CHAPTER 3: Decision Structures and Boolean Logic

3.1 The if Statement

The if statement is used to create a decision structure, which allows a program to have more than one path of execution. The if statement causes one or more statements to execute only when a Boolean expression is true.

A *sequence structure* is a set of statements that execute in the order that they appear.

A *control structure* is a logical design that controls the order in which a set of statements execute.

A decision structure (also known as a *selection structure*) is a form of a structure, that a specific action is performed only if a certain condition exists. If the condition does not exist, the action is not performed.

In the flowchart, the diamond symbol indicates some condition that must be tested.

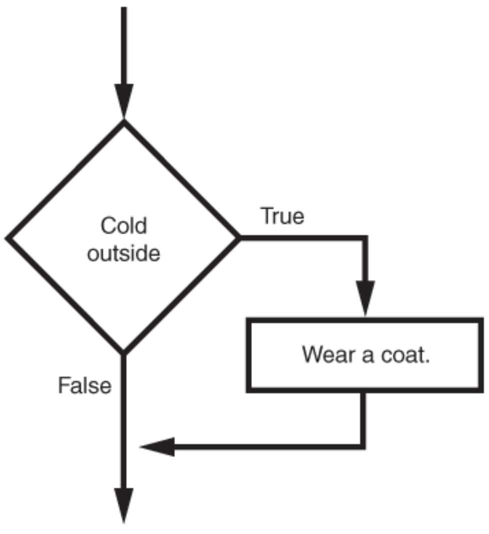


Figure 3-1 A simple decision structure

In Python we use the if statement to write a single alternative decision structure.

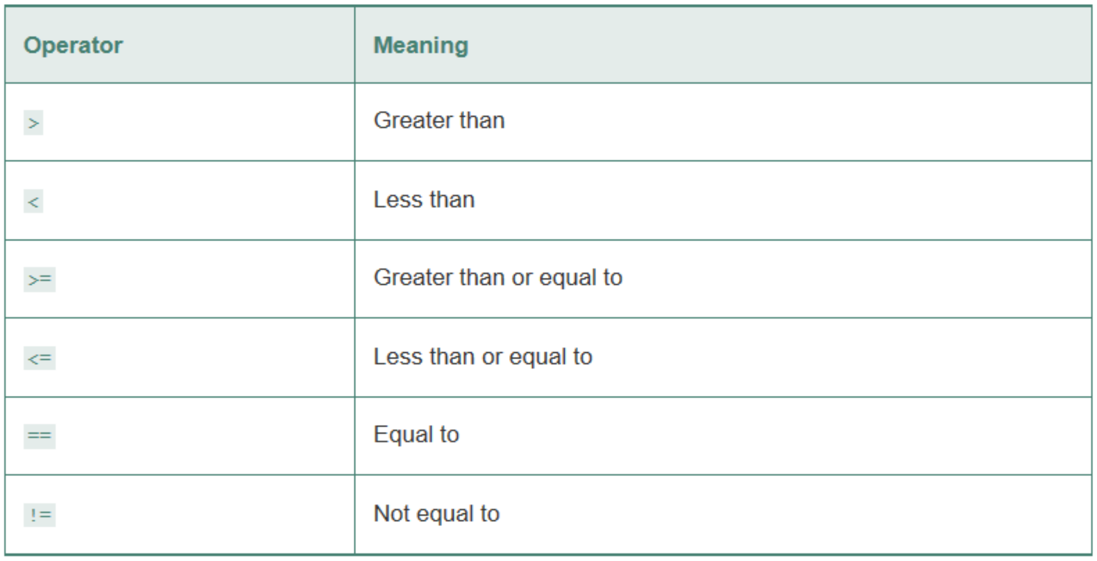
```
if condition:
    statement
    statement
    etc.
```

When the if statement executes, the *condition* is tested. If the *condition* is true, the statements that appear in the block following the if clause are executed. If the condition is false, the statements in the block are skipped.

Boolean Expressions and Relational Operators

The expressions that are tested by the if statement are called *Boolean expressions*.

Table 3-1 Relational operators



Operator	Meaning
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal to
`<=`	Less than or equal to
`==`	Equal to
`!=`	Not equal to

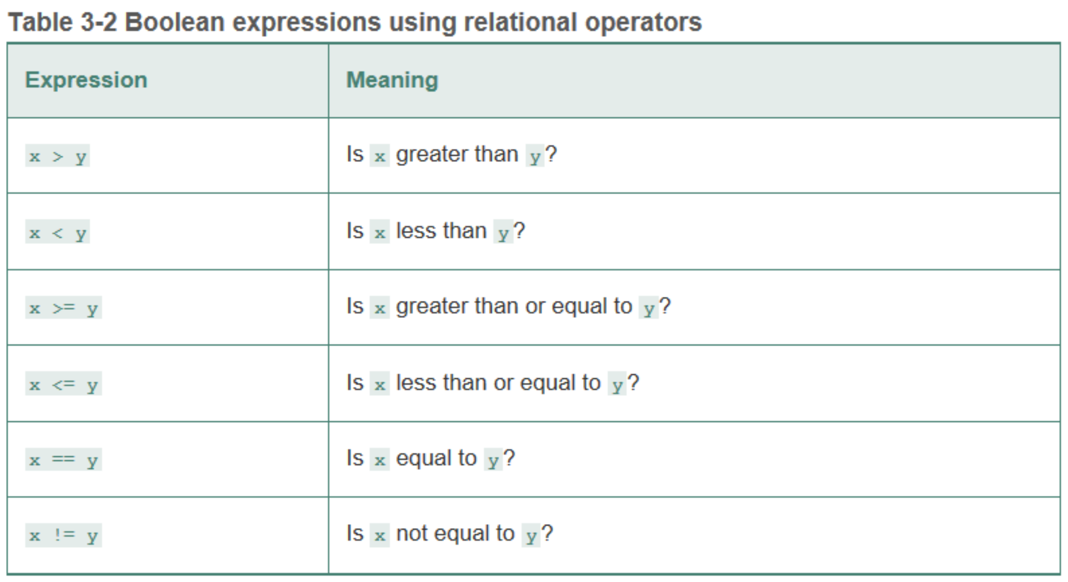
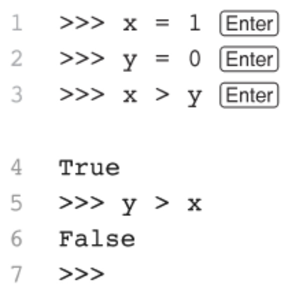


Table 3-2 Boolean expressions using relational operators

Expression	Meaning
`x > y`	Is `x` greater than `y`?
`x < y`	Is `x` less than `y`?
`x >= y`	Is `x` greater than or equal to `y`?
`x <= y`	Is `x` less than or equal to `y`?
`x == y`	Is `x` equal to `y`?
`x != y`	Is `x` not equal to `y`?

Example:



```
>>> x = 1 Enter
>>> y = 0 Enter
>>> x > y Enter

True
>>> y > x
False
>>>
```

3.2 The if-else Statement

An if-else statement will execute one block of statements if its condition is true, or another block if its condition is false.

A *dual alternative decision structure* is a structure which has two possible paths of execution—one path is taken if a condition is true, and the other path is taken if the condition is false.

The flowchart for a dual alternative decision structure.

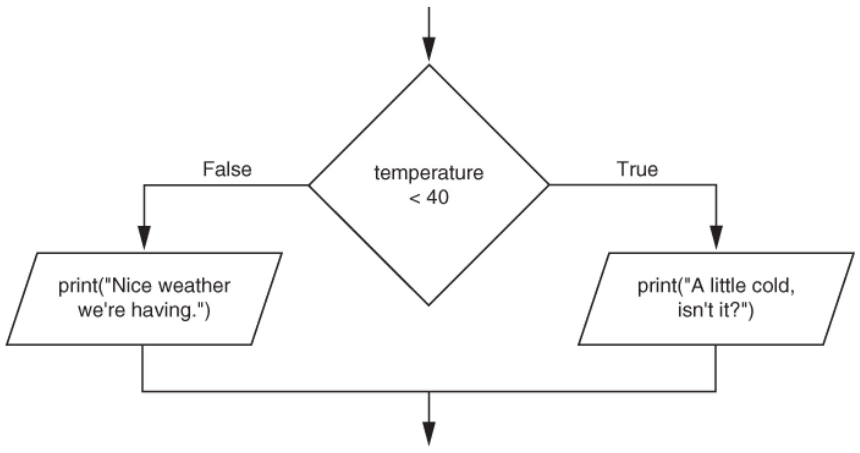

False
temperature < 40
True
print("Nice weather we're having.")
print("A little cold, isn't it?")

Figure 3-5 A dual alternative decision structure

In code we write a dual alternative decision structure as an if-else statement. The general format of the if-else statement is:

```
if condition:
    statement
    statement
    etc.
else:
    statement
    statement
    etc.
```

Indentation in the if-else Statement

When you write an if-else statement, follow these guidelines for indentation:

- Make sure the if clause and the else clause are aligned.
- The if clause and the else clause are each followed by a block of statements. Make sure the statements in the blocks are consistently indented.

3.3 Comparing Strings

Python allows you to compare strings. This allows you to create decision structures that test the value of a string.

```
name1 = 'Mary'
name2 = 'Mark'
if name1 == name2:
        print('The names are the same.')
else:
        print('The names are NOT the same.')
```

Other String Comparisons

In addition to determining whether strings are equal or not equal, you can also determine whether one string is greater than or less than another string.

- The uppercase characters A through Z are represented by the numbers 65 through 90.
- The lowercase characters a through z are represented by the numbers 97 through 122.
- When the digits 0 through 9 are stored in memory as characters, they are represented by the numbers 48 through 57. (For example, the string 'abc123' would be stored in memory as the codes 97, 98, 99, 49, 50, and 51.)
- A blank space is represented by the number 32.

In addition to establishing a set of numeric codes to represent characters in memory, ASCII also establishes an order for characters. The character "A" comes before the character "B", which comes before the character "C", and so on.

When a program compares characters, it actually compares the codes for the characters.

3.4 Nested Decision Structures and the if-elif-else Statement

To test more than one condition, a decision structure can be nested inside another decision structure.

Example:
The program determines whether a bank customer qualifies for a loan. To qualify, two conditions must exist:
(1) the customer must earn at least $30,000 per year
(2) the customer must have been employed for at least two years.

Flowchart:

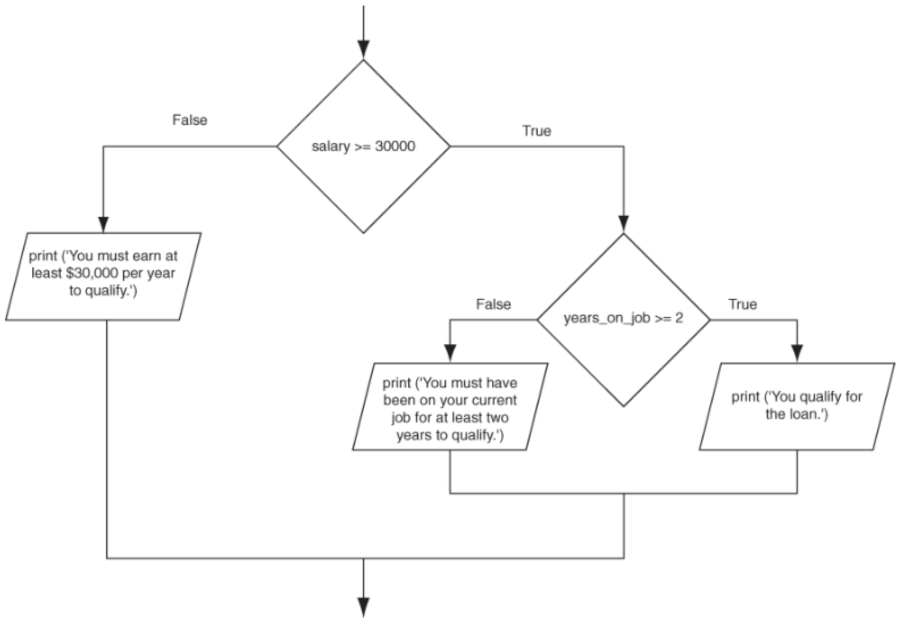

salary >= 30000
False
True
print ('You must earn at least $30,000 per year to qualify.')
years_on_job >= 2
False
True
print ('You must have been on your current job for at least two years to qualify.')
print ('You qualify for the loan.')

Figure 3-12 A nested decision structure

Testing a Series of Conditions

A decision structure with numerous other decision structures nested inside it.

Example:

The following 10-point grading scale for an exam is assigned:

Test Score	Grade
90 and above	A
80–89	B
70–79	C
60–69	D
Below 60	F

The program determines the grade for the given score.

Flowchart:

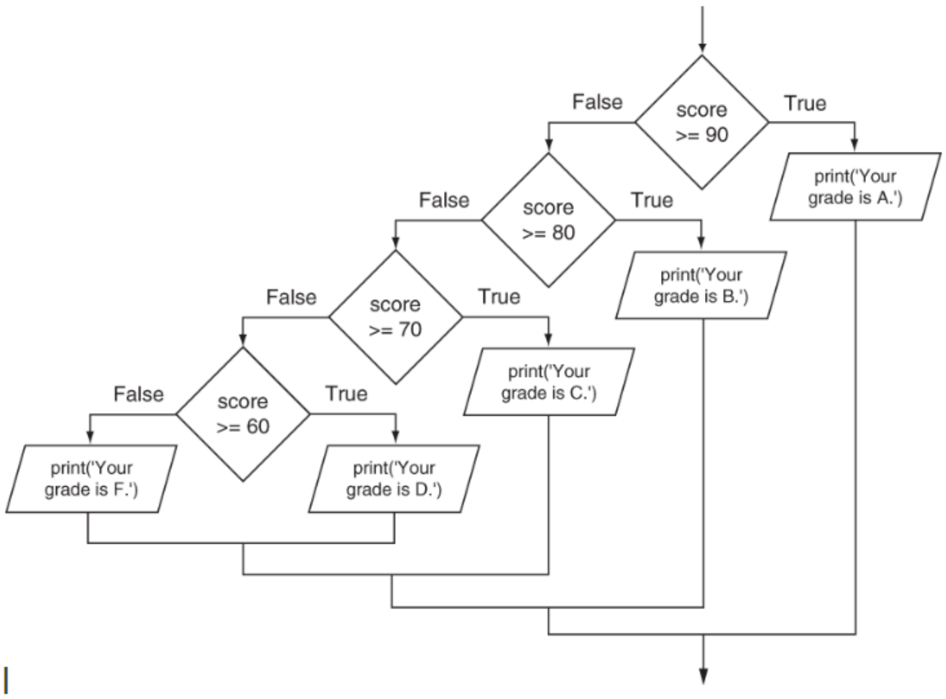

score >= 90
True
False
print('Your grade is A.')
score >= 80
True
False
print('Your grade is B.')
score >= 70
True
False
print('Your grade is C.')
score >= 60
True
False
print('Your grade is D.')
print('Your grade is F.')

Figure 3-15 Nested decision structure to determine a grade

The if-elif-else Statement

The if-elif-else Statement provides a special version of the decision structure, which makes this type of logic simpler to write.

The general format of the if-elif-else statement is:

```
if condition_1:
    statement
    statement
    etc.

elif condition_2:
    statement
    statement
    etc.
Insert as many elif clauses as necessary . . .
else:
    statement
    statement
    etc.
```

When the statement executes, *condition_1* is tested. If *condition_1* is true, the block of statements that immediately follow is executed, up to the elif clause. The rest of the structure is ignored. If condition_1 is false, however, the program jumps to the very next elif clause and tests condition_2 . If it is true, the block of statements that immediately follow is executed, up to the next elif clause. The rest of the structure is then ignored. This process continues until a condition is found to be true, or no more elif clauses are left. If no condition is true, the block of statements following the else clause is executed.

if-elif-else flowchart

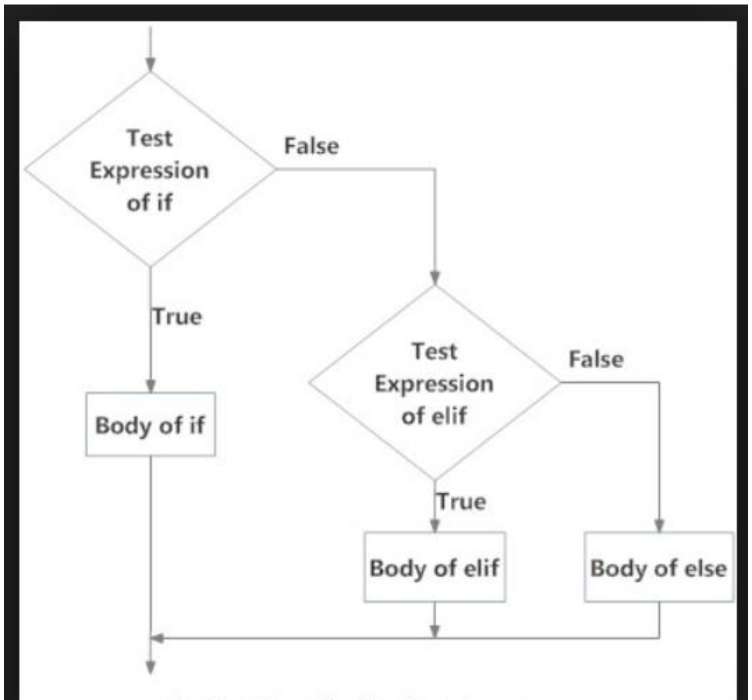
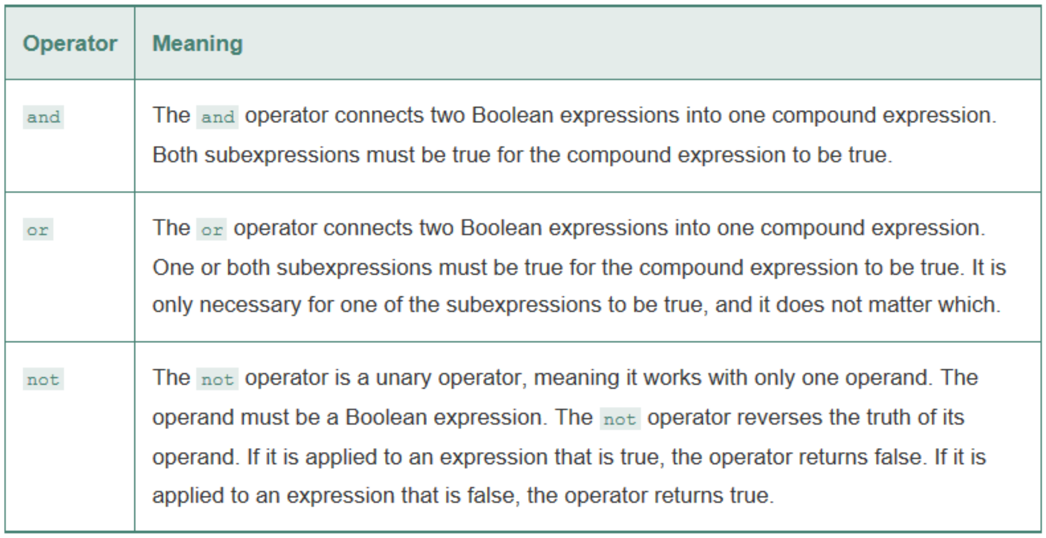


Fig: Operation of if...elif...else statement

3.5 Logical Operators

The logical **and** operator and the logical **or** operator allow you to connect multiple Boolean expressions to create a compound expression. The logical **not** operator reverses the truth of a Boolean expression.

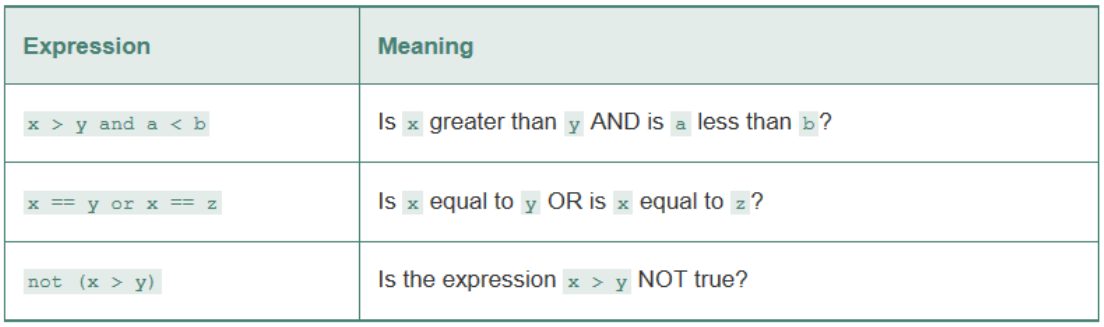
Table 3-3 Logical operators



Operator	Meaning
`and`	The `and` operator connects two Boolean expressions into one compound expression. Both subexpressions must be true for the compound expression to be true.
`or`	The `or` operator connects two Boolean expressions into one compound expression. One or both subexpressions must be true for the compound expression to be true. It is only necessary for one of the subexpressions to be true, and it does not matter which.
`not`	The `not` operator is a unary operator, meaning it works with only one operand. The operand must be a Boolean expression. The `not` operator reverses the truth of its operand. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

Example:

Table 3-4 Compound Boolean expressions using logical operators



Expression	Meaning
`x > y and a < b`	Is `x` greater than `y` AND is `a` less than `b`?
`x == y or x == z`	Is `x` equal to `y` OR is `x` equal to `z`?
`not (x > y)`	Is the expression `x > y` NOT true?

The and Operator

The **and** operator takes two Boolean expressions as operands and creates a compound Boolean expression that is true only when both subexpressions are true.

The or Operator

The **or** operator takes two Boolean expressions as operands and creates a compound Boolean expression that is true when either of the subexpressions is true.

Short-Circuit Evaluation

Both the **and** and **or** operators perform *short-circuit evaluation.*

How it works with the **and** operator: If the expression on the left side of the and operator is false, the expression on the right side will not be checked.

How it works with the **or** operator: If the expression on the left side of the or operator is true, the expression on the right side will not be checked.

The not Operator

The **not** operator is a unary operator that takes a Boolean expression as its operand and reverses its logical value.

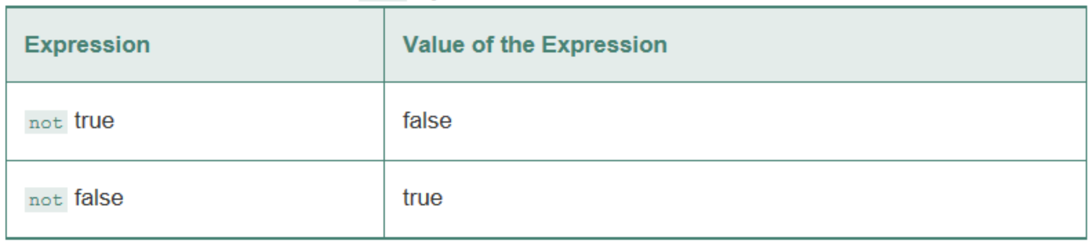


Table 3-7 Truth table for the `not` operator

Expression	Value of the Expression
`not` true	false
`not` false	true

3.6 Boolean Variables

A Boolean variable can reference one of two values: True or False.

In addition to int, float, and str (string) variables, Python provides a bool data type. The bool data type variables provide reference to one of two possible values: True or False.

Example:

```
hungry = True
sleepy = False
```

Boolean variables used as flags

A *flag* is a variable that signals when some condition exists in the program. When the flag variable is set to False, it indicates the condition does not exist. When the flag variable is set to True, it means the condition does exist.

Example:

Let suppose a salesperson has a quota of $50,000. Assuming sales references the amount that the salesperson has sold, the following code determines whether the quota has been met:

```
if sales >= 50000.0:
    sales_quota_met = True
else:
    sales_quota_met = False
```

The sales_quota_met variable can be used as a flag to indicate whether the sales quota has been met.

The flag can be used in the program in the following way:

```
if sales_quota_met:
    print('You have met your sales quota!')
```

This code displays 'You have met your sales quota!' if the bool variable sales_quota_met is True.

We do not have to use the == operator to explicitly compare the sales_quota_met variable with the value True. This code is equivalent to the following:

```
if sales_quota_met == True:
   print('You have met your sales quota!')
```