**Monad:**

**Pure functional programming**

* In Haskell, functions are *pure*:
  + Same argument ⟹ same result, there are no hidden dependencies
  + The returned value is the only result
    - There are no other effects that can change the result of the program
* Is this good or bad?
  + + It helps keep your code clean and modular.
  + + It's good for automated testing.
  + - Limits what you can do with functions...

**Monad**

**Example : A data Expr = Val Int | Div Expr Expr**

**Such expression can be evaluated as follow**

**eval::Expr->Int**

**eval(Val n) =n**

**eval(Div x y)= eval x `div` eval y**

**Execution on cmd: eval(Div (Val 2) (Val 0))**

**this function not take care of division by zero**

**safediv:: Int->Int->Maybe Int**

**safediv \_0 =Nothing**

**safediv n m = Just (n `div` m)**

**eval::Expr->Maybe Int**

**eval(Val n) =Just n**

**eval(Div x y)= case eval x of**

**Nothing->Nothing**

**Just n -> case eval y of**

**Nothing->Nothing**

**Just m -> safediv n m**

**Using Monad**

**Eval:: Expr ->Maybe Int**

**Eval(Val n)=return n**

**Eval(Div x y)=eval x >>=(\n->**

**Eval y >>=(\m->**

**Safediv n m))**

**Simpler form of Monad using DO**

**Eval:: Expr->Maybe Int**

**Eval(Val n) = return n**

**Eval(Div x y ) =do n <-eval x**

**m <-eval y**

**safediv n m.**

**Monads;-** Monads are very heavily used in Haskell.

The IO monad serves as the central repository for imperative language features—not only I/O and random numbers, but also mutable global variables and shared-memory synchronization.

Additional monads (with accessible hidden state) support partial functions and various container classes (lists and sets).

When coupled with lazy evaluation, monadic containers in turn provide a natural foundation for backtracking search, nondeterminism, and the functional equivalent of iterators. (In the list monad, for example, hidden state can carry the continuation needed to generate the tail of an infinite list.)

The inability to extract values from the IO monad reflects the fact that the physical world is imperative, and that a language that needs to interact with the physical world in nontrivial ways must include imperative features.

Put another way, the IO monad (unlike monads in general) is more than syntactic sugar: by hiding the state of the physical world it makes it possible to express things that could not otherwise be expressed in a functional way, provided that we are willing to enforce a sequential evaluation order. The beauty of monads is that they confine sequentiality to a relatively small fraction of the typical program, so that side effects cannot interfere with the bulk of the computation.