Kotlin Programming

Duration: 5 Days

Day 1

Get started with Kotlin

* Learn Kotlin fundamentals
* Install Kotlin
* Create your powerful application with Kotlin
* Is anything missing?

Kotlin Multiplatform

* Kotlin Multiplatform use cases
* How Kotlin Multiplatform works
* Get started

Kotlin for server side

* Frameworks for server-side development with Kotlin
* Deploying Kotlin server-side applications
* Products that use Kotlin on the server side
* Next steps

Kotlin for Android

Kotlin for JavaScript

* Use cases for Kotlin/JS
* Kotlin/JS frameworks
* Kotlin/JS, Today and Tomorrow
* Get started with Kotlin/JS
* New Kotlin/JS IR compiler
* Join the Kotlin/JS community

Kotlin Native

* Why Kotlin/Native?
* Target platforms
* Interoperability
* Sharing code between platforms
* How to get started

Kotlin for data science

* Interactive editors
* Libraries

Kotlin for competitive programming

* Simple example: Reachable Numbers problem
* Functional operators example: Long Number problem
* More tips and tricks
* Learning Kotlin

What's new in Kotlin 1.7.20

* Support for Kotlin K2 compiler plugins
* Language
* Kotlin/JVM
* Kotlin/Native
* Kotlin/JS
* Gradle
* Standard library
* Documentation updates
* Install Kotlin 1.7.20

Install the EAP Plugin for IntelliJ IDEA or Android Studio

* If you run into any problems

Configure your build for EAP

* Configure in Gradle
* Configure in Maven

Basic syntax

* Package definition and imports
* Program entry point
* Print to the standard output
* Functions
* Variables
* Creating classes and instances
* Comments
* String templates
* Conditional expressions
* for loop
* while loop
* when expression
* Ranges
* Collections
* Nullable values and null checks
* Type checks and automatic casts

Idioms

* Create DTOs (POJOs/POCOs)
* Default values for function parameters
* Filter a list
* Check the presence of an element in a collection
* String interpolation
* Instance checks
* Read-only list
* Read-only map
* Access a map entry
* Traverse a map or a list of pairs
* Iterate over a range
* Lazy property
* Extension functions
* Create a singleton
* Instantiate an abstract class
* If-not-null shorthand
* If-not-null-else shorthand
* Execute a statement if null
* Get first item of a possibly empty collection
* Execute if not null
* Map nullable value if not null
* Return on when statement
* try-catch expression
* if expression
* Builder-style usage of methods that return Unit
* Single-expression functions
* Call multiple methods on an object instance (with)
* Configure properties of an object (apply)
* Java 7's try-with-resources
* Generic function that requires the generic type information
* Nullable Boolean
* Swap two variables
* Mark code as incomplete (TODO)
* What's next?

Coding conventions

* Configure style in IDE
* Source code organization
* Naming rules
* Formatting
* Documentation comments
* Avoid redundant constructs
* Idiomatic use of language features
* Coding conventions for libraries

Day 2

Basic types

Numbers

* Integer types
* Floating-point types
* Literal constants for numbers
* Numbers representation on the JVM
* Explicit number conversions
* Operations on numbers

Booleans

Characters

Strings

* String literals
* String templates

Arrays

* Primitive type arrays

Unsigned integer types

* Unsigned arrays and ranges
* Unsigned integers literals
* Use cases

Type checks and casts

* is and !is operators
* Smart casts
* "Unsafe" cast operator
* "Safe" (nullable) cast operator
* Generics type checks and casts

Conditions and loops

* If expression
* When expression
* For loops
* While loops
* Break and continue in loops
* Returns and jumps
* Break and continue labels
* Return to labels

Exceptions

* Exception classes
* Checked exceptions
* The Nothing type
* Java interoperability
* Packages and imports
* Default imports
* Imports
* Visibility of top-level declarations

Classes

* Constructors
* Creating instances of classes
* Class members
* Inheritance
* Abstract classes
* Companion objects

Inheritance

* Overriding methods
* Overriding properties
* Derived class initialization order
* Calling the superclass implementation
* Overriding rules
* Properties
* Declaring properties
* Getters and setters
* Compile-time constants
* Late-initialized properties and variables
* Overriding properties
* Delegated properties

Interfaces

* Implementing interfaces
* Properties in interfaces
* Interfaces Inheritance
* Resolving overriding conflicts

Functional (SAM) interfaces

* SAM conversions
* Migration from an interface with constructor function to a functional interface
* Functional interfaces vs. type aliases

Visibility modifiers

* Packages
* Class members
* Modules

Day 3

Extensions

* Extension functions
* Extensions are resolved statically
* Nullable receiver
* Extension properties
* Companion object extensions
* Scope of extensions
* Declaring extensions as members
* Note on visibility

Data classes

* Properties declared in the class body
* Copying
* Data classes and destructuring declarations
* Standard data classes

Sealed classes

* Location of direct subclasses
* Sealed classes and when expression

Generics: in, out, where

* Variance
* Type projections
* Generic functions
* Generic constraints
* Type erasure
* Underscore operator for type arguments

Nested and inner classes

* Inner classes
* Anonymous inner classes

Enum classes

* Anonymous classes
* Implementing interfaces in enum classes
* Working with enum constants

Inline classes

* Members
* Inheritance
* Representation
* Inline classes vs type aliases
* Inline classes and delegation

Day 4

Object expressions and declarations

* Object expressions
* Object declarations

Delegation

* Overriding a member of an interface implemented by delegation

Delegated properties

* Standard delegates
* Delegating to another property
* Storing properties in a map
* Local delegated properties
* Property delegate requirements
* Translation rules for delegated properties
* Providing a delegate

Type aliases

Functions

* Function usage
* Function scope
* Generic functions
* Tail recursive functions

High-order functions and lambdas

* Higher-order functions
* Function types
* Lambda expressions and anonymous functions

Inline functions

* noinline
* Non-local returns
* Reified type parameters
* Inline properties
* Restrictions for public API inline functions

Operator overloading

* Unary operations
* Binary operations
* Infix calls for named functions

Type-safe builders

* How it works
* Scope control: @DslMarker
* Full definition of the com.example.html package
* Using builders with builder type inference
* Writing your own builders
* How builder inference works

Null safety

* Nullable types and non-null types
* Checking for null in conditions
* Safe calls
* Elvis operator
* The !! operator
* Safe casts
* Collections of a nullable type

Equality

* Structural equality
* Referential equality
* Floating-point numbers equality

This expressions

* Qualified this
* Implicit this

Asynchronous programming techniques

* Threading
* Callbacks
* Futures, promises, and others
* Reactive extensions
* Coroutines

Coroutines

* How to start
* Sample projects

Annotations

* Usage
* Constructors
* Instantiation
* Lambdas
* Annotation use-site targets
* Java annotations
* Repeatable annotations

Destructuring declarations

* Example: returning two values from a function
* Example: destructuring declarations and maps
* Underscore for unused variables
* Destructuring in lambdas

Day 5

Reflection

* JVM dependency
* Class references
* Callable references

Configure compilations

* Configure all compilations
* Configure compilations for one target
* Configure one compilation
* Create a custom compilation
* Use Java sources in JVM compilations
* Configure interop with native languages
* Compilation for Android
* Compilation of the source set hierarchy
* Some Java issues addressed in Kotlin
* What Java has that Kotlin does not
* What Kotlin has that Java does not

**Calling Java from Kotlin**

* Getters and setters
* Methods returning void
* Escaping for Java identifiers that are keywords in Kotlin
* Null-safety and platform types
* Mapped types
* Java generics in Kotlin
* Java arrays
* Java varargs
* Operators
* Checked exceptions
* Object methods
* Inheritance from Java classes
* Accessing static members
* Java reflection
* SAM conversions
* Using JNI with Kotlin
* Using Lombok-generated declarations in Kotlin

**Calling Kotlin from Java**

* Properties
* Package-level functions
* Instance fields
* Static fields
* Static methods
* Default methods in interfaces
* Visibility
* KClass
* Handling signature clashes with @JvmName
* Overloads generation
* Checked exceptions
* Null-safety
* Variant generics

Create a RESTful web service with a database using Spring Boot

* Before you start
* Bootstrap the project
* Explore the project build file
* Explore the Spring Boot application
* Create a data class and a controller
* Run the application
* Add database support
* Configure the database
* Execute HTTP requests

Test code using JUnit in JVM

* Add dependencies
* Add the code to test it
* Create a test
* Run a test

Mixing Java and Kotlin in one project

* Adding Java source code to an existing Kotlin project
* Adding Kotlin source code to an existing Java project
* Converting an existing Java file to Kotlin with J2K

Using Java records in Kotlin

* Using Java records from Kotlin code
* Declare records in Kotlin

Strings in Java and Kotlin

* Concatenate strings
* Build a string
* Create a string from collection items
* Set default value if the string is blank
* Replace characters at the beginning and end of a string
* Replace occurrences
* Split a string
* Take a substring
* Use multiline strings

Collections in Java and Kotlin

* Operations that are the same in Java and Kotlin
* Operations that differ a bit
* Operations that don't exist in Java's standard library
* Mutability
* Covariance
* Ranges and progressions
* Comparison by several criteria
* Sequences
* Removal of elements from a list
* Traverse a map
* Get the first and the last items of a possibly empty collection
* Create a set from a list
* Group elements
* Filter elements
* Collection transformation operations

Nullability in Java and Kotlin

* Support for nullable types
* Platform types
* Checking the result of a function call
* Default values instead of null
* Functions returning a value or null
* Aggregate operations
* Casting types safely