Chapter 5 assignment

Lists,Tuples and Dictionaries

EXERCISES

5.3 Use a list to solve the following problem: Read in 20 numbers. As each number is read, print it only if it is not a duplicate of a number already read.

5.4 Use a list of lists to solve the following problem. A company has four salespeople (1 to 4) who sell five different products (1 to 5). Once a day, each salesperson passes in a slip for each differ- ent type of product sold. Each slip contains:

a) The salesperson number.

b) The product number.

c) The number of that product sold that day.

Thus, each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all of the slips for last month is available. Write a program that will read all this information for last month's sales and summarise the total sales by salesperson by product. All totals should be stored in list **sales**. After processing all the information for last month, display the results in tabu- lar format, with each of the columns representing a particular salesperson and each of the rows rep- resenting a particular product. Cross-total each row to get the total sales of each product for last month; cross-total each column to get the total sales by salesperson for last month. Your tabular print out should include these cross-totals to the right of the totaled rows and at the bottom of the totaled columns.

5.5 (*The Sieve of Eratosthenes*) A prime integer is any integer greater than 1 that is evenly divisible only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:

a) Create a list with all elements initialised to 1 (true). List elements with prime subscripts will remain 1. All other list elements will eventually be set to zero.

b) Starting with list element 2, every time a list element is found whose value is 1, loop through the remainder of the list and set to zero every element whose subscript is a multiple of the subscript for the element with value 1. For list subscript 2, all elements be- yond 2 in the list that are multiples of 2 will be set to zero (subscripts 4, 6, 8, 10, etc.); for list subscript 3, all elements beyond 3 in the list that are multiples of 3 will be set to zero (subscripts 6, 9, 12, 15, etc.); and so on.

When this process is complete, the list elements that are still set to 1 indicate that the subscript is a prime number. These subscripts can then be printed. Write a program that uses a list of 1000 elements to determine and print the prime numbers between 2 and 999. Ignore element 0 of the list.

5.6 (*Bubble Sort*) Sorting data (i.e. placing data into some particular order, such as ascending or descending) is one of the most important computing applications. Python lists provide a **sort** meth- od. In this exercise, readers implement their own sorting function, using the bubble-sort method. In the bubble sort (or *sinking* sort), the smaller values gradually "bubble" their way upward to the top of the list like air bubbles rising in water, while the larger values sink to the bottom of the list. The pro- cess that compares each adjacent pair of elements in a list in turn and swaps the elements if the second element is less than the first element is called a pass. The technique makes several passes through the list. On each pass, successive pairs of elements are compared. If a pair is in increasing order, bubble sort leaves the values as they are. If a pair is in decreasing order, their values are swapped in the list. After the first pass, the largest value is guaranteed to sink to the highest index of a list. After the second pass, the second largest value is guaranteed to sink to the second highest index of a list, and so on. Write a program that uses the function **bubbleSort** to sort the items in a list.

5.7 (*Binary Search*) When a list is sorted, a high-speed binary search technique can find items in the list quickly. The binary search algorithm eliminates from consideration one-half of the elements in the list being searched after each comparison. The algorithm locates the middle element of the list and compares it with the search key. If they are equal, the search key is found, and the subscript of that element is returned. Otherwise, the problem is reduced to searching one half of the list. If the search key is less than the middle element of the list, the first half of the list is searched. If the search key is not the middle element in the specified piece of the original list, the algorithm is repeated on one-quarter of the original list. The search continues until the search key is equal to the middle ele- ment of the smaller list or until the smaller list consists of one element that is not equal to the search key (i.e. the search key is not found.)
Even in a worst-case scenario, searching a list of 1024 elements will take only 10 comparisons during a binary search. Repeatedly dividing 1024 by 2 (because after each comparison we are able to eliminate from the consideration half the list) yields the values 512, 256, 128, 64, 32, 16, 8, 4, 2 and 1. The number 1024 (210) is divided by 2 only ten times to get the value 1. Dividing by 2 is equivalent to

one comparison in the binary-search algorithm. A list of 1,048,576 (220) elements takes a maximum of 20 comparisons to find the key. A list of one billion elements takes a maximum of 30 comparisons to find the key. The maximum number of comparisons needed for the binary search of any sorted list can be determined by finding the first power of 2 greater than or equal to the number of elements in the list.

Write a program that implements function binarySearch, which takes a sorted list and a search key as arguments. The function should return the index of the list value that matches the search key (or -1, if the search key is not found).

5.8 Create a dictionary of 20 random values in the range 1–99. Determine whether there are any duplicate values in the dictionary. (*Hint*: you may want to sort the list first.)