Code Academy

Chapter Three

PART ONE: DECISIONS, RELATIONAL OPERATORS

Part1: Decisions and Relational Operators

Goals

- To implement decisions using the if statement
- To compare integers, floating-point numbers, and Strings

Contents

- The `if` Statement
- Relational Operators

The `if` Statement

Code Academy

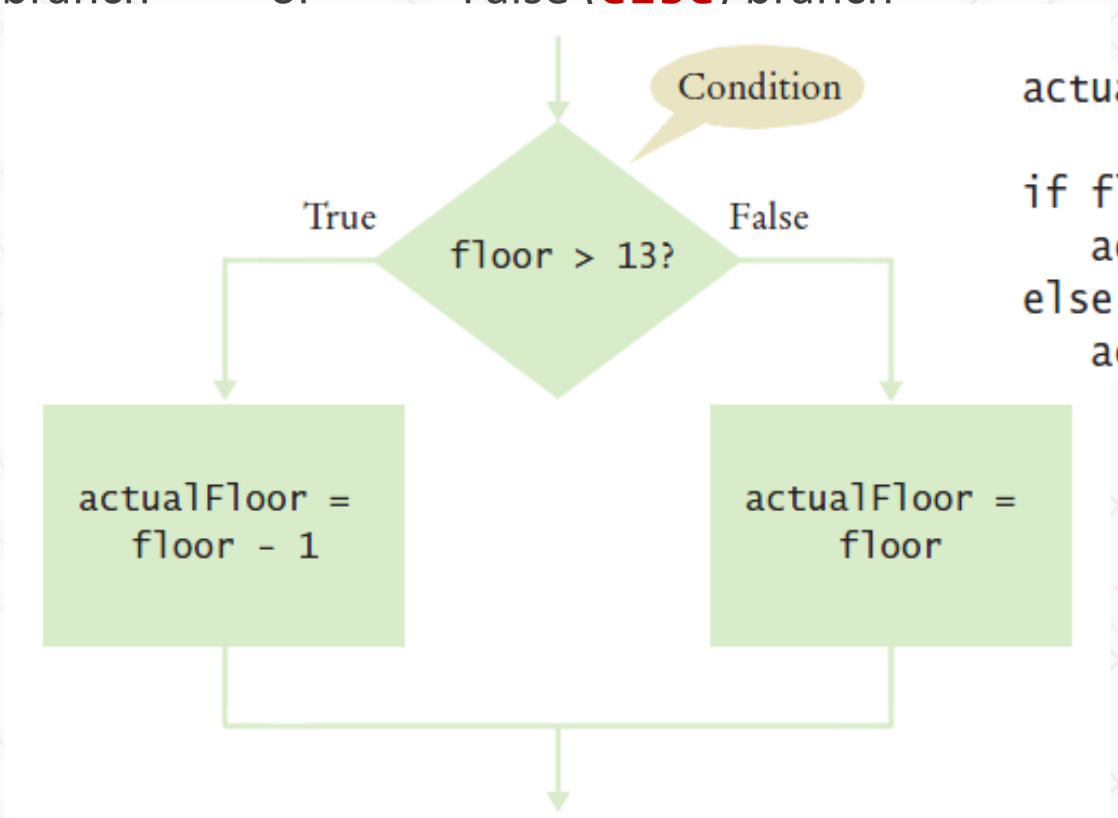
- A computer program often needs to make decisions based on input, or circumstances
- When a condition is fulfilled one set of statements is executed . Otherwise, another set of statements is executed.
- For example, buildings often ‘skip’ the 13th floor, and elevators should too. (In some countries number 13 is considered unlucky)
  - 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`

  Note: If and else statement together are called a compound statement.

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

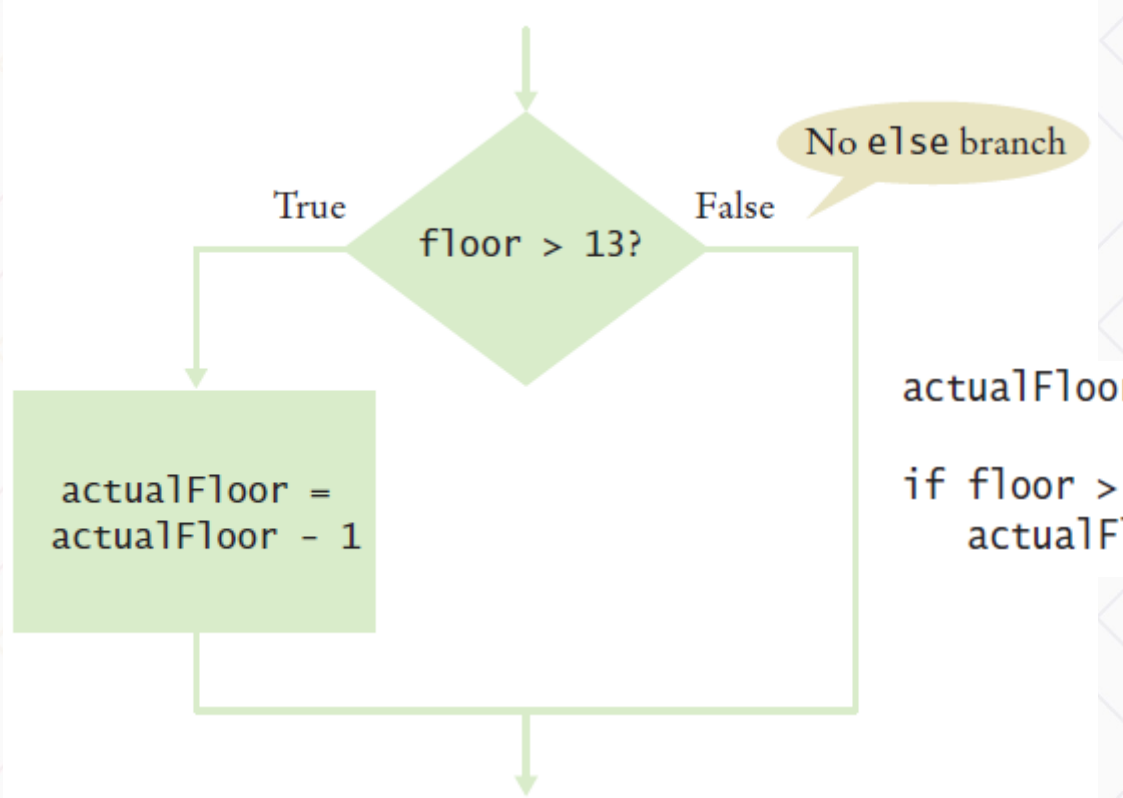



```
actualFloor = 0

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

Flowchart with only a True Branch

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




```
actualFloor = floor

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

Syntax 3.1: The `if` Statement

Syntax

```
if condition :
   statements
```

```
if condition :
   statements1
else :
   statements2
```

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.

Code Academy

Our First Example: 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
```

- **Run the program again:**
  - What happens if you enter 13?
  - What happens if you enter a value less than 13?

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***

Compound Statements

Code Academy

- 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

Code Academy

```
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)
```

Code Academy

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


```
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
>=	≥	Greater than or equal
<	<	Less than
<=	≤	Less than or equal
==	=	Equal
!=	≠	Not equal

Assignment vs. Equality Testing

- Assignment: *makes* something true. It saves a value into a variable.

```
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")
```

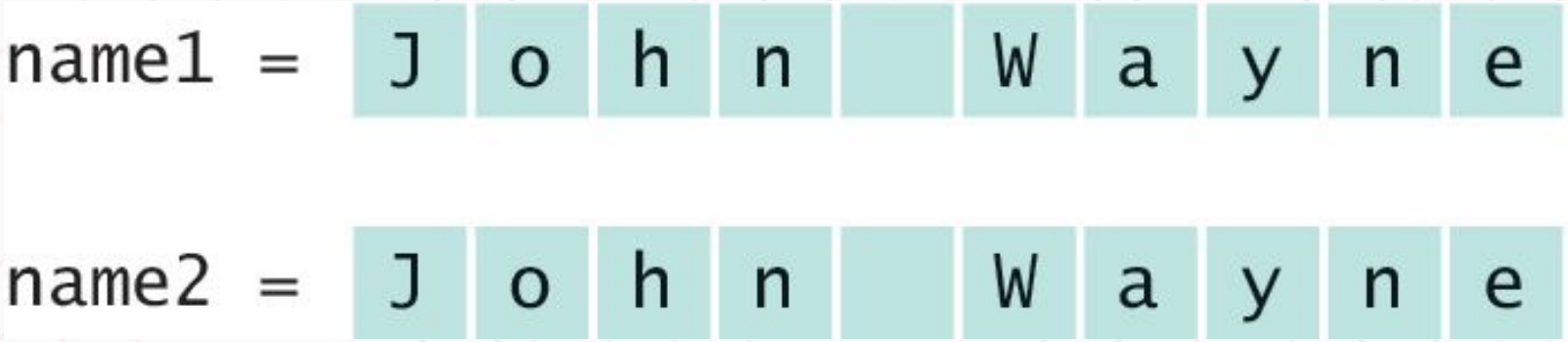
- Checking if two strings are not equal

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

Code Academy

Checking for String Equality (1)

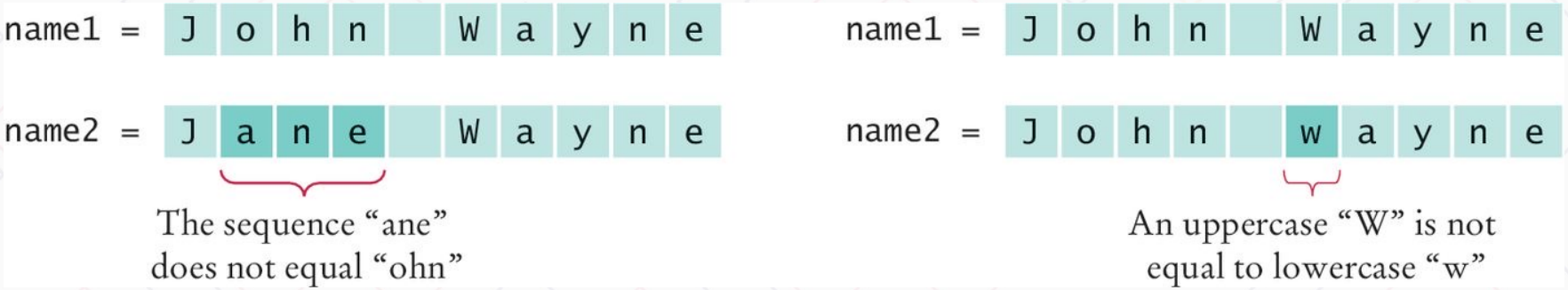
- For two strings to be equal, they must be of the same length and contain the same sequence of characters:



Checking for String Equality (2)

Code Academy

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



Relational Operator Examples

Table 2 Relational Operator Examples

Expression	Value	Comment
`3 <= 4`	`True`	3 is less than 4; <= tests for "less than or equal".
`3 =< 4`	**Error**	The "less than or equal" operator is <=, not =<. The "less than" symbol comes first.
`3 > 4`	`False`	> is the opposite of <=.
`4 < 4`	`False`	The left-hand side must be strictly smaller than the right-hand side.
`4 <= 4`	`True`	Both sides are equal; <= 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 == 0.333333333`	`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.

Code Academy

What is the output of compare.py program?

```
1  ##
2  # compare.py
3  # This program demonstrates comparisons of numbers and strings.
4  #
5
6  from math import sqrt
7
8  # Comparing integers
9  m = 2
10 n = 4
11
12 if m * m == n :
13    print("2 times 2 is four.")
14
15 # Comparing floating-point numbers.
16 x = sqrt(2)
17 y = 2.0
18
19 if x * x == y :
20    print("sqrt(2) times sqrt(2) is 2")
21 else :
22    print("sqrt(2) times sqrt(2) is not two but %.18f" % (x * x))
23
24 EPSILON = 1E-14
25 if abs(x * x - y) < EPSILON :
26    print("sqrt(2) times sqrt(2) is approximately 2")
```

→*Program continues next slide….*

What is the output of the compare.py program? (continued)

```
27
28 # Comparing strings
29 s = "120"
30 t = "20"
31
32 if s == t :
33    comparison = "is the same as"
34 else :
35    comparison = "is not the same as"
36
37 print("The string '%s' %s the string '%s'." % (s, comparison, t))
38
39 u = "1" + t
40 if s != u :
41    comparison = "not "
42 else :
43    comparison = ""
44
45 print("The strings '%s' and '%s' are %sidentical." % (s, u, comparison))
46
```

What is the output the program: compare.py? (continued)

```
In [10]: runfile('D:/python/src/compare.py', wdir='D:/python/src')
2 times 2 is four.
sqrt(2) times sqrt(2) is not two but 2.0000000000000044
sqrt(2) times sqrt(2) is approximately 2
The string '120' is not the same as the string '20'.
The strings '120' and '120' are identical.

In [11]:
```

Common Error (Floating Point)

- Floating-point numbers have only a limited precision, and calculations can introduce round-off errors.
- You must take these inevitable round-offs into account when comparing floating point numbers.

Common Error (Floating Point, 2)

- For example, the following code multiplies the square root of 2 by itself.
- Ideally, we expect to get the answer 2:

```
r = math.sqrt(2.0)
if r * r == 2.0 :
   print("sqrt(2.0) squared is 2.0")
else :
   print("sqrt(2.0) squared is not 2.0 but", r * r)
```

```
Output:
sqrt(2.0) squared is not 2.0 but 2.0000000000000004
```

Code Academy

The Use of EPSILON

- Use a very small value to compare the difference to determine if floating-point values are *'close enough'*
  - The magnitude of their difference should be less than some threshold
  - Mathematically, we would write that x and y are close enough if:

$$|x - y| < \varepsilon$$

```
EPSILON = 1E-14
r = math.sqrt(2.0)
if abs(r * r - 2.0) < EPSILON :
   print("sqrt(2.0) squared is approximately 2.0")
```

Lexicographical Order

Code Academy

- To compare Strings in ‘dictionary’ like order:

  string1 < string2

- Notes
  - All UPPERCASE letters come before lowercase
  - ‘space’ comes before all other printable characters
  - Digits (0-9) come before all letters
  - See Appendix A for the Basic Latin (ASCII) Subset of Unicode

Operator Precedence

Code Academy

- The arithmetic operators have higher precedence than comparison operators
  - ***Calculations are done first before the comparison***
  - Normally your calculations are on the 'right side' of the comparison or assignment operator.

```
actualFloor = floor + 1
```

Calculations

```
if floor > height + 1 :
```

→ But it is possible to have calculations on both sides of the comparison operator

```
if x + y == y + x:
        print("Yes")
else:
        print("No")
```

Implementing an `if` Statement (1)

1) Decide on a branching condition

```
original price < 128?
```

2) Write pseudocode for the true branch

```
discounted price = 0.92 x original price
```

3) Write pseudocode for the false branch

```
discounted price = 0.84 x original price
```

Implementing an `if` Statement (2)

4) Double-check relational operators

- Test values below, at, and above the comparison (127, 128, 129)

5) Remove duplication

discounted price = ____ x original price

6) Test both branches

discounted price = 0.92 x 100 = 92

discounted price = 0.84 x 200 = 168

A Third Example

Code Academy

- The university bookstore has a Kilobyte Day sale every October 24 (10.24), giving an 8 percent discount on all computer accessory purchases if the price is less than $128, and a 16 percent discount if the price is at least $128. Compute the price after discount

```
if originalPrice < 128 :
   discountRate = 0.92
else :
   discountRate = 0.84
discountedPrice = discountRate * originalPrice
```

The Sale Example: sale.py

Code Academy

```
1  ##
2  # sale.py
3  # Compute the discount for a given purchase.
4  #
5
6  # Obtain the original price.
7  originalPrice = float(input("Original price before discount: "))
8
9  # Determine the discount rate.
10 if originalPrice < 128 :
11    discountRate = 0.92
12 else :
13    discountRate = 0.84
14
15 # Compute and print the discount.
16 discountedPrice = discountRate * originalPrice
17 print("Discounted price: %.2f" % discountedPrice)
```

- Run the program several time using different values
  - Use values less than 128
  - Use values greater that 128
  - Enter 128
- What results do you get?

Summary: `if` Statement

- The if statement allows a program to carry out different actions depending on the nature of the data to be processed.
- Relational operators (`< <= > >= == !=`) are used to compare numbers and `Strings`.
- `Strings` are compared in lexicographic order.