Functions in Kotlin

Kevin Jones

@kevinrjones www.rocksolidknowledge.com

Functions in Kotlin

Don't need to be part of a class

Are introduced with the 'fun' keyword

Can have default parameters

Can have named parameters

Can 'extend' existing types

Function Declaration

```
fun connect(addr: URI) : Boolean {
}
```

Calling From Java

How are top level functions compiled?

Function Declaration

```
// could use @file:JvmName("UriFunctions")
package rsk;

fun connect(addr: URI) : Boolean {
}
```

What Java Sees

```
// assume filename of Util.kt


package rsk;

// if @JvmName then class would be UriFunctions

class UtilKt () {

    static boolean connect(URI addr) {

    }

}
```

Function Expressions

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

Local Functions

```
fun parsePage(page: String) : Page {
    fun parseHeader(header: String) {
    }
}
```

Default Parameters

Kotlin supports default parameters

Default Parameters

```
fun connect(addr: URI, useProxy:Boolean = true) : Boolean {
}
```

Default Parameters and Java

@JvmOverloads

Named Parameters

Kotlin supports named parameters

Named Parameters

```
fun connect(addr: URI, useProxy:Boolean = true) : Boolean {
}

connect(URI(...), useProxy = false)
```

Extension Functions

Can 'add' functions to classes not owned by you

Kotlin generates static functions

Cuts down on use of utility classes

Makes code easier to read

Extension Functions

```
fun InputStream.readLine(): String { ... }
```

Extension Functions

Add to existing classes

Add to your own classes

Infix Functions

Member or extension functions

Have a single parameter

Marked with infix keyword

'infix' Functions

```
infix fun Header.plus(Header other) : Header { ... }


h1,h2, h3:Header

h3 = h1 plus h2
```

Operator Overloading

Predefined set of operators can be overloaded

Overloading Operators

Unary

+, -, !, ++, --

Binary

+, -, *, /, %, ..

+=, -=, *=, /=, %=

in, [], (),

==, !=

>, <, >= ,<=

Overloaded Operators

```
operator infix fun Header.plus(Header other) : Header { ...
}
```

```
h1,h2, h3:Header

h3 = h1 plus h2

h3 = h1 + h2
```

Tail Recursive Functions

Use tailrec keyword

Have correct 'form'

Kotlin optimises away the recursion

Tail Recursion

```
tailrec fun fib(n: Int, a: BigInteger, b: BigInteger)
                                          : BigInteger {
    return if (n == 0) b else fib(n - 1, a + b, a)
}


fun main(args: Array<String>) {
    // on my machine fails without tailrec
    println(fib(8280, BigInteger("1"), BigInteger("0")))
}
```

Summary

Much easier to use functions

Do not need classes

Can use default and named parameters

Can create extension functions

Can create infix functions

Support for overloaded operators

Support for tail recursion

And One Last Thing

Member functions

Member Functions

```
class Parser() {
    fun parsePage(page: String) : Page {
    }
}
```

What's Next