Introduction to meta-programming in Scala

Alessandro Marrella

About me

- Software Engineer
- Working with Scala professionally for 3 years
- ♡ automating stuff
- ♡ functional programming
- github.com/amarrella
- linkedin.com/in/alessandromarrella

CREDITS

- Ólafur Páll Geirsson (@olafurpg)
- Scala Center / EPFL (scala.epfl.ch)
- Twitter

+ many other contributors in the Scala community

Meta-programming

- Treating other programs as **Data**
- Ability to
  - **Read**
  - **Generate**
  - **Transform**
  - **Analyze**

  other programs

WHY?

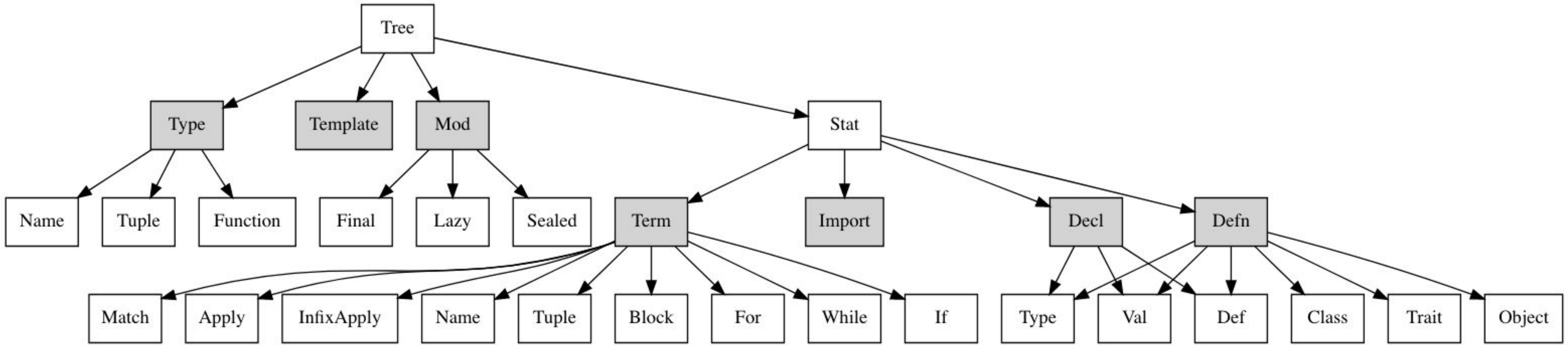
- Reduce lines of code & dev time
- Reduce boilerplate, errors and boring stuff

Scalameta

- **Library** to read, analyze, transform and generate Scala programs
  - Syntactic API: parse, transform & prettyprint
  - Semantic API: understand symbols and types
- **Ecosystem**
  - Scalameta (Trees & SemanticDB)
  - Scalafmt: code formatting
  - Scalafix: code linting & rewriting
  - Metals: language server

& more(scalameta.org/docs/misc/built-with-scalameta.html)

Syntax trees



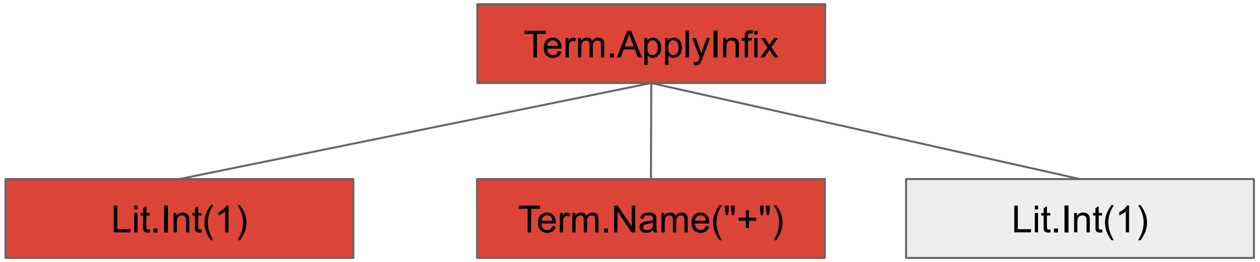
Syntax trees

```
1
```

```
Lit.Int(1)
```

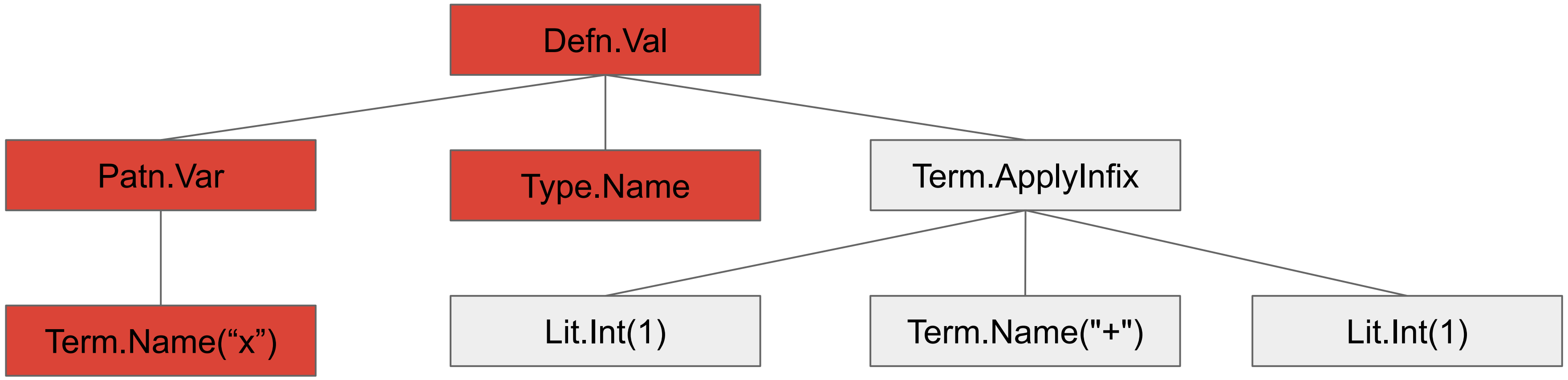
Syntax trees

1+1



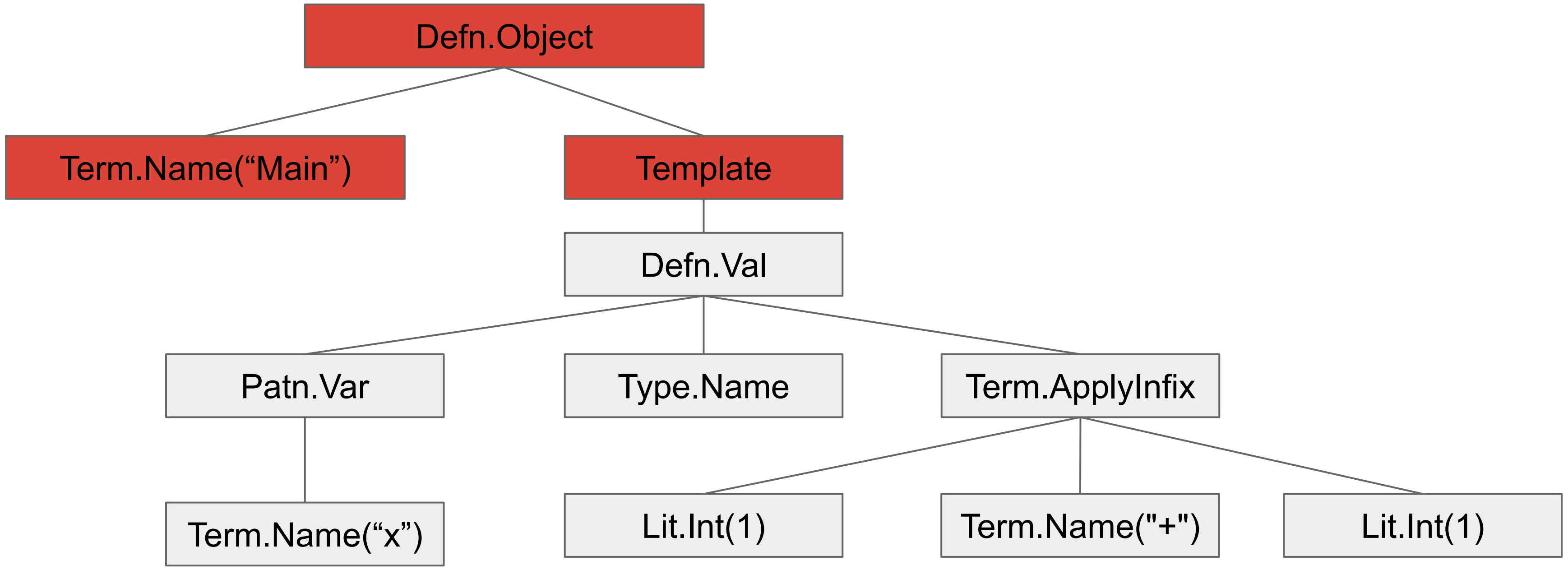
Syntax trees

val x: Int = 1+1



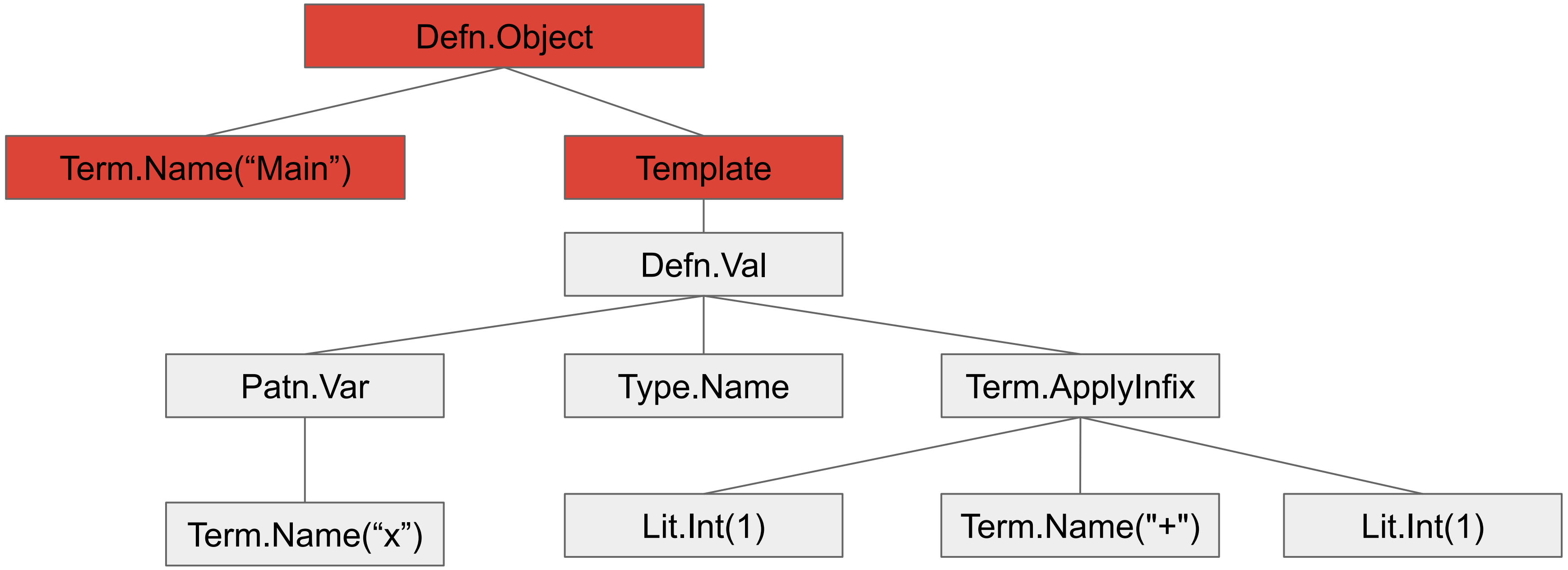
Syntax trees

object Main { val x: Int = 1+1 }



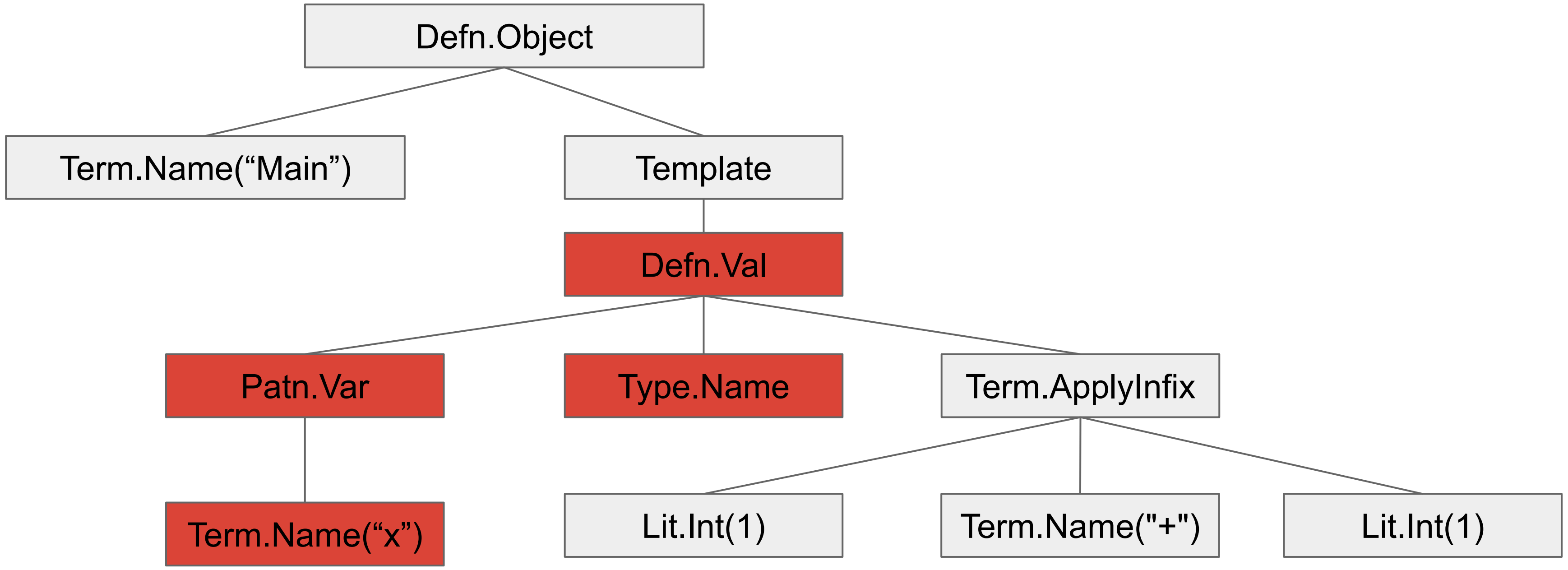
Syntax trees: from tree to code

object Main { }



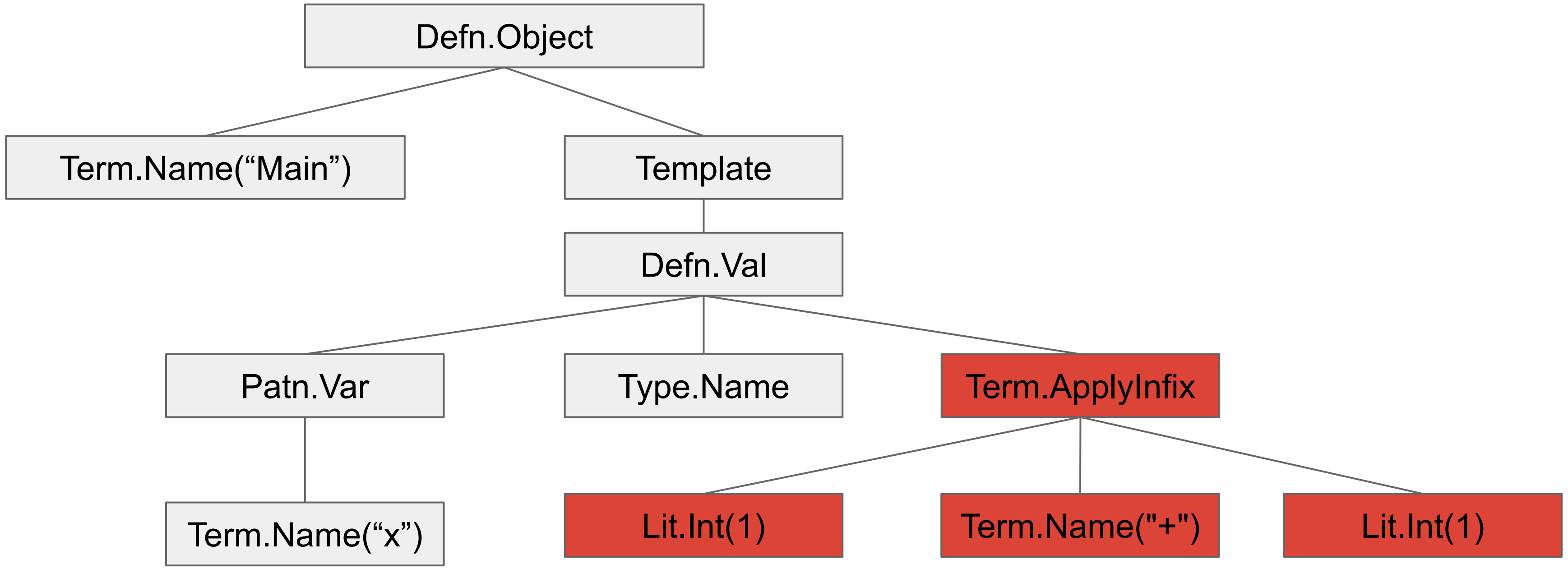
Syntax trees: from tree to code

object Main { val x: Int }



Syntax trees: from tree to code

object Main { val x: Int = 1 + 1 }



Scalameta

- Scalameta Trees are *lossless*
- Scalafmt: uses trees to figure out how to format
- Scalafix: uses trees to apply syntactic rules (more later)
- Explicit syntax or quasiquotes

```
q”object $name { val $val: $t = $expr }”
```

https://scalameta.org/docs/trees/guide.html

SemanticDB

- Compiles scala programs to semantic information
- Decouples producing and consuming semantic information

SemanticDB

```
package com.alessandromarrella.presentation

import cats.effect.IO

object IOExample {

   val x = IO(println("Hello world"))

}
```

SemanticDB

```scala
package com.alessandromarrella.presentation

import cats.effect.IO

object IOExample {

   val x = IO(println("Hello world"))

}
```

Summary:

Schema => SemanticDB v4

Uri => src/main/scala/com/alessandromarrella/presentation/IOExample.scala

Text => empty

Language => Scala

Symbols => 2 entries

Occurrences => 11 entries

SemanticDB

```scala
package com.alessandromarrella.presentation

import cats.effect.IO

object IOExample {

   val x = IO(println("Hello world"))

}
```

Symbols:

```
com/alessandromarrella/presentation/IOExample. => final object IOExample extends AnyRef { +1 decls }

com/alessandromarrella/presentation/IOExample.x. => val method x: IO[Unit]
```

SemanticDB

```
package com.alessandromarrella.presentation

import cats.effect.IO

object IOExample {

   val x = IO(println("Hello world"))

}
```

Occurrences:

position

```
[0:8..0:11) => com/

[0:12..0:30) =>
com/alessandromarrella/

[0:31..0:43) =>
com/alessandromarrella/presen
tation/
```

SemanticDB

```
package com.alessandromarrella.presentation

import cats.effect.IO

object IOExample {

   val x = IO(println("Hello world"))

}
```

Occurrences:

```
[2:7..2:11) => cats/

[2:12..2:18) => cats/effect/

[2:19..2:21) =>
cats/effect/IO# <- class

[2:19..2:21) =>
cats/effect/IO. <- object
```

SemanticDB

```scala
package com.alessandromarrella.presentation

import cats.effect.IO

object IOExample {

  val x = IO(println("Hello world"))

}
```

Occurrences:

```
[4:7..4:16) <=
com/alessandromarrella/presentation/IOExample.

[6:8..6:9) <=
com/alessandromarrella/presentation/IOExample.x.

[6:12..6:14) =>
cats/effect/IO.

[6:15..6:22) =>
scala/Predef.println(+1).
```

It "knows"

Method overload

SemanticDB

- Compiles the program to a SemanticDB file
- Symbol information
- Disambiguation
- Scalafix can use it for semantic rules (more later)
- Metals uses it for code navigation

https://scalameta.org/docs/semanticdb/guide.html

Scalafix

- **Refactoring** and **linting** tool
- **Syntactic** and **semantic** rules
- As a user:
  - Apply rules
  - Integrate with your CI
- As a dev:
  - Create & publish rules

Applying rules

Add scalafix dependency

project/plugins.sbt

```
addSbtPlugin("ch.epfl.scala" % "sbt-scalafix" % "0.9.1")
```

Run the rule (e.g. http4s 0.18 to 0.20 upgrade)

```
$ sbt ";scalafixEnable; scalafix github:http4s/http4s/v0_20"
```

Creating new rules

Create a new project with the Giter8 template

```
$ sbt new scalacenter/scalafix.g8
```

You get a new directory with:

build.sbt readme.md **input** **output** **rules** test

Example: No return Unit rule

```
def foo(x: Int): Unit
```

What does calling *foo(42)* do?

Example: No return Unit rule

`def foo(x: Int): Unit`

What does calling *foo(42)* do?

- Nothing
- Performs a side effect

Example: No return Unit rule

`def foo(x: Int): Unit`

What does calling *foo(42)* do?

- Nothing
- Performs a side effect

We won’t consider:

- Throwing exceptions
- Returning null

We don’t want the user to be able to return Unit.

Example: No return Unit rule

Demo

https://github.com/amarrella/noreturnunit

Thank you!

hello@alessandromarrella.com
linkedin.com/in/alessandromarrella
twitter.com/amarrella