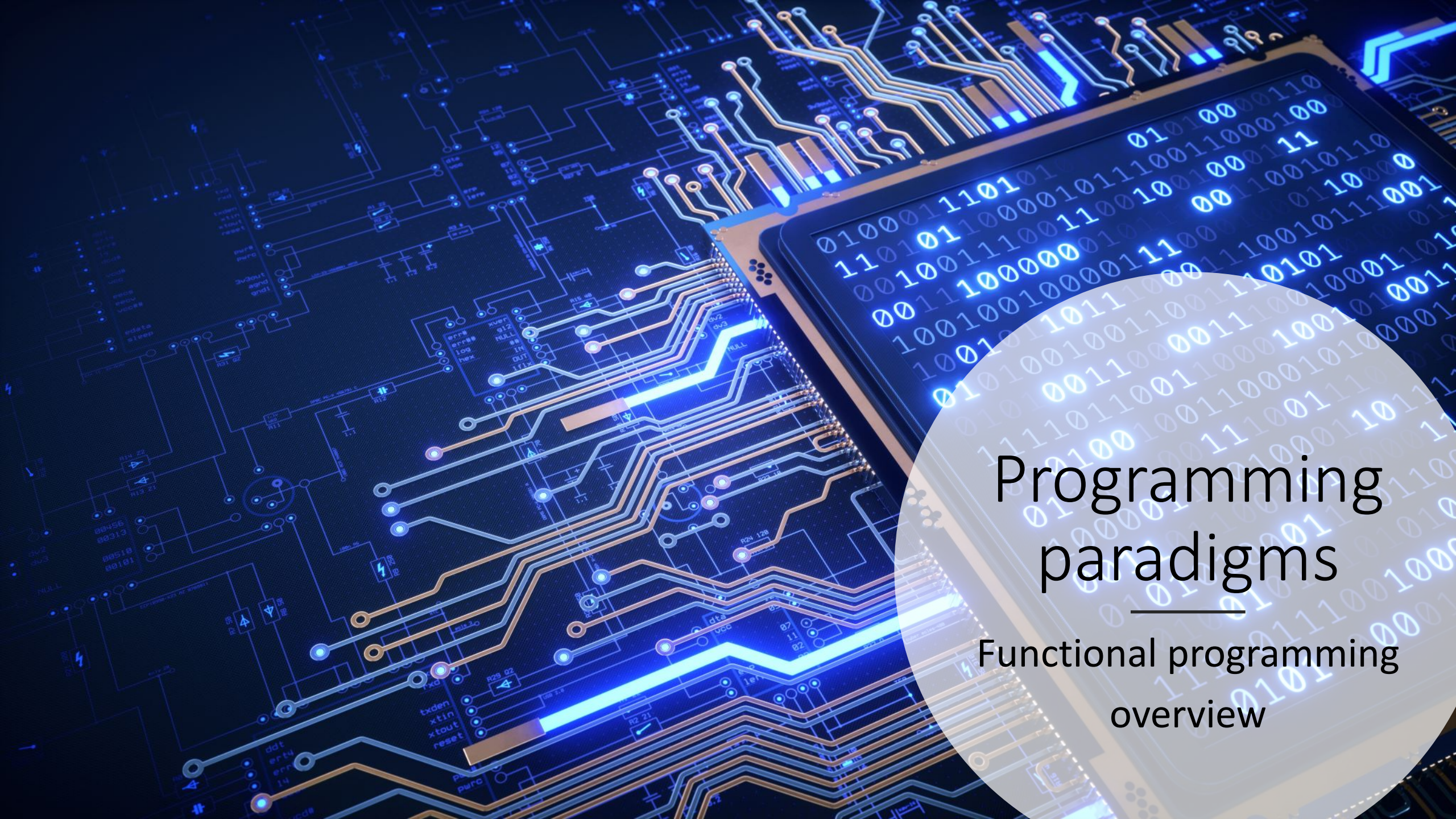
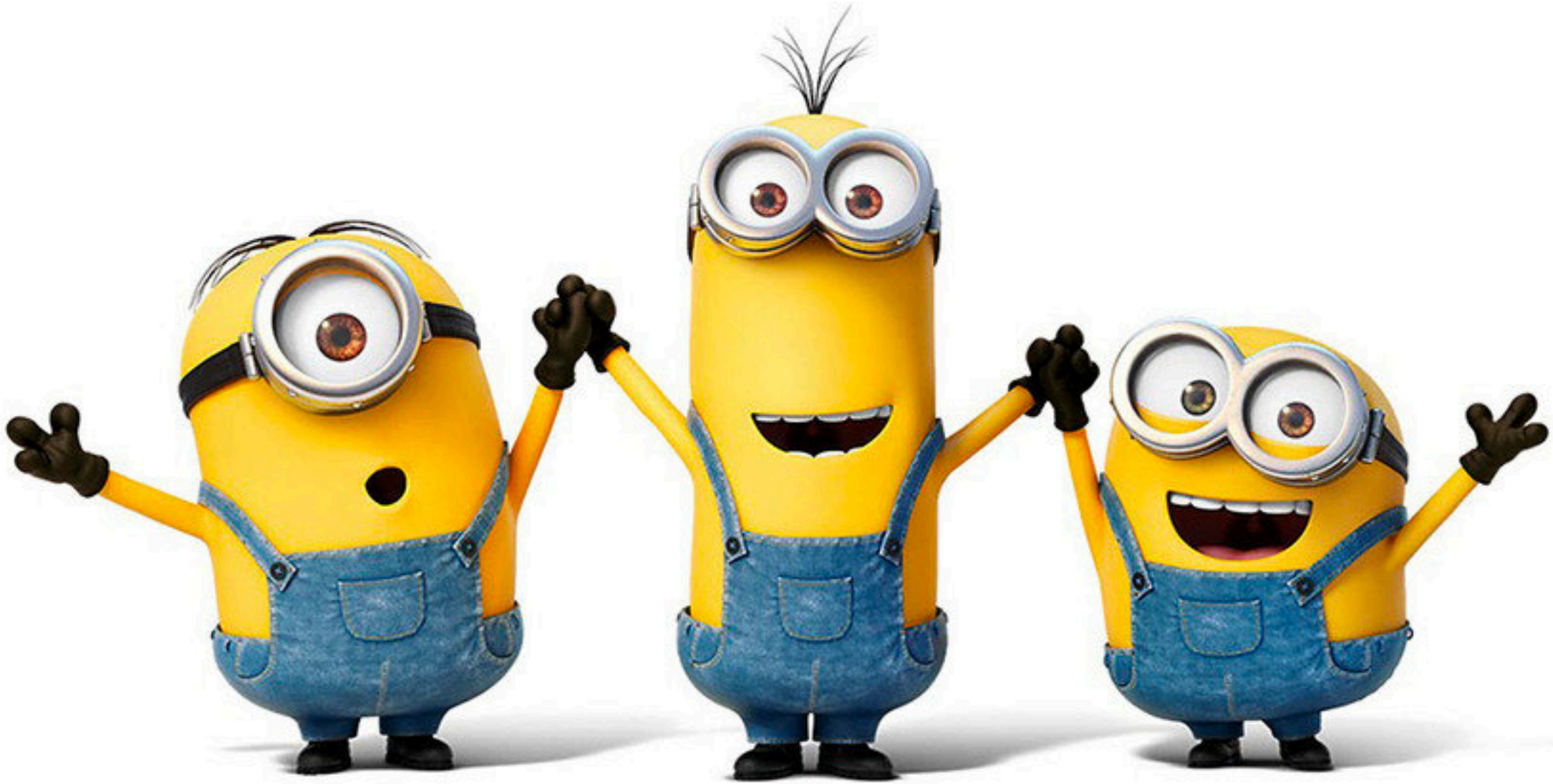


Programming paradigms

Functional programming overview

Programming?


```
('b' + 'a' + + 'a' + 'a').toLowerCase()
```




WAT

```
> []+[]
''
> []+{}
[object Object]
> {}+[]
0
> {}+{}
NaN
```


38

'b' + 'a' + + 'a' + ,a’

...is evaluated as....

('b') + ('a') + (+'a') + (,a’)

+'a' attempts to convert 'a' to a number using the unary plus operator. Because 'a' is not a number, the result is NaN (*"Not-A-Number"*)

Programming?

Paradigm

In science and philosophy, a paradigm is a distinct set of concepts or thought patterns, including theories, research methods, postulates, and standards for what constitutes legitimate contributions to a field.

The Paradigms of Programming

Robert W. Floyd
Stanford University

Paradigm(pæ·radim, −dəim) . . . [a. F. *paradigme*, ad. L. *paradigma*, a. Gr. παραδειγμα pattern, example, f. παραδεικνυ·ναι to exhibit beside, show side by side. . .]
1. A pattern, exemplar, example.
1752 J. Gill *Trinity* v. 91
The archetype, paradigm, exemplar, and idea, according to which all things were made.
From the Oxford English Dictionary.

Today I want to talk about the paradigms of programming, how they affect our success as designers of computer programs, how they should be taught, and how they should be embodied in our programming languages.

A familiar example of a paradigm of programming is the technique of *structured programming*, which appears to be the dominant paradigm in most current treatments of programming methodology. Structured programming, as formulated by Dijkstra [6], Wirth [27, 29], and Parnas [21], among others, consists of two phases.

In the first phase, that of top-down design, or stepwise refinement, the problem is decomposed into a very small number of simpler subproblems. In programming the solution of simultaneous linear equations, say, the first level of decomposition would be into a stage of triangularizing the equations and a following stage of back-substitution in the triangularized system. This gradual decomposition is continued until the subproblems that arise are simple enough to cope with directly. In the simultaneous equation example, the back substitution process would be further decomposed as a backwards iteration of a process which finds and stores the value of the *i*th variable from the *i*th equation. Yet further decomposition would yield a fully detailed algorithm.

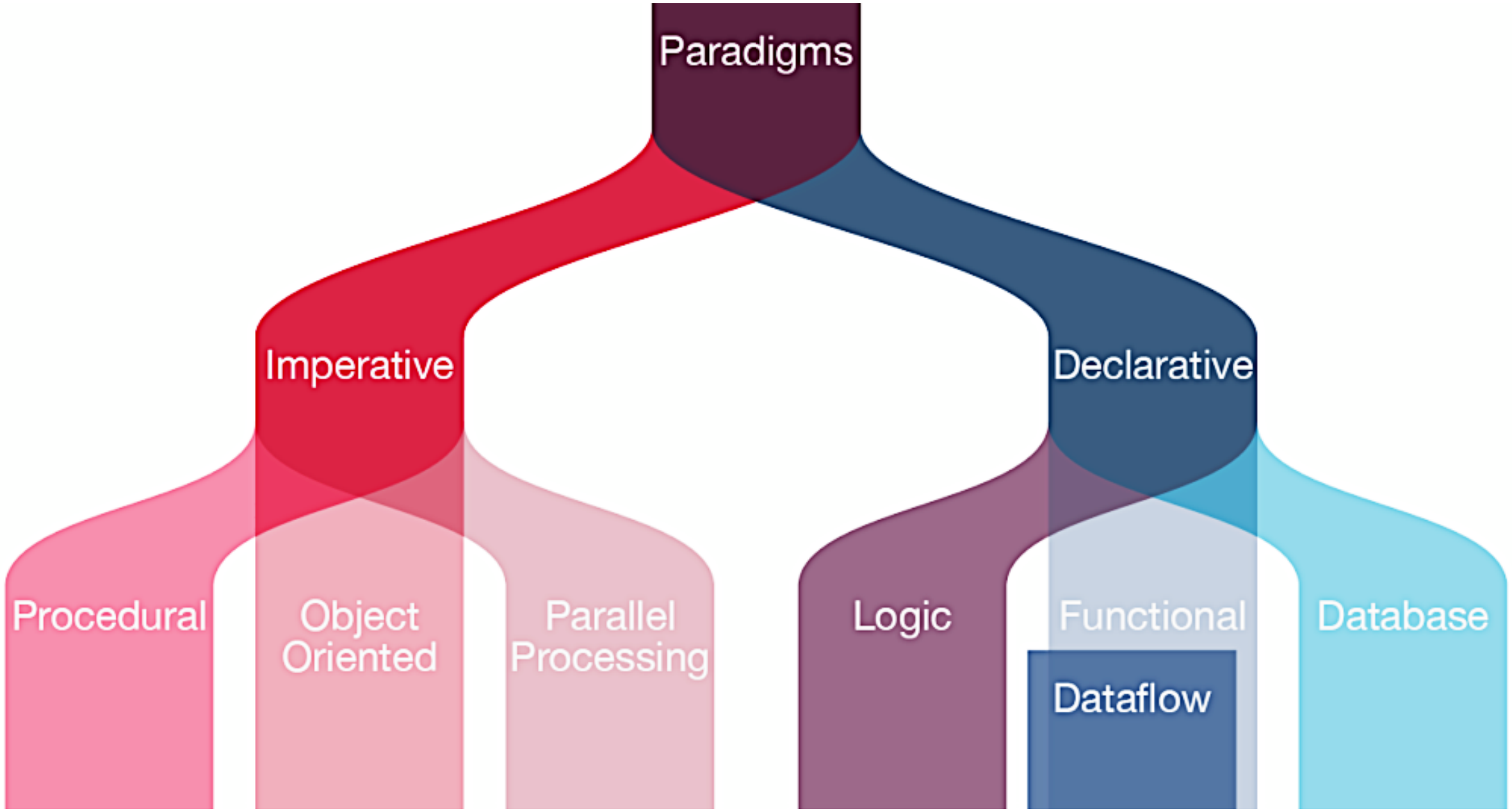
1978 ACM Turing Award Lecture

“I believe the best chance we have to improve the general practice of programming is to attend to our paradigms.

Robert W. Floyd
“The paradigms of programming.”, 1979. p. 456

Usually the first problems you solve with the new paradigm are the ones that were unsolvable with the old paradigm.

Joel A. Barker


Paradigms
Imperative
Declarative
Procedural
Object Oriented
Parallel Processing
Logic
Functional
Dataflow
Database

Imperative programming paradigm

It is one of the oldest programming paradigm.

It features close relation to machine architecture.

It is based on Von Neumann architecture.

It works by changing the program state through assignment statements. It performs step by step task by changing state.

The main focus is on how to achieve the goal. The paradigm consist of several statements and after execution of all the result is stored.

Imperative programming paradigm

Advantage:

- Very simple to implement
- It contains loops, variables etc.

Disadvantage:

- Complex problem cannot be solved
- Less efficient and less productive

Declarative programming paradigm

It is divided as Logic, Functional, Database.

In computer science the declarative programming is a style of building programs that expresses logic of computation without talking about its control flow.

It often considers programs as theories of some logic.It may simplify writing parallel programs. The focus is on what needs to be done rather how it should be done basically emphasize on what code code is actually doing. It just declare the result we want rather how it has be produced. This is the only difference between imperative (how to do) and declarative (what to do) programming paradigms. Getting into deeper we would see logic, functional and database.

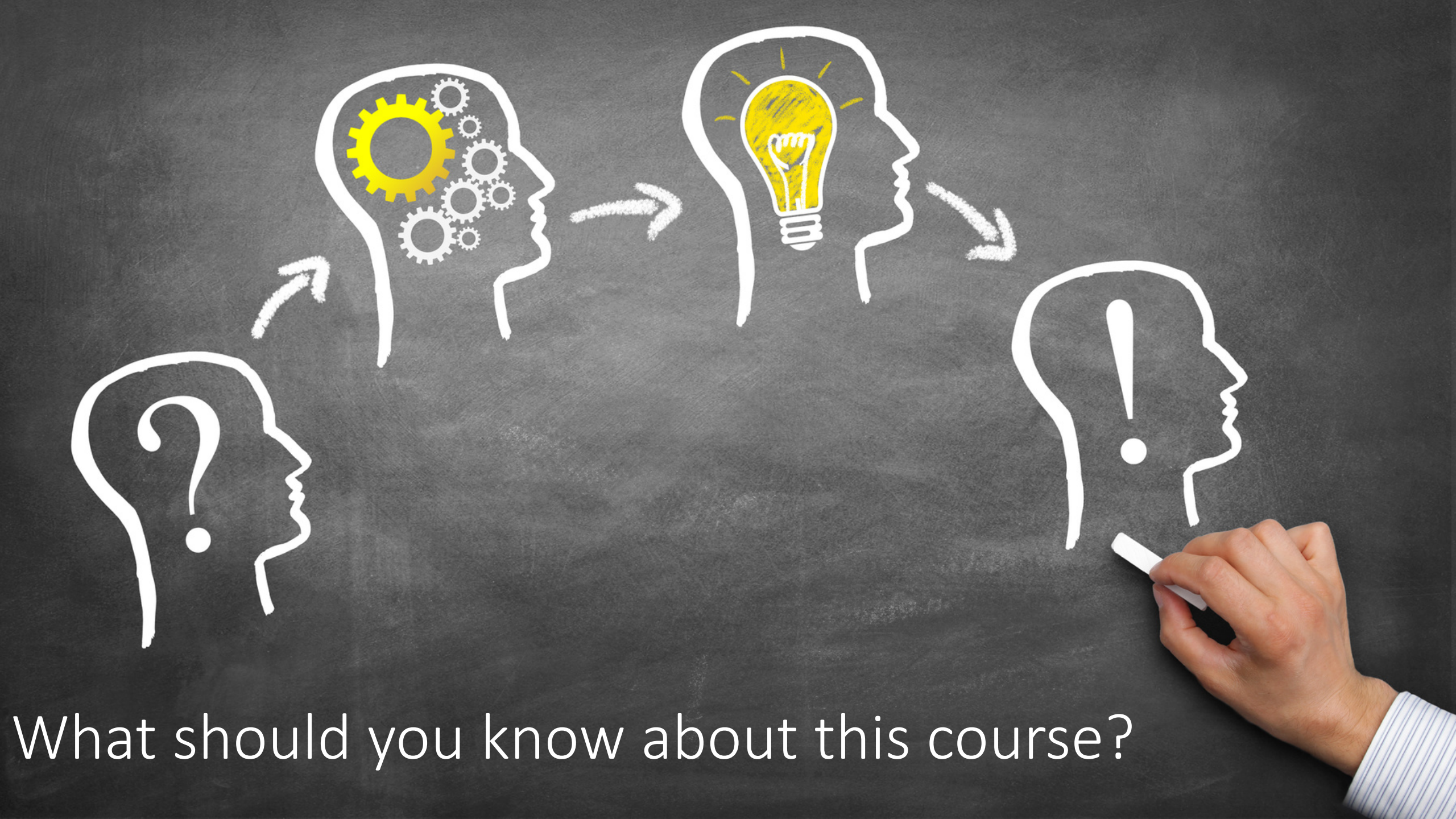
Imperative:

Programming with an explicit sequence of commands that update state.

Declarative:

Programming by specifying the result you want, not how to get it.

You Should Know This!



What should you know about this course?

Access to course (presentations etc.)

https://eportal.pwr.edu.pl/

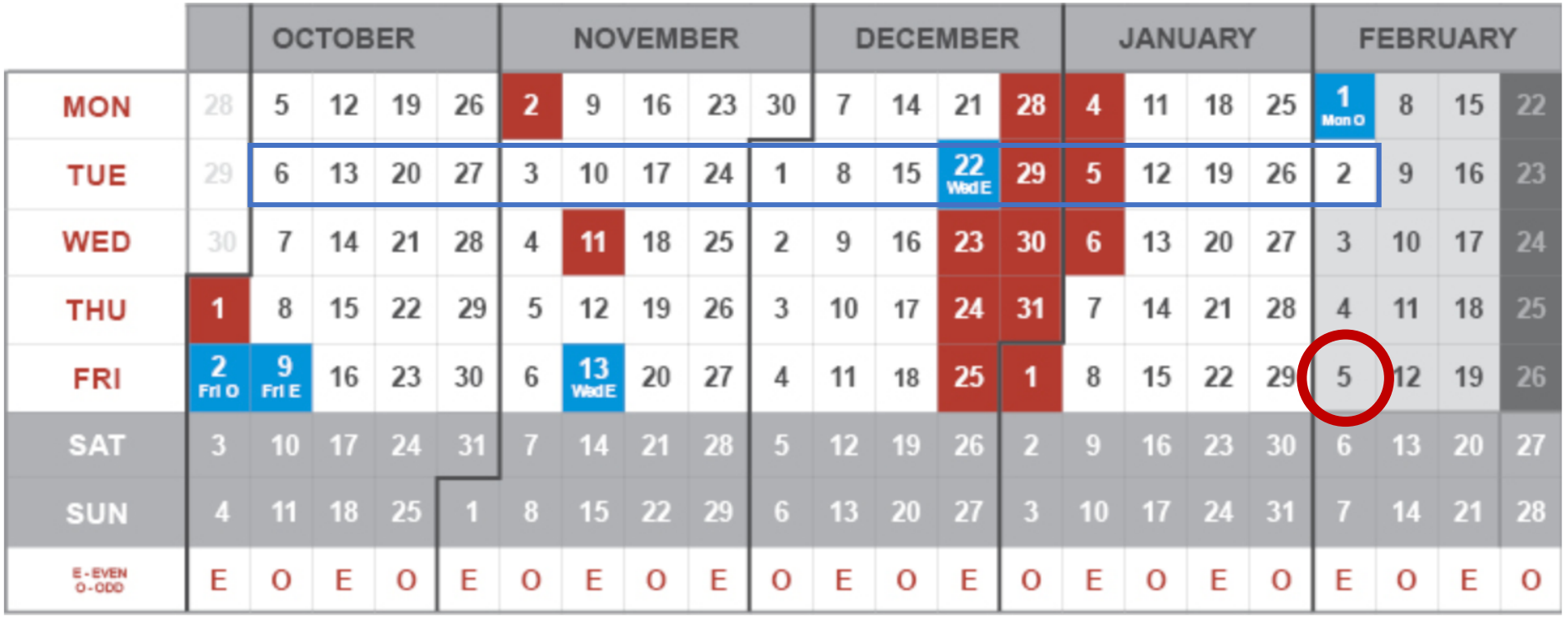
Search for:
Programming paradigms INZ004409W Z01-74a

Pass:
PP20_4409w

How to pass?

Exam:

Feb 5, 2021 – 10:00-14:00 (via zoom)



		OCTOBER				NOVEMBER					DECEMBER				JANUARY				FEBRUARY			
MON	28	5	12	19	26	2	9	16	23	30	7	14	21	28	4	11	18	25	1 Mon O	8	15	22
TUE	29	6	13	20	27	3	10	17	24	1	8	15	22 Wed E	29	5	12	19	26	2	9	16	23
WED	30	7	14	21	28	4	11	18	25	2	9	16	23	30	6	13	20	27	3	10	17	24
THU	1	8	15	22	29	5	12	19	26	3	10	17	24	31	7	14	21	28	4	11	18	25
FRI	2 Fri O	9 Fri E	16	23	30	6	13 Wed E	20	27	4	11	18	25	1	8	15	22	29	5	12	19	26
SAT	3	10	17	24	31	7	14	21	28	5	12	19	26	2	9	16	23	30	6	13	20	27
SUN	4	11	18	25	1	8	15	22	29	6	13	20	27	3	10	17	24	31	7	14	21	28
E - EVEN O - ODD	E	O	E	O	E	O	E	O	E	O	E	O	E	O	E	O	E	O	E	O	E	O

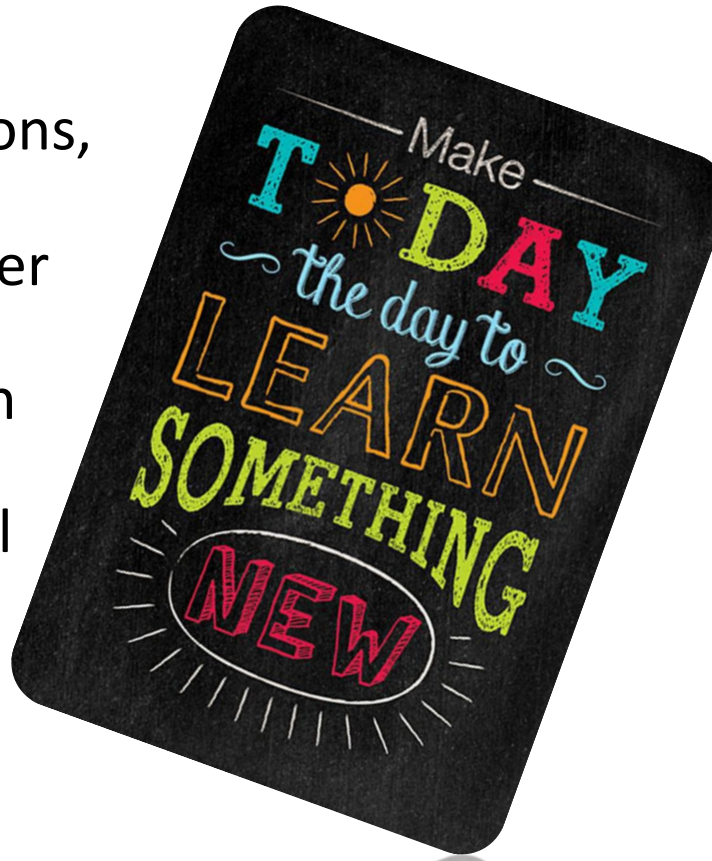
Other option:

Tutorials activity:

<50%	exam
50%-60%	3.0
60%-70%	3.5
70%-80%	4.0
80%-90%	4.5
90%-100%	5.0
100%	5.5

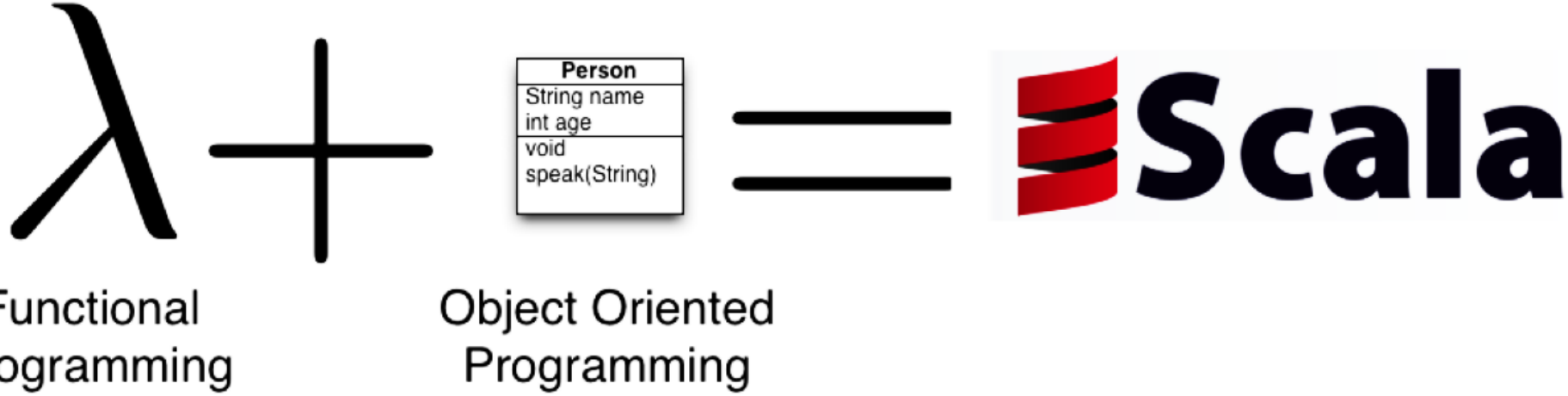
During that course

- ✓Introduction to the subject.
- ✓Functional programming in an interactive environment.
- ✓Basics of functional programming: expanded and collapsed functions, tail recursion, pattern matching. Higher order functions.
- ✓Algebraic data types: definition and use. Lazy evaluation. Parameter transfer.
- ✓Object-oriented programming. Subtypes, variability and restriction polymorphism.
- ✓Concurrent programming. Threads and memory sharing. Low level mechanisms.
- ✓Executors, thread pools. Asynchronous concurrency. "Future" mechanism.
- ✓Messaging. The mechanism of actors.
- ✓Reactive programming.
- ✓Transactional Memory.



Scala (labs)

It combines object-oriented and functional programming.



https://www.scala-lang.org/

Scala (history)

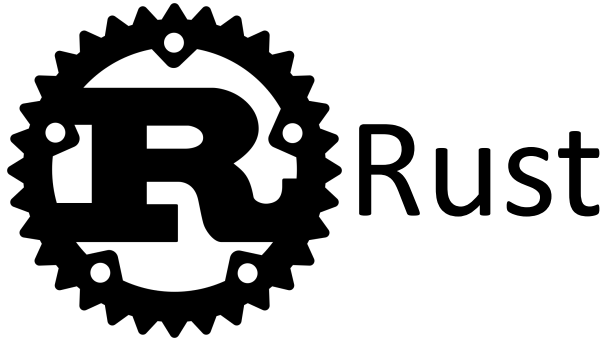
- It was created and developed by Martin Odersky.
- Martin started working on Scala in 2001 at the Ecole Polytechnique Federale de Lausanne (EPFL).
- It was officially released on January 20, 2004.
- 03.2006 Scala 2.0 was released
- Scala = "scalable language"

Hieronymus Bosch "A visual guide to the Scala language" oil on oak panels, 1490-1510

The left panel shows the functional features, the main one describes the type system, and the right the object oriented parts.

(source: http://classicprogrammerpaintings.com/post/142321815809/hieronymus-bosch-a-visual-guide-to-the-scala)

Additionally



Imperative:

Programming with an explicit sequence of commands that update state.

Declarative:

Programming by specifying the result you want, not how to get it.

Imperative vs declarative

An imperative approach (HOW):

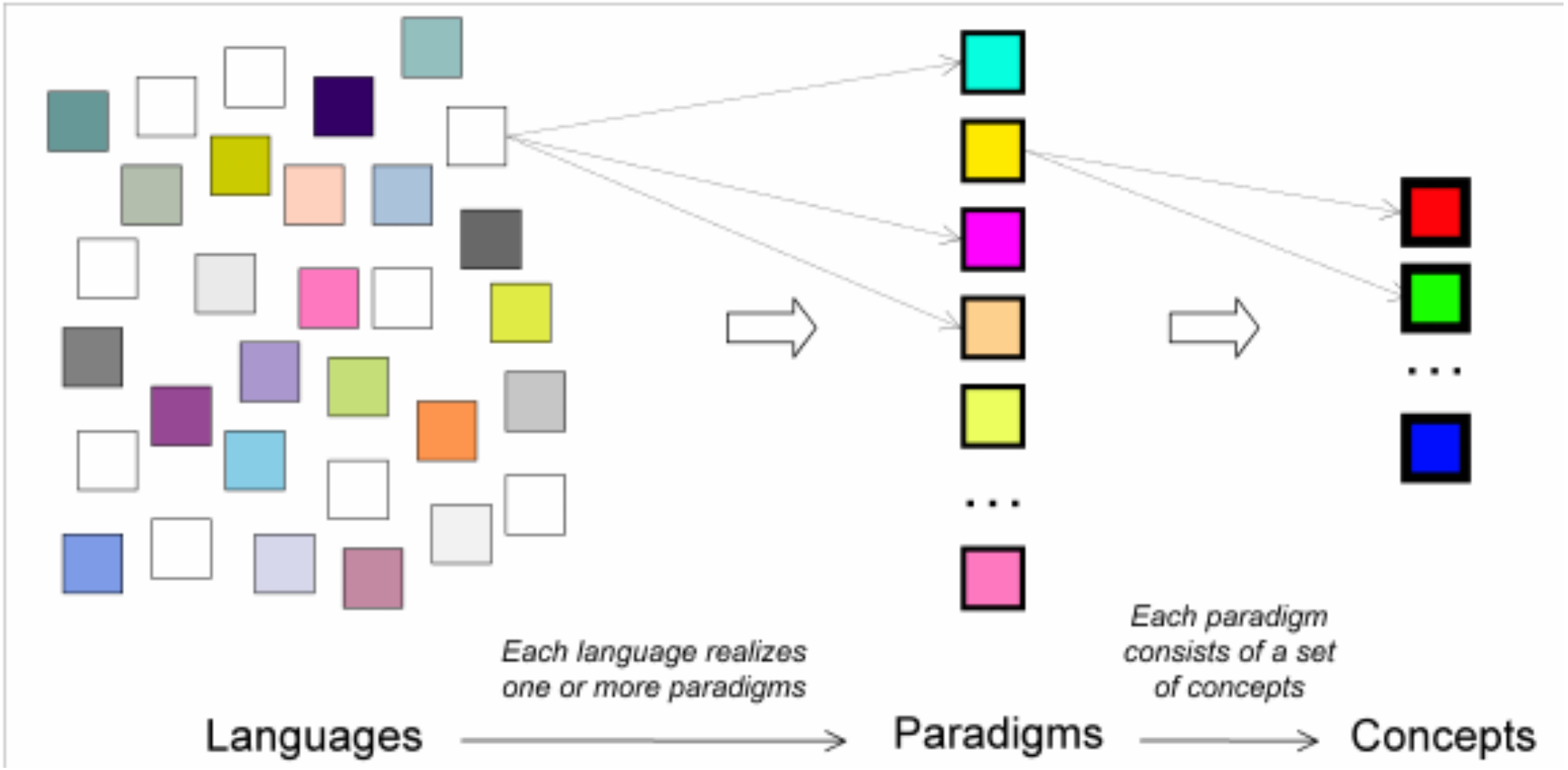
“I see that table located under the Scala’ sign is empty.

My wife and I are going to walk over there and sit down.”

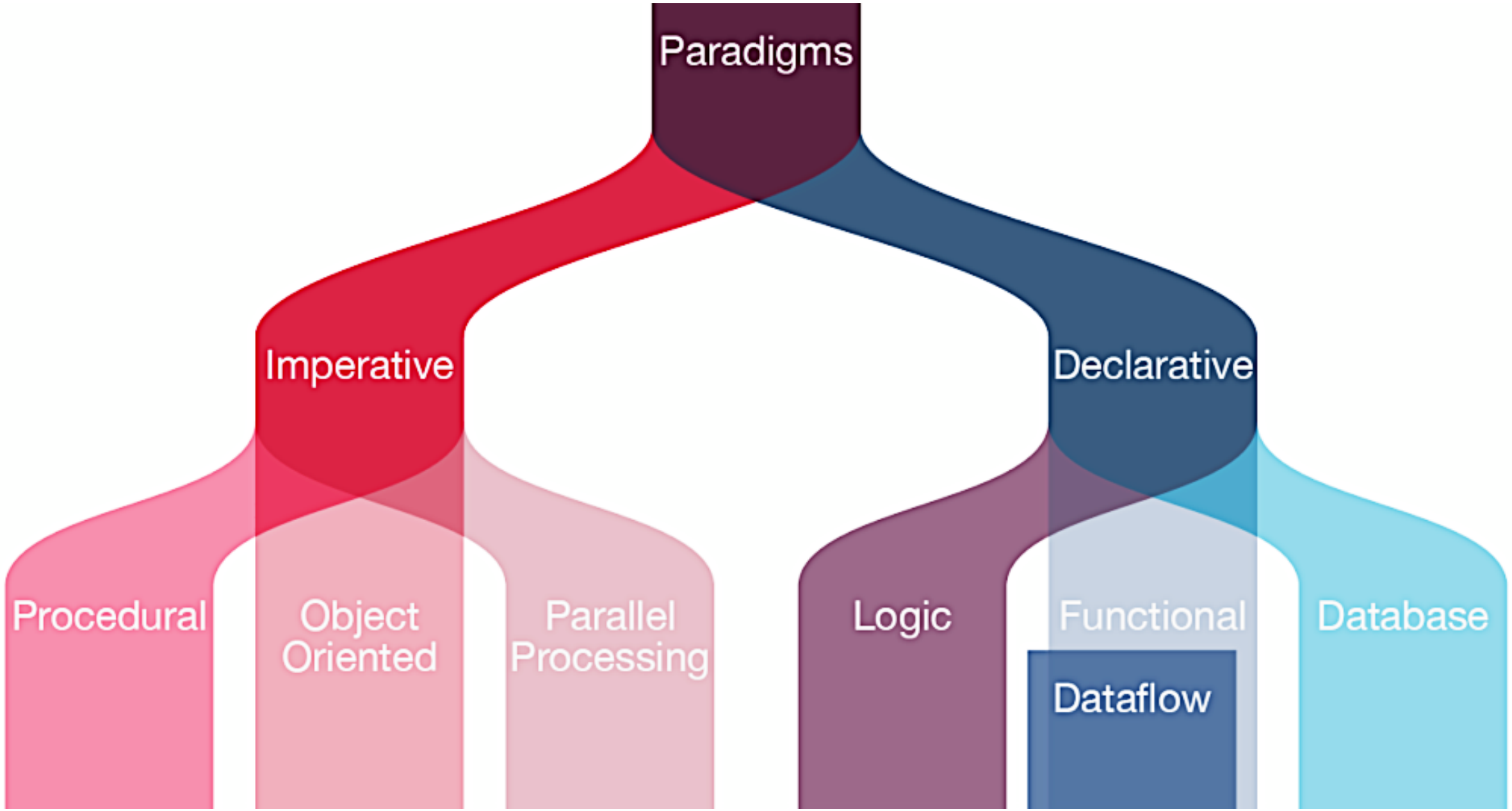
A declarative approach (WHAT):

“Table for two, please.”

Languages vs Paradigms vs Concepts



- **Structured**: Programming with clean, goto-free, nested control structures.
- **Procedural**: Imperative programming with procedure calls.
- **Functional** (Applicative): Programming with function calls that avoid any global state.
- **Function-Level** (Combinator): Programming with no variables at all.
- **Object-Oriented**: Programming by defining objects that send messages to each other. Objects have their own internal (encapsulated) state and public interfaces. Object orientation can be:
  - **Class-based**: Objects get state and behavior based on membership in a class.
  - **Prototype-based**: Objects get behavior from a prototype object.
- **Event-Driven**: Programming with emitters and listeners of asynchronous actions.
- **Flow-Driven**: Programming processes communicating with each other over predefined channels.
- **Logic** (Rule-based): Programming by specifying a set of facts and rules. An engine infers the answers to questions.
- **Constraint**: Programming by specifying a set of constraints. An engine finds the values that meet the constraints.
- **Aspect-Oriented**: Programming cross-cutting concerns applied transparently.
- **Reflective**: Programming by manipulating the program elements themselves.
- **Array**: Programming with powerful array operators that usually make loops unnecessary.


Paradigms
Imperative
Declarative
Procedural
Object Oriented
Parallel Processing
Logic
Functional
Dataflow
Database

Procedural programming paradigm

This paradigm emphasizes on procedure in terms of under lying machine model. There is no difference in between procedural and imperative approach. It has the ability to reuse the code and it was boon at that time when it was in use because of its reusability.

Examples of **Procedural** programming paradigm:

- **C** : developed by Dennis Ritchie and Ken Thompson
- **Pascal** : developed by Niklaus Wirth

Object oriented programming

The program is written as a collection of classes and object which are meant for communication. The smallest and basic entity is object and all kind of computation is performed on the objects only.

More emphasis is on data rather procedure. It can handle almost all kind of real life problems which are today in scenario.

Examples of **Object Oriented** programming paradigm:

Simula : first OOP language

Java : developed by James Gosling at Sun Microsystems

C++ : developed by Bjarne Stroustrup

Objective-C : designed by Brad Cox

Parallel processing approach

Parallel processing is the processing of program instructions by dividing them among multiple processors. A parallel processing system posses many numbers of processor with the objective of running a program in less time by dividing them. This approach seems to be like divide and conquer. Examples are NESL (one of the oldest one) and C/C++ also supports because of some library function.

Logic programming paradigms

It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc. In logic programming we have a knowledge base which we know before and along with the question and knowledge base which is given to machine, it produces result. In normal programming languages, such concept of knowledge base is not available but while using the concept of artificial intelligence, machine learning we have some models like Perception model which is using the same mechanism.

In logical programming the main emphasize is on knowledge base and the problem. The execution of the program is very much like proof of mathematical statement, e.g., Prolog

Functional programming paradigms

The functional programming paradigms has its roots in mathematics and it is language independent. The key principal of this paradigms is the execution of series of mathematical functions. The central model for the abstraction is the function which are meant for some specific computation and not the data structure. Data are loosely coupled to functions.The function hide their implementation. Function can be replaced with their values without changing the meaning of the program. Some of the languages like perl, javascript mostly uses this paradigm.

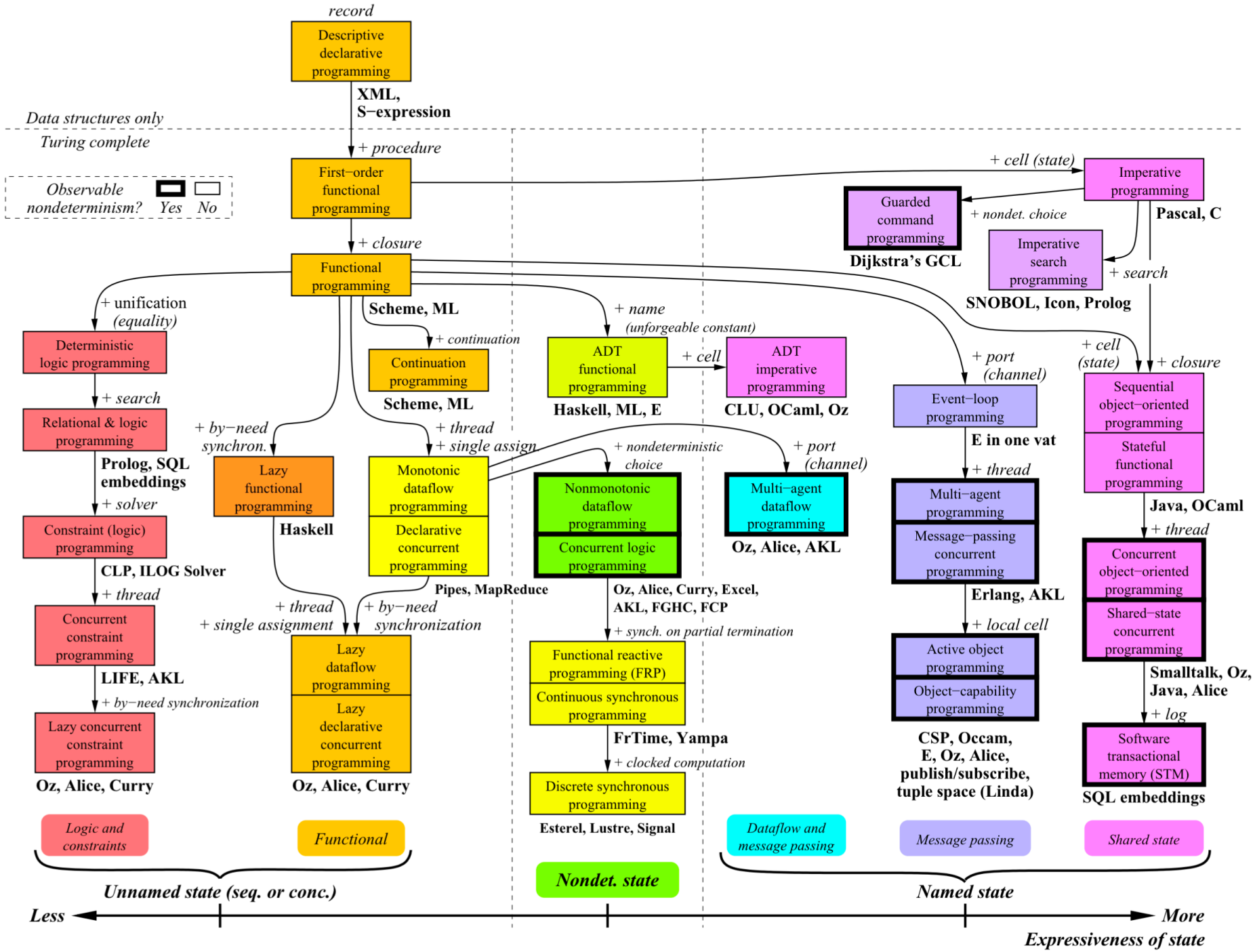
Database/Data driven programming approach

This programming methodology is based on data and its movement. Program statements are defined by data rather than hard-coding a series of steps. A database program is the heart of a business information system and provides file creation, data entry, update, query and reporting functions. There are several programming languages that are developed mostly for database application. For example SQL. It is applied to streams of structured data, for filtering, transforming, aggregating (such as computing statistics), or calling other programs. So it has its own wide application.

Application of paradigms

The paradigm defines the general approach to describing the problem.

- if I know an algorithm that is linear (or tree-like) and not very complex, then I choose a pure imperative paradigm
- the algorithm contains many repetitive fragments, it is convenient to reach for the structural / procedural paradigm
- the algorithm contains abstract classes that classify data by their properties - object-oriented paradigm
- I don't know how, but I know what I'm starting with, what I can do and what I want to get at the end - the declarative paradigm
- … and I can reduce it to a mathematical description – a declarative functional paradigm
- … and I can reduce it to Boolean expressions – a declarative logical paradigm


record
Descriptive declarative programming
XML, S−expression
Data structures only
Turing complete
+ procedure
First−order functional programming
Observable nondeterminism? Yes No
+ closure
Functional programming
+ cell (state)
Imperative programming
Pascal, C
Guarded command programming
+ nondet. choice
Dijkstra's GCL
Imperative search programming
+ search
SNOBOL, Icon, Prolog
+ unification (equality)
Scheme, ML
+ name (unforgeable constant)
+ port (channel)
+ cell (state)
+ closure
Deterministic logic programming
+ continuation
Continuation programming
Scheme, ML
ADT functional programming
+ cell
ADT imperative programming
Haskell, ML, E
CLU, OCaml, Oz
Event−loop programming
E in one vat
Sequential object−oriented programming
Stateful functional programming
Java, OCaml
+ search
Relational & logic programming
+ by−need synchron.
+ thread
+ single assign
+ nondeterministic choice
+ port (channel)
+ thread
Prolog, SQL embeddings
Lazy functional programming
Haskell
Monotonic dataflow programming
Declarative concurrent programming
Pipes, MapReduce
Nonmonotonic dataflow programming
Concurrent logic programming
Oz, Alice, Curry, Excel, AKL, FGHC, FCP
Multi−agent dataflow programming
Oz, Alice, AKL
Multi−agent programming
Message−passing concurrent programming
Erlang, AKL
+ thread
Concurrent object−oriented programming
Shared−state concurrent programming
Smalltalk, Oz, Java, Alice
+ solver
Constraint (logic) programming
CLP, ILOG Solver
+ thread
+ single assignment
+ by−need synchronization
+ synch. on partial termination
+ local cell
Concurrent constraint programming
LIFE, AKL
Lazy dataflow programming
Lazy declarative concurrent programming
Oz, Alice, Curry
Functional reactive programming (FRP)
Continuous synchronous programming
FrTime, Yampa
Active object programming
Object−capability programming
CSP, Occam, E, Oz, Alice, publish/subscribe, tuple space (Linda)
+ log
+ by−need synchronization
Lazy concurrent constraint programming
Oz, Alice, Curry
+ clocked computation
Discrete synchronous programming
Esterel, Lustre, Signal
Software transactional memory (STM)
SQL embeddings
Logic and constraints
Functional
Dataflow and message passing
Message passing
Shared state
Nondet. state
Unnamed state (seq. or conc.)
Named state
Less
More
Expressiveness of state

#only Java™

What is modern programming language?

Programming language

Traditionally it is said that a programming language must give some opportunities:

- operating on patterns
- operating on basic data structures
- execution of conditional statement
- looping or designation of recursive structures

so that any algorithm (universal language) can be used in it.

Not every language used for programming:

- markup languages (e.g. HTML)
- some scripting languages
- most macro languages
- ”programmable calculators"

Programming language is important

- The programming language is used for precise description of tasks which should be performed by the computer.
- The programming language is also used to share ideas between people. Like any language, it provides resources for verbalizing ideas about various fields
- A well-designed high programming language level facilitates both of these tasks by leading to resistant, fast and easy to understand programs.

Programming language is not important

- The programming language can be compared to a complicated tool: its effective useage requires knowledge of many details.
- However, tools in the modern world are changing very quickly.
  A programmer who knows one language can be compared to a composer who can only compose works for the viola ...
- Programming languages are evolving while basic programming concepts and techniques remain (relatively) stable.

Scala Introduction

Hello World in Java:

```
public class HelloJava {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Hello World in Scala:

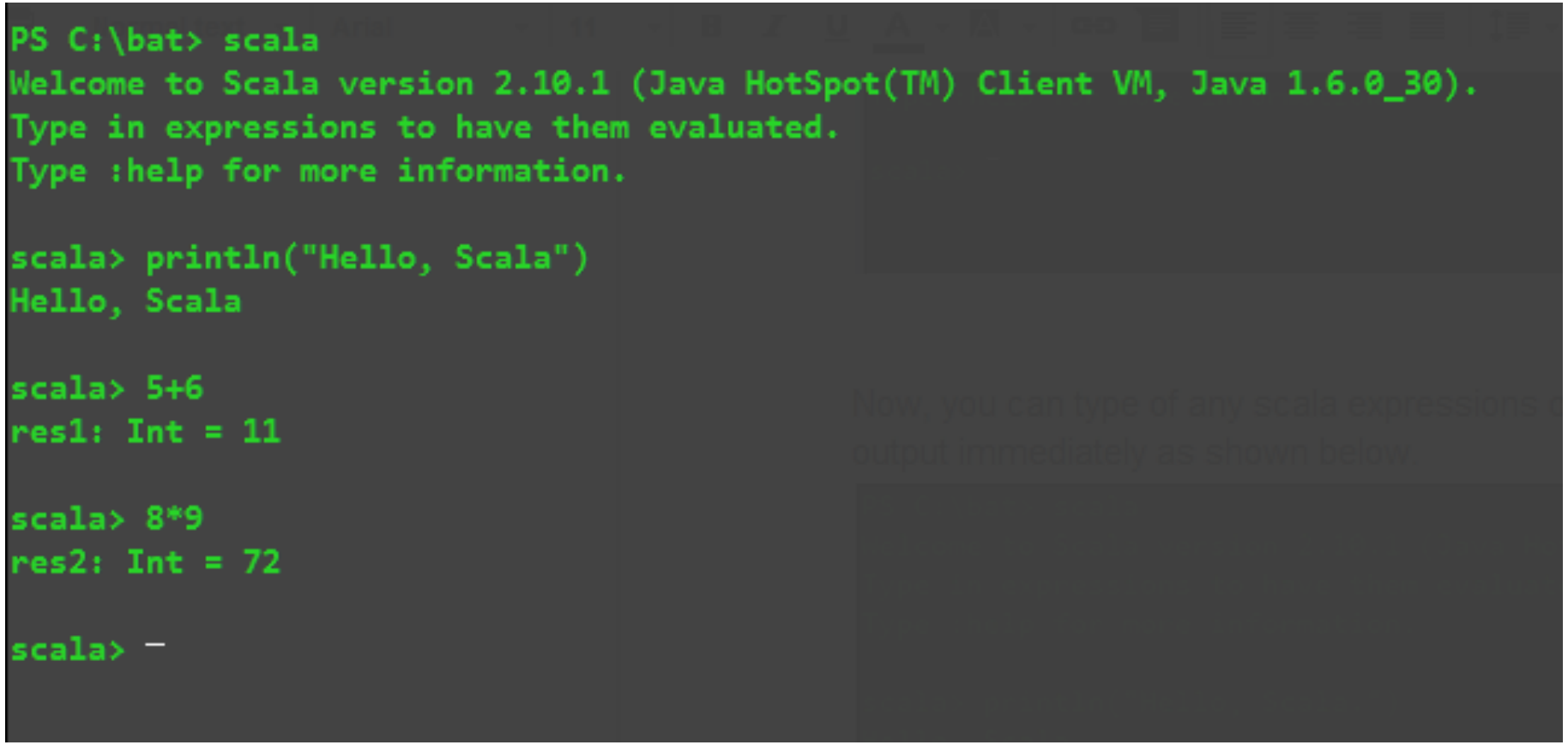
```
object HelloScala {
    def main(args: Array[String]): Unit = {
        println("Hello World!")
    }
}
```

Scala vs Java

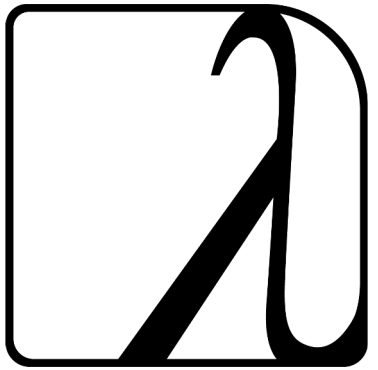
Scala has a set of features that completely differ from Java.
Some of these are:

- All types are objects
- Type inference
- Nested Functions
- Functions are objects
- Domain specific language (DSL) support
- Traits
- Closures
- Concurrency support inspired by Erlang

Cykl REPL (REPL = Read-Evaluate-Print Loop).



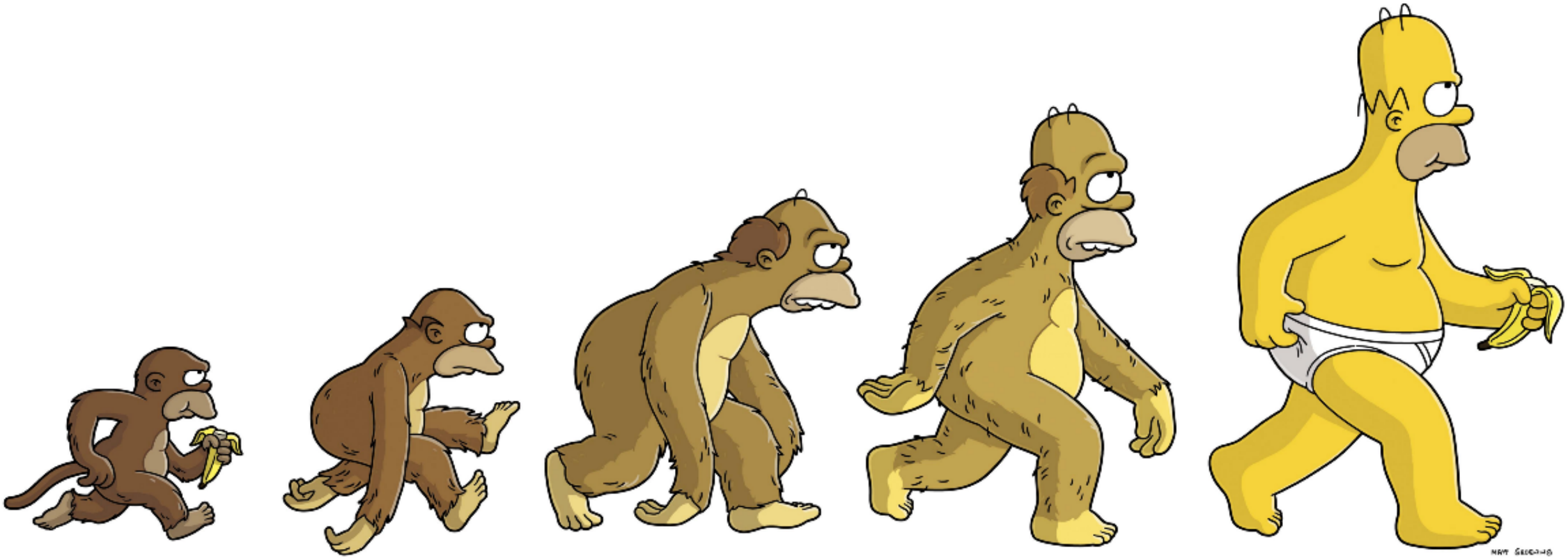
Functional programming



- The theoretical basis is lambda calculus with types
- A breakthrough in the development of the LISP language
- A variety of declarative programming
- The emphasis is on evaluating the function
- Once defined function always returns the same value for given argument values

We are changing the way of thinking

- instead of operations we have calculations of expressions
- instead of assignments we have defining constants and functions
- we have recursion instead of loops


MACHINE
ASSEMBLY
PROCEDURAL
OBJECT ORIENTED
FUNCTIONAL

A comprehensive step-by-step guide

Programming in Scala

Second Edition

Updated for Scala 2.8

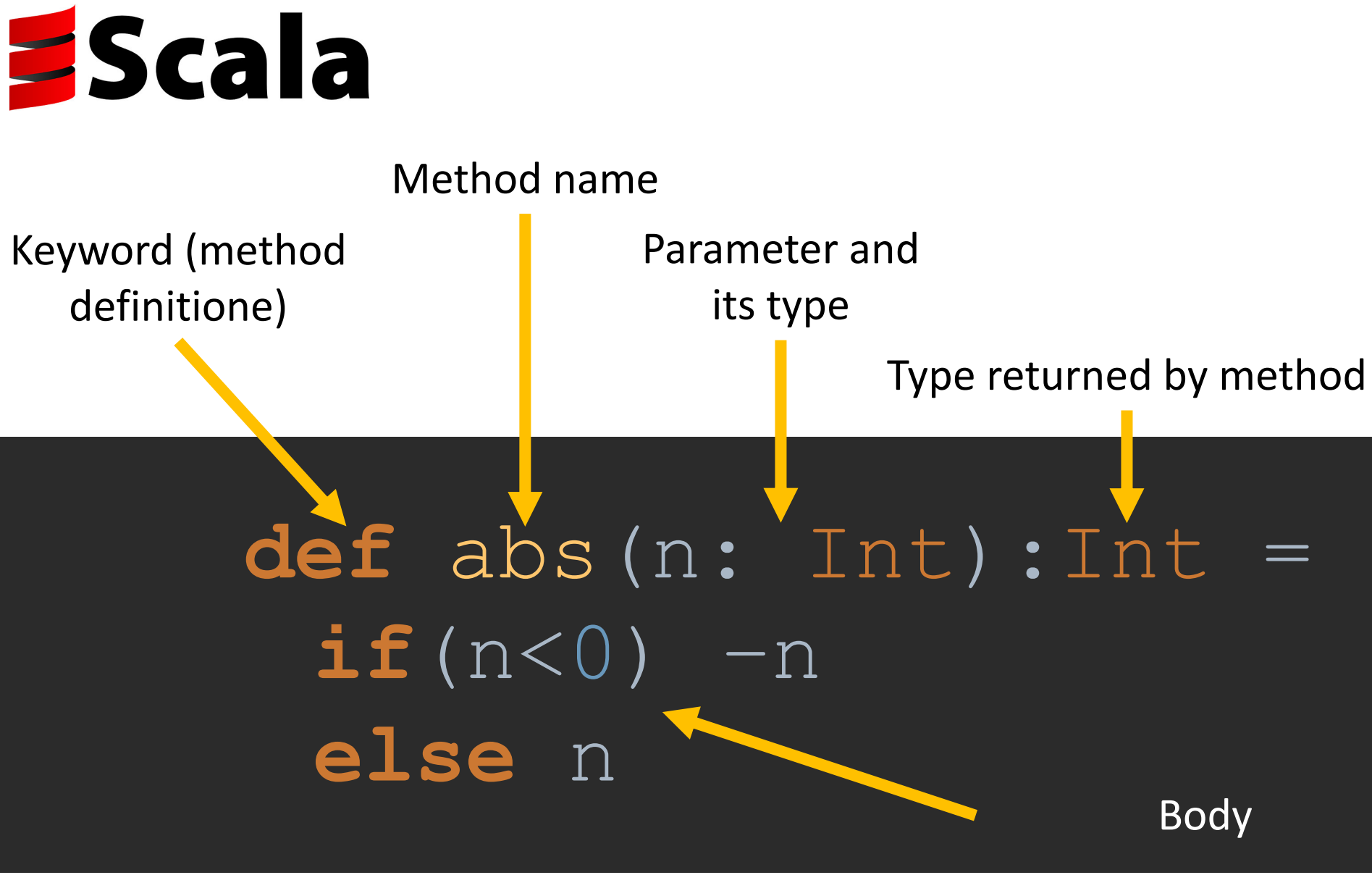
Martin Odersky
Lex Spoon
Bill Venners

artima

Functional Programming in Scala

Paul Chiusano
Rúnar Bjarnason

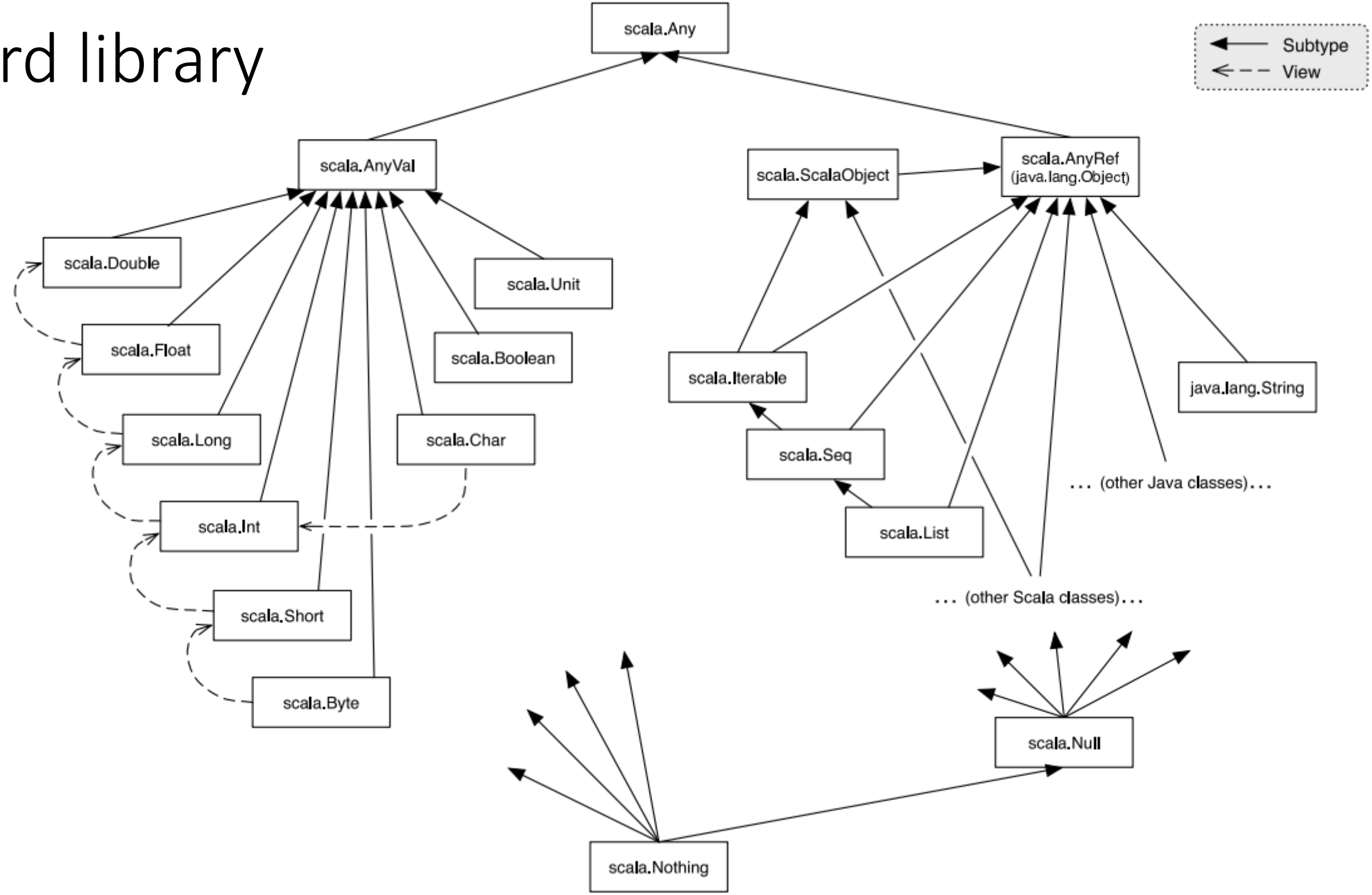
MANNING


Scala
Method name
Keyword (method definitione)
Parameter and its type
Type returned by method
def abs(n: Int):Int =
if(n<0) -n
else n
Body

Data Types

Byte - 8 bit signed value. Range from -128 to 127
Short - 16 bit signed value. Range -32768 to 32767
Int - 32 bit signed value. Range -2147483648 to 2147483647
Long - 64 bit signed value. -9223372036854775808 to 9223372036854775807
Float - 32 bit IEEE 754 single-precision float
Double - 64 bit IEEE 754 double-precision float
Char - 16 bit unsigned Unicode character. Range from U+0000 to U+FFFF
String - A sequence of Chars
Boolean - Either the literal true or the literal false
Unit - Corresponds to no value
Null - null or empty reference
Nothing - The subtype of every other type; includes no values
Any - The supertype of any type; any object is of type Any
AnyRef - The supertype of any reference type

Standard library



```
def abs(n: Int):Int =
  if(n<0) -n
  else n
```

```
def abs(n: Int)=
  if(n<0) -n
  else n
```

Anonymous function

Anonymous function:

```
((x:Int) => x+x) (2)
```

A functional expression can be associated with an identifier:

```
val double = (x:Int) => x+x
double(2)
```

```
def double(x:Int) = x+x
```

Currying:
uncurried and curried representation

Currying is the process of transforming a function that takes multiple arguments into a function that takes just a single argument and returns another function if any arguments are still needed.

```
def plus(x:Int, y:Int) = x+y
plus(2,1)

def plus2(x:Int)(y:Int) = x+y
plus2(2)(1)
```

Recursive

Mutual recursion:
two or more functions call each other.

```
def evenR(n: Int): Boolean =
  if (n==0) true else oddR(n-1)
def oddR(n: Int) :Boolean =
  if (n==0) false else evenR(n-1)
```


DEMO
TIME

michal.szczepanik@pwr.edu.pl

Useful links

- Scala and online compiler
  https://www.scala-lang.org/
- Compilers for other languages
  https://www.jdoodle.com/