Recursion Exercises

8

A recursive function is a function that calls itself. In this chapter, you will use recursive functions to solve a variety of problems. The programs that you write will help you learn to:

- Identify the base case(s) for a recursive function
- Identify the recursive case(s) for a recursive function
- Write a non-trivial recursive function
- Use a recursive function that you have written to solve a problem

Exercise 164: Total the Values

(*Solved—28 Lines*)

Write a program that reads values from the user until a blank line is entered. Display the total of all of the values entered by the user (or 0.0 if the first value entered is a blank line). Complete this task using recursion. Your program may not use any loops.

> Hint: The body of your recursive function will need to read one value from the user, and then determine whether or not to make a recursive call. Your function does not need to take any parameters, but it will need to return a numeric result.

B. Stephenson, *The Python Workbook*, DOI 10.1007/978-3-319-14240-1_8

Exercise 165: Greatest Common Divisor

(*24 Lines*)

Euclid was a Greek mathematician who lived approximately 2,300 years ago. His algorithm for computing the greatest common divisor of two positive integers, `a` and `b`, is both efficient and recursive. It is outlined below:

If *b* is 0 **then**
  Return *a*
Else
  Set *c* equal to the remainder when *a* is divided by *b*
  Return the greatest common divisor of *b* and *c*

Write a program that implements Euclid's algorithm and uses it to determine the greatest common divisor of two integers entered by the user.

Exercise 166: Recursive Decimal to Binary

(*34 Lines*)

In Exercise 78 you wrote a program that used a loop to convert a decimal number to its binary representation. In this exercise you will perform the same task using recursion.

Write a recursive function that converts a non-negative decimal number to binary. Treat 0 and 1 as base cases which return a string containing the appropriate digit. For all other positive integers, *n*, you should compute the next digit using the remainder operator and then make a recursive call to compute the digits of n // 2. Finally, you should concatenate the result of the recursive call (which will be a string) and the next digit (which you will need to convert to a string) and return this string as the result of the function.

Write a main program that uses your recursive function to convert a non-negative integer entered by the user from decimal to binary. Your program should display an appropriate error message if the user enters a negative value.

Exercise 167: Recursive Palindrome

(*Solved—29 Lines*)

The notion of a palindrome was introduced previously in Exercise 72. In this exercise you will write a recursive function that determines whether or not a string is a palindrome. The empty string is a palindrome, as is any string containing only

one character. Any longer string is a palindrome if its first and last characters match, and if the string formed by removing the first and last characters is also a palindrome.

Write a main program that reads a string from the user. Use your recursive function to determine whether or not the string is a palindrome. Then display an appropriate message for the user.

Exercise 168: Recursive Square Root

(*20 Lines*)

Exercise 71 explored how iteration can be used to compute the square root of a number. In that exercise a better approximation of the square root was generated with each additional iteration of a loop. In this exercise you will use the same approximation strategy, but you will use recursion instead of iteration.

Create a square root function that takes two parameters. The first parameter, n, will be the number for which the square root is being computed. The second parameter, *guess*, will be the current guess for the square root. The guess parameter should have a default value of 1.0. Do not provide a default value for the first parameter.

Your square root function will be recursive. The base case occurs when $guess^2$ is within 10^{-12} of n. In this case your function should return $guess$ because it is close enough to the square root of n. Otherwise your function should return the result of calling itself recursively with n as the first parameter and $\frac{guess+\frac{n}{guess}}{2}$ as the second parameter.

Write a main program that demonstrate your square root function by computing the square root of several different values. When you call your square root function from the main program you should only pass one parameter to it so that the default value for $guess$ is used.

Exercise 169: String Edit Distance

(*Solved—42 Lines*)

The edit distance between two strings is a measure of their similarity—the smaller the edit distance, the more similar the strings are with regard to the minimum number of insert, delete and substitute operations needed to transform one string into the other.

Consider the strings `kitten` and `sitting`. The first string can be transformed into the second string with the following operations: Substitute the `k` with an `s`, substitute the `e` with an `i`, and insert a `g` at the end of the string. This is the smallest number of operations that can be performed to transform `kitten` into `sitting`. As a result, the edit distance is 3.

Write a recursive function that computes the edit distance between two strings. Use the following algorithm:

```
Let s and t be the strings
If the length of s is 0 then
   Return the length of t
Else if the length of t is 0 then
   Return the length of s
Else
   Set cost to 0
   If the last character in s does not equal the last character in t then
      Set cost to 1
   Set d1 equal to the edit distance between all characters except the last one
   in s, and all characters in t, plus 1
   Set d2 equal to the edit distance between all characters in s, and all
   characters except the last one in t, plus 1
   Set d3 equal to the edit distance between all characters except the last one
   in s, and all characters except the last one in t, plus cost
   Return the minimum of d1, d2 and d3
```

Use your recursive function to write a program that reads two strings from the user and displays the edit distance between them.

Exercise 170: Possible Change

(*41 Lines*)

Create a program that determines whether or not it is possible to construct a particular total using a specific number of coins. For example, it is possible to have a total of $1.00 using four coins if they are all quarters. However, there is no way to have a total of $1.00 using 5 coins. Yet it is possible to have $1.00 using 6 coins by using 3 quarters, 2 dimes and a nickel. Similarly, a total of $1.25 can be formed using 5 coins or 8 coins, but a total of $1.25 can not be formed using 4, 6 or 7 coins.

Your program should read both the dollar amount and the number of coins from the user. It should display a clear message indicating whether or not the entered dollar amount can be formed using the number of coins indicated. Assume the existence of quarters, dimes, nickels and pennies when completing this problem. Your solution must use recursion. It can not contain any loops.

Exercise 171: Spelling with Element Symbols

(*68 Lines*)

Each chemical element has a standard symbol that is one, two or three letters long. One game that some people like to play is to determine whether or not a word can be spelled using only element symbols. For example, silicon can be spelled using

the symbols Si, Li, C, O and N. However, hydrogen can not be spelled with any combination of element symbols.

Write a recursive function that determines whether or not a word can be spelled using only element symbols. Your function will take two parameters: the word that you are trying to spell and a list of the symbols that can be used. Your function will return two results: a Boolean value indicating whether or not a spelling was found, and the string of symbols used to achieve the spelling (or an empty string if no spelling exists). Your function should ignore capitalization when searching for a spelling.

Create a program that uses your function to find and display all of the element names that can be spelled using only element symbols. Display the names of the elements along with the sequences of symbols. For example, one line of your output will be:

```
Silver can be spelled as SiLvEr
```

Your program will use the elements data set, which can be downloaded from the author's website. This data set includes the names and symbols of all 118 chemical elements.

Exercise 172: Element Sequences

(*Solved—83 Lines*)

Another game that some people play with the names of chemical elements involves constructing a sequence of elements where each element in the sequence begins with the last letter of its predecessor. For example, if a sequence begins with Hydrogen, then the next element must be an element that begins with N, such as Nickel. The element following Nickel must begin with L, such as Lithium. The element sequence that is constructed can not contain any duplicates.

Write a program that reads the name of an element from the user. Your program should use a recursive function to find the longest sequence of elements that begins with the entered element. Then it should display the sequence. Ensure that your program responds in a reasonable way if the user does not enter a valid element name.

> Hint: It may take your program up to two minutes to find the longest sequence for some elements. As a result, you might want to use elements like Molybdenum and Magnesium as your first test cases. Each has a longest sequence that is only 8 elements long which your program should find in a fraction of a second.

Exercise 173: Run-Length Decoding

(*33 Lines*)

Run-length encoding is a simple data compression technique that can be effective when repeated values occur at adjacent positions within a list. Compression is

achieved by replacing groups of repeated values with one copy of the value, followed by the number of times that the value should be repeated. For example, the list `["A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "B", "B", "B", "B", "A", "A", "A", "A", "A", "A", "B"]` would be compressed as `["A", 12, "B", 4, "A", 6, "B", 1]`. Decompression is performed by replicating each value in the list the number of times indicated.

Write a recursive function that decompresses a run-length encoded list. Your recursive function will take a run-length compressed list as its only parameter. It will return the decompressed list as its only result. Create a main program that displays a run-length encoded list and the result of decoding it.

Exercise 174: Run-Length Encoding

(*Solved—36 Lines*)

Write a recursive function that implements the run-length compression technique described in Exercise 173. Your function will take a list or a string as its only parameter. It should return the run-length compressed list as its only result. Include a main program that reads a string from the user, compresses it, and displays the run-length encoded result.

> Hint: You may want to include a loop inside the body of your recursive function.