

Chapter 3

Conditional statements and Boolean logic

1. Learning objectives

By the end of this chapter, students should be able to:

- Understand Boolean values and Boolean expressions.
- Use comparison operators to compare values.
- Write simple if statements.
- Use if-else to handle two alternatives.
- Use if-elif-else to handle multiple alternatives.
- Combine conditions using and, or, and not.
- Use conditional statements with user input.
- Apply conditional logic to simple practical and engineering situations.
- Identify and correct common errors in conditional statements.

2. Introduction

In Chapter 2, we learned how to store values in variables, perform calculations, receive input from the user, and display results. The programs we have written so far generally execute their instructions in sequence.

For example:

```
temperature = 25
print(f"Temperature = {temperature} °C")
```

The program displays the temperature regardless of its value. However, many programs need to **make decisions**.

For example:

- display a warning if a temperature is too high;
- classify a value as low, normal, or high;
- determine whether a measurement is within an acceptable range;
- decide whether a condition is satisfied before performing an operation.

Python allows us to make such decisions using **conditional statements**. A conditional or branching statement evaluates a condition and executes different instructions depending on whether the condition is `True` or `False`.

3. Concepts

3.1. Boolean values

A Boolean value represents one of two logical states, `True` or `False`. `True` means that a condition is satisfied, while `False` means that it is not.

For example:

```
>>> temperature = 25
>>> temperature > 20

True

>>> print(temperature > 30)

False
```

In general, Boolean values are used in conditional statements to make decisions.

3.2.Relational operators

Python provides several operators for comparing values. The meaning and an example of each operator are indicated in *Table 3.1*

Table 3.1 Python logical operators

Operator	Meaning	Example
>	greater than	speed > 10
<	less than	speed < 10
>=	greater than or equal to	speed >= 10
<=	less than or equal to	speed <= 10
==	equal to	speed == 10
!=	not equal to	Speed != 10

3.3.Assignment versus Comparison

It is important to distinguish between = and ==, where the = operator is used for **assignment** and the == operator is used for **comparison**.

For example

```
temperature = 25
```

This stores 25 in the variable temperature. Where:

```
temperature == 25
```

This asks whether temperature is equal to 25 or not. The result is either True or False.

4. The if Statement

The if statement allows a program to execute an instruction only when a condition is true. The general syntax is:

```
if condition:
    statement
```

For example:

```
number = 20
if number > 0:
    print("Positive number")
```

Output:

Positive number

If the number is changed to `number = -30`, the condition is false, so the message is not displayed.

5. Indentation

Indentation is an essential part of Python syntax. The statements belonging to an `if` statement must be indented. Python commonly uses **four spaces** for indentation.

For example:

```
if number > 0:
    print("Positive number")
```

The `print()` statement belongs to the `if` block because it is indented.

6. The if-else Statement

Sometimes a program needs to choose between two alternatives. Generally, the `if` statement is followed by another statement called **`else`** that allows to run another code or program when the condition in the `if` statement is false.

The `if-else` structure is:

```
if condition:
    statement_if_true
else:
    statement_if_false
```

Example:

```
temperature = 25

if temperature > 35:
    print("High temperature")
else:
    print("Temperature is acceptable")
```

Output:

Temperature is acceptable

If the temperature is 40, the output becomes:

High temperature

Exactly one of the two blocks is executed.

7. The if-elif-else Statement

When there are more than two possible situations, **if** statement can be followed by one or more optional **elif** and an **else** statement, which is very useful to test various conditions. The basic syntax of **if-elif-else** statement is expressed as follows:

```
if condition1:
    statement1
elif condition2:
    statement2
else:
    statement3
```

For example:

```
score = 13

if score >= 15:
    print("Excellent")
elif score >= 10:
    print("Good")
else:
    print("Needs improvement")
```

Output:

Good

A program can contain more than one **elif**.

For example:

```
hour = 15

if hour < 12:
    print("Morning")
elif hour < 17:
    print("Afternoon")
elif hour < 20:
    print("Evening")
else:
    print("Night")
```

Output:

Afternoon

Python evaluates the conditions from top to bottom. When it finds the first condition that is `True`, it executes its block and skips the remaining alternatives.

8. Logical operators

Sometimes we need more than one condition. Python provides three logical operators:

- `and`
- `or`
- `not`

They allow us to combine or modify Boolean expressions.

The `and` operator

The `and` operator is `True` only when **both conditions are true**.

For example:

```
temperature = 30

if temperature >= 20 and temperature <= 40:
    print("Temperature is within the desired range")
```

Both conditions are true, so the message is displayed.

The `or` operator

The `or` operator is `True` when **at least one condition is true**.

For example:

```
score = -5

if score < 0 or score > 20:
    print("Score is outside the normal range")
```

The first condition is true, so the message is displayed.

The `not` operator

The `not` operator reverses a Boolean value.

For example:

```
>>> print(not True)
False
>>> print(not False)
True
```

It can also be used with a Boolean expression:

```
temperature = 25

if not (speed > 80):
    print("Speed is not high")
```

9. Combining conditions

Comparison and logical operators can be combined.

For example:

```
temperature = 45
pressure = 90

if (temperature > 40) or (pressure > 100):
    print("Warning")
```

Another example:

```
if age > 0 and age < 18:
    print("Under 18")
```

Python provide a more concise alternative for and combination:

```
if 0 < age < 18:
    print("Under 18")
```

10. Nested conditional statements

A conditional statement can contain another conditional statement.

For example:

```
temperature = 25
pressure = 95

if temperature <= 30:
    if pressure <= 100:
        print("Conditions are acceptable")
```

This is called a **nested conditional statement**.

Nested conditions are useful when one decision depends on another. However, they should not be used when a simpler condition can express the same logic clearly.

For example, the previous program can also be written as:

```
if temperature <= 30 and pressure <= 100:
    print("Conditions are acceptable")
```

For simple situations, this version is usually easier to read.

11. Using Conditions with `input()`

In Chapter 2, we learned that `input()` returns a string. Therefore, when numerical input is required, we need type conversion.

Example:

```
total_purchase = float(input("Enter total purchase (DA): "))
if total_purchase > 5000:
    discount = total_purchase * 0.10
    final_price = total_purchase - discount
    print(f"Discount 10%: {discount:.2f} ")
    print(f"Final price: {final_price:.2f} ")
else:
    print(f"No Discount! ")
    print(f"Final price: {total_purchase:.2f} ")
```

A possible interaction is:

```
Enter total purchase (DA): 8000
Discount 10%: 800.00
Final price: 7200.00
```

This example combines several concepts learned in Chapters 2 and 3: `input()`, `float()`, variables, comparison, conditional statements and output function

12. Worked Examples

Example 1 - Battery voltage

```
battery_voltage = 11.5
if battery_voltage < 12.0:
    print("Voltage is below the reference value.")
```

Example 2 - Even or Odd number

```
number = int(input("Enter a number "))
if number % 2 == 0:
    print("Even number")
else:
    print("Odd number ")
```

Example 3 - Positive, Negative or Null

```
x = int(input("Enter a number: "))
if x > 0:
    type = 'positive'
```

```
elif x < 0:
    type = 'negative'
else:
    type = 'null'
print("The entered number is ", type)
```

Example 4 - Two Operating Conditions

Suppose a process operates normally when:

- temperature is between 20 °C and 80 °C;
- pressure is between 90 kPa and 110 kPa.

```
temperature = float(input("Enter temperature (°C): "))
pressure = float(input("Enter pressure (kPa): "))

if (20 <= temperature <= 80) and (90 <= pressure <= 110):
    print("Operating conditions are acceptable")
else:
    print("Check operating conditions")
```

Example 5 - Formatted Conditional Output

We can combine conditional statements with the formatted output learned in Chapter 2.

```
voltage = float(input("Enter voltage (V): "))

if voltage > 10:
    print(f"Voltage = { voltage:.2f} V")
    print("Status = High")
else voltage > 3:
    print(f"Voltage = { voltage:.2f} V")
    print("Status = Medium")
else:
    print(f"Voltage = { voltage:.2f} V")
    print("Status = Low")
```

Possible output:

```
Voltage = 5.6 V
Status = Medium
```

13.Common Beginner Mistakes

1. Using = Instead of ==

For assignment we use `x = 10`, where for a comparison we use `x == 10`

Incorrect:

```
if temperature = 30:
    print("Temperature is 30")
```

Correct:

```
if temperature == 30:
    print("Temperature is 30")
```

2. Forgetting the Colon

Incorrect:

```
if temperature > 30
    print("High temperature")
```

Correct:

```
if temperature > 30:
    print("High temperature")
```

3. Incorrect Indentation

Incorrect:

```
if temperature > 30:
print("High temperature")
```

Correct:

```
if temperature > 30:
    print("High temperature")
```

4. Forgetting Type Conversion

Incorrect:

```
temperature = input("Enter temperature: ")

if temperature > 30:
    print("High temperature")
```

The value returned by `input()` is a string.

Correct:

```
temperature = float(input("Enter temperature: "))

if temperature > 30:
    print("High temperature")
```

5. Incorrect Order of Conditions

Consider:

```
temperature = 50

if temperature > 25:
    print("Normal")
elif temperature > 45:
    print("High")
```

The output will be always:

Normal

The second condition is never reached because `temperature > 20` is already `True`.

A better arrangement is:

```
if temperature > 45:
    print("High")
elif temperature > 25:
    print("Normal")
```

The order of conditions matters.

14.Exercises

Exercise 1

Ask the user to enter two numbers then display:

- First number is larger if the first is larger.
- Second number is larger if the second is larger.
- The numbers are equal if they are equal.

Exercise 2

Ask the user to enter a stored username and password. For this exercise, use:

```
username = "student"
password = "python123"
```

Display "Login successful" only when both values are correct. Otherwise, display "Invalid login".

Exercise 3

Ask the user to enter a speed.

Use the following table to display the classification.

Speed	Classification
Below 40	Low
40–80	Normal
Above 80	High

Exercise 4

Write a program that asks the user to enter:

- a distance in meters;
- a time in seconds.

Calculate the speed: $S = \frac{d}{t}$

Then classify the speed using classification table provided in Exercise 2.

Display the distance, time, calculated speed, and classification using formatted output.

Example of possible output format is:

```
Engineering Experiment
-----
Distance: 150 m
Time: 50 s
Speed: 15 m/s
Classification: Normal
```

Exercise 5

Using **if-else statements**, write a Python program to calculate the determinant of a second-degree equation and find its solutions

15.Summary

- Programs do not always need to execute the same instructions for every input. Conditional statements allow Python programs to respond differently depending on the values they receive.
- Comparisons such as `>`, `<`, `>=`, `<=`, `==`, and `!=` produce Boolean results. These results can be used as conditions in an `if` statement.
- The `if` statement handles a single decision. `if-else` provides two alternatives, while `if-elif-else` allows a program to choose between several alternatives.
- Logical operators extend this decision-making ability. `and` requires all combined conditions to be true, `or` requires at least one condition to be true, and `not` reverses a Boolean result.

- Conditional statements can be combined with variables, user input, type conversion, arithmetic operations, and formatted output. These concepts allow us to create simple programs that respond to data and make meaningful decisions.

16.Review Questions

Conceptual Questions

1. What is a Boolean value?
2. What is a Boolean expression?
3. What are the six comparison operators introduced in this chapter?
4. What is the difference between = and ==?
5. What is the purpose of an if statement?
6. When should if-else be used?
7. When should if-elif-else be used?
8. What is a nested conditional statement?

Code Questions

9. What will the following program display?

```
x = 12

if x > 10:
    print("A")
else:
    print("B")
```

10. What is wrong with this code?

```
if x = 10:
    print("Ten")
```

11. What is wrong with this code?

```
if x > 10:
    print("Ten")
```

12. What is wrong with this code?

```
value = input("Enter a value: ")

if value > 10:
    print("Large")
```

13. Rewrite the following condition using a logical operator:

```
if temperature >= 20:
    if temperature <= 30:
        print("Normal")
```

17.Key Terms

Term	Meaning
Boolean	A data type with two possible values: "True" or "False".
Boolean expression	An expression that evaluates to "True" or "False".
Relational operator	An operator used to compare values, such as "=", "!=", ">", "<", ">=", and "<=".
Condition	A Boolean expression used to determine whether a particular action should be performed.
Conditional statement	A statement that allows a program to make decisions based on a condition.
"if" statement	A conditional statement that executes a block of code when a condition is "True".
"else" statement	A statement used to test another condition when previous conditions are "False".
"elif" statement	A statement used to test another condition when previous conditions are "False".
Logical operator	An operator used to combine or modify Boolean expressions, such as "and", "or", and "not".
"and" operator	Returns "True" when both conditions are "True".
"or" operator	Returns "True" when at least one of the conditions is "True".
"not" operator	Reverses a Boolean value or Boolean expression.
Nested conditional	A conditional statement placed inside another conditional statement.
Indentation	The spaces at the beginning of a line that define a block of code in Python.
Comparison	The process of checking the relationship between two values.

18.References

1. **Python Software Foundation.** *Python 3 Documentation — The Python Tutorial and Python User Guide.*
2. **Severance, C. R.** *Python for Everybody: Exploring Data Using Python 3.*
1. **Downey, A. B.** *Think Python. How to Think Like a Computer Scientist. 2nd Edition,*
2. **Sweigart, A.** *Automate the Boring Stuff with Python.*