Introduction to Scala

cerenode.io

About Us

- Data Science Company
- Build web-apps to visualize graph data
  - Semantic search engine, Analytics Dashboard
- Data Ingestion Engine (ETL)
- Stack
  - Scala, AngularJS, D3.js, Graph Databases, Elasticsearch

Program Outline

- Scala Introductions
- Setup Scala
- Basic Data Types and Operations
- Scala Collections
- Scala OOP Concepts
- Scala as a Functional Language
- Scala Functions, Higher Order Functions, Lambda and Closures
- Pattern Matching
- Type Classes
- Scala Futures And Concurrency
- Scala Async And Akka

History

- Designed by Prof. Martin Odersky
- Initial release 2003
- Latest version is 2.12.6
- Commercial support for Scala provided by Lightbend Inc. (formally known as Typesafe)

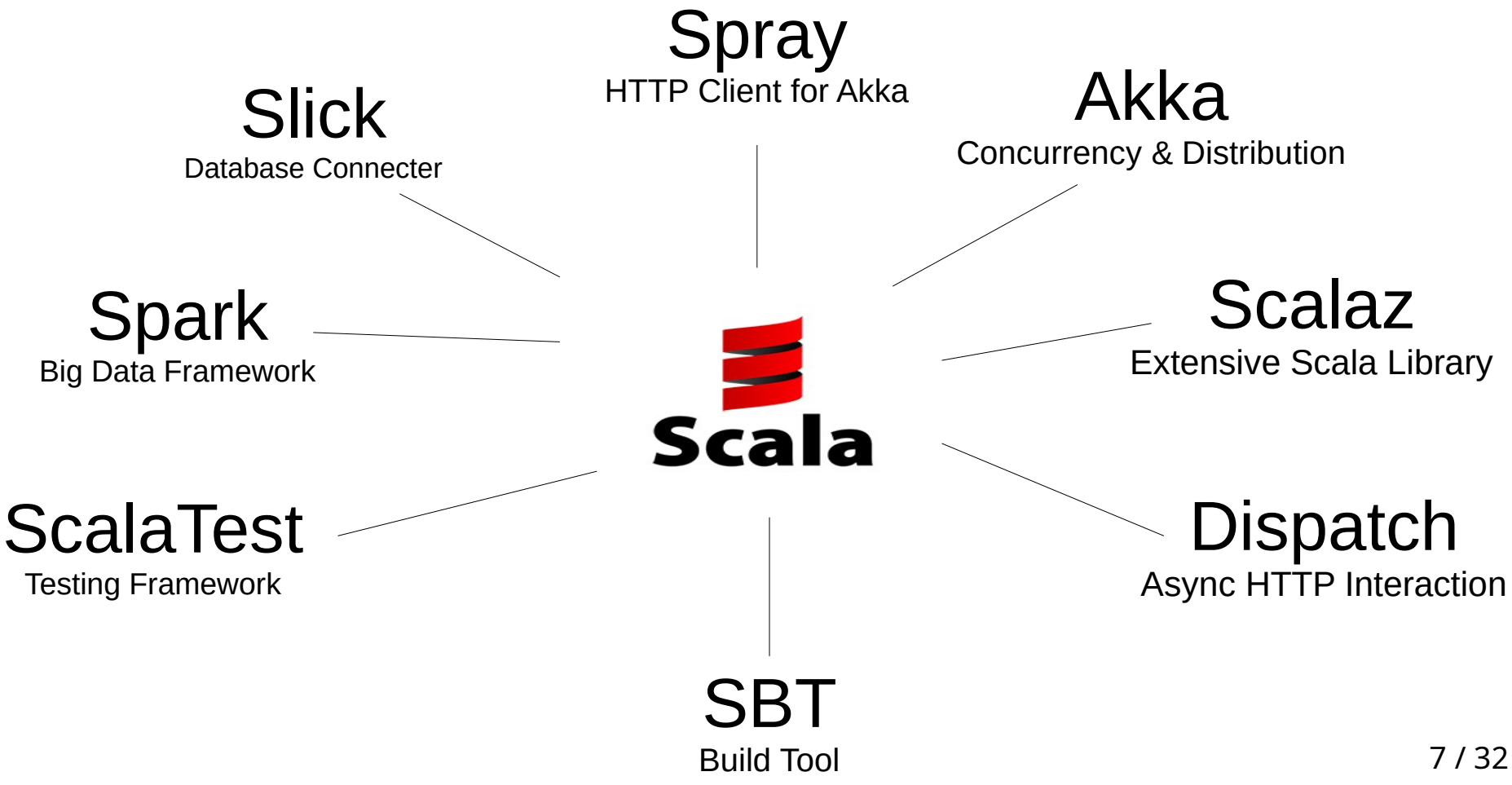
What is Scala?

- Scala is a modern multi-paradigm programming language
  - Object Oriented
  - Functional
- Concise and elegant
- Statically typed language
- Compiles to byte code and runs on top of JVM
- Interoperable with Java

Scalable Language

- Growable
  - Used to create small scripts to enterprise size projects
- Extensible
  - Can be moulded into languages (DSLs)

Extensible


Spray
HTTP Client for Akka
Slick
Database Connecter
Akka
Concurrency & Distribution
Spark
Big Data Framework
Scala
Scalaz
Extensive Scala Library
ScalaTest
Testing Framework
Dispatch
Async HTTP Interaction
SBT
Build Tool

Who is using Scala?

Object Oriented Aspects

- Classes and Objects
- Abstraction
- Inheritance
- Polymorphism
- Encapsulation

Functional Aspects

- Functions are first-class
  - functions are treated as objects
- Higher order functions
  - pass functions as parameters to another function
- Immutability
  - usage of variables that don’t change value
- Pattern Matching
  - similar to ‘switch‘ statements in C
- Composing functions
  - f(x) compose g(x) = f(g(x))

Advantages

- Higher order functions
- Type inference
- Concurrency
- Lazy computation
- Less boilerplate
- Good IDE support
- Open source libraries
- Performance
- JVM Tools
- Scriptable
- **R**ead **E**val **P**rint **L**oop

Concurrency and Parallelism

- Data-parallel operation on collections (eg. List, Array, etc.)
- Actors for concurrency
  - Actors are concurrent processes
  - Communicate using messages
- Futures for asynchronous programming
  - Futures are used to store values returned by an asynchronous function

Read Eval Print Loop

- Interactive Interpreter
- Evaluates expression in Scala

```
scala> 1 + 2
res4: Int = 3

scala>
```

Expressions

- Single unit of code that returns a value
- Computable statements

```
1 + 1
“Hello” + “ World”
```

Datatypes

- Scala has no primitives, only objects

- 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

Datatypes

- 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

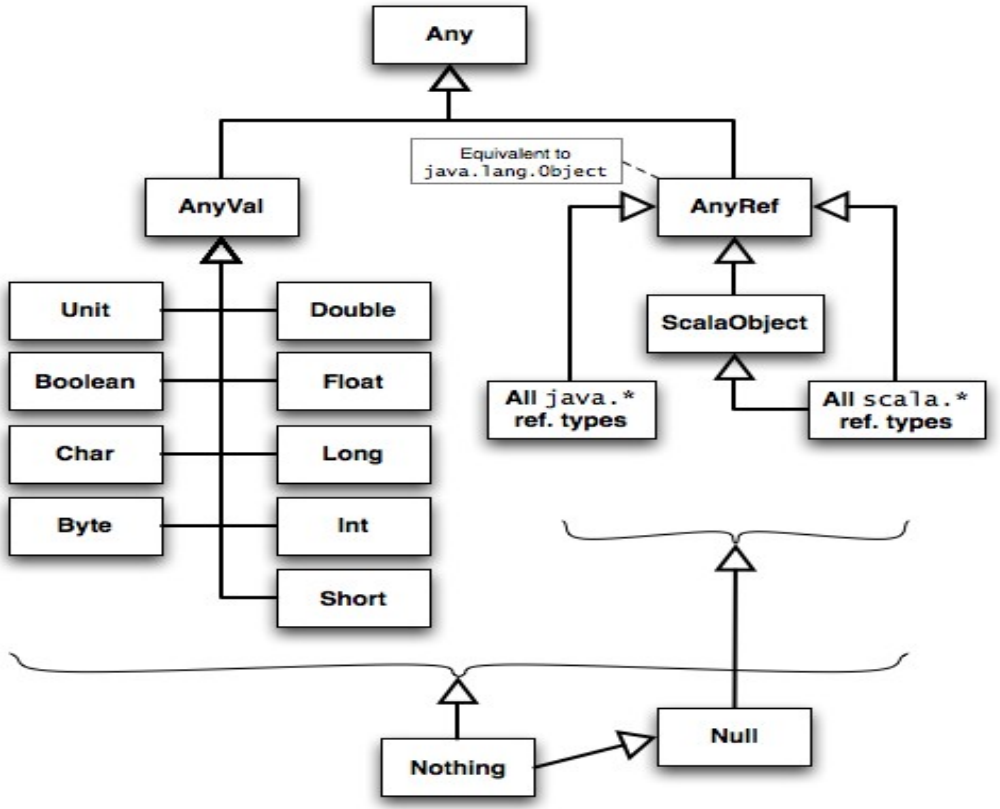
- Any

  The supertype of any type; any object is of type Any

- AnyRef

  The supertype of any reference type

Datatype Hierarchy



Identifiers

- Identifiers are names of variables, classes, objects and methods
- Should start with letter or an underscore, followed by letters, digits or underscores

```
stmt, _thrice, new_variable_1    // valid
45th, 4_9_3, abc$@def            // invalid
```

- Case-sensitive

  `pi, PI, Pi` are all different identifiers

- Keyword cannot be used as an identifier

Keywords

abstract	false	lazy	protected	true
case	final	match	return	type
catch	finally	new	sealed	val
class	for	Null	super	var
def	forSome	object	this	while
do	if	override	throw	with
else	implicit	package	trait	yield
extends	import	private	try	-
:	=	=>	<-	<:
<%	>:	#	@	

Variables and Values

- Variables and values are used to name results of expressions
- Syntax:

```
var | val <identifier> : <type> = <expression>
```

- Variable

```
var k: String = “code”
k = “something”   // valid
```

- Value

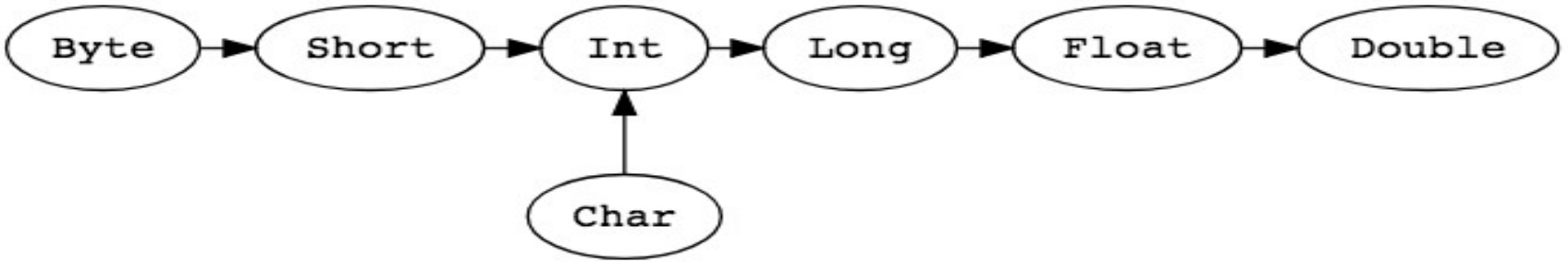
```
val k: Int = 1
k = 2    //invalid
```

Type Inference

- Compiler often infers the type of an expression without explicitly specifying the type

```
val sum = 1 + 2 + 3
val welcomeMsg = “Hello”
```

Type Casting



- Example

```
val m: Int = 7
val n: Long = m  // valid
```

- Casting is unidirectional

```
val p: Short = m // invalid
```

String Interpolation

- Embed variables inside strings
- `s` interpolator
- `f` interpolator
- `raw` interpolator
- User defined interpolator

Format specifiers for f interpolator

Format specifier	Description
%c	Character
%d	Decimal number (integer, base 10)
%e	Exponential floating-point number
%f	Floating-point number
%i	Integer (base 10)
%o	Octal number (base 8)
%s	A string of characters
%u	Unsigned decimal (integer) number
%x	Hexadecimal number (base 16)
%%	Print a “percent” character
\%	Print a “percent” character

Block

- Multiple expressions are combined using ‘{‘ and ‘}’
- The last expression in the block is the return value for the entire block

```
val stmt = {
      val x = 5 * 10
      x + 20
   }
// stmt: Int = 70
```

Conditional Expressions

- IF...ELSE structure

```
if(<Boolean_expression>) {
   // execute if <Boolean_expression> is true
} else {
   // execute if <Boolean_expression> is false
}
```

Less Boilerplate

```
public class Person {
    private String name;
    private Int age;
    public Person(String name, Int age)
    {
        this.name = name;
        this.age = age;
    }
    public Int getAge() {
        return age;
    }
    ....
}
```

The Scala Way

```
class Person(val name: String, val age: Int)
```

- No need of special getter or setter functions
- Accessing properties of objects in Scala

```
val p1 = new Person(“John”, 24)
println(p1.name)     // prints the name ‘John’
```

Install Scala

- Scala 2.12.x requires Java 8 JDK

  http://www.oracle.com/technetwork/java/javase/downloads/index.html

- Download it from

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

Install IntelliJ

- IntelliJ requires Java 8 JDK
- Download IntelliJ installation file from

  https://www.jetbrains.com/idea/download

  https://www.jetbrains.com/help/idea/installing-updating-and-uninstalling-repository-plugins.html
- Setup first Scala project

  http://allaboutscala.com/tutorials/chapter-1-getting-familiar-intellij-ide/scala-tutorial-first-hello-world-application/

Functional Programming Approach

- Immutability
- Concise code
- Assigning a block to a val

Scala Best Practices

- val vs var
- Naming of variables
  - test, nextInt, supportedDocSet