Chapter 4

If statements

Quite often in programs we only want to do something provided something else is true. Python's `if` statement is what we need.

4.1 A Simple Example

Let's try a guess-a-number program. The computer picks a random number, the player tries to guess, and the program tells them if they are correct. To see if the player's guess is correct, we need something new, called an *if statement*.

```
from random import randint

num = randint(1,10)
guess = eval(input('Enter your guess: '))
if guess==num:
    print('You got it!')
```

The syntax of the if statement is a lot like the `for` statement in that there is a colon at the end of the if condition and the following line or lines are indented. The lines that are indented will be executed only if the condition is true. Once the indentation is done with, the if block is concluded.

The guess-a-number game works, but it is pretty simple. If the player guesses wrong, nothing happens. We can add to the if statement as follows:

```
if guess==num:
    print('You got it!')
else:
    print('Sorry.  The number is ', num)
```

We have added an `else` statement, which is like an "otherwise."

4.2 Conditional operators

The comparison operators are ==, >, <, >=, <=, and !=. That last one is for *not equals*. Here are a few examples:

Expression	Description
`if x>3:`	if x is greater than 3
`if x>=3:`	if x is greater than or equal to 3
`if x==3:`	if x is 3
`if x!=3:`	if x is not 3

There are three additional operators used to construct more complicated conditions: `and`, `or`, and `not`. Here are some examples:

```
if grade>=80 and grade<90:
    print('Your grade is a B.')

if score>1000 or time>20:
    print('Game over.')

if not (score>1000 or time>20):
    print('Game continues.')
```

Order of operations In terms of order of operations, `and` is done before `or`, so if you have a complicated condition that contains both, you may need parentheses around the `or` condition. Think of `and` as being like multiplication and `or` as being like addition. Here is an example:

```
if (score<1000 or time>20) and turns_remaining==0:
    print('Game over.')
```

4.3 Common Mistakes

Mistake 1 The operator for equality consists of two equals signs. It is a really common error to forget one of the equals signs.

Incorrect	Correct
`if x=1:`	`if x==1:`

Mistake 2 A common mistake is to use `and` where `or` is needed or vice-versa. Consider the following if statements:

```
if x>1 and x<100:
if x>1 or x<100:
```

The first statement is the correct one. If x is any value between 1 and 100, then the statement will be true. The idea is that x has to be *both* greater than 1 *and* less than 100. On the other hand, the second statement is not what we want because for it to be true, *either* x has to be greater than 1 *or* x has to be less than 100. But every number satisfies this. The lesson here is if your program is not working correctly, check your **and**'s and **or**'s.

Mistake 3 Another very common mistake is to write something like below:

```
if grade>=80 and <90:
```

This will lead to a syntax error. We have to be explicit. The correct statement is

```
if grade>=80 and grade<90:
```

On the other hand, there is a nice shortcut that does work in Python (though not in many other programming languages):

```
if 80<=grade<90:
```

4.4 elif

A simple use of an if statement is to assign letter grades. Suppose that scores 90 and above are A's, scores in the 80s are B's, 70s are C's, 60s are D's, and anything below 60 is an F. Here is one way to do this:

```
grade = eval(input('Enter your score: '))

if grade>=90:
    print('A')
if grade>=80 and grade<90:
    print('B')
if grade>=70 and grade<80:
    print('C')
if grade>=60 and grade<70:
    print('D')
if grade<60:
    print('F')
```

The code above is pretty straightforward and it works. However, a more elegant way to do it is shown below.

```
grade = eval(input('Enter your score: '))

if grade>=90:
    print('A')
elif grade>=80:
    print('B')
elif grade>=70:
    print('C'):
```