



Scala Syntax Summary

Principles of Functional Programming

Language Elements Seen So Far:

We have seen language elements to express types, expressions and definitions.

Below, we give their context-free syntax in Extended Backus-Naur form (EBNF), where

- | denotes an alternative,

- [...] an option (0 or 1),

- {...} a repetition (0 or more).

Types

```
Type          = SimpleType | FunctionType
FunctionType  = SimpleType '=>' Type
               | '(' [Types] ')' '=>' Type
SimpleType    = Ident
Types         = Type {', ' Type}
```

A *type* can be:

- ▶ A *numeric type*: Int, Double (and Byte, Short, Char, Long, Float),
- ▶ The Boolean type with the values true and false,
- ▶ The String type,
- ▶ A *function type*, like Int => Int, (Int, Int) => Int.

Later we will see more forms of types.

Expressions

```
Expr          = InfixExpr | FunctionExpr
               | if Expr then Expr else Expr
InfixExpr     = PrefixExpr | InfixExpr Operator InfixExpr
Operator      = ident
PrefixExpr    = ['+' | '-' | '!' | '~' ] SimpleExpr
SimpleExpr    = ident | literal | SimpleExpr '.' ident
               | Block
FunctionExpr  = Bindings '=>' Expr
Bindings      = ident
               | '(' [Binding {',' Binding}] ')'
Binding       = ident [':' Type]
Block         = '{' {Def ';' } Expr '}'
               | <indent> {Def ';' } Expr <outdent>
```

Expressions (2)

An *expression* can be:

- ▶ An *identifier* such as `x`, `isGoodEnough`,
- ▶ A *literal*, like `0`, `1.0`, `"abc"`,
- ▶ A *function application*, like `sqrt(x)`,
- ▶ An *operator application*, like `-x`, `y + x`,
- ▶ A *selection*, like `math.abs`,
- ▶ A *conditional expression*, like `if x < 0 then -x else x`,
- ▶ A *block*, like `{ val x = abs(y) ; x * 2 }`
- ▶ An *anonymous function*, like `x => x + 1`.

Definitions

```
Def           = FunDef   |   ValDef
FunDef        = def ident { '(' [Parameters] ')' }
               [ ':' Type ] '=' Expr
ValDef        = val ident [ ':' Type ] '=' Expr
Parameter     = ident ':' [ '=>' ] Type
Parameters    = Parameter { ',', Parameter }
```

A *definition* can be:

- ▶ A *function definition*, like `def square(x: Int) = x * x`
- ▶ A *value definition*, like `val y = square(2)`

A *parameter* can be:

- ▶ A *call-by-value parameter*, like `(x: Int)`,
- ▶ A *call-by-name parameter*, like `(y: => Double)`.