CHAPTER 3

DECISIONS

CHAPTER GOALS

To implement decisions using `if` statements

To compare integers, floating-point numbers, and strings

To write statements using Boolean expressions

To develop strategies for testing your programs

To validate user input

CHAPTER CONTENTS

One of the essential features of computer programs is their ability to make decisions. Like a train that changes tracks depending on how the switches are set, a program can take different actions depending on inputs and other circumstances.

In this chapter, you will learn how to program simple and complex decisions. You will apply what you learn to the task of checking user input.

3.1 The `if` Statement

The `if` statement allows a program to carry out different actions depending on the nature of the data to be processed.

The `if` statement is used to implement a decision (see Syntax 3.1 on page 94). When a condition is fulfilled, one set of statements is executed. Otherwise, another set of statements is executed.

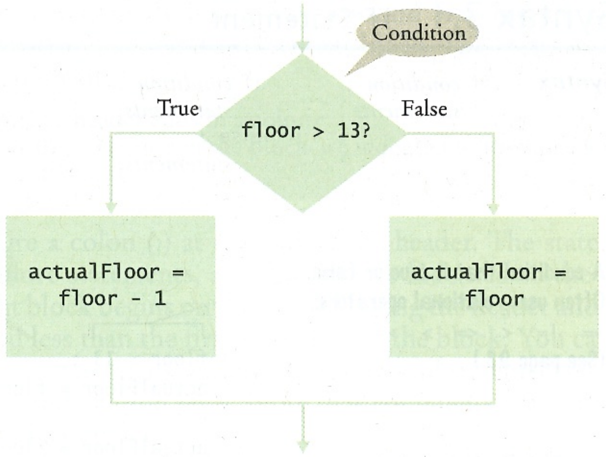
Here is an example using the `if` statement: In many countries, the number 13 is considered unlucky. Rather than offending superstitious tenants, building owners sometimes skip the thirteenth floor; floor 12 is immediately followed by floor 14. Of course, floor 13 is not usually left empty or, as some conspiracy theorists believe, filled with secret offices and research labs. It is simply called floor 14. The computer that controls the building elevators needs to compensate for this foible and adjust all floor numbers above 13.

Let's simulate this process in Python. We will ask the user to type in the desired floor number and then compute the actual floor. When the input is above 13, then we need to decrement the input to obtain the actual floor.

This elevator panel "skips" the thirteenth floor. The floor is not actually missing—the computer that controls the elevator adjusts the floor numbers above 13.

An `if` statement is like a fork in the road. Depending upon a decision, different parts of the program are executed.

Figure 1
Flowchart for if Statement



For example, if the user provides an input of 20, the program determines the actual floor as 19. Otherwise, we simply use the supplied floor number.

```
actualFloor = 0

if floor > 13 :
   actualFloor = floor - 1
else :
   actualFloor = floor
```

The flowchart in Figure 1 shows the branching behavior.

In our example, each branch of the `if` statement contains a single statement. You can include as many statements in each branch as you like. Sometimes, it happens that there is nothing to do in the `else` branch of the statement. In that case, you can omit it entirely, such as in this example:

```
actualFloor = floor

if floor > 13 :
   actualFloor = actualFloor - 1
```

See Figure 2 for the flowchart.

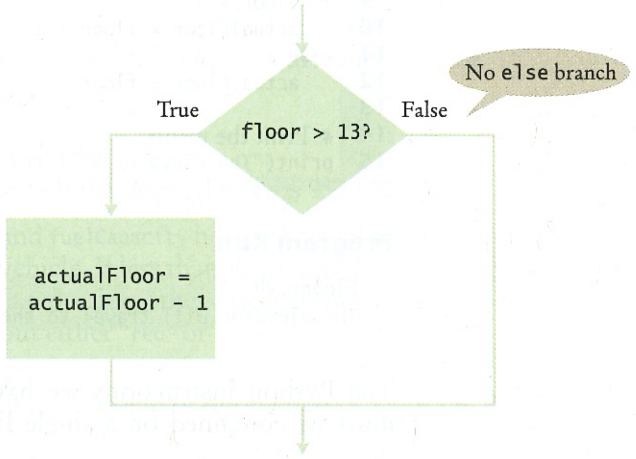


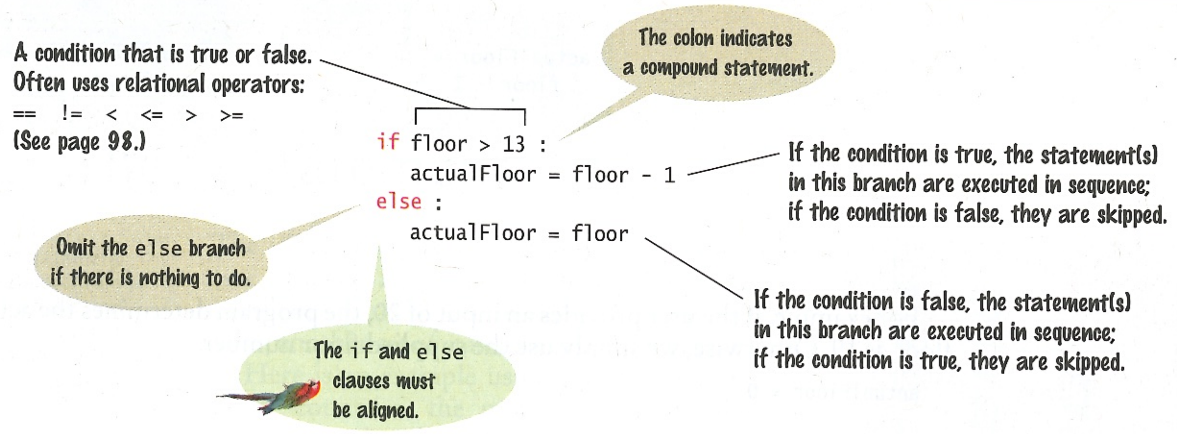
Figure 2
Flowchart for if Statement with No else Branch

Syntax 3.1 if Statement

Syntax

```
if condition :
   statements

if condition :
   statements_1
else :
   statements_2
```



```
if floor > 13 :
   actualFloor = floor - 1
else :
   actualFloor = floor
```

A condition that is true or false. Often uses relational operators: == != < <= > >= (See page 98.)
The colon indicates a compound statement.
If the condition is true, the statement(s) in this branch are executed in sequence; if the condition is false, they are skipped.
Omit the else branch if there is nothing to do.
If the condition is false, the statement(s) in this branch are executed in sequence; if the condition is true, they are skipped.
The if and else clauses must be aligned.

The following program puts the `if` statement to work. This program asks for the desired floor and then prints out the actual floor.

ch03/elevatorsim.py

```python
##
#   This program simulates an elevator panel that skips the 13th floor.
#

# Obtain the floor number from the user as an integer.
floor = int(input("Floor: "))

# Adjust floor if necessary.
if floor > 13 :
   actualFloor = floor - 1
else :
   actualFloor = floor

# Print the result.
print("The elevator will travel to the actual floor", actualFloor)
```

Program Run

```
Floor: 20
The elevator will travel to the actual floor 19
```

The Python instructions we have used so far have been simple statements that must be contained on a single line (or explicitly continued to the next line—see

Compound statements consist of a header and a statement block.

Special Topic 2.3). Some constructs in Python are **compound statements,** which span multiple lines and consist of a *header* and a **statement block.** The `if` statement is an example of a compound statement.

```
if totalSales > 100.0 :   # The header ends in a colon.
   discount = totalSales * 0.05    # Lines in the block are indented to the same level
   totalSales = totalSales - discount
   print("You received a discount of", discount)
```

Compound statements require a colon (:) at the end of the header. The statement block is a group of one or more statements, all of which are indented to the same indentation level. A statement block begins on the line following the header and ends at the first statement indented less than the first statement in the block. You can use any number of spaces to indent statements within a block, but all statements within the block must have the same indentation level. Note that comments are not statements and thus can be indented to any level.

Statement blocks, which can be nested inside other blocks, signal that one or more statements are part of the given compound statement. In the case of the `if` construct, the statement block specifies the instructions that will be executed if the condition is true or skipped if the condition is false.

SELF CHECK

1. In some Asian countries, the number 14 is considered unlucky. Some building owners play it safe and skip *both* the thirteenth and the fourteenth floor. How would you modify the sample program to handle such a building?
2. Consider the following `if` statement to compute a discounted price:
```
if originalPrice > 100 :
   discountedPrice = originalPrice - 20
else :
   discountedPrice = originalPrice - 10
```
What is the discounted price if the original price is 95? 100? 105?
3. Compare this `if` statement with the one in Self Check 2:
```
if originalPrice < 100 :
   discountedPrice = originalPrice - 10
else :
   discountedPrice = originalPrice - 20
```
Do the two statements always compute the same value? If not, when do the values differ?
4. Consider the following statements to compute a discounted price:
```
discountedPrice = originalPrice
if originalPrice > 100 :
   discountedPrice = originalPrice - 10
```
What is the discounted price if the original price is 95? 100? 105?
5. The variables `fuelAmount` and `fuelCapacity` hold the actual amount of fuel and the size of the fuel tank of a vehicle. If less than 10 percent is remaining in the tank, a status light should show a red color; otherwise it shows a green color. Simulate this process by printing out either `"red"` or `"green"`.

Practice It Now you can try these exercises at the end of the chapter: R3.5, R3.6, P3.32.

Common Error 3.1

Tabs

Block-structured code has the property that nested statements are indented by one or more levels:

```
if totalSales > 100.0 :
↑  discount = totalSales * 0.05
|  totalSales = totalSales - discount
|  print("You received a discount of $%.2f" % discount)
else :
↑  diff = 100.0 - totalSales
|  if diff < 10.0 :
|  ↑  print("If you were to purchase our item of the day you can receive a 5% discount.")
|  else :
|  ↑  print("You need to spend $%.2f more to receive a 5% discount." % diff)
|  |  ↑
|  |  |
0  1  2  Indentation level
```

Python requires block-structured code as part of its syntax. The alignment of statements within a Python program specifies which statements are part of a given statement block.

How do you move the cursor from the leftmost column to the appropriate indentation level? A perfectly reasonable strategy is to hit the space bar a sufficient number of times. With most editors, you can use the Tab *key* instead. A tab moves the cursor to the next indentation level. Some editors even have an option to fill in the tabs automatically.

While the Tab *key* is nice, some editors use *tab characters* for alignment, which is not so nice. Python is very picky as to how you align the statements within a statement block. All of the statements must be aligned with either blank spaces or tab characters, but not a mixture of the two. In addition, tab characters can lead to problems when you send your file to another person or a printer. There is no universal agreement on the width of a tab character, and some software will ignore tab characters altogether. It is therefore best to save your files with spaces instead of tabs. Most editors have a setting to automatically convert all tabs to spaces.

Look at the documentation of your development environment to find out how to activate this useful setting.

Programming Tip 3.1

Avoid Duplication in Branches

Look to see whether you *duplicate code* in each branch. If so, move it out of the `if` statement. Here is an example of such duplication:

```
if floor > 13 :
   actualFloor = floor - 1
   print("Actual floor:", actualFloor)
else :
   actualFloor = floor
   print("Actual floor:", actualFloor)
```

The output statement is exactly the same in both branches. This is not an error—the program will run correctly. However, you can simplify the program by moving the duplicated statement, like this:

```
if floor > 13 :
   actualFloor = floor - 1
else :
   actualFloor = floor
print("Actual floor:", actualFloor)
```

Removing duplication is particularly important when programs are maintained for a long time. When there are two sets of statements with the same effect, it can easily happen that a programmer modifies one set but not the other.

Special Topic 3.1

Conditional Expressions

Python has a conditional operator of the form

$value_1$ `if` *condition* `else` $value_2$

The value of that expression is either $value_1$ if the condition is true or $value_2$ if it is false. For example, we can compute the actual floor number as

```
actualFloor = floor - 1 if floor > 13 else floor
```

which is equivalent to

```
if floor > 13 :
   actualFloor = floor - 1
else :
   actualFloor = floor
```

Note that a conditional expression is a single statement that must be contained on a single line or continued to the next line (see Special Topic 2.3). Also note that a colon is not needed because a conditional expression is not a compound statement.

You can use a conditional expression anywhere that a value is expected, for example:

```
print("Actual floor:", floor - 1 if floor > 13 else floor)
```

We don't use the conditional expression in this book, but it is a convenient construct that you will find in some Python programs.

3.2 Relational Operators

Use relational operators (< <= > >= == !=) to compare numbers and strings.

In Python, you use a relational operator to check whether one value is greater than another.

In this section, you will learn how to compare numbers and strings in Python.

Every `if` statement contains a condition. In many cases, the condition involves comparing two values. For example, in the previous examples we tested `floor > 13`. The comparison `>` is called a **relational operator**. Python has six relational operators (see Table 1).

As you can see, only two Python relational operators (`>` and `<`) look as you would expect from the mathematical notation. Computer keyboards do not have keys for ≥, ≤, or ≠, but the `>=`, `<=`, and `!=` operators are easy to remember because they look similar. The `==` operator is initially confusing to most newcomers to Python.