Loop Exercises

3

The exercises that appear in this chapter should all be completed using loops. In some cases the exercise specifies what type of loop to use. In other cases you must make this decision yourself. Some of the exercises can be completed easily with both `for` loops and `while` loops. Other exercises are much better suited to one type of loop than the other. In addition, some of the exercises require multiple loops. When multiple loops are involved, one loop might need to be nested inside the other. Carefully consider your choice of loops as you design your solution to each problem.

Exercise 61: Average

(*26 Lines*)

In this exercise you will create a program that computes the average of a collection of values entered by the user. The user will enter 0 as a sentinel value to indicate that no further values will be provided. Your program should display an appropriate error message if the first value entered by the user is 0.

> Hint: Because the 0 marks the end of the input it should **not** be included in the average.

Exercise 62: Discount Table

(*18 Lines*)

A particular retailer is having a 60 percent off sale on a variety of discontinued products. The retailer would like to help its customers determine the reduced price of the merchandise by having a printed discount table on the shelf that shows the

B. Stephenson, *The Python Workbook*, DOI 10.1007/978-3-319-14240-1_3

original prices and the prices after the discount has been applied. Write a program that uses a loop to generate this table, showing the original price, the discount amount, and the new price for purchases of $4.95, $9.95, $14.95, $19.95 and $24.95. Ensure that the discount amounts and the new prices are rounded to 2 decimal places when they are displayed.

Exercise 63: Temperature Conversion Table

(*22 Lines*)

Write a program that displays a temperature conversion table for degrees Celsius and degrees Fahrenheit. The table should include rows for all temperatures between 0 and 100 degrees Celsius that are multiples of 10 degrees Celsius. Include appropriate headings on your columns. The formula for converting between degrees Celsius and degrees Fahrenheit can be found on the internet.

Exercise 64: No More Pennies

(*Solved—38 Lines*)

February 4, 2013 was the last day that pennies were distributed by the Royal Canadian Mint. Now that pennies have been phased out retailers must adjust totals so that they are multiples of 5 cents when they are paid for with cash (credit card and debit card transactions continue to be charged to the penny). While retailers have some freedom in how they do this, most choose to round to the closest nickel.

Write a program that reads prices from the user until a blank line is entered. Display the total cost of all the entered items on one line, followed by the amount due if the customer pays with cash on a second line. The amount due for a cash payment should be rounded to the nearest nickel. One way to compute the cash payment amount is to begin by determining how many pennies would be needed to pay the total. Then compute the remainder when this number of pennies is divided by 5. Finally, adjust the total down if the remainder is less than 2.5. Otherwise adjust the total up.

Exercise 65: Compute the Perimeter of a Polygon

(*Solved—42 Lines*)

Write a program that computes the perimeter of a polygon. Begin by reading the x and y values for the first point on the perimeter of the polygon from the user. Then continue reading pairs of x and y values until the user enters a blank line for the

x-coordinate. Each time you read an additional coordinate you should compute the distance to the previous point and add it to the perimeter. When a blank line is entered for the x-coordinate your program should add the distance from the last point back to the first point to the perimeter. Then it should display the total perimeter. Sample input and output is shown below, with user input shown in bold:

```
Enter the x part of the coordinate: 0
Enter the y part of the coordinate: 0
Enter the x part of the coordinate: (blank to quit): 1
Enter the y part of the coordinate: 0
Enter the x part of the coordinate: (blank to quit): 0
Enter the y part of the coordinate: 1
Enter the x part of the coordinate: (blank to quit):
The perimeter of that polygon is 3.414213562373095
```

Exercise 66: Compute a Grade Point Average

(*62 Lines*)

Exercise 51 included a table that shows the conversion from letter grades to grade points at a particular academic institution. In this exercise you will compute the grade point average of an arbitrary number of letter grades entered by the user. The user will enter a blank line to indicate that all of the grades have been provided. For example, if the user enters `A`, followed by `C+`, followed by `B`, followed by a blank line then your program should report a grade point average of 3.1.

You may find your solution to Exercise 51 helpful when completing this exercise. Your program does not need to do any error checking. It can assume that each value entered by the user will always be a valid letter grade or a blank line.

Exercise 67: Admission Price

(*Solved—38 Lines*)

A particular zoo determines the price of admission based on the age of the guest. Guests 2 years of age and less are admitted without charge. Children between 3 and 12 years of age cost $14.00. Seniors aged 65 years and over cost $18.00. Admission for all other guests is $23.00.

Create a program that begins by reading the ages of all of the guests in a group from the user, with one age entered on each line. The user will enter a blank line to indicate that there are no more guests in the group. Then your program should display the admission cost for the group with an appropriate message. The cost should be displayed using two decimal places.

Exercise 68: Parity Bits

(*Solved—25 Lines*)

A parity bit is a simple mechanism for detecting errors in data transmitted over an unreliable connection such as a telephone line. The basic idea is that an additional bit is transmitted after each group of 8 bits so that a single bit error in the transmission can be detected.

Parity bits can be computed for either even parity or odd parity. If even parity is selected then the parity bit that is transmitted is chosen so that the total number of one bits transmitted (8 bits of data plus the parity bit) is even. When odd parity is selected the parity bit is chosen so that the total number of one bits transmitted is odd.

Write a program that computes the parity bit for groups of 8 bits entered by the user using even parity. Your program should read strings containing 8 bits until the user enters a blank line. After each string is entered by the user your program should display a clear message indicating whether the parity bit should be 0 or 1. Display an appropriate error message if the user enters something other than 8 bits.

Hint: You should read the input from the user as a string. Then you can use the `count` method to help you determine the number of zeros and ones in the string. Information about the `count` method is available online.

Exercise 69: Approximate π

(*23 Lines*)

The value of π can be approximated by the following infinite series:

$$\pi \approx 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \frac{4}{10 \times 11 \times 12} - \cdots$$

Write a program that displays 15 approximations of π. The first approximation should make use of only the first term from the infinite series. Each additional approximation displayed by your program should include one more term in the series, making it a better approximation of π than any of the approximations displayed previously.

Exercise 70: Caesar Cipher

(*Solved—35 Lines*)

One of the first known examples of encryption was used by Julius Caesar. Caesar needed to provide written instructions to his generals, but he didn't want his enemies

to learn his plans if the message slipped into their hands. As result, he developed what later became known as the Caesar Cipher.

The idea behind this cipher is simple (and as a result, it provides no protection against modern code breaking techniques). Each letter in the original message is shifted by 3 places. As a result, A becomes D, B becomes E, C becomes F, D becomes G, etc. The last three letters in the alphabet are wrapped around to the beginning: X becomes A, Y becomes B and Z becomes C. Non-letter characters are not modified by the cipher.

Write a program that implements a Caesar cipher. Allow the user to supply the message and the shift amount, and then display the shifted message. Ensure that your program encodes both uppercase and lowercase letters. Your program should also support negative shift values so that it can be used both to encode messages and decode messages.

Exercise 71: Square Root

(*14 Lines*)

Write a program that implements Newton's method to compute and display the square root of a number entered by the user. The algorithm for Newton's method follows:

Read x from the user
Initialize *guess* to $x/2$
While *guess* is not good enough **do**
    Update *guess* to be the average of *guess* and $x/guess$

When this algorithm completes, *guess* contains an approximation of the square root. The quality of the approximation depends on how you define "good enough". In the author's solution, *guess* was considered good enough when the absolute value of the difference between $guess * guess$ and x was less than or equal to 10^{-12}.

Exercise 72: Is a String a Palindrome?

(*Solved—23 Lines*)

A string is a palindrome if it is identical forward and backward. For example "anna", "civic", "level" and "hannah" are all examples of palindromic words. Write a program that reads a string from the user and uses a loop to determines whether or not it is a palindrome. Display the result, including a meaningful output message.

Exercise 73: Multiple Word Palindromes

(*35 Lines*)

There are numerous phrases that are palindromes when spacing is ignored. Examples include "go dog", "flee to me remote elf" and "some men interpret nine memos", among many others. Extend your solution to Exercise 72 so that it ignores spacing while determining whether or not a string is a palindrome. For an additional challenge, extend your solution so that is also ignores punctuation marks and treats uppercase and lowercase letters as equivalent.

Exercise 74: Multiplication Table

(*Solved—18 Lines*)

In this exercise you will create a program that displays a multiplication table that shows the products of all combinations of integers from 1 times 1 up to and including 10 times 10. Your multiplication table should include a row of labels across the top of it containing the numbers 1 through 10. It should also include labels down the left side consisting of the numbers 1 through 10. The expected output from the program is shown below:

```
    1   2   3   4   5   6   7   8   9  10
1   1   2   3   4   5   6   7   8   9  10
2   2   4   6   8  10  12  14  16  18  20
3   3   6   9  12  15  18  21  24  27  30
4   4   8  12  16  20  24  28  32  36  40
5   5  10  15  20  25  30  35  40  45  50
6   6  12  18  24  30  36  42  48  54  60
7   7  14  21  28  35  42  49  56  63  70
8   8  16  24  32  40  48  56  64  72  80
9   9  18  27  36  45  54  63  72  81  90
10 10  20  30  40  50  60  70  80  90 100
```

When completing this exercise you will probably find it helpful to be able to print out a value without moving down to the next line. This can be accomplished by added `end=""` as the last parameter to your print statement. For example, `print("A")` will display the letter `A` and then move down to the next line. The statement `print("A", end="")` will display the letter `A` without moving down to the next line, causing the next print statement to display its result on the same line as the letter `A`.

Exercise 75: Greatest Common Divisor

(*Solved—17 Lines*)

The greatest common divisor of two positive integers, n and m, is the largest number, d, which divides evenly into both n and m. There are several algorithms that can be used to solve this problem, including:

Initialize d to the smaller of m and n.
While d does not evenly divide m or d does not evenly divide n **do**
    Decrease the value of d by 1
Report d as the greatest common divisor of n and m

Write a program that reads two positive integers from the user and uses this algorithm to determine and report their greatest common divisor.

Exercise 76: Prime Factors

(*27 Lines*)

The prime factorization of an integer, n, can be determined using the following steps:

Initialize *factor* to two
While *factor* is less than or equal to n **do**
    If n is evenly divisible by *factor* **then**
        Conclude that *factor* is a factor of n
        Divide n by *factor* using integer division
    Else
        Increase *factor* by one

Write a program that reads an integer from the user. If the value entered by the user is less than 2 then your program should display an appropriate error message. Otherwise your program should display the prime numbers that can be multiplied together to compute n, with one factor appearing on each line. For example:

```
Enter an integer (2 or greater): 72
The prime factors of 72 are:
2
2
2
3
3
```

Exercise 77: Binary to Decimal

(*18 Lines*)

Write a program that converts a binary (base 2) number to decimal (base 10). Your program should begin by reading the binary number from the user as a string. Then it should compute the equivalent decimal number by processing each digit in the binary number. Finally, your program should display the equivalent decimal number with an appropriate message.

Exercise 78: Decimal to Binary

(*Solved—26 Lines*)

Write a program that converts a decimal (base 10) number to binary (base 2). Read the decimal number from the user as an integer and then use the division algorithm shown below to perform the conversion. When the algorithm completes, *result* contains the binary representation of the number. Display the result, along with an appropriate message.

Let *result* be an empty string
Let q represent the number to convert
repeat
    Set r equal to the remainder when q is divided by 2
    Convert r to a string and add it to the beginning of *result*
    Divide q by 2, discarding any remainder, and store the result back into q
until q is 0

Exercise 79: Maximum Integer

(*Solved—34 Lines*)

This exercise examines the process of identifying the maximum value in a collection of integers. Each of the integers will be randomly selected from the numbers between 1 and 100. The collection of integers may contain duplicate values, and some of the integers between 1 and 100 may not be present.

Take a moment and think about how you would handle this problem on paper. Many people would check each integer in sequence and ask themself if the number that they are currently considering is larger than the largest number that they have seen previously. If it is, then they forget the previous maximum number and remember the current number as the new maximum number. This is a reasonable approach, and will result in the correct answer when the process is performed carefully. If you were performing this task, how many times would you expect to need to update the maximum value and remember a new number?

While we can answer the question posed at the end of the previous paragraph using probability theory, we are going to explore it by simulating the situation. Create a program that begins by selecting a random integer between 1 and 100. Save this integer as the maximum number encountered so far. After the initial integer has been selected, generate 99 additional random integers between 1 and 100. Check each integer as it is generated to see if it is larger than the maximum number encountered so far. If it is then your program should update the maximum number encountered and count the fact that you performed an update. Display each integer after you generate it. Include a notation with those integers which represent a new maximum.

After you have displayed 100 integers your program should display the maximum value encountered, along with the number of times the maximum value was updated during the process. Partial output for the program is shown below, with… representing the remaining integers that your program will display. Run your program several times. Is the number of updates performed on the maximum value what you expected?

```
30
74 <== Update
58
17
40
37
13
34
46
52
80 <== Update
37
97 <== Update
45
55
73
...

The maximum value found was 100
The maximum value was updated 5 times
```

Exercise 80: Coin Flip Simulation

(*47 Lines*)

What's the minimum number of times you have to flip a coin before you can have three consecutive flips that result in the same outcome (either all three are heads or all three are tails)? What's the maximum number of flips that might be needed? How

many flips are needed on average? In this exercise we will explore these questions by creating a program that simulates several series of coin flips.

Create a program that uses Python's random number generator to simulate flipping a coin several times. The simulated coin should be fair, meaning that the probability of heads is equal to the probability of tails. Your program should flip simulated coins until either 3 consecutive heads of 3 consecutive tails occur. Display an `H` each time the outcome is heads, and a `T` each time the outcome is tails, with all of the outcomes shown on the same line. Then display the number of flips needed to reach 3 consecutive flips with the same outcome. When your program is run it should perform the simulation 10 times and report the average number of flips needed. Sample output is shown below:

```
H T T T (4 flips)
H H T T H T H T T H H T H T T H T T T (19 flips)
T T T (3 flips)
T H H H (4 flips)
H H H (3 flips)
T H T T H T H H T T H H T H T H H H (18 flips)
H T T H H H (6 flips)
T H T T T (5 flips)
T T H T T H T H T H H H (12 flips)
T H T T T (5 flips)
On average, 7.9 flips were needed.
```