

KOTLIN

FROM A JAVA DEVELOPER'S PERSPECTIVE

WHAT IS KOTLIN?

- cross platform
- statically typed
- more concise than Java
- compiles to JVM byte code

WHAT IS KOTLIN?

Growth in contributors	
1 Kotlin	2.6x
2 HCL	2.2x
3 TypeScript	1.9x
4 PowerShell	1.7x
5 Rust	1.7x
6 CMake	1.6x
7 Go	1.5x
8 Python	1.5x
9 Groovy	1.4x
10 SQLPL	1.4x

-- Fastest growing languages on GitHub in 2018

JAVA INTEROPERABILITY

“ Kotlin is designed with Java Interoperability in mind. Existing Java code can be called from Kotlin in a natural way, and Kotlin code can be used from Java rather smoothly as well. ”

kotlinlang.org

JAVA INTEROPERABILITY

```
import java.util.Calendar

fun calendarDemo() {
    val calendar = Calendar.getInstance()
    if (calendar.firstDayOfWeek == Calendar.SUNDAY) {
        calendar.firstDayOfWeek = Calendar.MONDAY
    }
}
```

LESS VERBOSE

“ Kotlin's modern language features allow you to focus on expressing your ideas and write less boilerplate code. Less code written also means less code to test and maintain. ”

developer.android.com

LESS VERBOSE

Java

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

Kotlin

```
fun main() {  
    println("Hello World")  
}
```

SEMICOLONS ARE OPTIONAL

```
println("Hello World");
```

is equivalent to

```
println("Hello World")
```


Unit RETURN TYPE CAN BE OMITTED

```
fun main(): Unit {  
    println("Hello World")  
}
```

is equivalent to

```
fun main() {  
    println("Hello World")  
}
```

TYPE INFERENCE

```
val a = 3  
val b: Int  
b = 5
```

is equivalent to

```
val a: Int = 3 // type Int gets inferred  
val b: Int  
b = 5
```

VISIBILITY MODIFIERS

- `private`
- `protected`
- `internal`
- `public` (default)

```
private val a = 3  
val b = 5 // public by default
```

ACCESSING PROPERTIES

Java

```
String name = somePerson.getName();
```

Kotlin

```
val name = somePerson.name
```

CREATING NEW INSTANCES

No `new` keyword

```
Person somePerson = new Person("Dennis");
```

Kotlin

```
val somePerson = Person("Dennis")
```

NAMED ARGUMENTS

```
class User(val username: String, val password: String)

val admin = User(username = "admin", password = "12345")

val user = User(password = "54321", username = "user")
```

DEFAULT ARGUMENTS

```
class User(  
    val username: String,  
    val password: String = "not-so-secure-password"  
)  
  
val admin = User(username = "admin")
```

SINGLE-EXPRESSION FUNCTIONS

```
fun theAnswer() = 42
```

is equivalent to

```
fun theAnswer(): Int {  
    return 42  
}
```


var AND val

```
var a = "mutable String"  
val b = "immutable String"  
  
a = "new value"  
b = "new value" // compilation error
```

NULL SAFETY

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

Tony Hoare

NULL SAFETY

```
var a: String = "abc"  
a = null // compilation error  
  
var b: String? = "abc"  
b = null // ok
```

SAFE CALL OPERATOR - ? .

```
val a: String = "abc"  
a.length  
  
val b: String? = "abc"  
  
b.length // compilation error  
  
b?.length
```

ELVIS OPERATOR - ? :

If the expression to the left of ? : is not null, the elvis operator returns it, otherwise it returns the expression to the right.

```
val b: String? = "abc"  
val l = b?.length ?: -1
```

TEMPLATE STRINGS

```
val name = "Dennis"  
println("My name is $name")
```

```
My name is Dennis
```

DATA CLASSES

- `equals()`
- `hashCode()`
- `toString()`
- `copy()`
- **Destructuring Declarations**

DATA CLASSES

```
data class User(  
    val username: String,  
    val password: String  
)  
  
val admin = User("admin", "12345")
```


DESTRUCTURING DECLARATIONS

```
data class User(  
    val username: String,  
    val password: String  
)  
  
val (username, password) = User("admin", "12345")
```

```
for ((key, value) in map) {  
    println("Key: $key, value: $value")  
}
```

SMART CASTS

`is` check

```
if (x is String) {  
    println(x.length) // x is automatically cast to String  
}
```

negative `is` check

```
if (x !is String) return  
println(x.length) // x is automatically cast to String
```

SMART CASTS

when-expressions

```
when (x) {  
    is Int -> println(x + 1)  
    is String -> println(x.length + 1)  
    is IntArray -> println(x.sum())  
}
```

COLLECTIONS

Immutable

```
val x = listOf(a, b, c)
val y = setOf(a, b, c)
val z = mapOf(
    Pair("a", a),
    Pair("b", b),
    Pair("c", c)
)
```

COLLECTIONS

Mutable

```
val x = mutableListOf(a, b, c)
val y = mutableSetOf(a, b, c)
val z = mutableMapOf(
    Pair("a", a),
    Pair("b", b),
    Pair("c", c)
)
```

RANGES

ascending

```
for (i in 1..4) {  
    println("Current Value is $i")  
}
```

descending

```
for (i in 4 downTo 1) {  
    println("Current Value is $i")  
}
```

RANGES WITH STEP

ascending

```
for(i in 1..4 step 2) {  
    println("Current Value is $i")  
}
```

descending

```
for(i in 4 downTo 1 step 2) {  
    println("Current Value is $i")  
}
```

EXTENSION FUNCTIONS

“Provide the ability to extend a class with new functionality without having to inherit from the class or use any type of design pattern such as Decorator.”

kotlinlang.org

UTIL CLASSES IN JAVA

```
public class CollectionUtil {  
    public static void swap(  
        List<Integer> list,  
        int index1,  
        int index2  
    ) {  
        Integer tmp = list.get(index1);  
        list.set(index1, list.get(index2));  
        list.set(index2, tmp);  
    }  
}
```

```
List<Integer> list = Arrays.asList(1,2,3);  
CollectionUtil.swap(list, 0, 2);
```

EXTENSION FUNCTIONS

Replacement of Java Util function

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {  
    val tmp = this[index1]  
    this[index1] = this[index2]  
    this[index2] = tmp  
}
```

```
val list = mutableListOf(1, 2, 3)  
list.swap(0, 2)
```

COROUTINES

lightweight threads (no context switching)

```
suspend fun main() = coroutineScope {  
    for(i in 0 until 10) {  
        launch {  
            delay(1000L - i * 10)  
            print("$i ")  
        }  
    }  
}
```

9 8 7 6 5 4 3 2 1 0

MORE AWESOME FEATURES

- Kotlin Native
- Scope functions
- Infix operator
- Ability to create DSLs
- Contracts
- ...

FURTHER INFORMATION

- <https://kotlinlang.org/docs/reference/>
- <https://play.kotlinlang.org/koans/overview>
- <https://kotlinlang.org/docs/books.html>
- Medium - Roman Elizarov
- YouTube - KotlinConf 2018

THANK YOU!

Questions?

Follow me on <https://github.com/d3ns0n>