Advanced Programming with Scala

7,5 hp

- Critically reflect and describe principles of functional programming.
- Critically reflect and describe the main differences between functional- and imperative programming.
- Critically reflect on efficiency issues on functional programming.
- Demonstrate abilities to independently develop programs within the functional programming paradigm and in particular using the Scala programming language.

Sep.	W37	L1. Introduction to functional programming. Scala. L2. Basics of the language. Declarations. Functions. Lab 1	
	W38	L3. Lists. Pattern matching. L4. Higher-order functions. Lazy and eager evaluation. Lab 2	Lab
	W39	L5. Object oriented programming in Scala L6. Functional data structures 1 Assignment (formative assessment)	
	W40	L7. Functional data structures 2 Lab 3 Assignment (formative assessment)	
Nov.	W41	L8. Parallelism L9. Advanced typing Assignment: submission	W43
	W42	Assignment: re-submission	W44 Exam

Lectures

- Theoretical
- Different rooms
- Broadcasted via Zoom
- A laptop might help

Labs

- Practical
- Lab D204 (need card)
- May bring own laptop

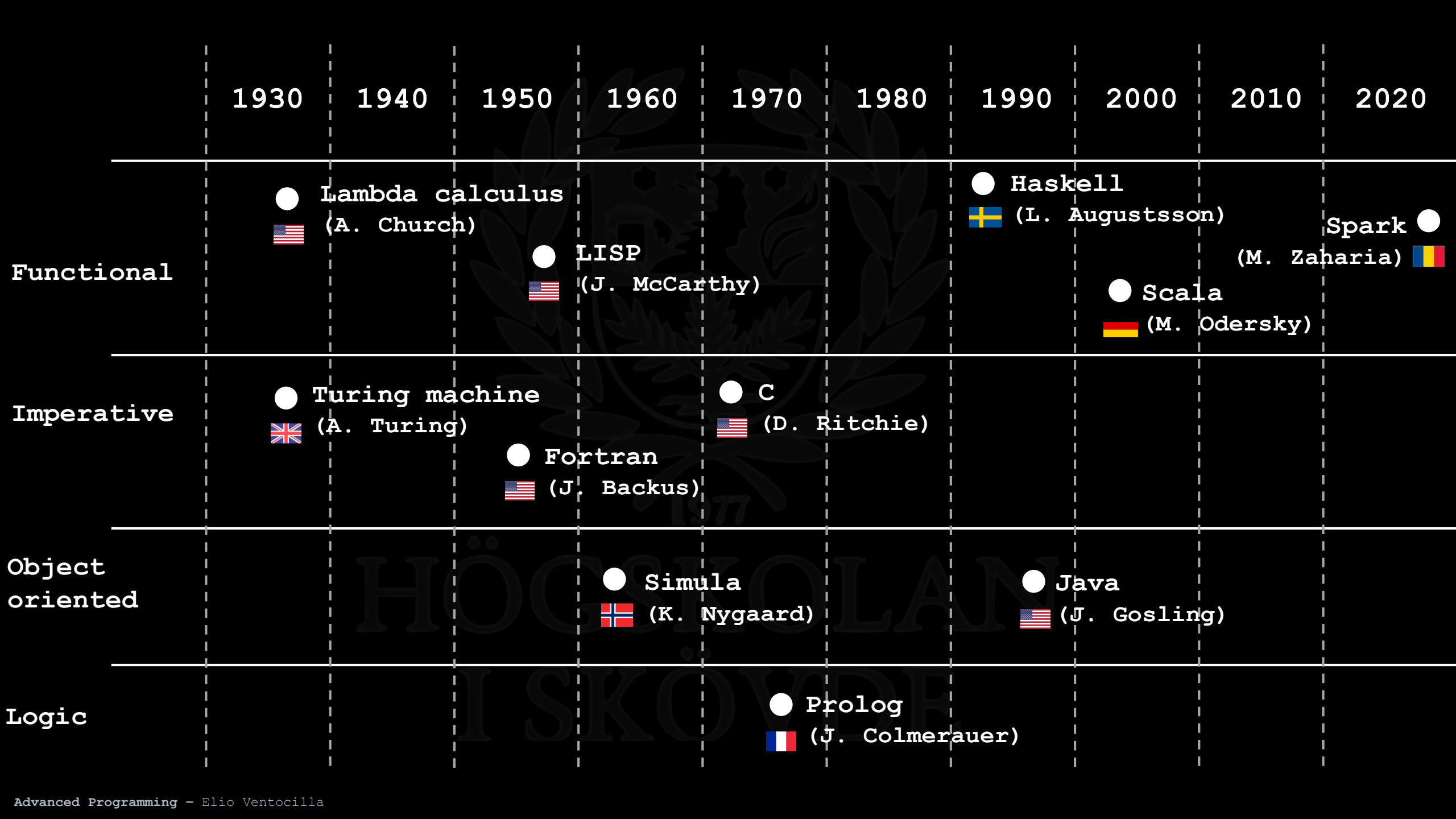
Assignment

- U/G (Fail/Pass)
- Code
- Two chances
- Individual

Exam

- A-F
- Written
- Two chances
- Individual

canvas.his.se


1930
1940
1950
1960
1970
1980
1990
2000
2010
2020
Functional
Lambda calculus
(A. Church)
LISP
(J. McCarthy)
Haskell
(L. Augustsson)
Spark
(M. Zaharia)
Scala
(M. Odersky)
Imperative
Turing machine
(A. Turing)
Fortran
(J. Backus)
C
(D. Ritchie)
Object oriented
Simula
(K. Nygaard)
Java
(J. Gosling)
Logic
Prolog
(J. Colmerauer)

	Functional	Imperative	Logic
Programs as	Functions	Commands	Relations
Blocks as	Expressions	Statements	Horn clauses
Programs made by	Composing functions	Command sequences	List of rules
Values	Immutable	Mutable	Immutable
Repetition	Recursion	Loops	Recursion

Adapted from *Scala: From a Functional Programming Perspective* (V. Torra, 2017)

Imperative

```
int sum(int[] arr) {
  int sum = 0;
  for (x in arr) {
    sum += x;
  }
  return sum;
}
```

Functional

```
def sum(arr: List[Int]) = {
  arr match {
    case Nil => 0
    case h :: t => h + sum(t)
  }
}
```

Imperative

```
int sum(int[] arr) {
  int sum = 0;
  for (x in arr) {
    sum += x;
  }
  return sum;
}
```

Blocks as

statements

vs.

expressions

Functional

```
def sum(arr: List[Int]) = {
  arr match {
    case Nil => 0
    case h :: t => h + sum(t)
  }
}
```

Expression

"a construct that will be evaluated to yield a value"

Watt, D. A. (2004) *Programming language design concepts*

Imperative

```
int sum(int[] arr) {
  int sum = 0;
  for (x in arr) {
    sum += x;
  }
  return sum;
}
```

Repetitions

loops
vs
recursion

Functional

```
def sum(arr: List[Int]) = {
  arr match {
    case Nil => 0
    case h :: t => h + sum(t)
  }
}
```

A declarative paradigm:
You state what should be done, and not how it should be done

E.g., SQL

Imperative

```
int sum(int[] arr) {
  int sum = 0;
  for (x in arr) {
    sum += x;
  }
  return sum;
}
```

Values

mutable
vs
immutable

Functional

```
def sum(arr: List[Int]) = {
  arr match {
    case Nil => 0
    case h :: t => h + sum(t)
  }
}
```

Imperative

```
int sum(int[] arr) {
  int sum = 0;
  for (x in arr) {
    sum += x;
  }
  return sum;
}
```

Programs made by

command sequences
vs
composing functions

Functional

```
def sum(arr: List[Int]) = {
  arr match {
    case Nil => 0
    case h :: t => h + sum(t)
  }
}
```

LISP:

```
(defun sum (arr)
  (if (null arr) 0
    (+ (first arr) (sum (rest arr)))))
```

Object oriented

```
class MyArray {
    private int[] arr = ...;

    public int sum() {
        int sum = 0;
        for (x in arr) {
            sum += x;
        }
        return sum;
    }
}
```

Logic

```
sum([], 0).

sum([H|T], Sum) :-
    sum_list(T, Rest),
    Sum is H + Rest.
```

Why Functional programming?

No side effects! Which means…

- You know what to expect.
- Easier to test.
- Stable parallelism.

Higher-order functions + recursion means…

- More concise code.
- More powerful ways to express solutions.

Why Scala?

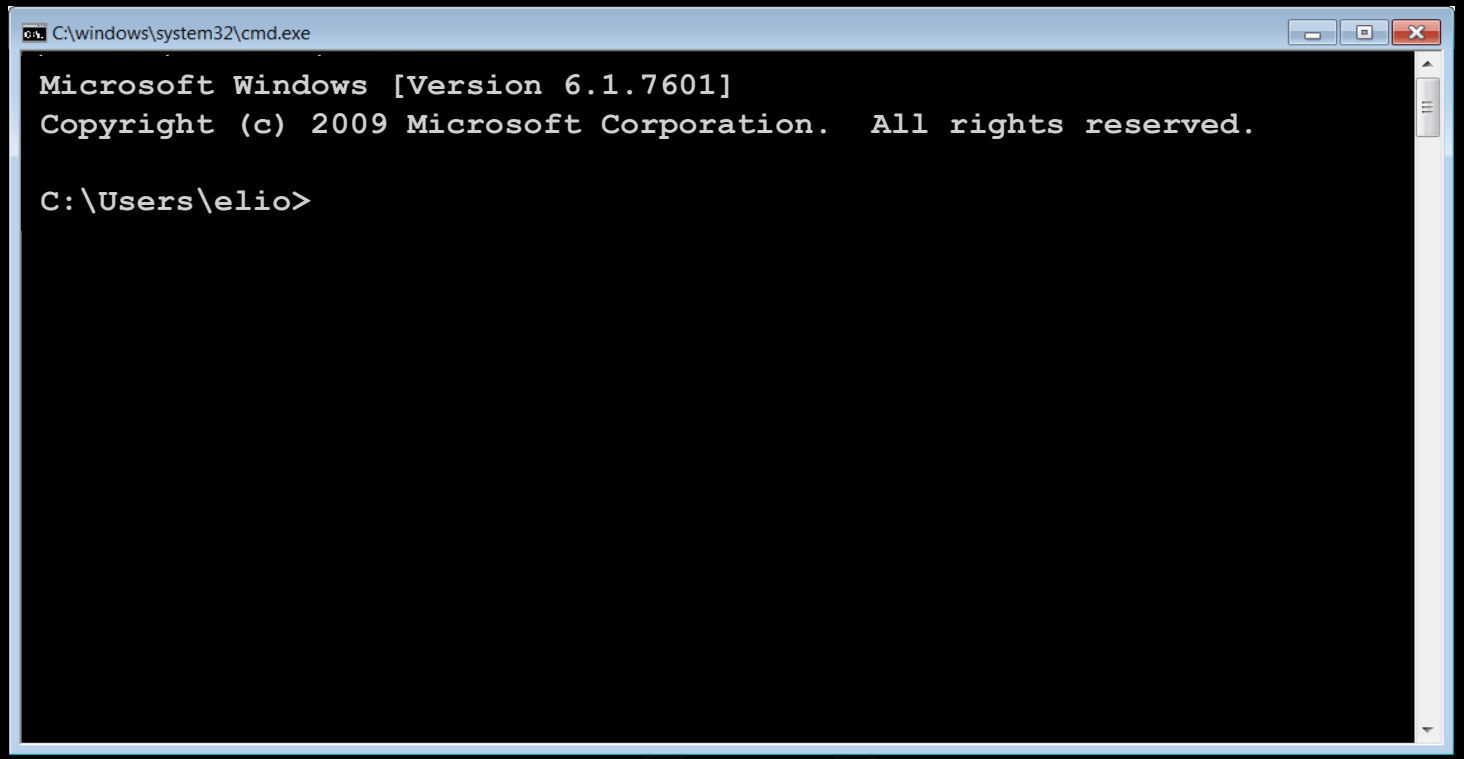
- Pedagogical purposes.
- Functional / imperative / object oriented.
- Works along with Java.
- Useful for Big Data.
- Useful in the market.

Functional principles

- We use pure functions:
  - Always return a value
  - No side effects
  - Referentially transparent
- Functions are first-class citizens
  - Can be passed as parameters
  - Can be returned
- We rely ~~only~~ on expressions. If inevitable, we use immutable values.

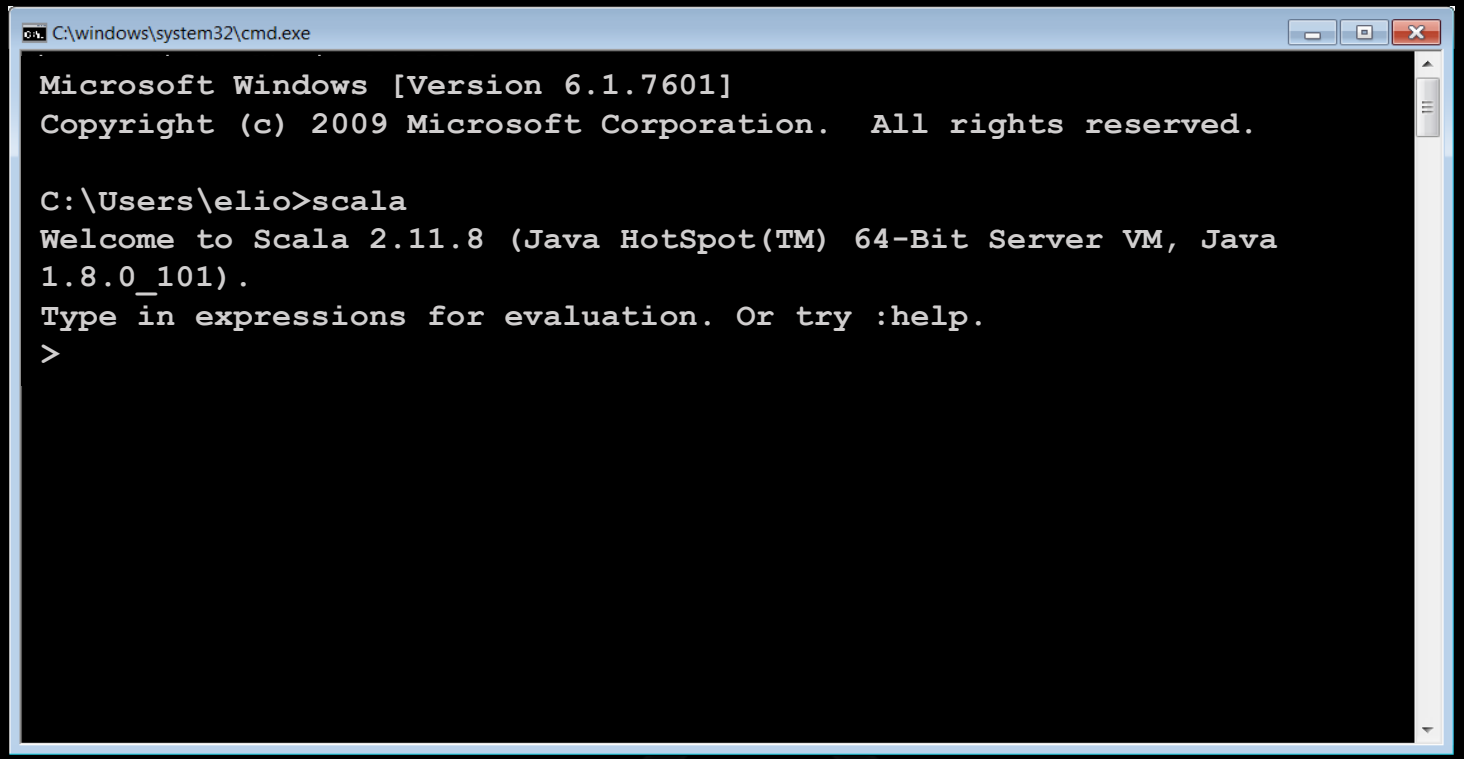
Derived concepts

- The substitution model
- Higher-order functions
- Anonymous functions
- Currification
- Recursion



CMD / terminal

Command	Description
`scala`	Enter the REPL
`scala <filename.scala>`	Run Scala app (file should have a main)
`scalac <filename.scala>`	Compile Scala file



REPL: Read, eval, print, loop

Command	Description
`:q`	Quit the REPL
`:paste`	Paste mode in REPL
`:load <filename.scala>`	Load the given file into the REPL
`Ctrl + L`	Clear screen

Literals

```
2018          // Int
3.14          // Double
3.14f         // Float
45430000001   // Long
0x00FF        // Int
```

```
true          // Boolean
'c'           // Char
"hello"       // String
```

Expressions: arithmetic

```
2 + 2       // 4: Int

2 - 3       // -1: Int

2 * 3       // 6: Int

10 / 3      // 3: Int

10 % 0      // 0: Int
```

Expressions: logical

```
1 > 2                // false: Boolean

1 < 2                // true: Boolean

1 == 2               // false: Boolean

1 != 2               // true: Boolean

1 >= 2               // false: Boolean

1 <= 2               // true: Boolean

true || false        // true: Boolean

true && false        // false: Boolean
```

Expressions: bitwise

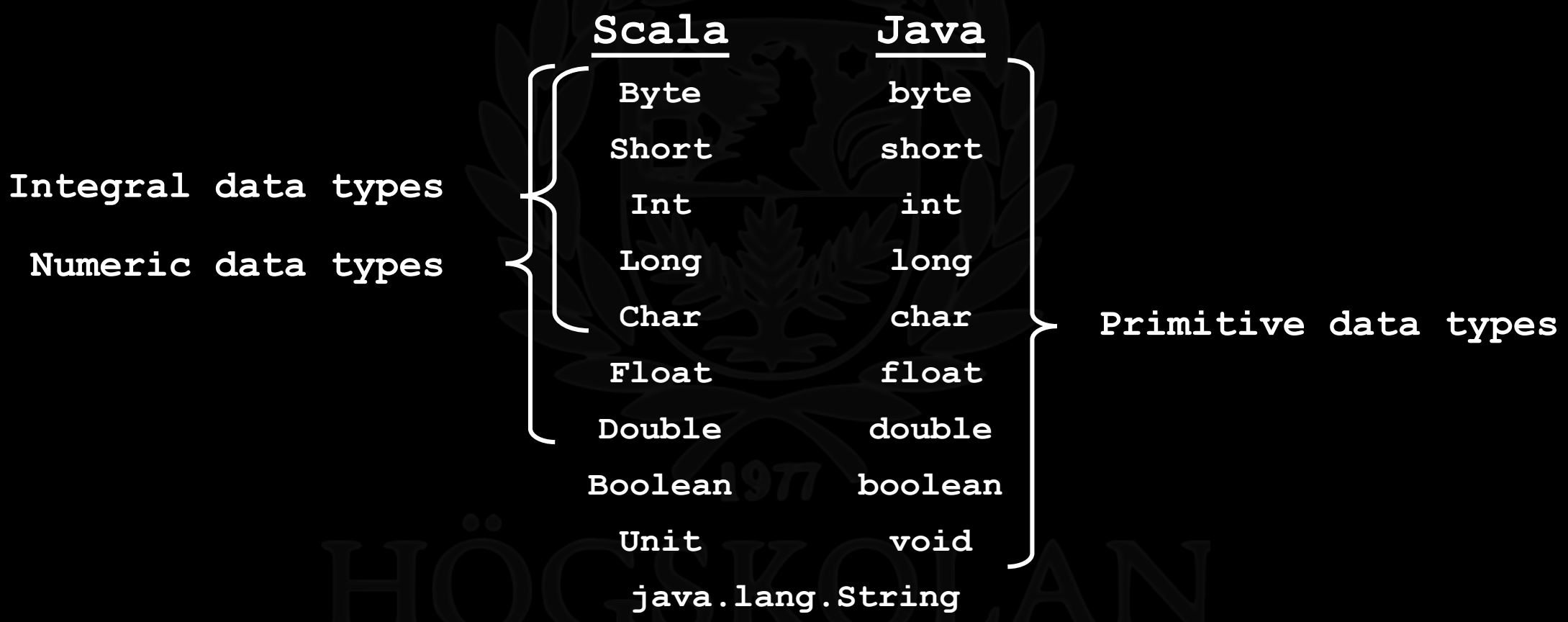
```
1 & 2                   // 0: Int

1 | 2                   // 3: Int

1 ^ 3                   // 2: Int

~1                      // -2: Int
```

Declarations

```
val year: Int = 2018

val pi: Double = 3.14

val earth: Long = 45430000001

val hello: String = "Hello"

val c: Char = 'c'

val isTrue: Boolean = true
```

Declarations: type inference

```
val year = 2018           // Int

val pi = 3.14             // Double

val earth = 45430000001 // Long

val hello = "Hello"       // String

val c = 'c'               // Char

val isTrue = true         // Boolean
```



scala and **java.lang** libraries
are automatically imported