SPARK with Scala

Module 1 begins

What is Scala?

- Scala is an acronym for “**Scalable Language”**. This means that Scala grows with you. You can play with it by typing one-line expressions and observing the results.

- But you can also rely on it for large mission critical systems, as many companies, including Twitter, LinkedIn, or Intel do with advent of frameworks like PLAY, AKKA etc.

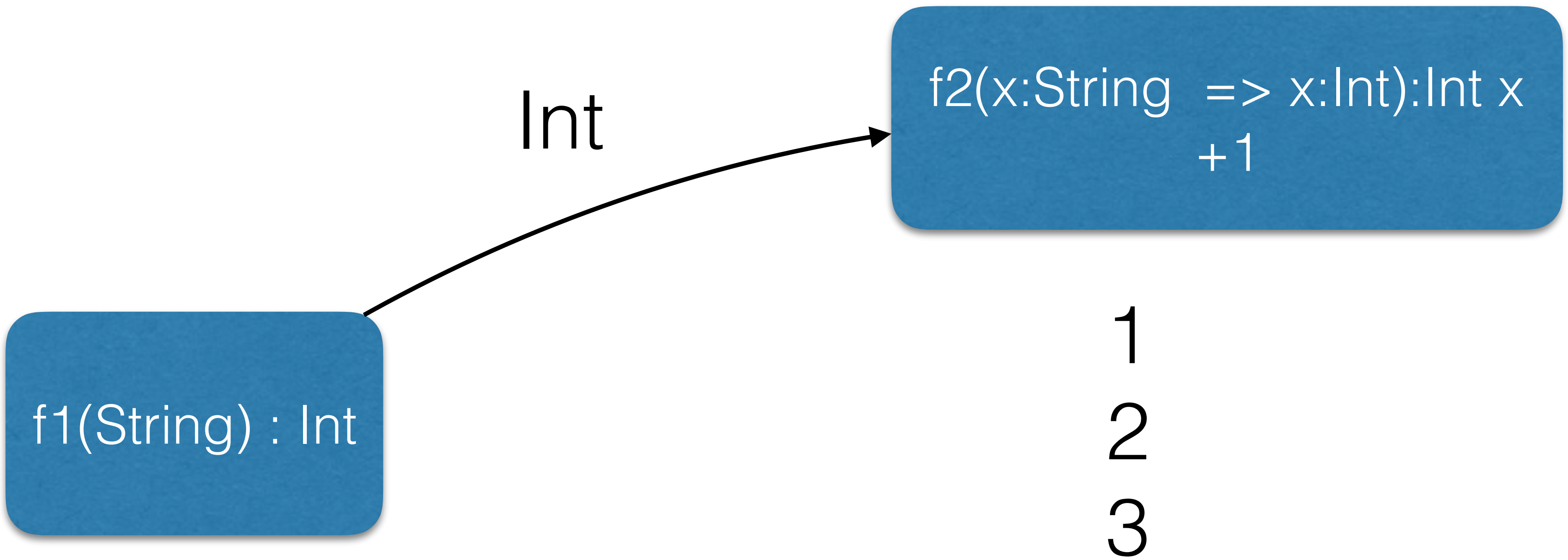
- It can act both as a scripting language as well as a regular objected oriented language like java.

- Scala is a **pure-bred object-oriented language**. Conceptually, **every value is an object** and **every operation is a method-call**. The language supports advanced component architectures through **classes** and **traits.**

- Many traditional design patterns in other languages are already natively supported.

- Even though its syntax is fairly conventional, Scala is also a **full-blown functional language**. It has everything you would expect, including first-class functions, a library **with efficient immutable data structures**, and **a general preference of immutability over mutation**.

- Scala **runs on the JVM**. **Java and Scala classes can be freely mixed**, no matter whether they reside in different projects or in the same.



Int

f1(String) : Int

f2(x:String => x:Int):Int x +1

1 - filter(from the text)
2 - (all words having vowels)
3 - (words count)

1
2
3
4
5

lambdas

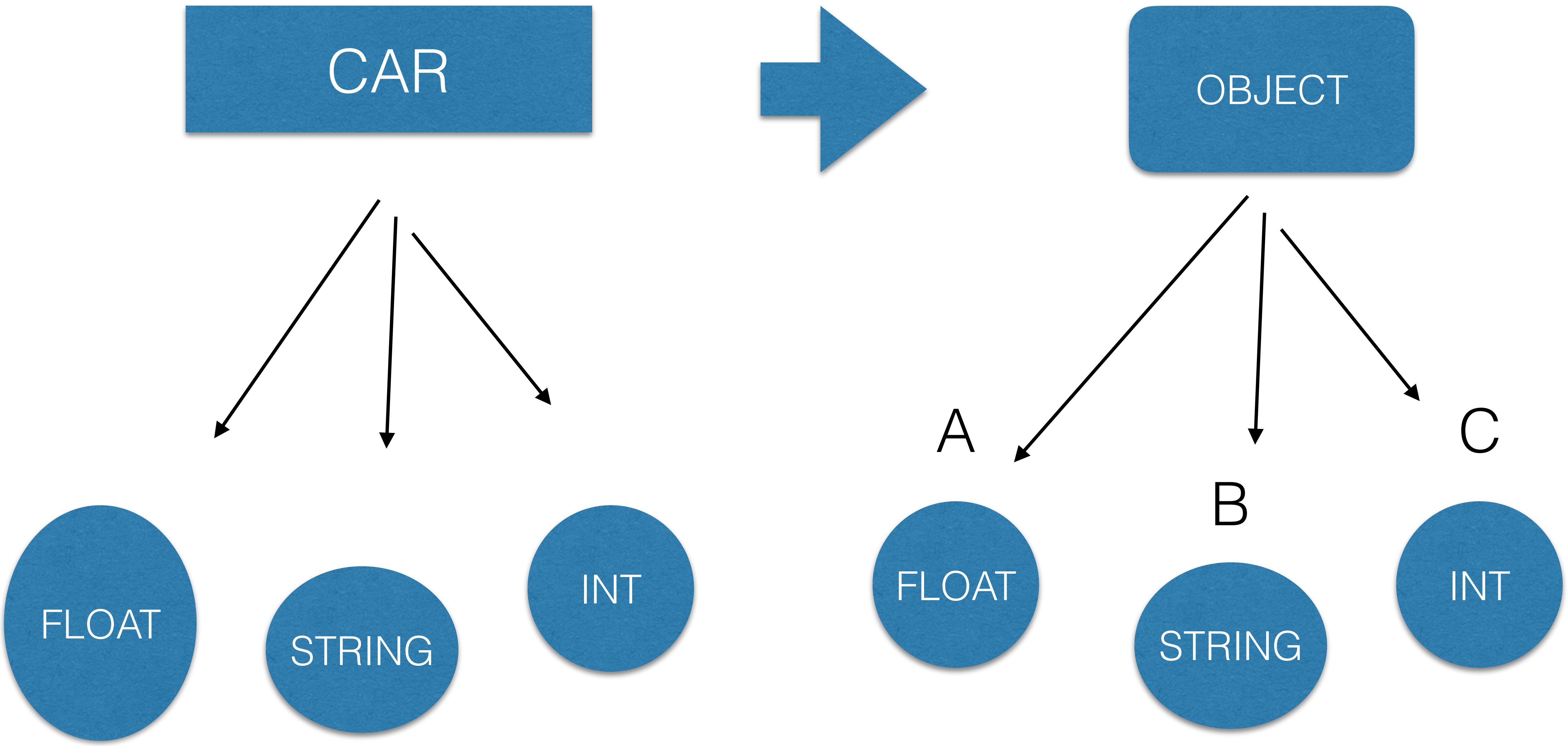
JAVA8 - Functional
Python
SCALA

Introduction to Scala for Spark

- Scala is a language which
  - Type Inferential
  - STATIC TYPING
  - TYPE SAFE: Throws errors at the time of compilation rather at run time unlike java
- others like:
  - PATTERN MATCHING,
  - CLEAN SYNTAX and API,
  - CURRYING,
  - PARTIAL FUNCTIONS and much more....
  - Tough to reach the ceiling: things become obscure for a programmer.

-gatives:

- --> SCALA is difficult, analogically it is similar to carrying the luggage on your shoudlers and climb, or learn to operate a crane which would do this job easier, but u need to invest time to learn operating crane which for some might be a difficult exercise. But this is worth an investment.



CAR

OBJECT

FLOAT

STRING

INT

A

B

C

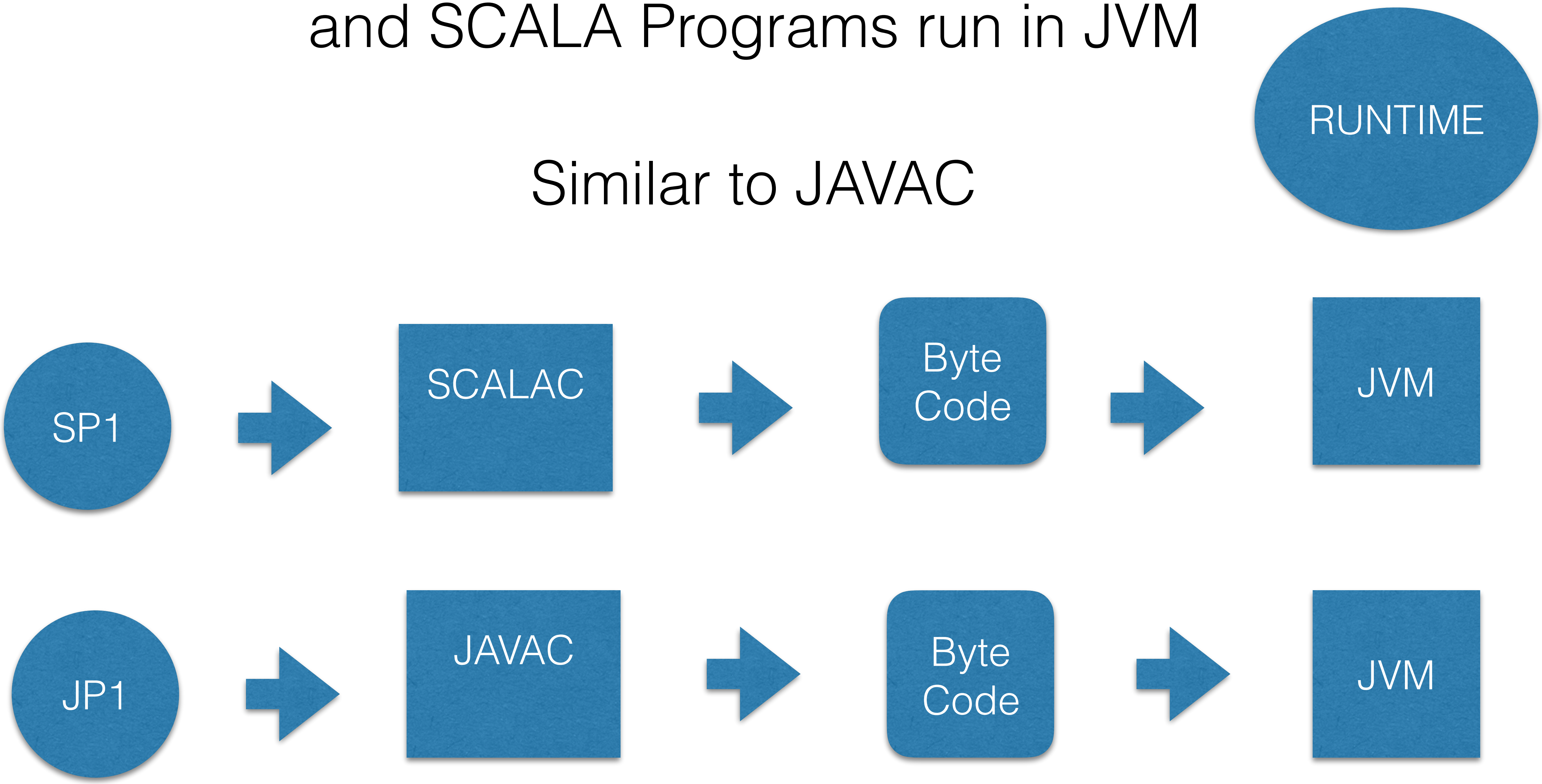
FLOAT

STRING

INT

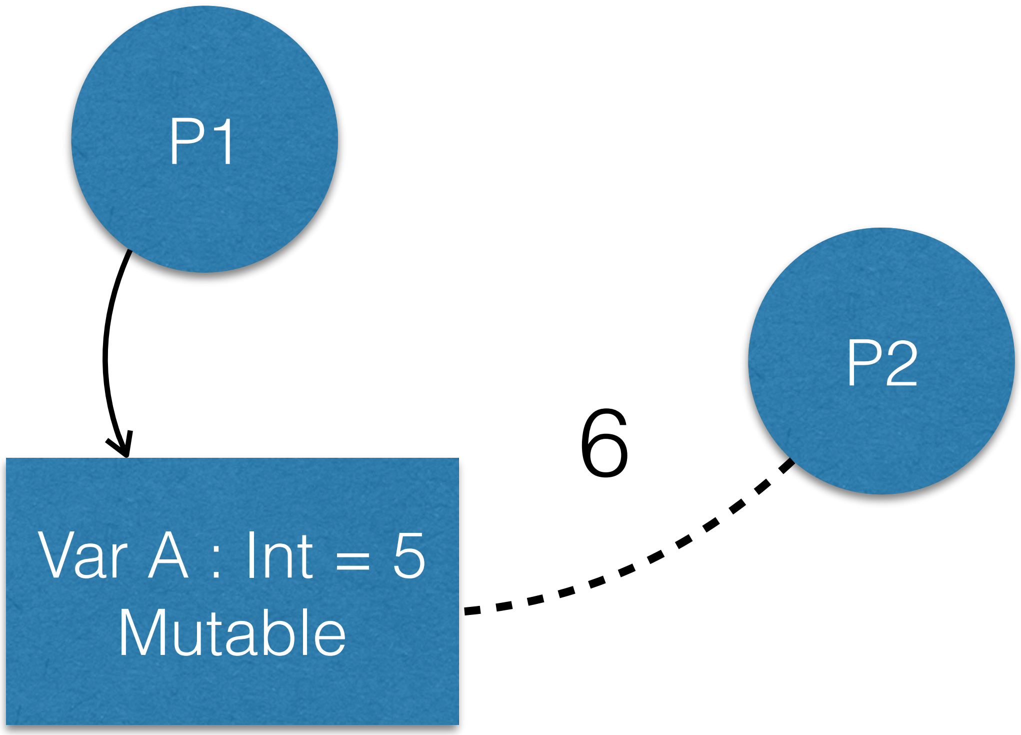
SCALA is Inherently JAVA
SCALAC is the scala compiler
SCALAC produces a BYTE CODE
and SCALA Programs run in JVM

Similar to JAVAC



SCALAC - Compiler

Converts human readable code into Byte Code



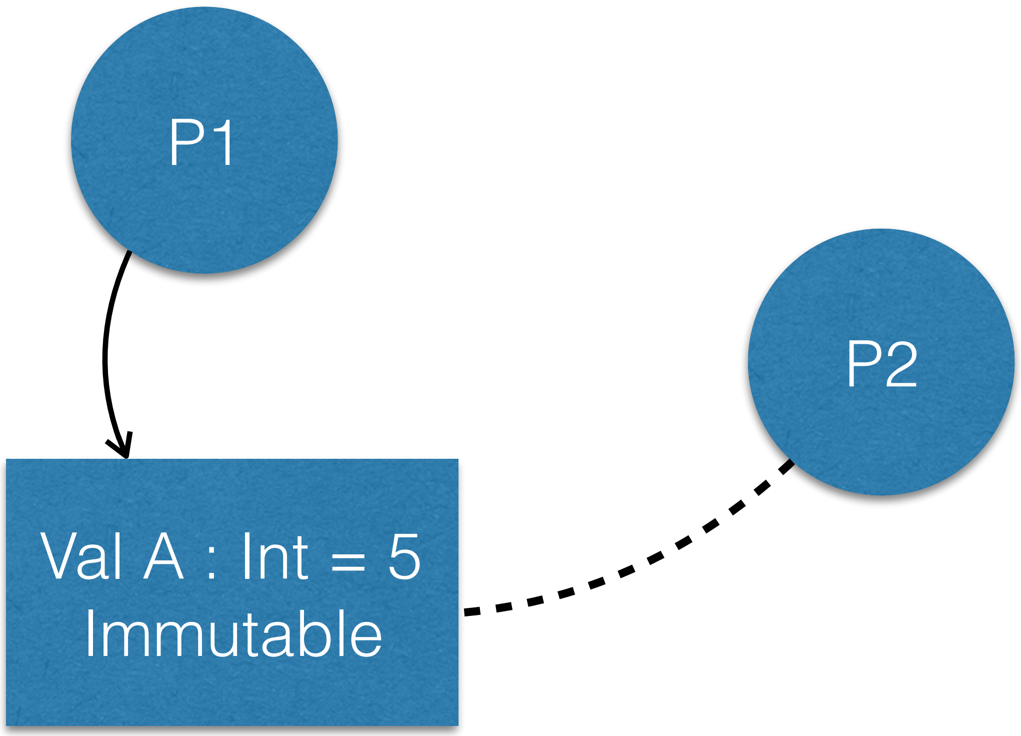
P1

P2

6

Var A : Int = 5
Mutable

Scala prefers Immutability



P1

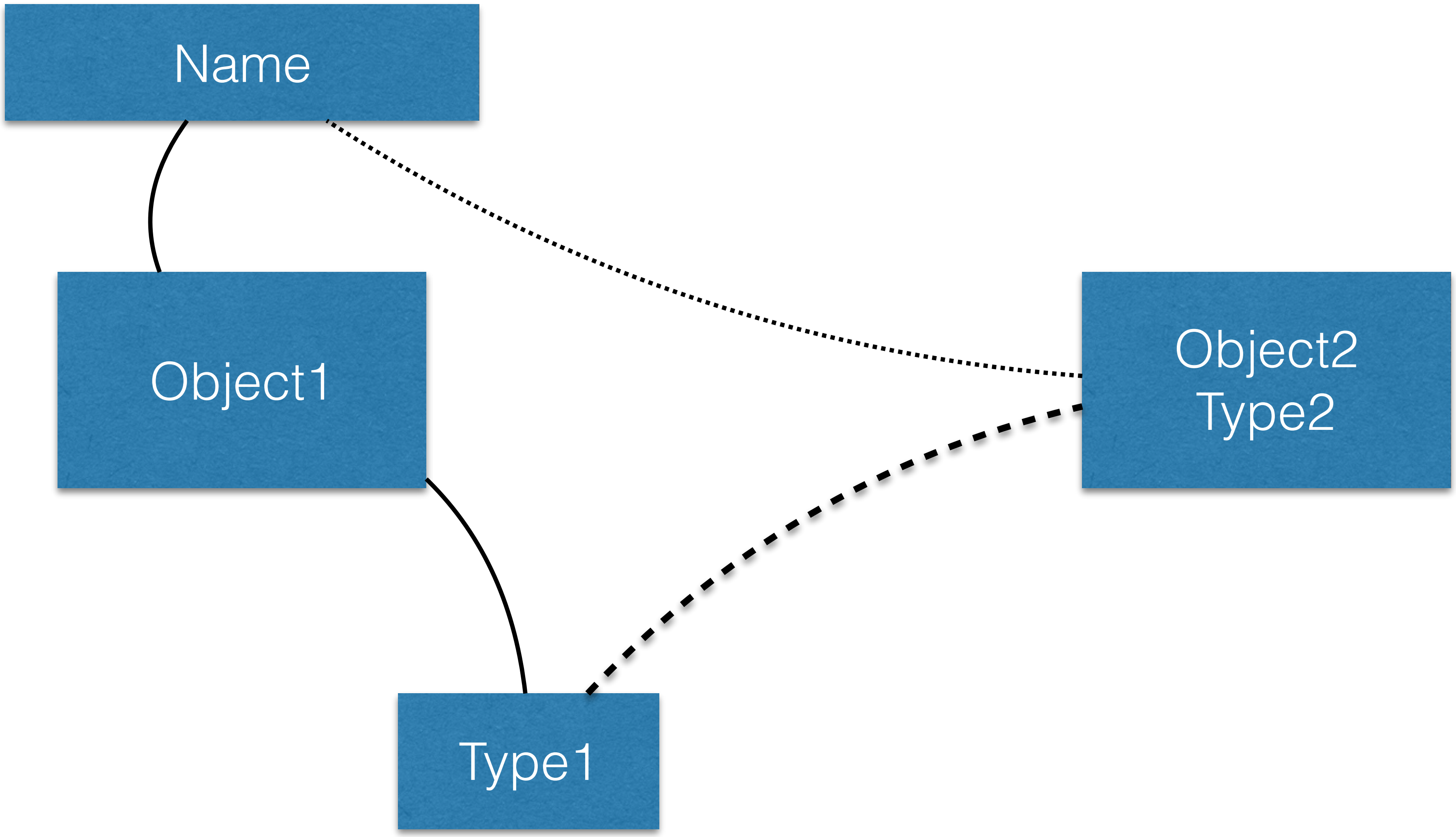
P2

Val A : Int = 5
Immutable

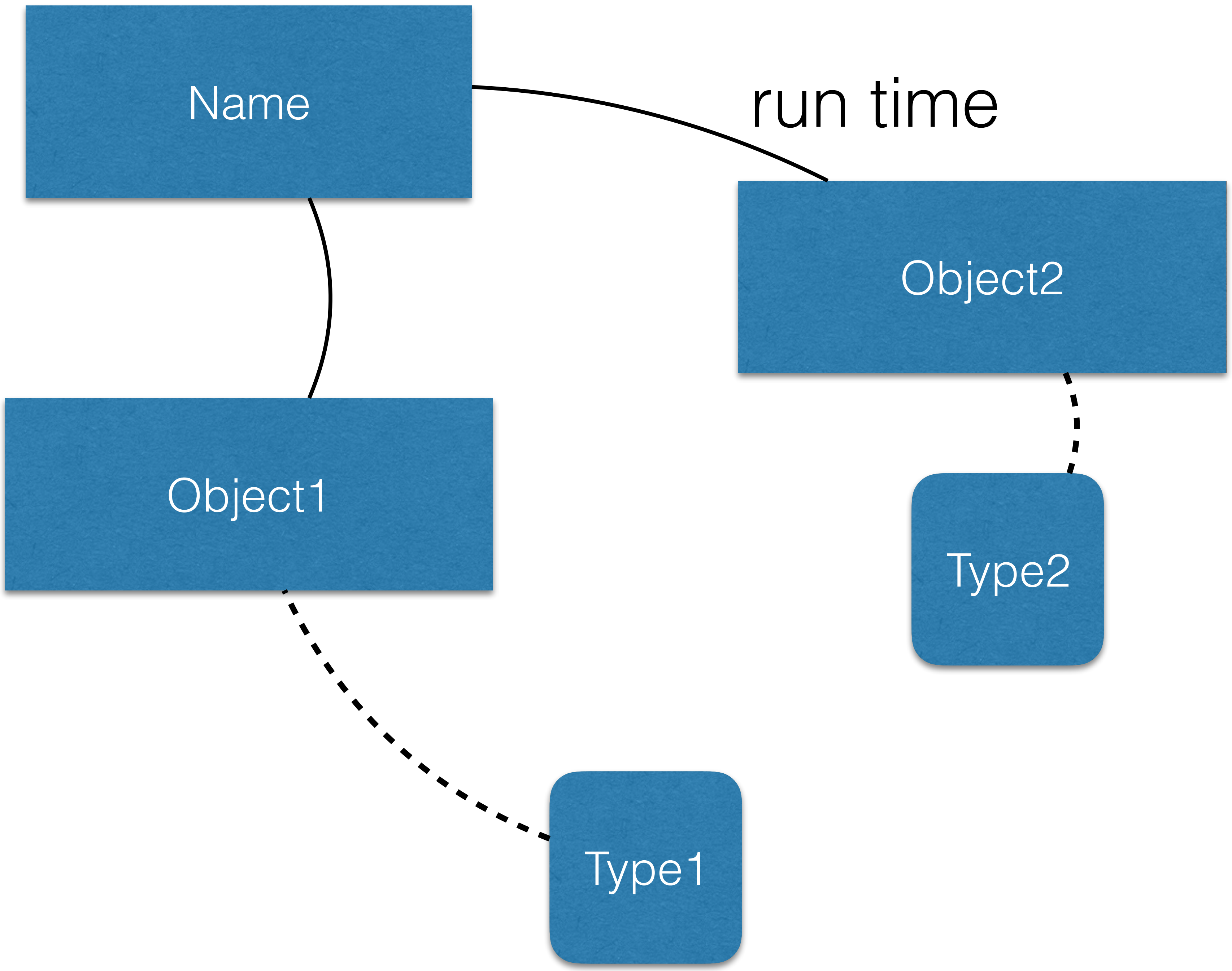
Types

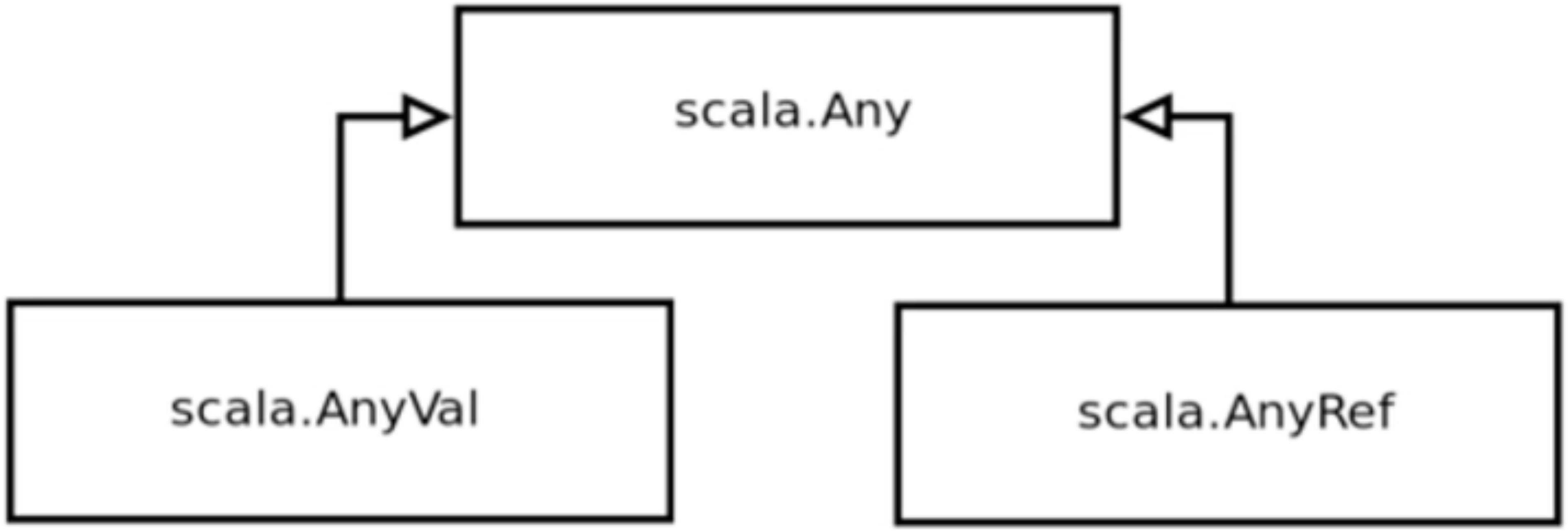
- Python, Ruby and Javascript - Dynamically Typed
- Scala, Java, C, C++ etc - Statically Typed

Static Typing



Dynamic Typed

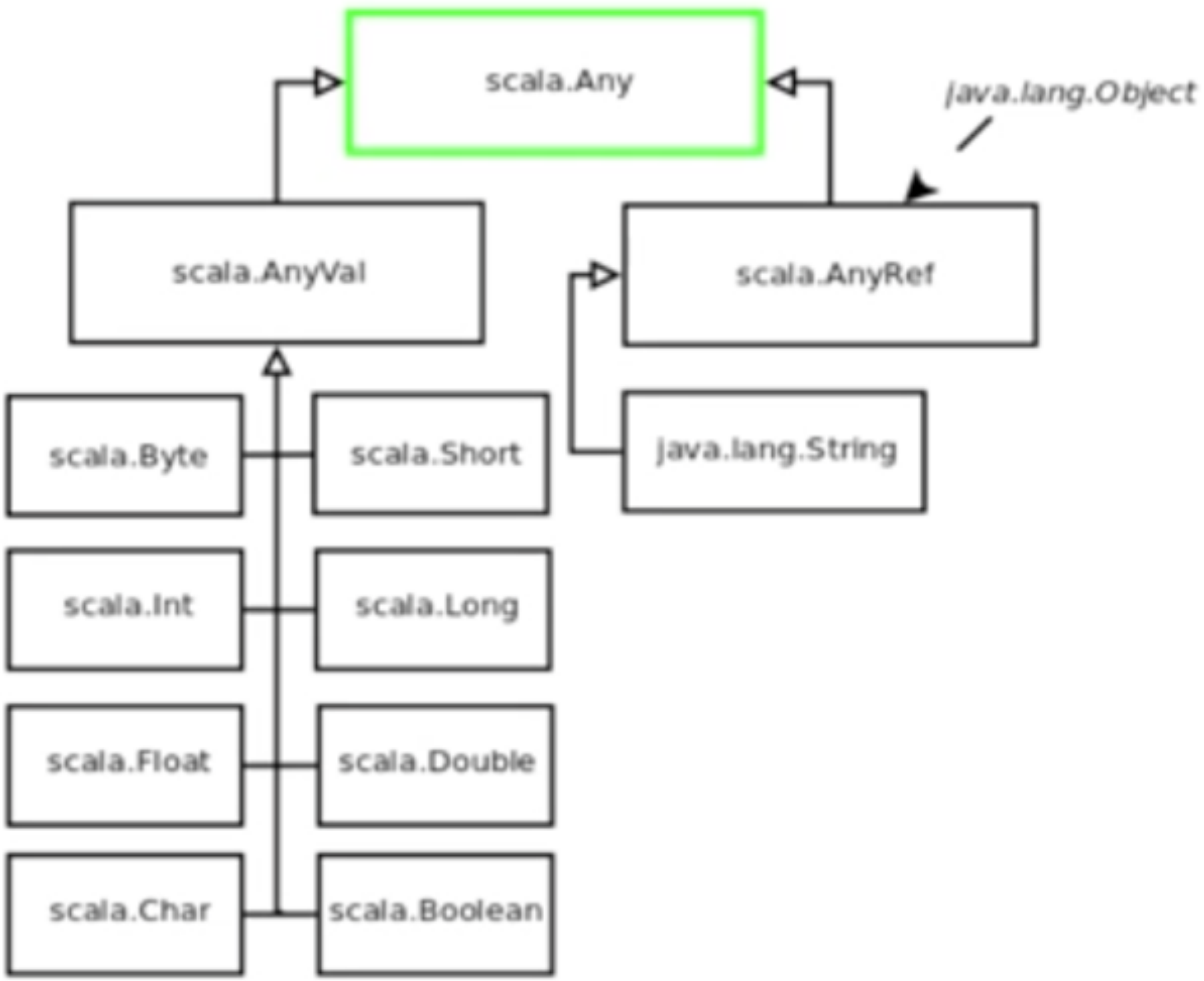




scala.Any

scala.AnyVal

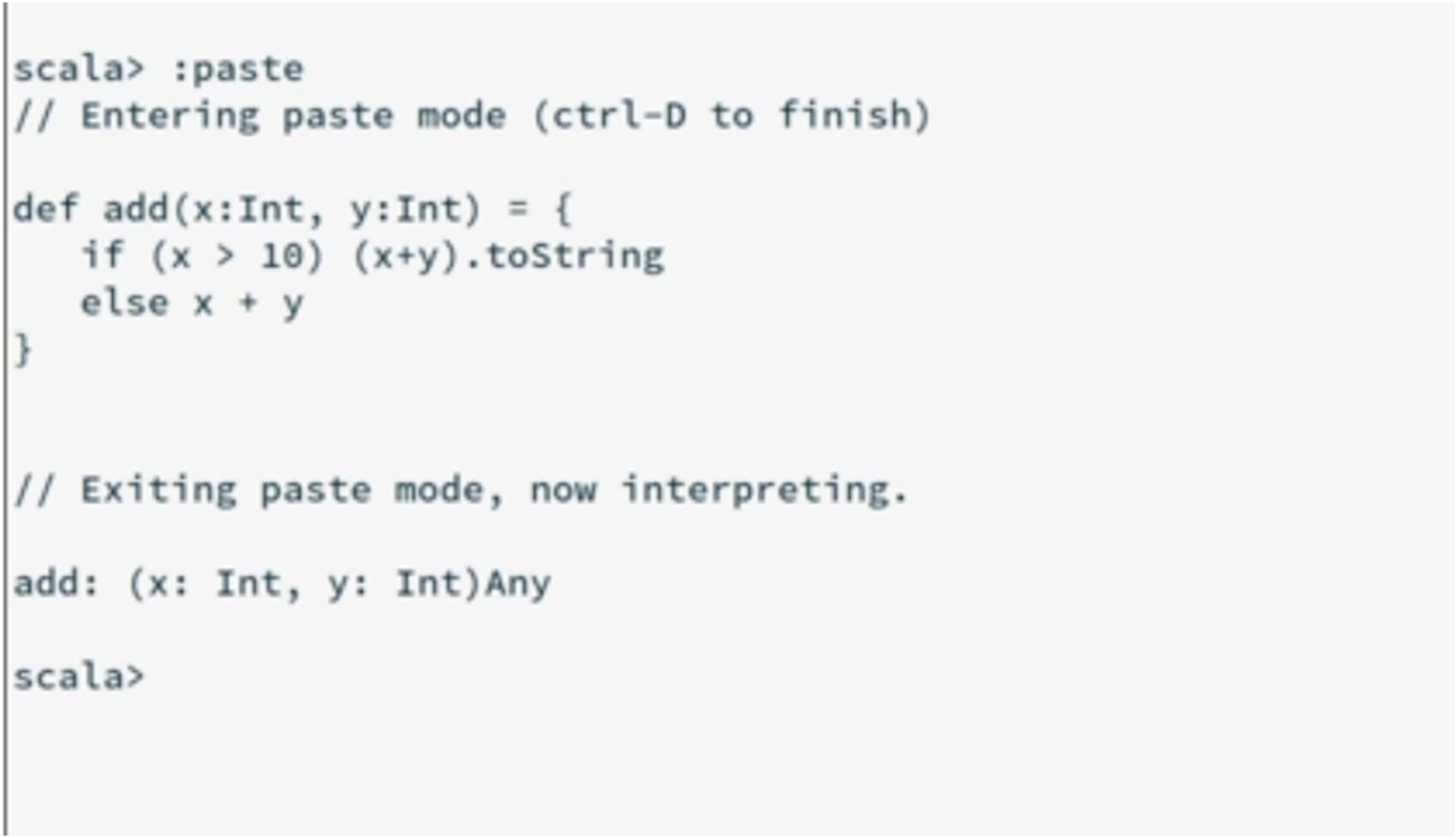
scala.AnyRef


scala.Any
java.lang.Object
scala.AnyVal
scala.AnyRef
scala.Byte
scala.Short
java.lang.String
scala.Int
scala.Long
scala.Float
scala.Double
scala.Char
scala.Boolean

VOID

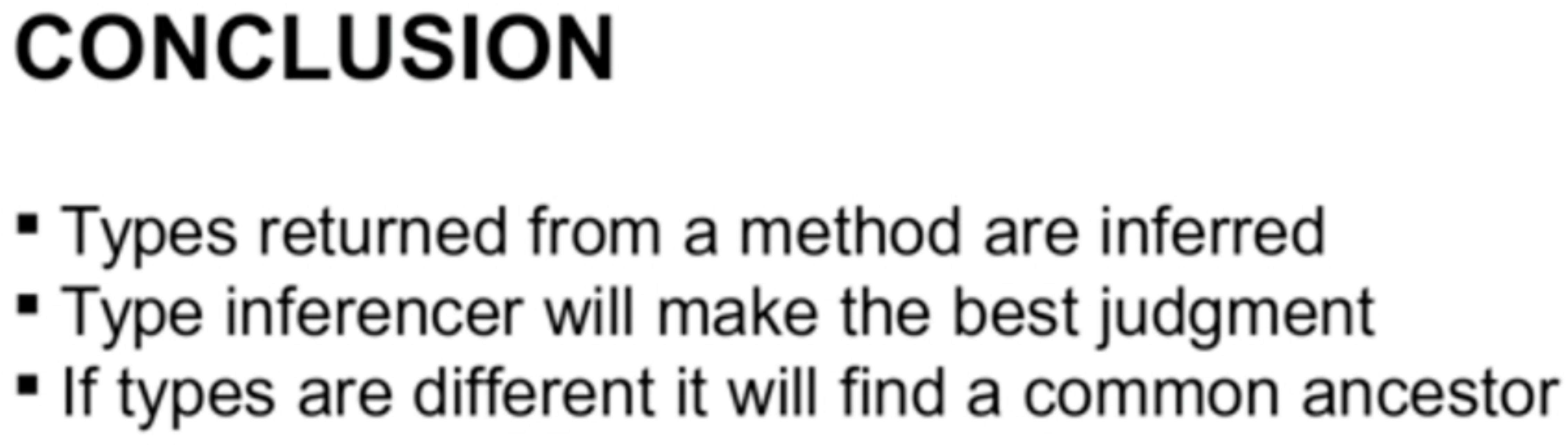
UNIT

Type Inference



```
scala> :paste
// Entering paste mode (ctrl-D to finish)

def add(x:Int, y:Int) = {
   if (x > 10) (x+y).toString
   else x + y
}


// Exiting paste mode, now interpreting.

add: (x: Int, y: Int)Any

scala>
```

Type Inference - Conclusion

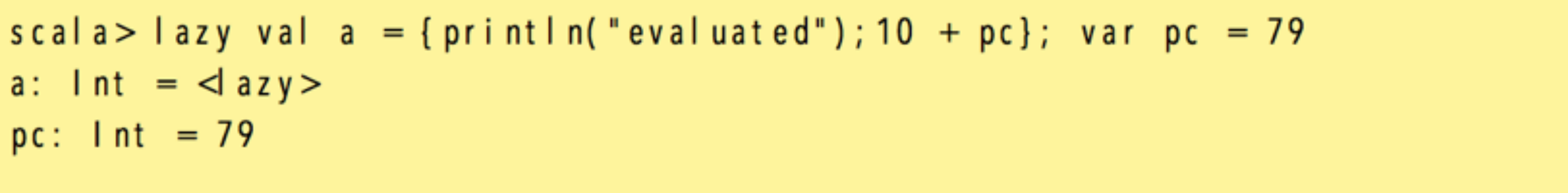


CONCLUSION

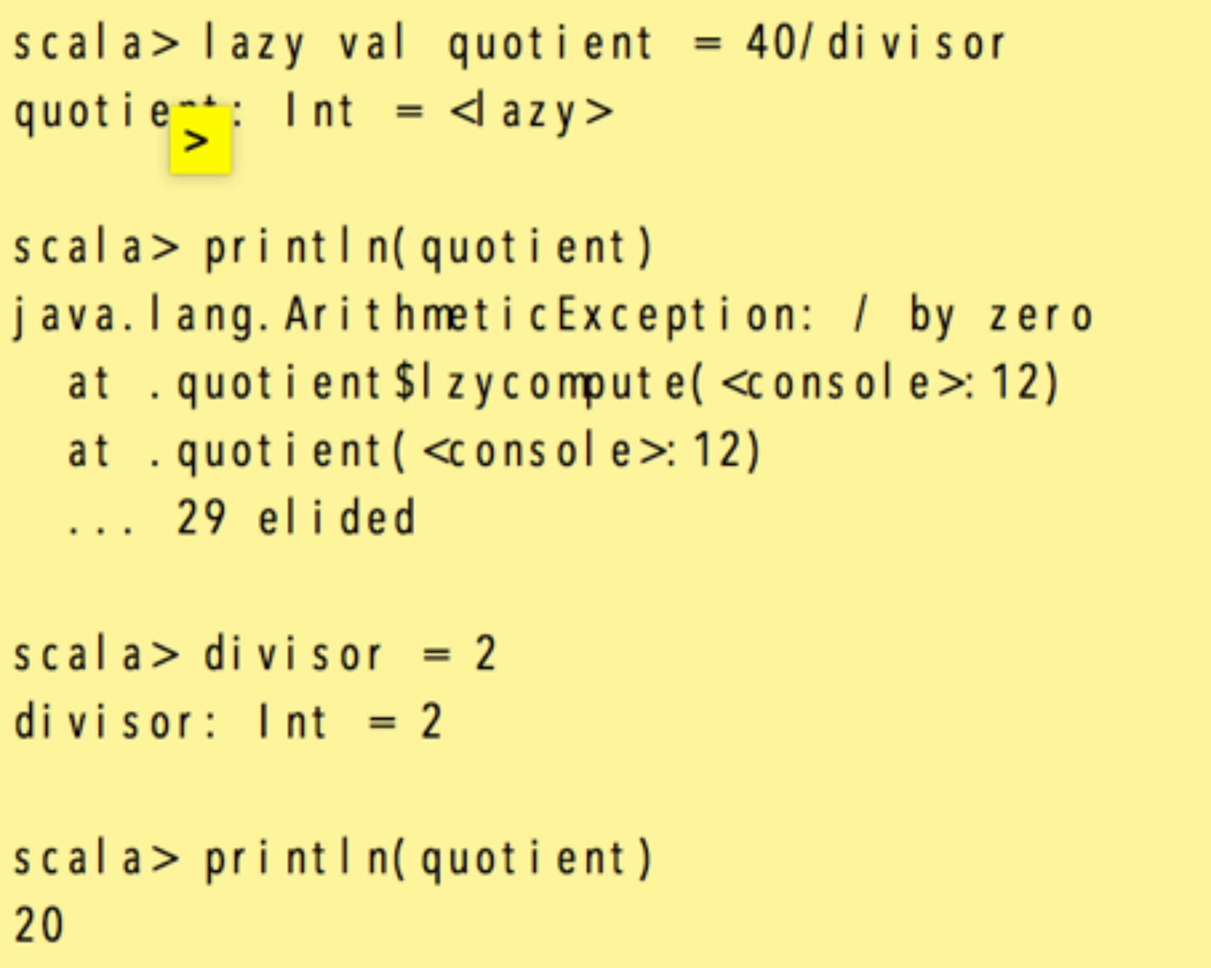
- Types returned from a method are inferred
- Type inferencer will make the best judgment
- If types are different it will find a common ancestor

Lazy Vals

lazy val a = {println("evaluated");10 + pc}; var pc = 79

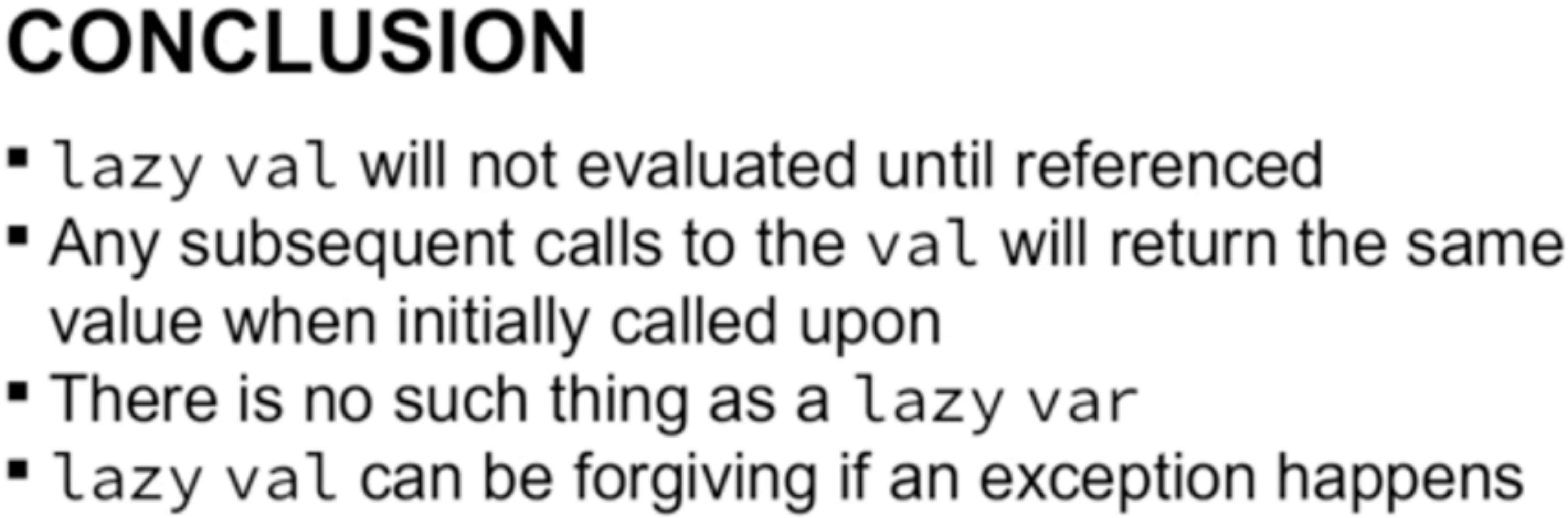


```
scala> lazy val a = {println("evaluated");10 + pc}; var pc = 79
a: Int = <lazy>
pc: Int = 79
```



```
scala> lazy val quotient = 40/divisor
quotient: Int = <lazy>

scala> println(quotient)
java.lang.ArithmeticException: / by zero
  at .quotient$lzycompute(<console>:12)
  at .quotient(<console>:12)
  ... 29 elided

scala> divisor = 2
divisor: Int = 2

scala> println(quotient)
20
```

Lazy Vals

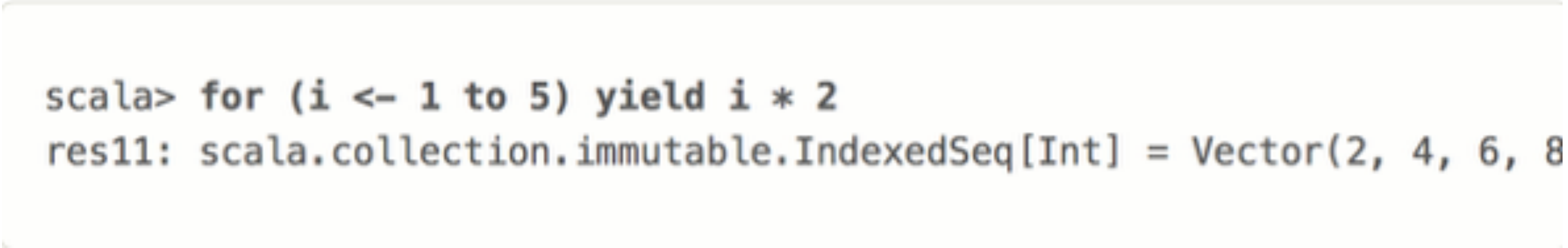


CONCLUSION

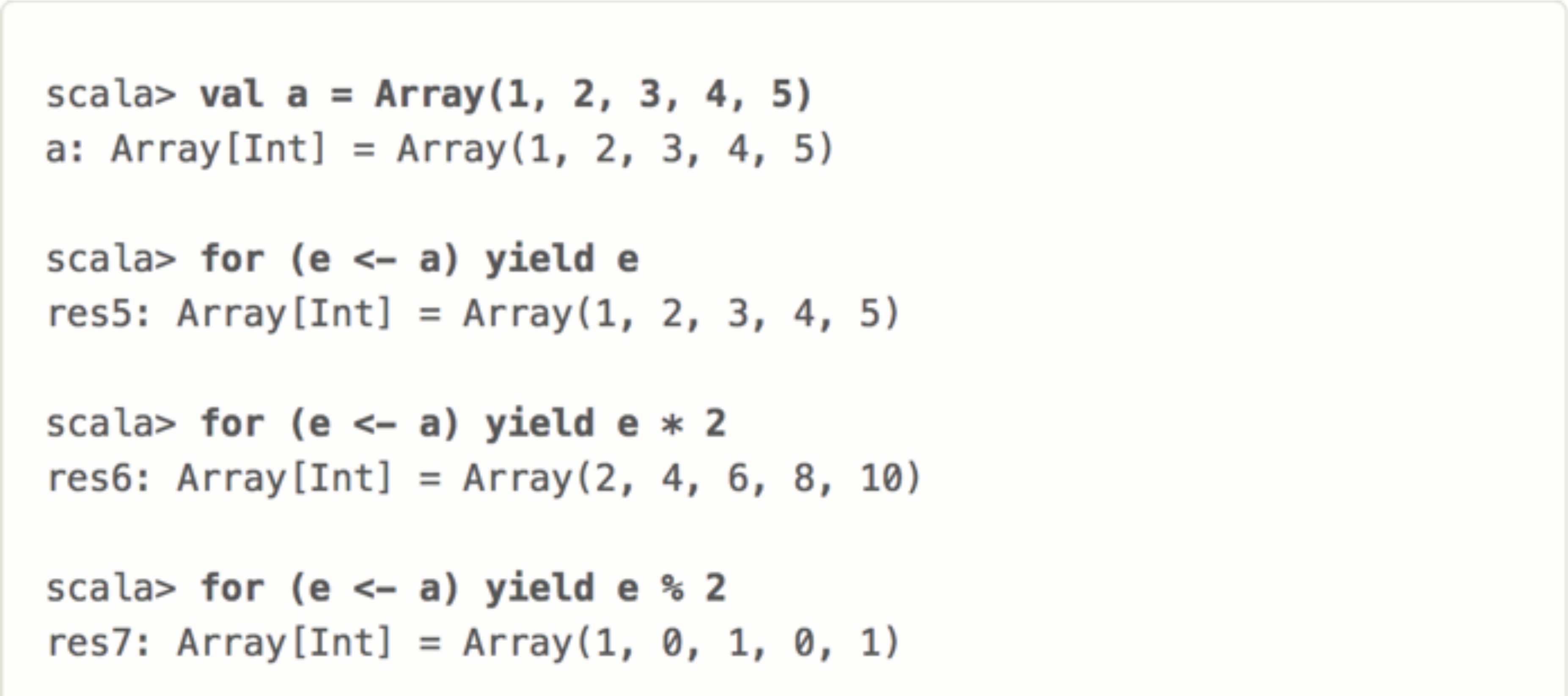
- `lazy val` will not evaluated until referenced
- Any subsequent calls to the `val` will return the same value when initially called upon
- There is no such thing as a `lazy var`
- `lazy val` can be forgiving if an exception happens

For Comprehension with Yield

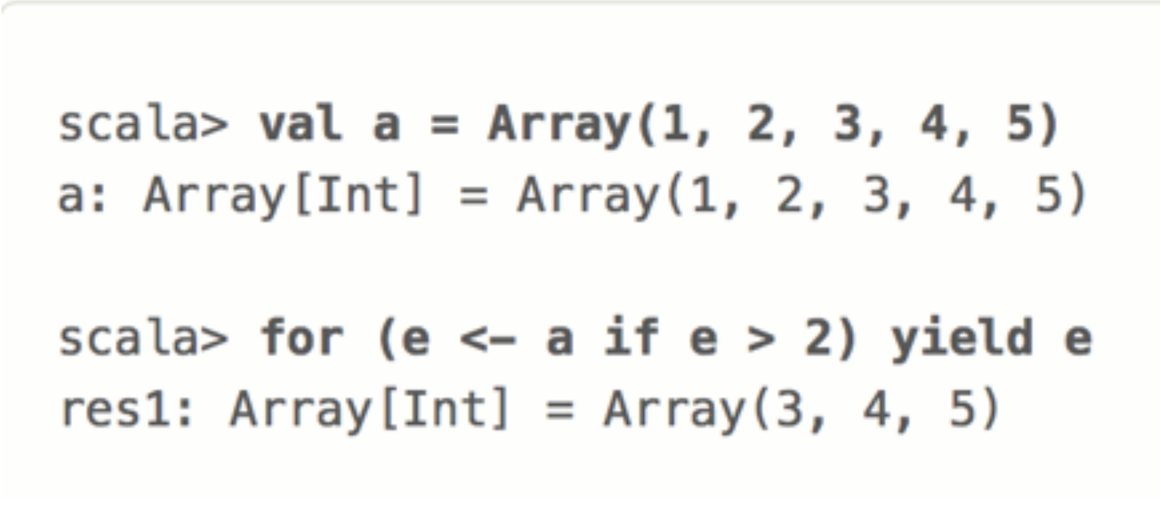
- Consider a FOR COMPREHENSION with yield as a FOR LOOP WITH BUFFER which results the same collection as input but with processed values as provided with YIELD.



```
scala> for (i <- 1 to 5) yield i * 2
res11: scala.collection.immutable.IndexedSeq[Int] = Vector(2, 4, 6, 8
```



```
scala> val a = Array(1, 2, 3, 4, 5)
a: Array[Int] = Array(1, 2, 3, 4, 5)

scala> for (e <- a) yield e
res5: Array[Int] = Array(1, 2, 3, 4, 5)

scala> for (e <- a) yield e * 2
res6: Array[Int] = Array(2, 4, 6, 8, 10)

scala> for (e <- a) yield e % 2
res7: Array[Int] = Array(1, 0, 1, 0, 1)
```



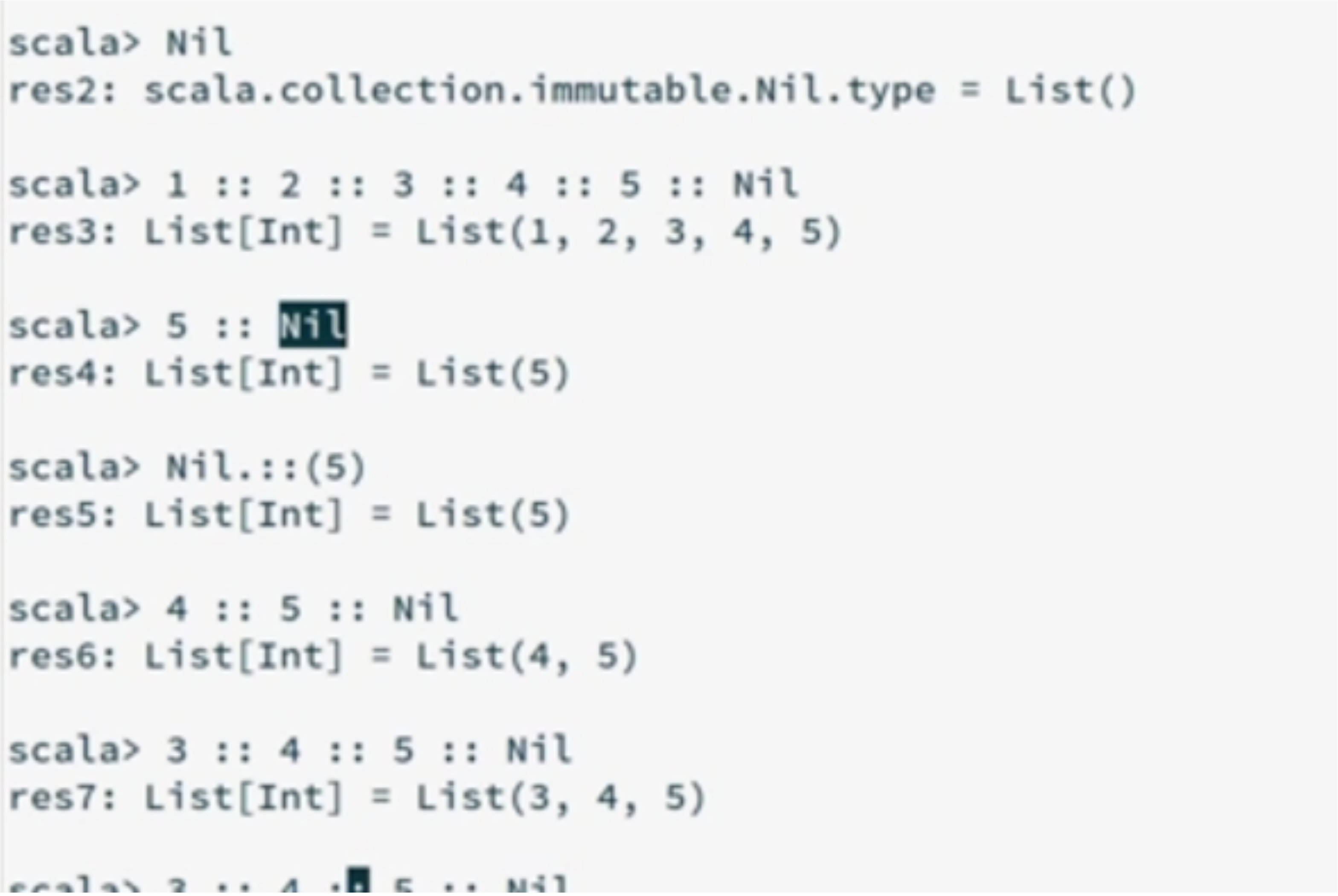
```
scala> val a = Array(1, 2, 3, 4, 5)
a: Array[Int] = Array(1, 2, 3, 4, 5)

scala> for (e <- a if e > 2) yield e
res1: Array[Int] = Array(3, 4, 5)
```

Importance of Clean APIs

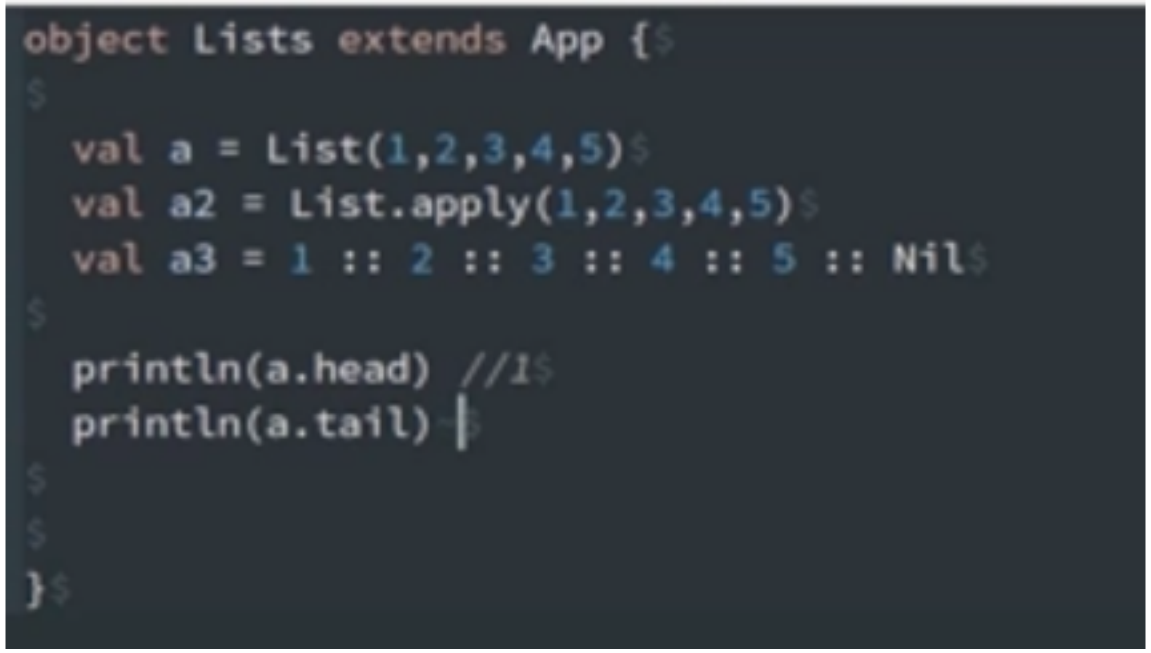
- Lot of importance has been given to maintain cleaner APIs in SCALA collections.
  - Learning implementation of a function available in a particular collection helps to implement the same function for any other collection, as many important functions are intentionally maintained across various collections.
  - This eases the life of a developer in a big way.

http://www.scala-lang.org/api/current/scala/collection/immutable/List.html

LISTS



```
scala> Nil
res2: scala.collection.immutable.Nil.type = List()

scala> 1 :: 2 :: 3 :: 4 :: 5 :: Nil
res3: List[Int] = List(1, 2, 3, 4, 5)

scala> 5 :: Nil
res4: List[Int] = List(5)

scala> Nil.::(5)
res5: List[Int] = List(5)

scala> 4 :: 5 :: Nil
res6: List[Int] = List(4, 5)

scala> 3 :: 4 :: 5 :: Nil
res7: List[Int] = List(3, 4, 5)

scala> 3 :: 4 :: 5 :: Nil
```



```
object Lists extends App {$
$
  val a = List(1,2,3,4,5)$
  val a2 = List.apply(1,2,3,4,5)$
  val a3 = 1 :: 2 :: 3 :: 4 :: 5 :: Nil$
$
  println(a.head) //1$
  println(a.tail) $
$
$
}$
```