Composing Functions with Expressions

Vitthal Srinivasan
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

Refresh basics of writing Scala code

Test conditions with if/else expressions

Iterate with for loops and while loops

Match patterns in many different ways

Jogging Through Basics

Let's Refresh Some **Basics of Scala**

- Types and type inference
- String interpolation
- Immutable and mutable data
- Return values from expression blocks

Demo

Utilise Type Inference to avoid specifying types of values and variables

Implications of Type Inference

Variable types

Function return types

Type parameters to generic functions

It's often OK to not specify these types - Type Inference will kick in

Why Type Inference Matters

Statically Typed Languages

Java, C++, C#

Variable types specified in code, known at compile-time

Code is more verbose

Type-related bugs caught at compile-time

Dynamically Typed Languages

Python, Javascript

Variable types not specified, only known at run-time

Code is easier to write, more concise

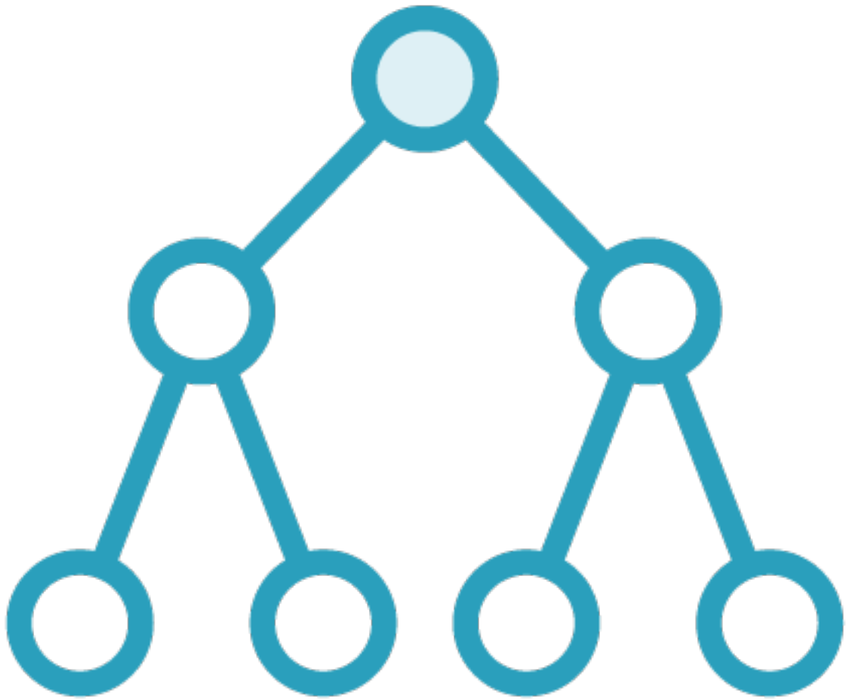
Type-related bugs can cause nasty run-time issues

Why Type Inference Matters

Scala achieves the best of both worlds through Type Inference

Scala has a Unified Type System - everything is an instance of a class

Unified Type System

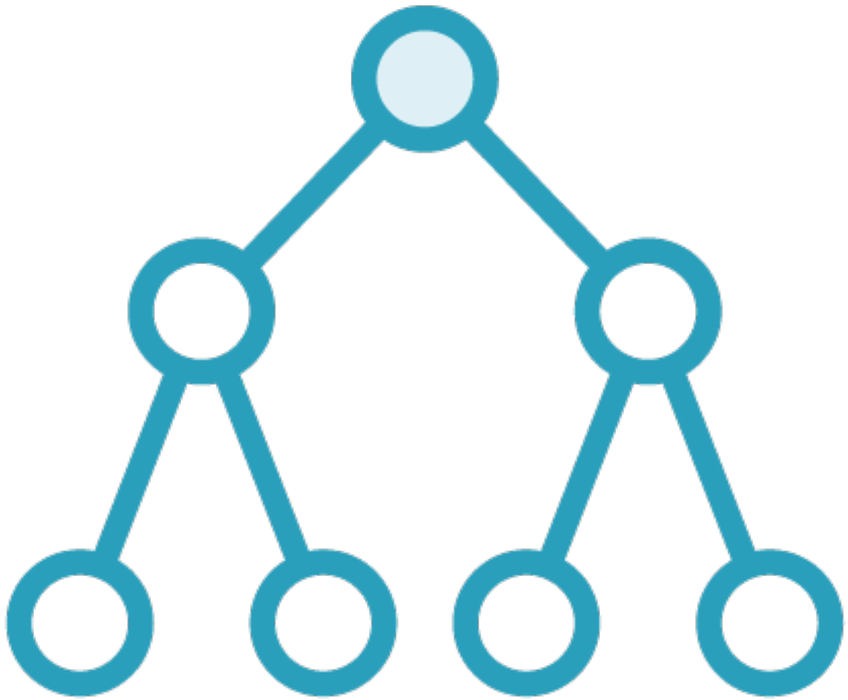


All values and variables are instances of a class - no exceptions

To run on the JVM, the distinction between value and reference types must still exist

Scala squares the circle by creating a Unified Type System

Unified Type System



Any is the universal base class in Scala

AnyVal descends from Any and is the base for all Java value types

AnyRef also descends from Any and is the base for all Java reference types

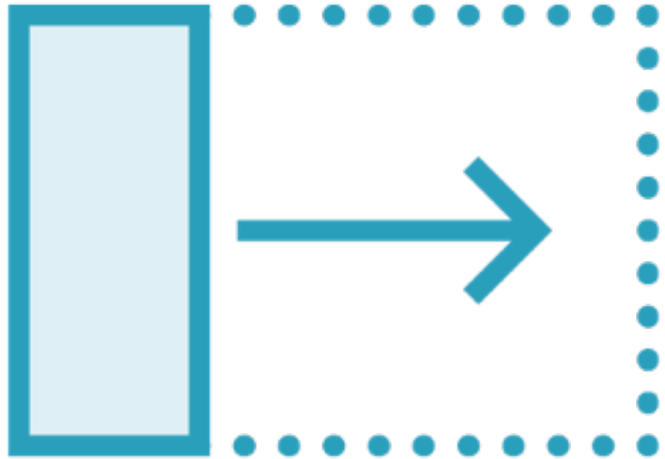
Demo

Any is a type from which all values and variables descend

AnyVal is the common base class for Java value types

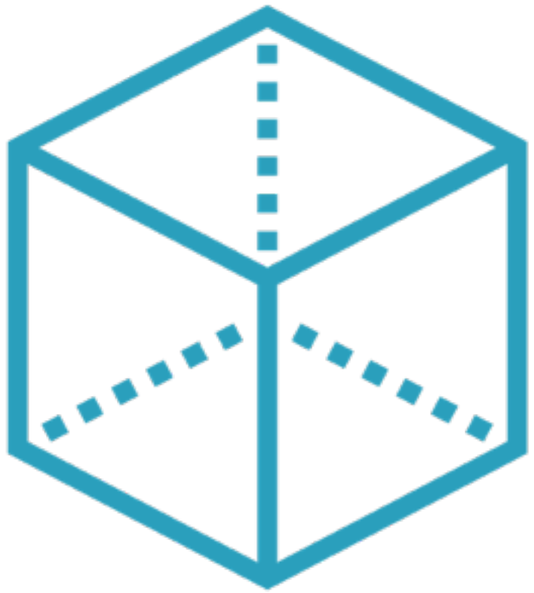
AnyRef is the common base class for Java reference types

AnyVal and AnyRef



AnyVal

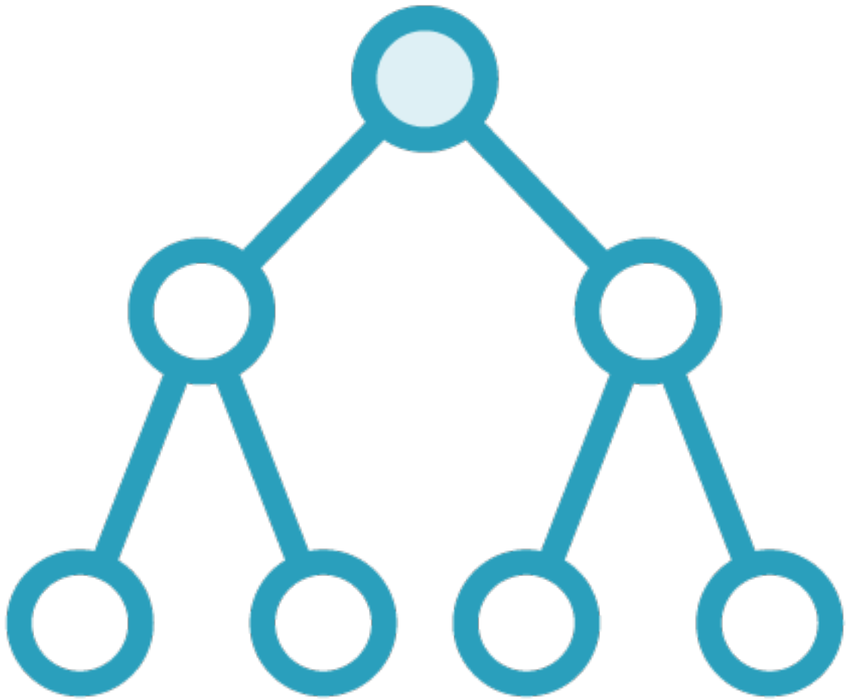
Base class for int, boolean, double, char



AnyRef

Base class for classes, strings, collections, arrays

Emptiness in Scala



null is the same as null in Java

Null is the type of null, and descends from AnyRef

Nothing is a type that extends Any

Nil is a singleton List[Nothing]

Unit is the same as void in Java

Let's Refresh Some **Basics of Scala**

- Types and type inference
- String interpolation
- Immutable and mutable data
- Return values from expression blocks

Demo

String interpolation in Scala

Let's Refresh Some **Basics of Scala**

- Types and type inference
- String interpolation
- Immutable and mutable data
- Return values from expression blocks

Demo

Prefer immutable data

Let's Refresh Some **Basics of Scala**

- Types and type inference
- String interpolation
- Immutable and mutable data
- Return values from expression blocks

Demo

Last expression in a block is its return value

The last expression in a block is the return value for that block

If/else Expressions

If/Else Expression Blocks

```
if (boolean expression)
    {expression block #1}
else
    {expression block #2}
```

An expression block made of expression blocks

If/Else Expression Blocks

```
if (boolean expression)
    {expression block #1}
else
    {expression block #2}
```

The return value of the if/else expression is that of expression block #1 if the boolean expression evaluates to true

If/Else Expression Blocks

```
if (boolean expression)
    {expression block #1}
else
    {expression block #2}
```

The return value of the if/else expression is that of expression block #2 if the boolean expression evaluates to false

If Expression Blocks

```
if (boolean expression)
    {expression block #1}
```

What is the return value if the condition evaluates to false? `Nothing`

Demo

If/else expressions are expression blocks like any others

For Loops and While Loops

Demo

For loops can be statements...

...Or expressions

Demo

For loop iterators can be

- Value bindings
- Numeric ranges

Demo

For loops can include conditions called Pattern Guards

Demo

For loops can be nested more concisely than in Java

While Loops

While loops are statements - no value returned

Loop variable needs to be mutable

Value binding needs to be explicit

Cannot be chained or composed

Infrequently used due to clunky syntax

Demo

While loops are syntactically awkward and seldom used

Pattern Matching

Pattern Matching

Pattern matches are similar to switch statements

No fall-through

No breaks

Matches can be on type, value or condition

Demo

Pattern matches are similar to switch statements in Java

Demo

Conditions inside an individual case can be specified using

- Pattern guards
- OR-ed expressions

Demo

Scala.MatchError results if no case clause is matched

Specify a catch-all to avoid this, using

- Value bindings
- The wildcard operator _

Demo

Match expressions can be used to perform safe downcasts

Course Outline: Think Functional, Talk Functional

Strong Basics

Simple constructs have a functional twist in Scala

Functions

Play in the big leagues in Scala, on par with objects

Collections

The pipes and links in functional chains

Summary

Used if/else expressions to initialise values and variables

Iterated over ranges with for loops

Understood why while loops are best avoided

Matched several complex patterns