

## 8. Recursion

Recursion is the process of repeating items in a self-similar way.



picture by [Pavlos Mavridis](#)

The same way you can call a function inside of other functions, you can call a function inside of itself. A function that calls itself is called a recursive function. Recursion is important because you can solve some problems by solving similar sub-problems. Recursive solutions usually have less code and are more elegant than their iterative equivalents if the problem you solve is recursive in nature.

Let's take a simple example. To print all the numbers from `1` to `N`, the first thing we have to do is print all the numbers from `1` to `N-1` and then print `N`.

```
func printFirstNumbers(_ N: Int) {  
    if N > 1 {  
        printFirstNumbers(N - 1)  
    }  
    print(N)  
}  
  
printFirstNumbers(3)  
// 1  
// 2  
// 3
```

Okay ... the example from above works ... but what really happens?

To understand what happens we can take a look at a modified version of the `printFirstNumbers` function. This version will print all the steps it takes.

```
func printFirstNumbers(_ N: Int) {
    print("start printFirstNumbers(\(N))")

    if N > 1 {
        print("printFirstNumbers(\(N)) calls printFirstNumbers(\(N-1))")

        printFirstNumbers(N - 1)
    }

    print("printFirstNumbers(\(N)) will print \(N)")

    print("end printFirstNumbers(\(N))")
}

printFirstNumbers(3)
// start printFirstNumbers(3)
// printFirstNumbers(3) calls printFirstNumbers(2)
// start printFirstNumbers(2)
// printFirstNumbers(2) calls printFirstNumbers(1)
// start printFirstNumbers(1)
// printFirstNumbers(1) will print 1
// end printFirstNumbers(1)
// printFirstNumbers(2) will print 2
// end printFirstNumbers(2)
// printFirstNumbers(3) will print 3
// end printFirstNumbers(3)
```

The computer knows where to continue the execution of `printFirstNumbers(2)` after `printFirstNumbers(1)` finishes by using a data structure known as a [call stack](#). The call stack keeps information about the currently active functions. When `printFirstNumbers(1)` starts executing `printFirstNumbers(3)` and `printFirstNumbers(2)` are still active and they need to resume control right after the if statement.

Notice that `printFirstNumbers(1)` did not call `printFirstNumbers` again. That's known as a base case. You need to have at least one base case inside a recursive function in order to prevent infinite calls - or what is known as stack overflow.

Let's take another example. This time instead of printing the numbers from `1` to `N` let's do it from `N` to `1`.

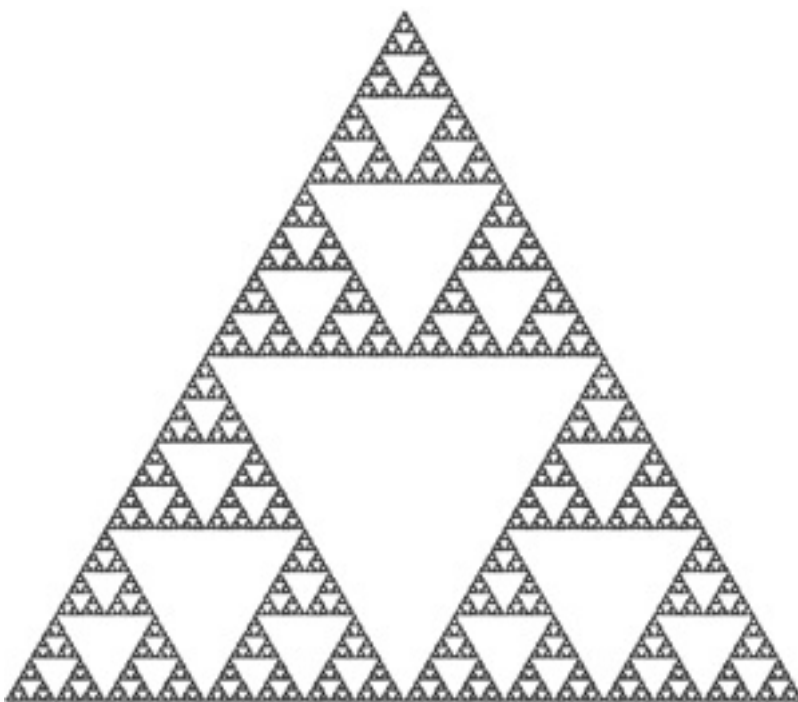
To count from `N` to `1` all we need to do is print `N` then print all the numbers from `N-1` to `1`.

```
func printFrom(_ N: Int) {
    print(N)
    if N > 1 {
        printFrom(N - 1)
    }
}
```

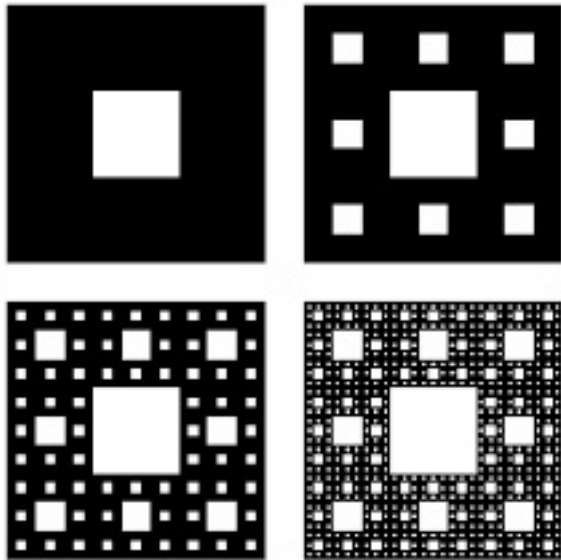
```
    }  
}  
  
printFrom(5)  
// 5  
// 4  
// 3  
// 2  
// 1
```

You can find another example of recursion if you google [recursion](#). The results page will ask you "Did you mean: recursion" which will take you to the same page...

Here are some visual examples of recursion. The Sierpinski triangle and carpet.

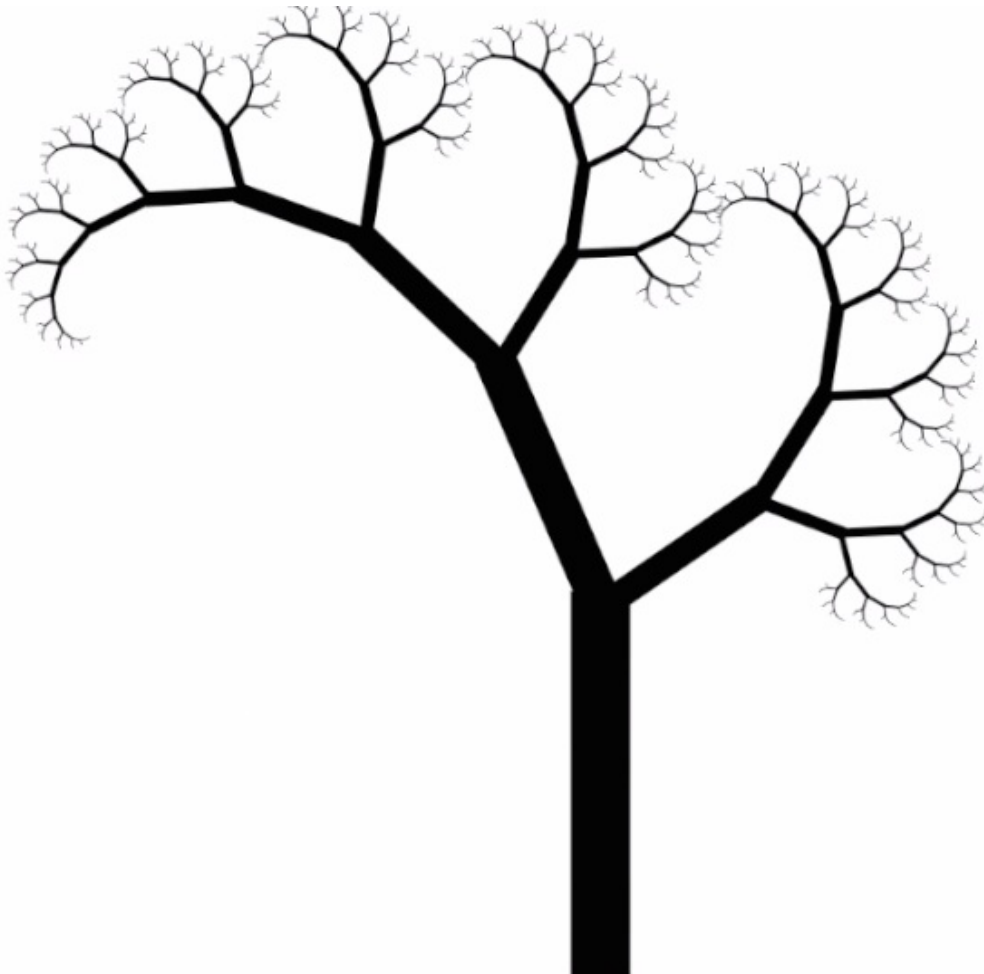



Sierpinski triangle



Sierpinski carpet

And [here](#) you can find a recursive drawing editor made by Toby Schachman. It's super easy to use - and a lot of fun. If you didn't understand what recursion is all about I highly encourage you to take a few minutes to play with it.



 An example image that could be generated using recursive drawing

Things to remember about recursion:

- you can call a function inside of itself
- you always have at least one base case in order to prevent the function calling itself infinite times