FACTFILE: GCSE DIGITAL TECHNOLOGY

Unit 5

DIGITAL DEVELOPMENT PRACTICE

Rewarding Learning

Building a Solution

Learning Outcomes

Students should be able to:

use the following features of an integrated development environment (IDE) to support the creation of a solution from a structured design:

- code editor;
- simple debugging tools;
- compiler;
- error diagnostics;
- run-time environment, and;
- graphical user interface (GUI) where appropriate.

Content

- What is an IDE?
- Getting Started with IDLE
- Editing Python Code
- Error Diagnostics
- IDLE's Debugger

What is an IDE?

An integrated development environment (IDE) is a software application that is designed to support the software developer. It provides a single environment and interface to access a range of common tools that might be used by a computer programmer.

Integrated Development Environment (IDE)

"[An integrated development environment] is software that performs the various stages of software design and implementation in a single integrated system. It will usually include facilities for project management, design, graphical design, programming, testing and producing documentation."

BCS Glossary of Computing, 13th edition, p 238.

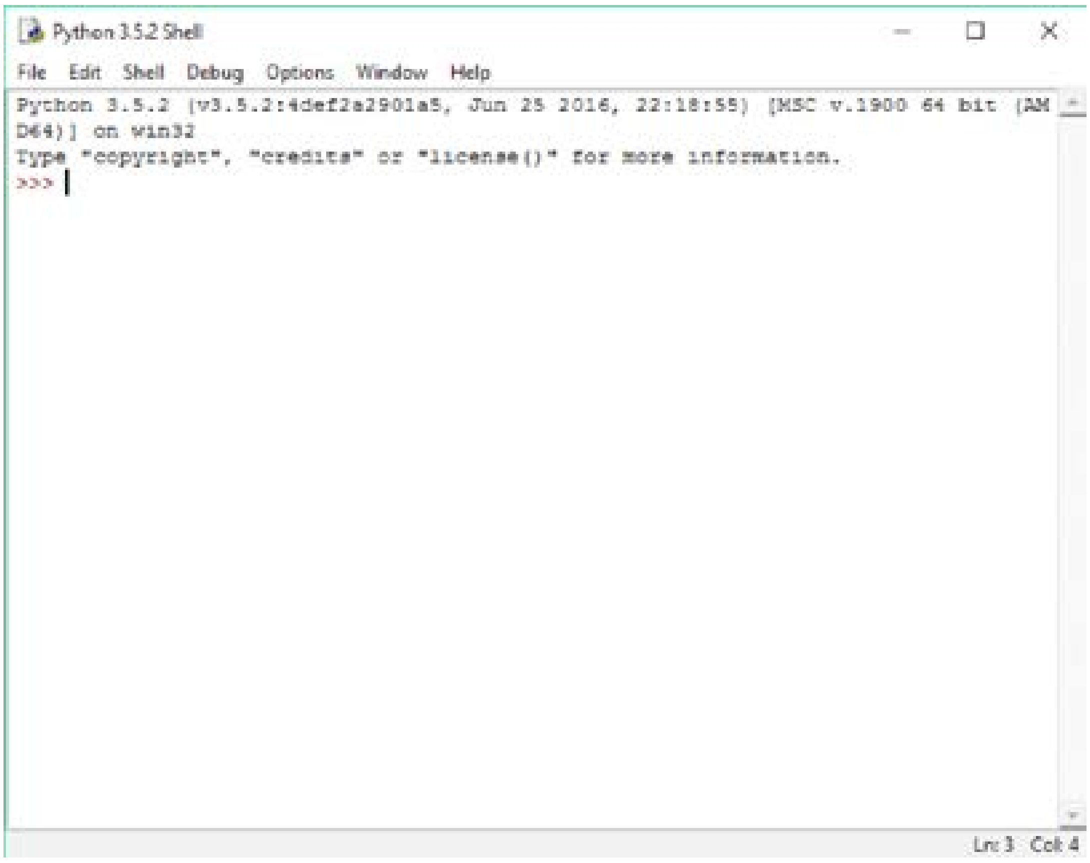
Getting Started with IDLE

IDLE – which stands for **I**ntegrated **D**eve**L**opment **E**nvironment – is the IDE that is bundled with the Python. It provides many of the most common features that you would expect from an IDE.

If you use IDLE most of your interaction will occur within these two windows:

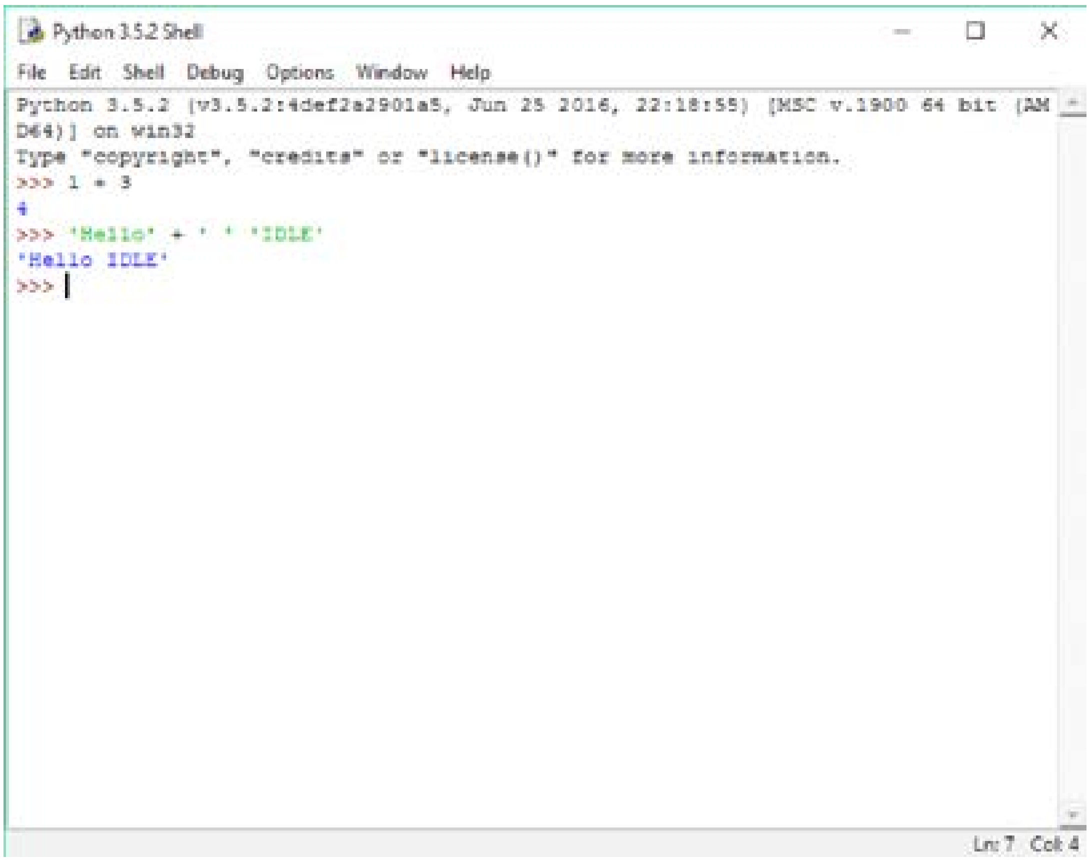
- The *shell window* is where the where you will normally interact with your program when it is running. This is the default for any input and output that is within the control of your program.
- The *editor window* is where you type your program code. From here you can save your program or instruct Python to execute it.

When you launch IDLE the *shell window* will open first – this is what it looks like.



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

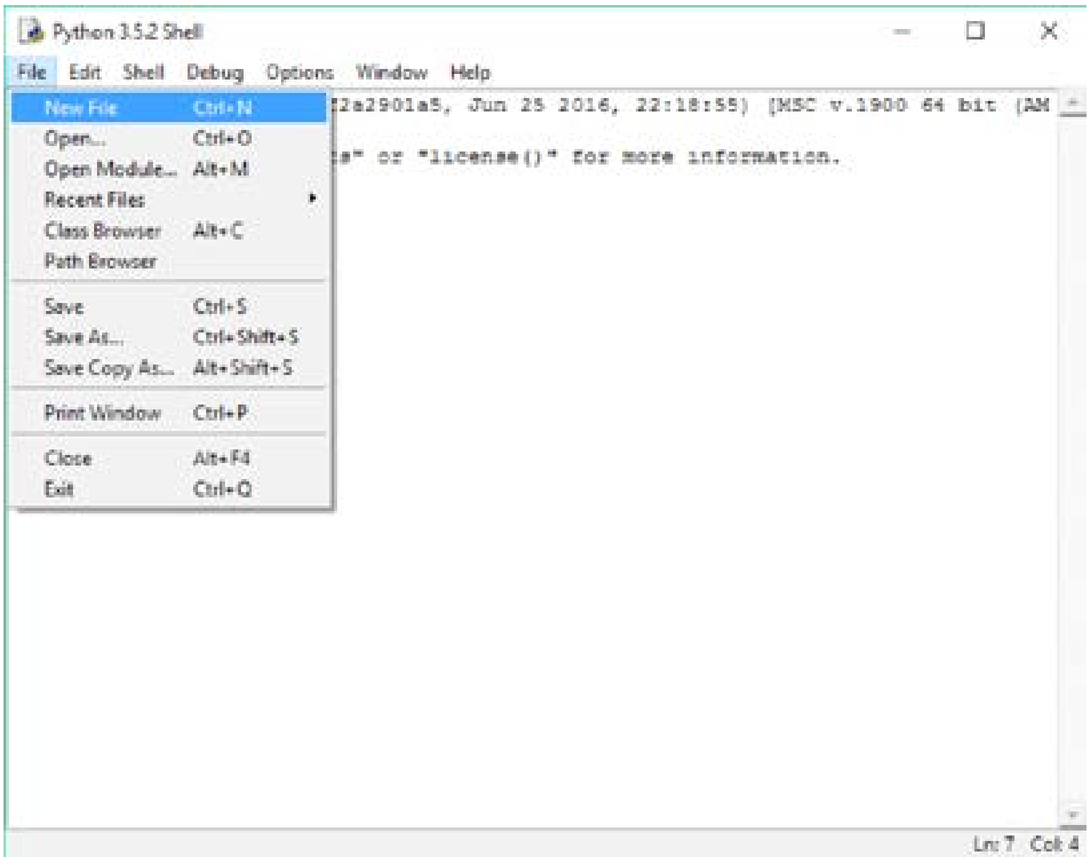
You will see the Python prompt displayed and you can interact with this in the normal way.



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 1 + 3
4
>>> 'Hello' + ' ' + 'IDLE'
'Hello IDLE'
>>> 
```

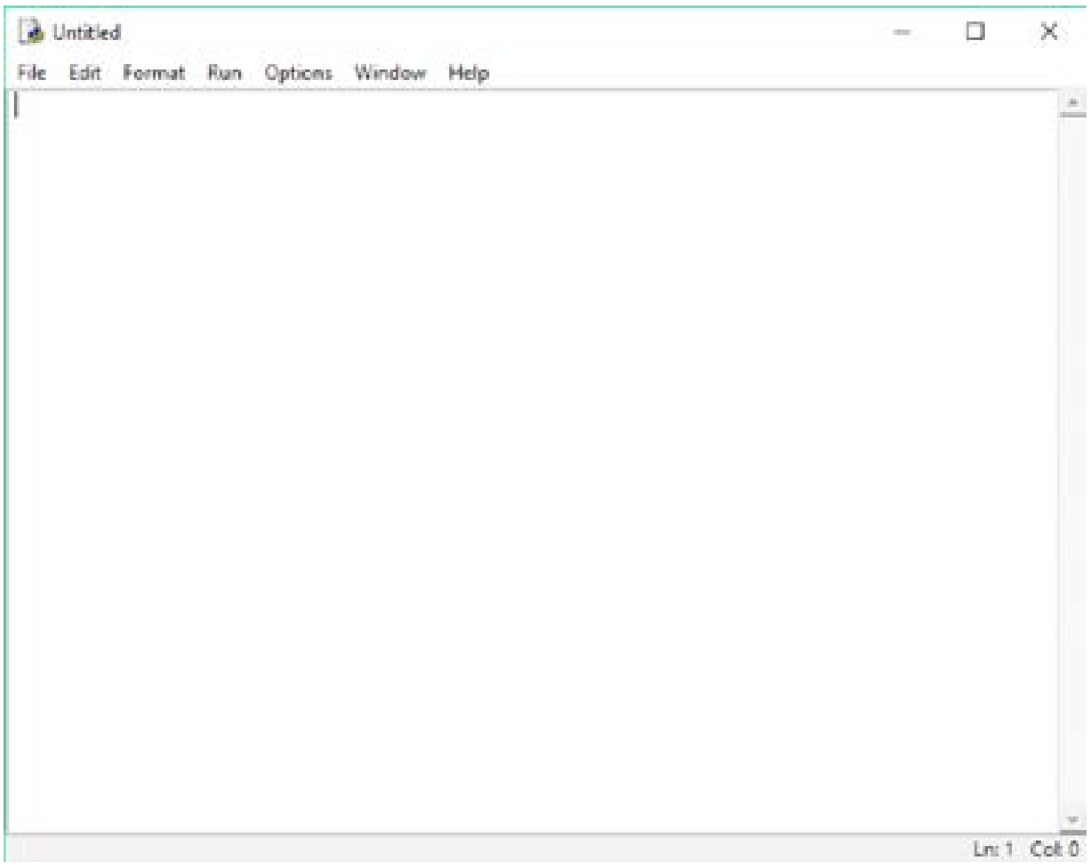
The menus along the top of the shell window allow you to access a range of IDLE's functionality. For example:

- The *file menu* enables you to create new files, open existing files and save files.
- The *debug menu* gives you access to a range of features that will help you to debug your programs.
- The options menu allows you to configure a range of IDLE options, including fonts and colours.



```
f2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
ts" or "license()" for more information.
```

Upon creating a new file using the file menu, the *edit window* will appear – this is what it looks like.

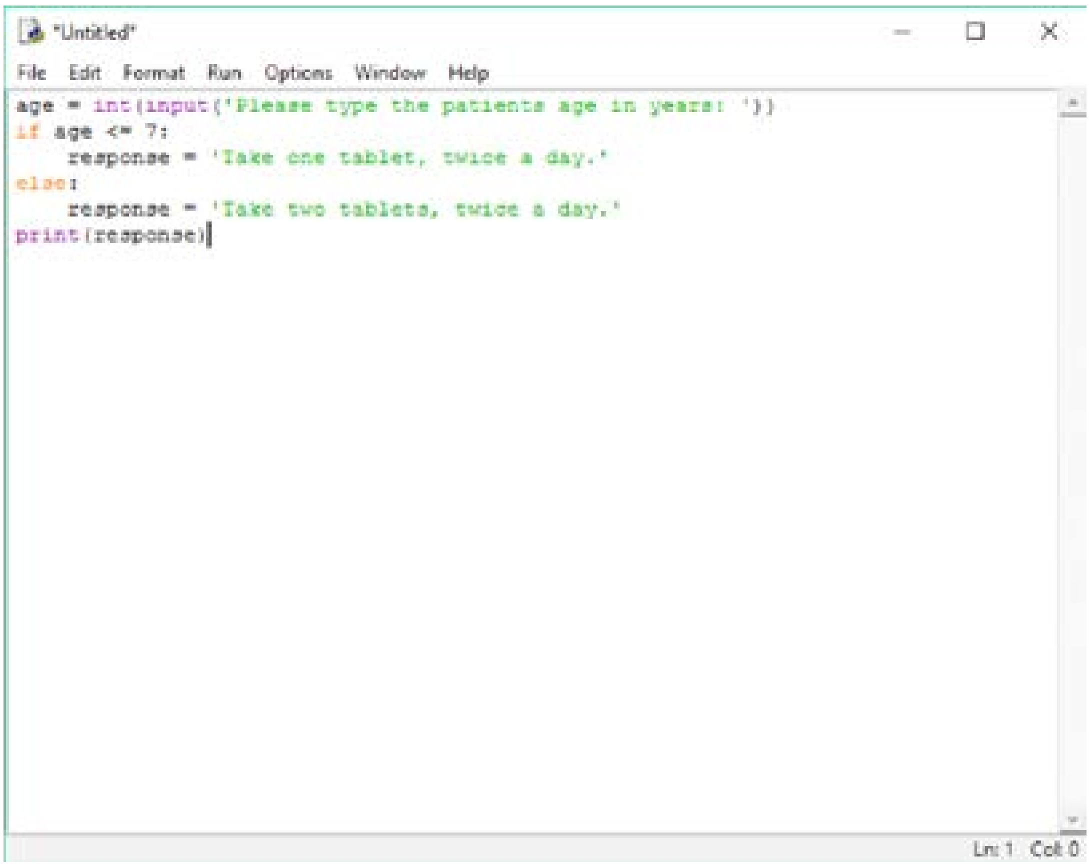


Some of the menu items from the shell window are repeated in the edit window – the *format* and *run* menus, however, are new.

- The *format menu* enables you to perform a range of formatting tasks, such as applying indentation.
- The *run menu* enables you to instruct Python to execute your program.

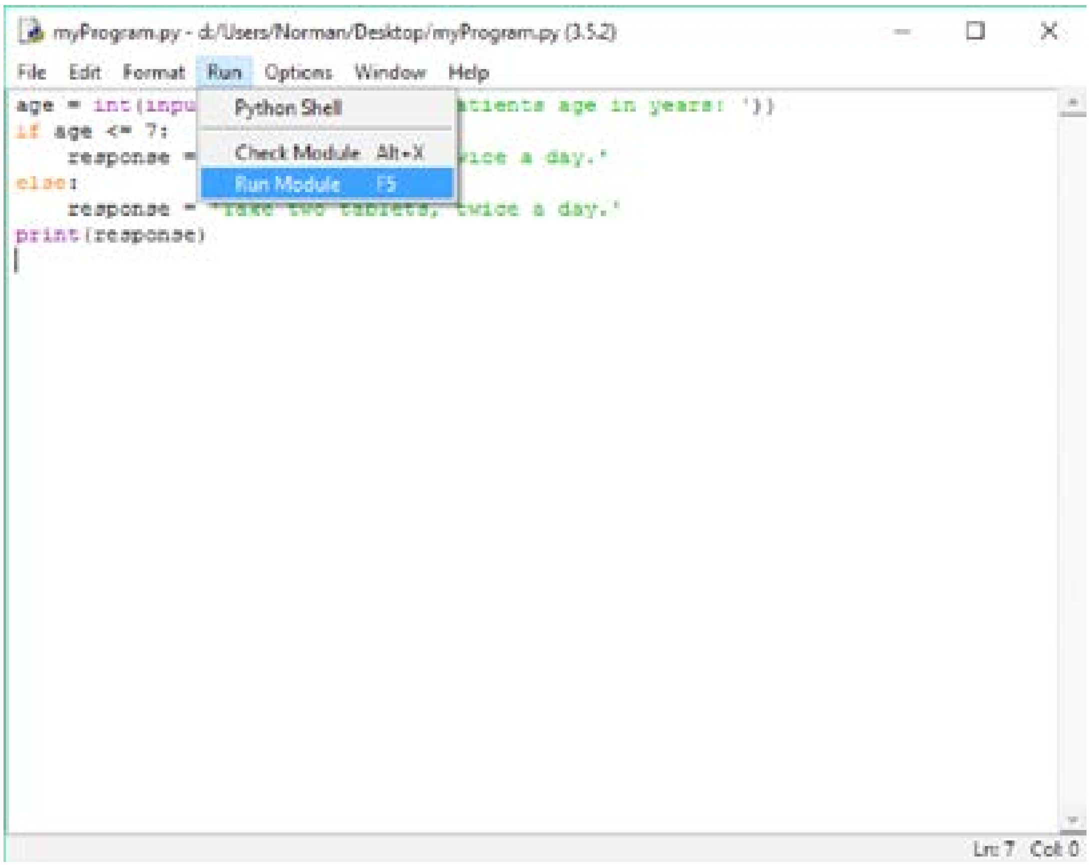
Editing Python Code

You can enter and edit Python code into the edit window. As you type, IDLE adds colour highlighting to your code. You can see below that, for example, string values are displayed in green; function names are displayed in purple, and reserved words such as 'if' and 'else' are displayed in orange. This makes it much easier for the programmer to read and understand her code.



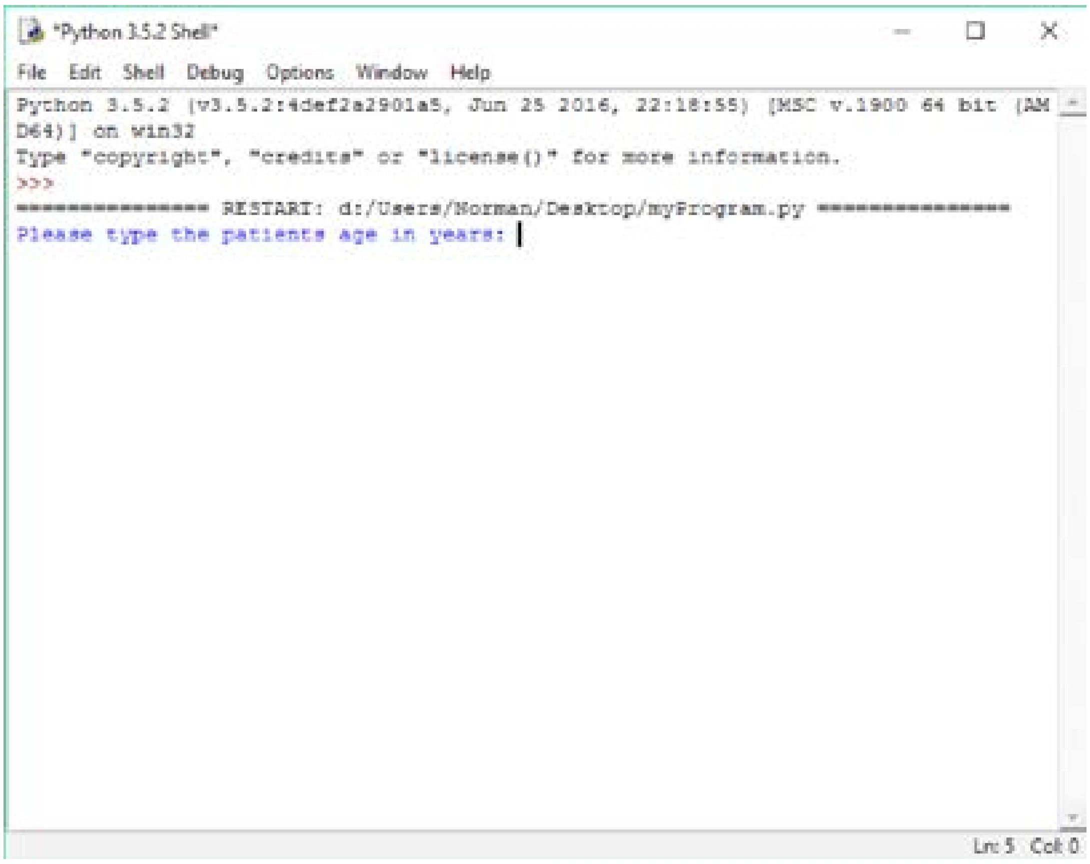
```
age = int(input('Please type the patients age in years: '))
if age <= 7:
    response = 'Take one tablet, twice a day.'
else:
    response = 'Take two tablets, twice a day.'
print(response)
```

Before your program can be executed, it must be saved – this is done using the *save as* option in file menu. Once it is saved it can be executed by selecting the *run module* option in the *run* menu.

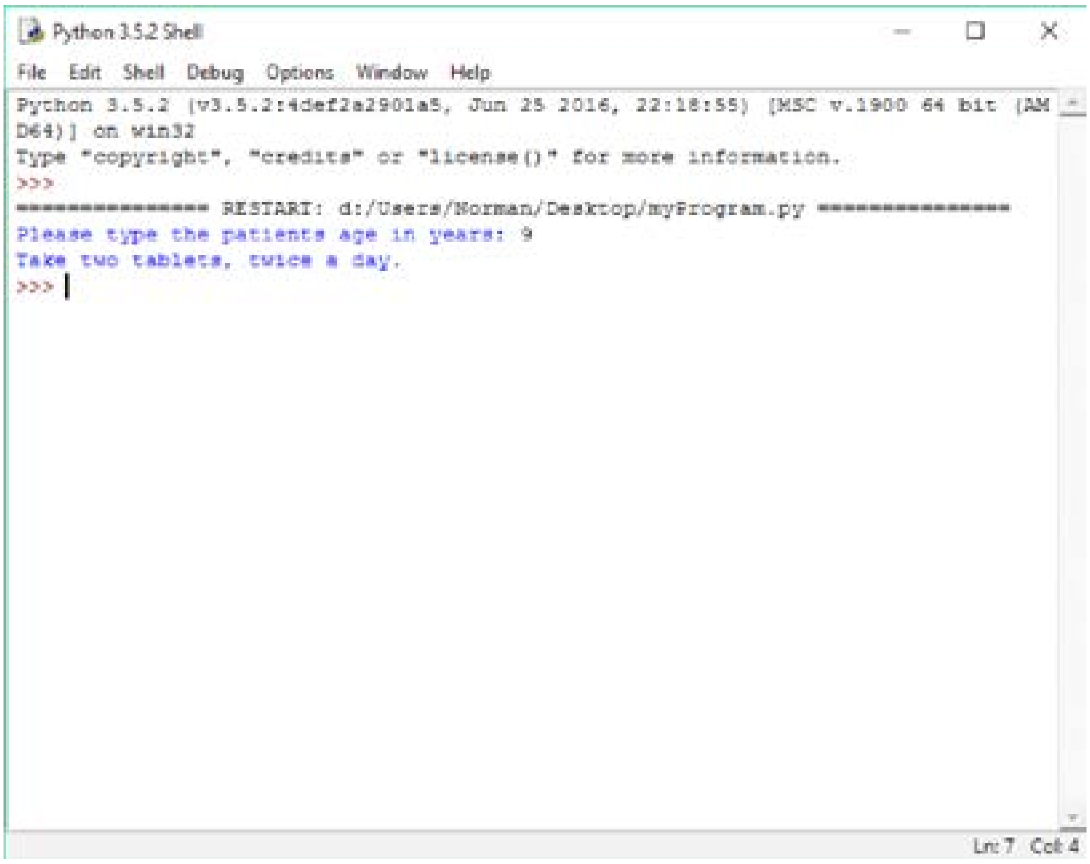

myProgram.py - d:/Users/Norman/Desktop/myProgram.py (3.5.2)
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5
Ln: 7 Col: 0

```
age = int(inpu                        atients age in years: '))
if age <= 7:
    response =                        wice a day.'
else:
    response = 'Take two tablets, twice a day.'
print(response)
```

When the program is executed, an interaction takes place in the shell window.



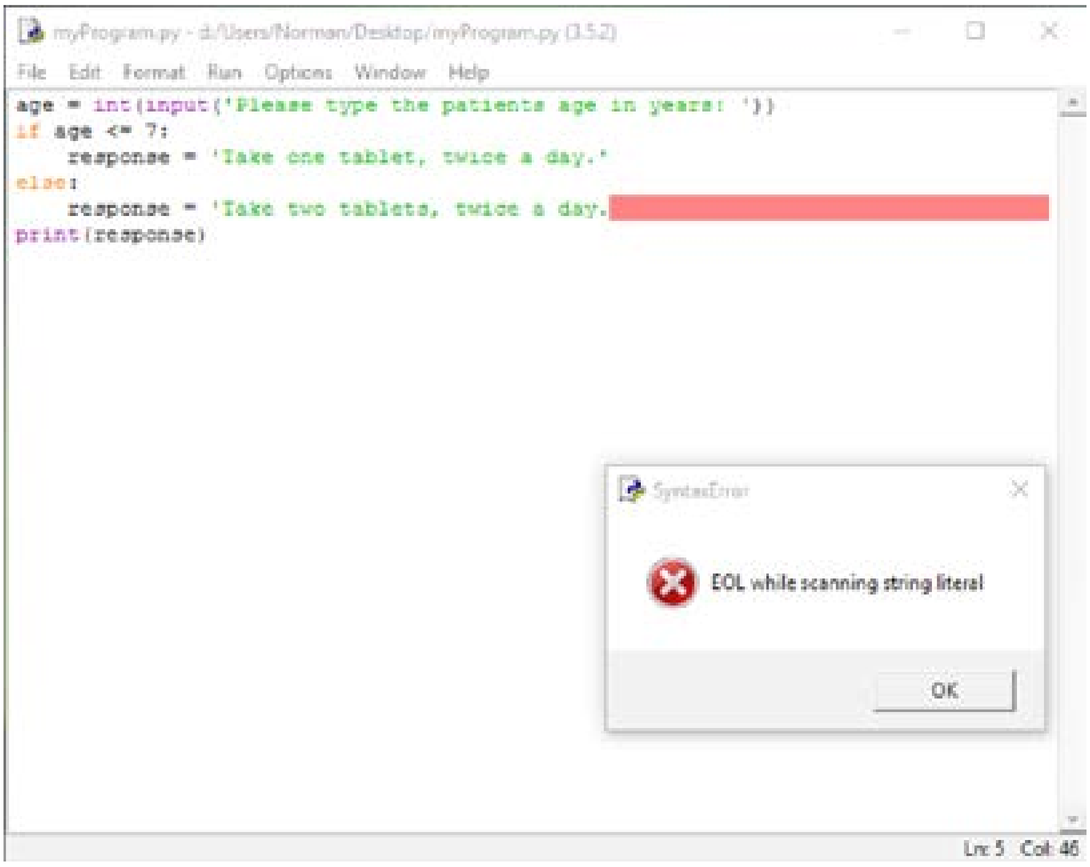
```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
=============== RESTART: d:/Users/Norman/Desktop/myProgram.py ===============
Please type the patients age in years: 
```



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: d:/Users/Norman/Desktop/myProgram.py ================
Please type the patients age in years: 9
Take two tablets, twice a day.
>>> 
```

Error Diagnostics

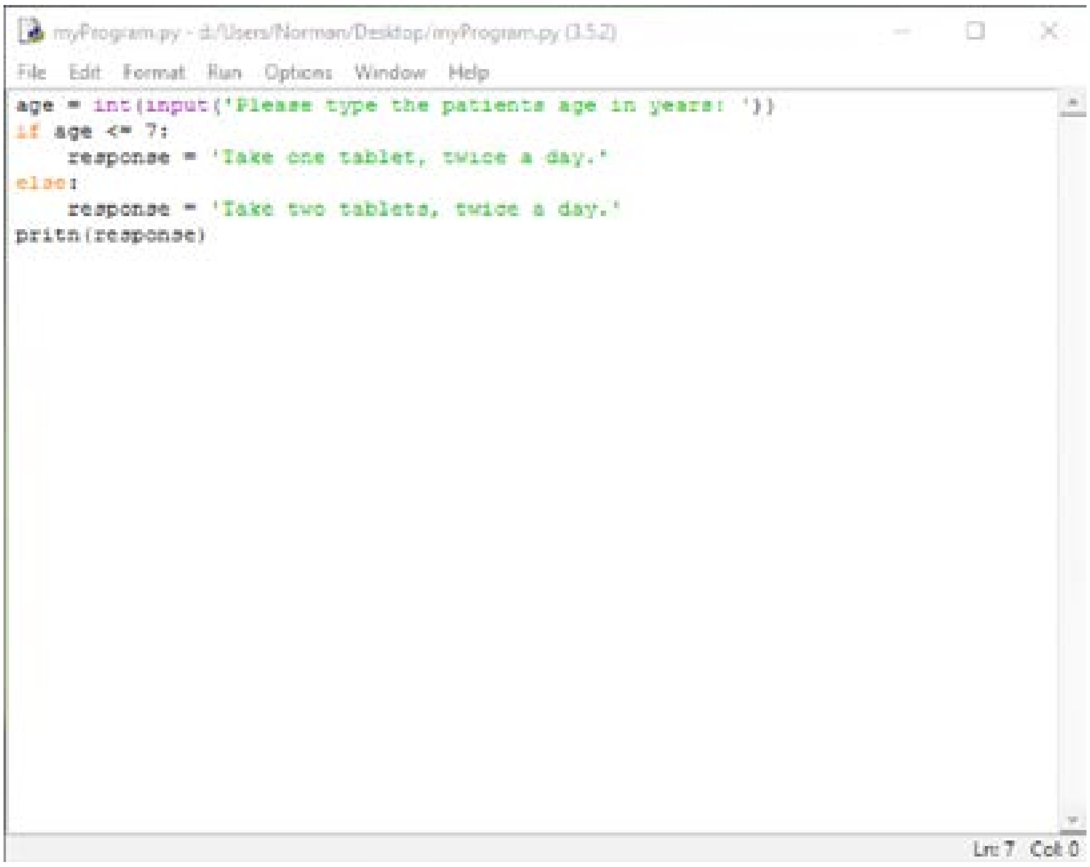
One of the most important tasks of an IDE is to help the programmer to find and correct any errors that might be in the code. In the following example a quotation mark is missing at the end of the highlighted string. When the run menu is used to try to execute the code, an alert dialog box informs the user of the problem ('EOL' means end of line).


myProgram.py - d:/Users/Norman/Desktop/myProgram.py (3.5.2)
File Edit Format Run Options Window Help
SyntaxError
OK
Ln: 5 Col: 46

```
age = int(input('Please type the patients age in years: '))
if age <= 7:
    response = 'Take one tablet, twice a day.'
else:
    response = 'Take two tablets, twice a day.
print(response)
```

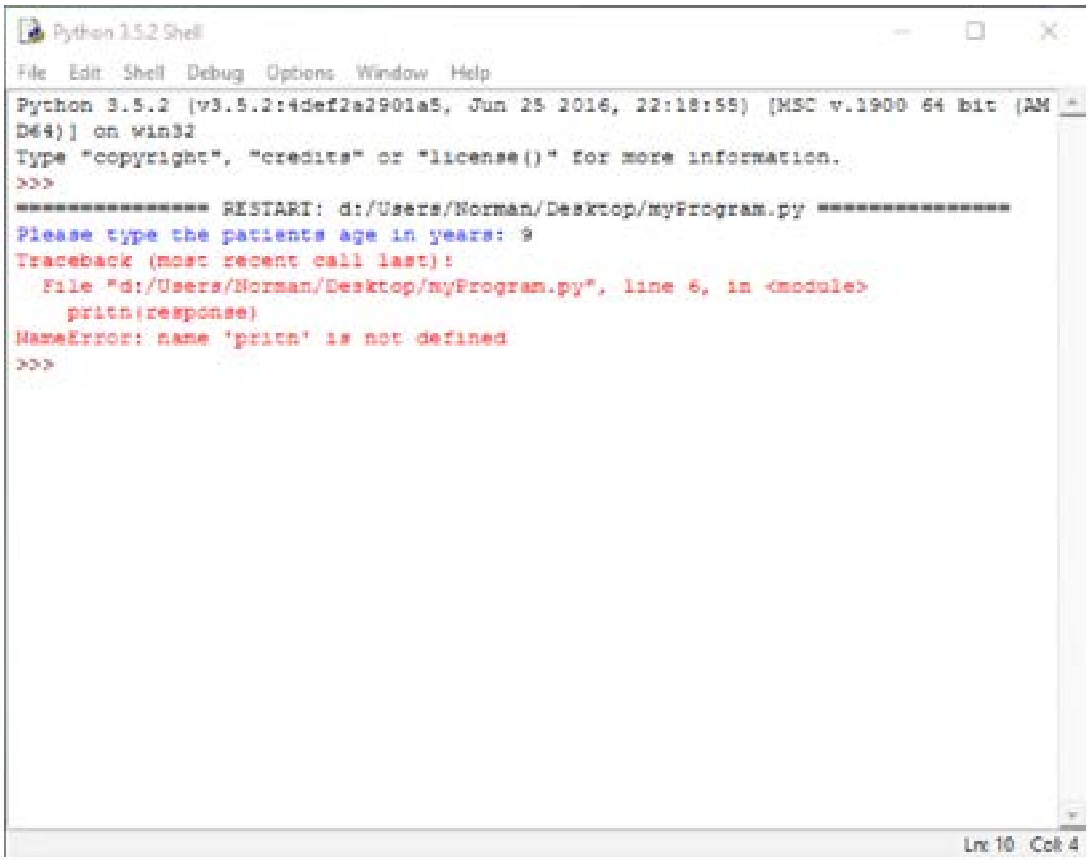
EOL while scanning string literal

Some errors cannot be diagnosed until program execution begins. For these *run-time* errors a diagnostic message is normally printed in the shell window. The following example shows a short Python program containing an error.


myProgram.py - d:/Users/Norman/Desktop/myProgram.py (3.5.2)
File Edit Format Run Options Window Help
Ln: 7 Col: 0

```
age = int(input('Please type the patients age in years: '))
if age <= 7:
    response = 'Take one tablet, twice a day.'
else:
    response = 'Take two tablets, twice a day.'
pritn(response)
```

The following diagnostic error report is produced.



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: d:/Users/Norman/Desktop/myProgram.py ================
Please type the patients age in years: 9
Traceback (most recent call last):
  File "d:/Users/Norman/Desktop/myProgram.py", line 6, in <module>
    pritn(response)
NameError: name 'pritn' is not defined
>>>
```

Question: What is the error and how can it be fixed?

Exercise: The following Fact File takes code that contains bugs and diagnoses and corrects them.

- U4FF7: Error Handling
  Section: Fun with Debugging

Type the *buggy* code into the IDLE edit window and see how the errors are reported. Correct the errors.

IDLE’s Debugger

Debugging is the process of finding and correcting any errors that might be in a program. Often this can be accomplished by closely examining the code and carefully observing the behaviour of the program as it executes – you can see an example of this process in U4FF7. Sometimes it also helps to add some lines of temporary code to, for example, display the values of important variables at key points in the execution. For more complex programs, however, a special piece of software – known as a *debugger* – is sometimes used.

Debugger

“[Debuggers] are programs that provide a range of facilities enabling the programmer to investigate the conditions when errors occur.”

BCS Glossary of Computing, 13th edition, p 300.

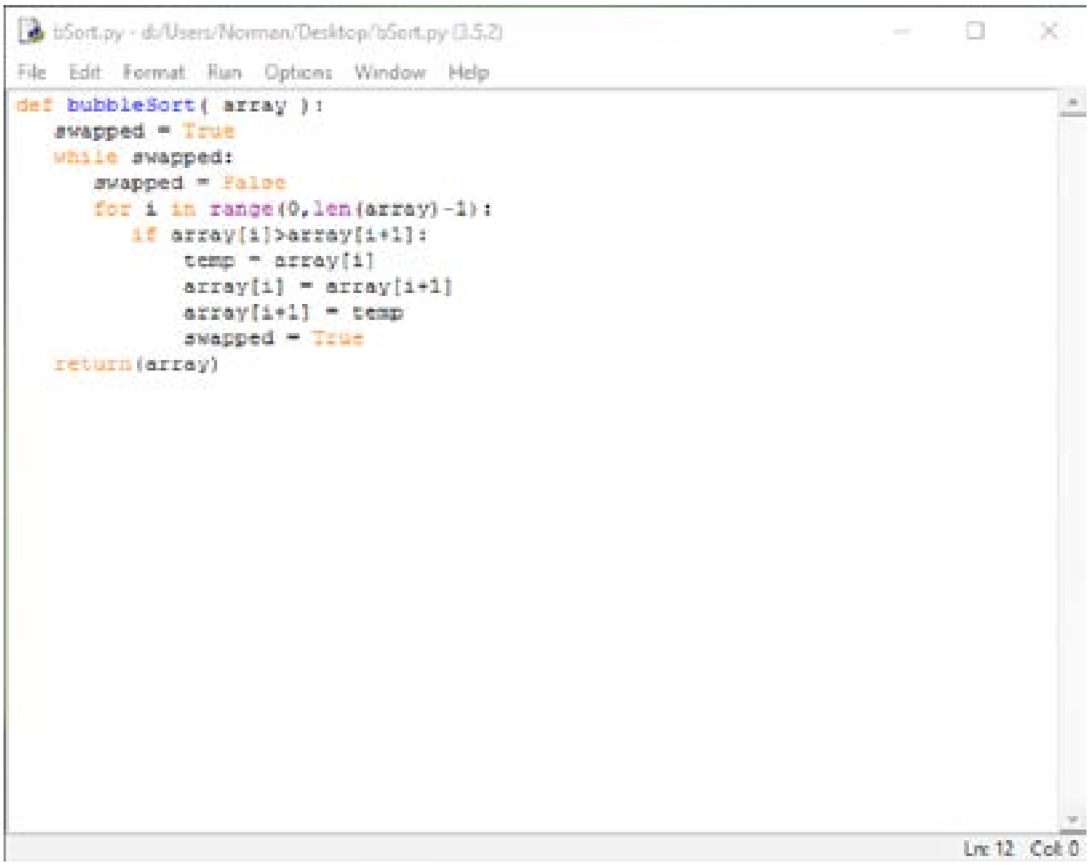
IDEs typically come bundled with integrated debuggers, and IDLE is no exception.

The IDLE debugger enables the programmer to control the execution of her program. Typically she will instruct Python to execute the program line-by-line, reporting the values of variables at each step. This is intended to help the programmer understand what is happening as the program executes, and so help her find the source of any errors.

The bubble sort program that was explained in the following fact file is used below to demonstrate the IDLE debugger.

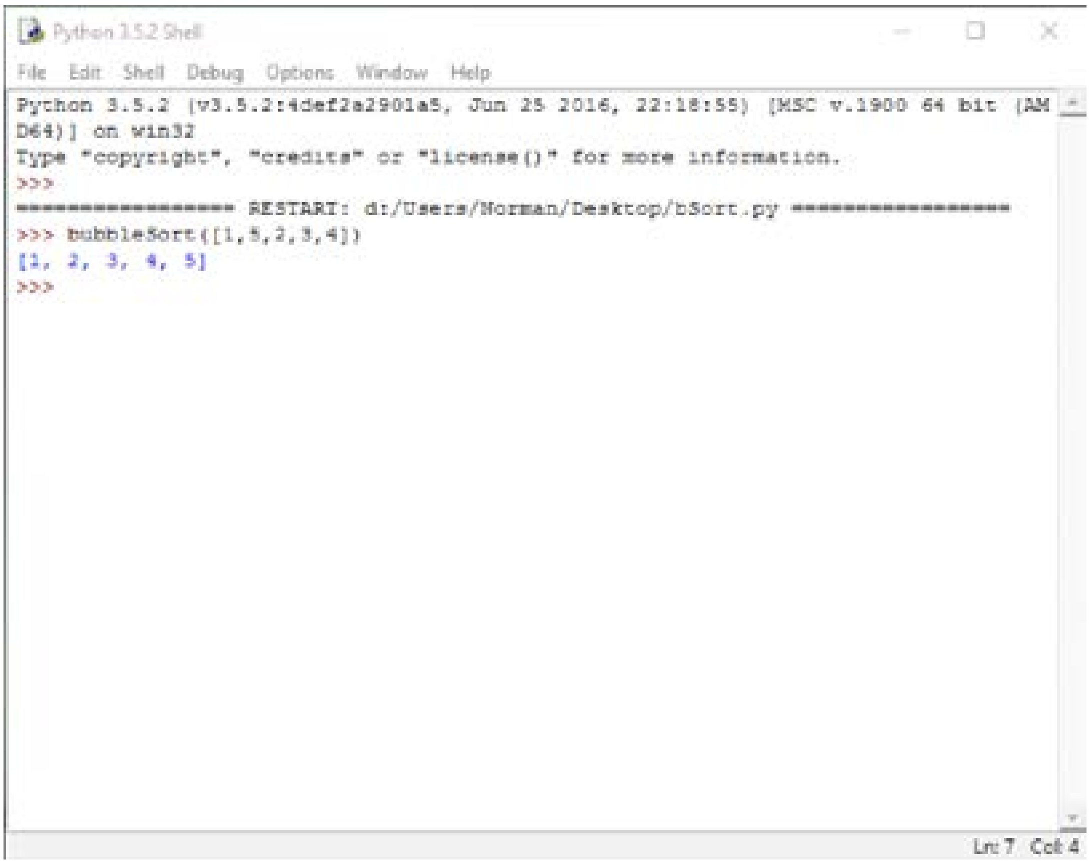
- U4FF6: Data Programming Constructs 2

Here is the program.


bSort.py - d:/Users/Norman/Desktop/bSort.py (3.5.2)
File Edit Format Run Options Window Help
Ln: 12 Col: 0

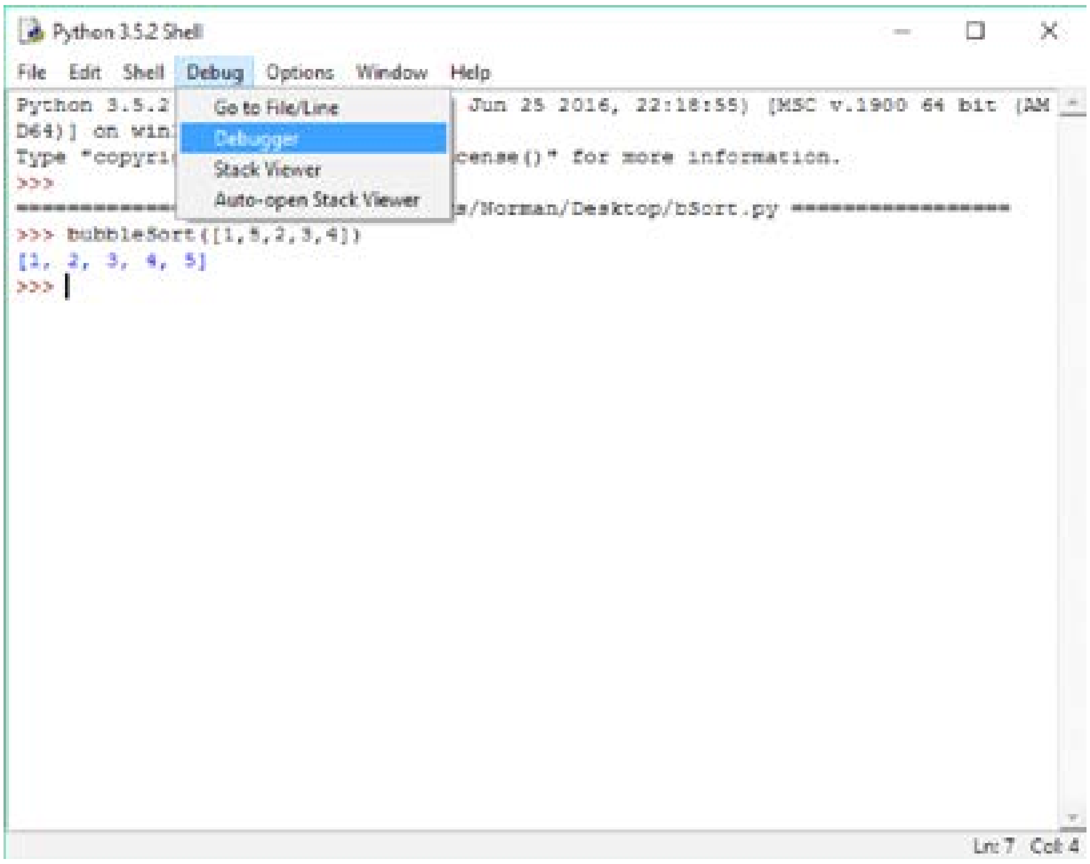
```
def bubbleSort( array ):
    swapped = True
    while swapped:
        swapped = False
        for i in range(0,len(array)-1):
            if array[i]>array[i+1]:
                temp = array[i]
                array[i] = array[i+1]
                array[i+1] = temp
                swapped = True
    return(array)
```

And here is what happens when it is executed without the debugger.



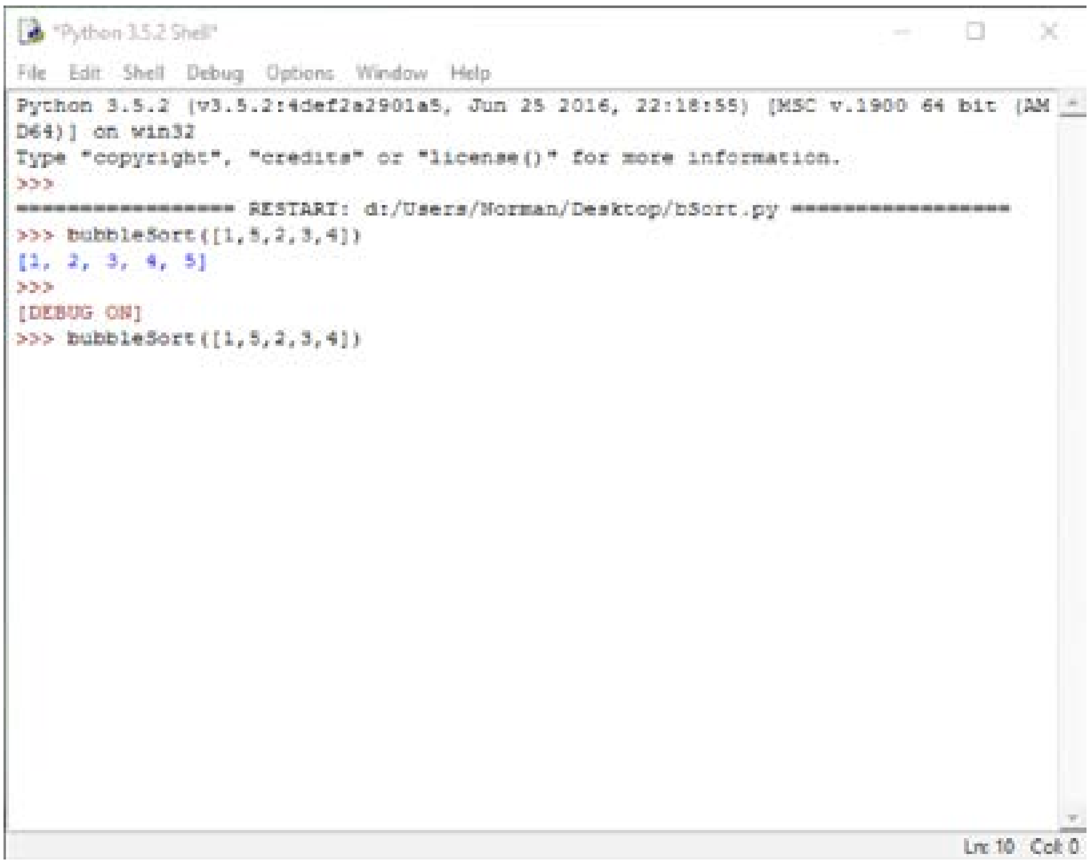
```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: d:/Users/Norman/Desktop/bSort.py ==================
>>> bubbleSort([1,5,2,3,4])
[1, 2, 3, 4, 5]
>>>
```

To see the effect of the debugger, we first switch on debugging mode using the *debug* menu.



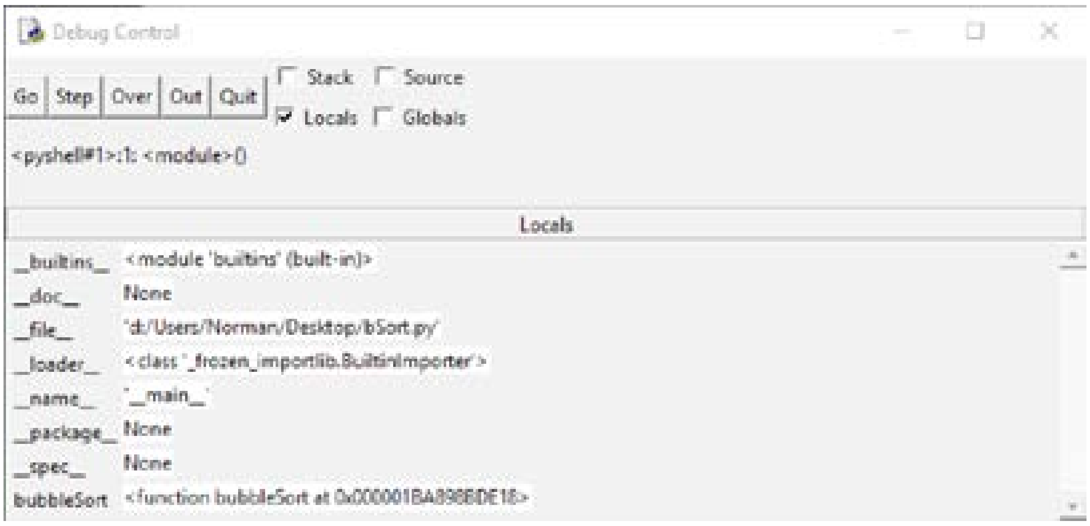
```
Python 3.5.2                  Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win
Type "copyri                  cense()" for more information.
>>>
=============                 s/Norman/Desktop/bSort.py ==================
>>> bubbleSort([1,5,2,3,4])
[1, 2, 3, 4, 5]
>>> 
```

When the bubbleSort function is invoked, again execution begins under the control of the debugger, which, in turn, is under the control of the user.



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: d:/Users/Norman/Desktop/bSort.py ==================
>>> bubbleSort([1,5,2,3,4])
[1, 2, 3, 4, 5]
>>>
[DEBUG ON]
>>> bubbleSort([1,5,2,3,4])
```

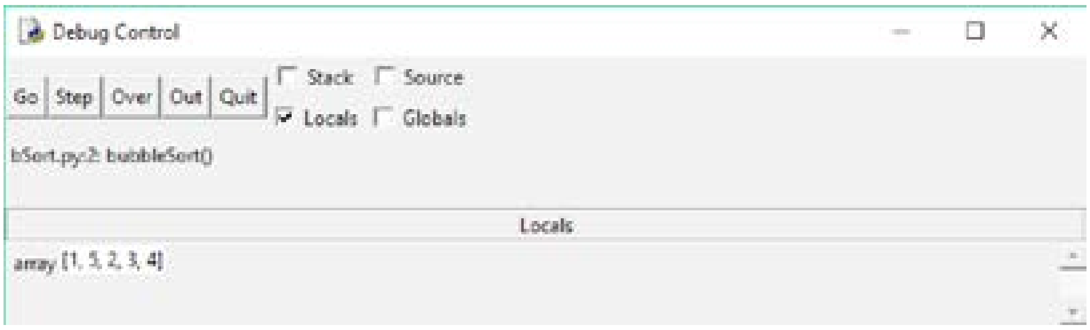
The debugger stops the program execution to await instructions from the user. This is what the debugger window looks like.



```
<pyshell#1>:1: <module>()
```

Locals	
__builtins__	<module 'builtins' (built-in)>
__doc__	None
__file__	'd:/Users/Norman/Desktop/bSort.py'
__loader__	<class '_frozen_importlib.BuiltinImporter'>
__name__	'__main__'
__package__	None
__spec__	None
bubbleSort	<function bubbleSort at 0x000001BA898BDE18>

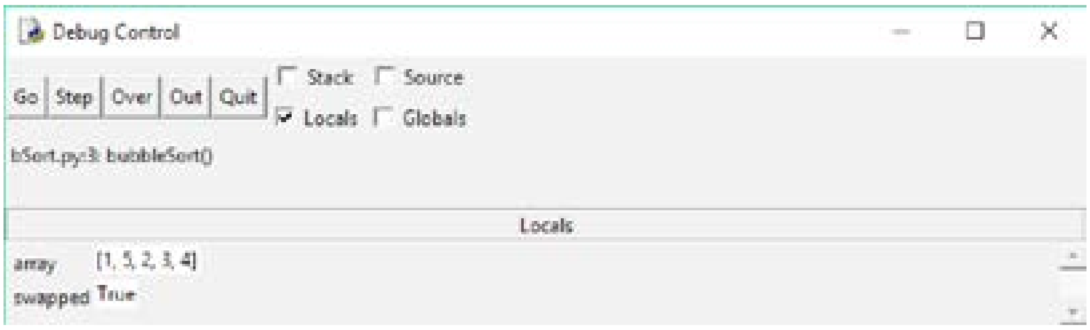
By clicking on the *Step* button, the user instructs the debugger to execute the next line of code.



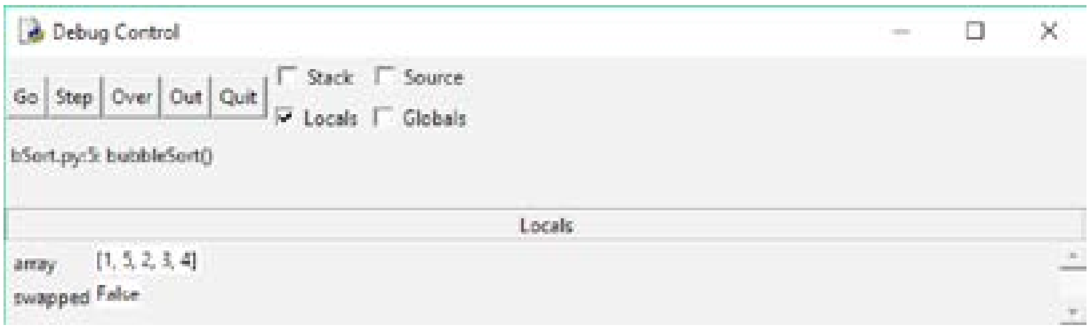
```
bSort.py:2: bubbleSort()

Locals
array [1, 5, 2, 3, 4]
```

At this point the value of the variable *array* is reported – [1,5,2,3,4].

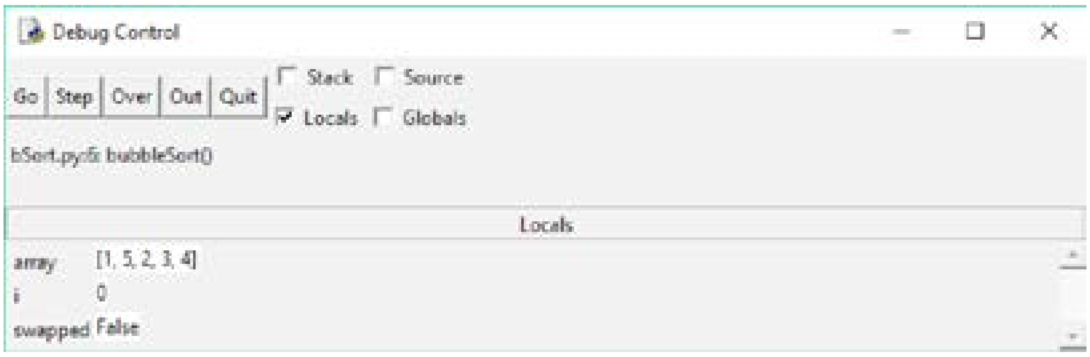
By repeatedly clicking on the *Step* button, the programmer steps through the execution and can see how the variable values change at each step.



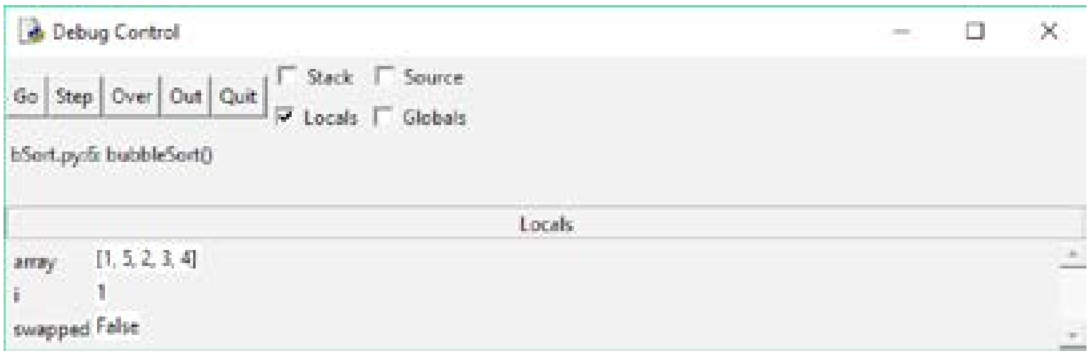
```
bSort.py:3: bubbleSort()

Locals
array     [1, 5, 2, 3, 4]
swapped True
```



```
bSort.py:5: bubbleSort()

Locals
array     [1, 5, 2, 3, 4]
swapped False
```

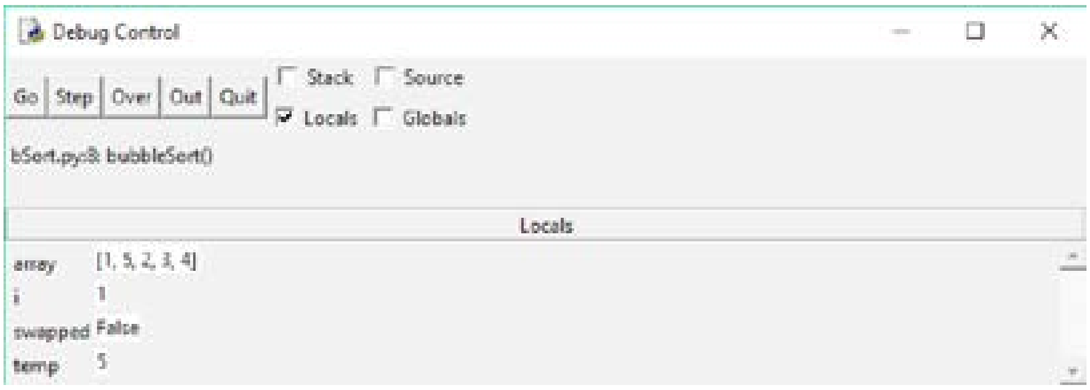


```
bSort.py:6: bubbleSort()

Locals
array     [1, 5, 2, 3, 4]
i         0
swapped False
```



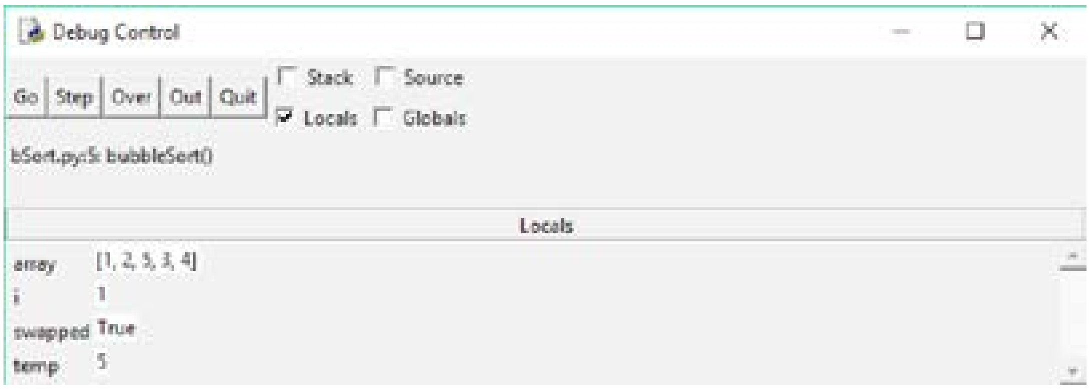
[ ] Stack [ ] Source
[✓] Locals [ ] Globals

```
bSort.py:6: bubbleSort()

Locals
array   [1, 5, 2, 3, 4]
i       1
swapped False
```



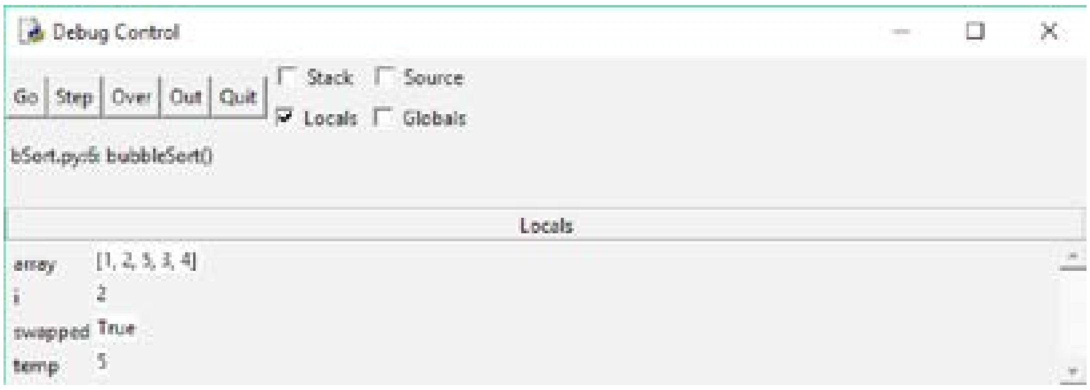
[ ] Stack [ ] Source
[✓] Locals [ ] Globals

```
bSort.py:8: bubbleSort()

Locals
array   [1, 5, 2, 3, 4]
i       1
swapped False
temp    5
```



[ ] Stack [ ] Source
[✓] Locals [ ] Globals

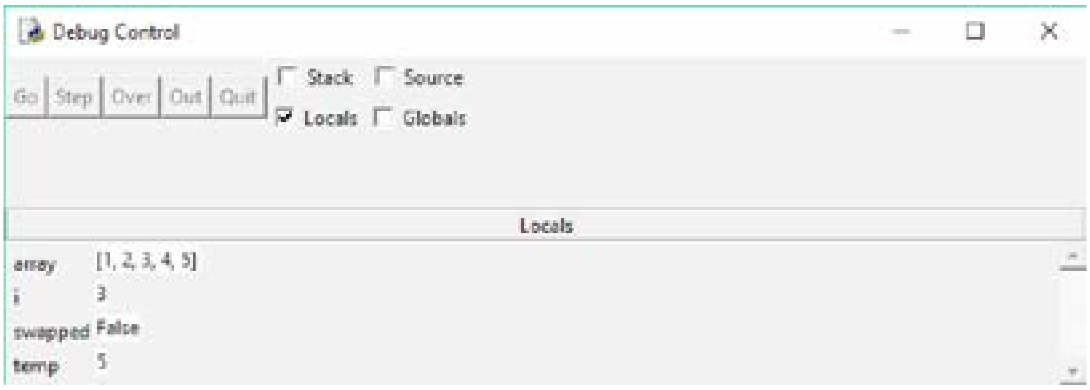
```
bSort.py:5: bubbleSort()

Locals
array   [1, 2, 5, 3, 4]
i       1
swapped True
temp    5
```



[ ] Stack [ ] Source
[✓] Locals [ ] Globals

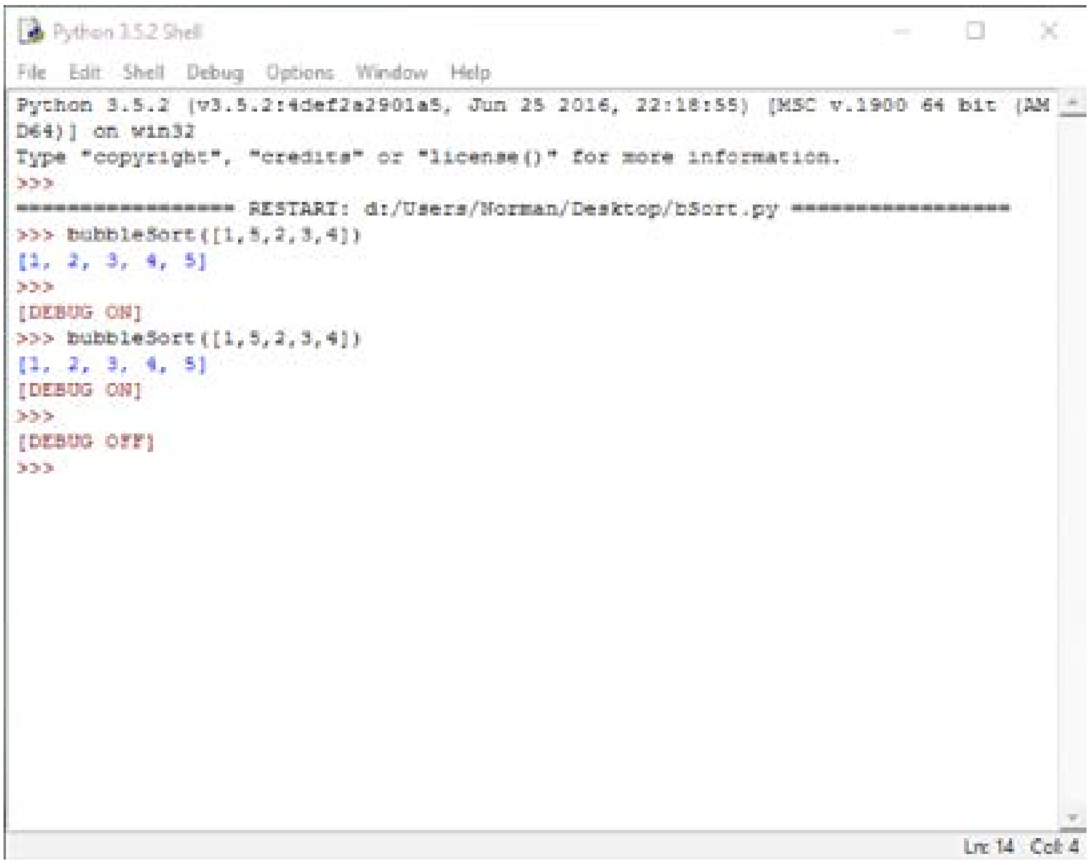
```
bSort.py:6: bubbleSort()

Locals
array   [1, 2, 5, 3, 4]
i       2
swapped True
temp    5
```

After many more steps (which are not shown) the debugger window finally looks like this.



```
array   [1, 2, 3, 4, 5]
i       3
swapped False
temp    5
```

Closing the debugging window results in debugging mode being switched off.



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: d:/Users/Norman/Desktop/bSort.py ==================
>>> bubbleSort([1,5,2,3,4])
[1, 2, 3, 4, 5]
>>>
[DEBUG ON]
>>> bubbleSort([1,5,2,3,4])
[1, 2, 3, 4, 5]
[DEBUG ON]
>>>
[DEBUG OFF]
>>>
```

Exercise: Copy the bubble sort code and run it in Python with the debugger switched on. Step through the execution and make a note on how the variable, *array*, changes at each step.

Question: Why do you think the bubble sort algorithm is so named?

While IDLE's debugger enables the programmer to do much more than we have seen here, you will find that simply stepping through your program's execution in the way that we have shown is very effective in helping you locate and correct program bugs.

Resources

- Python Documentation – Debugger:
  https://docs.python.org/3.3/library/pdb.html?highlight=step#pdbcommand-step
- Python Documentation – IDLE:
  https://docs.python.org/3.3/library/idle.html
- TutorialsPoint, Python 3 Tutorial:
  https://www.tutorialspoint.com/python3/index.htm