Haskell is **faster than C++**, more **concise than Perl**, more **regular than Python**, more **flexible than Ruby**, more **typeful than C#**, more **robust than Java**, and has absolutely **nothing in common with PHP**.

-Audrey Tang (2005)

"SQL, Lisp, and Haskell are the only programming languages that I've seen where **one spends more time thinking than typing**."

- Philip Greenspun

Laziness is next to Godliness

-from the #haskell IRC channel

What is Haskell?

- Polymorphic: Values can have different types
  - Parametric polymorphism = id :: a -> a
  - Ad-hoc polymorphism = (+)
- Statically Typed: Type checking done at compile time
  - Similar to Java and C
- Lazy: Expressions are not evaluated until results are needed
  - Infinite lists and curried functions
- Pure: Referentially transparent
  - Computations yield the same result on each invocation
  - For example, if y = f x then g h y y is same as g h (f x) (f x)
  - Python -> def (x): return x + 1 ----PURE, def (x): return x + y ----NOT PURE

Laziness

- Unevaluated expressions are stored as *thunks*
  - Thunks are values that are yet to be evaluated
    - (&&) = A && _, if A is False, then False, else _
    - take 5 [1..], take 1 [2,undefined]
- Curried Functions and partial function application
  - Every haskell function takes one argument!
  - Function application is left associative
    - add :: int -> int -> int ⇒ add :: int -> (int -> int)
- Example: QuickSort
  - Find minimum of list with quickSort

What’s wrong with Laziness?

- Hard to debug
- Unpredictable Resource consumption (huge thunks on heap)
- Undefined order of execution
- Bigger burden on the compiler

Purity - How can code be impure?

- Global variables
- Mutable state
- Input/Output
- Network Connections

These are necessary operations, handled by Haskell through:

- Monads
  - Composable computation descriptions

Monad - A **VERY** general overview

- Represent a sequence of steps
- Controlled execution order
- Used to embed impure code into pure code

```
main = do
    putStrLn “Your name? ”
    n <- getLine
    putStrLn (“Hello ,” ++ n)
```

Purity - Pros and Cons

Pros

- Simplifies reasoning about code
- Compiler optimization

Cons

- Lot’s of things programmer’s do are impure...

Syntax

Haskell	ML
1 : 2 : 3 : [ ]	1 :: 2 :: 3 :: [ ]
id x = x	fun id x = x
True	true
[1, 2] ++ [3, 4] = [1,2,3,4]	[1,2] @ [3,4] = [1,2,3,4]
(x:xs)	x::xs

More Syntax

Haskell

```
len [ ] = 0

len (_ : xs) = len xs + 1
```

map, foldl, foldr demo

ML

```
fun len [ ] = 0

        | len(_::xs) = len x + 1
```

Conclusion??

Further readings:

- www.haskell.org & wiki.haskell.org – Official homepage and wiki
- learnyouahaskell.com – A beginner/intermediate tutorial
- http://haskellbook.com - A beginner/intermediate tutorial
- book.realworldhaskell.org – A intermediate/advanced tutorial
- Pete Johnson - Taught me Haskell, changed my life

```
myMap :: (a -> b) -> [a] -> [b]

myMap f = foldr((:).f) []
```