GDG Android Athens
Meetup - 2017/10/11

Is this Swift for Android?

A short introduction to the **Kotlin** language

Antonis Lilis
Mobile Engineer @ ADVANTAGE
FINANCIAL SYSTEMS EXPERTS

Some History

- **2011:** JetBrains **unveiled** Project Kotlin, a new language for the JVM
- **2012:** JetBrains **open sourced** the project under the Apache 2 license
  - Swift was introduced later at Apple's 2014 WWDC
- **2016:** Kotlin **v1.0** is released
- **2017:** Google announced **first-class support for Kotlin on Android**
- Kotlin is technically 6, but in reality **1 year old**

The Kotlin Language

- **Statically Typed**
  - Type validation at compile time
- Supports **Type Inference**
  - Type automatically determined from the context
- Both **Object Oriented** and **Functional**
- **First-class functions**
  - You can store them in variables, pass them as parameters, or return them from other functions
- Can be **mixed with Java** in the same project

Constants and Variables

- **val** (from value)
  - **Immutable** reference
- **var** (from variable)
  - **Mutable** reference
- **Nullable** Types
  - Defined **Explicitly**

```
val someInt: Int = 42
var someString = "forty-two"
var someValue: Int? = 23

someInt = 23 //It is constant
someString = "twenty-three"
someString = 5 //It is a String
someString = null //Cannot be null
someValue = null
```

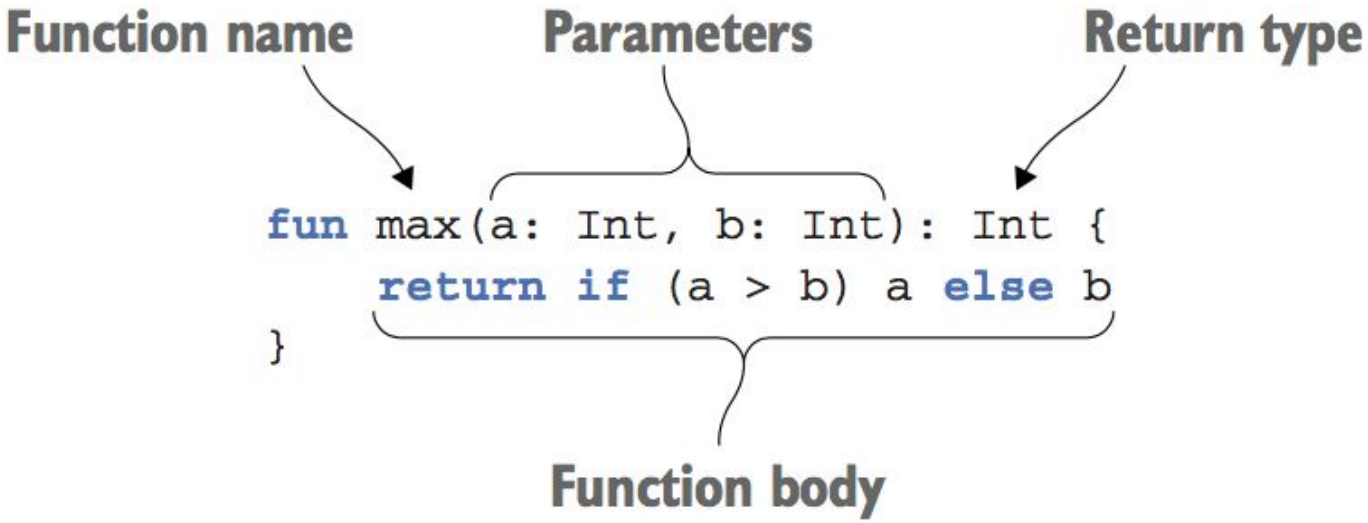
Control Flow

- Classic loops:
  - **if**
  - **for**
  - **while / do-while**
- **when**
  - Replaces the switch operator
  - No breaks, no errors

```
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> { //block
        print("not 1 or 2")
    }
}

for (i in 1..100) {
}
for (i in 100 downTo 1 step 2) {
}
for (i in 0 until 100) {
}

val list = arrayListOf("1", "2", "3")
for (item in list) {
    println("item: $item")
}
```

Functions

- **Named** arguments
- Can be declared at the **top level** of a file (without belonging to a class)
- Can be **Nested**



Functions

- **Default** parameter values
- Can have a **block or expression body**

```
fun doSomethingWith(letter: Char, number: Int = 42) {
    val res = "The letter is ${letter} and the number is $number"
    println(res)
}

fun test() {
    doSomethingWith(letter = 'C', number = 1)

    doSomethingWith(letter = 'A')

    doSomethingWith( letter: 'A')
}

//Expression body
fun max(a: Int, b: Int): Int = if (a>b) a else b
```

Classes

```
class MyView : View {
    constructor(ctx: Context): super(ctx) {
        //Initialization stuff
    }
    //...
}

class MyViewShort(ctx: Context) : View(ctx) {
    //...
}

class Car(val brand: String, val isUsed: Boolean = false)

val car = Car( brand: "Ford")

data class Bike(val brand: String, val isUsed: Boolean = false)
```

"**Any**" is the analogue of java **Object**: a superclass of all classes

data classes: **autogenerated** implementations of universal **methods** (equals, hashCode etc)

Properties

- **first-class** language feature
- combination of the **field** and its **accessors**

```
class House {

    var street: String = "Ermou"
    var number: String = "1"
    var city: String = "Athens"

    var state: String? = null

    var zip: String = ""
        set(value) {
            state = "TK.$value"
        }

    val prettyAddress: String
        get() = "$street $number, $city"

}
```

Modifiers

- **Access** modifiers
  - **final** (default)
  - **open**
  - **abstract**

- **Visibility** modifiers
  - **public** (default)
  - **internal**
  - **protected**
  - **private**

"Design and document for inheritance or else prohibit it"

Joshua J. Bloch, Effective Java

No static keyword

- **Top-level** functions and properties

  (e.g. for utility classes)
- **Companion objects**
- The **object keyword**:

  declaring a class and creating an instance,

  combined (**Singleton**)

```
class Foo {
    companion object {
        fun bar() {
            //...
        }
    }
}

object Singleton {
    fun doSomething() {
        //..
    }
}


Foo.bar()

Singleton.doSomething()
```

Extensions

- Enable **adding methods and properties** to other people's classes
  - Of Course without access to private or protected members of the class

```
fun String.lastChar(): Char
        = this.get(this.length - 1)

val String.lastChar: Char
    get() = get(length - 1)


val last: Char = "hi".lastChar()

"hello".lastChar
```

Null Checks

- **Safe-call** operator **?.**
- **Elvis** operator **?:**
- **Not-null assertion** operator **!!**

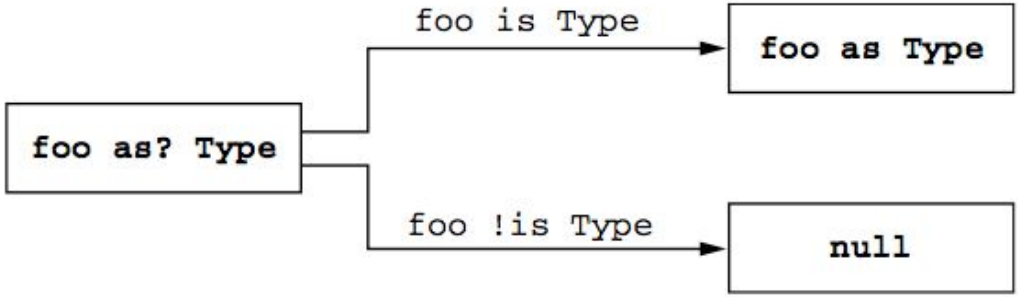
```
var myview: MyView? = MyView(ctx)

myview?.bar()

val t: String = s ?: ""

fun rootOfAllEvils(s: String?) {
    val sNotNull: String = s!!
    println(sNotNull.length)
}
```

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

Safe Casting

- **Safe cast** operator **as?**
- **Smart cast**
  - combining type checks and casts

```
var myview: MyView? = MyView(ctx)

val view = myview as? View

if (view is MyView) {
    view.bar()
}
```



Collections

- Kotlin **enhances** the **Java** collection classes (**List, Set, Map**)

```
class Car(val brand: String, val age: Int, val horsePower: Int)

val fleet = listOf(
        Car( brand: "Ford",  age: 1,  horsePower: 100),
        Car( brand: "Mazda",  age: 2,  horsePower: 120),
        Car( brand: "Opel",  age: 2,  horsePower: 95))

fleet.maxBy { it.horsePower }

fleet.filter { it.age == 2 }

fleet.filter { it.age == 2 }.maxBy { it.horsePower }

fleet.forEach { print("brand: $it.brand") }
```

Delegation

- **Composition over Inheritance** design pattern
- **Native support** for delegation (**implicit** delegation)
- **Zero Boilerplate** code
- Supports both **Class Delegation** and **Delegated Properties**

```
interface Nameable {
    var name: String
}

class Ford : Nameable {
    override var name = "Ford"
}

class Car(name: Nameable)
    : Nameable by name


val car = Car(Ford())
print(car.name) //Ford
```

Class Car **inherits from an interface** Nameable and **delegates** all of its public **methods to** a delegate **object** defined with the **by keyword**

Coroutines

- Introduced in Kotlin **1.1** (March)
- A way to write **asynchronous code sequentially**
- **Multithreading** in a way that is easily debuggable and maintainable
- Based on the idea of **suspending function execution**
- More **lightweight** and efficient than threads

```
async(UI) {
    val r1 = bg { fetchResult1() }
    val r2 = bg { fetchResult2() }
    updateUI(r1.await(), r2.await())
}
```

Source Code Layout

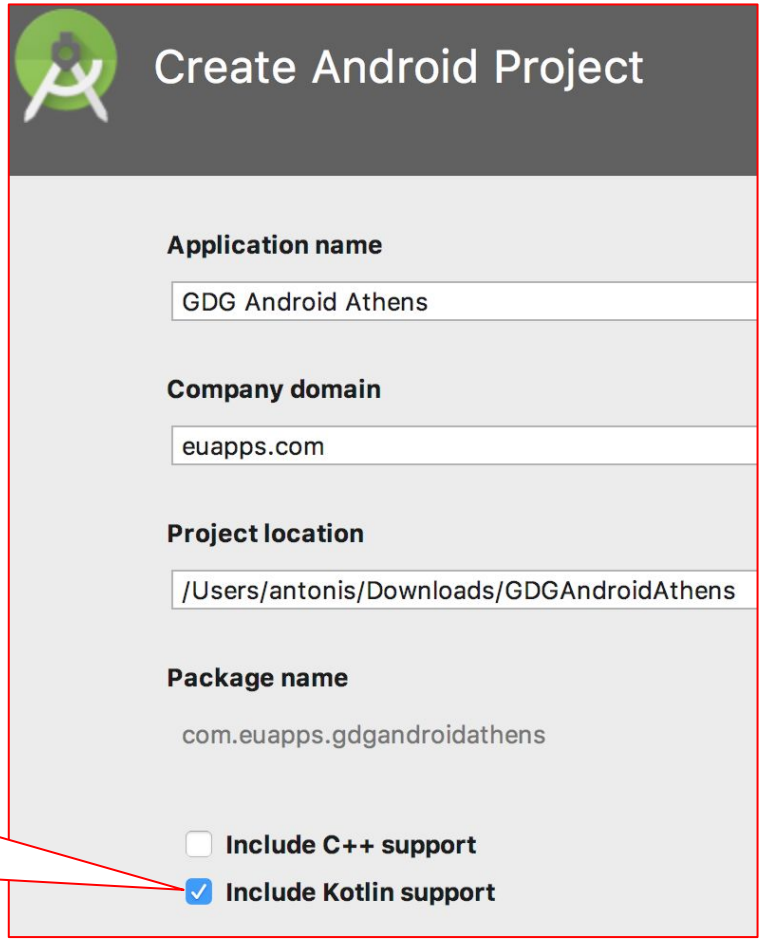
- **Packages** (similar to that in Java)
- **Multiple classes** can fit **in the same file**
- You can choose **any name for files** (not restricted to class name)
- The **import** keyword is not restricted to importing classes
  - Top-level functions and properties can be imported
- Kotlin does **no**t impose any **restrictions** on the layout of source files on disk
- Good practice to **follow Java's directory layout**
  - Especially if mixed with java



Kotlin for Android

Android Studio 3.0 Preview
has Kotlin Support out of the box

```
apply plugin: 'com.android.application'

apply plugin: 'kotlin-android'

apply plugin: 'kotlin-android-extensions'
```



Hello Android

Let’s create a simple **XML Layout**

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

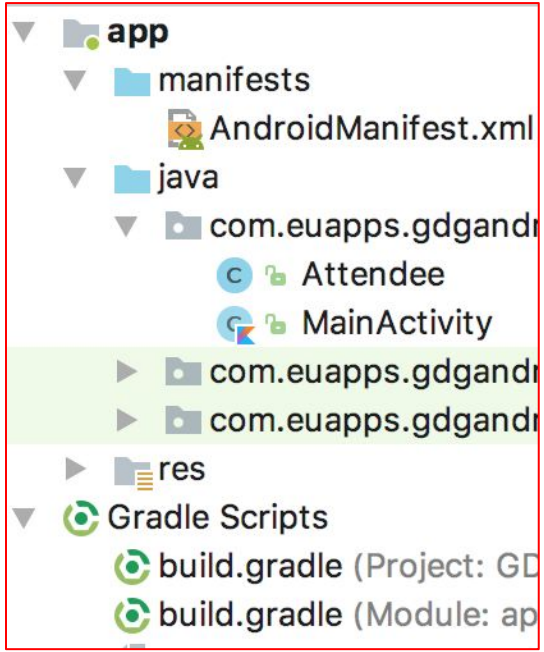
Hello Android

...and an **Activity**

```
package com.euapps.gdgandroidathens

import android.app.Activity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : Activity() {

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

        textView.setText("Hello GDG Android Athens")
    }
}
```

No findViewById: Kotlin Android Extensions allows you to import a reference to a View

Let's Mix with some Java




```java
package com.euapps.gdgandroidathens;

public class Attendee {

    private String name;
    private String email;

    public Attendee(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public String getName() {
        return name;
    }
}
```

Java from Kotlin

```kotlin
package com.euapps.gdgandroidathens

import android.app.Activity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : Activity() {

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

        val attendee = Attendee( name: "Antonis",
                email: "antonis.lilis@gmail.com")

        textView.setText("Hello ${attendee.name}")
    }
}
```

Convert Java to Kotlin


```
package com.euapps.gdgandroidathens

class Attendee(val name: String, private val email: String)
```

Utility

- **Top-level** computed **property**
- String **extension**

Meetup.java | Utils.kt | Attendee.kt | MainActivity.kt

```kotlin
package com.euapps.gdgandroidathens

val String.isValidEmail: Boolean
    get() {
        //Dummy implementation
        return isNotEmpty() &&
                contains('@') &&
                contains('.')
    }
```

Kotlin from Java

- **Seamless** integration
- **Static class is generated** for the **top-level** declarations

```java
package com.euapps.gdgandroidathens;

import java.util.ArrayList;
import java.util.List;

public class Meetup {
    private String title;
    private List<Attendee> attendees;

    public Meetup(String title) {
        this.title = title;
        this.attendees = new ArrayList<Attendee>();
    }

    public void addAttendee(String name, String email) {

        if (UtilsKt.isValidEmail(email)) {

            Attendee attendee = new Attendee(name, email);
            attendees.add(attendee);
        }
    }
}
```

Libraries

- You can use **Any Java Library**

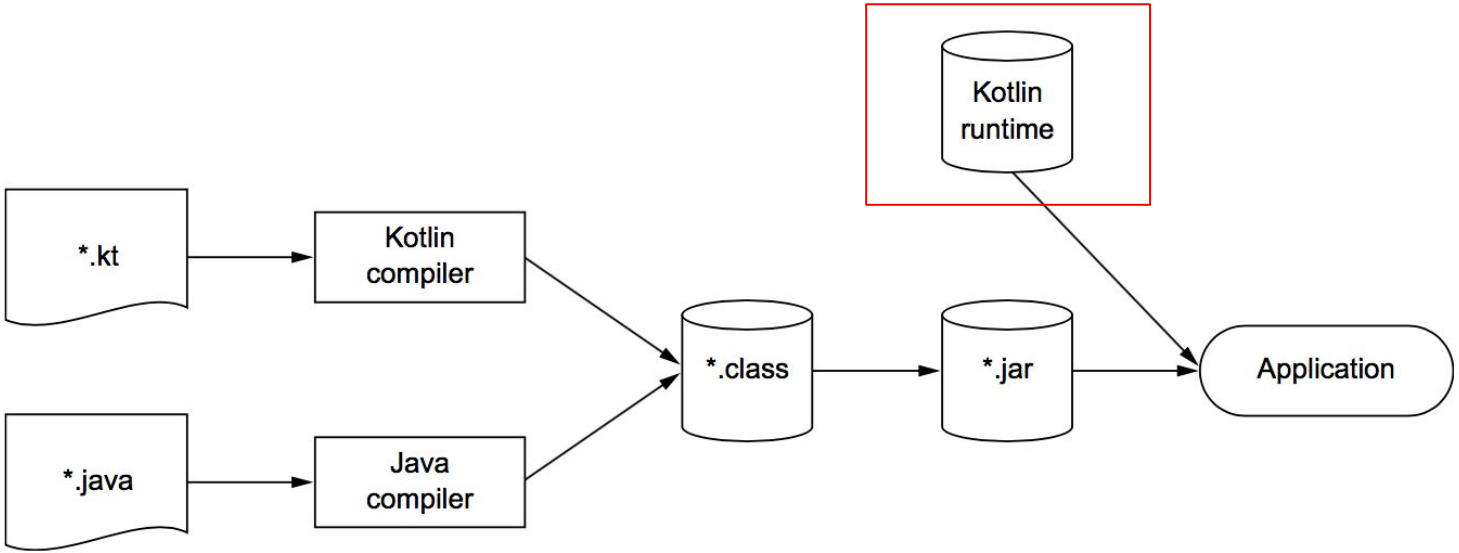
- **Kotlin libraries**
  - eg. **RxKotlin** - Kotlin Adaptor for RxJava
  - A nice curated list at https://kotlin.link

- The **Anko library** developed by the **Kotlin Team**
  - Easy Asynchronous Tasks
  - Layout Handling
  - SQL Lite utilities
  - Much more...

Anko

Any Disadvantages?

- An app built with Kotlin will likely result in a **larger file** package size than one built purely in Java
- The **build time** for Kotlin is a little slower using Gradle



Kotlin vs Swift

Run Kotlin REPL (in module GDGAndroidAthens)

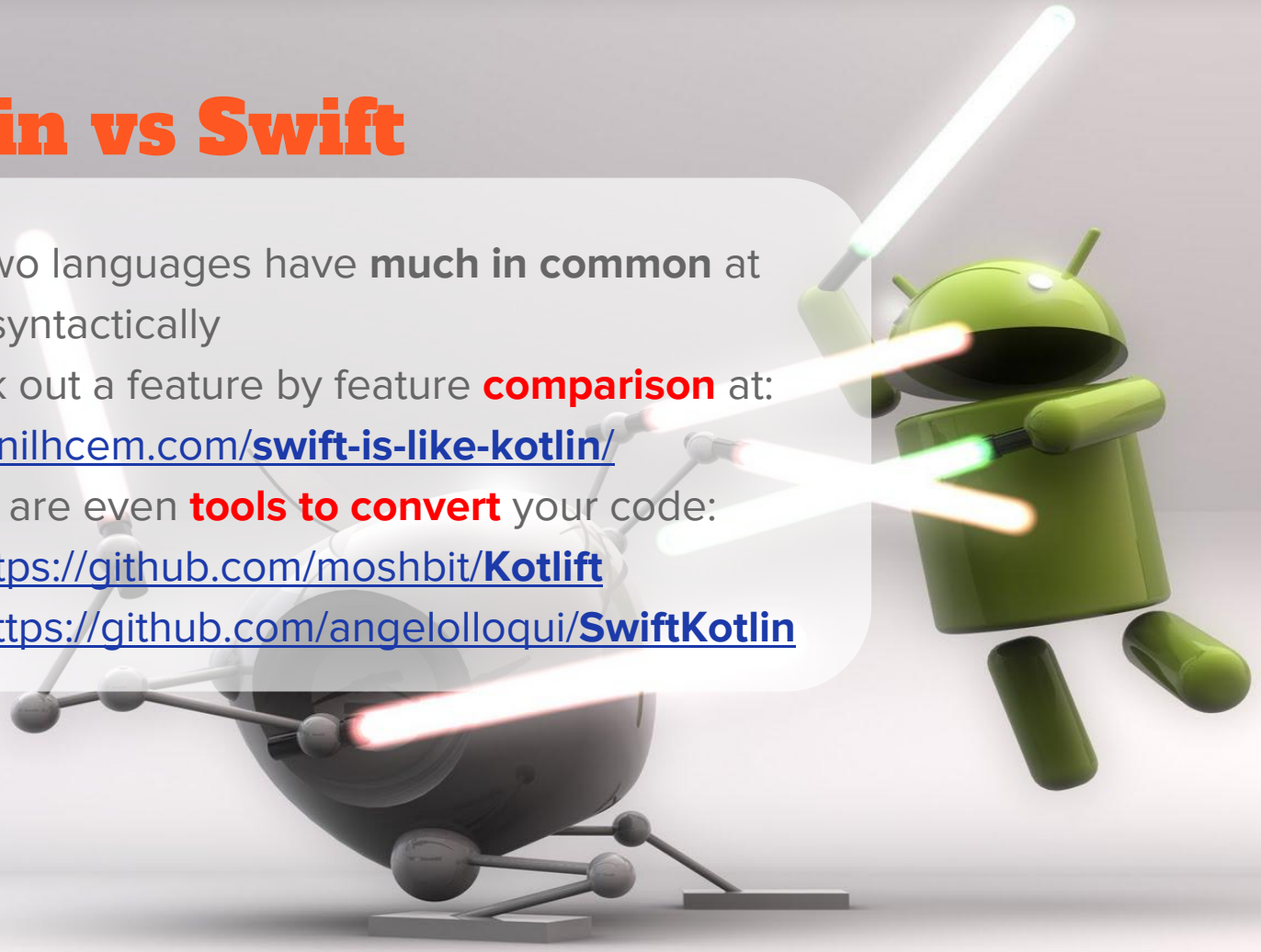
```kotlin
val helloString = "Hello Kotlin"
print(helloString)
Hello Kotlin

var myVariable = 42
val explicitDouble: Double = 70.0
print("my Variable is ${myVariable}")
my Variable is 42

for (index in 1..5) {
    println("$index times 5 is ${index * 5}")
}
1 times 5 is 52 times 5 is 103 times 5 is 154 times

class Shape {
    var numberOfSides = 0
    fun simpleDescription(): String {
        return "A shape with $numberOfSides sides."
    }
}
var shape = Shape()
shape.numberOfSides = 7
var shapeDescription = shape.simpleDescription()
fun area(width: Int, height: Int) : Int {
    return width * height
}
area(width = 2, height = 3)
6
```

MyPlayground

```swift
let helloString = "Hello Swift"
print(helloString)
var myVariable = 42
let explicitDouble: Double = 70
print("my Variable is \(myVariable)")
for index in 1...5 {
    print("\(index) times 5 is \(index * 5)")
}
class Shape {
    var numberOfSides = 0
    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
var shape = Shape()
shape.numberOfSides = 7
var shapeDescription = shape.simpleDescription()
func area(width: Int, height: Int) -> Int {
    return width * height
}
area(width: 2, height: 3)
```

Kotlin vs Swift

- The two languages have **much in common** at least syntactically
- Check out a feature by feature **comparison** at: http://nilhcem.com/**swift-is-like-kotlin**/
- There are even **tools to convert** your code: eg. https://github.com/moshbit/**Kotlift** and https://github.com/angelolloqui/**SwiftKotlin**



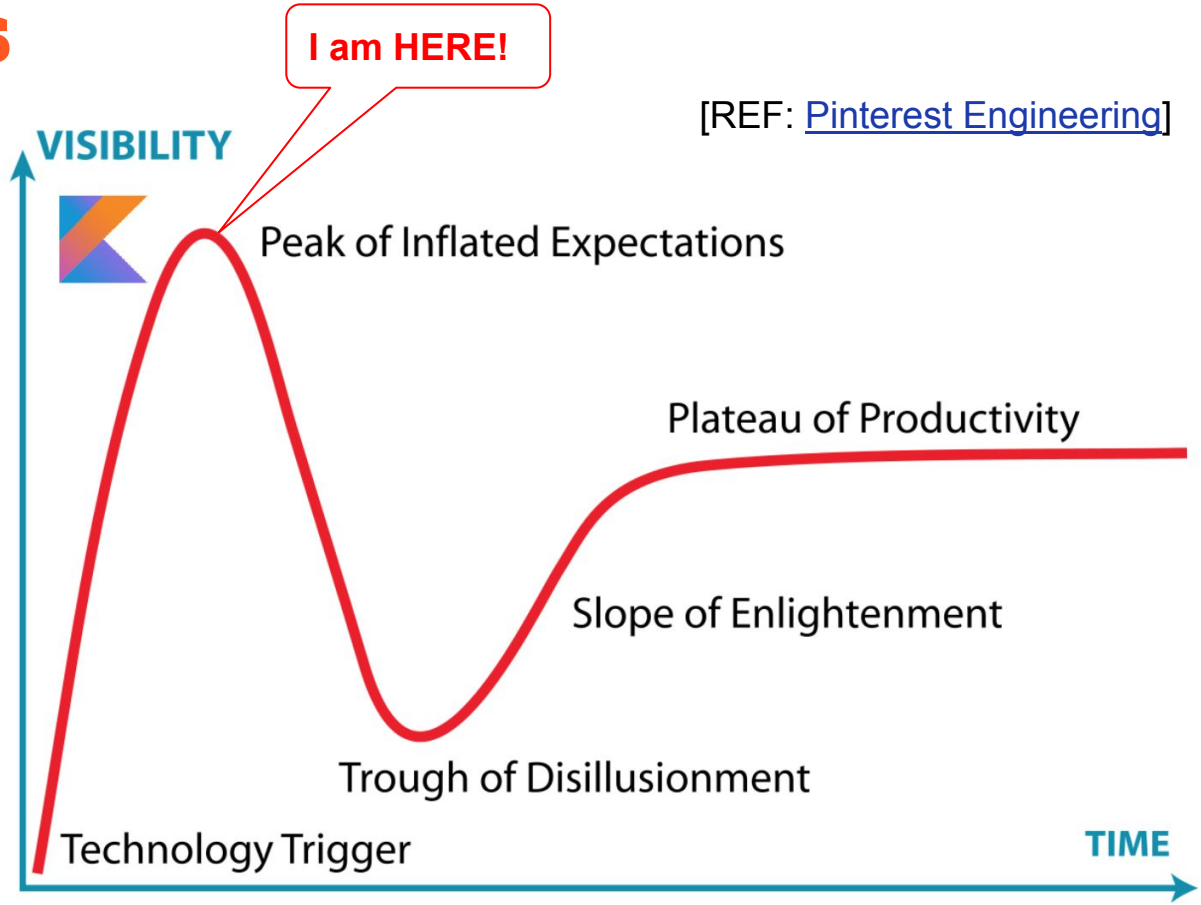
Final Thoughts

- The **learning curve**
  - IMHO comparatively small
- **Not so popular** yet
  - **41st** in TIOBE Index for October (was 80th in May)
- Development **Stability**
  - Tools still in **Beta**
  - **Static Analysis** Tools
- **Reversibility**
  - Once you Go Kotlin...

IMHO **Kotlin is here to stay**

(at least on **Mobile**)

[REF: Pinterest Engineering]



Kotlin on the Hype Curve

Thank you!

Questions?

http://antonis.me/