the file on the previous page is area.py.) After the file is saved, go to the Run menu and select "Run Module". If everything in your file is okay, the result of running your program should appear in the original IDLE window. For this program, here is what appears:

The perimeter of your rectangle is 26

The area of your rectangle is 40

If you have a mistake in your program, then an error message will appear in the original IDLE window, when you try to run your program. There are many different error messages, some are very helpful and others are quite ambiguous. Through practice you will get a better idea of what you did wrong based on the error message. Most of the time, the error reflects the very first thing that the interpreter had a problem with. This may or may not be the root cause of the problem.

If we replace the third line of the program with

perimeter = 2\*(length + width

the error message we receive is:

invalid syntax

with area highlighted. This is because the compiler is looking for a close parenthesis, and once it sees area, realizes that it won't find one that's valid. Thus, we have to look at the previous line to find our error.

**Order of Statements**

Note that the order of statements in this program is important. If we do the following:

perimeter = 2\*(length + width)

area = length\*width

length = 5

width = 8

print("The perimeter of your rectangle is", perimeter)

print("The area of your rectangle is", area)

then our program will not work. The reason is that when we arrive at the first line, the variable length has no known value, thus, the computer can NOT evaluate the value of the right hand side of this assignment statement.

Based on what information is necessary, here is an alternate ordering of statements which will work:

width = 8

length = 5

perimeter = 2\*(length + width)

print("The perimeter of your rectangle is", perimeter)

area = length\*width

print("The area of your rectangle is", area)

The key here is to note that we don't need to calculate the area to print the perimeter, so we we can switch the order of those two statements without any ill effects. Similarly, we can switch the first two statements because the length is not necessary in order to assign the width.

**Printing out variables and multiple entities in a single print statement**

From the program above, the following line illustrates how to print out the value of a variable and how to print out multiple items of information in a single line:

print("The area of your rectangle is", area)

To print out the value of a single variable (and nothing else), just don't use quotes:

print(area)

To print out multiple components (strings and variables), separate out each component with a comma:

print("The area of your rectangle is", area)

Here is an example that uses more separate items:

a = 5

b = 7

print(a,"+",b,"=",(a+b))

This print statement prints out 5 separate components. When you run it, you'll notice that it separates out each component with a space. If you want to avoid this, do the following:

print(a,"+",b,"=",(a+b),sep="")

Basically, the print function takes in a variable number of pieces of information. Each piece of information is separated by a comma. If you want no space printed out between pieces of data, then for the last piece of information to the print function, you have to put sep="", which basically means that you want the separator to be no space. In these two examples, the output would be

5 + 7 = 12

5+7=12

In fact, you can choose any item to be a separator. Consider the following statement:

print(5,7,9,sep=" plus ")

The output would be

5 plus 7 plus 9