

# Web development in Kotlin

Suman Venkat  
Sebastian Muskalla

42 / TUM Heilbronn | May 7, 2025



# Agenda

- |   |  |
|---|--|
| <p><b>01 Introduction</b><br/>Introduction of SIT, trainers and participants</p> <p><b>02 About Kotlin</b></p> <p><b>03 Morning session</b><br/>Kotlin basics</p> | <p><b>04 Lunch break + Q &amp; A</b></p> <p><b>05 Afternoon session</b><br/>Web development</p> <p><b>06 Q &amp; A</b></p> |
|---|--|

01

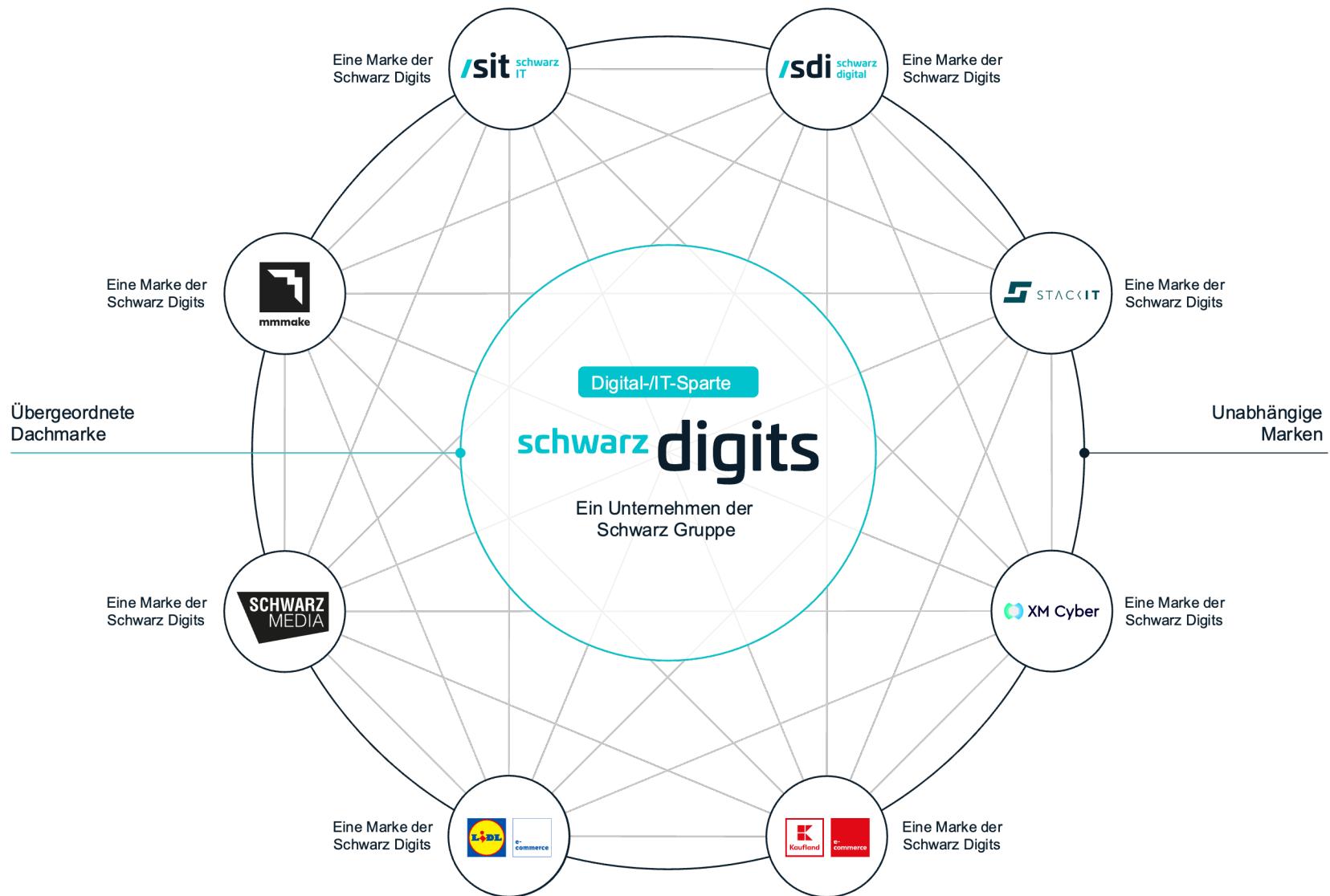
# Introduction

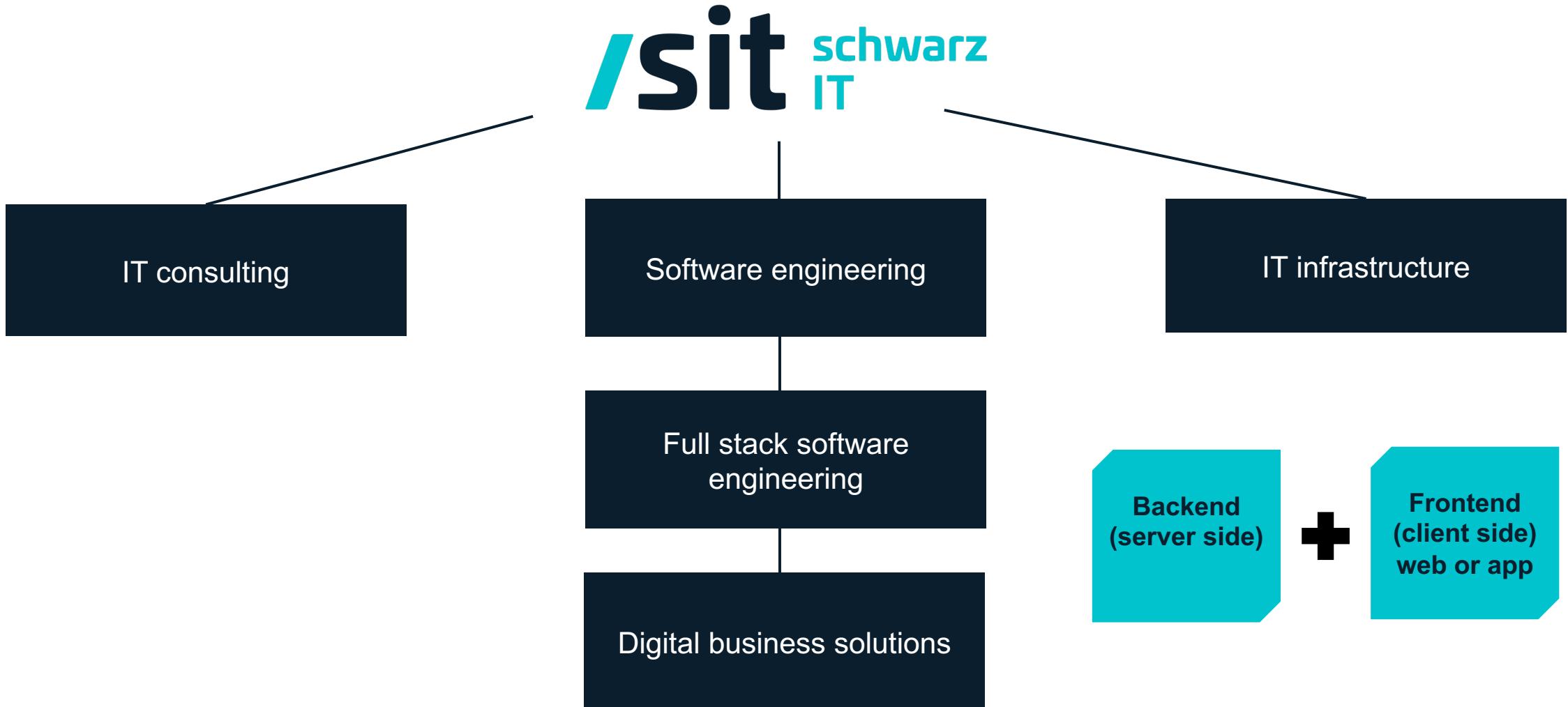
# SCHWARZ

---



# Schwarz Digits







# Sebastian Muskalla

Sebastian.Muskalla@mail.schwarz

2015 M.Sc. (mathematics)

2022 PhD (theoretical computer science)

2023 Full stack software engineer @ Schwarz IT



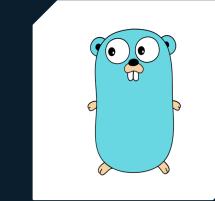


# Suman Venkat

Suman.Venkat@mail.schwarz

2019 Junior Consultant @ Schwarz IT

2021 Full stack software engineer



# Introduction: You!

- **Your name**
- **How long have you been at 42 / TUM Heilbronn?**
- **With which technologies / programming languages do you have experience?**
- **What do you hope to learn in this workshop?**

02

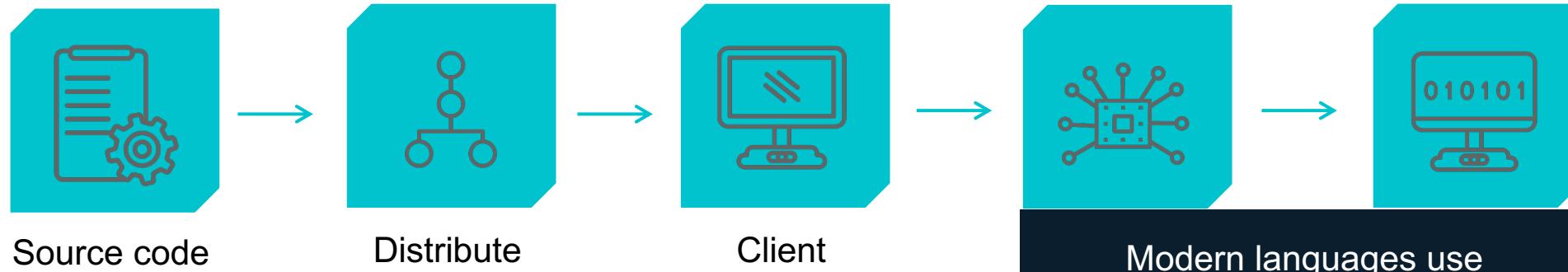
# About Kotlin

# Compiled and interpreted languages

## Compiled languages

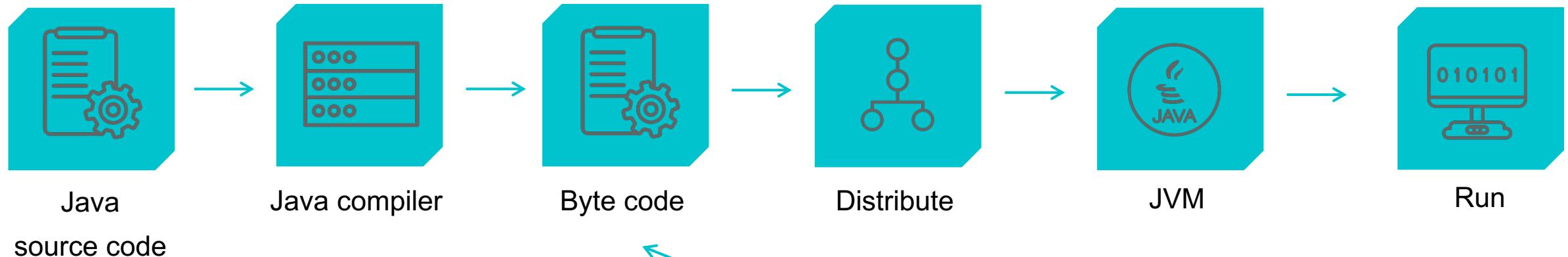


## Interpreted languages



## JIT compilation

# Java and the Java virtual machine (JVM)



- Originally released in 1996
- The most popular programming language of the last 25 years [1]
  - Business sector
  - Android
- Failed attempts to replace it: Groovy, Scala, ...
- Dropping popularity in the last few years:
  - JavaScript/TypeScript, Python
  - **Kotlin**

[1] <https://www.tiobe.com/tiobe-index/>

 HelloWorld.java

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hello, World!");  
4     }  
5 }
```



A screenshot of a Java code editor window titled "HelloWorld.kt". The window has a dark background and a light gray header bar. The code editor area contains the following Java code:

```
1 fun main() {  
2     println("Hello, world!")  
3 }
```

# Kotlin

**2010:** Start of development

**2011:** Public announcement by JetBrains

**2012:** Open source'd (Apache license)

**2016:** First stable release

**2017:** Kotlin Multiplatform beta

**2019:** Declared by Google as preferred language for Android apps

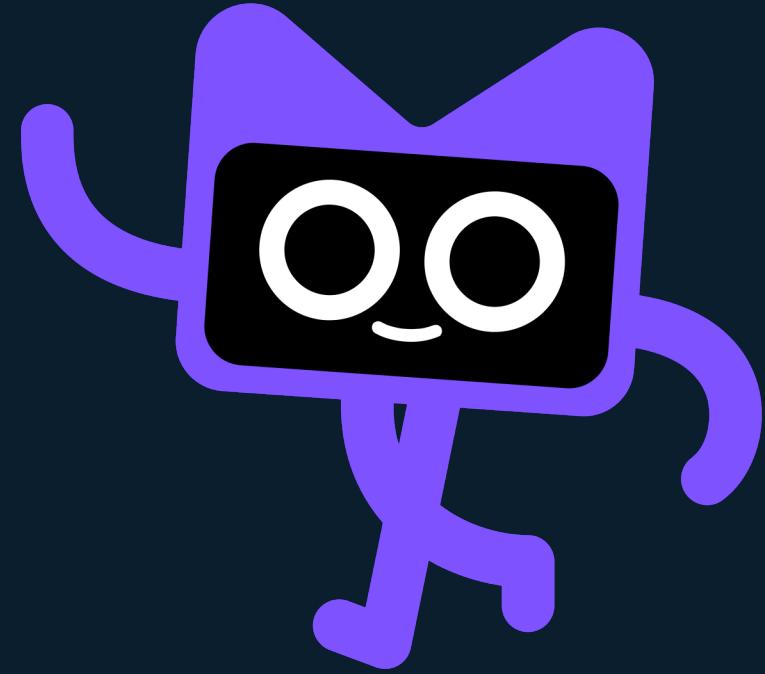
**2023:** Kotlin Multiplatform stable

**2024:** Kotlin 2.0



# Why Kotlin?

- Less verbose
- Full Java interoperability
- Null-safety
- Declarative programming

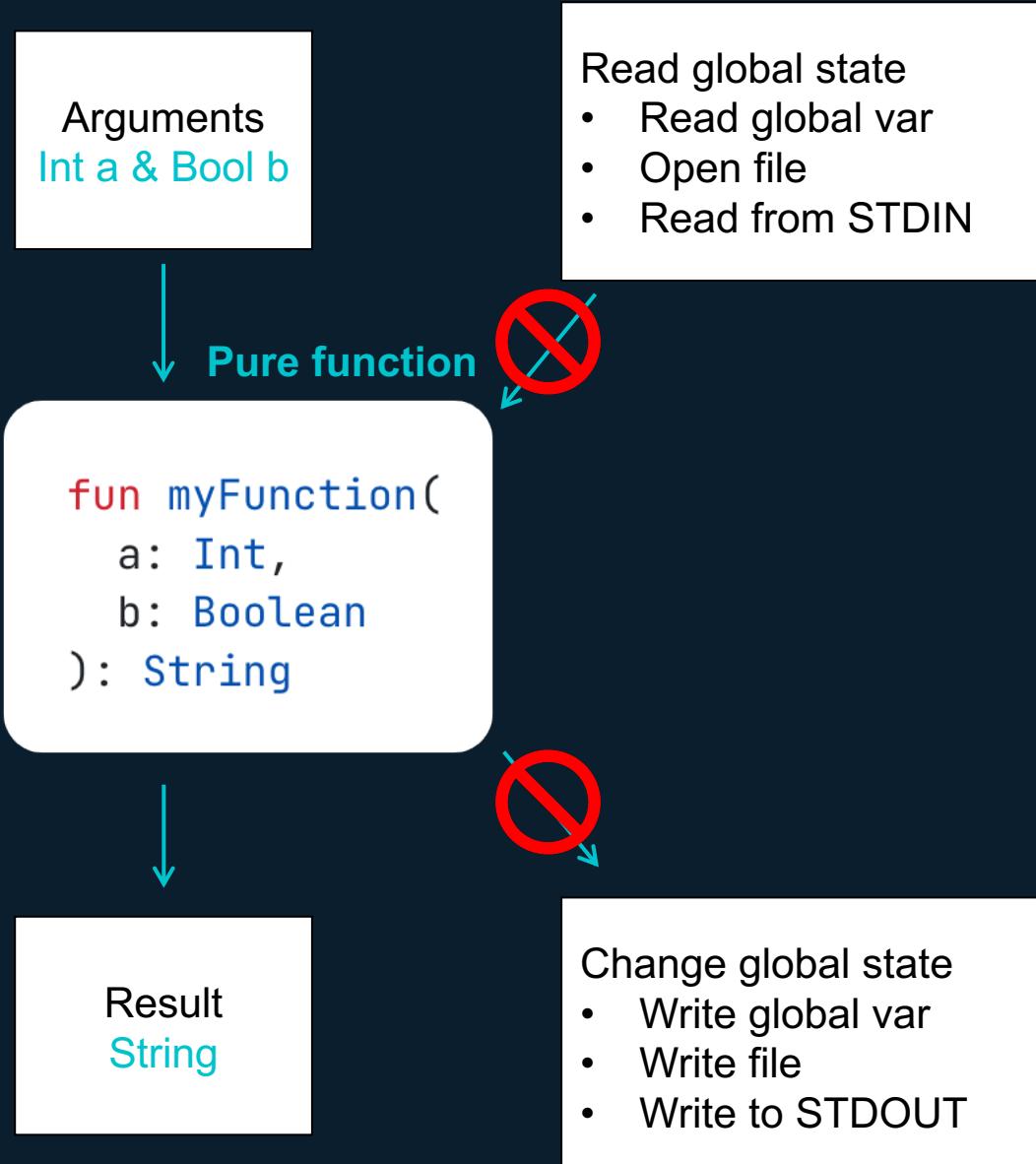


Kodee

# Declarative programming

- Telling the computer **what to do**, not **how to do it**
- Focus on **immutability**
- Functional programming with **pure functions**
- Higher-order functions

Makes code easier to **read, test, debug, maintain**



# Kotlin setup check



<https://github.com/SIT-Kotlin-Workshop>

03

Morning session

# Immutable data

**Immutable data:** State cannot be modified creation

For variables:

- **var:** Mutable variable
- **val:** Immutable value – no reassignment possible

For lists:

- **listOf, List:** Immutable list – no modification possible
- **mutableListOf, MutableList:** Mutable list

Prefer immutable data!

ImmutableVariables.kt

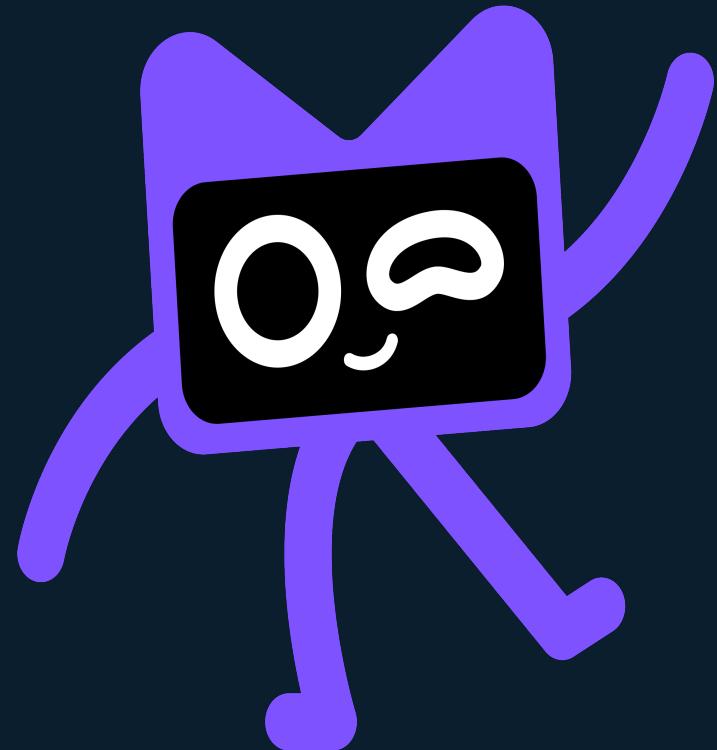
```
1
2 var number = 42
3 number = 1337 // OK
4
5 val number2 = 2 * 668 + 1
6 number2 = 42 // ERROR
7
```

ImmutableLists.kt

```
1 val list = listOf(42)
2 list.add(1337) // ERROR
3
4 val list2 = mutableListOf(1)
5 list2.add(2) // OK
6 list2 = mutableListOf(1, 2, 3) // ERROR
7
```

# Basic syntax overview

- While Loop & Do-While Loop
- For Loop
- If-Then(-Else)
- When
- Functions
  - With body
  - With expression



 BasicSyntax.kt (1)

- □ ×

```
1 println("Countdown:")
2 var number = 10
3
4 while (number > 0) {
5     println("$number ...")
6     number--
7 }
8
9 println("Go!")
```

 BasicSyntax.kt (2)- □ ×

```
1 do {
2     println("Enter \"break\" to terminate loop!")
3     print("Input: ")
4     val input = readln()
5 } while (input != "break")
6
7 println("Exited loop!")
```

 BasicSyntax.kt (3)- □ ×

```
1  for (i in 1..5) {
2      println(i)
3  }
4
5  for (i in 1..<5) {
6      println(i)
7  }
8
9  for (i in 10 downTo 1 step 2) {
10     println(i)
11 }
12
13 val mascots = listOf("Rust: Ferris", "Java: Duke", "Kotlin: Kodee")
14 for (mascot in mascots) {
15     println(mascot)
16 }
```



BasicSyntax.kt (4)

- □ ×

```
1 val number = 5
2 if (number > 2) {
3     println("Number is greater than 2!")
4 }
5
6 if (number > 3) {
7     println("Number is also greater than 3!")
8 } else {
9     println("Number is less than 3!")
10 }
```

 BasicSyntax.kt (5)

- □ ×

```
1 val number = 5
2
3 val output =
4     if (number > 4) {
5         "Number is greater than 4!"
6     } else {
7         "Number is less than 4!"
8     }
9
10 println(output)
```



BasicSyntax.kt (6)

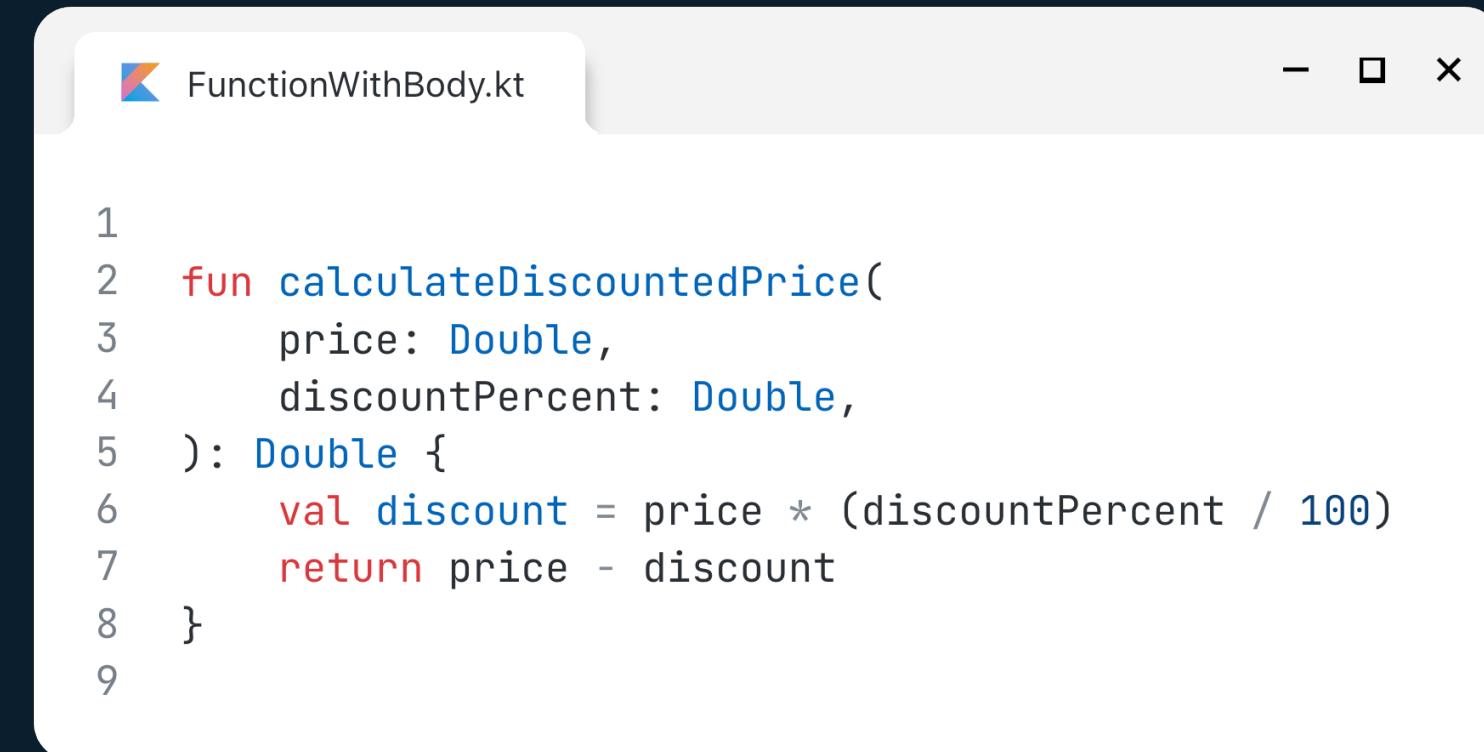
- □ ×

```
1 val number = 5
2
3 when {
4     number == 1 → println("One")
5     number == 2 → println("Two")
6 }
7
8 when (number) {
9     1 → println("One")
10    2 → println("Two")
11 }
```

 BasicSyntax.kt (7)

- □ ×

```
1 val number = 5
2
3 val output =
4     when (number) {
5         1 → "One"
6         2 → "Two"
7         else → "Many"
8     }
9
10 println(output)
```



A screenshot of a Java code editor window titled "FunctionWithBody.kt". The window has a light gray header bar with a close button (X) on the right. The main area contains the following Java code:

```
1
2 fun calculateDiscountedPrice(
3     price: Double,
4     discountPercent: Double,
5 ): Double {
6     val discount = price * (discountPercent / 100)
7     return price - discount
8 }
9
```

 FunctionExpression.kt

```
1
2 fun calculateDiscountedPriceExpression(
3     price: Double,
4     discountPercent: Double,
5 ) = price - (price * (discountPercent / 100))
```



Return type : `Double` is optional and can be inferred

```
1 val number = 5
2
3 val output =
4     when (number) {
5         1 → "One"
6         2 → "Two"
7         else → "Many"
8     }
9
10 val output2 =
11     if (number > 4) {
12         "Number is greater than 4!"
13     } else {
14         "Number is less than 4!"
15     }
16
17 for (i in 1..5) {
18     println(i)
19 }
20
21 val names = listOf("Suman", "Sebastian")
22 for (name in names) {
23     println(name)
24 }
```

When

If-Else

For

```
1
2 fun square(x : Int) : Int {
3     return x * x
4 }
5
6 fun square(x : Int) = x * x
7
8 fun repeatName(
9     name : String,
10    times : Int,
11 ) = name.repeat(times)
12
13 // Calling it
14 println(repeat("Kotlin!", 42))
15
16 println(
17     repeat(
18         name = "Kotlin!",
19         times = 42,
20     )
21 )
22
23
24
```

Function  
with body

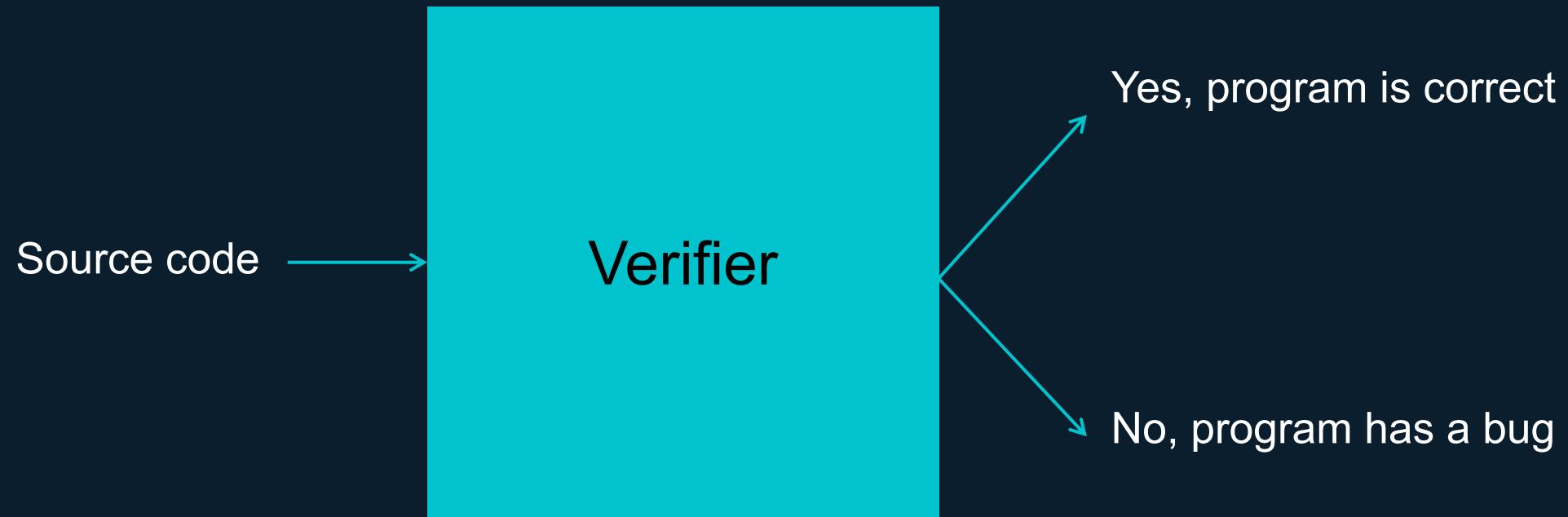
Function  
expression

Function  
call

# Basic syntax

# Exercises

# Automated testing





 SimpleTest.kt (1)

- □ ×

```
1 // Built-in assertions
2 // Only enabled with special flag
3 assert(
4     "kotlin".uppercase(Locale.GERMAN) =
5         "kotlin".uppercase(Locale.ENGLISH)
6 )
7
8 // Using assertk
9 assertThat(
10    "kotlin".uppercase(Locale.GERMAN) =
11        "kotlin".uppercase(Locale.ENGLISH)
12 ).isTrue()
```

 SimpleTest.kt (2)

```
1 assertThat(  
2     "kotlin".uppercase(Locale.of("tr", "TR")) =  
3     "kotlin".uppercase(Locale.ENGLISH)  
4 ).isTrue()
```

```
@Test
fun `simple assert - false`() {
    assertThat(actual: "kotlin".uppercase(Locale.of(language: "tr", country: "TR")) == "kotlin".uppercase(Locale.ENGLISH))
        .isTrue()
}
```

SimpleTest.simple assert - false ✘



✖ Tests failed: 1 of 1 test – 16 ms

expected to be true

org.opentest4j.AssertionFailedError Create breakpoint : expected to be true

```
>      at schwarz.it.kotlin.workshop.basic.testing.SimpleTest.simple assert - false(SimpleTest.kt:28) <1 internal line>
      at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
      at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
```



SimpleTest.kt (3)

```
1 assertThat(  
2     "kotlin".uppercase(Locale.of("tr", "TR"))  
3 ).isEqualTo(  
4     "kotlin".uppercase(Locale.ENGLISH)  
5 )
```

```
@Test
fun `better assert - false`() {
    assertThat("kotlin".uppercase(Locale.of(language: "tr", country: "TR")))
        .isEqualTo("kotlin".uppercase(Locale.ENGLISH))
}
```

SimpleTest.better assert - false ✘



✖ Tests failed: 1 of 1 test – 19 ms

Expected :KOTLIN

Actual :KOTLİN

[<Click to see difference>](#)

org.opentest4j.AssertionFailedError Create breakpoint : expected:<"KOTL[I]N"> but was:<"KOTL[İ]N">

```
>   at schwarz.it.kotlin.workshop.basic.testing.SimpleTest.better assert - false(SimpleTest.kt:34) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
```

# Testing

- Use **testing framework + assertion library** to write expressive tests (Here: JUnit 5 + assertk)
- Detecting **regressions**
- **Code coverage**
- **Test driven development**
- **Automatic test execution** as part of CI/CD

Several types of tests

- **Unit tests:** Test component in isolation
- **Integration tests:** Test interaction between components  
(requires starting web framework, later)

SimpleTest.kt (4)

```
1 import assertk.assertThat
2 import assertk.assertions.contains
3 import org.junit.jupiter.api.Test
4
5 class SimpleTest() {
6
7     @Test
8     fun `another test`() {
9
10         val mascots = listOf(
11             "Duke",
12             "Ferris",
13             "Kodee",
14             "Gopher"
15         )
16
17         assertThat(mascots)
18             .contains("Kodee")
19     }
20
21 }
```

# Testing

# Exercises

# Immutable classes in Kotlin

- **Data classes:**

Kotlin's data **class** feature simplifies the creation of immutable classes by automatically generating useful methods such as:

- equals()
- hashCode()
- toString()
- copy()

Immutable class

```
data class CartItem(val name: String, val quantity: Int, val price: Double)
val originalItem = CartItem("T-Shirt", 1, 5.99)
// compile-time error
originalItem.price = 4.99
// Apply discount by creating a new copy
val discountedItem = originalItem.copy(price = 4.99)
println(originalItem)
println(discountedItem)
```

All properties are read-only

CartItem(name=T-Shirt, quantity=1, price=5.99)

CartItem(name=T-Shirt, quantity=1, price=3.99)

# Nullability

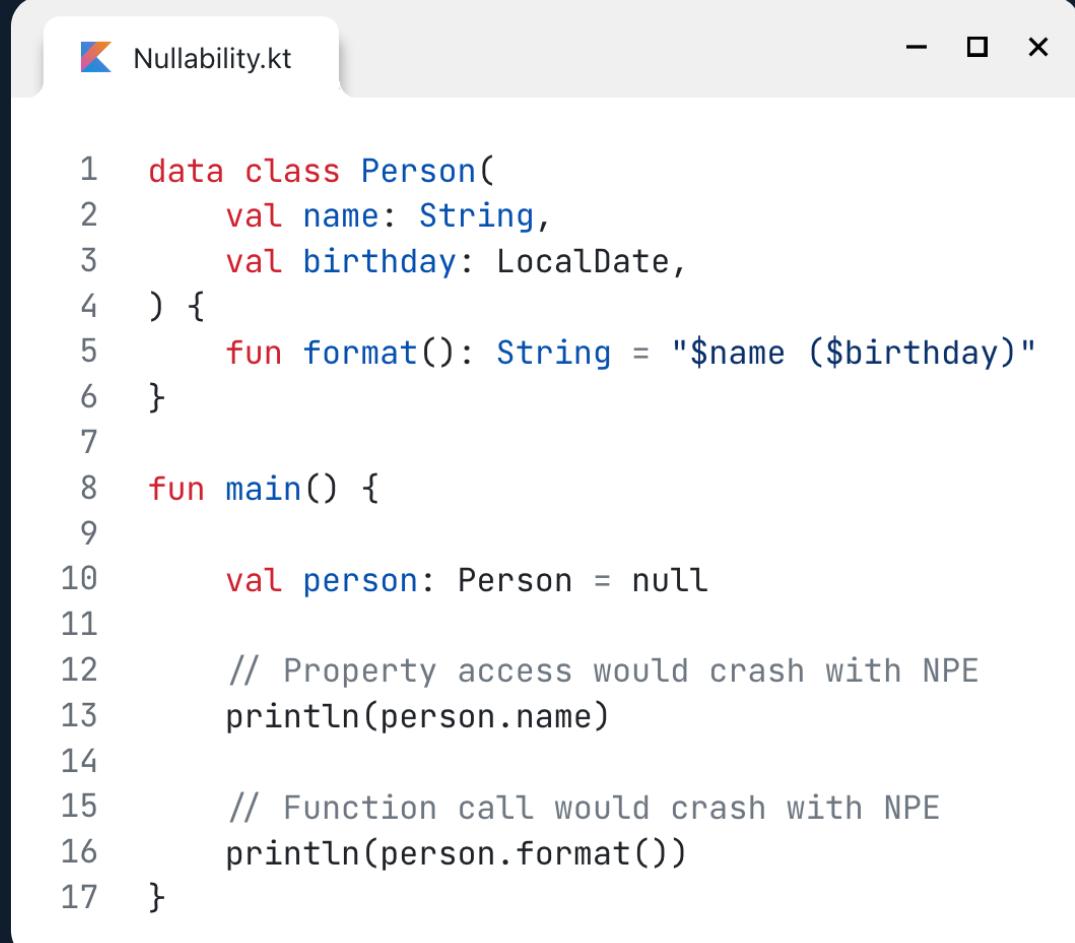
In Java and other languages, the special value `null` is allowed for (almost) all types.

It can mean

- `not initialized`
- `undefined / not provided` (as an argument of a function)
- `error` (as a return value of a function)
- ...

## Problems:

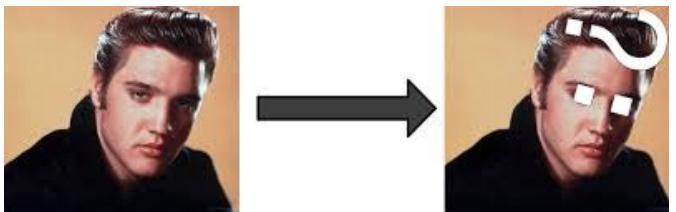
- Don't know which of the above
- Don't know when objects are null
- Accessing null crashes with `NPE` (`NullPointerException`)



```
1 data class Person(
2     val name: String,
3     val birthday: LocalDate,
4 ) {
5     fun format(): String = "$name ($birthday)"
6 }
7
8 fun main() {
9
10    val person: Person = null
11
12    // Property access would crash with NPE
13    println(person.name)
14
15    // Function call would crash with NPE
16    println(person.format())
17 }
```

# Null safety

- In Kotlin, `null` is only allowed as value for nullable types of shape `Type?`
- Compiler enforces that no NPEs happen at runtime
- **Safe navigation ?:**
  - only call property / function if object is not null
- **Elvis operator ?:**
  - Take alternative when object is null
- Smart casting



```
Null safety.kt
```

```
1  data class Company(  
2      val companyName: String,  
3      val numberOfEmployees: Int,  
4  )  
5  
6  data class Worker(  
7      val name: String,  
8      val company: Company,  
9  )  
10  
11 private fun main() {  
12     // val worker : Worker = null  
13     val worker: Worker? = null  
14  
15     // println(worker.name)  
16     println(worker?.name)  
17     println(worker?.company?.companyName)  
18  
19     println(worker?.name ?: "Unknown")  
20  
21     if (worker != null) {  
22         println(worker.name)  
23     }  
24 }
```

# Immutable Data & Nullability

## Exercises

04

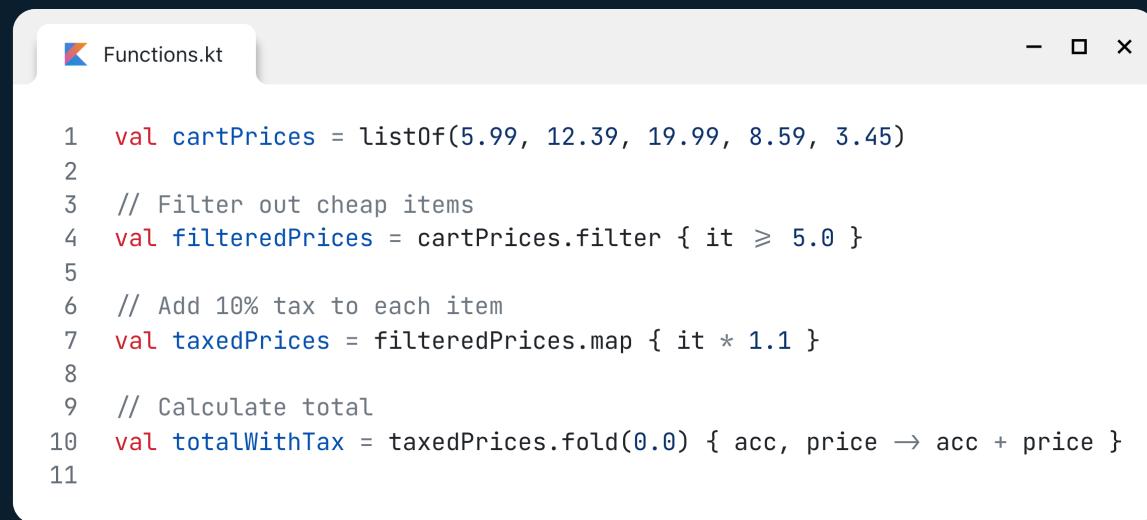
Lunch break  
+ Q & A / Networking

05

# Afternoon session

# Web development

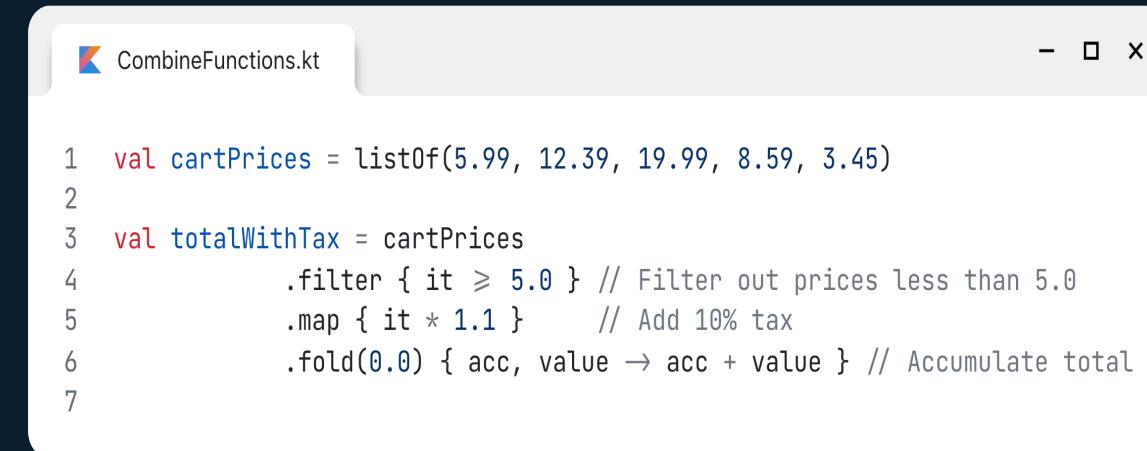
# Functional programming with higher-order functions



A screenshot of a code editor window titled "Functions.kt". The code is written in Kotlin and performs the following steps:

- Define a list of cart prices.
- Filter out items with a price less than or equal to 5.0.
- Add 10% tax to each item.
- Calculate the total sum of the taxed prices.

```
1 val cartPrices = listOf(5.99, 12.39, 19.99, 8.59, 3.45)
2
3 // Filter out cheap items
4 val filteredPrices = cartPrices.filter { it ≥ 5.0 }
5
6 // Add 10% tax to each item
7 val taxedPrices = filteredPrices.map { it * 1.1 }
8
9 // Calculate total
10 val totalWithTax = taxedPrices.fold(0.0) { acc, price → acc + price }
11
```



A screenshot of a code editor window titled "CombineFunctions.kt". The code is written in Kotlin and uses the `filter`, `map`, and `fold` functions to achieve the same result as the first code snippet, but in a more concise and readable way.

- Define a list of cart prices.
- Calculate the total sum of the prices after filtering out those less than or equal to 5.0, mapping each price to its value plus 10% tax, and accumulating the total.

```
1 val cartPrices = listOf(5.99, 12.39, 19.99, 8.59, 3.45)
2
3 val totalWithTax = cartPrices
4     .filter { it ≥ 5.0 } // Filter out prices less than 5.0
5     .map { it * 1.1 }    // Add 10% tax
6     .fold(0.0) { acc, value → acc + value } // Accumulate total
7
```

ImperativeApproach.kt

```
1 fun imperativeEvenDouble(input: List<Int>): List<Int> {  
2     val result = mutableListOf<Int>()  
3     for (number in input) {  
4         if (number % 2 == 0) {  
5             result.add(number * 2)  
6         }  
7     }  
8     return result  
9 }
```

FunctionalApproach.kt

```
1 fun functionalEvenDouble(input: List<Int>): List<Int> {  
2     return input.filter { it % 2 == 0 }  
3         .map { it * 2 }  
4 }
```

# Functional programming

## ▪ Functions as First-Class citizens

- Assigned to variables
- Passed as arguments to other functions
- Returned from other functions

## ▪ Higher-order Functions

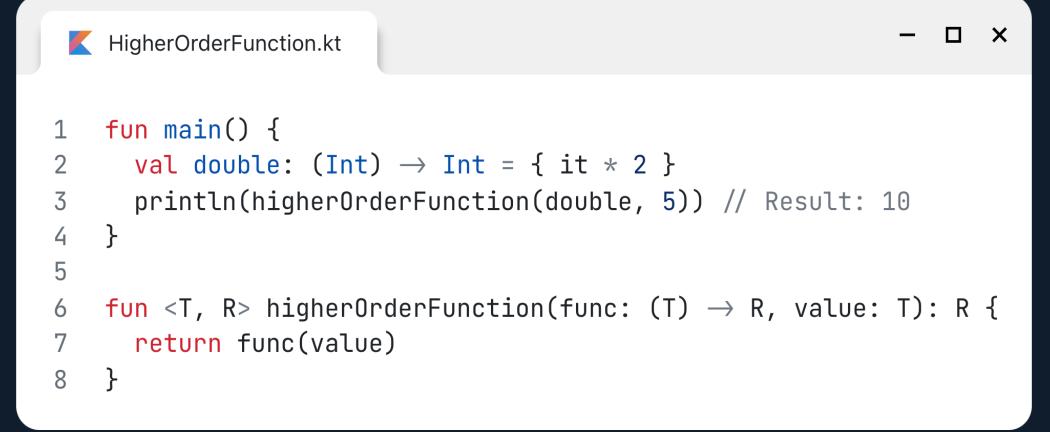
- Functions that take other functions as parameters or return them
- Common examples: `map`, `filter`, `fold`, `reduce`

## ▪ Lambdas

- Anonymous functions that can be used as values
- Often passed to higher-order functions for concise, functional-style code

```
1 val greet: (String) → String = { name → "Hello, $name" }
2 println(greet("World")) // Output: Hello, World
```

```
1 fun <T, R> higherOrderFunction(func: (T) → R, value: T): R {
2     return func(value)
3 }
```



```
HigherOrderFunction.kt - x
- x

1 fun main() {
2     val double: (Int) → Int = { it * 2 }
3     println(higherOrderFunction(double, 5)) // Result: 10
4 }
5
6 fun <T, R> higherOrderFunction(func: (T) → R, value: T): R {
7     return func(value)
8 }
```

# Functional programming with higher-order functions

## Exercises

# Backend web development using Ktor



A screenshot of a code editor window titled "StaticRoutes.kt". The window has a standard OS X style title bar with a close button. The code itself is a single function definition:

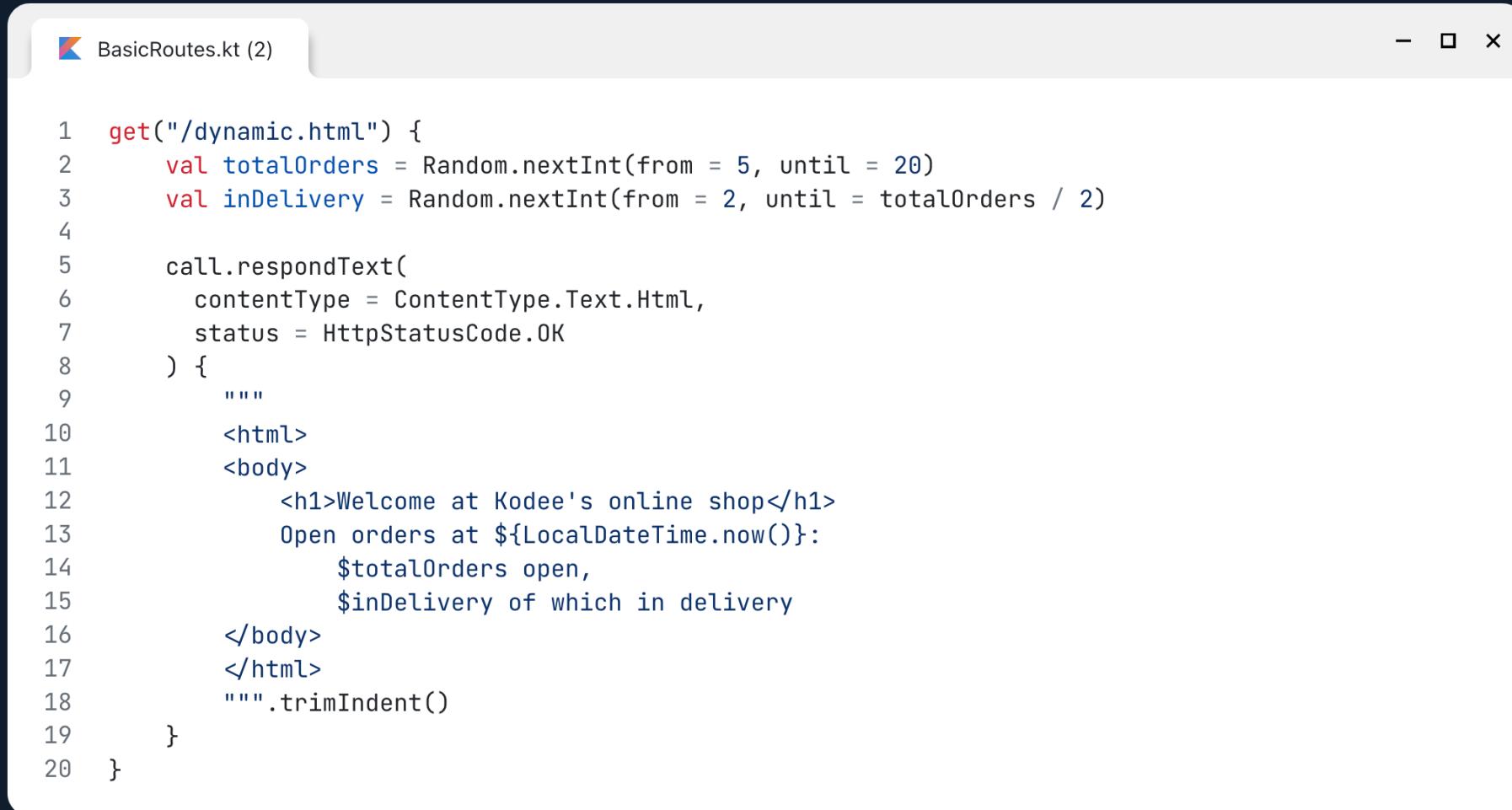
```
1 fun Route.staticRoutes() {  
2     staticResources("/frontend", "static") {  
3         default("index.html")  
4     }  
5 }
```

<http://localhost:8080/frontend/index.html>

 BasicRoutes.kt (1)

```
1 fun Route.basicRoutes() {
2     route("/basic") {
3         get("/static.html") {
4             call.respondText(
5                 contentType = ContentType.Text.Html,
6                 status = HttpStatusCode.OK
7             ) {
8                 """
9                     <html>
10                    <body>
11                        <h1>Welcome at Kodee's online shop</h1>
12                    </body>
13                    </html>
14                 """.trimIndent()
15             }
16         }
17     }
18 }
```

<http://localhost:8080/basic/static.html>



The screenshot shows a code editor window titled "BasicRoutes.kt (2)". The code is written in Kotlin and defines a route for handling requests to "/dynamic.html". The route uses a lambda expression to handle the GET request. Inside the lambda, it generates a random number of total orders (between 5 and 20) and a random number of those orders in delivery (between 2 and half of the total). It then constructs an HTML response with an h1 header, a timestamp, and a message indicating the number of open and in-delivery orders. The code uses triple quotes to define the entire HTML content.

```
1 get("/dynamic.html") {
2     val totalOrders = Random.nextInt(from = 5, until = 20)
3     val inDelivery = Random.nextInt(from = 2, until = totalOrders / 2)
4
5     call.respondText(
6         contentType = ContentType.Text.Html,
7         status = HttpStatusCode.OK
8     ) {
9         """
10            <html>
11                <body>
12                    <h1>Welcome at Kodee's online shop</h1>
13                    Open orders at ${LocalDateTime.now()}:
14                        $totalOrders open,
15                        $inDelivery of which in delivery
16                </body>
17            </html>
18        """.trimIndent()
19    }
20 }
```

<http://localhost:8080/basic/dynamic.html>

# Web technologies

## The beginnings:

- Web server delivers static HTML



## Later:

- Dynamic content: Web server renders page on request



## 15-25 years ago: AJAX etc.

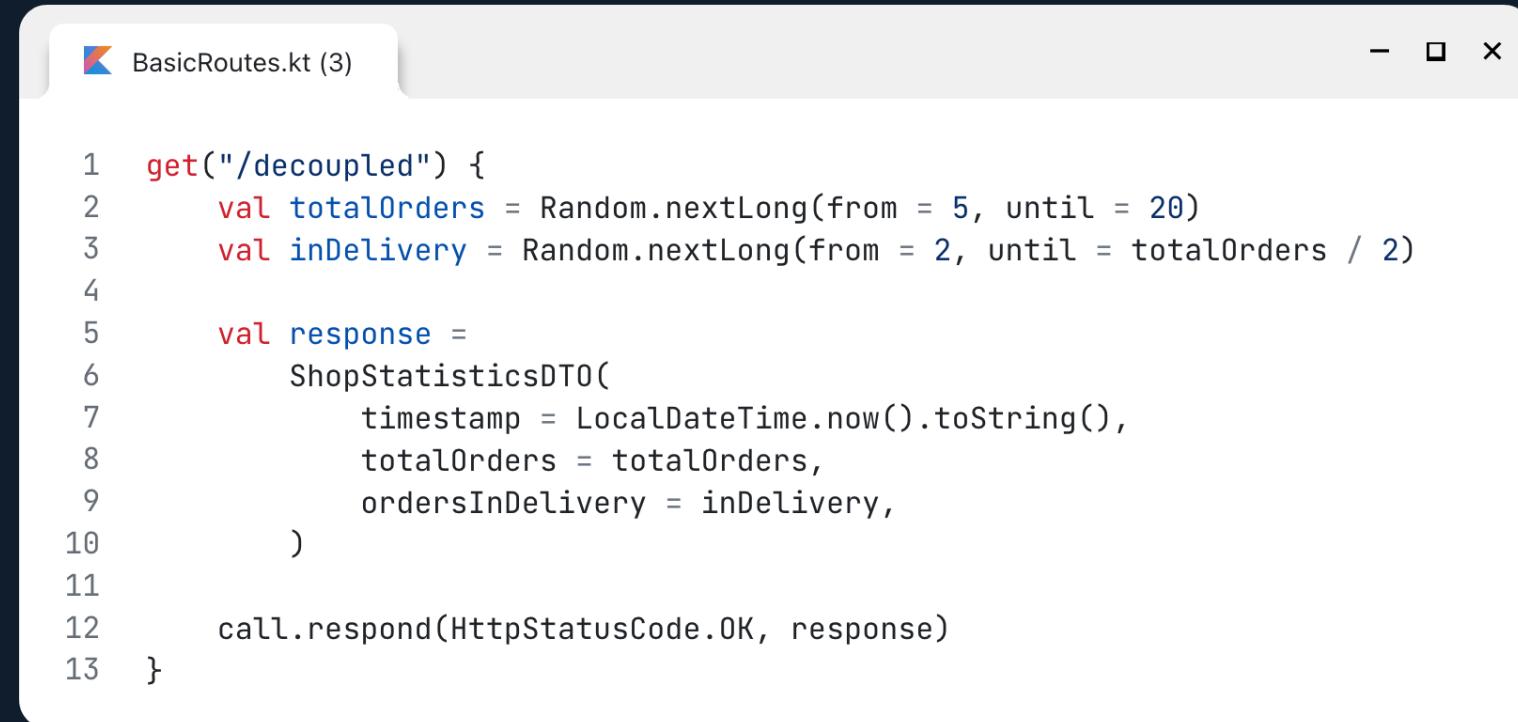
- Browser and backend communicate asynchronously via API
- REST
- JSON



 ShopStatisticsDTO.kt

- □ ×

```
1 @Serializable
2 data class ShopStatisticsDTO(
3     val timestamp: String,
4     val totalOrders: Long,
5     val ordersInDelivery: Long,
6 )
```



```
BasicRoutes.kt (3)

1 get("/decoupled") {
2     val totalOrders = Random.nextLong(from = 5, until = 20)
3     val inDelivery = Random.nextLong(from = 2, until = totalOrders / 2)
4
5     val response =
6         ShopStatisticsDTO(
7             timestamp = LocalDateTime.now().toString(),
8             totalOrders = totalOrders,
9             ordersInDelivery = inDelivery,
10        )
11
12     call.respond(HttpStatusCode.OK, response)
13 }
```

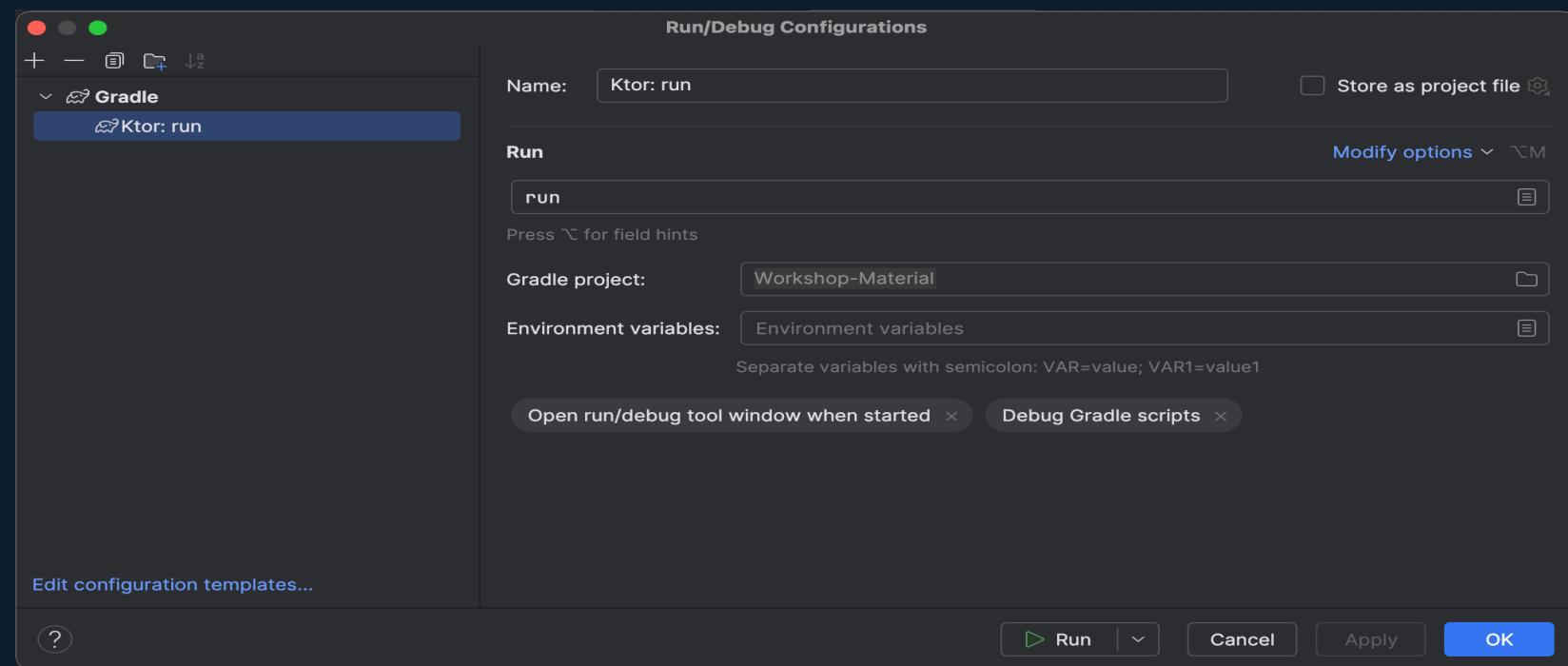
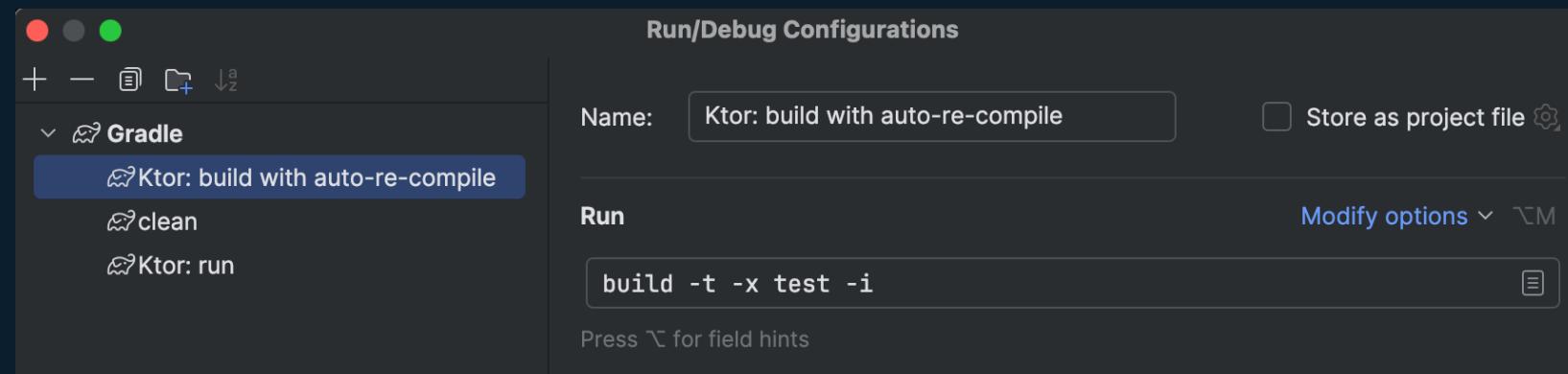
<http://localhost:8080/basic/decoupled>

<http://localhost:8080/frontend/basic.html>

 index.kt

- □ ×

```
1 class BasicRoutesTest {  
2     @Test  
3     fun `endpoint decoupled - returns statistics`() =  
4         integrationTest { client →  
5             val response = client.get("/basic/decoupled")  
6  
7             assertThat(response.status).isEqualTo(HttpStatusCode.OK)  
8  
9             val body = response.body<ShopStatisticsDTO>()  
10  
11            assertThat(body.totalOrders).isLessThan(20)  
12            assertThat(body.ordersInDelivery).isLessThan(body.totalOrders)  
13            assertThat(body.ordersInDelivery).isGreaterThanOrEqualTo(2)  
14        }  
15    }  
16}
```



# **Advanced project demo in IDE / browser**

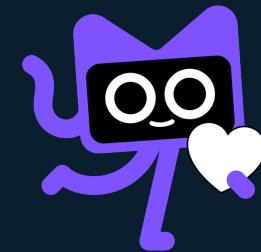
<http://localhost:8080/frontend/advanced.html>

# Backend web development using Ktor

## Exercises

# Further reading

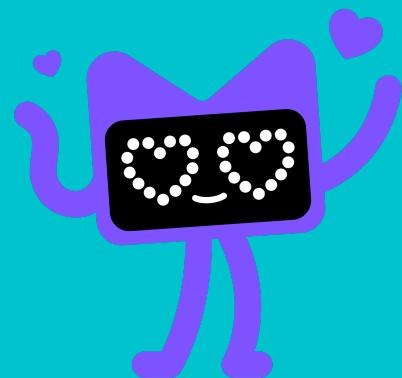
- **Databases and ORM libraries**  
e.g. <https://github.com/JetBrains/Exposed>
- **Dependency injection**
- **Layered architecture**
- **OpenAPI specifications**
- **Frontend technology**  
e.g. [Vue.js](#)



06

Q&A

**Thank you for  
attending!**



**Sebastian Muskalla**

Digital Business Solutions  
Schwarz IT

[Sebastian.Muskalla@mail.schwarz](mailto:Sebastian.Muskalla@mail.schwarz)

**Suman Venkat**

Digital Business Solutions  
Schwarz IT

[Suman.Venkat@mail.schwarz](mailto:Suman.Venkat@mail.schwarz)