Wrapping Up

Coroutines, built upon the concept of continuations, provide a great way to program concurrency. Coroutines are functions with multiple entry points, and they carry state between invocations. These functions can call into each other and resume execution from where they left off in the previous call. Coroutines also may yield the flow of control to other pending tasks that may need the thread and other resources to execute. You may vary the thread of execution of coroutines and, using `async()` and `await()`, perform tasks in parallel and receive the result in the future.

All these concepts nicely lay a foundation for creating powerful abstractions for concurrent programming. We can apply these to creating infinite sequences and also powerful iterators over boundless collections. We can also use coroutines to program asynchronous applications, which we'll focus on in the next chapter.
