Recursion is very powerful tool, both mentally and in software development, for solving problems. Recursive functions have two main parts: a few easy to solve problems called “base cases,” and then a case for more complicated problems where **the function is called inside of itself**. The central philosophy of recursive programming is that problems can be broken down into simpler parts, and then combining those simple answers results in the answer to a complex problem.

Imagine you wanted to write a function that adds together all of the numbers in a vector. You could of course accomplish this with a loop:

```
vector_sum_loop <- function(v){
  result <- 0
  for(i in v){
    result <- result + i
  }
  result
}

vector_sum_loop(c(5, 40, 91))
[1] 136
```

You could also think about how to solve this problem recursively. First ask yourself: what’s the base case of finding the sum of a vector? If the vector only contains one element, then the sum is just the value of that element. In the more complex case the vector has more than one element. We can remove the first element of the vector, but then what should we do with the rest of the vector? Thankfully we have a function for computing the sum of all of the elements of a vector because we’re writing that function right now! So we’ll add the value of the first element of the vector to whatever the cumulative sum is of the rest of the vector. The resulting function is illustrated below:

```
vector_sum_rec <- function(v){
  if(length(v) == 1){
    v
  } else {
    v[1] + vector_sum_rec(v[-1])
  }
}

vector_sum_rec(c(5, 40, 91))
[1] 136
```

Another useful exercise for thinking about applications for recursion is computing the Fibonacci sequence. The Fibonacci sequence is a sequence of integers that starts: 0, 1, 1, 2, 3, 5, 8 where each proceeding integer is the sum of the previous two integers. This fits into a recursive mental framework very nicely since each subsequent number depends on the previous two numbers.

Let’s write a function to compute the nth digit of the Fibonacci sequence such that the first number in the sequence is 0, the second number is 1, and then all proceeding numbers are the sum of the n - 1 and the n - 2 Fibonacci number. It is immediately evident that there are three base cases:

1. n must be greater than 0.
2. When n is equal to 1, return 0.
3. When n is equal to 2, return 1.

And then the recursive case:

- Otherwise return the sum of the n - 1 Fibonacci number and the n - 2 Fibonacci number.

Let’s turn those words into code: