Reasons to Love Kotlin

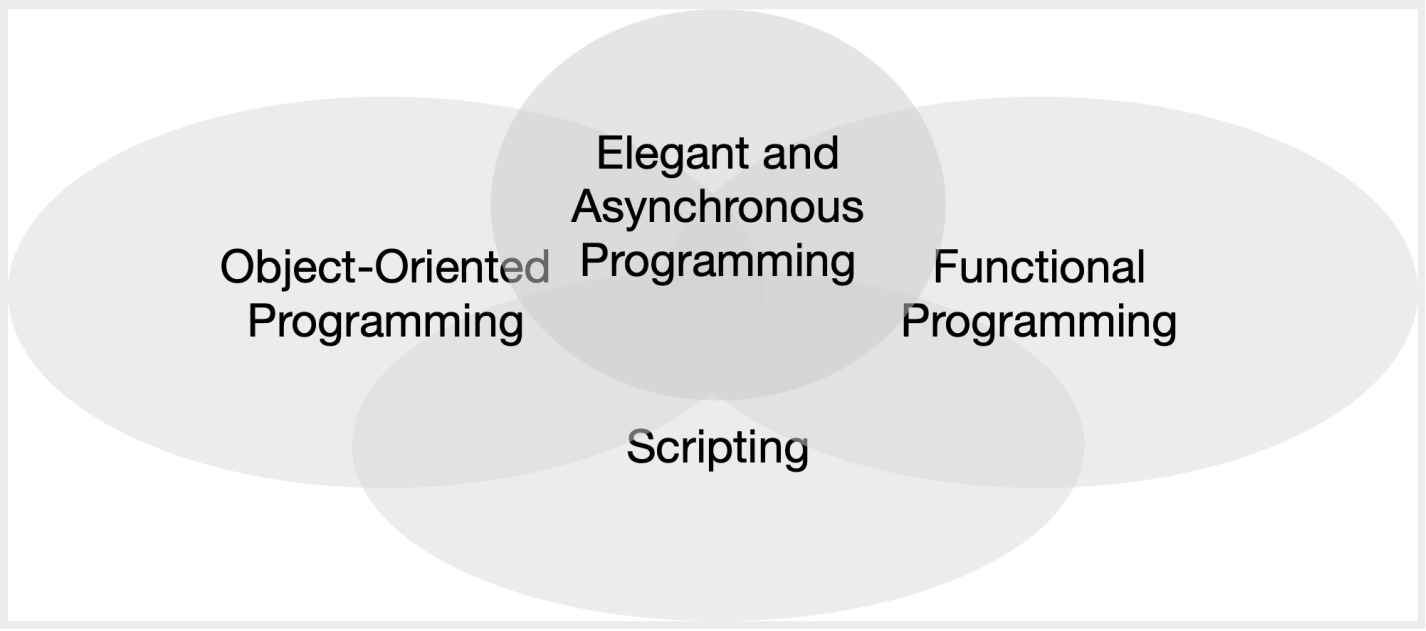
We'll cover the following

Once you dig in, Kotlin feels more like a Swiss Army knife than a cocktail—you can do so much with this language with so little code. The language supports multiple paradigms. It's statically typed with a healthy dose of strong type inference. It may be compiled to Java bytecode, transpiled to JavaScript, or it may target native binaries using Kotlin/Native. It is highly fluent and elegant, and it's a charm to work with. Let's further explore the reasons to adopt Kotlin.

Multi-Paradigm programming

Kotlin treats us like an adult, it offers choices and lets us pick the approach that's best suited for the problem at hand. The language has few ceremonies; you're not required to write everything in classes nor are you required to compile every piece of code. The language is largely unopinionated and offers different programming paradigms for you to choose or even intermix.

You can see the different programming paradigms supported in Kotlin in the following figure.



You can write procedural code—that is, code and functions directly in a file—like in JavaScript or C, and you can run it as a script, without any extra compilation steps, but with the exceptional type safety you expect from a statically typed language. The benefit is that you can do rapid prototyping of your ideas or illustrate how a particular design pattern may be used, but without being drowned in the ceremonies that other languages often impose. That gives you the shortest time from idea to demo.

Much like in Java, you can create classes and write object-oriented code in Kotlin, but without much boilerplate code. Thus, it takes less code to achieve the same results as in Java. Kotlin guides you along to create your hierarchy of classes intentionally rather than accidentally. Classes are final by default, and if you intend a class to serve as a base class, you must specify that explicitly. Also, delegation has a language-level syntax, so we can select prudently between inheritance and delegation.

Though the mainstream world has predominantly used the imperative style of programming, code written using the functional style is less complex, more expressive, concise, elegant, and fluent. Kotlin provides exceptional support for both the imperative and functional style of programming. You can readily benefit from the key functional capabilities you're used to from other languages that support the paradigm.

You can make immediate use of the elegance and low ceremony of Kotlin syntax to create internal domain-specific languages (DSLs). In addition to creating your own

fluent APIs, you can also benefit from fluency in a number of different libraries, for example, the Kotlin API for the Spring framework.

In addition to programming concurrency using the Java Development Kit (JDK), you may also write asynchronous programs using Kotlin's coroutines. This feature is critical for applications that make use of cloud services or are deployed as microservices; it allows you to interact efficiently with other services to exchange data asynchronously.

Statically typed with type inference

Statically typed languages offer compile-time type safety, but Kotlin walks a few extra miles to prevent common errors that are likely in other statically typed languages. For instance, the Kotlin type system distinguishes nullable types from non-nullable types. It also has very strong type inference, in the same vein of languages like Scala, F#, and Haskell. You don't have to spend your time keying in type details that are obvious to everyone looking at the code. At the same time, when the type may not be 100 percent clear, Kotlin requires that you specify it. It's not overly zealous—it supports type inference to the right measure, so we can be productive and at the same time the code can be type safe.

One language for full-stack development

Just like `javac` compiles Java source code to bytecode to run on the Java Virtual Machine (JVM), `kotlinc-jvm` compiles the Kotlin code to bytecode to run on virtual machines. You can write your server-side code and Android applications using Kotlin, and target the specific version of the virtual machine that you'd like to use for deployment. Thus your Spring code on the back end and your Android or iOS native code on the devices all may be written using the same language. Where necessary, you may also intermix Kotlin code with Java code—no legacy code has to be left behind.

Kotlin also transpiles to JavaScript. You can write Kotlin code that may transform to JavaScript and run in Node.js on the server side, or in browsers on the web front end.

Using Kotlin/Native, you can compile code to native binary to run on targeted platforms and to WebAssembly to run within browsers.

Fluent and elegant

Some languages impose high ceremony and force you to create boilerplate code. Some developers argue that IDEs remove the burden of having to write that code manually. True, but even if the IDEs were to vomit that boilerplate code, your team has to spend the time and effort maintaining that code each day. Languages like Scala, Groovy, and Ruby synthesize code that programmers will have to otherwise write. Likewise, Kotlin creates a few things for you, like fields, getters, and setters, implicitly following the JavaBean convention. Less effort, better results.

Kotlin makes a few things optional. For example, a semicolon is optional. Not having to place the; symbol leads to a more fluent syntax—a must for creating easy-to-read internal DSLs. Kotlin also provides an infix annotation that we can use, making the dot and parenthesis optional. With these capabilities you can write fluent and elegant code like this:

```
operate robot {
  turn left
  turn right
  move forward
}
```

Yes, it's not some fiction—that's real Kotlin code; you'll learn to create your own fluent code like this later in the course, without the need for any parsers or external tools.

The next lesson explains why you should choose Kotlin.