Wrapping Up

The power of recursion is sadly not usable for large-sized problems due to potential stack overflow error. Kotlin provides `tailrec` to perform tail call optimization on specially written recursions. By converting the recursion into iteration under the hood, the language provides the capability for you to enjoy the power of recursion without suffering the consequences of stack overflow. And, even though the language doesn't directly provide a memoization function, using the features of the language, we can build the ability to elegantly memoize or cache results of computations. By using this approach, we can greatly reduce the computation time of programs that make use of the algorithmic technique called dynamic programming. We've seen how to implement memoization at the function level and also how to devise a delegate-based solution.

In this chapter we've seen the elegance of recursion and memorization. In the next part, we'll explore another charming technique that also has impact on performance and ease of programming—the relatively new innovation in Kotlin—coroutines.