5.4. The Three Laws of Recursion

Like the robots of Asimov, all recursive algorithms must obey three important laws:

1. A recursive algorithm must have a **base case**.
2. A recursive algorithm must change its state and move toward the base case.
3. A recursive algorithm must call itself, recursively.

Let's look at each one of these laws in more detail and see how it was used in the `listsum` algorithm. First, a base case is the condition that allows the algorithm to stop recursing. A base case is typically a problem that is small enough to solve directly. In the `listsum` algorithm the base case is a list of length 1.

To obey the second law, we must arrange for a change of state that moves the algorithm toward the base case. A change of state means that some data that the algorithm is using is modified. Usually the data that represents our problem gets smaller in some way. In the `listsum` algorithm our primary data structure is a list, so we must focus our state-changing efforts on the list. Since the base case is a list of length 1, a natural progression toward the base case is to shorten the list. This is exactly what happens on line 5 of ActiveCode 2 (pythondsCalculatingtheSumofaListofNumbers.html#lst-recsum) when we call `listsum` with a shorter list.

The final law is that the algorithm must call itself. This is the very definition of recursion. Recursion is a confusing concept to many beginning programmers. As a novice programmer, you have learned that functions are good because you can take a large problem and break it up into smaller problems. The smaller problems can be solved by writing a function to solve each problem. When we talk about recursion it may seem that we are talking ourselves in circles. We have a problem to solve with a function, but that function solves the problem by calling itself! But the logic is not circular at all; the logic of recursion is an elegant expression of solving a problem by breaking it down into a smaller and easier problems.

In the remainder of this chapter we will look at more examples of recursion. In each case we will focus on designing a solution to a problem by using the three laws of recursion.

Self Check

Q-1: How many recursive calls are made when computing the sum of the list [2,4,6,8,10]?

- **A. 6**
- **B. 5**
- **C. 4**
- **D. 3**

Check Me Compare me

Activity: 5.4.1 Multiple Choice (question_recsimp_1)

Q-2: Suppose you are going to write a recusive function to calculate the factorial of a number. fact(n) returns n * n-1 * n-2 * … Where the factorial of zero is defined to be 1. What would be the most appropriate base case?

(pythondsCalculatingtheSumofaListofNumbers.html) (pythondsConvert

- ○ **A. n == 0**
- ○ **B. n == 1**
- ○ **C. n >= 0**
- ○ **D. n <= 1**

Check Me Compare me

Activity: 5.4.2 Multiple Choice (question_recsimp_2)

You have attempted 1 of 3 activities on this page

user not logged in

(pythondsCalculatingtheSumofaListofNumbers.html)

(pythondsConvert