**Principle of Recursion**

Recursion is an approach to solving problems using a function that calls itself as a subroutine.

You might wonder how we can implement a function that calls itself. The trick is that each time a recursive function calls itself, it reduces the given problem into subproblems. The recursion call continues until it reaches a point where the subproblem can be solved without further recursion.

A recursive function should have the following properties so that it does not result in an infinite loop:

1. A simple base case (or cases) — a terminating scenario that does not use recursion to produce an answer.
2. A set of rules, also known as recurrence relation that reduces all other cases towards the base case.

Note that there could be multiple places where the function may call itself.

Example

Let's start with a simple programming problem:

Print a string in reverse order.

You can easily solve this problem iteratively, *i.e.* looping through the string starting from its last character. But how about solving it recursively?

First, we can define the desired function as printReverse(str[0...n-1]), where str[0] represents the first character in the string. Then we can accomplish the given task in two steps:

1. printReverse(str[1...n-1]): print the substring str[1...n-1] in reverse order.
2. print(str[0]): print the first character in the string.

Notice that we call the function itself in the first step, which by definition makes the function recursive.

Here is the code snippet:

A screenshot of a computer code

AI-generated content may be incorrect.

Next, you will find an exercise that is slightly different from the above example. You should try to solve it using recursion.

Note: For this exercise, we also provide a detailed solution in this Explore chapter.