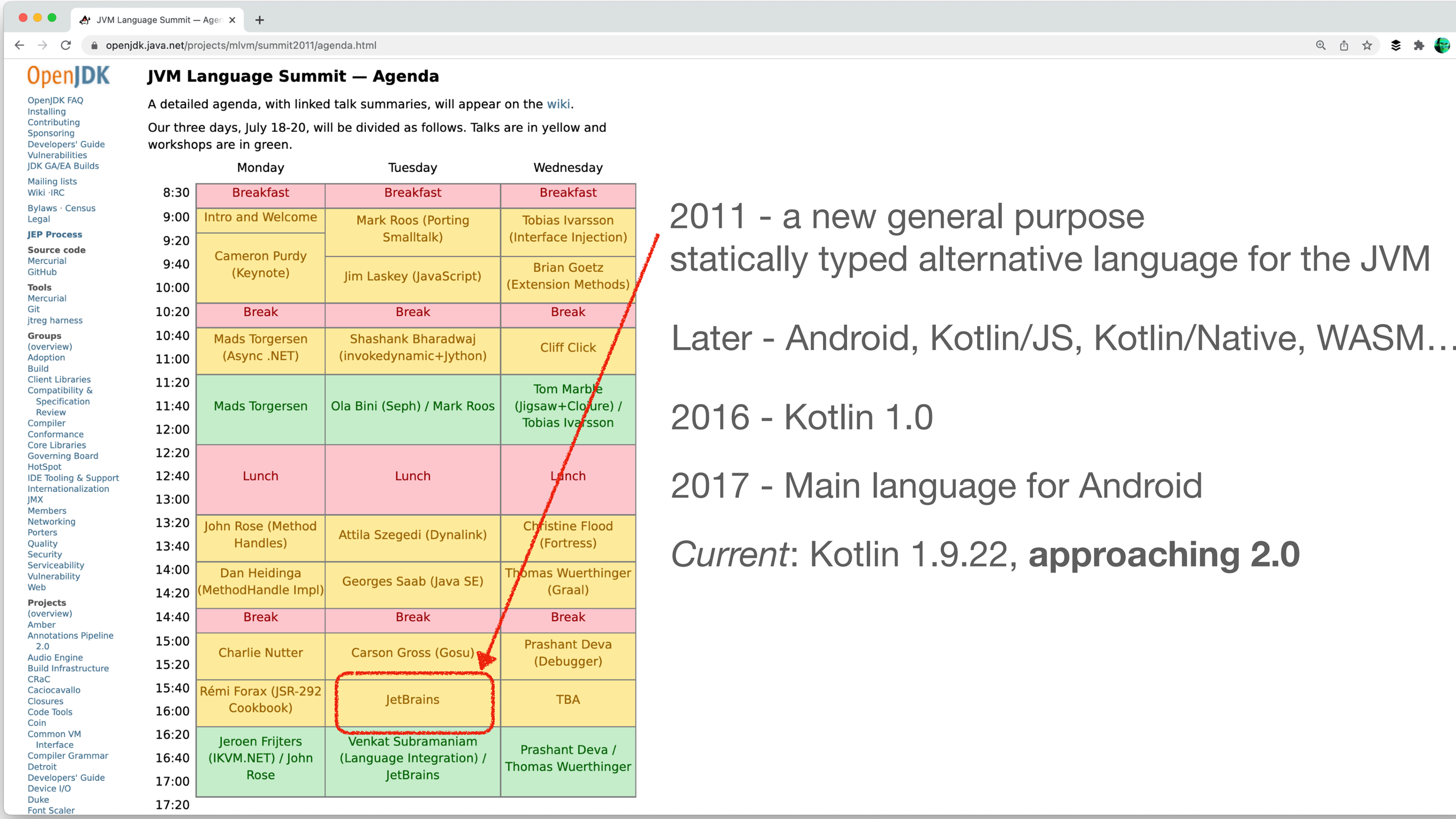
Idiomatic Kotlin

@antonarhipov

JET
BRAINS


JET BRAINS
QD
TC
IJ
@antonarhipov
Java™
Champions


JVM Language Summit — Ager
openjdk.java.net/projects/mlvm/summit2011/agenda.html
OpenJDK
OpenJDK FAQ
Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds
Mailing lists
Wiki · IRC
Bylaws · Census
Legal
JEP Process
Source code
Mercurial
GitHub
Tools
Mercurial
Git
jtreg harness
Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters
Quality
Security
Serviceability
Vulnerability
Web
Projects
(overview)
Amber
Annotations Pipeline 2.0
Audio Engine
Build Infrastructure
CRaC
Caciocavallo
Closures
Code Tools
Coin
Common VM Interface
Compiler Grammar
Detroit
Developers' Guide
Device I/O
Duke
Font Scaler
JVM Language Summit — Agenda
A detailed agenda, with linked talk summaries, will appear on the wiki.
Our three days, July 18-20, will be divided as follows. Talks are in yellow and workshops are in green.
Monday
Tuesday
Wednesday
8:30 Breakfast Breakfast Breakfast
9:00 Intro and Welcome Mark Roos (Porting Smalltalk) Tobias Ivarsson (Interface Injection)
9:20 Cameron Purdy (Keynote)
9:40 Jim Laskey (JavaScript) Brian Goetz (Extension Methods)
10:00
10:20 Break Break Break
10:40 Mads Torgersen (Async .NET) Shashank Bharadwaj (invokedynamic+Jython) Cliff Click
11:00
11:20 Mads Torgersen Ola Bini (Seph) / Mark Roos Tom Marble (Jigsaw+Closure) / Tobias Ivarsson
11:40
12:00
12:20 Lunch Lunch Lunch
12:40
13:00
13:20 John Rose (Method Handles) Attila Szegedi (Dynalink) Christine Flood (Fortress)
13:40
14:00 Dan Heidinga (MethodHandle Impl) Georges Saab (Java SE) Thomas Wuerthinger (Graal)
14:20
14:40 Break Break Break
15:00 Charlie Nutter Carson Gross (Gosu) Prashant Deva (Debugger)
15:20
15:40 Rémi Forax (JSR-292 Cookbook) JetBrains TBA
16:00
16:20 Jeroen Frijters (IKVM.NET) / John Rose Venkat Subramaniam (Language Integration) / JetBrains Prashant Deva / Thomas Wuerthinger
16:40
17:00
17:20
2011 - a new general purpose statically typed alternative language for the JVM
Later - Android, Kotlin/JS, Kotlin/Native, WASM…
2016 - Kotlin 1.0
2017 - Main language for Android
Current: Kotlin 1.9.22, approaching 2.0

Me becoming Kotlin advocate:

Expectation

Reality

Home
Get started
Kotlin overview
What's new
Basics
Basic syntax
Idioms
Kotlin by example ↗
Coding conventions
Concepts
Multiplatform programming
Platforms
Releases and roadmap
Standard library
Official libraries
API reference
Language reference
Tools
Learning materials
Other resources

Idioms

Edit page Last modified: 23 June 2021

A collection of random and frequently used idioms in Kotlin. If you have a favorite idiom, contribute it by sending a pull request.

Create DTOs (POJOs/POCOs)

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

provides a `Customer` class with the following functionality:

- getters (and setters in case of `var` s) for all properties
- `equals()`
- `hashCode()`
- `toString()`
- `copy()`
- `component1()` , `component2()` , ..., for all properties (see Data classes)

Default values for function parameters

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

Functions & expressions

Extensions & stdlib

Null-safety

Type-safe builders, a.k.a DSL

Top-Level (Extension) Functions

```
fun main() {
    doSomething()
}

fun doSomething() {

}
```

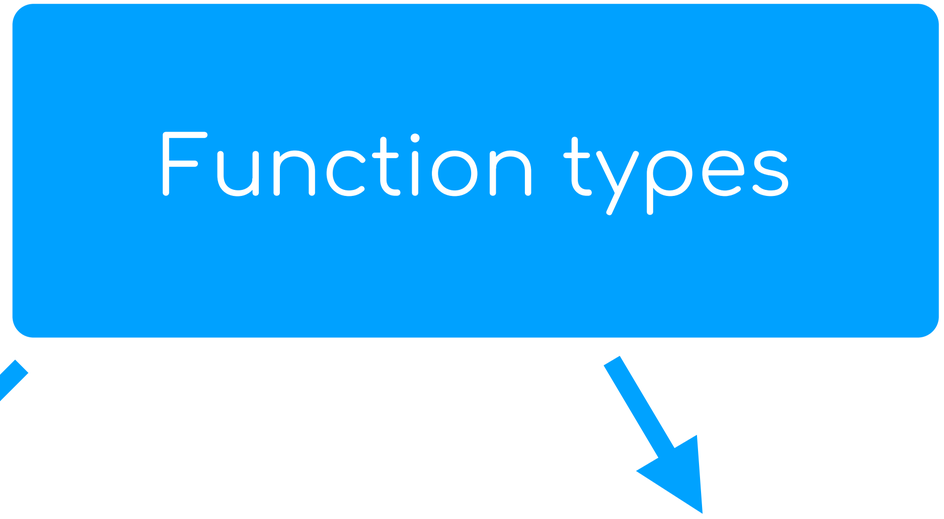
```
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}
```

```
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}

fun doMoreStuff(callback: () → Unit) {
    callback()
}
```

```
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}

fun doMoreStuff(callback: () → Unit) {
    callback()
}

fun finishWork() {
    TODO("Not implemented yet")
}
```

Just functions,
no classes!

```
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}

fun doMoreStuff(callback: () -> Unit) {
    callback()
}

fun finishWork() {
    TODO("Not implemented yet")
}
```




```
val callback: () -> Unit = { }
```

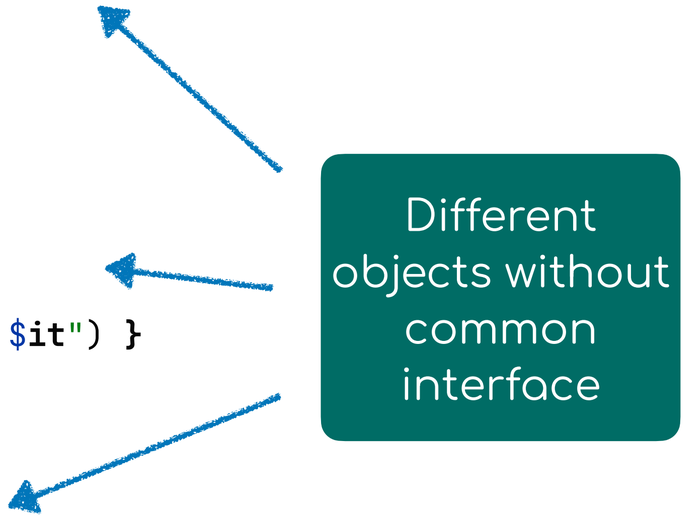
```
fun stopProducer() {
    if (this::producer.isInitialized) {
        runBlocking {
            runCatching {
                producer.close()
            }.onFailure {
                println("failed to close queue producer: $it")
            }
        }
    }
}

fun stopBroker() {
    if (this::broker.isInitialized) {
        runBlocking {
            runCatching {
                broker.close()
            }.onFailure { println("failed to close queue producer: $it") }
        }
    }
}

fun stopService() {
    if (this::service.isInitialized) {
        runBlocking {
            runCatching {
                service.close()
            }.onFailure { println("failed to close queue producer: $it") }
        }
    }
}
```

Go to IDE

Duplicates

```
fun stopProducer() {
    if (this::producer.isInitialized) {
        runBlocking {
            runCatching {
                producer.close()
            }.onFailure {
                println("failed to close queue producer: $it")
            }
        }
    }
}

fun stopBroker() {
    if (this::broker.isInitialized) {
        runBlocking {
            runCatching {
                broker.close()
            }.onFailure { println("failed to close queue producer: $it") }
        }
    }
}

fun stopService() {
    if (this::service.isInitialized) {
        runBlocking {
            runCatching {
                service.close()
            }.onFailure { println("failed to close queue producer: $it") }
        }
    }
}
```

```
fun stopProducer() {
    if (this::producer.isInitialized) {
        runBlocking {
            runCatching {
                producer.close()
            }.onFailure {
                println("failed to close queue producer: $it")
            }
        }
    }
}

fun stopBroker() {
    if (this::broker.isInitialized) {
        runBlocking {
            runCatching {
                broker.close()
            }.onFailure { println("failed to close queue producer: $it") }
        }
    }
}

fun stopService() {
    if (this::service.isInitialized) {
        runBlocking {
            runCatching {
                service.close()
            }.onFailure { println("failed to close queue producer: $it") }
        }
    }
}
```



```
fun stopResource(predicate: () → Boolean, close: suspend () → Unit) {
    if (predicate()) {
        runBlocking {
            runCatching {
                close()
            }.onFailure {
                println("failed to close the resource: $it")
            }
        }
    }
}
```

```
fun stopResource(predicate: () → Boolean, close: suspend () → Unit) {
    if (predicate()) {
        runBlocking {
            runCatching {
                close()
            }.onFailure {
                println("failed to close the resource: $it")
            }
        }
    }
}
```

```
fun stopResource(predicate: () → Boolean, close: suspend () → Unit) {
    if (predicate()) {
        runBlocking {
            runCatching {
                close()
            }.onFailure {
                println("failed to close the resource: $it")
            }
        }
    }
}
```

```
class StringUtils {
    companion object {
        fun isPhoneNumber(s: String) =
            s.length == 7 && s.all { it.isDigit() }
    }
}
```

```
class StringUtils {
    companion object {
        fun isPhoneNumber(s: String) =
            s.length == 7 && s.all { it.isDigit() }
    }
}
```

```
object StringUtils {
    fun isPhoneNumber(s: String) =
        s.length == 7 && s.all { it.isDigit() }
}
```

```
class StringUtils {
    companion object {
        fun isPhoneNumber(s: String) =
            s.length == 7 && s.all { it.isDigit() }
    }
}
```

```
object StringUtils {
    fun isPhoneNumber(s: String) =
        s.length == 7 && s.all { it.isDigit() }
}
```

```
fun isPhoneNumber(s: String) =
    s.length == 7 && s.all { it.isDigit() }
```

```
class StringUtils {
    companion object {
        fun isPhoneNumber(s: String) =
            s.length == 7 && s.all { it.isDigit() }
    }
}
```

```
object StringUtils {
    fun isPhoneNumber(s: String) =
        s.length == 7 && s.all { it.isDigit() }
}
```

```
fun isPhoneNumber(s: String) =
    s.length == 7 && s.all { it.isDigit() }
```

```
fun String.isPhoneNumber() =
    length == 7 && all { it.isDigit() }
```

Extension or a member?

https://kotlinlang.org/docs/coding-conventions.html#extension-functions

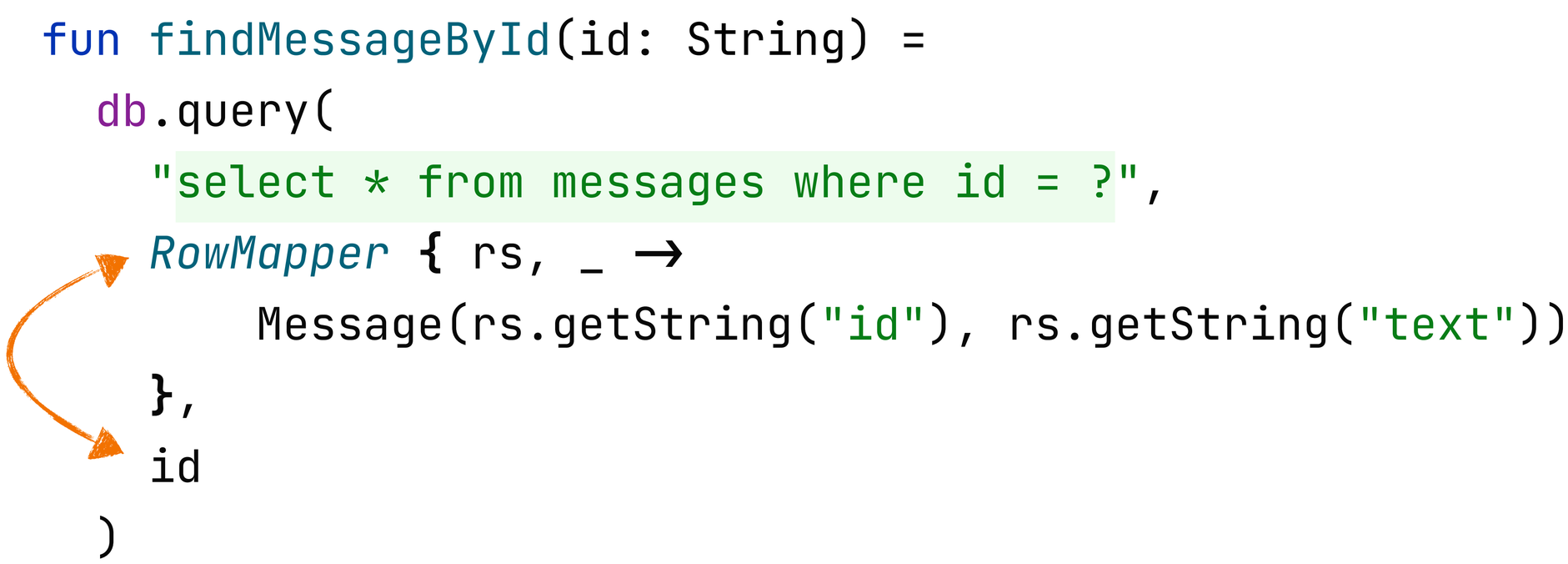
- Use extension functions liberally
- Restrict the visibility to minimize API pollution
- As necessary, use local extension functions, member extension functions, or top-level extension functions with private visibility

```
val db: JdbcTemplate = ...

fun findMessageById(id: String) =
  db.query(
    "select * from messages where id = ?",
    RowMapper { rs, _ →
        Message(rs.getString("id"), rs.getString("text"))
    },
    id
  )
```

```java
@Override
public <T> List<T> query(String sql, RowMapper<T> rowMapper, @Nullable Object... args)
    throws DataAccessException {
  return result(query(sql, args, new RowMapperResultSetExtractor<>(rowMapper)));
}
```

```
fun findMessageById(id: String) =
  db.query(
    "select * from messages where id = ?",
    RowMapper { rs, _ ->
        Message(rs.getString("id"), rs.getString("text"))
    },
    id
  )
```

```
fun findMessageById(id: String) =
  db.query(
    "select * from messages where id = ?",
    RowMapper { rs, _ →
        Message(rs.getString("id"), rs.getString("text"))
    },
    id
  )
```


fun findMessageById(id: String) =
db.query(
"select * from messages where id = ?",
RowMapper { rs, _ →
Message(rs.getString("id"), rs.getString("text"))
},
id
)

```
fun findMessageById(id: String) =
  db.query(
    "select * from messages where id = ?",
    id,
    RowMapper { rs, _ →
        Message(rs.getString("id"), rs.getString("text"))
    }
  )
```

```
fun findMessageById(id: String) =
  db.query(
    "select * from messages where id = ?",
    id,
    { rs, _ →
        Message(rs.getString("id"), rs.getString("text"))
    }
  )
```

SAM conversion

```
fun findMessageById(id: String) =
  db.query(
    "select * from messages where id = ?",
    id)
    { rs, _ →
        Message(rs.getString("id"), rs.getString("text"))
    }
```

Trailing lambda parameter

```
fun findMessageById(id: String) =
  db.query("select * from messages where id = ?", id ) { rs, _ →
        Message(rs.getString("id"), rs.getString("text"))
  }
```

```
fun <T> JdbcOperations.query(sql: String, vararg args: Any, function: (ResultSet, Int) → T): List<T> =
    query(sql, RowMapper { rs, i → function(rs, i) }, *args)
```

Extension function!

```
fun findMessageById(id: String) =
  db.query("select * from messages where id = ?", id ) { rs, _ →
        Message(rs.getString("id"), rs.getString("text"))
  }
```

Top-level functions, Extensions,

Single-expression functions,

Trailing lambda, Varargs,

SAM conversion

Scope functions

Home
Get started
Kotlin overview
What's new
Basics
Concepts
Multiplatform programming
Platforms
Releases and roadmap
Standard library
Collections
Scope functions
Opt-in requirements
Official libraries
API reference
Language reference
Tools
Learning materials
Other resources

Scope functions

Edit page Last modified: 30 August 2021

The Kotlin standard library contains several functions whose sole purpose is to execute a block of code within the context of an object. When you call such a function on an object with a lambda expression provided, it forms a temporary scope. In this scope, you can access the object without its name. Such functions are called *scope functions*. There are five of them: `let`, `run`, `with`, `apply`, and `also`.

Basically, these functions do the same: execute a block of code on an object. What's different is how this object becomes available inside the block and what is the result of the whole expression.

Here's a typical usage of a scope function:

```
Person("Alice", 20, "Amsterdam").let {
    println(it)
    it.moveTo("London")
    it.incrementAge()
    println(it)
}
```

Target platform: JVM Running on kotlin v.1.5.30

If you write the same without `let`, you'll have to introduce a new variable and repeat its name whenever you use it.

```
val alice = Person("Alice", 20, "Amsterdam")
```

Scope functions
Function selection
Distinctions
Context object: this or it
Return value
Functions
let
with
run
apply
also
takeIf and takeUnless

```
val dataSource = BasicDataSource()
dataSource.driverClassName = "com.mysql.jdbc.Driver"
dataSource.url = "jdbc:mysql://domain:3309/db"
dataSource.username = "username"
dataSource.password = "password"
dataSource.maxTotal = 40
dataSource.maxIdle = 40
dataSource.minIdle = 4
```

```
val dataSource = BasicDataSource()
dataSource.driverClassName = "com.mysql.jdbc.Driver"
dataSource.url = "jdbc:mysql://domain:3309/db"
dataSource.username = "username"
dataSource.password = "password"
dataSource.maxTotal = 40
dataSource.maxIdle = 40
dataSource.minIdle = 4
```

```
val dataSource = BasicDataSource().apply {
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
    maxTotal = 40
    maxIdle = 40
    minIdle = 4
}
```

```kotlin
public inline fun <T> T.apply(block: T.() -> Unit): T {
    block()
    return this
}
```

```kotlin
val dataSource = BasicDataSource().apply {
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
    maxTotal = 40
    maxIdle = 40
    minIdle = 4
}
```

```kotlin
public inline fun <T> T.apply(block: T.() -> Unit): T {
    block()
    return this
}
```

Lambda with receiver

```kotlin
val dataSource = BasicDataSource().apply {
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
    maxTotal = 40
    maxIdle = 40
    minIdle = 4
}
```

`let()` as a helper for a complex condition

```
if (some.complex.expression.let { it is Type && it.has.some.property }) {
    ...
}
```

`let()` as a helper for a complex condition

```
if (some.complex.expression.let { it is Type && it.has.some.property }) {
    ...
}
```

`let()` as a helper for a complex condition

```
if (some.complex.expression.let { it is Type && it.has.some.property }) {
    ...
}
```

`let()` as a helper for a complex condition

```
if (some.complex.expression.let { it is Type && it.has.some.property }) {
    ...
}
```

```
if (retrieveOrder().let { it is Subscription && it.customer.name == "Anton"}) {
    ...
}
```

Function selection

To help you choose the right scope function for your purpose, we provide the table of key differences between them.

Function	Object reference	Return value	Is extension function
`let`	`it`	Lambda result	Yes
`run`	`this`	Lambda result	Yes
`run`	-	Lambda result	No: called without the context object
`with`	`this`	Lambda result	No: takes the context object as an argument.
`apply`	`this`	Context object	Yes
`also`	`it`	Context object	Yes

```
fun makeDir(path: String) = path.let { File(it) }.also { it.mkdirs() }
```

```
fun makeDir(path: String) = path.let { File(it) }.also { it.mkdirs() }
```

Don’t overuse the scope functions!

```
fun makeDir(path: String) = path.let { File(it) }.also { it.mkdirs() }
```

Don't overuse the scope functions!

```
fun makeDir(path: String) : File {
    val file = File(path)
    file.mkdirs()
    return file
}
```

This is simpler!

```
fun makeDir(path: String) = path.let { File(it) }.also { it.mkdirs() }
```

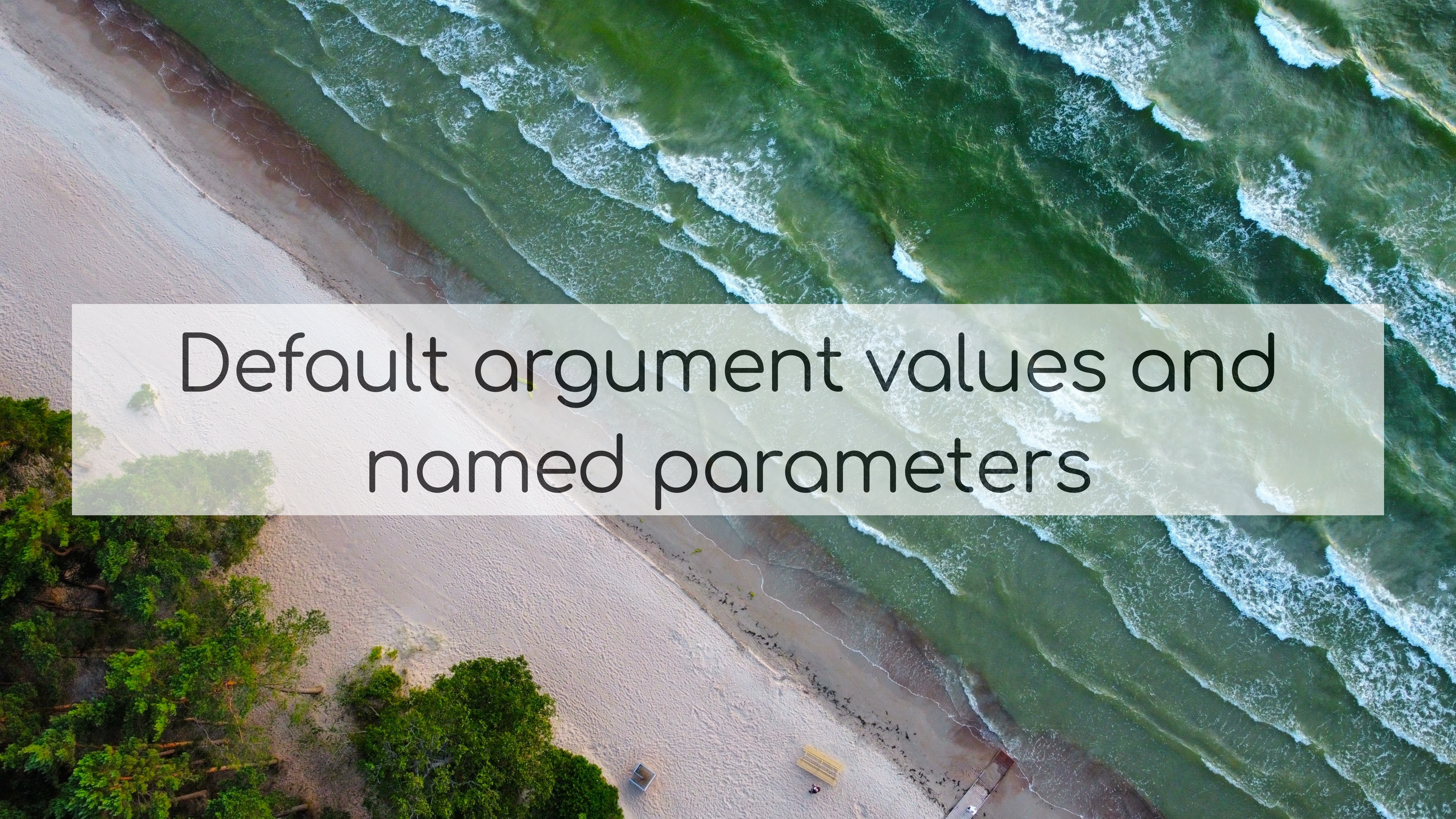
Don't overuse the scope functions!

```
fun makeDir(path: String) : File {
    val file = File(path)
    file.mkdirs()
    return file
}
```

This is simpler!

OK, this one is actually fine :)

```
fun makeDir(path: String) = File(path).also { it.mkdirs() }
```



Default argument values and named parameters

```
fun find(name: String){
    find(name, true)
}

fun find(name: String, recursive: Boolean){
}
```

Function overloading

```
fun find(name: String){
    find(name, true)
}

fun find(name: String, recursive: Boolean){
}
```

Function overloading

```
fun find(name: String, recursive: Boolean = true){
}
```

Default argument value

```
fun find(name: String){
    find(name, true)
}

fun find(name: String, recursive: Boolean){
}
```

Function overloading

```
fun find(name: String, recursive: Boolean = true){
}
```

Default argument value

```
fun main() {
    find("myfile.txt")
}
```

```
class Figure(
    val width: Int = 1,
    val height: Int = 1,
    val depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "This is a 3d figure",
)
```

```
Figure(Color.RED, "Red figure")
```

```
class Figure(
    val width: Int = 1,
    val height: Int = 1,
    val depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "This is a 3d figure",
)
```

```
Figure(Color.RED, "Red figure")
```

Compilation error

```
class Figure(
    val width: Int = 1,
    val height: Int = 1,
    val depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "This is a 3d figure",
)
```

```
Figure(color = Color.RED, description = "Red figure")
```

Default argument values diminish the need for overloading in most cases

Default argument values diminish the need for overloading in most cases

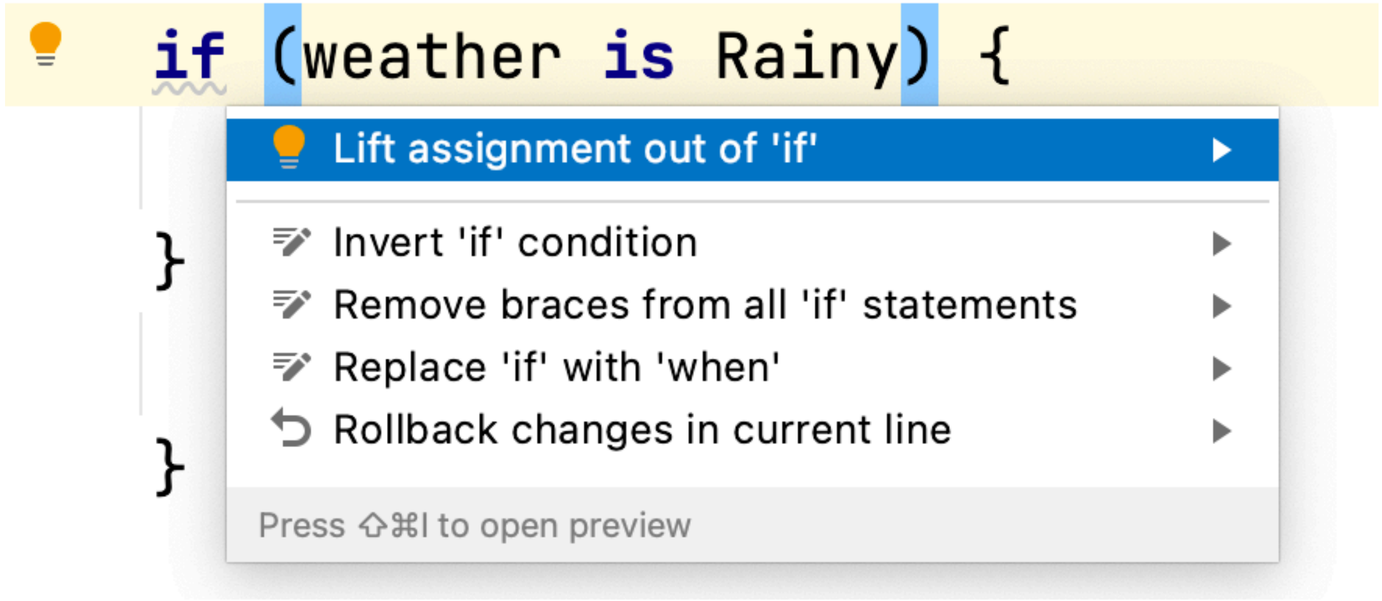
Named parameters is a necessary tool for working with default argument values

Expressions

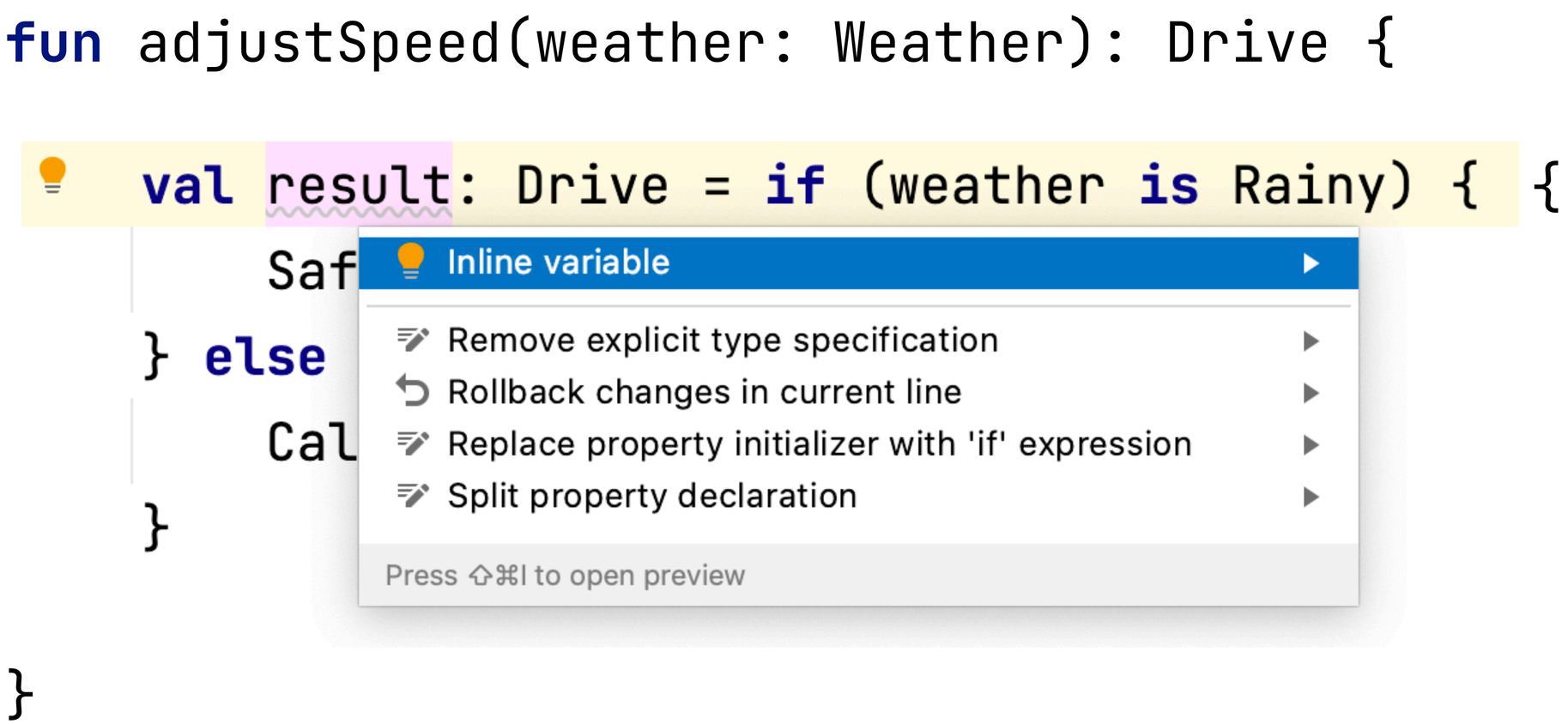
Let’s transform this code using Alt+Enter

```
fun adjustSpeed(weather: Weather): Drive {
    val result: Drive

    if (weather is Rainy) {
        result = Safe()
    } else {
        result = Calm()
    }

    return result
}
```

```
fun adjustSpeed(weather: Weather): Drive {
    val result: Drive

    if (weather is Rainy) {
    }
    }
}
```

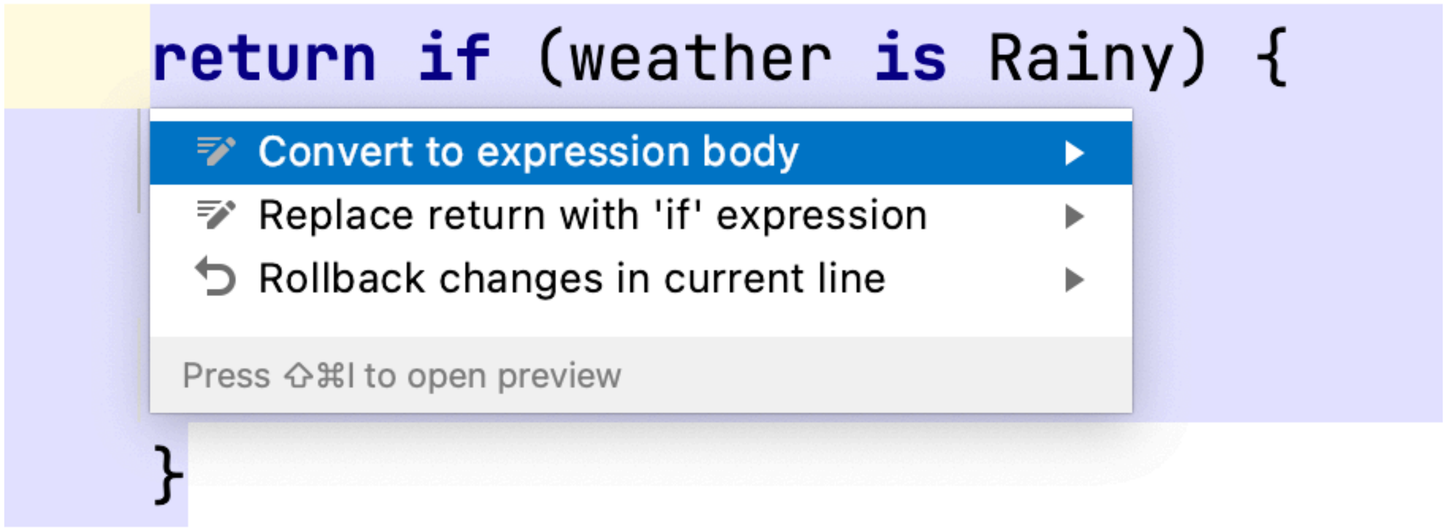


```
fun adjustSpeed(weather: Weather): Drive {

    val result: Drive = if (weather is Rainy) {
        Safe()
    } else {
        Calm()
    }

    return result
}
```


fun adjustSpeed(weather: Weather): Drive {
val result: Drive = if (weather is Rainy) {
Saf
} else
Cal
}
}
Inline variable
Remove explicit type specification
Rollback changes in current line
Replace property initializer with 'if' expression
Split property declaration
Press ⇧⌘I to open preview

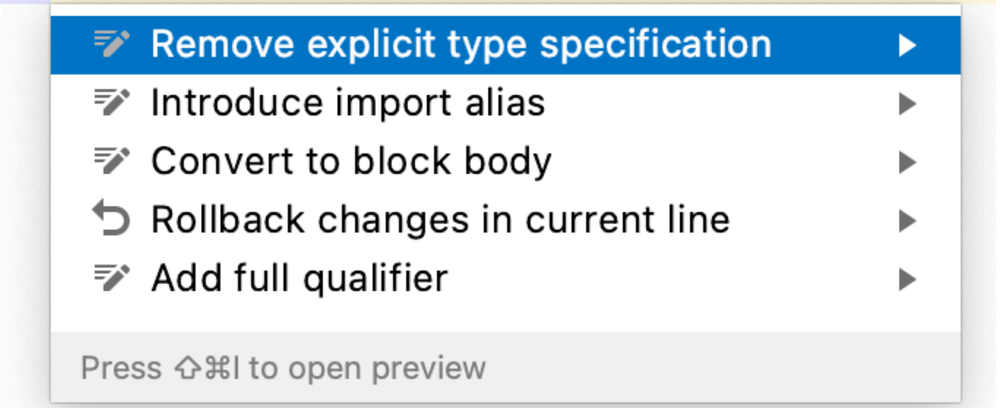
```
fun adjustSpeed(weather: Weather): Drive {

    return if (weather is Rainy) {
        Safe()
    } else {
        Calm()
    }

}
```

```
fun adjustSpeed(weather: Weather): Drive {
    return if (weather is Rainy) {
    }
}
```



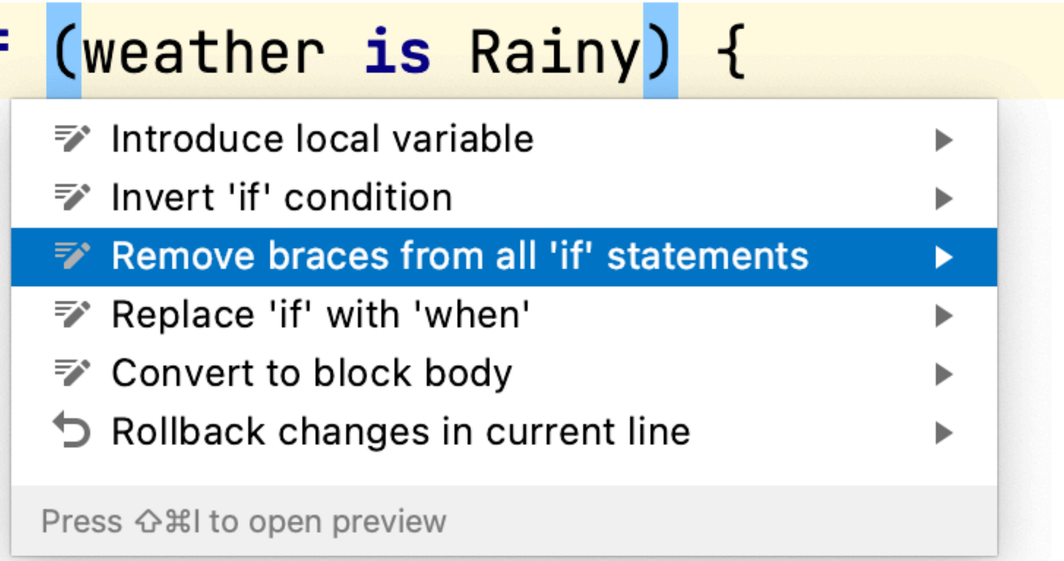
```
fun adjustSpeed(weather: Weather): Drive = if (weather is Rainy) {
    Safe()
} else {
    Calm()
}
```

```
fun adjustSpeed(weather: Weather): Drive = if (weather is Rainy) {
    Safe()
} else {
    Calm()
}
```



```
fun adjustSpeed(weather: Weather) = if (weather is Rainy) {
    Safe()
} else {
    Calm()
}
```

```
fun adjustSpeed(weather: Weather) = if (weather is Rainy) {
    Safe()
} else {
    Calm()
}
```



Is it concise?

Sure!

```
fun adjustSpeed(weather: Weather) = if (weather is Rainy) Safe() else Calm()
```

Is it readable?

It depends!

```
fun adjustSpeed(weather: Weather) = if (weather is Rainy) Safe() else Calm()
```

```
fun adjustSpeed(weather: Weather) = if (weather is Rainy) Safe() else Calm()
```

What does the function return?

For public API, keep the return type in the signature

```
fun adjustSpeed(weather: Weather): Drive = ...

fun adjustSpeed(weather: Weather) = ...
```

For private API it is generally OK to use type inference

What's new in Kotlin
Kotlin 1.8.20
Kotlin 1.8.0
Earlier versions
Kotlin 1.7.20
Kotlin 1.7.0
Kotlin 1.6.20
Kotlin 1.6.0
Kotlin 1.5.30
Kotlin 1.5.20
Kotlin 1.5.0
Kotlin 1.4.30
Kotlin 1.4.20
Kotlin 1.4.0
Kotlin 1.3
Kotlin 1.2
Kotlin 1.1
Releases and roadmap

To compile your module in the explicit API mode, add the following lines to your Gradle build script:

Kotlin Groovy

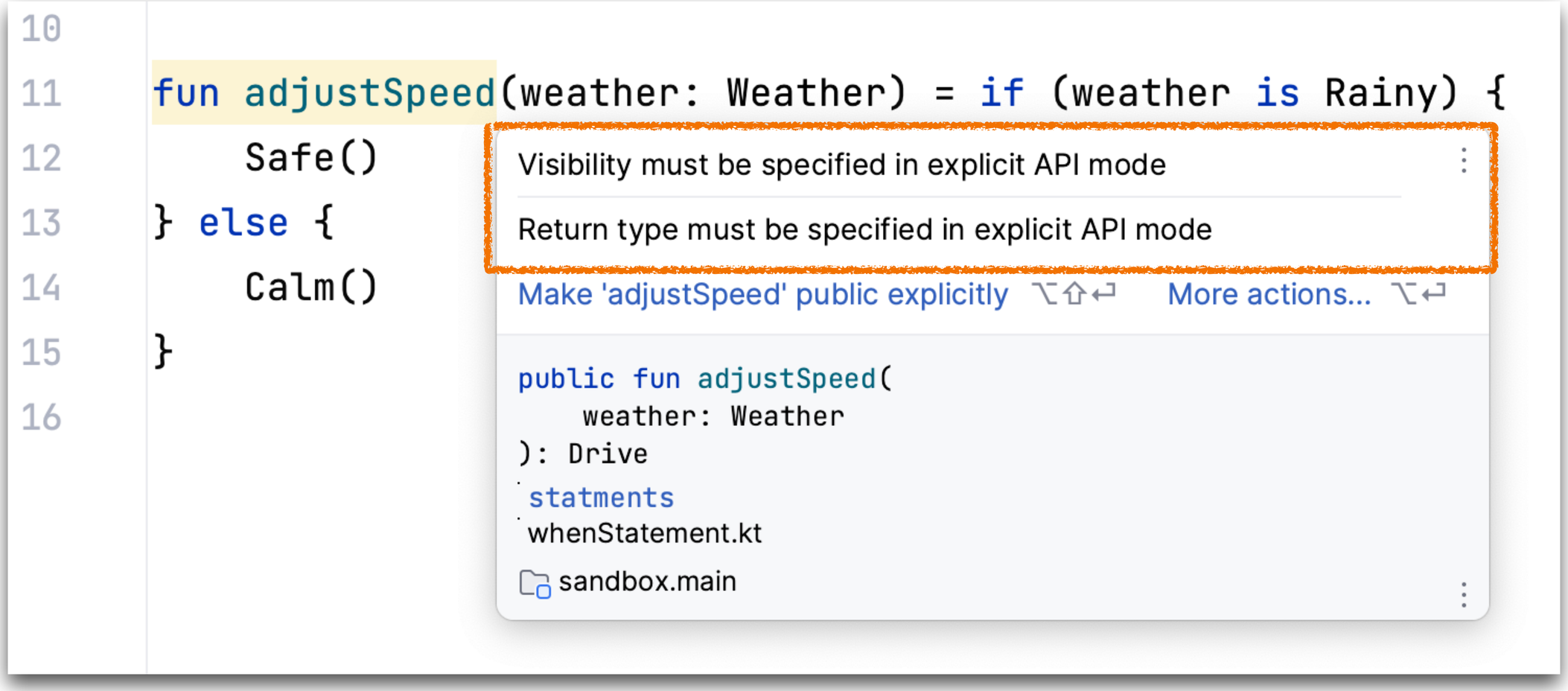
```
kotlin {
    // for strict mode
    explicitApi()
    // or
    explicitApi = ExplicitApiMode.Strict

    // for warning mode
    explicitApiWarning()
    // or
    explicitApi = ExplicitApiMode.Warning
}
```

When using the command-line compiler, switch to explicit API mode by adding the `-Xexplicit-api` compiler option with the value `strict` or `warning`.

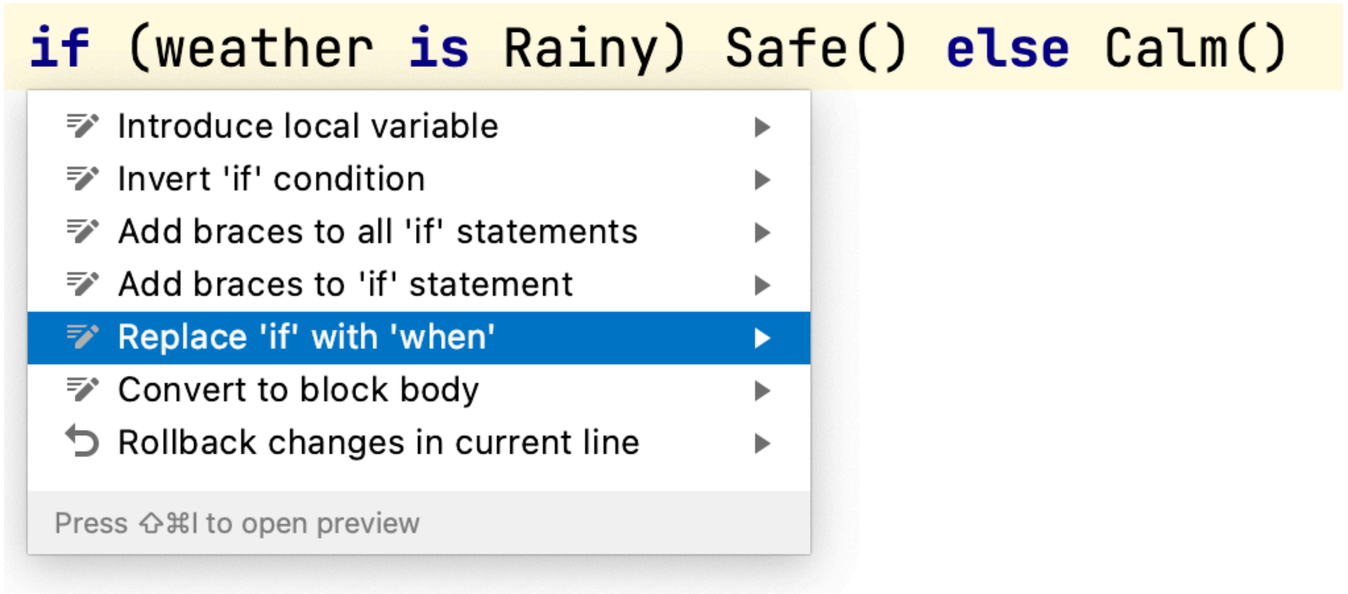
```
-Xexplicit-api={strict|warning}
```

Find more details about the explicit API mode in the KEEP ↗.

What's new in Kotlin 1.4
Language features and improvements
SAM conversions for Kotlin interfaces
Explicit API mode for library authors
Mixing named and positional arguments
Trailing comma
Callable reference improvements
Using break and continue inside when expressions included in loops
New tools in the IDE
New flexible Project Wizard
Coroutine Debugger


10
11 fun adjustSpeed(weather: Weather) = if (weather is Rainy) {
12 Safe()
13 } else {
14 Calm()
15 }
16
Visibility must be specified in explicit API mode
Return type must be specified in explicit API mode
Make 'adjustSpeed' public explicitly ⌥⇧↵
More actions... ⌥↵
public fun adjustSpeed(
weather: Weather
): Drive
statments
whenStatement.kt
sandbox.main

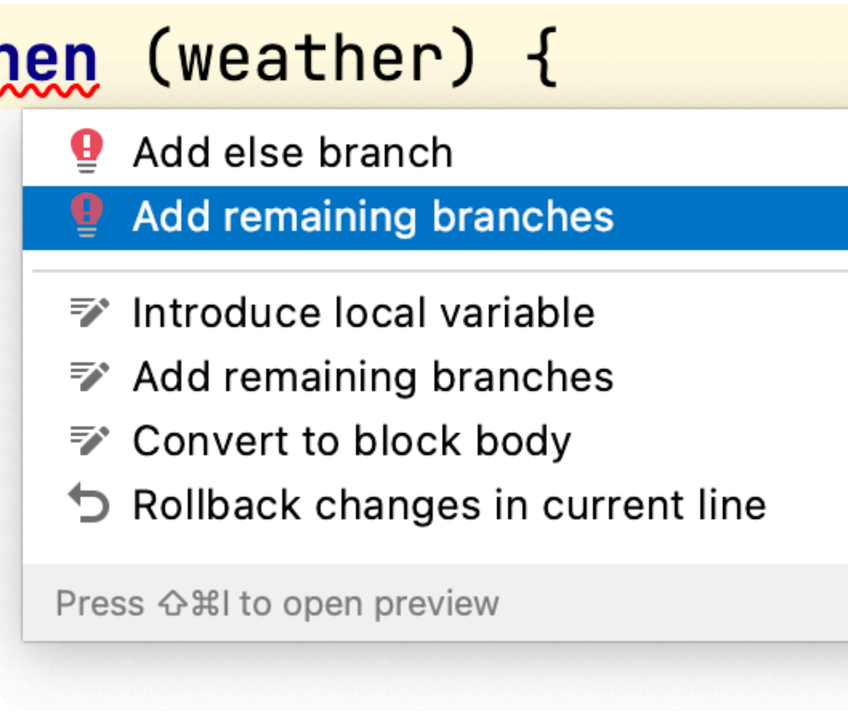
```
fun adjustSpeed(weather: Weather) = if (weather is Rainy) Safe() else Calm()
```



```
abstract class Weather
class Sunny : Weather()
class Rainy : Weather()


fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy → Safe()
    else → Calm()
}
```

```
sealed class Weather
class Sunny : Weather()
class Rainy : Weather()


fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy → Safe()
//     else → Calm()
}
```

```
sealed class Weather
class Sunny : Weather()
class Rainy : Weather()


fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy → Safe()
//     else → Calm()
}
```



```
sealed class Weather
class Sunny : Weather()
class Rainy : Weather()


fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy → Safe()
    is Sunny → TODO()
}
```

```
sealed class Weather
class Sunny : Weather()
class Rainy : Weather()
```

Use expressions!

Use **when** as expression body

Use sealed classes with **when** expression

```
fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy → Safe()
    is Sunny → TODO()
}
```

The *when* expression
is your friend!

if ⟷ when

```
val condition: Boolean = expression()
```

```
if (condition) {
    doThis()
} else {
    doThat()
}
```

```
when(condition) {
    true → doThis()
    false → doThat()
}
```

Use `try` as expression body

```
fun tryParse(number: String) : Int? {
    try {
        return Integer.parseInt(number)
    } catch (e: NumberFormatException) {
        return null
    }
}
```

Use `try` as expression body

```
fun tryParse(number: String) = try {
    Integer.parseInt(number)
} catch (e: NumberFormatException) {
    null
}
```

Null-safety

```
class Nullable {
    fun someFunction(){}
}

fun createNullable(): Nullable? = null
```

```
class Nullable {
    fun someFunction(){}
}

fun createNullable(): Nullable? = null

fun main() {
    val n: Nullable? = createNullable()

    n.someFunction()
}
```

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

Surround with null check ⌥⇧⏎ More actions... ⌥⏎

intro.Nullable
public final fun **someFunction**(): Unit

· sandbox.main

Consider using null-safe call

```
val order = retrieveOrder()

if (order == null || order.customer == null || order.customer.address == null){
    throw IllegalArgumentException("Invalid Order")
}
val city = order.customer.address.city
```

Consider using null-safe call

```
val order = retrieveOrder()

val city = order?.customer?.address?.city
```

Consider using null-safe call

```
val order = retrieveOrder()

val city = order?.customer?.address?.city
    ?: throw IllegalArgumentException("Invalid Order")
```

?.let

```
val order = retrieveOrder()

if (order ≠ null){
    processCustomer(order.customer)
}
```

?.let

```
val order = retrieveOrder()

if (order ≠ null){
    processCustomer(order.customer)
}
```

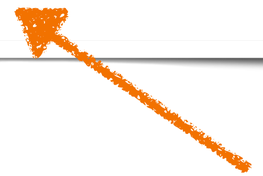
```
retrieveOrder()?.let {
    processCustomer(it.customer)
}
```

or

```
retrieveOrder()?.customer?.let(::processCustomer)
```

?.let

```
val order = retrieveOrder()

if (order ≠ null) {
    processCustomer(order.customer)
} else {
    // ignored
}
```



Don’t overuse the safe call

Use it only if you don’t care about **null**

Avoid not-null assertions !!

```
val order = retrieveOrder()

val city = order!!.customer!!.address!!.city
```

“You may notice that the double exclamation mark looks a bit rude:
it’s almost like you’re yelling at the compiler. This is intentional.” - ***Kotlin in Action***

Avoid not-null assertions !!

```
class MyTest {
    class State(val data: String)

    private var state: State? = null

    @BeforeEach
    fun setup() {
        state = State("abc")
    }

    @Test
    fun foo() {
        assertEquals("abc", state!!.data)
    }
}
```

Avoid not-null assertions `!!` - use `lateinit`

```
class MyTest {
    class State(val data: String)

    private var state: State? = null

    @BeforeEach
    fun setup() {
        state = State("abc")
    }

    @Test
    fun foo() {
        assertEquals("abc", state!!.data)
    }
}
```

```
class MyTest {
    class State(val data: String)

    private lateinit var state: State

    @BeforeEach
    fun setup() {
        state = State("abc")
    }

    @Test
    fun foo() {
        assertEquals("abc", state.data)
    }
}
```

Use elvis operator

```
class Person(val name: String?, val age: Int?)

val p = retrievePerson() ?: Person()
```

Use elvis operator as **return** and **throw**

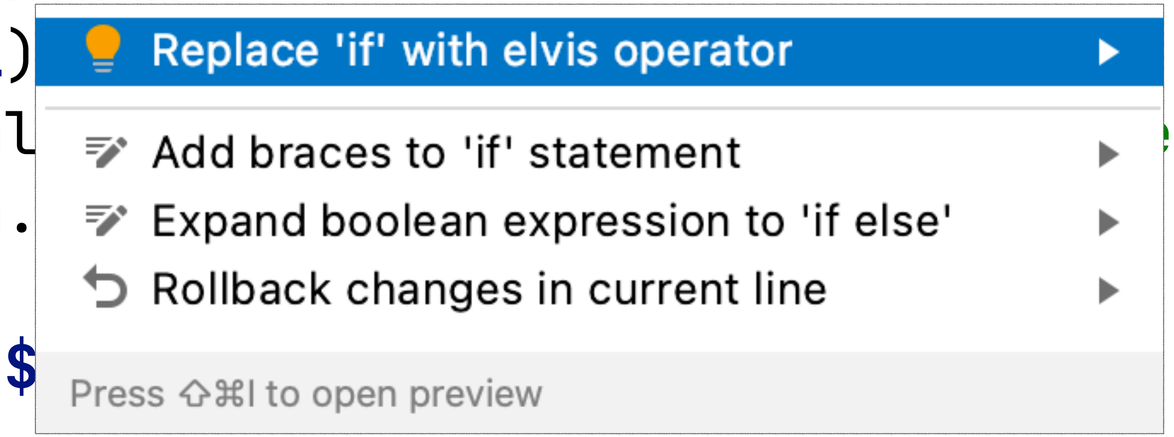
```
class Person(val name: String?, val age: Int?)

fun processPerson(person: Person) {
    val name = person.name
    if (name == null)
        throw IllegalArgumentException("Named required")
    val age = person.age
    if (age == null) return
    println("$name: $age")
}
```

Use elvis operator as **return** and **throw**

```
class Person(val name: String?, val age: Int?)

fun processPerson(person: Person) {
    val name = person.name
    if (name == null)
        throw IllegalArgumentException("Named required")
    val age = person.age
    if (age == null) return
    println("$name: $age")
}
```

Use elvis operator as **return** and **throw**

```
class Person(val name: String?, val age: Int?)

fun processPerson(person: Person) {
    val name = person.name
    if (name == null)
        throw Illegal                                    ed")
    val age = person.
    if (age == null)
    println("$name: $
}
```



Use elvis operator as **return** and **throw**

```
class Person(val name: String?, val age: Int?)

fun processPerson(person: Person) {
    val name = person.name ?:
        throw IllegalArgumentException("Named required")
    val age = person.age ?: return
    println("$name: $age")
}
```

Consider using safe cast for type checking

```
override fun equals(other: Any?) : Boolean {
    val command = other as Command
    return command.id == id
}
```

Consider using safe cast for type checking

```
override fun equals(other: Any?) : Boolean {
    val command = other as Command
    return command.id == id
}
```

```
override fun equals(other: Any?) : Boolean {
    return (other as? Command)?.id == id
}
```

Domain Specific Languages

buildString

```java
//Java
String name = "Joe";
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 5; i++) {
    sb.append("Hello, ");
    sb.append(name);
    sb.append("!\n");
}
System.out.println(sb);
```

```kotlin
//Kotlin
val name = "Joe"
val s = buildString {
    repeat(5) {
        append("Hello, ")
        append(name)
        appendLine("!")
    }
}
println(s)
```

kotlinx.html

```
System.out.appendHTML().html {
    body {
        div {
            a("http://kotlinlang.org") {
                target = ATarget.blank
                +"Main site"
            }
        }
    }
}
```

Ktor

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        routing {
            get("/html-dsl") {
                call.respondHtml {
                    body {
                        h1 { +"HTML" }
                        ul {
                            for (n in 1..10) {
                                li { +"$n" }
                            }
                        }
                    }
                }
            }
        }
    }.start(wait = true)
}
```

Ktor

Ktor's routing

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        routing {
            get("/html-dsl") {
                call.respondHtml {
                    body {
                        h1 { +"HTML" }
                        ul {
                            for (n in 1..10) {
                                li { +"$n" }
                            }
                        }
                    }
                }
            }
        }
    }.start(wait = true)
}
```

Ktor

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        routing {
            get("/html-dsl") {
                call.respondHtml {
                    body {
                        h1 { +"HTML" }
                        ul {
                            for (n in 1..10) {
                                li { +"$n" }
                            }
                        }
                    }
                }
            }
        }
    }.start(wait = true)
}
```

Ktor's routing

kotlinx.html

Lambda with receiver

`T.() → Unit`

```java
final ClientBuilder builder = new ClientBuilder();

builder.setFirstName("Anton");
builder.setLastName("Arhipov");

final TwitterBuilder twitterBuilder = new TwitterBuilder();
twitterBuilder.setHandle("@antonarhipov");
builder.setTwitter(twitterBuilder.build());

final CompanyBuilder companyBuilder = new CompanyBuilder();
companyBuilder.setName("JetBrains");
companyBuilder.setCity("Tallinn");
builder.setCompany(companyBuilder.build());


final Client client = builder.build();
System.out.println("Created client is: " + client);
```

```
val builder = ClientBuilder()

builder.firstName = "Anton"
builder.lastName = "Arhipov"

val twitterBuilder = TwitterBuilder()
twitterBuilder.handle = "@antonarhipov"
builder.twitter = twitterBuilder.build()

val companyBuilder = CompanyBuilder()
companyBuilder.name = "JetBrains"
companyBuilder.city = "Tallinn"
builder.company = companyBuilder.build()

val client = builder.build()
println("Created client is: $client")
```

```
val client = ClientBuilder().apply {
    firstName = "Anton"
    lastName = "Arhipov"

    twitter = TwitterBuilder().apply {
        handle = "@antonarhipov"
    }.build()

    company = CompanyBuilder().apply {
        name = "JetBrains"
        city = "Tallinn"
    }.build()
}.build()
println("Created client is: $client")
```

```
val client = ClientBuilder().apply {
    firstName = "Anton"
    lastName = "Arhipov"

    twitter = TwitterBuilder().apply {
        handle = "@antonarhipov"
    }.build()

    company = CompanyBuilder().apply {
        name = "JetBrains"
        city = "Tallinn"
    }.build()
}.build()
println("Created client is: $client")
```

```
val client = ClientBuilder().apply {
    firstName = "Anton"
    lastName = "Arhipov"

    twitter = TwitterBuilder().apply {
        handle = "@antonarhipov"
    }.build()

    company = CompanyBuilder().apply {
        name = "JetBrains"
        city = "Tallinn"
    }.build()
}.build()
println("Created client is: $client")
```

```kotlin
val client = ClientBuilder().apply {
    firstName = "Anton"
    lastName = "Arhipov"

    twitter = TwitterBuilder().apply {
        handle = "@antonarhipov"
    }.build()

    company = CompanyBuilder().apply {
        name = "JetBrains"
        city = "Tallinn"
    }.build()
}.build()
println("Created client is: $client")
```

```kotlin
fun client(c: ClientBuilder.() -> Unit): Client {
    val builder = ClientBuilder()
    c(builder)
    return builder.build()
}

fun ClientBuilder.company(block: CompanyBuilder.() -> Unit) {
    company = CompanyBuilder().apply(block).build()
}

fun ClientBuilder.twitter(block: TwitterBuilder.() -> Unit) {
    twitter = TwitterBuilder().apply(block).build()
}
```

```
val person = person {
    firstName = "Anton"
    lastName = "Arhipov"
    twitter {
       handle = "@antonarhipov"
    }
    company {
      name = "JetBrains"
      city = "Tallinn"
    }
}
println("Hello, $person!")
```

```
fun client(c: ClientBuilder.() → Unit): Client {
    val builder = ClientBuilder()
    c(builder)
    return builder.build()
}

fun ClientBuilder.company(block: CompanyBuilder.() → Unit) {
    company = CompanyBuilder().apply(block).build()
}

fun ClientBuilder.twitter(block: TwitterBuilder.() → Unit) {
    twitter = TwitterBuilder().apply(block).build()
}
```

Standard Library

Verify parameters using `require()`

```
class Person(
    val name: String?,
    val age: Int
)

fun processPerson(person: Person) {
    if (person.age < 18) {
        throw IllegalArgumentException("Adult required")
    }
}
```

Verify parameters using `require()`

```
class Person(
    val name: String?,
    val age: Int
)

fun processPerson(person: Person) {
    if (person.age < 18) {
        throw IllegalArgumentException("Adult required")
    }
}
```

```
fun processPerson(person: Person) {
    require(person.age ⩾ 18) { "Adult required" }
}
```

Select objects by type with `filterIsInstance`

```
fun findAllStrings(objects: List<Any>) =
    objects.filter { it is String }
```

Select objects by type with `filterIsInstance`

```
fun findAllStrings(objects: List<Any>) =
    objects.filter { it is String }
```

```
fun findAllStrings(objects: List<Any>) =
    objects.filterIsInstance<String>()
```

Select objects by type with `filterIsInstance`

```
fun findAllStrings(objects: List<Any>) : List<Any> =
    objects.filter { it is String }
```

```
fun findAllStrings(objects: List<Any>) : List<String> =
    objects.filterIsInstance<String>()
```

Apply operation to non-null elements `mapNotNull`

```
data class Result(
    val data: Any?,
    val error: String?
)

fun listErrors(results: List<Result>): List<String> =
    results.map { it.error }.filterNotNull()
```

```
fun listErrors(results: List<Result>): List<String> =
    results.mapNotNull { it.errorMessage }
```

compareBy compares by multiple keys

```
class Person(
    val name: String,
    val age: Int
)

fun sortPersons(persons: List<Person>) =
    persons.sortedWith(Comparator<Person> { person1, person2 →
        val rc = person1.name.compareTo(person2.name)
        if (rc ≠ 0)
            rc
        else
            person1.age - person2.age
    })
```

compareBy compares by multiple keys

```
class Person(
    val name: String,
    val age: Int
)

fun sortPersons(persons: List<Person>) =
    persons.sortedWith(Comparator<Person> { person1, person2 →
        val rc = person1.name.compareTo(person2.name)
        if (rc ≠ 0)
            rc
        else
            person1.age - person2.age
    })
```

```
fun sortPersons(persons: List<Person>) =
    persons.sortedWith(compareBy(Person::name, Person::age))
```

groupBy to group elements

```
class Request(
    val url: String,
    val remoteIP: String,
    val timestamp: Long
)

fun analyzeLog(log: List<Request>) {
    val map = mutableMapOf<String, MutableList<Request>>()
    for (request in log) {
        map.getOrPut(request.url) { mutableListOf() }
            .add(request)
    }
}
```

groupBy to group elements

```
class Request(
    val url: String,
    val remoteIP: String,
    val timestamp: Long
)

fun analyzeLog(log: List<Request>) {
    val map = mutableMapOf<String, MutableList<Request>>()
    for (request in log) {
        map.getOrPut(request.url) { mutableListOf() }
            .add(request)
    }
}
```

```
fun analyzeLog(log: List<Request>) {
    val map = log.groupBy(Request::url)
}
```

Use **coerceIn** to ensure numbers in range

```
fun updateProgress(value: Int) {
    val actualValue = when {
        value < 0 → 0
        value > 100 → 100
        else → value
    }
}
```

```
fun updateProgress(value: Int) {
    val actualValue = value.coerceIn(0, 100)
}
```

@antonarhipov

https://speakerdeck.com/antonarhipov

https://github.com/antonarhipov