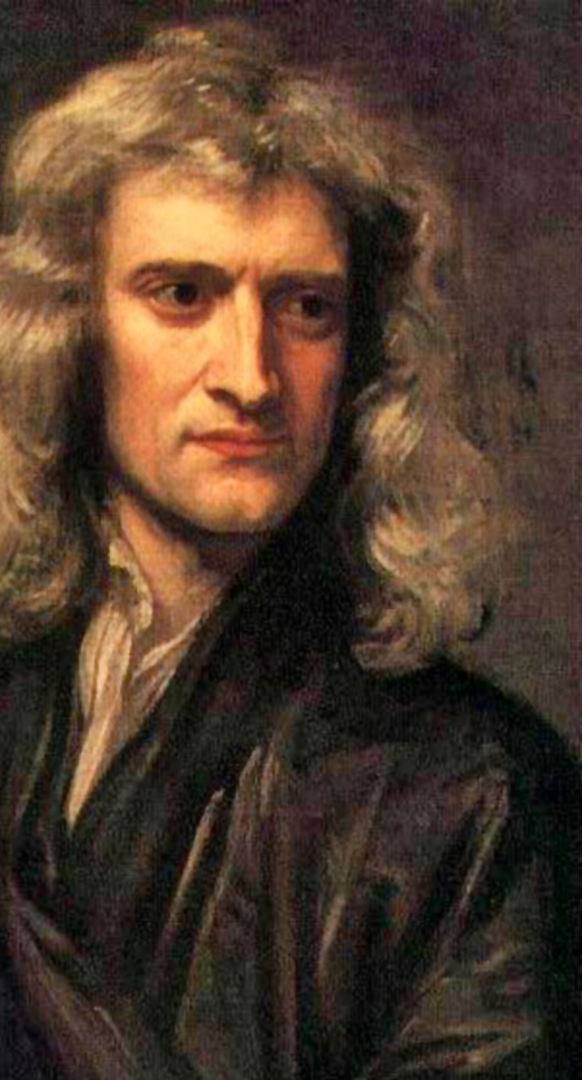
Shoulders of Giants

Andrey Breslav



Sir Isaac Newton

If I have seen further it is by standing on the shoulders of Giants

Letter to Robert Hooke, 1675

Bernard of Charters

[Dicebat Bernardus Carnotensis nos esse quasi]

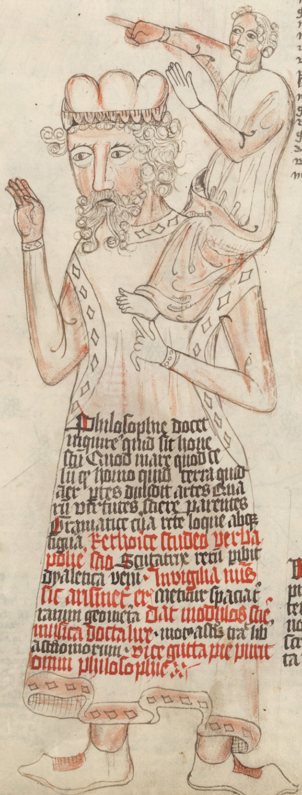
nanos gigantium humeris incidentes.

XII c.

Ysiderus spricht yn xv puch Ethymologiarum das daz fleisch von vier elementen zw sammen gemacht ist et cetera.

Als die sel in irem spiegel vnd in irem begyn drivaltig vnd ein yn wer tugent vnd yn wer macht dem gotez pilde zw geaynet ist yn ainem liecht der chunst vnd yn ainer gestalt der heiligen drivaltichait die sich dreyerlay beweist ein yegleicher philosophus ist naturleich vnd ist redleich vnd ist tadleich zw dem ersten so spricht er dez wesens daz selb weist er in die sach macht dez vaters Czw dem andern so spricht er von dem vnnemens redleichait daz sich weist in die weyshait dez suns Daz dritt beweisset vns die demut dez lebens daz vns weiset yn die gute dez heiligen geist Daz tailt sich yn die chunst die do haisset methaphysica mathematica et physica zw dem erst so spricht er von den dingen des zw dem andern so spricht er von der zal vn der figur Czw dem von der natur vn von der tugent vnd von der eyngiessenten machung darumb so weist er ein daz erst wegunm dez vaters Daz ander in dem pild dez suns Daz dritt yn der gab dez heiligen geist Czw dem andern tail er sich yn die chunst grammaticaz die da macht gewaltig auff leger der chunst die da haist loyca rethorica Die do stent ist yn yrm wesen als dye heilig drivaltichait Czw dem dritten daz tail er yn die chunst die do haisset manastica ycomomica vnd politica zw dem ersten dez vaters stettichait zw dem andern dez suns aimslichait zw dem dritten dez heiligen geist freylichait Alle die chunst habent ir regel gewislich vnd vntrugenlich Recht als der man vnd daz gestirn gien von de ewigen ee yn vnser gemut darumb vnser gemut vnd vns sin sulches leuchtes wirt erfult vnd ein gegossen nur ez gar blint ez mag per im selben begreiffen daz ewig liecht etc.

Gregorius in dem zehente puche moralium spricht allez daz do ist daz man zw dem menschen ayfen Daz mensch ist der hymel wan er mit wegerung anhengent dem obristen dingen vnd auch ist er die helle wenn er mit seyner chobung sicht selbn betrubt mit den vndristen vinster nussn Er ist auch der ertreich daz do mit gutten werken mit guter hoffnung frucht pringet Er ist daz mer der do mettlicher sachen porinet vnd daz wett mit seiner vnstet ragende ist Als augustinus spricht in dem xv puche von der stat Plumus der ander vnd gelertist mensch lemer der mesch der werlt lauff vnd leibez natur nisgent ist ye mer daz fleische des menschen genaygetist zw der werlt



Nostri maiores vita studioque priores corpore immentes forz tenere gigantis Pergunt veros nos ipsos equiparamur Et perscrutamur sensu suo dogmata vero.

Why I’m giving this talk

The right question

Why don’t all language designers give such talks?

Every language has learned from others

Is it morally wrong? Or maybe even illegal?

“In science and in art, there are, and can be, few, if any things, which in an abstract sense are strictly new and original throughout. Every book in literature, science, and art borrows, and must necessarily borrow, and use much which was well known and used before.”

Supreme Court Justice David Souter
Campbell v. Acuff-Rose Music, Inc., 510 U.S. 569 (1994)

Does it make them worse?

- Is originality what really matters to you as a developer?
  - Put in on the scale with productivity and maintainability ☺

- But I want to
  - … stand out
  - … be proud of my language (and its authors)
  - ... win arguments with fans of other languages
  - … write awesome software, maybe? ☺

What do language designers think?

- The Kotlin team had productive discussions with
  - Brian Goetz
  - Martin Odersky
  - Erik Meijer
  - Chris Lattner
  - … looking forward to having more!

My ideal world

- Everyone builds on top of others' ideas
- Everyone openly admits it and says thank you
- Difference from academia: it's OK to not look at some prior work ☺

So, what languages has Kotlin learned from?

From Java: classes!

- One superclass
- Multiple superinterfaces
- No state in interfaces
- Constructors (no destructors)
- Nested/inner classes
- Annotations
- Method overloading

- One root class
- toString/equals/hashCode
- Autoboxing
- Erased generics
- Runtime safety guarantees

Leaving things out: Java

- Monitor in every object
- finalize()
- Covariant arrays
- Everything implicitly nullable
- Primitive types are not classes
- Mutable collection interfaces
- Static vs instance methods
- Raw types
- …

From Java…

```
class C : BaseClass, I1, I2 {

    class Nested { ... }

    inner class Inner { ... }

}
```

From Scala, C#...

```
class C(p: Int, val pp: I1) : B(p, pp) {

    internal var foo: Int
        set(v) { field = check(v) }

    override fun toString() = "..."
}
```

Not Kotlin

```
class Person {
    private var _name: String
    def name = _name
    def name_=(aName: String) { _name = aName }
 }
```

Not Kotlin

```
class Person {
    private string _Name;
    public string Name {
        get { return _Name; }
        set { _Name = value }
    }
}
```

Kotlin

```
class Person {
    private var _name = "..."
    var name: String
        get() = _name
        set(v) { _name = v }
}
```

Leaving things out: Interfaces vs Scala Traits

- No state allowed
  - no fields
  - no constructors
- Order doesn’t matter
  - no linearization rules
- Delegation through **by**

Some (more) differences…

```
class C(d: Intf) : Intf by d {

    init { println(d) }

    fun def(x: Int = 1) { ... }

    fun test() {
        def(x = 2)
    }
}
```

And a bit more…

```
val person = Person("Jane", "Doe")

val anonymous = object : Base() {
    override fun foo() { ... }
}
```

From Scala…

```
object Singleton : Base(foo) {
    val bar = ...

    fun baz(): String { ... }
}

Singleton.baz()
```

Companion objects

```
class Data private (val x: Int) {
    companion object {
        fun create(x) = Data(x)
    }
}

val data = Data.create(10)
```

Companion objects

```
class Data private (val x: Int) {
    companion object {
        @JvmStatic
        fun create(x) = Data(x)
    }
}

val data = Data.create(10)
```

BTW: Not so great ideas

- Companion objects ☹
- Inheritance by delegation ☹

From C#...

```
class Person(
    val firstName: String,
    val lastName: String
)

fun Person.fullName() {
  return "$firstName $lastName"
}
```

Not Kotlin

```
implicit class RichPerson(p: Person) {
    def fullName = s"${p.firstName} ${p.lastName}"
}

class PersonUtil {
    public static string FullName(this Person p) {
        return $"{p.firstName} {p.lastName}"
    }
}
```

It’s Groovy time!

```
mylist
    .filter { it.foo }
    .map { it.bar }
```

Not Kotlin

```
mylist.filter(_.foo).map(_.bar)
```

```
mylist.stream()
  .filter((it) -> it.getFoo())
  .map((it) -> it.getBar())
  .collect(Collectors.toList())
```

```
mylist.Where(it => it.foo).Select(it => it.bar)
```

Not Kotlin (yet…)

```
for (it <- mylist if it.foo)
    yield it.bar
```

```
from it in mylist where it.foo select it.bar
```

```
[it.bar for it in mylist if it.foo]
```

It’s Groovy time!

```
val foo = new Foo()
foo.bar()
foo.baz()
foo.qux()
```

```
with(foo) {
    bar()
    baz()
    qux()
}
```

DSLs: Type-Safe Builders

```
a(href = "http://my.com") {
    img(src = "http://my.com/icon.png")
}
```

Not Kotlin ☺

```
10 PRINT "Welcome to Baysick Lunar Lander v0.9"
20 LET ('dist := 100)
30 LET ('v := 1)
40 LET ('fuel := 1000)
50 LET ('mass := 1000)

60 PRINT "You are drifting towards the moon."
```

Not Kotlin ☺

```
object Lunar extends Baysick {
  def main(args:Array[String]) = {
    10 PRINT "Welcome to Baysick Lunar Lander v0.9"
    20 LET ('dist := 100)
    30 LET ('v := 1)
    40 LET ('fuel := 1000)
    50 LET ('mass := 1000)

    60 PRINT "You are drifting towards the moon."
  }
}
```

From Scala: Data Classes

```
data class Person(
    val first: String,
    val last: String
) {
    override fun equals(other: Any?): Boolean
    override fun hashCode(): Int
    override fun toString(): String
    fun component1() = first
    fun component2() = last
    fun copy(first: String = this.first, last: String = this.last)
}

val (first, last) = myPerson
```

Not Kotlin

```
val greeting = x match {
  case Person(f, l) => s"Hi, $f $l!"
  case Pet(n) => s"Hi, $n!"
  _ => "Not so greetable O_o"
}
```

From Gosu: Smart casts

```
val greeting = when (x) {
  is Person -> "Hi, ${x.first} ${x.last}!"
  is Pet -> "Hi, ${x.name}!"
  else -> "Not so greetable o_O"
}
```

From Groovy, C#: Elvis and friends

```
val middle = p.middleName ?: "N/A"

persons.firstOrNull()?.firstName ?: "N/A"

nullable!!.foo

(foo as Person).bar
(foo as? Person)?.bar
```

Java/Scala/C#: Generics

```
abstract class Read<out T> {
    abstract fun read(): T
}

interface Write<in T> {
    fun write(t: T)
}
```

Java/Scala/C#: Generics

```
class RW<T>(var t: T): Read<T>, Write<T> {
    override fun read(): T = t
    override fun write(t: T) {
        this.t = t
    }
}

fun foo(from: RW<out T>, to: RW<in T>) {
    to.write(from.read())
}
```

Do other languages learn from Kotlin?

- I can't be 100% sure, but looks like they do
  - xTend
  - Hack
  - Swift
  - Java?
  - C#?

- But let them speak for themselves!

My ideal world

- Everyone builds on top of others' ideas
- Everyone openly admits it and says thank you
- Difference from academia: it's OK to not look at some prior work ☺

P.S. My Personal Workaround

In practice, languages are often
selected by passion, not reason.

Much as I'd like it to be the other way around,
I see no way of changing that.

So, I'm trying to make Kotlin a
language that is loved for a reason ☺