Kotlin -- the "other" language for Android

- "modern" programming language, released in July 2011 by JetBrains, still changing (1.3 has "experimental" features)
- generates code for JVM, browsers (javascript), native
- compatible with JDK 6, 100% interoperable with Java
- (designed for less typing, not readability IMO *Nelson*)
- due to interop with Java, has packages and import statements
- Comments like C++, // one line, /* ... until */
  
  ```
  package my.pack
  import java.util.*
  ```
- Function definitions:
  ```
  fun sum (a: Int, b: Int): Int {
    return a + b
  }
  ```
- Or as an expression
  ```
  fun sum (a: Int, b: Int) = a+b
  ```
- Return type for a "procedure" Unit (optional)
Functions may have default arguments

```kotlin
fun nofun(b: Array<Byte>, off: Int = 0, len: Int = b.size) {...}
```

Can call using name:

```kotlin
nofun(len=10, b=myarray)
```

Can’t call java code this way

May have a "vararg" parameter

```kotlin
fun foo(vararg strings: String) {...}
```

foo(strings = *arrayOf("a", "b", "c"))

must use the named form and the "spread operator *"

Explicit return "expression"

```kotlin
return result
```

Parameters (appear to be) val parameters
Generic functions

```kotlin
fun <T> asList(vararg ts: T): List<T> {
    val result = ArrayList<T>()
    for (t in ts) result.add(t)
    return result
}
val list = asList(1,2,3)
val a = arrayOf(1,2,3)
val list = asList(-1,0,*a,4)
```

Functions can be local to other functions

Functions can be member functions

```kotlin
class Simple() {
    fun jane() { print("jane") }
}
```

Other function stuff ... infix, inline, higher-order and lambdas, tail recursive
Data in Kotlin

□ "constants" -- known as "Read-only variables" -- val
  
  val a: Int = 1
  val b = 2
  val c: Int
  c = 3 // deferred assignment, allowed only once

□ true variables use var

  var x = 5
  x += 1

□ "Top-level" (e.g. global) variables declared outside functions

□ "properties" .. e.g. instance variables, member variables

class info {
  var name: String = ... // May have an initial value
  var phone: String
  val id: Int
}

class info {
    var name: String = ... // May have an initial value
    var phone: String
    val id: Int
}

- Use similar to other languages
val result = info() // no new keyword
result.name = "..."
result.phone = "..."
result.id = 1

- can add getters and setters
class name {
    var vname: Int
        get() = vname
        set(value) = { vname = value }
}

- More complications with getters/setters (read docs)
Standard Data types
- Double (64 bits), Float (32 bits)
- Long (64), Int (32), Short (16), Byte (8)
- Boolean (true, false)
- String, Char (e.g. '1', not a special int)

Literals: 123 vs 123L, 0x0f, 0b00110011
- 123.5 vs 123.4f (doubles by default)
- 1_000_000 is legal

Array done by the Array class

class Array<T> private constructor() {
    val size: Int
    operator fun get(index: Int): T
    operator fun set(index: Int, value: T): Unit
    operator fun iterator(): Iterator<T>
    ...
}
val constArray = arrayOf(1,2,3)

Other ways to create arrays
Strings
- Strings are immutable
- Elements can be indexed using s[i], char
- Can be iterated (for (c in str) { ... })
- + is concatenation, "str"+var converts var to string
  - "String templates are preferable to concatenation"

String Literals
- "....." May have escaped characters, \t, \b, \n, \, \", \\ and \\
- """" string """" (no escaped chars, may be multi-line)
  ```
  val text=""
  |First Line.
  |Second line.
  |"""".trimMargin()  // gets rid of leading space to and including |
  .trimMargin(">")  // changes the space
String Templates

```kotlin
var a = 1
var s1 = "a is $a"
a = 2
val s2 = \"${s1.replace("is","was\")}\, but now is \$a\""
```

Control Flow

- If "statements" .... although Kotlin community prefers "expression"
- `if (cond) { then_part } else { else_part}`
- Conditional expressions
  - `if (a > b) a else b`
- When expressions
  ```kotlin
  when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> {
      print("x is not 1 or 2")
    }
  }
  ```
Control Flow (page 2)

- More when stuff
  - 0, 1 -> ...
  - funcall() -> ... (return true selects)
  - in 1..10 ->
  - !in 10..20 ->

- For Loops
  - for (item in collection)
  - supports iterator() and next() and hasNext()
  - for (i in 1..3) ...
  - for (i in 6 downTo 0 step 2) ...

- While Loops
  - while (cond ) {...}
  - do { ... } while (cond)

- break and continue -- work ... enclosing loop
- break and continue can work with a label

loop@ for ( i in 1..100) { for (j in 1..100) { if (...) break@loop } }

Classes -- as usual, a major idea

- class name { ... }
- constructor: primary and secondary
  - Primary: class Myclass constructor(arg: type) {...}
    - code is in an "init" block (can have several)
  - secondary: constructor (args) {...code ...}
- Inheritance
  - Default superclass "Any", has equals(), hashCode() and toString()
  - Can do own:
    - class Base(p: Int) {...}
    - class Derived(p: Int) : Base(p) {...}
- Method and property definition
  - inline declaration
  - fun mymethod() ...
  - val x = ...
  - var y = ...
  - visibility modifiers:
    - private -- inside class only (and members)
    - public -- any client with access to class
(visibility modifiers)
- protected -- private, visible in subclasses
- internal -- any client inside this module

Note: outer class does not see private members of inner classes

Overriding:
- function overriding, can change visibility
- data members, val can’t override a var (var can override a val)

Nullable values, Int?, classname? (must be marked)

Checking if an object is a specific type: is operator

fun getStringLength(obj: Any): Int? {
    if (obj is String) { return obj.length }  // return obj.length }  // return null
}
Kotlin (other elements)

- Standard Library Collections: List, Set, Map
  - iterators, various kinds
- Many other stuff in standard library, now includes coroutines
- Most of the rest of issues are class related
Other final items

- **Styles and Themes**
  - Markup in strings ... eg: `<i>italic</i>`
    - `i` - italic, `b` - bold, `u` - underlined
    - `sup` - superscript, `sub` - subscript, `strike`
  - `big`, `small`, `monospace`

- Styles -- can be defined in XML
- Applied to views
- Various attributes: `textColor`, `typeface`, `fontsize`, ...
- Can be directly applied via code.
- Many standard styles defined
  - See Android/Sdi/platforms/android-xx/data/res/values

- **Themes**
  - Problem with styles ... need to be applied to each view requiring that style
  - Theme can be applied to the application or an activity
    - Done in the manifest: `android:theme="@style/..."`
Standard themes
- Theme.NoTitleBar, Theme.NoTitleBar.Fullscreen
- Theme.Translucent, ...
- Holo.Light, Holo.Light.DarkActionBar
- Theme.AppCompat.*

Networking ...
- Talked about web services ...
- Can do full network programming with sockets
- java.net.Socket and SSLSocket
- android.bluetoothBluetoothSocket
- You can set up listening sockets as well as outgoing sockets
- Bluetooth
  - Have to make sure devices are paired first
  - Use the BluetoothSocket for communications
  - Do network communication on a background thread
Android has a WiFi peer-to-peer subsystem
- Can send photos over WiFi peer-to-peer, shouldn’t over bluetooth
  - getSystemService(Context.WIFI_P2P_SERVICE)
  - discover peers
  - connecting to them
  - communication ... again socket based

NFC - Near Field Communication -- typically less than 4cm
- small amounts of data
- Uses Android Beam .. simple API for NFC messages

Geocoding, Geofencing

Google Play Services, Firebase
- Adds and other ways to monetize