Introduction to Recursion and Memoization

Hands down, recursion is one of the coolest programming techniques. Once we're able to formulate a solution to a problem using solutions of subproblems, we can implement that using recursion. Recursion is intriguing, a bit enigmatic, but also highly expressive.

While recursion is very powerful, it unfortunately runs into runtime stack overflow for large-size problems, making it ineffective when it's much needed. A technique called tail call optimization can solve this problem, and Kotlin provides support for that.

Recursion plays a significant role in programming, but algorithms often memoize data to achieve significant performance improvements. Kotlin doesn't provide built-in support for memoization, but, using the techniques you've learned so far, you can build it quite easily with great fluency.

In this chapter, we'll explore the power of recursion, where it falls short, and how tail call optimization can address those issues. Then we'll look at techniques to memoize the results of function calls to improve performance, while keeping the elegance of recursion.