Chapter 5

Lists and Dictionaries

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

Chapter 5 describes lists and dictionaries. Students learn to construct, use, and manipulate lists. They also learn to define simple functions that expect parameters and return values. Finally, they learn to construct, use, and manipulate dictionaries.

Objectives

After completing this chapter, students will be able to:

- Construct lists and access items in those lists
- Use methods to manipulate lists
- Perform traversals of lists to process items in the lists
- Define simple functions that expect parameters and return values
- Construct dictionaries and access entries in those dictionaries
- Use methods to manipulate dictionaries
- Decide whether a list or a dictionary is an appropriate data structure for a given application

Teaching Tips

Lists

1. Introduce the terms: *list* and *item*/*element*.
2. Provide some examples of real-life lists. You can use the bullet points on Page 160 in the text as a guide.
3. Explain that each item in a list has a unique *index* that specifies its position (from 0 to length – 1).

Teaching Tip	For more on Python lists, visit: http://diveintopython.org/native_data_types/lists.html.

List Literals and Basic Operators

1. Provide a few examples to show students what list literals look like. Explain that when an element is an expression or variable, its value is included in the list (not the expression itself).

2. Use a couple of examples to show how to create lists using the `range` and `list` functions.

3. Use several examples to show how the `len` function and the subscript, concatenation (+), and equality (==) operators work with lists.

4. Show how to print lists, both by printing the whole list at once and by iterating through the list in a loop that prints each element individually.

5. Use one or more examples to show how the `in` operator can be used to detect the presence or absence of an element in a list.

6. Use Table 5.1 to review the operators and functions covered in this section.

Replacing an Element in a List

1. Explain that a list is *mutable*, which means that the list itself maintains its identity but its *state*—its length and its contents—can change.

2. Use the examples provided in the book to show how to use the subscript operator to replace an element in a list. Stress that the subscript is used to reference the target of the assignment – the position of the replaced element within the list.

3. Explain that a sublist of elements may be replaced by using the slice operator, and demonstrate using the example on Page 165 of the text.

List Methods for Inserting and Removing Elements

1. Use Table 5.2 to explain that the `list` type includes several methods for inserting and removing elements. Use the examples provided in the book to show how to use these methods.

2. Stress the difference between the list `extend` and `append` methods.

Teaching Tip	In Python, lists can be used as stacks and as queues. For more information, read: http://docs.python.org/tut/node7.html.

Searching a List

1. Remind students that the `in` operator can be used to determine whether or not an element is present in a list.

2. Provide an example to illustrate how to use the method `index` to locate an element's position in a list. Be sure to point out that because the `index` method raises an error if the element is not found in the list, it is good practice to use the `in` operator to check whether an element is present prior to using the `index` method to locate that element.

Sorting a List

1. Provide an example to illustrate how to use the `sort` method to arrange the elements of a list in ascending order.

Mutator Methods and the Value `None`

1. Explain that *mutator* methods usually do not return a value. Use an example to show that in these cases, Python automatically returns the special value `None`.

Aliasing and Side Effects

1. Use the example provided on Page 169 of the book and Figure 5.1 to show how the mutable property of lists leads to interesting phenomena. Make sure students understand what *side effect* and *aliasing* are in the context of lists.

2. Use the example provided on Page 170 of the book and Figure 5.2 to show that aliasing can be avoided by copying the contents of an object instead of copying the reference to the object. Show how the slicing operator can be used to simplify this process.

Equality: Object Identity and Structural Equivalence

1. Use the example provided on Page 171 of the book and Figure 5.3 to define the terms *identity* and *structural equivalence* as they apply to variables in Python. Show that the equality operator (`==`) tests for structural equivalence, not for object identity. Show how to use the `is` operator to test for object identity.

Example: Using a List to Find the Median of a Set of Numbers

1. Use the example provided in the book to show how to use a list to find the *median* of a set of numbers. Make sure that students use this example to understand how different methods can be applied to lists in order to achieve a desired result.

Tuples

1. Explain that a *tuple* resembles a list, but it is immutable. Provide a few examples to illustrate how to create and use tuples, stressing the syntax differences between creating lists and creating tuples.

2. Note that most of the operators and functions used with lists can be used in a similar fashion with tuples.

Teaching Tip	"Assigning a tuple to a several different variables is called *tuple unpacking*, while assigning multiple values to a tuple in one variable is called *tuple packing*. When unpacking a tuple, or performing multiple assignment, you must have the same number of variables being assigned to as values being assigned." *Reference:* http://en.wikibooks.org/wiki/Python_Programming/Tuples.

Quick Quiz 1

1. What is a list in Python?
   Answer: In Python, a list is written as a sequence of data values separated by commas. The entire sequence is enclosed in square brackets (`[` and `]`).

2. True or False: The `is` operator can be used to detect the presence or absence of a given element in a list.
   Answer: False

3. True or False: A list is immutable.
   Answer: False

4. What is a tuple?
   Answer: A tuple is a type of sequence that resembles a list—although unlike a list, a tuple is immutable. You indicate a tuple literal in Python by enclosing its elements in parentheses instead of square brackets.

Defining Simple Functions

1. Explain that defining functions allows programmers to organize their code in existing scripts more effectively.

The Syntax of Simple Function Definitions

1. Describe the syntax of a simple function definition. Use the `square(x)` example provided in the book to show how to define and use a simple value-returning function.

2. Stress that it is important to provide a docstring that contains information about what the function does.

3. Note that a function can be defined in a Python shell, but that it is more convenient to define it in an IDLE window.

4. Point out that a function must be defined in a script before it is called in that script.

Parameters and Arguments

1. Explain the difference between parameters and arguments.

Teaching Tip	Python allows the programmer to provide default parameters in the header of a function definition. For more information, read: http://docs.python.org/ref/function.html.

The `return` Statement

1. Explain that you should place a `return` statement at each exit point of a function when the function should explicitly return a value. Describe the syntax of this statement. Point out that a function may include more than one `return` statement, for instance when an `if-else` statement is used in the function.

2. Stress that if a function contains no `return` statement, Python transfers control to the caller after the last statement in the function's body is executed and the special value `None` is automatically returned.

Boolean Functions

1. Explain that a *Boolean function* usually tests its argument for the presence or absence of some property.

2. Use the example provided in the book to show how to define and use a Boolean function.

Defining a `main` Function

1. Describe the role of the `main` function. Stress that the definition of `main` and other functions can appear in no particular order in the script, as long as `main` is called at the end of the script.

2. Use an example to illustrate how to define and use a `main` function.

Case Study: Generating Sentences

1. Guide students as they step through this section.

Request

1. Ask students to write a program that generates sentences.

Analysis

1. Use Table 5.3 to explain that the program created in this section will generate sentences from a simplified subset of English.

Design

1. Explain why it is important to assign the task of generating each phrase to a separate function. Briefly describe the role of the `sentence`, `nounPhrase`, and `main` functions.

Implementation (Coding)

1. Stress that the variables for the data should be initialized just below the `import` statement.

Testing

1. Describe the difference between the bottom-up and top-down testing approaches. Note that a wise programmer can mix bottom-up and top-down testing as needed.

Quick Quiz 2

1. Why is it useful to define your own functions?
   Answer: Some scripts might be useful enough to package as functions that can be used in other scripts. Moreover, defining your own functions allows you to organize your code in existing scripts more effectively.

2. True or False: The function's header contains the function's name and a parenthesized list of argument names.
   Answer: True

3. The header of a function consists of the reserved word ____________________, followed by the function's name, followed by a parenthesized list of parameters, followed by a colon.
   Answer: `def`

4. A(n) ____________________ function usually tests its argument for the presence or absence of some property.
   Answer: Boolean

Dictionaries

1. Use an example to explain that a dictionary organizes information by *association*, not by position. Note that data structures organized by association are also called *tables* or *association lists*.

2. Explain that in Python, a *dictionary* associates a set of *keys* with data values.

Teaching Tip	Here are some useful resources with more information about Python dictionaries: • http://diveintopython.org/getting_to_know_python/dictionaries.html • www.developer.com/article.php/630721 • http://en.wikibooks.org/wiki/Python_Programming/Dictionaries

Dictionary Literals

1. Provide several examples to show how to create dictionary literals in Python. Note that `{}` is an empty dictionary.

2. Explain that in a dictionary literal, the keys can be data of any immutable type, including other data structures.

Adding Keys and Replacing Values

1. Use a few examples to show how to use the subscript operator (`[]`) to add a new key/value pair to a dictionary, as well as to replace a value at an existing key.

Accessing Values

1. Use a few examples to show how to use the subscript operator (`[]`) to obtain the value associated with a key.

2. Explain that if the existence of a key is uncertain, you can test for it using the dictionary method `has_key`. Use an example to show that an easier strategy is to use the method `get`, and explain the syntax of this method.

Removing Keys

1. Provide a couple of examples to show how you can use the method `pop` to delete an entry from a dictionary.

2. Explain what happens if the key provided to the `pop` method is absent from the dictionary when the `pop` method is used with one argument. Next explain when it is used with two arguments.

Traversing a Dictionary

1. Provide examples to show how to traverse a dictionary using a loop and the method `items` (which returns a list of tuples).

2. Explain how the method `keys` can be used to help generate a specific order for traversing a dictionary.

3. Use Table 5.4 to briefly describe the dictionary operations studied in this chapter.

Example: The Hexadecimal System Revisited

1. Use the example provided in the book to show how to use a dictionary to create a hex-to-binary *lookup table* to aid in the conversion process.

Example: Finding the Mode of a List of Values

1. Use the code provided in the book to show how to use a dictionary to find the *mode* of a list of values.

Teaching Tip	Besides lists, tuples, and dictionaries, Python also includes a data type for sets. For more information, read: http://docs.python.org/tut/node7.html#SECTION007400000000000000000.

Case Study: Nondirective Psychotherapy

1. Guide students as they step through this section.

Teaching Tip	Nondirective psychotherapy is also called *client-centered* or *person-centered psychotherapy*. For more information, view the Britannica Online Encyclopedia entry on the subject: http://www.britannica.com/EBchecked/topic/417641/nondirective-psychotherapy.

Request

1. Ask students to write a program that emulates a nondirective psychotherapist.

Analysis

1. Use Figure 5.4 to describe the user interface of the program.

Design

1. Explain why it is a good idea to create a set of collaborating functions that share a common data pool.

Implementation (Coding)

1. Ask students to enter and run the `doctor.py` script.

Testing

1. Describe a few situations in which the program does not work as expected. Note that, with a little work, you can make the replies more realistic.

Quick Quiz 3

1. What is a dictionary?
   Answer: A dictionary organizes information by association, not position. In computer science, data structures organized by association are also called tables or association lists. In Python, a dictionary associates a set of keys with data values.

2. True or False: A dictionary associates a set of indexes with data values.
   Answer: False

3. True or False: `[]` is an empty dictionary.
   Answer: False

4. If the existence of a dictionary key is uncertain, the programmer can test for it using the dictionary method `has_key`. However, an easier strategy is to use the method ____________________.
   Answer: `get`

Class Discussion Topics

1. Ask your students to brainstorm for three situations in which they would use a list, three situations in which they would use a tuple, and three situations in which they would use a dictionary.

2. If your students are familiar with other programming languages, ask them if those languages provide data structures with functionality similar to that of Python's dictionaries.

Teaching Tip	A dictionary is like a *hash* in Perl, a *Hashtable* in Java, and an instance of the *Scripting.Dictionary* object in Visual Basic.

Additional Projects

1. In addition to lists, tuples, and dictionaries, Python also includes a data type for sets. Ask students to do some research to learn about this data structure and how to use it.

2. Lists can be used as stacks and queues. Ask students to write scripts that use a list as a stack and as a queue, respectively. Tip: you may ask them to read the "Data Structures" section of the Python Tutorial before working in this exercise. (http://docs.python.org/tut/node7.html).

Additional Resources

1. Python Tutorial: Data Structures:
   http://docs.python.org/tut/node7.html

2. Dive Into Python: Introducing Lists:
   http://diveintopython.org/native_data_types/lists.html

3. Python Programming/Tuples:
   http://en.wikibooks.org/wiki/Python_Programming/Tuples

4. Python Tutorial: Function Definitions:
   http://docs.python.org/ref/function.html

5. Python Programming/Dictionaries:
   http://en.wikibooks.org/wiki/Python_Programming/Dictionaries

Key Terms

- **alias:** A situation in which two or more names in a program can refer to the same memory location. An alias can cause subtle side effects.
- **association:** A pair of items consisting of a key and a value.
- **dictionary:** A data structure that allows the programmer to access items by specifying key values.
- **element:** A value that is stored in an array or a collection. Also known as item.
- **function(s):** A chunk of code that can be treated as a unit and called to perform a task.
- **function heading:** The portion of a function implementation containing the function's name and parameter names.
- **grammar:** The set of rules for constructing sentences in a language.
- **identity:** The property of an object that it is the same thing at different points in time, even though the values of its attributes might change.
- **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.
- **logical structure:** The organization of the components in a data structure, independent of their organization in computer memory.
- **mutator:** A method used to change the value of an attribute of an object.
- **natural ordering:** The placement of data items relative to each other by some internal criteria, such as numeric value or alphabetical value.
- **`None` value:** A special value that indicates that no object can be accessed.

- **object identity:** The property of an object that it is the same thing at different points in time, even though the values of its attributes might change.
- **side effect:** A change in a variable that is the result of some action taken in a program, usually from within a method.
- **state:** The set of all the values of the variables of a program at any point during its execution.
- **structural equivalence:** A criterion of equality between two distinct objects in which one or more of their attributes are equal.
- **tuple:** A linear, immutable collection.
- **value:** An item that is associated with a key and is located by a key in a collection.