State Monad

Andrzej Wasowski & Zhoulai Fu

19 Sep, 2019

Today

- Concepts
- Demos

State Monad is a (relatively) simple ***design pattern*** *in functional programming.*

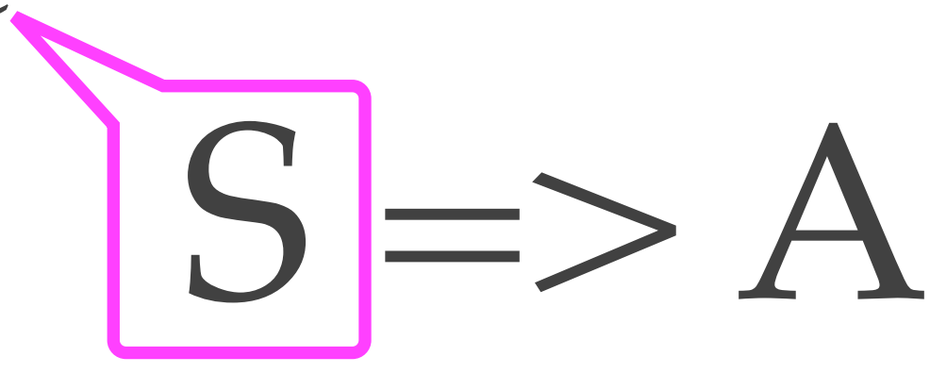
Pure Functional Programming

$$S => A$$

- Calculate an answer (A) out of the source (S)
- Scientific computation, IO…
- Scala Example: Append an element to a list produces a new list

Pure Functional Programming

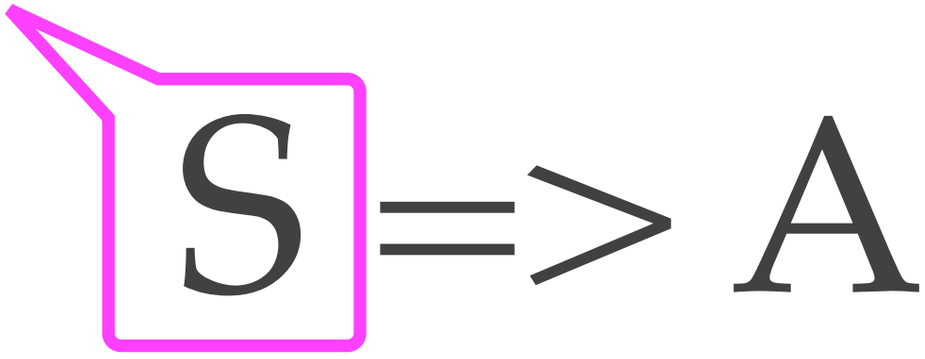
Immutable



- Calculate an answer (A) out of the source (S)
- Scientific computation, IO…
- Scala Example: Append an element to a list produces a new list

But, the real world often has to deal with changes

Immutability breaks!



- Database
- Google
- Scala example: Generating a random data

Back to pure functional programming

Immutable

$$S => A,$$

$$S$$

And, this its what characterizes a so-called state monad.

A state monad wraps S => S, A

```
trait State[S,A]{
  def transition (initial:S):(S,A)
}

object State{
  def apply[S,A](f:S => (S,A)):State[S,A] =
    new State[S,A] {
      def transition(initial:S): (S,A) = f(initial)
    }
}
```

So, a state monad does not compute;
it wraps.

Combinators can then be added on top of this structure.
To practice in your exercises.

```
trait State[S,A]{
  def transition (initial:S):(S,A)

  //combinators
  def map[B] (f: A=>B):State[S,B] = ???
  def flatMap[B] (f: A => State[S,B]):State[S,B] = ???
}
```

- map, map2, flatMap
- Useful, in cases like: derive a random Int list generator from a random Int generator
- Again, a monad does not compute; it wraps