

State Monad

Andrzej Wasowski & Zhoulai Fu

19 Sep, 2019

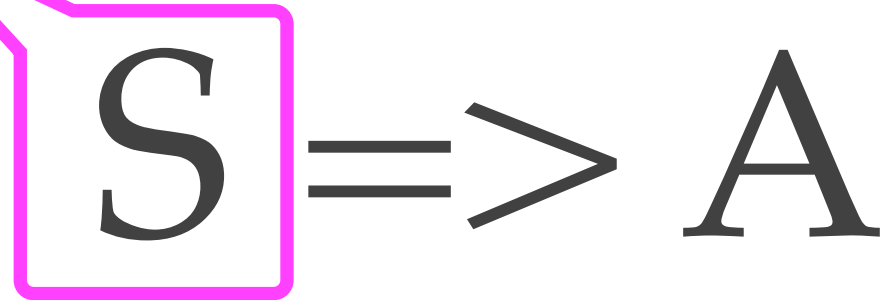
Today

- ❖ Concepts: Why and what
- ❖ Demos

*State Monad is a **design pattern** in functional programming.*

Pure Functional Programming

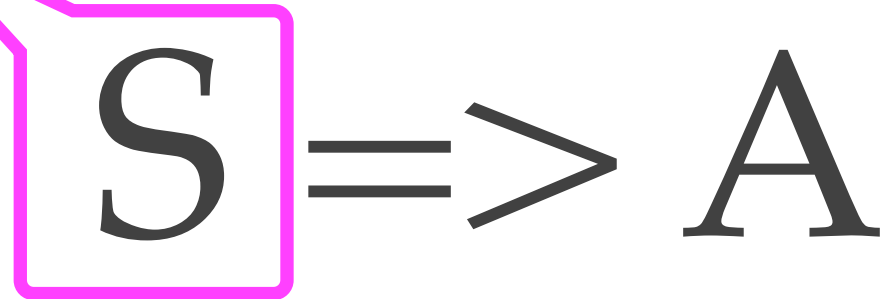
Immutable



- ❖ 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!

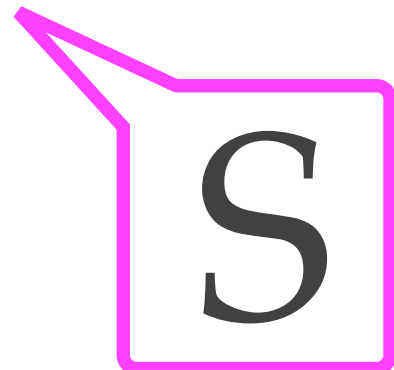
 $S = > A$

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

DEMO

For the sake of pure functional programming

Immutable

 $S \Rightarrow A,$
 S

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

A state monad wraps $S \Rightarrow S, A$

```
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)  
    }  
}
```

A state monad does not compute; it wraps.

A state monad for a random integer generator (adapted from our textbook)

```
case class MyRandom(seed:Long)

def transition(r:MyRandom): (MyRandom, Int) = {
  val newSeed = (r.seed * 0x5DEECE66DL + 0xBL) & 0xFFFFFFFFFFFFFL
  val n = (newSeed >>> 16).toInt
  (MyRandom(newSeed),n)
}
val rng=State[MyRandom, Int](transition)

println(rng.transition(MyRandom(3)))
```



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

```
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 integer list generator from a random integer generator
- ❖ Again, a monad does not compute; it wraps