Intro to Kotlin

What is Kotlin?

- An open source JVM language headed by JetBrains
  - Unveiled: July 2011 (in development for about year)
  - Open sourced: February 2012
  - 1.0: February 15, 2016
- At JetBrains, they have essentially stopped writing Java, to focus on Kotlin
- Named after an island in Russia
- Also compiles to JavaScript for some reason?

Why Kotlin?

- On Android, Java support is stuck on a weird hybrid of versions 6 and 7
- This version of Java does not have lambdas
- Java is pretty verbose in general

- Retrolambda can convert Java 8 lambdas to bytecode compatible with Android
- Jack compiler introducing the path ahead for future versions of Java

Ok, So Why Kotlin?

- Null safety
  - Nullability baked directly in the type system
  - Android APIs rely heavily on null
  - Takes out an entire subset of bugs
- Kotlin syntax is very concise compared to Java, with type inference
- Extension functions to add functionality to classes you don't own
- Small run time compared to Scala, Groovy
  - Std lib is mostly comprised of extension functions on Java types
- Java interop is a core component of Kotlin
- Android community is strong and growing

Warning… Code incoming

Syntax Crash Course

```
class Foo : Bar {
    override fun baz(fizz: String) : String {
        val fuzz = fizz.toUpperCase()
        return fuzz
    }
}

data class FizzBuzz(val x: String, var y: String)  // isEqual, hashCode, etc.

fun topLevelFunction(x: String?) : String? {
    return x?.toUpperCase()
}

fun anotherFunction(x: String) : String {
    return x.toUpperCase()
}
val x = topLevelFunction("hey")  // x: String?
anotherFunction(x)  // compilation error, cannot pass a String?
```

Extension Functions

```
// Java
class StringUtils {
    public static String addThreeSpaces(String x) {
        return x + "   "
    }
}
String x = "hey";
String x3 = StringUtils.addThreeSpaces(x);

// Kotlin
fun String.addThreeSpaces(): String {
    return this + "   "
}
val x = "hey"
val x3 = x.addThreeSpaces()

fun String.addThreeSpaces() = this + “   “  // compressed syntax for a single line
```

Lambdas / Higher-order functions

```
val x : () -> Unit = {println("Hey")}  // Unit == void

fun foo(w: Int, lambda: () -> Int) : Int {
    val y = w * lambda()
    return y
}

// if lambda is last argument of function, can define it outside of parentheses
val z = foo(2) {
    2
}
z == 4
```

last statement is the return value, or do **return@foo** if you need multiple return locations

Shared Preferences Example

In Java

```
SharedPreferences prefs = getSharedPreferences();
SharedPreferences.Editor editor = prefs.edit();
editor.putString("hi", "hey");
editor.putBoolean(“flag”, true);
editor.apply();
```

Kotlin

```
fun SharedPreferences.transaction(transaction: (SharedPreferences.Editor) -> Unit) {
    val editor = edit()
    transaction(editor)
    editor.apply()
}

val prefs = getSharedPreferences()
prefs.transaction { editor ->
    editor.putString("hi", "hey")
    editor.putBoolean("flag", true)
}
```

Shared Preferences Example -- Full Inception Mode

Kotlin

```
inline fun SharedPreferences.transaction(transaction: SharedPreferences.Editor.() -> Unit){
    val editor = edit()
    editor.transaction()
    editor.apply()
}

val prefs = getSharedPreferences()
prefs.transaction {
    // we are an extension function on Editor
    this.putString("hi", "hey")
    putBoolean("flag", true)
}
```

Zero Cost Abstraction

```
// Generated Bytecode (represented as Java syntax)
SharedPreferences prefs = getSharedPreferences();
SharedPreferences.Editor editor = prefs.edit();
editor.putString("hi", "hey");
editor.putBoolean(“flag”, true);
editor.apply();
```

Other cool things

- In latest release, coroutines were added
  - Similar to Python 3 async io or C# async/await
- The Gradle build system now supports writing build scripts in Kotlin
- Kotlin Android extensions
  - Attaches fields to Android activity classes for UI elements from XML
  - Old: (Button) findViewById(R.id.button)
  - New: this.button
- Anko DSL for Android is an interesting showcase for how far extension functions can go

Anko by JetBrains

```
// in onCreate of Activity
verticalLayout {
    val name = editText()
    button("Say Hello") {
        onClick { toast("Hello, ${name.text}!") }
    }
}
```

- Builds Android UI components programmatically, instead of XML
    - Very painful to do this traditionally, instantiating View objects manually and associating parent/child view relationships
- Keeps the structure of the UI -- Similar to HTML
- Completely type safe
    - E.g. A button can't be the top level component, since the button function only exists when in the context of a ViewGroup (verticalLayout)

Not So Great Things

- There is a runtime cost
  - Much less than Scala or Groovy
- Sometimes the Kotlin plugin for IntelliJ isn’t smart
- Generics are handled slightly differently in Kotlin, sometimes some interesting syntax to get something that would work in Java
  - out / in, reified, * ???
- Static functions and fields are more annoying to add to a class
  - Companion objects are a strange concept
- By default, Kotlin classes can not be inherited
  - Must declare class as open -- concept comes from the book Effective Java