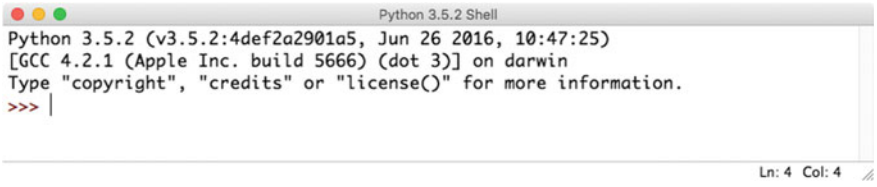
Chapter 2
The Python Shell—IDLE

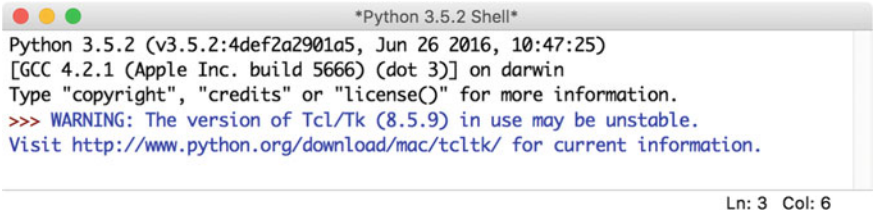
IDLE, the Integrated Development and Learning Environment, is a shell that comes bundled with the Python distribution on download.

Locate the Python 3.5 folder, open it, find IDLE and open it seeing a large empty window headed by



the final >>> prompt is where you can enter code interactively for the Python interpreter.

If on any platform you see the following warning message



it is safe to ignore this and type <**return**> or <**enter**> carrying on with the rest of the chapter. You will only be using 'Tcl/Tk' within the 'spyder' environment of Chap. 3. To clear this warning you have to install the latest Tcl/Tk 8.5 release which was 8.5.18.0 at the time of writing. Visit the ActiveState Tcl/Tk download page at http://www.activestate.com/activetcl/downloads and scroll down to

I. Gray, *Snake Charming—The Musical Python*,
DOI 10.1007/978-3-319-60660-6_2

DOWNLOAD TCL: OTHER PLATFORMS AND VERSIONS

Version	Windows (x86)	Windows (64-bit, x64)	Mac OS X (10.5+, x86_64/x86)	Linux (x86)	Linux (x86_64)
8.6.4.1	Windows Installer (EXE)	Windows Installer (EXE)	Mac Disk Image (DMG)	AS Package	AS Package
8.5.18.0	Windows Installer (EXE)	Windows Installer (EXE)	Mac Disk Image (DMG)	AS Package	AS Package

from where you can download the latest 8.5 version to install for your platform.

2.1 Basic Python Syntax

Python itself has an 'official' tutorial accessible as described in Chap. 1 or as in Chap. 3. The aim here is slightly less lofty and is intended as a 'get you started' subset of Python. More advanced concepts such as functions or datatypes such as lists will be introduced in the projects as and when required.

2.1.1 Comments

Comments to be ignored by the interpreter are introduced by the # (hash) character (<**option**> or <**alt**> + 3 on many keyboards). Multi-line comments can be delimited (before and after) by ''' (three single quotes).

2.1.2 Indentation and Block Structure

Python is block structured but uses code indentation, rather begin...end, to delimit related blocks of code.

2.1.3 Input and Output

The most basic functions are input() and print().

2.1.4 Declaration of Simple Types and Type Casting

Only the numeric type of int and float will be used here: an int holds an integer value and has no decimal point: a float hold a floating point value and has a decimal point. The two types have different internal representations. Before being used an alphanumeric variable must be declared to be of a particular type by

1. assigning it to a constant as in a = 0.0
2. making multiple assignments as in a, b, c = 0.0, 1, 2
3. assigning it to the value of an expression as in a = b/c, where b and c have previously been declared.

Variables of one type may be cast as the other by using the int() and float() functions.

2.1.5 Arithmetic Operators and Precedence

The basic arithmetic operators are $+, -, *, /$ supplemented by ** for exponentiation, // for floor (integer) division and % for remainder. Parsing of arithmetic expressions is carried out left to right obeying the following order of precedence

1. (highest) ** and unary minus but see note below
2. $*, /, //, \%$
3. (lowest) $+, -$

Note that ** is higher than left unary minus, ** is lower than a right unary minus in terms of precedence.

This order of precedence may be changed by using parentheses ().

2.1.6 Conditional Expressions, Relational and Logical Operators

Python has a cast of 'all the usual suspects' for its relational operators, which are

1. $==$ equals. Note that a single $=$ is only used for declaration or assignment
2. $!=$ not equals
3. $>$ greater than
4. $>=$ greater than or equals
5. $<$ less than
6. $<=$ less than or equals.

These all return Boolean values of True or False and are ranked below '$+, -$' in order of precedence. They can be chained together to form multiple relational expressions such as $x < y < z$.

Logical operators can be used to combine such Boolean values into composite conditional expressions with a value of True or False. They rank lower than the relational operator and have the following order of precedence

1. **not** expr—True if expr is False, False otherwise
2. expr1 **and** expr2—True if both expr1 and expr2 are True, False otherwise
3. expr1 **or** expr1—True if either expr1 or expr2 is True, False otherwise.

Conditional expressions can thus be made of one, or more, relational expressions tied together with logical operators.

2.1.7 Conditional Statements

Multiple line statements are much more easily entered within an Editor shell (see later section).

Conditional statements are those which allow the selection of a consequence or alternative statement depend on the evaluation of a conditional expression. They come in three flavours which are best illustrated by examples.

1. conditional statements have a simple structure and are most often used for boolean assignments, thus `out=x and not y or not x and y`
2. **inline if** is used for conditional assignment and has a simple structure, thus `abs=−x if x<0 else x`
3. **if, elif, else** conditional statements will perform blocks of statements, after each**:**, dependent on the result of the conditional expression. **elif** short for **else if** allows for nested conditional statements. As described earlier indentation is used to distinguish blocks. For example

```
if number<0:
    print('negative')
elif number>0:
    print ('positive')
else:
    print('zero')
```

2.1.8 Looping Statements

Multiple line statements are much more easily entered within an Editor shell (see later section).

Looping statements are those which allow the controlled repetition of a statement or group of statements. They again come in two flavours which are best illustrated by examples. The examples are in the Python editor section and cover both simple versions of **for** loops and **while** loops. No attempt has been made to explain list types, or **in** (membership) operators as such details are deferred to later Project Code chapters.

2.2 Entering Python Code

IDLE supports two types of shell

1. the Python shell which support interactive development via the Python interpreter
2. one or more Editor shells which allow you to edit and save Python code.

2.2.1 The Python Interpreter

Type the following into the IDLE Python interpreter shell, (comments are optional), followed by <**return**> or <**enter**> for each line and observe the output.

```
# multiple declarations
a,b=355,113
# multiple assignments
p,q,r=a/b,a//b,a%b
# multiple prints
print(p,q,r)
s=float(a)+r/b
print(s)
p==s
# generally unsafe to use == (equality) on floats
# Boolean declarations
x,y,out=False,True,False
# 'exclusive or' as a conditional expression
out=x and not y or not x and y
# 'abs' as an inline if statement
abs=-x if x<0 else x
```

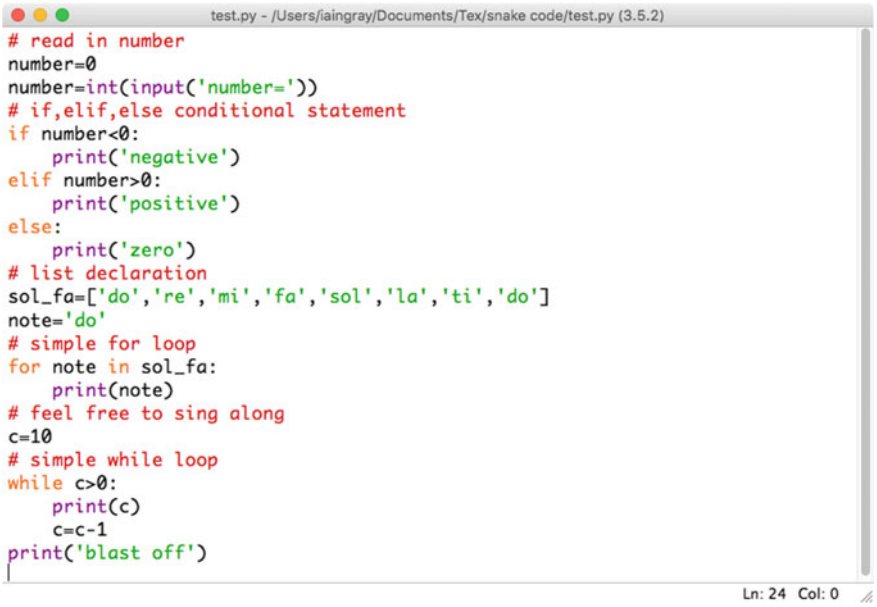
2.2.2 The Python Editor

Unfortunately the Python interpreter shell vanishes on quitting IDLE. to recall your work you must create a new editor shell, type in your program and save it with a .py extension. Then on restarting IDLE open your file to run it, or 'run module' in the Python shell via the Run menu in the Editor shell. The editor offers many features like automatic syntax highlighting, auto-completion of language words, auto-indentation of blocks and highlighting parenthesised expressions.

Open a new Editor shell, and type the following into the IDLE editor shell, (comments are optional). Save your work in 'test.py', reopen as described above and observe the output. Note that although strings are used this is not a 'hello world' program

```
# read in number
number=0
number=int(input('number='))
# if, elif, else conditional statement
if number<0:
    print('negative')
elif number>0:
    print ('positive')
else:
    print('zero')
# list declaration
sol_fa=['do','re','mi','fa','sol','la','ti','do']
note='do'
# simple for loop
for note in sol_fa:
    print(note)
# feel free to sing along
c=10
# simple while loop
while c>o:
    print(c)
    c=c—1
print('blast off')
```

Finally the syntax highlighted IDLE editor window for ‘test.py’ looks like



http://www.springer.com/978-3-319-60659-0

Snake Charming - The Musical Python

Gray, I.

2017, XIV, 121 p. 66 illus., 59 illus. in color., Softcover
ISBN: 978-3-319-60659-0