Chapter 4

Strings and Text Files

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

Chapter 4 provides an introduction to strings and text files in Python. Students learn how use and manipulate strings, including some of the most useful string methods. Next they learn to create, read from, and write to text files. Finally, they learn how to use library functions to access and navigate a file system.

Objectives

After completing this chapter, students will be able to:

- Access individual characters in a string
- Retrieve a substring from a string
- Search for a substring in a string
- Convert a string representation of a number from one base to another base
- Use string methods to manipulate strings
- Open a text file for output and write strings or numbers to the file
- Open a text file for input and read strings or numbers from the file
- Use library functions to access and navigate a file system

Teaching Tips

Accessing Characters and Substrings in Strings

1. Explain to students that in this section the internal structure of a string is examined. Make sure to introduce the term *substring*.

Teaching Tip	For a brief introduction to text processing in Python, read: www.ibm.com/developerworks/linux/library/l-python5.html.

The Structure of Strings

1. Explain that a string is a *data structure* consisting of zero or more characters. Use the example provided in the book and Figure 4.1 to describe the structure of a string in Python. Stress that the numbering of the string's characters starts at 0.

2. Stress that a string is an *immutable data structure*.

3. Note that the length of a string is the number of characters it contains. A string with no characters has a length of zero (0).

The Subscript Operator

1. Describe the syntax of the *subscript operator*. Introduce the term *index*. Note that the index is usually in the range of 0 and (length of the string – 1). Explain why you can use negative indexes in Python.

2. Provide a few examples so that students become familiar with how the subscript operator works with strings.

3. Provide an example to explain that the subscript operator is useful when working with a count controlled loop.

Slicing for Substrings

1. Use a few examples to show how Python's subscript operator can be used to obtain a substring through a process called *slicing*.

Testing for a Substring with the `in` Operator

1. Use a couple of examples to show how to use the `in` operator to test for a substring within a string.

Quick Quiz 1

1. What is a data structure?
   Answer: A data structure is a compound unit that consists of several smaller pieces of data.

2. True or False: The string is an immutable data structure.
   Answer: True

3. True or False: To extract a substring, the programmer places a semicolon (;) in the subscript.
   Answer: False

4. The operator ____________________ returns `True` if the target string is somewhere in the search string; otherwise, it returns `False`.
   Answer: `in`

Data Encryption

1. Explain that by using *sniffing software*, an attacker can easily observe data crossing a network. Stress that the problem is even more pervasive in wireless networks.

2. Note that *data encryption* can be used to protect information transmitted on networks. Briefly describe how encryption works, introducing some important terms like *keys*, *encrypt*, *decrypt*, *cipher text*, and *plain text*.

3. Explain how a *Caesar cipher* works, and use the code provided in the book to show how the encryption and decryption functionality can be implemented in Python.

4. Explain why a Caesar cipher is easy to break using modern computers. Provide a brief introduction to more complex encryption methods, like the *block cipher* described in the book.

Teaching Tip	For more information about the Caesar cipher, read: http://www.cs.trincoll.edu/~crypto/historical/substitution.html, and http://www.cs.trincoll.edu/~crypto/historical/caesar.html.

Strings and Number Systems

1. Explain that, because *binary numbers* can be long strings of 0s and 1s, computer scientists often use other number systems, such as *octal* (base eight) and *hexadecimal* (base 16) as shorthand for these numbers. Point out that the number system people use in their day-to-day life is the *decimal*, or base ten, number system.

2. Demonstrate the syntax for identifying the number system being used.

Teaching Tip	You can find an online number system conversion tool at: www.cstc.org/data/resources/60/convtop.html.

The Positional System for Representing Numbers

1. Use a few examples and Figure 4.2 to explain that in *positional notation*, a digit has a *positional value* that is determined by raising the base to the power of the position ($base^{position}$).

2. Explain how the quantity represented by a number is calculated by multiplying a digit by its positional value and adding the results. Explain why this doesn't work for base 11 or higher.

Converting Binary to Decimal

1. Use one or more examples to show students how to manually convert a binary number (*bit string*) to its decimal equivalent.

2. Use the code provided in the book to show how to implement binary-to-decimal conversion in Python.

Converting Decimal to Binary

1. Use the code provided in the book to show how to implement decimal-to-binary conversion in Python.

Conversion Shortcuts

1. Use Table 4.1 and one or more examples to describe a quicker way to compute the decimal value of a bit string.

Octal and Hexadecimal Numbers

1. Use Figure 4.3 to show how to convert from octal to binary. Explain that to convert binary to octal, you begin at the right and factor the bits into groups of three bits each. Point out that the reverse process is carried out to convert from binary to octal.

2. Explain the structure of a hexadecimal number. Point out that the quantities 10…15 are represented by the letters A…F.

3. Use Figure 4.4 to show how to convert from hexadecimal to binary. Explain that to convert binary to hex, you begin at the right and factor the bits into groups of four and look up the corresponding hex digits. Point out that the reverse process is carried out to convert from binary to hexadecimal.

String Methods

1. Explain that Python includes a set of string operations called *methods* that make tasks like counting the words in a single sentence easy.

2. Explain that a method is always called with a given data value called an *object*. Stress that a method knows about the internal state of the object with which it is called. Demonstrate the syntax of calling a method:
   ```
   <object_name>.<method_name>(<argument1>,
   <argument2>…<argumentN>)
   ```

3. Point out that methods can receive arguments and return a value.

4. Note that in Python, all data values are objects and every data type includes a set of methods to use with objects of that type.

5. Use one or more examples to show students how to use the string methods available in Python.

6. Provide a brief overview of each of the string methods listed in Table 4.2.

Teaching Tip	For a complete list of the string methods available in Python, visit: http://docs.python.org/lib/string-methods.html.

Quick Quiz 2

1. What is positional notation?
   Answer: All of the number systems we have examined use positional notation—that is, the value of each digit in a number is determined by the digit's position in the number. In other words, each digit has a positional value. The positional value of a digit is determined by raising the base of the system to the power specified by the position ($base^{position}$)

2. True or False: The octal system uses a base of 8 and the digits 0…7.
   Answer: True

3. True or False: Each digit in the hexadecimal number is equivalent to three digits in the binary number.
   Answer: False

4. Unlike a function, a method is always called with a given data value called a(n) ____________________, which is placed before the method name in the call.
   Answer: object

Text Files

1. Explain that a text file is a software object that stores data on a permanent medium such as a disk or CD.

2. Briefly describe the advantages of using text files for input in a program (versus interactive human input).

Text Files and Their Format

1. Show an example of a text file, and note that you can easily create one in a text editor.

2. Stress that all data output to or input from a text file must be strings.

Writing Text to a File

1. Use an example to show how to write to a text file using a `file` object and the `open`, `write`, and `close` methods. Stress that failure to close an output file can result in data being lost.

2. Explain what happens if you write to a file that does not previously exist, or if you write to an existing file.

Teaching Tip	For more information on `file` objects, read: http://docs.python.org/lib/bltin-file-objects.html.

Writing Numbers to a File

1. Note that the `file` method `write` expects a string as an argument. For this reason, other types of data must be converted to strings before being written to the output file.

2. Use the example provided in the book to show how to use `str` to convert a number to a string before writing it to a file.

Reading Text from a File

1. Use an example to show how to read from a text file using a `file` object and the `open` and `read` methods. Point out that if the specified pathname is not accessible, Python raises an error.

2. Stress that after input is finished, `read` returns an empty string.

3. Explain that in order to re-read a file it must be reopened.

4. Introduce the `readline` method, and explain how it can be used to read a specific number of lines from a file.

5. Demonstrate how to use `for` and `while` loops to read the contents of a file.

Reading Numbers from a File

1. Use the examples provided in the book to show how to read numbers from a text file. Point out that you must know the type of white spaces used in order to efficiently process the numbers read from a file.

2. Use Table 4.3 to briefly review the `file` methods studied in this section.

Accessing and Manipulating Files and Directories on Disk

1. Explain why it is important to include error recovery in Python programs that interact with files.

2. Use Tables 4.4 and 4.5 and the code snippets provided in the book to show how to use some system functions that are useful when working with files and directories on disk.

Case Study: Text Analysis

1. Explain the *Flesch Index* and what it measures.

Request

1. Explain to students that they are asked to write a program that computes the Flesch index and grade level for text stored in a text file.

Analysis

1. Use Table 4.6 to provide an overview of the items used in the text-analysis program and their definitions.

Design

1. Use Table 4.7 to briefly describe each of the tasks in the text analysis program.

Implementation (Coding)

1. Ask students whether they made any mistakes while typing the code in their computers. Were their mistakes syntax or logic errors?

Testing

1. Explain that *bottom-up* testing can be used to test the text-analysis program. Introduce the term *driver*.

Teaching Tip	Bottom-up testing: "An integration testing technique that tests the low-level components first using test drivers for those components that have not yet been developed to call the low-level components for test." For more information, read: http://foldoc.org/?bottom-up+testing.

Quick Quiz 3

1. Data can be output to a text file using a(n) ____________________ object.
   Answer: `file`

2. True or False: You can close a file with the `append` method.
   Answer: False

3. True or False: The `file` method `write` expects a string as an argument.
   Answer: True

4. After input is finished, another call to `read` would return a(n) ____________________ to indicate that the end of the file has been reached.

Answer: empty string

Class Discussion Topics

1. Are students familiar with string manipulation functions in other programming languages? If so, ask them to compare how strings are manipulated in one or more other languages with respect to how they are manipulated in Python.

2. Ask students to brainstorm for situations in which they would need to use each of the functions listed in Tables 4.4 and 4.5 for file/directory manipulation.

Additional Projects

1. Ask students to do some research on other string methods available in Python. They should compile a list of at least five other useful methods that were not studied in this chapter.

2. Ask students to do some research on how brute force attacks and frequency analysis attacks can be used to break a Caesar cipher. Have them summarize their findings in a one-page write-up.

Additional Resources

1. String Methods:
   http://docs.python.org/lib/string-methods.html

2. Python String Methods:
   http://python.about.com/od/pythonstandardlibrary/a/string-methods.htm

3. Charming Python: Text Processing in Python:
   www.ibm.com/developerworks/linux/library/l-python5.html

4. File Objects:
   http://docs.python.org/lib/bltin-file-objects.html

Key Terms

- **binary digit:** A digit, either 0 or 1, in the binary number system. Program instructions are stored in memory using a sequence of binary digits. See also bit.
- **bit:** A binary digit.
- **block cipher:** An encryption method that replaces characters with other characters located in a two-dimensional grid of characters.

- **bottom-up testing:** A method of coding and testing a program that starts with lower-level modules and a test driver module.
- **Caesar cipher:** An encryption method that replaces characters with other characters a given distance away in the character set.
- **data decryption:** The process of translating encrypted data to a form that can be used.
- **data encryption:** The process of transforming data so that others cannot use it.
- **data structure:** A compound unit consisting of several data values.
- **driver:** A function used to test other functions.
- **encryption:** The process of transforming data so that others cannot use it.
- **file:** A data structure that resides on a secondary storage medium accessing/manipulating.
- **immutable data structure:** a data structure whose internal data or state cannot be changed
- **index:** The relative position of a component of a linear data structure or collection.
- **key(s):** An item that is associated with a value and which is used to locate that value in a collection.
- **method(s):** A chunk of code that can be treated as a unit and invoked by name. A method is called with an object or class.
- **object:** A collection of data and operations, in which the data can be accessed and modified only by means of the operations.
- **positional notation:** The type of representation used in based number systems, in which the position of each digit denotes a power in the system's base.
- **slicing:** An operation that returns a subsection of a sequence, for example, a sublist or a substring.
- **substring:** A string that represents a segment of another string.
- **text files:** Files that contain characters and are readable and writable by text editors.