en.wikipedia.org

Kotlin (programming language) - Wikipedia

21-27 minutos

Kotlin

Paradigm	Multi-paradigm
Designed by	JetBrains
Developer	JetBrains
First appeared	2011
Stable release	1.3.61 / 27 November 2019; 2 months ago[1]
Typing discipline	Inferred, static, strong
Platform	JVM, JavaScript, LLVM
OS	Cross-platform
License	Apache License 2.0

Filename extensions	• .kt • .kts
Website	kotlinlang.org
Influenced by	

- C#
- Gosu
- Groovy
- Java
- ML
- Python
- Scala
- Swift

Kotlin ()[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 its 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 or native code (via LLVM). Language development costs are borne by JetBrains, while the Kotlin Foundation protects the Kotlin trademark.[4]

On 7 May 2019, Google announced that the Kotlin programming language is now its preferred language for Android app developers.[5] 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 lets the user choose between targeting Java 6 or Java 8 compatible bytecode.[6]

History[edit]

In July 2011, JetBrains unveiled Project Kotlin, a new language for the JVM, which had been under development for a year.[7] 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.[7] 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.[8]

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[9] (though the programming language Java was perhaps named after the coffee).[10]

JetBrains hopes that the new language will drive IntelliJ IDEA sales.[11]

Kotlin v1.0 was released on 15 February 2016.[12] 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.[13]

Kotlin v1.2 was released on 28 November 2017.[14] Sharing code between JVM and Javascript platforms feature was newly added to this release.

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.[15]

Design[edit]

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.[16]

Semicolons are optional as a statement terminator; in most cases a newline is sufficient for the compiler to deduce that the statement has ended.[17]

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

Variables in Kotlin can be immutable, declared with the val keyword, or mutable, declared with the var keyword.[18]

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 methods (called member functions in Kotlin) of object-oriented programming, Kotlin also supports procedural programming with the use of functions.[19] 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.

Syntax[edit]

Functional programming style[edit]

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[edit]

As in 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[20]). 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 methods[edit]

Similar to C#, Kotlin allows a user to add methods to any class without the formalities of creating a derived class with new methods. Instead, Kotlin adds the concept of an extension method which allows a function to be "glued" onto the public method list of any class without being formally placed inside of the class. In other words, an extension method is a helper method that has access to all the public interface of a class which it can use to create a new method interface to a target class and this method will appear exactly like a method of the class, appearing as part of code completion inspection of class methods. For example:

```
package MyStringExtensions

fun String.lastChar(): Char = get(length - 1)

>>> println("Kotlin".lastChar())
```

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

```
// Overloading '+' operator using an extension method
operator fun Point.plus(other: Point): Point {
    return Point(x + other.x, y + other.y)
}

>>> val p1 = Point(10, 20)
>>> val p2 = Point(30, 40)
>>> println(p1 + p2)
Point(x=40, y=60)
```

Unpack arguments with spread operator[edit]

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

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

Deconstructor methods[edit]

Not to be confused with the destructor method common in object oriented languages.

A deconstructor's job is to decompose a class object into a tuple of elemental objects. For example, a 2D coordinate class might be deconstructed into a tuple of integer x and integer y.

For example, the collection object contains a deconstructor method that splits each collection item into an index and an element variable:

```
for ((index, element) in collection.withIndex()) {
     println("$index: $element")
}
```

Nested functions[edit]

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

```
class User(val id: Int, val name: String, val address: String)

fun saveUserToDb(user: User) {
    fun validate(user: User, value: String, fieldName: String) {
        if (value.isEmpty()) {
            throw IllegalArgumentException("Can't save user ${user.id}: empty $fieldName")
        }
    }

    validate(user, user.name, "Name")
    validate(user, user.address, "Address")
    // Save user to the database
    ...
}
```

Classes are final by default[edit]

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.

```
// open on the class means this class will allow derived classes
open class MegaButton  {

    // no-open on a function means that
    //    polymorphic behavior disabled if function overridden in derived class
    fun disable() { ... }

    // open on a function means that
    //    polymorphic behavior allowed if
function is overridden in derived class
    open fun animate() { ... }
}

class GigaButton: MegaButton {

    // Explicit use of override keyword
required to override a function in derived class
    override fun animate() { println("Giga
Click!") }
}
```

Abstract classes are open by default[edit]

Abstract classes define abstract or "Pure Virtual" placeholder function that will be defined in a derived class. Abstract classes are open by default.

```
// No need for the open keyword here, it’s
already open by default
abstract class Animated {

    // This virtual function is already open
by default as well
    abstract fun animate()

    open fun stopAnimating() { }

    fun animateTwice() { }
}
```

Classes are public by default[edit]

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[edit]

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[edit]

Kotlin provides Data Classes to define classes that store only properties. In Java programming, classes that store only properties are not unusual, but regular classes are used for this purpose. Kotlin has given provision to exclusively define classes that store properties alone. These data classes do not have any methods but only properties. A data class does not contain a body, unlike a regular class. **data** keyword is used before **class** keyword to define a data class.

```
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)
```

Anko library[edit]

Anko is a library specifically created for Kotlin to help build Android UI applications.[21]

```
fun Activity.showAreYouSureAlert(process: () -> Unit) {
    alert(
        title   = "Are you sure?",
        message = "Are you really sure?")
        {
            positiveButton("Yes") { process() }
            negativeButton("No") { cancel() }
        }
}
```

Kotlin interactive shell[edit]

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

Kotlin as a scripting language[edit]

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) }
```

To run a script, you pass the `-script` option to the compiler with the corresponding script file:

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

Kotlin features in an overly 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"} }

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

Variables in Kotlin can be immutable, declared with the val keyword or mutable, declared with the var keyword.[18]

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()
```

Kotlin provides support for higher order functions and anonymous functions or lambdas.[22]

```
// 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") }
```

Tools[edit]

- IntelliJ IDEA has plug-in support for Kotlin.[23] IntelliJ IDEA 15 is the first version to bundle the Kotlin plugin in the IntelliJ Installer, and provide Kotlin support out of the box.[24]
- JetBrains also provides a plugin for Eclipse.[25][26]
- Integration with common Java build tools is supported including Apache Maven,[27] Apache Ant,[28] and Gradle.[29]
- Android Studio (based on IntelliJ IDEA) has official support for Kotlin, starting from Android Studio 3.[30]
- Emacs has a Kotlin Mode in its Melpa package repository.
- Vim has a plugin maintained on Github[31]

Applications[edit]

One of the obvious applications of Kotlin is Android development. The platform was stuck on Java 7 for a while (with some contemporary language features made accessible through the use of Retrolambda[32] or the Jack toolchain[33]) and Kotlin introduces many improvements for programmers such as null-pointer safety, extension functions and infix notation. Accompanied by full Java compatibility and good IDE support (*Android Studio*[34]) it is intended to improve code readability, give an easier way to extend Android SDK classes and speed up development.[35]

Kotlin was announced as an official Android development language at Google I/O 2017. It became the third language fully supported for Android, in addition to Java and C++.[36]

Adoption[edit]

In 2018, Kotlin was the fastest growing language on GitHub with 2.6 times more developers compared to 2017.[37] It's the fourth most loved programming language according to the 2019 Stack Overflow survey.[38]

Kotlin was also awarded the O'Reilly Open Source Software Conference Breakout Award for 2019.[39]

A number of companies have publicly stated using Kotlin:

- DripStat[40]
- Basecamp[41]
- Pinterest[42]
- Coursera[43]
- Netflix[44]
- Uber[45]
- Square[46]
- Trello[47]
- 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 in its codebase.[48]

See also[edit]

- Comparison of programming languages

References[edit]

- This article contains quotations from Kotlin tutorials which are released under an Apache 2.0 license.

1. ^ https://github.com/JetBrains/kotlin/releases/latest
2. ^ *"What is the correct English pronunciation of Kotlin?". October 16, 2019. Retrieved November 9, 2019.*
3. ^ *"kotlin-stdlib". kotlinlang.org. JetBrains. Retrieved 20 April 2018.*
4. ^ “Kotlin Foundation”
5. ^ *"Kotlin is now Google's preferred language for Android app development". TechCrunch. Retrieved 2019-05-08.*
6. ^ *"Kotlin FAQ". “Kotlin lets you choose between generating Java 6 and Java 8 compatible bytecode. More optimal byte code may be generated for higher versions of the platform.”*
7. ^ Jump up to: *a b* *Krill, Paul (22 July 2011). "JetBrains readies JVM language Kotlin". infoworld.com. InfoWorld. Retrieved 2 February 2014.*
8. ^ *Waters, John (22 February 2012). "Kotlin Goes Open Source". ADTmag.com/. 1105 Enterprise Computing Group. Retrieved 2 February 2014.*
9. ^ Mobius (2015-01-08), *Андрей Бреслав — Kotlin для Android: коротко и ясно*, retrieved 2017-05-28
10. ^ Kieron Murphy (1996-10-04). *"So why did they decide to call it Java?". Archived* from the original on 2019-03-15.
11. ^ *"Why JetBrains needs Kotlin". “we expect Kotlin to drive the sales of IntelliJ IDEA”*
12. ^ *"Kotlin 1.0 Released: Pragmatic Language for JVM and Android | Kotlin Blog". Blog.jetbrains.com. 2016-02-15. Retrieved 2017-04-11.*
13. ^ *Shafirov, Maxim (17 May 2017). "Kotlin on Android. Now*

official". *“Today, at the Google I/O keynote, the Android team announced first-class support for Kotlin.”*

14. ^ *"Kotlin 1.2 Released: Sharing Code between Platforms | Kotlin Blog"*. *Blog.jetbrains.com. 2017-11-28.*
15. ^ *"Kotlin is now Google's preferred language for Android app development"*. *TechCrunch. Retrieved 2019-05-08.*
16. ^ *"JVM Languages Report extended interview with Kotlin creator Andrey Breslav"*. *Zeroturnaround.com. 22 April 2013. Retrieved 2 February 2014.*
17. ^ *"Semicolons"*. *jetbrains.com. Retrieved 8 February 2014.*
18. ^ Jump up to: ***a*** ***b*** *"Basic Syntax"*. *Kotlin. Jetbrains. Retrieved 19 January 2018.*
19. ^ *"functions"*. *jetbrains.com. Retrieved 8 February 2014.*
20. ^ *"Kotlin Examples: Learn Kotlin Programming By Example"*.
21. ^ Anko Github
22. ^ *"Higher-Order Functions and Lambdas"*. *Kotlin. Jetbrains. Retrieved 19 January 2018.*
23. ^ *"Kotlin :: JetBrains Plugin Repository"*. *Plugins.jetbrains.com. 2017-03-31. Retrieved 2017-04-11.*
24. ^ *"What's New in IntelliJ IDEA 2017.1"*. *Jetbrains.com. Retrieved 2017-04-11.*
25. ^ *"Getting Started with Eclipse Neon – Kotlin Programming Language"*. *Kotlinlang.org. 2016-11-10. Retrieved 2017-04-11.*
26. ^ *"JetBrains/kotlin-eclipse: Kotlin Plugin for Eclipse"*. *GitHub. Retrieved 2017-04-11.*
27. ^ *"Using Maven – Kotlin Programming Language"*. *kotlinlang.org. Retrieved 2017-05-09.*

28. ^ *"Using Ant – Kotlin Programming Language". kotlinlang.org. Retrieved 2017-05-09.*
29. ^ *"Using Gradle – Kotlin Programming Language". kotlinlang.org. Retrieved 2017-05-09.*
30. ^ https://developer.android.com/kotlin/index.html
31. ^ *"udalov/kotlin-vim: Kotlin plugin for Vim. Featuring: syntax highlighting, basic indentation, Syntastic support". GitHub. Retrieved 2019-08-30.*
32. ^ *"orfjackal/retrolambda: Backport of Java 8's lambda expressions to Java 7, 6 and 5". GitHub. Retrieved 2017-05-09.*
33. ^ *"Jack (Java Android Compiler Kit) | Android Open Source Project". source.android.com. Retrieved 2016-04-15.*
34. ^ *"JetBrains Plugin Repository :: Kotlin". plugins.jetbrains.com. Retrieved 2016-04-15.*
35. ^ *"Will Kotlin Replace Java?". themindstudios.com. Retrieved 2017-03-10.*
36. ^ Lardinois, Frederic (2017-05-17). *"Google makes Kotlin a first-class language for writing Android apps". techcrunch.com. Retrieved 2018-06-28.*
37. ^ *"The state of the Octoverse". Retrieved 2019-07-24.*
38. ^ *"Developer Survey Results". Retrieved 2019-07-24.*
39. ^ *"Kotlin wins Breakout Project of the Year award at OSCON '19". Retrieved 2019-07-24.*
40. ^ *"Kotlin in Production – What works, Whats broken". Blog.dripstat.com. 2016-09-24. Retrieved 2017-04-11.*
41. ^ *"How we made Basecamp 3's Android app 100% Kotlin – Signal v. Noise". Signal v. Noise. 2017-04-29. Retrieved 2017-05-01.*

42. ^ "Droidcon NYC 2016 - Kotlin in Production". *Retrieved 2019-07-24.*

43. ^ "Becoming bilingual@coursera". *Retrieved 2019-07-24.*

44. ^ "Rob Spieldenner on twitter". *Retrieved 2019-07-24.*

45. ^ "2017 Who's using Kotlin?". *Retrieved 2019-07-24.*

46. ^ "square/sqldelight". *Retrieved 2019-07-24.*

47. ^ "Dan Lew on twitter". *Retrieved 2019-07-24.*

48. ^ "Kotlin 1.1 Released with JavaScript Support, Coroutines and more". *Retrieved 2017-05-01.*

External links[edit]

- Official website