4 Basic Python Programming: Some Practice and *if*

In this tutorial we will slow down a bit. You'll learn a few new things including how to write to a file. For the most part, though, we will be combining things you already know in different ways to give you a bit of practice.

4.1 Fun with files

Download the file harryPotter.txt from the website (http://www.zacharski.org/python). (The file contains a few lines from J.K. Rowling's *Harry Potter and the Socercer's Stone*) Can you write a function that opens the file, reads the lines of the file into a list, closes the file, and then prints the lines to the screen? Try to write this on your own before proceeding.

Try it now on your own!

Let's look at one way of doing this. We want to write a function that will:

1) open the file harryPotter.txt
2) read the lines of the file into a list
3) close the file
4) print the lines to the screen.

Let's call the function print_file:

```
def print_file():
    """read a file and then print the lines to the screen"""
    #1) open the file harryPotter.txt
```

```
     #2) read the lines of the file into a list
     #3) close the file
     #4) print the lines to the screen.
```

We know how to do steps 1, 2, and 3:

```
def print_file():
   """read a file and then print the lines to the screen"""
   infile = open(r'C:\AI\python\book\harryPotter.txt', 'r')
   lines = list(infile)
   infile.close()
   #4) print the lines to the screen.
```

In our discussion of the *for* loop in the previous tutorial, we saw how you can print the contents of a list:

```
>>> for noun in ['dog', 'cat', 'poodle', 'Maine Coon cat']:
...     print noun
```

Using this information we can fill in step 4 of our printFile function:

```
def print_file():
   """read a file and then print the lines to the screen"""
   infile = open(r'C:\AI\python\book\harryPotter.txt', 'r')
   lines = list(infile)
   infile.close()
   for line in lines:
       print line
```

When we run this function we get:

```
>>> print_file()
"This way!" Harry mouthed to the others and, petrified, they began to

creep down a long gallery full of suits of armor. They could hear

Filch getting nearer. Neville suddenly let out a frightened squeak

and broke into a run.

"RUN!" Harry yelled, and the four of them sprinted down the gallery

not looking to see whether Filch was following. Harry in the lead,

they  ripped through a tapestry.

"I think we lost him," Harry panted.

Ron looked shocked.

Harry shook his head.
```

Now lets make a number of modifications to this function. (The answers are in the solutions section at the end of this tutorial.)

4.1.1 Double space to single space

When you look at the file harryPotter.txt you'll notice that the lines are single spaced but when you print the file with print_file you get a blank line appearing between each line. Why is this? There are two (or more) possible fixes to this problem. Can you name them and alter the function to incorporate one of the fixes?

4.1.2 Replacing a string

Can you print the file to the screen as before, but this time, replace every occurrence of *Harry* with your favorite name? So, for example, if we replace every occurrence with *Jose Guerra* the output would be something like:

> "This way!" **Jose Guerra** mouthed to the others and, petrified, they began to
> creep down a long gallery full of suits of armor. They could hear
> Filch getting nearer. Neville suddenly let out a frightened squeak
> and broke into a run.
> "RUN!" **Jose Guerra** yelled, and the four of them sprinted down the gallery
> not looking to see whether Filch was following. **Jose Guerra** in the lead,
> they ripped
> through a tapestry.
> "I think we lost him," **Jose Guerra** panted.
> Ron looked shocked.
> **Jose Guerr**a shook his head.

4.2 Writing to a file.

Writing to a file is similar to reading from a file. First you open the file for writing:

```
>>> outfile = open(r'C:\AI\python\my_output_file.txt', 'w')
```

To write text to a file you use the write method:

```
>>> outfile.write('This is the first line\n')
```

Finally, you close the file:

```
>>> outfile.close()
```

Let's write a function that takes two arguments: an input file name, and an output file name. It reads the lines from the input file and writes them to the output file. A few pages ago we wrote the following function to print a text file to the screen:

```
def print_file():
   """read a file and then print the lines to the screen"""
   infile = open(r'C:\AI\python\book\harryPotter.txt', 'r')
   lines = list(infile)
   infile.close()
   for line in lines:
       print line
```

Let's use this function as a basic template. First, let's change the name of the function to *make_story* and alter it so it takes two arguments:

```
def make_story(inputfilename, outputfilename):
   """read a file and then copies it to an output file"""
   infile = open(r'C:\AI\python\book\harryPotter.txt', 'r')
   lines = list(infile)
   infile.close()
   for line in lines:
       print line
```

Next, let’s change the line where we open the input file to make use of the inputfilename argument:

```
def make_story(inputfilename, outputfilename):
   "read a file and then copies it to an output file"
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   for line in lines:
       print line
```

Now we need to add the lines that open and close the output file:

```
def make_story(inputfilename, outputfilename):
   """read a file and then copies it to an output file"""
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   outfile = open(outputfilename, 'w')
   for line in lines:
       print line
   outfile.close()
```

Finally, we change the print line to one that writes to the file:

```
def make_story(inputfilename, outputfilename):
   """read a file and then copies it to an output file"""
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   outfile = open(outputfilename, 'w')
   for line in lines:
       outfile.write(line)
   outfile.close()
```

Now, just to make the program a bit fancier, let’s add raw_input functions to ask the user for the filenames:

```
# make_story.py

def make_story(inputfilename, outputfilename):
   """read a file and then copies it to an output file"""
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   outfile = open(outputfilename, 'w')
   for line in lines:
       outfile.write(line)
   outfile.close()
```

```
in_name  = raw_input('Enter input filename:')
out_name = raw_input('Enter output filename')
make_story(in_name, out_name)
```

When you run this program, you will get dialog boxes appearing that ask for the two filenames. The program then copies the text from the input file to the output file.

4.2.1 Your Job—should you choose to accept it.

I'd like you to alter make_story.py so it will replace various character names with names of your choosing—just like you did before.[1] The first thing you will need to do is to get some stories to work with. One good site is Project Guttenberg (http://promo.net/pg/). On that website, select the 'browse' function and download a few fun texts.

Here are a few examples:

Tarzan of the Apes (replacing 'Tarzan' with 'Winnie the Pooh')

Not so, however, with **Winnie the Pooh**, the man-child. His life
amidst the dangers of the jungle had taught him to meet
emergencies with self-confidence, and his higher intelligence
resulted in a quickness of mental action far beyond the powers
of the apes.

So the scream of Sabor, the lioness, galvanized the brain
and muscles of little **Winnie the Pooh** into instant action.

Before him lay the deep waters of the little lake, behind
him certain death; a cruel death beneath tearing claws and
rending fangs.

Winnie the Pooh had always hated water except as a medium for
quenching his thirst. He hated it because he connected it with
the chill and discomfort of the torrential rains, and he feared
it for the thunder and lightning and wind which accompanied them.

Tom Swift and His Airship
(replacing *Tom* with *Peter,* and *Swift* with *Zacharski*)

Peter Zacharski, the young inventor, whose acquaintance some of you have
previously made, gave one look at the gauge, and seeing that the
pressure was steadily mounting, endeavored to reach, and open, a stop-
cock, that he might relieve the strain. One trial showed him that the
valve there had jammed too, and catching up a roll of blue prints the
lad made a dash for the door of the shop. He was not a second behind
his companion, and hardly had they passed out of the structure before
there was a loud explosion which shook the building, and shattered all
the windows in it.

[1] This idea is from Jen Doyon. See http://www.georgetown.edu/cball/perl/perl-11.html

Pieces of wood, bits of metal, and a cloud of sawdust and shavings
flew out of the door after the man and the youth, and this was
followed by a cloud of yellowish smoke.

"Are you hurt, **Peter**?" cried Mr. Sharp, as he swung around to look back
at the place where the hazardous experiment had been conducted.

You can download the text of Peter Pan from http://www.zacharski.org/python

4.3 Sorting and reporting

4.3.1 Sorting a file.

Recall that you can sort a list as follows:

```
>>> a = ['Clara', 'Sara', 'Ann', 'Ben', 'Hillary', 'Adam']
>>> a.sort()
>>> a
['Adam', 'Ann', 'Ben', 'Clara', 'Hillary', 'Sara']
>>>
```

Can you write a program that will take an input file like:

```
Carloz Alverez     551-1332
Marcia O'Rourke    523-6671
Lily McMaster      732-1921
Adam Nowicki       282-8992
Terumi Yobuko      451-3290
Jonathan Ginsberg  646-9902
Susie Swarton      951-6520
Peter Grayson      721-1205
Christine Mansouri 721-1207
John Adams         646-6521
Antonio Folerin    523-6673
```

(you can get this file, addresses.txt, at http://www.zacharski.org/python) and produce a output file where the lines are sorted alphabetically?

```
Adam Nowicki       282-8992
Antonio Folerin    523-6673
Carloz Alverez     551-1332
Christine Mansouri 721-1207
John Adams         646-6521
Jonathan Ginsberg  646-9902
Lily McMaster      732-1921
```

```
Marcia O’Rourke    523-6671
Peter Grayson      721-1205
Susie Swarton      951-6520
Terumi Yobuko      451-3290
```

For hints see the hints section at the end of this tutorial.

4.3.2 Splitting

Let’s say you have the following string:

```
>>> a = "Pregunta si nos vamos a quedar mucho tiempo"
```

You can convert this string into a sequential list of words by using the split method:

```
>>> b = a.split()
>>> b
['Pregunta', 'si', 'nos', 'vamos', 'a', 'quedar', 'mucho', 'tiempo']
```

What the split method does is split the string based on occurrences of spaces. Also recall that you can join two lists as follows:

```
>>> c = ['Ya', 'te', 'pedi', 'disculpas']
>>> c
['Ya', 'te', 'pedi', 'disculpas']
>>> b = b + c
>>> b
['Pregunta', 'si', 'nos', 'vamos', 'a', 'quedar', 'mucho', 'tiempo', 'Ya',
'te', 'pedi', 'disculpas']
>>>
```

Now, knowing this, can you write a function that reads in a text file and prints a list of all the words in the file sorted alphabetically? That is, given the Harry Potter text

```
“This way!” Harry mouthed to the others and, petrified, they began to
creep down a long gallery full of suits of armor. They could hear
Filch getting nearer. Neville suddenly let out a frightened squeak
and broke into a run.
“RUN!” Harry yelled, and the four of them sprinted down the gallery
not looking to see whether Filch was following. Harry in the lead,
they  ripped
through a tapestry.
“I think we lost him,” Harry panted.
Ron looked shocked.
Harry shook his head.
```

Your program would output:

```
>>> sort_words_in_a_file(r"d:/AI/Python/book/harryPotter.txt")
Filch
Filch
Harry
Harry
Harry
Harry
Harry
Neville
```

```
Ron
They
a
a
a
a
and
and
and,
armor.
began
```

For one solution see the solutions section at the end of this chapter. Also note that the sort function sorts capital letters ahead of lower case ones.

4.3.3 Reporting

To tell how many characters there are in a string you can use the `len` function:

```
>>> s1 = 'agua'
>>> len(s1)
4
>>> s2 = 'Estás bien seguro de que es agua la que vaciaste en el bebedero?'
>>> len(s2)
64
```

You can also use the `len` function to count how many elements are in a list:

```
>>> c = ['Ya', 'te', 'pedi', 'disculpas']
>>> len(c)
4
>>>
```

Can you write a function that reads in a file and reports how many characters, words, and lines are in the file? For example:

```
>>> wc(r'C:\AI\python\book\harryPotter.txt')
C:\AI\python\book\harryPotter.txt:
   Total characters:  461
   Total words:       83
   Total lines:       11
```

Try it now on your own!
One solution is shown at the end of this chapter.

4.4 The *if* statement

In chapter 2 we described how to find out if an item is in a list:

```
>>> linguists = ['Ann', 'Ben', 'Polly', 'Dora']
>>> 'Polly' in linguists
True
```

```
>>> 'Steve' in linguists
False
```

and we mentioned that 'True' denotes a 'yes' answer and 'False' denotes 'no'. Suppose we want to have a print statement execute conditional on the answer to this true/false query—something like “If Polly is a linguist print 'Polly is smart'. We can do this by using the *if* statement:

```
>>> if 'Polly' in linguists:
...     print 'Polly is a linguist'
...
Polly is a linguist
```

The template of this statement is:

```
if expression:
    code block
```

If the expression part of the *if*-statement evaluates to False then the code block is skipped; otherwise the statements in the code block execute. The code block can consist of one or more lines of Python statements:

```
>>> if 'Polly' in linguists:
...     print 'Polly is a linguist'
...     print 'Polly is smart'
...
Polly is a linguist
Polly is smart
```

The *if*-statement can have an *else* clause. So, for example, we can define the function `is_linguist`:

```
def is_linguist(name):
    """check to see if name is a linguist"""
    if name in ['Ann', 'Ben', 'Sally', 'Polly']:
        print name + ' is a linguist'
    else:
        print name + ' is not a linguist'
```

and we can try this function out:

```
>>> is_linguist('Polly')
Polly is a linguist
>>> is_linguist('Abe')
Abe is not a linguist
```

An *if*-statement can also contain an *else if* clause (which is coded as *elif*):

```
def is_what(name):
    """check the occupation of the person named name"""
    if name in ['Ann', 'Ben', 'Sally', 'Polly']:
        print name + ' is a linguist'
    elif name in ['Abe', 'Bebe', 'Clara']:
        print name + ' is a pianist'
    else:
        print "I'm not sure what " + name + ' does'

>>> is_what('Sally')
Sally is a linguist
```

```
>>> is_what('Clara')
Clara is a pianist
>>> is_what('Steve')
I'm not sure what Steve does
>>>
```

Finally, an *if*-statement can have multiple *elif* clauses:

```
if name in ['Ann', 'Ben', 'Sally', 'Polly']:
    print name + ' is a linguist'
elif name in ['Abe', 'Bebe', 'Clara']:
    print name + ' is a pianist'
elif name in ['Jim', 'Francis', 'Jewel']:
    print name + ' is a writer'
else:
    print "I'm not sure what " + name + ' does'
```

4.4.1 A detailed look at *if*-statement expressions.

Again, the *if*-statement template is:

```
if expression:
    code block
```

and if the expression evaluates to False then the code block is skipped, otherwise the block is executed. Let's spend a bit of time discussing what can be an expression.

The expression can be one of the following arithmetic comparison operators:

Operator	***Function***
`a == b`	*a* is equal to *b*
`a != b`	*a* is not equal to *b*
`a < b`	*a* is less than *b*
`a > b`	*a* is greater than *b*
`a <= b`	*a* is less than or equal to *b*
`a >= b`	*a* is greater than or equal to *b*

```
>>> a = 4
>>> a < 5
True
>>> a < 2
False
>>> a > 0
True
>>> a > 10
False
>>> a == 4
True
>>> a == 5
```

```
False
>>> a != 5
True
>>> len('this') > 5
False
>>> len('this') == 4
True
>>> len('comparison')
10
>>> len('reference') == len('magnitude')
True
```

These operators can also be applied to strings. If a and b are strings a < b is interpreted as does a come before b in alphabetically.

```
>>> 'adam' < 'ben'
True
>>> 'ben' < 'adam'
False
>>> name = 'Adam'
>>> name == 'Adam'
True
```

A common program error is to confuse the '=' and'==' operators, as in

```
>>> name = 'Adam'
>>> name == 'Adam'
```

The first, (name = 'Adam') is an assignment operation—we assign the name *name* to the value 'Adam'. The second (name == 'Adam') is a comparison operation—we are asking whether the value of *name* is equal to 'Adam'.

We can also use the Boolean operators **`and`**, **`or`**, and **`not`** in expressions:

```
>>> age = 15
>>> age < 20 and age > 10
True
>>> name = 'Ann'
>>> name == 'Ann' or name == 'Alice' or name == 'Fred'
True
>>> name = 'Brenda'         name now equals “Brenda”
>>> name == 'Ann' or name == 'Alice' or name == 'Fred'
False
>>> not name == 'Ann'       “Brenda” does not equal “Ann”
True
```

Finally, there are a number of string expressions. Some of the more useful ones are shown in the following table:

Expression	***Function***	***Example***

string.startswith(str)	Returns True if the string starts with the string *str* and returns False othewise.	`a = 'theory'`
		`b = '1997'`
		`a.startswith('the')`

Expression	***Function***	***Example***

		`True`
		`b.startswith('the')`
		`False`

string.endswith(str)	Returns True is the string ends with the string *str*	`a.endswith('y')`
		`True`
		`a.endswith('eory')`
		`True`
		`b.endswith('y')`
		`False`

string.isalpha()	Returns True if the string only alphabetic	`a.isalpha()`
		`True`
		`b.isalpha()`
		`False`

string.isdigit()	Returns True if the string contains only digits.	`a.isdigit()`
		`False`
		`b.isdigit()`
		`True`

string.isspace()	Returns True if the string contains only spaces.	`a.isspace()`
		`False`
		`c = '          '`
		`c.isspace()`
		`True`

string.islower()	Returns True if the string contains all lower cased letters	`a.islower()`
		`True`
		`b.islower()`
		`False`

string.isupper()	Returns True if the string contains all upper cased letters.	`a.isupper()`
		`False`
		`d = 'THEORY'`
		`d.isupper()`
		`True`

Suppose I want to write a function `license` that takes an integer as an argument. If the integer is equal or larger than 18, it outputs *Old enough to get a license.* Otherwise it outputs *Too young.* A skeleton of this function is

```
def license(age):
    """Check if age is at least 18 so person can get license"""
    if blah blah blah:
        print "Old enough to get a license"
    blah blah blah
        blah
```

You need to determine what the *blah blah blahs* should be.

Try it now on your own!
One solution is shown at the end of this chapter.

4.4.2 A challenge

Consider the following example:

```
def three_letter_words(input):
    """print the three letter words that occur in the input"""
    # first split the input into a list
    in_list = input.split()
    # now loop through the list
    for word in in_list:
        # print each word
        print word
```

Here's what happens when we run it:

```
>>> three_letter_words('The dog saw the lazy cat chasing the big rat')
The
dog
saw
the
lazy
cat
chasing
the
big
rat
>>>
```

1) Can you alter this function so it only prints words that are three letters long? If you are having problems look at the hint section at the end of this tutorial.
2) Can you alter the function you just wrote so it takes two arguments: a string, and an integer specifying the length of the words to retrieve. That is `letter_words(sentence, 3)` will retrieve all the three letter words in *sentence* and `letter_words(sentence, 4)` will retrieve all the four letter words.

Try it now on your own!
One solution is shown at the end of this chapter.

4.4.3 Another example

Suppose I have a list of determiners:

```
det = ['a', 'the', 'this', 'that', 'these', 'those']
```

and a list of prepositions:

```
prep = ['of', 'in', 'by', 'to', 'on']
```

I want to write a function, tag, that takes as input a sentence, tags the determiners and prepositions with their parts of speech and returns a sentence with these tags. For example,

```
>>> tag('this dog gave the cheese to those mice on the mat.')
this-det dog gave the-det cheese to-prep those-det mice on-prep the-det mat.
```

The rough outline of what the function might look like is shown here:

```
def tag(input):
    """Tag the determiners and prepositions in the input"""
    # specify the determiner and preposition lists
    # initialize the result string
    # for each word in the input
        # if the word is a determiner add it and the det tag to
        # the result string
        # else if the word is a preposition add it plus
        # the prep tag to the result string
        # else just add the word to the result string

    # return the result
```

If you don't have an extremely clear picture of how to write a particular function it is often a good idea to simplify the function to something you know. For example, we can simplify the tag function to:

```
def tag(input):
    """Tag the determiners and prepositions in the input"""
    # initialize the result string
    # for each word in the input
        # add the word to the result string

    # return the result
```

Stop reading, code this function and test it out.

If you are having problems go back through this tutorial and find a similar example.

How did you do? I imagine you came up with something similar to the following:

```
def tag(input):
    """Tag the determiners and prepositions in the input"""
    # initialize the result string
    result_string = ''
    word_list = input.split()
    # for each word in the input
    for word in word_list:
        # add the word to the result string
        result_string = result_string + word + ' '

    # return the result
    return result_string
```

Now we only have the highlighted parts remaining to be coded:

```
def tag(input):
    """Tag the determiners and prepositions in the input"""
    # specify the determiner and preposition lists
    # initialize the result string
    result_string = ''
    word_list = input.split()
    # for each word in the input
    for word in word_list:
        # if the word is a determiner add it and the det tag to
        # the result string
        # else if the word is a preposition add it plus
        # the prep tag to the result string
        # else add the word to the result string
        result_string = result_string + word + ' '

    # return the result
    return result_string
```

Specifying the determiner and proposition lists can be done by:

```
# specify the determiner and preposition lists
det = ['a', 'the', 'this', 'that', 'these', 'those']
prep = ['of', 'in', 'by', 'to', 'on']
```

Checking to see if a word is a determiner can be done by checking whether the word is in the determiner list:

```
if word in det:
```

So our revised function is:

```
def tag(input):
    """Tag the determiners and prepositions in the input"""
    # specify the determiner and preposition lists
    det = ['a', 'the', 'this', 'that', 'these', 'those']
    prep = ['of', 'in', 'by', 'to', 'on']
    # initialize the result string
    result_string = ''
    word_list = input.split()
    # for each word in the input
    for word in word_list:
        # if the word is a determiner add it and the det tag to
        # the result string
        if word in det:
            result_string = result_string + word + '-det '
        # else if the word is a preposition add it plus
        # the prep tag to the result string
        # else add the word to the result string
        else:
            result_string = result_string + word + ' '

    # return the result
    return result_string
```

and we can test this function:

```
>>> tag('this dog gave the cheese to those mice on the mat.')
'this-det dog gave the-det cheese to those-det mice on the-det mat. '
```

Finally, we can add the test for prepositions:

```
def tag(input):
    """Tag the determiners and prepositions in the input"""
    # specify the determiner and preposition lists
    det = ['a', 'the', 'this', 'that', 'these', 'those']
    prep = ['of', 'in', 'by', 'to', 'on']
    # initialize the result string
    result_string = ''
    word_list = input.split()
    # for each word in the input
    for word in word_list:
        # if the word is a determiner add it and the det tag to
        # the result string
        if word in det:
            result_string = result_string + word + '-det '
        # else if the word is a preposition add it plus
        # the prep tag to the result string
        elif word in prep:
            result_string = result_string + word + '-prep '
        # else add the word to the result string
        else:
            result_string = result_string + word + ' '

    # return the result
    return result_string
```

4.4.4 A stop list

In section 4.3.2 above, you wrote a function that reads in a text file and prints a list of all the words in the file sorted alphabetically. Now I'd like to change that function as follows. In addition to the file containing the English text, you have a text file that contains a list of words that you do not want to appear in the output. For example, this stop list file might contain the following words:

```
stoplist.txt
```

```
a
A
the
The
and
or
on
under
in
by
it
them
they
that
```

and the text to analyze is:

```
story.txt

The posters show crop circles, those huge geometric shapes
in fields of corn and wheat, which were seen all over the
world in the 1970s. Their origin was explained in 1991 when
several hoaxers came forward and demonstrated how they made
them; it was not difficult, they said. Like many supernatural
events, however, crop circles live on after their unmasking,
and most people today have forgotten, or never knew, that they
were explained. "Signs" uses them to evoke the possibility that ...
well, the possibility of anything.
```

Then the function call:

```
>>> sort_words_in_a_file('story.txt','stoplist.txt')
```

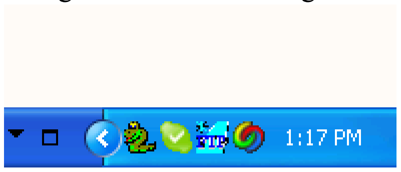
Would print a sorted list of the words in the file that were not in the stoplist.
One possible solution to this problem is shown at the end of this tutorial.

4.5 *While*

The *while* statement is a looping statement of the form:

```
while expression:
      code block to repeat
```

Before I talk about *while* I should mention that if you ever mess something up rather badly and the Python interpreter appears to be locked up, you often fix it by right-clicking the PythonWin icon in the system tray (in the lower right corner) and selecting "Break into running code".



If that doesn't work, you can always exit by clicking on the standard windows close button—the 'x' in the top right of the window.
With that warning, let's dive into while. Here's an example:

```
>>> i = 0
>>> while i < 5:
...   print i
...   i = i + 1
...
0
1
2
3
4
```

Let's go through this step by step. The first thing I did was set *i* equal to 0. The next line starts the while statement. The expression *i* < 5 is true so the code block executes—*i* is printed (in this case '0') and *i* is incremented by 1, so *i* now equals 2. Now we are at the end of the code block so we return to the while statement (while i < 5:); i is smaller than 5 (2 < 5) so we execute the while loop printing *i* and incrementing *i*. Let's skip ahead a bit and consider the iteration where *i* is equal to 4. In this case, *i* < 5 still holds, we print *i* and increment it so now *i* equals 5. Now the while expression *i* < 5 evaluates to False and we skip the code block.

The warning I mentioned earlier occurs when you erroneously create an infinite loop by doing something like the following:

```
i = 0
while i < 5
   print i
```

Since *i* is never incremented *i* < 5 is always true and we are in an infinite loop of printing '0'.

4.5.1 A guessing game using while.

Let's quickly review the information about the random module, which we discussed in chapter 2. Before we can use any functions in a module we need to import the module using the 'import' statement. So to import 'random' we use:

```
>>> import random
```

We can use the choice function in the random module to select a random element from a list:

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.choice(a)
6
```

To select a random number between 0 and 100, we could create a list of 100 elements and use the choice function. However, there is another way. The random module has a function called 'randint' that takes two integer arguments and returns a random integer between those two integers. For example,

```
>>> random.randint(0, 100)
```

will return a random integer between 0 and 100. Let's use this function to create a guessing game function. The function will first select a random number and then allow the user to guess the number. If the user's guess is incorrect it will keeping asking the user until the correct number is guessed. For example:

```
>>> guess()
What is your guess? 51
No, that's not it. What is your guess? 25
No, that's not it. What is your guess? 77
...
No, that's not it. What is your guess? 63
You guessed it!
```

We can create such a guessing function using a while loop. The rough outline is

```
def guess():
    """Guess the number"""
    # choose a random number from 0 - 100
    # get user guess
    # while the guess is not correct
        # get a user guess
    # print congratulations message
```

We are going to get a user guess by using the function `raw_input.` Recall that raw input returns a string, which we will need to convert to an integer:

```
a = int(raw_input('What is your guess?'))
```

Here's the complete function.

```
def guess():
    """Guess the number"""
    # choose a random number from 0 - 100
    num = random.randint(0, 100)
    # get user guess
    a = int(raw_input('What is your guess?'))
    # while the guess isn't correct
    while a != num:
        # get another user guess
        a = int(raw_input('Nope. What is your next guess?'))
    print "You guessed it!"
```

Can you alter this program so it gives the user clues. For example:

```
>>> guess()
What is your guess? 50
Too high. Try again 25
Too high. Try again 12
Too low. Try again 18
Too low. Try again 21
You guessed it!
```

4.5 Summary

Throughout this chapter we have been using bits that we have learned in other chapters and combining those bits in various ways to do interesting things. Feel free to experiment and try your own exercises. Don't worry if your functions aren't perfect. For example, when we wrote a function to sort the words in a file, we didn't eliminate duplicate words—if there were 37 occurrences of 'a' we printed all 37 of them—and we didn't worry about punctuation or capitalization (*The, "The, and 'the'* were all sorted in different places in the list). Nevertheless, the program you did write was pretty cool.

5 Hints

sorting a file

You can start with the make_story function:

```
def make_story(inputfilename, outputfilename):
   """read a file and then copies it to an output file"""
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   # ??????????????????
   outfile = open(outputfilename, 'w')
   for line in lines:
       outfile.write(line)
   outfile.close()
```

You will need to add one line right where I put the '???????' comment.

Another hint:
The lines read from the input file are in a list. Is there a way to sort the list?
For a solution, see the following "Solutions" section.

4.4.2 A challenge

Hint 1
What part of the function do we need to alter? It's the section I highlight here in bold.

```
def three_letter_words(input):
    """print the three letter words that occur in the input"""
    # first split the input into a list
    in_list = input.split()
    # now loop through the list
    for word in in_list:
        # print each word
        print word
```

In English, we need to add something like the following:

```
def three_letter_words(input):
    """print the three letter words that occur in the input"""
    # first split the input into a list
    in_list = input.split()
    # now loop through the list
    for word in in_list:
        # if the word is a 3 letter word
           # print each word
           print word
```

Hint 2:

How do we test whether a word is a three letter word? The answer is in this footnote.[2]

Solutions

3.1.1 Double space to single space

The print statement inserts a newline the end of the string. So, for example:

```
def print_some_lines():
    print 'This is line one'
    print 'and line two'
    print 'and line three'
```

prints

```
>>> print_some_lines()
This is line one
and line two
and line three
```

(A newline is printed at the end of every print statement.)

When you read the lines of a file the newlines are preserved. That is, if the file looks like:

This is line one
and line two
and line three

when you read the lines in as in:

```
lines = list(infile)
```

the list, lines, will be

```
['This is line one\n', 'and line two\n', 'and line three\n']
```

When you write the statement

```
>>> print lines[0]
```

two consecutive newlines will be printed. The first comes from the string, 'This is line one\n' itself. The second comes from the print statement.

[2] `len(word) == 3`

Two methods for solving the problem
To solve the problem you can either remove the newline from the string using the replace method, which we covered in section 3.2.2. of the third tutorial. Another alternative is to prevent print from adding a newline. This is accomplished by adding a terminating comma to the statement:

```
print lines[0],
```

3.2 Replacing a substring

You can use the replace function:

```
line = line.replace('Eddie Willers', 'Carlos Montero')
```

4.3.1 sorting a file

```
def sort_file(inputfilename, outputfilename):
   "sort lines in a file"
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   lines.sort()
   outfile = open(outputfilename, 'w')
   for line in lines:
       outfile.write(line)
   outfile.close()
```

4.3.2 sorting words in a file

There are a number of improvements that can be made to this function. We'll look at solutions to these in the next chapter.

```
def sort_words_in_a_file(inputfilename):
   """sort words in a file"""
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   # this is the list of words, initially empty
   words = []
   for line in lines:
       # add the words in this line to the word list
       words = words + line.split()
   # now sort the word list
   words.sort()
   # and print the results
   for word in words:
       print word
```

4.3.3 reporting

```
def wc(inputfilename):
   """reports on characters, words, and lines in a file"""
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   # set initial counts to zero
   words = 0
   chars = 0
   num_lines = 0
   for line in lines:
       # increment the line count
       num_lines = num_lines + 1
       # add number of characters in line to total
       chars = chars + len(line.strip())
       words = words + len(line.split())
   # now print results
   print inputfilename + ':'
   print '   Total characters: ',
   print chars
   print '   Total words:      ',
   print words
   print '   Total lines:      ',
   print num_lines
```

4.4 *if* statements

4.4.1.a - Old enough for drivers' license?

```
def license(age):
    """Check if age is at least 18 so person can get license"""
    if age >= 18:
        print "Old enough to get a license"
    else:
        print "Too young"
```

4.4.3A challenge

```
def three_letter_words(input):
    """print the three letter words that occur in the input"""
    # first split the input into a list
    in_list = input.split()
    # now loop through the list
    for word in in_list:
        # if word is 3 letters long
        if len(word) == 3:
            # print each word
            print word


def letter_words(input, length):
    """print the words of length that occur in the input"""
    # first split the input into a list
    in_list = input.split()
    # now loop through the list
    for word in in_list:
        # if word is 3 letters long
        if len(word) == length:
            # print each word
            print word
```

4.4.4 A stop list

```
def sort_words_in_a_file(inputfilename, stoplistfile):
   """sort words in a file"""
   # first read in the stop list
   infile = open(stoplistfile)
   lines = list(infile)
   infile.close()
   # initialize stoplist to null list
   stoplist = []
   # for each line in the file add a stoplist entry
   for line in lines:
       stoplist.append(line.strip())

   # now open and read the lines from the inputfile.
   infile = open(inputfilename, 'r')
   lines = list(infile)
   infile.close()
   # this is the list of words, initially empty
   words = []
   for line in lines:
       # add the words in this line to the word list
       for wrd in line.split():
           if not wrd in stoplist:
               words.append(wrd)
   # now sort the word list
   words.sort()
   # and print the results
   for word in words:
       print word
```

4.5.1 The guessing game

```
def guess():
    """Guess the number"""
    # choose a random number from 0 - 100
    num = random.randint(0,100)
    prompt = 'What is your guess?'
    # get user guess
    a = int(raw_input(prompt))
    # while the guess isn't correct
    while a != num:
        if a < num:
            prompt = 'Too low. Try again'
        else:
            prompt = 'Too high. Try again.'
        # get another user guess
        a = int(raw_input(prompt))
    print "You guessed it!"
```