State Monad

```
runState :: s -> State s a -> (a, s)
runState s (State f) = f s
```

```
main :: IO ()
main = do
     [IO ()]      putStrLn "What's your name?"
  >>
     [IO String]  name <- getLine
  >>=
     [IO ()]      putStrLn $ "Hi" ++ name
```

class Monad where

$(>>) :: ma \to mb \to mb$ (called "then", throw away the result from previous computation)

↑ e.g. IO ()

↗ borrowed by 'Promise' in JS.

$ma >> mb = ma >>= \backslash\_ \to mb$

"imperative programming uses Monads everywhere, sequencing computation with side effects by default"

```
runReader :: e -> Reader e a -> a
runReader e (Reader f) = f e
```

Parallism in Haskell is deterministic.
No matter how many threads are provided,
the end result is the same.