Kotlin.amsterdam

The dangerous side of Kotlin

bol.com

Bart Enkelaar

Site Reliability Engineer

@BartEnkelaar

benkelaar@bol.com

Ahold Delhaize

bol.com

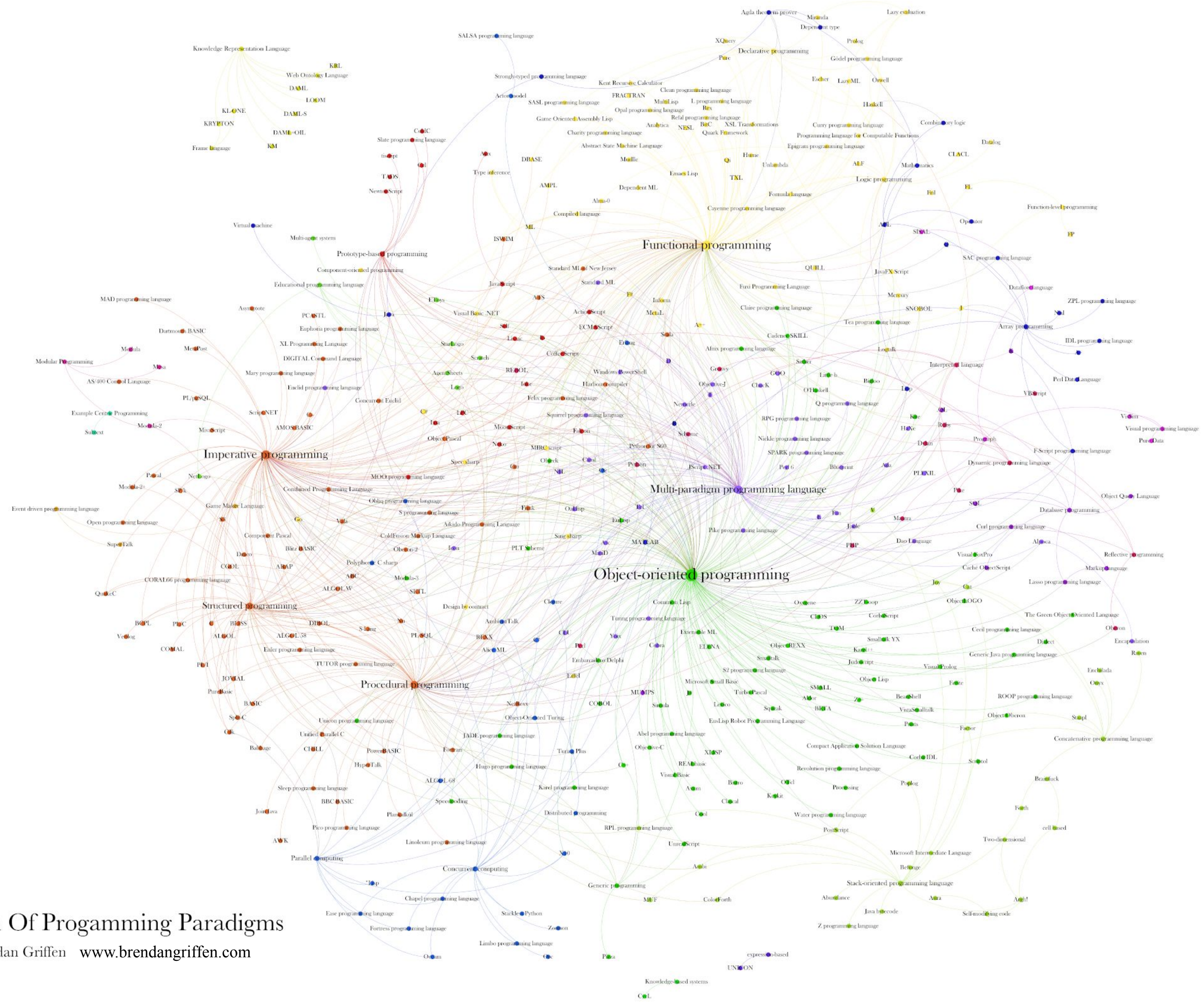

feestbeesten


Hallo
bol.com

Kotlin

DANGEROUS Kotlin

DANGEROUS Kotlin



The Graph Of Progamming Paradigms

By Brendan Griffen www.brendangriffen.com

- Fix our problem
- Easy to write
- Easy to read

I call it my billion-dollar mistake. It was the invention of the null reference in 1965.

- Sir Tony Hoare, FRS, FREng

```kotlin
data class Coffee(val amount: MilliLiters) {
    fun merge(that: Coffee) = this + that

    private operator fun plus(extra: Coffee) = Coffee( amount: amount + 
}

typealias MilliLiters = Double
typealias CoffeeMerge = (Coffee, Coffee) -> Coffee

private val merge: CoffeeMerge = Coffee::merge

fun Double.toOunces() = this / 29.574

inline fun <reified T> T?.getClassString() = T::class.java.toString()
```


I ♥ Whiskey

Extension functions

```kotlin
private fun String.asUrl() = URL( spec: this)
```

```kotlin
private fun String?.asUrl() = this?.let { URL(it) }
```

```kotlin
private fun String?.toUrl() =
    try { this?.let { URL(it) } }
    catch (e: MalformedURLException) { null }
```

```kotlin
data class PromotionReport(
        val eans: Set<String>?,
        val supplierId: String?,
        val attribute: List<PromotedAttribute>?,
        val promotedAttributesCount: Int?
): BeagleEntity

fun PromotionReport.getEans(): Set<String> = this.eans.orEmpty()

fun PromotionReport.getTraceId(): String? {
    return this.attribute?.firstOrNull { it.id == AttributeId.TRACE_ID.valu
}

fun PromotionReport.getTraceIdOrThrow() = this.getTraceId() ?: throw Illega
```

```kotlin
val value = BigDecimal( val: "3.5")

val moreValue = value * 4

operator fun BigDecimal.times(int: Int) = this * BigDecimal(in
```

```kotlin
inline class Money(private val value: BigDecimal) {
    constructor(value: Number) : this(BigDecimal(value.toString()))

    fun toCurrency(type: CurrencyType) = Currency( money: this, type)

    operator fun times(number: Number) = this * Money(number)
    operator fun times(money: Money) = this * money.value
    operator fun times(bigDecimal: BigDecimal) = Money( value: value * bigDecimal)
}
```

Great power...

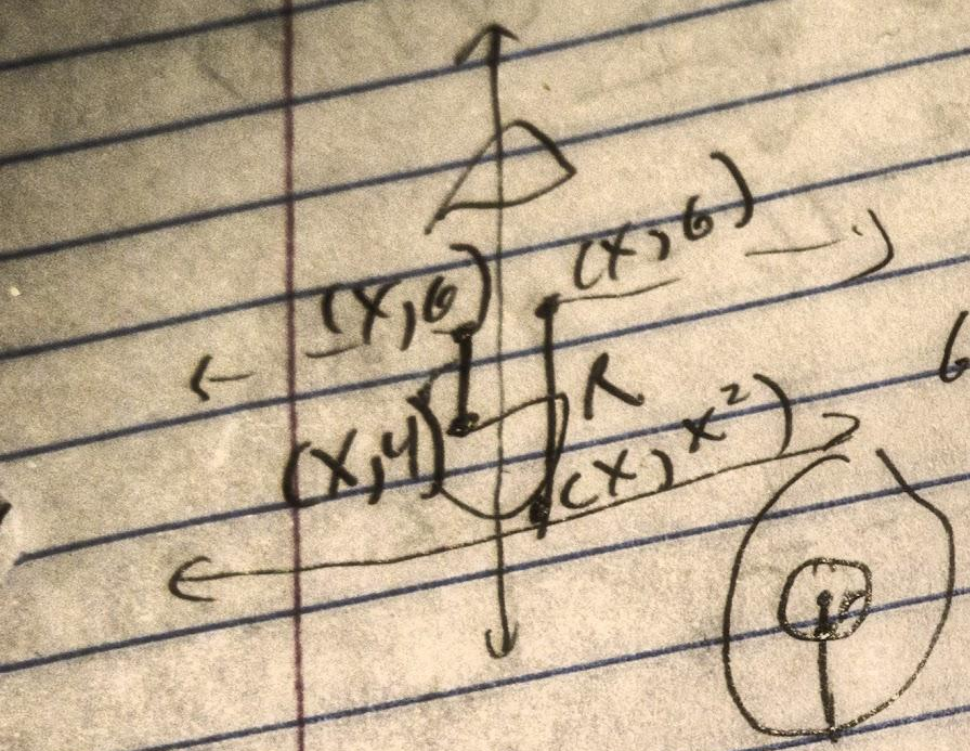
- Can introduce duplication
- Can break encapsulation
- Can interfere with OO principles

Operator overloading

```
inline class Money(private val value: BigDecimal) {
    constructor(value: Number) : this(BigDecimal(value.toString()))

    fun toCurrency(type: CurrencyType) = Currency( money: this, type)

    operator fun times(number: Number) = this * Money(number)
    operator fun times(money: Money) = this * money.value
    operator fun times(bigDecimal: BigDecimal) = Money( value: value * bigDecimal)
}
```



$$\int \pi (R^2 - r^2)\, dx$$

$6 - x^2 = R$

$$\pi \int (6-x^2)^2 - r^2\, dx$$

$$\pi \int (6-x^2)^2 - (2)^2\, dx$$

$$\pi \int_{-2}^{2} (6-x^2)^2 - 4\, dx$$

$$\pi \int_{-2}^{2} 36 - 12x^2 +$$

$6 - 4$

$2 = r$

limits

$x = 4$

$x = x^2$

4

StockTransactions
PriceTransactions

StockValueTransactions

$$x\Delta S * y\Delta P = xy\Delta V$$

```
val stockValueTransactions = stockTransactions * priceTransactions
```

Great but...

- Only works if concept aligns
- Can surprise readers
- Risky for its seductive elegance

Lambdas with receiver

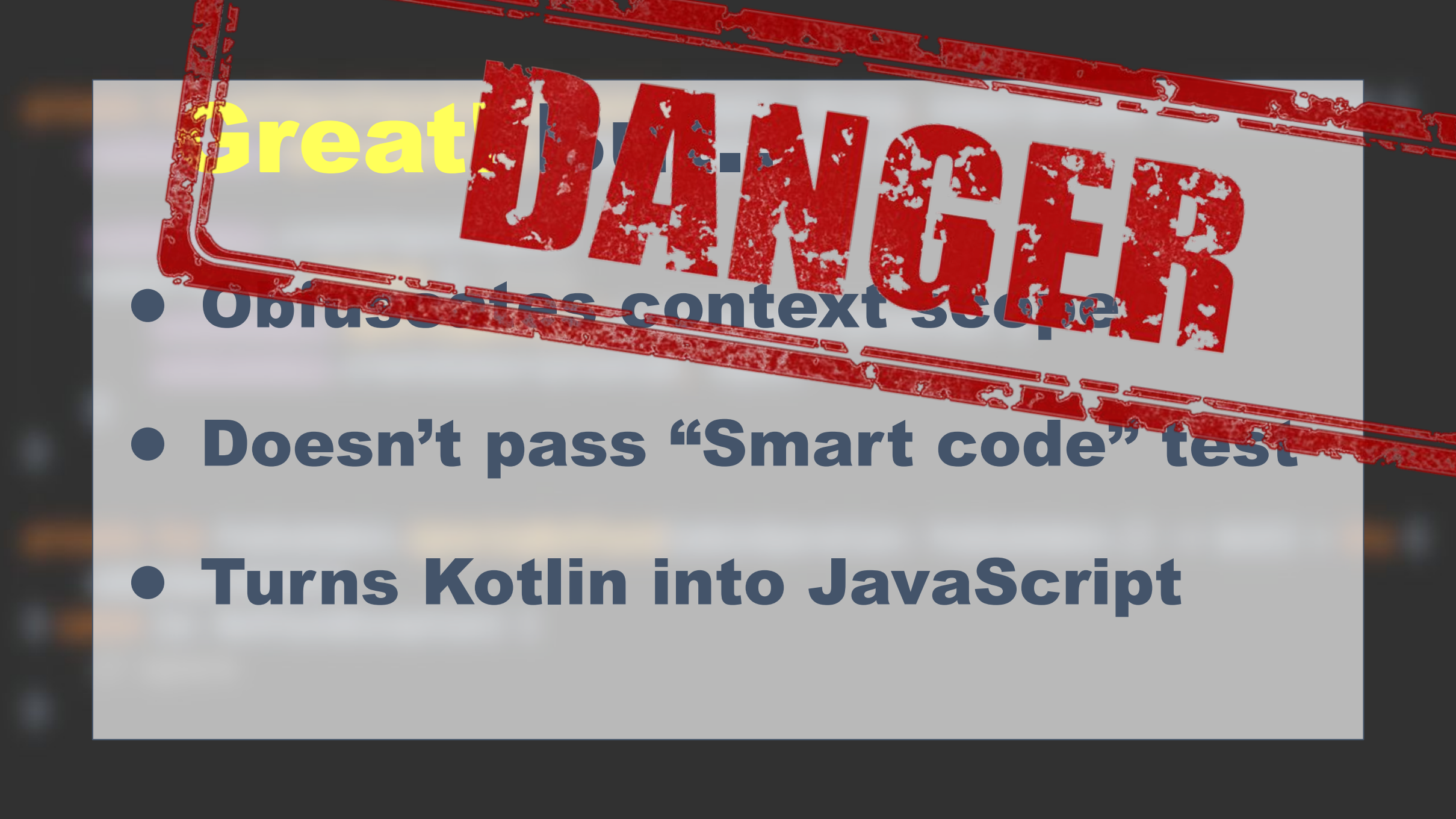
```
private fun configureTopicAndSubscription(topic: String, subscriptions: List<String>) {
    pubSubAdmin.ignoringNotFound { deleteTopic(topic) }

}

priva                                                              y {
    a
} catch (e: NotFoundException) {
    // ignore
}
```

Whenever I read JavaScript I want to shout "this is shit!", but I can never figure out what "this" refers to...

- A tweet I once read who's author I've regrettably forgotten


Great
DANGER
• Obfuscates context scope
• Doesn't pass "Smart code" test
• Turns Kotlin into JavaScript

Thanks

Attributions

- “Dangerous pirate” by Felix Lichtenfeld via Pixabay
- “Danger stamp” by Gerd Altmann via Pixabay
- Math picture by Robert Couse-Baker from Pxhere
- Graph of programming paradigms by Brendan Griffen

- All other images Creative Commons license

Bart Enkelaar

Site Reliability Engineer

@BartEnkelaar

benkelaar@bol.com

https://github.com/benkelaar/dangerous-kotlin

Bonus

```kotlin
fun <T, U, V> receivingStuff(instance: T, functionWithReceiver: T.(U) -> V) = { input: U -> instance.functionWithReceiver(input) }

// fun <T, U, V> receivingStuff(instance: T, functionWithReceiver: T.(U) -> V) = instance.functionWithReceiver
// or
// fun <T, U, V> receivingStuff(instance: T, functionWithReceiver: T.(U) -> V) = instance::functionWithReceiver
```

```kotlin
fun <T, U, V> curry(function: (T, U) -> V) = { t: T -> { u: U -> function(t, u) } }

fun <T, U, V, W> curry(function: (T, U, V) -> W) = { t: T -> { u: U -> { v: V -> function(t, u, v) } } }

fun <T, U, V> uncurry(function: (T) -> (U) -> V) = { t: T, u: U -> function(t)(u) }

fun <T, U, V, W> uncurry(function: (T) -> (U) -> (V) -> W) = { t: T, u: U, v: V -> function(t)(u)(v) }
```

```kotlin
fun <T, U, V> curriedReceivingStuff(instance: T, functionWithReceiver: T.(U) -> V) = curry(functionWithReceiver)(instance)
```

Main takeaway

Don’t blindly apply techniques - Think!

Outline

- Intro
  - Don’t blindly apply techniques - Think!
  - Kotlin is amazing
  - But be careful
- Quick recap of why Kotlin is awesome
  - All the power of the JVM
  - Null safety
  - Brevity
  - Modern language features
    - data classes
    - functional support
    - coroutines
    - type reification, tailrec
- Venture into the dark side - Features gone wrong
  - Extension functions
  - Operator overloading
  - Lambda with receivers
  - Deep optional chains
- Here be dragons, the real evil stuff
  - Java interop - Nullability, Function types
- Outro
  - All of these features can be used for good
  -
  - Don’t blindly apply techniques - Think!