

Chapter Three

PART ONE: DECISIONS, RELATIONAL
OPERATORS



Chapter Goals

- To implement decisions using the if statement
- To compare integers, floating-point numbers, and Strings
- To write statements using the Boolean data type
- To develop strategies for testing your programs
- To validate user input

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.

Contents

- The **if** Statement
- Relational Operators
- Nested Branches
- Multiple Alternatives
- Problem Solving: Flowcharts
- Problem Solving: Test Cases
- Boolean Variables and Operators
- Analyzing Strings
- Application: Input Validation



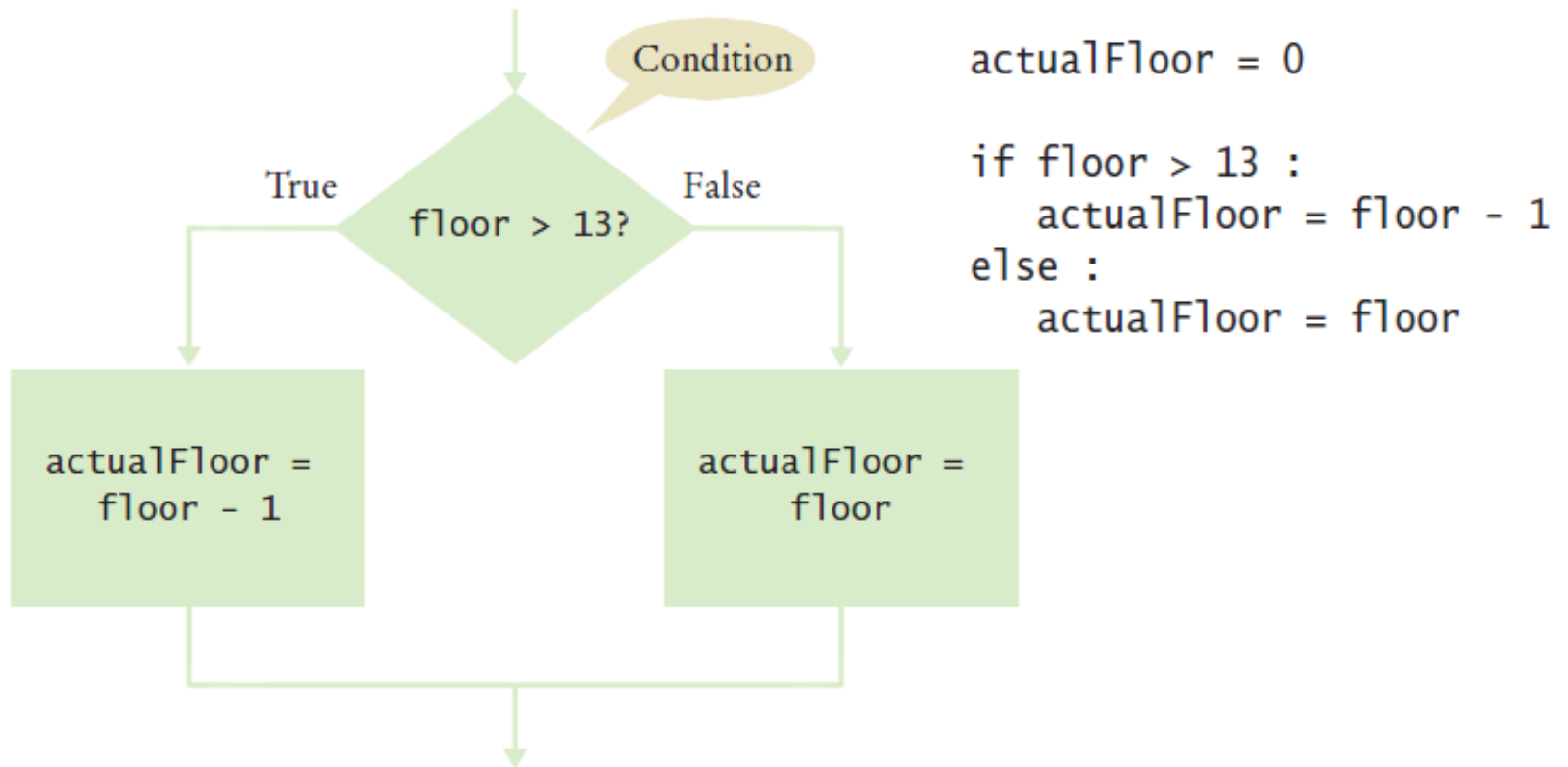
The **if** Statement

- A computer program often needs to make decisions based on input, or circumstances
- For example, buildings often ‘skip’ the 13th floor, and elevators should too
 - The 14th floor is really the 13th floor
 - So every floor above 12 is really ‘floor - 1’
 - If floor > 12, Actual floor = floor - 1
- The two keywords of the if statement are:
 - **if**
 - **else**

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

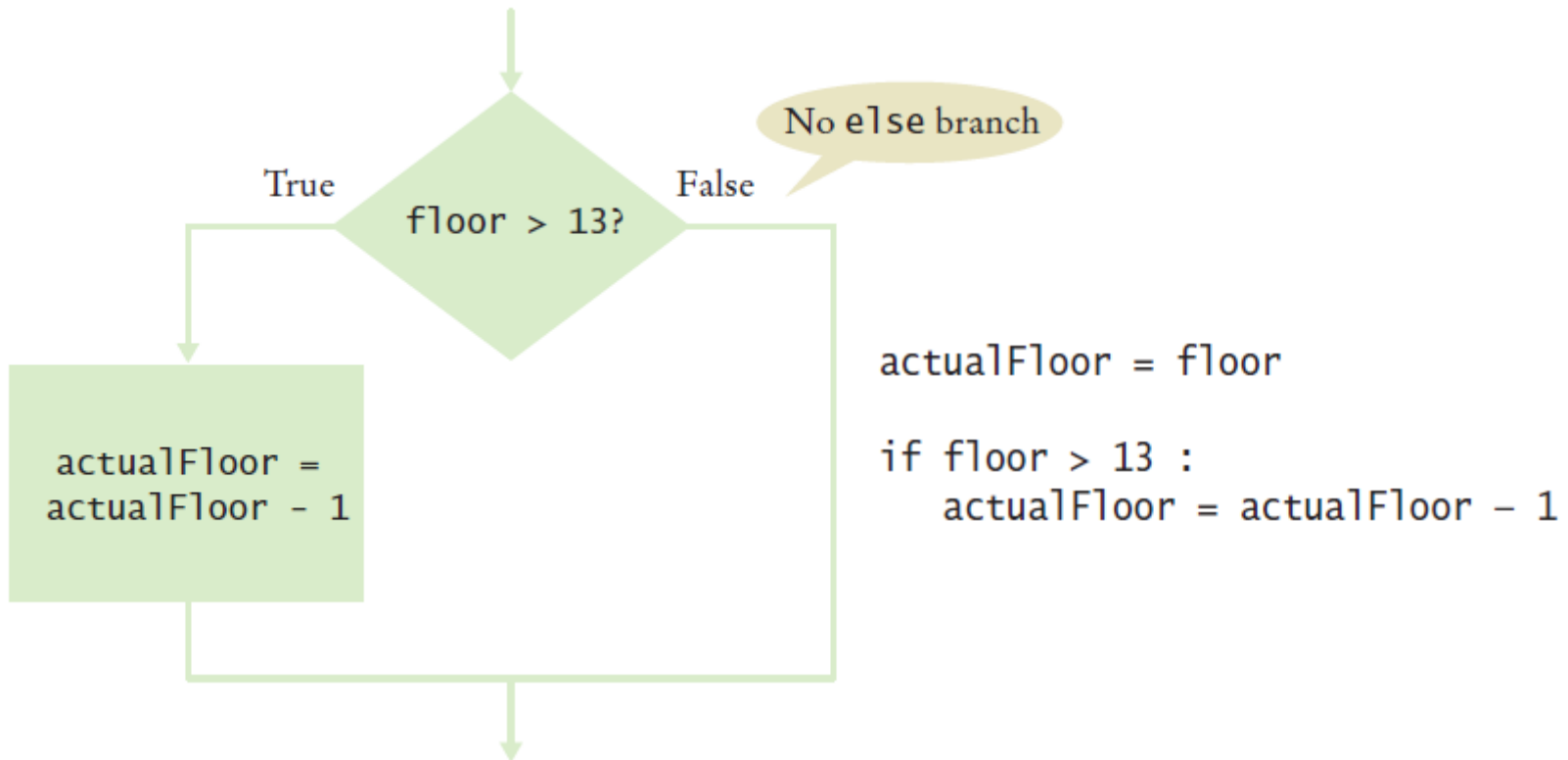
Flowchart of the **if** Statement

- One of the two branches is executed once
 - True (**if**) branch or False (**else**) branch



Flowchart with only a True Branch

- An **if** statement may not need a 'False' (**else**) branch



Syntax 3.1: The **if** Statement

Syntax **if** *condition* :
 statements

if *condition* :
 *statements*₁
 else :
 *statements*₂

A condition that is true or false.
Often uses relational operators:

`==` `!=` `<` `<=` `>` `>=`

(See page 98.)

The colon indicates
a compound statement.

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

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.



Elevatorsim.py

```
1  ##
2  # This program simulates an elevator panel that skips the 13th floor.
3  #
4
5  # Obtain the floor number from the user as an integer.
6  floor = int(input("Floor: "))
7
8  # Adjust floor if necessary.
9  if floor > 13 :
10     actualFloor = floor - 1
11 else :
12     actualFloor = floor
13
14 # Print the result.
15 print("The elevator will travel to the actual floor", actualFloor)
```

Program Run

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

Our First Example

- Open the file:
 - `elevatorsim.py`
 - This is a slightly modified program
- Run the program
 - Try a value that is less than 13
 - What is the result?
 - Run the program again with a value that is greater than 13
 - What is the result?
- What happens if you enter 13?

Our First Example (2)

- Revised Problem Statement (1):
 - Check the input entered by the user:
 - If the input is 13, set the value to 14 and print a message
 - Modify the elevatorsim program to test the input

The relational operator for equal is “==”

- Modified Problem Statement (2)
 - In some countries the number 14 is considered unlucky.
 - What is the revised algorithm?
 - Modify the elevatorsim program to “skip” both the 13th and 14th floor

Compound Statements

- Some constructs in Python are **compound statements**.
- **compound statements** span multiple lines and consist of a *header* and a statement block

The if statement is an example of a compound statement

- Compound statements require a colon “:” at the end of the header.
- The statement block is a group of one or more statements, *all indented to the same column*
- The statement block ***starts on the line after the header*** and ***ends at the first statement indented less than the first statement in the block***

***If you use PyCharm; PyCharm properly indents the statement block.
at the end of the block enter a blank line and wing will shift back to
the first column in the current block***

Compound Statements

- Statement blocks can be nested inside other types of blocks (we will learn about more blocks later)
- Statement blocks signal that one or more statements are part of a given compound statement
- In the case of the if construct the statement block specifies:
 - The instructions that are executed if the condition is true
 - Or skipped if the condition is false

Statement blocks are visual cues that allow you to follow the logic and flow of a program

Tips on Indenting Blocks

- Let PyCharm do the indenting for you...

```
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
```

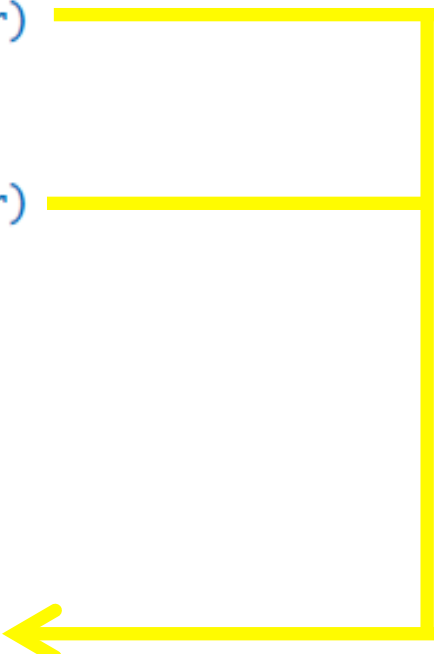
This is referred to as “block structured” code. Indenting consistently is not only syntactically required in Python, it also makes code much easier to follow.

A Common Error

- Avoid duplication in branches
- If the same code is duplicated in each branch then move it out of the **if** statement.

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


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



The Conditional Operator

- A “shortcut” you may find in existing code
 - It is not used in this book
 - The shortcut notation ***can*** be used anywhere that a value is expected

True branch Condition False branch



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

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

Complexity is BAD....

This “shortcut” is difficult to read and a poor programming practice

Relational Operators

- Every **if** statement has a condition
 - Usually compares two values with an operator

```
if floor > 13 :  
    ..  
if floor >=  
    13 : ..  
if floor <  
    13 : ..  
if floor <= 13 :  
    ..  
if floor ==  
    13 : ..
```

Table 1 Relational Operators

Python	Math Notation	Description
>	>	Greater than
>=	\geq	Greater than or equal
<	<	Less than
<=	\leq	Less than or equal
==	=	Equal
!=	\neq	Not equal

Assignment vs. Equality Testing

- Assignment: *makes* something true.

```
floor = 13
```

- Equality testing: *checks* if something is true.

```
if floor == 13 :
```

Comparing Strings

- Checking if two strings are equal

```
if name1 == name2 :  
    print("The strings are identical")
```

- Checking if two strings are not equal

```
if name1 != name2 :  
    print("The strings are not identical")
```

Checking for String Equality (1)

- For two strings to be equal, they must be of the same length and contain the same sequence of characters:

name1 = J o h n W a y n e


name2 = J o h n W a y n e

Checking for String Equality (2)

- If any character is different, the two strings will not be equal:


name1 = J o h n W a y n e

name2 = J a n e W a y n e


The sequence “ane”
does not equal “ohn”




name1 = J o h n W a y n e

name2 = J o h n w a y n e


An uppercase “W” is not
equal to lowercase “w”



Relational Operator Examples (1)

Table 2 Relational Operator Examples

Expression	Value	Comment
$3 \leq 4$	True	3 is less than 4; \leq tests for “less than or equal”.
 $3 \leq 4$	Error	The “less than or equal” operator is \leq , not \leq . The “less than” symbol comes first.
$3 > 4$	False	$>$ is the opposite of \leq .
$4 < 4$	False	The left-hand side must be strictly smaller than the right-hand side.
$4 \leq 4$	True	Both sides are equal; \leq tests for “less than or equal”.
$3 == 5 - 2$	True	$==$ tests for equality.
$3 != 5 - 1$	True	$!=$ tests for inequality. It is true that 3 is not $5 - 1$.
 $3 = 6 / 2$	Error	Use $==$ to test for equality.
$1.0 / 3.0 == 0.33333333$	False	Although the values are very close to one another, they are not exactly equal. See Common Error 3.2 on page 101.
 $"10" > 5$	Error	You cannot compare a string to a number.

Relational Operator Examples (2)

Table 2 Relational Operator Examples

 <code>3 = 6 / 2</code>	Error	Use <code>==</code> to test for equality.
<code>1.0 / 3.0 == 0.333333333</code>	False	Although the values are very close to one another, they are not exactly equal. See Common Error 3.2 on page 101.
 <code>"10" > 5</code>	Error	You cannot compare a string to a number.

Another Example

- Open the file:
 - `compare.py`
- Run the program
 - What are the results?