Chapter 2

Simple Python Code

In this chapter we will look at the basic elements of programming in Python. There isn't a lot you can do without the control structures of Chapter 3, so we will move quickly through this material. You might want to review this after you have read Chapters 3 and 4. Though none of this is difficult, there are some subtleties here that you might not appreciate unil you have some programming experience.

2.1 A first program

In this chapter we will begin to write programs using the simplest elements of Python. There are several factors that make this a complex, and sometimes frustrating, process. One is that programs are more formal than human communication, and everything in a program needs to be correct in order for it to run. If you are just starting to learn a new human language, such as Spanish, you might put a verb in the wrong tense or use the wrong gender for a noun but people will still know what you mean. Computers aren't as good as people at finding the essence of what you are saying. If your program isn't completely correct, the computer will not run it. Fortunately, many of the formal details of programs are the same in program after program; you can use them almost like incantations to achieve what you want.

Another factor that makes programming complicated at the start is that we want to eventually write complex programs that do useful things. In this chapter we are trying to lay the groundwork for a full understanding of programming, so at times we will use techniques that are more complex than necessary for the task immediately at hand, but that use a style that will be easy to adapt later on.

There are four primary types of instructions in Python, and in most programming languages: definitions, expressions, statements and comments. An *executable* instruction does something. For example, the line of code $x = 3 + 4$ adds together the numbers 3 and 4 and stores the result in variable x. There are two kinds of executable instructions. An *expression* represents a value, such as `3+4`. A *statement* does something, such as **print** `x`. A *definition* creates something and gives it a name for later use. Nothing appears to happen when you execute a definition, but the system's memory is modified to include the newly defined object. Finally, a *comment* is a note for human readers of the program. Comments are ignored by the computer; their sole purpose is to help humans understand the program. Comments in Python start with the symbol # and extend to the end of the line.

The Python system starts reading your program at the top; it reads and executes statements until it gets to the end. Of course, all that happens when it executes a definition is that its memory is modifed to include the new object. Consider Program 2.1.1:

```
# This asks for the user's name
# and prints a greeting.

def main():
    name = input( "Who are you? " )
    if name == 'bob':
        print( "Bob rules!" )
    else:
        print( "Hello, " + name + "!" )
    print( "Goodbye." )

main()
```

Program 2.1.1: Our first program

This starts with two lines of comments:

```
# This asks for the user's name
# and prints a greeting.
```

Anyone reading the program can tell the program's purpose from these two lines without reading any further. The program then has a definition:

```
def main():
    name = input( "Who are you? " )
    if name == 'bob':
        print( "Bob rules!" )
    else:
        print( "Hello, " + name + "!" )
    print( "Goodbye." )

main()
```

This defines an object called main(). Objects like this that have parentheses as part of their names are called *functions*. These are the one of the primary building blocks of programs. Most of the programs we will write will have a sequence of function definitions at the start. Note that there are many lines indented underneath the line

```
def main():
```

These lines make up the body of the function - the instructions to be executed when the function is used. Python is different from most programing languages in that it uses indentation as a structural part of programs. Other languages uses braces, such as { and } or words such as begin and end to group together statements. Python's use of white space (space characters and tabs) makes it very visual; many people think this makes Python programs easier to read.

The final line of Program 2.1.1 is

```
main()
```

This invokes, or calls, function main() that was defined earlier. The system calls a function by executing the statements in its body one at a time. First, we have

```
name = input( "Who are you? " )
```

This is complex. The string "Who are you? " is printed and the system halts and waits for the user to type a line of text that is terminated by the *Return* key. Whatever the user types is given as a string to the variable name.

The next instruction in main() is an *if*-statement:

```
if name ==   'bob':
        print( "Bob rules!" )
    else:
        print( "Hello, " + name + "!" )
```

This compares the value stored in variable name to the string 'bob'. If they match, the system prints 'Bob rules'. If they don't match it prints the word 'Hello', followed by the value in variable name followed by an exclamation point.

The last line of function main() just prints the word "Goodbye." This line is not part of the *if*-statement and so it is executed regardless of what name the user enters. Note how Python's use of indentation helps us to understand the functionality of the program. If we indented the last line, so that main() reads

```
def main():
    name = input( "Who are you? " )
    if name == 'bob':
        print( "Bob rules!" )
    else:
        print( "Hello, " + name + "!" )
        print( "Goodbye." )
```

then, in the case where the user enters bob the system will only print "Bob rules!" and if the user enters 'Mary' the system will respond

Hello, Mary!
Goodbye.

We run this program by opening it in a Python window inside Idle, or else typing it into a new file in Idle. One of the menus in this window is called Run, and there is an option in this menu called Run module F5 Either select this option or else press the F5 key to run the program. Here is a typical run, with what the user prints in bold and what the user types in normal font:

Who are you? `Mary`
Hello, Mary!
Goodbye.

Program 2.1.1 has a structure that we will use over and over. Until we discuss functions more completely in Chapter 5, all of our programs will have the following form

```
# comment on what the program does
def main():
        ⋮
main()
```

The use of main() here as the function that holds the body of the program goes back to the C language, which required a function called ain()} to fill this role. Python doesn't care what the is; our programs would run just as well with the structure:

```
def stuff():
        ⋮

stuff()
```

We will use the name "main" as a concession to tradition.

Note that Program 2.1.1 is more complex than it actually needs to be. Python is a scripting language, which means that it can execute sequences of statements without an encompassing program. Program 2.1.2, below, runs in the same way as Program 2.1.1. For very short programs this can be useful, but as soon as the programs gain a little complexity this style becomes very hard to follow. For this reason we will stick to the style of Program 2.1.1 for all of our subsequent programs.

```
# This asks for the user's name
# and prints a greeting.

name = input( "Who are you? " )
if name == 'bob':
     print( "Bob rules!" )
else:
     print( "Hello, " + name + "!" )
print( "Goodbye." )
```

Program 2.1.2: Our second program

So far, programming should look easy. Programming is a great spectator sport. It is easy to nod your head and say "Yes, I understand." as long as someone else is doing the coding. When you start writing your own programs matters become a little more complex. In the next section we will look at some

of the mistakes you might make, and how the system will respond to these mistakes.