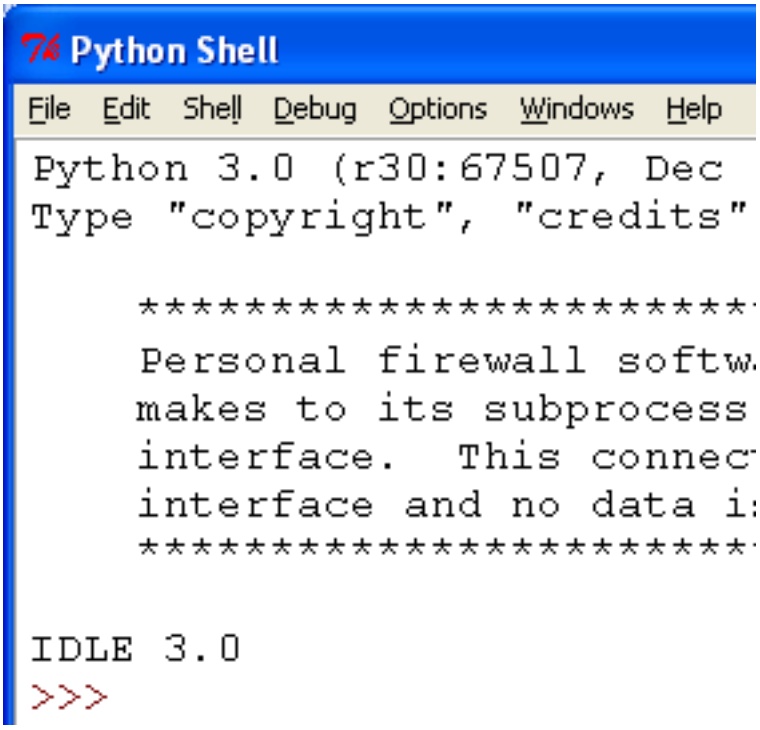

Look at the Python Shell. ...
In the Shell the last line should look like

```
>>>
```

The `>>>` is the *prompt*, telling you Idle is waiting for you to type something. Continuing on the same line enter

```
6+3
```

Be sure to end with the Enter key. After the Shell responds, you should see something like

```
>>> 6+3
9
>>>
```

The shell evaluates the line you entered, and prints the result. You see Python does arithmetic. At the end you see a further prompt `>>>` where you can enter your next line.... The result line, showing `9`, that is produced by the computer, does not start with ">>>".

1.3. Whirlwind Introduction To Types and Functions

Python directly recognizes a variety of types of data. Here are a few:

Numbers: 3, 6, -7, 1.25
Character strings: `'hello'`, `'The answer is: '`
Lists of objects of any type: `[1, 2, 3, 4]`, `['yes', 'no', 'maybe']`
A special datum meaning nothing: `None`

Python has large collection of built-in functions that operate on different kinds of data to produce all kinds of results. To make a function do its action, parentheses are required. These parentheses surround the parameter or parameters, as in a function in algebra class.

The general syntax to execute a function is

`functionName ( parameters )`

One function is called `type`, and it returns the type of any object. The Python Shell will evaluate functions. In the Shell the last line should look like

```
>>>
```

Continuing on the same line enter

```
type(7)
```

Always remember to end with the Enter key. After the Shell responds, you should see something like

```
>>> type(7)
<class 'int'>
>>>
```

In the result, `int` is short for integer. The work *class* is basically a synonym for type in Python. At the end you see a further prompt where you can enter your next line....

For the rest of this section, at the `>>>` prompt in the Python *Shell*, individually enter each line below that is set off in `typewriter` font. So next enter

```
type(1.25)
```

Note the name in the last result is `float`, not real or decimal, coming from the term "floating point", for reasons that will be explained later, in Section 1.14.1. Enter

```
type('hello')
```

In your last result you see another abbreviation: `str` rather than string. Enter

```
type([1, 2, 3])
```

Strings and lists are both sequences of parts (characters or elements). We can find the length of that sequence with another function with the abbreviated name `len`. Try both of the following, separately, in the *Shell*:

```
len([2, 4, 6])
len('abcd')
```

Some functions have no parameters, so nothing goes between the parentheses. For example, some types serve as no-parameter functions to create a simple value of their type. Try

```
list()
```

You see the way an empty list is displayed.

Functions may also take more than one parameter. Try

```
max(5, 11, 2)
```

Above, max is short for maximum.

Some of the names of types serve as conversion functions (where there is an obvious meaning for the conversion). Try each of the following, one at a time, in the *Shell*:

```
str(23)
int('125')
```

An often handy Shell feature: an earlier Shell line may to copied and edited by clicking anywhere in the previously displayed line and then pressing ENTER. For instance you should have entered several lines starting with `len`. click on any one, press ENTER, and edit the line for a different test.

1.4. Integer Arithmetic

1.4.1. Addition and Subtraction. We start with the integers and integer arithmetic, not because arithmetic is exciting, but because the symbolism should be mostly familiar. Of course arithmetic is important in many cases, but Python is probably more often used to manipulate text and other sorts of data, as in the sample program in Section 1.2.2.

Python understands numbers and standard arithmetic. *For the whole section on integer arithmetic*, where you see a set-off line in `typewriter` font, type individual lines at the >>> prompt in the Python *Shell*. Press Enter after each line to get Python to respond:

```
77
2 + 3
5 - 7
```

Python should evaluate and print back the value of each expression. Of course the first one does not require any calculation. It appears the shell just echoes back what you printed. Do note that the line with the value *produced* by the shell does not start with >>> and appears at the left margin. Hence you can distinguish what you type (after the ">>>" prompt) from what the computer responds.