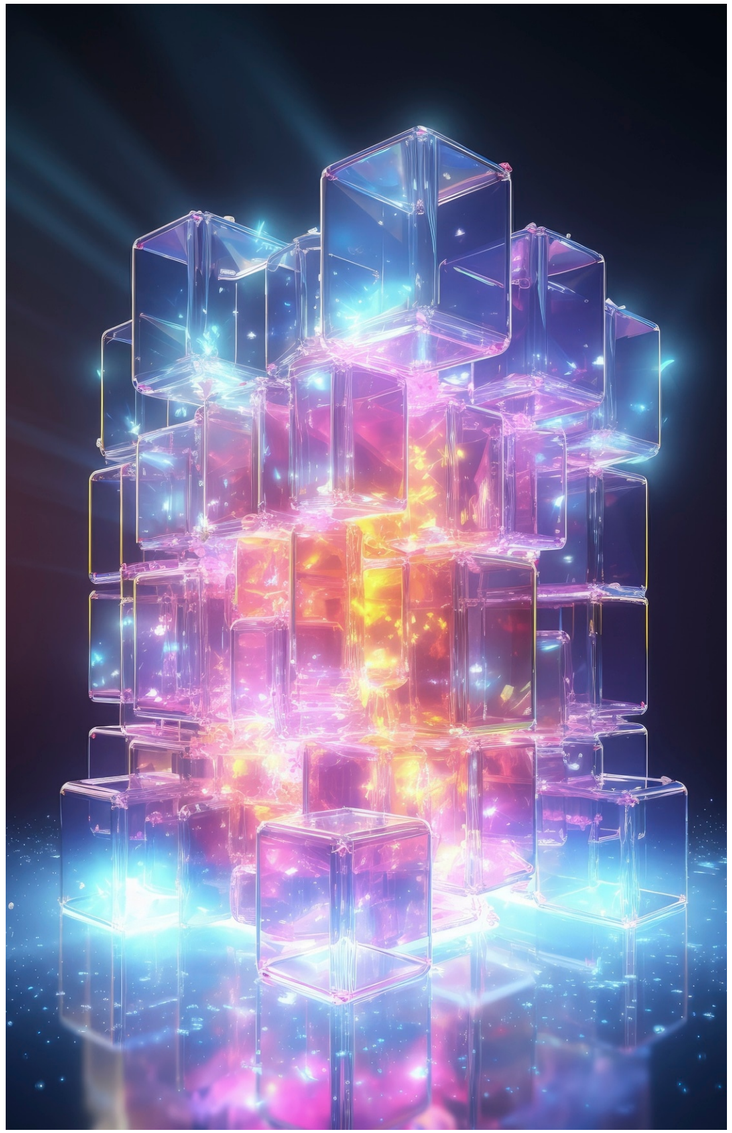

Monad Functional Programming

Monads

What are Monads?

- Structures in functional programming that encapsulate sequential computations.
- They enable handling side effects in a controlled manner and composing operations cleanly.

Key Features:

- Encapsulation: They contain a value and a series of operations defined on that value.
- Composition: Allow chaining operations easily and safely.
- Side Effect Control: Manage side effects such as I/O, exceptions, mutable state, etc.

Real-World Applications

Usos Comunes:

- Manejo de errores.
- Acceso a datos externos.
- Programación asíncrona.
- Manipulación de estado.

Ejemplo de Mónada:

- MaybeRepresenta un valor opcional.
- Ayuda a manejar casos donde un valor puede no estar presente.
- Evita el manejo explícito de nulos o excepciones.

Example

Maybe se utiliza en este ejemplo para encapsular operaciones y manejar de manera segura los casos especiales, como valores nulos o resultados inválidos, gracias a la capacidad de la mónada para encadenar operaciones y controlar el flujo de ejecución de manera efectiva.

```haskell
-- Definición de una función que divide dos números, pero puede devolver Nothing si el divisor es cero
safeDivide :: Double -> Double -> Maybe Double
safeDivide _ 0 = Nothing
safeDivide x y = Just (x / y)

-- Función que suma dos números y luego divide el resultado por un tercer número de forma segura
-- Utiliza la mónada Maybe para manejar el caso en que la división sea por cero
safeDivideExample :: Double -> Double -> Double -> Maybe Double
safeDivideExample x y z =
    Just (x + y) >>= \sumResult ->  -- Realiza la suma
    safeDivide sumResult z          -- Llama a la función de división segura

-- Ejemplo de uso
main :: IO ()
main = do
    let result = safeDivideExample 10 5 2    -- 10 + 5 = 15, 15 / 2 = 7.5
    case result of
        Just value -> putStrLn $ "Resultado: " ++ show value
        Nothing    -> putStrLn "Error"
```