Do notation

```
pure :: ∀ a f . Applicative f => a -> f a
empty :: ∀ a f . Plus f => f a

countThrows :: Int -> Array (Array Int)
countThrows n = do
  x <- 1..6          -- [1,2,3,4,5,6]
  y <- 1..6
  if x + y == n
    then pure [x,y]
    else empty

pure :: Int -> [Int]
pure x = [x]
empty :: Int -> [Int]
empty = []

func :: ∀ m . Applicative m => Plus m => Int -> Int -> Int -> m (Array Int)
func n x y = if x+y == n
               then pure [x,y]
               else empty

\n -> [1,2,3,4,5,6] |> chain (\x ->
    [1,2,3,4,5,6] |> chain (\y ->
      func n x y   -- Array (Array Int)
    )
)
```