5.3. Calculating the Sum of a List of Numbers

We will begin our investigation with a simple problem that you already know how to solve without using recursion. Suppose that you want to calculate the sum of a list of numbers such as: $[1, 3, 5, 7, 9]$. An iterative function that computes the sum is shown in ActiveCode 1. The function uses an accumulator variable (`theSum`) to compute a running total of all the numbers in the list by starting with 0 and adding each number in the list.

Run Load History Show CodeLens

```
def listsum(numList):
    theSum = 0
    for i in numList:
        theSum = theSum + i
    return theSum

print(listsum([1,3,5,7,9]))
```

Activity: 5.3.1 Iterative Summation (lst_itsum)

Pretend for a minute that you do not have `while` loops or `for` loops. How would you compute the sum of a list of numbers? If you were a mathematician you might start by recalling that addition is a function that is defined for two parameters, a pair of numbers. To redefine the problem from adding a list to adding pairs of numbers, we could rewrite the list as a fully parenthesized expression. Such an expression looks like this:

$$((((1+3)+5)+7)+9)$$

We can also parenthesize the expression the other way around,

$$(1+(3+(5+(7+9))))$$

Notice that the innermost set of parentheses, $(7+9)$, is a problem that we can solve without a loop or any special constructs. In fact, we can use the following sequence of simplifications to compute a final sum.

(WhatIsRecursion.html)

$$\begin{aligned} total = &\ (1+(3+(5+(7+9)))) \\ total = &\ (1+(3+(5+16))) \\ total = &\ (1+(3+21)) \\ total = &\ (1+24) \\ total = &\ 25 \end{aligned}$$

(TheThreeLawsof

How can we take this idea and turn it into a Python program? First, let's restate the sum problem in terms of Python lists. We might say the the sum of the list `numList` is the sum of the first element of the list (`numList[0]`), and the sum of the numbers in the rest of the list (`numList[1:]`). To state it in a functional form:

$$listSum(numList) = first(numList) + listSum(rest(numList))$$

In this equation $first(numList)$ returns the first element of the list and $rest(numList)$ returns a list of everything but the first element. This is easily expressed in Python as shown in ActiveCode 2.

Run Load History Show CodeLens

```
1 def listsum(numList):
2    if len(numList) == 1:
3         return numList[0]
4    else:
5         return numList[0] + listsum(numList[1:])
6
7 print(listsum([1,3,5,7,9]))
8
```

Activity: 5.3.2 Recursive Summation (lst_recsum)

There are a few key ideas in this listing to look at. First, on line 2 we are checking to see if the list is one element long. This check is crucial and is our escape clause from the function. The sum of a list of length 1 is trivial; it is just the number in the list. Second, on line 5 our function calls itself! This is the reason that we call the `listsum` algorithm recursive. A recursive function is a function that calls itself.

Figure 1 shows the series of **recursive calls** that are needed to sum the list $[1, 3, 5, 7, 9]$. You should think of this series of calls as a series of simplifications. Each time we make a recursive call we are solving a smaller problem, until we reach the point where the problem cannot get any smaller.

(WhatIsRecursion.html)

(TheThreeLawsof

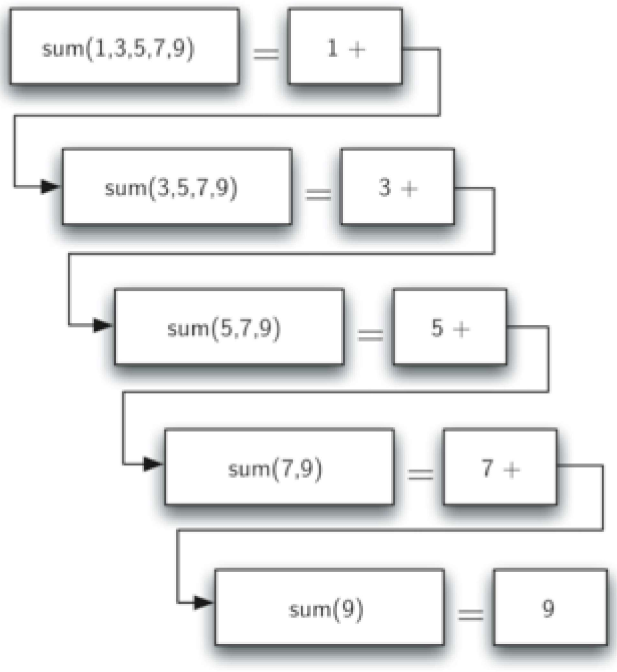

Figure 1: Series of Recursive Calls Adding a List of Numbers

When we reach the point where the problem is as simple as it can get, we begin to piece together the solutions of each of the small problems until the initial problem is solved. Figure 2 shows the additions that are performed as `listsum` works its way backward through the series of calls. When `listsum` returns from the topmost problem, we have the solution to the whole problem.

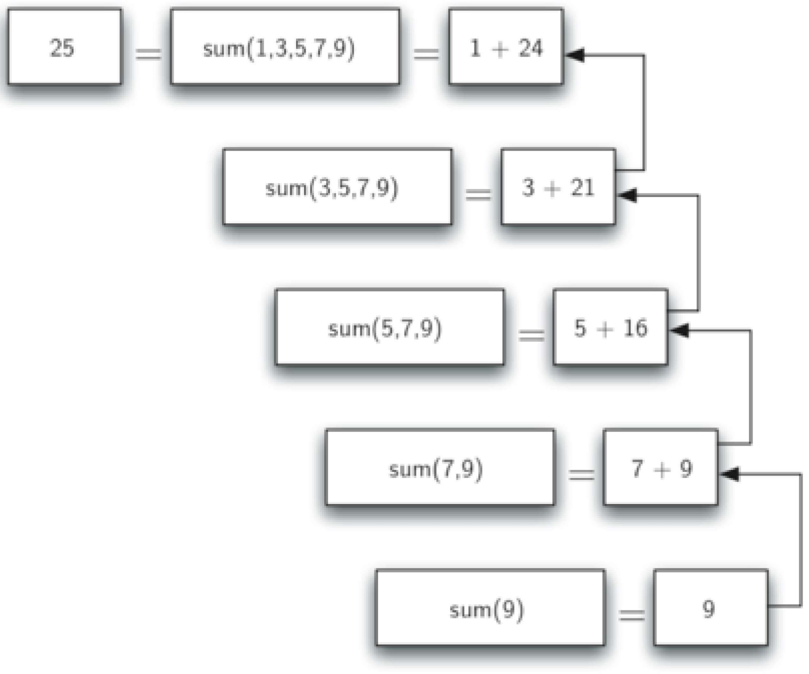

Figure2: Series of Recursive Returns from Adding a List of Numbers

You have attempted 1 of 3 activities on this page

(WhatIsRecursion.html) (TheThreeLawsof

user not logged in

(WhatIsRecursion.html)

(TheThreeLawsof