




Kotlin

By: Kenny, Daniel, and Parth

Language Overview

- Designed by JetBrains on July 22, 2011
- 15th most popular language from Github public repos
- Great general purpose language with an emphasis on Android app development



# Ranking	Programming Language	Percentage (Change)	Trend
1	JavaScript	18.772% (-1.494%)	
2	Python	16.488% (-1.089%)	
3	Java	11.546% (+1.369%)	
4	Go	8.134% (-0.153%)	
5	C++	7.000% (+0.143%)	
6	Ruby	6.948% (+0.146%)	
7	TypeScript	6.655% (+0.406%)	
8	PHP	5.574% (+0.295%)	
9	C#	3.673% (+0.044%)	
10	C	3.127% (+0.175%)	
11	Scala	2.042% (+0.389%)	^
12	Shell	2.031% (+0.052%)	v
13	Dart	1.082% (+0.337%)	^
14	Rust	0.898% (-0.038%)	v
15	Kotlin	0.751% (+0.029%)	^
16	Swift	0.620% (-0.196%)	v
17	Groovy	0.381% (+0.001%)	^
18	Objective-C	0.378% (-0.120%)	v
19	Elixir	0.347% (-0.020%)	^
20	DM	0.302% (-0.114%)	v

Language Overview

- Designed to operate alongside Java
 - Main advantage over Java is that it is more concise
- Is an object oriented and functional language

```
StringBuilder sb = new StringBuilder();
```

Becomes in Kotlin

```
val sb = StringBuilder()
```

How far we got

- Fully implemented
- Not fully tested due to clunky formatting for concrete syntax

```
(define p1 '{local
  [{z = {+ 9 16}}
   {y = 25}]
 in
  {+ z y}))
```

```
val p1 = arrayOf<Any>("+", 9.0, 16.0)
val p2 = arrayOf<Any>("z", "=", p1)
val p3 = arrayOf<Any>("y", "=", 25.0)
val p4 = arrayOf<Any>("+", "z", "y")
val p5 = arrayOf<Any>(p2, p3)
val p6 = arrayOf<Any>("local", p5, "in", p4)
```

Interp Basic Tests

```
assertEquals(interp(NumC(n: 3.1), Env(ArrayList(topEnv.b))), NumV(n: 3.1))
// 3 + 4
assertEquals(interp(AppC (IdC (s: "+"), arrayListOf(NumC (n: 3.0), NumC (n: 4.0))), Env(ArrayList(topEnv.b))), NumV(n: 7.0))
// 3 - 4
assertEquals(interp(AppC (IdC (s: "-"), arrayListOf(NumC (n: 3.0), NumC (n: 4.0))), Env(ArrayList(topEnv.b))), NumV(n: -1.0))
// 3 * 4
assertEquals(interp(AppC (IdC (s: "*"), arrayListOf(NumC (n: 3.0), NumC (n: 4.0))), Env(ArrayList(topEnv.b))), NumV(n: 12.0))
// 3 / 4
assertEquals(interp(AppC (IdC (s: "/"), arrayListOf(NumC (n: 3.0), NumC (n: 4.0))), Env(ArrayList(topEnv.b))), NumV(n: .75))
// 3 <= 4
assertEquals(interp(AppC (IdC (s: "<="), arrayListOf(NumC (n: 3.0), NumC (n: 4.0))), Env(ArrayList(topEnv.b))), BoolV(b: true))
// 4 <= 3
assertEquals(interp(AppC (IdC (s: "<="), arrayListOf(NumC (n: 4.0), NumC (n: 3.0))), Env(ArrayList(topEnv.b))), BoolV(b: false))
// Ray equal? Ray
assertEquals(interp(AppC (IdC (s: "equal?"), arrayListOf(StrC (s: "Ray"), StrC (s: "Ray"))), Env(ArrayList(topEnv.b))), BoolV(b: true))
// Ray equal? ray
assertEquals(interp(AppC (IdC (s: "equal?"), arrayListOf(StrC (s: "Ray"), StrC (s: "ray"))), Env(ArrayList(topEnv.b))), BoolV(b: false))
// (substring Parth 3 4)
assertEquals(interp(AppC (IdC (s: "substring"), arrayListOf(StrC(s: "Parth"), NumC (n: 3.0), NumC (n: 4.0))), Env(ArrayList(topEnv.b))), StrV(s: "t"))
```

More Involved Tests

```
// '{local {[x = 5]} in {+ x 5}}
val t1 = AppC(LamC(arrayListOf<String>("x"),
    AppC(IdC(s: "+"), arrayListOf<ExprC>(IdC(s: "x"), NumC(n: 5.0)))),
    arrayListOf<ExprC>(NumC(n: 5.0)))

// '{lam {x y} {+ x y} 2 3}
val t2 = AppC(LamC(arrayListOf<String>("x", "z"),
    AppC(IdC(s: "+"),
        arrayListOf<ExprC>(IdC(s: "x"), IdC(s: "z")))),
    arrayListOf<ExprC>(NumC(n: 2.0), NumC(n: 3.0)))

// '{local
//   [{a = {lam {x y} {+ x y}}}]
//   [{b = {lam {z} {equal? z 4}}}]
//   in
//   {if {b {a 4 5}} "succ" "fail"}}
val t3 = AppC(LamC(arrayListOf<String>("a", "b"), IfC(AppC(IdC(s: "b"),
    arrayListOf<ExprC>(AppC(IdC(s: "a"), arrayListOf<ExprC>(NumC(n: 4.0), NumC(n: 5.0)))))),
    StrC(s: "succ"), StrC(s: "fail"))),
    arrayListOf<ExprC>(LamC(arrayListOf<String>("x", "y"), AppC(IdC(s: "+"),
        arrayListOf<ExprC>(IdC(s: "x"), IdC(s: "y")))),
        LamC(arrayListOf<String>("z"), AppC(IdC(s: "equal?"), arrayListOf<ExprC>(IdC(s: "z"), NumC(n: 4.0))))))

assertEquals(interp(t1, Env(ArrayList(topEnv.b))), NumV(n: 10.0))
assertEquals(interp(t2, Env(ArrayList(topEnv.b))), NumV(n: 5.0))
assertEquals(interp(t3, Env(ArrayList(topEnv.b))), StrV(s: "fail"))
```

Parse Tests

```
fun mainTest() {  
    val p1 = arrayOf<Any>("+", 9.0, 16.0)  
    val p2 = arrayOf<Any>("z", "=", p1)  
    val p3 = arrayOf<Any>("y", "=", 25.0)  
    val p4 = arrayOf<Any>("+", "z", "y")  
    val p5 = arrayOf<Any>(p2, p3)  
    val p6 = arrayOf<Any>("local", p5, "in", p4)  
  
    val a1 = arrayOf<Any>("if", arrayOf<Any>("<=", 3.0, 5.0), "\"Hi", 3.0)  
  
    assertEquals(main(a1), actual: "\"Hi")  
    assertEquals(main(p6), actual: "50.0")  
}
```

ExprC Definition

```
//===== Data Definitions =====  
//ExprC definition  
open class ExprC()  
class NumC(val n: Double) : ExprC() {  
    override fun equals(other: Any?)  
        = (other is NumC)  
        && this.n == other.n  
}  
class IdC(val s: String) : ExprC() {  
    override fun equals(other: Any?)  
        = (other is IdC)  
        && this.s == other.s  
}  
class StrC(val s: String) : ExprC() {  
    override fun equals(other: Any?)  
        = (other is StrC)  
        && this.s == other.s  
}  
class IfC(val test: ExprC, val thn: ExprC, val els: ExprC) : ExprC() {  
    override fun equals(other: Any?)  
        = (other is IfC)  
        && this.test == other.test  
        && this.thn == other.thn  
        && this.els == other.els  
}  
class LamC(val args: ArrayList<String>, val body: ExprC) : ExprC() {  
    override fun equals(other: Any?)  
        = (other is LamC)  
        && this.args == other.args  
        && this.body == other.body
```

```
}  
class LamC(val args: ArrayList<String>, val body: ExprC) : ExprC() {  
    override fun equals(other: Any?)  
        = (other is LamC)  
        && this.args == other.args  
        && this.body == other.body  
}  
class AppC(val name: ExprC, val args: List<ExprC>) : ExprC() {  
    override fun equals(other: Any?)  
        = (other is AppC)  
        && this.name == other.name  
        && this.args == this.args  
}  
  
//Binding  
class Binding(val name: String, val the_value: Value) {  
    override fun equals(other: Any?)  
        = (other is Binding)  
        && this.name == other.name  
        && this.the_value == other.the_value  
}  
  
//Env  
class Env(val b: ArrayList<Binding>) {  
    override fun equals(other: Any?)  
        = (other is Env)  
        && this.b == other.b  
}
```


Value Definition

```
// Value definition
open class Value
class NumV(val n: Double) : Value() {
    override fun equals(other: Any?)
        = (other is NumV)
        && this.n == other.n
    override fun toString(): String
        = this.n.toString()
}

class BoolV(val b: Boolean) : Value() {
    override fun equals(other: Any?)
        = (other is BoolV)
        && this.b == other.b
}

class StrV(val s: String) : Value() {
    override fun equals(other: Any?)
        = (other is StrV)
        && this.s == other.s
}

class CloV(val param: ArrayList<String>, val body: ExprC, val env: Env) : Value() {
    override fun equals(other: Any?)
        = (other is CloV)
        && this.param == other.param
        && this.body == other.body
        && this.env == other.env
}

class PrimV(val p: String) : Value() {
    override fun equals(other: Any?)
        = (other is PrimV)
        && this.p == other.p
}
```

Interp

```
// interp: Interprets the given expression, using the list of funcs to resolve applications.
fun interp(exp: ExprC, env: Env): Value {
    return when(exp) {
        is NumC -> NumV(exp.n)
        is IdC -> (envLookup(exp.s, env))
        is StrC -> StrV(exp.s)
        is LamC -> CloV(exp.args, exp.body, env)
        is IfC -> when(val test = interp(exp.test, env)) {
            is BoolV -> if (test.b) interp(exp.thn, env)
                       else interp(exp.els, env)
            else -> throw Exception("If test not evaluating to a boolean $exp")
        }
        is AppC -> when(val app = interp(exp.name, env)) {
            is CloV -> if ((app.param).size == (exp.args).size) {
                val argval = (exp.args).map {arg -> interp(arg, env)};
                val newEnv = makeBindings(app.param, (argval as ArrayList<Value>), app.env);
                interp(app.body, newEnv);
            } else {
                throw Exception("Args wrong arity $app.param")
            }
            is PrimV -> {
                val v = (exp.args).map {arg -> interp(arg, env)};
                primInterp(app, (v as ArrayList<Value>));
            }
            else -> throw Exception("Not a function $exp.name")
        }
    }
    else -> throw Exception("Interp did not receive an ExprC $exp")
}
```

Parse

```
// parse: takes in an Sexp, returns an ExprC of the proper form
fun parse(p: Array<Any>): ExprC {
    return when(p.size) {
        1 -> when(val s = p[0]) {
            is Double -> NumC(s)
            is String -> if(s[0] == '"') {
                StrC(s)
            } else {
                if(s in reserved) throw Exception("Invalid id $s") else IdC(s)
            }
        }
        else -> throw Exception("Invalid singleton parse $s")
    }
}

else -> when(p[0]) {
    "if" -> IfC(parse(prepare(p[1])), parse(prepare(p[2])), parse(prepare(p[3])))
    "local" -> {val (s, exp) = localHelper(p[1] as Array<Any>);
        AppC(LamC(s, parse(prepare(p[3]))), exp.map{e -> parse(prepare(e))});
    }
    "lam" -> {lamHelper(p[1] as Array<Any>);
        LamC(p[1] as ArrayList<String>, parse(prepare(p[2])))
    }
    else -> AppC(parse(prepare(p[0])), ((p.drop(1)).map{ arg -> parse(prepare(arg))}))
}
}
```

Values in Kotlin

- Values: Objects, Closures, Strings, Numbers, Boolean
- Tries to avoid Null values
 - nullable and non-nullable references

Language Syntax

- Program entry point is in main function
- Lexical Scoping with visibility
- Body of function can be set with = rather than {} with single expression
- Range syntax (x in 1..5)
- When: similar to switch statements

```
// interp: Interprets the given expression, using the list of funcs to resolve applications.
fun interp(exp: ExprC, env: Env): Value {
    return when(exp) {
        is NumC -> NumV(exp.n)
        is IdC -> (envLookup(exp.s, env))
        is StrC -> StrV(exp.s)
        is LamC -> CloV(exp.args, exp.body, env)
        is IfC -> when(val test = interp(exp.test, env)) {
            is BoolV -> if (test.b) interp(exp.thn, env)
                       else interp(exp.els, env)
            else -> throw Exception("If test not evaluating to a boolean $exp")
        }
        is AppC -> when(val app = interp(exp.name, env)) {
            is CloV -> if ((app.param).size == (exp.args).size) {
                val argval = (exp.args).map {arg -> interp(arg, env)};
                val newEnv = makeBindings(app.param, (argval as ArrayList<Value>), app.env);
                interp(app.body, newEnv);
            } else {
                throw Exception("Args wrong arity $app.param")
            }
            is PrimV -> {
                val v = (exp.args).map {arg -> interp(arg, env)};
                primInterp(app, (v as ArrayList<Value>));
            }
            else -> throw Exception("Not a function $exp.name")
        }
    }
    else -> throw Exception("$interp did not receive an ExprC $exp")
}
```

Type System

- Nullable types
- Type inference
- Subtyping
- Unreachable code analysis
- Same types as Java
- Nullable Any? Is supertype of all types
- Cast with "as" keyword

```
+  
var a: String = "abc" // Regular initialization means non-null by default  
a = null // compilation error  
  
Null can not be a value of a non-null type String
```

Target platform: JVM Running on kotlin v1.4.30

To allow nulls, you can declare a variable as nullable string, written `String?` :

```
+  
var b: String? = "abc" // can be set null  
b = null // ok  
print(b)  
  
null
```

Target platform: JVM Running on kotlin v1.4.30

Would we take a job in Kotlin?

- Yes
- It's cleaner and more concise than Java
- Can convert Java files into Kotlin and vice versa
- One of the main languages used for Android development

Thank you!