Dictionary Exercises 6

Dictionaries are another data structure that Python programmers can use to manage larger amounts of data. While many of the exercises in this chapter can be solved with lists or if statements, most (or even all) of them have solutions that are well suited to dictionaries. As a result, you should use dictionaries to solve all of these exercises instead of (or in addition to) using the constructs that you have been introduced to in the previous chapters. Completing the exercises in this chapter will help you learn to:

- Create a new variable that holds a dictionary
- Add a key-value pair to a dictionary
- Update the value associated with a key in a dictionary
- Iterate over all of the keys and/or values in a dictionary
- Write functions that take dictionaries as parameters

Exercise 128: Reverse Lookup

(*Solved—40 Lines*)

Write a function named `reverseLookup` that finds all of the keys in a dictionary that map to a specific value. The function will take the dictionary and the value to search for as its only parameters. It will return a (possibly empty) list of keys from the dictionary that map to the provided value.

Include a main program that demonstrates the `reverseLookup` function as part of your solution to this exercise. Your program should create a dictionary and then show that the `reverseLookup` function works correctly when it returns multiple keys, a single key, and no keys. Ensure that your main program only runs when the file containing your solution to this exercise has not been imported into another program.

© Springer International Publishing Switzerland 2014

B. Stephenson, *The Python Workbook*, DOI 10.1007/978-3-319-14240-1_6

Exercise 129: Two Dice Simulation

(*Solved—42 Lines*)

In this exercise you will simulate 1,000 rolls of two dice. Begin by writing a function that simulates rolling a pair of six-sided dice. Your function will not take any parameters. It will return the total that was rolled on two dice as its only result.

Write a main program that uses your function to simulate rolling two six-sided dice 1,000 times. As your program runs, it should count the number of times that each total occurs. Then it should display a table that summarizes this data. Express the frequency for each total as a percentage of the total number of rolls. Your program should also display the percentage expected by probability theory for each total. Sample output is shown below.

```
Total    Simulated    Expected
           Percent     Percent
    2         2.90        2.78
    3         6.90        5.56
    4         9.40        8.33
    5        11.90       11.11
    6        14.20       13.89
    7        14.20       16.67
    8        15.00       13.89
    9        10.50       11.11
   10         7.90        8.33
   11         4.50        5.56
   12         2.60        2.78
```

Exercise 130: Text Messaging

(*21 Lines*)

On some basic cell phones, text messages can be sent using the numeric keypad. Because each key has multiple letters associated with it, multiple key presses are needed for most letters. Pressing the number once generates the first letter on the key. Pressing the number 2, 3, 4 or 5 times generates the second, third, fourth or fifth character listed for that key.

Key	Symbols
1	. , ? ! :
2	A B C
3	D E F
4	G H I

5	J K L
6	M N O
7	P Q R S
8	T U V
9	W X Y Z
0	*space*

Write a program that displays the key presses that must be made to enter a text message read from the user. Construct a dictionary that maps from each letter or symbol to the key presses. Then use the dictionary to generate and display the presses for the user's message. For example, if the user enters `Hello, World!` then your program should output `4433555555666110966677755531111`. Ensure that your program handles both uppercase and lowercase letters. Ignore any characters that aren't listed in the table above such as semicolons and brackets.

Exercise 131: Morse Code

(*15 Lines*)

Morse code is an encoding scheme that uses dashes and dots to represent numbers and letters. In this exercise, you will write a program that uses a dictionary to store the mapping from letters and numbers to Morse code. Use a period to represent a dot, and a hyphen to represent a dash. The mapping from letters and numbers to dashes and dots is shown in Table 6.1.

Your program should read a message from the user. Then it should translate each letter and number in the message to Morse code, leaving a space between each sequence of dashes and dots. Your program should ignore any characters that are not letters or numbers. The Morse code for `Hello, World!` is shown below:

```
.... . .-.. .-.. --- .-- --- .-. .-.. -..
```

Table 6.1 Morse Code Letters and Numbers

Letter	Code	Letter	Code	Letter	Code	Number	Code
A	.-	J	.---	S	...	1	.----
B	-...	K	-.-	T	-	2	..---
C	-.-.	L	.-..	U	..-	3	...--
D	-..	M	--	V	...-	4	....-
E	.	N	-.	W	.--	5	.....
F	..-.	O	---	X	-..-	6	-....
G	--.	P	.--.	Y	-.--	7	--...
H	....	Q	--.-	Z	--..	8	---..
I	..	R	.-.	0	-----	9	----.

Morse code was originally developed in the nineteenth century for use over telegraph wires. It is still used today, over 160 years after it was first created.

Exercise 132: Postal Codes

(*24 Lines*)

In a Canadian postal code, the first, third and fifth characters are letters while the second, fourth and sixth characters are numbers. The province can be determined from the first character of a postal code, as shown in the following table. No valid postal codes currently begin with D, F, I, O, Q, U, W, or Z.

Province	First character(s)
Newfoundland	A
Nova Scotia	B
Prince Edward Island	C
New Brunswick	E
Quebec	G, H and J
Ontario	K, L, M, N and P
Manitoba	R
Saskatchewan	S
Alberta	T
British Columbia	V
Nunavut	X
Northwest Territories	X
Yukon	Y

The second character in a postal code identifies whether the address is rural or urban. If that character is a 0 then the address is rural. Otherwise it is urban.

Create a program that reads a postal code from the user and displays the province associated with it, along with whether the address is urban or rural. For example, if the user enters `T2N 1N4` then your program should indicate that the postal code is for an urban address in Alberta. If the user enters `X0A 1B2` then your program should indicate that the postal code is for a rural address in Nunavut or Northwest Territories. Use a dictionary to map from the first character of the postal code to the province name. Display a meaningful error message if the postal code begins with an invalid character.

Exercise 133: Write Out Numbers in English

(*65 Lines*)

While the popularity of cheques as a payment method has diminished in recent years, some companies still issue them to pay employees or vendors. The amount being paid normally appears on a cheque twice, with one occurrence written using digits, and the other occurrence written using English words. Repeating the amount in two different forms makes it much more difficult for an unscrupulous employee or vendor to modify the amount on the cheque before depositing it.

In this exercise, your task is to create a function that takes an integer between 0 and 999 as its only parameter, and returns a string containing the English words for that number. For example, if the parameter to the function is 142 then your function should return "`one hundred forty two`". Use one or more dictionaries to implement your solution rather than large if/elif/else constructs. Include a main program that reads an integer from the user and displays its value in English words.

Exercise 134: Unique Characters

(*Solved—14 Lines*)

Create a program that determines and displays the number of unique characters in a string entered by the user. For example, `Hello, World!` has 10 unique characters while `zzz` has only one unique character. Use a dictionary or set to solve this problem.

Exercise 135: Anagrams

(*Solved—39 Lines*)

Two words are anagrams if they contain all of the same letters, but in a different order. For example, "evil" and "live" are anagrams because each contains one e, one i, one l, and one v. Create a program that reads two strings from the user, determines whether or not they are anagrams, and reports the result.

Exercise 136: Anagrams Again

(*48 Lines*)

The notion of anagrams can be extended to multiple words. For example, "William Shakespeare" and "I am a weakish speller" are anagrams when capitalization and spacing are ignored.

Extend your program from Exercise 135 so that it is able to check if two phrases are anagrams. Your program should ignore capitalization, punctuation marks and spacing when making the determination.

Exercise 137: Scrabble™ Score

(*Solved—18 Lines*)

In the game of Scrabble™, each letter has points associated with it. The total score of a word is the sum of the scores of its letters. More common letters are worth fewer points while less common letters are worth more points. The points associated with each letter are shown below:

One point	A, E, I, L, N, O, R, S, T and U
Two points	D and G
Three points	B, C, M and P
Four points	F, H, V, W and Y
Five points	K
Eight points	J and X
Ten points	Q and Z

Write a program that computes and displays the Scrabble™ score for a word. Create a dictionary that maps from letters to point values. Then use the dictionary to compute the score.

A Scrabble™ board includes some squares that multiply the value of a letter or the value of an entire word. We will ignore these squares in this exercise.

Exercise 138: Create a Bingo Card

(*Solved—58 Lines*)

A Bingo card consists of 5 columns of 5 numbers. The columns are labeled with the letters `B`, `I`, `N`, `G` and `O`. There are 15 numbers that can appear under each letter. In particular, the numbers that can appear under the `B` range from 1 to 15, the numbers that can appear under the `I` range from 16 to 30, the numbers that can appear under the `N` range from 31 to 45, and so on.

Write a function that creates a random Bingo card and stores it in a dictionary. The keys will be the letters `B`, `I`, `N`, `G` and `O`. The values will be the lists of five numbers that appear under each letter. Write a second function that displays the Bingo card with the columns labeled appropriately. Use these functions to write a program that displays a random Bingo card. Ensure that the main program only runs when the file containing your solution has not been imported into another program.

You may be aware that Bingo cards often have a "free" space in the middle of the card. We won't consider the free space in this exercise.

Exercise 139: Checking for a Winning Card

(*102 Lines*)

A winning Bingo card contains a line of 5 numbers that have all been called. Players normally mark the numbers that have been called by crossing them out or marking them with a Bingo dauber. In our implementation we will mark that a number has been called by replacing it with a 0 in the Bingo card dictionary.

Write a function that takes a dictionary representing a Bingo card as its only parameter. If the card contains a line of five zeros (vertical, horizontal or diagonal) then your function should return `True`, indicating that the card has won. Otherwise the function should return `False`.

Create a main program that demonstrates your function by creating several Bingo cards, displaying them, and indicating whether or not they contain a winning line. You should demonstrate your function with at least one card with a horizontal line, at least one card with a vertical line, at least one card with a diagonal line, and at least one card that has some numbers crossed out but does not contain a winning line. You will probably want to import your solution to Exercise 138 when completing this exercise.

Hint: Because there are no negative numbers on a Bingo card, finding a line of 5 zeros is the same problem as finding a line of 5 entries that sum to zero. You may find the summation problem easier to solve.

Exercise 140: Play Bingo

(*88 Lines*)

In this exercise you will write a program that simulates a game of Bingo for a single card. Begin by generating a list of all of the valid Bingo calls (B1 through O75). Once the list has been created you can randomize the order of its elements by calling the `shuffle` function in the `random` module. Then your program should consume calls out of the list, crossing out numbers on the card, until the card contains a crossed out line (horizontal, vertical or diagonal). Simulate 1,000 games and report the minimum, maximum and average number of calls that must be made before the card wins. Import your solutions to Exercises 138 and 139 when completing this exercise.