8.2. Boolean Values and Boolean Expressions

The Python type for storing true and false values is called bool, named after the British mathematician, George Boole. George Boole created *Boolean Algebra*, which is the basis of all modern computer arithmetic.

There are only two **boolean values**. They are True and False. Capitalization is important, since true and false are not boolean values (remember Python is case sensitive).

print(True)

print(type(True))

print(type(False))

​

**Note**

Boolean values are not strings!

It is extremely important to realize that True and False are not strings. They are not surrounded by quotes. They are the only two values in the data type bool. Take a close look at the types shown below.

print(type(True))

print(type("True"))

​

A **boolean expression** is an expression that evaluates to a boolean value. The equality operator, ==, compares two values and produces a boolean value related to whether the two values are equal to one another.

print(5 == 5)

print(5 == 6)

​

In the first statement, the two operands are equal, so the expression evaluates to True. In the second statement, 5 is not equal to 6, so we get False.

The == operator is one of six common **comparison operators**; the others are:

x != y *# x is not equal to y*

x > y *# x is greater than y*

x < y *# x is less than y*

x >= y *# x is greater than or equal to y*

x <= y *# x is less than or equal to y*

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols. A common error is to use a single equal sign (=) instead of a double equal sign (==). Remember that = is an assignment operator and == is a comparison operator. Also, there is no such thing as =< or =>.

Note too that an equality test is symmetric, but assignment is not. For example, if a == 7 then 7 == a. But in Python, the statement a = 7 is legal and 7 = a is not. (Can you explain why?)

**Check your understanding**

condition-2-1: Which of the following is a Boolean expression? Select all that apply.

Top of Form

A. True  
B. 3 == 4  
C. 3 + 4  
D. 3 + 4 == 7  
E. "False"

8.3. Logical operators

There are three **logical operators**: and, or, and not. The semantics (meaning) of these operators is similar to their meaning in English. For example, x > 0 and x < 10 is true only if x is greater than 0 *and* at the same time, x is less than 10. How would you describe this in words? You would say that x is between 0 and 10, not including the endpoints.

n % 2 == 0 or n % 3 == 0 is true if *either* of the conditions is true, that is, if the number is divisible by 2 *or* divisible by 3. In this case, one, or the other, or both of the parts has to be true for the result to be true.

Finally, the not operator negates a boolean expression, so not  x > y is true if x > y is false, that is, if x is less than or equal to y.

x = 5

print(x>0 and x<10)

​

n = 25

print(n%2 == 0 or n%3 == 0)

​

**Common Mistake!**

There is a very common mistake that occurs when programmers try to write boolean expressions. For example, what if we have a variable number and we want to check to see if its value is 5, 6, or 7? In words we might say: “number equal to 5 or 6 or 7”. However, if we translate this into Python, number == 5 or 6 or 7, it will not be correct. The or operator must join the results of three equality checks. The correct way to write this is number == 5 or number == 6 or number == 7.

This may seem like a lot of typing but it is absolutely necessary. You cannot take a shortcut.

Well, actually, you can take a shortcut but not that way. Later in this chapter you’ll learn about the in operator for strings and sequences: you could write number in [5, 6, 7].

**Check your understanding**

condition-3-1: What is the correct Python expression for checking to see if a number stored in a variable x is between 0 and 5.

Top of Form

A. x > 0 and < 5  
B. 0 < x < 5  
C. x > 0 or x < 5  
D. x > 0 and x < 5

8.4. The in and not in operators

The in operator tests if one string is a substring of another:

print('p' in 'apple')

print('i' in 'apple')

print('ap' in 'apple')

print('pa' in 'apple')

​

Note that a string is a substring of itself, and the empty string is a substring of any other string. (Also note that computer scientists like to think about these edge cases quite carefully!)

print('a' in 'a')

print('apple' in 'apple')

print('' in 'a')

print('' in 'apple')

​

The not in operator returns the logical opposite result of in.

print('x' not in 'apple')

8.5. Precedence of Operators

Arithmetic operators take precedence over logical operators. Python will always evaluate the arithmetic operators first (\*\* is highest, then multiplication/division, then addition/subtraction). Next comes the relational operators. Finally, the logical operators are done last. This means that the expression x\*5 >= 10 and y-6 <= 20 will be evaluated so as to first perform the arithmetic and then check the relationships. The and will be done last. Many programmers might place parentheses around the two relational expressions, (x\*5 >= 10) and (y-6 <= 20). It is not necessary to do so, but causes no harm and may make it easier for people to read and understand the code.

The following table summarizes the operator precedence from highest to lowest. A complete table for the entire language can be found in the [Python Documentation](http://docs.python.org/py3k/reference/expressions.html#expression-lists).

| **Level** | **Category** | **Operators** |
| --- | --- | --- |
| 7(high) | exponent | \*\* |
| 6 | multiplication | \*,/,//,% |
| 5 | addition | +,- |
| 4 | relational | ==,!=,<=,>=,>,< |
| 3 | logical | not |
| 2 | logical | and |
| 1(low) | logical | or |

8.6. Conditional Execution: Binary Selection

In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. **Selection statements**, sometimes also referred to as **conditional statements**, give us this ability. The simplest form of selection is the **if statement**. This is sometimes referred to as **binary selection** since there are two possible paths of execution.

x = 15

​

if x % 2 == 0:

print(x, "is even")

else:

print(x, "is odd")

​

The syntax for an if statement looks like this:

**if** BOOLEAN EXPRESSION:

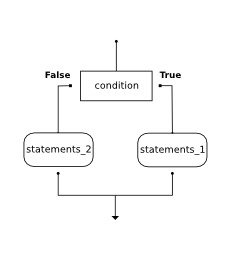
STATEMENTS\_1 *# executed if condition evaluates to True*

**else**:

STATEMENTS\_2 *# executed if condition evaluates to False*

The boolean expression after the if statement is called the **condition**. If it is true, then the indented statements get executed. If not, then the statements indented under the else clause get executed.

**Flowchart of a if statement with an else**



As with the function definition from the last chapter and other compound statements like for, the if statement consists of a header line and a body. The header line begins with the keyword if followed by a *boolean expression* and ends with a colon (:).

The indented statements that follow are called a **block**. The first unindented statement marks the end of the block.

Each of the statements inside the first block of statements is executed in order if the boolean expression evaluates to True. The entire first block of statements is skipped if the boolean expression evaluates to False, and instead all the statements under the else clause are executed.

There is no limit on the number of statements that can appear under the two clauses of an if statement, but there has to be at least one statement in each block.

**Check your understanding**

condition-6-1: How many lines of code can appear in the indented code block below the if and else lines in a conditional?

Top of Form

A. Just one.  
B. Zero or more.  
C. One or more.  
D. One or more, and each must contain the same number.

Bottom of Form

condition-6-2: What does the following code print? (choose from output a, b, c or nothing)

**if** (4 + 5 == 10):

**print**("TRUE")

**else**:

**print**("FALSE")

Top of Form

A. TRUE  
B. FALSE  
C. TRUE on one line and FALSE on the next  
D. Nothing will be printed

Bottom of Form

condition-6-3: What does the following code print?

**if** (4 + 5 == 10):

**print**("TRUE")

**else**:

**print**("FALSE")

**print**("TRUE")

a. TRUE

b.

TRUE

FALSE

c.

FALSE

TRUE

d.

TRUE

FALSE

TRUE

Top of Form

A. Output a  
B. Output b  
C. Output c  
D. Output d

8.7. Chained conditionals

Python provides an alternative way to write nested selection such as the one shown in the previous section. This is sometimes referred to as a **chained conditional**.

**if** x < y:

**print**("x is less than y")

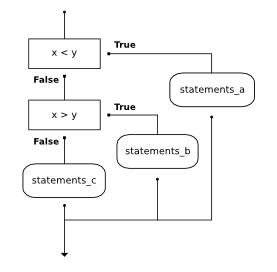
**elif** x > y:

**print**("x is greater than y")

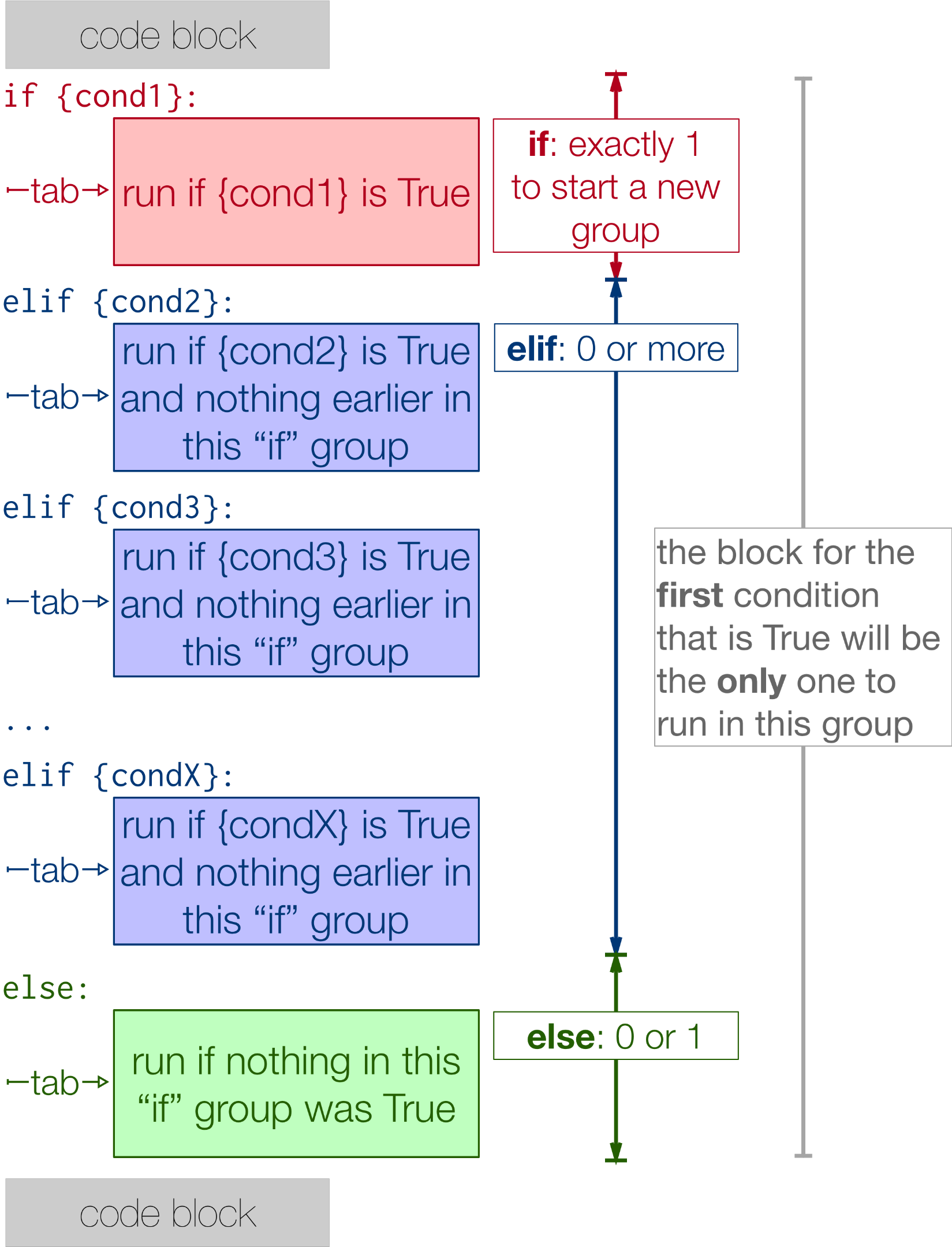
**else**:

**print**("x and y must be equal")

The flow of control can be drawn in a different orientation but the resulting pattern is identical to the one shown above.



elif is an abbreviation of else if. Again, exactly one branch will be executed. There is no limit of the number of elif statements but only a single (and optional) final else statement is allowed and it must be the last branch in the statement.



Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

Here is the same program using elif.

x = 10

y = 10

​

if x < y:

print("x is less than y")

elif x > y:

print("x is greater than y")

else:

print("x and y must be equal")

Bottom of Form

Bottom of Form

Bottom of Form