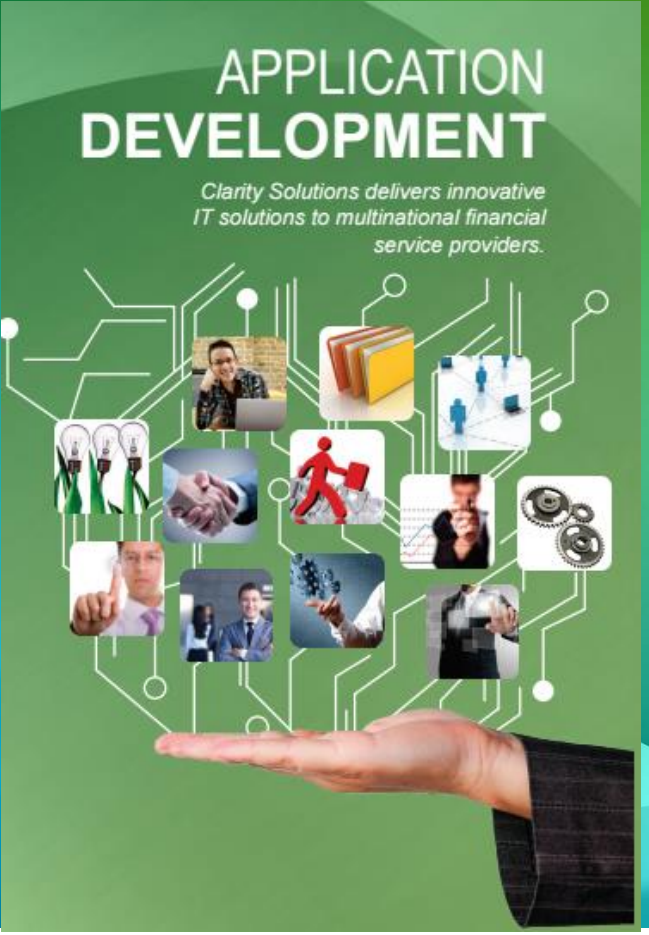


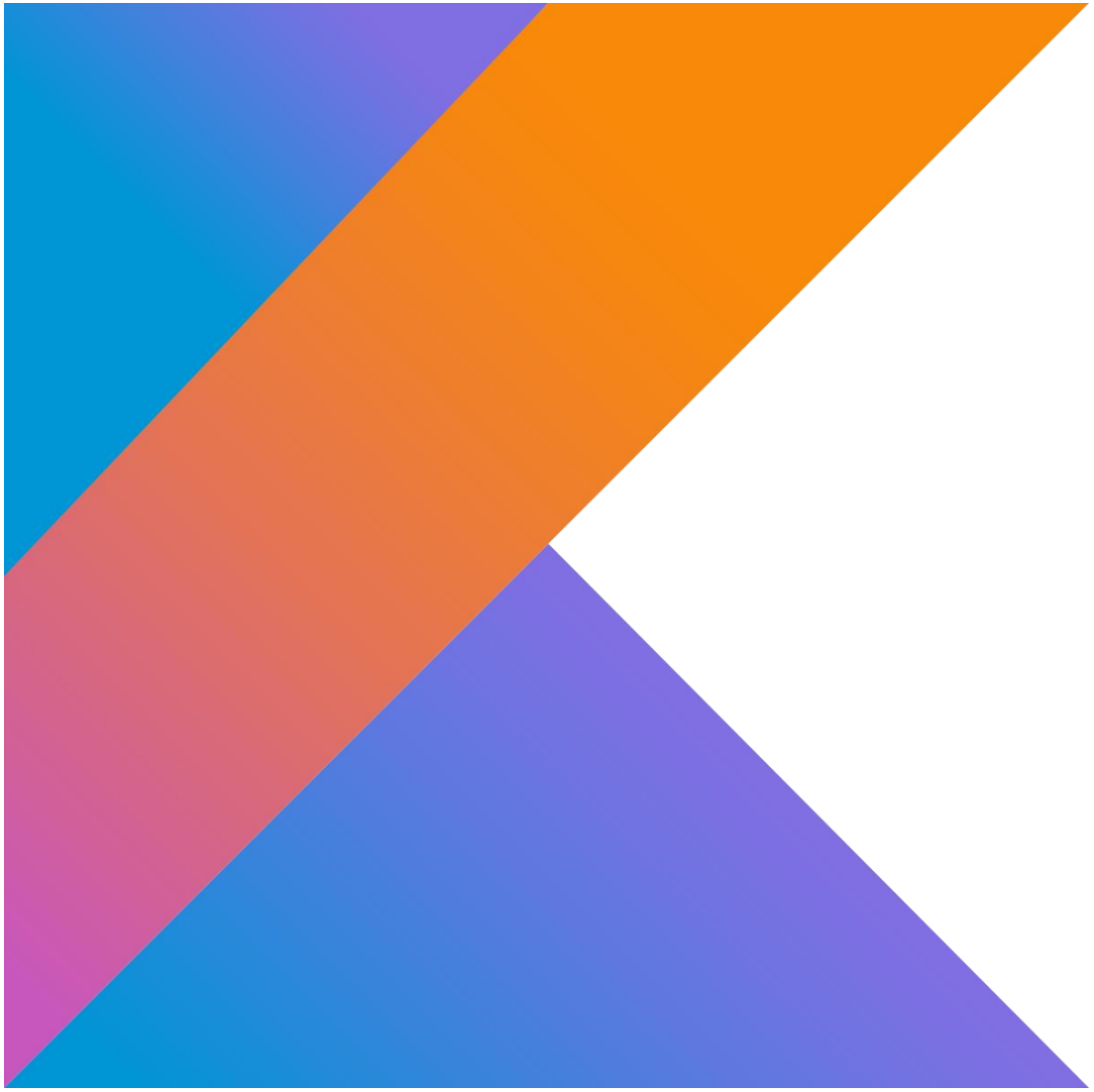
Kotlin

suIT Solutions
Turn IT on!

Zalavári Bálint

Kotlin

Első release: 2011.
V1.0 → 2016. február
JVM-en futó nyelv
JetBrains fejleszti



Kotlin

Tervezési alapelvek:

- Legyen jobb, mint a java.
- Legyen teljes mértékben átjárható a java-val

Kotlin

2018. május óta a hivatalosan preferált Android fejlesztési nyelv.
Lehet fordítani native kódra is.
Lehet fordítani iOS-re is.
Lehet fordítani JavaScript-re is.

variables

variable

```
var a: String = "initial"
var b: String
var c = 3
```

Value (readonly)

```
val d: Int = 1
val e = 3
```

```
var variable: String
    get() {
        return getAsString();
    }
    set(value) {
        setAsString(value.toInt())
    }
```

```
val value: String
    get() {
        return getAsString();
    }
```

Null safety

```
var neverNull: String = "This can't be null"

var nullable: String? = "You can keep a null here"

var inferredNonNull = "The compiler assumes non-null"
```

when

When vezérlő

```
when (obj) {
    1 -> println("One")
    "Hello" -> println("Greeting")
    is Long -> println("Long")
    !is String -> println("Not a string")
    else -> println("Unknown")
}
```

When kifejezés

```
val result = when (obj) {
    1 -> "one"
    "Hello" -> 1
    is Long -> false
    else -> 42
}
```

loops

For each

```
for (cake in cakes) {
    println("Yummy, it's a $cake cake!")
}
```

While

```
while (cakesEaten < 5) {
    eatACake()
    cakesEaten++
}
```

do-while

```
do {
    bakeACake()
    cakesBaked++
} while (cakesBaked < cakesEaten)
```

Range ciklusban

```
for (i in 0..3) {
    print(i)
}


for (i in 0 until 3) {
    print(i)
}


for (i in 2..8 step 2) {
    print(i)
}


for (i in 3 downTo 0) {
    print(i)
}
for (c in 'a'..'d') {
    print(c)
}


for (c in 'z' downTo 's' step 2) {
    print(c)
}
```

Range feltételben

```
if (x in 1..5) {
    print("x is in range from 1 to 5")
}


if (x !in 6..10) {
    print("x is not in range from 6 to 10")
}
```

Equality Checks

Két fajta egyenlőség vizsgálat van
== → .equals-t hívja meg
=== → referenciát vizsgál

```
val authors = setOf("Shakespeare", "Hemingway", "Twain")
val writers = setOf("Twain", "Shakespeare", "Hemingway")


println(authors == writers)
println(authors === writers)
```

Conditional Expression

Nincs Elvis operátor, helyette az if-et lehet használni kifejezésként.

```
val max = if (a > b) a else b
```

Companion Objects

Nincsenek static function-ök/variablek.
Helyette van companion object.

```
class BigBen {
    companion object Bonger {
        fun getBongs(nTimes: Int) {
            for (i in 1..nTimes) {
                print("BONG ")
            }
        }
    }
}


fun main() {
    BigBen.getBongs(12)
}
```

Vagy package level function/variable.

```
package hu.mik.prog5.kotlindemo

fun getBongs(nTimes: Int) {
    for (i in 1..nTimes) {
        print(bongSound)
    }
}


const val bongSound: String = "Bong "
```

Lambda

Function paraméterként

```
fun calculate(x: Int, y: Int, operation: (Int, Int) -> Int): Int {
    return operation(x, y)
}


fun sum(x: Int, y: Int) = x + y


fun main() {
    val sumResult = calculate(4, 5, ::sum)
    val mulResult = calculate(4, 5) { a, b -> a * b }
    println("sumResult $sumResult, mulResult $mulResult")
}
```

Function visszatérési értékként

```
fun operation(): (Int) -> Int {
    return ::square
}


fun square(x: Int) = x * x


fun main() {
    val func = operation()
    println(func(2))
}
```

Lambda

Használata:

```
val upperCase1: (String) -> String = { str: String -> str.toUpperCase() }

val upperCase2: (String) -> String = { str -> str.toUpperCase() }

val upperCase3 = { str: String -> str.toUpperCase() }

// val upperCase4 = { str -> str.toUpperCase() }

val upperCase5: (String) -> String = { it.toUpperCase() }

val upperCase6: (String) -> String = String::toUpperCase
```

let

Csak akkor hajtja végre, ha az objektum, amin meghívtuk nem null. Visszaadja a benne lévő utolsó kifejezés értékét. `it`-tel lehet meghivatkozni benne az objektumot, amin meghívtuk.

```
val empty = "test".let {
    customPrint(it)
    it.isEmpty()
}
println(" is empty: $empty")
```

```
str?.let {
    print("\t")
    customPrint(it)
    println()
}
```

run

Ugyanaz, mint a let, csak itt `this`-szel lehet meghivatkozni az objektumot. Szebb, ha metódust akarunk hívni az objektumon.

```
str?.run {
    println("\tis empty? " + isEmpty())
    println("\tlength = $length")
    length
}
```

with

Ne kelljen egy hivatkozást sokszor kiírni és ne is kelljen változóba kirakni.

```
with(configuration) {
    println("$host:$port")
}


// instead of:
println("${configuration.host}:${configuration.port}")
```

apply

Végrehajtja az utasításokat, majd visszaadja az objektumot magát.

```
val jake = Person()
val stringDescription = jake.apply {
    name = "Jake"
    age = 30
    about = "Android developer"
}.toString()
```

also

Ugyanaz, mint az apply, csak itt `it`-tel hivatkozunk.

```
val jake = Person("Jake", 30, "Android developer")
        .also {
            writeCreationLog(it)
        }
```

Smart cast

Smart cast non-nullable-re

Rövidre zárás

Smart cast non-nullable-re már a feltételben is

Smart cast leszármazottra

```
val date: ChronoLocalDate? = LocalDate.now()


if (date != null) {
    println(date.isLeapYear)
}


if (date != null && date.isLeapYear) {
    println("It's a leap year!")
}


if (date == null || !date.isLeapYear) {
    println("There's no Feb 29 this year...")
}


if (date is LocalDate) {
    val month = date.monthValue
    println(month)
}
```

És még sok más

- Delegation
- Destructuring Declarations
- Corutines
- ...

És jöjjön a várva várt gyakorlat…

Köszönöm a figyelmet!

facebook.com/suitsolutions.eu

linkedin.com/company/suit-solutions-ltd

suitsolutions.eu