SCALACON / 04.11.2021

# Describable, typesafe computations

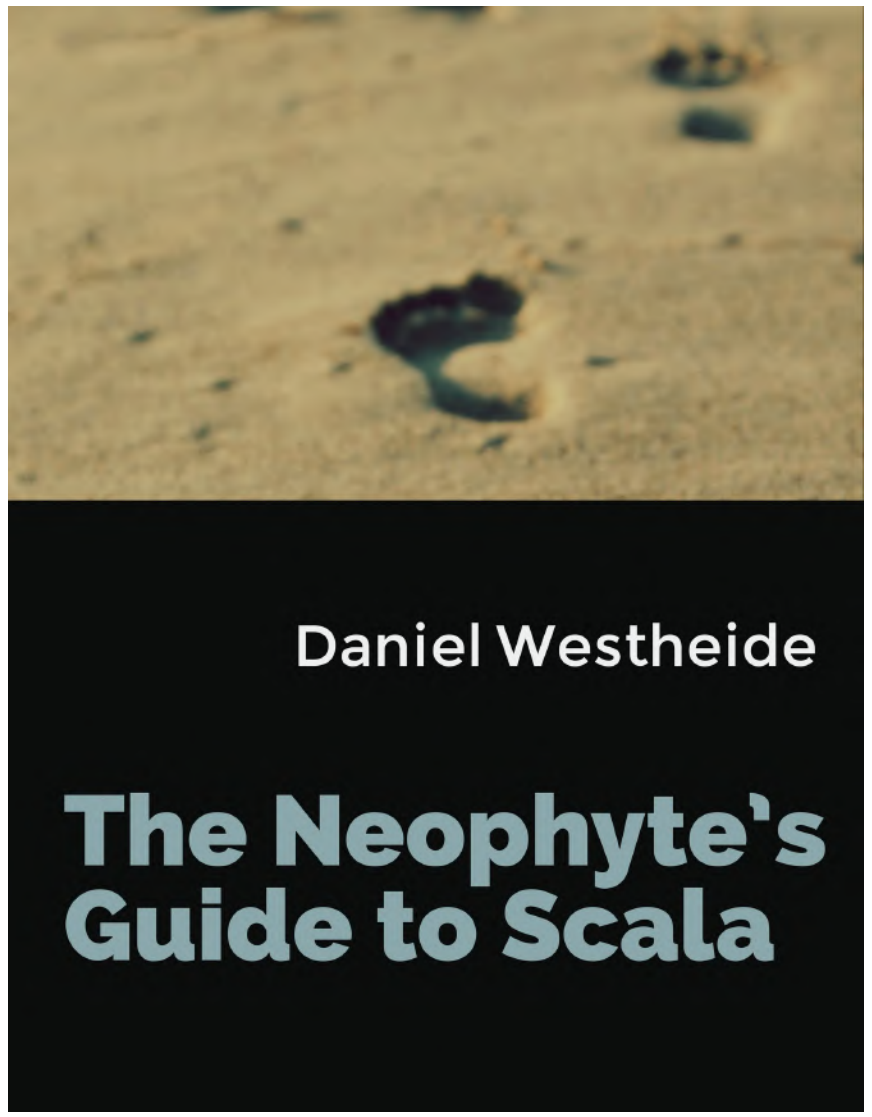
INNOQ

DANIEL WESTHEIDE
TWITTER: @KAFFEECODER

# About me

Senior consultant at INNOQ.

Author of three Scala books.



Twitter

LinkedIn

Disney Streaming

**Scala**

Finagle

Play Framework

Akka

Apache Spark

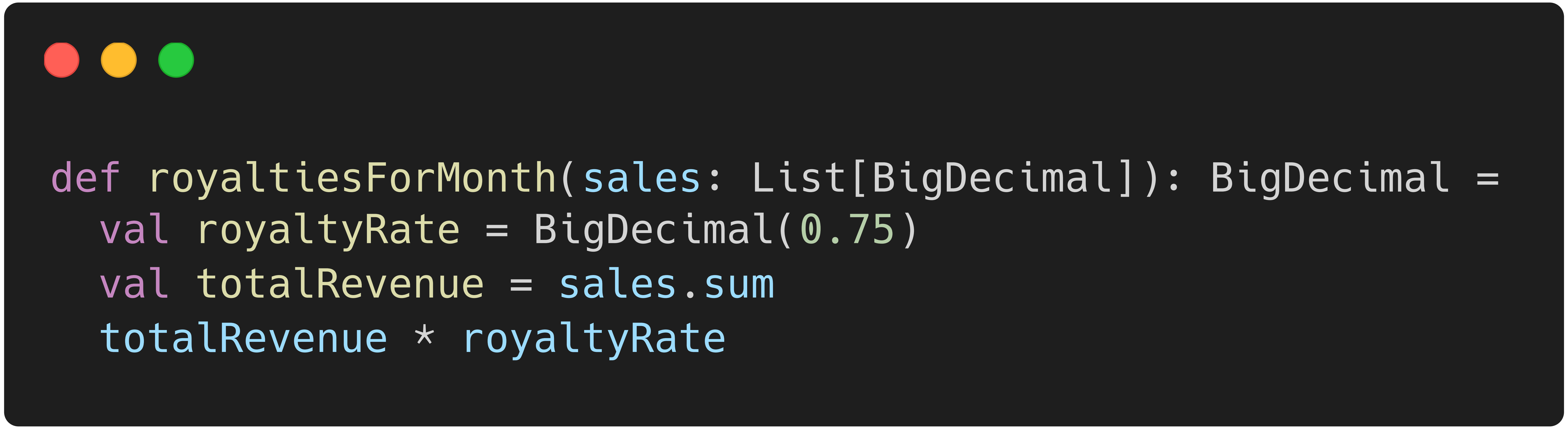
## Quality goals

correctness
auditability

## Example use case

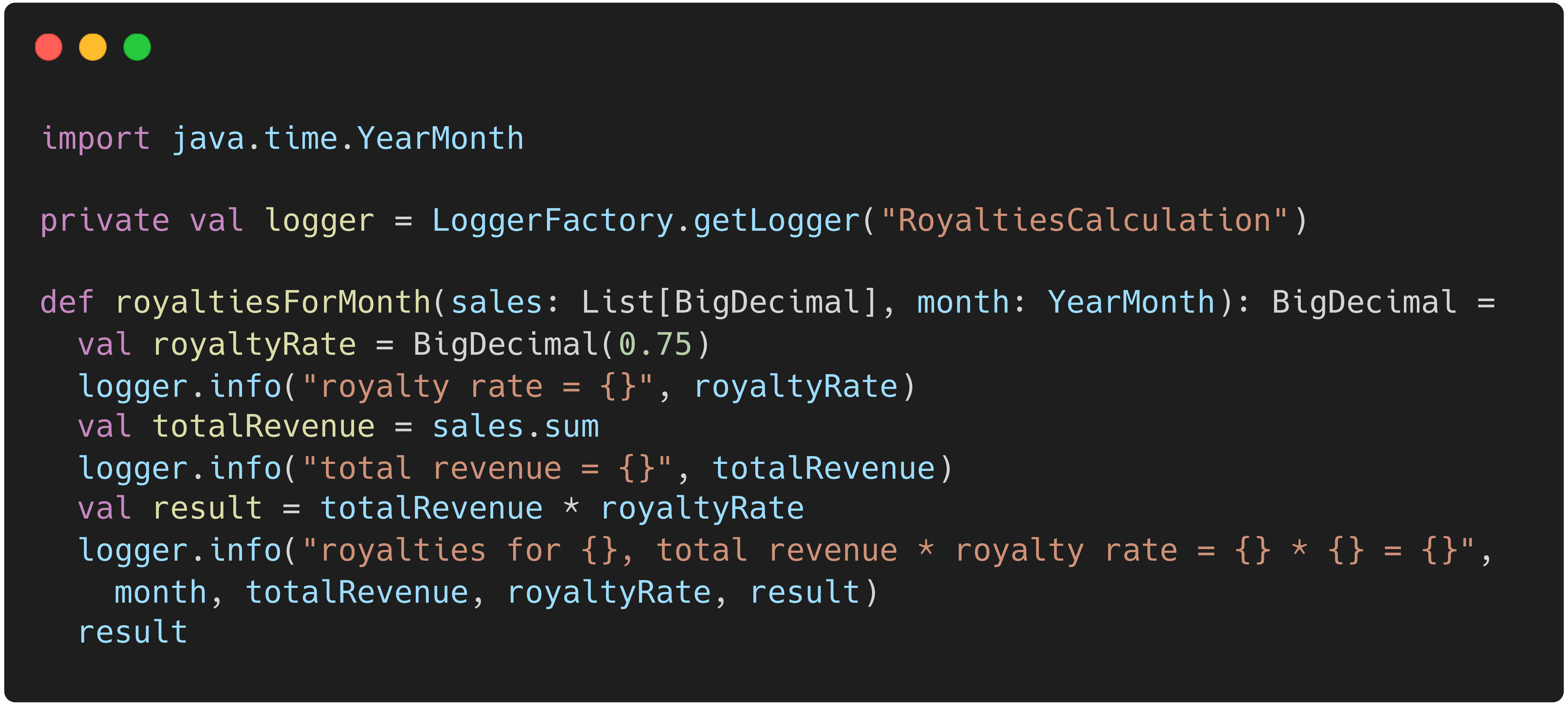
- **SelfPub**, a platform for creating and selling self-published digital books
- The royalty rate is 75% of the revenue
- SelfPub keeps at least EUR 0.80 of the revenue of each sold book

# SelfPub royalties calculation



```scala
def royaltiesForMonth(sales: List[BigDecimal]): BigDecimal =
  val royaltyRate = BigDecimal(0.75)
  val totalRevenue = sales.sum
  totalRevenue * royaltyRate
```

# Calculation with logging

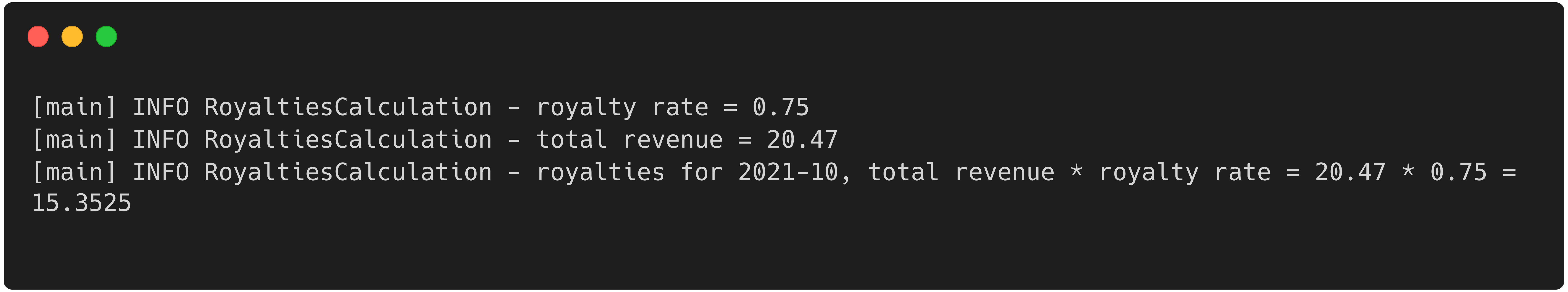


```scala
import java.time.YearMonth

private val logger = LoggerFactory.getLogger("RoyaltiesCalculation")

def royaltiesForMonth(sales: List[BigDecimal], month: YearMonth): BigDecimal =
  val royaltyRate = BigDecimal(0.75)
  logger.info("royalty rate = {}", royaltyRate)
  val totalRevenue = sales.sum
  logger.info("total revenue = {}", totalRevenue)
  val result = totalRevenue * royaltyRate
  logger.info("royalties for {}, total revenue * royalty rate = {} * {} = {}",
    month, totalRevenue, royaltyRate, result)
  result
```

## Log output

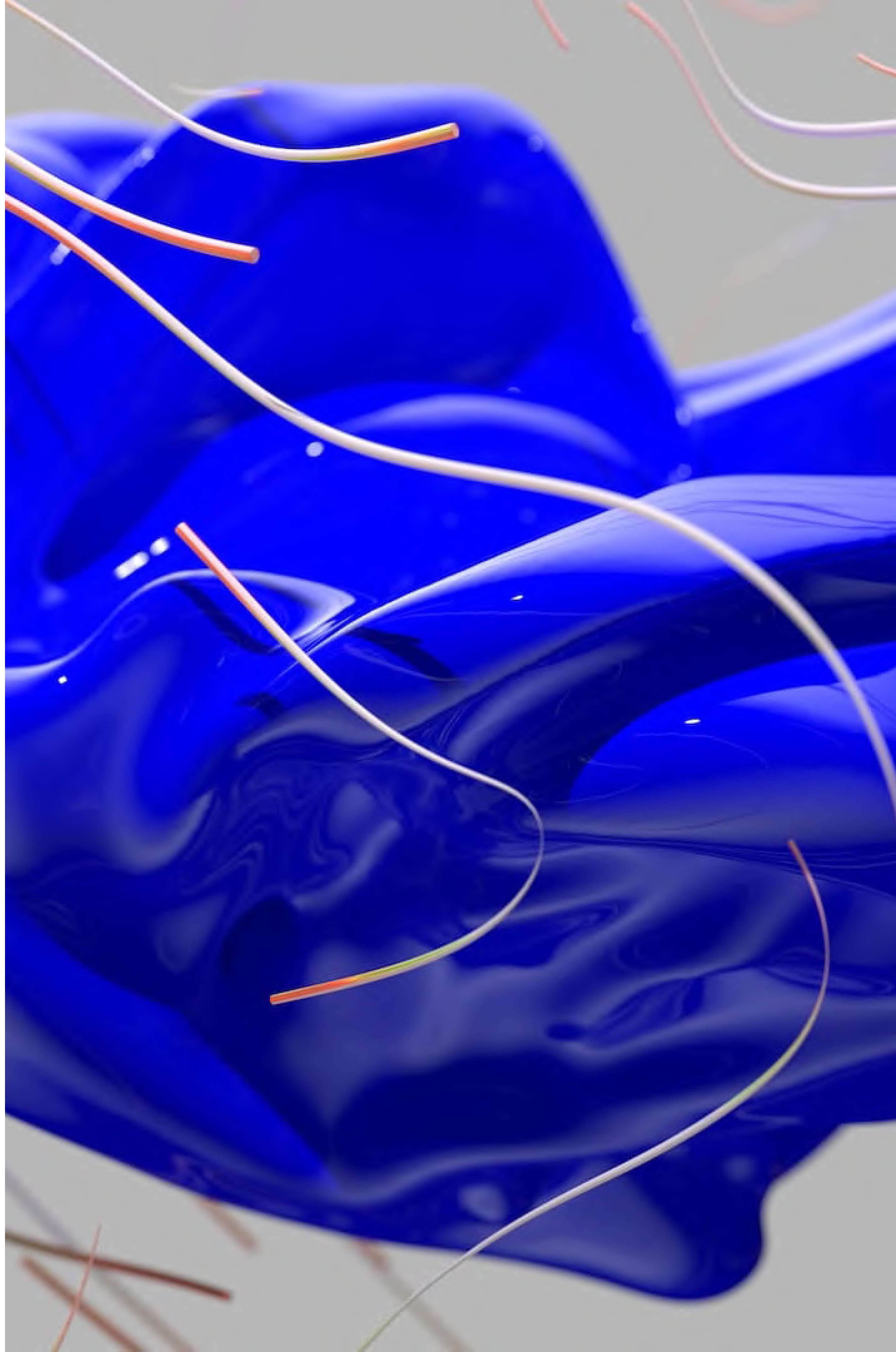


```
[main] INFO RoyaltiesCalculation - royalty rate = 0.75
[main] INFO RoyaltiesCalculation - total revenue = 20.47
[main] INFO RoyaltiesCalculation - royalties for 2021-10, total revenue * royalty rate = 20.47 * 0.75 =
15.3525
```

## Assessment

- easy to implement
- difficult to read and maintain
- complection of calculation and logging
- error-prone
- difficult to make calculation log available to stakeholders
- logging as a side-effect

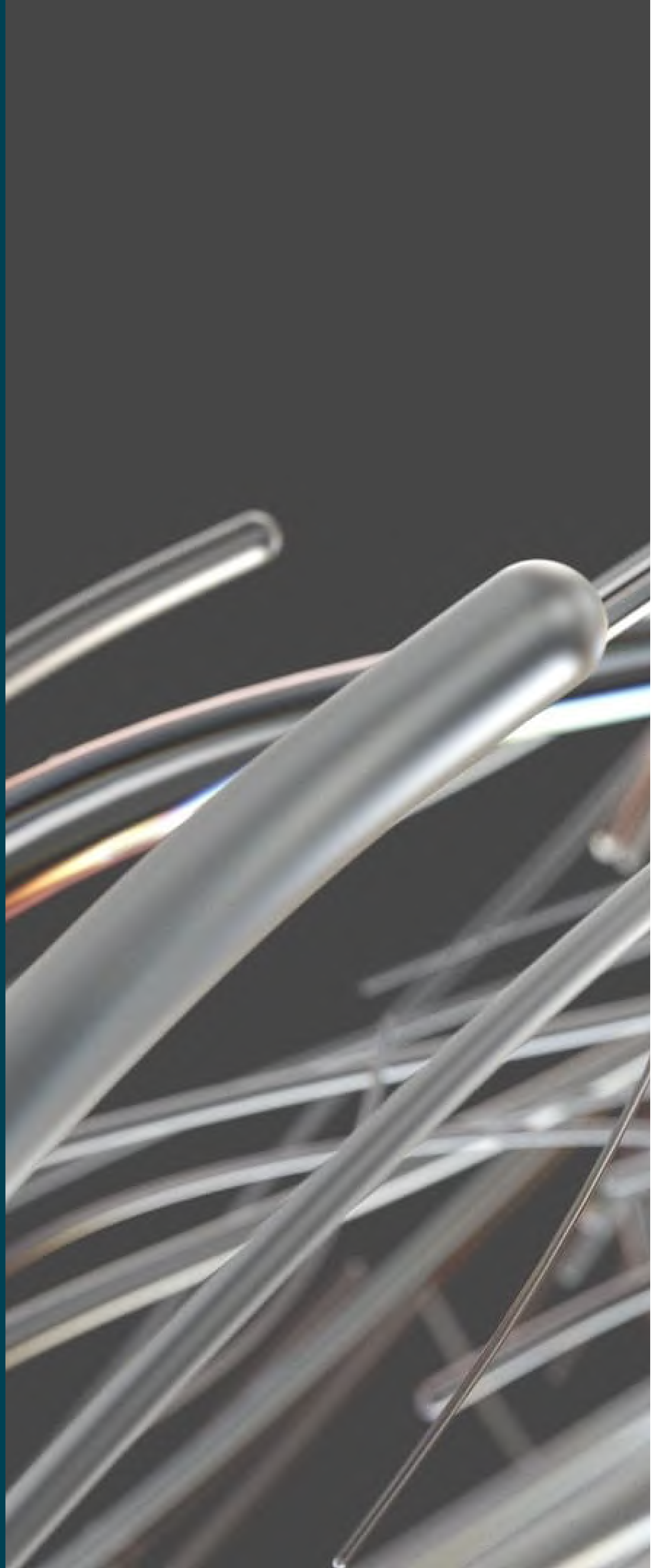
😞



Enter

## TreeLog

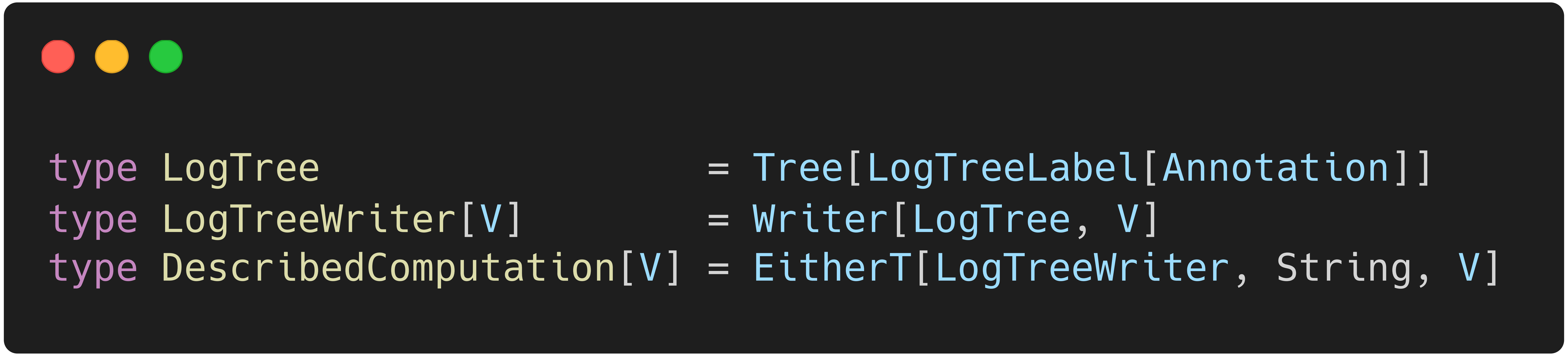
**"[..] a veritable complect of computation and description in perfect harmony"**



## TreeLog

- functional Scala library by Lance and Channing Walton
- Github: https://github.com/lancewalton/treelog
- *"TreeLog achieves this remarkable feat with a Writer monad writing to a Tree representing the hierarchical log of computation."*
- based on Cats or Scalaz
- introduces the concept of a **described computation**

# TreeLog types



```scala
type LogTree                    = Tree[LogTreeLabel[Annotation]]
type LogTreeWriter[V]           = Writer[LogTree, V]
type DescribedComputation[V] = EitherT[LogTreeWriter, String, V]
```

# SelfPub royalties with TreeLog

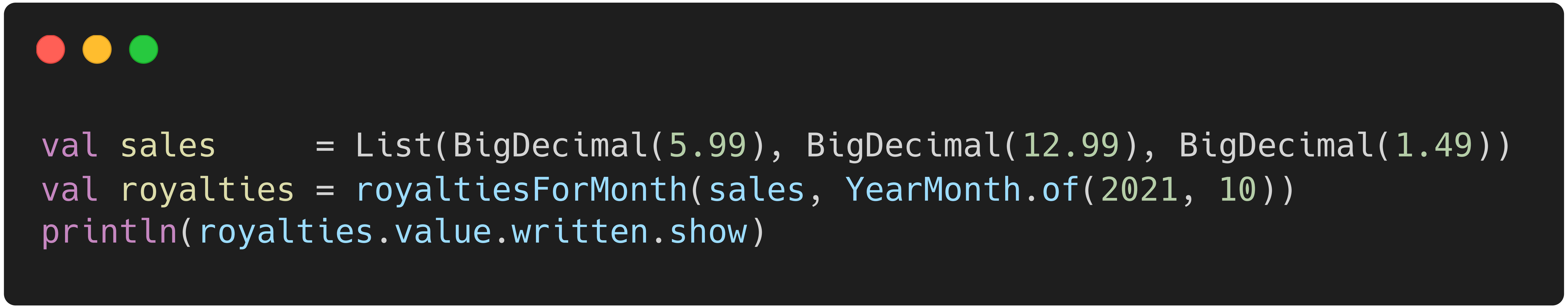
```scala
import java.time.YearMonth
import cats.implicits._
import treelog.LogTreeSyntaxWithoutAnnotations._

def royaltiesForMonth(
    sales: List[BigDecimal],
    month: YearMonth
): DescribedComputation[BigDecimal] =
  s"Revenue share for $month" ~< {
    for
      royaltyRate  <- BigDecimal(0.75) ~> (rate => s"Royalty rate = $rate")
      totalRevenue <- sales.sum ~> (revenue => s"Total revenue = $revenue")
      revenueShare <- (totalRevenue * royaltyRate) ~> (revShare =>
        s"revenue share = total revenue * royalty rate = $revShare"
      )
    yield revenueShare
  }
```

## Accessing the result value

```scala
val sales     = List(BigDecimal(5.99), BigDecimal(12.99), BigDecimal(1.49))
val royalties = royaltiesForMonth(sales, YearMonth.of(2021, 10))
royalties.value.value // : Either[String, BigDecimal] = Right(15.3525)
```

## Showing the tree log



```scala
val sales     = List(BigDecimal(5.99), BigDecimal(12.99), BigDecimal(1.49))
val royalties = royaltiesForMonth(sales, YearMonth.of(2021, 10))
println(royalties.value.written.show)
```

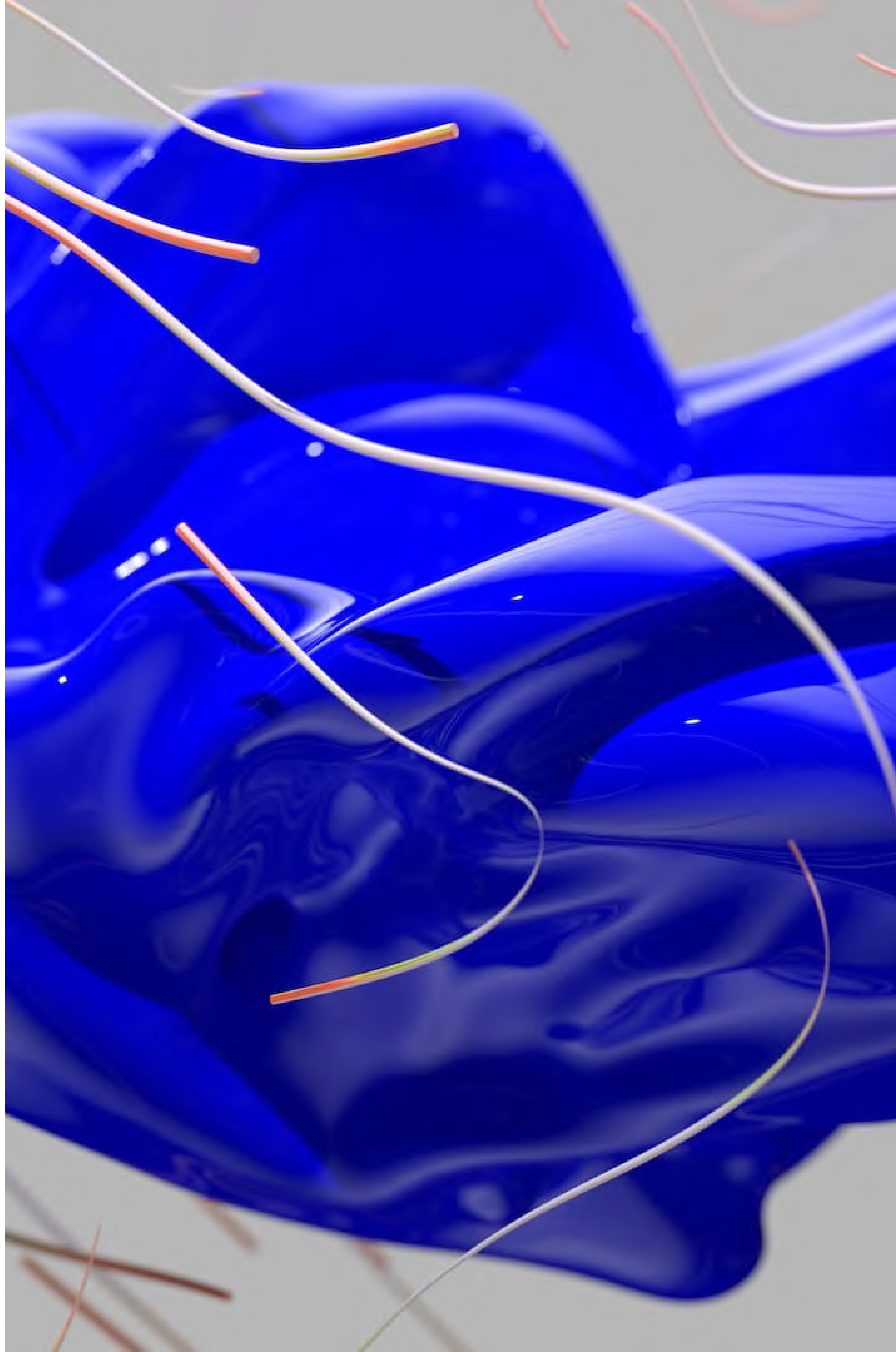
## SelfPub royalty calculation log

```
Revenue share for 2021-10
  Royalty rate = 0.75
  Total revenue = 20.47
  revenue share = total revenue * royalty rate = 15.3525
```

## Assessment

- described computation
- hierarchical log with tree structure
- Cats or Scalaz dependency
- monadic API using monad transformers
  - elegant implementation
  - obstacle in less experienced teams
- still verbose and error-prone
- referentially transparent

🙂



## An alternative approach

## Goals

- familiar API that looks similar to plain calculations
- a log that can be read and understood by stakeholders
- guaranteed consistency between log and actual calculation
- reduced surface area for calculation bugs
- ability to serialize the complete calculation

## Solution

- self-describing calculations
- calculations as data
- representing different quantity types in the type system
- typeclasses for all operators that are supported as self-describing calculations

## calclog

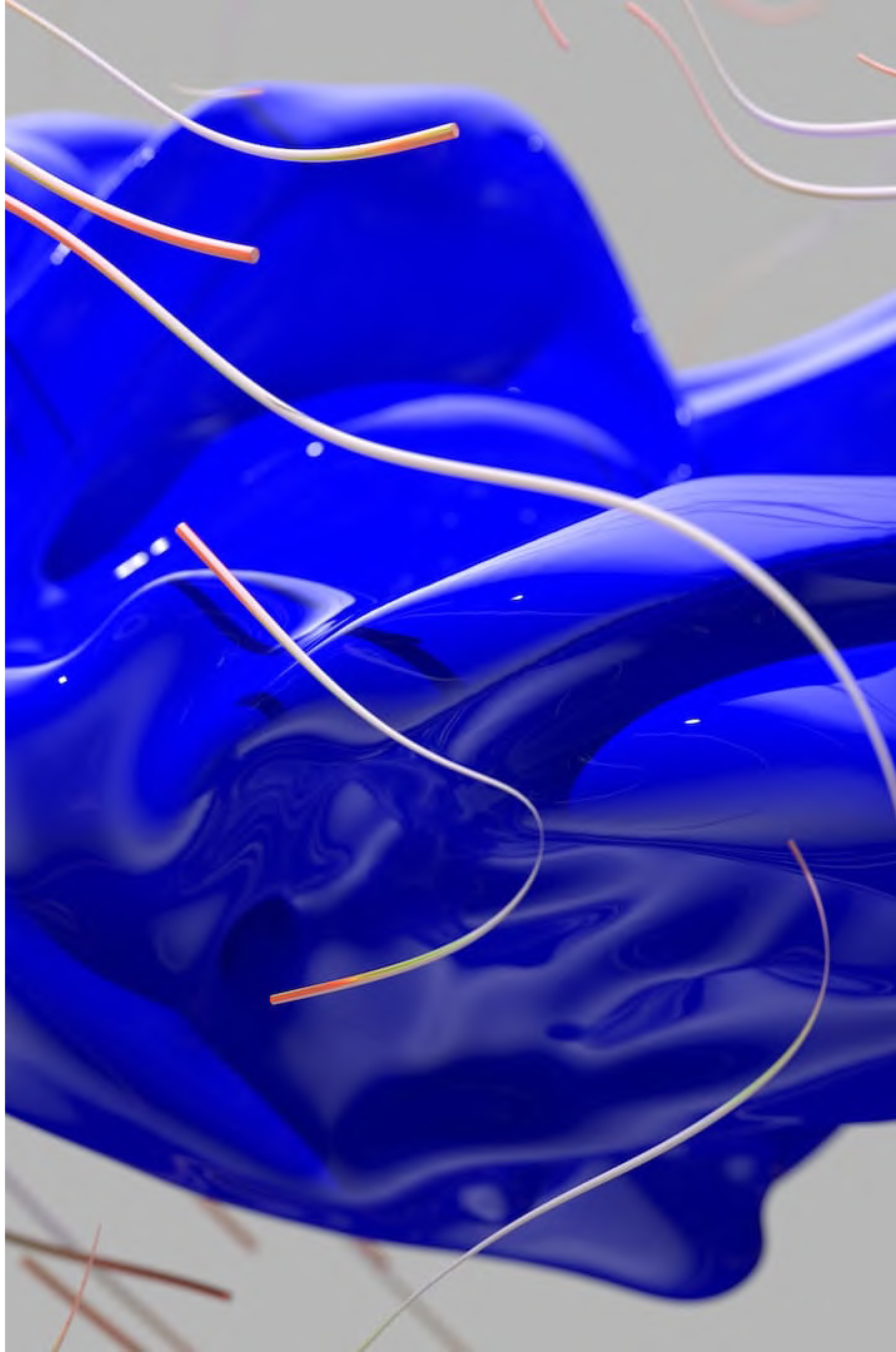
- open-source library built on the same principles
- additional goals:
    - not only open for new quantity types, but also for new operators
    - zero dependencies
- GitHub: https://github.com/dwestheide/calclog/

## calclog

it's demo time...

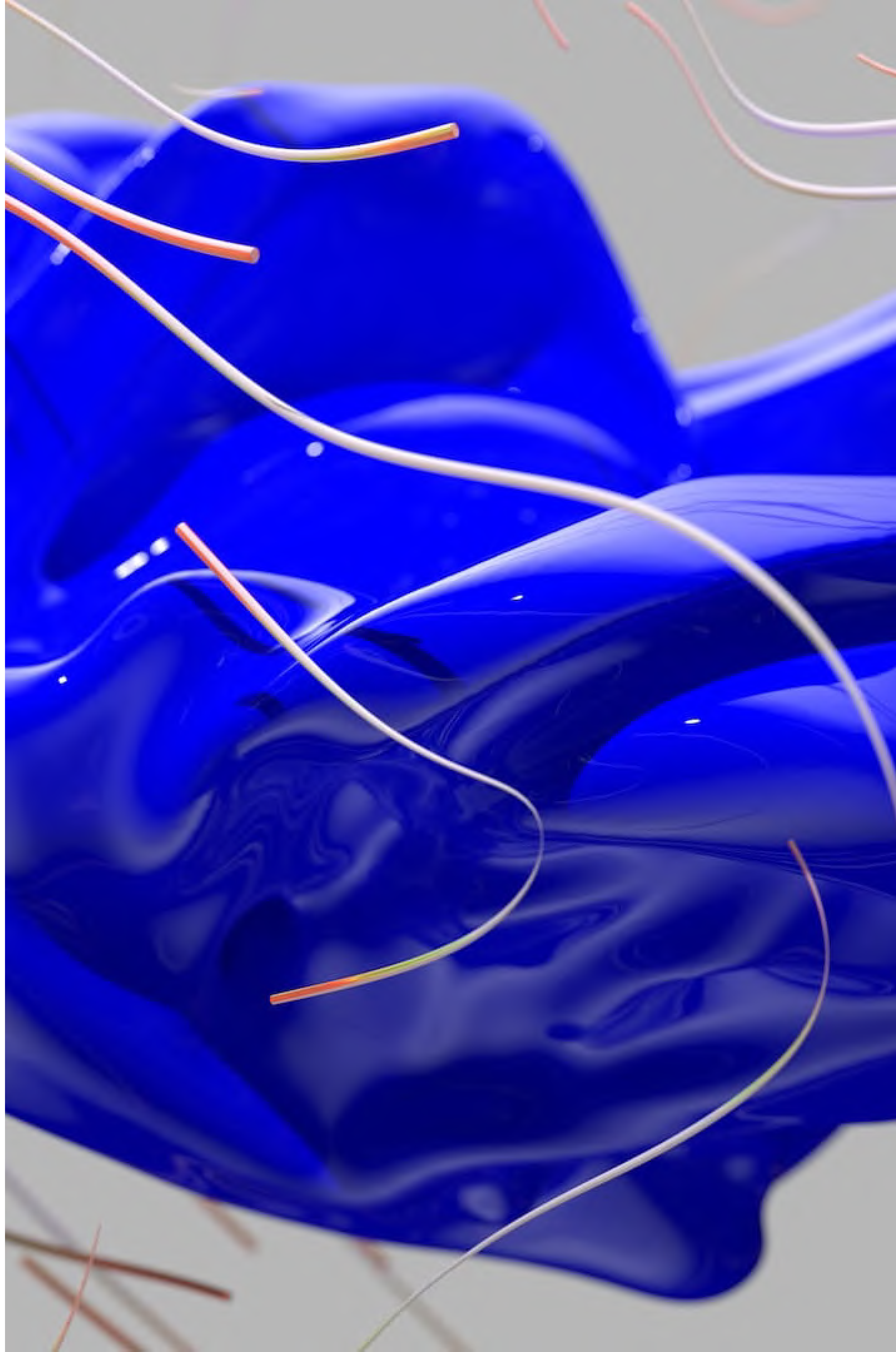
## Summary (1)

- There are a few options for creating an audit log of business-critical calculations
- Simple logging is usually not sufficient
- *treelog* is a great choice if your team is experienced with category theory and monad transformers, and they love using monads even for logging



## Summary (2)

- *calclog* ensures that your audit log is consistent with your calculation
- it literally derives the log from the calculation
- involves some ceremony to add support for your types and operators
- no monadic API
- flexible regarding formatting or serialization
- another approach worth exploring: macros



# Thank you! Questions?

INNOQ
www.innoq.com

Daniel Westheide
https://danielwestheide.com
@kaffeecoder

**innoQ Deutschland GmbH**

| | | | | | | |
|---|---|---|---|---|---|---|
| Krischerstr. 100 | Ohlauer Str. 43 | Ludwigstr. 180E | Kreuzstr. 16 | Hermannstr. 13 | Erftstr. 15-17 | Königstorgraben 11 |
| 40789 Monheim | 10999 Berlin | 63067 Offenbach | 80331 München | 20095 Hamburg | 50672 Köln | 90402 Nürnberg |
| +49 2173 333660 | | | | | | |