

LEARN DEPENDENTLY- TYPED PROGRAMMING WITH IDRIS

A black and white photograph of a dog, possibly a pit bull mix, lying on a patterned rug. The dog is looking towards the camera. The text "WHO I AM" is overlaid in large, bold, white capital letters across the center of the image. The background is dark, and the rug has a light-colored pattern. A dark, curved object, possibly a chair or part of a car, is visible in the lower-left corner.

WHO I AM

▶ **@puffnfresh**

- ▶ **Tiny contributor to Idris (18 commits)**
- ▶ **Played with dependent types for 2 years**
 - ▶ **Been doing Idris for 6 months**

A dark, shaggy dog is sitting on a brick path partially covered in snow. The dog is facing away from the camera, looking towards the left. The background shows a snowy area with some bare branches. The overall scene is dimly lit, suggesting an overcast day.

ASSUMPTIONS

- ▶ **Small experience with Haskell**
- ▶ **Have an install of Idris (can be tricky)**

```
$ brew install ghc cabal-install  
$ cabal update  
$ cabal install alex  
$ cabal install idris
```

Try Idris

Console

Compile

```
module Main

import JavaScript

main : IO ()
main = alert "Hello world!"
```

Compile to JavaScript

Hashly

```
var __IDRRT__bigInt = (function() {
// Copyright (c) 2005 Tom Wu
// All Rights Reserved.
// See "LICENSE" for details.

// Basic JavaScript BN library - subset useful for RSA encryption.

// Bits per digit
var dbits;

// JavaScript engine analysis
var canary = 0xdeadbeefcafe;
var j_lm = ((canary&0xffffffff)===0xefcafe);

// (public) Constructor
function BigInteger(a,b,c) {
  if(a != null)
    if("number" == typeof a) this.fromNumber(a,b,c);
```

A dark, moody photograph of a person in a forest, with the word 'OUTLINE' overlaid in large white letters. The person is wearing a dark, textured jacket and is looking down. The background is a dense forest with bare trees. The overall tone is somber and atmospheric.

OUTLINE

- 1. Overview of dependent types and Idris**
- 2. Work through exercises, I lead**
- 3. Work through exercises, I help**

A dark, dimly lit room, likely an office or study area. In the foreground, a desk is visible with a black mug, a red pen, and a white trash can. A chair is partially visible on the left. The floor is dark, and there are some white markings or shadows on it. The word "MOTIVATION" is written in large, bold, white capital letters across the center of the image.

MOTIVATION

*Bad news: most software
cannot be reasoned about*


– Paul Phillips

- ▶ **Curry-Howard; programs are proofs**
 - ▶ **Let's make our proofs interesting**
- ▶ **Therefore let's use a powerful type system**



MISCONCEPTIONS

- ▶ **Idris is harder than Haskell**
- ▶ **Dependent types are hard**



**DEPENDENT TYPES
EVERYTHING IS A TERM**

```
isIdris : Bool  
isIdris = True
```

```
one : Nat  
one = if isIdris then S Z else Z
```

```
StringList : Type  
StringList = if isIdris then List Char else Int
```


- ▶ **Types and kinds are values in universes**
 - ▶ **Types can depend on values**
- ▶ ***Free* polymorphism, type constructors**

```
the : (t : Type) -> (x : t) -> t
```

```
the _ a = a
```

```
one : Nat
```

```
one = the Nat Z
```

`id1` : {t : Type} -> (x : t) -> t

`id1` {t} a = a

`id2` : (x : t) -> t

`id2` a = a

`id3` : t -> t

`id3` a = a

```
Option : Type -> Type  
Option = Maybe
```

A black and white photograph of a sloth sitting on a car seat. The sloth is positioned in the center-left of the frame, looking towards the camera. The car seat has a textured, ribbed pattern. The word "TOTALITY" is written in large, bold, white capital letters across the middle of the image, partially overlapping the sloth and the seat. The background is dark and out of focus, showing more of the car's interior.

TOTALITY

```
$ idris --total
```

```
$ idris --warnpartial
```

```
%default total
```

```
total plusOne : Nat -> Nat
```

```
plusOne Z = S Z
```

```
plusOne (S n) = S (S n)
```

*I am often asked 'how do I
implement a server as a
program in your
terminating language?'*

– Conor McBride

*I reply that I do not: a
server is a coprogram in a
language guaranteeing
liveness*

– Conor McBride

- ▶▶ **We always make progress**
- ▶ **Watch out for the totality checker!**
 - ▶ **Church-Rosser theorem**
- ▶ **Evaluation is really normalisation!**
 - ▶ **Can still do it all!**

A black dog, possibly a Labrador Retriever, is sitting on a balcony railing. The dog is looking out over a city street. In the background, there are several cars parked on the street, a brick building, and trees. The word "EQUALITY" is written in large, white, bold letters across the middle of the image.

EQUALITY

```
data (=) : a -> b -> Type where  
  refl : x = x
```

```
x : 1 = 1
```

```
x = refl
```

```
y : 1 + 1 = 2
```

```
y = refl
```

$x : \{a : \text{Nat}\} \rightarrow a - a = Z$

$x \{a=Z\} = \text{refl}$

$x \{a=S\ k\} = x \{a=k\}$

$y : \{a : \text{Nat}\} \rightarrow a - a = Z$

$y \{a\} = \text{replace } \{P = \lambda x \Rightarrow (a - x = Z)\}$

$(\text{plusZeroRightNeutral } a)$

$(\text{minusPlusZero } a\ Z)$

```
x : {a : Nat} -> a - a = Z
```

```
x = ?xproof
```

```
xproof = proof
```

```
  intros
```

```
  rewrite (minusPlusZero a Z)
```

```
  rewrite (plusZeroRightNeutral a)
```

```
  trivial
```

- ▶ *The* problem of dependent types
 - ▶ Values are unified
- ▶ Checked for syntactic/term equality



WHY IDRIS?

- ▶ **LLVM, C, Java, JS backends**
 - ▶ **FFI**
- ▶ **Lots of syntactic sugar**
 - ▶ **Tactic rewriting**
- ▶ **Allows more lying/cheating**
- ▶ **REPL, editor modes, doc tools**


```

3
4
5
6
7
8
9 Idris> :i Monad
10 Methods:
11
12 Prelude.Monad.>>= : (m a) -> (a -> m b) -> m b
13
14 Instances:
15
16 Monad PrimIO
17 Monad IO
18 Monad Maybe
19 (e : Type) -> Monad (Either e)
20 Monad List
21 (n : Nat) -> Monad (Vect n)
22 Idris> the Nat 1
23 1 : Nat
24 Idris> :t filter
25 Prelude.List.filter : (a -> Bool) -> (List a) -> List a
26 Prelude.Vect.filter : (a -> Bool) -> (Vect n a) -> (p ** Vect p a)
27 Idris>

```

Version 0.9.9.2
<http://www.idris-lang.org/>
Type :? for help

```

52 total minus : Nat -> Nat -> Nat
53 minus Z right = Z
54 minus left Z = left
55 minus (S left) (S right) = minus left right
56
57 total power : Nat -> Nat -> Nat
58 power base Z = S Z
59 power base (S exp) = mult base $ power base exp
60
61 hyper : Nat -> Nat -> Nat -> Nat
62 hyper Z a b = S b
63 hyper (S Z) a Z = a
64 hyper (S(S Z)) a Z = Z
65 hyper n a Z = S Z
66 hyper (S pn) a (S pb) = hyper pn a (hyper (S pn) a pb)
67
68
69 -----
70 -- Comparisons
71 -----
72
73 data LTE : Nat -> Nat -> Type where
74 lteZero : LTE Z right
75 lteSucc : LTE left right -> LTE (S left) (S right)
76

```

```

151 , prim_indexB64x2 x (prim_truncBigInt_B32 1)
152 )
153
154 instance Show Bits64x2 where
155 show x =
156 case viewB64x2 x of
157 (a, b) =>
158 "<" ++ prim_toStrB64 a
159 ++ ", " ++ prim_toStrB64 b
160 ++ ">"
161
162 instance (Show a, Show b) => Show (a, b) where
163 show (x, y) = "(" ++ show x ++ ", " ++ show y ++ ")"
164
165 instance Show a => Show (List a) where
166 show xs = "[" ++ show' "" xs ++ "]" where
167 show' acc [] = acc
168 show' acc [x] = acc ++ show x
169 show' acc (x :: xs) = show' (acc ++ show x ++ ", ") xs
170
171 instance Show a => Show (Vect n a) where
172 show xs = "[" ++ show' xs ++ "]" where
173 show' : Vect n a -> String
174 show' [] = ""
175 show' [x] = show x
176 show' (x :: xs) = show x ++ ", " ++ show' xs
177
178 instance Show a => Show (Maybe a) where
179 show Nothing = "Nothing"
180 show (Just x) = "Just " ++ show x
181
182 ---- Functor instances
183
184 instance Functor PrimIO where
185 map f io = prim_io_bind io (prim_io_return . f)
186
187 instance Functor IO where
188 map f io = io_bind io (\b => io_return (f b))
189
190 instance Functor Maybe where
191 map f (Just x) = Just (f x)
192 map f Nothing = Nothing
193
194 instance Functor (Either e) where
195 map f (Left l) = Left l
196 map f (Right r) = Right (f r)
197
198 ---- Applicative instances
199
200 instance Applicative PrimIO where
201 pure = prim_io_return

```

```
data Parity : Nat -> Type where
  Even : (n : Nat) -> Parity (n + n)
  Odd  : (n : Nat) -> Parity (S (n + n))
```



A black cat is curled up on the keyboard of a silver laptop. The laptop screen is open and shows a blank white page. The text "HOW TO IDRIS" is overlaid in large, bold, white capital letters across the center of the image. The background is a desk with various items like cables and a mouse.

HOW TO IDRIS

- ▶ **Idris Tutorial**
- ▶ **Idris library docs**
- ▶ **Idris library source**
- ▶ **Beginning Haskell: a Project Based Approach**

A fluffy black dog is sitting at a desk, looking directly at the camera. In the background, there is a computer monitor displaying some code or text. In the foreground, a black Microsoft keyboard is visible. The scene is dimly lit, and the overall tone is somewhat somber.

LET'S GO

- ▶ **printf**
- ▶ **Equality proofs**
- ▶ **Verified algebra**
- ▶ **Vector filtering**

<http://goo.gl/gfCJne>