Chapter 3

Control Statements

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

Chapter 3 describes the different control statements available in Python. Students learn how to write `for` and `while` loops; students then learn to use these loops to repeat actions, traverse lists, count up and down, and generate random numbers. Selection statements are also covered, including one-way selection statements (`if`), two-way selection statements (`if-else`), and multi-way selection statements.

Objectives

After completing this chapter, students will be able to:

- Write a loop to repeat a sequence of actions a fixed number of times
- Write a loop to traverse the sequence of characters in a string
- Write a loop that counts down and a loop that counts up
- Write an entry-controlled loop that halts when a condition becomes false
- Use selection statements to make choices in a program
- Construct appropriate conditions for condition-controlled loops and selection statements
- Use logical operators to construct compound Boolean expressions
- Use a selection statement and a break statement to exit a loop that is not entry-controlled

Teaching Tips

Definite Iteration: The `for` Loop

1. Introduce the terms: *loop, pass/iteration, definite iteration*, and *indefinite iteration*.

Teaching Tip	For more information on Python loops, visit: http://en.wikibooks.org/wiki/Python_Programming/Loops.

Executing a Statement a Given Number of Times

1. Note that Python's *`for` loop* is the control statement that most easily supports definite iteration.

2. Describe the syntax of a `for` loop; identify its header and body and explain the structure of each. Stress that the statements in the loop body must be indented and aligned in the same column.

3. Use a few examples to show how to write a `for` loop.

Count-Controlled Loops

1. Use a few examples to show how to create count-controlled `for` loops.

2. Point out that the second (or only) argument of `range` should be greater by one than the desired upper bound of the count.

3. Give examples of how user input can be used to determine the for loop range.

Augmented Assignment

1. Explain the term *augmented assignment operations* and provide examples of such operations. Stress that the augmented assignment operators have the same precedence as the assignment operator.

Loop Errors: Off-by-One Error

1. Explain the term *off-by-one error*, pointing out that these are logic errors and not syntax errors. Note that this type of error is one of the most common types of errors in `for` loops. Use an example to illustrate this type of error.

Traversing the Contents of a Data Sequence

1. Explain that `for` loops can iterate over any sequence of elements, not only a numeric sequence. Provide a few examples to show how to write `for` loops that traverse list elements and characters in a string.

Specifying the Steps in the Range

1. Describe how a third variant of the `range` function can be used to specify a *step value*, and provide one or more examples to show how this kind of `for` loop is written.

Loops That Count Down

1. Use an example to note that one can provide a negative third argument to the `range` function in order to create loops that count down.

Formatting Text for Output

1. Use the examples provided in the book to show how to specify a *field width* and nicely display output in *tabular format*.

2. Describe the roles of *format strings* and the *format operator* (`%`), and explain how to use them to format strings, integers, and floating-point numbers.

Teaching Tip	You can find some useful information about string formatting operations in Python at: http://docs.python.org/lib/typesseq-strings.html, section 5.6.2.

Case Study: An Investment Report

1. Guide students as they step through this section.

Request

1. Explain that students are requested to write a program that computes an investment report.

Analysis

1. Provide an overview of the input the program should receive (as listed in the bullet points on Page 87) and the values the program should compute.

2. Use Figure 3.1 to describe the output that the program should generate. Can students figure out how will they use the format operator?

Design

1. Explain to students the four principal parts of the program.

2. Point out that the pseudocode for this algorithm is very simple—students should only look at the pseudocode provided in the book to check their own design.

3. Explain the term *prototype* as it applies to designing a program.

Implementation (Coding)

1. Review the code provided in the book with students. Point out the descriptive choices of variable names, as well as the end-of-line comments explaining the various computation stages.

2. Make sure students understand how the format operator (%) is used in the code.

Testing

1. Explain to students that when testing a program that contains a loop, they must first make sure that the number of iterations of the loop is correct.

2. Point out that the second phase of testing is to examine the effect of different inputs on the results, including the format of the results.

3. Show students the sample data set provided in Table 3.1, and explain why this is a good data set.

Quick Quiz 1

1. What is an iteration?
   Answer: In a loop, each repetition of the action is known as a pass or an iteration.

2. True or False: When two arguments are supplied to `range`, the count ranges from the first argument to the second argument minus 1.
   Answer: True

3. True or False: The third argument of the `range` function specifies the upper bound of the count.
   Answer: False

4. The total number of data characters and additional spaces for a given datum in a formatted string is called its ____________________.
   Answer: field width

Selection: `if` and `if-else` Statements

1. Explain that *selection statements* allow a computer to make choices based on a *condition*.

Teaching Tip	For more information on loops and conditionals in Python, read: www.sthurlow.com/python/lesson04/.

The Boolean Type, Comparisons, and Boolean Expressions

1. Explain that the *Boolean data type* consists of two values: true and false.

2. Point out that Boolean expressions can be created in a number of ways, including variables bound to Boolean values, function calls that return Boolean values, and comparisons.

3. Use Table 3.2 and a few examples to show how to use comparison operators to create Boolean expressions. Be sure to point out to students that to check whether two terms are equal you must use == and not =, which means assignment, and that use of the wrong operator is often a cause of logic errors in programs that contain comparisons.

4. Explain to students the order in which comparison operators are applied in complex expressions, pointing out the order of precedence of these operators.

`if-else` Statements

1. Describe the syntax of *`if-else` statements* and provide a few examples to demonstrate how to write *two-way selection statements*.

2. Use Figure 3.2 to describe the semantics of this statement, and stress that the condition must be a Boolean expression. Emphasize that each sequence of statements must be indented at least one space beyond the `if` and `else`.

One-Way Selection Statements

1. Use Figure 3.3 to describe the semantics of the `if` (*one-way selection*) statement.

2. Describe its syntax, and use a few examples to show how to use one-way selection statements.

Multi-Way `if` Statements

1. Use the grading example provided in the book (see Table 3.3) to show how to write *multi-way selection statements* in Python.

2. Explain the syntax of a multi-way `if` statement, emphasizing the use of the term `elif` when entering each alternate condition statement.

Logical Operators and Compound Boolean Expressions

1. Use the example provided in the book to show how to use *logical operators* to create *compound Boolean expressions*, which can then be used to write simpler selection statements.

2. Provide an overview of the *truth tables* for the `and`, `or`, and `not` logical operators, using Figure 3.4 as a guide.

3. Stress that the logical operators are evaluated after comparisons but before the assignment operator. Note that the `not` operator has higher precedence than `and` and `or` operators. Table 3.4 shows the operator precedence (from highest to lowest) of the arithmetic, logical, and assignment operators.

Short-Circuit Evaluation

1. Explain what *short-circuit evaluation* is, and use one or more examples to show how this feature can be useful for avoiding errors like division by zero.

Teaching Tip	For more information on short-circuit evaluation, read: http://www.freenetpages.co.uk/hp/alan.gauld/tutfctnl.htm.

Testing Selection Statements

1. Provide some tips on how to effectively test selection statements, including making sure to consider the program's behavior for all possible branches or alternatives of a selection and examining the conditions.

Quick Quiz 2

1. In Python, ____________________ literals can be written in several ways, but most programmers prefer the use of the standard values `True` and `False`.
   Answer: Boolean

2. What is a two-way selection statement?
   Answer: The `if-else` statement is the most common type of selection statement. It is also called a two-way selection statement, because it directs the computer to make a choice between two alternative courses of action.

3. True or False: Python includes all three Boolean or logical operators, `and`, `or`, and `xor`.
   Answer: False

4. True or False: There are times when short-circuit evaluation is advantageous.
   Answer: True

Conditional Iteration: The `while` Loop

1. Explain that the `while` loop can be used to describe conditional iteration. Introduce the term *sentinel*.

The Structure and Behavior of a `while` Loop

1. Describe the syntax of a `while` loop, identifying its *continuation condition*. Stress that improper setting of the continuation condition may lead to an infinite loop.

2. Use Figure 3.5 and an example to describe the semantics of a `while` loop. Introduce the term *loop control variable*.

3. Explain why a `while` loop is also called an *entry-control loop*.

Count Control with a `while` Loop

1. Use one or more examples to show how to create count control loops using a `while`.

2. Explain why using a while loop for a count controlled loop can potentially be a source of errors in loop logic.

The `while` True Loop and the `break` Statement

1. Use one or more examples to show how it is sometimes convenient to create `while True` loops in which the loop's *termination condition* is indicated using a selection statement within the loop.

2. Note that some computer scientists believe `while True` loops go against the nature of the `while` statement. Explain that an alternative to a `while True` loop is to use a Boolean variable (i.e., a flag) to control the loop.

Teaching Tip	For more information on the `break` statement, read: http://docs.python.org/ref/break.html.

Teaching Tip	Besides the `break` statement, `continue` statements are also useful in loops. For more information, visit: http://docs.python.org/ref/continue.html.

Random Numbers

1. Use the simple guessing game example provided in the book to show how to use the `random` module and the `randint` function to generate *random numbers*.

Teaching Tip	For more information on random number generators, visit: http://en.wikipedia.org/wiki/Random_number_generator.

Loop Logic, Errors, and Testing

1. Provide some tips on how to effectively test `while` loops.

2. Explain that the `Control+c` key combination can be used to halt a loop that appears to be infinite during testing.

Case Study: Approximating Square Roots

1. Guide students as they step through this section.

Request

1. Explain to students that they are asked to write a program that computes square roots.

Analysis

1. Describe the user interface of the program to be created.

Design

1. Spend some time explaining Newton's square root approximation algorithm.

Teaching Tip	For more information on Newton's and other square root approximation algorithms, visit: http://mathworld.wolfram.com/NewtonsMethod.html.

Implementation (Coding)

1. Show students the code for the program, going over each statement and paying specific attention to the `while` loop.

2. You may ask students to modify the program so that it counts how many iterations the loop performs. How does the number change if the tolerance is modified?

Testing

1. Explain the steps required to test this program: testing how the program functions for valid and invalid inputs, as well as using Python's own most accurate estimate of the square root provides a benchmark for assessing the correctness of our own algorithm.

Quick Quiz 3

1. What is conditional iteration?
   Answer: Conditional iteration requires that a condition be tested within the loop to determine whether the loop should continue.

2. The `while` loop is also called a(n) ______________________ loop, because its condition is tested at the top of the loop.
   Answer: entry-control

3. True or False: A `while` loop cannot be used for a count-controlled loop.
   Answer: False

4. A(n) ______________________ statement causes an exit from the loop that contains it.
   Answer: `break`

Class Discussion Topics

1. Are students familiar with the repeat-until loops available in some programming languages? If so, ask them to compare and contrast these types of loops with the `while` loop studied in this chapter.

2. Are students familiar with the `switch` (or `case`) selection statements available in some programming languages? If so, ask them if they think `switch` statements are more convenient than multi-way `if` statements (or the other way around).

Additional Projects

1. Ask your students to write a menu-driven program that calculates the total price for a picnic lunch that the user is purchasing for a group of friends. The user is first asked to enter her budget for the lunch. She has the option of buying apples, cheese, and bread. Set the price per apple, price per pound of cheese, and price per loaf of bread in constant variables. Use a nested repetition/selection structure to ask the user what type of item and how much of each item she would like to purchase. Keep a running total of the items purchased inside the loop. Exit the loop if the total has exceeded the user's budget. In addition, provide a sentinel value that allows the user to exit the purchasing loop at any time.

2. Ask students to write a program that uses a loop to calculate the *gcd* of two positive numbers using the Euclidean method. (http://mathworld.wolfram.com/EuclideanAlgorithm.html).

Additional Resources

1. Python Programming/Loops:
   http://en.wikibooks.org/wiki/Python_Programming/Loops

2. Python Tutorial: More Control Flow Tools:
   http://docs.python.org/tut/node6.html

3. A Beginner's Python Tutorial: Loops and Conditionals:
   www.sthurlow.com/python/lesson04/

4. The `break` Statement:
   http://docs.python.org/ref/break.html

Key Terms

- **augmented assignment:** An assignment operation that performs a designated operation, such as addition, before storing the result in a variable.

- **Boolean expression:** An expression whose value is either true or false. See also compound Boolean expression and simple Boolean expression.
- **compound Boolean expression:** Refers to the complete expression when logical connectives and negation are used to generate Boolean values. See also Boolean expression and simple Boolean expression.
- **continuation condition:** A Boolean expression that is checked to determine whether or not to continue iterating within a loop. If this expression is True, iteration continues.
- **control statement:** A statement that allows the computer to repeat or select an action.
- **count-controlled loop:** A loop that stops when a counter variable reaches a specified limit.
- **definite iteration:** The process of repeating a given action a preset number of times.
- **field width:** The number of columns used for the output of text
- **`for` loop:** A structured loop used to traverse a sequence.
- **format operator %:**
- **format string:** A special syntax within a string that allows the programmer to specify the number of columns within which data are placed in a string.
- **`if-else` statement:** A selection statement that allows a program to perform alternative actions based on a condition.
- **indefinite iteration:** The process of repeating a given action until a condition stops the repetition.
- **infinite loop:** A loop in which the controlling condition is not changed in such a manner to allow the loop to terminate.
- **Iteration:** See loop
- **logical operator:** Either of the logical connective operators `and`, `or`, or `not`.
- **loop(s):** A type of statement that repeatedly executes a set of statements. See also control statements.
- **loop body:** The action(s) performed on each iteration through a loop.
- **loop header:** Information at the beginning of a loop that includes the conditions for continuing the iteration process.
- **multi-way selection statement:** Code description of the process of testing several conditions and responding accordingly**.** See also extended if statement
- **off-by-one error:** Usually seen with loops, this error shows up as a result that is one less or one greater than the expected value.
- **prototype:** A trimmed down version of a class or software system that still functions and allows the programmer to study its essential features.
- **selection statement:** A control statement that selects some particular logical path based on the value of an expression. Also referred to as a conditional statement.
- **sentinel value (sentinel):** A special value that indicates the end of a set of data or of a process.
- **short-circuit evaluation:** The process by which a compound Boolean expression halts evaluation and returns the value of the first subexpression that evaluates to true, in the case of or, or false, in the case of and.
- **simple Boolean expression:** An expression in which two numbers or variable values are compared using a single relational operator. See also Boolean expression and compound Boolean expression.
- **step value:** The amount by which a counter is incremented or decremented in a count-controlled loop.
- **termination condition:** A Boolean expression that is checked to determine whether or not to stop iterating within a loop. If this expression is True, iteration stops.

- **truth table:** A means of listing all of the possible values of a Boolean expression.
- **`while` loop(s):** A pretest loop that examines a Boolean expression before causing a statement to be executed.