Let's play with Kotlin

A (super duper amazing) alternative to Java

What's wrong with Java?

- It's the best language ever, isn't it?
- Java 8 is cool but 7 or lower are still mostly used
- Syntax tends to be verbose and not modern anymore (compare to most recent languages)
  - Extensions
  - Closures
  - Optionals

Why Kotlin?

- Sweet alternative to Java
- Offers most recent paradigms
- IDE Integration available
- Not in beta anymore 😉


ARE YOU READY TO
HAVE SOME FUN?
I AM!
memes.com

Introduction

- From JetBrains
- Development started in 2011
- Not a super-language. Compiles directly for the JVM
- http://try.kotlinlang.org/

- No semi-colon
- 4-space indentation

```
package main

fun main() {
    println("Hello World!")
}
```

Variables

```
val constant: Int = 1
var x = 5

x += 5
val y = x

println("x: ${x}")
```

Functions

```
fun sum(a: Int, b: Int): Int {
    return a + b
}

fun myFunction() {
    // Do some stuff
}
```

Conditionals

```
fun max(a: Int, b: Int): Int {
    if (a > b) {
        return a
    } else {
        // Else is always required except if returning
        return b
    }
}

fun max(a: Int, b: Int) = if (a > b) a else b
// Return type is not needed for automatically spotted
```

Conditionals

```
val a = 4, b = 5
val max = if (a > b) {
    a
} else {
    b
}
```

Conditionals

```
var x = 0
x += 1
x = 4
```

```
when (x) {
    0 -> println("foo")
    2, 3 -> println("bar")
    else -> {
        println("default case")
    }
}
```

Loops

```
for (arg in args) {
    // ...
}
```

```
while (i < 0)
    // ...

do {
    // ...
} while (i > 0)
```

Ok so far?

Ranges

```
for (x in 0..5)
    println("$x")
// 0 1 2 3 4 5
```

```
if (x in 0..5)
    // ...
if (x !in 0..5)
    // ...
```

Ranges

```
for (i in 4 downTo 0 step 2) {
    // 4 2 0
}

// You can use:
//  - rangeTo
//  - downTo
//  - reversed
//  - step
```

Loops again

```
for ((i, value) in a.withIndex())
    // ...
```

```
foo@ for (i in 0..100) {
    for (j in 0..100) {
        if (condition)
            break@foo
    }
}
```

Optionals

```
var a = "foo"

a = null  // Have fun, dude
// Forbidden for a is a String
// Forced to use String? instead

var b: String? = "foo"
b = null
```

Optionals

```
fun parseInt(input: String): Int? {
    // Parse string
    if (isAnInteger) {
        return value
    } else {
        return null
    }
}

val a = parseInt("45")

if (a == null) {
    println("Not an integer")
    return
}

// a is automatically unwrapped after the condition above
var b = a + 90
```

Optionals

```
fun toInt(o: Any): Int? {
    if (o is Int) {
        return o + 15
    }

    return null
}
```

Lambdas

```
val f: (Int) -> Int = { x ->
    return x + 1
}

val g: (Int) -> Int = { return it + 1 }
// Default name
```

Lambdas

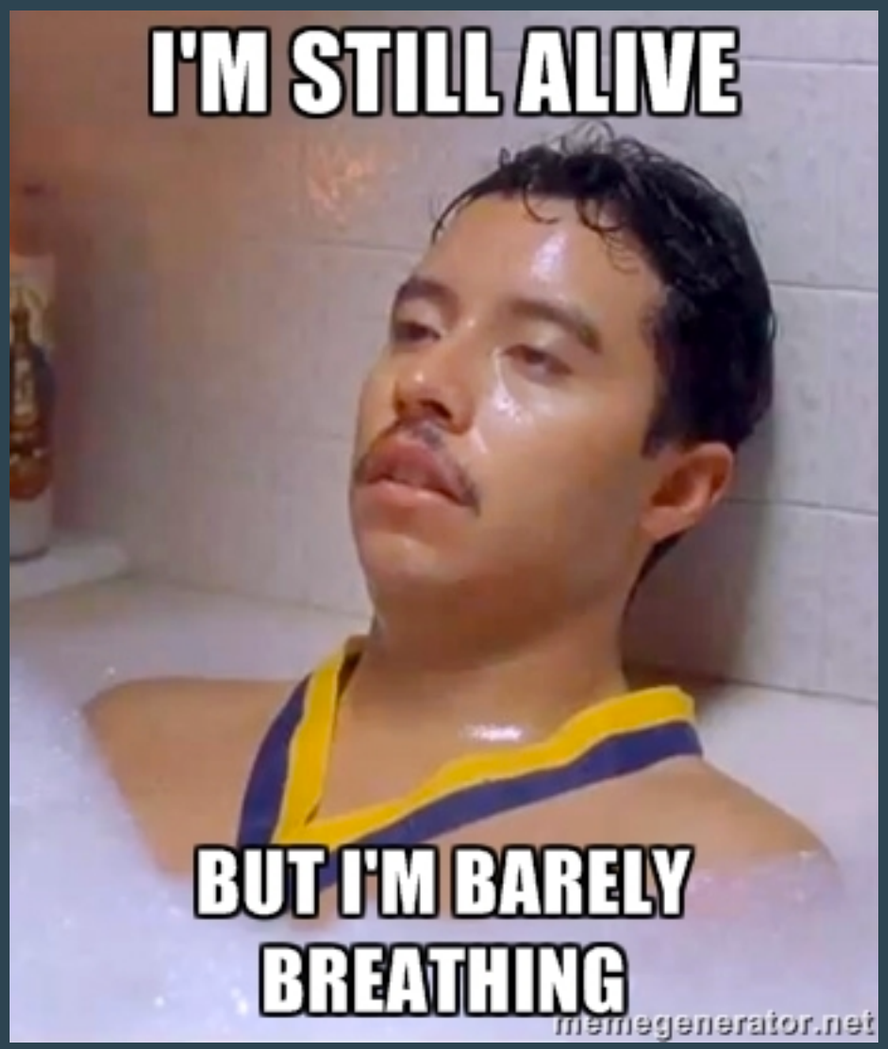
```
fun foo(a: Int, f: () -> Int): Int {
    return a + f()
}

foo(45, { return 56 })

// If last param is a function, it could be passed
like this
foo(45) { return 56 }

fun g() = 56
foo(45, ::g)
```

Still following?



Classes - Constructors

```
class User constructor(name: String) {

}

// OR if there is no annotation or visibility keyword
class User(name: String) {

}

// OR if a single line is not enough

class User(name: String) {
    init() {
        // Do your stuff over there
    }
}
```

Classes - Constructors

```
// Fast init
class User(val name: String, var age: Int) {
    private val name: String
    private var age: Int
}
```

Classes - Constructors

```
// Secondary constructor
class User(val name: String) {
    constructor(name: String, age: Int):
this(name) {
    }
}

// Every class can have:
//  - a unique primary constructor
//  - multiple secondary constructors
```

Classes - Properties

```
class A(var name: String) {
    public var name: String
}

val a = A("Ben")
println(a.name)
a.name = "Bob"
```

Classes - Properties

```
class MyList {
    // There is no field in Kotlin, only
properties
    var size = 0
        public get
        private set
    public var isEmpty: Boolean = true
        get() = this.size == 0
}
```

Classes - Properties

```
class A {
    // Lazy var
    lateinit var foo = "bar"
}
```

Classes - Methods

```
class A {
    fun foo(): String {
        return "foo"
    }

    fun bar() = "bar"

    fun foobar(a: Int, b: Int): Int {
        return a * b / 4
    }
}

val a = A()
println(a.foo())
println(a.bar())
println(a.foobar(8, 2))
```

Data Classes

```
// Only for holding data
// Includes
//  - getters/setters
//  - equals
//  - hashCode
//  - toString
//  - copy
data class User(val name: String = "", val age: Int =
0)

val jack = User(name = "Jack", age = 35)
val olderJack = jack.copy(age = 56)
val (name, age) = jack // Destruction allowed
```

Enough for tonight!

¿Preguntas?