Lesson 2:

Rolling a dice

Instructor: Ahmet Geymen

About this lesson

- Lesson 2:
  - Kotlin fundamentals
  - Add a button to an app

Get started

Kotlin fundamentals

Kotlin Playground

https://developer.android.com/training/kotlinplayground

Conditionals

Control flow

- Kotlin features several ways to implement conditional logic:
  - If/Else statements
  - When statements
  - For loops
  - While loops

if/else statements

```
if ( condition ) {
    body 1
} else {
    body 2
}
```

if/else statements

```
if ( condition 1 ) {
    body 1
} else if ( condition 2 ) {
    body 2
} else {
    body 3
}
```

if/else statements

```
if ( condition 1 ) {
    body 1
} else if ( condition 2 ) {
    body 2
}
```

if/else statements

```
fun main() {
    val trafficLightColor = "Green"

    if (trafficLightColor == "Red") {
        println("Stop")
    } else if (trafficLightColor == "Yellow") {
        println("Slow")
    } else {
        println("Go")
    }
}
```

Ranges

- Data type containing a span of comparable values (e.g., integers from 1 to 100 inclusive)
- Ranges are bounded
- Objects within a range can be mutable or immutable

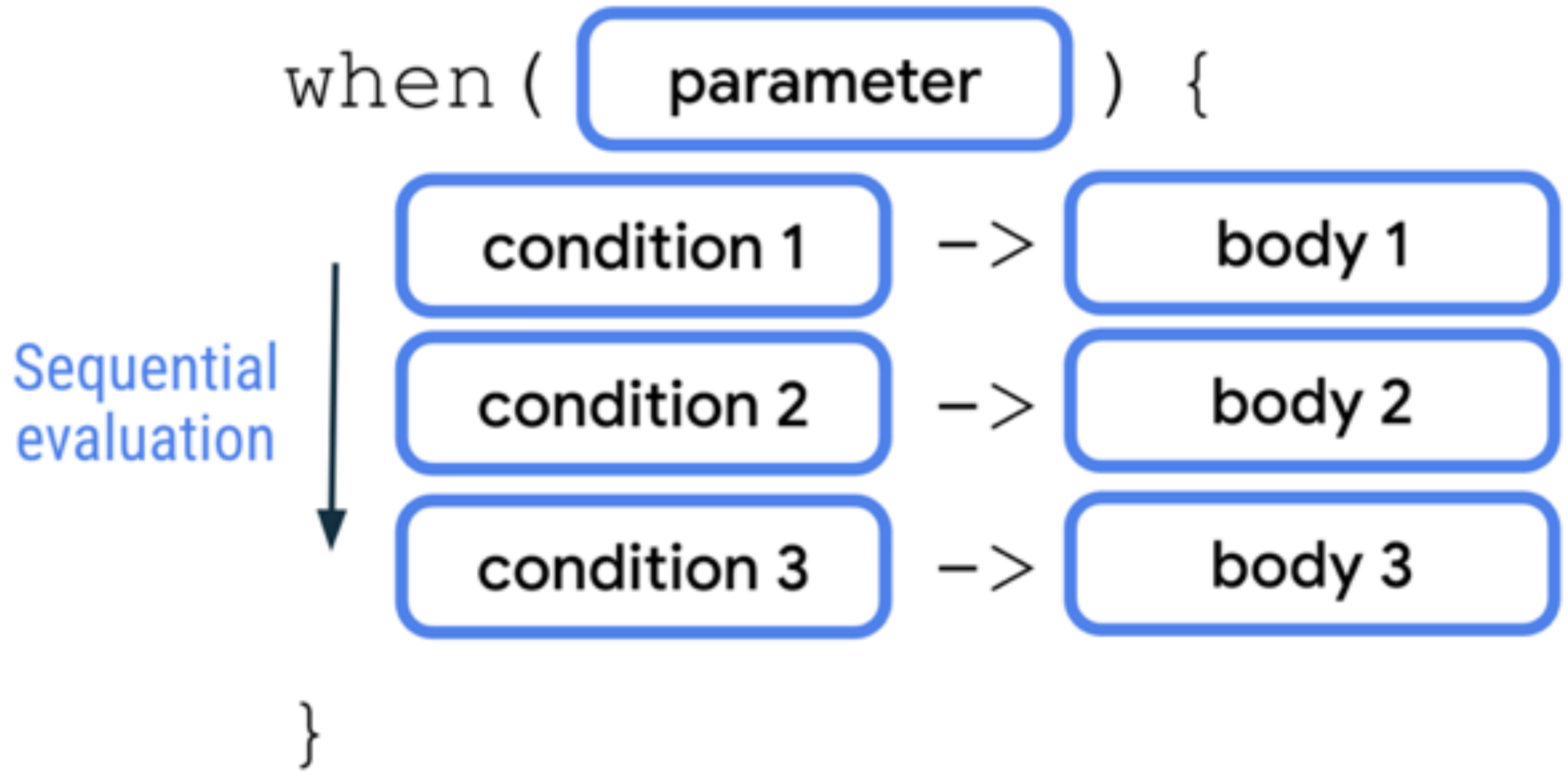
```
..    //inclusive, closed-ended range
..<   //exclusive, open-ended range
```

Ranges in if/else statements

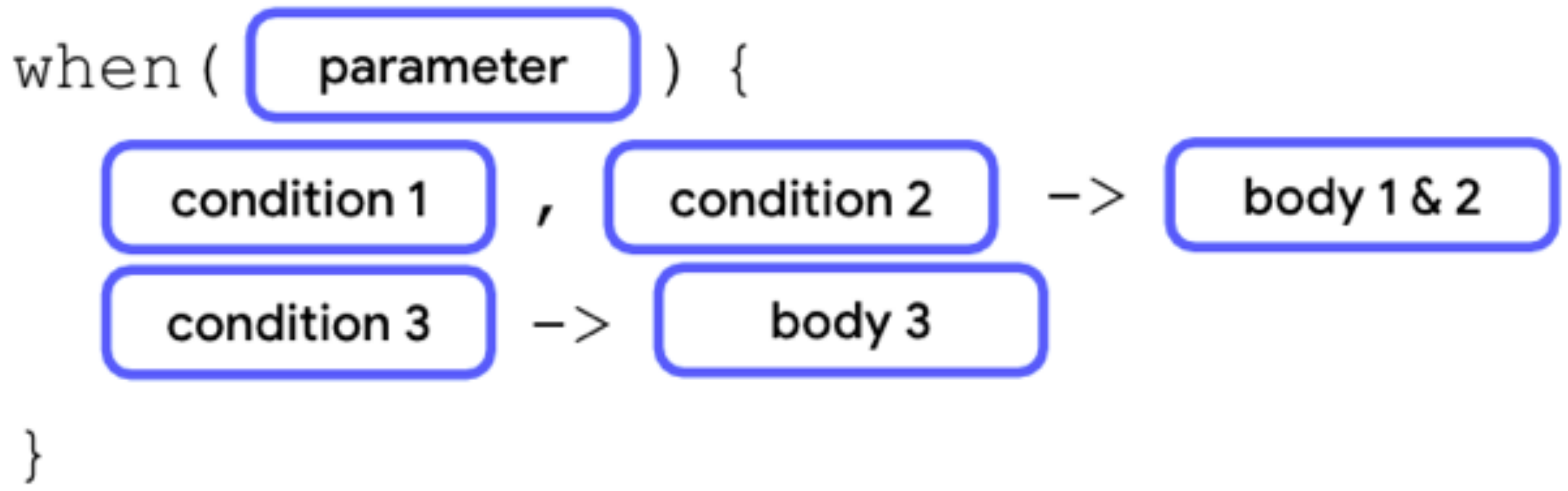
```
val numberOfStudents = 50
if (numberOfStudents in 1..100) {
    println(numberOfStudents)
}

=> 50
```

There are no spaces around the "range to" operator (1..100)

when statements

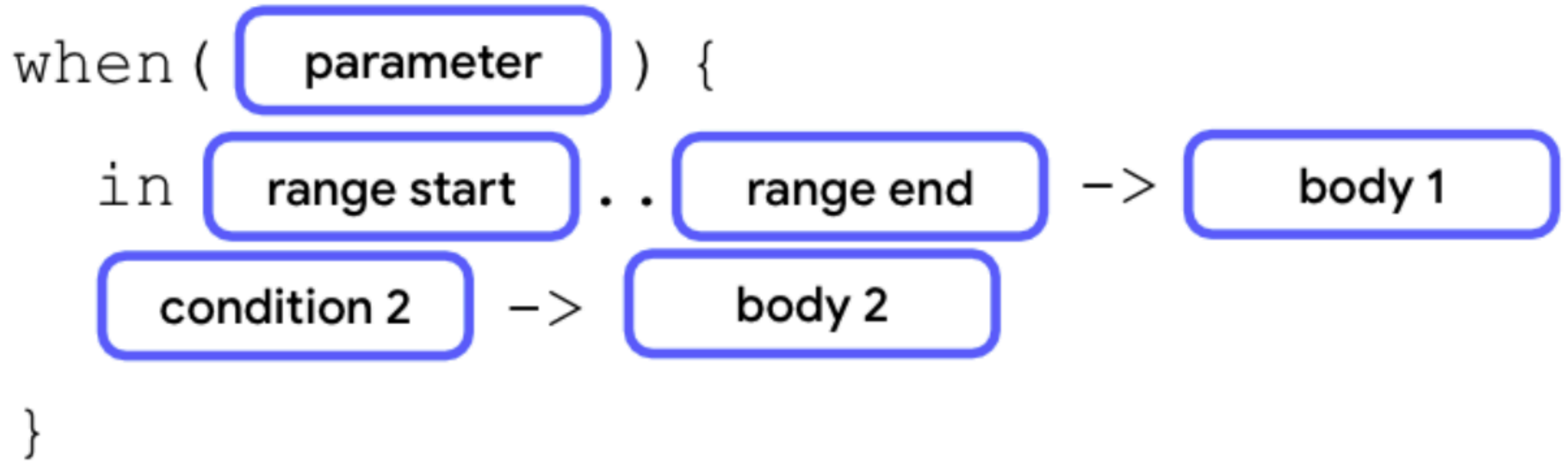


when statements



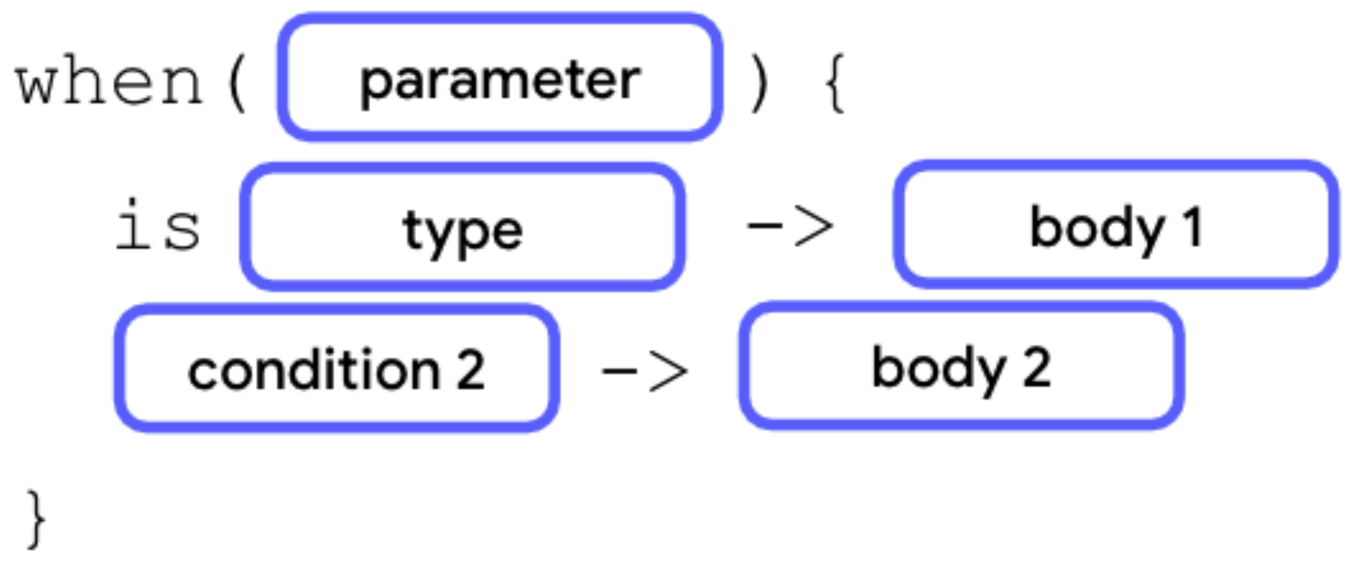
```
when ( parameter ) {
    condition 1 , condition 2 -> body 1 & 2
    condition 3 -> body 3
}
```

when statements



```
when ( parameter ) {
    in range start .. range end -> body 1
    condition 2 -> body 2
}
```

when statements



```
when ( parameter ) {
    is type -> body 1
    condition 2 -> body 2
}
```

when statements

```
fun main() {
    val trafficLightColor = "Black"

    when (trafficLightColor) {
        "Red" -> println("Stop")
        "Yellow" -> println("Slow")
        "Green" -> println("Go")
        else -> println("Invalid traffic-light color")
    }
}
```

if/else expressions

```
val name = if( condition ) {
    body 1
} else {
    body 2
}
```

if/else expressions

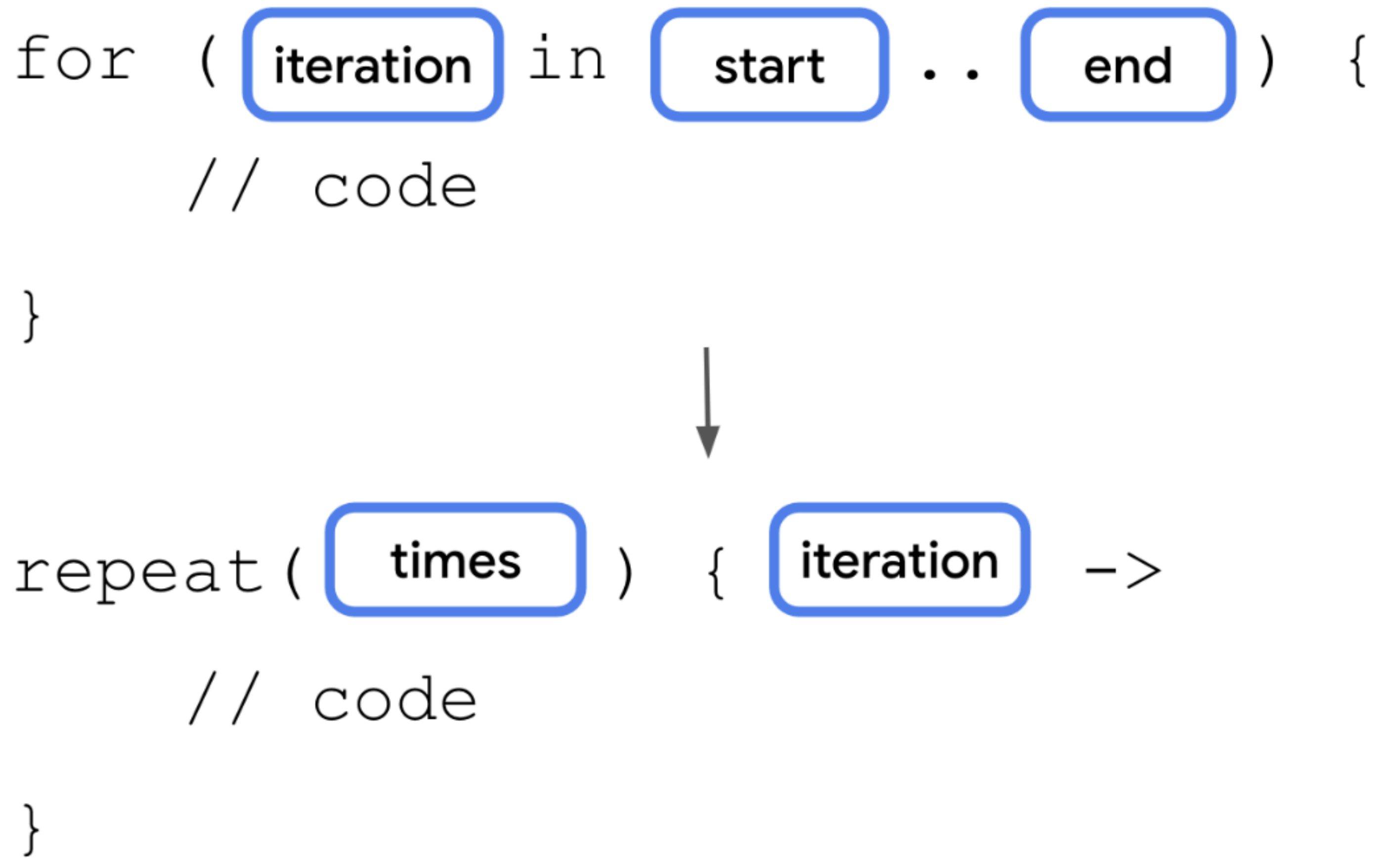
```
val name = if ( condition ) {
    body 1
} else {
    body 2
}
```

```
val name = if ( condition ) expression 1 else expression 2
```

if/else expressions

```
fun main() {
    val trafficLightColor = "Black"

    val message =
      if (trafficLightColor == "Red") "Stop"
      else if (trafficLightColor == "Yellow") "Slow"
      else if (trafficLightColor == "Green") "Go"
      else "Invalid traffic-light color"

    println(message)
}
```

repeat loops



```
for ( iteration in start .. end ) {
    // code
}
```

```
repeat( times ) { iteration ->
    // code
}
```

repeat loops

```
repeat(2) {
    println("Hello!")
}


=>
Hello!
Hello!
```

Null Safety

Null safety

- In Kotlin, variables cannot be null by default
- You can explicitly assign a variable to null using the safe call operator
- Allow null-pointer exceptions using the !! operator
- You can test for null using the elvis (?:) operator

Variables cannot be null

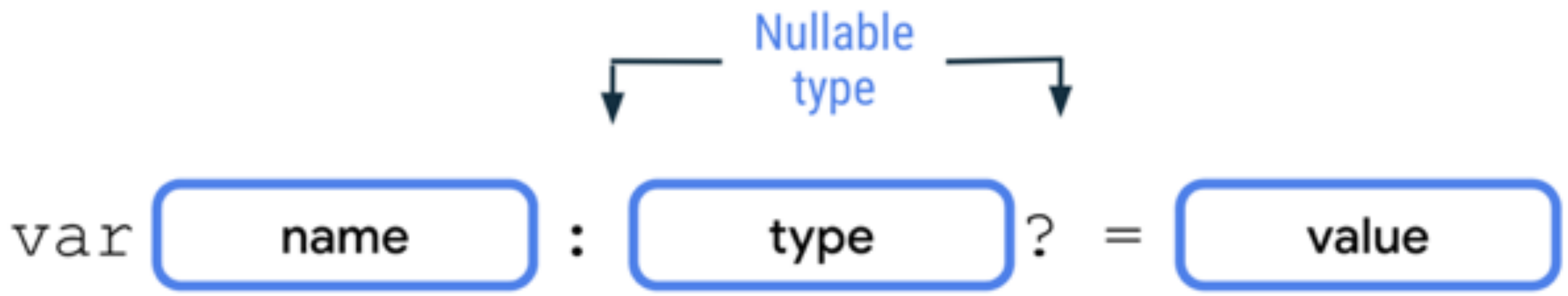
- In Kotlin, `null` variables are not allowed by default.
- Declare an `Int` and assign `null` to it.

```
var numberOfBooks: Int = null
```

```
⇒ error: null can not be a value of a non-null
type Int
```

Safe call operator

- Nullable types are variables that can hold `null`.
- Non-null types are variables that can't hold `null`.



Safe call operator

- The safe call operator (?), after the type indicates that a variable can be `null`.
- Declare an `Int?` as nullable

```
var numberOfBooks: Int? = null
```

In general, do not set a variable to null as it may have unwanted consequences.

Handle nullable variables

```
var numberOfBooks: Int? = 6
numberOfBooks = numberOfBooks.dec()
println(numberOfBooks)
```

Handle nullable variables

```
var numberOfBooks: Int? = 6
numberOfBooks = numberOfBooks.dec()
println(numberOfBooks)


=>
Only safe (?.) or non-null asserted (!!.) calls
are allowed on a nullable receiver of type Int?
```

Handle nullable variables

```
var numberOfBooks: Int? = 6
numberOfBooks = numberOfBooks?.dec()
println(numberOfBooks)


=> 5
```

nullable variable ?. method/property

Handle nullable variables

```
var numberOfBooks: Int? = null
numberOfBooks = numberOfBooks?.dec()
println(numberOfBooks)


=> null
```

The !! operator

- If you're certain a variable won't be null, use `!!` to force the variable into a non-null type. Then you can call methods/properties on it.

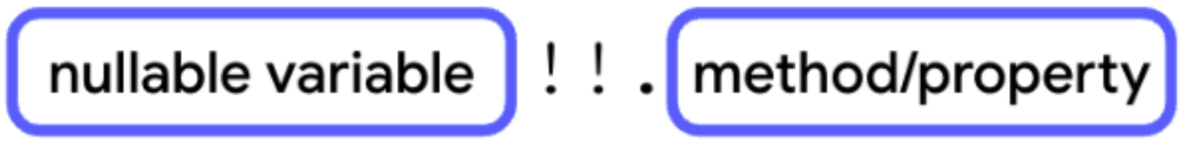
```
val len = s!!.length

// throws NullPointerException if s is null
```

KOÇ UNIVERSITY

The !! operator

- If you’re certain a variable won’t be null, use `!!` to force the variable into a non-null type. Then you can call methods/properties on it.

```
val len = s!!.length

// throws NullPointerException if s is null
```



Warning: Because !! will throw an exception, it should only be used when it would be exceptional to hold a null value.

Elvis operator

- Chain null tests with the `?:` operator.

```
numberOfBooks = numberOfBooks?.dec() ?: 0
```

val name = nullable variable ?. method/property ?: default value

Workshop