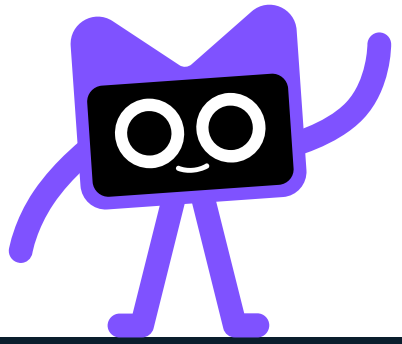
/sit schwarz IT



Web development in Kotlin

Suman Venkat
Sebastian Muskalla

42 / TUM Heilbronn | May 7, 2025

Agenda

01 Introduction
Introduction of SIT, trainers and participants

02 About Kotlin

03 Morning session
Kotlin basics

04 Lunch break + Q & A

05 Afternoon session
Web development

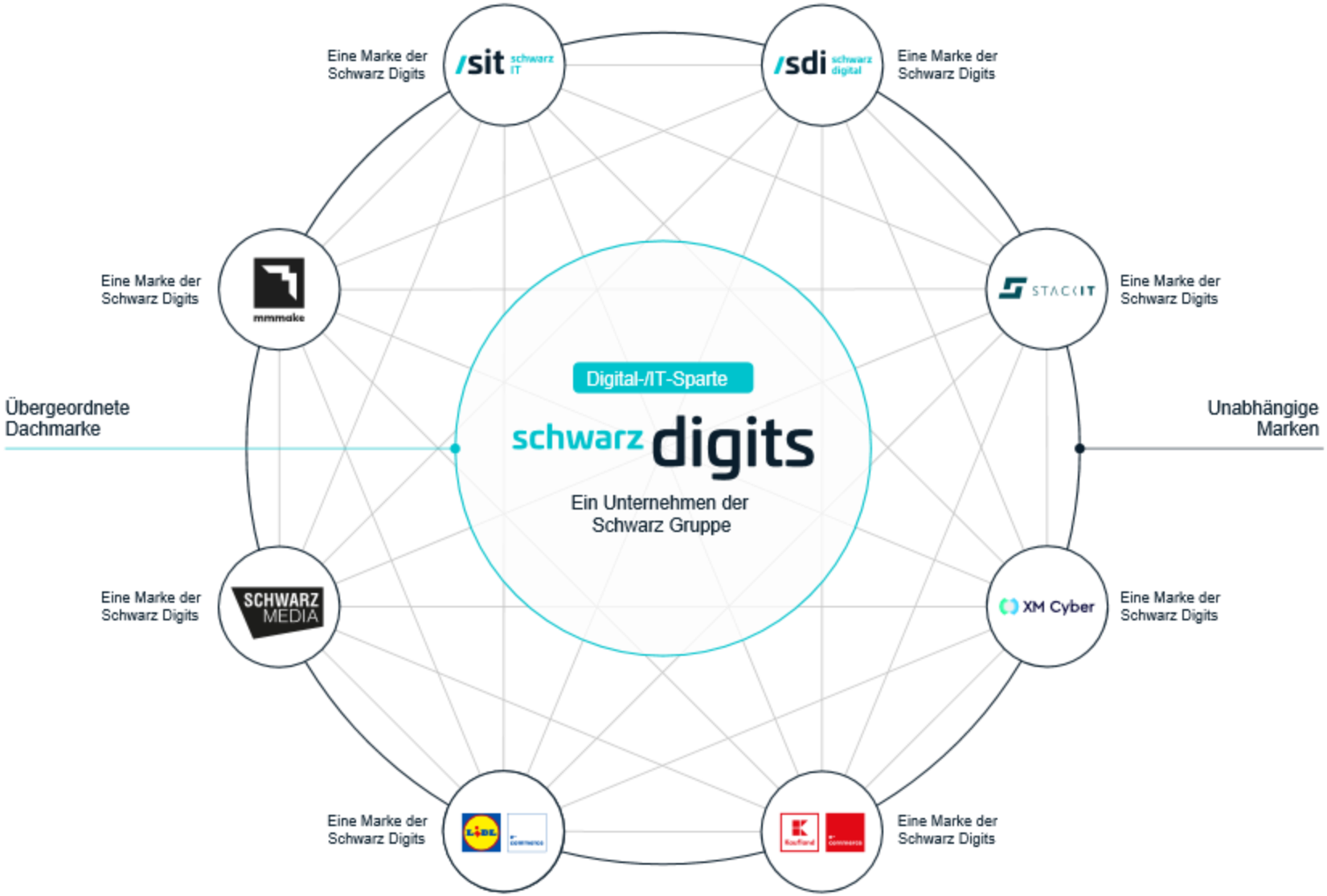
06 Q & A

01

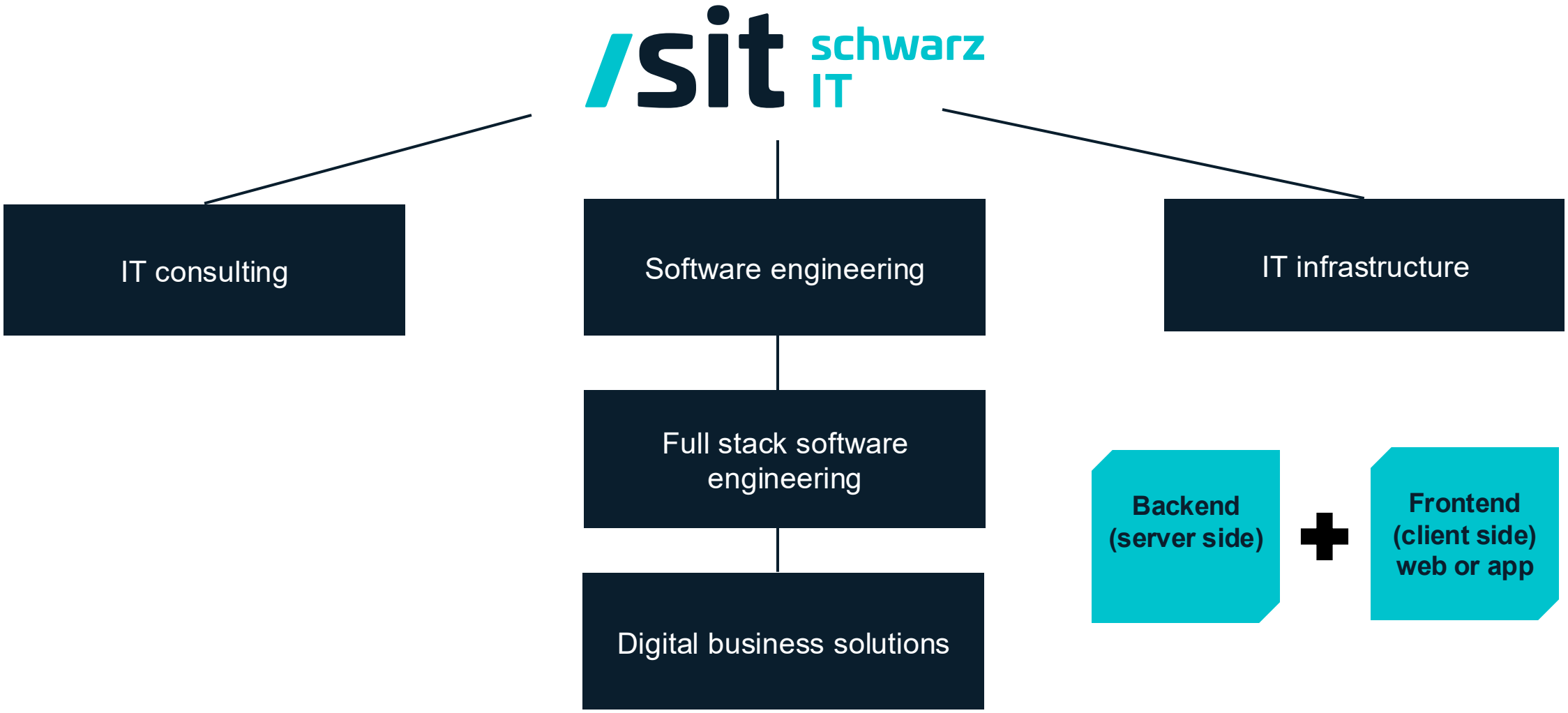
Introduction


SCHWARZ
LIDL
Kaufland
SCHWARZ
Produktion
pre
zero
schwarz
digits

Schwarz Digits



Schwarz IT


IT consulting
Software engineering
IT infrastructure
Full stack software engineering
Digital business solutions
Backend (server side)
Frontend (client side) web or app

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?

/sit schwarz IT

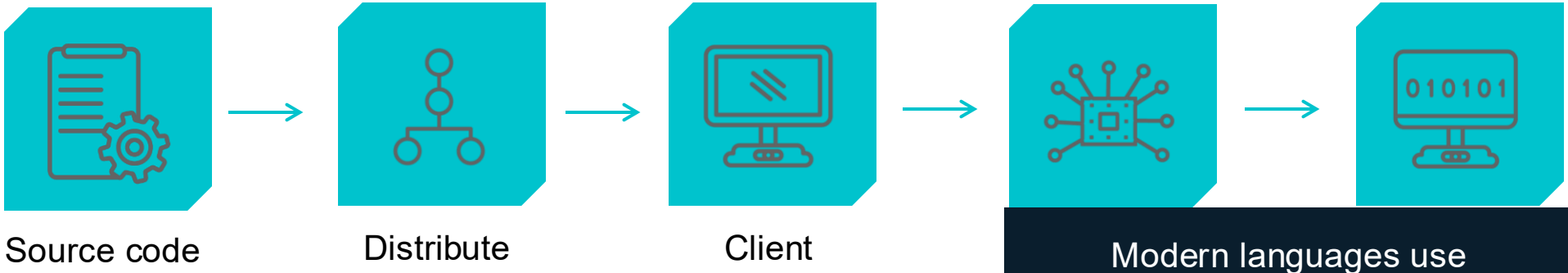
02

About Kotlin

Compiled and interpreted languages

Compiled languages

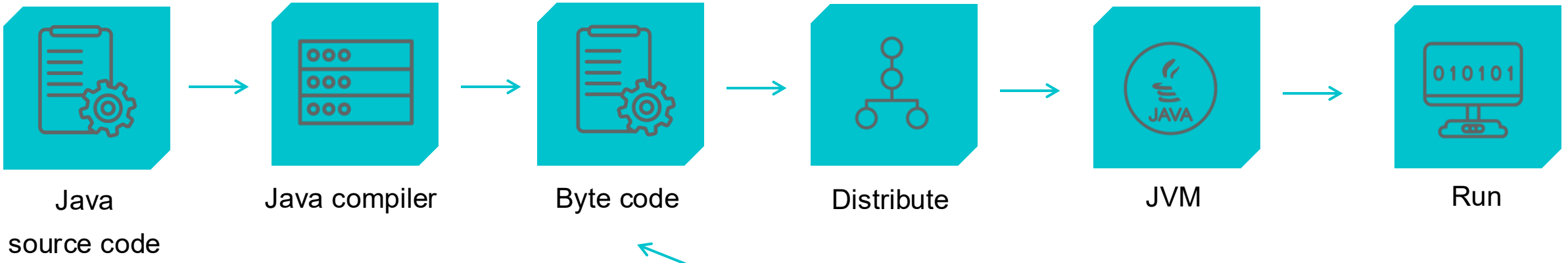
Interpreted languages



Modern languages use

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

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

/sit schwarz IT

Kotlin

2010: Start of development

2011: Public announcement by JetBrains

2012: Open source‘d (Apache license)

2016: First stable release

2017: Kotlin Multiplatform beta

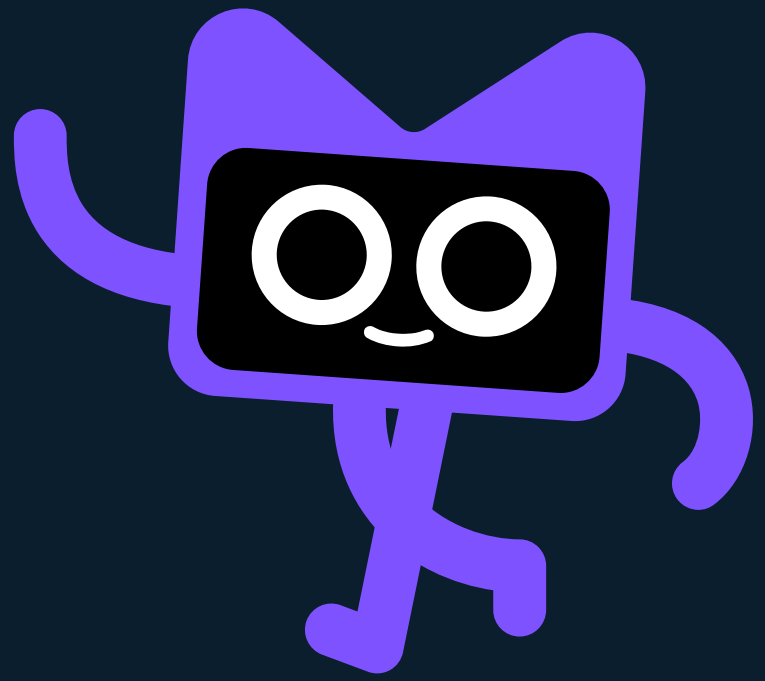
2019: Declared by Google as preferred language for Android apps

2023: Kotlin Multiplatform stable

2024: Kotlin 2.0

Why Kotlin?

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

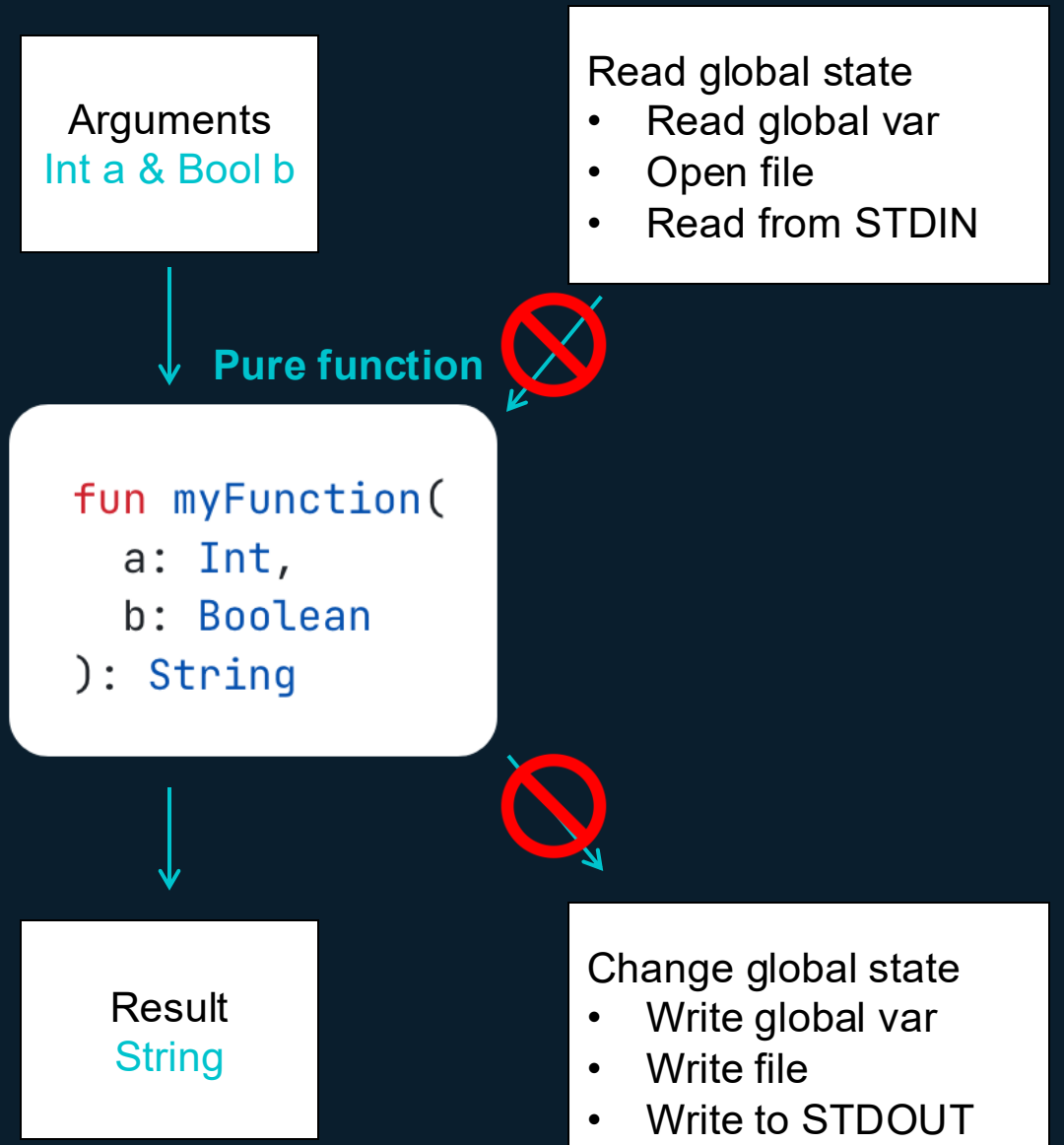


Kodee

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

```kotlin

var number = 42
number = 1337 // OK

val number2 = 2 * 668 + 1
number2 = 42 // ERROR

```

ImmutableLists.kt

```kotlin
val list = listOf(42)
list.add(1337) // ERROR

val list2 = mutableListOf(1)
list2.add(2) // OK
list2 = mutableListOf(1, 2, 3) // ERROR

```

Basic syntax overview

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

BasicSyntax.kt (1)

```kotlin
println("Countdown:")
var number = 10

while (number > 0) {
    println("$number...")
    number--
}

println("Go!")
```

BasicSyntax.kt (2)

```kotlin
do {
    println("Enter \"break\" to terminate loop!")
    print("Input: ")
    val input = readln()
} while (input ≠ "break")

println("Exited loop!")
```

BasicSyntax.kt (3)

```kotlin
for (i in 1..5) {
    println(i)
}

for (i in 1..<5) {
    println(i)
}

for (i in 10 downTo 1 step 2) {
    println(i)
}

val mascots = listOf("Rust: Ferris", "Java: Duke", "Kotlin: Kodee")
for (mascot in mascots) {
    println(mascot)
}
```

BasicSyntax.kt (4)

```kotlin
val number = 5
if (number > 2) {
    println("Number is greater than 2!")
}

if (number > 3) {
    println("Number is also greater than 3!")
} else {
    println("Number is less than 3!")
}
```

BasicSyntax.kt (5)

```kotlin
val number = 5

val output =
    if (number > 4) {
        "Number is greater than 4!"
    } else {
        "Number is less than 4!"
    }

println(output)
```

BasicSyntax.kt (6)

```kotlin
val number = 5

when {
    number == 1 -> println("One")
    number == 2 -> println("Two")
}

when (number) {
    1 -> println("One")
    2 -> println("Two")
}
```

BasicSyntax.kt (7)

```kotlin
val number = 5

val output =
    when (number) {
        1 -> "One"
        2 -> "Two"
        else -> "Many"
    }

println(output)
```

FunctionWithBody.kt

```kotlin

fun calculateDiscountedPrice(
    price: Double,
    discountPercent: Double,
): Double {
    val discount = price * (discountPercent / 100)
    return price - discount
}

```

FunctionExpression.kt

```kotlin

fun calculateDiscountedPriceExpression(
    price: Double,
    discountPercent: Double,
) = price - (price * (discountPercent / 100))

```

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

BasicSyntaxOverview.kt

```kotlin
val number = 5

val output =
    when (number) {
        1 -> "One"
        2 -> "Two"
        else -> "Many"
    }

val output2 =
    if (number > 4) {
        "Number is greater than 4!"
    } else {
        "Number is less than 4!"
    }

for (i in 1..5) {
    println(i)
}

val names = listOf("Suman", "Sebastian")
for (name in names) {
    println(name)
}
```

When

If-Else

For

FunctionsOverview.kt

```kotlin

fun square(x : Int) : Int {
    return x * x
}

fun square(x : Int) = x * x

fun repeatName(
    name : String,
    times : Int,
) = name.repeat(times)

// Calling it
println(repeat("Kotlin!", 42))

println(
    repeat(
        name = "Kotlin!",
        times = 42,
    )
)
```

Function with body

Function expression

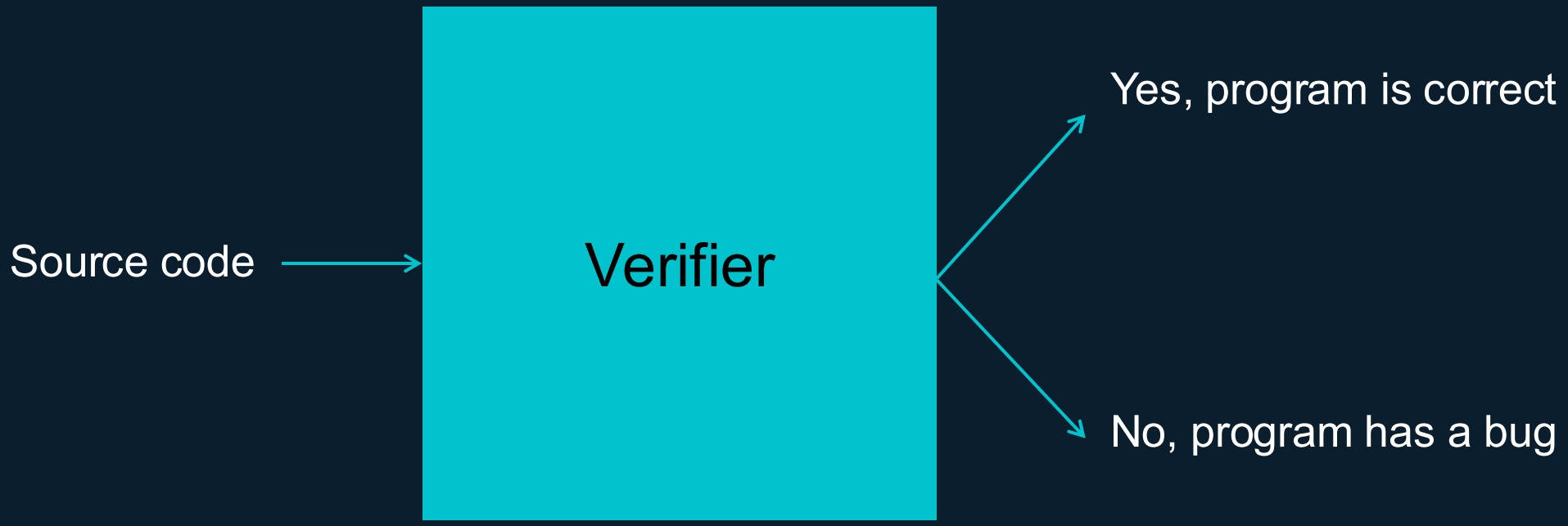
Function call

/sit schwarz IT

Basic syntax

Exercises

Automated testing


Source code
Verifier
Yes, program is correct
No, program has a bug


Source code
Test scenario
Expected outcome
Test framework
Yes, outcome is as expected
No, program has a bug

SimpleTest.kt (1)

```kotlin
// Built-in assertions
// Only enabled with special flag
assert(
    "kotlin".uppercase(Locale.GERMAN) ==
        "kotlin".uppercase(Locale.ENGLISH)
)

// Using assertk
assertThat(
    "kotlin".uppercase(Locale.GERMAN) ==
        "kotlin".uppercase(Locale.ENGLISH)
).isTrue()
```

SimpleTest.kt (2)

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

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

npleTest.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  )
```

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

npleTest.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)

```kotlin
import assertk.assertThat
import assertk.assertions.contains
import org.junit.jupiter.api.Test

class SimpleTest() {

@Test
fun `another test`() {

    val mascots = listOf(
        "Duke",
        "Ferris",
        "Kodee",
        "Gopher"
    )

    assertThat(mascots)
        .contains("Kodee")
}

}
```

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

ImmutableClass.kt

```kotlin
1  data class CartItem(val name: String, val quantity: Int, val price: Double)
2
3  val originalItem = CartItem("T-Shirt", 1, 5.99)
4
5  // compile-time error
6  originalItem.price = 4.99
7
8  // Apply discount by creating a new copy
9  val discountedItem = originalItem.copy(price = 4.99)
10
11 println(originalItem)
12 println(discountedItem)
13
```

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:

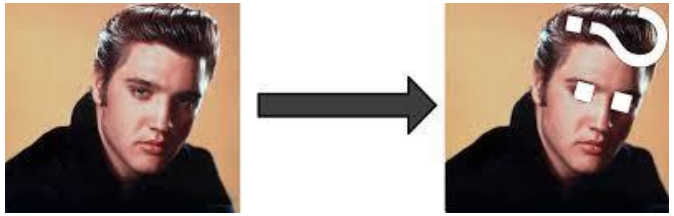
- Don‘t know which of the above
- Don‘t know when objects are null
- Accessing null crashes with NPE (NullPointerException)

Nullability.kt

```kotlin
data class Person(
    val name: String,
    val birthday: LocalDate,
) {
    fun format(): String = "$name ($birthday)"
}

fun main() {

    val person: Person = null

    // Property access would crash with NPE
    println(person.name)

    // Function call would crash with NPE
    println(person.format())
}
```

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

```kotlin

data class Company(
    val companyName: String,
    val numberOfEmployees: Int,
)

data class Worker(
    val name: String,
    val company: Company,
)

private fun main() {
    // val worker : Worker = null

    val worker: Worker? = null

    // println(worker.name)

    println(worker?.name)
    println(worker?.company?.companyName)

    println(worker?.name ?: "Unknown")

    if (worker ≠ null) {
        println(worker.name)
    }
}
```

Immutable Data & Nullability

Exercises

04

Lunch break
+ Q & A / Networking

/sit schwarz IT

05

Afternoon session
Web development

Functional programming with higher-order functions

Functions.kt

```kotlin
val cartPrices = listOf(5.99, 12.39, 19.99, 8.59, 3.45)

// Filter out cheap items
val filteredPrices = cartPrices.filter { it >= 5.0 }

// Add 10% tax to each item
val taxedPrices = filteredPrices.map { it * 1.1 }

// Calculate total
val totalWithTax = taxedPrices.fold(0.0) { acc, price -> acc + price }

```

CombineFunctions.kt

```kotlin
val cartPrices = listOf(5.99, 12.39, 19.99, 8.59, 3.45)

val totalWithTax = cartPrices
            .filter { it >= 5.0 } // Filter out prices less than 5.0
            .map { it * 1.1 }     // Add 10% tax
            .fold(0.0) { acc, value -> acc + value } // Accumulate total

```

ImperativeApproach.kt

```kotlin
fun imperativeEvenDouble(input: List<Int>): List<Int> {
  val result = mutableListOf<Int>()
  for (number in input) {
    if (number % 2 == 0) {
      result.add(number * 2)
    }
  }
  return result
}
```

FunctionalApproach.kt

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

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 funtions for concise, functional-style code

```kotlin
val greet: (String) -> String = { name -> "Hello, $name" }
println(greet("World")) // Output: Hello, World
```

```kotlin
fun <T, R> higherOrderFunction(func: (T) -> R, value: T): R {
  return func(value)
}
```

HigherOrderFunction.kt

```kotlin
fun main() {
  val double: (Int) -> Int = { it * 2 }
  println(higherOrderFunction(double, 5)) // Result: 10
}

fun <T, R> higherOrderFunction(func: (T) -> R, value: T): R {
  return func(value)
}
```

Functional programming with higher-order functions

Exercises

Backend web development using Ktor

StaticRoutes.kt

```kotlin
fun Route.staticRoutes() {
    staticResources("/frontend", "static") {
        default("index.html")
    }
}
```

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

BasicRoutes.kt (1)

```kotlin
fun Route.basicRoutes() {
    route("/basic") {
        get("/static.html") {
            call.respondText(
              contentType = ContentType.Text.Html,
              status = HttpStatusCode.OK
            ) {
                """
                <html>
                <body>
                    <h1>Welcome at Kodee's online shop</h1>
                </body>
                </htmL>
                """.trimIndent()
            }
        }
    }
}
```

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

BasicRoutes.kt (2)

```kotlin
get("/dynamic.html") {
    val totalOrders = Random.nextInt(from = 5, until = 20)
    val inDelivery = Random.nextInt(from = 2, until = totalOrders / 2)

    call.respondText(
      contentType = ContentType.Text.Html,
      status = HttpStatusCode.OK
    ) {
        """
        <html>
        <body>
            <h1>Welcome at Kodee's online shop</h1>
            Open orders at ${LocalDateTime.now()}:
                $totalOrders open,
                $inDelivery of which in delivery
        </body>
        </html>
        """.trimIndent()
    }
}
```

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

```kotlin
@Serializable
data class ShopStatisticsDTO(
    val timestamp: String,
    val totalOrders: Long,
    val ordersInDelivery: Long,
)
```

BasicRoutes.kt (3)

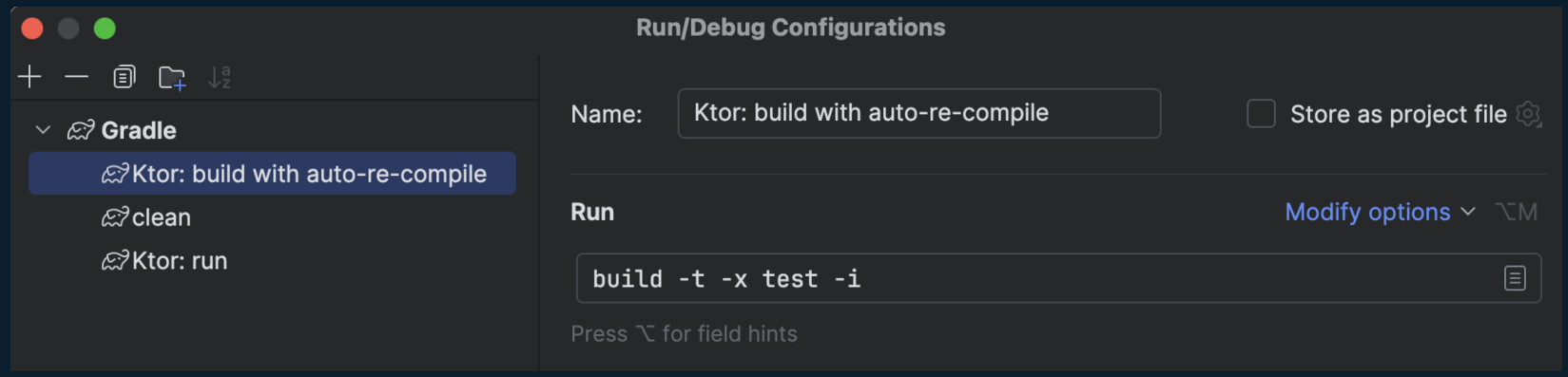
```kotlin
get("/decoupled") {
    val totalOrders = Random.nextLong(from = 5, until = 20)
    val inDelivery = Random.nextLong(from = 2, until = totalOrders / 2)

    val response =
        ShopStatisticsDTO(
            timestamp = LocalDateTime.now().toString(),
            totalOrders = totalOrders,
            ordersInDelivery = inDelivery,
        )

    call.respond(HttpStatusCode.OK, response)
}
```

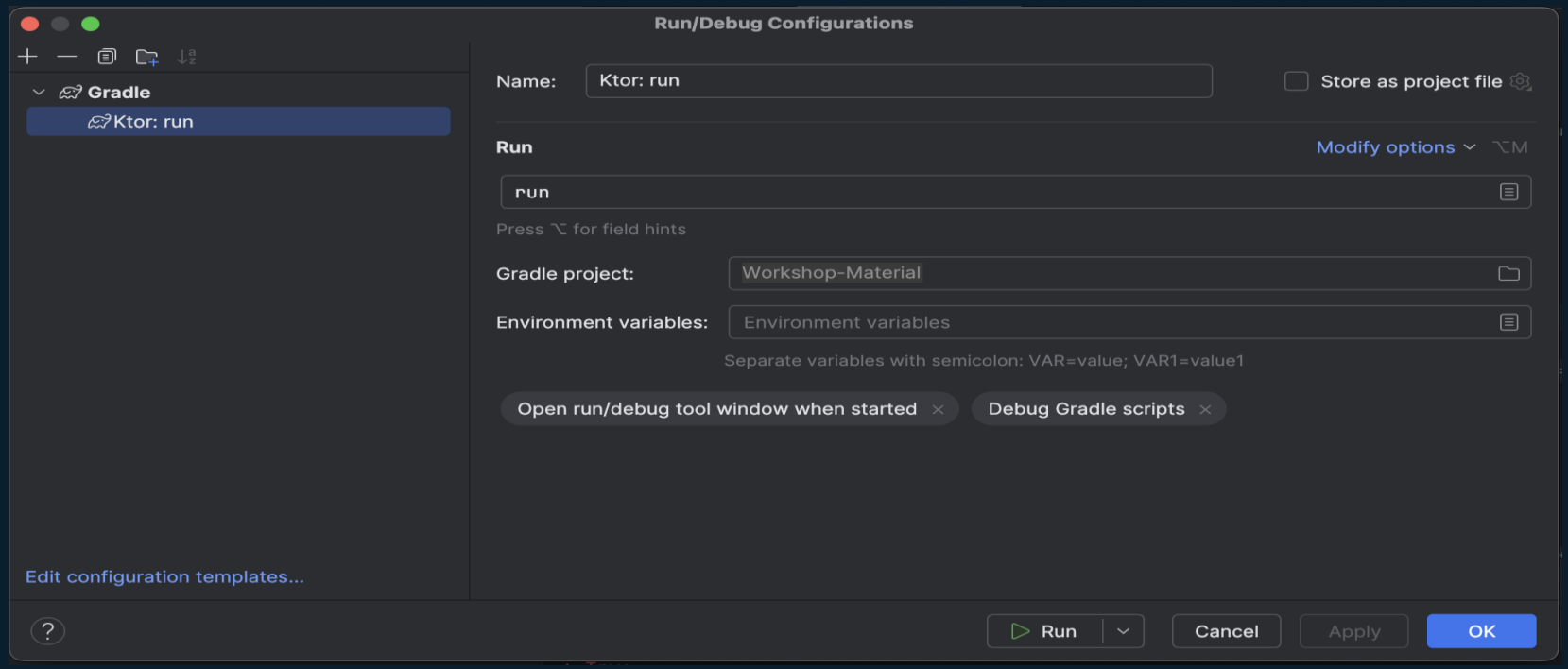
http://localhost:8080/basic/decoupled

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

index.kt

```kotlin
class BasicRoutesTest {
    @Test
    fun `endpoint decoupled - returns statistics`() =
        integrationTest { client →
            val response = client.get("/basic/decoupled")

            assertThat(response.status).isEqualTo(HttpStatusCode.OK)

            val body = response.body<ShopStatisticsDTO>()

            assertThat(body.totalOrders).isLessThan(20)
            assertThat(body.ordersInDelivery).isLessThan(body.totalOrders)
            assertThat(body.ordersInDelivery).isGreaterThanOrEqualTo(2)
        }
}

```


Run/Debug Configurations
Gradle
Ktor: build with auto-re-compile
clean
Ktor: run
Name:
Ktor: build with auto-re-compile
Store as project file
Run
Modify options
⌥M
build -t -x test -i
Press ⌥ for field hints


Run/Debug Configurations
Gradle
Ktor: run
Name:
Ktor: run
Store as project file
Run
Modify options
⌥M
run
Press ⌥ for field hints
Gradle project:
Workshop-Material
Environment variables:
Environment variables
Separate variables with semicolon: VAR=value; VAR1=value1
Open run/debug tool window when started
Debug Gradle scripts
Edit configuration templates...
Run
Cancel
Apply
OK

Advanced project demo in IDE / browser

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

Backend web development using Ktor

Exercises

http://localhost:8080/frontend/exercise.html

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

Further reading

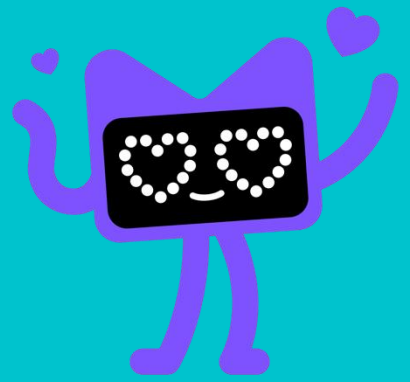
- Databases and ORM libraries
  e.g. https://github.com/JetBrains/Exposed
- Dependency injection
- Layered architecture
- OpenAPI specifications
- Frontend technology
  e.g. Vue.js

/sit schwarz IT

06

Q&A

Thank you for attending!



Sebastian Muskalla
Digital Business Solutions
Schwarz IT

Sebastian.Muskalla@mail.schwarz

Suman Venkat
Digital Business Solutions
Schwarz IT

Suman.Venkat@mail.schwarz