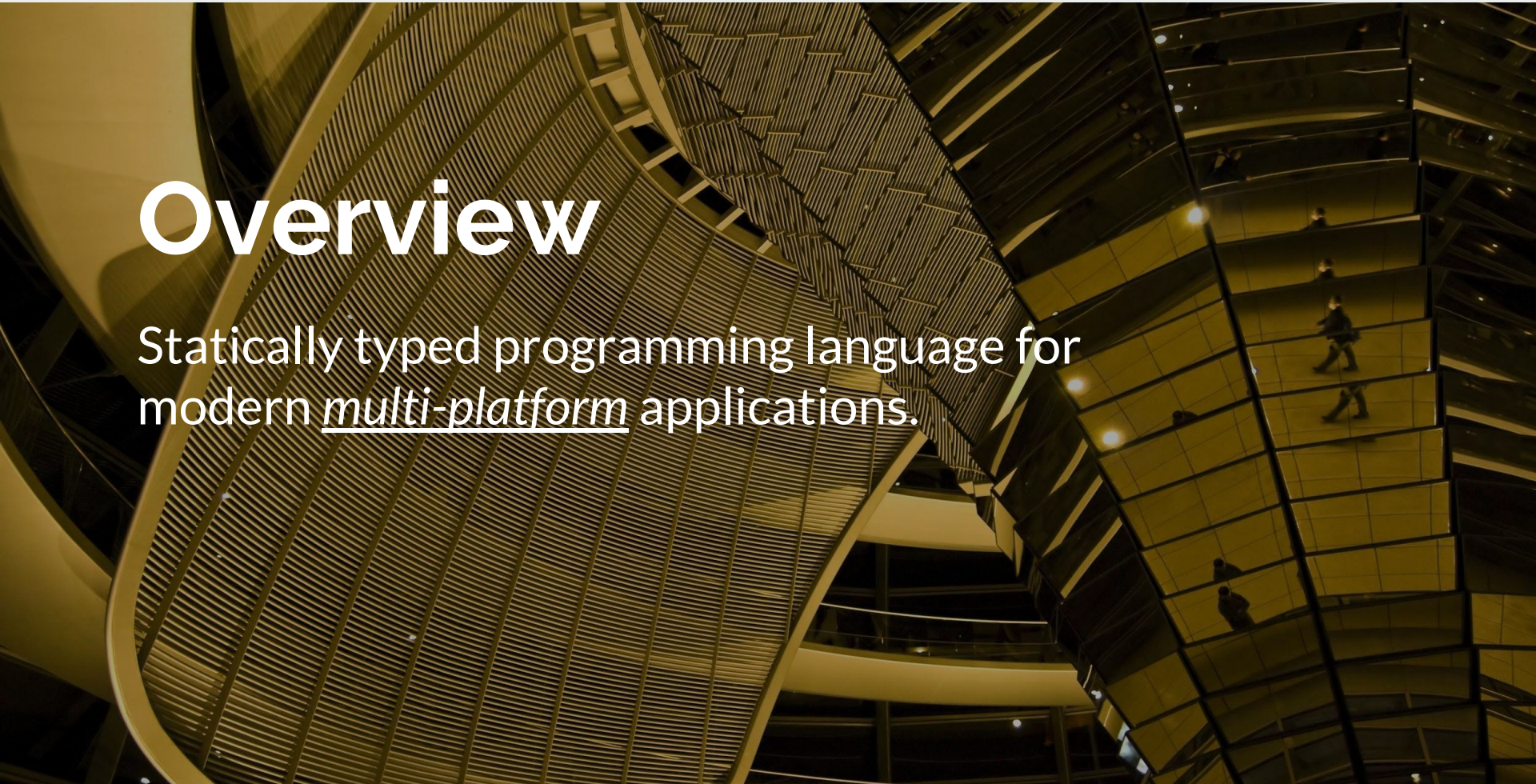
Kotlin for Android

Modern. Expressive. Safe.

Han Yin @ Jun, 2018

TOC

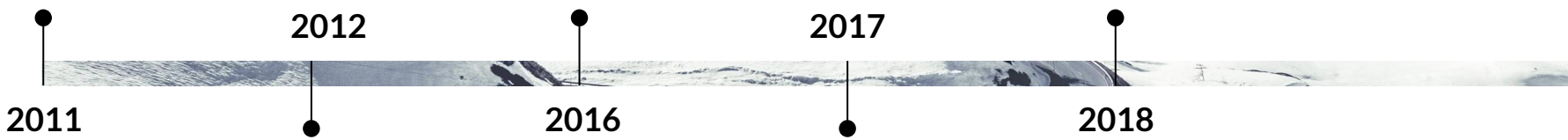
- Overview, Major advantages, Case study
- Language features by examples
- Migration advices, Tutorials, Documentations & Samples



Overview

Statically typed programming language for modern *multi-platform* applications.

Overview: a brief history



2011

JetBrains* unveiled *Project Kotlin

A new language for the *JVM*, which had been under development for a year.

2012

Open sourced under the *Apache 2* license

JetBrains hopes that the new language will drive *IntelliJ IDEA* sales.

2016

***Kotlin* v1.0 was released on February 15**

This is considered to be the first officially stable release and *JetBrains* has committed to long-term backwards compatibility starting with this version.

2017

Google* announced first-class support for *Kotlin* on *Android

Android ❤ *Kotlin*

2018

***Kotlin* v1.2 was released on November 28**

Sharing Code between *JVM* and *Javascript* platforms feature was newly added to this release.

Major advantages @ Language

1 **Concise**

Drastically reduce the amount of boilerplate code.

2 **Safe**

Avoid entire classes of errors such as null pointer exceptions.

3 **Inter-operable**

Leverage existing libraries for the ***JVM***, ***Android***, and the *browser*.

4 **Tool-friendly**

Choose any ***Java*** IDE or build from the command line.

Major advantages @ Android

1 ## Compatibility

Fully compatible with ***JDK 6*** and ***Android*** build system.

2 ## Performance

A ***Kotlin*** app runs as fast as an equivalent ***Java*** one, due to similar bytecode structure.

3 ## Footprint

Kotlin has a very compact runtime library, which can be further reduced through the use of ***ProGuard***.

4 ## Compilation Time

Incremental builds are usually as fast or faster than with ***Java***, due to efficient incremental compilation.

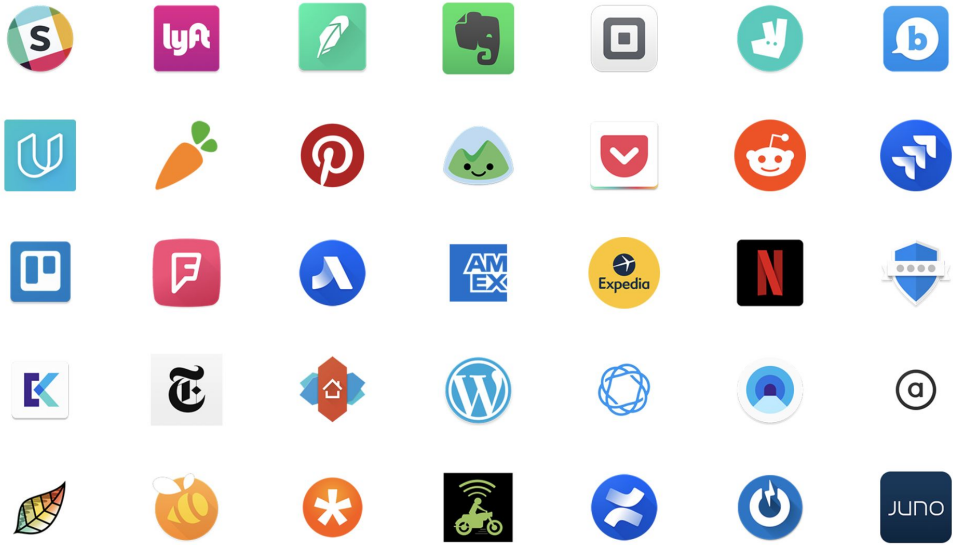
Case study

1. ## Pinterest

   Successfully introduced Kotlin into their app, used by 150M people every month.

2. ## Basecamp

   100% Kotlin code. Reports a huge difference in developer happiness, great improvements in work quality and speed.



Language features

Null-safety
Smart casts
Type inference
Type projections
Declaration-site variance
Separate interfaces for read-only and mutable collections

Range expressions
String templates
Properties
Primary constructors
Data classes
First-class delegation
Companion objects
Singletons

Operator overloading
Extension functions
Inline functions
Lambda expressions

* *Coroutines*

Java vs. *Kotlin*

```java
View view = getLayoutInflater().inflate(layoutResource, group);
view.setVisibility(View.GONE)
System.out.println("View " + view + " has visibility " + view.getVisibility() + ".");
```

```kotlin
val view = layoutInflater.inflate(layoutResource, group)
view.visibility = View.GONE
println("View $view has visibility ${view.visibility}.")
```

100% interoperable with *Java*

- For a ***Java*** developer, getting started with ***Kotlin*** is very easy. The automated *Java to Kotlin converter* included in the ***Kotlin*** plugin helps with the first steps.
- *Kotlin Koans* offer a guide through the key features of the language with a series of interactive exercises.

```kotlin
// Calling Java code from Kotlin
class KotlinClass {
    fun kotlinDoSomething() {
        val javaClass = JavaClass()
        javaClass.javaDoSomething()
        println(JavaClass().prop)
    }
}

==================================

// Calling Kotlin code from Java
public class JavaClass {
    public String getProp() { return "Hello"; }
    public void javaDoSomething() {
        new KotlinClass().kotlinDoSomething();
    }
}
```

Null-safety, Mutable & Read-only variable:

```
1  var str: String
2  println(str) // non-null variables must be initialized.
3  str = null   // compile error: null cannot be a value of a non-null type
4
5  val str2: String? = null    // nullable type
6  println(str2.length())      // compile error
7
8  fun parseInt(str: String): Int? {...}
```

Lateinit

```
1  class MyActivity : AppCompatActivity() {
2      // non-null, but not initalized
3      lateinit var recyclerView: RecyclerView
4
5      override fun onCreate(savedInstanceState: Bundle?) {
6          // …
7          // initialized here
8          recyclerView = findViewById(R.id.recycler_view)
9      }
10 }
```

Elvis operator

```
1  // Java “null-erplate”s
2  if (person.age == null) {
3      return -1;
4  }
5  int age = person.age;
6  String name = person.name != null ? person.name : "unknown";
7
8  // With Elvis
9  val age = person.age ?: return -1
10 val name: String = person.name ?: "unknown"
```

Conditional expressions & When expressions

```
1  fun maxOf(a: Int, b: Int) = if (a > b) a else b
2
3  fun describe(value: Number?): String =
4      when (value) {
5          null, 0      -> "None"
6          in 1..5      -> "Tiny"
7          2018         ->  "This year"
8          is Long      -> "Is a Long"
9          !is Int      -> "Not an Integer"
10         else         -> "Unknown"
11     }
```

Smart casts

```
1  if (context is Activity)
2      context.finish()    // no need for casting
```

For loops & Range expressions

```
1  for(i in 1..100) {…}
2  for (x in 1..10 step 2) {…}
3
4  for(i in 100 downTo 1){…}
5  for (x in 9 downTo 0 step 3) {…}
6
7  val array = arrayOf(“a”, “b”, “x”)  // "arrayOf"
8  for(i in 1 until array.size step 2 ){…}
```

Easy bundle: *listOf*, *arrayOf*, *mapOf*, *setOf*, etc.

```
1  val bundle = bundleOf(
2      "KEY_INT " to 1,
3      "KEY_LONG" to 2L,
4      "KEY_BOOLEAN" to true,
5      "KEY_NULL" to null
6      "KEY_ARRAY" to arrayOf(1, 2)
7  )
```

```
// "arrayOf"
```

Break, Continue & Return labels

```
xyz@ for (i in 1..100) {
      for (j in 1..100) {
          if (……) break@xyz
      }
}
```

```
fun foo() {
    listOf(1, 2, 3, 4, 5).forEach {
        if (it == 3)
            // local return to the caller of the lambda,
            // i.e. the forEach loop
            return@forEach
        print(it)
    }
}
// Outputs: 1245
```

Named arguments & Default arguments

```
1  fun format(str: String,
2      normalizeCase: Boolean = true,
3      upperCaseFirstLetter: Boolean = true,
4      divideByCamelHumps: Boolean = false,
5      wordSeparator: Char = ' ') {
6      ……
7  }
8
9  // Call function with named arguments.
10 format(str, normalizeCase = true, upperCaseFirstLetter = true)
```

Data classes, Parcelable

```
1  // a POJO with getters, setters, equals(), hashCode(), toString(), and copy()
2  @Parcelize
3  data class User(val name: String, val email: String, val company: String)
```

Properties & Fields

- To use a property, simply refer to it by name, as if it were a field in ***Java***.
- Fields cannot be declared directly in ***Kotlin*** classes.
- However, when a property needs a backing field, ***Kotlin*** provides it automatically.

```
class User {
    // properties
    val id: String = ""                 // immutable. just getter

    var firstname: String = ""          // default getter and setter

    var surname: String = ""            // custom getter, default setter
        get() = field.toUpperCase()     // custom getter declaration

    var fullname: String = ""
        get() {                         // custom getter declaration
            return when (field.isNotEmpty()) {
                true -> field
                false -> "${firstname} ${surname}"
            }
        }

    var email: String = ""              // default getter, custom setter
        set(value) {                    // custom setter declaration
            // "value" = name of the setter parameter
            // "field" = property's backing field; generated
            if(isEmailValid(value)) field = value
        }
}
```

String templates

```
1  var a = 1
2  // simple name in template:
3  val s1 = "a is $a"
4
5  a = 2
6  // arbitrary expression in template:
7  val s2 = "${s1.replace("is", "was")}, but now is $a"
8  // outputs: a was 1, but now is 2
9
10 val items = listOf("apple", "banana", "kiwifruit")
11 for (index in items.indices) {
12     println("item at $index is ${items[index]}")
13 }
```

```
return when (field.isNotEmpty()) {
    true -> field
    false -> "${firstname} ${surname}"
}
```

Singleton

```
1  object SimpleSingleton {
2      const val answer = 42;
3      fun greet(name: String) = "Hello, $name!"
4  }
5
6  val sum = 33 + SimpleSingleton.42    // 75
7  SimpleSingleton.greet("world")       // "Hello, world!"
```

Sealed class

```
1  sealed class Expr
2  data class Const(val number: Double)        : Expr()
3  data class Sum(val e1: Expr, val e2: Expr)  : Expr()
4  object NotANumber                           : Expr()
5
6  fun eval(expr: Expr): Double = when(expr) {
7      is Const -> expr.number
8      is Sum -> eval(expr.e1) + eval(expr.e2)
9      NotANumber -> Double.NaN
10     // No need for an "else" here
11 }
```

Destructuring declarations

```
1  data class ApiResponse(val resultCode: Int, val data: JSONObject)
2  fun requestService(): ApiResponse {
3      // Stub
4      return ApiResponse(0, JSONObject())
5  }
6
7  val (result, data) = requestService()
```

Destructuring for loops

```
1  // iterating over an array with destructuring
2  for( (index, element) in array.withIndex()) {…}
3
4  // iterating over a map with destructuring
5  val map = mapOf(1 to "one", 2 to "two")     // "mapOf"
6  for( (key, value) in map){…}
```

Extend functionality without inheritance

```java
// com/example/util/DateUtils.java
static boolean isTuesday(Date date) {
    return date.getDay() == 2;
}

// com/example/TuesdayActivity.java
boolean isTue = DateUtils.isTuesday(date);
```

```kotlin
// com/example/util/DateExtensions.kt
fun Date:isTuesday() = day == 2

// com/example/TuesdayActivity.kt
import com.example.util.isTuesday
val isTue = date.isTuesday()
```

Operator overloading

```
1  data class Point(val x: Int, val y: Int)
2  operator fun Point.unaryMinus() = Point(-x, -y)
3
4  val point = Point(15, 20)
5  println(-point)     // outputs: "-15, -20"
```

```
1  /** Adds a span to the entire text. */
2  inline operator fun Spannable.plusAssign(span: Any) =
3  setSpan(span, 0, length, SPAN_INCLUSIVE_EXCLUSIVE)
4
5  // Use it like this
6  val spannable = “Eureka!!!!”.toSpannable()
7  spannable += StyleSpan(BOLD) // Make the text bold with +=
8  spannable += UnderlineSpan() // Make the text underline with +=
```

Type aliases

```
1  typealias NodeSet = Set<Node>
2  typealias MyHandler = (Int, String, Any) -> Unit
```

Lazy Evaluation

```
1 val preference: String by lazy {
2   sharedPreferences.getString(PREFERENCE_KEY)
3 }
```

Higher-Order Functions & Lambdas

```
// funtions are first-class

// A lambda expression
{ a, b -> a + b }

// an anonymous function
fun(s: String): Int { return s.toIntOrNull() ?: 0 }

// a Higher-Order Function is a function that takes functions as parameters, or returns a function
fun <T, R> Collection<T>.fold(
        initial: R,
        combine: (acc: R, nextElement: T) -> R
        ): R {
    var accumulator: R = initial
    for (element: T in this) {
        accumulator = combine(accumulator, element)
    }
    return accumulator
}
```

```
// A lambda expression or anonymous function can access its closure
// (i.e. the variables declared in the outer scope).
var sum = 0
ints.filter { it > 0 }.forEach {
    sum += it   // Unlike Java, the variables captured in the closure can be modified
}
print(sum)
```

```java
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v){
        doSomething();
    }
});

button.setOnClickListener( { doSomething() } )  // type: (View!) -> Unit

// If the last parameter of a function accepts a function, a lambda expression that is passed as the
// corresponding argument can be placed outside the parentheses
button.setOnClickListener() { doSomething() }

// If the lambda is the only argument to that call, the parentheses can be omitted entirely
button.setOnClickListener { doSomething() }

// If the compiler can figure the signature out itself, it is allowed not to declare the only parameter and omit ->.
// The parameter will be implicitly declared under the name it
ints.filter { it -> it > 0 }
ints.filter { it > 0 }      // type: (Int) -> Boolean

// If the lambda parameter is unused, you can place an underscore instead of its name:
map.forEach { _, value -> println("$value!") }
```

Filter

```java
int[] all = {1, 2, 3, 4, 5, 6, 7, 8, 9};
List<Integer> filters = new ArrayList<>();
for (int a : all) {
    if (a % 3 == 0) {
        filters.add(a);
    }
}
```

```kotlin
val all = arrayOf(1, 2, 3, 4, 5, 6, 7, 8, 9)
val filters = all.filter { it % 3 == 0 }
```

Map

```
1  String[] names = {"Ali", "Gus", "Said", "Jeff"};
2  int[] namesLength = new int[names.length];
3  for (int i = 0; i < names.length ; i ++) {
4      namesLength[i] = names[i].length();
5  }
```

```
1  val names = arrayOf("Ali", "Gus", "Said", "Jeff");
2  val namesLength = names.map { it.length }
```

Reduce

```
1  String[] texts = {"Ali", "Gus", "Said", "Jeff"};
2  StringBuffer sb = new StringBuffer();
3  for (int i = 0; i < texts.length ; i ++) {
4      sb.append(texts[i].substring(0, 1));
5  }
6  String result = sb.toString();
```

```
1  val texts = arrayOf("Ali", "Gus", "Said", "Jeff")
2  val result = texts.map { it.substring(0,1) }.reduce { r, s -> "$r$s"}
```

Performant custom control structures

```
1  employees
2      .filter { it.group.startsWith("Product") }
3      .sortedBy { it.lastName }
4      .map { "${it.firstName}: ${it.comment.trim()}"}
```

Standard functions: *run, with, let, also, apply*

```
1  // Provide inner scope
2  var mood = "I am sad"
3  run {
4      val mood = "I am happy"
5      println(mood)           // I am happy
6  }
7  println(mood)               // I am sad
8
9  // "Run" returns the last object within the scope
10 run {
11     if (firstTimeView) introView else normalView
12 }.show()
13
14 // For this Java sample
15 if (webview.settings != null) {
16     webview.settings.javaScriptEnabled = true;
17     webview.settings.databaseEnabled = true;
18 }
```

```
20 // "Run" serves as an Extension Function
21 webview.settings?.run {
22     javaScriptEnabled = true
23     databaseEnabled = true
24 }
25
26 // "With"
27 with(webview.settings) {
28     databaseEnabled = true
29     javaScriptEnabled = true
30 }
31
32 // "Also"
33 val list = (1..20).toList()
34             .filter { it % 5 == 0 }
35             .also { println(it) }   // [5, 10, 15, 20]
36             .map { it -> it / 2 }
37 println(list)                      // [2, 5, 7, 10]
```

Finally, say goodbye to *findViewById*!

```kotlin
import kotlinx.android.synthetic.main.content_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // No need to call findViewById(R.id.textView) as TextView
        textView.text = "Kotlin for Android rocks!"
    }
}
```

Android KTX

- ***Android KTX*** is a set of *Kotlin extensions* that is part of the ***Android Jetpack*** family.
- It optimizes *Jetpack and Android platform APIs* for ***Kotlin*** use, while doesn't add any new features to the existing *Android APIs*.
- The purpose of ***Android KTX*** is to make Android development with ***Kotlin*** more concise, pleasant, and idiomatic by leveraging ***Kotlin*** language features, such as:
    - Extension functions and properties
    - Lambdas
    - Named parameters and parameter default values

Migration advices and Materials

- Advice from *Google*
- Advice from *Basecamp*

Android Developers: A gradual approach

1. **Start by writing tests in *Kotlin*.**
   Tests are not bundled with your app during packaging, they are a safe place to add *Kotlin* code.
2. **Write new code in *Kotlin*.**
   Before converting existing Java code, try adding small pieces of new Kotlin code like a small class or top-level helper function.
3. **Update existing code to *Kotlin*.**
   Consider extracting small bits of Java functionality and converting to Kotlin classes and top-level functions.

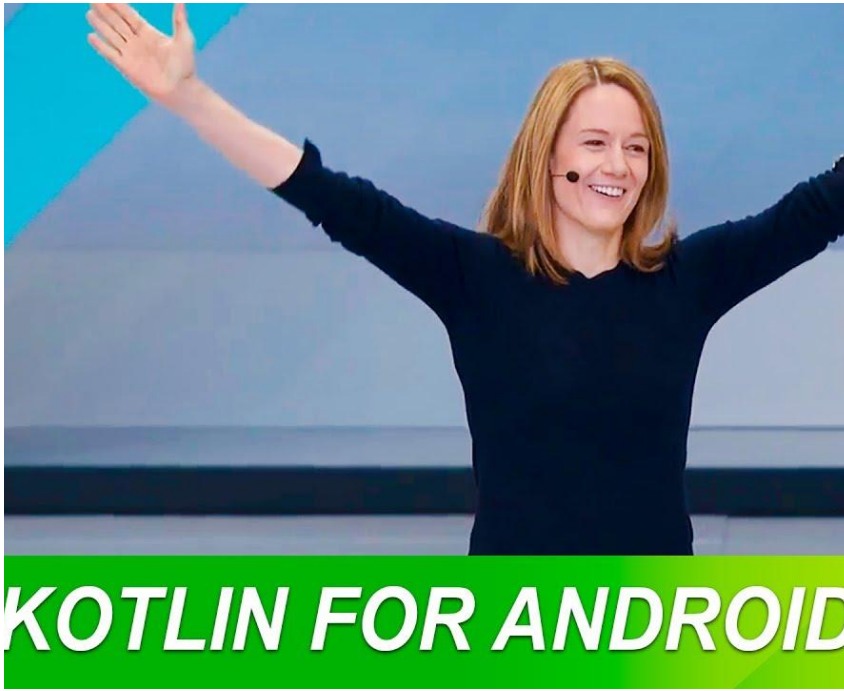
Kotlin fully supported in *Android Studio 3.0* and higher

Basecamp

01

1. **Write real production code.**
   It's something you can see working in your app right away and it's more fun! That feeling of accomplishment and seeing something work shouldn't be discounted — it builds your confidence and keeps you motivated.

2. **Don't try to learn the whole language at once**
   Find a few key concepts that click in your brain (not what others tell you are the best parts of the language). Focus just on those concepts and practice using them to their fullest, without feeling overwhelmed.



Basecamp

02

3. **Question all your Java habits**
   Regularly question whether you are doing something the "*Kotlin way*". When you see code that feels long or complicated, pause and take another look at it.

4. **Use cool downs for Kotlin**
   A great time to work on Kotlin conversions is when you're cooling down off a big release and watching for stability and customer issues.

5. **Learn from the auto converter**
   Auto convert the class, but keep the Java class handy. Put them side by side, and see how the Kotlin compares.

Tutorials

android
announces
support for
Kotlin

[Jake Wharton's talk about Kotlin for Android.]

From Android KW December 2015 hosted by Matthew P.

[How to Kotlin]

From the Lead Kotlin Language Designer (Google I/O '18)

[#31DaysOfKotlin]

From Android Developer Advocates Florina Muntenescu and Sean McQuillan

Documentations & Samples

Kotlin is an open source language with its own documentation and community:

01 | Get Started with Kotlin on Android

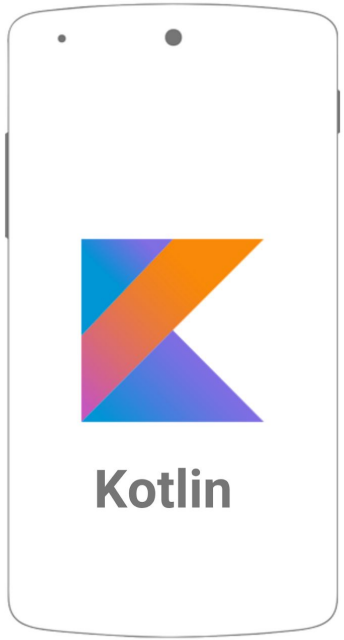
02 | Add Kotlin code

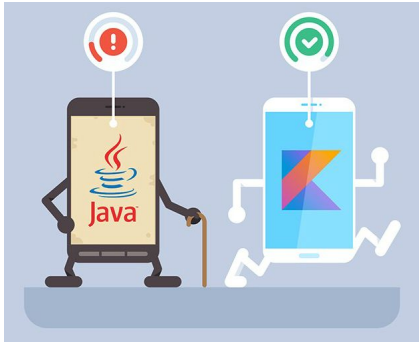
03 | Android Kotlin sample apps

04 | Getting started with Android and Kotlin

05 | Kotlin Koans Online

06 | Udacity Kotlin Bootcamp for Programmers





Thank you!