

 무료 전자 책

배우기

Scala Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#scala

.....	1
1:	2
.....	2
.....	2
Examples.....	2
" Hello World.....	2
Hello World	3
.....	3
.....	4
Hello World	4
REPL	5
Scala Quicksheet.....	5
2: Implicits	7
.....	7
.....	7
Examples.....	7
.....	7
.....	7
.....	8
"	9
REPL	9
3: Java	11
Examples.....	11
Java	11
.....	11
Java	11
- scala-java8-compat.....	12
4: JSON	14
Examples.....	14
JSON with spray-json.....	14
SBT	14
.....	

JSON	14
JSON	14
DSL	14
-	14
.....	15
Circe JSON	15
JSON with play-json	16
JSON json4s	19
5: Quasiquotes	21
Examples	21
quasiquotes	21
6: Scala.js	22
.....	22
Examples	22
Scala.js console.log	22
.....	22
.....	22
.....	22
DOM	22
SBT	22
Sbt	22
.....	22
:	23
JavaScript :	23
7: ScalaCheck	24
.....	24
Examples	24
Scalacheck	24
8: Scaladoc	27
.....	27
.....	

Examples.....	27
Scaladoc.....	27
9: ScalaTest	29
Examples.....	29
Hello World Spec	29
Cheatsheet.....	29
SBT ScalaTest	30
10: While	31
.....	31
.....	31
.....	31
Examples.....	31
While	31
Do-While	31
11: XML	32
Examples.....	32
Beautify Pretty-Print XML.....	32
12:	33
.....	33
.....	33
.....	33
Examples.....	33
Continuations.....	33
.....	34
13:	35
.....	35
Examples.....	35
.....	35
()......	35
.....	36

14:	37
.....	37
:	37
Examples.....	37
.....	37
.....	37
.....	38
.....	38
PartialFunctions	38
15:	39
.....	39
Examples.....	39
case	39
16: (SAM)	40
.....	40
Examples.....	40
.....	40
17:	41
.....	41
Examples.....	41
.....	41
.....	41
18:	42
.....	42
.....	42
.....	42
Examples.....	42
.....	42
.....	42
.....	43
19:	44
.....	

.....	44
.....	44
Examples.....	44
.....	44
.....	44
.....	45
20:	47
Examples.....	47
.....	47
21:	49
.....	49
Examples.....	49
.....	49
.....	49
.....	50
22:	51
.....	51
Examples.....	51
Hello	51
f	51
.....	51
.....	52
.....	52
.....	53
23:	54
Examples.....	54
.....	54
24:	55
.....	55
.....	55
Examples.....	55

0	55
.....	55
.....	55
.....	56
.....	56
.....	56
25:	57
.....	57
.....	57
Examples.....	57
val var	57
val var.....	57
.....	57
!.....	58
" ?".....	58
result	58
.....	59
.....	59
.....	59
.....	59
26: , Def	60
.....	60
Examples.....	60
, Def.....	60
var	60
.....	60
def	61
.....	61
.....	62
"	62
.....	63
.....	63

27:	65
.....	65
Examples.....	65
.....	65
.....	65
28:	67
Examples.....	67
.....	67
`collect`	67
.....	68
.....	68
.....	69
29:	70
.....	70
Examples.....	70
.....	70
.....	70
.....	71
Immutability.....	72
.....	73
.....	73
30:	74
Examples.....	74
.....	74
.....	74
.....	74
.....	75
.....	75
-	75
31:	77
.....	77
.....	77

Examples.....	77
.....	77
32:	78
.....	78
Examples.....	78
ApplyUsage.....	78
FunctorUsage.....	78
ArrowUsage.....	78
33:	80
Examples.....	80
dpkg Linux	80
.....	80
OSX	81
34:	82
Examples.....	82
.....	82
.....	82
.....	82
35:	84
.....	84
Examples.....	84
.....	84
.....	84
- referent	85
36:	86
Examples.....	86
.....	86
.....	86
37:	88
.....	88
Examples.....	88
.....	88

.....	89
allValues	90
38:	92
Examples.....	92
.....	92
.....	92
.....	92
.....	93
map getOrElse	93
.....	93
Java	93
.....	93
try-catch	94
Either Option	94
39:	95
.....	95
Examples.....	95
.....	95
Null Option	95
.....	96
.....	96
.....	96
40:	98
Examples.....	98
.....	98
Gradle Scala	98
.....	98
.....	102
41: 0	103
Examples.....	103
.....	103
.....	103

.....	103
Ints	103
.....	103
42:	105
Examples.....	105
.....	105
43:	106
Examples.....	106
.....	106
.....	106
.....	106
.....	107
.....	107
44:	109
Examples.....	109
.....	109
.....	109
.....	109
.....	109
45:	111
.....	111
Examples.....	111
.....	111
.....	111
.....	112
46:	114
Examples.....	114
Cake Pattern.....	114
47:	115
Examples.....	115
.....	115
.....	115

.....	115
(scala.util.control.TailCalls).....	116
48:	117
.....	117
Examples.....	117
.....	117
.....	117
49:	119
.....	119
Examples.....	119
.....	119
.....	119
-	120
.....	121
.....	121
.....	122
50: ().....	123
.....	123
.....	123
Examples.....	123
.....	123
.....	123
51:	124
.....	124
Examples.....	124
.....	124
,	124
.....	124
.....	124
.....	125
Currying	125
Currying	127

52:	128
Examples	128
.....	128
n	128
Cheatsheet	129
Cheatsheet	129
.....	130
.....	130
2	130
.....	130
.....	131
.....	131
.....	131
.....	132
.....	132
.....	133
.....	133
53:	135
.....	135
Examples	135
.....	135
: {} vs ()	135
.....	136
.....	136
.....	137
.....	137
.....	138
.....	139
.....	139
.....	140
54:	141
.....	141

Examples.....	141
.....	141
.....	141
55:	143
.....	143
.....	143
.....	143
Examples.....	143
.....	143
.....	143
.....	144
56:	145
.....	145
Examples.....	145
.....	145
57:	146
.....	146
Examples.....	146
.....	146
.....	146
.....	146
58:	148
.....	148
.....	148
Examples.....	148
.....	148
.....	149
Seq	149
()......	150
.....	151
.....	151
.....	151

.....	152
(@).....	152
.....	152
tableswitch lookupswitch	153
.....	154
.....	155
59:	156
Examples.....	156
If	156
60:	157
.....	157
.....	157
Examples.....	157
For	157
.....	157
Nested For Loop.....	157
.....	158
For	159
Desugaring.....	159
61:	161
Examples.....	161
Apache Spark UDF.....	161
62:	162
.....	162
Examples.....	162
.....	162
.....	162
.....	163
.....	164
.....	166

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [scala-language](#)

It is an unofficial and free Scala Language ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Scala Language.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1:

(Scala)

scalafiddle.net

2.10.1	2013-03-13
2.10.2	2013-06-06
2.10.3	2013-10-01
2.10.4	2014-03-24
2.10.5	2015-03-05
2.10.6	2015-09-18
2.11.0	2014-04-21
2.11.1	2014-05-21
2.11.2	2014-07-24
2.11.4	2014-10-30
2.11.5	2014-01-14
2.11.6	2015-03-05
2.11.7	2015-06-23
2.11.8	2016-03-08
2.11.11	2017-04-19
2.12.0	2016-11-03
2.12.1	2016-12-06
2.12.2	2017-04-19

Examples

" Hello World

HelloWorld.scala HelloWorld.scala .

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

JVM :

```
$ scalac HelloWorld.scala
```

:

```
$ scala Hello
```

```
main Hello .main .
```

```
, ,., Hello scala Hello main . .scala main .
```

```
args (). .
```

```
object HelloWorld {
  def main(args: Array[String]): Unit = {
    println("Hello World!")
    for {
      arg <- args
    } println(s"Arg=$arg")
  }
}
```

:

```
$ scalac HelloWorld.scala
```

:

```
$ scala HelloWorld 1 2 3
Hello World!
Arg=1
Arg=2
Arg=3
```

Hello World

```
object HelloWorld extends App {
  println("Hello, world!")
}
```

```
App main .HelloWorld "".
```

2.11.0

, App *Delayed Initialization* . , .

2.11.0

, App *Delayed Initialization* . , .

DelayedInit App . .

App this.args this.args .

```
object HelloWorld extends App {  
  println("Hello World!")  
  for {  
    arg <- this.args  
  } println(s"Arg=$arg")  
}
```

App , main , (override) main .

Hello World

Scala . HelloWorld.scala .

```
println("Hello")
```

(\$).

```
$ scala HelloWorld.scala  
Hello
```

.scala .scala (: scala HelloWorld) .class .

: scala .

bash ' ' . HelloWorld.sh .

```
#!/bin/sh  
exec scala "$0" "$@"  
!#  
println("Hello")
```

#! #! !# ' , bash . .

' . .

```
$ chmod a+x HelloWorld.sh
```

(: *chmod* *chmod* .)

.

```
$ ./HelloWorld.sh
```

REPL

scala **REPL** (Read-Eval-Print Loop) .

```
nford:~ $ scala
Welcome to Scala 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_66).
Type in expressions for evaluation. Or try :help.

scala>
```

REPL . , val poem = "As halcyons we shall be" .

```
scala> val poem = "As halcyons we shall be"
poem: String = As halcyons we shall be
```

val :

```
scala> print(poem)
As halcyons we shall be
```

val .

```
scala> poem = "Brooding on the open sea"
<console>:12: error: reassignment to val
    poem = "Brooding on the open sea"
```

REPL val ().

```
scala> val poem = "Brooding on the open sea"
poem: String = Brooding on the open sea
```

REPL . REPL . , , .

Scala Quicksheet

int	val x = 3
int	var x = 3
	val x: Int = 27
	lazy val y = print("Sleeping in.")
	val f = (x: Int) => x * x
	val f: Int => Int = (x: Int) => x * x
	def f(x: Int) = x * x

	<code>def f(x: Int): Int = x * x</code>
	<code>class Hopper(someParam: Int) { ... }</code>
	<code>object Hopper(someParam: Int) { ... }</code>
	<code>trait Grace { ... }</code>
	<code>Seq(1,2,3).head</code>
	<code>val result = if(x > 0) "Positive!"</code>
	<code>Seq(1,2,3).tail</code>
	<code>for { x <- Seq(1,2,3) } print(x)</code>
	<code>for { x <- Seq(1,2,3) y <- Seq(4,5,6) } print(x + ":" + y)</code>
<code>list execute</code>	<code>List(1,2,3).foreach { println }</code>
	<code>print("Ada Lovelace")</code>
	<code>List('b','c','a').sorted</code>

: <https://riptutorial.com/ko/scala/topic/216/-->

2: Implicit

- `val x : T = ???`

```
.  
'.'  
  
1., .  
2..  
3.,, .
```

Examples

```
. .
```

```
case class Foo(i: Int)  
  
// without the implicit  
Foo(40) + 2    // compilation-error (type mismatch)  
  
// defines how to turn a Foo into an Int  
implicit def fooToInt(foo: Foo): Int = foo.i  
  
// now the Foo is converted to Int automatically when needed  
Foo(40) + 2    // 42
```

```
: 42 Foo(42) . .
```

```
implicit def intToFoo(i: Int): Foo = Foo(i)
```

float .

```
. .
```

```
.
```

```
scala.Predef . .
```

```
.
```

```
.
```

```
// import the duration methods  
import scala.concurrent.duration._  
  
// a normal method:  
def doLongRunningTask(timeout: FiniteDuration): Long = timeout.toMillis  
  
val timeout = 1.second
```

```
// timeout: scala.concurrent.duration.FiniteDuration = 1 second

// to call it
doLongRunningTask(timeout) // 1000
```

```
// import the duration methods
import scala.concurrent.duration._

// dummy methods that use the implicit parameter
def doLongRunningTaskA()(implicit timeout: FiniteDuration): Long = timeout.toMillis
def doLongRunningTaskB()(implicit timeout: FiniteDuration): Long = timeout.toMillis

// we define the value timeout as implicit
implicit val timeout: FiniteDuration = 1.second

// we can now call the functions without passing the timeout parameter
doLongRunningTaskA() // 1000
doLongRunningTaskB() // 1000
```

scalac

implicitNotFound .

```
@annotation.implicitNotFound(msg = "Select the proper implicit value for type M[${A}]!")
case class M[A](v: A) {}

def usage[A](implicit x: M[A]): A = x.v

//Does not work because no implicit value is present for type `M[Int]`
//usage[Int] //Select the proper implicit value for type M[Int]!
implicit val first: M[Int] = M(1)
usage[Int] //Works when `second` is not in scope
implicit val second: M[Int] = M(2)
//Does not work because more than one implicit values are present for the type `M[Int]`
//usage[Int] //Select the proper implicit value for type M[Int]!
```

. Akka ActorSystem() . , typeclasses (: ,) FP .

Int , Long , String . .

String withoutVowels . .

```
object StringUtil {
  implicit class StringEnhancer(str: String) {
    def withoutVowels: String = str.replaceAll("[aeiou]", "")
  }
}
```

(String) (withoutVowels) "" (str). ().

```
import StringUtil.StringEnhancer // Brings StringEnhancer into implicit scope

println("Hello world".withoutVowels) // Hll wrld
```

.

```
implicit def toStringEnhancer(str: String): StringEnhancer = new StringEnhancer(str)
```

.

```
implicit class StringEnhancer(val str: String) extends AnyVal {
  /* conversions code here */
}
```

```
withoutVowels    StringEnhancer    .
```

''

.

```
case class Example(p1:String, p2:String)(implicit ctx1:SomeCtx1, ctx2:SomeCtx2)
```

```
( SomeCtx1 )      SomeCtx1 .
```

```
implicitly        .
```

```
Example("something","somethingElse")(new SomeCtx1(), implicitly[SomeCtx2])
```

REPL

REPL implicits :

```
scala> :implicits
```

```
Predef.scala      Predef.scala .
```

```
scala> :implicits -v
```

(implicit) :

```
scala> reflect.runtime.universe.reify(expr) // No quotes. reify is a macro operating directly
on code.
```

(:

```
scala> import reflect.runtime.universe._
scala> reify(Array("Alice", "Bob", "Eve").mkString(", "))
resX: Expr[String] = Expr[String](Predef.refArrayOps(Array.apply("Alice", "Bob",
```

```
"Eve") (Predef.implicitly)).mkString(", ")
```

)

Implicits : <https://riptutorial.com/ko/scala/topic/1732/implicits>

3: Java

Examples

Java

Java :

```
import scala.collection.JavaConverters._

val scalaList = List(1, 2, 3)
JavaLibrary.process(scalaList.asJava)
```

Java Java Scala .

```
import scala.collection.JavaConverters._

val javaCollection = JavaLibrary.getList
val scalaCollection = javaCollection.asScala
```

Java . .asJava .asScala .

JVM , (invariant) . new .

```
val a = Array("element")
```

a Array[String] .

```
val acs: Array[CharSequence] = a
//Error: type mismatch; found   : Array[String]   required: Array[CharSequence]
```

String CharSequence Array[String] Array[CharSequence] .

TraversableLike ArrayOps Array .

```
val b: Array[Int] = a.map(_.length)
```

(TraversableOnce) ClassTag toArray .

```
List(0).toArray
//> res1: Array[Int] = Array(0)
```

TraversableOnce Java .

Java

JavaConverters .

	Java
	java.util.Iterator
	java.util.Enumeration
	java.util.Iterable
	java.util.Collection
.	java.util.List
mutable.Set	java.util.Set
.	java.util.Map
mutable.ConcurrentMap	java.util.concurrent.ConcurrentMap

Java .

	Java
	java.util.List
mutable.Seq	java.util.List
	java.util.Set
	java.util.Map

:

[Java Scala](#)

- **scala-java8-compat**

[Scala Java 8](#) .

[Readme](#) .

scala.FunctionN java.util.function

```
import java.util.function._
import scala.compat.java8.FunctionConverters._

val foo: Int => Boolean = i => i > 7
def testBig(ip: IntPredicate) = ip.test(9)
println(testBig(foo.asJava)) // Prints true
```

```
val bar = new UnaryOperator[String]{ def apply(s: String) = s.reverse }
List("cod", "herring").map(bar.asScala)    // List("doc", "gnirrih")

def testA[A](p: Predicate[A])(a: A) = p.test(a)
println(testA(asJavaPredicate(foo))(4))   // Prints false
```

scala.Option java.util , OptionalDouble, OptionalInt OptionalLong.

```
import scala.compat.java8.OptionConverters._

class Test {
  val o = Option(2.7)
  val oj = o.asJava           // Optional[Double]
  val ojd = o.asPrimitive    // OptionalDouble
  val ojds = ojd.asScala     // Option(2.7) again
}
```

Scala Java 8

```
import java.util.stream.IntStream

import scala.compat.java8.StreamConverters._
import scala.compat.java8.collectionImpl.{Accumulator, LongAccumulator}

val m = collection.immutable.HashMap("fish" -> 2, "bird" -> 4)
val parStream: IntStream = m.parValueStream
val s: Int = parStream.sum
// 6, potentially computed in parallel
val t: List[String] = m.seqKeyStream.toScala[List]
// List("fish", "bird")
val a: Accumulator[(String, Int)] = m.accumulate // Accumulator[(String, Int)]

val n = a.stepper.fold(0)(_ + _.length) +
  a.parStream.count // 8 + 2 = 10

val b: LongAccumulator = java.util.Arrays.stream(Array(2L, 3L, 4L)).accumulate
// LongAccumulator
val l: List[Long] = b.toList // List(2L, 3L, 4L)
```

Java : <https://riptutorial.com/ko/scala/topic/2441/java-->

4: JSON

Examples

JSON with spray-json

spray-json JSON . "".

SBT

SBT spray-json .

```
libraryDependencies += "io.spray" %% "spray-json" % "1.3.2"
```

(1.3.2) .

spray-json repo.spray.io .

```
import spray.json._
import DefaultJsonProtocol._
```

JSON DefaultJsonProtocol . JSON .

JSON

```
// generates an intermediate JSON representation (abstract syntax tree)
val res = """"{ "foo": "bar" }""".parseJson // JsValue = {"foo":"bar"}

res.convertTo[Map[String, String]] // Map(foo -> bar)
```

JSON

```
val values = List("a", "b", "c")
values.toJson.prettyPrint // ["a", "b", "c"]
```

DSL

DSL .



JSON .

```
case class Address(street: String, city: String)
case class Person(name: String, address: Address)

// create the formats and provide them implicitly
implicit val addressFormat = jsonFormat2(Address)
implicit val personFormat = jsonFormat2(Person)

// serialize a Person
Person("Fred", Address("Awesome Street 9", "SuperCity"))
val fredJsonString = fred.toJson.prettyPrint
```

JSON .

```
{
  "name": "Fred",
  "address": {
    "street": "Awesome Street 9",
    "city": "SuperCity"
  }
}
```

JSON deserialize .

```
val personRead = fredJsonString.parseJson.convertTo[Person]
//Person(Fred,Address(Awesome Street 9,SuperCity))
```



JsonFormat . JSON

```
implicit object BetterPersonFormat extends JsonFormat[Person] {
  // deserialization code
  override def read(json: JsValue): Person = {
    val fields = json.asJsObject("Person object expected").fields
    Person(
      name = fields("name").convertTo[String],
      address = fields("home").convertTo[Address]
    )
  }

  // serialization code
  override def write(person: Person): JsValue = JsObject(
    "name" -> person.name.toJson,
    "home" -> person.address.toJson
  )
}
```

Circe JSON

```
import io.circe._
import io.circe.generic.auto._
import io.circe.parser._
import io.circe.syntax._

case class User(id: Long, name: String)

val user = User(1, "John Doe")

// {"id":1,"name":"John Doe"}
val json = user.asJson.noSpaces

// Right(User(1L, "John Doe"))
val res: Either[Error, User] = decode[User](json)
```

JSON with play-json

play-json json .

SBT : libraryDependencies += "com.typesafe.play" %% "play-json" % "2.4.8"

```
import play.api.libs.json._
import play.api.libs.functional.syntax._ // if you need DSL
```

DefaultFormat . JSON .

json

```
// generates an intermediate JSON representation (abstract syntax tree)
val res = Json.parse("""{ "foo": "bar" }""") // JsValue = {"foo":"bar"}

res.as[Map[String, String]] // Map(foo -> bar)
res.validate[Map[String, String]] // JsSuccess(Map(foo -> bar),)
```

json

```
val values = List("a", "b", "c")
Json.stringify(Json.toJson(values)) // ["a", "b", "c"]
```

DSL

```
val json = parse("""{ "foo": [{"foo": "bar"}]}""")
(json \ "foo").get // Simple path: [{"foo":"bar"}]
(json \ \ "foo") // Recursive path: List([{"foo":"bar"}], "bar")
(json \ "foo")(0).get // Index lookup (for JsArrays): {"foo":"bar"}
```

JsSuccess / JsError .get , array(i) .

```
case class Address(street: String, city: String)
case class Person(name: String, address: Address)
```

```
// create the formats and provide them implicitly
implicit val addressFormat = Json.format[Address]
implicit val personFormat = Json.format[Person]

// serialize a Person
val fred = Person("Fred", Address("Awesome Street 9", "SuperCity"))
val fredJsonString = Json.stringify(Json.toJson(Json.toJson(fred)))

val personRead = Json.parse(fredJsonString).as[Person] //Person(Fred,Address(Awesome Street
9,SuperCity))
```

(: scala Json) JsonFormat .

```
case class Address(street: String, city: String)

// create the formats and provide them implicitly
implicit object AddressFormatCustom extends Format[Address] {
  def reads(json: JsValue): JsResult[Address] = for {
    street <- (json \ "Street").validate[String]
    city <- (json \ "City").validate[String]
  } yield Address(street, city)

  def writes(x: Address): JsValue = Json.obj(
    "Street" -> x.street,
    "City" -> x.city
  )
}

// serialize an address
val address = Address("Awesome Street 9", "SuperCity")
val addressJsonString = Json.stringify(Json.toJson(Json.toJson(address)))
//{"Street":"Awesome Street 9","City":"SuperCity"}

val addressRead = Json.parse(addressJsonString).as[Address]
//Address(Awesome Street 9,SuperCity)
```

json (json is_alive isAlive) :

```
case class User(username: String, friends: Int, enemies: Int, isAlive: Boolean)

object User {

  import play.api.libs.functional.syntax._
  import play.api.libs.json._

  implicit val userReads: Reads[User] = (
    (JsPath \ "username").read[String] and
    (JsPath \ "friends").read[Int] and
    (JsPath \ "enemies").read[Int] and
    (JsPath \ "is_alive").read[Boolean]
  ) (User.apply _)
}
```

Json

```
case class User(username: String, friends: Int, enemies: Int, isAlive: Option[Boolean])
```

```
object User {

  import play.api.libs.functional.syntax._
  import play.api.libs.json._

  implicit val userReads: Reads[User] = (
    (JsPath \ "username").read[String] and
    (JsPath \ "friends").read[Int] and
    (JsPath \ "enemies").read[Int] and
    (JsPath \ "is_alive").readNullable[Boolean]
  ) (User.apply _)
}
```

json

Json .

```
{
  "field": "example field",
  "date": 1459014762000
}
```

:

```
case class JsonExampleV1(field: String, date: DateTime)
object JsonExampleV1{
  implicit val r: Reads[JsonExampleV1] = (
    (__ \ "field").read[String] and
    (__ \ "date").read[DateTime] (Reads.DefaultJodaDateReads)
  ) (JsonExampleV1.apply _)
}
```

. json .

```
{
  "id": 91,
  "data": "Some data"
}
```

:

```
case class MyIdentifier(id: Long)

case class JsonExampleV2(id: MyIdentifier, data: String)
```

(Long) idenfier .

```
object JsonExampleV2 {
  implicit val r: Reads[JsonExampleV2] = (
    (__ \ "id").read[Long].map(MyIdentifier) and
    (__ \ "data").read[String]
  ) (JsonExampleV2.apply _)
}
```

JSON json4s

json4s json .

SBT :

```
libraryDependencies += "org.json4s" %% "json4s-native" % "3.4.0"
//or
libraryDependencies += "org.json4s" %% "json4s-jackson" % "3.4.0"
```

```
import org.json4s.JsonDSL._
import org.json4s._
import org.json4s.native.JsonMethods._

implicit val formats = DefaultFormats
```

DefaultFormats .

json

```
// generates an intermediate JSON representation (abstract syntax tree)
val res = parse("""{ "foo": "bar" }""")           // JValue = {"foo":"bar"}
res.extract[Map[String, String]]                  // Map(foo -> bar)
```

json

```
val values = List("a", "b", "c")
compact(render(values))           // ["a", "b", "c"]
```

DSL

```
json \ "foo"           //Simple path: JArray(List(JObject(List((foo,JString(bar))))))
json \\ "foo"          //Recursive path: ~List([{"foo":"bar"}], "bar")
(json \ "foo")(0)      //Index lookup (for JsArrays): JObject(List((foo,JString(bar))))
("foo" -> "bar") ~ ("field" -> "value") // {"foo":"bar","field":"value"}
```

```
import org.json4s.native.Serialization.{read, write}

case class Address(street: String, city: String)
val addressString = write(Address("Awesome stree", "Super city"))
// {"street":"Awesome stree","city":"Super city"}

read[Address](addressString) // Address(Awesome stree,Super city)
//or
parse(addressString).extract[Address]
```

() serialize deserialize .

```
trait Location
```

```

case class Street(name: String) extends Location
case class City(name: String, zipcode: String) extends Location
case class Address(street: Street, city: City) extends Location
case class Locations (locations : List[Location])

implicit val formats = Serialization.formats(ShortTypeHints(List(classOf[Street],
classOf[City], classOf[Address])))

val locationsString = write(Locations(Street("Lavelle Street"):: City("Super city","74658")))

read[Locations](locationsString)

```

```

class AddressSerializer extends CustomSerializer[Address](format => (
{
  case JObject(JField("Street", JString(s)) :: JField("City", JString(c)) :: Nil) =>
    new Address(s, c)
},
{
  case x: Address => ("Street" -> x.street) ~ ("City" -> x.city)
}
))

implicit val formats = DefaultFormats + new AddressSerializer
val str = write[Address](Address("Awesome Stree", "Super City"))
// {"Street":"Awesome Stree","City":"Super City"}
read[Address](str)
// Address(Awesome Stree,Super City)

```

JSON : <https://riptutorial.com/ko/scala/topic/2348/json>

5: Quasiquotes

Examples

quasiquotes

Tree .

```
object macro {
  def addCreationDate(): java.util.Date = macro impl.addCreationDate
}

object impl {
  def addCreationDate(c: Context)(): c.Expr[java.util.Date] = {
    import c.universe._

    val date = q"new java.util.Date()" // this is the quasiquote
    c.Expr[java.util.Date](date)
  }
}
```

.

Quasiquotes : <https://riptutorial.com/ko/scala/topic/4032/quasiquotes>

6: Scala.js

Scala.js JavaScript Scala , JVM JVM. , , JavaScript .

Examples

Scala.js console.log

```
println("Hello Scala.js") // In ES6: console.log("Hello Scala.js");
```

```
val lastNames = people.map(p => p.lastName)
// Or shorter:
val lastNames = people.map(_.lastName)
```

```
class Person(val firstName: String, val lastName: String) {
  def fullName(): String =
    s"$firstName $lastName"
}
```

```
val personMap = Map(
  10 -> new Person("Roger", "Moore"),
  20 -> new Person("James", "Bond")
)
val names = for {
  (key, person) <- personMap
  if key > 15
} yield s"$key = ${person.firstName}"
```

DOM

```
import org.scalajs.dom
import dom.document

def appendP(target: dom.Node, text: String) = {
  val pNode = document.createElement("p")
  val textNode = document.createTextNode(text)
  pNode.appendChild(textNode)
  target.appendChild(pNode)
}
```

SBT

Sbt

```
libraryDependencies += "org.scala-js" %% "scalajs-dom" % "0.9.1" // (Triple %%)
```

```
sbt run
```

:

```
sbt ~run
```

JavaScript :

```
sbt fastOptJS
```

Scala.js : <https://riptutorial.com/ko/scala/topic/9426/scala-js>

7: ScalaCheck

ScalaCheck Scala Scala Java . ScalaCheck Haskell QuickCheck .

ScalaCheck Scala Scala sbt . ScalaTest specs2 .

Examples

Scalacheck

scalatest scalacheck . .

- " "-
- " "- && boolean .
- " "- ("argument" | : :) && Props.all .
- " "- ("command" | : :) Props.all . &&

```
import org.scalatest.prop.Checkers
import org.scalatest.{Matchers, WordSpecLike}

import org.scalacheck.Gen._
import org.scalacheck.Prop._
import org.scalacheck.Prop

object Splitter {
  def splitLineByColon(message: String): (String, String) = {
    val (command, argument) = message.indexOf(":") match {
      case -1 =>
        (message, "")
      case x: Int =>
        (message.substring(0, x), message.substring(x + 1))
    }
    (command.trim, argument.trim)
  }

  def splitLineByColonWithBugOnCommand(message: String): (String, String) = {
    val (command, argument) = splitLineByColon(message)
    (command.trim + 2, argument.trim)
  }

  def splitLineByColonWithBugOnArgument(message: String): (String, String) = {
    val (command, argument) = splitLineByColon(message)
    (command.trim, argument.trim + 2)
  }
}

class ScalaCheckSpec extends WordSpecLike with Matchers with Checkers {

  private val COMMAND_LENGTH = 4

  "ScalaCheckSpec " should {
```

```
    "show pass example" in {
```

```

check {
  Prop.forAll(listOfN(COMMAND_LENGTH, alphaChar), alphaStr) {
    (chars, expArgument) =>
      val expCommand = new String(chars.toArray)
      val line = s"$expCommand:$expArgument"
      val (c, p) = Splitter.splitLineByColon(line)
      Prop.all("command" |: c =? expCommand, "argument" |: expArgument =? p)
  }
}

```

```

"show simple example without custom error message " in {
  check {
    Prop.forAll(listOfN(COMMAND_LENGTH, alphaChar), alphaStr) {
      (chars, expArgument) =>
        val expCommand = new String(chars.toArray)
        val line = s"$expCommand:$expArgument"
        val (c, p) = Splitter.splitLineByColonWithBugOnArgument(line)
        c === expCommand && expArgument === p
    }
  }
}

```

```

"show example with error messages on argument" in {
  check {
    Prop.forAll(listOfN(COMMAND_LENGTH, alphaChar), alphaStr) {
      (chars, expArgument) =>
        val expCommand = new String(chars.toArray)
        val line = s"$expCommand:$expArgument"
        val (c, p) = Splitter.splitLineByColonWithBugOnArgument(line)
        Prop.all("command" |: c =? expCommand, "argument" |: expArgument =? p)
    }
  }
}

```

```

"show example with error messages on command" in {
  check {
    Prop.forAll(listOfN(COMMAND_LENGTH, alphaChar), alphaStr) {
      (chars, expArgument) =>
        val expCommand = new String(chars.toArray)
        val line = s"$expCommand:$expArgument"
        val (c, p) = Splitter.splitLineByColonWithBugOnCommand(line)
        Prop.all("command" |: c =? expCommand, "argument" |: expArgument =? p)
    }
  }
}

```

0:

```

[info] - should show example // passed
[info] - should show simple example without custom error message *** FAILED ***
[info]   (ScalaCheckSpec.scala:73)
[info]   Falsified after 0 successful property evaluations.
[info]   Location: (ScalaCheckSpec.scala:73)

```

```

[info]     Occurred when passed generated values (
[info]       arg0 = List(), // 3 shrinks
[info]       arg1 = ""
[info]     )
[info] - should show example with error messages on argument *** FAILED ***
[info]   (ScalaCheckSpec.scala:86)
[info]   Falsified after 0 successful property evaluations.
[info]   Location: (ScalaCheckSpec.scala:86)
[info]   Occurred when passed generated values (
[info]     arg0 = List(), // 3 shrinks
[info]     arg1 = ""
[info]   )
[info]   Labels of failing property:
[info]     Expected "" but got "2"
[info]     argument
[info] - should show example with error messages on command *** FAILED ***
[info]   (ScalaCheckSpec.scala:99)
[info]   Falsified after 0 successful property evaluations.
[info]   Location: (ScalaCheckSpec.scala:99)
[info]   Occurred when passed generated values (
[info]     arg0 = List(), // 3 shrinks
[info]     arg1 = ""
[info]   )
[info]   Labels of failing property:
[info]     Expected "2" but got ""
[info]     command

```

ScalaCheck : <https://riptutorial.com/ko/scala/topic/9430/scalacheck->

8: Scaladoc

- , , .
- /** .
- * .
- */

	—
@constructor detail	.
	—
@return detail	.
, /	—
@param x detail	value x .
@tparam x detail	type x .
@throws detail	.
	—
@see detail	.
@note detail	.
@example detail	.
@usecase detail	.
	—
@author detail	.
@version detail	.
@deprecated detail	.

Examples

Scaladoc

```
/**
```

```
* Explain briefly what method does here
* @param x Explain briefly what should be x and how this affects the method.
* @param y Explain briefly what should be y and how this affects the method.
* @return Explain what is returned from execution.
*/
def method(x: Int, y: String): Option[Double] = {
  // Method content
}
```

Scaladoc : <https://riptutorial.com/ko/scala/topic/4518/scaladoc>

9: ScalaTest

Examples

Hello World Spec

HelloWorldSpec.scala src/test/scala . . .

```
import org.scalatest.{FlatSpec, Matchers}

class HelloWorldSpec extends FlatSpec with Matchers {

  "Hello World" should "not be an empty String" in {
    val helloWorld = "Hello World"
    helloWorld should not be ("")
  }
}
```

- [ScalaTest FlatSpec Matchers](#) .
- FlatSpec [BDD \(Behavior-Driven Development\)](#) FlatSpec

Cheatsheet

.

```
val helloWorld = "Hello World"
val helloWorldCount = 1
val helloWorldList = List("Hello World", "Bonjour Le Monde")
def sayHello = throw new IllegalStateException("Hello World Exception")
```

val .

```
helloWorld shouldBe a [String]
```

String .

.

```
helloWorld shouldEqual "Hello World"
helloWorld should === ("Hello World")
helloWorldCount shouldEqual 1
helloWorldCount shouldBe 1
helloWorldList shouldEqual List("Hello World", "Bonjour Le Monde")
helloWorldList === List("Hello World", "Bonjour Le Monde")
```

:

```
helloWorld should not equal "Hello"
helloWorld !== "Hello"
```

```
helloWorldCount should not be 5
helloWorldList should not equal List("Hello World")
helloWorldList != List("Hello World")
helloWorldList should not be empty
```

/ .

```
helloWorld should have length 11
helloWorldList should have size 2
```

.

```
val exception = the [java.lang.IllegalStateException] thrownBy {
  sayHello
}
exception.getClass shouldEqual classOf[java.lang.IllegalStateException]
exception.getMessage should include ("Hello World")
```

SBT ScalaTest

SBT build.sbt .

```
libraryDependencies += "org.scalactic" %% "scalactic" % "3.0.0"
libraryDependencies += "org.scalatest" %% "scalatest" % "3.0.0" % "test"
```

ScalaTest .

ScalaTest : <https://riptutorial.com/ko/scala/topic/5506/scalatest->

10: While

- while (boolean_expression) {block_expression}
- do {block_expression} while (boolean_expression) while (boolean_expression)

boolean_expression	true false .
block_expression	boolean_expression true .

```
while do-while    block_expression    .
```

```
while do-while    false                .
```

Examples

While

```
var line = 0
var maximum_lines = 5

while (line < maximum_lines) {
  line = line + 1
  println("Line number: " + line)
}
```

Do-While

```
var line = 0
var maximum_lines = 5

do {
  line = line + 1
  println("Line number: " + line)
} while (line < maximum_lines)
```

```
do / while        break / continue    .
```

```
if(initial_condition) do if(filter) {
  ...
} while(continuation_condition)
```

While : <https://riptutorial.com/ko/scala/topic/650/while->

11: XML

Examples

Beautify Pretty-Print XML

PrettyPrinter XML ' '. xml .

```
import scala.xml.{PrettyPrinter, XML}
val xml = XML.loadString("<a>Alana<b><c>Beth</c><d>Catie</d></b></a>")
val formatted = new PrettyPrinter(150, 4).format(xml)
print(formatted)
```

150 4 .

```
<a>
  Alana
  <b>
    <c>Beth</c>
    <d>Catie</d>
  </b>
</a>
```

XML.loadFile("nameoffile.xml") xml .

XML : <https://riptutorial.com/ko/scala/topic/1453/xml->

12:

```
"" . . . shift / reset .
```

: <https://github.com/scala/scala-continuations>

- reset {...} //
- shift {...} //
- A @cpsParam [B, C] // C A => B
- @cps [A] // @cpsParam [A, A]
- @suspendable // @cpsParam [Unit, Unit]

```
shift reset Int.+ Long . . " " API .
```

```
. , flatMap .
```

Examples

Continuations.

```
// Takes a callback and executes it with the read value
def readFile(path: String)(callback: Try[String] => Unit): Unit = ???

readFile(path) { _.flatMap { file1 =>
  readFile(path2) { _.foreach { file2 =>
    processFiles(file1, file2)
  }}
}}
```

```
readFile readFile readFile .
```

continuations .

```
reset { // Reset is a delimiter for continuations.
  for { // Since the callback hell is relegated to continuation library machinery.
    // a for-comprehension can be used
    file1 <- shift(readFile(path1)) // shift has type ((A => B) => C) => A
    // We use it as (((Try[String] => Unit) => Unit) => Try[String])
    // It takes all the code that occurs after it is called, up to the end of reset, and
    // makes it into a closure of type (A => B).
    // The reason this works is that shift is actually faking its return type.
    // It only pretends to return A.
    // It actually passes that closure into its function parameter (readFile(path1) here),
    // And that function calls what it thinks is a normal callback with an A.
    // And through compiler magic shift "injects" that A into its own callsite.
    // So if readFile calls its callback with parameter Success("OK"),
    // the shift is replaced with that value and the code is executed until the end of reset,
    // and the return value of that is what the callback in readFile returns.
    // If readFile called its callback twice, then the shift would run this code twice too.
    // Since readFile returns Unit though, the type of the entire reset expression is Unit
  }
}
```

```

    // Think of shift as shifting all the code after it into a closure,
    // and reset as resetting all those shifts and ending the closures.
    file2 <- shift(readFile(path2))
  } processFiles(file1, file2)
}

// After compilation, shift and reset are transformed back into closures
// The for comprehension first desugars to:
reset {
  shift(readFile(path1)).flatMap { file1 => shift(readFile(path2)).foreach { file2 =>
processFiles(file1, file2) } }
}
// And then the callbacks are restored via CPS transformation
readFile(path1) { _.flatMap { file1 => // We see how shift moves the code after it into a
closure
  readFile(path2) { _.foreach { file2 =>
    processFiles(file1, file2)
  }}
}} // And we see how reset closes all those closures
// And it looks just like the old version!

```

```

shift  reset,      reset.shift  (((A => B) => C) => A)  (((A => B) => C) => (A @cpsParam[B,
C])) . CPS  .reset  shift  ""'.

```

```

reset  A @cpsParam[B, C]  A  .  A => B  B . C "" CPS  C .

```

, [Scaladoc](#) .

```

val sessions = new HashMap[UUID, Int=>Unit]
def ask(prompt: String): Int @suspendable = // alias for @cpsParam[Unit, Unit]. @cps[Unit] is
also an alias. (@cps[A] = @cpsParam[A,A])
  shift {
    k: (Int => Unit) => {
      println(prompt)
      val id = uuidGen
      sessions += id -> k
    }
  }

def go(): Unit = reset {
  println("Welcome!")
  val first = ask("Please give me a number") // Uses CPS just like shift
  val second = ask("Please enter another number")
  printf("The sum of your numbers is: %d\n", first + second)
}

```

```

ask  ""  . go  .

```

: <https://riptutorial.com/ko/scala/topic/8312/>-

13:

```

, , .

Function1 ( ). apply ., .

, .

. Seq map "higher-order function" . .
```

Examples

Scala

```
object MyObject {
  def mapMethod(input: Int): String = {
    int.toString
  }
}

Seq(1, 2, 3).map(MyObject.mapMethod) // Seq("1", "2", "3")
```

MyObject.mapMethod map ., map . List[A] (A List[A] map .

```
def map[B] (f: (A) => B): List[B]
```

f: (A) => B A A B . A B . mapMethod Int (A) String (B) . mapMethod map . .

```
Seq(1, 2, 3).map(x:Int => int.toString)
```

.

()

1 .

- .
- .
- : .

```
object HOF {
  def main(args: Array[String]) {
    val list =
    List(("Srini","E"), ("Subash","R"), ("Ranjith","RK"), ("Vicky","s"), ("Sudhar","s"))
    //HOF
```

```

    val fullNameList= list.map(n => getFullName(n._1, n._2))

    }

    def getFullName(firstName: String, lastName: String): String = firstName + "." +
    lastName
    }

```

`map` `getFullName(n._1,n._2)` . **HOF (Higher order function).**

Scala notation : `def func(arg: => String)` **lazy evaluation** . `String String` . , .

.

```

def calculateData: String = {
    print("Calculating expensive data! ")
    "some expensive data"
}

def dumbMediator(preconditions: Boolean, data: String): Option[String] = {
    print("Applying mediator")
    preconditions match {
        case true => Some(data)
        case false => None
    }
}

def smartMediator(preconditions: Boolean, data: => String): Option[String] = {
    print("Applying mediator")
    preconditions match {
        case true => Some(data)
        case false => None
    }
}

smartMediator(preconditions = false, calculateData)

dumbMediator(preconditions = false, calculateData)

```

`smartMediator` **None** "Applying mediator" .

`dumbMediator` **None** "Calculating expensive data! Applying mediator" "Calculating expensive data!
Applying mediator" . "Calculating expensive data! Applying mediator" .

.

: <https://riptutorial.com/ko/scala/topic/1642/>-

14:

.

■
■

: , . def (def), .

- (: Function1) . , , , .
- .
- .
- .

Examples

.

.

```
(x: Int, y: Int) => x + y
```

val .

```
val sum = (x: Int, y: Int) => x + y
```

. , map .

```
// Returns Seq("FOO", "BAR", "QUX")
Seq("Foo", "Bar", "Qux").map((x: String) => x.toUpperCase)
```

. :

```
Seq("Foo", "Bar", "Qux").map((x) => x.toUpperCase)
```

.

```
Seq("Foo", "Bar", "Qux").map(x => x.toUpperCase)
```

. .

```
Seq("Foo", "Bar", "Qux").map(_.toUpperCase)
```

— . — . , .

```
// Returns "FooBarQux" in both cases
Seq("Foo", "Bar", "Qux").reduce((s1, s2) => s1 + s2)
Seq("Foo", "Bar", "Qux").reduce(_ + _)
```

, — .

.

```
val sayHello = () => println("hello")
```

. , f(x) g(x) , h(x) = f(g(x)) .

Function1 . **Scala composition** : andThen compose Function1 . compose .

```
val f: B => C = ...
val g: A => B = ...

val h: A => C = f compose g
```

andThen (h(x) = g(f(x))) 'DSL' andThen .

```
val f: A => B = ...
val g: B => C = ...

val h: A => C = f andThen g
```

f g . h .

```
def andThen(g: B => C): A => C = new (A => C) {
  def apply(x: A) = g(self(x))
}
```

f g h f g . .

PartialFunctions

```
trait PartialFunction[-A, +B] extends (A => B)
```

PartialFunction Function1 . . Function1 true isDefinedAt .

() .

```
{ case i: Int => i + 1 } // or equivalently { case i: Int => i + 1 }
```

PartialFunctions .

: <https://riptutorial.com/ko/scala/topic/477/>

15:

, ., () .

Lisp, Ruby, Erlang . Scala . . .

:

' , - _

:

```
'ATM
'IPv4
'IPv6
'map_to_operations
'data_format_2006

// Using the `Symbol.apply` method

Symbol("hakuna matata")
Symbol("To be or not to be that is a question")
```

:

```
'8'th_division
'94_pattern
'bad-format
```

Examples

case

, , *argumentList*

```
def loadData(dataSource: Symbol): Try[String] = dataSource match {
  case 'database => loadDatabase() // Loading data from database
  case 'file => loadFile() // Loading data from file
  case 'prompt => askUser() // Asking user for data
  case 'argumentList => argumentListExtract() // Accessing argument list for data
  case _ => Failure(new Exception("Unsupported data source"))
}
```

Symbol String

.

: <https://riptutorial.com/ko/scala/topic/6419/>

16: (SAM)

Java 8 .

Examples

: 2.12+ (2.11.X -Xexperimental -Xfuture) .

SAM .

2.11.8

```
trait Runnable {  
  def run(): Unit  
}  
  
val t: Runnable = () => println("foo")
```

.

2.11.8

```
trait Runnable {  
  def run(): Unit  
  def concrete: Int = 42  
}  
  
val t: Runnable = () => println("foo")
```

(SAM) : <https://riptutorial.com/ko/scala/topic/3664/-----sam-->

17:

- `objectToSynchronizeOn.synchronized { /* code to run */ }`
- `synchronized { /* code to run, can be suspended with wait */ }`

Examples

`synchronized` . [Java JVM](#) .

```
anInstance.synchronized {  
  // code to run when the intrinsic lock on `anInstance` is acquired  
  // other thread cannot enter concurrently unless `wait` is called on `anInstance` to suspend  
  // other threads can continue of the execution of this thread if they `notify` or  
  `notifyAll` `anInstance`'s lock  
}
```

`object` .

▪

```
/* within a class, def, trait or object, but not a constructor */  
synchronized {  
  /* code to run when an intrinsic lock on `this` is acquired */  
  /* no other thread can get the this lock unless execution is suspended with  
   * `wait` on `this`  
   */  
}
```

: [https://riptutorial.com/ko/scala/topic/3371/-](https://riptutorial.com/ko/scala/topic/3371/)

18:

Scala `scala.Dynamic` `[ scala.Dynamic ] [Dynamic scaladoc]` `. [ scaladoc ] :`
<http://www.scala-lang.org/api/2.12.x/scala/Dynamic.html>

- `Foo Dynamic`
- `foo.field`
- `foo.field = value`
- `foo.method (args)`
- `foo.method ( Arg = x, y)`

`Dynamic scala.language.dynamics -language:dynamics dynamics . Dynamic .`

Examples

:

```
class Foo extends Dynamic {  
  // Expressions are only rewritten to use Dynamic if they are not already valid  
  // Therefore foo.realField will not use select/updateDynamic  
  var realField: Int = 5  
  // Called for expressions of the type foo.field  
  def selectDynamic(fieldName: String) = ???  
  def updateDynamic(fieldName: String) (value: Int) = ???  
}
```

.

```
val foo: Foo = ???  
foo.realField // Does NOT use Dynamic; accesses the actual field  
foo.realField = 10 // Actual field access here too  
foo.unrealField // Becomes foo.selectDynamic(unrealField)  
foo.field = 10 // Becomes foo.updateDynamic("field") (10)  
foo.field = "10" // Does not compile; "10" is not an Int.  
foo.x() // Does not compile; Foo does not define applyDynamic, which is used for methods.  
foo.x.apply() // DOES compile, as Nothing is a subtype of () => Any  
// Remember, the compiler is still doing static type checks, it just has one more way to  
// "recover" and rewrite otherwise invalid code now.
```

:

```
class Villain(val minions: Map[String, Minion]) extends Dynamic {  
  def applyDynamic(name: String) (jobs: Task*) = jobs.foreach(minions(name).do)  
  def applyDynamicNamed(name: String) (jobs: (String, Task)*) = jobs.foreach {  
    // If a parameter does not have a name, and is simply given, the name passed as ""  
    case ("", task) => minions(name).do(task)  
    case (subsys, task) => minions(name).subsystems(subsys).do(task)  
  }  
}
```

```

val gru: Villain = ???
gru.blu() // Becomes gru.applyDynamic("blu")()
// Becomes gru.applyDynamicNamed("stu")(("foer", ???), ("boomer", ???), ("", ???),
//      ("computer breaker", ???), ("foer", ???))
// Note how the `???` without a name is given the name ""
// Note how both occurrences of `foer` are passed to the method
gru.stu(foer = ???, boomer = ???, ???, `computer breaker` = ???, foer = ???)
gru.ERR("a") // Somehow, scalac thinks "a" is not a Task, though it clearly is (it isn't)

```

() .

```

val dyn: Dynamic = ???
dyn.x(y) = z

```

```

dyn.selectDynamic("x").update(y, z)

```

```

dyn.x(y)

```

```

dyn.applyDynamic("x")(y)

```

: <https://riptutorial.com/ko/scala/topic/8296/>-

19:

. [] [] . [] : <http://docs.scala-lang.org/overviews/macros/paradise.html>

- `def x () = x_impl // x , x_impl .`
- `def macroTransform (annottees : Any *) : Any = impl //`

`scala.language.macros -language:macros . . . . .`

Examples

```
import scala.annotation.{compileTimeOnly, StaticAnnotation}
import scala.reflect.macros.whitebox.Context

@compileTimeOnly("enable macro paradise to expand macro annotations")
class noop extends StaticAnnotation {
  def macroTransform(annottees: Any*): Any = macro linkMacro.impl
}

object linkMacro {
  def impl(c: Context)(annottees: c.Expr[Any]*): c.Expr[Any] = {
    import c.universe._

    c.Expr[Any] (q"{..$annottees}")
  }
}
```

`@compileTimeOnly` `paradise` . `SBT` .

```
@noop
case class Foo(a: String, b: Int)

@noop
object Bar {
  def f(): String = "hello"
}

@noop
def g(): Int = 10
```

AST . AST AST .

```
import reflect.macros.blackbox.Context

object Macros {
  // This macro simply sees if the argument is the result of an addition expression.
  // E.g. isAddition(1+1) and isAddition("a"+1).
  // but !isAddition(1+1-1), as the addition is underneath a subtraction, and also
```

```
// !isAddition(x.+), and !isAddition(x.+(a,b)) as there must be exactly one argument.
def isAddition(x: Any): Boolean = macro isAddition_impl

// The signature of the macro implementation is the same as the macro definition,
// but with a new Context parameter, and everything else is wrapped in an Expr.
def isAddition_impl(c: Context)(expr: c.Expr[Any]): c.Expr[Boolean] = {
  import c.universe._ // The universe contains all the useful methods and types
  val plusName = TermName("+").encodedName // Take the name + and encode it as $plus
  expr.tree match { // Turn expr into an AST representing the code in isAddition(...)
    case Apply(Select(_, `plusName`), List(_)) => reify(true)
    // Pattern match the AST to see whether we have an addition
    // Above we match this AST
    //           Apply (function application)
    //           /      \
    //       Select  List(_) (exactly one argument)
    // (selection ^ of entity, basically the . in x.y)
    //       /      \
    //   -      \
    //       `plusName` (method named +)
    case _ => reify(false)
    // reify is a macro you use when writing macros
    // It takes the code given as its argument and creates an Expr out of it
  }
}
```

Tree . reify Expr q (quasiquote) Tree . q (expr.tree Tree) .

```
// No Exprs, just Trees
def isAddition_impl(c: Context)(tree: c.Tree): c.Tree = {
  import c.universe._
  tree match {
    // q is a macro too, so it must be used with string literals.
    // It can destructure and create Trees.
    // Note how there was no need to encode + this time, as q is smart enough to do it itself.
    case q"${_} + ${_}" => q"true"
    case _ => q"false"
  }
}
```

Context .

().

```
import reflect.macros.blackbox.Context

def debtMark(message: String): Unit = macro debtMark_impl
def debtMarkImpl(c: Context)(message: c.Tree): c.Tree = {
  message match {
    case Literal(Constant(msg: String)) => c.info(c.enclosingPosition, msg, false)
    // false above means "do not force this message to be shown unless -verbose"
    case _ => c.abort(c.enclosingPosition, "Message must be a
string literal.")
    // Abort causes the compilation to completely fail. It's not even a compile error, where
    // multiple can stack up; this just kills everything.
  }
  q"()" // At runtime this method does nothing, so we return ()
}
```

, ??? !!! ??? . ??? , !!! .

```
import reflect.macros.blackbox.Context

def ??? : Nothing = macro impl_???
def !!! : Nothing = macro impl_!!!

def impl_??? (c: Context): c.Tree = {
  import c.universe._
  c.warning(c.enclosingPosition, "Unimplemented!")
  q"${termNames.ROOTPKG}.scala.Predef.???"
  // If someone were to shadow the scala package, scala.Predef.??? would not work, as it
  // would end up referring to the scala that shadows and not the actual scala.
  // ROOTPKG is the very root of the tree, and acts like it is imported anew in every
  // expression. It is actually named _root_, but if someone were to shadow it, every
  // reference to it would be an error. It allows us to safely access ??? and know that
  // it is the one we want.
}

def impl_!!! (c: Context): c.Tree = {
  import c.universe._
  c.error(c.enclosingPosition, "Unimplemented!")
  q"${termNames.ROOTPKG}.scala.Predef.???"
}
```

: <https://riptutorial.com/ko/scala/topic/3808/>

20:

Examples

`m.flatMap(f).flatMap(g) == m.flatMap(x => f(x).flatMap(g))`

`List[T], Set[T], Option[T]`

Monad `M` `flatMap` `unit` `flatMap` `M[T]`

```
trait M[T] {  
  def flatMap[U](f: T => M[U]): M[U]  
}  
  
def unit[T](x: T): M[T]
```

1. `(m.flatMap(f).flatMap(g) == m.flatMap(x => f(x).flatMap(g)))`
`, m.flatMap(x => f(x).flatMap(g))`
2. `unit(x).flatMap(f) == f(x)`
`, f(x).flatMap(unit)`
3. `m.flatMap(unit) == m`
`" : " : "`

:

```
val m = List(1, 2, 3)  
def unit(x: Int): List[Int] = List(x)  
def f(x: Int): List[Int] = List(x * x)  
def g(x: Int): List[Int] = List(x * x * x)  
val x = 1
```

1. :

```
(m.flatMap(f).flatMap(g) == m.flatMap(x => f(x).flatMap(g))) //Boolean = true  
//Left side:  
List(1, 4, 9).flatMap(g) // List(1, 64, 729)  
//Right side:  
m.flatMap(x => (x * x) * (x * x) * (x * x)) //List(1, 64, 729)
```

2.

```
unit(x).flatMap(x => f(x)) == f(x)  
List(1).flatMap(x => x * x) == 1 * 1
```

3.

```
//m flatMap unit == m
m.flatMap(unit) == m
List(1, 2, 3).flatMap(x => List(x)) == List(1,2,3) //Boolean = true
```

•

(List[T] , Option[T]) (Either[T] , Future[T]). for (flatMap)

```
val a = List(1, 2, 3)
val b = List(3, 4, 5)
for {
  i <- a
  j <- b
} yield(i * j)
```

•

```
a flatMap {
  i => b map {
    j => i * j
  }
}
```

•

: <https://riptutorial.com/ko/scala/topic/4112/>

21:

vals, . . vars, .

- Odersky, Spoon, Venners *Scala*

Odersky .

Examples

▪

. .

-
-
- API (map , flatMap map , fold)

.

- Cake pattern Reader Monad .
- implicit .

. . . .

▪

- .
- for map .

.

```
if (userAuthorized.nonEmpty) {  
  makeRequest().map {  
    case Success(response) =>  
      someProcessing(..)  
      if (resendToUser) {  
        sendToUser(...)  
      }  
    ...  
  }  
}
```

Either Validation .

```
for {  
  user      <- authorizeUser  
  response  <- requestToThirdParty(user)  
  _        <- someProcessing(...)  
} {
```

```
sendToUser  
}
```

:

- val, var .
- [recursion comprehensions](#) .
- . val .
- , CRQ CQRS .

.

- var (:) .
- mutable .

: [https://riptutorial.com/ko/scala/topic/4376/-](https://riptutorial.com/ko/scala/topic/4376/)

22:

Scala 2.10.0 .

Examples

Hello

s .

```
val name = "Brian"
println(s"Hello $name")
```

"Hello Brian" .

f

```
val num = 42d
```

f num

```
println(f"$num%2.2f")
42.00
```

e num

```
println(f"$num%e")
4.200000e+01
```

num **A** 16

```
println(f"$num%a")
0x1.5p5
```

<https://docs.oracle.com/javase/6/docs/api/java/util/Formatter.html#detail> .

.

```
def f(x: String) = x + x
val a = "A"

s"${a}" // "A"
s"${f(a)}" // "AA"
```

\$ (f) . f String f .

```
s"$f(a)" // compile-time error (missing argument list for method f)
```

.

```
my"foo${bar}baz"
```

.

```
new scala.StringContext("foo", "baz").my(bar)
```

```
scala.StringContext my      .      s      :
```

```
implicit class MyInterpolator(sc: StringContext) {  
  def my(subs: Any*): String = {  
    val pit = sc.parts.iterator  
    val sit = subs.iterator  
    // Note parts.length == subs.length + 1  
    val sb = new java.lang.StringBuilder(pit.next())  
    while(sit.hasNext) {  
      sb.append(sit.next().toString)  
      sb.append(pit.next())  
    }  
    sb.toString  
  }  
}
```

```
my"foo${bar}baz" my"foo${bar}baz" .
```

```
new MyInterpolation(new StringContext("foo", "baz")).my(bar)
```

. .

```
case class Let(name: Char, value: Int)  
  
implicit class LetInterpolator(sc: StringContext) {  
  def let(value: Int): Let = Let(sc.parts(0).charAt(0), value)  
}  
  
let"a=${4}" // Let(a, 4)  
let"b=${"foo"}" // error: type mismatch  
let"c=" // error: not enough arguments for method let: (value: Int)Let
```

Scala [quasiquotes API](#) () .

```
n"p0${i0}p1" new StringContext("p0", "p1").n(i0) new StringContext("p0", "p1").n(i0)  
StringContext .unapply unapplySeq n .
```

```
, StringContext . heavy scala.util.matching.Regex unapplySeq .
```

```
implicit class PathExtractor(sc: StringContext) {  
  object path {  
    def unapplySeq(str: String): Option[Seq[String]] =
```

```

    sc.parts.map(Regex.quote).mkString("^", "([^\n/]+)", "$").r.unapplySeq(str)
  }
}

"/documentation/scala/1629/string-interpolation" match {
  case path"/documentation/${topic}/${id}/${_}" => println(s"$topic, $id")
  case _ => ???
}

```

path apply .

.

```
println(raw"Hello World In English And French\nEnglish:\tHello World\nFrench:\t\tBonjour Le
Monde")
```

.

```
Hello World In English And French\nEnglish:\tHello World\nFrench:\t\tBonjour Le Monde
```

\n \t .

```
println("Hello World In English And French\nEnglish:\tHello World\nFrench:\t\tBonjour Le
Monde")
```

:

```
Hello World In English And French
English:      Hello World
French:       Bonjour Le Monde
```

: <https://riptutorial.com/ko/scala/topic/1629/>-

23:

Examples

```
import scala.reflect.runtime.universe._  
val mirror = runtimeMirror(getClass.getClassLoader)  
val module = mirror.staticModule("org.data.TempClass")
```

: <https://riptutorial.com/ko/scala/topic/5824/>

24:

(def , val , var class) .

-
-
- [this]
- [fromWhere]
-
- protected [fromWhere]

Examples

()

public .

```
package com.example {
  class FooClass {
    val x = "foo"
  }
}

package an.other.package {
  class BarClass {
    val foo = new com.example.FooClass
    foo.x // <- Accessing a public value from another package
  }
}
```

private .

```
package com.example {
  class FooClass {
    private val x = "foo"
    def aFoo(otherFoo: FooClass) {
      otherFoo.x // <- Accessing from another instance of the same class
    }
  }
  class BarClass {
    val f = new FooClass
    f.x // <- This will not compile
  }
}
```

.

```
package com.example {
  class FooClass {
    private val x = "foo"
    private[example] val y = "bar"
  }
}
```

```

}
class BarClass {
  val f = new FooClass
  f.x // <- Will not compile
  f.y // <- Will compile
}
}

```

" - ", .

```

class FooClass {
  private[this] val x = "foo"
  def aFoo(otherFoo: FooClass) = {
    otherFoo.x // <- This will not compile, accessing x outside the object instance
  }
}

```

.

```

class FooClass {
  protected val x = "foo"
}
class BarClass extends FooClass {
  val y = x // It is a subclass instance, will compile
}
class ClassB {
  val f = new FooClass
  f.x // <- This will not compile
}

```

package protected scope .

```

package com.example {
  class FooClass {
    protected[example] val x = "foo"
  }
  class ClassB extends FooClass {
    val y = x // It's in the protected scope, will compile
  }
}
package com {
  class BarClass extends com.example.FooClass {
    val y = x // <- Outside the protected scope, will not compile
  }
}

```

: <https://riptutorial.com/ko/scala/topic/9705/>

25:

▪

`final` ,

, .

: <http://docs.scala-lang.org/style/naming-conventions.html>

:

```
val (a,b) = (1,2)
// a: Int = 1
// b: Int = 2
```

▪

```
val (A,B) = (1,2)
// error: not found: value A
// error: not found: value B
```

Examples

val var

val var

```
scala> val a = 123
a: Int = 123

scala> a = 456
<console>:8: error: reassignment to val
    a = 456

scala> var b = 123
b: Int = 123

scala> b = 321
b: Int = 321
```

- `val` .Java `final` .
- `var` **Java**

```
val mut = scala.collection.mutable.Map.empty[String, Int]
mut += ("123" -> 123)
mut += ("456" -> 456)
mut += ("789" -> 789)
```

```

val imm = scala.collection.immutable.Map.empty[String, Int]
imm + ("123" -> 123)
imm + ("456" -> 456)
imm + ("789" -> 789)

scala> mut
  Map(123 -> 123, 456 -> 456, 789 -> 789)

scala> imm
  Map()

scala> imm + ("123" -> 123) + ("456" -> 456) + ("789" -> 789)
  Map(123 -> 123, 456 -> 456, 789 -> 789)

```

. "" .

()

!

2 Map ma mb Map .

```
def merge2Maps(ma: Map[String, Int], mb: Map[String, Int]): Map[String, Int]
```

for ((k, v) <- map) .

```

def merge2Maps(ma: ..., mb: ...): Map[String, Int] = {

  for ((k, v) <- mb) {
    ???
  }

}

```

for . for :

```

// this:
for ((k, v) <- map) { ??? }

// is equivalent to:
map.foreach { case (k, v) => ??? }

```

" ?"

foreach . foreach "" . var result .

result

ma mb scala.collection.immutable.Map, result ma :

```
val result = mutable.Map() ++ ma
```

```
mb ma key mb.
```

```
mb.foreach { case (k, v) => result += (k -> v) }
```

```
" " .
```

```
def merge2Maps(ma: Map[String, Int], mb: Map[String, Int]): Map[String, Int] = {  
  val result = scala.collection.mutable.Map() ++ ma  
  mb.foreach { case (k, v) => result += (k -> v) }  
  result.toMap // to get back an immutable Map  
}
```

```
:
```

```
scala> merge2Maps(Map("a" -> 11, "b" -> 12), Map("b" -> 22, "c" -> 23))  
Map(a -> 11, b -> 22, c -> 23)
```

```
foreach ? .foldLeft :
```

```
def merge2Maps(ma: Map[String, Int], mb: Map[String, Int]): Map[String, Int] = {  
  mb.foldLeft(ma) { case (result, (k, v)) => result + (k -> v) }  
  // or more concisely mb.foldLeft(ma) { _ + _ }  
}
```

```
"".foldLeft zero ma .
```

```
Map , Map () .
```

```
.foldLeft . .
```

: <https://riptutorial.com/ko/scala/topic/6298/---->

26: , Def

val `foo` = `bar`.

, Int PlusOne .Int , + val .

```
trait PlusOne {  
  val i: Int  
  
  val incr = i + 1  
}  
  
class IntWrapper(val i: Int) extends PlusOne
```

i IntWrapper .incr 1 incr , i 0.(Nil , null .)

val ., lazy val def. lazy val a val , .

Sciddle-Js () .

Examples

, Def

var

var Java . var , var :

```
scala> var x = 1  
x: Int = 1  
  
scala> x = 2  
x: Int = 2  
  
scala> x = "foo bar"  
<console>:12: error: type mismatch;  
found   : String("foo bar")  
required: Int  
    x = "foo bar"  
    ^
```

var .

val . val .

```
scala> val y = 1  
y: Int = 1
```

```
scala> y = 2
<console>:12: error: reassignment to val
    y = 2
    ^
```

val . . .

```
scala> val arr = new Array[Int](2)
arr: Array[Int] = Array(0, 0)

scala> arr(0) = 1

scala> arr
res1: Array[Int] = Array(1, 0)
```

def

def . . .

```
scala> def z = 1
z: Int

scala> z = 2
<console>:12: error: value z_= is not a member of object $iw
    z = 2
    ^
```

val y def z . def val var . . .

```
scala> val a = {println("Hi"); 1}
Hi
a: Int = 1

scala> def b = {println("Hi"); 1}
b: Int

scala> a + 1
res2: Int = 2

scala> b + 1
Hi
res3: Int = 2
```

val / var / def . . .

```
scala> val x = (x: Int) => s"value=$x"
x: Int => String = <function1>

scala> var y = (x: Int) => s"value=$x"
y: Int => String = <function1>

scala> def z = (x: Int) => s"value=$x"
```

```

z: Int => String

scala> x(1)
res0: String = value=1

scala> y(2)
res1: String = value=2

scala> z(3)
res2: String = value=3

```

lazy val **a** val . , val .

val lazy . **REPL**:

```

scala> lazy val foo = {
  |   println("Initializing")
  |   "my foo value"
  | }
foo: String = <lazy>

scala> val bar = {
  |   println("Initializing bar")
  |   "my bar value"
  | }
Initializing bar
bar: String = my bar value

scala> foo
Initializing
res3: String = my foo value

scala> bar
res4: String = my bar value

scala> foo
res5: String = my foo value

```

. lazy val foo . val println bar . println foo - . bar println **execute** .



1. val .

```

lazy val tiresomeValue = {(1 to 1000000).filter(x => x % 113 == 0).sum}
if (scala.util.Random.nextInt > 1000) {
  println(tiresomeValue)
}

```

tiresomeValue . lazy val .

2. .

```

object comicBook {
  def main(args:Array[String]): Unit = {
    gotham.hero.talk()
    gotham.villain.talk()
  }
}

class Superhero(val name: String) {
  lazy val toLockUp = gotham.villain
  def talk(): Unit = {
    println(s"I won't let you win ${toLockUp.name}!")
  }
}

class Supervillain(val name: String) {
  lazy val toKill = gotham.hero
  def talk(): Unit = {
    println(s"Let me loosen up Gotham a little bit ${toKill.name}!")
  }
}

object gotham {
  val hero: Superhero = new Superhero("Batman")
  val villain: Supervillain = new Supervillain("Joker")
}

```

lazy . java.lang.NullPointerException . lazy .

def .

```

def printValue(x: Int) {
  println(s"My value is an integer equal to $x")
}

def printValue(x: String) {
  println(s"My value is a string equal to '$x'")
}

printValue(1)    // prints "My value is an integer equal to 1"
printValue("1") // prints "My value is a string equal to '1'"

```

,, .

def , . . , printUs() printUs() .

```

// print out the three arguments in order.
def printUs(one: String, two: String, three: String) =
  println(s"$one, $two, $three")

```

().

```

printUs("one", "two", "three")
printUs(one="one", two="two", three="three")
printUs("one", two="two", three="three")
printUs(three="three", one="one", two="two")

```

one, two, three .

X .. .

```
printUs("one", two="two", three="three") // prints 'one, two, three'
printUs(two="two", three="three", "one") // fails to compile: 'positional after named
argument'
```

, Def : <https://riptutorial.com/ko/scala/topic/3155/----def>

27:

. ParArray , ParVector , mutable.ParHashMap , immutable.ParHashMap ParRange .

Examples

par . seq . Vector ParVector .

```
scala> val vect = (1 to 5).toVector
vect: Vector[Int] = Vector(1, 2, 3, 4, 5)

scala> val parVect = vect.par
parVect: scala.collection.parallel.immutable.ParVector[Int] = ParVector(1, 2, 3, 4, 5)

scala> parVect.seq
res0: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3, 4, 5)
```

par .

```
scala> vect.map(_ * 2)
res1: scala.collection.immutable.Vector[Int] = Vector(2, 4, 6, 8, 10)

scala> vect.par.map(_ * 2)
res2: scala.collection.parallel.immutable.ParVector[Int] = ParVector(2, 4, 6, 8, 10)
```

, , .

.

.

() .

```
scala> val list = (1 to 1000).toList
list: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10...)

scala> list.reduce(_ - _)
res0: Int = -500498

scala> list.reduce(_ - _)
res1: Int = -500498

scala> list.reduce(_ - _)
res2: Int = -500498

scala> val listPar = list.par
listPar: scala.collection.parallel.immutable.ParSeq[Int] = ParVector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10...)

scala> listPar.reduce(_ - _)
res3: Int = -408314
```

```
scala> listPar.reduce(_ - _)
res4: Int = -422884

scala> listPar.reduce(_ - _)
res5: Int = -301748
```

foreach . reduce map reduce map .

```
scala> val wittyOneLiner = Array("Artificial", "Intelligence", "is", "no", "match", "for",
"natural", "stupidity")

scala> wittyOneLiner.foreach(word => print(word + " "))
Artificial Intelligence is no match for natural stupidity

scala> wittyOneLiner.par.foreach(word => print(word + " "))
match natural is for Artificial no stupidity Intelligence

scala> print(wittyOneLiner.par.reduce(_ + " " + _))
Artificial Intelligence is no match for natural stupidity

scala> val list = (1 to 100).toList
list: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15...
```

: <https://riptutorial.com/ko/scala/topic/3882/>-

28:

Examples

.

```
sealed trait SuperType
case object A extends SuperType
case object B extends SuperType
case object C extends SuperType

val pfA: PartialFunction[SuperType, Int] = {
  case A => 5
}

val pfB: PartialFunction[SuperType, Int] = {
  case B => 10
}

val input: Seq[SuperType] = Seq(A, B, C)

input.map(pfA orElse pfB orElse {
  case _ => 15
}) // Seq(5, 10, 15)
```

`orElse` . . , .

. [Akka Actor](#) .

`collect`

(case _) . `collect` . / .

1

.

```
val sqRoot: PartialFunction[Double, Double] = { case n if n > 0 => math.sqrt(n) }
```

`collect` **combinator** :

```
List(-1.1, 2.2, 3.3, 0).collect(sqRoot)
```

.

```
List(-1.1, 2.2, 3.3, 0).filter(sqRoot.isDefinedAt).map(sqRoot)
```

2

```
sealed trait SuperType // `sealed` modifier allows inheritance within current build-unit only
```

```

case class A(value: Int) extends SuperType
case class B(text: String) extends SuperType
case object C extends SuperType

val input: Seq[SuperType] = Seq(A(5), B("hello"), C, A(25), B(""))

input.collect {
  case A(value) if value < 10    => value.toString
  case B(text) if text.nonEmpty => text
} // Seq("5", "hello")

```

.

- . case .
- .
- (if) .

partial , .Function1 PartialFunction . case .

```

val pf: PartialFunction[Boolean, Int] = {
  case true => 7
}

pf.isDefinedAt(true) // returns true
pf(true) // returns 7

pf.isDefinedAt(false) // returns false
pf(false) // throws scala.MatchError: false (of class java.lang.Boolean)

```

. Function1 **total** .

. case .

```

sealed trait SuperType // `sealed` modifier allows inheritance within current build-unit only
case object A extends SuperType
case object B extends SuperType
case object C extends SuperType

val input: Seq[SuperType] = Seq(A, B, C)

input.map {
  case A => 5
  case _ => 10
} // Seq(5, 10, 10)

```

match .:

```

input.map { item =>
  item match {
    case A => 5
    case _ => 10
  }
} // Seq(5, 10, 10)

```

case .

```
val input = Seq("A" -> 1, "B" -> 2, "C" -> 3)

input.map { case (a, i) =>
  a + i.toString
} // Seq("A1", "B2", "C3")
```

.

```
val numberNames = Map(1 -> "One", 2 -> "Two", 3 -> "Three")

// 1. No extraction
numberNames.map(it => s"${it._1} is written ${it._2}" )

// 2. Extraction within a normal function
numberNames.map(it => {
  val (number, name) = it
  s"$number is written $name"
})

// 3. Extraction via a partial function (note the brackets in the parentheses)
numberNames.map({ case (number, name) => s"$number is written $name" })
```

. throw.

: <https://riptutorial.com/ko/scala/topic/1638/>

29:

- `case Foo () //`
- `case Foo (a1 : A1, ..., aN : AN) // a1 ... aN case .`
- `case // case`

Examples

`equals .`

:

```
class Foo(val i: Int)
val a = new Foo(3)
val b = new Foo(3)
println(a == b) // "false" because they are different objects
```

:

```
case class Foo(i: Int)
val a = Foo(3)
val b = Foo(3)
println(a == b) // "true" because their members have the same value
```

`case . . :`

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

`... .`

```
class Person(val name: String, val age: Int)
  extends Product with Serializable
{
  def copy(name: String = this.name, age: Int = this.age): Person =
    new Person(name, age)

  def productArity: Int = 2

  def productElement(i: Int): Any = i match {
    case 0 => name
    case 1 => age
    case _ => throw new IndexOutOfBoundsException(i.toString)
  }

  def productIterator: Iterator[Any] =
    scala.runtime.ScalaRunTime.typedProductIterator(this)

  def productPrefix: String = "Person"

  def canEqual(obj: Any): Boolean = obj.isInstanceOf[Person]
```

```

override def hashCode(): Int = scala.runtime.ScalaRunTime._hashCode(this)

override def equals(obj: Any): Boolean = this.eq(obj) || obj match {
  case that: Person => this.name == that.name && this.age == that.age
  case _ => false
}

override def toString: String =
  scala.runtime.ScalaRunTime._toString(this)
}

```

/ case .

```

object Person extends AbstractFunction2[String, Int, Person] with Serializable {
  def apply(name: String, age: Int): Person = new Person(name, age)

  def unapply(p: Person): Option[(String, Int)] =
    if(p == null) None else Some((p.name, p.age))
}

```

object / case (). toString hashCode . / ().

```

object Foo extends Product with Serializable {
  def productArity: Int = 0

  def productIterator: Iterator[Any] =
    scala.runtime.ScalaRunTime.typedProductIterator(this)

  def productElement(i: Int): Any =
    throw new IndexOutOfBoundsException(i.toString)

  def productPrefix: String = "Foo"

  def canEqual(obj: Any): Boolean = obj.isInstanceOf[this.type]

  override def hashCode(): Int = 70822 // "Foo".hashCode()

  override def toString: String = "Foo"
}

```

case .

.

- public ().

```

case class Dog1(age: Int)
val x = Dog1(18)
println(x.age) // 18 (success!)

class Dog2(age: Int)
val x = new Dog2(18)
println(x.age) // Error: "value age is not a member of Dog2"

```

- toString , equals , hashCode (), copy , apply unapply .

```
case class Dog(age: Int)
val d1 = Dog(10)
val d2 = d1.copy(age = 15)
```

- .

```
sealed trait Animal // `sealed` modifier allows inheritance within current build-unit only
case class Dog(age: Int) extends Animal
case class Cat(owner: String) extends Animal
val x: Animal = Dog(18)
x match {
  case Dog(x) => println(s"It's a $x years old dog.")
  case Cat(x) => println(s"This cat belongs to $x.")
}
```

Immutability

Scala `val` `val ., .` mutator `. :`

```
case class Foo(i: Int)

val fooInstance = Foo(1)
val j = fooInstance.i // get
fooInstance.i = 2      // compile-time exception (mutation: reassignment to val)
```

case `var` case `.`

```
case class Bar(var i: Int)

val barInstance = Bar(1)
val j = barInstance.i // get
barInstance.i = 2      // set
```

case 'mutable' case `.`

```
import scala.collection._

case class Bar(m: mutable.Map[Int, Int])

val barInstance = Bar(mutable.Map(1 -> 2))
barInstance.m.update(1, 3) // mutate m
barInstance                // Bar(Map(1 -> 3))
```

" m m., m , . instanceA instanceB :

```
import scala.collection.mutable

case class Bar(m: mutable.Map[Int, Int])

val m = mutable.Map(1 -> 2)
val barInstanceA = Bar(m)
val barInstanceB = Bar(m)
```

```
barInstanceA.m.update(1,3)
barInstanceA // Bar = Bar(Map(1 -> 3))
barInstanceB // Bar = Bar(Map(1 -> 3))
m // scala.collection.mutable.Map[Int,Int] = Map(1 -> 3)
```

copy .

. .

```
case class Person(firstName: String, lastName: String, grade: String, subject: String)
val putu = Person("Putu", "Kevin", "A1", "Math")
val mark = putu.copy(firstName = "Ketut", lastName = "Mark")
// mark: People = People(Ketut,Mark,A1,Math)
```

copy (firstName lastName) (grade = A1 , subject = Math) .

. , name Person . , name String encode. , Person name String String .

```
def logError(message: ErrorMessage): Unit = ???
case class Person(name: String)
val maybeName: Either[String, String] = ??? // Left is error, Right is name
maybeName.foreach(logError) // But that won't stop me from logging the name as an error!
```

.

```
case class PersonName(value: String)
case class ErrorMessage(value: String)
case class Person(name: PersonName)
```

PersonName ErrorMessage String .

```
val maybeName: Either[ErrorMessage, PersonName] = ???
maybeName.foreach(reportError) // ERROR: tried to pass PersonName; ErrorMessage expected
maybeName.swap.foreach(reportError) // OK
```

PersonName / String **box / unbox** . PersonName ErrorMessage .

```
case class PersonName(val value: String) extends AnyVal
case class ErrorMessage(val value: String) extends AnyVal
```

: <https://riptutorial.com/ko/scala/topic/1022/>-

30:

Examples

```
import scala.concurrent.Future
import scala.concurrent.ExecutionContext.Implicits.global

object FutureDivider {
  def divide(a: Int, b: Int): Future[Int] = Future {
    // Note that this is integer division.
    a / b
  }
}
```

divide 55 0 .

.map { _ => "FutureDivider divide 55 0 = " + a b ? }

```
object Calculator {
  def calculateAndReport(a: Int, b: Int) = {
    val eventualQuotient = FutureDivider divide(a, b)

    eventualQuotient map {
      quotient => println(quotient)
    }
  }
}
```

Future 55 0 . "Creating a Future" divide 55 0 ? 0 ArithmeticException . ? .

recover { _ => ... }

```
object Calculator {
  def calculateAndReport(a: Int, b: Int) = {
    val eventualQuotient = FutureDivider divide(a, b)

    eventualQuotient recover {
      case ex: ArithmeticException => println(s"It failed with: ${ex.getMessage}")
    }
  }
}
```

failed . .

```
object Calculator {
  def calculateAndReport(a: Int, b: Int) = {
    val eventualQuotient = FutureDivider divide(a, b)

    // Note the use of the dot operator to get the failed projection and map it.
    eventualQuotient.failed.map {
      ex => println(s"It failed with: ${ex.getMessage}")
    }
  }
}
```

```
}
}
```

▪

• • •

```
object Calculator {
  def calculateAndReport(a: Int, b: Int) = {
    val eventualQuotient = FutureDivider divide(a, b)

    eventualQuotient map {
      quotient => println(s"Quotient: $quotient")
    } recover {
      case ex: ArithmeticException => println(s"It failed with: ${ex.getMessage}")
    }
  }
}
```

`.List[Future[Int]]` `List[Int]` `.List[Future[Int]]` `Future[List[Int]]` `.Future` `sequence` .

```
def listOfFuture: List[Future[Int]] = List(1,2,3).map(Future(_))
def futureOfList: Future[List[Int]] = Future.sequence(listOfFuture)
```

`sequence` `F` `G` `F[G[T]]` `G[F[T]]` .

`traverse` . `x => x` `sequence` .

```
def listOfFuture: List[Future[Int]] = List(1,2,3).map(Future(_))
def futureOfList: Future[List[Int]] = Future.traverse(listOfFuture)(x => x)
```

`,` `,` `listOfFuture` `..` `Future` .

```
def futureOfList: Future[List[Int]] = Future.traverse(List(1,2,3))(Future(_))
```

`List(1,2,3)` `identity` `x => x` `Future(_)` `Int` `Future` . `List[Future[Int]]` .

▪

for comprehension .

`f1`, `f2`, `f3` **3** `Future[String]` `one`, `two`, `three` .

```
val fCombined =
  for {
    s1 <- f1
    s2 <- f2
    s3 <- f3
  } yield (s"$s1 - $s2 - $s3")
```

`fCombined` `one - two - three` `Future[String]` .

ExectContext .

flatMap . .

```
val result1 = for {  
  first <- Future {  
    Thread.sleep(2000)  
    System.currentTimeMillis()  
  }  
  second <- Future {  
    Thread.sleep(1000)  
    System.currentTimeMillis()  
  }  
} yield first - second  
  
val fut1 = Future {  
  Thread.sleep(2000)  
  System.currentTimeMillis()  
}  
val fut2 = Future {  
  Thread.sleep(1000)  
  System.currentTimeMillis()  
}  
val result2 = for {  
  first <- fut1  
  second <- fut2  
} yield first - second
```

result1 result2 .

yield <http://docs.scala-lang.org/tutorials/FAQ/yield.html> .

: <https://riptutorial.com/ko/scala/topic/3245/>

31:

- `{selfId => /      selfId this /}`
- `{selfId : OtherType => /*   selfId   OtherType . */`
- `{selfId : OtherType2 OtherType1 => /* selfId OtherType1 OtherType2 */`

.

Examples

. this ().

. .

```
trait Container[+T] {  
  def add(o: T): Unit  
}  
  
trait PermanentStorage[T] {  
  /* Constraint on self type: it should be Container  
   * we can refer to that type as `identifier`, usually `this` or `self`  
   * or the type's name is used. */  
  identifier: Container[T] =>  
  
  def save(o: T): Unit = {  
    identifier.add(o)  
    //Do something to persist too.  
  }  
}
```

Container PermanentStorage .

: <https://riptutorial.com/ko/scala/topic/4639/->

32:

Scalaz Scala .

Scala . (: Functor , Monad) .

Examples

ApplyUsage

```
import scalaz._
import Scalaz._

scala> Apply[Option].apply2(some(1), some(2))((a, b) => a + b)
res0: Option[Int] = Some(3)

scala> val intToString: Int => String = _.toString

scala> Apply[Option].ap(1.some)(some(intToString))
res1: Option[String] = Some(1)

scala> Apply[Option].ap(none)(some(intToString))
res2: Option[String] = None

scala> val double: Int => Int = _ * 2

scala> Apply[List].ap(List(1, 2, 3))(List(double))
res3: List[Int] = List(2, 4, 6)

scala> :kind Apply
scalaz.Apply's kind is X[F[A]]
```

FunctorUsage

```
import scalaz._
import Scalaz._

scala> val len: String => Int = _.length
len: String => Int = $$Lambda$1164/969820333@7e758f40

scala> Functor[Option].map(Some("foo"))(len)
res0: Option[Int] = Some(3)

scala> Functor[Option].map(None)(len)
res1: Option[Int] = None

scala> Functor[List].map(List("qwer", "adsfg"))(len)
res2: List[Int] = List(4, 5)

scala> :kind Functor
scalaz.Functor's kind is X[F[A]]
```

ArrowUsage

```

import scalaz._
import Scalaz._
scala> val plus1 = (_: Int) + 1
plus1: Int => Int = $$Lambda$1167/1113119649@6a6bfd97

scala> val plus2 = (_: Int) + 2
plus2: Int => Int = $$Lambda$1168/924329548@6bbe050f

scala> val rev = (_: String).reverse
rev: String => String = $$Lambda$1227/1278001332@72685b74

scala> plus1.first apply (1, "abc")
res0: (Int, String) = (2,abc)

scala> plus1.second apply ("abc", 2)
res1: (String, Int) = (abc,3)

scala> rev.second apply (1, "abc")
res2: (Int, String) = (1,cba)

scala> plus1 *** rev apply(7, "abc")
res3: (Int, String) = (8,cba)

scala> plus1 &&& plus2 apply 7
res4: (Int, Int) = (8,9)

scala> plus1.product apply (1, 2)
res5: (Int, Int) = (2,3)

scala> :kind Arrow
scalaz.Arrow's kind is X[F[A1,A2]]

```

: <https://riptutorial.com/ko/scala/topic/9893/>-

33:

Examples

dpkg Linux

.deb . [Scala](#) . scala-xxxdeb .

scala deb :

```
sudo dpkg -i scala-x.x.x.deb
```

.

```
which scala
```

PATH . :

```
scala
```

Scala REPL ().

[Lightbend](#) [curl](#) :

```
curl -O http://downloads.lightbend.com/scala/2.xx.x/scala-2.xx.x.tgz
```

```
tar /usr/local/share /opt/bin .
```

```
unzip scala-2.xx.x.tgz
mv scala-2.xx.x /usr/local/share/scala
```

```
~/.profile ~/.bash_profile ~/.bashrc PATH ~/.bashrc .
```

```
$SCALA_HOME=/usr/local/share/scala
export PATH=$SCALA_HOME/bin:$PATH
```

.

```
which scala
```

PATH . scala :

```
scala
```

Scala REPL ().

OSX

[MacPorts](#) Mac OSX .

```
port list | grep scala
```

. (Scala 2.11) :

```
sudo port install scala2.11
```

(2.11 .)

\$PATH . :

```
which scala
```

.

```
scala
```

Scala REPL .

: [https://riptutorial.com/ko/scala/topic/2921/-](https://riptutorial.com/ko/scala/topic/2921/)

34:

Examples

(/).

+ - * / %		a + b
== != > < >= <=		a > b
&& & !		a && b
& ^ ~ << >> >>>		a & b , ~a , a >>> b
= += -= *= /= %= <<= >>= &= ^= ==		a += b

Java .

: () list::(value) value :: list . (1 :: 2 :: 3 :: Nil 1 :: (2 :: (3 :: Nil)))

.

```
class Team {
  def +(member: Person) = ...
}
```

.

```
ITTeam + Jack
```

```
ITTeam.+(Jack)
```

unary_ unary_ . : unary_!

```
class MyBigInt {
  def unary_! = ...
}

var a: MyBigInt = new MyBigInt
var b = !a
```

() []		
	! ~	

Multiplicative	* / %	
	+ -	
	>> >>> <<	
	> >= < <=	
	== !=	
	&	
xor	^	
	&&	
	= += -= *= /= %= >>= <<= &= ^= =	
	,	

. > >>> :

()
* / %
+ -
:
= !
< >
&
^
()
()

(: += , *=) . (=) <= , >= , == != , . , .

: <https://riptutorial.com/ko/scala/topic/6604/>-

35:

., . " .

Examples

genRandom 4 .

```
def genRandom: Stream[String] = {
  val random = scala.util.Random.nextFloat()
  println(s"Random value is: $random")
  if (random < 0.25) {
    Stream.empty[String]
  } else {
    ("%.3f : A random number" format random) #:: genRandom
  }
}

lazy val randos = genRandom // getRando is lazily evaluated as randos is iterated through

for {
  x <- randos
} println(x) // The number of times this prints is effectively randomized.
```

lazily recurses #:: #:: .

.

```
// factorial
val fact: Stream[BigInt] = 1 #:: fact.zipWithIndex.map{case (p,x)=>p*(x+1)}
fact.take(10) // (1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880)
fact(24) // 620448401733239439360000

// the Fibonacci series
val fib: Stream[BigInt] = 0 #:: fib.scan(1:BigInt)(_+_)
fib.take(10) // (0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
fib(124) // 36726740705505779255899443

// random Ints between 10 and 99 (inclusive)
def rndInt: Stream[Int] = (util.Random.nextInt(90)+10) #:: rndInt
rndInt.take(10) // (20, 95, 14, 44, 42, 78, 85, 24, 99, 85)
```

Var, Val, Def . def . **A** val . .

```
// def with extra output per calculation
def fact: Stream[Int] = 1 #:: fact.zipWithIndex.map{case (p,x)=>print("!");p*(x+1)}
fact(5) // !!!!!!!!!!!!!!! 120
fact(4) // !!!!!!!!!!! 24
fact(7) // !!!!!!!!!!!!!!! 5040

// now as val
val fact: Stream[Int] = 1 #:: fact.zipWithIndex.map{case (p,x)=>print("!");p*(x+1)}
fact(5) // !!!!! 120
```

```
fact(4) // 24
fact(7) // !! 5040
```

Stream val .

```
val rndInt: Stream[Int] = (util.Random.nextInt(90)+10) #:: rndInt
rndInt.take(5) // (79, 79, 79, 79, 79)
```

- referent

```
// Generate stream that references itself in its evaluation
lazy val primes: Stream[Int] =
  2 #:: Stream.from(3, 2)
    .filter { i => primes.takeWhile(p => p * p <= i).forall(i % _ != 0) }
    .takeWhile(_ > 0) // prevent overflowing

// Get list of 10 primes
assert(primes.take(10).toList == List(2, 3, 5, 7, 11, 13, 17, 19, 23, 29))

// Previously calculated values were memoized, as shown by toString
assert(primes.toString == "Stream(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ?)")
```

: <https://riptutorial.com/ko/scala/topic/3702/>

36:

Examples

(: +, -, *, ++). , 1 + 2 1.(2) . ' (infix operators)' .

, .

```
class Matrix(rows: Int, cols: Int, val data: Seq[Seq[Int]]){
  def +(that: Matrix) = {
    val newData = for (r <- 0 until rows) yield
      for (c <- 0 until cols) yield this.data(r)(c) + that.data(r)(c)

    new Matrix(rows, cols, newData)
  }
}
```

.

```
val a = new Matrix(2, 2, Seq(Seq(1,2), Seq(3,4)))
val b = new Matrix(2, 2, Seq(Seq(1,2), Seq(3,4)))

// could also be written a.+(b)
val sum = a + b
```

. . .

. . + add .

unary_ . unary_+ , unary_- , unary_! unary_! unary_~ :

```
class Matrix(rows: Int, cols: Int, val data: Seq[Seq[Int]]){
  def +(that: Matrix) = {
    val newData = for (r <- 0 until rows) yield
      for (c <- 0 until cols) yield this.data(r)(c) + that.data(r)(c)

    new Matrix(rows, cols, newData)
  }

  def unary_- = {
    val newData = for (r <- 0 until rows) yield
      for (c <- 0 until cols) yield this.data(r)(c) * -1

    new Matrix(rows, cols, newData)
  }
}
```

.

```
val a = new Matrix(2, 2, Seq(Seq(1,2), Seq(3,4)))
val negA = -a
```

• •
: <https://riptutorial.com/ko/scala/topic/2271/>-

37:

sealed trait case objects .

1. .

2. "Match is not exhaustive" . . scala.MatchError :

```
def isWeekendWithBug(day: WeekDays.Value): Boolean = day match {  
  case WeekDays.Sun | WeekDays.Sat => true  
}  
  
isWeekendWithBug(WeekDays.Fri)  
scala.MatchError: Fri (of class scala Enumeration$Val)
```

:

```
def isWeekendWithBug(day: WeekDay): Boolean = day match {  
  case WeekDay.Sun | WeekDay.Sat => true  
}  
  
Warning: match may not be exhaustive.  
It would fail on the following inputs: Fri, Mon, Thu, Tue, Wed  
def isWeekendWithBug(day: WeekDay): Boolean = day match {  
                                     ^
```

.

Examples

Enumeration .

```
object WeekDays extends Enumeration {  
  val Mon, Tue, Wed, Thu, Fri, Sat, Sun = Value  
}  
  
def isWeekend(day: WeekDays.Value): Boolean = day match {  
  case WeekDays.Sat | WeekDays.Sun => true  
  case _ => false  
}  
  
isWeekend(WeekDays.Sun)  
res0: Boolean = true
```

.

```
object WeekDays extends Enumeration {  
  val Mon = Value("Monday")  
  val Tue = Value("Tuesday")  
  val Wed = Value("Wednesday")  
  val Thu = Value("Thursday")  
  val Fri = Value("Friday")  
  val Sat = Value("Saturday")
```

```

    val Sun = Value("Sunday")
  }

  println(WeekDays.Mon)
  >> Monday

  WeekDays.withName("Monday") == WeekDays.Mon
  >> res0: Boolean = true

```

not-so-typesafe .

```

object Parity extends Enumeration {
  val Even, Odd = Value
}

WeekDays.Mon.isInstanceOf[Parity.Value]
>> res1: Boolean = true

```

Enumeration sealed .

```

sealed trait WeekDay

object WeekDay {
  case object Mon extends WeekDay
  case object Tue extends WeekDay
  case object Wed extends WeekDay
  case object Thu extends WeekDay
  case object Fri extends WeekDay
  case object Sun extends WeekDay
  case object Sat extends WeekDay
}

```

sealed WeekDay . WeekDay .

. .

```

val allWeekDays = Seq(Mon, Tue, Wed, Thu, Fri, Sun, Sat)

```

sealed . .

```

sealed trait CelestialBody

object CelestialBody {
  case object Earth extends CelestialBody
  case object Sun extends CelestialBody
  case object Moon extends CelestialBody
  case class Asteroid(name: String) extends CelestialBody
}

```

sealed . .

```

sealed trait WeekDay { val name: String }

object WeekDay {

```

```

    case object Mon extends WeekDay { val name = "Monday" }
    case object Tue extends WeekDay { val name = "Tuesday" }
    (...)
}

```

:

```

sealed case class WeekDay(name: String)

object WeekDay {
    object Mon extends WeekDay("Monday")
    object Tue extends WeekDay("Tuesday")
    (...)
}

```

allValues

. allElements .

.

```

import EnumerationMacros._

sealed trait WeekDay
object WeekDay {
    case object Mon extends WeekDay
    case object Tue extends WeekDay
    case object Wed extends WeekDay
    case object Thu extends WeekDay
    case object Fri extends WeekDay
    case object Sun extends WeekDay
    case object Sat extends WeekDay
    val allWeekDays: Set[WeekDay] = sealedInstancesOf[WeekDay]
}

```

.

```

import scala.collection.immutable.TreeSet
import scala.language.experimental.macros
import scala.reflect.macros.blackbox

/**
 * A macro to produce a TreeSet of all instances of a sealed trait.
 * Based on Travis Brown's work:
 * http://stackoverflow.com/questions/13671734/iteration-over-a-sealed-trait-in-scala
 * CAREFUL: !!! MUST be used at END OF code block containing the instances !!!
 */
object EnumerationMacros {
    def sealedInstancesOf[A]: TreeSet[A] = macro sealedInstancesOf_impl[A]

    def sealedInstancesOf_impl[A: c.WeakTypeTag](c: blackbox.Context) = {
        import c.universe._

        val symbol = weakTypeOf[A].typeSymbol.asClass

        if (!symbol.isClass || !symbol.isSealed)

```

```

    c.abort(c.enclosingPosition, "Can only enumerate values of a sealed trait or class.")
  else {

    val children = symbol.knownDirectSubclasses.toList

    if (!children.forall(_.isModuleClass)) c.abort(c.enclosingPosition, "All children must
be objects.")
    else c.Expr[TreeSet[A]] {

      def sourceModuleRef(sym: Symbol) =
Ident(sym.asInstanceOf[scala.reflect.internal.Symbols#Symbol
      ].sourceModule.asInstanceOf[Symbol]
    )

    Apply(
      Select(
        reify(TreeSet).tree,
        TermName("apply")
      ),
      children.map(sourceModuleRef(_))
    )
  }
}
}
}

```

: [https://riptutorial.com/ko/scala/topic/1499/-](https://riptutorial.com/ko/scala/topic/1499/)

38:

Examples

map, getOrElse flatMap Try :

```
import scala.util.Try

val i = Try("123".toInt)    // Success(123)
i.map(_ + 1).getOrElse(321) // 124

val j = Try("abc".toInt)    // Failure(java.lang.NumberFormatException)
j.map(_ + 1).getOrElse(321) // 321

Try("123".toInt) flatMap { i =>
  Try("234".toInt)
    .map(_ + i)
} // Success(357)
```

Try :

```
Try(parsePerson("John Doe")) match {
  case Success(person) => println(person.surname)
  case Failure(ex)    => // Handle error ...
}
```

/

```
def getPersonFromWebService(url: String): Either[String, Person] = {

  val response = webServiceClient.get(url)

  response.webService.status match {
    case 200 => {
      val person = parsePerson(response)
      if(!isValid(person)) Left("Validation failed")
      else Right(person)
    }

    case _ => Left(s"Request failed with error code $response.status")
  }
}
```

```
getPersonFromWebService("http://some-webservice.com/person") match {
  case Left(errorMessage) => println(errorMessage)
  case Right(person)      => println(person.surname)
}
```

```
val maybePerson: Option[Person] = getPersonFromWebService("http://some-
webservice.com/person").right.toOption
```

null null . , (Some) (None) Option .

try-catch `Option` .

`Option[T]` `Some(value)` (`T`) `None` :

```
def findPerson(name: String): Option[Person]
```

`None` . , `Person` `Some` . `Option` .

```
findPerson(personName) match {  
  case Some(person) => println(person.surname)  
  case None => println(s"No person found with name $personName")  
}
```

map getOrElse

```
val name = findPerson(personName).map(_.firstName).getOrElse("Unknown")  
println(name) // Prints either the name of the found person or "Unknown"
```

```
val name = findPerson(personName).fold("Unknown")(_.firstName)  
// equivalent to the map getOrElse example above.
```

Java

`Option` **null** **Java** :

```
val s: Option[String] = Option("hello")  
s.orNull              // "hello": String  
s.getOrElse(null)    // "hello": String  
  
val n: Option[Int] = Option(42)  
n.orNull              // compilation failure (Cannot prove that Null <: Int.)  
n.getOrElse(null)    // 42
```

`Future` `exception` `recover` `recover` .

,

```
def runFuture: Future = Future { throw new FairlyStupidException }  
  
val itWillBeAwesome: Future = runFuture
```

`... Future` `Exception` . `FairlyStupidException` `Exception` `FairlyStupidException` .

```
val itWillBeAwesomeOrIllRecover = runFuture recover {  
  case stupid: FairlyStupidException =>  
    BadRequest("Another stupid exception!")  
}
```

```
recover A PartialFunction Throwable, Future .
```

```
Future .
```

```
def runNotFuture: Unit = throw new FairlyStupidException

try {
  runNotFuture
} catch {
  case e: FairlyStupidException => BadRequest("Another stupid exception!")
}
```

```
Future . . .
```

try-catch

```
Try, Option Either ( ) - , .catch .
```

```
try {
  // ... might throw exception
} catch {
  case ioe: IOException => ... // more specific cases first
  case e: Exception => ...
  // uncaught types will be thrown
} finally {
  // ...
}
```

Either Option

```
Either Option scala.util.control.Exception .
```

```
import scala.util.control.Exception._

val plain = "71a"
val optionInt: Option[Int] = catching(classOf[java.lang.NumberFormatException]) opt {
  plain.toInt }
val eitherInt = Either[Throwable, Int] = catching(classOf[java.lang.NumberFormatException])
  either { plain.toInt }
```

: <https://riptutorial.com/ko/scala/topic/910/>-

39:

- `class [+ T] (value : T) Option [T] .`
- `[]`
- `[T] (: T)`

`Some(value) None .`

Examples

`Option 0 . None Some(x) x .`

```
val option: Option[String] = ???

option.map(_.trim) // None if option is None, Some(s.trim) if Some(s)
option.foreach(println) // prints the string if it exists, does nothing otherwise
option.forall(_.length > 4) // true if None or if Some(s) and s.length > 4
option.exists(_.length > 4) // true if Some(s) and s.length > 4
option.toList // returns an actual list
```

Null Option

`Java ( ) null . null Option . Option null .`

`None null Option . Some null Option .`

`.`

```
val nothing = Option(null) // None
val something = Option("Aren't options cool?") // Some("Aren't options cool?")
```

`null Java .`

```
val resource = Option(JavaLib.getResource())
// if null, then resource = None
// else resource = Some(resource)
```

`getResource() null resource None . Some(resource) . Option Option . None value == null`
`isDefined .`

```
val resource: Option[Resource] = Option(JavaLib.getResource())
if (resource.isDefined) { // resource is `Some(_)` type
  val r: Resource = resource.get
  r.connect()
}
```

`null .`

```
val resource: Option[Resource] = Option(JavaLib.getResource())
if (resource.isEmpty) { // resource is `None` type.
  System.out.println("Resource is empty! Cannot connect.")
}
```

Option foreach Option (" Option.get) .

```
val resource: Option[Resource] = Option(JavaLib.getResource())
resource foreach (r => r.connect())
// if r is defined, then r.connect() is run
// if r is empty, then it does nothing
```

Resource (Option[Resource]) Null Option . getOrElse .

```
lazy val defaultResource = new Resource()
val resource: Resource = Option(JavaLib.getResource()).getOrElse(defaultResource)
```

Option , Option null :

```
val resource: Option[Resource] = ???
JavaLib.sendResource(resource.orNull)
JavaLib.sendResource(resource.getOrElse(defaultResource)) //
```

Option . Option 0 .

Option : Some None .

Some None .

Option null . NullPointerException Map , FlatMap .

```
val countries = Map(
  "USA" -> "Washington",
  "UK" -> "London",
  "Germany" -> "Berlin",
  "Netherlands" -> "Amsterdam",
  "Japan" -> "Tokyo"
)

println(countries.get("USA")) // Some(Washington)
println(countries.get("France")) // None
println(countries.get("USA").get) // Washington
println(countries.get("France").get) // Error: NoSuchElementException
println(countries.get("USA").getOrElse("Nope")) // Washington
println(countries.get("France").getOrElse("Nope")) // Nope
```

Option[A] . .

Option flatMap . . Option .

```
val firstOption: Option[Int] = Option(1)
val secondOption: Option[Int] = Option(2)
```

```
val myResult = for {
  firstValue <- firstOption
  secondValue <- secondOption
} yield firstValue + secondValue
// myResult: Option[Int] = Some(3)
```

None None .

```
val firstOption: Option[Int] = Option(1)
val secondOption: Option[Int] = None

val myResult = for {
  firstValue <- firstOption
  secondValue <- secondOption
} yield firstValue + secondValue
// myResult: Option[Int] = None
```

: Monad . (Monad)

. Option Iterable .toIterable Option Iterable .

```
val option: Option[Int] = Option(1)
val iterable: Iterable[Int] = Iterable(2, 3, 4, 5)

// does NOT compile since we cannot mix Monads in a for comprehension
// val myResult = for {
//   optionValue <- option
//   iterableValue <- iterable
//} yield optionValue + iterableValue

// It does compile when adding a .toIterable on the option
val myResult = for {
  optionValue <- option.toIterable
  iterableValue <- iterable
} yield optionValue + iterableValue
// myResult: Iterable[Int] = List(2, 3, 4, 5)
```

: . .toIterable .toIterable () .

: <https://riptutorial.com/ko/scala/topic/2293/>-

Examples

1. `SCALA_PROJECT/build.gradle` .

```
group 'scala_gradle'
version '1.0-SNAPSHOT'

apply plugin: 'scala'

repositories {
    jcenter()
    mavenCentral()
    maven {
        url "https://repo.typesafe.com/typesafe/maven-releases"
    }
}

dependencies {
    compile group: 'org.scala-lang', name: 'scala-library', version: '2.10.6'
}

task "create-dirs" << {
    sourceSets*.scala.srcDirs*.each { it.mkdirs() }
    sourceSets*.resources.srcDirs*.each { it.mkdirs() }
}
```

2. `gradle tasks` `gradle tasks` .

3. `src/scala, src/resources` `gradle create-dirs` .

4. `gradle build` .

Gradle Scala

Basic Setup `Scala Gradle`

`Gradle` , .

`Gradle` .

`Gradle` `Gradle API` . `Java` . `Groovy` . `Scala` , , `gradle` [github repo](#) .

.

- 2 `scalaVersion` `scalaVersion`
 - `"2.12"`
 - `"0"`
-

```
withScalaVersion (sbt %% . )
• createDirs
```

1. gradle build.gradle .

```
apply plugin: 'scala'
apply plugin: 'maven'

repositories {
    mavenLocal()
    mavenCentral()
}

dependencies {
    compile gradleApi()
    compile "org.scala-lang:scala-library:2.12.0"
}
```

- ```
:
```
- Scala Gradle Scala Plugin .
  - Gradle Maven Plugin . jar Maven install
  - compile gradleApi() gradle-api-<gradle\_version>.jar classpath .

## 2. .

```
package com.btesila.gradle.plugins

import org.gradle.api.{Plugin, Project}

class ScalaCustomPlugin extends Plugin[Project] {
 override def apply(project: Project): Unit = {
 project.getPlugins.apply("scala")
 }
}
```

- ```
:
```
- Project Plugin apply .
 - apply Project .
 - Gradle Scala Plugin .

3. scalaVersion .

```
ScalaVersion . .
```

```
class ScalaVersion {
    var major: String = "2.12"
    var minor: String = "0"
}
```

```
Gradle . gradle Project ExtensionContainer . .
```

apply .

- scalaVersion
- ScalaVersion ScalaVersion .

```
var scalaVersion = new ScalaVersion
if (!project.getExtensions.getExtraProperties.has("scalaVersion"))
    project.getExtensions.getExtraProperties.set("scalaVersion", scalaVersion)
else
    scalaVersion =
project.getExtensions.getExtraProperties.get("scalaVersion").asInstanceOf[ScalaVersion]
```

.

```
ext {
    scalaVersion.major = "2.12"
    scalaVersion.minor = "0"
}
```

4. scalaVersion scala-lang .

```
project.getDependencies.add("compile", s"org.scala-lang:scala-
library:${scalaVersion.major}.${scalaVersion.minor}")
```

.

```
compile "org.scala-lang:scala-library:2.12.0"
```

5. withScalaVersion .

```
val withScalaVersion = (lib: String) => {
    val libComp = lib.split(":")
    libComp.update(1, s"${libComp(1)}_${scalaVersion.major}")
    libComp.mkString(":")
}
project.getExtensions.getExtraProperties.set("withScalaVersion", withScalaVersion)
```

6. createDirs .

DefaultTask **Gradle** .

```
class CreateDirs extends DefaultTask {
    @TaskAction
    def createDirs(): Unit = {
        val sourceSetContainer =
this.getProject.getConvention.getPlugin(classOf[JavaPluginConvention]).getSourceSets

        sourceSetContainer.forEach { sourceSet =>
            sourceSet.getAllSource.getSrcDirs.forEach(file => if (!file.getName.contains("java"))
file.mkdirs())
        }
    }
}
```

```
}  
}
```

: SourceSetContainer . Gradle Scala Plugin Java .

apply createDir .

```
project.getTasks.create("createDirs", classOf[CreateDirs])
```

ScalaCustomPlugin .

```
class ScalaCustomPlugin extends Plugin[Project] {  
  override def apply(project: Project): Unit = {  
    project.getPlugins.apply("scala")  
  
    var scalaVersion = new ScalaVersion  
    if (!project.getExtensions.getExtraProperties.has("scalaVersion"))  
      project.getExtensions.getExtraProperties.set("scalaVersion", scalaVersion)  
    else  
      scalaVersion =  
project.getExtensions.getExtraProperties.get("scalaVersion").asInstanceOf[ScalaVersion]  
  
    project.getDependencies.add("compile", s"org.scala-lang:scala-  
library:${scalaVersion.major}.${scalaVersion.minor}")  
  
    val withScalaVersion = (lib: String) => {  
      val libComp = lib.split(":")  
      libComp.update(1, s"${libComp(1)}_${scalaVersion.major}")  
      libComp.mkString(":")  
    }  
    project.getExtensions.getExtraProperties.set("withScalaVersion", withScalaVersion)  
  
    project.getTasks.create("createDirs", classOf[CreateDirs])  
  }  
}
```

Maven

gradle install .
~/.m2/repository .

Gradle ?

Gradle apply id apply . , Gradle ID scala .

```
apply plugin: 'scala'
```

.

```
apply plugin: "com.btesila.scala.plugin"
```

```
com.btesila.scala.plugin id .
```

```
id .
```

src / main / resources / META-INF / gradle-plugin / com.btesil.scala.plugin.properties

```
implementation-class=com.btesila.gradle.plugins.ScalaCustomPlugin
```

```
, gradle install .
```

1. Gradle

```
buildscript {
    repositories {
        mavenLocal()
        mavenCentral()
    }

    dependencies {
        //modify this path to match the installed plugin project in your local repository
        classpath 'com.btesila:working-with-gradle:1.0-SNAPSHOT'
    }
}

repositories {
    mavenLocal()
    mavenCentral()
}

apply plugin: "com.btesila.scala.plugin"
```

2. gradle createDirs gradle createDirs - .

3. .

```
ext {
    scalaVersion.major = "2.11"
    scalaVersion.minor = "8"
}

println(project.ext.scalaVersion.major)
println(project.ext.scalaVersion.minor)
```

4.

```
dependencies {
    compile withScalaVersion("com.typesafe.scala-logging:scala-logging:3.5.0")
}
```

! .

: <https://riptutorial.com/ko/scala/topic/3304/>-

41: ()

Examples

Option . ().

```
sealed abstract class Option[+A] {
  def isEmpty: Boolean
  def get: A

  final def fold[B](ifEmpty: => B)(f: A => B): B =
    if (isEmpty) ifEmpty else f(this.get)

  // lots of methods...
}

case class Some[A](value: A) extends Option[A] {
  def isEmpty = false
  def get = value
}

case object None extends Option[Nothing] {
  def isEmpty = true
  def get = throw new NoSuchElementException("None.get")
}
```

B fold .

. x A x *type* .

```
def f[A](x: A): A = x

f(1)           // 1
f("two")       // "two"
f[Float](3)    // 3.0F
```

Scala . : * A * .

```
def g[A](x: A): A = 2 * x // Won't compile
```

Ints

```
trait IntList { ... }

class Cons(val head: Int, val tail: IntList) extends IntList { ... }

class Nil extends IntList { ... }
```

, ?

```
trait List[T] {  
  def isEmpty: Boolean  
  def head: T  
  def tail: List[T]  
}  
  
class Cons[T](val head: T, val tail: List[T]) extends List[T] {  
  def isEmpty: Boolean = false  
}  
  
class Nil[T] extends List[T] {  
  def isEmpty: Boolean = true  
  
  def head: Nothing = throw NoSuchElementException("Nil.head")  
  
  def tail: Nothing = throw NoSuchElementException("Nil.tail")  
}
```

() : <https://riptutorial.com/ko/scala/topic/782/----->

42:

Examples

. . .

```
, Projection type Apply[A] , def apply[A](a: A): Apply[A] def apply[A](a: A):  
Apply[A] .
```

```
trait Projection {  
  type Apply[A] // <: Any  
  def apply[A](a: A): Apply[A]  
}
```

```
type Apply[A] ( ) .
```

```
map . , HList .
```

```
sealed trait HList {  
  type Map[P <: Projection] <: HList  
  def map[P <: Projection](p: P): Map[P]  
}
```

```
HNil , . trait HNil HNil HNil.type :
```

```
sealed trait HNil extends HList  
case object HNil extends HNil {  
  type Map[P <: Projection] = HNil  
  def map[P <: Projection](p: P): Map[P] = HNil  
}
```

```
HCons . ( P#Apply[H] ) HList ( T#Map[P] ) HList :
```

```
case class HCons[H, T <: HList](head: H, tail: T) extends HList {  
  type Map[P <: Projection] = HCons[P#Apply[H], T#Map[P]]  
  def map[P <: Projection](p: P): Map[P] = HCons(p.apply(head), tail.map(p))  
}
```

```
. HCons[Option[String], HCons[Option[Int], HNil]] .
```

```
HCons("1", HCons(2, HNil)).map(new Projection {  
  type Apply[A] = Option[A]  
  def apply[A](a: A): Apply[A] = Some(a)  
})
```

: <https://riptutorial.com/ko/scala/topic/3738/-->

43:

Examples

+ (covariant) . "Producer A (covariant)".

```
trait Producer[+A] {  
  def produce: A  
}
```

""" .A Producer[X] <: Producer[Y] X <: Y . Producer[Cat] Producer[Animal] .

() . Co[Cat] <: Co[Animal] Co[Cat] def handle(a: Cat): Unit Co[Animal] Animal def handle(a: Cat): Unit !

```
trait Co[+A] {  
  def produce: A  
  def handle(a: A): Unit  
}
```

. B A . Option[X] <: Option[Y] X <: Y , Option[X] def getOrElse[B >: X](b: => B): B X -
Option[Y] Y .

```
trait Option[+A] {  
  def getOrElse[B >: A](b: => B): B  
}
```

.trait A[B] "A B". A[Cat] A[Animal] . A[Cat] <: A[Animal] A[Cat] >: A[Animal] Cat Animal
A[Cat] >: A[Animal] .

.

- . "Handler A".

```
trait Handler[-A] {  
  def handle(a: A): Unit  
}
```

contravariant type """ .A Handler[X] <: Handler[Y] X >: Y . Handler[Animal] Handler[Cat]
Handler[Animal] .

() . Contra[Animal] <: Contra[Cat] Contra[Animal] <: Contra[Cat] Contra[Animal] def produce:
Animal Contra[Cat] def produce: Animal !

```
trait Contra[-A] {  
  def handle(a: A): Unit  
  def produce: A  
}
```

contravariance contravariant type . Handler[Animal] Handler[Cat] " " .

. ofCat ofAnimal .

```
implicit def ofAnimal: Handler[Animal] = ???
implicit def ofCat: Handler[Cat] = ???

implicitly[Handler[Cat]].handle(new Cat)
```

scala.math.Ordering scala.math.Ordering T () typeclass .

```
trait Person
object Person {
  implicit def ordering[A <: Person]: Ordering[A] = ???
}
```

* (covariant) .

```
trait Animal { def name: String }
case class Dog(name: String) extends Animal

object Animal {
  def printAnimalNames(animals: Seq[Animal]) = {
    animals.foreach(animal => println(animal.name))
  }
}

val myDogs: Seq[Dog] = Seq(Dog("Curly"), Dog("Larry"), Dog("Moe"))

Animal.printAnimalNames(myDogs)
// Curly
// Larry
// Moe
```

, Seq[Dog] Seq[Animal] : A (kinded Seq).

* .

(covariant) (covariant argument) . T .

```
trait LocalVariance[T]{
  /// ??? throws a NotImplementedError
  def produce: T = ???
  // the implicit evidence provided by the compiler confirms that S is a
  // subtype of T.
  def handle[S](s: S)(implicit evidence: S <:: T) = {
    // and we can use the evidence to convert s into t.
    val t: T = evidence(s)
    ???
  }
}

trait A {}
trait B extends A {}
```

```
object Test {  
  val lv = new LocalVariance[A] {}  
  
  // now we can pass a B instead of an A.  
  lv.handle(new B {})  
}
```

: [https://riptutorial.com/ko/scala/topic/1651/-](https://riptutorial.com/ko/scala/topic/1651/)

44:

Examples

. ' ' .

```
val i = 1 + 2           // the type of i is Int
val s = "I am a String" // the type of s is String
def squared(x : Int) = x*x // the return type of squared is Int
```

. . .

```
val i: Int = 1 + 2
val s: String = "I am a String"
def squared(x : Int): Int = x*x
```

.

```
case class InferedPair[A, B](a: A, b: B)

val pairFirstInst = InferedPair("Husband", "Wife") //type is InferedPair[String, String]

// Equivalent, with type explicitly defined
val pairSecondInst: InferedPair[String, String]
    = InferedPair[String, String]("Husband", "Wife")
```

Java 7 [Diamond Operator](#) .

. , .

```
def add(a, b) = a + b // Does not compile
def add(a: Int, b: Int) = a + b // Compiles
def add(a: Int, b: Int): Int = a + b // Equivalent expression, compiles
```

.

```
// Does not compile
def factorial(n: Int) = if (n == 0 || n == 1) 1 else n * factorial(n - 1)
// Compiles
def factorial(n: Int): Int = if (n == 0 || n == 1) 1 else n * factorial(n - 1)
```

▪

.

.

```
def get[T]: Option[T] = ???
```

```
Nothing . . . NotNothing      ( RuntimeClass Nothing ClassTags RuntimeClass ).
```

```
@implicitNotFound("Nothing was inferred")
sealed trait NotNothing[-T]

object NotNothing {
  implicit object notNothing extends NotNothing[Any]
  //We do not want Nothing to be inferred, so make an ambiguous implicit
  implicit object `\n The error is because the type parameter was resolved to Nothing` extends
  NotNothing[Nothing]
  //For classtags, RuntimeClass can also be inferred, so making that ambiguous too
  implicit object ` \n The error is because the type parameter was resolved to RuntimeClass`
  extends NotNothing[RuntimeClass]
}

object ObjectStore {
  //Using context bounds
  def get[T: NotNothing]: Option[T] = {
    ???
  }

  def newArray[T](length: Int = 10)(implicit ct: ClassTag[T], evNotNothing: NotNothing[T]):
  Option[Array[T]] = ???
}
```

:

```
object X {
  //Fails to compile
  //val nothingInferred = ObjectStore.get

  val anOption = ObjectStore.get[String]
  val optionalArray = ObjectStore.newArray[AnyRef]()

  //Fails to compile
  //val runtimeClassInferred = ObjectStore.newArray()
}
```

: <https://riptutorial.com/ko/scala/topic/4918/>-

45:

(: [Apache Spark](#)) Serializable .

Examples

trait .

```
trait Show[A] {  
  def show(a: A): String  
}
```

. .

```
object Show {  
  implicit val intShow: Show[Int] = new Show {  
    def show(x: Int): String = x.toString  
  }  
  
  implicit val dateShow: Show[java.util.Date] = new Show {  
    def show(x: java.util.Date): String = x.getTime.toString  
  }  
  
  // ..etc  
}
```

.

```
def log[A](a: A)(implicit showInstance: Show[A]): Unit = {  
  println(showInstance.show(a))  
}
```

.

```
def log[A: Show](a: A): Unit = {  
  println(implicitly[Show[A]].show(a))  
}
```

log . Show[A] A log

```
log(10) // prints: "10"  
log(new java.util.Date(1469491668401L)) // prints: "1469491668401"  
log(List(1,2,3)) // fails to compile with  
                  // could not find implicit value for evidence parameter of type  
Show[List[Int]]
```

Show . String . toString toString . Show log String .

type .

```
trait Show[A] {
  def show: String
}
```

() , . Person Show .

```
class Person(val fullName: String) {
  def this(firstName: String, lastName: String) = this(s"$firstName $lastName")
}
```

Person companion Show :

```
object Person {
  implicit val personShow: Show[Person] = new Show {
    def show(p: Person): String = s"Person(${p.fullname})"
  }
}
```

class Person object Person .

Simple Class type type companion .

import . Simple Type Class log :

```
object MyShow {
  implicit val personShow: Show[Person] = new Show {
    def show(p: Person): String = s"Person(${p.fullname})"
  }
}

def logPeople(people: Person*): Unit = {
  import MyShow.personShow
  people foreach { p => log(p) }
}
```

. " " . / / .
.

```
// The mathematical definition of "Addable" is "Semigroup"
trait Addable[A] {
  def add(x: A, y: A): A
}
```

().

```
object Instances {

  // Instance for Int
  // Also called evidence object, meaning that this object saw that Int learned how to be
  added
  implicit object addableInt extends Addable[Int] {
    def add(x: Int, y: Int): Int = x + y
  }
}
```

```

}

// Instance for String
implicit object addableString extends Addable[String] {
  def add(x: String, y: String): String = x + y
}

}

```

Addable .

```

import Instances._
val three = addableInt.add(1,2)

```

1.add(2) . Addable "Operation Class"("Ops Class") .

```

object Ops {
  implicit class AddableOps[A](self: A)(implicit A: Addable[A]) {
    def add(other: A): A = A.add(self, other)
  }
}

```

add Int String .

```

object Main {

  import Instances._ // import evidence objects into this scope
  import Ops._       // import the wrappers

  def main(args: Array[String]): Unit = {

    println(1.add(5))
    println("mag".add("net"))
    // println(1.add(3.141)) // Fails because we didn't create an instance for Double

  }
}

```

"Ops" :

```

import simulacrum._

@typeclass trait Addable[A] {
  @op("|+|") def add(x: A, y: A): A
}

```

: <https://riptutorial.com/ko/scala/topic/3835/>-

46:

Examples

Cake Pattern.

```
//create a component that will be injected
trait TimeUtil {
  lazy val timeUtil = new TimeUtilImpl()

  class TimeUtilImpl{
    def now() = new DateTime()
  }
}

//main controller is depended on time util
trait MainController {
  _ : TimeUtil => //inject time util into main controller

  lazy val mainController = new MainControllerImpl()

  class MainControllerImpl {
    def printCurrentTime() = println(timeUtil.now()) //timeUtil is injected from TimeUtil
  }
}

object MainApp extends App {
  object app extends MainController
    with TimeUtil //wire all components

  app.mainController.printCurrentTime()
}
```

MainController TimeUtil .

TimeUtil MainController (_ : TimeUtil =>). TimeUtil , MainController TimeUtil .

(: TimeUtilImpl) . .

. Guice Binding .

: [https://riptutorial.com/ko/scala/topic/5909/-](https://riptutorial.com/ko/scala/topic/5909/)

47:

Examples

```
.  
.  
.  
  
.    (@tailrec)    .  
.  
.
```

```
def fact(i : Int) : Int = {  
    if(i <= 1) i  
    else i * fact(i-1)  
}  
  
println(fact(5))
```

```
.  
  
(fact 5)  
(* 5 (fact 4))  
(* 5 (* 4 (fact 3)))  
(* 5 (* 4 (* 3 (fact 2))))  
(* 5 (* 4 (* 3 (* 2 (fact 1)))))  
(* 5 (* 4 (* 3 (* 2 (* 1 (fact 0))))))  
(* 5 (* 4 (* 3 (* 2 (* 1 * 1)))))  
(* 5 (* 4 (* 3 (* 2))))  
(* 5 (* 4 (* 6)))  
(* 5 (* 24))  
120
```

@tailrec .could not optimize @tailrec annotated method fact: it contains a recursive call not in tail position

```
.  
  
def fact_with_tailrec(i : Int) : Long = {  
    @tailrec  
    def fact_inside(i : Int, sum: Long) : Long = {  
        if(i <= 1) sum  
        else fact_inside(i-1,sum*i)  
    }  
    fact_inside(i,1)  
}  
  
println(fact_with_tailrec(5))
```

```
.  
  
(fact_with_tailrec 5)  
(fact_inside 5 1)  
(fact_inside 4 5)
```

```
(fact_inside 3 20)
(fact_inside 2 60)
(fact_inside 1 120)
```

```
fact_inside . . , fact_with_tail 1000000 fact_with_tail 3 . .
```

(scala.util.control.TailCalls)

StackOverflowError . [TailCall](#) .

[TailCalls scaladoc](#)

```
import scala.util.control.TailCalls._

def isEven(xs: List[Int]): TailRec[Boolean] =
  if (xs.isEmpty) done(true) else tailcall(isOdd(xs.tail))

def isOdd(xs: List[Int]): TailRec[Boolean] =
  if (xs.isEmpty) done(false) else tailcall(isEven(xs.tail))

// Does this List contain an even number of elements?
isEven((1 to 100000).toList).result

def fib(n: Int): TailRec[Int] =
  if (n < 2) done(n) else for {
    x <- tailcall(fib(n - 1))
    y <- tailcall(fib(n - 2))
  } yield (x + y)

// What is the 40th entry of the Fibonacci series?
fib(40).result
```

: <https://riptutorial.com/ko/scala/topic/3889/>

48:

- `re.findAllIn (s : CharSequence) : MatchIterator`
- `re.findAllMatchIn (s : CharSequence) : []`
- `re.findFirstIn (s : CharSequence) : []`
- `re.findFirstMatchIn (s : CharSequence) : []`
- `re.findPrefixMatchIn (s : CharSequence) : []`
- `re.findPrefixOf (s : CharSequence) : []`
- `re.replaceAllIn (s : CharSequence, Replacer : Match => String) : String`
- `re.replaceAllIn (s : CharSequence, replacement : String) : String`
- `re.replaceFirstIn (s : CharSequence, replacement : String) : String`
- `re.replaceSomeIn (s : CharSequence, Replacer : Match => Option [String]) :`
- `re.split (s : CharSequence) : Array []`

Examples

[scala.collection.immutable.StringOps](#) `r` [scala.util.matching.Regex](#) `.` [Scala](#) `.`

```
val r0: Regex = """"(\d{4})-(\d{2})-(\d{2})""".r // :)
val r1: Regex = """"(\d{4})-(\d{2})-(\d{2})""".r // :(
```

[scala.util.matching.Regex](#) [java.util.regex.Pattern](#) (wrapper) [Scala](#) `API` `.` `x` `.`

```
val dateRegex = """"(?x:
  (\d{4}) # year
  -(\d{2}) # month
  -(\d{2}) # day
)""".r
```

`r`, `def r(names: String*): Regex` `.` `.`

```
""""(\d{4})-(\d{2})-(\d{2})""".r("y", "m", "d").findFirstMatchIn(str) match {
  case Some(matched) =>
    val y = matched.group("y").toInt
    val m = matched.group("m").toInt
    val d = matched.group("d").toInt
    java.time.LocalDate.of(y, m, d)
  case None => ???
}
```

```
val re = """"\((.*?)\)""".r

val str =
"(The) (example) (of) (repeating) (pattern) (in) (a) (single) (string) (I) (had) (some) (trouble) (with) (once) "

re.findAllMatchIn(str).map(_.group(1)).toList
res2: List[String] = List(The, example, of, repeating, pattern, in, a, single, string, I, had,
some, trouble, with, once)
```

: [https://riptutorial.com/ko/scala/topic/2891/-](https://riptutorial.com/ko/scala/topic/2891/)

49:

- (1, _ / * * /) = valueToBeExtracted
- valueToBeExtracted match {case (extractedValue1, _) => ???}
- val (tuple1, tuple2, tuple3) = tupleWith3Elements
- Foo {def unapply (foo : Foo) : [String] = (foo.x); }

Examples

x y .

```
val (x, y) = (1337, 42)
// x: Int = 1337
// y: Int = 42
```

_ : .

```
val (_, y: Int) = (1337, 42)
// y: Int = 42
```

.

```
val myTuple = (1337, 42)
myTuple._1 // res0: Int = 1337
myTuple._2 // res1: Int = 42
```

(22) ._1 ._22 (,).

.

```
val persons = List("A." -> "Lovelace", "G." -> "Hopper")
val names = List("Lovelace, A.", "Hopper, G.")

assert {
  names ==
    (persons map { name =>
      s"${name._2}, ${name._1}"
    })
}

assert {
  names ==
    (persons map { case (given, surname) =>
      s"$surname, $given"
    })
}
```

. .

```
case class Person(name: String, age: Int) // Define the case class
```

```
val p = Person("Paola", 42) // Instantiate a value with the case class type

val Person(n, a) = p // Extract values n and a
// n: String = Paola
// a: Int = 42
```

, n a val . p". :

```
val p2 = Person("Angela", 1337)

val List(Person(n1, a1), Person(_, a2)) = List(p, p2)
// n1: String = Paola
// a1: Int = 42
// a2: Int = 1337
```

```
:
    • " . .
    • . _ .val .
.
```

```
val ls = List(p1, p2, p3) // List of Person objects
ls.map(person => person match {
  case Person(n, a) => println("%s is %d years old".format(n, a))
})
```

person Person n a .

-

unapply Option .

```
class Foo(val x: String)

object Foo {
  def unapply(foo: Foo): Option[String] = Some(foo.x)
}

new Foo("42") match {
  case Foo(x) => x
}
// "42"
```

get isEmpty , unapply Option . Bar unapply Bar .

```
class Bar(val x: String) {
  def get = x
  def isEmpty = false
}

object Bar {
  def unapply(bar: Bar): Bar = bar
}
```

```
new Bar("1337") match {
  case Bar(x) => x
}
// "1337"
```

```
unapply Boolean . Boolean get isEmpty . DivisibleByTwo ( ).
```

```
object DivisibleByTwo {
  def unapply(num: Int): Boolean = num % 2 == 0
}

4 match {
  case DivisibleByTwo() => "yes"
  case _ => "no"
}
// yes

3 match {
  case DivisibleByTwo() => "yes"
  case _ => "no"
}
// no
```

```
unapply unapply . .
```

.

```
case class Pair(a: String, b: String)
val p: Pair = Pair("hello", "world")
val x Pair y = p
//x: String = hello
//y: String = world
```

2-tuple .

```
object Foo {
  def unapply(s: String): Option[(Int, Int)] = Some((s.length, 5))
}
val a Foo b = "hello world!"
//a: Int = 12
//b: Int = 5
```

.

```
scala> val address = """.+:(\d+)""".r
address: scala.util.matching.Regex = .+:(\d+)

scala> val address(host, port) = "some.domain.org:8080"
host: String = some.domain.org
port: String = 8080
```

MatchError .

```
scala> val address(host, port) = "something not a host and port"
scala.MatchError: something not a host and port (of class java.lang.String)
```

Windows

```
object UserPrincipalName {
  def unapply(str: String): Option[(String, String)] = str.split('@') match {
    case Array(u, d) if u.length > 0 && d.length > 0 => Some((u, d))
    case _ => None
  }
}

object DownLevelLogonName {
  def unapply(str: String): Option[(String, String)] = str.split('\\') match {
    case Array(d, u) if u.length > 0 && d.length > 0 => Some((d, u))
    case _ => None
  }
}

def getDomain(str: String): Option[String] = str match {
  case UserPrincipalName(_, domain) => Some(domain)
  case DownLevelLogonName(domain, _) => Some(domain)
  case _ => None
}
```

```
object UserPrincipalName {
  def unapply(obj: Any): Option[(String, String)] = obj match {
    case upn: UserPrincipalName => Some((upn.username, upn.domain))
    case str: String => str.split('@') match {
      case Array(u, d) if u.length > 0 && d.length > 0 => Some((u, d))
      case _ => None
    }
    case _ => None
  }
}
```

tryParse Option .

```
UserPrincipalName.unapply("user@domain") match {
  case Some((u, d)) => ???
  case None => ???
}
```

: <https://riptutorial.com/ko/scala/topic/930/>

50: ()

- (val meters : Double) AnyVal .
- =

.

Examples

```
type Meter = Double
```

Double .

```
type Second = Double
var length: Meter = 3
val duration: Second = 1
length = duration
length = 0d
```

/ () .

```
case class Meter(meters: Double) extends AnyVal
case class Gram(grams: Double) extends AnyVal
```

.

```
var length = Meter(3)
var weight = Gram(4)
//length = weight //type mismatch; found : Gram required: Meter
```

AnyVal JVM (Double).

(: Velocity aka MeterPerSecond) .

-
-
- [ScalaQuantity](#)

() : <https://riptutorial.com/ko/scala/topic/5966/---->

51:

- `aFunction (10) _ // '_'` .
- `nArityFunction.curried // n-arity` .
- `anotherFunction (x) (_ : String) (z) // .` .

Examples

```
def multiply(factor: Int)(numberToBeMultiplied: Int): Int = factor * numberToBeMultiplied

val multiplyBy3 = multiply(3)_ // resulting function signature Int => Int
val multiplyBy10 = multiply(10)_ // resulting function signature Int => Int

val sixFromCurriedCall = multiplyBy3(2) //6
val sixFromFullCall = multiply(3)(2) //6

val fortyFromCurriedCall = multiplyBy10(4) //40
val fortyFromFullCall = multiply(10)(4) //40
```

```
def numberOrCharacterSwitch(toggleNumber: Boolean)(number: Int)(character: Char): String =
  if (toggleNumber) number.toString else character.toString

// need to explicitly specify the type of the parameter to be curried
// resulting function signature Boolean => String
val switchBetween3AndE = numberOrCharacterSwitch(_: Boolean)(3)('E')

switchBetween3AndE(true) // "3"
switchBetween3AndE(false) // "E"
```

```
def minus(left: Int, right: Int) = left - right

val numberMinus5 = minus(_: Int, 5)
val fiveMinusNumber = minus(5, _: Int)

numberMinus5(7) // 2
fiveMinusNumber(7) // -2
```

2 .

```
def add: (Int, Int) => Int = (x,y) => x + y
val three = add(1,2)
```

`add` `Int (` `Int` `An Int )`

```
val addCurried: (Int) => (Int => Int) = add2.curried
//           ^~~ take *one* Int
//           ^~~~ return a *function* from Int to Int
```

```
val add1: Int => Int = addCurried(1)
val three: Int = add1(2)
val allInOneGo: Int = addCurried(1)(2)
```

. currying :

```
def add3: (Int, Int, Int) => Int = (a,b,c) => a + b + c + d
def add3Curr: Int => (Int => (Int => Int)) = add3.curried

val x = add3Curr(1)(2)(42)
```

([Wikipedia](#)) ,

.

, (2)

```
val f: (A, B) => C // a function that takes two arguments of type `A` and `B` respectively
                  // and returns a value of type `C`
```

```
val curriedF: A => B => C // a function that take an argument of type `A`
                        // and returns *a function*
                        // that takes an argument of type `B` and returns a `C`
```

arity-2 .

```
def curry[A, B, C](f: (A, B) => C): A => B => C = {
  (a: A) => (b: B) => f(a, b)
}
```

:

```
val f: (String, Int) => Double = {(_, _) => 1.0}
val curriedF: String => Int => Double = curry(f)
f("a", 1)           // => 1.0
curriedF("a")(1)    // => 1.0
```

Scala .

1.. curriedF :

```
def curriedFasAMethod(str: String)(int: Int): Double = 1.0
val curriedF = curriedFasAMethod _
```

2. (:A => B => C (A, B) => C):Function.uncurried

```
val f: (String, Int) => Double = Function.uncurried(curriedF)
f("a", 1) // => 1.0
```

Currying

currying

1. (1)
2. (2)

1

```
val totalYearlyIncome: (Int, Int) => Int = (income, bonus) => income + bonus
```

2-arity

```
val totalYearlyIncomeCurried: Int => Int => Int = totalYearlyIncome.curried
```

```
Int => (Int => Int)
```

```
val partialTotalYearlyIncome: Int => Int = totalYearlyIncomeCurried(10000)
```

```
partialTotalYearlyIncome(100)
```

2

```
val carManufacturing: (String, String) => String = (wheels, body) => wheels + body
```

```
class CarWheelsFactory {  
  def applyCarWheels(carManufacturing: (String, String) => String): String => String =  
    carManufacturing.curried("applied wheels..  
}  
  
class CarBodyFactory {  
  def applyCarBody(partialCarWithWheels: String => String): String =  
    partialCarWithWheels("applied car body..  
}
```

CarWheelsFactory

```

val carWheelsFactory = new CarWheelsFactory()
val carBodyFactory   = new CarBodyFactory()

val carManufacturing: (String, String) => String = (wheels, body) => wheels + body

val partialCarWheelsApplied: String => String =
  carWheelsFactory.applyCarWheels(carManufacturing)
val carCompleted = carBodyFactory.applyCarBody(partialCarWheelsApplied)

```

Currying .

. .

.

```

case class CreditCard(creditInfo: CreditCardInfo, issuer: Person, account: Account)

object CreditCard {
  def getPremium(totalCards: Int, creditCard: CreditCard): Double = { ... }
}

```

. :

```

val creditCards: List[CreditCard] = getCreditCards()
val allPremiums = creditCards.map(CreditCard.getPremium).sum //type mismatch; found : (Int,
CreditCard) => Double required: CreditCard => ?

```

CreditCard.getPremium . ! . getPremium .

.

```

object CreditCard {
  def getPremium(totalCards: Int)(creditCard: CreditCard): Double = { ... }
}

val creditCards: List[CreditCard] = getCreditCards()

val getPremiumWithTotal = CreditCard.getPremium(creditCards.length)_

val allPremiums = creditCards.map(getPremiumWithTotal).sum

```

: <https://riptutorial.com/ko/scala/topic/1636/>

52:

Examples

```
.  
  
val names = List("Kathryn", "Allie", "Beth", "Serin", "Alana")
```

```
sorted()    math.Ordering    .math.Ordering    .
```

```
names.sorted  
// results in: List(Alana, Allie, Beth, Kathryn, Serin)
```

```
sortWith    .
```

```
names.sortWith(_.length < _.length)  
// results in: List(Beth, Allie, Serin, Alana, Kathryn)
```

```
sortBy    .
```

```
//A set of vowels to use  
val vowels = Set('a', 'e', 'i', 'o', 'u')  
  
//A function that counts the vowels in a name  
def countVowels(name: String) = name.count(l => vowels.contains(l.toLowerCase))  
  
//Sorts by the number of vowels  
names.sortBy(countVowels)  
//result is: List(Kathryn, Beth, Serin, Allie, Alana)
```

```
`reverse :    .
```

```
names.sorted.reverse  
//results in: List(Serin, Kathryn, Beth, Allie, Alana)
```

Java `java.util.Arrays.sort` . `scala.util.Sorting.quickSort`

```
java.util.Arrays.sort(data)  
scala.util.Sorting.quickSort(data)
```

unboxing / boxing . [Scala Collection](#) , `sortWith` `sortBy` .

n

```
x n    fill    . List    fill    .
```

```
// List.fill(n) (x)  
scala > List.fill(3) ("Hello World")
```

```
res0: List[String] = List(Hello World, Hello World, Hello World)
```

Cheatsheet

```
List Vector    .    .Vector List    .
```

```
List[Int]()           // Declares an empty list of type Int
List.empty[Int]        // Uses `empty` method to declare empty list of type Int
Nil                   // A list of type Nothing that explicitly has nothing in it

List(1, 2, 3)          // Declare a list with some elements
1 :: 2 :: 3 :: Nil     // Chaining element prepending to an empty list, in a LISP-style
```

```
List(1, 2, 3).headOption // Some(1)
List(1, 2, 3).head       // 1

List(1, 2, 3).lastOption // Some(3)
List(1, 2, 3).last       // 3, complexity is O(n)

List(1, 2, 3)(1)         // 2, complexity is O(n)
List(1, 2, 3)(3)         // java.lang.IndexOutOfBoundsException: 4
```

```
0 :: List(1, 2, 3)       // List(0, 1, 2, 3)
```

```
List(1, 2, 3) :+ 4       // List(1, 2, 3, 4), complexity is O(n)
```

0

```
List(1, 2) ::: List(3, 4) // List(1, 2, 3, 4)
List.concat(List(1,2), List(3, 4)) // List(1, 2, 3, 4)
List(1, 2) ++ List(3, 4) // List(1, 2, 3, 4)
```

```
List(1, 2, 3).find(_ == 3)           // Some(3)
List(1, 2, 3).map(_ * 2)              // List(2, 4, 6)
List(1, 2, 3).filter(_ % 2 == 1)     // List(1, 3)
List(1, 2, 3).fold(0)((acc, i) => acc + i * i) // 1 * 1 + 2 * 2 + 3 * 3 = 14
List(1, 2, 3).foldLeft("Foo")(_ + _.toString) // "Foo123"
List(1, 2, 3).foldRight("Foo")(_ + _.toString) // "123Foo"
```

Cheatsheet

```
map Map    .
```

```
Map[String, Int]()
val m1: Map[String, Int] = Map()
val m2: String Map Int = Map()
```

```
tuples    .    .
```

```
val l = List(("a", 1), ("b", 2), ("c", 3))
val m = l.toMap // Map(a -> 1, b -> 2, c -> 3)
```

```
val m = Map("a" -> 1, "b" -> 2, "c" -> 3)

m.get("a") // Some(1)
m.get("d") // None
m("a")     // 1
m("d")     // java.util.NoSuchElementException: key not found: d

m.keys     // Set(a, b, c)
m.values   // MapLike(1, 2, 3)
```

```
Map("a" -> 1, "b" -> 2) + ("c" -> 3) // Map(a -> 1, b -> 2, c -> 3)
Map("a" -> 1, "b" -> 2) + ("a" -> 3) // Map(a -> 3, b -> 2)
Map("a" -> 1, "b" -> 2) ++ Map("b" -> 3, "c" -> 4) // Map(a -> 1, b -> 3, c -> 4)
```

(map, find, forEach) tuples. (_1, _2) case .

```
m.find(_._1 == "a") // Some((a,1))
m.map {
  case (key, value) => (value, key)
} // Map(1 -> a, 2 -> b, 3 -> c)
m.filter(_._2 == 2) // Map(b -> 2)
m.foldLeft(0){
  case (acc, (key, value: Int)) => acc + value
} // 6
```

" map . .

```
val someFunction: (A) => (B) = ???
collection.map(someFunction)
```

```
collection.map((x: T) => /*Do something with x*/)
```

2

```
// Initialize
val list = List(1,2,3)
// list: List[Int] = List(1, 2, 3)

// Apply map
list.map((item: Int) => item*2)
// res0: List[Int] = List(2, 4, 6)

// Or in a more concise way
list.map(_*2)
// res1: List[Int] = List(2, 4, 6)
```

```
filter    ".map    Boolean.
```

```
val someFunction: (a) => Boolean = ???  
collection.filter(someFunction)
```

.

```
collection.filter((x: T) => /*Do something that returns a Boolean*/)
```

```
val list = 1 to 10 toList  
// list: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
  
// Filter out all elements that aren't evenly divisible by 2  
list.filter((item: Int) => item % 2==0)  
// res0: List[Int] = List(2, 4, 6, 8, 10)
```

```
case class Person(firstName: String,  
                  lastName: String,  
                  title: String)  
  
// Make a sequence of people  
val people = Seq(  
  Person("Millie", "Fletcher", "Mrs"),  
  Person("Jim", "White", "Mr"),  
  Person("Jenny", "Ball", "Miss") )  
  
// Make labels using map  
val labels = people.map( person =>  
  s"${person.title}. ${person.lastName}"  
)  
  
// Filter the elements beginning with J  
val beginningWithJ = people.filter(_.firstName.startsWith("J"))  
  
// Extract first names and concatenate to a string  
val firstNames = people.map(_.firstName).reduce( (a, b) => a + "," + b )
```

.

. .

```
. . List() ( ) List() , .
```

. List, Sets, Vectors, Seqs Arrays .

```
val numList = List[Int](1, 2, 3, 4, 5)  
numList.reduce((n1, n2) => n1 + n2) // 15  
  
val numSet = Set[Int](1, 2, 3, 4, 5)  
numSet.reduce((n1, n2) => n1 + n2) // 15  
  
val numArray = Array[Int](1, 2, 3, 4, 5)
```

```
numArray.reduce((n1, n2) => n1 + n2) // 15
```

Traversable .

List Vector . Vector List .

Traversable foreach . . .

- **Map** - map , flatMap collect .

```
List(1, 2, 3).map(num => num * 2) // double every number = List(2, 4, 6)
```

```
// split list of letters into individual strings and put them into the same list
List("a b c", "d e").flatMap(letters => letters.split(" ")) // = List("a", "b", "c", "d", "e")
```

- - toList , toArray . 'to' (, 'toList' List).

```
val array: Array[Int] = List[Int](1, 2, 3).toArray // convert list of ints to array of ints
```

- - isEmpty , nonEmpty , size hasDefiniteSize . ().

```
List().isEmpty // true
List(1).nonEmpty // true
```

- - head , last , find Option .

```
val list = List(1, 2, 3)
list.head // = 1
list.last // = 3
```

- - filter , tail , slice , drop .

```
List(-2, -1, 0, 1, 2).filter(num => num > 0) // = List(1, 2)
```

- - partition , splitAt , span groupBy .

```
// split numbers into < 0 and >= 0
List(-2, -1, 0, 1, 2).partition(num => num < 0) // = (List(-2, -1), List(0, 1, 2))
```

- - exists , forall count .

```
List(1, 2, 3, 4).forall(num => num > 0) // = true, all numbers are positive
List(-3, -2, -1, 1).forall(num => num < 0) // = false, not all numbers are negative
```

- **Folds** - foldLeft (/: foldLeft , foldRight (:\\), reduceLeft reduceRight . .

fold .

```
val nums = List(1,2,3,4,5)
```

```
var initialValue: Int = 0;
var sum = nums.fold(initialValue){
  (accumulator, currentElementBeingIterated) => accumulator + currentElementBeingIterated
}
println(sum) //prints 15 because 0+1+2+3+4+5 = 15
```

fold() . . . :

```
def sum(x: Int, y: Int) = x + y
val nums = List(1, 2, 3, 4, 5)
var initialValue: Int = 0
val sum = nums.fold(initialValue)(sum)
println(sum) // prints 15 because 0 + 1 + 2 + 3 + 4 + 5 = 15
```

.

```
initialValue = 2;
sum = nums.fold(initialValue){
  (accumulator, currentElementBeingIterated) => accumulator + currentElementBeingIterated
}
println(sum) //prints 17 because 2+1+2+3+4+5 = 17
```

fold foldLeft foldRight .

foldLeft() ().foldRight() ().fold() foldLeft() . fold() foldLeft() .

```
def fold[A1 >: A](z: A1)(op: (A1, A1) => A1): A1 = foldLeft(z)(op)
```

fold() , foldLeft() foldRight() ., foldLeft() foldRight() fold() .

fold() foldLeft() foldRight() . .

foldLeft() foldLeft() foldRight() .foldRight() .

foreach . . :

```
scala> val x = List(1,2,3)
x: List[Int] = List(1, 2, 3)

scala> x.foreach { println }
1
2
3
```

foreach . foreach . map, filter, for comprehension .

```
def myFunc(a: Int) : Int = a * 2
List(1,2,3).foreach(myFunc) // Returns nothing
```

reduce() , reduceLeft() reduceRight .reduce

```
val nums = List(1,2,3,4,5)
```

```

sum = nums.reduce({ (a, b) => a + b })
println(sum) //prints 15

val names = List("John", "Koby", "Josh", "Matilda", "Zac", "Mary Poppins")

def findLongest(nameA:String, nameB:String):String = {
  if (nameA.length > nameB.length) nameA else nameB
}

def findLastAlphabetically(nameA:String, nameB:String):String = {
  if (nameA > nameB) nameA else nameB
}

val longestName:String = names.reduce(findLongest(_,_))
println(longestName) //prints Mary Poppins

//You can also omit the arguments if you want
val lastAlphabetically:String = names.reduce(findLastAlphabetically)
println(lastAlphabetically) //prints Zac

```

.:

1. reduce .
2. .
3. .

: <https://riptutorial.com/ko/scala/topic/686/>

53:

- `class MyClass{} // curly braces are optional here as class body is empty`
- `class MyClassWithMethod {def method: MyClass = ???}`
- `new MyClass() //Instantiate`
- `object MyObject // Singleton object`
- `class MyClassWithGenericParameters[V1, V2](v1: V1, i: Int, v2: V2)`
- `class MyClassWithImplicitFieldCreation[V1](val v1: V1, val i: Int)`
- `new MyClassWithGenericParameters(2.3, 4, 5)    new MyClassWithGenericParameters[Double, Any](2.3, 4, 5)`
- `class MyClassWithProtectedConstructor protected[my.pack.age](s: String)`

Examples

" . . .

```
class MyClass{} // curly braces are optional here as class body is empty
```

new .

```
var instance = new MyClass()
```

:

```
var instance = new MyClass
```

Scala . :

```
class MyClass(arg : Int) // Class definition
var instance = new MyClass(2) // Instance instantiation
instance.arg // not allowed
```

MyClass Int . arg MyClass .

```
class MyClass(arg : Int){
  val prop = arg // Class field declaration
}
```

```
var obj = new MyClass(2)
obj.prop // legal statement
```

public .

```
class MyClass(val arg : Int) // Class definition with arg declared public
var instance = new MyClass(2) // Instance instantiation
instance.arg //arg is now visible to clients
```

: {} vs ()

MyClass .

```
class MyClass
```

.

```
val obj = new MyClass()
```

.

```
val obj = new MyClass
```

. . .

```
val newThread = new Thread { new Runnable {  
    override def run(): Unit = {  
        // perform task  
        println("Performing task.")  
    }  
}  
}  
newThread.start // prints no output
```

Performing task. Performing task.,, . . new Thread {} . Thread :

```
val newThread = new Thread {  
    //creating anonymous class extending Thread  
}
```

(Runnable). public Thread(Runnable target) (()) public Thread() run() . .

```
val newThread = new Thread ( new Runnable {  
    override def run(): Unit = {  
        // perform task  
        println("Performing task.")  
    }  
}  
)
```

, {} () .

Scala Java . Scala *Singleton Objects* . new . .

```
object Factorial {  
    private val cache = Map[Int, Int]()  
    def getCache = cache  
}
```

object 'class' 'trait' . . .

```
Factorial.getCache() //returns the cache
```

Java

```
. . . , Factorial , Factorial . . .
```

```
class Factorial(num : Int) {  
  
    def fact(num : Int) : Int = if (num <= 1) 1 else (num * fact(num - 1))  
  
    def calculate() : Int = {  
        if (!Factorial.cache.contains(num)) { // num does not exists in cache  
            val output = fact(num) // calculate factorial  
            Factorial.cache += (num -> output) // add new value in cache  
        }  
  
        Factorial.cache(num)  
    }  
}  
  
object Factorial {  
    private val cache = scala.collection.mutable.Map[Int, Int]()  
}  
  
val factfive = new Factorial(5)  
factfive.calculate // Calculates the factorial of 5 and stores it  
factfive.calculate // uses cache this time  
val factfiveagain = new Factorial(5)  
factfiveagain.calculate // Also uses cache
```

cache .

```
object Factorial class Factorial . private . Factorial private cache .
```

.

(,).

```
object Dog {  
    def bark: String = "Raf"  
}  
  
Dog.bark() // yields "Raf"
```

, .

```
class Dog(val name: String) {  
  
}  
  
object Dog {  
    def apply(name: String): Dog = new Dog(name)  
}  
  
val dog = Dog("Barky") // Object  
val dog = new Dog("Barky") // Class
```

```
: variable.isInstanceOf[Type]
```

```
( ):
```

```
variable match {  
  case _: Type => true  
  case _ => false  
}
```

```
isInstanceOf ( ) .
```

```
val list: List[Any] = List(1, 2, 3)           //> list : List[Any] = List(1, 2, 3)  
  
val upcasting = list.isInstanceOf[Seq[Int]]     //> upcasting : Boolean = true  
  
val shouldBeFalse = list.isInstanceOf[List[String]]  
                                     //> shouldBeFalse : Boolean = true
```

```
val chSeqArray: Array[CharSequence] = Array("a") //> chSeqArray : Array[CharSequence] =  
Array(a)  
val correctlyReified = chSeqArray.isInstanceOf[Array[String]]  
                                     //> correctlyReified : Boolean = false  
  
val stringIsACharSequence: CharSequence = ""    //> stringIsACharSequence : CharSequence = ""  
  
val sArray = Array("a")                  //> sArray : Array[String] = Array(a)  
val correctlyReified = sArray.isInstanceOf[Array[String]]  
                                     //> correctlyReified : Boolean = true  
  
//val arraysAreInvariantInScala: Array[CharSequence] = sArray  
//Error: type mismatch; found   : Array[String] required: Array[CharSequence]  
//Note: String <: CharSequence, but class Array is invariant in type T.  
//You may wish to investigate a wildcard type such as `_ <: CharSequence`. (SLS 3.2.10)  
//Workaround:  
val arraysAreInvariantInScala: Array[_ <: CharSequence] = sArray  
                                     //> arraysAreInvariantInScala : Array[_ <:  
CharSequence] = Array(a)  
  
val arraysAreCovariantOnJVM = sArray.isInstanceOf[Array[CharSequence]]  
                                     //> arraysAreCovariantOnJVM : Boolean = true
```

```
: variable.asInstanceOf[Type]
```

```
:
```

```
variable match {  
  case _: Type => true  
}
```

```
:
```

```
val x = 3                                     //> x : Int = 3  
x match {
```

```

    case _: Int => true//better: do something
    case _ => false
  }
                                     //> res0: Boolean = true

x match {
  case _: java.lang.Integer => true//better: do something
  case _ => false
}
                                     //> res1: Boolean = true

x.isInstanceOf[Int]
                                     //> res2: Boolean = true

//x.isInstanceOf[java.lang.Integer]//fruitless type test: a value of type Int cannot also be
a Integer

trait Valuable { def value: Int}
case class V(val value: Int) extends Valuable

val y: Valuable = V(3)
y.isInstanceOf[V]
y.asInstanceOf[V]
                                     //> y  : Valuable = V(3)
                                     //> res3: Boolean = true
                                     //> res4: V = V(3)

```

: JVM . (JS,) / .

. .(.)

```

class Foo(x: Int, y: String) {
  val xy: String = y * x
  /* now xy is a public member of the class */
}

class Bar {
  ...
}

```

val .

```

class Baz(val z: String)
// Baz has no other members or methods, so the body may be omitted

val foo = new Foo(4, "ab")
val baz = new Baz("I am a baz")
foo.x // will not compile: x is not a member of Foo
foo.xy // returns "abababab": xy is a member of Foo
baz.z // returns "I am a baz": z is a member of Baz
val bar0 = new Bar
val bar1 = new Bar() // Constructor parentheses are optional here

```

.

```

class DatabaseConnection
  (host: String, port: Int, username: String, password: String) {
  /* first connect to the DB, or throw an exception */
  private val driver = new AwesomeDB.Driver()
  driver.connect(host, port, username, password)
  def isConnected: Boolean = driver.isConnected
  ...
}

```

. IO connect disconnect .

'' . def this(...) = e . e .

```
class Person(val fullName: String) {  
  def this(firstName: String, lastName: String) = this(s"$firstName $lastName")  
}
```

```
// usage:  
new Person("Grace Hopper").fullName // returns Grace Hopper  
new Person("Grace", "Hopper").fullName // returns Grace Hopper
```

. .

```
class Person private(val fullName: String) {  
  def this(firstName: String, lastName: String) = this(s"$firstName $lastName")  
}
```

```
new Person("Ada Lovelace") // won't compile  
new Person("Ada", "Lovelace") // compiles
```

.

: <https://riptutorial.com/ko/scala/topic/2047/->

54:

23 ?

. Tuple1 Tuple22 . . .

0 ?

Tuple0 Unit .

Examples

2 22 . . 2 ('pair') .

```
scala> val x = (1, "hello")
x: (Int, String) = (1,hello)
scala> val y = 2 -> "world"
y: (Int, String) = (2,world)
scala> val z = 3 → "foo" //example of using U+2192 RIGHTWARD ARROW
z: (Int, String) = (3,foo)
```

x 2 . . _1 , _22 . , x._1 x . x._2 . , .

2 (key -> value) Maps .

```
scala> val m = Map[Int, String](2 -> "world")
m: scala.collection.immutable.Map[Int,String] = Map(2 -> world)

scala> m + x
res0: scala.collection.immutable.Map[Int,String] = Map(2 -> world, 1 -> hello)

scala> (m + x).toList
res1: List[(Int, String)] = List((2,world), (1,hello))
```

, 1 a .

. .

```
scala> val l = List(1 -> 2, 2 -> 3, 3 -> 4)
l: List[(Int, Int)] = List((1,2), (2,3), (3,4))
```

- .

```
scala> l.map((e1: Int, e2: Int) => e1 + e2)
```

.

```
<console>:9: error: type mismatch;
 found   : (Int, Int) => Int
```

```
required: ((Int, Int)) => ?  
      l.map((e1: Int, e2: Int) => e1 + e2)
```

```
.    .    _1 _2 .
```

```
scala> l.map(e => e._1 + e._2)  
res1: List[Int] = List(3, 5, 7)
```

```
case    .
```

```
scala> l.map{ case (e1: Int, e2: Int) => e1 + e2}  
res2: List[Int] = List(3, 5, 7)
```

```
.
```

: <https://riptutorial.com/ko/scala/topic/4971/>

55:

- `@AnAnnotation def someMethod = {...}`
- `@AnAnnotation class someClass {...}`
- `@AnnotatioWithArgs (annotation_args) def someMethod = {...}`

@	.
SomeAnnotation	
constructor_args	() . .

Scala-lang [Java](#) .

Examples

[deprecated](#) .

```
@deprecated
def anUnusedLegacyMethod(someArg: Any) = {
  ...
}
```

```
@deprecated def anUnusedLegacyMethod(someArg: Any) = {
  ...
}
```

```
/**
 * @param num Numerator
 * @param denom Denominator
 * @throws ArithmeticException in case `denom` is `0`
 */
class Division @throws[ArithmeticException](/*no annotation parameters*/) protected (num: Int,
denom: Int) {
  private[this] val wrongValue = num / denom

  /** Integer number
   * @param num Value */
  protected[Division] def this(num: Int) {
    this(num, 1)
  }
}

object Division {
  def apply(num: Int) = new Division(num)
  def apply(num: Int, denom: Int) = new Division(num, denom)
}
```

```
( protected ) . ( @throws () () :) .

: ( ) .

( ) :
```

```
case class Division @throws[ArithmeticException]("denom is 0") (num: Int, denom: Int) {
  private[this] val wrongValue = num / denom
}
```

scala.annotation.StaticAnnotation scala.annotation.ClassfileAnnotation Scala .

```
package animals
// Create Annotation `Mammal`
class Mammal(indigenous:String) extends scala.annotation.StaticAnnotation

// Annotate class Platypus as a `Mammal`
@Mammal(indigenous = "North America")
class Platypus{}
```

API .

```
scala>import scala.reflect.runtime.{universe => u}

scala>val platypusType = u.typeOf[Platypus]
platypusType: reflect.runtime.universe.Type = animals.reflection.Platypus

scala>val platypusSymbol = platypusType.typeSymbol.asClass
platypusSymbol: reflect.runtime.universe.ClassSymbol = class Platypus

scala>platypusSymbol.annotations
List[reflect.runtime.universe.Annotation] = List(animals.reflection.Mammal("North America"))
```

: <https://riptutorial.com/ko/scala/topic/3783/>-

56:

ParseResult Cases

ParseResult ParseResult .

- Match Success.
- , .
- , .

Examples

```
import scala.util.parsing.combinator._

class SimpleParser extends RegexParsers {
  // Define a grammar rule, turn it into a regex, and apply it the input.
  def word: Parser[String] = """[A-Z][a-z]+""".r ^^ { _.toString }
}

object SimpleParser extends SimpleParser {
  val parseAlice = parse(word, "Alice went to Alamo Square.")
  val parseBarb = parse(word, "barb went Upside Down.")
}

//Successfully finds a match
println(SimpleParser.parseAlice)
//Fails to find a match
println(SimpleParser.parseBarb)
```

```
[1.6] parsed: Alice
res0: Unit = ()

[1.1] failure: string matching regex `[A-Z][a-z]+' expected but `b' found

barb went Upside Down.
^
```

Alice [1.6] 1 6 .

: [https://riptutorial.com/ko/scala/topic/3730/-](https://riptutorial.com/ko/scala/topic/3730/)

```
.., com.sql org.http connection . com.sql.connection org.http.connection .
```

Examples

```
package com {
  package utility {
    package serialization {
      class Serializer
      ...
    }
  }
}
```

```
package . . , math1.scala math2.scala .
```

math1.scala

```
package org {
  package math {
    package statistics {
      class Interval
    }
  }
}
```

math2.scala

```
package org {
  package math{
    package probability {
      class Density
    }
  }
}
```

.

```
import org.math.probability.Density
import org.math.statistics.Interval

object Study {

  def main(args: Array[String]): Unit = {
    var a = new Interval()
    var b = new Density()
  }
}
```

Java .

. (:

```
io.super.math
```

: <https://riptutorial.com/ko/scala/topic/8231/>

58:

- `partialFunction`
- `( ) //`

	.
case	.

Examples

.

```
def f(x: Int): String = x match {  
  case 1 => "One"  
  case 2 => "Two"  
  case _ => "Unknown!"  
}  
  
f(2) // "Two"  
f(3) // "Unknown!"
```

: _ () .

```
def g(x: Int): String = x match {  
  case 1 => "One"  
  case 2 => "Two"  
}  
  
g(1) // "One"  
g(3) // throws a MatchError
```

(case _ => <do something>)

. **backticks** .

One two :

```
val One: Int = 1  
val two: Int = 2
```

.

```
def g(x: Int): String = x match {  
  case One => "One"  
  case `two` => "Two"  
}
```

Java . case . .

```
def f(x: Int): String = x match {
  case _ => "Default"
  case 1 => "One"
}

f(5) // "Default"
f(1) // "Default"
```

. . .

. . . .

```
def tabulate(char: Char, tab: List[(Char, Int)]): List[(Char, Int)] = tab match {
  case Nil => List((char, 1))
  case (`char`, count) :: tail => (char, count + 1) :: tail
  case head :: tail => head :: tabulate(char, tail)
}
```

char " . ., tabulate('x', ...) case .

```
case('x', count) => ...
```

. . .

Seq

```
def f(ints: Seq[Int]): String = ints match {
  case Seq() =>
    "The Seq is empty !"
  case Seq(first) =>
    s"The seq has exactly one element : $first"
  case Seq(first, second) =>
    s"The seq has exactly two elements : $first, $second"
  case s @ Seq(_, _, _) =>
    s"s is a Seq of length three and looks like ${s}" // Note individual elements are not
    bound to their own names.
  case s: Seq[Int] if s.length == 4 =>
    s"s is a Seq of Ints of exactly length 4" // Again, individual elements are not bound
    to their own names.
  case _ =>
    "No match was found!"
}
```

:

```
def f(ints: Seq[Int]): String = ints match {
  case Seq(first, second, tail @ _*) =>
    s"The seq has at least two elements : $first, $second. The rest of the Seq is $tail"
  case Seq(first, tail @ _*) =>
    s"The seq has at least one element : $first. The rest of the Seq is $tail"
  // alternative syntax
  // here of course this one will never match since it checks
```

```
// for the same thing as the one above
case first +: tail =>
  s"The seq has at least one element : $first. The rest of the Seq is $tail"
case _ =>
  "The seq didn't match any of the above, so it must be empty"
}
```

.

```
Nil :: List . Seq( ...) +: Seq( ...) .
```

```
:: WrappedArray, Vector . :
```

```
scala> def f(ints:Seq[Int]) = ints match {
  | case h :: t => h
  | case _ => "No match"
  | }
f: (ints: Seq[Int])Any

scala> f(Array(1,2))
res0: Any = No match
```

+:

```
scala> def g(ints:Seq[Int]) = ints match {
  | case h+:t => h
  | case _ => "No match"
  | }
g: (ints: Seq[Int])Any

scala> g(Array(1,2).toSeq)
res4: Any = 1
```

()

case if .

```
def checkSign(x: Int): String = {
  x match {
    case a if a < 0 => s"$a is a negative number"
    case b if b > 0 => s"$b is a positive number"
    case c => s"$c neither positive nor negative"
  }
}
```

():

```
def f(x: Option[Int]) = x match {
  case Some(i) if i % 2 == 0 => doSomething(i)
  case None => doSomethingIfNone
}
```

MatchError . / .

```
def f(x: Option[Int]) = x match {
  case Some(i) if i % 2 == 0 => doSomething(i)
  case _ => doSomethingIfNoneOrOdd
}
```

```
case class Student(name: String, email: String)

def matchStudent1(student: Student): String = student match {
  case Student(name, email) => s"$name has the following email: $email" // extract name and email
}
```

```
def matchStudent2(student: Student): String = student match {
  case Student("Paul", _) => "Matched Paul" // Only match students named Paul, ignore email
  case Student(name, _) if name == "Paul" => "Matched Paul" // Use a guard to match students named Paul, ignore email
  case s if s.name == "Paul" => "Matched Paul" // Don't use extractor; use a guard to match students named Paul, ignore email
  case Student("Joe", email) => s"Joe has email $email" // Match students named Joe, capture their email
  case Student(name, email) if name == "Joe" => s"Joe has email $email" // use a guard to match students named Joe, capture their email
  case Student(name, email) => s"$name has email $email." // Match all students, capture name and email
}
```

Option :

```
def f(x: Option[Int]) = x match {
  case Some(i) => doSomething(i)
  case None => doSomethingIfNone
}
```

fold, map / getOrElse :

```
def g(x: Option[Int]) = x.fold(doSomethingIfNone)(doSomething)
def h(x: Option[Int]) = x.map(doSomething).getOrElse(doSomethingIfNone)
```

, '' :

```
sealed trait Shape
case class Square(height: Int, width: Int) extends Shape
case class Circle(radius: Int) extends Shape
case object Point extends Shape

def matchShape(shape: Shape): String = shape match {
  case Square(height, width) => "It's a square"
  case Circle(radius) => "It's a circle"
  //no case for Point because it would cause a compiler warning.
}
```

```
}
```

Shape case class Shape match throw. . .

```
val emailRegex: Regex = "(.+)?@(.+)?\\.?(.+)"

"name@example.com" match {
  case emailRegex(userName, domain, topDomain) => println(s"Hi $userName from $domain")
  case _ => println(s"This is not a valid email.")
}
```

. userName domain .topDomain .String str .r new Regex(str) .r .

(@)

@ . .

```
sealed trait Shape
case class Rectangle(height: Int, width: Int) extends Shape
case class Circle(radius: Int) extends Shape
case object Point extends Shape

(Circle(5): Shape) match {
  case Rectangle(h, w) => s"rectangle, $h x $w."
  case Circle(r) if r > 9 => s"large circle"
  case c @ Circle(_) => s"small circle: ${c.radius}" // Whole matched object is bound to c
  case Point => "point"
}
```

> res0: String = small circle: 5

..:

```
case Circle(r) if r > 9 => s"large circle"
```

.

```
case c @ Circle(_) if c.radius > 9 => s"large circle"
```

.

```
Seq(Some(1), Some(2), None) match {
  // Only the first element of the matched sequence is bound to the name 'c'
  case Seq(c @ Some(1), _) => head
  case _ => None
}
```

> res0: Option[Int] = Some(1)

asInstanceOf[B] .

```
val anyRef: AnyRef = ""
```

```
anyRef match {
  case _: Number      => "It is a number"
  case _: String       => "It is a string"
  case _: CharSequence => "It is a char sequence"
}
//> res0: String = It is a string
```

```
anyRef match {
  case _: Number      => "It is a number"
  case _: CharSequence => "It is a char sequence"
  case _: String       => "It is a string"
}
//> res1: String = It is a char sequence
```

" . " . :

```
case class Foo(s: String)
case class Bar(s: String)
case class Woo(s: String, i: Int)

def matcher(g: Any):String = {
  g match {
    case Bar(s) => s + " is classy!"
    case Foo(_) => "Someone is wicked smart!"
    case Woo(s, _) => s + " is adventuresome!"
    case _ => "What are we talking about?"
  }
}

print(matcher(Foo("Diana"))) // prints 'Diana is classy!'
print(matcher(Bar("Hadas"))) // prints 'Someone is wicked smart!'
print(matcher(Woo("Beth", 27))) // prints 'Beth is adventuresome!'
print(matcher(Option("Katie"))) // prints 'What are we talking about?'
```

Foo Woo (_) . (Hadas 27) . " .

tableswitch lookupswitch

@switch match tableswitch . .

@switch final val . tableswitch / lookupswitch .

```
import annotation.switch

def suffix(i: Int) = (i: @switch) match {
  case 1 => "st"
  case 2 => "nd"
  case 3 => "rd"
  case _ => "th"
}
```

```
scala> suffix(2)
res1: String = "2nd"

scala> suffix(4)
res2: String = "4th"
```

(2.8+) - @switch:

.

Java :

- `tableswitch` : " "
- `lookupswitch` : " "

| `case` .

```
def f(str: String): String = str match {
  case "foo" | "bar" => "Matched!"
  case _ => "No match."
}

f("foo") // res0: String = Matched!
f("bar") // res1: String = Matched!
f("fubar") // res2: String = No match.
```

.

```
sealed class FooBar
case class Foo(s: String) extends FooBar
case class Bar(s: String) extends FooBar

val d = Foo("Diana")
val h = Bar("Hadas")

// This matcher WILL NOT work.
def matcher(g: FooBar):String = {
  g match {
    case Foo(s) | Bar(s) => print(s) // Won't work: s cannot be resolved
    case Foo(_) | Bar(_) => _ // Won't work: _ is an unbound placeholder
    case _ => "Could not match"
  }
}
```

(_) .

```
def matcher(g: FooBar):String = {
  g match {
    case Foo(_) | Bar(_) => "Is either Foo or Bar." // Works fine
    case _ => "Could not match"
  }
}
```

, :

```
def matcher(g: FooBar):String = {
  g match {
    case Foo(s) => s
    case Bar(s) => s
    case _ => "Could not match"
  }
}
```

List :

```
val pastries = List(("Chocolate Cupcake", 2.50),
                    ("Vanilla Cupcake", 2.25),
                    ("Plain Muffin", 3.25))
```

```
pastries foreach { pastry =>
  pastry match {
    case ("Plain Muffin", price) => println(s"Buying muffin for $price")
    case p if p._1 contains "Cupcake" => println(s"Buying cupcake for ${p._2}")
    case _ => println("We don't sell that pastry")
  }
}
```

. if **tuple** .

: [https://riptutorial.com/ko/scala/topic/661/-](https://riptutorial.com/ko/scala/topic/661/)

59:

Examples

If

(), if . .

```
if(x < 1984) {  
    println("Good times")  
} else if(x == 1984) {  
    println("The Orwellian Future begins")  
} else {  
    println("Poor guy...")  
}
```

if evaluation :

```
val result = if(x > 0) "Greater than 0" else "Less than or equals 0"  
\ \ result: String = Greater than 0
```

if result .

if ., String . else if if Any .

```
val result = if(x > 0) "Greater than 0"  
// result: Any = Greater than 0
```

println Unit :

```
val result = if(x > 0) println("Greater than 0")  
// result: Unit = ()
```

Java if Scala . .

if .

: <https://riptutorial.com/ko/scala/topic/4171/>

60:

- {}
- {clauses}
- ()
- ()

...	for /
	for .
	".yield for Unit .
	for .

Examples

For

```
for (x <- 1 to 10)
  println("Iteration number " + x)
```

```
x 1 10      . for Unit .
```

for-loop 'sequence comprehension' yield .

```
for ( x <- 1 to 10 if x % 2 == 0)
  yield x
```

.

```
scala.collection.immutable.IndexedSeq[Int] = Vector(2, 4, 6, 8, 10)
```

A .

Nested For Loop

.

```
for {
  x <- 1 to 2
  y <- 'a' to 'd'
} println("(" + x + "," + y + ")")
```

(to . .)

.

```
(1,a)
(1,b)
(1,c)
(1,d)
(2,a)
(2,b)
(2,c)
(2,d)
```

.

```
for (
  x <- 1 to 2
  y <- 'a' to 'd'
) println("(" + x + "," + y + ")")
```

yield val .

```
val a = for {
  x <- 1 to 2
  y <- 'a' to 'd'
} yield "(%s,%s)".format(x, y)
// a: scala.collection.immutable.IndexedSeq[String] = Vector((1,a), (1,b), (1,c), (1,d),
(2,a), (2,b), (2,c), (2,d))
```

'for comprehension' .

```
for {
  x <- Option(1)
  y <- Option("b")
  z <- List(3, 4)
} {
  // Now we can use the x, y, z variables
  println(x, y, z)
  x // the last expression is *not* the output of the block in this case!
}

// This prints
// (1, "b", 3)
// (1, "b", 4)
```

Unit .

M(:Option) yield yield Unit M .

```
val a = for {
  x <- Option(1)
  y <- Option("b")
} yield {
  // Now we can use the x, y variables
  println(x, y)
  // whatever is at the end of the block is the output
  (7 * x, y)
}
```

```

}

// This prints:
// (1, "b")
// The val `a` is set:
// a: Option[(Int, String)] = Some((7,b))

```

yield (Option List) . .

For .

.

```

val map = Map(1 -> "a", 2 -> "b")
for (number <- map) println(number) // prints (1,a), (2,b)
for ((key, value) <- map) println(value) // prints a, b

```

.

```

val list = List(1,2,3)
for(number <- list) println(number) // prints 1, 2, 3

```

Desugaring

for . withFilter withFilter , foreach , flatMap map . for .

A for pN , gN cN for .

```

for(p0 <- x0 if g0; p1 <- g1 if c1) { ??? }

```

...withFilter foreach **de-sugar.**

```

g0.withFilter({ case p0 => c0 case _ => false }).foreach({
  case p0 => g1.withFilter({ case p1 => c1 case _ => false }).foreach({
    case p1 => ???
  })
})

```

for / yield .

```

for(p0 <- g0 if c0; p1 <- g1 if c1) yield ???

```

...withFilter flatMap map flatMap map .

```

g0.withFilter({ case p0 => c0 case _ => false }).flatMap({
  case p0 => g1.withFilter({ case p1 => c1 case _ => false }).map({
    case p1 => ???
  })
})

```

```
(map flatMap .)
```

```
for {
```

```
: https://riptutorial.com/ko/scala/topic/669/-
```

61:

Examples

Apache Spark UDF

```
import org.apache.spark.sql.functions._

// Create a function that uses the content of the column inside the dataframe
val code = (param: String) => if (param == "myCode") 1 else 0
// With that function, create the udf function
val myUDF = udf(code)
// Apply the udf to a column inside the existing dataframe, creating a dataframe with the
additional new column
val newDataframe = aDataframe.withColumn("new_column_name", myUDF(col(inputColumn)))
```

: <https://riptutorial.com/ko/scala/topic/8241/---->

62:

- ATrait {...}
- AClass (...) ATrait {...} .
- class AClass ATclass BClass .
- AClass ATrait BTrait .
- class AClass CTrait BTrait ATrait .
- ATrait BTrait .

Examples

.

. . . .

```
class Ball {
  def roll(ball : String) = println("Rolling : " + ball)
}

trait Red extends Ball {
  override def roll(ball : String) = super.roll("Red-" + ball)
}

trait Green extends Ball {
  override def roll(ball : String) = super.roll("Green-" + ball)
}

trait Shiny extends Ball {
  override def roll(ball : String) = super.roll("Shiny-" + ball)
}

object Balls {
  def main(args: Array[String]) {
    val ball1 = new Ball with Shiny with Red
    ball1.roll("Ball-1") // Rolling : Shiny-Red-Ball-1

    val ball2 = new Ball with Green with Shiny
    ball2.roll("Ball-2") // Rolling : Green-Shiny-Ball-2
  }
}
```

super roll() . . , .

.

```
trait Identifiable {
  def getIdentifier: String
  def printIndentification(): Unit = println(getIdentifier)
}

case class Puppy(id: String, name: String) extends Identifiable {
  def getIdentifier: String = s"$name has id $id"
}
```

```
Identifiable      AnyRef . Identifiable getIdentifier    Puppy . Puppy Identifiable
printIdentification .
```

REPL :

```
val p = new Puppy("K9", "Rex")
p.getIdentifier // res0: String = Rex has id K9
p.printIdentification() // Rex has id K9
```

Java Traits Scala . [mixin](#) .

```
trait traitA {
  def name = println("This is the 'grandparent' trait.")
}

trait traitB extends traitA {
  override def name = {
    println("B is a child of A.")
    super.name
  }
}

trait traitC extends traitA {
  override def name = {
    println("C is a child of A.")
    super.name
  }
}

object grandChild extends traitB with traitC

grandChild.name
```

```
grandChild traitB traitC traitA.traitA traitA. () :
```

```
C is a child of A.
B is a child of A.
This is the 'grandparent' trait.
```

```
super class trait . grandChild .
```

grandChild -> traitC -> traitB -> traitA -> AnyRef -> Any

```
trait Printer {
  def print(msg : String) = println (msg)
}

trait DelimitWithHyphen extends Printer {
  override def print(msg : String) {
```

```

    println("-----")
    super.print(msg)
  }
}

trait DelimitWithStar extends Printer {
  override def print(msg : String) {
    println("*****")
    super.print(msg)
  }
}

class CustomPrinter extends Printer with DelimitWithHyphen with DelimitWithStar

object TestPrinter{
  def main(args: Array[String]) {
    new CustomPrinter().print("Hello World!")
  }
}

```

```

*****
-----
Hello World!

```

CustomPrinter .

CustomPrinter -> DelimitWithStar -> DelimitWithHyphen -> Printer -> AnyRef -> Any

. . super() . .

```

class Shape {
  def paint (shape: String): Unit = {
    println(shape)
  }
}

trait Color extends Shape {
  abstract override def paint (shape : String) {
    super.paint(shape + "Color ")
  }
}

trait Blue extends Color {
  abstract override def paint (shape : String) {
    super.paint(shape + "with Blue ")
  }
}

trait Border extends Shape {
  abstract override def paint (shape : String) {
    super.paint(shape + "Border ")
  }
}

trait Dotted extends Border {
  abstract override def paint (shape : String) {

```

```

    super.paint(shape + "with Dotted ")
  }
}

class MyShape extends Shape with Dotted with Blue {
  override def paint (shape : String) {
    super.paint(shape)
  }
}

```

• ,

1. First Shape .

Shape -> AnyRef -> Any

2. Dotted .

Dotted -> Border -> Shape -> AnyRef -> Any

3. Blue . Blue .

Blue -> Color -> Shape -> AnyRef -> Any

MyShape (**Step 2**) Shape -> AnyRef -> Any Shape -> AnyRef -> Any . . Blue .

Blue -> Color -> Dotted -> Border -> Shape -> AnyRef -> Any

4. Circle .

-> -> -> -> -> -> AnyRef -> Any

super . . new MyShape().paint("Circle ") .

Circle with Blue Color with Dotted Border

•

: <https://riptutorial.com/ko/scala/topic/1056/>

S. No		Contributors
1		4444 , Andy Hayden , Ani Menon , Community , David G. , David Portabella , dk14 , Donald.McLean , Gabriele Petronella , Grzegorz Oledzki , implicitdef , isaias-b , J Atkin , Jean , Jonathan , mammothbane , marcospereira , Marek Skiba , mdarwin , Nathaniel Ford , NeoWelkin , Nicofisi , Priya , rolve , Shoe , sschaef , Thomas Andrews , Tyler James Harden , Ven , Vogon Jeltz
2	Implicits	Andy Hayden , dimitrisli , Gábor Bakos , HTNW , implicitdef , ipoteka , Jose Antonio Jimenez Saez , Michael Zajac , Nathaniel Ford , nattyddubbs , Simon , spiffman , Suma , Timo , vsminkov
3	Java	Andrzej Jozwik , Dan Hulme , Gábor Bakos , mvn , the21st , thekingofkings
4	JSON	ipoteka , John , Muki , Nathaniel Ford , pedrorijo91 , suj1th , void , Wogan , zoitol
5	Quasiquotes	gregghz
6	Scala.js	Camilo Sampedro
7	ScalaCheck	Andrzej Jozwik
8	Scaladoc	Camilo Sampedro , Gábor Bakos , Nathaniel Ford
9	ScalaTest	Nadim Bahadoor , Nathaniel Ford
10	While	J Cracknell , Nathaniel Ford
11	XML	Nathaniel Ford , Rockie Yang , vsny
12		dmitry , HTNW
13		acjay , ches , Nathaniel Ford , nukie , Rajat Jain , Srin
14		Aravindh S , Arche , Camilo Sampedro , ches , corvus_192 , Dawny33 , Gábor Bakos , Gabriele Petronella , implicitdef , ipoteka , Jean , jwvh , michael_s , Nathaniel Ford , raam86 , rjsvaljean , ScientiaEtVeritas , Shastick , stefanobaghino , Sven Koschnicke , vise890 , wheaties
15		ZbyszekKr
16	(SAM)	Gábor Bakos , Gabriele Petronella , Nathaniel Ford

17		Gábor Bakos
18		HTNW
19		gregghz, HTNW, Nathaniel Ford
20		ipoteka, Nathaniel Ford
21		corvus_192, ipoteka, Nathaniel Ford, RamenChef, Sarvesh Kumar Singh, Shuklaswag
22		Andy Hayden, Ayberk, Brian, implicitdef, J Cracknell, Nadim Bahadoor
23		Sachin Janani
24		Camilo Sampedro
25		Filippo Vitale
26	, Def	Aamir, John Starich, jwvh, linkhyrule5, Nathaniel Ford, Shastick, Shuklaswag, stefanobaghino, ZbyszekKr
27		Nathaniel Ford, Shuklaswag
28		acjay, Akash Sethi, David Leppik, dimitrisli, jwvh, Suma, Tzach Zohar
29		Andy Hayden, Dan Simon, dk14, Gábor Bakos, HTNW, J Cracknell, keegan, made raka teja, Marc Grue, Nathaniel Ford, pedrorijo91, Rumoku, ScientiaEtVeritas, suj1th, Suma
30		isaias-b, kevin628, Nathaniel Ford, nukie, Shastick
31		Gábor Bakos, irundaia
32		chengpohi
33		Hristo Iliev, Matas Vaitkevicius, Nathaniel Ford, Rjk
34		Gábor Bakos, Shaido, Suminda Sirinath S. Dharmasena
35		jwvh, Nathaniel Ford, Oleg Pyzhcov
36		corvus_192, implicitdef, inzi, mnoronha, Nathaniel Ford, Simon
37		Andy Hayden, Cortwave, Daniel Schröter, Gábor Bakos, implicitdef, ipoteka, Nathaniel Ford, phantomastray, Red Mercury
38		Andy Hayden, Graham, John Starich, made raka teja,

		mnoronha , Nathaniel Ford , Simon , Suma , tacos_tacos_tacos , Tzach Zohar
39		Bruce Lowe , CPS , earldouglas , evan.oman , Governa , John Starich , Matthew Scharley , Nathaniel Ford , R Pieters , ScientiaEtVeritas , suj1th , Tzach Zohar , Vasiliy Levykin
40		Bianca Tesila , Nathaniel Ford , Rjk
41	()	akauppi , Andy Hayden , Eero Helenius , Nathaniel Ford , vivek
42		J Cracknell
43		acjay , J Cracknell , Reactormonk
44		Gábor Bakos , Nathaniel Ford , suj1th
45		Arseniy Zhizhelev , Daniel C. Sobral , Gábor Bakos , gregghz , Nathaniel Ford , TomTom , Yawar
46		Hoang Ong
47		Dmitry Bystritsky , Gábor Bakos , jilen , jwvh , michael_s , ScientiaEtVeritas , teldosas
48		dmitry , J Cracknell , Nathaniel Ford
49		Andy Hayden , Dan Hulme , Dan Simon , Gábor Bakos , gilad hoch , Idloj , J Cracknell , jwvh , knutwalker , Łukasz , Martin Seeler , Michael Ahlers , Nathaniel Ford , Suma , W.P. McNeill
50	()	Gábor Bakos
51		Adamos Loizou , alphaloop , Amr Gawish , dimitrisli , Luka Jacobowitz , Nathaniel Ford , rjsvaljean , Suma , vise890
52		Anton , Camilo Sampedro , deepkimo , Donald.McLean , doub1ejack , EdgeCaseBerg , Filippo Vitale , George , implicitdef , ipoteka , Jason , John Starich , Mr D , Nathaniel Ford , raam86 , Shastick , Suma , Tundebabzy , Vasiliy Levykin
53		Aamir , Gábor Bakos , mdarwin , mirosval , MSmedberg , Nathaniel Ford , ScientiaEtVeritas , steve , Sudhir Singh , Tzach Zohar , vivek
54		corvus_192 , evan.oman , Lawsy , Nathaniel Ford
55		Gábor Bakos , Nathaniel Ford , Thomas Matecki
56		Nathaniel Ford
57		Alex Javarotti , Nathaniel Ford , NetanelRabinowitz

58		Ali Dehghani, Andrzej Jozwik, Andy Hayden, CPS, Dan Simon, Daniel Werner, Filippo Vitale, Gábor Bakos, implicitdef, insan-e, jilen, jozic, JRomero, Justin Bailey, Louis F., mammothbane, Matt, Nadim Bahadoor, Nathaniel Ford, Peter Neyens, Sergio, Shastick, Shoe, Simon, Suma, T.Grottker, user6062072, vdebergue, vsminkov, Yagüe
59		corvus_192, Nathaniel Ford, ScientiaEtVeritas
60		Andy Hayden, J Cracknell, jwvh, LivingRobot, Nathaniel Ford, ScientiaEtVeritas
61		Camilo Sampedro
62		André Laszlo, Andy Hayden, Donald.McLean, Louis F., Nathaniel Ford, Rumoku, Sudhir Singh, Vogon Jeltz