Non-Programmer's Tutorial for Python 2.6

Wikibooks.org

March 13, 2013

Contents

1 Front matter

All example Python source code in this tutorial is granted to the public domain. Therefore you may modify it and relicense it under any license you please. Since you are expected to learn programming, the GNU Free Documentation License would require you to keep all programs that are derived from the source code in this tutorial under that license. Since the python source code is granted to the public domain, that requirement is waived.

This tutorial was originally written in LaTeX and was available at: `http://www.honors.montana.edu/~jjc/easytut/`. It was moved here because the other server is going away and it was being read at least ten times a day. This document is available as LaTeX, HTML, PDF, and Postscript. Go to `http://jjc.freeshell.org/easytut/` (Also could try `http://web.archive.org/web/*/http://www.honors.montana.edu/~jjc/easytut/` or `http://www.geocities.com/jrincayc/easytut.tar.gz`) to see all these forms. There are also versions of this in Korean, Spanish, Italian and Greek in the tar file.

The *Non-Programmers' Tutorial For Python* is a tutorial designed to be an introduction to the Python programming language. This guide is for someone with no programming experience.

If you have programmed in other languages I recommend using Python Tutorial for Programmers[1] written by Guido van Rossum.

If you have any questions or comments please use the discussion pages or see ../Authors[2] for author contact information. I welcome questions and comments about this tutorial. I will try to answer any questions you have as best I can.

Thanks go to James A. Brown for writing most of the Windows install info. Thanks also to Elizabeth Cogliati for complaining enough :) about the original tutorial (that is almost unusable for a non-programmer), for proofreading, and for many ideas and comments on it. Thanks to Joe Oppegaard for writing almost all the exercises. Thanks to everyone I have missed.

1.0.1 Other resources

- Python Home Page[3]
- Python Documentation[4]
- Python Tutorial for Programmers[5]

1 `http://docs.python.org/tutorial/`
2 `http://en.wikibooks.org/wiki/..%2FAuthors`
3 `http://www.python.org`
4 `http://www.python.org/doc/`
5 `http://www.python.org/doc/current/tut/tut.html`

- LaTeX, PDF, and Postscript, and Zip versions[6]

See also chapter The End[7] for some more comments.

6 http://www.honors.montana.edu/~jjc/easytut/
7 Chapter 19 on page 111

2 Intro

2.1 First things first

So, you've never programmed before. As we go through this tutorial, I will attempt to teach you how to program. There really is only one way to learn to program. **You** must read *code* and write *code* (as computer programs are often called). I'm going to show you lots of code. You should type in code that I show you to see what happens. Play around with it and make changes. The worst that can happen is that it won't work. When I type in code it will be formatted like this:

```
##Python is easy to learn
print "Hello, World!"
```

That's so it is easy to distinguish from the other text. If you're reading this on the web, you'll notice the code is in color -- that's just to make it stand out, and to make the different parts of the code stand out from each other. The code you enter will probably not be colored, or the colors may be different, but it won't affect the code as long as you enter it the same way as it's printed here.

If the computer prints something out it will be formatted like this:

```
Hello, World!
```

(Note that printed text goes to your screen, and does not involve paper. Before computers had screens, the output of computer programs would be printed on paper.)

If you try this program out and you get a syntax error, check and see what version of python you have. If you have python 3.0, you should be using the Non-Programmer's Tutorial for Python 3.0[1]. This article was made for Python 2.6

There will often be a mixture of the text you type (which is shown in **bold**) and the text the program prints to the screen, which would look like this:

```
Halt!
Who Goes there? Josh
You may pass, Josh
```

(Some of the tutorial has not been converted to this format. Since this is a wiki, you can convert it when you find it.)

1 http://en.wikibooks.org/wiki/Non-Programmer%27s%20Tutorial%20for%20Python%203.0

I will also introduce you to the terminology of programming - for example, that programming is often referred to as *coding*. This will not only help you understand what programmers are talking about, but also help the learning process.

Now, on to more important things. In order to program in Python you need the Python software. If you don't already have the Python software go to `http://www.python.org/download/` and get the proper version for your platform. Download it, read the instructions and get it installed.

2.1.1 Installing Python

For Python programming you need a working Python installation and a text editor. Python comes with its own editor *IDLE*, which is quite nice and totally sufficient for the beginning. As you get more into programming, you will probably switch to some other editor like *emacs*, *vi* or another.

The Python download page is http://www.python.org/download[2]. The most recent version is 3.1, but any *Python 2.x* version since 2.2 will work for this tutorial. Be careful with the upcoming *Python 3*, though, as some major details will change and break this tutorial's examples. A version of this tutorial for Python 3 is at Non-Programmer's Tutorial for Python 3[3]. There are various different installation files for different computer platforms available on the download site. Here are some specific instructions for the most common operating systems:

Linux, BSD and Unix users

You are probably lucky and Python is already installed on your machine. To test it type `python` on a command line. If you see something like that in the following section, you are set.

If you have to install Python, just use the operating system's package manager or go to the repository where your packages are available and get Python. Alternatively, you can compile Python from scratch after downloading the source code. If you get the source code make sure you compile in the Tk extension if you want to use IDLE.

Mac users

Starting from Mac OS X (Tiger), Python ships by default with the operating system, but you might want to update to the newer version (check the version by starting `python` in a command line terminal). Also IDLE (the Python editor) might be missing in the standard installation. If you want to (re-)install Python, have a look at the Mac page on the Python download site[4].

2 `http://www.python.org/download`
3 `http://en.wikibooks.org/wiki/Non-Programmer%27s%20Tutorial%20for%20Python%203`
4 `http://www.python.org/download/mac/`

Windows users

Some computer manufacturers pre-install Python. To check if you already have it installed, open command prompt (cmd in run menu) or MS-DOS and type python. If it says "Bad command or file name" you will need to download the appropriate Windows installer (the normal one, if you do not have a 64-bit AMD or Intel chip). Start the installer by double-clicking it and follow the procedure. Python for windows can be downloaded from the official site of python[5]

2.1.2 Interactive Mode

Go into IDLE (also called the Python GUI). You should see a window that has some text like this:

```
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit
 (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.


   ****************************************************************
   Personal firewall software may warn about the connection IDLE
   makes to its subprocess using this computer's internal loopback
   interface.  This connection is not visible on any external
   interface and no data is sent to or received from the Internet.
   ****************************************************************


IDLE 1.2.1
>>>
```

The `>>>` is Python's way of telling you that you are in interactive mode. In interactive mode what you type is immediately run. Try typing `1+1` in. Python will respond with `2`. Interactive mode allows you to test out and see what Python will do. If you ever feel you need to play with new Python statements, go into interactive mode and try them out.

2.1.3 Creating and Running Programs

Go into IDLE if you are not already. In the menu at the top, select `File` then `New Window`. In the new window that appears, type the following:

```
print "Hello, World!"
```

Now save the program: select `File` from the menu, then `Save`. Save it as "`hello.py`" (you can save it in any folder you want). Now that it is saved it can be run.

Next run the program by going to `Run` then `Run Module` (or if you have a older version of IDLE use `Edit` then `Run script`). This will output `Hello, World!` on the `*Python Shell*` window.

5 `http://www.python.org/getit/`

For a more in-depth introduction to IDLE, a longer tutorial with screenshots can be found at http://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/index.html[6]

Running Python Programs in Unix

If you are using Unix (such as Linux, Mac OSX, or BSD), if you make the program executable with `chmod`, and have as the first line:

```
#!/usr/bin/env python2
```

you can run the python program with `./hello.py` like any other command.

Note: In some computer environments, you need to write:

```
!/usr/bin/env python
```

Program file names

It is very useful to stick to some rules regarding the file names of Python programs. Otherwise some things *might* go wrong unexpectedly. These don't matter as much for programs, but you can have weird problems if you don't follow them for module names (modules will be discussed later).

1. Always save the program with the extension `.py`. Do not put another dot somewhere else in the file name.
2. Only use standard characters for file names: letters, numbers, dash (`-`) and underscore (`_`).
3. White space ("") should not be used at all (use e.g. underscores instead).
4. Do not use anything other than a letter (particularly no numbers!) at the beginning of a file name.
5. Do not use "non-english" characters (such as `ä`, `ö`, `ü`, `å` or `ß`) in your file names—or, even better, do not use them at all when programming.

2.1.4 Using Python from the command line

If you don't want to use Python from the command line, you don't have to, just use IDLE. To get into interactive mode just type `python` without any arguments. To run a program, create it with a text editor (Emacs has a good Python mode) and then run it with `python program_name`.

Additionally, to use Python within Vim, you may want to visit Using vim as a Python IDE[7]

6 `http://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/index.html`
7 `http://www.ibiblio.org/obp/pybiblio/tips/elkner/vim4python.php`

2.1.5 Where to get help

At some point in your Python career you will probably get stuck and have no clue about how to solve the problem you are supposed to work on. This tutorial only covers the basics of Python programming, but there is a lot of further information available.

Python documentation

First of all, Python is very well documented. There might even be copies of these documents on your computer, which came with your Python installation: *** The official Python Tutorial[8] by Guido van Rossum is often a good starting point for general questions**.

- For questions about standard modules (you will learn what this is later), the Python Library Reference[9] is the place to look.
- If you really want to get to know something about the details of the language, the Python Reference Manual[10] is comprehensive but quite complex for beginners.

Python user community

There are a lot of other Python users out there, and usually they are nice and willing to help you. This very active user community is organised mostly through mailing lists and a newsgroup:

- The tutor mailing list[11] is for folks who want to ask questions regarding how to learn computer programming with the Python language.
- The python-help mailing list[12] is python.org's help desk. You can ask a group of knowledgeable volunteers questions about all your Python problems.
- The Python newsgroup [news:comp.lang.python comp.lang.python] (Google groups archive[13]) is the place for general Python discussions, questions and the central meeting point of the community.

In order not to reinvent the wheel and discuss the same questions again and again, people will appreciate very much if you *do a web search for a solution to your problem before contacting these lists!*

8 `http://docs.python.org/tut/tut.html`
9 `http://docs.python.org/lib/lib.html`
10 `http://docs.python.org/ref/ref.html`
11 `http://mail.python.org/mailman/listinfo/tutor`
12 `http://www.python.org/community/lists/#python-help`
13 `http://groups.google.com/group/comp.lang.python/`

3 Hello, World

3.0.6 What you should know

You should know how to edit programs in a text editor or IDLE, save the file and run the file once the files have been saved to your disk.

3.0.7 Printing

Programming tutorials since the beginning of time have started with a little program called "Hello, World!"[1] The syntax changed in Python 3.0. If you are using Python 3.0, you should be reading Non-Programmer's Tutorial for Python 3[2] instead. So here is the Python 2.6 example:

```
print "Hello, World!"
```

If you are using the command line to run programs then type it in with a text editor, save it as `hello.py` and run it with `python hello.py`

Otherwise go into IDLE, create a new window, and create the program as in section Creating and Running Programs[3].

When this program is run here's what it prints:

```
Hello, World!
```

Now I'm not going to tell you this every time, but when I show you a program I recommend that you type it in and run it. I learn better when I type it in and you probably do too.

Now here is a more complicated program:

```
print "Jack and Jill went up a hill"
print "to fetch a pail of water;"
print "Jack fell down, and broke his crown,"
print "and Jill came tumbling after."
```

When you run this program it prints out:

1 List of "Hello, world!" programs in many programming languages ^{http://en.wikibooks.org/wiki/Computer%20Programming%2FHello%20world}

2 http://en.wikibooks.org/wiki/Non-Programmer%27s%20Tutorial%20for%20Python%203

3 Chapter 2.1.3 on page 7

```
Jack and Jill went up a hill
to fetch a pail of water;
Jack fell down, and broke his crown,
and Jill came tumbling after.
```

When the computer runs this program it first sees the line:

```
print "Jack and Jill went up a hill"
```

so the computer prints:

```
Jack and Jill went up a hill
```

Then the computer goes down to the next line and sees:

```
print "to fetch a pail of water;"
```

So the computer prints to the screen:

```
to fetch a pail of water;
```

The computer keeps looking at each line, follows the command and then goes on to the next line. The computer keeps running commands until it reaches the end of the program.

Terminology

Now is probably a good time to give you a bit of an explanation of what is happening - and a little bit of programming terminology.

What we were doing above was using a *command* called `print`. The `print` command is followed by one or more *arguments*. So in this example

```
print "Hello, World!"
```

there is one *argument*, which is `"Hello, World!"`. Note that this argument is a group of characters enclosed in double quotes ("). This is commonly referred to as a *string of characters*, or *string*, for short. Another example of a string is `"Jack and Jill went up a hill"`.

A command and its arguments are collectively referred to as a *statement*, so

```
print "Hello, World!"
```

is an example of a statement.

That's probably more than enough terminology for now.

3.0.8 Expressions

Here is another program:

```
print "2 + 2 is", 2 + 2
print "3 * 4 is", 3 * 4
print "100 - 1 is", 100 - 1
print "(33 + 2) / 5 + 11.5 is", (33 + 2) / 5 + 11.5
```

And here is the *output* when the program is run:

```
2 + 2 is 4
3 * 4 is 12
100 - 1 is 99
(33 + 2) / 5 + 11.5 is 18.5
```

As you can see, Python can turn your six hundred dollar computer into a 2 dollar calculator.

In this example, the print command is followed by two arguments, with each of the arguments separated by a comma. So with the first line of the program

```
print "2 + 2 is", 2 + 2
```

The first argument is the string `"2 + 2 is"` and the second argument is the *mathematical expression* `2 + 2`, which is commonly referred to as an *expression*.

What is important to note is that a string is printed as is (the string is what is within the double quotes but doesn't include the double quotes themselves. So the string is printed without the enclosing double quotes.) But an *expression* is *evaluated*, (in other words, converted) to its actual value.

Python has six basic operations for numbers:

Operation	**Symbol**	**Example**
Power (exponentiation)	`**`	`5 ** 2 == 25`
Multiplication	`*`	`2 * 3 == 6`
Division	`/`	`14 / 3 == 4`
Remainder (modulo)	`%`	`14 % 3 == 2`
Addition	`+`	`1 + 2 == 3`
Subtraction	`-`	`4 - 3 == 1`

Notice that division follows the rule, **if there are no decimals to start with, there will be no decimals to end with**. The following program shows this:

```
print "14 / 3 = ", 14 / 3
print "14 % 3 = ", 14 % 3
print
print "14.0 / 3.0 =", 14.0 / 3.0
print "14.0 % 3.0 =", 14.0 % 3.0
print
print "14.0 / 3 =", 14.0 / 3
print "14.0 % 3 =", 14.0 % 3
print
print "14 / 3.0 =", 14 / 3.0
print "14 % 3.0 =", 14 % 3.0
print
```

With the output:

```
14 / 3 = 4
14 % 3 = 2

14.0 / 3.0 = 4.66666666667
14.0 % 3.0 = 2.0

14.0 / 3 = 4.66666666667
14.0 % 3 = 2.0

14 / 3.0 = 4.66666666667
14 % 3.0 = 2.0
```

Notice how Python gives different answers for some problems depending on whether or not decimal values are used.

The order of operations is the same as in math:

- parentheses `()`
- exponents `**`
- multiplication `*`, division `/`, and remainder `%`
- addition `+` and subtraction `-`

So use parentheses to structure your formulas when needed.

3.0.9 Talking to humans (and other intelligent beings)

Often in programming you are doing something complicated and may not in the future remember what you did. When this happens, the program should probably be commented. A *comment* is a note to you and other programmers explaining what is happening. For example:

```
# Not quite PI, but an incredible simulation
print 22.0 / 7.0    # 355/113 is even more incredible rational approx
 to PI
```

Which outputs

```
3.14285714286
```

Notice that the comment starts with a hash: `#`. Comments are used to communicate with others who read the program and your future self to make clear what is complicated.

Note that any text can follow a comment, and that when the program is run, the text after the `#` through to the end of that line is ignored. The `#` does not have to be at the beginning of a new line:

```
# Output PI on the screen
print 22.0 / 7.0 # Well, just a good approximation
```

3.0.10 Examples

Each chapter (eventually) will contain examples of the programming features introduced in the chapter. You should at least look over them and see if you understand them. If you don't, you may want to type them in and see what happens. Mess around with them, change them and see what happens.

Denmark.py

```
print "Something's rotten in the state of Denmark."
print "                -- Shakespeare"
```

Output:

```
Something's rotten in the state of Denmark.
                -- Shakespeare
```

School.py

```
# This is not quite true outside of USA
# and is based on my dim memories of my younger years
print "First Grade"
print "1 + 1 =", 1 + 1
print "2 + 4 =", 2 + 4
print "5 - 2 =", 5 - 2
print
print "Third Grade"
print "243 - 23 =", 243 - 23
print "12 * 4 =", 12 * 4
print "12 / 3 =", 12 / 3
print "13 / 3 =", 13 / 3, "R", 13 % 3
print
print "Junior High"
print "123.56 - 62.12 =", 123.56 - 62.12
print "(4 + 3) * 2 =", (4 + 3) * 2
print "4 + 3 * 2 =", 4 + 3 * 2
print "3 ** 2 =", 3 ** 2
print
```

Output:

```
First Grade
1 + 1 = 2
2 + 4 = 6
5 - 2 = 3

Third Grade
243 - 23 = 220
12 * 4 = 48
12 / 3 = 4
13 / 3 = 4 R 1

Junior High
123.56 - 62.12 = 61.44
(4 + 3) * 2 = 14
4 + 3 * 2 = 10
3 ** 2 = 9
```

3.0.11 Exercises

1. Write a program that prints your full name and your birthday as separate strings.
2. Write a program that shows the use of all 6 math functions.

Solution

1. Write a program that prints your full name and your birthday as separate strings.

```
print "Ada Lovelace", "born on", "November 27, 1852"
```

2. Write a program that shows the use of all 6 math operations.

```
Anything along these lines is acceptable:

Addition
print "2 + 5 = ", 2 + 5

subtraction
print "9 - 3 = ", 9 - 3

multiplication
print "3 * 3 = ", 3 * 3

division
print "90 / 5 = ", 90 / 5

exponents
print "7 to the power of 2 (squared) = ", 7 ** 2

remainder
print "the remainder when doing 22 / 9 = ", 22 % 9
```

Footnotes

4 Who Goes There?

4.0.12 Input and Variables

Now I feel it is time for a really complicated program. Here it is:

```
print "Halt!"
user_reply = raw_input("Who goes there? ")
print "You may pass,", user_reply
```

When **I** ran it, here is what **my** screen showed:

```
Halt!
Who goes there? Josh
You may pass, Josh
```

Note: After running the code by pressing F5, the Python shell will only give output:

```
Halt!
Who goes there?
```

You need to enter your name in the Python shell, and then press Enter to get the rest of the output.

Of course when you run the program your screen will look different because of the `raw_input()` statement. When you ran the program you probably noticed (you did run the program, right?) how you had to type in your name and then press Enter. Then the program printed out some more text and also your name. This is an example of *input*. The program reaches a certain point and then waits for the user to input some data that the program can use later.

Of course, getting information from the user would be useless if we didn't have anywhere to put that information and this is where variables come in. In the previous program `user_reply` is a *variable*. Variables are like a box that can store some piece of data. Here is a program to show examples of variables:

```
a = 123.4
b23 = ,Spam,
first_name = "Bill"
b = 432
c = a + b
print "a + b is",c
print "first_name is",first_name
print "Sorted Parts, After Midnight or",b23
```

And here is the output:

```
a + b is 555.4
first_name is Bill
Sorted Parts, After Midnight or Spam
```

The variables in the above program are `a`, `b23`, `first_name`, `b`, and `c`. A variable in Python can store any type of data - in this example we stored some strings (e.g. "Bill") and some numbers (e.g. 432).

Note the difference between strings and variable names. Strings are marked with quotation marks, which tells the computer "don't try to understand, just take this text as it is":

```
print "first_name"
```

This would print the text:

```
first_name
```

as-is. Variable names are written without any quotation marks and instruct the computer "use the value I've previously stored under this name":

```
print first_name
```

which would print (after the previous example):

```
Bill
```

4.0.13 Assignment

Okay, so we have these boxes called variables and also data that can go into the variable. The computer will see a line like `first_name = "Bill"` and it reads it as "Put the string `Bill` into the box (or variable) `first_name`. Later on it sees the statement `c = a + b` and it reads it as "put the sum of `a + b` or `123.4 + 432` which equals `555.4` into `c`". The right hand side of the statement (`a + b`) is *evaluated* and the result is stored in the variable on the left hand side (`c`). This is called *assignment*, and you should not confuse the assignment equal sign (`=`) with "equality" in a mathematical sense here (that's what `==` will be used for later).

Here is another example of variable usage:

```
a = 1
print a
a = a + 1
print a
a = a * 2
print a
```

And of course here is the output:

```
1
2
4
```

Even if it is the same variable on both sides the computer still reads it as "First find out the data to store and then find out where the data goes".

One more program before I end this chapter:

```
number = input("Type in a number: ")
text = raw_input("Type in a string: ")
print "number =", number
print "number is a", type(number)
print "number * 2 =", number * 2
print "text =", text
print "text is a", type(text)
print "text * 2 =", text * 2
```

The output I got was:

```
Type in a Number: 12.34
Type in a String: Hello
number = 12.34
number is a <type 'float'>
number * 2 = 24.68
text = Hello
text is a <type 'str'>
text * 2 = HelloHello
```

Notice that `number` was gotten with `input()` while `text` was gotten with `raw_input()`. `raw_input()` returns a string while `input()` returns a number. When you want the user to type in a number use `input()` but if you want the user to type in a string use `raw_input()`.

Only use `input()` when you trust your users with the computer the program runs on.

Everything entered into `input()` dialog is evaluated as a Python expression and thus allows the user to take control of your program. For example, entering `__import__('os').system('dir')` executes the `dir` command. You should instead get a string and convert it to the neccessary type like `int(raw_input())` or `float(raw_input())`.

The second half of the program uses `type()` which tells what a variable is. Numbers are of type `int` or `float`, which are short for *integer* and *floating point* (mostly used for decimal numbers), respectively. Text strings are of type `str`, short for *string*. Integers and floats can be worked on by mathematical functions, strings cannot. Notice how when python multiplies a number by an integer the expected thing happens. However when a string is multiplied by an integer the result is that multiple copies of the string are produced (i.e., `text * 2 = HelloHello`).

The operations with strings do different things than operations with numbers. Here are some interactive mode examples to show that some more.

```
>>> "This" + " " + "is" + " joined."
'This is joined.'
>>> "Ha, " * 5
'Ha, Ha, Ha, Ha, Ha, '
>>> "Ha, " * 5 + "ha!"
'Ha, Ha, Ha, Ha, Ha, ha!'
>>>
```

This could also be done as a program:

```
print "This" + " " + "is" + " joined."
print "Ha, " * 5
print "Ha, " * 5 + "ha!"
```

Here is the list of some string operations:

Operation	**Symbol**	**Example**
Repetition	`*`	`"i" * 5 == "iiiii"`
Concatenation	`+`	`"Hello, " + "World!" == "Hello, World!"`

4.0.14 Examples

Rate_times.py

```
# This program calculates rate and distance problems
print "Input a rate and a distance"
rate = input("Rate: ")
distance = input("Distance: ")
print "Time:", (distance / rate)
```

Sample runs:

```
Input a rate and a distance
Rate: 5
Distance: 10
Time: 2
```

```
Input a rate and a distance
Rate: 3.52
Distance: 45.6
Time: 12.9545454545
```

Area.py

```
# This program calculates the perimeter and area of a rectangle
print "Calculate information about a rectangle"
length = input("Length: ")
```

```
width = input("Width: ")
print "Area", length * width
print "Perimeter", 2 * length + 2 * width
```

Sample runs:

```
Calculate information about a rectangle
Length: 4
Width: 3
Area 12
Perimeter 14
```

```
Calculate information about a rectangle
Length: 2.53
Width: 5.2
Area 13.156
Perimeter 15.46
```

temperature.py

```
# Converts Fahrenheit to Celsius
temp = input("Fahrenheit temperature: ")
print (temp - 32.0) * 5.0 / 9.0
```

Sample runs:

```
Fahrenheit temperature: 32
0.0
```

```
Fahrenheit temperature: -40
-40.0
```

```
Fahrenheit temperature: 212
100.0
```

```
Fahrenheit temperature: 98.6
37.0
```

4.0.15 Exercises

Write a program that gets 2 string variables and 2 integer variables from the user, concatenates (joins them together with no spaces) and displays the strings, then multiplies the two numbers on a new line.

Solution

Write a program that gets 2 string variables and 2 integer variables from the user, concatenates (joins them together with no spaces) and displays the strings, then multiplies the two numbers on a new line.

```
string1 = raw_input(‚String 1: ‚)
string2 = raw_input(‚String 2: ‚)
int1 = input(‚Integer 1: ‚)
int2 = input(‚Integer 2: ‚)
print string1 + string2
print int1 * int2
```

Another Solution

```
print "this is an exercise"
number_1 = input("please input the first number: ")
number_2 = input("Please input the second number: ")

string_1 = raw_input("Please input the first half of the word: ")
string_2 = raw_input("please input the second half of the word: ")

print "the word you input is ‚" + string_1 + string_2 + "‚"
print "the result of the 2 numbers is:", number_1 * number_2
```

5 Count to 10

5.0.16 While loops

Here we present our first *control structure*. Ordinarily the computer starts with the first line and then goes down from there. Control structures change the order that statements are executed or decide if a certain statement will be run. Here's the source for a program that uses the `while` control structure:

```
a = 0
while a < 10:
    a = a + 1
    print a
```

And here is the extremely exciting output:

```
1
2
3
4
5
6
7
8
9
10
```

And you thought it couldn't get any worse after turning your computer into a five dollar calculator?

So what does the program do? First it sees the line `a = 0` and sets `a` to zero. Then it sees `while a < 10:` and so the computer checks to see if `a < 10`. The first time the computer sees this statement `a` is zero so it is less than 10. In other words as long as *a* is less than ten the computer will run the tabbed in statements. This eventually makes *a* equal to ten (by adding one to *a* again and again), and the `while a < 10` is not true any longer. Reaching that point the program will not run the indented lines any longer.

Always remember to put a colon ":" after the "while" statement!

Here is another example of the use of `while`:

```
a = 1
s = 0
print ‚Enter Numbers to add to the sum.‚
print ‚Enter 0 to quit.‚
while a != 0:
    print ‚Current Sum:‚, s
    a = input(‚Number? ‚)
    s = s + a
print ‚Total Sum =‚, s
```

```
Enter Numbers to add to the sum.
Enter 0 to quit.
Current Sum: 0
Number? 200
Current Sum: 200
Number? -15.25
Current Sum: 184.75
Number? -151.85
Current Sum: 32.9
Number? 10.00
Current Sum: 42.9
Number? 0
Total Sum = 42.9
```

Notice how `print 'Total Sum =', s` is only run at the end. The `while` statement only affects the lines that are indented with whitespace. The `!=` means "does not equal" so "`while a != 0:`" means: "until *a* is zero, run the tabbed statements that follow."

Infinite loops

Now that we have while loops, it is possible to have programs that run forever. An easy way to do this is to write a program like this:

```
while 1 == 1:
   print "Help, I,m stuck in a loop."
```

The "==" operator is used to test equality of the expressions on the two sides of the operator, just as "<" was used for "less than" before (you will get a complete list of all comparison operators in the next chapter).

This program will output `Help, I'm stuck in a loop.` until the heat death of the universe or until you stop it, because 1 will forever be equal to 1. The way to stop it is to hit the Control (or *Ctrl*) button and *C* (the letter) at the same time. This will kill the program. (Note: sometimes you will have to hit enter after the Control-C.)

5.0.17 Examples

Fibonacci.py

```
# This program calculates the Fibonacci sequence
a = 0
b = 1
count = 0
max_count = 20
while count < max_count:
    count = count + 1
    # we need to keep track of a since we change it
    old_a = a
    old_b = b
    a = old_b
    b = old_a + old_b
    # Notice that the , at the end of a print statement keeps it
    # from switching to a new line
    print old_a,
```

Output:

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

Note the output on a single line by use of a comma at the end of the `print` statement.

Password.py

```
# Waits until a password has been entered.  Use Control-C to break
 out without
# the password

# Note that this must not be the password so that the
# while loop runs at least once.
password = "no password"

# note that != means not equal
while password != "unicorn":
    password = raw_input("Password: ")
print "Welcome in"
```

Sample run:

```
Password: auo
Password: y22
Password: password
Password: open sesame
Password: unicorn
Welcome in
```

5.0.18 Exercises

Write a program that asks the user for a Login Name and password. Then when they type "lock", they need to type in their name and password to unlock the program.

Solution

Write a program that asks the user for a Login Name and password. Then when they type "lock", they need to type in their name and password to unlock the program.

```
name = raw_input("What is your UserName: ")
password = raw_input("What is your Password: ")
print "To lock your computer type lock."
command = ""
input1 = ""
input2 = ""
while command != "lock":
    command = raw_input("What is your command: ")
while input1 != name:
    input1 = raw_input("What is your username: ")
```

```
while input2 != password:
    input2 = raw_input("What is your password: ")
print "Welcome back to your system!"
```

If you would like the program to run continuously, just add a `while 1 == 1:` loop around the whole thing. You will have to indent the rest of the program when you add this at the top of the code, but don't worry, you don't have to do it manually for each line! Just highlight everything you want to indent and click on "Indent" under "Format" in the top bar of the python window. Note that you can use empty strings like this: `""`.

Another way of doing this could be:

```
name = raw_input(‚Set name: ‚)
password = raw_input(‚Set password: ‚)
while 1 == 1:
    nameguess=passwordguess=key=""    multiple assignment
    while (nameguess != name) or (passwordguess != password):
        nameguess = raw_input(‚Name? ‚)
        passwordguess = raw_input(‚Password? ‚)
    print "Welcome,", name, ". Type lock to lock."
    while key != "lock":
        key = raw_input("")
```

Notice the `or` in `while (name != "user") or (password != "pass"):`, which we haven't yet introduced. You can probably figure out how it works.

```
login = "john"
password = "tucker"
logged=2

while logged != 0:
    while login != "Phil":
            login = raw_input("Login : ")
    while password != "McChicken":
            password = raw_input("Password: ")
    logged = 1

    print "Welcome!"
    print "To leave type lock "

    while logged == 1:
        leave = raw_input (">> ")
        if leave == "lock":
            logged = 0
```

```
print "Goodbye!!"
```

This method, although a bit more crude also works. Notice it uses the as of yet un-introduced `if` function.

6 Decisions

6.0.19 If statement

As always I believe I should start each chapter with a warm-up typing exercise, so here is a short program to compute the absolute value of a number:

```
n = input("Number? ")
if n < 0:
   print "The absolute value of", n, "is", -n
else:
   print "The absolute value of", n, "is", n
```

Here is the output from the two times that I ran this program:

```
Number? -34
The absolute value of -34 is 34
```

```
Number? 1
The absolute value of 1 is 1
```

So what does the computer do when it sees this piece of code? First it prompts the user for a number with the statement "`n = input("Number?  ")`". Next it reads the line "`if n < 0:`". If `n` is less than zero Python runs the line "`print "The absolute value of", n, "is", -n`". Otherwise it runs the line "`print "The absolute value of", n, "is", n`".

More formally Python looks at whether the *expression* `n < 0` is true or false. An `if` statement is followed by an indented *block* of statements that are run when the expression is true. Optionally after the `if` statement is an `else` statement and another indented *block* of statements. This second block of statements is run if the expression is false.

There are a number of different tests that an expression can have. Here is a table of all of them:

operator	**function**
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`==`	equal
`!=`	not equal
`<>`	another way to say not equal (old style, not recommended)

Another feature of the `if` command is the `elif` statement. It stands for else if and means if the original `if` statement is false but the `elif` part is true, then do the `elif` part. And if neither the `if` or `elif` expressions are true, then do what's in the `else` block. Here's an example:

```
a = 0
while a < 10:
    a = a + 1
    if a > 5:
        print a, ">", 5
    elif a <= 7:
        print a, "<=", 7
    else:
        print "Neither test was true"
```

and the output:

```
1 <= 7
2 <= 7
3 <= 7
4 <= 7
5 <= 7
6 > 5
7 > 5
8 > 5
9 > 5
10 > 5
```

Notice how the `elif a <= 7` is only tested when the `if` statement fails to be true. There can be more than one `elif` expression, allowing multiple tests to be done in a single `if` statement.

6.0.20 Examples

```
# This Program Demonstrates the use of the == operator
# using numbers
print 5 == 6
# Using variables
x = 5
y = 8
print x == y
```

And the output

```
False
False
```

High_low.py

```
# Plays the guessing game higher or lower

# This should actually be something that is semi random like the
# last digits of the time or something else, but that will have to
# wait till a later chapter.  (Extra Credit, modify it to be random
# after the Modules chapter)
number = 78
guess = 0

while guess != number:
    guess = input("Guess a number: ")
    if guess > number:
        print "Too high"
    elif guess < number:
        print "Too low"

print "Just right"
```

Sample run:

```
Guess a number: 100
Too high
Guess a number: 50
Too low
Guess a number: 75
Too low
Guess a number: 87
Too high
Guess a number: 81
Too high
Guess a number: 78
Just right
```

even.py

```
# Asks for a number.
# Prints if it is even or odd

number = input("Tell me a number: ")
if number % 2 == 0:
    print number, "is even."
elif number % 2 == 1:
    print number, "is odd."
else:
    print number, "is very strange."
```

Sample runs:

```
Tell me a number: 3
3 is odd.
```

```
Tell me a number: 2
2 is even.
```

```
Tell me a number: 3.14159
3.14159 is very strange.
```

average1.py

```
# keeps asking for numbers until 0 is entered.
# Prints the average value.

count = 0
sum = 0.0
number = 1 # set to something that will not exit the while loop
 immediately.

print "Enter 0 to exit the loop"
```

```
while number != 0:
    number = input("Enter a number: ")
    if number != 0:
        count = count + 1
        sum = sum + number

print "The average was:", sum / count
```

Sample runs:

```
Enter 0 to exit the loop
Enter a number: 3
Enter a number: 5
Enter a number: 0
The average was: 4.0
```

```
Enter 0 to exit the loop
Enter a number: 1
Enter a number: 4
Enter a number: 3
Enter a number: 0
The average was: 2.66666666667
```

average2.py

```
# keeps asking for numbers until count numbers have been entered.
# Prints the average value.

sum = 0.0

print "This program will take several numbers then average them"
count = input("How many numbers would you like to average: ")
current_count = 0

while current_count < count:
    current_count = current_count + 1
    print "Number", current_count
    number = input("Enter a number: ")
    sum = sum + number

print "The average was:", sum / count
```

Sample runs:

```
This program will take several numbers then average them
How many numbers would you like to average: 2
Number 1
Enter a number: 3
Number 2
Enter a number: 5
The average was: 4.0
```

```
This program will take several numbers then average them
How many numbers would you like to average: 3
Number 1
Enter a number: 1
Number 2
Enter a number: 4
Number 3
```

```
Enter a number: 3
The average was: 2.66666666667
```

6.0.21 Exercises

Modify the higher or lower program from this section to keep track of how many times the user has entered the wrong number. If it is more than 3 times, print "That must have been complicated." Note that the program does not have to quit asking for the number before it is guessed, it just has to print this after the number is guessed.

Write a program that asks for two numbers. If the sum of the numbers is greater than 100, print "That is a big number."

Write a program that asks the user their name, if they enter your name say "That is a nice name", if they enter "John Cleese" or "Michael Palin", tell them how you feel about them ;), otherwise tell them "You have a nice name."

Solution

Modify the higher or lower program from this section to keep track of how many times the user has entered the wrong number. If it is more than 3 times, print "That must have been complicated."

```
number = 42
guess = 0
count = 0
while guess != number:
    count = count + 1
    guess = input(‚Guess a number: ‚)
    if guess > number:
        print ‚Too high‚
    elif guess < number:
        print ‚Too low‚
    else:
        print ‚Just right‚
        break
    if count > 2:
        print ‚That must have been complicated.‚
        break
```

Write a program that asks for two numbers. If the sum of the numbers is greater than 100, print "That is a big number."

```
number1 = input(,1st number: ,)
number2 = input(,2nd number: ,)
if number1 + number2 > 100:
    print ,That is a big number.,
```

Write a program that asks the user their name, if they enter your name say "That is a nice name", if they enter "John Cleese" or "Michael Palin", tell them how you feel about them ;), otherwise tell them "You have a nice name."

```
name = raw_input(,Your name: ,)
if name == ,Ada,:
    print ,That is a nice name.,
elif name == ,John Cleese, or name == ,Michael Palin,:
    print ,... some funny text.,
else:
    print ,You have a nice name.,
```

7 Debugging

7.0.22 What is debugging?

"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs." — *Maurice Wilkes discovers debugging*, 1949

By now if you have been messing around with the programs you have probably found that sometimes the program does something you didn't want it to do. This is fairly common. Debugging is the process of figuring out what the computer is doing and then getting it to do what you want it to do. This can be tricky. I once spent nearly a week tracking down and fixing a bug that was caused by someone putting an `x` where a `y` should have been.

This chapter will be more abstract than previous chapters.

7.0.23 What should the program do?

The first thing to do (this sounds obvious) is to figure out what the program should be doing if it is running correctly. Come up with some test cases and see what happens. For example, let's say I have a program to compute the perimeter of a rectangle (the sum of the length of all the edges). I have the following test cases:

height	**width**	**perimeter**
3	4	14
2	3	10
4	4	16
2	2	8
5	1	12

I now run my program on all of the test cases and see if the program does what I expect it to do. If it doesn't then I need to find out what the computer is doing.

More commonly some of the test cases will work and some will not. If that is the case you should try and figure out what the working ones have in common. For example here is the output for a perimeter program (you get to see the code in a minute):

```
Height: 3
Width: 4
perimeter = 15
```

```
Height: 2
Width: 3
perimeter = 11
```

```
Height: 4
Width: 4
perimeter = 16
```

```
Height: 2
Width: 2
perimeter = 8
```

```
Height: 5
Width: 1
perimeter = 8
```

Notice that it didn't work for the first two inputs, it worked for the next two and it didn't work on the last one. Try and figure out what is in common with the working ones. Once you have some idea what the problem is finding the cause is easier. With your own programs you should try more test cases if you need them.

7.0.24 What does the program do?

The next thing to do is to look at the source code. One of the most important things to do while programming is reading source code. The primary way to do this is code walkthroughs.

A code walkthrough starts at the first line, and works its way down until the program is done. `While` loops and `if` statements mean that some lines may never be run and some lines are run many times. At each line you figure out what Python has done.

Lets start with the simple perimeter program. Don't type it in, you are going to read it, not run it. The source code is:

```
height = input("Height: ")
width = input("Width: ")
print "perimeter =", width + height + width + width
```

Question: **What is the first line Python runs?**

Answer: The first line is always run first. In this case it is: `height = input("Height: ")`

What does that line do?

Prints `Height: `, waits for the user to type a number in, and puts that in the variable height.

What is the next line that runs?

In general, it is the next line down which is: `width = input("Width: ")`

What does that line do?

Prints `Width: `, waits for the user to type a number in, and puts what the user types in the variable width.

What is the next line that runs?

When the next line is not indented more or less than the current line, it is the line right afterwards, so it is: `print "perimeter = ", width + height + width + width` (It may also run a function in the current line, but that's a future chapter.) What does that line do?

First it prints `perimeter = `, then it prints `width + height + width + width`.

Does `width + height + width + width` calculate the perimeter properly?

Let's see, perimeter of a rectangle is the bottom (width) plus the left side (height) plus the top (width) plus the right side (huh?). The last item should be the right side's length, or the height.

Do you understand why some of the times the perimeter was calculated "correctly"?

It was calculated correctly when the width and the height were equal.

The next program we will do a code walkthrough for is a program that is supposed to print out 5 dots on the screen. However, this is what the program is outputting:

```
. . . .
```

And here is the program:

```
number = 5
while number > 1:
    print ".",
    number = number - 1
print
```

This program will be more complex to walkthrough since it now has indented portions (or control structures). Let us begin.

What is the first line to be run?

The first line of the file: `number = 5`

What does it do?

Puts the number 5 in the variable number.

What is the next line?

The next line is: `while number > 1:`

What does it do?

Well, `while` statements in general look at their expression, and if it is true they do the next indented block of code, otherwise they skip the next indented block of code.

So what does it do right now?

If `number > 1` is true then the next two lines will be run.

So is `number > 1`?

The last value put into `number` was `5` and `5 > 1` so yes.

So what is the next line?

Since the `while` was true the next line is: `print ".",`

What does that line do?

Prints one dot and since the statement ends with a ',' the next print statement will not be on a different screen line.

What is the next line?

`number = number - 1` since that is following line and there are no indent changes.

What does it do?

It calculates `number - 1`, which is the current value of `number` (or 5) subtracts 1 from it, and makes that the new value of number. So basically it changes `number`'s value from 5 to 4.

What is the next line?

Well, the indent level decreases so we have to look at what type of control structure it is. It is a `while` loop, so we have to go back to the `while` clause which is `while number > 1:`

What does it do?

It looks at the value of number, which is 4, and compares it to 1 and since `4 > 1` the while loop continues.

What is the next line?

Since the while loop was true, the next line is: `print ".",`

What does it do?

It prints a second dot on the line.

What is the next line?

No indent change so it is: `number = number - 1`

And what does it do?

It takes the current value of number (4), subtracts 1 from it, which gives it 3 and then finally makes 3 the new value of number.

What is the next line?

Since there is an indent change caused by the end of the while loop, the next line is: `while number > 1:`

What does it do?

It compares the current value of number (3) to 1. `3 > 1` so the while loop continues.

What is the next line?

Since the while loop condition was true the next line is: `print ".",`

And it does what?

A third dot is printed on the line.

What is the next line?

It is: `number = number - 1`

What does it do?

It takes the current value of number (3) subtracts from it 1 and makes the 2 the new value of number.

What is the next line?

Back up to the start of the while loop: `while number > 1:`

What does it do?

It compares the current value of number (2) to 1. Since `2 > 1` the while loop continues.

What is the next line?

Since the while loop is continuing: `print ".",`

What does it do?

It discovers the meaning of life, the universe and everything. I'm joking. (I had to make sure you were awake.) The line prints a fourth dot on the screen.

What is the next line?

It's: `number = number - 1`

What does it do?

Takes the current value of number (2) subtracts 1 and makes 1 the new value of number.

What is the next line?

Back up to the while loop: `while number > 1:`

What does the line do?

It compares the current value of number (1) to 1. Since `1 > 1` is false (one is not greater than one), the while loop exits.

What is the next line?

Since the while loop condition was false the next line is the line after the while loop exits, or: `print`

What does that line do?

Makes the screen go to the next line.

Why doesn't the program print 5 dots?

The loop exits 1 dot too soon.

How can we fix that?

Make the loop exit 1 dot later.

And how do we do that?

There are several ways. One way would be to change the while loop to: `while number > 0:` Another way would be to change the conditional to: `number >= 1` There are a couple others.

7.0.25 How do I fix the program?

You need to figure out what the program is doing. You need to figure out what the program should do. Figure out what the difference between the two is. Debugging is a skill that has to be practiced to be learned. If you can't figure it out after an hour, take a break, talk to someone about the problem or contemplate the lint in your navel. Come back in a while and you will probably have new ideas about the problem. Good luck.

8 Defining Functions

8.0.26 Creating Functions

To start off this chapter I am going to give you an example of what you could do but shouldn't (so don't type it in):

```
a = 23
b = -23

if a < 0:
    a = -a
if b < 0:
    b = -b
if a == b:
    print "The absolute values of", a, "and", b, "are equal"
else:
    print "The absolute values of", a, "and", b, "are different"
```

with the output being:

```
The absolute values of 23 and 23 are equal
```

The program seems a little repetitive. Programmers hate to repeat things -- that's what computers are for, after all! (Note also that finding the absolute value changed the value of the variable, which is why it is printing out 23, and not -23 in the output.) Fortunately Python allows you to create functions to remove duplication. Here is the rewritten example:

```
def absolute_value(n):
    if n < 0:
        n = -n
    return n

a = 23
b = -23

if absolute_value(a) == absolute_value(b):
    print "The absolute values of", a, "and", b, "are equal"
else:
    print "The absolute values of", a, "and", b, "are different"
```

with the output being:

```
The absolute values of 23 and -23 are equal
```

The key feature of this program is the `def` statement. `def` (short for define) starts a function definition. `def` is followed by the name of the function `absolute_value`. Next comes a '(' followed by the parameter `n` (`n` is passed from the program into the function when the function is called). The statements after the ':' are executed when the function is used. The statements continue until either the indented statements end or a `return` is encountered. The `return` statement returns a value back to the place where the function was called.

Notice how the values of `a` and `b` are not changed. Functions can be used to repeat tasks that don't return values. Here are some examples:

```
def hello():
    print "Hello"

def area(w, h):
    return w * h

def print_welcome(name):
    print "Welcome", name

hello()
hello()

print_welcome("Fred")
w = 4
h = 5
print "width =", w, "height =", h, "area =", area(w, h)
```

with output being:

```
Hello
Hello
Welcome Fred
width = 4 height = 5 area = 20
```

That example shows some more stuff that you can do with functions. Notice that you can use no arguments or two or more. Notice also when a function doesn't need to send back a value, a return is optional.

8.0.27 Variables in functions

When eliminating repeated code, you often have variables in the repeated code. In Python, these are dealt with in a special way. So far all variables we have seen are global variables. Functions have a special type of variable called local variables. These variables only exist while the function is running. When a local variable has the same name as another variable (such as a global variable), the local variable hides the other. Sound confusing? Well, these next examples (which are a bit contrived) should help clear things up.

```
a = 4

def print_func():
    a = 17
    print "in  print_func a = ", a

print_func()
print "a = ", a
```

When run, we will receive an output of:

```
in print_func a = 17
a = 4
```

Variable assignments inside a function do not override global variables, they exist only inside the function. Even though `a` was assigned a new value inside the function, this newly assigned value was only relevant to `print_func`, when the function finishes running, and the `a`'s values is printed again, we see the originally assigned values.

8.0.28 Complex example

```
a_var = 10
b_var = 15
e_var = 25

def a_func(a_var):
    print "in a_func a_var = ", a_var
    b_var = 100 + a_var
    d_var = 2 * a_var
    print "in a_func b_var = ", b_var
    print "in a_func d_var = ", d_var
    print "in a_func e_var = ", e_var
    return b_var + 10

c_var = a_func(b_var)

print "a_var = ", a_var
print "b_var = ", b_var
print "c_var = ", c_var
print "d_var = ", d_var
```

The output is:

```
 in a_func a_var =  15
 in a_func b_var =  115
 in a_func d_var =  30
 in a_func e_var =  25
 a_var =  10
 b_var =  15
 c_var =  125
 d_var =

 Traceback (most recent call last):
  File "C:\Python24\def2", line 19, in -toplevel-
     print "d_var = ", d_var

 NameError: name 'd_var' is not defined
```

In this example the variables `a_var`, `b_var`, and `d_var` are all local variables when they are inside the function `a_func`. After the statement `return b_var + 10` is run, they all cease to exist. The variable `a_var` is automatically a local variable since it is a parameter name. The variables `b_var` and `d_var` are local variables since they appear on the left of an equals sign in the function in the statements `b_var = 100 + a_var` and `d_var = 2 * a_var` .

Inside of the function `a_var` has no value assigned to it. When the function is called with `c_var = a_func(b_var)`, 15 is assigned to `a_var` since at that point in time `b_var` is 15, making the call to the function `a_func(15)`. This ends up setting `a_var` to 15 when it is inside of `a_func`.

As you can see, once the function finishes running, the local variables `a_var` and `b_var` that had hidden the global variables of the same name are gone. Then the statement `print "a_var = ", a_var` prints the value `10` rather than the value `15` since the local variable that hid the global variable is gone.

Another thing to notice is the `NameError` that happens at the end. This appears since the variable `d_var` no longer exists since `a_func` finished. All the local variables are deleted when the function exits. If you want to get something from a function, then you will have to use `return something`.

One last thing to notice is that the value of `e_var` remains unchanged inside `a_func` since it is not a parameter and it never appears on the left of an equals sign inside of the function `a_func`. When a global variable is accessed inside a function it is the global variable from the outside.

Functions allow local variables that exist only inside the function and can hide other variables that are outside the function.

8.0.29 Examples

temperature2.py

```
# converts temperature to fahrenheit or celsius

def print_options():
    print "Options:"
    print " ,p, print options"
    print " ,c, convert from celsius"
    print " ,f, convert from fahrenheit"
    print " ,q, quit the program"

def celsius_to_fahrenheit(c_temp):
    return 9.0 / 5.0 * c_temp + 32

def fahrenheit_to_celsius(f_temp):
    return (f_temp - 32.0) * 5.0 / 9.0

choice = "p"
while choice != "q":
    if choice == "c":
        temp = input("Celsius temperature: ")
        print "Fahrenheit:", celsius_to_fahrenheit(temp)
    elif choice == "f":
        temp = input("Fahrenheit temperature: ")
        print "Celsius:", fahrenheit_to_celsius(temp)
    elif choice != "q":
        print_options()
    choice = raw_input("option: ")
```

Sample Run:

```
Options:
 'p' print options
 'c' convert from celsius
 'f' convert from fahrenheit
 'q' quit the program
option: c
Celsius temperature: 30
Fahrenheit: 86.0
option: f
Fahrenheit temperature: 60
Celsius: 15.5555555556
option: q
```

area2.py

```
# By Amos Satterlee
print
def hello():
    print "Hello!"

def area(width, height):
    return width * height

def print_welcome(name):
    print "Welcome,", name

name = raw_input("Your Name: ")
hello(),
print_welcome(name)
print
print "To find the area of a rectangle,"
print "enter the width and height below."
print
w = input("Width: ")
while w <= 0:
    print "Must be a positive number"
    w = input("Width: ")

h = input("Height: ")
while h <= 0:
    print "Must be a positive number"
    h = input("Height: ")

print "Width =", w, "Height =", h, "so Area =", area(w, h)
```

Sample Run:

```
Your Name: Josh
Hello!
Welcome, Josh

To find the area of a rectangle,
enter the width and height below.

Width: -4
Must be a positive number
Width: 4
Height: 3
Width = 4 Height = 3 so Area = 12
```

8.0.30 Exercises

Rewrite the area2.py program from the Examples above to have a separate function for the area of a square, the area of a rectangle, and the area of a circle (`3.14 * radius ** 2`). This program should include a menu interface.

Solution

Rewrite the area2.py program from the Examples above to have a separate function for the area of a square, the area of a rectangle, and the area of a circle (`3.14 * radius ** 2`). This program should include a menu interface.

```
def square(length):
    return length * length

def rectangle(width , height):
    return width * height

def circle(radius):
    return 3.14 * radius ** 2

def options():
    print
    print "Options:"
    print "s = calculate the area of a square."
    print "c = calculate the area of a circle."
    print "r = calculate the area of a rectangle."
    print "q = quit"
    print

print "This program will calculate the area of a square, circle or
 rectangle."
choice = "x"
options()
while choice != "q":
    choice = raw_input("Please enter your choice: ")
    if choice == "s":
        length = input("Length of square: ")
        print "The area of this square is", square(length)
        options()
    elif choice == "c":
        radius = input("Radius of the circle: ")
        print "The area of the circle is", circle(radius)
        options()
    elif choice == "r":
        width = input("Width of the rectangle: ")
        height = input("Height of the rectangle: ")
        print "The area of the rectangle is", rectangle(width,
height)
        options()
    elif choice == "q":
        print "",
    else:
        print "Unrecognized option."
        options()
```

9 Advanced Functions Example

Some people find this section useful, and some find it confusing. If you find it confusing you can skip it (or just look at the examples.) Now we will do a walk through for the following program:

```
def mult(a, b):
    if b == 0:
        return 0
    rest = mult(a, b - 1)
    value = a + rest
    return value
print "3 * 2 = ", mult(3, 2)
```

Output

```
3 * 2 =  6
```

Basically this program creates a positive integer multiplication function (that is far slower than the built in multiplication function) and then demonstrates this function with a use of the function. This program demonstrates the use of recursion, that is a form of iteration (repetition) in which there is a function that repeatedly calls itself until an exit condition is satisfied. It uses repeated additions to give the same result as mutiplication: e.g. 3 + 3 (addition) gives the same result as 3 * 2 (multiplication).

RUN 1

***Question:* What is the first thing the program does?**

Answer: The first thing done is the function mult is defined with all the lines except the last one.

function mult defined

```
def mult(a, b):
    if b == 0:
        return 0
    rest = mult(a, b - 1)
    value = a + rest
    return value
```

This creates a function that takes two parameters and returns a value when it is done. Later this function can be run.

What happens next?

The next line after the function, `print "3 * 2 = ", mult(3, 2)` is run.

And what does this do?

It prints `3 * 2 = ` and the return value of `mult(3, 2)`

And what does `mult(3, 2)` return?

We need to do a walkthrough of the `mult` function to find out.

RUN 2

What happens next?

The variable `a` gets the value 3 assigned to it and the variable `b` gets the value 2 assigned to it.

And then?

The line `if b == 0:` is run. Since `b` has the value 2 this is false so the line `return 0` is skipped.

And what then?

The line `rest = mult(a, b - 1)` is run. This line sets the local variable `rest` to the value of `mult(a, b - 1)`. The value of `a` is 3 and the value of `b` is 2 so the function call is `mult(3,1)`

So what is the value of `mult(3, 1)` ?

We will need to run the function `mult` with the parameters 3 and 1.

RUN 2

```
def mult(3, 2):
    if b == 0:
        return 0
    rest = mult(3, 2 - 1)
    rest = mult(3, 1)
    value = 3 + rest
    return value
```

RUN 3

So what happens next?

The local variables in the *new* run of the function are set so that `a` has the value 3 and `b` has the value 1. Since these are local values these do not affect the previous values of `a` and `b`.

And then?

Since `b` has the value 1 the if statement is false, so the next line becomes `rest = mult(a, b - 1)`.

What does this line do?

This line will assign the value of `mult(3, 0)` to rest.

So what is that value?

We will have to run the function one more time to find that out. This time `a` has the value 3 and `b` has the value 0.

So what happens next?

The first line in the function to run is `if b == 0:`. `b` has the value 0 so the next line to run is `return 0`

And what does the line `return 0` do?

This line returns the value 0 out of the function.

So?

So now we know that `mult(3, 0)` has the value 0. Now we know what the line `rest = mult(a, b - 1)` did since we have run the function `mult` with the parameters 3 and 0. We have finished running `mult(3, 0)` and are now back to running `mult(3, 1)`. The variable `rest` gets assigned the value 0.

What line is run next?

The line `value = a + rest` is run next. In this run of the function, `a = 3` and `rest = 0` so now `value = 3`.

What happens next?

The line `return value` is run. This returns 3 from the function. This also exits from the run of the function `mult(3, 1)`. After `return` is called, we go back to running `mult(3, 2)`.

Where were we in `mult(3, 2)`?

We had the variables `a = 3` and `b = 2` and were examining the line `rest = mult(a, b - 1)`.

So what happens now?

The variable `rest` get 3 assigned to it. The next line `value = a + rest` sets `value` to `3 + 3` or 6.

So now what happens?

The next line runs, this returns 6 from the function. We are now back to running the line `print "3 * 2 = ", mult(3, 2)` which can now print out the 6.

What is happening overall?

Basically we used two facts to calculate the multiple of the two numbers. The first is that any number times 0 is 0 (`x * 0 = 0`). The second is that a number times another number is equal to the first number plus the first number times one less than the second number (`x * y = x + x * (y - 1)`). So what happens is `3 * 2` is first converted into `3 + 3 * 1`. Then `3 * 1` is converted into `3 + 3 * 0`. Then we know that any number times 0 is 0 so `3 * 0` is 0. Then we can calculate that `3 + 3 * 0` is `3 + 0` which is `3`. Now we know what `3 * 1` is so we can calculate that `3 + 3 * 1` is `3 + 3` which is `6`.

This is how the whole thing works:

```
3 * 2
3 + 3 * 1
3 + 3 + 3 * 0
3 + 3 + 0
3 + 3
6
```

Should you still have problems with this example, look at the process backwards. What is the last step that happens? We can easily make out that the result of `mult(3, 0)` is `0`. Since `b` is `0`, the function `mult(3, 0)` will return `0` and stop.

So what does the previous step do? `mult(3, 1)` does not return `0` because `b` is not `0`. So the next lines are executed: `rest = mult (a, b - 1)`, which is `rest = mult (3, 0)`, which is `0` as we just worked out. So now the variable `rest` is set to `0`.

The next line adds the value of `rest` to `a`, and since `a` is `3` and `rest` is `0`, the result is `3`.

Now we know that the function `mult(3, 1)` returns `3`. But we want to know the result of `mult(3,2)`. Therefore, we need to jump back to the start of the program and execute it one more round: `mult(3, 2)` sets `rest` to the result of `mult(3, 1)`. We know from the last round that this result is `3`. Then `value` calculates as `a + rest`, i. e. `3 + 3`. Then the result of 3 * 2 is printed as 6.

The point of this example is that the function `mult(a, b)` starts itself inside itself. It does this until `b` reaches `0` and then calculates the result as explained above.

Recursion

Programming constructs of this kind are called *recursive* and probably the most intuitive definition of *recursion* is:

Recursion

If you still don't get it, see *recursion.*

These last two sections were recently written. If you have any comments, found any errors or think I need more/clearer explanations please email. I have been known in the past to make simple things incomprehensible. If the rest of the tutorial has made sense, but this section didn't, it is probably my fault and I would like to know. Thanks.

9.0.31 Examples

factorial.py

```
#defines a function that calculates the factorial

def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)

print "2! =", factorial(2)
print "3! =", factorial(3)
print "4! =", factorial(4)
print "5! =", factorial(5)
```

Output:

```
2! = 2
3! = 6
4! = 24
5! = 120
```

countdown.py

```
def count_down(n):
    print n
    if n > 0:
        return count_down(n-1)

count_down(5)
```

Output:

```
5
4
3
2
1
0
```

Commented function_interesting.py

```
# The comments below have been numbered as steps, to make explanation
# of the code easier. Please read according to those steps.
# (step number 1, for example, is at the bottom)


def mult(a, b): # (2.) This function will keep repeating itself,
 because....
    if b == 0:
        return 0
    rest = mult(a, b - 1) # (3.) ....Once it reaches THIS, the
 sequence starts over again and goes back to the top!
```

```
    value = a + rest
    return value # (4.) therefore, "return value" will not happen
 until the program gets past step 3 above

print "3 * 2 = ", mult(3, 2) # (1.) The "mult" function will first
 initiate here


# The "return value" event at the end can therefore only happen
# once b equals zero (b decreases by 1 everytime step 3 happens).
# And only then can the print command at the bottom be displayed.

# See it as kind of a "jump-around" effect. Basically, all you
# should really understand is that the function is reinitiated
# WITHIN ITSELF at step 3. Therefore, the sequence "jumps" back
# to the top.
```

Commented factorial.py

```
# Another "jump-around" function example:

def factorial(n): # (2.) So once again, this function will REPEAT
 itself....
    if n <= 1:
        return 1
    return n * factorial(n - 1) # (3.) Because it RE-initiates HERE,
 and goes back to the top.

print "2! =", factorial(2) # (1.) The "factorial" function is
 initiated with this line
print "3! =", factorial(3)
print "4! =", factorial(4)
print "5! =", factorial(5)
```

Commented countdown.py

```
# Another "jump-around", nice and easy:


def count_down(n): # (2.) Once again, this sequence will repeat
 itself....
    print n
    if n > 0:
        return count_down(n-1) # (3.) Because it restarts here, and
 goes back to the top

count_down(5) # (1.) The "count_down" function initiates here
```

10 Lists

10.0.32 Variables with more than one value

You have already seen ordinary variables that store a single value. However other variable types can hold more than one value. The simplest type is called a list. Here is an example of a list being used:

```
which_one = input("What month (1-12)? ")
months = [‚January‚, ‚February‚, ‚March‚, ‚April‚, ‚May‚, ‚June‚,
 ‚July‚,
          ‚August‚, ‚September‚, ‚October‚, ‚November‚, ‚December‚]

if 1 <= which_one <= 12:
    print "The month is", months[which_one - 1]
```

and an output example:

```
What month (1-12)? 3
The month is March
```

In this example the `months` is a list. `months` is defined with the lines `months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',` and `'August', 'September', 'October', 'November', 'December']` (note that a `\` could also be used to split a long line, but that is not necessary in this case because Python is intelligent enough to recognize that everything within brackets belongs together). The `[` and `]` start and end the list with commas (`,`) separating the list items. The list is used in `months[which_one - 1]`. A list consists of items that are numbered starting at 0. In other words if you wanted January you would use `months[0]`. Give a list a number and it will return the value that is stored at that location.

The statement `if 1 <= which_one <= 12:` will only be true if `which_one` is between one and twelve inclusive (in other words it is what you would expect if you have seen that in algebra).

Lists can be thought of as a series of boxes. Each box has a different value. For example, the boxes created by `demolist = ['life', 42, 'the universe', 6, 'and', 7]` would look like this:

box number	**0**	**1**	**2**	**3**	**4**	**5**
demolist	"life"	42	"the universe"	6	"and"	7

Each box is referenced by its number so the statement `demolist[0]` would get `'life'`, `demolist[1]` would get 42 and so on up to `demolist[5]` getting 7.

10.0.33 More features of lists

The next example is just to show a lot of other stuff lists can do (for once I don't expect you to type it in, but you should probably play around with lists until you are comfortable with them.). Here goes:

```
demolist = ["life", 42, "the universe", 6, "and", 7]
print "demolist = ",demolist
demolist.append("everything")
print "after ‚everything‚ was appended demolist is now:"
print demolist
print "len(demolist) =", len(demolist)
print "demolist.index(42) =", demolist.index(42)
print "demolist[1] =", demolist[1]

# Next we will loop through the list
c = 0
while c < len(demolist):
    print "demolist[", c, "] =", demolist[c]
    c = c + 1

del demolist[2]
print "After ‚the universe‚ was removed demolist is now:"
print demolist
if "life" in demolist:
    print "‚life‚ was found in demolist"
else:
    print "‚life‚ was not found in demolist"

if "amoeba" in demolist:
    print "‚amoeba‚ was found in demolist"

if "amoeba" not in demolist:
    print "‚amoeba‚ was not found in demolist"

demolist.sort()
print "The sorted demolist is", demolist
```

The output is:

```
demolist =  ['life', 42, 'the universe', 6, 'and', 7]
after 'everything' was appended demolist is now:
['life', 42, 'the universe', 6, 'and', 7, 'everything']
len(demolist) = 7
demolist.index(42) = 1
demolist[1] = 42
demolist[ 0 ] = life
demolist[ 1 ] = 42
demolist[ 2 ] = the universe
demolist[ 3 ] = 6
demolist[ 4 ] = and
demolist[ 5 ] = 7
demolist[ 6 ] = everything
After 'the universe' was removed demolist is now:
['life', 42, 6, 'and', 7, 'everything']
'life' was found in demolist
```

```
'amoeba' was not found in demolist
The sorted demolist is [6, 7, 42, 'and', 'everything', 'life']
```

This example uses a whole bunch of new functions. Notice that you can just `print` a whole list. Next the `append` function is used to add a new item to the end of the list. `len` returns how many items are in a list. The valid indexes (as in numbers that can be used inside of the `[]`) of a list range from 0 to `len - 1`. The `index` function tells where the first location of an item is located in a list. Notice how `demolist.index(42)` returns 1, and when `demolist[1]` is run it returns 42. The line `# Next we will loop through the list` is a just a reminder to the programmer (also called a *comment*). Python will ignore any lines that start with a `#`. Next the lines:

```
c = 0
while c < len(demolist):
    print ,demolist[,, c, ,] =,, demolist[c]
    c = c + 1
```

create a variable `c`, which starts at 0 and is incremented until it reaches the last index of the list. Meanwhile the `print` statement prints out each element of the list. The `del` command can be used to remove a given element in a list. The next few lines use the `in` operator to test if an element is in or is not in a list. The `sort` function sorts the list. This is useful if you need a list in order from smallest number to largest or alphabetical. Note that this rearranges the list. In summary, for a list, the following operations occur:

example	**explanation**
`demolist[2]`	accesses the element at index 2
`demolist[2] = 3`	sets the element at index 2 to be 3
`del demolist[2]`	removes the element at index 2
`len(demolist)`	returns the length of `demolist`
`"value" in demolist`	is *True* if `"value"` is an element in `demolist`
`"value" not in demolist`	is *True* if `"value"` is not an element in `demolist`
`demolist.sort()`	sorts `demolist`
`demolist.index("value")`	returns the index of the first place that `"value"` occurs
`demolist.append("value")`	adds an element `"value"` at the end of the list
`demolist.remove("value")`	removes the first occurrence of value from `demolist` (same as `del demolist[demolist.index("value")]`)

This next example uses these features in a more useful way:

```
menu_item = 0
namelist = []
while menu_item != 9:
    print "--------------------"
    print "1. Print the list"
    print "2. Add a name to the list"
    print "3. Remove a name from the list"
    print "4. Change an item in the list"
    print "9. Quit"
    menu_item = input("Pick an item from the menu: ")
    if menu_item == 1:
        current = 0
        if len(namelist) > 0:
            while current < len(namelist):
                print current, ".", namelist[current]
                current = current + 1
        else:
            print "List is empty"
    elif menu_item == 2:
        name = raw_input("Type in a name to add: ")
        namelist.append(name)
    elif menu_item == 3:
        del_name = raw_input("What name would you like to remove: ")
        if del_name in namelist:
            # namelist.remove(del_name) would work just as fine
            item_number = namelist.index(del_name)
            del namelist[item_number]
            # The code above only removes the first occurrence of
            # the name.  The code below from Gerald removes all.
            # while del_name in namelist:
            #       item_number = namelist.index(del_name)
            #       del namelist[item_number]
        else:
            print del_name, "was not found"
    elif menu_item == 4:
        old_name = raw_input("What name would you like to change: ")
        if old_name in namelist:
            item_number = namelist.index(old_name)
            new_name = raw_input("What is the new name: ")
            namelist[item_number] = new_name
        else:
            print old_name, "was not found"

print "Goodbye"
```

And here is part of the output:

```
--------------------
1. Print the list
2. Add a name to the list
3. Remove a name from the list
4. Change an item in the list
9. Quit

Pick an item from the menu: 2
Type in a name to add: Jack

Pick an item from the menu: 2
Type in a name to add: Jill

Pick an item from the menu: 1
0 . Jack
1 . Jill

Pick an item from the menu: 3
What name would you like to remove: Jack

Pick an item from the menu: 4
What name would you like to change: Jill
What is the new name: Jill Peters

Pick an item from the menu: 1
0 . Jill Peters
```

```
Pick an item from the menu: 9
Goodbye
```

That was a long program. Let's take a look at the source code. The line `namelist = []` makes the variable `namelist` a list with no items (or elements). The next important line is `while menu_item != 9:`. This line starts a loop that allows the menu system for this program. The next few lines display a menu and decide which part of the program to run.

The section

```
current = 0
if len(namelist) > 0:
    while current < len(namelist):
        print current, ".", namelist[current]
        current = current + 1
else:
    print "List is empty"
```

goes through the list and prints each name. `len(namelist)` tells how many items are in the list. If `len` returns `0`, then the list is empty.

Then, a few lines later, the statement `namelist.append(name)` appears. It uses the `append` function to add an item to the end of the list. Jump down another two lines, and notice this section of code:

```
item_number = namelist.index(del_name)
del namelist[item_number]
```

Here the `index` function is used to find the index value that will be used later to remove the item. `del namelist[item_number]` is used to remove a element of the list.

The next section

```
old_name = raw_input("What name would you like to change: ")
if old_name in namelist:
    item_number = namelist.index(old_name)
    new_name = raw_input("What is the new name: ")
    namelist[item_number] = new_name
else:
   print old_name, "was not found"
```

uses `index` to find the `item_number` and then puts `new_name` where the `old_name` was.

Congratulations, with lists under your belt, you now know enough of the language that you could do any computations that a computer can do (this is technically known as Turing-Completeness). Of course, there are still many features that are used to make your life easier.

10.0.34 Examples

test.py

```
## This program runs a test of knowledge

# First get the test questions
# Later this will be modified to use file io.
def get_questions():
    # notice how the data is stored as a list of lists
    return [["What color is the daytime sky on a clear day? ",
 "blue"],
            ["What is the answer to life, the universe and
 everything? ", "42"],
            ["What is a three letter word for mouse trap? ", "cat"]]

# This will test a single question
# it takes a single question in
# it returns True if the user typed the correct answer, otherwise
 False

def check_question(question_and_answer):
    # extract the question and the answer from the list
    question = question_and_answer[0]
    answer = question_and_answer[1]
    # give the question to the user
    given_answer = raw_input(question)
    # compare the user,s answer to the testers answer
    if answer == given_answer:
        print "Correct"
        return True
    else:
        print "Incorrect, correct was:", answer
        return False

# This will run through all the questions
def run_test(questions):
    if len(questions) == 0:
        print "No questions were given."
        # the return exits the function
        return
    index = 0
    right = 0
    while index < len(questions):
        # Check the question
        if check_question(questions[index]):
            right = right + 1
            index = index + 1
        # go to the next question
        else:
            index = index + 1
    # notice the order of the computation, first multiply, then
 divide
    print "You got", right * 100 / len(questions),\
           "% right out of", len(questions)

# now let,s run the questions

run_test(get_questions())
```

The values True and False point to 1 and 0, respectively. They are often used in sanity checks, loop conditions etc. You will learn more about this a little bit later (chapter ../Boolean Expressions/[1]).

Sample Output:

```
What color is the daytime sky on a clear day?green
Incorrect, correct was: blue
```

1 Chapter 12 on page 69

```
What is the answer to life, the universe and everything?42
Correct
What is a three letter word for mouse trap?cat
Correct
You got 66 % right out of 3
```

10.0.35 Exercises

Expand the test.py program so it has a menu giving the option of taking the test, viewing the list of questions and answers, and an option to quit. Also, add a new question to ask, "What noise does a truly advanced machine make?" with the answer of "ping".

Solution

Expand the test.py program so it has menu giving the option of taking the test, viewing the list of questions and answers, and an option to quit. Also, add a new question to ask, "What noise does a truly advanced machine make?" with the answer of "ping".

```
 This program runs a test of knowledge

questions = [["What color is the daytime sky on a clear day? ",
 "blue"],
             ["What is the answer to life, the universe and
 everything? ", "42"],
             ["What is a three letter word for mouse trap? ", "cat"],
             ["What noise does a truly advanced machine make?",
 "ping"]]

 This will test a single question
 it takes a single question in
 it returns True if the user typed the correct answer, otherwise
 False

def check_question(question_and_answer):
     extract the question and the answer from the list
    question = question_and_answer[0]
    answer = question_and_answer[1]
     give the question to the user
    given_answer = raw_input(question)
     compare the user,s answer to the testers answer
    if answer == given_answer:
        print "Correct"
        return True
    else:
        print "Incorrect, correct was:", answer
        return False
 This will run through all the questions

def run_test(questions):

    if len(questions) == 0:
        print "No questions were given."
         the return exits the function
        return
    index = 0
    right = 0
    while index < len(questions):
         Check the question
        if check_question(questions[index]):
            right = right + 1
         go to the next question
        index = index + 1
     notice the order of the computation, first multiply, then
 divide
    print ("You got", right * 100 / len(questions),
           "% right out of", len(questions))

showing a list of questions and answers
def showquestions():
    q = 0
    while q < len(questions):
        a = 0
        print "Q:" , questions[q][a]
        a = 1
        print "A:" , questions[q][a]
        q = q + 1

 now let,s define the menu function
def menu():
    print "---------"
    print "Menu:"
    print "1 - Take the test"
    print "2 - View a list of questions and answers"
    print "3 - View the menu"
    print "5 - Quit"
    print "---------"

choice = "3"
while choice != "5":
    if choice == "1":
        run_test(questions)
    elif choice == "2":
        showquestions()
    elif choice == "3":
        menu()
    print
    choice = raw_input("Choose your option from the menu above: ")
```

11 For Loops

And here is the new typing exercise for this chapter:

```
onetoten = range(1, 11)
for count in onetoten:
    print count
```

and the ever-present output:

```
1
2
3
4
5
6
7
8
9
10
```

The output looks awfully familiar but the program code looks different. The first line uses the `range` function. The `range` function uses two arguments like this `range(start, finish)`. `start` is the first number that is produced. `finish` is one larger than the last number. Note that this program could have been done in a shorter way:

```
for count in range(1, 11):
    print count
```

Here are some examples to show what happens with the `range` command:

```
>>> range(1, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(-32, -20)
[-32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22, -21]
>>> range(5,21)
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(21, 5)
[]
```

The next line `for count in onetoten:` uses the `for` control structure. A `for` control structure looks like `for variable in list:`. `list` is gone through starting with the first element of the list and going to the last. As `for` goes through each element in a list it puts each into `variable`. That allows `variable` to be used in each successive time the `for` loop is run through. Here is another example (you don't have to type this) to demonstrate:

```
demolist = ['life', 42, 'the universe', 6, 'and', 9, 'everything']
for item in demolist:
    print "The Current item is:",
    print item
```

The output is:

```
The Current item is: life
The Current item is: 42
The Current item is: the universe
The Current item is: 6
The Current item is: and
The Current item is: 9
The Current item is: everything
```

Notice how the `for` loop goes through and sets item to each element in the list. Notice how if you don't want `print` to go to the next line add a comma at the end of the statement (i.e. if you want to print something else on that line). So, what is `for` good for? The first use is to go through all the elements of a list and do something with each of them. Here's a quick way to add up all the elements:

```
list = [2, 4, 6, 8]
sum = 0
for num in list:
    sum = sum + num

print "The sum is:", sum
```

with the output simply being:

```
The sum is: 20
```

Or you could write a program to find out if there are any duplicates in a list like this program does:

```
list = [4, 5, 7, 8, 9, 1, 0, 7, 10]
list.sort()
prev = list[0]
del list[0]
for item in list:
    if prev == item:
        print "Duplicate of", prev, "found"
    prev = item
```

and for good measure:

```
Duplicate of 7 Found
```

Okay, so how does it work? Here is a special debugging version to help you understand (you don't need to type this in):

```
l = [4, 5, 7, 8, 9, 1, 0, 7, 10]
print "l = [4, 5, 7, 8, 9, 1, 0, 7, 10]", "\t\tl:", l
l.sort()
```

```
print "l.sort()", "\t\tl:", l
prev = l[0]
print "prev = l[0]", "\t\tprev:", prev
del l[0]
print "del l[0]", "\t\tl:", l
for item in l:
    if prev == item:
        print "Duplicate of", prev, "found"
    print "if prev == item:", "\t\tprev:", prev, "\titem:", item
    prev = item
    print "prev = item", "\t\tprev:", prev, "\titem:", item
```

with the output being:

```
l = [4, 5, 7, 8, 9, 1, 0, 7, 10]        l: [4, 5, 7, 8, 9, 1, 0, 7,
10]
l.sort()                l: [0, 1, 4, 5, 7, 7, 8, 9, 10]
prev = l[0]             prev: 0
del l[0]                l: [1, 4, 5, 7, 7, 8, 9, 10]
if prev == item:        prev: 0         item: 1
prev = item             prev: 1         item: 1
if prev == item:        prev: 1         item: 4
prev = item             prev: 4         item: 4
if prev == item:        prev: 4         item: 5
prev = item             prev: 5         item: 5
if prev == item:        prev: 5         item: 7
prev = item             prev: 7         item: 7
Duplicate of 7 found
if prev == item:        prev: 7         item: 7
prev = item             prev: 7         item: 7
if prev == item:        prev: 7         item: 8
prev = item             prev: 8         item: 8
if prev == item:        prev: 8         item: 9
prev = item             prev: 9         item: 9
if prev == item:        prev: 9         item: 10
prev = item             prev: 10        item: 10
```

The reason I put so many `print` statements in the code was so that you can see what is happening in each line. (By the way, if you can't figure out why a program is not working, try putting in lots of print statements so you can see what is happening.) First the program starts with a boring old list. Next the program sorts the list. This is so that any duplicates get put next to each other. The program then initializes a `prev`(ious) variable. Next the first element of the list is deleted so that the first item is not incorrectly thought to be a duplicate. Next a `for` loop is gone into. Each item of the list is checked to see if it is the same as the previous. If it is a duplicate was found. The value of `prev` is then changed so that the next time the `for` loop is run through `prev` is the previous item to the current. Sure enough, the 7 is found to be a duplicate. (Notice how `\t` is used to print a tab.)

The other way to use `for` loops is to do something a certain number of times. Here is some code to print out the first 9 numbers of the Fibonacci series:

```
a = 1
b = 1
for c in range(1, 10):
    print a,
    n = a + b
    a = b
    b = n
```

with the surprising output:

```
1 1 2 3 5 8 13 21 34
```

Everything that can be done with `for` loops can also be done with `while` loops but `for` loops give an easy way to go through all the elements in a list or to do something a certain number of times.

12 Boolean Expressions

Here is a little example of boolean expressions (you don't have to type it in):

```
a = 6
b = 7
c = 42
print 1, a == 6
print 2, a == 7
print 3, a == 6 and b == 7
print 4, a == 7 and b == 7
print 5, not a == 7 and b == 7
print 6, a == 7 or b == 7
print 7, a == 7 or b == 6
print 8, not (a == 7 and b == 6)
print 9, not a == 7 and b == 6
```

With the output being:

```
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
```

What is going on? The program consists of a bunch of funny looking `print` statements. Each `print` statement prints a number and an expression. The number is to help keep track of which statement I am dealing with. Notice how each expression ends up being either `False` or `True`. In Python, false can also be written as 0 and true as 1.

The lines:

```
print 1, a == 6
print 2, a == 7
```

print out a `True` and a `False` respectively just as expected since the first is true and the second is false. The third print, `print 3, a == 6 and b == 7`, is a little different. The operator `and` means if both the statement before and the statement after are true then the whole expression is true otherwise the whole expression is false. The next line, `print 4, a == 7 and b == 7`, shows how if part of an `and` expression is false, the whole thing is false. The behavior of `and` can be summarized as follows:

expression	**result**
true `and` true	true
true `and` false	false

expression	result
false and true	false
false and false	false

Notice that if the first expression is false Python does not check the second expression since it knows the whole expression is false.

The next line, `print 5, not a == 7 and b == 7`, uses the `not` operator. `not` just gives the opposite of the expression. (The expression could be rewritten as `print 5, a != 7 and b == 7`). Here is the table:

expression	result
not true	false
not false	true

The two following lines, `print 6, a == 7 or b == 7` and `print 7, a == 7 or b == 6`, use the `or` operator. The `or` operator returns true if the first expression is true, or if the second expression is true or both are true. If neither are true it returns false. Here's the table:

expression	result
true or true	true
true or false	true
false or true	true
false or false	false

Notice that if the first expression is true Python doesn't check the second expression since it knows the whole expression is true. This works since `or` is true if at least one half of the expression is true. The first part is true so the second part could be either false or true, but the whole expression is still true.

The next two lines, `print 8, not (a == 7 and b == 6)` and `print 9, not a == 7 and b == 6`, show that parentheses can be used to group expressions and force one part to be evaluated first. Notice that the parentheses changed the expression from false to true. This occurred since the parentheses forced the `not` to apply to the whole expression instead of just the `a == 7` portion.

Here is an example of using a boolean expression:

```
list = ["Life", "The Universe", "Everything", "Jack", "Jill", "Life",
 "Jill"]

# make a copy of the list. See the More on Lists chapter to explain
 what [:] means.
copy = list[:]
# sort the copy
copy.sort()
prev = copy[0]
del copy[0]

count = 0
```

```
# go through the list searching for a match
while count < len(copy) and copy[count] != prev:
    prev = copy[count]
    count = count + 1

# If a match was not found then count can,t be < len
# since the while loop continues while count is < len
# and no match is found

if count < len(copy):
    print "First Match:", prev
```

And here is the output:

```
First Match: Jill
```

This program works by continuing to check for match `while count < len(copy) and copy[count] is not equal to prev`. When either `count` is greater than the last index of `copy` or a match has been found the `and` is no longer true so the loop exits. The `if` simply checks to make sure that the `while` exited because a match was found.

The other "trick" of `and` is used in this example. If you look at the table for `and` notice that the third entry is "false and won't check". If `count >= len(copy)` (in other words `count < len(copy)` is false) then `copy[count]` is never looked at. This is because Python knows that if the first is false then they can't both be true. This is known as a short circuit and is useful if the second half of the `and` will cause an error if something is wrong. I used the first expression (`count < len(copy)`) to check and see if `count` was a valid index for `copy`. (If you don't believe me remove the matches "Jill" and "Life", check that it still works and then reverse the order of `count < len(copy) and copy[count] != prev` to `copy[count] != prev and count < len(copy)`.)

Boolean expressions can be used when you need to check two or more different things at once.

12.0.36 A note on Boolean Operators

A common mistake for people new to programming is a misunderstanding of the way that boolean operators works, which stems from the way the python interpreter reads these expressions. For example, after initially learning about "and " and "or" statements, one might assume that the expression `x == ('a' or 'b')` would check to see if the variable `x` was equivalent to one of the strings `'a'` or `'b'`. This is not so. To see what I'm talking about, start an interactive session with the interpreter and enter the following expressions:

```
>>> 'a' == ('a' or 'b')
>>> 'b' == ('a' or 'b')
>>> 'a' == ('a' and 'b')
>>> 'b' == ('a' and 'b')
```

And this will be the unintuitive result:

```
>>>'a' == ('a' or 'b')
True
>>>'b' == ('a' or 'b')
False
>>>'a' == ('a' and 'b')
False
>>>'b' == ('a' and 'b')
True
```

At this point, the `and` and `or` operators seem to be broken. It doesn't make sense that, for the first two expressions, `'a'` is equivalent to `'a'` or `'b'` while `'b'` is not. Furthermore, it doesn't make any sense that 'b' is equivalent to `'a'` and `'b'`. After examining what the interpreter does with boolean operators, these results do in fact exactly what you are asking of them, it's just not the same as what you think you are asking.

When the Python interpreter looks at an `or` expression, it takes the first statement and checks to see if it is true. If the first statement is true, then Python returns that object's value without checking the second statement. This is because for an `or` expression, the whole thing is true if one of the values is true; the program does not need to bother with the second statement. On the other hand, if the first value is evaluated as false Python checks the second half and returns that value. That second half determines the truth value of the whole expression since the first half was false. This "laziness" on the part of the interpreter is called "short circuiting" and is a common way of evaluating boolean expressions in many programming languages.

Similarly, for an `and` expression, Python uses a short circuit technique to speed truth value evaluation. If the first statement is false then the whole thing must be false, so it returns that value. Otherwise if the first value is true it checks the second and returns that value.

One thing to note at this point is that the boolean expression returns a value indicating `True` or `False`, but that Python considers a number of different things to have a truth value assigned to them. To check the truth value of any given object `x`, you can use the function `bool(x)` to see its truth value. Below is a table with examples of the truth values of various objects:

True	**False**
True	False
1	0
Numbers other than zero	The string 'None'
Nonempty strings	Empty strings
Nonempty lists	Empty lists
Nonempty dictionaries	Empty dictionaries

Now it is possible to understand the perplexing results we were getting when we tested those boolean expressions before. Let's take a look at what the interpreter "sees" as it goes through that code:

First case:

```
>>>'a' == ('a' or 'b')  # Look at parentheses first, so evaluate expression "('a' or
'b')"
                           # 'a' is a nonempty string, so the first
value is True
                           # Return that first value: 'a'
>>>'a' == 'a'           # the string 'a' is equivalent to the string 'a', so
expression is True
True
```

Second case:

```
>>>'b' == ('a' or 'b')  # Look at parentheses first, so evaluate expression "('a' or
'b')"
                           # 'a' is a nonempty string, so the first
value is True
                           # Return that first value: 'a'
>>>'b' == 'a'           # the string 'b' is not equivalent to the string 'a',
so expression is False
False
```

Third case:

```
>>>'a' == ('a' and 'b') # Look at parentheses first, so evaluate expression "('a' and
'b')"
                           # 'a' is a nonempty string, so the first
value is True, examine second value
                           # 'b' is a nonempty string, so second
value is True
                           # Return that second value as result of
whole expression: 'b'
>>>'a' == 'b'           # the string 'a' is not equivalent to the string 'b',
so expression is False
False
```

Fourth case:

```
>>>'b' == ('a' and 'b') # Look at parentheses first, so evaluate expression "('a' and
'b')"
                           # 'a' is a nonempty string, so the first
value is True, examine second value
                           # 'b' is a nonempty string, so second
value is True
                           # Return that second value as result of
whole expression: 'b'
>>>'b' == 'b'           # the string 'b' is equivalent to the string 'b', so
expression is True
True
```

So Python was really doing its job when it gave those apparently bogus results. As mentioned previously, the important thing is to recognize what value your boolean expression will return when it is evaluated, because it isn't always obvious.

Going back to those initial expressions, this is how you would write them out so they behaved in a way that you want:

```
>>>'a' == 'a' or 'a' == 'b'
True
>>>'b' == 'a' or 'b' == 'b'
True
>>>'a' == 'a' and 'a' == 'b'
False
>>> 'b' == 'a' and 'b' == 'b'
False
```

When these comparisons are evaluated they return truth values in terms of True or False, not strings, so we get the proper results.

12.0.37 Examples

password1.py

```
## This programs asks a user for a name and a password.
# It then checks them to make sure the user is allowed in.

name = raw_input("What is your name? ")
password = raw_input("What is the password? ")
if name == "Josh" and password == "Friday":
    print "Welcome Josh"
elif name == "Fred" and password == "Rock":
    print "Welcome Fred"
else:
    print "I don,t know you."
```

Sample runs

```
What is your name? Josh
What is the password? Friday
Welcome Josh
```

```
What is your name? Bill
What is the password? Money
I don't know you.
```

12.0.38 Exercises

Write a program that has a user guess your name, but they only get 3 chances to do so until the program quits.

Solution

Write a program that has a user guess your name, but they only get 3 chances to do so until the program quits.

```
print "Try to guess my name!"
count = 3
```

```
name = "Tony"
guess = raw_input("What is my name? ")
while count > 1 and guess != name:
    print "You are wrong!"
    guess = raw_input("What is my name? ")
    count = count - 1

if guess != name:
    print "You are wrong!"  this message isn,t printed in the third
 chance, so we print it now
    print "You ran out of chances."
    quit
else:
    print "Yes! My name is", name + "!"
```

13 Dictionaries

This chapter is about dictionaries. If you open a dictionary, you should notice every entry consists of two parts, a word and the word's definition. The word is the key to finding out what a word means, and what the word means is considered the value for that key. In Python, dictionaries have keys and values. Keys are used to find values. Here is an example of a dictionary in use:

```
def print_menu():
    print ‚1. Print Dictionary‚
    print ‚2. Add definition‚
    print ‚3. Remove word‚
    print ‚4. Lookup word‚
    print ‚5. Quit‚
    print

words = {}
menu_choice = 0
print_menu()

while menu_choice != 5:
    menu_choice = input("Type in a number (1-5): ")
    if menu_choice == 1:
        print "Definitions:"
        for x in words.keys():
            print x, ": ", words[x]
        print
    elif menu_choice == 2:
        print "Add definition"
        name = raw_input("Word: ")
        means = raw_input("Definition: ")
        words[name] = means
    elif menu_choice == 3:
        print "Remove word"
        name = raw_input("Word: ")
        if name in words:
            del words[name]
            print name, " was removed."
        else:
            print name, " was not found."
    elif menu_choice == 4:
        print "Lookup Word"
        name = raw_input("Word: ")
        if name in words:
            print "The definition of ", name, " is: ", words[name]
        else:
            print name, "No definition for ", name, " was found."
    elif menu_choice != 5:
        print_menu()
```

And here is my output:

```
1. Print Dictionary
2. Add definition
```

```
3. Remove word
4. Lookup word
5. Quit

Type in a number (1-5): 2
Add definition
Word: Python
Definition: A snake, a programming language, and a British comedy.
Type in a number (1-5): 2
Add definition
Word: Dictionary
Definition: A book where words are defined.
Type in a number (1-5): 1
Definitions:
Python: A snake, a programming language, and a British comedy.
Dictionary: A book where words are defined.

Type in a number (1-5): 4
Lookup Word
Word: Python
The definition of Python is: A snake, a programming language, and a
British comedy.
Type in a number (1-5): 3
Remove Word
Word: Dictionary
Dictionary was removed.
Type in a number (1-5): 1
Definitions:
Python: A snake, a programming language, and a British comedy.
Type in a number (1-5): 5
```

This program is similar to the name list from the earlier chapter on lists (note that lists use indexes and dictionaries don't). Here's how the program works:

- First the function `print_menu` is defined. `print_menu` just prints a menu that is later used twice in the program.
- Next comes the funny looking line `words = {}`. All that line does is tell Python that `words` is a dictionary.
- The next few lines just make the menu work.

```
for x in words.keys():
    print x, ": ", words[x]
```

- This goes through the dictionary and prints all the information. The function `words.keys()` returns a list that is then used by the `for` loop. The list returned by `keys()` is not in any particular order so if you want it in alphabetic order it must be sorted. Similar to lists the statement `words[x]` is used to access a specific member of the dictionary. Of course in this case `x` is a string.
- Next the line `words[name] = means` adds a word and definition to the dictionary. If `name` is already in the dictionary `means` replaces whatever was there before.

```
if name in words:
    del words[name]
```

- See if name is in words and remove it if it is. The expression `name in words` returns true if `name` is a key in `words` but otherwise returns false. The line `del words[name]` removes the key `name` and the value associated with that key.

```
if name in words:
    print "The definition of ", name, " is: ", words[name]
```

- Check to see if words has a certain key and if it does prints out the definition associated with it.
- Lastly if the menu choice is invalid it reprints the menu for your viewing pleasure.

A recap: Dictionaries have keys and values. Keys can be strings or numbers. Keys point to values. Values can be any type of variable (including lists or even dictionaries (those dictionaries or lists of course can contain dictionaries or lists themselves (scary right? :-))). Here is an example of using a list in a dictionary:

```
max_points = [25, 25, 50, 25, 100]
assignments = [,hw ch 1,, ,hw ch 2,, ,quiz   ,, ,hw ch 3,, ,test,]
students = {,#Max,: max_points}

def print_menu():
    print "1. Add student"
    print "2. Remove student"
    print "3. Print grades"
    print "4. Record grade"
    print "5. Print Menu"
    print "6. Exit"

def print_all_grades():
    print ,\t,,
    for i in range(len(assignments)):
        print assignments[i], ,\t,,
    print
    keys = students.keys()
    keys.sort()
    for x in keys:
        print x, ,\t,,
        grades = students[x]
        print_grades(grades)

def print_grades(grades):
    for i in range(len(grades)):
        print grades[i], ,\t,, ,\t,,
    print

print_menu()
menu_choice = 0
while menu_choice != 6:
    print
    menu_choice = input("Menu Choice (1-6): ")
    if menu_choice == 1:
        name = raw_input("Student to add: ")
        students[name] = [0] * len(max_points)
    elif menu_choice == 2:
        name = raw_input("Student to remove: ")
        if name in students:
            del students[name]
        else:
            print "Student:", name, "not found"
    elif menu_choice == 3:
        print_all_grades()
    elif menu_choice == 4:
        print "Record Grade"
        name = raw_input("Student: ")
        if name in students:
            grades = students[name]
            print "Type in the number of the grade to record"
            print "Type a 0 (zero) to exit"
            for i in range(len(assignments)):
                print i + 1, assignments[i], ,\t,,
            print
            print_grades(grades)
            which = 1234
            while which != -1:
                which = input("Change which Grade: ")
                which = which - 1
                if 0 <= which < len(grades):
                    grade = input("Grade: ")
                    grades[which] = grade
                elif which != -1:
                    print "Invalid Grade Number"
        else:
            print "Student not found"
    elif menu_choice != 6:
        print_menu()
```

and here is a sample output:

```
1. Add student
2. Remove student
3. Print grades
4. Record grade
5. Print Menu
6. Exit

Menu Choice (1-6): 3
       hw ch 1         hw ch 2         quiz            hw ch 3
  test
#Max    25              25              50              25
   100

Menu Choice (1-6): 5
1. Add student
2. Remove student
3. Print grades
4. Record grade
5. Print Menu
6. Exit

Menu Choice (1-6): 1
Student to add: Bill

Menu Choice (1-6): 4
Record Grade
Student: Bill
Type in the number of the grade to record
Type a 0 (zero) to exit
1   hw ch 1     2   hw ch 2     3   quiz        4   hw ch 3     5
test
0               0               0               0               0
Change which Grade: 1
Grade: 25
Change which Grade: 2
Grade: 24
Change which Grade: 3
Grade: 45
Change which Grade: 4
Grade: 23
Change which Grade: 5
Grade: 95
Change which Grade: 0
```

```
Menu Choice (1-6): 3
       hw ch 1         hw ch 2         quiz            hw ch 3
  test
#Max    25              25              50              25
   100
Bill    25              24              45              23
   95

Menu Choice (1-6): 6
```

Heres how the program works. Basically the variable `students` is a dictionary with the keys being the name of the students and the values being their grades. The first two lines just create two lists. The next line `students = {'#Max':  max_points}` creates a new dictionary with the key {`#Max`} and the value is set to be `[25, 25, 50, 25, 100]`, since thats what `max_points` was when the assignment is made (I use the key `#Max` since `#` is sorted ahead of any alphabetic characters). Next `print_menu` is defined. Next the `print_all_grades` function is defined in the lines:

```
def print_all_grades():
    print '\t',
    for i in range(len(assignments)):
        print assignments[i], '\t',
    print
    keys = students.keys()
    keys.sort()
    for x in keys:
        print x, '\t',
        grades = students[x]
        print_grades(grades)
```

Notice how first the keys are gotten out of the `students` dictionary with the `keys` function in the line `keys = students.keys()`. `keys` is a list so all the functions for lists can be used on it. Next the keys are sorted in the line `keys.sort()` since it is a list. `for` is used to go through all the keys. The grades are stored as a list inside the dictionary so the assignment `grades = students[x]` gives `grades` the list that is stored at the key `x`. The function `print_grades` just prints a list and is defined a few lines later.

The later lines of the program implement the various options of the menu. The line `students[name] = [0] * len(max_points)` adds a student to the key of their name. The notation `[0] * len(max_points)` just creates a list of 0's that is the same length as the `max_points` list.

The remove student entry just deletes a student similar to the telephone book example. The record grades choice is a little more complex. The grades are retrieved in the line `grades = students[name]` gets a reference to the grades of the student `name`. A grade is then recorded in the line `grades[which] = grade`. You may notice that `grades` is never put back into the students dictionary (as in no `students[name] = grades`). The reason for the missing statement is that `grades` is actually another name for `students[name]` and so changing `grades` changes `student[name]`.

Dictionaries provide a easy way to link keys to values. This can be used to easily keep track of data that is attached to various keys.

14 Using Modules

Here's this chapter's typing exercise (name it cal.py (`import` actually looks for a file named calendar.py and reads it in. If the file is named calendar.py and it sees a "import calendar" it tries to read in itself which works poorly at best.)):

```
import calendar
year = input("Type in the year number: ")
calendar.prcal(year)
```

And here is part of the output I got:

```
Type in the year number: 2001

                                 2001


       January                   February                    March


Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr
Sa Su
1  2  3  4  5  6  7                1  2  3  4                1  2  3
 4
8  9 10 11 12 13 14       5  6  7  8  9 10 11       5  6  7  8  9 10
11
15 16 17 18 19 20 21      12 13 14 15 16 17 18      12 13 14 15 16
17 18
22 23 24 25 26 27 28      19 20 21 22 23 24 25      19 20 21 22 23
24 25
29 30 31                  26 27 28                  26 27 28 29 30
31
```

(I skipped some of the output, but I think you get the idea.) So what does the program do? The first line `import calendar` uses a new command `import`. The command `import` loads a module (in this case the `calendar` module). To see the commands available in the standard modules either look in the library reference for python (if you downloaded it) or go to `http://docs.python.org/library/`. If you look at the documentation for the calendar module, it lists a function called `prcal` that prints a calendar for a year. The line `calendar.prcal(year)` uses this function. In summary to use a module `import` it and then use `module_name.function` for functions in the module. Another way to write the program is:

```
from calendar import prcal

year = input("Type in the year number: ")
prcal(year)
```

This version imports a specific function from a module. Here is another program that uses the Python Library (name it something like clock.py) (press Ctrl and the 'c' key at the same time to terminate the program):

```
from time import time, ctime

prev_time = ""
while True:
    the_time = ctime(time())
    if prev_time != the_time:
        print "The time is:", ctime(time())
        prev_time = the_time
```

With some output being:

```
The time is: Sun Aug 20 13:40:04 2000
The time is: Sun Aug 20 13:40:05 2000
The time is: Sun Aug 20 13:40:06 2000
The time is: Sun Aug 20 13:40:07 2000

Traceback (innermost last):
 File "clock.py", line 5, in ?
    the_time = ctime(time())

KeyboardInterrupt
```

The output is infinite of course so I canceled it (or the output at least continues until Ctrl+C is pressed). The program just does a infinite loop (`True` is always true, so `while True:` goes forever) and each time checks to see if the time has changed and prints it if it has. Notice how multiple names after the import statement are used in the line `from time import time, ctime`.

The Python Library contains many useful functions. These functions give your programs more abilities and many of them can simplify programming in Python.

14.0.39 Exercises

Rewrite the High_low.py program from section Decisions[1] to use a random integer between 0 and 99 instead of the hard-coded 78. Use the Python documentation to find an appropriate module and function to do this.

Solution

Rewrite the High_low.py program from section Decisions[2] to use an random integer between 0 and 99 instead of the hard-coded 78. Use the Python documentation to find an appropriate module and function to do this.

```
from random import randint
number = randint(0, 99)
```

1 Chapter 6.0.20 on page 30

```
guess = -1
while guess != number:
    guess = input ("Guess a number: ")
    if guess > number:
        print "Too high"
    elif guess < number:
            print "Too low"
print "Just right"
```

15 More on Lists

We have already seen lists and how they can be used. Now that you have some more background I will go into more detail about lists. First we will look at more ways to get at the elements in a list and then we will talk about copying them.

Here are some examples of using indexing to access a single element of a list:

```
>>> some_numbers = ['zero', 'one', 'two', 'three', 'four', 'five']
>>> some_numbers[0]
'zero'
>>> some_numbers[4]
'four'
>>> some_numbers[5]
'five'
```

All those examples should look familiar to you. If you want the first item in the list just look at index 0. The second item is index 1 and so on through the list. However what if you want the last item in the list? One way could be to use the `len()` function like `some_numbers[len(some_numbers) - 1]`. This way works since the `len()` function always returns the last index plus one. The second from the last would then be `some_numbers[len(some_numbers) - 2]`. There is an easier way to do this. In Python the last item is always index -1. The second to the last is index -2 and so on. Here are some more examples:

```
>>> some_numbers[len(some_numbers) - 1]
'five'
>>> some_numbers[len(some_numbers) - 2]
'four'
>>> some_numbers[-1]
'five'
>>> some_numbers[-2]
'four'
>>> some_numbers[-6]
'zero'
```

Thus any item in the list can be indexed in two ways: from the front and from the back.

Another useful way to get into parts of lists is using slicing. Here is another example to give you an idea what they can be used for:

```
>>> things = [0, 'Fred', 2, 'S.P.A.M.', 'Stocking', 42, "Jack", "Jill"]
>>> things[0]
0
>>> things[7]
'Jill'
>>> things[0:8]
```

```
[0, 'Fred', 2, 'S.P.A.M.', 'Stocking', 42, 'Jack', 'Jill']
>>> things[2:4]
[2, 'S.P.A.M.']
>>> things[4:7]
['Stocking', 42, 'Jack']
>>> things[1:5]
['Fred', 2, 'S.P.A.M.', 'Stocking']
```

Slicing is used to return part of a list. The slicing operator is in the form `things[first_index:last_index]`. Slicing cuts the list before the `first_index` and before the `last_index` and returns the parts inbetween. You can use both types of indexing:

```
>>> things[-4:-2]
['Stocking', 42]
>>> things[-4]
'Stocking'
>>> things[-4:6]
['Stocking', 42]
```

Another trick with slicing is the unspecified index. If the first index is not specified the beginning of the list is assumed. If the last index is not specified the whole rest of the list is assumed. Here are some examples:

```
>>> things[:2]
[0, 'Fred']
>>> things[-2:]
['Jack', 'Jill']
>>> things[:3]
[0, 'Fred', 2]
>>> things[:-5]
[0, 'Fred', 2]
```

Here is a (HTML inspired) program example (copy and paste in the poem definition if you want):

```
poem = ["<B>", "Jack", "and", "Jill", "</B>", "went", "up", "the",
        "hill", "to", "<B>", "fetch", "a", "pail", "of", "</B>",
        "water.", "Jack", "fell", "<B>", "down", "and", "broke",
        "</B>", "his", "crown", "and", "<B>", "Jill", "came",
        "</B>", "tumbling", "after"]

def get_bolds(text):
    true = 1
    false = 0
    ## is_bold tells whether or not the we are currently looking at
    ## a bold section of text.
    is_bold = false
    ## start_block is the index of the start of either an unbolded
    ## segment of text or a bolded segment.
    start_block = 0
    for index in range(len(text)):
        ## Handle a starting of bold text
        if text[index] == "<B>":
            if is_bold:
                print "Error: Extra Bold"
            ## print "Not Bold:", text[start_block:index]
            is_bold = true
            start_block = index + 1
        ## Handle end of bold text
        ## Remember that the last number in a slice is the index
        ## after the last index used.
        if text[index] == "</B>":
            if not is_bold:
                print "Error: Extra Close Bold"
            print "Bold [", start_block, ":", index, "]",
 text[start_block:index]
            is_bold = false
            start_block = index + 1

get_bolds(poem)
```

with the output being:

```
Bold [ 1 : 4 ] ['Jack', 'and', 'Jill']
Bold [ 11 : 15 ] ['fetch', 'a', 'pail', 'of']
Bold [ 20 : 23 ] ['down', 'and', 'broke']
Bold [ 28 : 30 ] ['Jill', 'came']
```

The `get_bold()` function takes in a list that is broken into words and tokens. The tokens that it looks for are `<B>` which starts the bold text and `</B>` which ends bold text. The function `get_bold()` goes through and searches for the start and end tokens.

The next feature of lists is copying them. If you try something simple like:

```
>>> a = [1, 2, 3]
>>> b = a
>>> print b
[1, 2, 3]
>>> b[1] = 10
>>> print b
[1, 10, 3]
>>> print a
[1, 10, 3]
```

This probably looks surprising since a modification to `b` resulted in `a` being changed as well. What happened is that the statement `b = a` makes `b` a *reference* to `a`. This means that `b` can be thought of as another name for `a`. Hence any modification to `b` changes `a` as well. However some assignments don't create two names for one list:

```
>>> a = [1, 2, 3]
>>> b = a * 2
>>> print a
[1, 2, 3]
>>> print b
[1, 2, 3, 1, 2, 3]
>>> a[1] = 10
>>> print a
[1, 10, 3]
>>> print b
[1, 2, 3, 1, 2, 3]
```

In this case `b` is not a reference to `a` since the expression `a * 2` creates a new list. Then the statement `b = a * 2` gives `b` a reference to `a * 2` rather than a reference to `a`. All assignment operations create a reference. When you pass a list as an argument to a function you create a reference as well. Most of the time you don't have to worry about creating references rather than copies. However when you need to make modifications to one list without changing another name of the list you have to make sure that you have actually created a copy.

There are several ways to make a copy of a list. The simplest that works most of the time is the slice operator since it always makes a new list even if it is a slice of a whole list:

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> b[1] = 10
>>> print a
[1, 2, 3]
>>> print b
[1, 10, 3]
```

Taking the slice `[:]` creates a new copy of the list. However it only copies the outer list. Any sublist inside is still a references to the sublist in the original list. Therefore, when the list contains lists, the inner lists have to be copied as well. You could do that manually but Python already contains a module to do it. You use the `deepcopy` function of the `copy` module:

```
>>> import copy
>>> a = [[1, 2, 3], [4, 5, 6]]
>>> b = a[:]
>>> c = copy.deepcopy(a)
>>> b[0][1] = 10
>>> c[1][1] = 12
>>> print a
[[1, 10, 3], [4, 5, 6]]
>>> print b
[[1, 10, 3], [4, 5, 6]]
>>> print c
[[1, 2, 3], [4, 12, 6]]
```

First of all notice that `a` is a list of lists. Then notice that when `b[0][1] = 10` is run both `a` and `b` are changed, but `c` is not. This happens because the inner arrays are still references when the slice operator is used. However with `deepcopy` `c` was fully copied.

So, should I worry about references every time I use a function or `=`? The good news is that you only have to worry about references when using dictionaries and lists. Numbers and strings create references when assigned but every operation on numbers and strings that modifies them creates a new copy so you can never modify them unexpectedly. You do have to think about references when you are modifying a list or a dictionary.

By now you are probably wondering why are references used at all? The basic reason is speed. It is much faster to make a reference to a thousand element list than to copy all the elements. The other reason is that it allows you to have a function to modify the inputed list or dictionary. Just remember about references if you ever have some weird problem with data being changed when it shouldn't be.

16 Revenge of the Strings

And now presenting a cool trick that can be done with strings:

```
def shout(string):
    for character in string:
        print "Gimme a " + character
        print "'" + character + "'"
shout("Lose")

def middle(string):
    print "The middle character is:", string[len(string) / 2]

middle("abcdefg")
middle("The Python Programming Language")
middle("Atlanta")
```

And the output is:

```
Gimme a L
'L'
Gimme a o
'o'
Gimme a s
's'
Gimme a e
'e'
The middle character is: d
The middle character is: r
The middle character is: a
```

What these programs demonstrate is that strings are similar to lists in several ways. The `shout()` function shows that `for` loops can be used with strings just as they can be used with lists. The `middle` procedure shows that that strings can also use the `len()` function and array indexes and slices. Most list features work on strings as well.

The next feature demonstrates some string specific features:

```
def to_upper(string):
    ## Converts a string to upper case
    upper_case = ""
    for character in string:
        if 'a' <= character <= 'z':
            location = ord(character) - ord('a')
            new_ascii = location + ord('A')
            character = chr(new_ascii)
        upper_case = upper_case + character
    return upper_case

print to_upper("This is Text")
```

with the output being:

```
THIS IS TEXT
```

This works because the computer represents the characters of a string as numbers from 0 to 255. Python has a function called `ord()` (short for ordinal) that returns a character as a number. There is also a corresponding function called `chr()` that converts a number into a character. With this in mind the program should start to be clear. The first detail is the line: `if 'a' <= character <= 'z':` which checks to see if a letter is lower case. If it is then the next lines are used. First it is converted into a location so that a = 0, b = 1, c = 2 and so on with the line: `location = ord(character) - ord('a')`. Next the new value is found with `new_ascii = location + ord('A')`. This value is converted back to a character that is now upper case.

Now for some interactive typing exercise:

```
>>> # Integer to String
>>> 2
2
>>> repr(2)
'2'
>>> -123
-123
>>> repr(-123)
'-123'
>>> `123`
'123'
>>> # String to Integer
>>> "23"
'23'
>>> int("23")
23
>>> "23" * 2
'2323'
>>> int("23") * 2
46
>>> # Float to String
>>> 1.23
1.23
>>> repr(1.23)
'1.23'
>>> # Float to Integer
>>> 1.23
1.23
>>> int(1.23)
1
>>> int(-1.23)
-1
>>> # String to Float
>>> float("1.23")
1.23
>>> "1.23"
'1.23'
>>> float("123")
123.0
>>> `float("1.23")`
'1.23'
```

If you haven't guessed already the function `repr()` can convert a integer to a string and the function `int()` can convert a string to an integer. The function `float()` can convert a string to a float. The `repr()` function returns a printable representation of something.

`...` converts almost everything into a string, too. Here are some examples of this:

```
>>> repr(1)
'1'
>>> repr(234.14)
'234.14'
>>> repr([4, 42, 10])
'[4, 42, 10]'
>>> `[4, 42, 10]`
'[4, 42, 10]'
```

The `int()` function tries to convert a string (or a float) into a integer. There is also a similar function called `float()` that will convert a integer or a string into a float. Another function that Python has is the `eval()` function. The `eval()` function takes a string and returns data of the type that python thinks it found. For example:

```
>>> v = eval('123')
>>> print v, type(v)
123 <type 'int'>
>>> v = eval('645.123')
>>> print v, type(v)
645.123 <type 'float'>
>>> v = eval('[1, 2, 3]')
>>> print v, type(v)
[1, 2, 3] <type 'list'>
```

If you use the `eval()` function you should check that it returns the type that you expect.

One useful string function is the `split()` method. Here's an example:

```
>>> "This is a bunch of words".split()
['This', 'is', 'a', 'bunch', 'of', 'words']
>>> text = "First batch, second batch, third, fourth"
>>> text.split(",")
['First batch', ' second batch', ' third', ' fourth']
```

Notice how `split()` converts a string into a list of strings. The string is split by whitespace by default or by the optional argument (in this case a comma). You can also add another argument that tells `split()` how many times the separator will be used to split the text. For example:

```
>>> list = text.split(",")
>>> len(list)
4
>>> list[-1]
' fourth'
>>> list = text.split(",", 2)
>>> len(list)
3
>>> list[-1]
' third, fourth'
```

16.0.40 Slicing strings (and lists)

Strings can be cut into pieces — in the same way as it was shown for lists in the previous chapter — by using the *slicing* "operator" `[:]`. The slicing operator works in the same way as before: text[first_index:last_index] (in very rare cases there can be another colon and a third argument, as in the example shown below).

In order not to get confused by the index numbers, it is easiest to see them as *clipping places*, possibilities to cut a string into parts. Here is an example, which shows the clipping places (in yellow) and their index numbers (red and blue) for a simple text string:

```
              0       1       2       ...      -2       -1
              ↓       ↓       ↓        ↓        ↓        ↓        ↓
text =   "        S       T       R        I        N        G        "
              ↑                                                   ↑
              [:                                                  :]
```

Note that the red indexes are counted from the beginning of the string and the blue ones from the end of the string backwards. (Note that there is no blue -0, which could seem to be logical at the end of the string. Because `-0 == 0`, (-0 means "beginning of the string" as well.) Now we are ready to use the indexes for slicing operations:

`text[1:4]`	→	`"TRI"`
`text[:5]`	→	`"STRIN"`
`text[:-1]`	→	`"STRIN"`
`text[-4:]`	→	`"RING"`
`text[2]`	→	`"R"`
`text[:]`	→	`"STRING"`
`text[::-1]`	→	`"GNIRTS"`

`text[1:4]` gives us all of the `text` string between clipping places 1 and 4, `"TRI"`. If you omit one of the [first_index:last_index] arguments, you get the beginning or end of the string as default: `text[:5]` gives `"STRIN"`. For both `first_index` and `last_index` we can use both the red and the blue numbering schema: `text[:-1]` gives the same as `text[:5]`, because the index -1 is at the same place as 5 in this case. If we do not use an argument containing a colon, the number is treated in a different way: `text[2]` gives us one character following the second clipping point, `"R"`. The special slicing operation `text[:]` means "from the beginning to the end" and produces a copy of the entire string (or list, as shown in the previous chapter).

Last but not least, the slicing operation can have a second colon and a third argument, which is interpreted as the "step size": `text[::-1]` is `text` from beginning to the end, with a step size of -1. -1 means "every character, but in the other direction". `"STRING"` backwards is `"GNIRTS"` (test a step length of 2, if you have not got the point here).

All these slicing operations work with lists as well. In that sense strings are just a special case of lists, where the list elements are single characters. Just remember the concept of *clipping places*, and the indexes for slicing things will get a lot less confusing.

16.0.41 Examples

```
# This program requires an excellent understanding of decimal numbers
def to_string(in_int):
    """Converts an integer to a string"""
    out_str = ""
    prefix = ""
    if in_int < 0:
        prefix = "-"
        in_int = -in_int
    while in_int / 10 != 0:
        out_str = chr(ord(,0,) + in_int % 10) + out_str
        in_int = in_int / 10
    out_str = chr(ord(,0,) + in_int % 10) + out_str
    return prefix + out_str

def to_int(in_str):
    """Converts a string to an integer"""
    out_num = 0
    if in_str[0] == "-":
        multiplier = -1
        in_str = in_str[1:]
    else:
        multiplier = 1
    for x in range(0, len(in_str)):
        out_num = out_num * 10 + ord(in_str[x]) - ord('0')
    return out_num * multiplier

print to_string(2)
print to_string(23445)
print to_string(-23445)
print to_int("14234")
print to_int("12345")
print to_int("-3512")
```

The output is:

```
2
23445
-23445
14234
12345
-3512
```

17 File IO

Here is a simple example of file IO (input/output):

```
# Write a file
out_file = open("test.txt", "w")
out_file.write("This Text is going to out file\nLook at it and see!")
out_file.close()

# Read a file
in_file = open("test.txt", "r")
text = in_file.read()
in_file.close()

print text
```

The output and the contents of the file `test.txt` are:

```
This Text is going to out file
Look at it and see!
```

Notice that it wrote a file called `test.txt` in the directory that you ran the program from. The `\n` in the string tells Python to put a *n*ewline where it is.

A overview of file IO is:

- Get a file object with the `open` function.
- Read or write to the file object (depending on how it was opened)
- Close it

The first step is to get a file object. The way to do this is to use the `open` function. The format is `file_object = open(filename, mode)` where `file_object` is the variable to put the file object, `filename` is a string with the filename, and `mode` is `"r"` to *r*ead a file or `"w"` to *w*rite a file (and a few others we will skip here). Next the file objects functions can be called. The two most common functions are `read` and `write`. The `write` function adds a string to the end of the file. The `read` function reads the next thing in the file and returns it as a string. If no argument is given it will return the whole file (as done in the example).

Now here is a new version of the phone numbers program that we made earlier:

```
def print_numbers(numbers):
    print "Telephone Numbers:"
    for x in numbers.keys():
        print "Name:", x, "\tNumber:", numbers[x]
    print

def add_number(numbers, name, number):
    numbers[name] = number

def lookup_number(numbers, name):
    if name in numbers:
        return "The number is " + numbers[name]
    else:
        return name + " was not found"

def remove_number(numbers, name):
    if name in numbers:
        del numbers[name]
    else:
        print name," was not found"

def load_numbers(numbers, filename):
    in_file = open(filename, "r")
    while True:
        in_line = in_file.readline()
        if not in_line:
            break
        in_line = in_line[:-1]
        name, number = in_line.split(",")
        numbers[name] = number
    in_file.close()

def save_numbers(numbers, filename):
    out_file = open(filename, "w")
    for x in numbers.keys():
        out_file.write(x + "," + numbers[x] + "\n")
    out_file.close()

def print_menu():
    print ,1. Print Phone Numbers,
    print ,2. Add a Phone Number,
    print ,3. Remove a Phone Number,
    print ,4. Lookup a Phone Number,
    print ,5. Load numbers,
    print ,6. Save numbers,
    print ,7. Quit,
    print

phone_list = {}
menu_choice = 0
print_menu()
while True:
    menu_choice = input("Type in a number (1-7): ")
    if menu_choice == 1:
        print_numbers(phone_list)
    elif menu_choice == 2:
        print "Add Name and Number"
        name = raw_input("Name: ")
        phone = raw_input("Number: ")
        add_number(phone_list, name, phone)
    elif menu_choice == 3:
        print "Remove Name and Number"
        name = raw_input("Name: ")
        remove_number(phone_list, name)
    elif menu_choice == 4:
        print "Lookup Number"
        name = raw_input("Name: ")
        print lookup_number(phone_list, name)
    elif menu_choice == 5:
        filename = raw_input("Filename to load: ")
        load_numbers(phone_list, filename)
    elif menu_choice == 6:
        filename = raw_input("Filename to save: ")
        save_numbers(phone_list, filename)
    elif menu_choice == 7:
        break
    else:
        print_menu()

print "Goodbye"
```

Notice that it now includes saving and loading files. Here is some output of my running it twice:

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7): 2
Add Name and Number
Name: Jill
Number: 1234
Type in a number (1-7): 2
Add Name and Number
Name: Fred
Number: 4321
Type in a number (1-7): 1
Telephone Numbers:
Name: Jill     Number: 1234
Name: Fred     Number: 4321

Type in a number (1-7): 6
Filename to save: numbers.txt
Type in a number (1-7): 7
Goodbye
```

```
1. Print Phone Numbers
2. Add a Phone Number
3. Remove a Phone Number
4. Lookup a Phone Number
5. Load numbers
6. Save numbers
7. Quit

Type in a number (1-7): 5
Filename to load: numbers.txt
Type in a number (1-7): 1
Telephone Numbers:
Name: Jill     Number: 1234
Name: Fred     Number: 4321

Type in a number (1-7): 7
Goodbye
```

The new portions of this program are:

```
def load_numbers(numbers, filename):
    in_file = open(filename, "r")
    while True:
        in_line = in_file.readline()
        if not in_line:
            break
        in_line = in_line[:-1]
        name, number = in_line.split(",")
        numbers[name] = number
    in_file.close()
```

```
def save_numbers(numbers, filename):
    out_file = open(filename, "w")
    for x in numbers.keys():
        out_file.write(x + "," + numbers[x] + "\n")
    out_file.close()
```

First we will look at the save portion of the program. First it creates a file object with the command `open(filename, "w")`. Next it goes through and creates a line for each of the phone numbers with the command `out_file.write(x + "," + numbers[x] + "\n")`. This writes out a line that contains the name, a comma, the number and follows it by a newline.

The loading portion is a little more complicated. It starts by getting a file object. Then it uses a `while True:` loop to keep looping until a `break` statement is encountered. Next it gets a line with the line `in_line = in_file.readline()`. The `readline` function will return a empty string when the end of the file is reached. The `if` statement checks for this and `break`s out of the `while` loop when that happens. Of course if the `readline` function did not return the newline at the end of the line there would be no way to tell if an empty string was an empty line or the end of the file so the newline is left in what `readline` returns. Hence we have to get rid of the newline. The line `in_line = in_line[:-1]` does this for us by dropping the last character. Next the line `name, number = in_line.split(",")` splits the line at the comma into a name and a number. This is then added to the `numbers` dictionary.

17.0.42 Exercises

Now modify the grades program from section ../Dictionaries/[1] so that is uses file IO to keep a record of the students.

Solution

Now modify the grades program from section ../Dictionaries/[2] so that is uses file IO to keep a record of the students.

```
assignments = [,hw ch 1,, ,hw ch 2,, ,quiz   ,, ,hw ch 3,, ,test,]
students = { }

def load_grades(gradesfile):
    inputfile = open(gradesfile, "r")
    grades = [ ]
    while True:
        student_and_grade = inputfile.readline()
        student_and_grade = student_and_grade[:-1]
        if not student_and_grade:
            break
        else:
            studentname, studentgrades = student_and_grade.split(",")
```

1 Chapter 13 on page 77

```
            studentgrades = studentgrades.split(" ")
            students[studentname] = studentgrades
    inputfile.close()
    print "Grades loaded."

def save_grades(gradesfile):
    outputfile = open(gradesfile, "w")
    for i in students.keys():
        outputfile.write(i + ",")
        for x in students[i]:
            outputfile.write(x + " ")
        outputfile.write("\n")
    outputfile.close()
    print "Grades saved."

def print_menu():
    print "1. Add student"
    print "2. Remove student"
    print "3. Load grades"
    print "4. Record grade"
    print "5. Print grades"
    print "6. Save grades"
    print "7. Print Menu"
    print "9. Quit"

def print_all_grades():
    keys = students.keys()
    if keys:
        keys.sort()
        print '\t',
        for i in range(len(assignments)):
            print assignments[i], '\t',
        print
        for x in keys:
            print x, '\t',
            grades = students[x]
            print_grades(grades)
    else:
        print "There are no grades to print."

def print_grades(grades):
    for i in range(len(grades)):
        print grades[i], '\t',
    print

print_menu()
```

```
menu_choice = 0
while menu_choice != 9:
    print
    menu_choice = input("Menu Choice: ")
    if menu_choice == 1:
        name = raw_input("Student to add: ")
        students[name] = [0] * len(assignments)
    elif menu_choice == 2:
        name = raw_input("Student to remove: ")
        if name in students:
            del students[name]
        else:
            print "Student:", name, "not found"
    elif menu_choice == 3:
        gradesfile = raw_input("Load grades from which file? ")
        load_grades(gradesfile)
    elif menu_choice == 4:
        print "Record Grade"
        name = raw_input("Student: ")
        if name in students:
            grades = students[name]
            print "Type in the number of the grade to record"
            print "Type a 0 (zero) to exit"
            for i in range(len(assignments)):
                print i + 1, assignments[i], '\t',
            print
            print_grades(grades)
            which = 1234
            while which != -1:
                which = input("Change which Grade: ")
                which = which - 1
                if 0 <= which < len(grades):
                    grade = raw_input("Grade: ")  Change from
 input() to raw_input() to avoid an error when saving
                    grades[which] = grade
                elif which != -1:
                    print "Invalid Grade Number"
        else:
            print "Student not found"
    elif menu_choice == 5:
        print_all_grades()
    elif menu_choice == 6:
        gradesfile = raw_input("Save grades to which file? ")
        save_grades(gradesfile)
    elif menu_choice != 9:
        print_menu()
```

18 Dealing with the imperfect

18.0.43 ...or how to handle errors

So you now have the perfect program, it runs flawlessly, except for one detail, it will crash on invalid user input. Have no fear, for Python has a special control structure for you. It's called `try` and it tries to do something. Here is an example of a program with a problem:

```
print "Type Control C or -1 to exit"
number = 1
while number != -1:
    number = int(raw_input("Enter a number: "))
    print "You entered:", number
```

Notice how when you enter `@#&` it outputs something like:

```
Traceback (innermost last):
 File "try_less.py", line 4, in ?
    number = int(raw_input("Enter a number: "))</source>

ValueError: invalid literal for int(): @#&
```

As you can see the `int()` function is unhappy with the number `@#&` (as well it should be). The last line shows what the problem is; Python found a `ValueError`. How can our program deal with this? What we do is first: put the place where the errors occurs in a `try` block, and second: tell Python how we want `ValueError`s handled. The following program does this:

```
print "Type Control C or -1 to exit"
number = 1
while number != -1:
    try:
        number = int(raw_input("Enter a number: "))
        print "You entered:", number
    except ValueError:
        print "That was not a number."
```

Now when we run the new program and give it `@#&` it tells us "That was not a number." and continues with what it was doing before.

When your program keeps having some error that you know how to handle, put code in a `try` block, and put the way to handle the error in the `except` block.

Here is a more complex example of Error Handling.

```
# Program by Mitchell Aikens 2012
# No copyright.
import math
```

```
def main():
        success = 0
        while (success == 0):
                try:
                        epact()
                        success = 1
                except ValueError:
                        print "Error. Please enter an integer value."
                        year = 0
                except NameError:
                        print "Error. Please enter an integer value."
                        year = 0
                except SyntaxError:
                        print "Error. Please enter an integer value."
                        year = 0
                finally:
                        print "Program Complete"

def epact():

    year = int(input("What year is it?\n"))
    C = year/100
    epactval = (8 + (C/4) - C + ((8*C + 13)/25) + 11 * (year%19))%30
    print "The Epact is: ",epactval

main()
```

The program above uses concepts from previous lessons as well as the current lesson. Let's look at the above program in sections.

After we define the function called "main", we tell it that we want to "try" function named "epact". It does so "while" there is no "success". The interpreter then goes to the the line `year = int(input("What year is it?\n"))`. The interpreter takes the value entered by the user and stores it in the variable named "year".

If the value entered is not an integer or a floating point number (which would be converted to an integer by the interpreter), an exception would be raised, and execution of the `try` block ends, just before `success` is assigned the value 1.

Let's look at some possible exceptions. the program above does not have an `except` clause for every possible exception, as there are numerous types or exceptions.

If the value entered for year is an alphabetical character, a `NameError` exception is raised. In the program above, this is caught by the `except NameError:` line, and the interpreter executes the print statement below the `except NameError:`, then it sets the value of "year" to 0 as a precaution, clearing it of any non-numeric number. The interpreter then jumps back to the first line of the `while` loop, and the process restarts.

The process above would be the same for the other exceptions we have. If an exception is raised, and there is an except clause for it in our program, the interpreter will jump to the statements under the appropriate except clause, and execute them.

The `finally` statement, is sometimes used in exception handling as well. Think of it as the trump card. Statements underneath the `finally` clause will be executed regardless of if we raise and exception or not. The `finally` statement will be executed after any `try` or `except` clauses prior to it.

Below is a simpler example where we are not looped, and the `finally` clause is executed regardless of exceptions.

```
#Program By Mitchell Aikens 2012
#Not copyright.

def main():
    try:
        number = int(input("Please enter a number.\n"))
        half = number/2
        print "Half of the number you entered is ",half
    except NameError:
        print "Error."
    except ValueError:
        print "Error."
    except SyntaxError:
        print "Error."
    finally:
        print "I am executing the finally clause."

main()
```

If we were to enter an alphabetic value for `number = int(input("Please enter a number.\n"))`, the output would be as follows:

```
Please enter a number.
t
Error.
I am executing the finally clause.
```

18.0.44 Exercises

Update at least the phone numbers program (in section ../File IO/[1]) so it doesn't crash if a user doesn't enter any data at the menu.

1 Chapter 17 on page 99

19 The End

For the moment I recommend looking at The Python Tutorial[1] by Guido van Rossum[2] for more topics. If you have been following this tutorial, you should be able to understand a fair amount of it. If you want to get deeper into Python, Dive Into Python[3] is a nice on-line textbook, although targeted at people with a more solid programming background. The Python Programming[4] wikibook can be worth looking at, too.

This tutorial is very much a work in progress. Thanks to everyone who has sent me emails about it. I enjoyed reading them, even when I have not always been the best replier.

Happy programming, may it change your life and the world.

1 `http://docs.python.org/tut/tut.html`
2 `http://www.python.org/~guido/`
3 `http://www.diveintopython.org/`
4 `http://en.wikibooks.org/wiki/Python%20Programming`

20 FAQ

***Question:* Can't use programs with input.**

Answer: If you are using IDLE then try using command line. This problem seems to be fixed in IDLE 0.6 and newer. If you are using an older version of IDLE try upgrading to Python 2.0 or newer.

Is there a printable version?

Yes, see the next question.

Is there a PDF or zipped version?

Yes, go to `http://www.honors.montana.edu/~jjc/easytut` for several different versions. Note that this will not always be up to date with the Wikibooks version. The Wikibook can be printed from the print version[1].

What is the tutorial written with?

Originally, LaTeX, see the `easytut.tex` file.

I can't type in programs of more than one line.

If the programs that you type in run as soon as you are typing them in, you need to edit a file instead of typing them in interactive mode. (Hint: interactive mode is the mode with the `>>>` prompt in front of it.)

My question is not answered here.

Ask on the talk page. Please post source code if at all relevant (even, (or maybe especially) if it doesn't work). Helpful things to include are what you were trying to do, what happened, what you expected to happen, error messages, version of Python, Operating System, and whether or not your cat was stepping on the keyboard. (The cat in my house has a fondness for space bars and control keys.)

I want to read it in a different language.

There are several translations that I know of. One is in Korean and is available at `http://home.hanmir.com/~johnsonj/easytut/easytut.html`. Another is in Spanish and at `http://www.honors.montana.edu/~jjc/easytut/easytut_es/`. Another is in Italian and is available at `http://www.python.it/doc/tut_begin/index.html`. Another is in Greek and available at `http://www.honors.montana.edu/~jjc/easytut/easytut_gr/`. Several people have said they are doing a translation in other languages such as French, but I never heard back from them. If you have done a translation or know of any translations, please either send it to me or send me a link.

1 `http://en.wikibooks.org/wiki/..%2FPrint%20version`

How do I make a GUI in Python?

You can use either TKinter at `http://www.python.org/topics/tkinter/` or WXPython at `http://www.wxpython.org/`

How do I make a game in Python?

The best method is probably to use PYgame at `http://pygame.org/`

How do I make an executable from a Python program?

Short answer: Python is an interpreted language so that is impossible. Long answer is that something similar to an executable can be created by taking the Python interpreter and the file and joining them together and distributing that. For more on that problem see `http://www.python.org/cgi-bin/faqw.py?req=all#4.28`. A project that does make executable python files is py2exe - see `http://www.py2exe.org`.

I need help with the exercises

Hint, the password program requires two variables, one to keep track of the number of times the password was typed in, and another to keep track of the last password typed in. Also you can download solutions from `http://www.honors.montana.edu/~jjc/easytut/`

What and when was the last thing changed?

- 2000-Dec-16, added error handling chapter.
- 2000-Dec-22, Removed old install procedure.
- 2001-Jan-16, Fixed bug in program, Added example and data to lists section.
- 2001-Apr-5, Spelling, grammar, added another how to break programs, url fix for PDF version.
- 2001-May-13, Added chapter on debugging.
- 2001-Nov-11, Added exercises, fixed grammar, spelling, and hopefully improved explanations of some things.
- 2001-Nov-19, Added password exercise, revised references section.
- 2002-Feb-23, Moved 3 times password exercise, changed l to list in list examples question. Added a new example to Decisions chapter, added two new exercises.
- 2002-Mar-14, Changed abs to my_abs since python now defines a abs function.
- 2002-May-15, Added a faq about creating an executable. Added a comment from about the list example. Fixed typos from Axel Kleiboemer.
- 2002-Jun-14, Changed a program to use while true instead of while 1 to be more clear.
- 2002-Jul-5, Rewrote functions chapter. Modified fib program to hopefully be clearer.
- 2003-Jan-3, Added average examples to the decisions chapter.
- 2003-Jan-19, Added comment about value of a_var. Fixed mistake in average2.py program.
- 2003-Sep-5, Changed idle instruction to Run->Run Module.
- 2004-Jun-1, Put on Wikibooks
- Since then all changes are visible through the Wikibooks version keeping system.

21 Contributors

Edits	User
55	33rogers[1]
1	Acannon828[2]
40	Adrignola[3]
6	Alain[4]
9	Allen Moore[5]
1	Aruziell[6]
1	Astroman3D[7]
2	Atelaes[8]
4	Balabalame[9]
4	Benjamin Meinl[10]
2	Benonsoftware[11]
4	Cm.squared[12]
1	Dablackwood[13]
4	Darklama[14]
7	Del45[15]
1	Dirk Hünniger[16]
1	Dm3da[17]
19	Dooglus[18]
1	Eva Griffeth[19]
2	Ezzieyguywuf[20]
1	Fishpi[21]

1 `http://en.wikibooks.org/w/index.php?title=User:33rogers`
2 `http://en.wikibooks.org/w/index.php?title=User:Acannon828`
3 `http://en.wikibooks.org/w/index.php?title=User:Adrignola`
4 `http://en.wikibooks.org/w/index.php?title=User:Alain`
5 `http://en.wikibooks.org/w/index.php?title=User:Allen_Moore`
6 `http://en.wikibooks.org/w/index.php?title=User:Aruziell`
7 `http://en.wikibooks.org/w/index.php?title=User:Astroman3D`
8 `http://en.wikibooks.org/w/index.php?title=User:Atelaes`
9 `http://en.wikibooks.org/w/index.php?title=User:Balabalame`
10 `http://en.wikibooks.org/w/index.php?title=User:Benjamin_Meinl`
11 `http://en.wikibooks.org/w/index.php?title=User:Benonsoftware`
12 `http://en.wikibooks.org/w/index.php?title=User:Cm.squared`
13 `http://en.wikibooks.org/w/index.php?title=User:Dablackwood`
14 `http://en.wikibooks.org/w/index.php?title=User:Darklama`
15 `http://en.wikibooks.org/w/index.php?title=User:Del45`
16 `http://en.wikibooks.org/w/index.php?title=User:Dirk_H%C3%BCnniger`
17 `http://en.wikibooks.org/w/index.php?title=User:Dm3da`
18 `http://en.wikibooks.org/w/index.php?title=User:Dooglus`
19 `http://en.wikibooks.org/w/index.php?title=User:Eva_Griffeth`
20 `http://en.wikibooks.org/w/index.php?title=User:Ezzieyguywuf`
21 `http://en.wikibooks.org/w/index.php?title=User:Fishpi`

1	G-Brain[22]
1	Geocachernemesis[23]
2	Herbythyme[24]
1	Jkover9000[25]
5	Jomegat[26]
5	JoshuaGolbez[27]
114	Jrincayc[28]
4	Jshadias[29]
4	Koex[30]
1	Krade[31]
1	Kristianpaul[32]
2	Legoktm[33]
1	LudoA[34]
1	Mabdul[35]
1	Mats Halldin[36]
6	Monobi[37]
1	MrChimp[38]
1	Msaikens[39]
1	Neoptolemus[40]
1	Ngch89[41]
2	NipplesMeCool[42]
1	Noclue[43]
4	OmnificienT[44]
1	Pancake[45]
1	Panic2k4[46]

22 http://en.wikibooks.org/w/index.php?title=User:G-Brain
23 http://en.wikibooks.org/w/index.php?title=User:Geocachernemesis
24 http://en.wikibooks.org/w/index.php?title=User:Herbythyme
25 http://en.wikibooks.org/w/index.php?title=User:Jkover9000
26 http://en.wikibooks.org/w/index.php?title=User:Jomegat
27 http://en.wikibooks.org/w/index.php?title=User:JoshuaGolbez
28 http://en.wikibooks.org/w/index.php?title=User:Jrincayc
29 http://en.wikibooks.org/w/index.php?title=User:Jshadias
30 http://en.wikibooks.org/w/index.php?title=User:Koex
31 http://en.wikibooks.org/w/index.php?title=User:Krade
32 http://en.wikibooks.org/w/index.php?title=User:Kristianpaul
33 http://en.wikibooks.org/w/index.php?title=User:Legoktm
34 http://en.wikibooks.org/w/index.php?title=User:LudoA
35 http://en.wikibooks.org/w/index.php?title=User:Mabdul
36 http://en.wikibooks.org/w/index.php?title=User:Mats_Halldin
37 http://en.wikibooks.org/w/index.php?title=User:Monobi
38 http://en.wikibooks.org/w/index.php?title=User:MrChimp
39 http://en.wikibooks.org/w/index.php?title=User:Msaikens
40 http://en.wikibooks.org/w/index.php?title=User:Neoptolemus
41 http://en.wikibooks.org/w/index.php?title=User:Ngch89
42 http://en.wikibooks.org/w/index.php?title=User:NipplesMeCool
43 http://en.wikibooks.org/w/index.php?title=User:Noclue
44 http://en.wikibooks.org/w/index.php?title=User:OmnificienT
45 http://en.wikibooks.org/w/index.php?title=User:Pancake
46 http://en.wikibooks.org/w/index.php?title=User:Panic2k4

1	QuiteUnusual[47]
1	Rmunn[48]
2	Robot Chicken[49]
4	Rpruyne[50]
1	Sam Hocevar[51]
1	Seshull[52]
185	Siebengang[53]
1	Spacebar265[54]
1	Specialized[55]
2	Spedley[56]
1	Switch32763[57]
2	TBOL3[58]
1	Taxman[59]
3	Tehdrago[60]
1	The Kid[61]
2	TheLostOne[62]
2	Tualha[63]
1	Van der Hoorn[64]
1	Webaware[65]
58	Whiteknight[66]
2	Wmcleod[67]
1	Wutsje[68]
2	Xania[69]
2	Zastard[70]

47 `http://en.wikibooks.org/w/index.php?title=User:QuiteUnusual`
48 `http://en.wikibooks.org/w/index.php?title=User:Rmunn`
49 `http://en.wikibooks.org/w/index.php?title=User:Robot_Chicken`
50 `http://en.wikibooks.org/w/index.php?title=User:Rpruyne`
51 `http://en.wikibooks.org/w/index.php?title=User:Sam_Hocevar`
52 `http://en.wikibooks.org/w/index.php?title=User:Seshull`
53 `http://en.wikibooks.org/w/index.php?title=User:Siebengang`
54 `http://en.wikibooks.org/w/index.php?title=User:Spacebar265`
55 `http://en.wikibooks.org/w/index.php?title=User:Specialized`
56 `http://en.wikibooks.org/w/index.php?title=User:Spedley`
57 `http://en.wikibooks.org/w/index.php?title=User:Switch32763`
58 `http://en.wikibooks.org/w/index.php?title=User:TBOL3`
59 `http://en.wikibooks.org/w/index.php?title=User:Taxman`
60 `http://en.wikibooks.org/w/index.php?title=User:Tehdrago`
61 `http://en.wikibooks.org/w/index.php?title=User:The_Kid`
62 `http://en.wikibooks.org/w/index.php?title=User:TheLostOne`
63 `http://en.wikibooks.org/w/index.php?title=User:Tualha`
64 `http://en.wikibooks.org/w/index.php?title=User:Van_der_Hoorn`
65 `http://en.wikibooks.org/w/index.php?title=User:Webaware`
66 `http://en.wikibooks.org/w/index.php?title=User:Whiteknight`
67 `http://en.wikibooks.org/w/index.php?title=User:Wmcleod`
68 `http://en.wikibooks.org/w/index.php?title=User:Wutsje`
69 `http://en.wikibooks.org/w/index.php?title=User:Xania`
70 `http://en.wikibooks.org/w/index.php?title=User:Zastard`

List of Figures

- GFDL: Gnu Free Documentation License. `http://www.gnu.org/licenses/fdl.html`
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. `http://creativecommons.org/licenses/by-sa/3.0/`
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. `http://creativecommons.org/licenses/by-sa/2.5/`
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. `http://creativecommons.org/licenses/by-sa/2.0/`
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. `http://creativecommons.org/licenses/by-sa/1.0/`
- cc-by-2.0: Creative Commons Attribution 2.0 License. `http://creativecommons.org/licenses/by/2.0/`
- cc-by-2.0: Creative Commons Attribution 2.0 License. `http://creativecommons.org/licenses/by/2.0/deed.en`
- cc-by-2.5: Creative Commons Attribution 2.5 License. `http://creativecommons.org/licenses/by/2.5/deed.en`
- cc-by-3.0: Creative Commons Attribution 3.0 License. `http://creativecommons.org/licenses/by/3.0/deed.en`
- GPL: GNU General Public License. `http://www.gnu.org/licenses/gpl-2.0.txt`
- LGPL: GNU Lesser General Public License. `http://www.gnu.org/licenses/lgpl.html`
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. `http://artlibre.org/licence/lal/de`
- CFR: Copyright free use.

- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[71]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

71 Chapter 22 on page 123

72 http://en.wikibooks.org/wiki/User%3ADsmurat
73 http://en.wikibooks.org/wiki/User%3APenubag

22 Licenses

22.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

22.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title

22.3 GNU Lesser General Public License