1.7 IF STATEMENTS

In Python an if statement begins with the word **if**, followed by an expression that evaluates to **True** or **False**, followed by a colon (**:**), then a series of statements that are executed if the expression is true. The names True and False are constants having the obvious meaning, and a variable that can take on these values is a *logical* or *Boolean* (named after the man who invented two state or logical algebra) variable. The expression is the only tricky part. It can be a constant like **True**, or a variable that has a **True** or **False** value, or a *relational expression* (one that compares two things) or a logical combination of any of these—anything that has a result that is true or false.

```
if True:          # Constant
if flag:          # Logical variable
if a < b:         # relational expression
if a<b and c>d:   # logical combination
```

A logical expression can be any arithmetic expressions being compared using any of the following operators:

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Logical combinations can be:

```
and     EG:    a==b and b==c
or      EG:    a==b or a==c
not     EG:    not (a == b)        # same as !=
```

The syntax is simple and yet allows a huge number of combinations. For example:

```
if p == q and not p == z and not z == p:
if pi**2 < 12:
if (a**b)**(c-d)/3 <= z**3:
```

The *consequent*, or the actions to be taken if the logical expression is true, follows the colon on the following lines. The next statement is indented more than the **if**, and all statements that follow immediately that have the same indentation

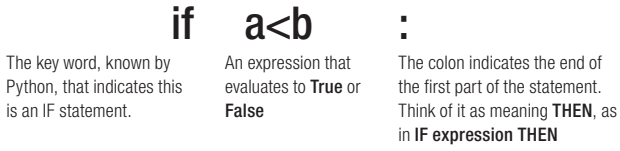

Figure 1.4
Syntax of an IF statement.

are a part of the consequent and are executed if the condition is true, otherwise none of them are. As an example, consider:

```
if a < b:
   a = a + 1
   b = b - 1
c = a - b
```

In this case the two statements following the ":" are indented by 4 more spaces than is the **if**. This tells Python that they are both a part of the **if** statement, and that if the value of **a** is smaller than the value of **b**, then both of those statements will be executed. Python calls such a group of statements a *suite*. The assignment to the variable c is indented to the same level as the **if**, so it will be executed in any case and is not conditional.

The use of indentation to connect statements into groups is unusual in programming languages. Most languages in use pretty much ignore spaces and line breaks altogether, and use a statement separator such as a semicolon to demark statements. So, in the Java language the above code would look like this:

```
if (a<b) {
  a = a + 1;
  b = b - 1;
}
c = a - b;
```

The braces { … } enclose the suite, which would probably be called a *block* in Java or C++. Notice that this code is also indented, but in Java this means nothing to the computer. Indentation is used for clarity, so that someone reading the code later can see more clearly what is happening.

Semicolons are used in Python too, but much more rarely. If it is desired to place more than one statement on a single line, then semicolons can be used to separate them. The Python **if** statement under consideration here could be written as:

```
if a < b:
```

```
a = a + 1;
b = b -1
c = a - b
```

This is harder to comprehend quickly and is therefore less desirable. There are too many symbols all grouped together. A program that is easy to read is also easier to modify and maintain. Code is written for computers to execute, but it is also for humans to read.

There are some special assignment operators that can be used for incrementing and decrementing variables. In the above code the statement **a = a + 1** could be written as **a += 1**, and **b = b – 1** can be written as **b -= 1**. There is no real advantage to doing this, but other languages permit it so Python adopted it too. There is another syntax that can be used to simplify certain code in languages like Java and C, and that is the increment operator "++" and the decrement operator "—." Python does not have these. However, an effect of the way that Python deals with variables and expressions is that "**++x**" is legal; so is "**++++x**." The value is simply **x**. The expression "**x++**" is not correct.

1.7.1 Else

An **if** statement is a two-way or *binary* decision. If the expression is true, then the indicated statements are executed. If it is not true, then it is possible to execute a distinct set of statements. This is needed for the *pick a number* program. In one case the computer wins, and in the other the human wins. An *else* clause is what will allow this.

The ***else*** is not really a statement on its own, because it has to be preceded by an **if**, so it's part of the **if** statement. It marks the part of the statement that is executed only when the condition in the **if** statement is false. It consists of the word else followed by a colon, followed by a suite (sequence of indented statements). So a trivial example is:

```
if True:
   print ("The condition was true")
else:
   print ("the condition was false")
```

The **else** as a clause is not required to accomplish any specific programming goals, and it can be implemented using another **if**. The code:

```
if a < b:
   print ("a < b")
else:
   print ("a >= b")
```

could also be written as:

```
if a < b:
   print ("a < b")
if  not (a<b):
   print ("a >= b")
```

The **else** is *expressive*, *efficient*, and *syntactically convenient*. It is expressive because it represents a way that humans actually communicate. The word *else* means pretty much the same thing in Python as it does in English. It is efficient because it avoids evaluating the same expression twice, which costs something in terms of execution speed. And it is syntactically convenient because it expresses an important element of the language in fewer symbols than when two **if**s are used.

The final Python code for the simple solution of the guess a number program can now be written. It is:

```
choice = 7
print ("Please guess a number between 1 and 10: ")
playerchoice = int(input())
if choice == playerchoice:
     print ("You win!")
else:
     print ("Sorry, You lose.")
```

1.8 DOCUMENTATION

There are some problems with this program, but is does work. A large problem is that it always chooses the same number every time it is executed (that number being 7). This will be fixed later on. A less critical problem is that it is *undocumented*; that is, there are no instructions to a player concerning how to use the program and there is no description of how the program works that another programmer might use if modifying this code. This can be fixed by providing *internal* and *external* documentation.

External documentation is like a manual for the user. Most programs have such a thing, and even though this program is quite simple, some degree of