**ICTPRG301**

**Lesson 4 Notes**

**Buggy Programming**

*NOTE. Please read the notes on Debugging.*

In this session we will return to variables to extend our knowledge. In the notes on Variables the concepts of Mutability and Scope were mentioned. Scope refers to the place in the program where a variable exists and can be used. Mutability refers to whether a variable can change its value during an operation on the variable.

As a general rule variables are usually classed as local in Python, which means that they exist only in the location where they are declared (ie in the function or the module). They can be classed as global (ie used everywhere) but this needs to be done explicitly.

a=5  
def inc(x):  
 a=a+x  
inc(a)

This code will throw an error on the line a=a+x. The left hand side of the assignment is fine, however the right hand side (a+x) is the problem. This side actually instructs Python to take the existing a and add the value of x to it. The problem is that inside the function there is currently no existing variable called a. One is being created but the other a that has already been created in the module does not exist in the function. This is the issue of **Scope.**

Most variables in Python are immutable (eg Strings, all types of numbers, booleans etc), however list (and some other collections) are mutable. What this means is that if you do an operation on a variable of type String then the original variable will not change. The only way to change a variable of this type is to re-assign it with a new value. However if you do an operation on a list the original list will change.

Let’s see how this works

Fire up the Wing IDE and enter the following in the shell:

a=5

This creates a variable called a and assigns it to the value 5. a is immutable, which means that it can’t be changed. Its current scope is the shell, which means that it can be used in the shell and nowhere else.

However you may not be aware that in Python everything is an object. Objects contain values and methods (functions). If you want to see the list of functions for an object there are two command for this dir and help. Type in:

dir(a)

The shell gives you a long list of functions that you can call on the variable a. Most you will never use, to understand them help(a) will give you some help to understand them, however it may be better to use Google with the search Python function\_name.

Type in:

a.bit\_length()

This command will give the amount of memory that is used to store that variable, you can see that it currently uses 3 bits of memory. Change the variable to 1000. (Remember it is immutable so the only way to do this is to reassign the variable a=1000). Now run the bit\_length command again (You can do this with the up arrow)

Create a variable

b=”hello”

Now have a look at the built in commands for a String using dir(b).

*(Unfortunately help(b) does not work, to see the help file for Strings you need to type in help(str). I have searched for a reason for this quirk in the language but no-one can give me a rational explanation).*

Some are the same as for a number variable but many are different. One of the commands is upper. Type in:

b.upper()

Did you see what this did? However you might think that b has been changed. This is not the case, Strings are immutable. Type into the shell:

b

to see that its value is still the same. Let’s look at some other built in functions. Try these to see what they do:

b.join(‘me’)  
b.join(‘me with this word’)  
b.count(‘b)  
b.count(‘pe’)  
b.replace(‘e’,’a’)  
b.find(‘l’)

b is a simple one word string, but whole sentences can be Strings. Create a new String:

c=’Monty Python Flying Circus TV Show is where the language Python got its name’

now try these

c.replace(‘Python’,’Cleese’)  
c.count(‘Python’)  
c.find(‘Show’  
c.find(‘Python’)  
c.split()

If you want to find out more on these and the other built in functions start looking at the Python documentation (<http://docs.python.org/library/stdtypes.html>)

In the shell create a list:

a=[1,2,3]

Type in dir(a) to see the list of built in commands (help also works). Print a to see what the list looks like. Then type in:

a.reverse()

and print out the list a again. Notice that it is changed. Lists are mutable, they can be changed. Type in the following:

a.append(‘help’)

and print out the list again. Now type in:

a.pop()

and print the list again. In fact pop does two things, it removes the last item from the list and it also returns that item so it can be assigned to a variable. To see this type in:

b = a.pop()

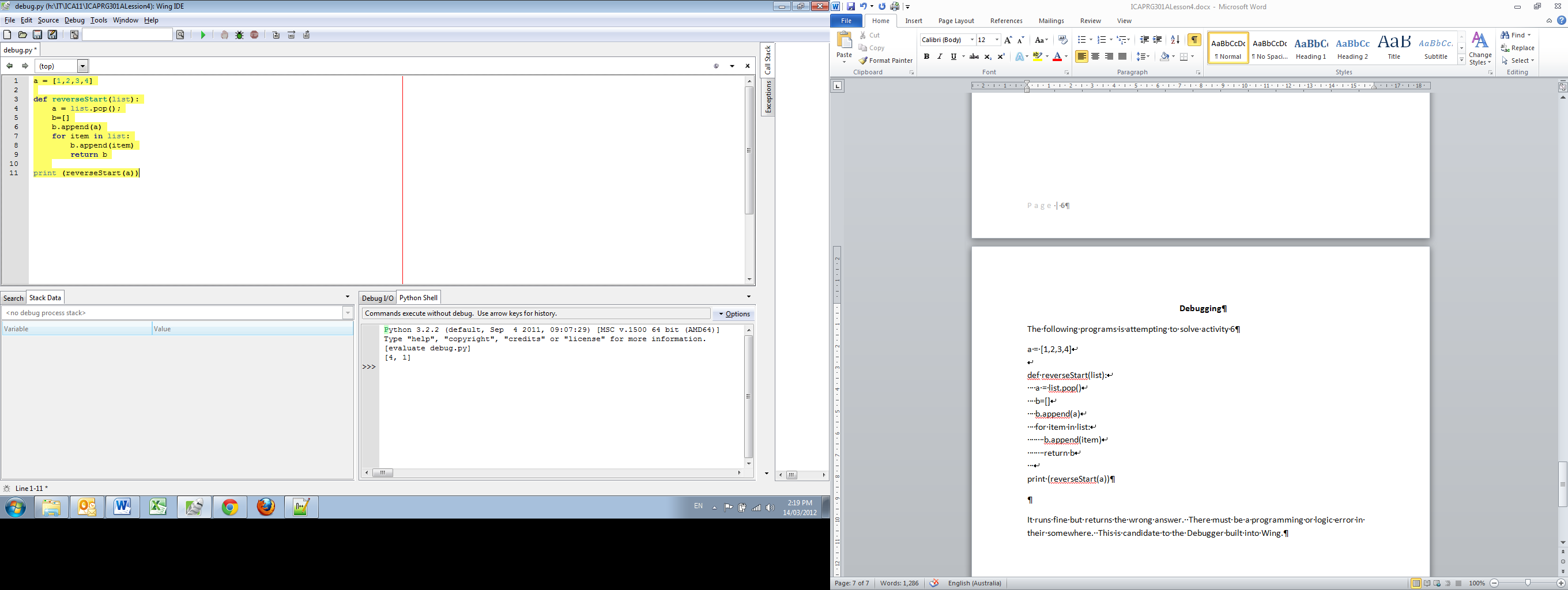
now print out b.

**Debugging**

The following programs is attempting to solve activity 5

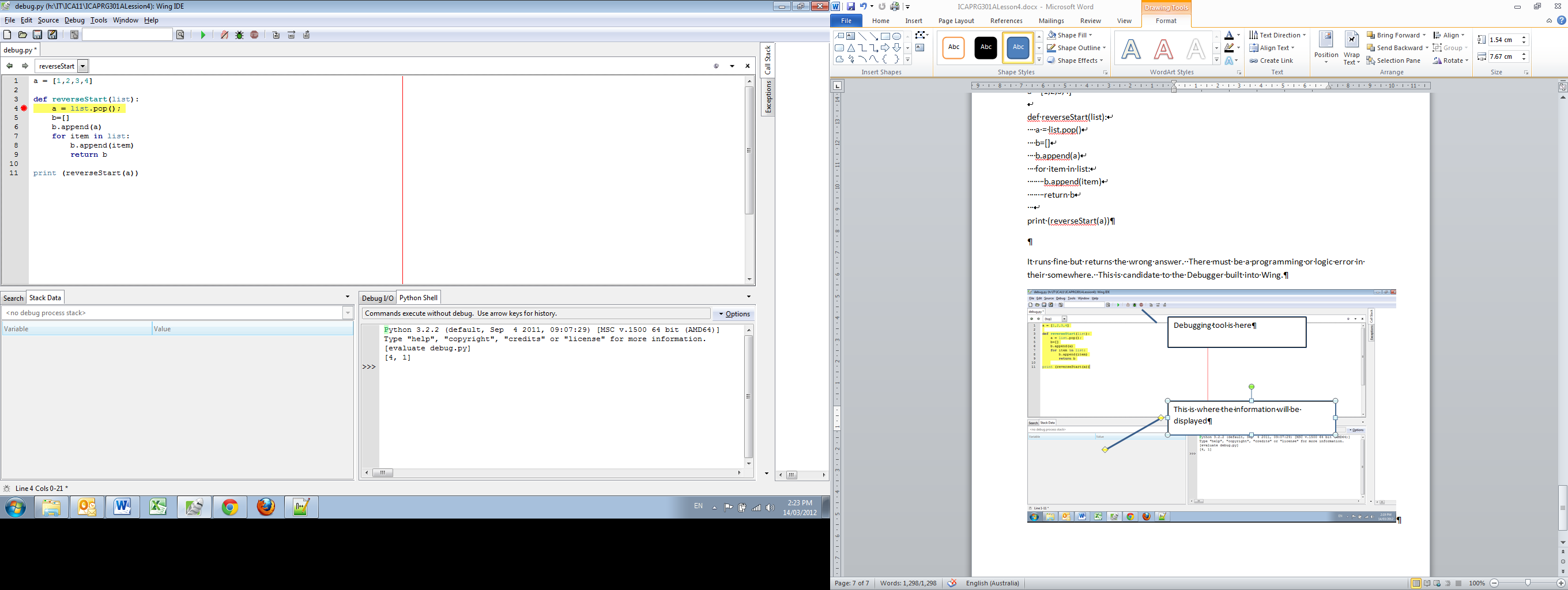
a = [1,2,3,4]  
  
def reverseStart(list):  
 a = list.pop()  
 b=[]  
 b.append(a)  
 for item in list:  
 b.append(item)  
 return b  
   
print (reverseStart(a))

It runs fine but returns the wrong answer. There must be a programming error in their somewhere. This is candidate to the Debugger built into Wing.

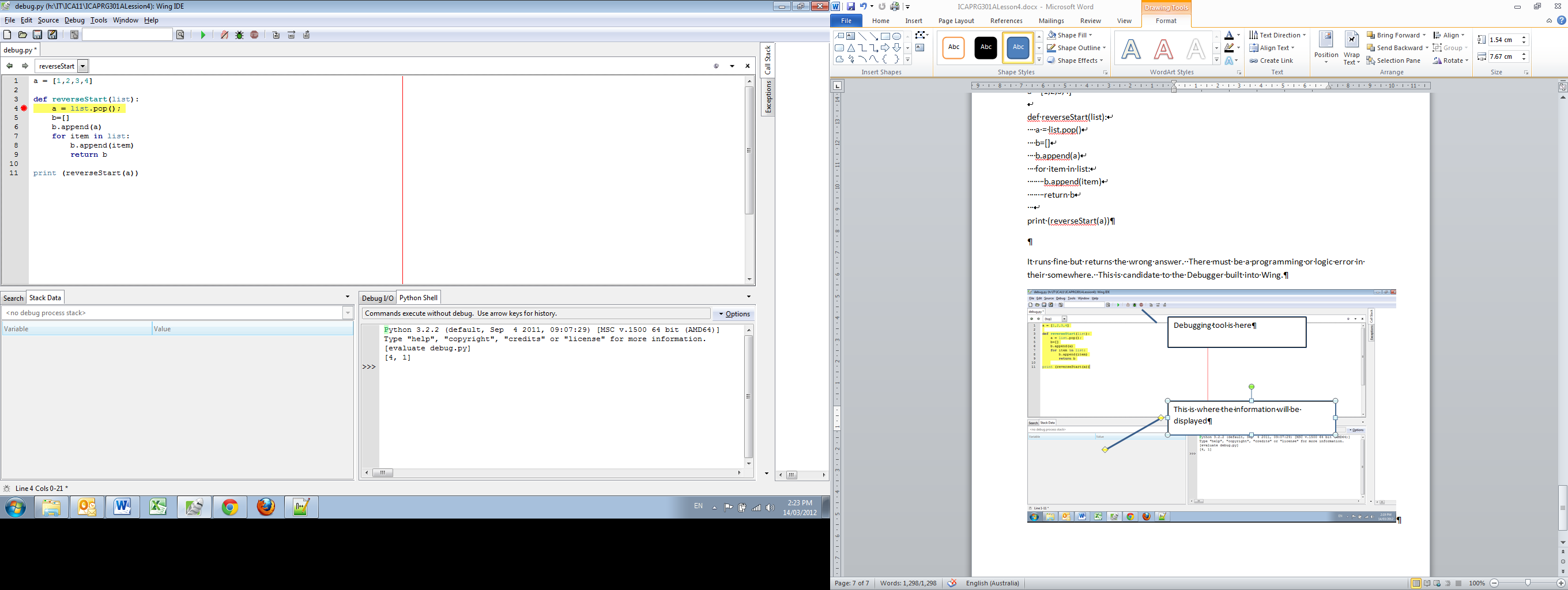


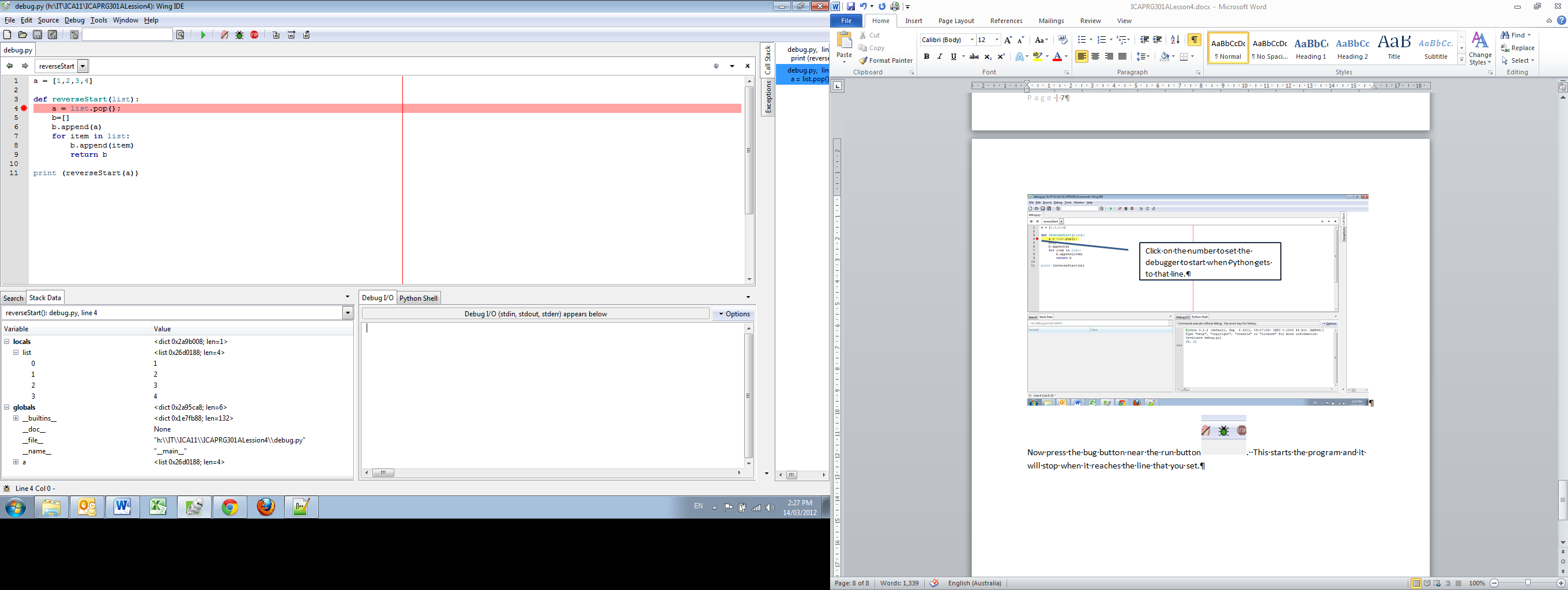
This is where the information will be displayed

Debugging tool is here



Click on the number to set the debugger to start when Python gets to that line.

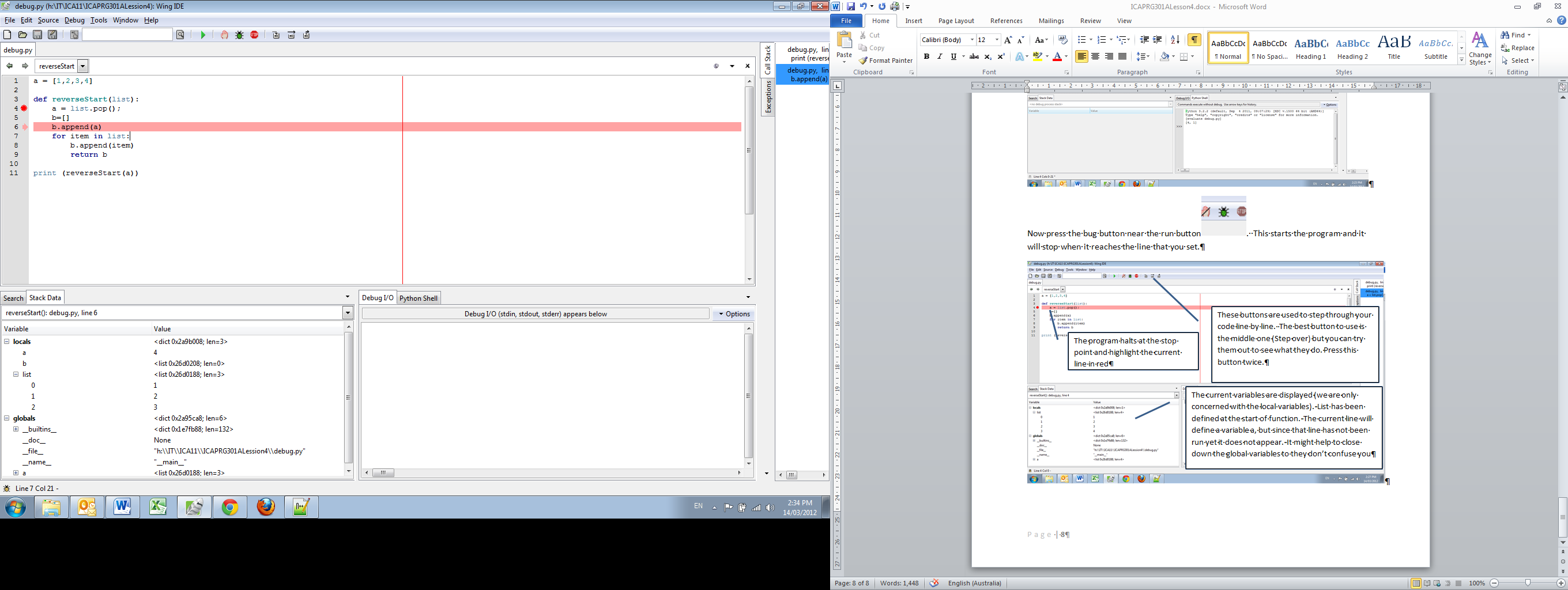
Now press the bug button near the run button. This starts the program and it will stop when it reaches the line that you set.



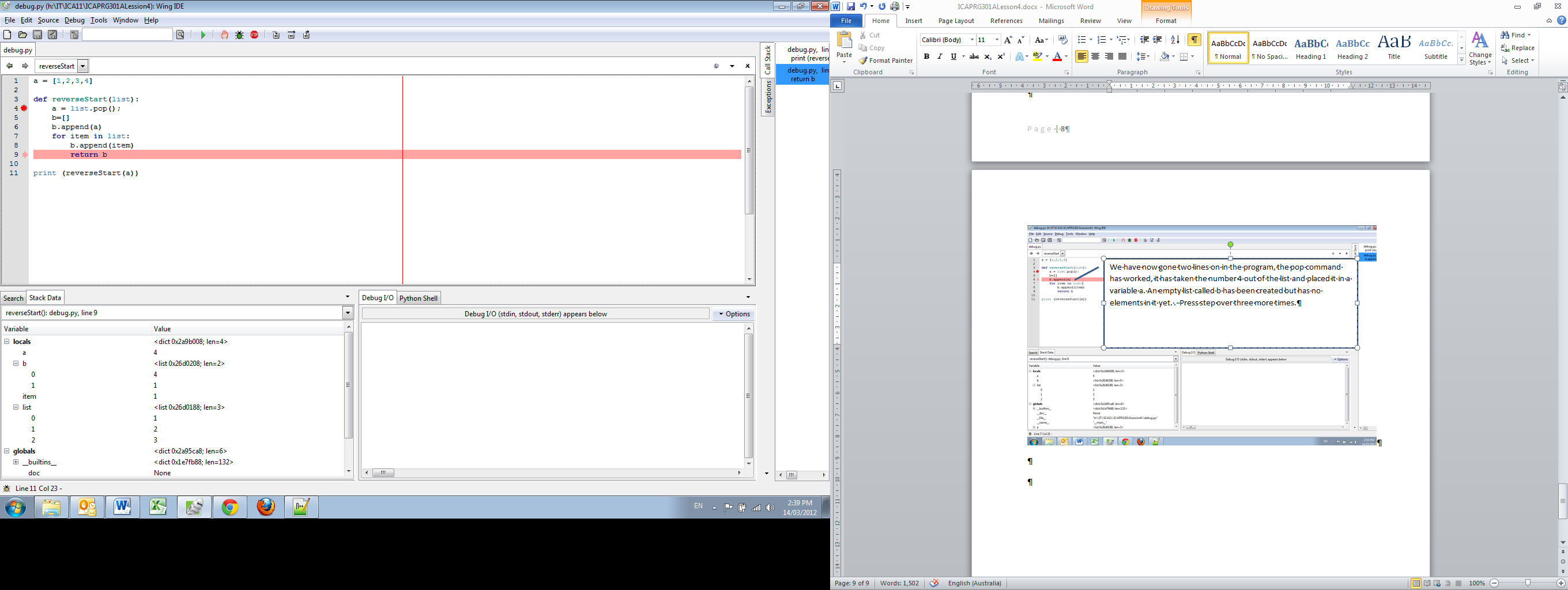
The program halts at the stop point and highlight the current line in red

These buttons are used to step through your code line by line. The best button to use is the middle one (Step over) but you can try them out to see what they do. Press this button twice.

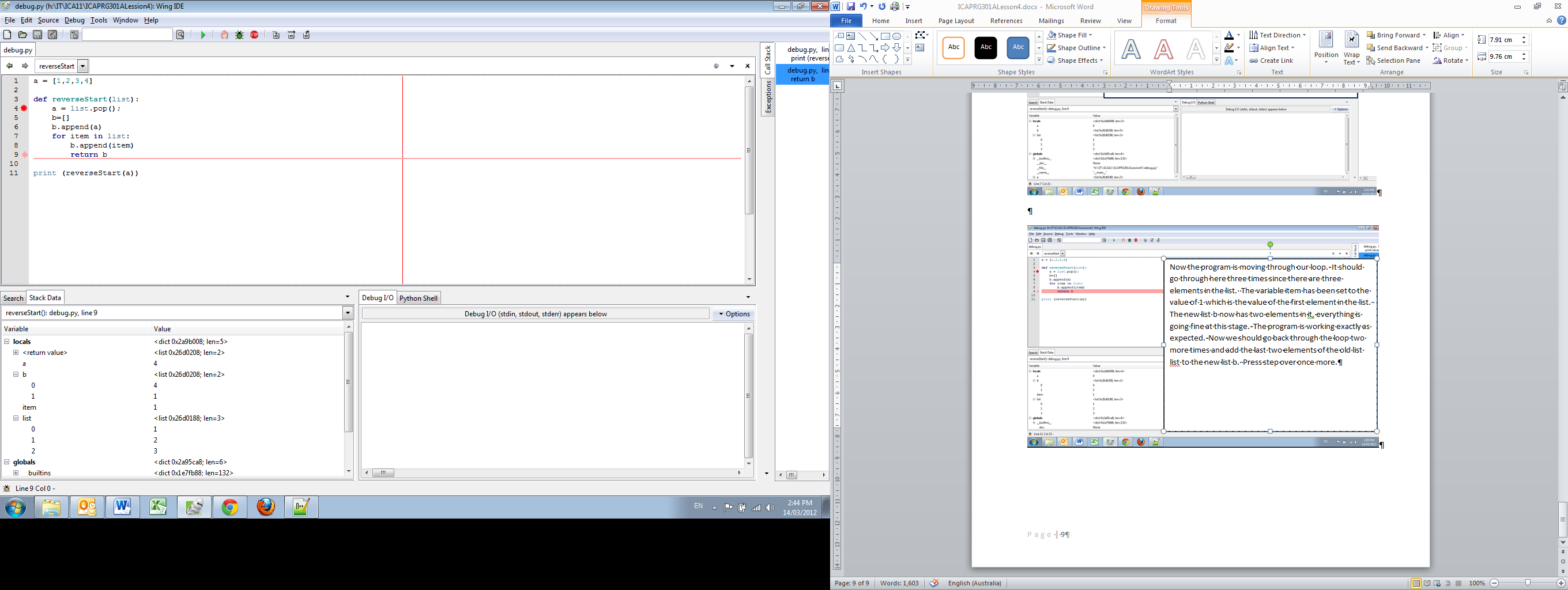
The current variables are displayed (we are only concerned with the local variables). List has been defined at the start of function. The current line will define a variable a, but since that line has not been run yet it does not appear. It might help to close down the global variables so they don’t confuse you



We have now gone two lines on in the program, the pop command has worked, it has taken the number 4 out of the list and placed it in a variable a. An empty list called b has been created but has no elements in it yet. . Press step over three more times.



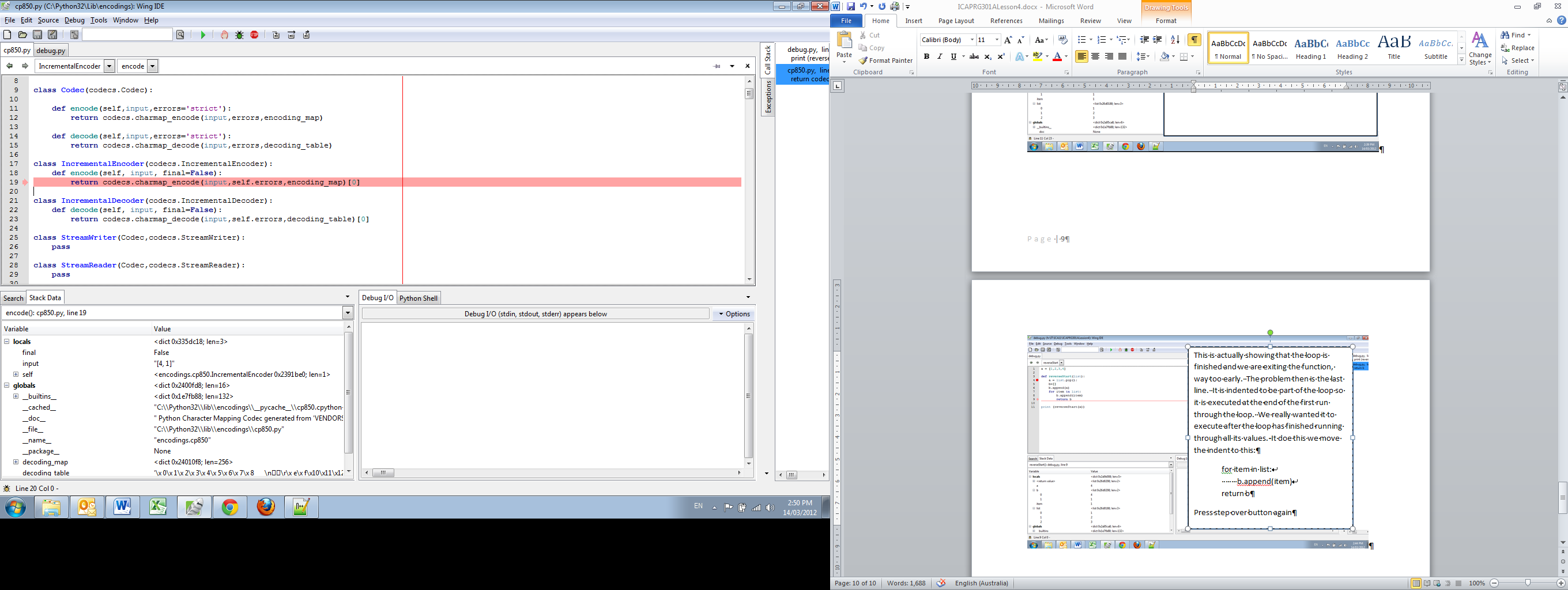
Now the program is moving through our loop. It should go through here three times since there are three elements in the list. The variable item has been set to the value of 1 which is the value of the first element in the list. The new list b now has two elements in it, everything is going fine at this stage. The program is working exactly as expected. Now we should go back through the loop two more times and add the last two elements of the old list to the new list b. Press step over once more.



This is actually showing that the loop is finished and we are exiting the function, way too early. The problem then is the last line. It is indented to be part of the loop so it is executed at the end of the first run through the loop. We really wanted it to execute after the loop has finished running through all its values. To do this we move the indent to this:

for item in list:  
 b.append(item)  
return b

Press step over button again



WHAT JUST HAPPENED, WHERE’S YOUR CODE.

Don’t panic, Wing is still stepping through Python code, however it has now started working through part of its own code that is run as part of the print statement. Debuggers don’t distinguish between your code and library code and will follow the code wherever it goes. Your code is still there (in the tab) the current code it is stepping through has been written by the authors of Python. If you keep pressing the step over button you will get back to your code soon. In the mean time it is interesting to see how good Python programmers actually structure their code.

This is how the debugger works. It is a good tool for getting rid of some bugs in your code.