Contents

0.1 Monad

1. Monad is monoid over the **endofunctor**

2. endofunctor is just domain and image are both same functor

0.2 definition of Monad

:: I → T
:: T × T → T

$$\begin{aligned}
\mu &: T \times T \to T \quad \text{where } T \text{ is endofunctor} \\
\mu T &: (T \times T) \times T \to T^2 \\
T\mu &: T \times (T \times T) \to T^2 \quad \text{Associativity law in Monoid} \\
\mu T &= T\mu \quad \text{from commutative diagram} \\
T\mu\mu &= T \\
\mu T\mu &= T \\
T\mu\mu &= \mu T\mu \\
\eta &: I \to T \\
\mu_a &: T \times Ta \to Ta \\
\eta_a &: Ia \to Ta \quad \text{where } I \text{ identity endofunctor}
\end{aligned}$$

```
f(x) = x² + x³
A = B → B
C E  F = f(9) = 4
```

Matrix

$$A = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} + B = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix}$$

Matrix multiplication

1. Matrix addition

   (a) matrix division

1 Top Level Header

*** Second Level header *** second Level 2 ** second Level 3

- this is what
- this is cool
- this is cool also
- this is nice
- this ia also good
- nasdfkasdkfjaskdfj asdf
- what the fuck
- askdfj
- ksjfkaskdf
- iajsdk
- askdfjaksd
- aksdjfaksj
- askdfjaksdfdf
- asdfkasdf
- nice
- cool

name	phone	email
dog	cat	rat
→		

1.1 My Alphebat

a	b	c	d

1.2 how to solve quadratic equation

1. how to create $f(x) = x + x + 2$
2. How to find the root of quadratic equation?
   (a) How to create squares of the equation?
   (b) How to solve the equation?
   (c) How to use the quadratic equation?
   (d) How to find the root of quadratic quation?
   (e) How to the creat the quadratic equation?
   (f) How to find the root
   (g) how to
3. hwo to asdf asdkfja sdf
4. how to wha tthat
5. how to find the solution of equation?
6. How to solve the java problem?
7. How to optimization of the problem?
8. How to move the cursor around
9. asdjfaksdjfkasjdkfjasdjf Polynomial Eqation:
10. $\sin + \cos = \sin + \cos$

```
f::(Monad m)=> m a -> (a -> m b) -> m b

transpose::[[a]] -> [[a]]
transpose [] -> repeat []
transpose (x:cs) = zipWith(:) x $ transpose cs
```

1.3 What is Functor

1. The definition of **Functor** in Haskell. Functor is the type class with two methods,

   ```
   class Functor f where
   fmap::(a -> b) -> f a -> f b
   ```

   The instance of **Functor** has to satisfy following two laws:

   ```
   fmap id = id
   fmap (f . g) = (fmap f) . (fmap g)
   ```

   (a) What is the difference between Functor and Monad Monad is the subtype of Functor

1.4 What is Monad

1. What is the definition of Monad? Monad is **Monoid** over the **endofunctor**

   ```
   class Applicative m => Monad m where
     return:: a -> m a
     return = pure
     (>>=)::(Monad m) => m a -> (a -> m b) -> m b
   ```

2. When to use Monad?

1.5 What is Monoid and Monad

1. What is the difference between Monad and Monoid? There are couple **axioms** for **Monoid**

   (a) id m = m id = m

   (b) m1 m2 m3 = m1 (m2 m3)

2. The mathematic definition of **Monad**
   (a) :: I → T
   (b) :: T T → T
   (c) T is the endofunctor which means from a category to itself. (T : C → C)

3. The domain and co-domain of are both **Functor**
   - is **Functor** composition
   - e.g. `if T = m a then T  T = m (m a)`
   - e.g. `T = [] then T  T = [[]]`
   - In Haskell, type constructor is like a **Functor**

```
class Applicative f => Monad f where
  return :: a -> f a
  join f (f a) -> f a
  fmap f (a -> b) -> (f a -> f b) -- f = (a -> m b)

-- definition in GHC
class Applicative m => Monad m where
  return :: a -> m a
  (>>=)::m a -> (a -> m b) -> m b

-- f = (a -> m b)
m >>= f = join $ fmap (a -> m b) m b
m >>= f = join $ fmap f $ m b
m >>= f = join $ m (f b)
m >>= f = join $ m (m b)
```

4. Maybe is Monad

```
instance Monad Maybe where
  return Nothing = Nothing
  (>>=) (Just a) f = Just f a
```

```
addMaybe::Maybe Int -> Maybe Int -> Maybe Int
addMaybe Nothing _ = Nothing
addMaybe _ Nothing = Nothing
addMaybe (Just a) (Just b) = Just (a + b)

-- other implementation
addMaybe::Maybe Int -> Maybe Int -> Maybe Int
addMaybe m1 m2 = do
a <- m1
b <- m2
return (a + b)
```

1.6 Applicative

1. How to use Applicative
2. What is Applicative
3. What is the difference between Monad and Applicative

```
class Functor f => Applicative f where
  pure:: a -> f a
 (<*>):: f (a -> b) -> f a -> f b

class Applicative f => Monad f where
  return:: a -> f a
  (>>=)::m a -> (a -> m b) -> m b
```