
COMPUTATION EXPRESSIONS IN CONTEXT
FP USER GROUP + CRAFT CONF - APRIL 2016

ANDREA MAGNORSKY

@SILVERSPOON

ROUNDCRISIS.COM

workday.


COMPUTATION EXPRESSIONS

```
1: let getHtml(url:string) =
2:    let req = WebRequest.Create url
3:    let response = req.GetResponse()
4:    use streatm = response.GetResponseStream()
5:    use reader = new StreamReader(streatm)
6:
7:    reader.ReadToEnd().Length
```

```
1: let getHtmlA(url:string) =
2:   async{
3:       let req = WebRequest.Create url
4:       let! response = req.AsyncGetResponse() // ding!
5:       use streatm = response.GetResponseStream()
6:       use reader = new StreamReader(streatm)
7:       return reader.ReadToEndAsync().Length // ding!
8:       }
```

```
1: let getHtmlA(url:string) =
2:   //async{
3:       let req = WebRequest.Create url
4:       let (*! *) response = req.AsyncGetResponse()
5:       use streatm = response.GetResponseStream()
6:       use reader = new StreamReader(streatm)
7:       (*return *) reader.ReadToEndAsync().Length
8:   //    }
```

```
1: let maybe = new MaybeBuilder()
2: let addNUmbers  =
3:     maybe {
4:         let x = 12
5:         let! y = Some 11
6:         let! z = Some 30
7:         return x + y + z
8:     }
```


© Caters News Agency

type SimplestBuilder () =

```
1:    member this.Bind(x,f) =
2:      printfn "Bind begin  %A %s " (DateTime.Now.TimeOf
3:      let y = f x
4:      printfn "Bind end  %A %s " (DateTime.Now.TimeOfDa
5:      y
6:    member this.Return(x) =
7:        printfn "Return %A %s"  (DateTime.Now.TimeOfDay
8:        x
```

ASYNC

```
1: open System
2:
3: let sleepWorkflow  = async{
4:     printfn "Starting at %O" DateTime.Now.TimeOfDay
5:     do! Async.Sleep 2000
6:     printfn "Finished at %O" DateTime.Now.TimeOfDay
7:     }
8:
9: Async.RunSynchronously sleepWorkflow
```

MBRACE

```
1: let job =
2:     cloud {
3:         return sprintf
4:         "run on worker '%s' " Environment.MachineName
5:     |> runtime.CreateProcess
```

```
let makeJob i =
    cloud {
        if i % 8 = 0 then failwith "fail"
        let primes = Sieve.getPrimes 1000000
        return sprintf
        "calculated %d primes %A on machine '%s'"
            primes.Length
            primes
            Environment.MachineName
    }

let jobs2 =
    [ for i in 1 .. 10 ->
        makeJob i |> cluster.CreateProcess ]
```

example source

NOTIONS OF COMPUTATION AND MONADS (1991)

COMPREHENDING MONADS (1990)

MONADS FOR FUNCTIONAL PROGRAMMING (1995)

Some `not-useful-right-away` info

- There is a strong link between monads and category theory
- Monads have 3 monadic laws that every monad must satisfy:
  - Left identity
  - Right identity and
  - Associativity

MONADS EH?

MONOIDS

MONOIDS

- Closures $a' \rightarrow a' \rightarrow a'$ (example int -> int -> int)
- Identity $x + I = x$
- Associativity $x + (y + z) = (x + y) + z$

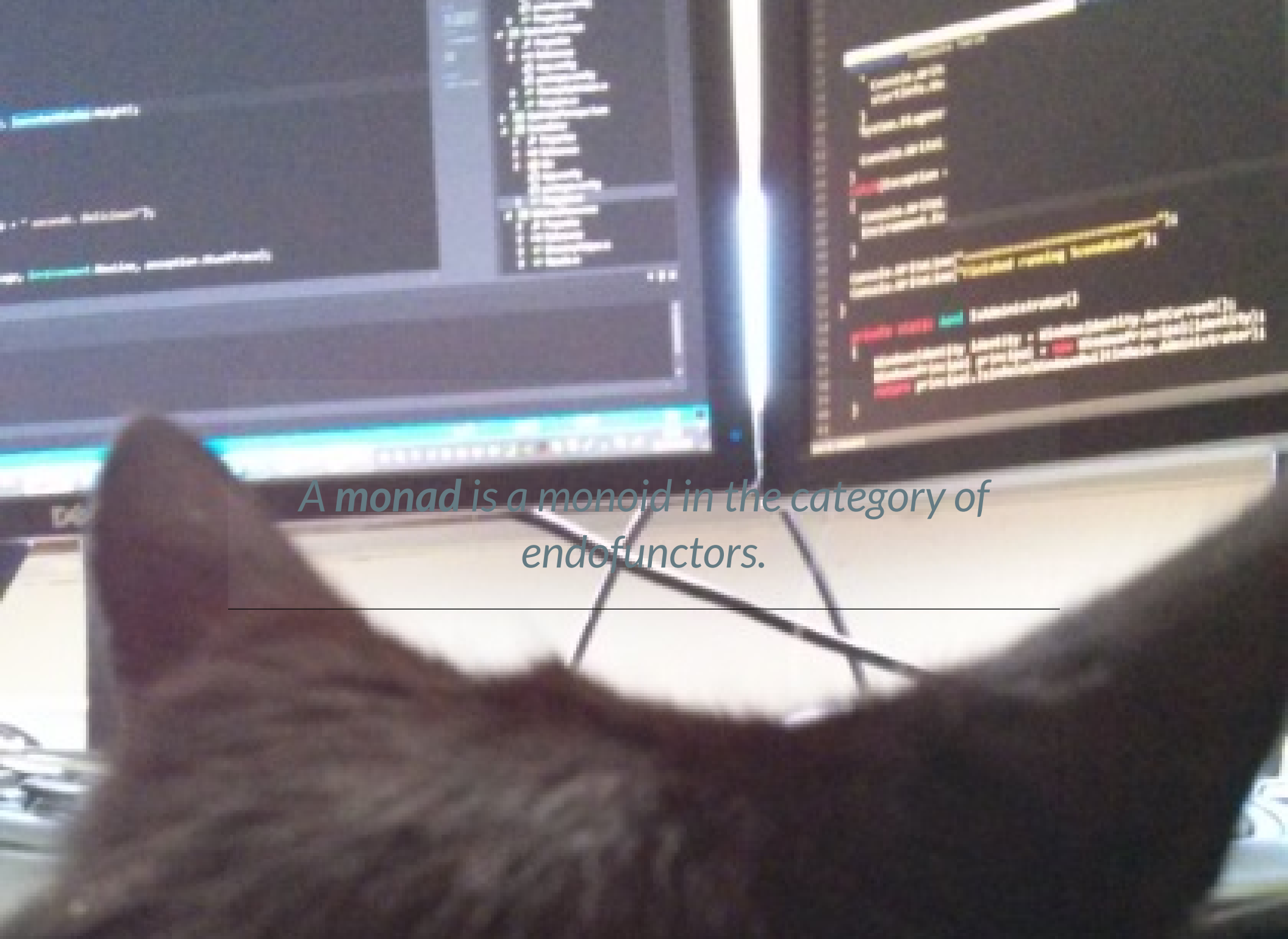
```
1: type Colour = { r: byte; g: byte; b: byte; a: byte }
2:
3: let addTwo c1 c2 = {
4:     r = c1.r + c2.r
5:     g = c1.g + c2.g
6:     b = c1.b + c2.b
7:     a = c1.a + c2.a
8: }
```

```
1: type Monoid<'a> =
2:     { neutral : 'a
3:       op : 'a -> 'a -> 'a }
```

```
1: let black = { r = 0uy; g = 0uy; b = 0uy; a = 0uy }
2:
3: let colourAdd : Monoid<Colour> = {
4:     neutral = black
5:     op = (addTwo)
6: }
```

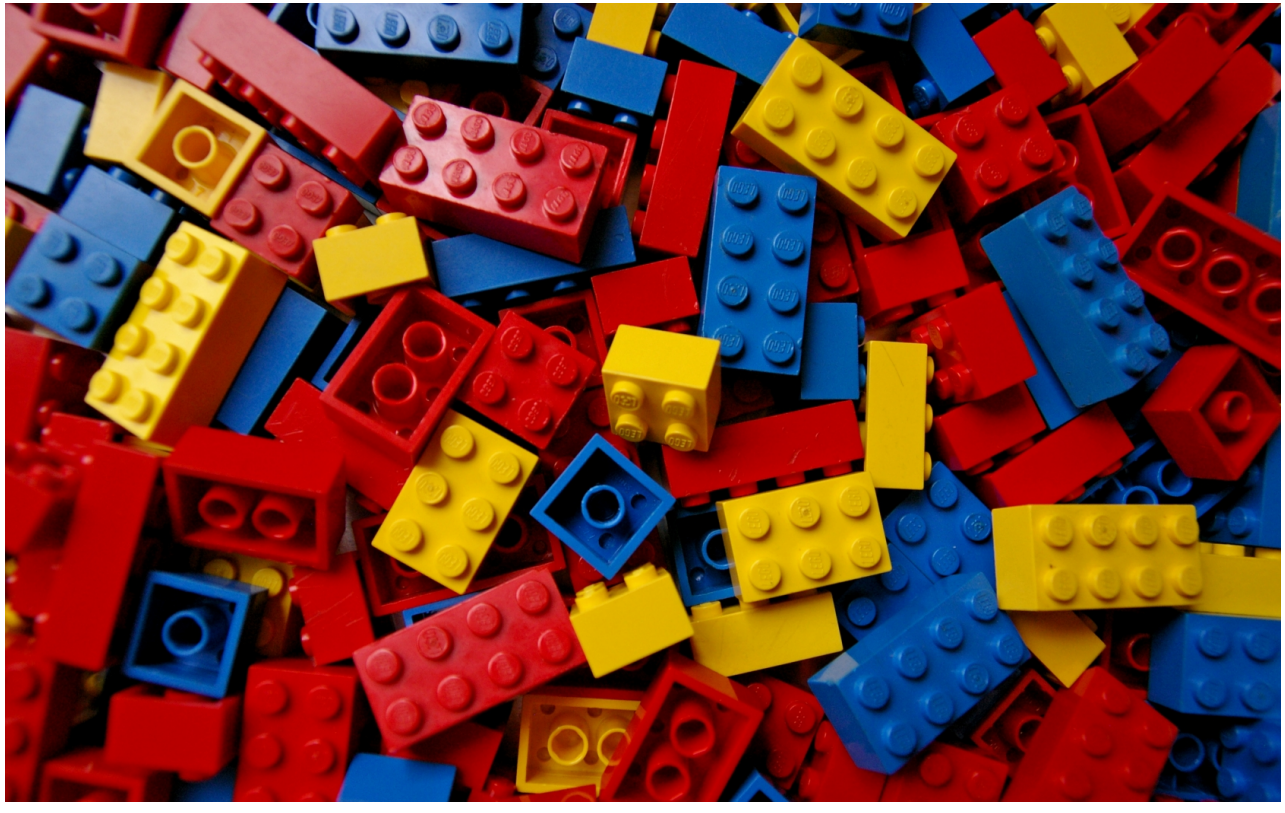
```
1: let c1 = { black with g = 254uy }
2: let c2 = { black with r = 254uy }
3:
4: let l = [ c1; c2; black ]
5:         |> List.reduce (addTwo)
```

```
 1: type T = Colour
 2:
 3: let M = colourAdd
 4: let Z = M.neutral
 5: let (++) = M.op
 6:
 7: [<Property>]
 8: let `` Z is the neutral element`` (v : T) =
 9:     Z ++ v = v && v ++ Z = v
10:
11: [<Property>]
12: let ``The op is associative`` (a : T, b : T, c : T)
13:     a ++ (b ++ c) = (a ++ b ++ c)
```


A monad is a monoid in the category of endofunctors.

*A **monad** is like a burrito.*

*Think of **monads** just like you would think about Legos.*



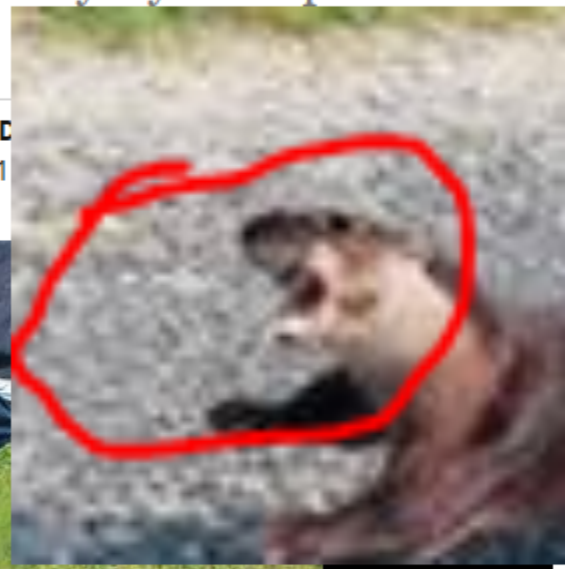
www.independent.ie/irish-news/village-terrorised-by-terminotter-26724916.html

Wednesday 15 April 2015

News Irish News

Village terrorised by Termin-otter

Lost animal turns on rescuers as they try to help it back to natural habitat



Gordon Deegan

PUBLISHED
19/04/2011

The otter chases a local in the Clare village of Tulla

IT was a rescue operation with added bite. An otter found shuffling up a village main street turned on its rescuers during a struggle to return the creature to water.

ERRORS

```
1: let division a b c d=
2:     match b with
3:     | 0 -> None
4:     | _ -> match c with
5:             | 0 -> None
6:             | _ -> match d with
7:                     | 0 -> None
8:                     | _ -> Some (((a/b)/c)/d)
```

1. EXTRACT THE CORE

```
1:  let divide a b =
2:      match b with
3:      | 0 -> None
4:      | _ -> Some (a/b )
```

2. WRITE THE BUILDER

```
1: type MaybeBuilder() =
2:     member __.Bind(value, func) =
3:         match value with
4:         | Some value -> func value
5:         | None -> None
6:     member __.Return value = Some value
```

3. PROFIT

```
1: let maybe  = MaybeBuilder()
2:
3: let divisionM a b c d=
4:     maybe{
5:         let! x = divide a b
6:         let! y = divide x c
7:         let! z = divide y d
8:         return z
9:     }
```

```
 1: [<Test>]
 2: let ``monad laws``() =
 3:     let ret (x: int) = choose.Return x
 4:     let n = sprintf "Choice : monad %s"
 5:     let inline (>>=) m f = choose.Bind(m,f)
 6:     fsCheck "left identity" <|
 7:         fun f a -> ret a >>= f = f a
 8:     fsCheck "right identity" <|
 9:         fun x -> x >>= ret = x
10:     fsCheck "associativity" <|
11:         fun f g v ->
12:             let a = (v >>= f) >>= g
13:             let b = v >>= (fun x -> f x >>= g)
14:             a = b
```

WHY LEARN THIS?

COMPUTATION EXPRESSIONS

Most monads are computation expressions,
not all computation expressions are monads

LET, FOR AND TRY .. WITH, BUT WITH A DIFFERENT SEMANTICS

MONOIDS AGAIN

```
1:    type Colour = { r : byte; g : byte; b : byte; a : b
2:
3:    let addColour c1 c2 =
4:        { r = c1.r + c2.r
5:          g = c1.g + c2.g
6:          b = c1.b + c2.b
7:          a = c1.a + c2.a }
8:
9:    let neutral = { r = 0uy; g = 0uy; b = 0uy; a = 0uy
```

```
1:    type MonoidBuilder ()=
2:      member this.Zero() = neutral
3:      member this.Combine(x, y) = addColour x  y
4:      member x.For(sequence, f) =
5:          let combine a b = x.Combine(a, f b)
6:          let Z = x.Zero()
7:          Seq.fold combine Z sequence
8:      member x.Yield (a) = a
```

```
1:    let monoid = new MonoidBuilder()
2:
3:    let monoidAdd xs= monoid {
4:          for x in xs do
5:            yield x
6:          }
```

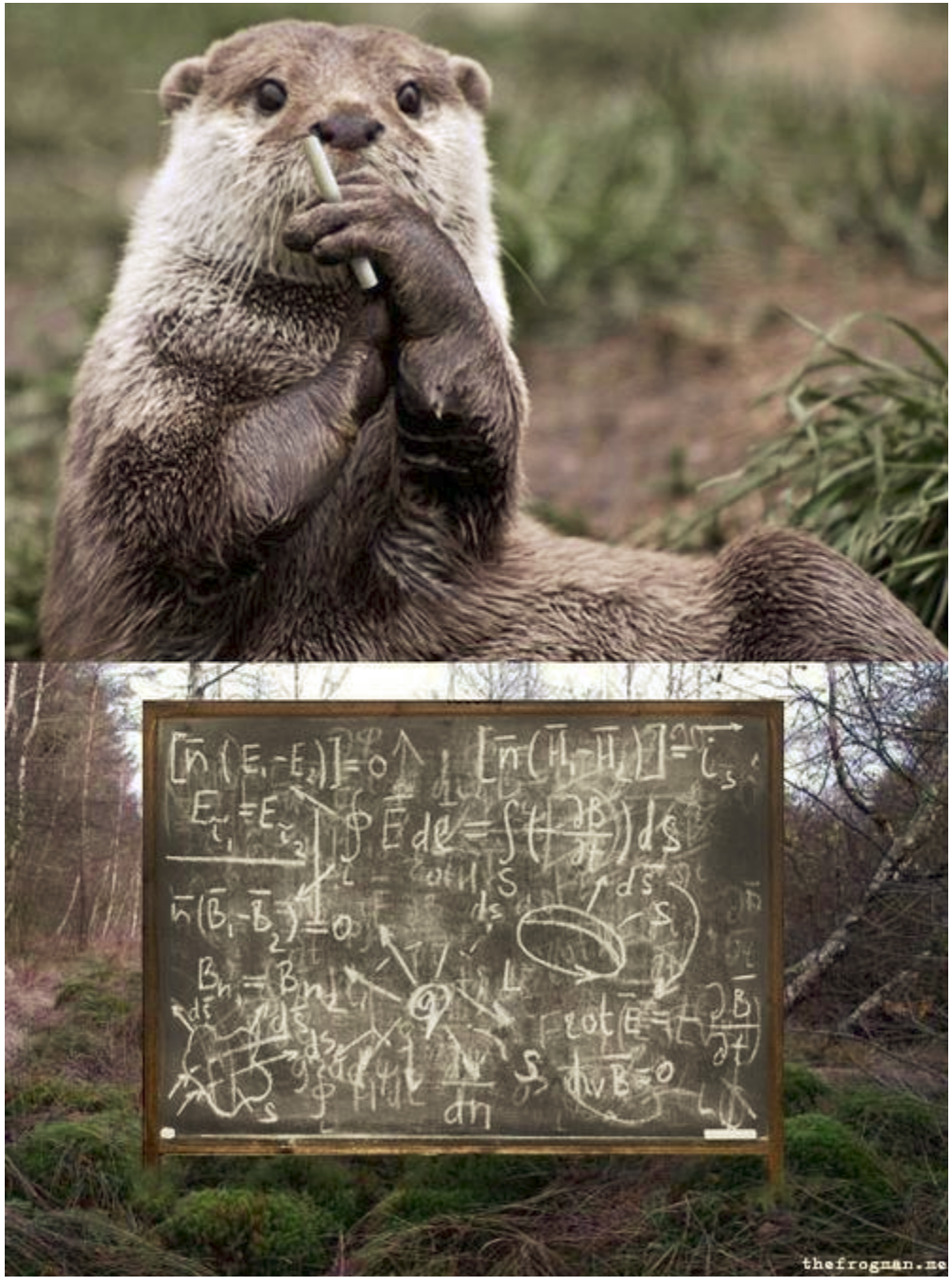
COMPUTATION EXPRESSIONS OR WORKFLOWS

Computation expressions have been available in F# since 2007 and they are fully documented in the F# language specification

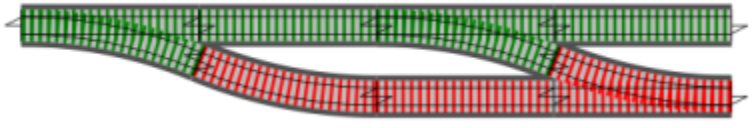
- Abstract computations
- Handling of effects

MAYBE PROBLEMS

- Slow
- Hard to debug
- Operator overuse


thefrogman.me

NEXT



- Railway Oriented Programming
- Chessie

THANKS

@SILVERSPOON

ROUNDCRISIS.COM

RESOURCES

- Abstraction, intuition, and the “monad tutorial fallacy”
- The "What are monads?" fallacy
- Beyond Foundations of F# - Workflows
- Monads, Arrows and idioms papers here.
- Why a monad is like a writing desk Video
- Understanding Monoids F# for fun and profit on monoids
- Understanding Monoids using F# From gettingsharp-er :)
- Syntax Matters: Writing abstract computations in F# paper by Tomas Petricek and Don Syme about computation expressions
- Monads for functional programming P.Wadler paper

RESOURCES

- F# language specification
- Try Joinads F# research extension for concurrent, parallel and asynchronous programming.
- Functors Applicatives and monads in pictures
- Monads explained from a maths point of view Video
- Comprehending Monads P.Wadler paper
- More freedom from side effects