Code Academy

Chapter Three

PART THREE: BOOLEAN VARIABLES AND OPERATORS, ANALYZING STRINGS, INPUT VALIDATION (SEC3.6-3.9)

Chapter3 Part2 & Part3: Goals & Contents

Goals

Part3:

- To write statements using the Boolean data type
- To develop strategies for testing your programs
- To validate user input

Contents

Part3:

- Problem Solving: Test Cases
- Boolean Variables and Operators
- Analyzing Strings
- Application: Input Validation

Boolean Variables

- Boolean Variables
  - A Boolean variable is often called a flag because it can either be up (`true`) or down (`false`)
  - boolean is a Python data type

```
failed = True
```

  - Boolean variables can either be `True` or `False`
- There are two Boolean Operators: `and, or`
  - They are used to combine multiple conditions

Combined Conditions: and

- Combining two conditions is often used in range checking
  - Is a value between two other values?
- Both sides of the *and* must be true for the result to be true

```
if temp > 0 and temp < 100 :
    print("Liquid")
```

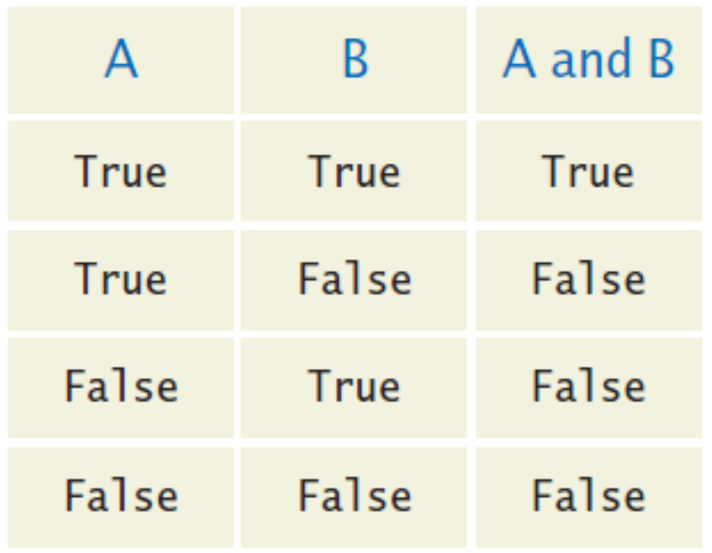
Examples

temp = 3

If 3 > 0 and 3 < 100 ➔ if True and True ➔ True

temp = -7

If -7 > 0 and -7 < 100 ➔ if False and True ➔ False

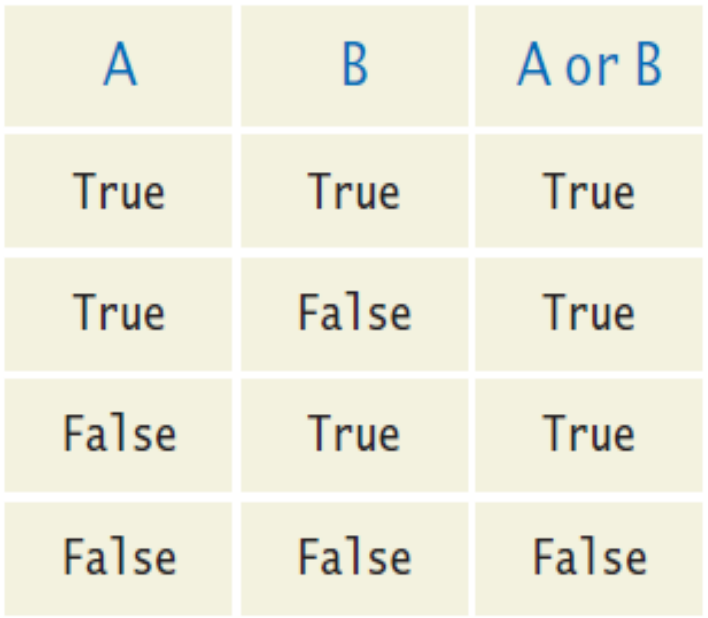


A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

Combined Conditions: or

- We use or if only one of two conditions need to be true
  - Use a compound conditional with an or:

```
if temp <= 0 or temp >= 100
   :
   print("Not liquid")
```



A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Examples

temp = -5

If -5 <= 0 or -5 >= 100 → if True or False → True

temp = 7

If 7 <= 0 or 7 >= 100 → if False or False → False

- If either condition is true
  - The result is true

The *not* operator: not

- If you need to invert a boolean variable or comparison, precede it with not.

Example: attending = True, grade = 70

```
if not attending or grade < 60 :
   print("Drop?")
```

If not (True) or 70 < 60

If False or False ➔ False

```
if attending and not(grade < 60):
   print("Stay")
```

If True and not (70 < 60)

If True and True ➔ True

- If you are using not, try to use simpler logic:

```
if attending and grade >= 60 :
   print("Stay")
```

A	not A
True	False
False	True

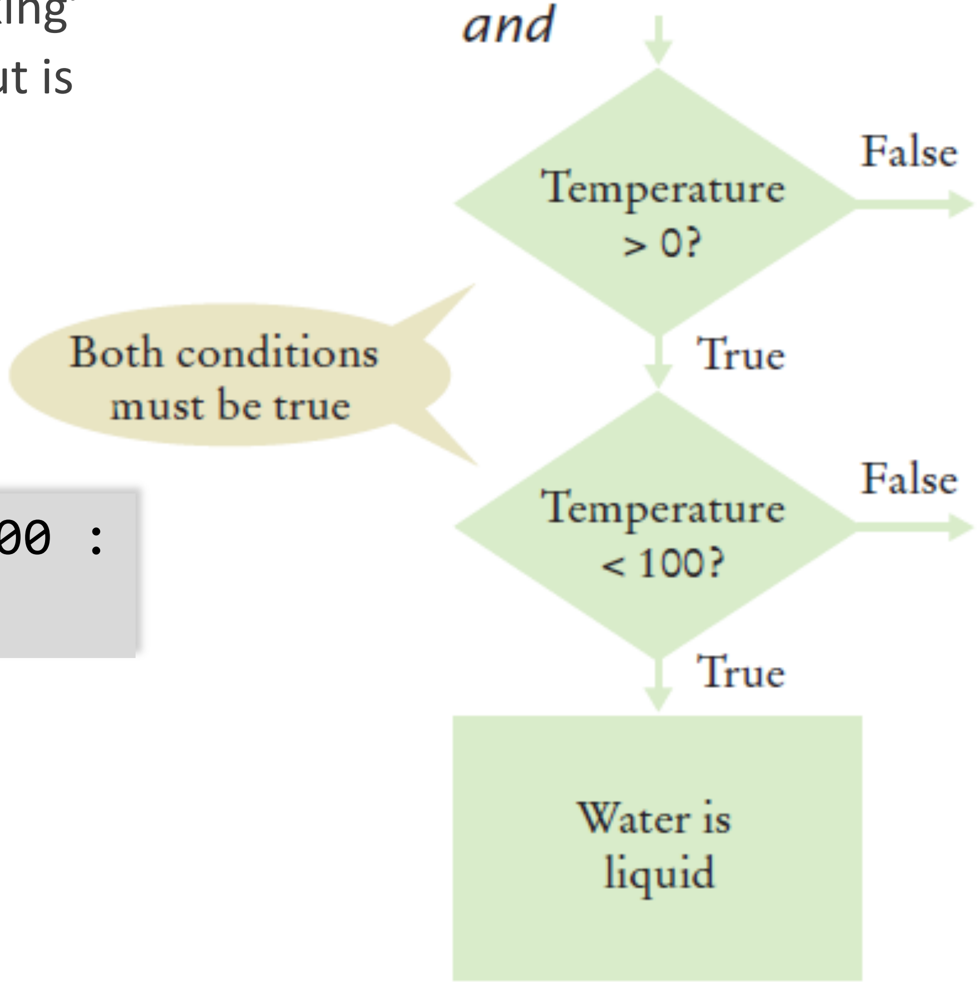
The *not* operator: inequality !

- A slightly different operator is used for the not when checking for inequality rather than negation.

- Example inequality:
  - The password that the user entered is not equal to the password on file.
  - `if userPassword != filePassword :`

Flowchart for *“and” Combination*

- This is often called ‘range checking’
  - Used to validate that the input is between two values

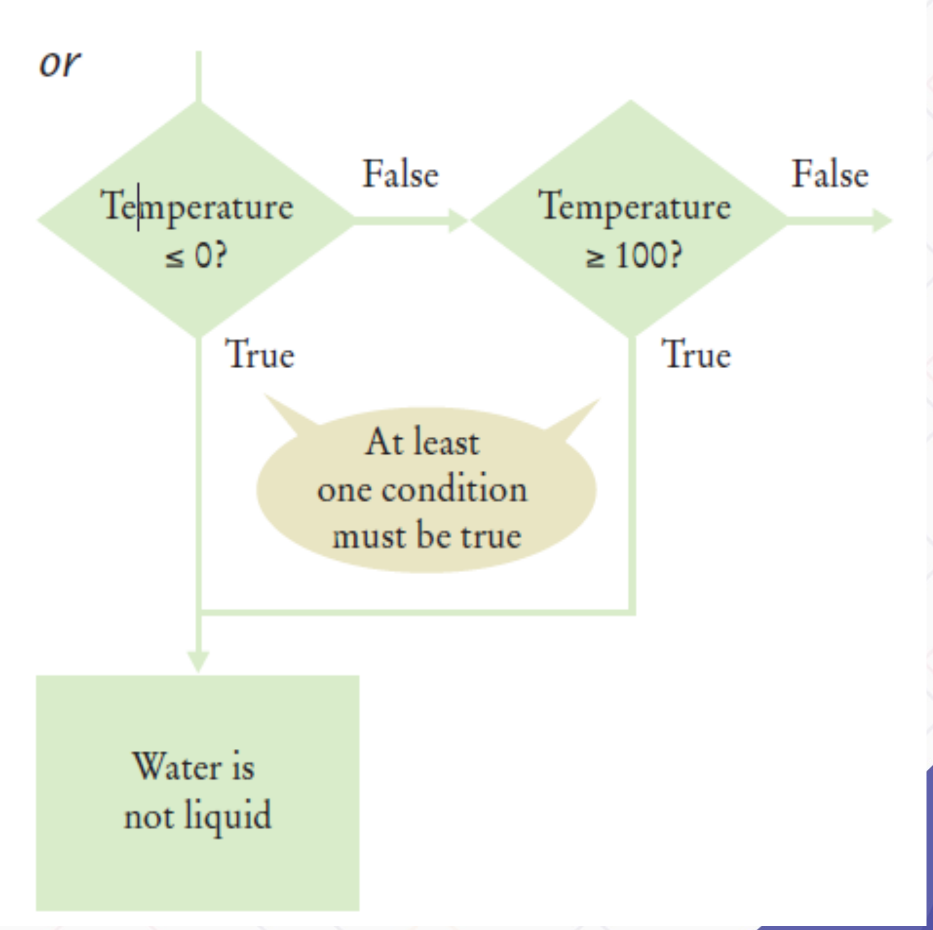


```
if temp > 0 and temp < 100 :
   print("Liquid")
```

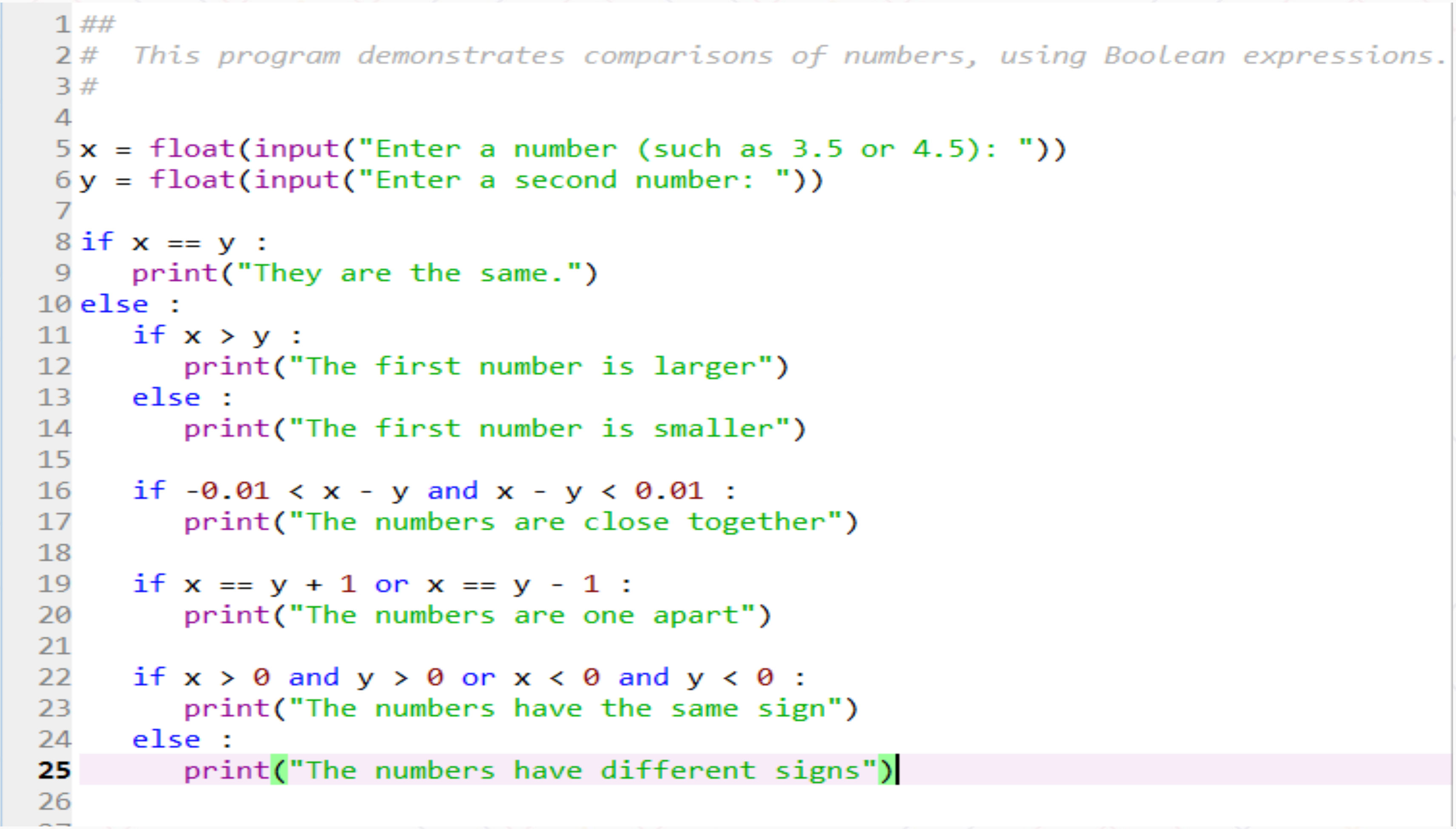
Flowchart for*“or” Combination*

- Another form of ‘range checking’
  - Checks if value is outside a range

```
if temp <= 0 or temp >= 100 :
    print("Not Liquid")
```



Comparison Example: Compare2.py



```
##
#  This program demonstrates comparisons of numbers, using Boolean expressions.
#

x = float(input("Enter a number (such as 3.5 or 4.5): "))
y = float(input("Enter a second number: "))

if x == y :
   print("They are the same.")
else :
   if x > y :
      print("The first number is larger")
   else :
      print("The first number is smaller")

   if -0.01 < x - y and x - y < 0.01 :
      print("The numbers are close together")

   if x == y + 1 or x == y - 1 :
      print("The numbers are one apart")

   if x > 0 and y > 0 or x < 0 and y < 0 :
      print("The numbers have the same sign")
   else :
      print("The numbers have different signs")

```

- Run the program with several inputs

Boolean Operator Examples

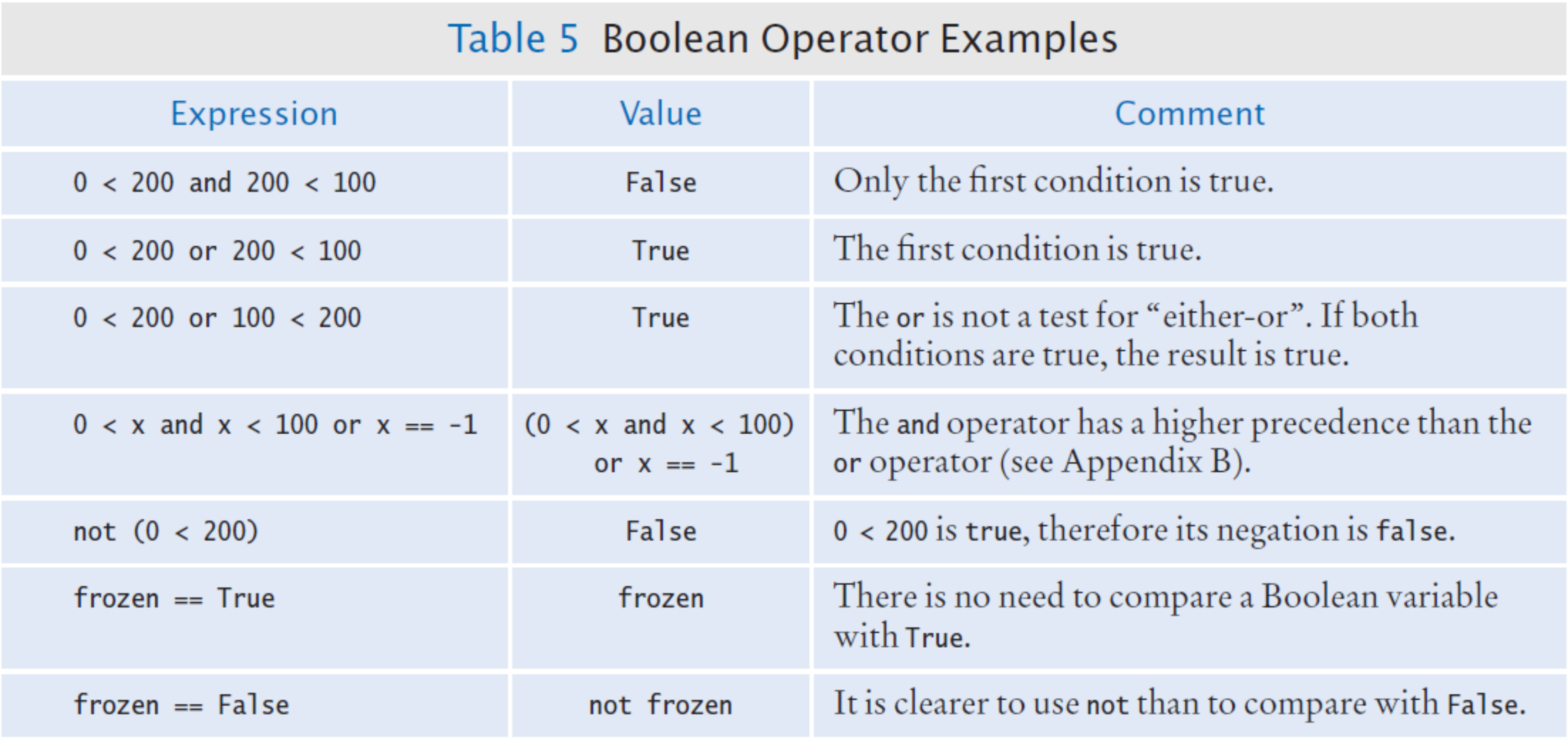


Table 5 Boolean Operator Examples

Expression	Value	Comment
0 < 200 and 200 < 100	False	Only the first condition is true.
0 < 200 or 200 < 100	True	The first condition is true.
0 < 200 or 100 < 200	True	The or is not a test for "either-or". If both conditions are true, the result is true.
0 < x and x < 100 or x == -1	(0 < x and x < 100) or x == -1	The and operator has a higher precedence than the or operator (see Appendix B).
not (0 < 200)	False	0 < 200 is true, therefore its negation is false.
frozen == True	frozen	There is no need to compare a Boolean variable with True.
frozen == False	not frozen	It is clearer to use not than to compare with False.

Common Errors with Boolean Conditions

Confusing and and or Conditions

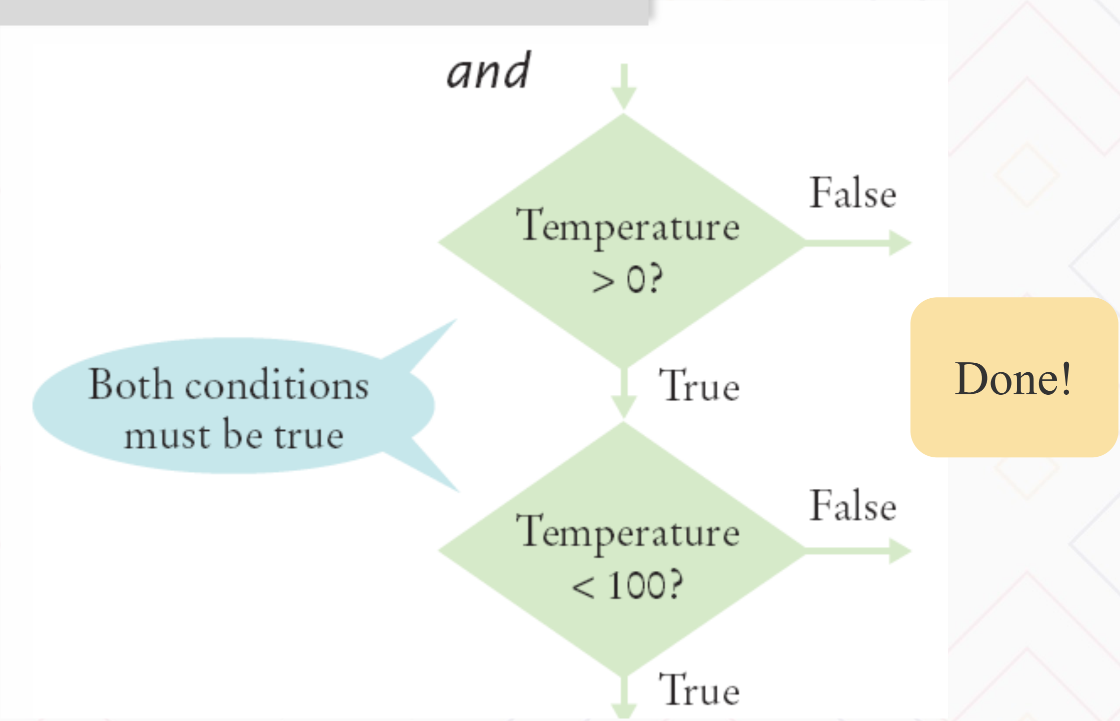
- It is a surprisingly common error to confuse and and or conditions.
- A value lies between 0 and 100 if it is at least 0 ***and*** at most 100.
- It lies outside that range if it is less than 0 ***or*** greater than 100.

- There is no golden rule; you just have to think carefully.

Short-circuit Evaluation: and

- Combined conditions are evaluated from left to right
  - If the left half of an *and* condition is false, why look further?

```
if temp > 0 and temp < 100 :
   print("Liquid")
```



and

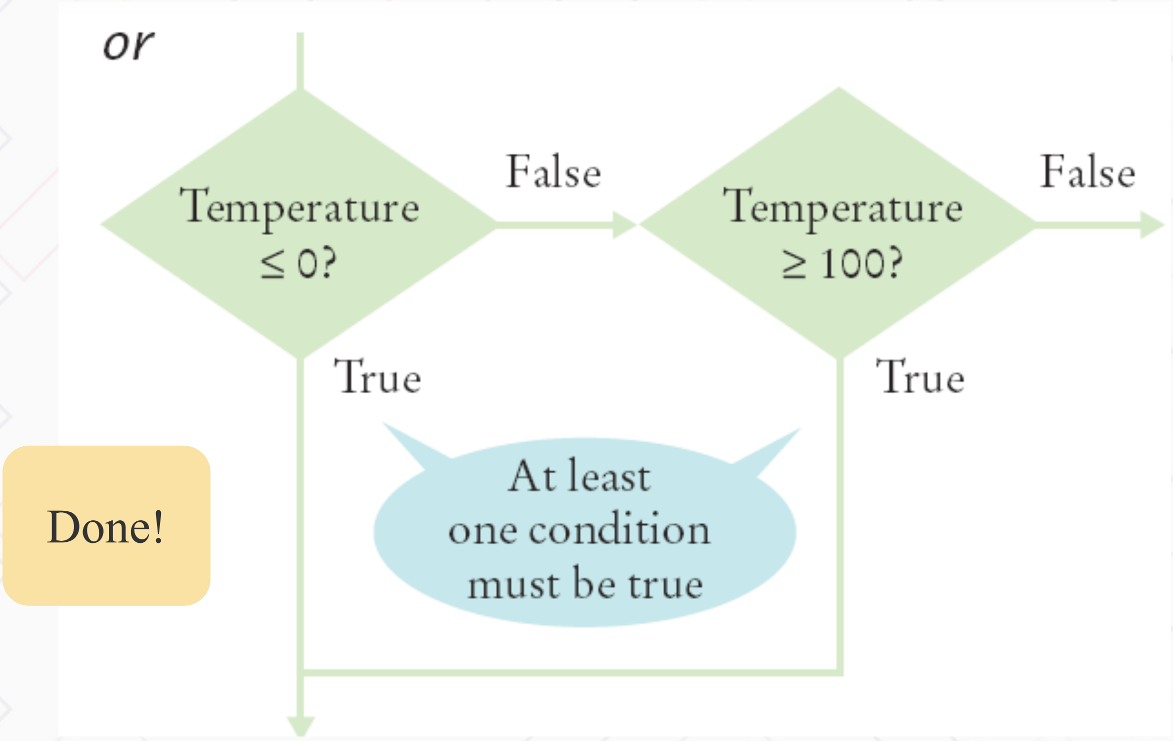
Both conditions must be true

Done!

Short-circuit evaluation: or

- If the left half of the *or* is true, why look further?

```
if temp <= 0 or temp >= 100 :
   print("Not Liquid")
```



De Morgan’s law

- De Morgan’s law tells you how to negate “and” and “or” conditions:
  - not(A and B) is the same as (notA) or (notB)
  - not(A or B) is the same as (notA) and (notB)

- Example: Shipping is higher to AK and HI

```
if (country != "USA"
   and state != "AK"
   and state != "HI") :
   shippingCharge = 20.00
```

```
if not(country=="USA"
 or state=="AK"
 or state=="HI") :
  shippingCharge = 20.00
```

- To simplify conditions with negations of *and* or *or* expressions, it’s a good idea to apply De Morgan’s law to move the negations to the innermost level.

Analyzing Strings – The in Operator

- Sometimes it's necessary to analyze or ask certain questions about a particular string.
  - Sometimes it is necessary to determine if a string contains a given substring. That is, one string contains an exact match of another string.
  - Given this code segment,

    ```
    name = "John Wayne"
    ```

  - the expression

    ```
    "Way" in name
    ```

  - yields `True` because the substring `"Way"` occurs within the string stored in variable name.
  - The **`not in`** operator is the inverse on the in operator

Substring: Suffixes

- Suppose you are given the name of a file and need to ensure that it has the correct extension

```
if filename.endswith(".html") :
   print("This is an HTML file.")
```

- The `endswith()` string method is applied to the string stored in filename and returns `True` if the string ends with the substring ".html" and `False` otherwise.

Operations for Testing Substrings

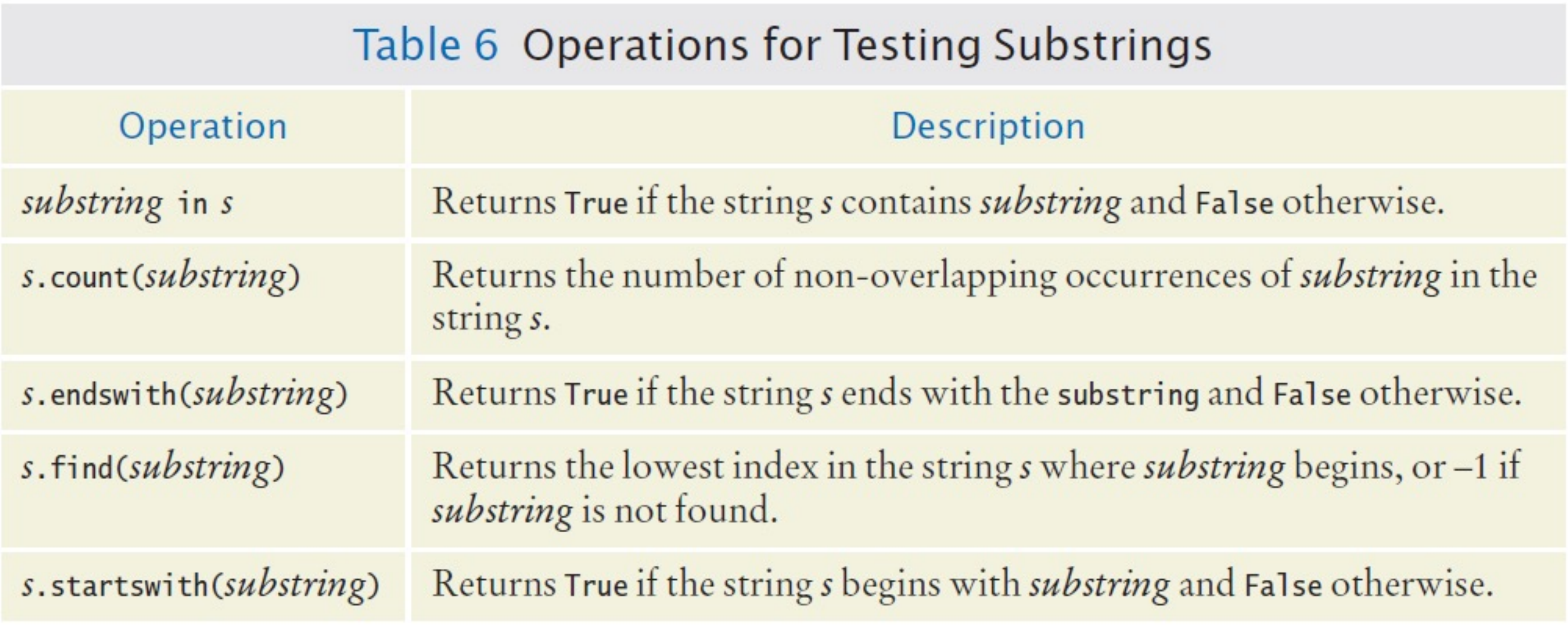


Table 6 Operations for Testing Substrings

Operation	Description
substring `in` *s*	Returns `True` if the string *s* contains *substring* and `False` otherwise.
s`.count(`*substring*`)`	Returns the number of non-overlapping occurrences of *substring* in the string *s*.
s`.endswith(`*substring*`)`	Returns `True` if the string *s* ends with the `substring` and `False` otherwise.
s`.find(`*substring*`)`	Returns the lowest index in the string *s* where *substring* begins, or –1 if *substring* is not found.
s`.startswith(`*substring*`)`	Returns `True` if the string *s* begins with *substring* and `False` otherwise.

Methods: Testing String Characteristics (1)

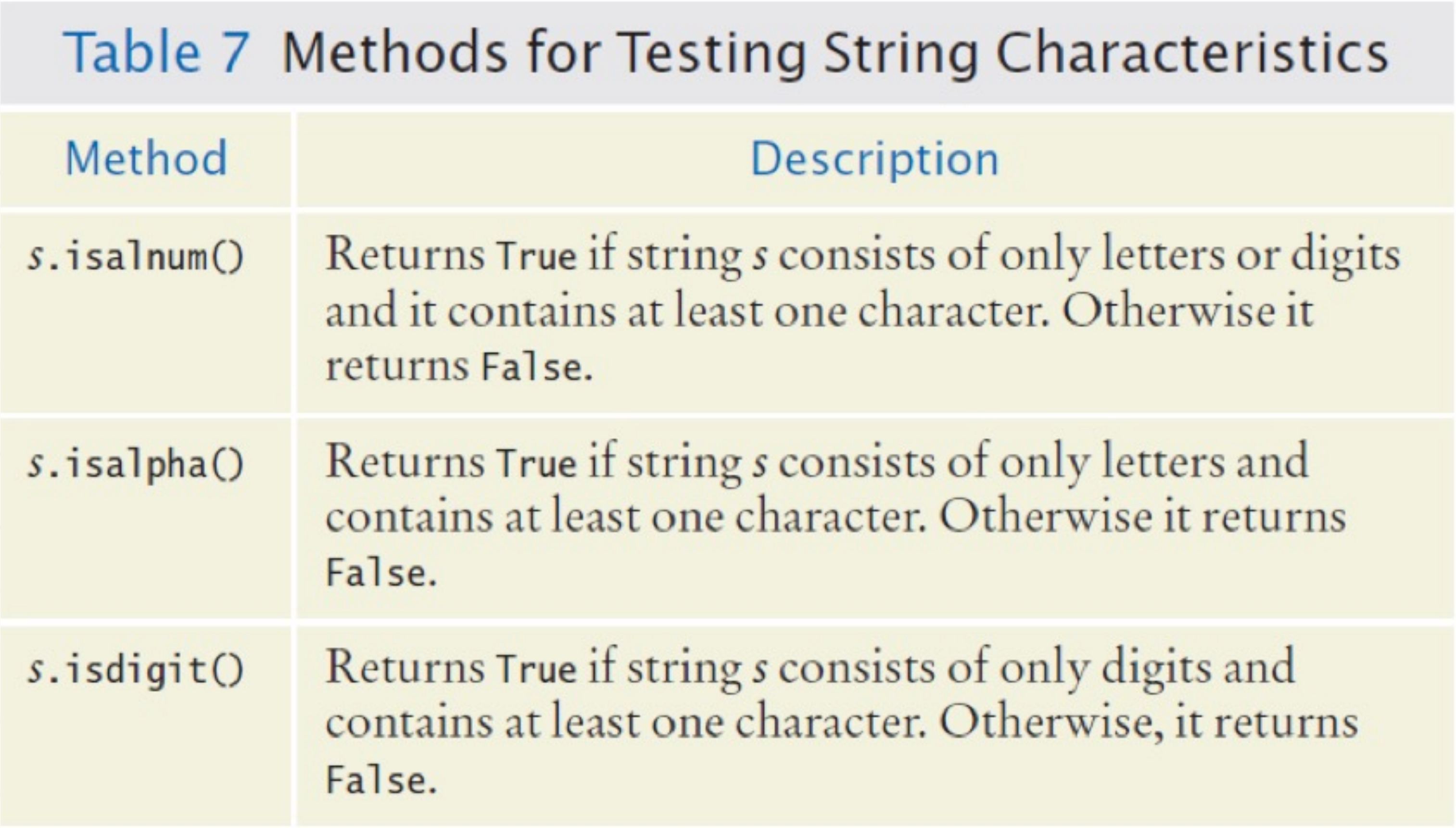


Table 7 Methods for Testing String Characteristics

Method	Description
`s.isalnum()`	Returns `True` if string *s* consists of only letters or digits and it contains at least one character. Otherwise it returns `False`.
`s.isalpha()`	Returns `True` if string *s* consists of only letters and contains at least one character. Otherwise it returns `False`.
`s.isdigit()`	Returns `True` if string *s* consists of only digits and contains at least one character. Otherwise, it returns `False`.

Methods for Testing String Characteristics (2)

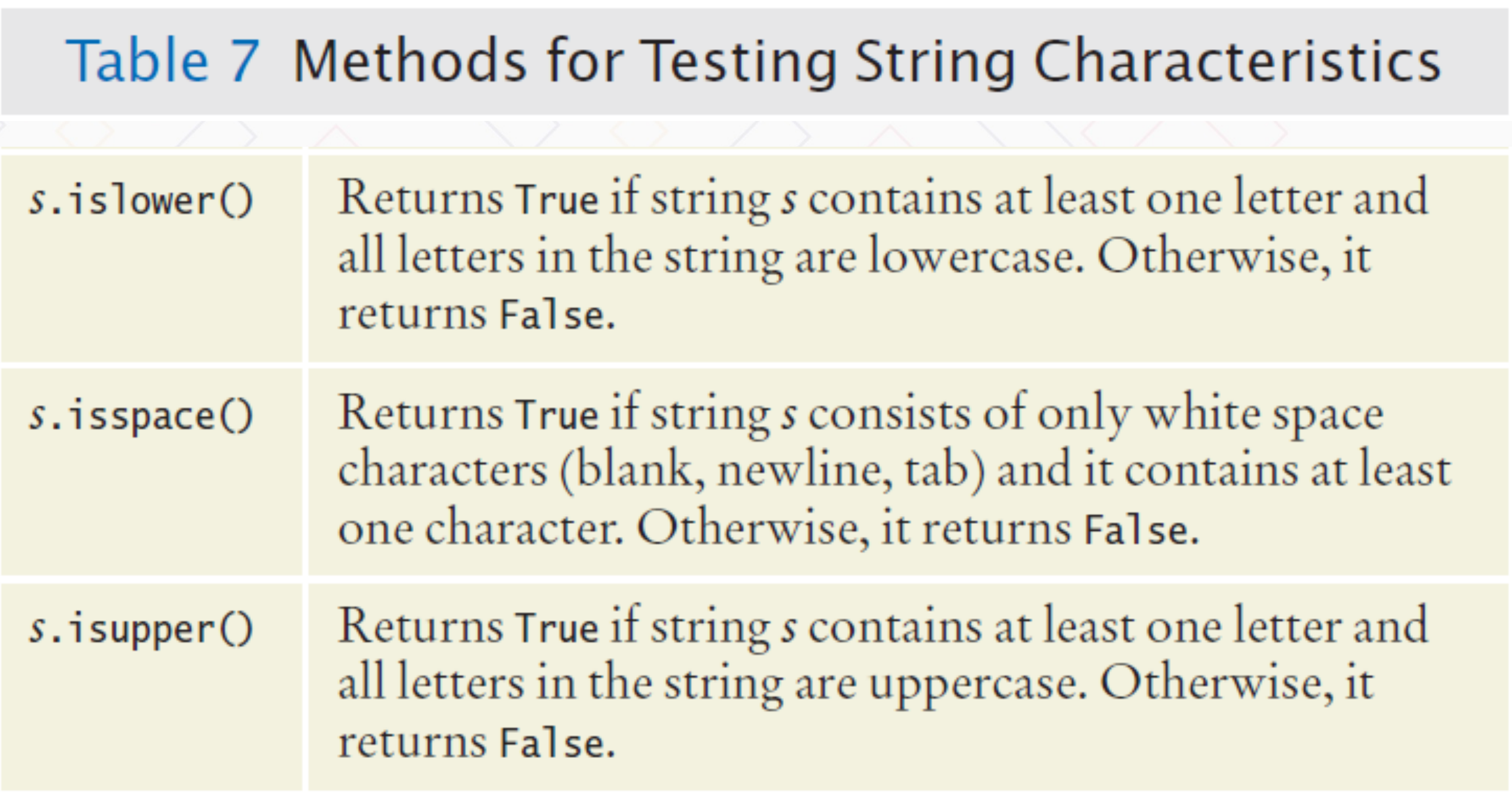


Table 7 Methods for Testing String Characteristics

s.islower()	Returns True if string *s* contains at least one letter and all letters in the string are lowercase. Otherwise, it returns False.
s.isspace()	Returns True if string *s* consists of only white space characters (blank, newline, tab) and it contains at least one character. Otherwise, it returns False.
s.isupper()	Returns True if string *s* contains at least one letter and all letters in the string are uppercase. Otherwise, it returns False.

Comparing and Analyzing Strings (1)

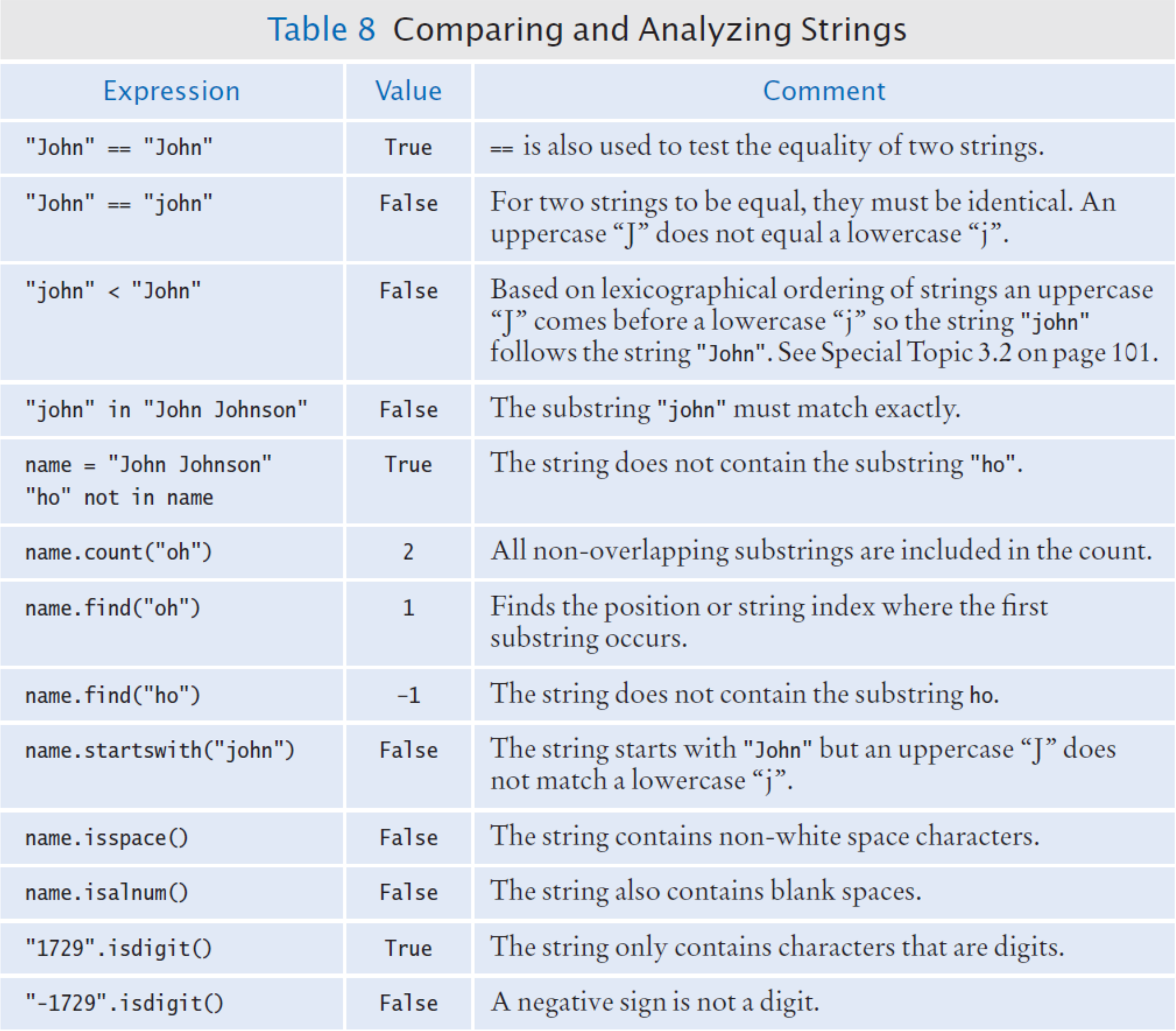


Table 8 Comparing and Analyzing Strings

Expression	Value	Comment
`"John" == "John"`	`True`	`==` is also used to test the equality of two strings.
`"John" == "john"`	`False`	For two strings to be equal, they must be identical. An uppercase “J” does not equal a lowercase “j”.
`"john" < "John"`	`False`	Based on lexicographical ordering of strings an uppercase “J” comes before a lowercase “j” so the string `"john"` follows the string `"John"`. See Special Topic 3.2 on page 101.
`"john" in "John Johnson"`	`False`	The substring `"john"` must match exactly.

`name = "John Johnson"`
`"ho" not in name`
	`True`	The string does not contain the substring `"ho"`.

`name.count("oh")`	`2`	All non-overlapping substrings are included in the count.
`name.find("oh")`	`1`	Finds the position or string index where the first substring occurs.
`name.find("ho")`	`-1`	The string does not contain the substring `ho`.
`name.startswith("john")`	`False`	The string starts with `"John"` but an uppercase “J” does not match a lowercase “j”.
`name.isspace()`	`False`	The string contains non-white space characters.
`name.isalnum()`	`False`	The string also contains blank spaces.
`"1729".isdigit()`	`True`	The string only contains characters that are digits.
`"-1729".isdigit()`	`False`	A negative sign is not a digit.

Comparing and Analyzing Strings (2)

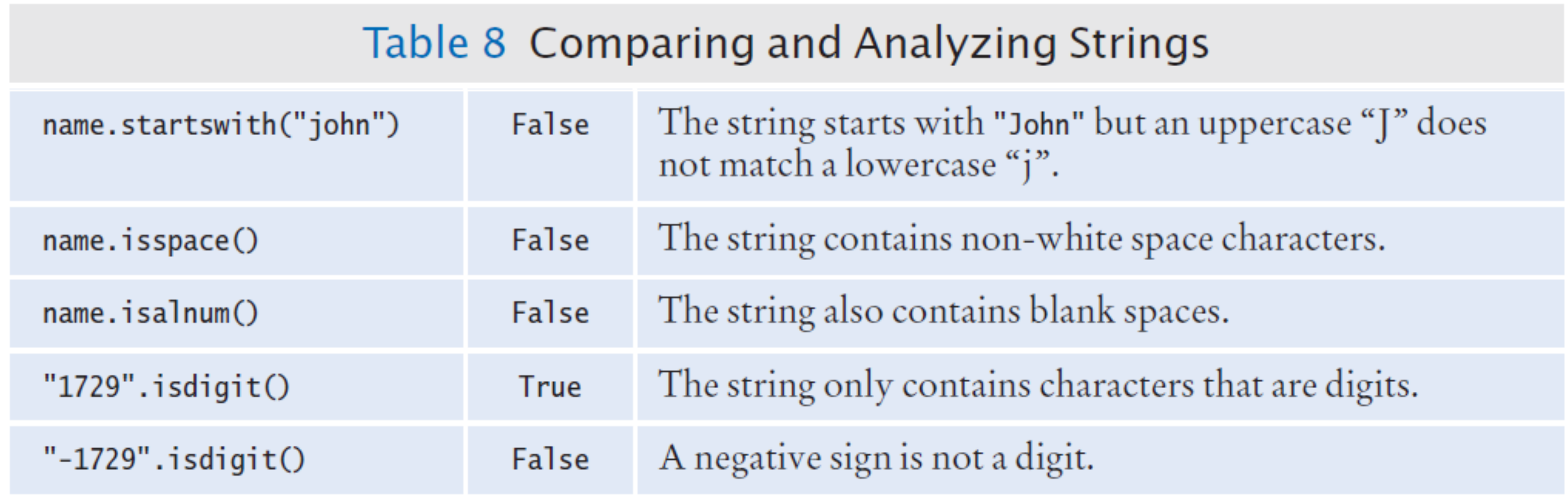
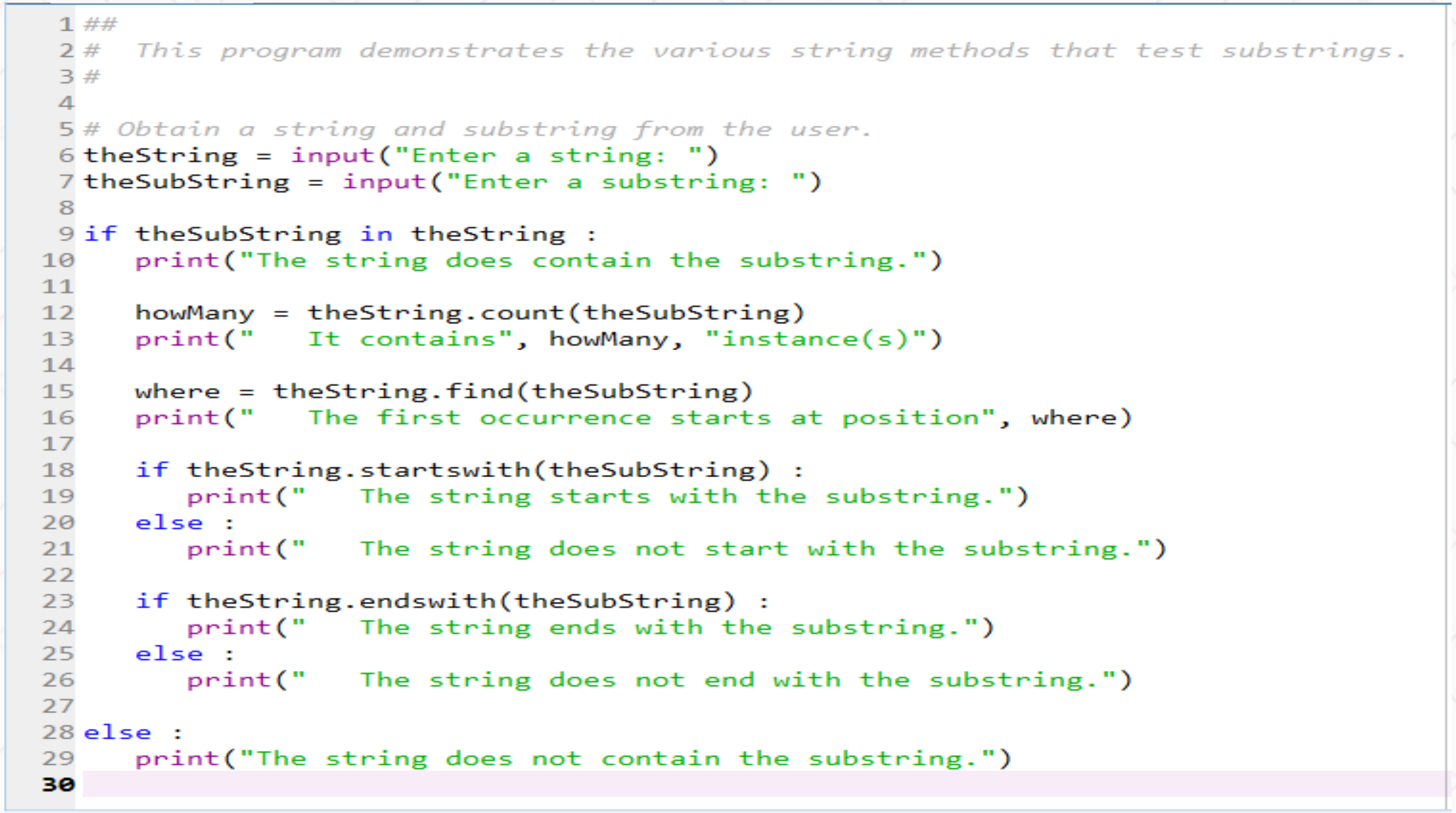


Table 8 Comparing and Analyzing Strings

`name.startswith("john")`	`False`	The string starts with `"John"` but an uppercase “J” does not match a lowercase “j”.
`name.isspace()`	`False`	The string contains non-white space characters.
`name.isalnum()`	`False`	The string also contains blank spaces.
`"1729".isdigit()`	`True`	The string only contains characters that are digits.
`"-1729".isdigit()`	`False`	A negative sign is not a digit.

Code Academy

Substring Example: Substrings.py



```python
##
#   This program demonstrates the various string methods that test substrings.
#

# Obtain a string and substring from the user.
theString = input("Enter a string: ")
theSubString = input("Enter a substring: ")

if theSubString in theString :
    print("The string does contain the substring.")

    howMany = theString.count(theSubString)
    print("   It contains", howMany, "instance(s)")

    where = theString.find(theSubString)
    print("   The first occurrence starts at position", where)

    if theString.startswith(theSubString) :
        print("   The string starts with the substring.")
    else :
        print("   The string does not start with the substring.")

    if theString.endswith(theSubString) :
        print("   The string ends with the substring.")
    else :
        print("   The string does not end with the substring.")

else :
    print("The string does not contain the substring.")

```

- Run the program and test several strings and substrings

Input Validation

- Accepting user input is dangerous
  - Consider the Elevator program:
  - Assume that the elevator panel has buttons labeled 1 through 20 (but not 13).

Input Validation

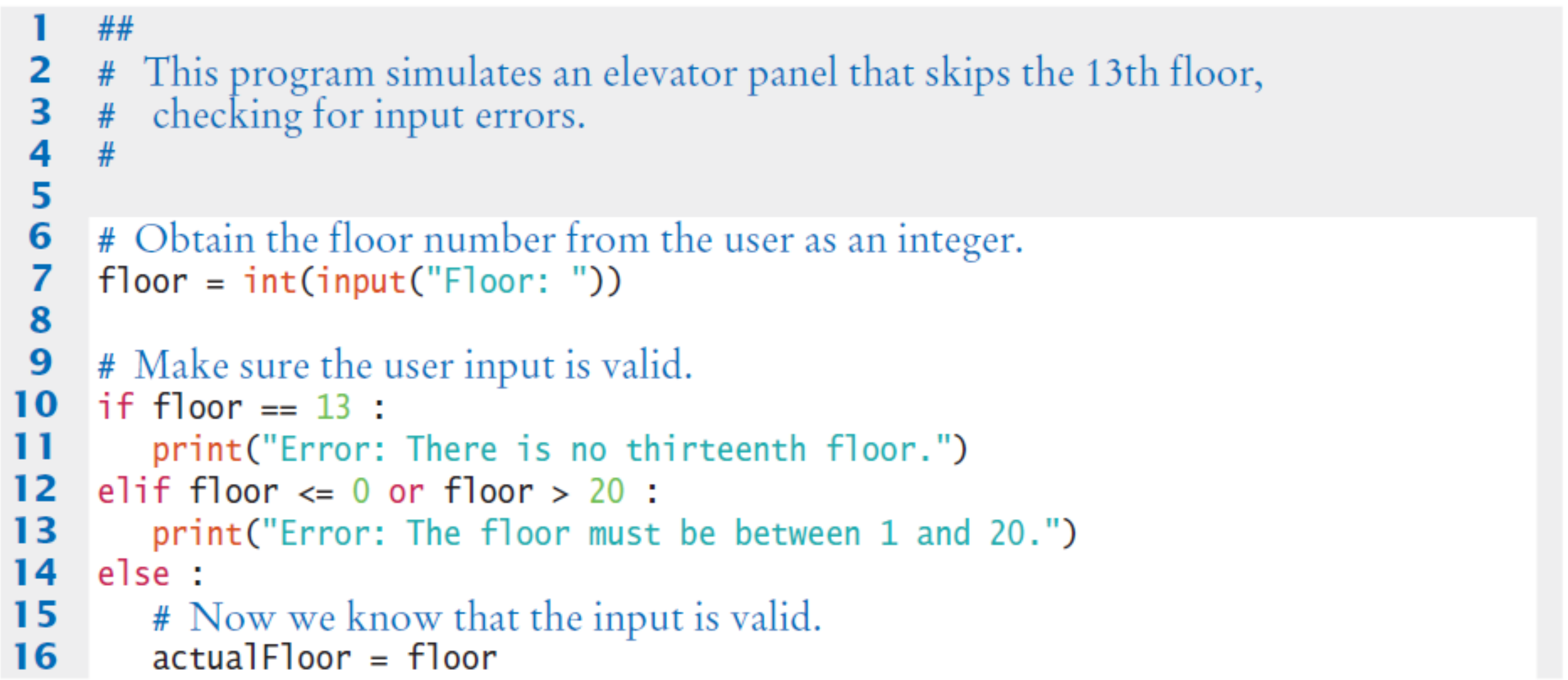
- The following are illegal inputs:
  - The number 13

```
if floor == 13 :
    print("Error: There is no thirteenth floor.")
```

  - Zero or a negative number
  - A number larger than 20

```
if floor <= 0 or floor > 20 :
    print("Error: The floor must be between 1 and 20.")
```

Elevatorsim2.py: Validating input data

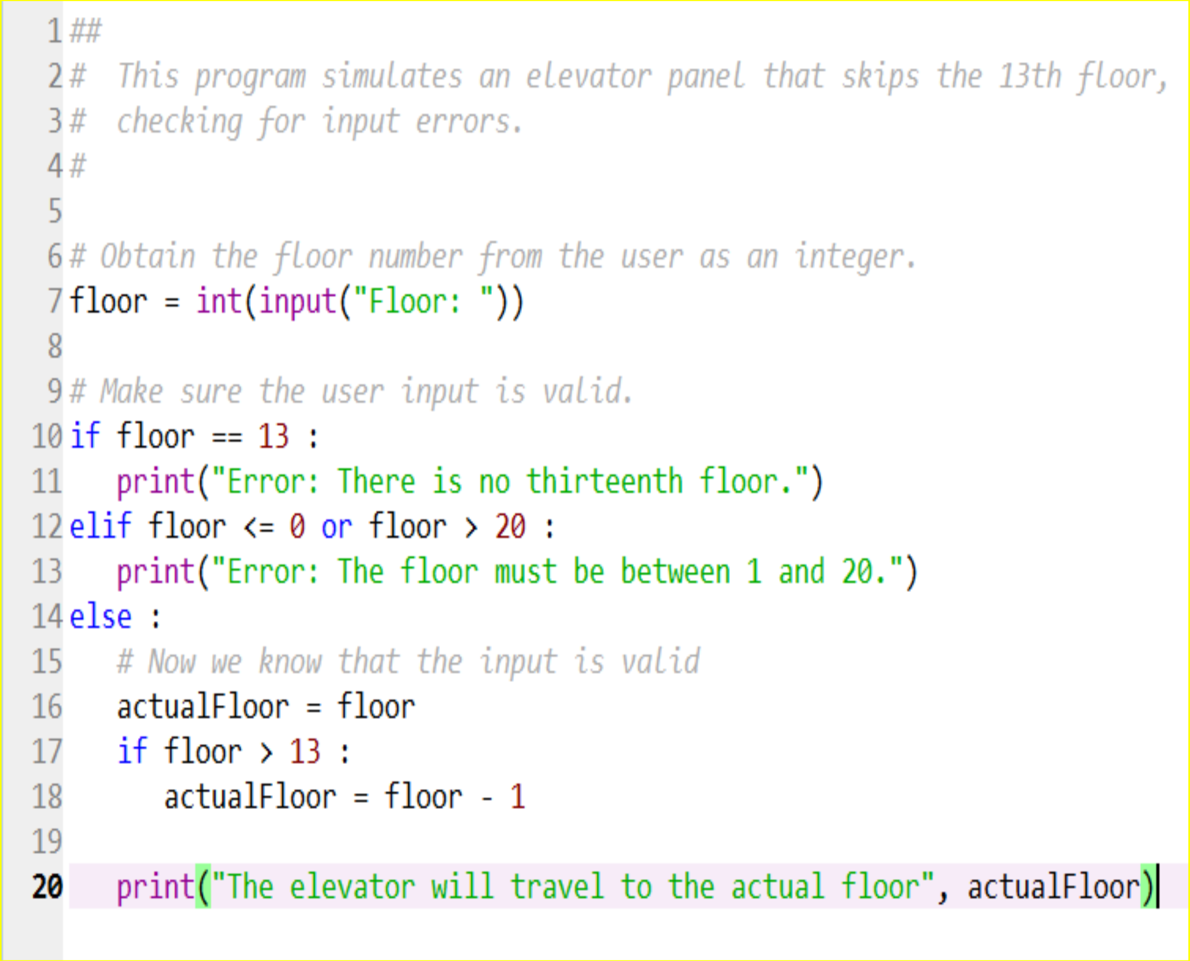


```
##
#  This program simulates an elevator panel that skips the 13th floor,
#   checking for input errors.
#

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

# Make sure the user input is valid.
if floor == 13 :
   print("Error: There is no thirteenth floor.")
elif floor <= 0 or floor > 20 :
   print("Error: The floor must be between 1 and 20.")
else :
   # Now we know that the input is valid.
   actualFloor = floor
```

Complete Program: elevatorsim2.py



```python
##
#  This program simulates an elevator panel that skips the 13th floor,
#  checking for input errors.
#

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

# Make sure the user input is valid.
if floor == 13 :
    print("Error: There is no thirteenth floor.")
elif floor <= 0 or floor > 20 :
    print("Error: The floor must be between 1 and 20.")
else :
    # Now we know that the input is valid
    actualFloor = floor
    if floor > 13 :
        actualFloor = floor - 1

    print("The elevator will travel to the actual floor", actualFloor)
```

- Test the program with a range of inputs including:
  - 12
  - 14
  - 13
  - -1
  - 0
  - 23
  - 19

Code Academy

Python Operator Precedence

Highest precedence at top, lowest at bottom.

Operators in the same box evaluate left to right.

Highest

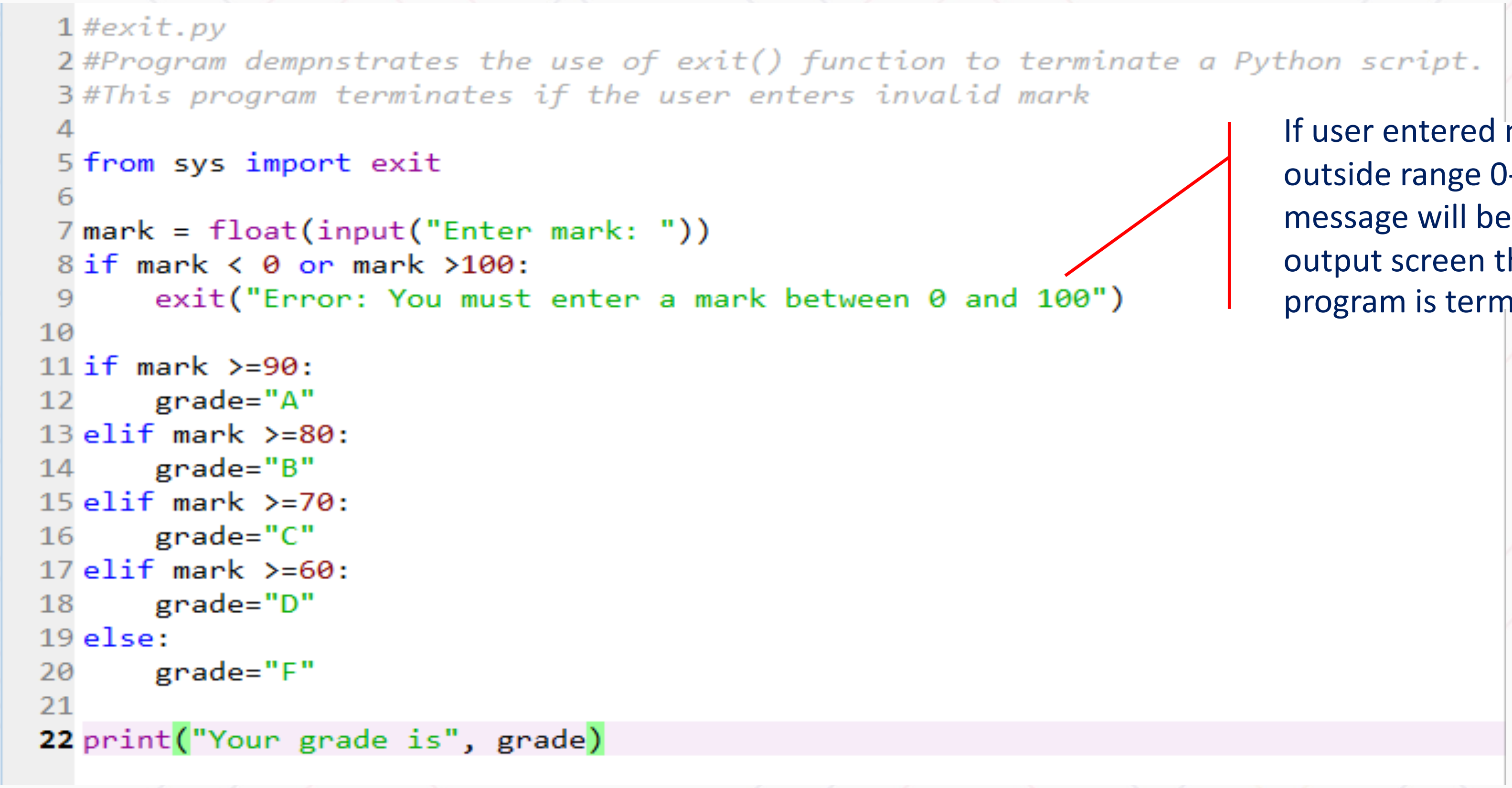
Operator	Description
()	Parentheses (grouping)
f(args...)	Function call
x[index:index]	Slicing
x[index]	Subscription
**	Exponentiation
+x, -x	Positive, negative
*, /, %	Multiplication, division, remainder
+, -	Addition, subtraction
in, not in, <, <=, >, >=, !=, ==	Comparisons, membership, identity
not x	Boolean NOT
and	Boolean AND
or	Boolean OR
=	Assignment operator

Lowest

For more information, see Python documentation on operator precedence (Section 5.15)

Terminating a Python Script: exit.py

- In some program, you may need to stop the program when invalid data is entered.
- The **exit()** function defined in the **sys** standard library module immediately aborts the program when executed. The program **exit.py** below demonstrates the use of exit function.



```python
#exit.py
#Program dempnstrates the use of exit() function to terminate a Python script.
#This program terminates if the user enters invalid mark

from sys import exit

mark = float(input("Enter mark: "))
if mark < 0 or mark >100:
    exit("Error: You must enter a mark between 0 and 100")

if mark >=90:
    grade="A"
elif mark >=80:
    grade="B"
elif mark >=70:
    grade="C"
elif mark >=60:
    grade="D"
else:
    grade="F"

print("Your grade is", grade)
```

If user entered mark value outside range 0-100, this message will be printed on output screen then the program is terminated

Terminating a Python Script: exit.py (2)

First Run

```
runfile('D:/python/src/exit.py', wdir='D:/python/src')

Enter mark: 102
C:\Users\pc1\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2889:
UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
An exception has occurred, use %tb to see the full traceback.

SystemExit: Error: You must enter a mark between 0 and 100
```

Second Run

```
runfile('D:/python/src/exit.py', wdir='D:/python/src')

Enter mark: 92
Your grade is A
```

Summary: Boolean

- Boolean
  - The type `boolean` has two values, **true** and **false**.
  - Python has two Boolean operators that combine conditions: **`and`** and **`or`**.
  - To invert a condition, use the *`not`* operator.
  - When checking for equality use the `!` operator.
  - The **and** and **`or`** operators are computed lazily:
    - As soon as the truth value is determined, no further conditions are evaluated.
  - De Morgan’s law tells you how to negate **and** and **`or`** conditions.