



Rapid Web API development with Kotlin and Ktor

About me



Trayan Iliev

- CEO of IPT – Intellectual Products & Technologies
<http://www.iproduct.org>
- Oracle® certified programmer 15+ Y
- end-to-end reactive fullstack apps with [Java](#), [ES6+](#), [TypeScript](#), [Angular](#), [React](#) and [Vue.js](#)
- 12+ years IT trainer: [Spring](#), [Java EE](#), [Node.js](#), [Express](#), [GraphQL](#), [SOA](#), [REST](#), [DDD](#) & [Reactive Microservices](#)
- Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker
- Organizer RoboLearn hackathons and IoT enthusiast

Course Schedule

- [\[Oct 4, 2021\]](#) Introduction to Kotlin
- [\[Oct 7, 2021\]](#) Kotlin basic syntax
- [\[Oct 11, 2021\]](#) Packages and imports
- [\[Oct 14, 2021\]](#) Classes and objects - I
- [\[Oct 18, 2021\]](#) Classes and objects - II
- [\[Oct 21, 2021\]](#) Functions and lambdas - I
- [\[Oct 25, 2021\]](#) Functions and lambdas - II
- [\[Oct 28, 2021\]](#) Collections and sequences
- [\[Nov 1, 2021\]](#) Exception handling. Annotations.

Course Schedule

- [\[Nov 4, 2021\]](#) Reflection. IO
- [\[Nov 8, 2021\]](#) Data Serialization. Interoperability with Java
- [\[Nov 11, 2021\]](#) Coroutines and concurrency
- [\[Nov 15, 2021\]](#) Coroutine Context and Dispatchers
- [\[Nov 18, 2021\]](#) Asynchronous Flow
- [\[Nov 22, 2021\]](#) Channels
- [\[Nov 25, 2021\]](#) Shared mutable state and concurrency
- [\[Nov 29, 2021\]](#) Test Driven Development (TDD). Logging
- [\[Dec 2, 2021\]](#) Databases and persistence. Spring Data in Kotlin

Course Schedule

- [\[Dec 6, 2021\]](#) Spring Boot, Spring MVC in Kotlin - I
- [\[Dec 9, 2021\]](#) Spring Boot, Spring MVC in Kotlin - II
- [\[Dec 13, 2021\]](#) Web Flux REST/ WebSocket API development in Kotlin
- [\[Dec 16, 2021\]](#) Rapid development of production REST APIs using Ktor - I
- [\[Dec 20, 2021\]](#) Rapid development of production REST APIs using Ktor - II

Where to Find The Code and Materials?

<https://github.com/iproduct/course-kotlin>

Kotlin - A modern programming language with less legacy debts

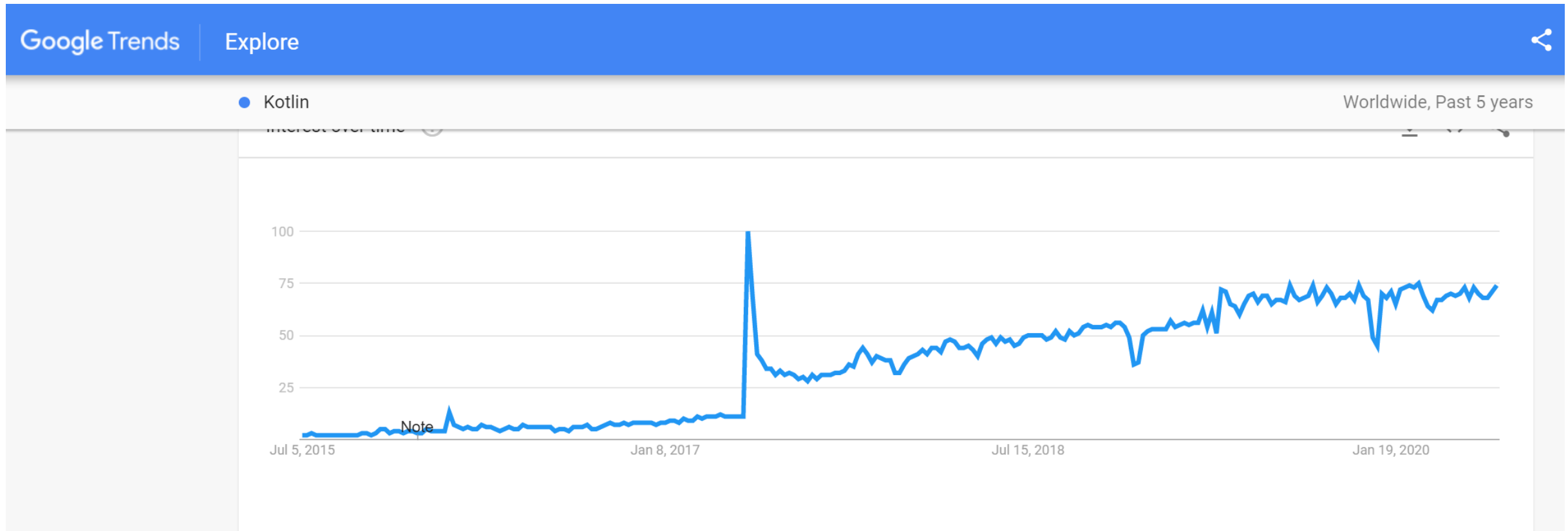
<https://kotlinlang.org/>

- Kotlin → since July 2011, 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.
- Kotlin docs admit how good Java is, but mention that the **Java** programming language has limitations and problems that are "**either impossible or very hard to fix due to backward-compatibility issues.**"
- Kotlin is a **statically typed** JVM-targeted language "**free of the legacy trouble and having the features so desperately wanted by the developers**", safer and more concise than Java, statically checking for pitfalls such as **null pointer dereference**, as well as **simpler than Scala**.

Kotlin Features

- [statically typed](#),
- [general-purpose programming language](#)
- [type inference](#) - allows more concise [syntax](#) than Java
- fully interoperate with [Java](#) (the [JVM](#) version of its [standard library](#) depends on the [Java Class Library](#))
- [cross-platform](#) - Kotlin mainly targets the JVM, but also compiles to [JavaScript](#) or [native code](#) (via [LLVM](#)).
- targeting [mobile, native](#) (including WebAssembly), [web, server-side, data science](#) and [cross-platform](#) application development
- open sourced under the [Apache 2 license](#).

Kotlin Popularity According to Google Trends



Source: [Google Trends](#)

Some Java Issues Addressed in Kotlin

- Null references are [controlled by the type system](#).
- [No raw types](#)
- Arrays in Kotlin are [invariant](#)
- Kotlin has proper [function types](#), as opposed to Java's SAM-conversions
- [Use-site variance](#) without wildcards
- Kotlin does not have checked [exceptions](#)

What Java Has that Kotlin Does Not

- Checked exceptions
- Primitive types that are not classes
- Static members
- Non-private fields
- Wildcard-types
- Ternary-operator `a ? b : c`

What Kotlin Has that Java Does Not - I

- Lambda expressions + Inline functions = performant custom control structures
- Extension functions
- Null-safety
- Smart casts
- String templates
- Properties
- Primary constructors

What Kotlin Has that Java Does Not - II

- First-class delegation
- Type inference for variable and property types
- Singletons
- Declaration-site variance & Type projections
- Range expressions
- Operator overloading
- Companion objects

What Kotlin Has that Java Does Not – III

- Data classes
- Separate interfaces for read-only and mutable collections
- Coroutines !!!

Kotlin Is Consize – Reduces the Boilerplate

// Create a POJO with getters, setters, `equals()`, `hashCode()`, `toString()` and `copy()` in a single line:

```
data class Customer(val name: String, val email: String, val company: String)
```

// Want a singleton? Create an object:

```
object ThisIsASingleton {  
    val companyName: String = "JetBrains"  
}
```

```
fun main() {
```

// Or filter a list using a lambda expression:

```
val positiveNumbers = listOf(0, 1, -1, 2, -2, 3, -3).filter { it > 0 }  
println(positiveNumbers);
```

```
val customerWithCompany = listOf(  
    Customer("Trayan Iliev", "office@iproduct.org", "IPT"),  
    Customer("John Smith", "john@nobrainer.com", "NoBrainer"),  
    Customer("Silvia Popova", "silvia@acme.com", "ACME Corp")  
).map({"${it.name} - ${it.company}")) // template strings and single argument lambdas iterator  
println(customerWithCompany); // => [Trayan Iliev - IPT, John Smith - NoBrainer, Silvia Popova - ACME Corp]  
}
```

Kotlin Is Safe – Avoids Null Pointer Exceptions

// Get rid of those pesky NullPointerExceptions

var output: String

output = **null** *// Compilation error*

// Kotlin protects you from mistakenly operating on nullable types

val *name*: String? = **null** *// Nullable type*

println(name.length()) *// Compilation error*

Kotlin Is Safe – Automatic Type Casting

```
data class Product(val name: String, val price: Double)
class Order(val number: Int, val products: List<Product>, val date: LocalDateTime = LocalDateTime.now() ){
    fun calculateTotal(): Double {
        return products
            .map { it.price }
            .reduce { acc, prod -> acc + prod }
    }
}
```

// And if you check a type is right, the compiler will auto-cast it for you

```
fun calculateTotal(obj: Any): Double? {
    if (obj is Order) return obj.calculateTotal()
    return 0.0
}
```

```
fun main() {
    val products = listOf(Product("Keyboard", 27.5), Product("Mouse", 17.1))
    val order = Order(1, products)
    println(calculateTotal((order))); // => 44.6
}
```

Kotlin Is Interoperable – JVM, Android

```
import io.reactivex.rxjava3.core.Flowable
import io.reactivex.rxjava3.schedulers.Schedulers
import kotlinx.coroutines.delay
import kotlinx.coroutines.runBlocking

fun main() = runBlocking {
    Flowable
        .fromCallable {
            Thread.sleep(1000) // imitate expensive computation
            "Done"
        }
        .subscribeOn(Schedulers.io())
        .observeOn(Schedulers.single())
        .subscribe(::println, Throwable::printStackTrace)
    delay(2000)
    println("Demo finished.")
}
```

Kotlin Is Interoperable – ... and in The Browser

```
fun getCommonWorldString() = "common-world"
```

```
import io.ktor.samples.fullstack.common.*  
import kotlin.browser.*
```

```
@Suppress("unused")  
@JsName("helloWorld")  
fun helloWorld(salutation: String) {  
    val message = "$salutation from Kotlin.JS ${getCommonWorldString()}"  
    document.getElementById("js-response")?.textContent = message  
}
```

```
fun main() {  
    document.addEventListener("DOMContentLoaded", {  
        helloWorld("Hi!")  
    })  
}
```

Kotlin Is Interoperable – ... Fullstack with Ktor as MPP

```
fun Application.main() {
    val currentDir = File(".").absoluteFile
    environment.log.info("Current directory: $currentDir")
    routing {
        get("/") {
            call.respondHtml {
                body {
                    +"Hello ${getCommonWorldString()} from Ktor"
                    div { id = "js-response" + "Loading..." }
                    script(src = "/static/ktor-samples-fullstack-mpp-frontend.js") {}
                }
            }
        }
        static("/static") { resource("ktor-samples-fullstack-mpp-frontend.js") }
    }
}

fun main(args: Array<String>) {
    embeddedServer(Netty, port = 8080) { main() }.start(wait = true)
}
```

Tool Support for Kotlin

Tool-friendly

Choose any Java IDE or build from the command line



IntelliJ IDEA

Bundled with both IntelliJ
IDEA Community Edition and
IntelliJ IDEA Ultimate

[Download](#) ➤ [Instructions](#)



Android Studio

Bundled with Android Studio

[Download](#) ➤ [Instructions](#)



Eclipse

Install the plugin from the
Eclipse Marketplace

[Instructions](#)



Standalone Compiler

Use any editor and build from
the command line

[Instructions](#)

Who Uses Kotlin? – to Mention just a Few



Gradle is [introducing Kotlin](#) as a language for writing build scripts



Corda is an open-source distributed ledger platform, supported by major banks, and [built entirely in Kotlin](#)



Evernote recently [integrated Kotlin](#) into their Android client



Coursera Android app is [partially written](#) in Kotlin



Spring makes [use of Kotlin's language features](#) to offer more concise APIs



All new code in the [Trello Android](#) app is in Kotlin

Ktor

Asynchronous client / server framework utilizing Kotlin features such as DSLs and coroutines



What is Ktor?


- **Ktor** is an OSS Apache 2 licensed framework for building **asynchronous servers and clients in connected systems** using the power of **Kotlin** programming language.
- Drawing inspiration from other frameworks, such as **Wasabi** and **Kara**, to leverage to the maximum extent the Kotlin language features such as **DSLs** and **coroutines**.
- It is **low-ceremony** in that it requires very **little code and configuration** to get a system up and running.
- Currently, the **Ktor client** works on **all platforms** Kotlin targets, that is, **JVM, JavaScript, and Native**. Right now, **Ktor server-side** is restricted to the **JVM**.

Generate a Ktor project

Estimated reading time: 1 minute



NOTE: You can also use the [Ktor IntelliJ plugin](#) instead. This page can also be accessed at start.ktor.io.

 **Ktor Project Generator (1.3.2)**

Configuration

Gradle project ☐ with Wrapper ☒

Server Engine: Netty

Ktor 1.3.2

Group: com.example

Name: ktor-demo

Version: 0.0.1-SNAPSHOT

Swagger (Optional)

or

Server

Filter Server Features

Templating

☐ **HTML DSL** (ktor-html-builder)
Generate HTML using Kotlin code like a pure-core template engine
[Documentation](#)

☐ **CSS DSL** (org.jetbrains.kotlin-css-jvm:1.0.0-pre.31-kotlin-1.2.41)
Generate CSS using Kotlin code
[Documentation](#)

☐ **Freemarker** (ktor-freemarker)
Serve HTML content using Apache's FreeMarker template engine
[Documentation](#)

☐ **Velocity** (ktor-velocity)
Serve HTML content using Apache's Velocity template engine

☐ Show marked dependencies only

Client

Filter Client Features

HttpClient Engine

☐ **HttpClient Engine** (ktor-client-core, ktor-client-core-jvm)
Core of the HttpClient. Required for libraries.
[Documentation](#)

☐ **Apache HttpClient Engine** (ktor-client-apache)
Engine for the Ktor HttpClient using Apache. Supports HTTP 1.x and HTTP 2.0.
[Documentation](#)

☐ **CIO HttpClient Engine** (ktor-client-cio)
Engine for the Ktor HttpClient using CIO (Corroutine I/O). Only supports HTTP 1.x.
[Documentation](#)

☐ **Jetty HttpClient Engine** (ktor-client-jetty)

☐ Show marked dependencies only

Source:
<https://kotlinlang.org/>

Simple Web Service with Ktor

```
fun main() {  
    val server = embeddedServer(Netty, 8080) {  
        routing {  
            get("/hello") {  
                call.respondText("<h2>Hello from Ktor and Kotlin!</h2>", ContentType.Text.Html)  
            }  
        }  
    }  
    server.start(true)  
}
```

... And that's all :)

```
data class Product(val name: String, val price: Double, var id: Int)
```

```
object Repo: ConcurrentHashMap<Int, Product>() {
```

```
    private idCounter = AtomicInteger()
```

```
    fun addProduct(product: Product) {
```

```
        product.id = idCounter.incrementAndGet()
```

```
        put(product.id, product)
```

```
    }
```

```
}
```

```
fun main() {
```

```
    embeddedServer(Netty, 8080, watchPaths = listOf("build/classes"), module= Application::mymodule).start(true)
```

```
}
```

```
fun Application.mymodule() {
```

```
    install(DefaultHeaders)
```

```
    install(CORS) { maxAgeInSeconds = Duration.ofDays(1).toSeconds() }
```

```
    install(Compression)
```

```
    install(CallLogging)
```

```
    install(ContentNegotiation) {
```

```
        gson {
```

```
            setDateFormat(DateFormat.LONG)
```

```
            setPrettyPrinting()
```

```
        }
```

```
}
```

```
routing {
  get("/products") {
    call.respond(Repo.values)
  }
  get("/products/{id}") {
    try {
      val item = Repo.get(call.parameters["id"]?.toInt())
      if (item == null) {
        call.respond(
          HttpStatusCode.NotFound,
          """{"error":"Product not found with id = ${call.parameters["id"]}"}""")
      }
    } else {
      call.respond(item)
    }
  }
  catch(ex :NumberFormatException) {
    call.respond(HttpStatusCode.BadRequest,
      """{"error":"Invalid product id: ${call.parameters["id"]}"}""")
  }
}
```

```

post("/products") {
    errorAware {
        val product: Product = call.receive<Product>(Product::class)
        println("Received Post Request: $product")
        Repo.addProduct(product)
        call.respond(HttpStatusCode.Created, product)
    }
}
}
}
}

```

```

private suspend fun <R> PipelineContext<*, ApplicationCall>.errorAware(block: suspend () -> R): R? {
    return try {
        block()
    } catch (e: Exception) {
        call.respondText(
            ""{"error": "$e"}"",
            ContentType.parse("application/json"),
            HttpStatusCode.InternalServerError
        )
        null
    }
}

```

Ktor Applications

- **Ktor Server Application** is a custom program **listening to one or more ports** using a **configured server engine**, **composed by modules** with the application logic, that install **features, like routing, sessions, compression**, etc. to handle **HTTP/S 1.x/2.x** and **WebSocket** requests.
- **ApplicationCall** – the context for handling routes, or directly intercepting the pipeline – provides access to two main properties **ApplicationRequest** and **ApplicationResponse**, as well as **request parameters, attributes, authentication, session, typesafe locations**, and the **application** itself. Example:

```
intercept(ApplicationCallPipeline.Call) {  
    if (call.request.uri == "/")  
        call.respondHtml {  
            body {  
                a(href = "/products") { + "Go to /products" }  
            }  
        }  
}
```

Routing DSL Using Higher Order Functions

- **routing**, **get**, and **post** are all **higher-order functions** (functions that take other functions as parameters or return functions).
- Kotlin has a convention that **if the last parameter to a function is another function**, we can place this **outside of the brackets**
- **routing** is a **lambda with receiver** == higher-order function taking as parameter an **extension function** => anything enclosed within **routing** has access to members of the type **Routing**.
- **get** and **post** are functions of the **Routing** type => also **lambdas with receivers**, with own members, such as **call**.
- This combination of **conventions** and **functions** allows to create elegant DSLs, such as Ktor's **routing DSL**.

Features

- A **feature** is a **singleton** (usually a **companion object**) that you can **install and configure for a pipeline**.
- Ktor includes some **standard features**, but you **can add your own** or other features from the community.
- You can install features in **any pipeline**, like the **application** itself, or specific **routes**.
- Features are **injected into the request and response pipeline**. Usually, an application would have a series of features such as **DefaultHeaders** which add headers to every outgoing response, **Routing** which allows us to define routes to handle requests, etc.

Installing Features

Using *install*:

```
fun Application.main() {  
    install(DefaultHeaders)  
    install(CallLogging)  
    install(Routing) {  
        get("/") {  
            call.respondText("Hello, World!")  
        }  
    }  
}
```



Using routing DSL:

```
fun Application.main() {  
    install(DefaultHeaders)  
    install(CallLogging)  
    routing {  
        get("/") {  
            call.respondText("Hello, World!")  
        }  
    }  
}
```

Standard Features I

- [Authenticating Clients](#)
- [Enable Automatic HEAD Responses](#)
- [Basic and Form authentication](#)
- [Controlling cache headers](#)
- [CallId](#)
- [Log the client requests](#)
- [Client/Server Sessions](#)
- [Enable HTTP Compression Facilities](#)

Standard Features - II

- [Easy '304 Not Modified' Responses](#)
- [Content conversion based on Content-Type and Accept headers](#)
- [Cookie/Header Sessions](#)
- [Enable Cross-Origin Resource Sharing \(CORS\)](#)
- [Data Conversion](#)
- [Send Headers Automatically](#)
- [Digest authentication](#)
- [DoubleReceive for request body](#)

Standard Features - III

- [XForwardedHeaderSupport \(Reverse Proxy Support\)](#)
- [Using Freemarker Templates](#)
- [JSON support using Gson](#)
- [Enable HTTP Strict Transport Security](#)
- [Emit HTML with a DSL](#)
- [Redirect HTTP requests to HTTPS](#)
- [JSON support using Jackson](#)
- [JWT and JWK authentication](#)
- [LDAP authentication](#)

Standard Features - IV

- [Type-safe Routing](#)
- [Metrics with Micrometer metrics](#)
- [Metrics with Dropwizard metrics](#)
- [Using Mustache Templates](#)
- [OAuth authentication](#)
- [Streaming Movies and Other Content](#)
- [Using Pebble Templates](#)
- [Structured Handling of HTTP Requests](#)
- [JSON support using `kotlinx.serialization`](#)

Standard Features - V

- [Handle Conversations with Sessions](#)
- [Add an URL for shutting down the server](#)
- [Serving Static Content](#)
- [Handle Exceptions and Customize Status Pages](#)
- [Session Storages](#)
- [Templates](#), [Using Thymeleaf Templates](#), [Using Velocity Templates](#)
- [Session Transformers](#)
- [Webjars support](#)
- [WebSockets](#)

Kotlin Basics

Basic types, type checks and casts, control flow



Basic types: Numbers

- **Ktor module** is a user-defined **function receiving the Application class** that is in charge of **configuring the server pipeline, install features, registering routes, handling requests**, etc. (e.g. `com.example.ApplicationKt.module`)
- Example:

```
fun main(args: Array<String>): Unit = io.ktor.server.netty.EngineMain.main(args)
```

```
fun Application.module(testing: Boolean = false) {  
    install(DefaultHeaders)  
    install(CallLogging)  
    routing {  
        get("/") {  
            call.respondText("Hello, Ktor World!")  
        }  
    }  
}
```


Basic types: Numbers

- Integer types:

Type	Size (bits)	Min value	Max value
Byte	8	-128	127
Short	16	-32768	32767
Int	32	-2,147,483,648 (-2^{31})	2,147,483,647 ($2^{31} - 1$)
Long	64	-9,223,372,036,854,775,808 (-2^{63})	9,223,372,036,854,775,807 ($2^{63} - 1$)

- Examples:

```
val one = 1 // Int
```

```
val threeBillion = 3000000000 // Long
```

```
val oneLong = 1L // Long
```

```
val oneByte: Byte = 1
```

Basic types: Numbers

- Floating-point types:

Type	Size (bits)	Significant bits	Exponent bits	Decimal digits
Float	32	24	8	6-7
Double	64	53	11	15-16

- Examples:

```
val pi = 3.14 // Double
// val two: Double = 1 // Error: type mismatch
val twoDouble = 1.0 // Double
val e = 2.7182818284 // Double
val eFloat = 2.7182818284f // Float, actual value is 2.7182817
fun printDouble(d: Double) { print(d) }
val i = 1
val d = 1.0
printDouble(d)
// printDouble(i) // Error: Type mismatch
// printDouble(eFloat ) // Error: Type mismatch
```

Basic types: Numbers

- Literal constants:
 - Decimals: 123
 - Longs are tagged by a capital L: 123L
 - Hexadecimals: 0x0F
 - Binaries: 0b00001011
 - Octal literals are not supported.
 - Doubles by default: 123.5, 123.5e10
 - Floats are tagged by f or F: 123.5f
- Examples:
 - `val oneMillion = 1_000_000`
 - `val creditCardNumber = 1234_5678_9012_3456L`
 - `val socialSecurityNumber = 999_99_9999L`
 - `val hexBytes = 0xFF_EC_DE_5E`
 - `val bytes = 0b11010010_01101001_10010100_10010010`

Numbers representation on the JVM

- On the JVM platform, numbers are stored as primitive types: `int`, `double`, etc.
- Exceptions are cases when you create a **nullable** number reference such as `Int?` or **use generics** -> numbers are boxed in Java classes `Integer`, `Double`, etc. Nullable references to the same number can be different objects.
- Example:

```
val a: Int = 100
val boxedA: Int? = a
val anotherBoxedA: Int? = a
val b: Int = 10000
val boxedB: Int? = b
val anotherBoxedB: Int? = b
println(boxedA === anotherBoxedA) // true
println(boxedB === anotherBoxedB) // false
println(b == b) // Prints 'true'
println(boxedB == anotherBoxedB) // Prints 'true'
```

Explicit conversions

- `toByte(): Byte`
- `toShort(): Short`
- `toInt(): Int`
- `toLong(): Long`
- `toFloat(): Float`
- `toDouble(): Double`
- `toChar(): Char`
- Example:

// Hypothetical code, does not actually compile:

```
val x: Int? = 1 // A boxed Int (java.lang.Integer)
```

```
val y: Long? = x // implicit conversion yields a boxed Long (java.lang.Long)
```

```
print(x == y) // Surprise! This prints "false" as Long's equals() checks whether the other is Long as well
```

```
val b: Byte = 1 // OK, literals are checked statically
```

```
// val i: Int = b // ERROR
```

```
val i1: Int = b.toInt()
```

```
val l = 1L + 3 // Long + Int => Long
```

Operators

- Arithmetic: `+`, `-`, `*`, `/`, `%`
- Examples:

```
println(1 + 2)
```

```
println(2_500_000_000L - 1L)
```

```
println(3.14 * 2.71)
```

```
println(10.0 / 3)
```

```
val x = 5 / 2
```

```
//println(x == 2.5) // ERROR: Operator '==' cannot be applied to 'Int' and 'Double'
```

```
println(x == 2)
```

```
val x2 = 5L / 2
```

```
println(x2 == 2L)
```

```
val x3 = 5 / 2.toDouble()
```

```
println(x3 == 2.5)
```

Operators

- Bitwise operations:
 - `shl(bits)` – signed shift left
 - `shr(bits)` – signed shift right
 - `ushr(bits)` – unsigned shift right
 - `and(bits)` – bitwise and
 - `or(bits)` – bitwise or
 - `xor(bits)` – bitwise xor
 - `inv()` – bitwise inversion
- Example:
`val x4 = (1 shl 9) and 0x000FF00`

Floating-point numbers comparison

- Equality checks: `a == b` and `a != b`
- Comparison operators: `a < b`, `a > b`, `a <= b`, `a >= b`
- Range instantiation and range checks: `a..b`, `x in a..b`, `x !in a..b`
- NaN is considered equal to itself
- NaN is considered greater than any other element including `POSITIVE_INFINITY`
- `-0.0` is considered less than `0.0`

Unsigned integer types

- **UByte**: an unsigned 8-bit integer, ranges from 0 to 255
- **UShort**: an unsigned 16-bit integer, ranges from 0 to 65535
- **UInt**: an unsigned 32-bit integer, ranges from 0 to $2^{32} - 1$
- **ULong**: an unsigned 64-bit integer, ranges from 0 to $2^{64} - 1$
- **UByteArray**: an array of unsigned bytes
- **UShortArray**: an array of unsigned shorts
- **UIntArray**: an array of unsigned ints
- **ULongArray**: an array of unsigned longs
- Ranges and progressions: **UIntRange**, **UIntProgression**, **ULongRange**, **ULongProgression**
- When you use unsigned arrays, you'll get a warning that indicates that this feature is not stable yet. To remove the warning, opt in using the **@ExperimentalUnsignedTypes** annotation.

Unsigned integer types - Examples

```
val ub: UByte = 1u // UByte, expected type provided
```

```
val us: UShort = 1u // UShort, expected type provided
```

```
val ul: ULong = 1u // ULong, expected type provided
```

```
val a1 = 42u // UInt: no expected type provided, constant fits in UInt
```

```
val a2 = 0xFFFF_FFFF_FFFFu // ULong: no expected type provided, constant doesn't fit in UInt
```

```
val a3 = 1UL // ULong, even though no expected type provided and constant fits into UInt
```

```
val u: UIntProgression = 1U..10U step 2
```

```
val ua: UIntArray = UIntArray(10)
```

```
var j = 0
```

```
for (e in u) {
```

```
    ua[j] = e
```

```
    j++
```

```
}
```

```
for (e in ua) {
```

```
    print("$e, ") // prints: 1, 3, 5, 7, 9, 0, 0, 0, 0, 0,
```

```
}
```

Booleans

- The type **Boolean** represents boolean objects: **true** and **false**.
- Boolean has a nullable counterpart **Boolean?** that also has the null value.
- Built-in operations on booleans include:
 - **||** – disjunction (logical OR)
 - **&&** – conjunction (logical AND)
 - **!** – negation (logical NOT)
- **||** and **&&** work lazily, nullable references are boxed on JVM
- Examples:

```
val myTrue: Boolean = true
val myFalse: Boolean = false
val boolNull: Boolean? = null
println(myTrue || myFalse)    // true
println(myTrue && myFalse)     // false
println(!myTrue)              // false
```

Characters

- Characters are represented by the type **Char**. Character literals go in single quotes: `'1'`.
- Special characters start from an escaping backslash `\`. The following escape sequences are supported: `\t`, `\b`, `\n`, `\r`, `\'`, `\"`, `\\` and `\$`.
- To encode any other character, use the Unicode escape sequence syntax: `'\uFF00'`
- Examples:

```
val aChar: Char = 'a'  
println(aChar)  
println('\n') //prints an extra newline character  
println('\uFF00')
```

Strings

- Strings in Kotlin are represented by the type **String**. Generally, a string value is a sequence of characters in double quotes (").
- Elements of a string are characters that you can access via the indexing operation: **s[i]**. You can iterate over these characters with a **for loop**.
- Strings are **immutable**. All operations that transform strings return their results in a **new String object**, leaving the original string unchanged.

```
val s = "abcd 123"
for (c in s) {
    println(c)
}
val str = "abcd"
val str1 = str.toUpperCase()
println(str1) // Create and print a new String object
println(str === str1) // different objects
val s2 = "abc" + 1
println(s2 + "def")
```

String Literals & String Templates

```
val s3 = "Hello, world!\n"
```

```
val text = """
```

```
    |Tell me and I forget.
```

```
    |Teach me and I remember.
```

```
    |Involve me and I learn.
```

```
    |(Benjamin Franklin)
```

```
    |
```

```
    """
```

```
val text2 = text.trimIndent()
```

```
val text3 = text.trimMargin()
```

```
println(text2)
```

```
println(text3)
```

```
val i3 = 10
```

```
println("i = $i") // prints "i = 10"
```

```
val s4 = "abc"
```

```
println("$s.length is ${s.length}") // prints "abc.length is 3"
```

Arrays

- Arrays in Kotlin are represented by the **Array** class. It has **get** and **set** functions that turn into **[]** by operator overloading conventions, and the **size** property, along with other useful member functions:

```
abstract class Array<T> private constructor() {  
    val size: Int  
    operator fun get(index: Int): T  
    operator fun set(index: Int, value: T): Unit  
  
    operator fun iterator(): Iterator<T>  
    // ...  
}
```

Array Examples

```
val a4 = arrayOf(1, 2, 3)
val a5: Array<String?> = arrayOfNulls(3)
val asc = Array(5) { i -> (i * i).toString() }
a4.forEach { println(it) }
a5.forEach { println(it) }
asc.forEach { println(it) }
val x2: IntArray = intArrayOf(1, 2, 3)
x2[0] = x2[1] + x2[2]
println(x2.asList())
```

```
val arr1 = IntArray(5)
val arr2 = IntArray(5) { 42 }
var arr3 = IntArray(20) { it * 2 }
arr1.forEach { println(it) }
arr2.forEach { println(it) }
arr3.filter { it % 3 == 0 }.forEach { print(it) } // prints: 0, 6, 12, 18, 24, 30, 36,
```


Type checks and casts: is and !is operators

```
fun f(obj: Any): Unit {  
    if (obj is String && obj.length > 0) {  
        println("$obj: ${obj.length}")  
    }  
  
    if (obj !is String) { // same as !(obj is String)  
        println("Not a String")  
    } else {  
        println(obj.length)  
    }  
}  
f("abc")
```

Type checks and casts: smart, unsafe and safe casts

```
fun g(x: Any) {  
    when (x) {  
        is Int -> println(x + 1)  
        is String -> println(x.length + 1)  
        is IntArray -> println(x.sum())  
    }  
}  
g(1)  
g("abc")  
g(IntArray(5) { it })
```

```
fun h(y: Any?) {  
    // val x: String = y as String  
    // val x: String? = y as String?  
    val x: String? = y as? String  
    println(x)  
}  
h(12)
```

Unchecked Casts

```
fun readDictionary(file: File): Map<String, *> = file.InputStream().use {  
    TODO("Read a mapping of strings to arbitrary elements.")  
}
```

```
// We saved a map with `Int`s into this file  
val intsFile = File("ints.dictionary")
```

```
// Warning: Unchecked cast: `Map<String, *>` to `Map<String, Int>`  
val intsDictionary: Map<String, Int> = readDictionary(intsFile) as Map<String, Int>
```

- To avoid unchecked casts, you can redesign the program structure. In the example above, you could use the `DictionaryReader<T>` and `DictionaryWriter<T>` interfaces with type-safe implementations for different types. You can introduce reasonable abstractions to move unchecked casts from the call site to the implementation details.
- `@Suppress("UNCHECKED_CAST")`

If expression

```
val a = 5
val b = 10
var max = a
if (a < b) max = b
```

// With else

```
if (a > b) {
    max = a
} else {
    max = b
}
```

// As expression

```
val max2 = if (a > b) a else b
```

```
val max3 = if (a > b) {
    print("Choose a")
    a
} else {
    print("Choose b")
    b
}
```

When expression

```
val x = 3
when (x) {
    1 -> println("x == 1")
    2 -> println("x == 2")
    else -> {
        println("x is neither 1 nor 2")
    }
}
```

```
enum class Bit {
    ZERO, ONE
}

val numericValue = when (getRandomBit()) {
    Bit.ZERO -> 0
    Bit.ONE -> 1
    // the 'else' clause is not required because all cases are covered
}
```

When expression examples

```
when (x) {  
    0, 1 -> print("x == 0 or x == 1")  
    else -> print("otherwise")  
}
```

```
when (x) {  
    s.toInt() -> print("s encodes x")  
    else -> print("s does not encode x")  
}
```

```
when (x) {  
    in 1..10 -> print("x is in the range")  
    in validNumbers -> print("x is valid")  
    !in 10..20 -> print("x is outside the range")  
    else -> print("none of the above")  
}
```

```
fun hasPrefix(x: Any) = when(x) {  
    is String -> x.startsWith("prefix")  
    else -> false  
}
```

```
when {  
    x.isOdd() -> print("x is odd")  
    y.isEven() -> print("y is even")  
    else -> print("x+y is odd")  
}
```

```
fun Request.getBody() =  
    when (val response = executeRequest()) {  
        is Authenticator.Success -> response.body  
        is HttpError -> throw HttpException(response.status)  
    }
```

For loops

```
val collection = listOf(1,2,3,4, 5)
for (item in collection) println(item)
```

```
for (i in 1..3) {
    println(i)
}
```

```
for (i in 6 downTo 0 step 2) {
    println(i)
}
```

```
val array = IntArray(5){ it * it}
```

```
for (i in array.indices) {
    println("$i -> ${array[i]}")
}
```

```
for ((index, value) in array.withIndex()) {
    println("the element at $index is $value")
}
```

While loops

```
var x = 5
while (x > 0) {
    x--
}
```

```
fun retrieveData() = "data ..."
```

```
do {
    val y = retrieveData()
} while (y != null) // y is visible here!
```


Returns and Jumps

Kotlin has three structural jump expressions:

- **return** by default returns from the nearest enclosing function or anonymous function
- **break** terminates the nearest enclosing loop
- **continue** proceeds to the next step of the nearest enclosing loop
- All of these expressions can be used as part of larger expressions:

```
data class Person(val name: String? = null, val email: String? =null, val age: Int? =0)
val person = Person("Ivan Petrov")
val s = person.name ?: return
println("Person name: $s")
```

Labeled break and continue

```
loop@ for (i in 1..100) {  
  for (j in 1..100) {  
    print(j)  
    if (j == 3) break@loop  
  }  
}  
println()
```

Labeled return - I

```
fun foo() {  
    listOf(1, 2, 3, 4, 5).forEach lit@{  
        if (it == 3) return@lit // local return to the caller of the lambda - the forEach loop  
        print(it)  
    }  
    println(" done with explicit label")  
}  
foo()
```

```
fun bar() {  
    listOf(1, 2, 3, 4, 5).forEach {  
        if (it == 3) return@forEach // local return to the caller of the lambda - the forEach loop  
        print(it)  
    }  
    println(" done with implicit label")  
}  
bar()
```

Labeled return - II

```
// simulate break
fun baz() {
    run loop@{
        listOf(1, 2, 3, 4, 5).forEach {
            if (it == 3) return@loop // non-local return from the lambda passed to run
            print(it)
        }
    }
    println(" done with nested loop")
}
baz()
```

Let's practice some Kotlin koans 😊

<https://play.kotlinlang.org/koans/overview>

And solve programming problem (for more ambitions)

<https://codeforces.com/contest/1157/problem/B>

Learn Kotlin by Example & Kotlin idioms

<https://play.kotlinlang.org/byExample/>

<https://kotlinlang.org/docs/idioms.html>

Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>