



# Scala Exercises

WHAT IS IT?

A BROWSER TOOL TO LEARN SCALA AND ITS COOLEST LIBS

## A BIT OF HISTORY

- › @RAFAPARADELA JS POC FOR THE @47DEG LABS
  - › V1 WENT LIVE ON MARCH 2ND. 2015
- › FOCUSED ON PORTING SCALA KOANS TO THE WEB
  - › SOCIAL COMPONENT

# V1 LIMITATIONS

- > SERVERLESS
- > "EVALUATION" CONSTRAINED TO EXACT STRING MATCHES
- > CAN'T TRACK USER PROGRESS ACROSS DEVICES
- > CAN'T PARSE OR EVALUATE MORE COMPLEX EXERCISES

# V1 STATS

- > 63K SESSIONS
- > 40% RETURNING
- > 180 PRS
- > 50 CONTRIBUTORS
- > 1 SECTION COMPLETED PER VISIT

# WHAT'S NEW IN V2?

# LIBS AT A GLANCE

The path to enlightenment.  
A series of exercises and challenges covering different technologies based in the Scala Programming Language.

**Std Lib**

Scala fuses object-oriented and functional programming in a statically typed programming language.

**Cats**

Cats is an experimental library intended to provide abstractions for functional programming in Scala.

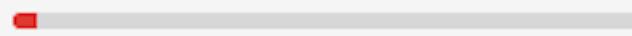
**Shapeless**

Shapeless is a type class and dependent type based generic programming library for Scala.

# OVERALL PROGRESS

## Std Lib

Scala fuses object-oriented and functional programming in a statically typed programming language.



2 / 42

## Cats

Cats is an experimental library intended to provide abstractions for functional programming in Scala.



1 / 2

## Shapeless

Shapeless is a type class and dependent type based generic programming library for Scala.

Start

# LIB PROGRESS

The screenshot shows a web page from the Scala Exercises / Cats website. The title bar says "Scala Exercises / Cats". On the left, there's a sidebar with a progress bar at 1/2 and a "Sections" list containing "Identity" and "Xor" (which is selected). The main content area discusses the Either type and its RightProjection instance. It shows code examples:

```
val e1: Either[String, Int] = Right(5)
e1.right.map(_ + 1)

val e2: Either[String, Int] = Left("hello")
e2.right.map(_ + 1)
```

Note the return types are themselves back to Either, we again must call right or left.

More often than not we want to just bias towards one often chosen. This is the primary difference between Xor and Either. There are more convenient methods on it, but the most crucial one is:

```
import cats.data.Xor

val right: String Xor Int = Xor.right(5)
right.map(_ + 1) should be(Xor.right(6))

val left: String Xor Int = Xor.left("Something")
left.map(_ + 1) should be(Xor.left("Something"))
```

Because Xor is right-biased, it is possible to define a flatMap method on it. However, but due to how it's encoded it may seem strange to fix the order of arguments. The Monad instance for Xor is consistent with the Either instance. Using Either would only introduce bias when Either is used.

# WRITE EXERCISES WITH CODE!

```
1 package catslib
2
3 import cats._
4 import org.scalatest._
5
6 /**
7  * identity
8  * The identity monad can be seen as the ambient monad that encodes the
9  * effect of having no effect. It is ambient in the sense that plain pure
10 * values are values of `Id`.
11 *
12 * It is encoded as:
13 *
14 * {{{
15 * type Id[A] = A
16 * }}}
17 *
18 * That is to say that the type Id[A] is just a synonym for A. We can
19 * freely treat values of type `A` as values of type Id[A], and
20 * vice-versa.
21 *
22 * {{{
23 * import cats._
24 *
25 * val x: Id[Int] = 1
26 * val y: Int = x
27 * }}}
28 *
29 */
30 object IdentitySection extends FlatSpec with Matchers with exercise.Section {
31
32 /**
33 * We can freely compare values of Id[T] with unadorned
34 * values of type T.
35 */
36 def identityType(res0: Int) {
37   val anId: Id[Int] = 42
38   anId should be (res0)
39 }
```

http://scala-exercises.47deg.com/

**Scala Exercises / Cats**

## Identity

The identity monad can be seen as the ambient monad that encodes the effect of having no effect. It is ambient in the sense that plain pure values are values of `Id`.

It is encoded as:

```
type Id[A] = A
```

That is to say that the type `Id[A]` is just a synonym for `A`. We can freely treat values of type `A` as values of type `Id[A]`, and vice-versa.

```
import cats._

val x: Id[Int] = 1
val y: Int = x
```

We can freely compare values of `Id[T]` with unadorned values of type `T`.

```
val anId: Id[Int] = 42
anId should be (_)
```

▶ Run

# WRITE EXERCISES WITH CODE!

A screenshot of a code editor window showing Scala code. The code defines an object `IdentitySection` that extends `FlatSpec` and `Matchers` with `exercise.Section`. It contains documentation for the `identity` method, which encodes the identity monad as an ambient monad. It also includes a code snippet demonstrating the usage of `Id[Int]` values.

```
1 package catslib
2
3 import cats._
4 import org.scalatest._
5
6 /**
7  * identity
8  *
9  * The identity monad can be seen as the ambient monad that encodes the
10 * effect of having no effect. It is ambient in the sense that plain pure
11 * values are values of `Id`.
12 *
13 * It is encoded as:
14 *
15 * {{{
16 * type Id[A] = A
17 * }}}
18 *
19 * That is to say that the type Id[A] is just a synonym for A. We can
20 * freely treat values of type 'A' as values of type 'Id[A]', and
21 * vice-versa.
22 *
23 * {{{
24 * import cats._
25 *
26 * val x: Id[Int] = 1
27 * val y: Int = x
28 * }}}
29 */
30 object IdentitySection extends FlatSpec with Matchers with exercise.Section {
31
32 /**
33 * We can freely compare values of 'Id[T] with unadorned
34 * values of type 'T'.
35 */
36 def identityType(res0: Int) {
37   val anId: Id[Int] = 42
38   anId should be (res0)
39 }
```

A screenshot of the Scala Exercises / Cats website. The page title is "Identity". It contains a detailed description of the identity monad and its encoding. Below the description are two code snippets: one showing the definition of `Id[A]` and another showing a comparison between `Id[Int]` and `Int` values.

The identity monad can be seen as the ambient monad that encodes the effect of having no effect. It is ambient in the sense that plain pure values are values of `Id`.

It is encoded as:

```
type Id[A] = A
```

That is to say that the type `Id[A]` is just a synonym for `A`. We can freely treat values of type `A` as values of type `Id[A]`, and vice-versa.

```
import cats._

val x: Id[Int] = 1
val y: Int = x
```

We can freely compare values of `Id[T]` with unadorned values of type `T`.

```
val anId: Id[Int] = 42
anId should be (_)
```

▶ Run

# WRITE EXERCISES WITH CODE!

A screenshot of a code editor window. The code is written in Scala and defines an identity monad. It includes documentation comments explaining the purpose of the monad and its encoding. The code also shows an example of using the identity monad with integers.

```
1 package catslib
2
3 import cats._
4 import org.scalatest._
5
6 /**
7  * identity
8  *
9  * The identity monad can be seen as the ambient monad that encodes the
10 * effect of having no effect. It is ambient in the sense that plain pure
11 * values are values of `Id`.
12 *
13 * It is encoded as:
14 */
15 type Id[A] = A
16
17
18 * That is to say that the type Id[A] is just a synonym for A. We can
19 * freely treat values of type `A` as values of type Id[A], and
20 * vice-versa.
21 *
22 */
23 import cats._
24
25 val x: Id[Int] = 1
26 val y: Int = x
27
28 */
29
30 object IdentitySection extends FlatSpec with Matchers with exercise.Section {
31
32 /**
33 * We can freely compare values of Id[T] with unadorned
34 * values of type T.
35 */
36 def identityType(res0: Int) {
37   val anId: Id[Int] = 42
38   anId should be (res0)
39 }
```

A screenshot of the Scala Exercises website. The page title is "Identity". It contains text explaining the identity monad and its encoding. Below the text is a code snippet showing how to define the identity monad. A callout box highlights the ability to compare values of `Id[T]` with unadorned values of type `T`. At the bottom, there is a code editor with a test case for the `identityType` function.

The identity monad can be seen as the ambient monad that encodes the effect of having no effect. It is ambient in the sense that plain pure values are values of `Id`.

It is encoded as:

```
type Id[A] = A
```

That is to say that the type `Id[A]` is just a synonym for `A`. We can freely treat values of type `A` as values of type `Id[A]`, and vice-versa.

```
import cats._

val x: Id[Int] = 1
val y: Int = x
```

We can freely compare values of `Id[T]` with unadorned values of type `T`.

```
val anId: Id[Int] = 42
anId should be (___)
```

▶ Run

# WRITE EXERCISES WITH CODE!

A screenshot of a code editor window showing Scala code. The code defines an object `IdentitySection` that extends `FlatSpec` and `Matchers`, and implements `exercise.Section`. It contains documentation for the `identity` method, which encodes the identity monad as an ambient monad. It also shows examples of creating an `Id[Int]` value and comparing it with an unadorned `Int` value. A test method `identityType` is defined to check if `anId` is equal to `42`.

```
1 package catslib
2
3 import cats._
4 import org.scalatest._
5
6 /**
7  * identity
8  *
9  * The identity monad can be seen as the ambient monad that encodes the
10 * effect of having no effect. It is ambient in the sense that plain pure
11 * values are values of `Id`.
12 *
13 * It is encoded as:
14 */
15 type Id[A] = A
16
17
18 * That is to say that the type Id[A] is just a synonym for A. We can
19 * freely treat values of type `A` as values of type `Id[A]`, and
20 * vice-versa.
21
22 */
23 import cats._
24
25 val x: Id[Int] = 1
26 val y: Int = x
27
28 */
29
30 object IdentitySection extends FlatSpec with Matchers with exercise.Section {
31
32 /**
33  * We can freely compare values of `Id[T]` with unadorned
34  * values of type `T`.
35 */
36 def identityType(res0: Int) {
37   val anId: Id[Int] = 42
38   anId should be (res0)
39 }
```

A screenshot of the [Scala Exercises](http://scala-exercises.47deg.com/) website. The page title is "Identity". It contains the same documentation and code examples as the code editor. Below the code, a text box states: "We can freely compare values of `Id[T]` with unadorned values of type `T`". A test runner interface is shown with the following code:  
`val anId: Id[Int] = 42  
anId should be (_)`  
A "Run" button is visible at the bottom right.

The page URL is <http://scala-exercises.47deg.com/>

Scala Exercises / Cats

## Identity

The identity monad can be seen as the ambient monad that encodes the effect of having no effect. It is ambient in the sense that plain pure values are values of `Id`.

It is encoded as:

```
type Id[A] = A
```

That is to say that the type `Id[A]` is just a synonym for `A`. We can freely treat values of type `A` as values of type `Id[A]`, and vice-versa.

```
import cats._

val x: Id[Int] = 1
val y: Int = x
```

We can freely compare values of `Id[T]` with unadorned values of type `T`.

```
val anId: Id[Int] = 42
anId should be (_)
```

▶ Run

# WRITE EXERCISES WITH CODE!

## REAL EVALUATION OF EXERCISES

The screenshot shows a web-based Scala exercise interface. At the top, there's a navigation bar with the title "Scala Exercises / Cats" and a user profile for "Israel Pérez". Below the navigation, a progress bar indicates "1 / 2". On the left, a sidebar titled "Sections" lists "Identity" (which is selected) and "Xor". The main content area contains text explaining the identity monad and its encoding:

The identity monad can be seen as the ambient monad that encodes the effect of having no effect. It is ambient in the sense that plain pure values are values of `Id`.  
It is encoded as:

```
type Id[A] = A
```

That is to say that the type `Id[A]` is just a synonym for `A`. We can freely treat values of type `A` as values of type `Id[A]`, and vice-versa.

```
import cats._

val x: Id[Int] = 1
val y: Int = x
```

```
val anId: Id[Int] = 42
anId should be ( _ )
```

At the bottom, there's a note about the `pure` method and the `map` method from `Functor`:

The `pure` method, which has type `A => Id[A]` just becomes the identity function. The `map` method from `Functor` just becomes function application:

```
import cats.Functor
```

# CONTRIBUTORS ARE PROPERLY CREDITED

The screenshot shows a web page from the Scala Exercises website. At the top, there's a navigation bar with the title "Scala Exercises / Cats". On the right side of the bar, there's a user profile for "Israel Pérez". Below the bar, a progress bar indicates "1 / 2".

**Sections**

- Identity
- Xor

In the main content area, there's a code snippet:

```
def flatMap[A, B](a: Id[A])(f: Id[A] => B): Id[B]
```

A explanatory text follows:

You'll notice that in the flatMap signature, since `Id[B]` is the same as `B` for all `B`, we can rewrite the type of the `f` parameter to be `A => B` instead of `A => Id[B]`, and this makes the signatures of the two functions the same, and, in fact, they can have the same implementation, meaning that for `Id`, `flatMap` is also just function application:

```
import cats.Monad

val one: Int = 1
Monad[Id].map(one)(_ + 1)
Monad[Id].flatMap(one)(_ + 1)
```

Below this, another code snippet is shown:

```
val fortytwo: Int = 42
Comonad[Id].coflatMap(fortytwo)(_ + 1) should be (_)
```

On the right of this snippet is a "Run" button.

At the bottom of the page, there's a footer with the Scala Exercise logo and links to GitHub, contact, terms, and privacy policies. It also mentions "6 contributors" with small profile icons and a "Edit exercises" link.

Scala Exercise is an Open Source project by 47 Degrees  
View on GitHub  
Contact us · Term of use · Privacy policy

6 contributors

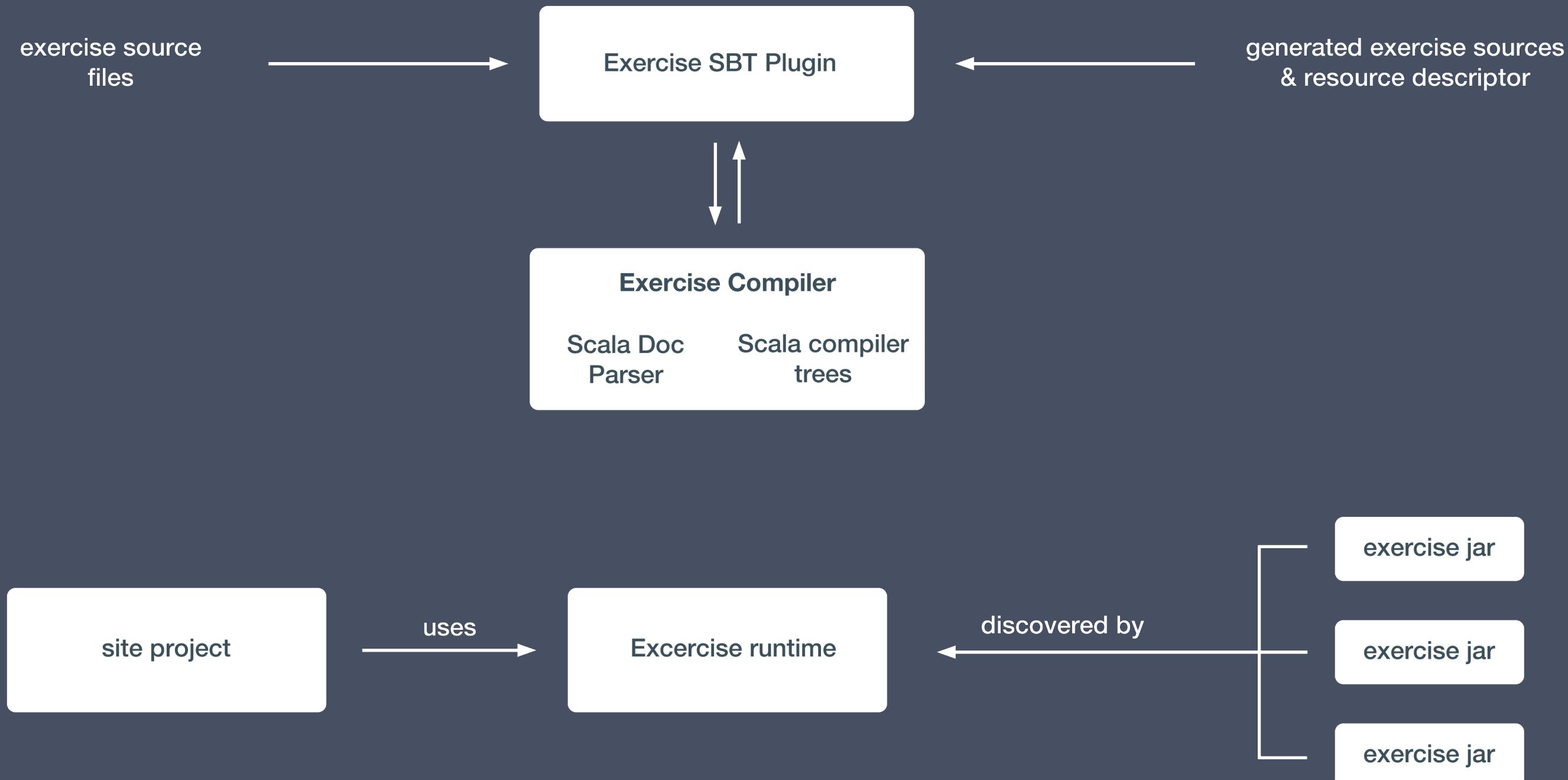
Edit exercises

# WE WANT YOU!



TO CONTRIBUTE TO SCALA  
EXERCISES!

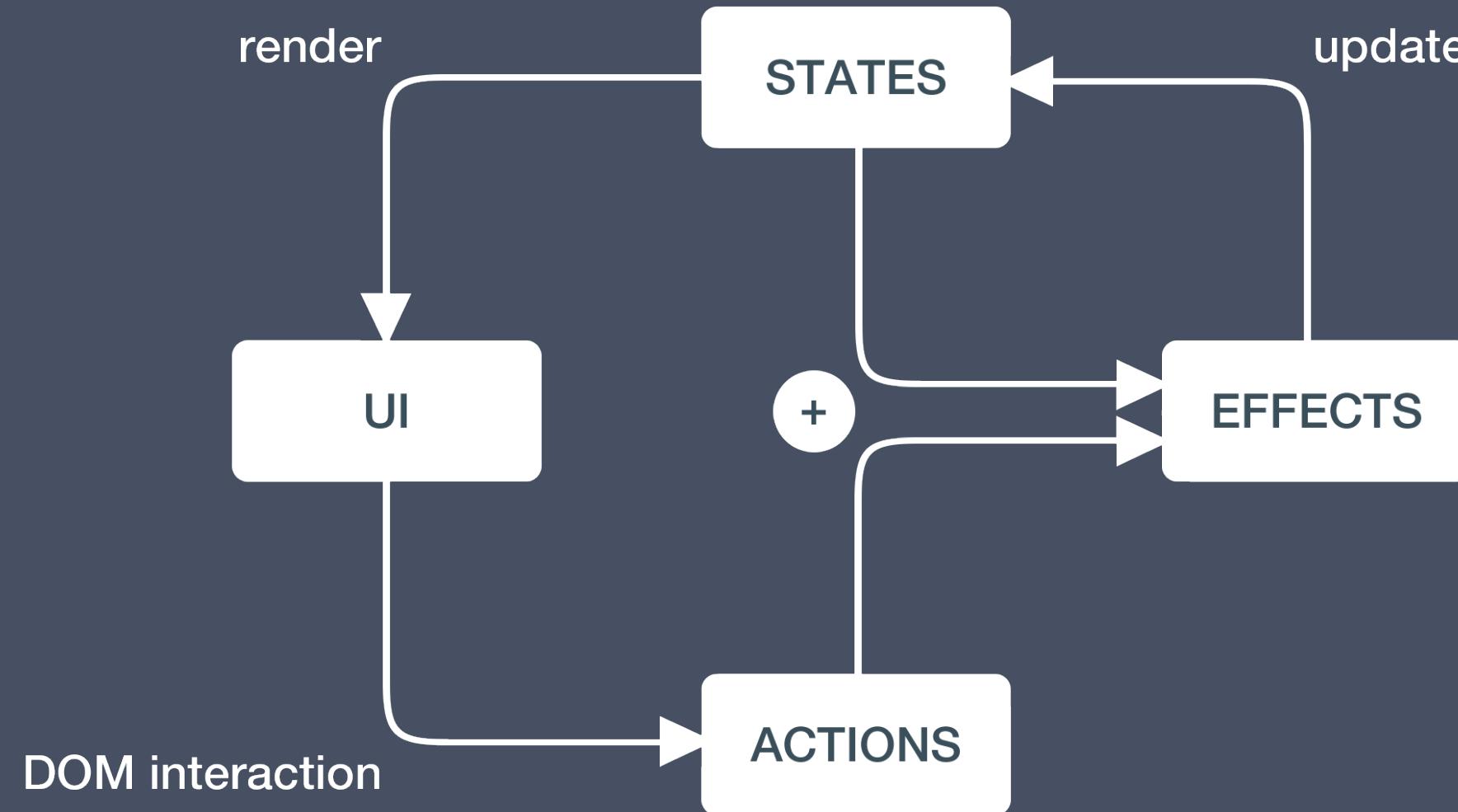
# EXERCISES COMPILER ARCHITECTURE



# TECHNOLOGIES

- › CLIENT [SCALajs. Cats]
- › SERVER [Play. Cats. Doobie]

# CLIENT ARCHITECTURE



# BACKEND ARCHITECTURE

```
type App = Coproduct [  
]  
]
```

# BACKEND ARCHITECTURE

```
type App = Coproduct [
```

ExerciseOp
- GetLibraries
- GetLibrary
- GetSection
- Evaluate

- GetLibraries
- GetLibrary
- GetSection
- Evaluate

```
]
```

# BACKEND ARCHITECTURE

```
type App = Coproduct [
```

ExerciseOp	DBResult
<ul style="list-style-type: none"><li>- GetLibraries</li><li>- GetLibrary</li><li>- GetSection</li><li>- Evaluate</li></ul>	<ul style="list-style-type: none"><li>- Success</li><li>- Failure</li></ul>

,

```
]
```

# BACKEND ARCHITECTURE

```
type App = Coproduct [
```

ExerciseOp
- GetLibraries
- GetLibrary
- GetSection
- Evaluate

DBResult
- Success
- Failure

UserOp
- GetUser
- GetUserByLogin
- CreateUser
- UpdateUser
- DeleteUser

```
]
```

# BACKEND ARCHITECTURE

```
type App = Coproduct [
```

ExerciseOp
- GetLibraries
- GetLibrary
- GetSection
- Evaluate

,

DBResult
- Success
- Failure

,

UserOp
- GetUser
- GetUserByLogin
- CreateUser
- UpdateUser
- DeleteUser

,

UserProgress
- UpdateProgress

```
]
```

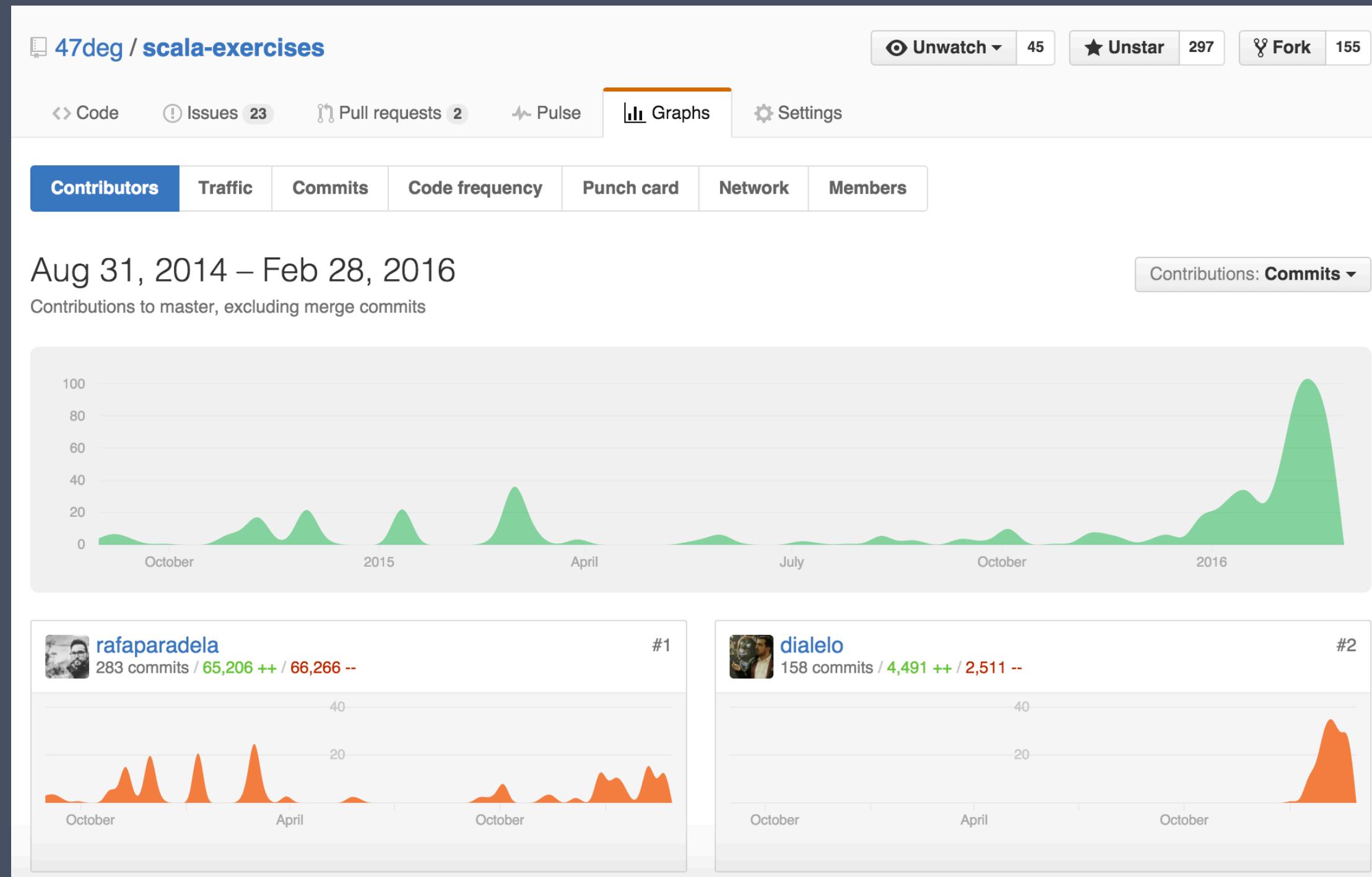
# FREE MONADS!



# FREE MONADAS LIBRES!



# WARNING! V2 IS UNDER HEAVY DEVELOPMENT!



ACTIVELY WORKING ON...

- > IMPROVED EVALUATION / FORK EVALUATION (FINCH)
- > SUPPORT FOR BLOCK STYLE EXERCISES (EX : IMPLEMENT A TYPE CLASS)
- > DOCS. CONTRIBUTION GUIDE...
- > SPLITTING INTO MULTIPLE REPOSITORIES

# WHAT WOULD BE COMING NEXT?

- > ALLOW GITHUB ORGANIZATIONS ?
- > SUPPORT OTHER EXERCISES FORMAT
  - > TOP LEVEL DOMAIN / ORG
  - > IT REALLY IS ENTIRELY UP TO YOU!

THANKS!

@RAULRAJA

@47DEG

- GITHUB : [HTTPS://GITHUB.COM/47DEG/SCALA-EXERCISES](https://github.com/47deg/scala-exercises)
- DECK: [HTTPS://GITHUB.COM/47DEG/SCALA-EXERCISES-V2-DECK](https://github.com/47deg/scala-exercises-v2-deck)
- V1: [HTTP://SCALA-EXERCISES.47DEG.COM](http://scala-exercises.47deg.com)