Recursion Outline

3 min

Recursion is a strategy for solving problems by defining the problem in terms of itself. For example, to **sum the elements of a list** we would take the first element and add it to the **sum of the remaining elements**.

How would we get that sum of remaining elements? Easy! We'd take the first element of the remaining elements and add it to the… Maybe you can understand why recursion can be a tricky subject!

In programming, recursion means a function definition will include an invocation of the function **within its own body.** Here's a pseudo-code example:

```
define function, speller
  if there are no more letters
    print "all done"
  print the first letter
  invoke speller with the given name minus the first letter
```

If we invoked this function with "Zoe" as the

Preview: Docs Loading link description

argument

, we would see "Z", "o", and "e" printed out before "all done".

We call the function a total of 4 times!

1. function called with "Zoe"
2. function called with "oe"
3. function called with "e"
4. function called with ""

Let's break the function into three chunks:

```
  if there are no more letters
    print "all done"
```

This section is the *base case*. We are **NOT** invoking the function under this condition. The equivalent base case from the previous exercise was when we had reached the front of the line.

```
  print the first letter
```

This section solves **a piece** of the problem. If we want to spell someone's name, we'll have to spell **at least** one letter.

```
  invoke speller with the given name minus the first letter
```

This section is the *recursive* step, calling the function with arguments which bring us closer to the base case. In this example, we're reducing the length of the name by a single letter. Eventually, there will be a function call with no letters given as the argument.

For comparison's sake, here is pseudo-code for an *iterative* approach to the same problem:

```
define function, speller
  for each letter in the name argument
    print the letter
  print "all done"
```

Instructions

Compare the two pseudo-code functions.

How are they different? What parts are similar?

How is it possible for a function to resolve if it refers to itself?