Kotlin (programming language)

Kotlin (/ˈkɒtliːn/)[2] is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin is designed to interoperate fully with Java, and the JVM version of Kotlin's standard library depends on the Java Class Library,[3] but type inference allows its syntax to be more concise. Kotlin mainly targets the JVM, but also compiles to JavaScript (for e.g. frontend web applications using React[4]) or native code (via LLVM), e.g. for native iOS apps sharing business logic with Android apps.[5] Language development costs are borne by JetBrains, while the Kotlin Foundation protects the Kotlin trademark.[6]

On 7 May 2019, Google announced that the Kotlin programming language is now its preferred language for Android app developers.[7] Since the release of Android Studio 3.0 in October 2017, Kotlin has been included as an alternative to the standard Java compiler. The Android Kotlin compiler targets Java 6 by default, but lets the programmer choose to target Java 8 up to 13, for optimization,[8] or more features.[9]

Contents

Kotlin

Paradigm	Multi-paradigm: object-oriented, functional, imperative, block structured, declarative, generic, reflective, concurrent
Designed by	JetBrains
Developer	JetBrains
First appeared	22 July 2011
Stable release	1.4.21 / 7 December 2020[1]
Typing discipline	Inferred, static, strong
Platform	▪ Android ▪ JVM ▪ iOS, macOS, watchOS, and tvOS ▪ Windows ▪ Linux ▪ JavaScript (Kotlin/JS (https://kotlinlang.org/docs/reference/js-overview.html)) ▪ WebAssembly ▪ LLVM (Kotlin/Native (https://kotlinla

	ng.org/docs/reference/native-overview.html))
OS	Cross-platform
License	Apache License 2.0
Filename extensions	.kt, .kts, .ktm
Website	kotlinlang.org (https://kotlinlang.org/)
Influenced by	
C# · Eiffel · Gosu · Groovy · Java · ML · Python · Scala · Swift	

History

In July 2011, JetBrains unveiled Project Kotlin, a new language for the JVM, which had been under development for a year.[10] JetBrains lead Dmitry Jemerov said that most languages did not have the features they were looking for, with the exception of Scala. However, he cited the slow compilation time of Scala as a deficiency.[10] One of the stated goals of Kotlin is to compile as quickly as Java. In February 2012, JetBrains open sourced the project under the Apache 2 license.[11]

The name comes from Kotlin Island, near St. Petersburg. Andrey Breslav mentioned that the team decided to name it after an island just like Java was named after the Indonesian island of Java[12] (though the programming language Java was perhaps named after the coffee).[13]

JetBrains hopes that the new language will drive IntelliJ IDEA sales.[14]

Kotlin v1.0 was released on 15 February 2016.[15] This is considered to be the first officially stable release and JetBrains has committed to long-term backwards compatibility starting with this version.

At Google I/O 2017, Google announced first-class support for Kotlin on Android.[16]

Kotlin v1.2 was released on 28 November 2017.[17] Sharing code between JVM and JavaScript platforms feature was newly added to this release (as of version 1.4 multiplatform programming is an alpha feature[18] upgraded from "experimental"). A full-stack demo has been made with the new Kotlin/JS Gradle Plugin.[19][20]

Kotlin v1.3 was released on 29 October 2018, bringing coroutines for asynchronous programming.

On 7 May 2019, Google announced that the Kotlin programming language is now its preferred language for Android app developers.[7]

Kotlin v1.4 was released in August 2020, with e.g. some slight changes to the support for Apple's platforms, i.e. to the Objective-C/Swift interop.[21]

Design

Development lead Andrey Breslav has said that Kotlin is designed to be an industrial-strength object-oriented language, and a "better language" than Java, but still be fully interoperable with Java code, allowing companies to make a gradual migration from Java to Kotlin.[22]

Semicolons are optional as a statement terminator; in most cases a newline is sufficient for the compiler to deduce that the statement has ended.[23]

Kotlin variable declarations and parameter lists have the data type come after the variable name (and with a colon separator), similar to BASIC, Pascal and TypeScript.

Variables in Kotlin can be read-only, declared with the `val` keyword, or mutable, declared with the `var` keyword.[24]

Class members are public by default, and classes themselves are final by default, meaning that creating a derived class is disabled unless the base class is declared with the `open` keyword.

In addition to the classes and member functions (equivalent to methods) of object-oriented programming, Kotlin also supports procedural programming with the use of functions.[25] Kotlin functions (and constructors) support default arguments, variable-length argument lists, named arguments and overloading by unique signature. Class member functions are virtual, i.e. dispatched based on the runtime type of the object they are called on.

Kotlin 1.3 adds (experimental) support for contracts[26] (inspired by Eiffel's design by contract[27] programming paradigm)

Syntax

Procedural programming style

Kotlin relaxes Java's restriction of allowing static methods and variables to exist only within a class body. Static objects and functions can be defined at the top level of the package without needing a redundant class level. For compatibility with Java, Kotlin provides a `JvmName` annotation which specifies a class name used when the package is viewed from a Java project. For example, `@file:JvmName("JavaClassName")`.

Main entry point

As in C, C++, C#, Java, and Go, the entry point to a Kotlin program is a function named "main", which may be passed an array containing any command-line arguments. (This is optional since Kotlin 1.3[28]). Perl and Unix shell style string interpolation is supported. Type inference is also supported.

```
// Hello, World! example
fun main() {
    val scope = "World"
    println("Hello, $scope!")
}

fun main(args: Array<String>) {
    for (arg in args) {
        println(arg)
    }
}
```

Extension functions

Similar to C#, Kotlin allows a user to add functions to any class without the formalities of creating a derived class with new functions. Instead, Kotlin adds the concept of an extension function which allows a function to be "glued" onto the public function list of any class without being formally placed inside of the class. In other

words, an extension function is a helper function that has access to all the public interface of a class which it can use to create a new function interface to a target class and this function will appear exactly like a function of the class, appearing as part of code completion inspection of class functions. For example:

```
1 package MyStringExtensions
2
3 fun String.lastChar(): Char = get(length - 1)
4
5 >>> println("Kotlin".lastChar())
```

By placing the preceding code in the top-level of a package, the String class is extended to include a `lastChar` function that was not included in the original definition of the String class.

```
1 // Overloading '+' operator using an extension function
2 operator fun Point.plus(other: Point): Point {
3     return Point(x + other.x, y + other.y)
4 }
5
6 >>> val p1 = Point(10, 20)
7 >>> val p2 = Point(30, 40)
8 >>> println(p1 + p2)
9 Point(x=40, y=60)
```

Unpack arguments with spread operator

Similar to Python, the spread operator asterisk (*) unpacks an array's contents as comma-separated arguments to a function:

```
1 fun main(args: Array<String>) {
2     val list = listOf("args: ", *args)
3     println(list)
4 }
```

Destructuring declarations

Destructuring declarations decompose an object into multiple variables at once, e.g. a 2D coordinate object might be *destructured* into two integers x and y.

For example, the `Map.Entry` object supports destructuring to simplify access to its key and value fields:

```
1 for ((key, value) in map) {
2     println("$key: $value")
3 }
```

Nested functions

Kotlin allows local functions to be declared inside of other functions or methods.

```
1 class User(val id: Int, val name: String, val address: String)
2
3 fun saveUserToDb(user: User) {
4     fun validate(user: User, value: String, fieldName: String) {
5         require(value.isNotEmpty()) { "Can't save user ${user.id}: empty $fieldName" }
6     }
```

```
 7
 8     validate(user, user.name, "Name")
 9     validate(user, user.address, "Address")
10     // Save user to the database
11     ...
12 }
```

Classes are final by default

In Kotlin, to derive a new class from a base class type, the base class needs to be explicitly marked as "open". This is in contrast to most object-oriented languages such as Java where classes are open by default.

Example of a base class that is open to deriving a new subclass from it.

```
 1 // open on the class means this class will allow derived classes
 2 open class MegaButton  {
 3
 4     // no-open on a function means that
 5     //    polymorphic behavior disabled if function overridden in derived class
 6     fun disable() { ... }
 7
 8     // open on a function means that
 9     //    polymorphic behavior allowed if function is overridden in derived class
10     open fun animate() { ... }
11 }
12
13 class GigaButton: MegaButton {
14
15     // Explicit use of override keyword required to override a function in derived class
16     override fun animate() { println("Giga Click!") }
17 }
```

Abstract classes are open by default

Abstract classes define abstract or "pure virtual" placeholder functions that will be defined in a derived class. Abstract classes are open by default.

```
 1 // No need for the open keyword here, it's already open by default
 2 abstract class Animated {
 3
 4     // This virtual function is already open by default as well
 5     abstract fun animate()
 6
 7     open fun stopAnimating() { }
 8
 9     fun animateTwice() { }
10 }
```

Classes are public by default

Kotlin provides the following keywords to restrict visibility for top-level declaration, such as classes, and for class members:

```
    public, internal, protected, and private.
```

When applied to a class member:

```
public (default): Visible everywhere
internal:         Visible in a module
protected:        Visible in subclasses
private:          Visible in a class
```

When applied to a top-level declaration

```
public (default):  Visible everywhere
internal:          Visible in a module
private:           Visible in a file
```

Example:

```
// Class is visible only to current module
internal open class TalkativeButton : Focusable {
    // method is only visible to current class
    private   fun yell() = println("Hey!")

    // method is visible to current class and derived classes
    protected fun whisper() = println("Let's talk!")
}
```

Primary constructor vs. secondary constructors

Kotlin supports the specification of a "primary constructor" as part of the class definition itself, consisting of an argument list following the class name. This argument list supports an expanded syntax on Kotlin's standard function argument lists, that enables declaration of class properties in the primary constructor, including visibility, extensibility and mutability attributes. Additionally, when defining a subclass, properties in super-interfaces and super-classes can be overridden in the primary constructor.

```
// Example of class using primary constructor syntax
// (Only one constructor required for this class)
open class PowerUser : User (
    protected val nickname: String,
    final override var isSubscribed: Boolean = true)
    {
         ...
    }
```

However, in cases where more than one constructor is needed for a class, a more general constructor can be used called **secondary constructor syntax** which closely resembles the constructor syntax used in most object-oriented languages like C++, C#, and Java.

```
// Example of class using secondary constructor syntax
// (more than one constructor required for this class)
class MyButton : View {

    // Constructor #1
    constructor(ctx: Context) : super(ctx) {
        // ...
    }

    // Constructor #2
    constructor(ctx: Context, attr: AttributeSet) : super(ctx, attr) {
        // ...
    }
}
```

Data Class

Kotlin provides Data Classes to define classes whose primary purpose is storing data. In Java, such classes are expected to provide a standard assortment of functions such as `equals`, `toString`, and `hashCode`, and `toString`. Kotlin's `data class` construct is similar to normal classes except in that these key functions are automatically generated from the class properties. Data classes are not required to declare any methods, though each must have at least one property. A data class often is written without a body, though it is possible to give a data class any methods or secondary constructors that are valid for any other class. The `data` keyword is used before the `class` keyword to define a data class.[29]

```
fun main(args: Array) {
    // create a data class object like any other class object
    var book1 = Book("Kotlin Programming", 250)
    println(book1)
    // output: Book(name=Kotlin Programming, price=250)
}

// data class with parameters and their optional default values
data class Book(val name: String = "", val price: Int = 0)
```

Kotlin interactive shell

```
$ kotlinc-jvm
type :help for help; :quit for quit
>>> 2 + 2
4
>>> println("Hello, World!")
Hello, World!
>>>
```

Kotlin as a scripting language

Kotlin can also be used as a scripting language. A script is a Kotlin source file (.kts) with top level executable code.

```
// list_folders.kts
import java.io.File
val folders = File(args[0]).listFiles { file -> file.isDirectory() }
folders?.forEach { folder -> println(folder) }
```

Scripts can be run by passing the `-script` option and the corresponding script file to the compiler.

```
$ kotlinc -script list_folders.kts "path_to_folder_to_inspect"
```

Null Safety

Kotlin makes a distinction between nullable and non-nullable data types. All nullable objects must be declared with a "?" postfix after the type name. Operations on nullable objects need special care from developers: null-check must be performed before using the value. Kotlin provides null-safe operators to help developers:

- `?.` (safe navigation operator) can be used to safely access a method or property of a possibly null object. If the object is null, the method will not be called and the expression evaluates to

null.

- ?: (null coalescing operator) often referred to as the Elvis operator:

```
fun sayHello(maybe: String?, neverNull: Int) {
    // use of elvis operator
    val name: String = maybe ?: "stranger"
    println("Hello $name")
}
```

An example of the use of the safe navigation operator:

```
// returns null if...
// - foo() returns null,
// - or if foo() is non-null, but bar() returns null,
// - or if foo() and bar() are non-null, but baz() returns null.
// vice versa, return value is non-null if and only if foo(), bar() and baz() are non-null
foo()?.bar()?.baz()
```

Lambdas

Kotlin provides support for higher order functions and anonymous functions or lambdas.[30]

```
// the following function takes a lambda, f, and executes f passing it the string, "lambda"
// note that (s: String) -> Unit indicates a lambda with a String parameter and Unit return type
fun executeLambda(f: (s: String) -> Unit) {
    f("lambda")
}
```

Lambdas are declared using braces, `{ }`. If a lambda takes parameters, they are declared within the braces and followed by the `->` operator.

```
// the following statement defines a lambda that takes a single parameter and passes it to the println function
val l = { c : Any? -> println(c) }
// lambdas with no parameters may simply be defined using { }
val l2 = { print("no parameters") }
```

Complex "hello world" example

```
fun main(args: Array<String>) {
    greet {
        to.place
    }.print()
}

// Inline higher-order functions
inline fun greet(s: () -> String) : String = greeting andAnother s()

// Infix functions, extensions, type inference, nullable types,
// lambda expressions, labeled this, Elvis operator (?:)
infix fun String.andAnother(other : Any?) = buildString()
{
    append(this@andAnother); append(" "); append(other ?: "")
}

// Immutable types, delegated properties, lazy initialization, string templates
val greeting by lazy { val doubleEl: String = "ll"; "he${doubleEl}o" }

// Sealed classes, companion objects
sealed class to { companion object { val place = "world"} }
```

```
22
23 // Extensions, Unit
24 fun String.print() = println(this)
```

Tools

- IntelliJ IDEA has plug-in support for Kotlin.[31] IntelliJ IDEA 15 was the first version to bundle the Kotlin plugin in the IntelliJ Installer, and provide Kotlin support out of the box.[32]
- JetBrains also provides a plugin for Eclipse.[33][34]
- Integration with common Java build tools is supported including Apache Maven,[35] Apache Ant,[36] and Gradle.[37]
- Android Studio (based on IntelliJ IDEA) has official support for Kotlin, starting from Android Studio 3.[38]
- Emacs has a Kotlin Mode in its Melpa package repository.
- Vim has a plugin maintained on Github.[39]
- Json2Kotlin (https://json2kotlin.com/) generates POJO style native Kotlin code for web service response mapping.

Applications

When Kotlin was announced as an official Android development language at Google I/O in May of 2017, it became the third language fully supported for Android, in addition to Java and C++.[40] As of 2020, Kotlin is still most widely used on Android, with Google estimating that 70% of the top 1000 apps on the Play Store are written in Kotlin. Google itself has 60 apps written in Kotlin, including Maps and Drive. Many Android apps, such as Google's Home, are in the process of being migrated to Kotlin, and so use both Kotlin and Java. Kotlin on Android is seen as beneficial for its null-pointer safety as well as for its features that make for shorter, more readable code.[41]

In addition to its prominent use on Android, Kotlin is gaining traction in server-side development. The Spring Framework officially added Kotlin support with version 5 on 04 January 2017.[42] To further support Kotlin, Spring has translated all its documentation to Kotlin and added built-in support for many Kotlin-specific features such as coroutines.[43] In addition to Spring, JetBrains has produced a Kotlin-first framework called Ktor for building web applications.[44]

In 2020, JetBrains found in a survey of developers who use Kotlin that 56% were using Kotlin for mobile apps, while 47% were using it for a web back-end. Just over a third of all Kotlin developers said that they were migrating to Kotlin from another language. Most Kotlin users were targeting Android (or otherwise on the JVM), with only 6% using Kotlin Native.[45]

Adoption

In 2018, Kotlin was the fastest growing language on GitHub with 2.6 times more developers compared to 2017.[46] It's the fourth most loved programming language according to the 2020 Stack Overflow Developer Survey.[47]

Kotlin was also awarded the O'Reilly Open Source Software Conference Breakout Award for 2019.[48]

Many companies/organisations have used Kotlin for backend development:

- Google[49]
- Norwegian Tax Administration[50]
- Gradle[51]
- Amazon[52]
- Square[53]
- JetBrains[54]
- Flux[55]
- Allegro[56]
- OLX[57]
- Shazam[58]
- Pivotal[59]
- Rocket Travel[60]
- Meshcloud[61]
- Zalando[62]

Some companies/organisations have used Kotlin for web development:

- JetBrains[63]
- Data2viz[64]
- Fritz2[65]
- Barclay's Bank[66]

A number of companies have publicly stated using Kotlin:

- DripStat[67]
- Basecamp[68]
- Pinterest[69]
- Coursera[70]
- Netflix[71]
- Uber[72]
- Square[73]
- Trello[74]
- Duolingo[75]
- Corda, a distributed ledger developed by a consortium of well-known banks (such as Goldman Sachs, Wells Fargo, J.P. Morgan, Deutsche Bank, UBS, HSBC, BNP Paribas, Société Générale), has over 90% Kotlin code in its codebase.[76]

See also

- Comparison of programming languages

References

1. "JetBrains/kotlin" (https://github.com/JetBrains/kotlin/releases/latest). *GitHub*.
2. "What is the correct English pronunciation of Kotlin?" (https://discuss.kotlinlang.org/t/what-is-the-correct-english-pronunciation-of-kotlin/2050). 16 October 2019. Retrieved 9 November 2019.
3. "kotlin-stdlib" (https://kotlinlang.org/api/latest/jvm/stdlib/index.html). *kotlinlang.org*. JetBrains. Retrieved 20 April 2018.
4. "Kotlin for JavaScript - Kotlin Programming Language" (https://kotlinlang.org/docs/reference/js-overview.html). *Kotlin*. Retrieved 20 August 2020.
5. "Kotlin for cross-platform mobile development" (https://www.jetbrains.com/lp/mobilecrossplatform/). *JetBrains: Developer Tools for Professionals and Teams*. Retrieved 20 August 2020.
6. "Kotlin Foundation - Kotlin Programming Language" (https://kotlinlang.org/foundation/kotlin-foundation.html). *Kotlin*.
7. "Kotlin is now Google's preferred language for Android app development" (http://social.techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/). *TechCrunch*. Retrieved 8 May 2019.
8. "Kotlin FAQ" (https://kotlinlang.org/docs/reference/faq.html). "Kotlin lets you choose the version of JVM for execution. By default, the Kotlin/JVM compiler produces Java 6 compatible bytecode. If you want to make use of optimizations available in newer versions of Java, you can explicitly specify the target Java version from 8 to 13. Note that in this case the resulting bytecode might not run on lower versions."
9. "What's New in Kotlin 1.4 - Kotlin Programming Language" (https://kotlinlang.org/docs/reference/whatsnew14.html). *Kotlin*. Retrieved 20 August 2020. "Kotlin can now generate type annotations in the JVM bytecode (target version 1.8+) [..] Note that the type annotations from the standard library aren't emitted in the bytecode for now because the standard library is compiled with the target version 1.6."
10. Krill, Paul (22 July 2011). "JetBrains readies JVM language Kotlin" (https://www.infoworld.com/d/application-development/jetbrains-readies-jvm-based-language-167875). *InfoWorld*. Archived (https://web.archive.org/web/20190907161741/https://www.infoworld.com/article/2622405/jetbrains-readies-jvm-based-language.html) from the original on 7 September 2019. Retrieved 2 February 2014.
11. Waters, John (22 February 2012). "Kotlin Goes Open Source" (https://adtmag.com/articles/2012/02/22/kotlin-goes-open-source.aspx). *ADTmag.com*. 1105 Enterprise Computing Group. Archived (https://web.archive.org/web/20140218225151/https://adtmag.com/articles/2012/02/22/kotlin-goes-open-source.aspx) from the original on 18 February 2014. Retrieved 2 February 2014.
12. Mobius (8 January 2015), *Андрей Бреслав — Kotlin для Android: коротко и ясно* (https://www.youtube.com/watch?v=VU_L2_XGQ9s), retrieved 28 May 2017
13. Kieron Murphy (4 October 1996). "So why did they decide to call it Java?" (https://www.javaworld.com/article/2077265/core-java/so-why-did-they-decide-to-call-it-java-.html). *JavaWorld*. Archived (https://web.archive.org/web/20190315171946/http://www.javaworld.com/article/2077265/so-why-did-they-decide-to-call-it-java-.html) from the original on 15 March 2019. Retrieved 14 October 2017.
14. "Why JetBrains needs Kotlin" (https://blog.jetbrains.com/kotlin/2011/08/why-jetbrains-needs-kotlin/). "we expect Kotlin to drive the sales of IntelliJ IDEA"
15. "Kotlin 1.0 Released: Pragmatic Language for JVM and Android | Kotlin Blog" (https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/). *Blog.jetbrains.com*. 15 February 2016. Retrieved 11 April 2017.
16. Shafirov, Maxim (17 May 2017). "Kotlin on Android. Now official" (https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/). "Today, at the Google I/O keynote, the Android team announced first-class support for Kotlin."
17. "Kotlin 1.2 Released: Sharing Code between Platforms | Kotlin Blog" (https://blog.jetbrains.com/kotlin/2017/11/kotlin-1-2-released/). *blog.jetbrains.com*. 28 November 2017.

18. "Multiplatform Projects - Kotlin Programming Language" (https://kotlinlang.org/docs/reference/multiplatform.html). *Kotlin*. Retrieved 20 August 2020. "Working on all platforms is an explicit goal for Kotlin, but we see it as a premise to a much more important goal: sharing code between platforms. With support for JVM, Android, JavaScript, iOS, Linux, Windows, Mac and even embedded systems like STM32, Kotlin can handle any and all components of a modern application."
19. "Kotlin/kotlin-full-stack-application-demo" (https://github.com/Kotlin/kotlin-full-stack-application-demo). Kotlin. 3 April 2020. Retrieved 4 April 2020.
20. "Kotlin full stack app demo: update all involving versions to work with 1.3.70 release" (https://youtrack.jetbrains.com/issue/KT-37029). *youtrack.jetbrains.com*. Retrieved 4 April 2020.
21. "What's New in Kotlin 1.4 - Kotlin Programming Language" (https://kotlinlang.org/docs/reference/whatsnew14.html). *Kotlin*. Retrieved 20 August 2020. "In 1.4.0, we slightly change the Swift API generated from Kotlin with respect to the way exceptions are translated."
22. "JVM Languages Report extended interview with Kotlin creator Andrey Breslav" (https://zeroturnaround.com/rebellabs/jvm-languages-report-extended-interview-with-kotlin-creator-andrey-breslav/). *Zeroturnaround.com*. 22 April 2013. Retrieved 2 February 2014.
23. "Semicolons" (https://confluence.jetbrains.com/display/Kotlin/Grammar#Grammar-Semicolons). *jetbrains.com*. Retrieved 8 February 2014.
24. "Basic Syntax" (https://kotlinlang.org/docs/reference/basic-syntax.html#defining-variables). *Kotlin*. Jetbrains. Retrieved 19 January 2018.
25. "functions" (https://confluence.jetbrains.com/display/Kotlin/Functions). *jetbrains.com*. Retrieved 8 February 2014.
26. "What's New in Kotlin 1.3 - Kotlin Programming Language" (https://kotlinlang.org/docs/reference/whatsnew13.html). *Kotlin*. Retrieved 4 April 2020.
27. "Design by Contract (DbC) design considerations" (https://discuss.kotlinlang.org/t/design-by-contract-dbc-design-considerations/1321). *Kotlin Discussions*. 16 August 2012. Retrieved 4 April 2020. "Implement the full semantics of Eiffel DbC and improve upon it."
28. "Kotlin Examples: Learn Kotlin Programming By Example" (https://play.kotlinlang.org/byExample/01_introduction/01_Hello%20world).
29. "Introduction to Data Classes in Kotlin" (https://www.callicoder.com/kotlin-data-classes/).
30. "Higher-Order Functions and Lambdas" (https://kotlinlang.org/docs/reference/lambdas.html). *Kotlin*. Jetbrains. Retrieved 19 January 2018.
31. "Kotlin :: JetBrains Plugin Repository" (https://plugins.jetbrains.com/plugin/6954-kotlin). *Plugins.jetbrains.com*. 31 March 2017. Retrieved 11 April 2017.
32. "What's New in IntelliJ IDEA 2017.1" (https://www.jetbrains.com/idea/whatsnew/). *Jetbrains.com*. Retrieved 11 April 2017.
33. "Getting Started with Eclipse Neon – Kotlin Programming Language" (https://kotlinlang.org/docs/tutorials/getting-started-eclipse.html). *Kotlinlang.org*. 10 November 2016. Retrieved 11 April 2017.
34. "JetBrains/kotlin-eclipse: Kotlin Plugin for Eclipse" (https://github.com/JetBrains/kotlin-eclipse). GitHub. Retrieved 11 April 2017.
35. "Using Maven – Kotlin Programming Language" (https://kotlinlang.org/docs/reference/using-maven.html). *kotlinlang.org*. Retrieved 9 May 2017.
36. "Using Ant – Kotlin Programming Language" (https://kotlinlang.org/docs/reference/using-ant.html). *kotlinlang.org*. Retrieved 9 May 2017.
37. "Using Gradle – Kotlin Programming Language" (https://kotlinlang.org/docs/reference/using-gradle.html). *kotlinlang.org*. Retrieved 9 May 2017.
38. "Kotlin and Android" (https://developer.android.com/kotlin). *Android Developers*.
39. "udalov/kotlin-vim: Kotlin plugin for Vim. Featuring: syntax highlighting, basic indentation, Syntastic support" (https://github.com/udalov/kotlin-vim). GitHub. Retrieved 30 August 2019.

40. Lardinois, Frederic (17 May 2017). "Google makes Kotlin a first-class language for writing Android apps" (https://techcrunch.com/2017/05/17/google-makes-kotlin-a-first-class-language-for-writing-android-apps/). *techcrunch.com*. Retrieved 28 June 2018.
41. "Kotlin programming language: How Google is using it to squash the code bugs that cause most crashes" (https://www.zdnet.com/article/google-were-using-kotlin-programming-language-to-squash-the-bugs-that-cause-most-crashes/). *ZDNet*.
42. "Introducing Kotlin support in Spring Framework 5.0" (https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0). *Spring*. Pivotal. Retrieved 29 September 2020.
43. "The State of Kotlin Support in Spring" (https://blog.jetbrains.com/kotlin/2020/08/the-state-of-kotlin-support-in-spring/). *JetBrains*. Retrieved 6 December 2020.
44. "Review of Microservices Frameworks: A Look at Spring Boot Alternatives" (https://dzone.com/articles/not-only-spring-boot-a-review-of-alternatives). *DZone*.
45. "Kotlin Programming - The State of Developer Ecosystem 2020" (https://www.jetbrains.com/lp/devecosystem-2020/kotlin/). *JetBrains*. Retrieved 29 September 2020.
46. "The state of the Octoverse" (https://web.archive.org/web/20190322190823/https://octoverse.github.com/projects). Archived from the original (https://octoverse.github.com/projects) on 22 March 2019. Retrieved 24 July 2019.
47. "Stack Overflow Developer Survey 2020" (https://insights.stackoverflow.com/survey/2020#most-loved-dreaded-and-wanted). Retrieved 28 May 2020.
48. "Kotlin wins Breakout Project of the Year award at OSCON '19" (https://blog.jetbrains.com/kotlin/2019/07/kotlin-wins-breakout-project-of-the-year-award-at-oscon-19/). Retrieved 24 July 2019.
49. "State of Kotlin on Android" (https://www.youtube.com/watch?v=AgPj1Q6D--c&feature=youtu.be&t=309). *YouTube*. Retrieved 29 September 2020.
50. "KotlinConf 2019: Kotlin Runs Taxes in Norway by Jarle Hansen & Anders Mikkelsen" (https://www.youtube.com/watch?v=K8XxaAba65g&list=PLQ176FUIyIUY6SKGl3Cj9yeYibBuRr3Hl&index=22). *YouTube*. Retrieved 29 September 2020.
51. "Gradle Kotlin DSL Primer" (https://docs.gradle.org/current/userguide/kotlin_dsl.html). *docs.gradle.org*. Retrieved 29 September 2020.
52. "QLDB at Amazon" (https://talkingkotlin.com/qldb/). *Talking Kotlin*. Retrieved 29 September 2020.
53. "Going Full Kotlin Multiplatform" (https://talkingkotlin.com/going-full-kotlin-multiplatform/). *Talking Kotlin*. Retrieved 29 September 2020.
54. "Kotless" (https://talkingkotlin.com/kotless/). *Talking Kotlin*. Retrieved 29 September 2020.
55. "Using Kotlin for backend development at Flux" (https://talkingkotlin.com/Using-Kotlin-for-backend-development-at-Flux/). *Talking Kotlin*. Retrieved 29 September 2020.
56. "Kotlin at Allegro" (https://talkingkotlin.com/kotlin-at-allegro/). *Talking Kotlin*. Retrieved 29 September 2020.
57. "Greenfield Kotlin at OLX" (https://talkingkotlin.com/greenfield-kotlin-at-olx/). *Talking Kotlin*. Retrieved 29 September 2020.
58. "Kotlin at Shazam" (https://talkingkotlin.com/kotlin-at-shazam/). *Talking Kotlin*. Retrieved 29 September 2020.
59. "Application Monitoring with Micrometer" (https://talkingkotlin.com/application-monitoring-with-micrometer/). *Talking Kotlin*. Retrieved 29 September 2020.
60. "Groovy and Kotlin Interop at Rocket Travel" (https://talkingkotlin.com/groovy-and-kotlin-interop-at-rocket-travel/). *Talking Kotlin*. Retrieved 29 September 2020.
61. "Kotlin on the backend at Meshcloud" (https://talkingkotlin.com/kotlin-on-the-backend-at-meshcloud/). *Talking Kotlin*. Retrieved 29 September 2020.
62. "Zally - An API Linter" (https://talkingkotlin.com/Zally-An-API-Linter/). *Talking Kotlin*. Retrieved 29 September 2020.

63. "KotlinConf 2019: Kotlin in Space by Maxim Mazin" (https://www.youtube.com/watch?v=JnmHqKLgYY4). *YouTube*. Retrieved 29 September 2020.
64. "KotlinConf 2017 - Frontend Kotlin from the Trenches by Gaetan Zoritchak" (https://www.youtube.com/watch?v=1Pu0TYJJ2Tw&list=PLQ176FUIyIUY6UK1cgVsbdPYA3X5WLam5&index=14). *YouTube*. Retrieved 29 September 2020.
65. "Fritz2" (https://talkingkotlin.com/fritz2/). *Talking Kotlin*. Retrieved 29 September 2020.
66. "Java/Kotlin Developer - Barclays - Prague - Wizbii" (https://www.wizbii.com/company/barclays/job/convertibles-trading-system-developer). *Wizbii.com*. Retrieved 29 September 2020.
67. "Kotlin in Production – What works, Whats broken" (https://blog.dripstat.com/kotlin-in-production-the-good-the-bad-and-the-ugly-2/). *Blog.dripstat.com*. 24 September 2016. Retrieved 11 April 2017.
68. "How we made Basecamp 3's Android app 100% Kotlin – Signal v. Noise" (https://m.signalvnoise.com/how-we-made-basecamp-3s-android-app-100-kotlin-35e4e1c0ef12). *Signal v. Noise*. 29 April 2017. Retrieved 1 May 2017.
69. "Droidcon NYC 2016 - Kotlin in Production" (https://www.youtube.com/watch?v=mDpnc45Wwll). Retrieved 24 July 2019.
70. "Becoming bilingual@coursera" (https://medium.com/coursera-engineering/becoming-bilingual-coursera-d8048dce73e3). Retrieved 24 July 2019.
71. "Rob Spieldenner on twitter" (https://twitter.com/robspieldenner/status/708355228832178176). Retrieved 24 July 2019.
72. "2017 Who's using Kotlin?" (https://www.reddit.com/r/androiddev/comments/5sihp0/2017_whos_using_kotlin/ddfmkf7/). Retrieved 24 July 2019.
73. "square/sqldelight" (https://github.com/square/sqldelight). Retrieved 24 July 2019.
74. "Dan Lew on Twitter" (https://twitter.com/danlew42/status/809065097339564032). Retrieved 24 July 2019.
75. "Duolingo on Twitter" (https://twitter.com/duolingo/status/1247876630984474626). Retrieved 13 April 2020.
76. "Kotlin 1.1 Released with JavaScript Support, Coroutines and more" (https://blog.jetbrains.com/kotlin/2017/03/kotlin-1-1/). Retrieved 1 May 2017.

External links

- Official website (https://kotlinlang.org/)