Check Point

MyProgrammingLab™

14.18 How do you create an empty dictionary?

14.19 Which of the following dictionaries are created correctly?

```
d = {1:[1, 2], 3:[3, 4]}
d = {[1, 2]:1, [3, 4]:3}
d = {(1, 2):1, (3, 4):3}
d = {1:"john", 3:"peter"}
d = {"john":1, "peter":3}
```

14.20 Each item in a dictionary has two parts. What are they called?

14.21 Suppose a dictionary named `students` is `{"john":3, "peter":2}`. What do the following statements do?

(a) `students["susan"] = 5`

(b) `students["peter"] = 5`

(c) `students["peter"] += 5`

(d) `del students["peter"]`

14.22 Suppose a dictionary named `students` is `{"john":3, "peter":2}`. What do the following statements do?

(a) `print(len(students))`

(b) `print(students.keys())`

(c) `print(students.values())`

(d) `print(students.items())`

14.23 Show the output of the following code:

```
def main():
    d = {"red":4, "blue":1, "green":14, "yellow":2}
    print(d["red"])
    print(list(d.keys()))
    print(list(d.values()))
    print("blue" in d)
    print("purple" in d)
    d["blue"] += 10
    print(d["blue"])

main() # Call the main function
```

14.24 Show the output of the following code:

```
def main():
    d = {}
    d["susan"] = 50
    d["jim"] = 45
    d["joan"] = 54
    d["susan"] = 51
    d["john"] = 53
    print(len(d))

main() # Call the main function
```

14.25 For a dictionary `d`, you can use `d[key]` or `d.get(key)` to return the value for the key. What are the differences between them?

14.7 Case Study: Occurrences of Words

Key Point

This case study writes a program that counts the occurrences of words in a text file and displays ten most frequently used words in decreasing order of their occurrence counts.

The program in this case study uses a dictionary to store an item consisting of a word and its count. The program determines whether each word is already a key in the dictionary. If not, the program adds a dictionary item with the word as the key and the value 1. Otherwise, the program increases the value for the word (key) by 1 in the dictionary. Assume the words are case-insensitive (for example, Good is treated the same as good). The program displays the ten most frequently used words in the file in decreasing order of their count.

Listing 14.5 shows the solution to the problem.

LISTING 14.5 CountOccurrenceOfWords.py

```
1  def main():
2      # Prompt the user to enter a file
3      filename = input("Enter a filename: ").strip()                    enter a file
4      infile = open(filename, "r") # Open the file                      open file
5
6      wordCounts = {} # Create an empty dictionary to count words       create a dictionary
7      for line in infile:
8          processLine(line.lower(), wordCounts)                         process each line
9
10     pairs = list(wordCounts.items()) # Get pairs from the dictionary  get pairs to list
11
12     items = [[x, y] for (y, x) in pairs] # Reverse pairs in the list  reverse pair
13
14     items.sort() # Sort pairs in items
15
16     for i in range(len(items) - 1, len(items) - 11, -1):
17         print(items[i][1] + "\t" + str(items[i][0]))                   display item
18
19 # Count each word in the line
20 def processLine(line, wordCounts):
21     line = replacePunctuations(line) # Replace punctuation with space  replace punctuation
22     words = line.split() # Get words from each line                   extract words
23     for word in words:
24         if word in wordCounts:
25             wordCounts[word] += 1                                     increase word count
26         else:
27             wordCounts[word] = 1                                      new word
28
29 # Replace punctuation in the line with a space
30 def replacePunctuations(line):
31     for ch in line:
32         if ch in "~@#$%^&*()_-+=~<>?/,.;:!{}[]|'\"":
33             line = line.replace(ch, " ")                              replace punctuation
34
35     return line
36
37 main() # Call the main function
```

```
Enter a filename: Lincoln.txt  Enter
that    13
the     11
we      10
to      8
here    8
a       7
and     6
```

```
of      5
nation  5
it      5
```

The program prompts the user to enter a filename (line 3) and opens the file (line 4). It creates a dictionary `wordCounts` (line 6) to store pairs of words and their occurrence counts. The words serve as the keys.

The program reads each line from the file and invokes `processLine(line, wordCounts)` to count the occurrence of each word in the line (lines 7–8). Suppose the `wordCounts` dictionary is `{"red":7, "blue":5, "green":2}`. How do you sort it? The dictionary object does not have the `sort` method. But the `list` object has it, so you can get the pairs into a list and then sort that list. The program obtains the list of pairs in line 10. If you apply the `sort` method to the list, the pairs will be sorted on their first element, but we need to sort each pair on their count (the second element). How can we do this? The trick is to reverse the pair. The program creates a new list with all the pairs reversed (line 12), and then applies the `sort` method (line 14). Now the list is sorted like this: `[[2, "green"], [5, "blue"], [7, "red"]]`.

sort pairs

The program displays the last ten pairs from the list to show the words with the highest count (lines 16–17).

`processLine`

The `processLine(line, wordCounts)` function invokes `replacePunctuations(line)` to replace all punctuation marks by spaces (line 21), then extracts words by using the `split` method (line 22). If a word is already in the dictionary, the program increases its count (line 25); otherwise, the program adds a new pair to the dictionary (line 27).

`replacePunctuations(line)`

The `replacePunctuations(line)` method checks each character in each line. If it is a punctuation mark, the program replaces it with a space (lines 32–33).

Now sit back and think how you would write this program without using a dictionary. You could use a nested list such as `[[key1, value1], [key2, value2], ... ]`, but your new program would be longer and more complex. You will find that a dictionary is a very efficient and powerful data structure for solving problems such as this.

Key Terms

data structure 476
dictionary 487
dictionary entry 488
dictionary item 488
hashable 479
immutable tuple 478
key/value pair 488
map 488
set 479
set difference 481
set intersection 481
set union 481
set symmetric difference 482
tuple 476

Chapter Summary

1. A *tuple* is a fixed list. You cannot add, delete, or replace elements in a tuple.

2. Since a tuple is a sequence, the common operations for sequences can be used for tuples.

3. Though you cannot add, delete, or replace elements in a tuple, you can change the content of individual elements if the elements are mutable.

4. A tuple is *immutable* if all its elements are immutable.