

# scalaz "For the Rest of Us" Cheat Sheet

Adam Rosien (adam@rosien.net)

29 August 2012

## Installation

In your build.sbt file:

```
libraryDependencies += "org.scalaz" %% "scalaz-core" % "6.0.4"
```

Then in your .scala files: <sup>1</sup>

```
import scalaz._
import Scalaz._
```

<sup>1</sup> Note that this is for scalaz 6. The imports (and many classes!) for scalaz 7 are much different.

## Style Stuff

Make your code a bit nicer to read.

Name	Scala	scalaz
"unix-pipey"	g(f(a))	a  > f  > g
ternary "operator"	if (p) "yes" else "no"	p ? "yes"   "no"
Option constructors	Some(42)	42.some
	None	none
Option.getOrElse	o.getOrElse("meh")	o   "meh"
Either constructors	Left("meh")	"meh".left
	Right(42)	42.right

## Memoization

```
def expensive(foo: Foo): Bar = ...

val memo = immutableHashMapMemo {
  foo: Foo => expensive(foo)
}

val f: Foo

memo(f) // $$$ (cache miss & fill)
memo(f) // 1¢ (cache hit)
```

Constructor	Backing store
immutableHashMapMemo[K, V]	HashMap
mutableHashMapMemo[K, V]	mutable.HashMap
weakHashMapMemo[K, V]	remove+gc unused entries
arrayMemo[V] (size: Int)	fixed size, K = Int

## Validation

Validation improves on Either: Success/Failure is more natural than Left/Right, and Validations can be composed together, accumulating failures.

Validation[X, A] constructors	"meh".fail 42.success
ValidationNel[X, A] constructors	"meh".failNel 42.successNel
Lift failure type into NonEmptyList	v.liftFailNel
De-construct into Failure or Success	v.fold( fail => ..., success => ...)
Combine Validations, accumulating failures (if any)	(ValidationNEL[X, A]  @  ValidationNEL[X, B]) { (A, B) => C } // ValidationNEL[X, C]

Table 1:

ValidationNEL[X, A] is a type alias for Validation[NonEmptyList[X], A]. NonEmptyList[X] is, well, a List that can't be empty.

## Lens

Lens is a composable "getter/setter" object, letting you "peek" into a deep structure, and also transform that "slot" you are pointing at.

Lens constructor	Lens[A, B](get: A => B, set: (A, B) => A)
compose	andThen[C](that: Lens[B, C]) = Lens[A, C]
pair	***[C, D](that: Lens[C, D]) = Lens[(A, C), (B, D)]
get	lens(a: A)
set	lens.set(a: A, b: B)
modify	lens.mod(a: A, f: B => B)