Programación Funcional en Haskell

Agustín Ramos Fonseca

#sgvirtual

¿Qué es Haskell?

Un lenguaje de programación...

Un lenguaje de programación...

- Funcional.

Un lenguaje de programación...

- Funcional.

  Las funciones son construcciones de primera clase.

Un lenguaje de programación...

- Funcional.

  Las funciones son construcciones de primera clase.

- Puro.

Un lenguaje de programación...

- Funcional.

  Las funciones son construcciones de primera clase.

- Puro.

  Las funciones no tienen efectos colaterales.

Un lenguaje de programación...

- Funcional.

  Las funciones son construcciones de primera clase.

- Puro.

  Las funciones no tienen efectos colaterales.

- De evaluación perezosa.

Un lenguaje de programación...

- Funcional.

  Las funciones son construcciones de primera clase.

- Puro.

  Las funciones no tienen efectos colaterales.

- De evaluación perezosa.

  Las expresiones se evalúan hasta que es requerido.

Un lenguaje de programación...

- Funcional.

  Las funciones son construcciones de primera clase.

- Puro.

  Las funciones no tienen efectos colaterales.

- De evaluación perezosa.

  Las expresiones se evalúan hasta que es requerido.

- De tipado estático.

Un lenguaje de programación...

- Funcional.

  Las funciones son construcciones de primera clase.

- Puro.

  Las funciones no tienen efectos colaterales.

- De evaluación perezosa.

  Las expresiones se evalúan hasta que es requerido.

- De tipado estático.

  Con inferencia de tipos.

Un poco de Historia

Un poco de Historia

LISP
(John McCarthy)

1958

Un poco de Historia

LISP
(John McCarthy)

ML y
Hope

1958

1970's

Un poco de Historia

LISP
(John McCarthy)

ML y
Hope

Miranda™
A Non-strict Polymorphic Functional Language

1958 1970's 1985

Un poco de Historia

LISP
(John McCarthy)

ML y
Hope

1958 1970's 1985

FPCA '87

Un poco de Historia

LISP
(John McCarthy)

1958

ML y
Hope

1970's

1985

FPCA '87

¡Necesitamos Haskell!

Un poco de Historia

LISP
(John McCarthy)

1958

ML y
Hope

1970's

Miranda™
A Non-strict Polymorphic Functional Language

1985

FPCA '87

Haskell 1.0

1990

¡Necesitamos Haskell!

Un poco de Historia

LISP
(John McCarthy)

1958

ML y
Hope

1970's

Miranda™
A Non-strict Polymorphic Functional Language

1985

FPCA '87

1990

Haskell 1.0

Haskell '98

1998

¡Necesitamos Haskell!

Un poco de Historia

LISP
(John McCarthy)

1958

ML y
Hope

1970's

Miranda™
A Non-strict Polymorphic Functional Language

1985

FPCA '87

Haskell 1.0

1990

Haskell '98

1998

Haskell '98
Revised Report

2003

¡Necesitamos Haskell!

Un poco de Historia

LISP
(John McCarthy)

1958

ML y
Hope

1970's

1985

FPCA '87

Haskell 1.0

1990

Haskell '98

1998

Haskell '98
Revised Report

2003

Haskell 2010

2010

¡Necesitamos Haskell!

Motivación de los Sistemas de Tipado Estático

```
import foo

def bar(baz, quux):
    x = baz.do_something()
    y = quux.do_something_else()
    return x + y
```

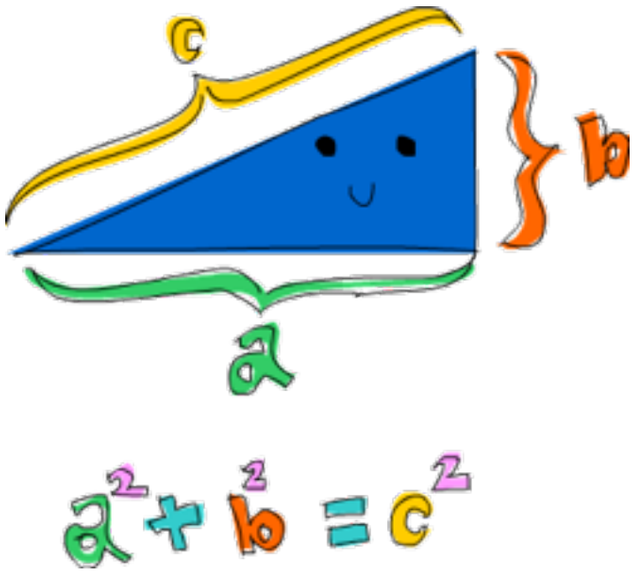
Motivación de los Sistemas de Tipado Estático

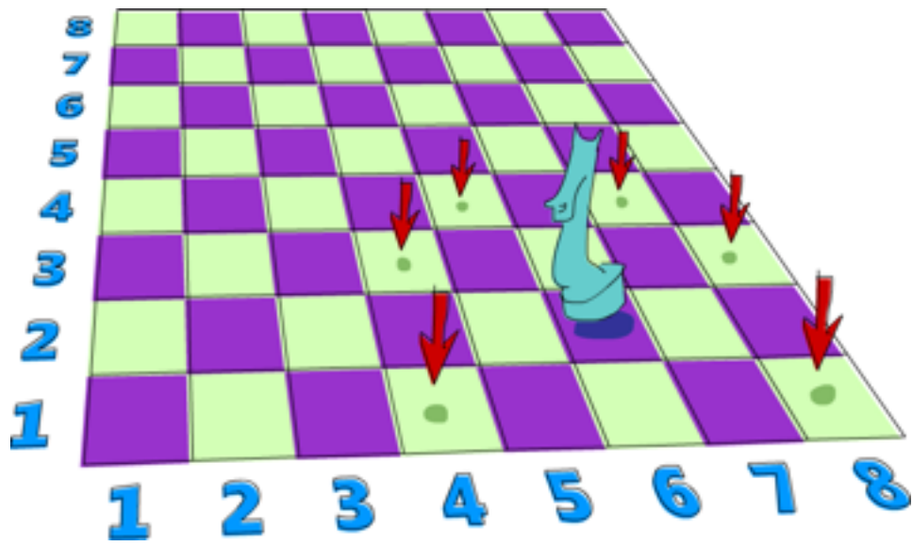
```
# Los imports pueden tener efectos colaterales,
# como escribir en un archivo... o lanzar un misil.
import foo

def bar(baz, quux):
    # baz puede ser cualquier objeto
    # y podría no tener el método do_something()
    x = baz.do_something()
    # quux puede ser cualquier objeto y podría
    # no ser capaz de ejecutaro do_something_else()
    y = quux.do_something_else()
    # El operador '+' podría estar sobrecargado y
    # hacer algo impredecible o simplemente fallar.
    return x + y
```

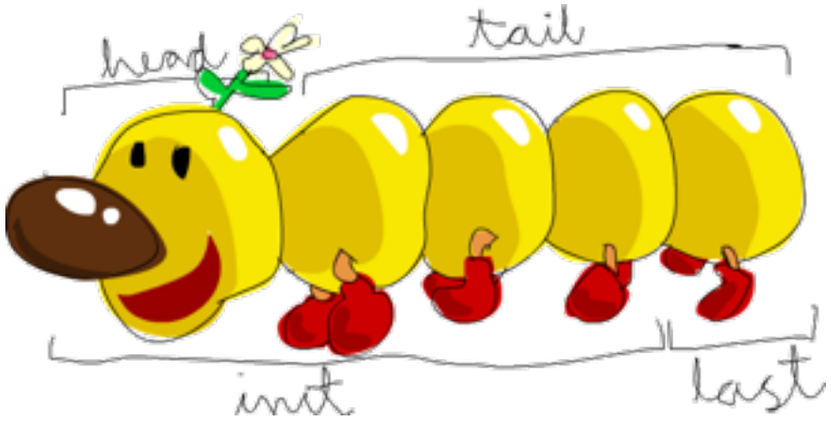
Características

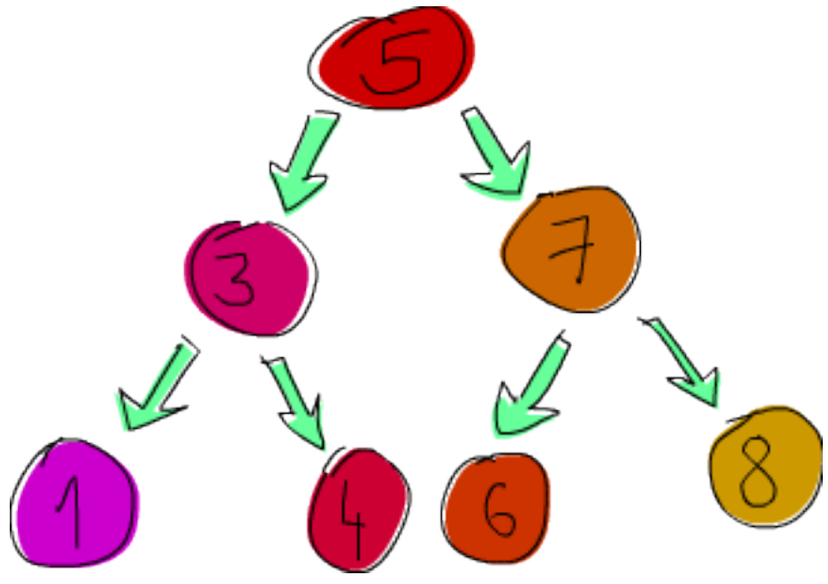
- List comprehensions.
- Pattern matching.
- Currying
- Composición de funciones.
- Infinite data structures.
- Agebraic Data Types
- Type classes.
- Applicative Functors
- Monads





Ejemplos





Tipos

```
Prelude > :t 2

2 :: Num a => a

Prelude > :t [1,2,3]

[1,2,3] :: Num t => [t]

Prelude > :t ["hola","mundo"]

["hola","mundo"] :: [[Char]]

Prelude > :t head

head :: [a] -> a

Prelude > :t [even,odd]

[even,odd] :: Integral a => [a -> Bool]
```

Listas

Lista de Num

[1,2,3,4,5]

Lista (rango) de Char

['a'..'z']

Lista de strings [Char]

["hola","mundo"]

Funciones sobre listas (1/2)

```
Prelude > head [1,2,3,4,5]

1

Prelude > tail [1..5]

[2,3,4,5]

Prelude > [1..5] !! 3

4

Prelude > take 2 [1..5]

[1,2]

Prelude > filter even [1..5]

[2,4]
```

Funciones sobre listas (2/2)

```
Prelude > map (\x -> x*x) [1..5]

[1,4,9,16,25]

Prelude > zip [1..5] ['a'..'e']

[(1,'a'),(2,'b'),(3,'c'),(4,'d'),(5,'e')]

Prelude > zipWith (\x y -> x + y) [1..5] [11..15]

[12,14,16,18,20]

Prelude > zipWith (+) [1..5] [11..15]

[12,14,16,18,20]

Prelude > foldl (+) 0 [1..5]

15
```

Funciones

Area de un círculo

```
circleArea r = pi * r^ 0
```

Secuencia de Fibonacci

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Funciones

QuickSort

```
quicksort :: Ord a => [a] -> [a]
quicksort []     = []
quicksort (p:xs) = (quicksort lesser) ++ [p] ++ (quicksort greater)
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

Funciones

```
Prelude > 1 + 2
3
Prelude > (+) 1 2
3
Prelude > (*) 2 3
6
Prelude > div 10 5
2
Prelude > 10 `div` 5
2
Prelude > ($) even 4
True
Prelude > even $ 4
True
```

List comprehensions

```
Prelude > [ x | x <- [1..10] , even x ]
[2,4,6,8,10]

Prelude > [ (x,y) | x <- [1..5], y <- [1..5] , even x, odd y ]
[(2,1),(2,3),(2,5),(4,1),(4,3),(4,5)]

Prelude > map ($3) $ map (*) [1..10]  -- La "tabla" del 3
[3,6,9,12,15,18,21,24,27,30]

Prelude > [ map ($x) $ map (*) [1..10] | x <-[2..9] ]   --Las "tablas" del 2 al 9
[[2,4,6,8,10,12,14,16,18,20],[3,6,9,12,15,18,21,24,27,30],[4,8,12,16,20,24,28,32,36,40],
[5,10,15,20,25,30,35,40,45,50],[6,12,18,24,30,36,42,48,54,60],
[7,14,21,28,35,42,49,56,63,70],[8,16,24,32,40,48,56,64,72,80],
[9,18,27,36,45,54,63,72,81,90]]
```

Pattern Matching

```
Prelude > let (x:xs) = [1..5]
Prelude > x
1
Prelude > xs
[2,3,4,5]
Prelude > let (x:y:xs) = [1..5]
Prelude > y
2
Prelude > let Just x = Just 5
Prelude > x
5
Prelude > let (a,b,c) = (3,"hola",[1,2,3])
Prelude > b
“hola”
```

Currying

```
Prelude > let sum x y z = x + y + z
Prelude > sum3 1 2 3
6
Prelude > :t sum
sum :: Num a => a -> a -> a -> a
Prelude > :t sum 1
sum 1 :: Num a => a -> a -> a
Prelude > :t sum 1 2
sum 1 2 :: Num a => a -> a
Prelude > let sum2and3 = sum 2 3
Prelude > sum2and3 5
10
Prelude > sum 2 4 $ 5
11
```

Composición de Funciones

```
Prelude > let foo = (*2).(+5)
Prelude > :t foo
foo :: Integer -> Integer
Prelude > foo 4
18
Prelude >
Prelude >
Prelude >

Prelude > let strToUpper = map toUpper
Prelude > let strToLower = map toLower
Prelude > map ($ "WaZoWzky") $  map ((reverse).) [strToUpper, strToLower]
["YKZWOZAW","ykzwozaw"]
```

Evaluación Perezosa

Ejemplo: El triángulo de Pascal

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

Algebraic Data Types

```
Prelude > data Toy = Buzz | Woody | Rex | Hamm deriving (Eq, Ord, Show)
Prelude > let x = Rex
Prelude > x
Rex

Prelude > data Maybe a = Nothing | Just a
Prelude > :t Just  5
Just  5 :: Num a => Maybe a
```

Algebraic Data Types

```
type Radius = Float
type Side   = Float
type Vertex = (Float, Float)

data Shape = Rectangle Side Side |
             Ellipse Radius Radius |
             RtTriangle Side Side |
              Polygon [Vertex]
              deriving Show
```

Algebraic Data Types

```
data Car = Car { company :: String
               , model :: String
               , year :: Int
               } deriving (Show)
```

```
Prelude > let stang = Car {company="Ford", model="Mustang", year=1967}
Prelude > company stang
"Ford"
Prelude > year stang
1967
```

Evaluación Perezosa

Solución:

```
pascal n = take n $
    iterate (\r -> zipWith (+) ([0] ++ r) (r ++ [0])) [1]
```

¿Por qué aprender Haskell?

¿Por qué aprender Haskell?

"My personal reason for using Haskell is that I have found that I write more bug-free code in less time using Haskell than any other language. I also find it very readable and extensible."

- Hal Daumé III

¿Por qué aprender Haskell?

"Learning Haskell may not get you a job programming in Haskell, but as Paul Graham postulates, it may still get you a job. Personally, I find that irrelevant. Learning the language has proven to be fun and rewarding and I plan to continue my Haskell adventures into 2012"

- Sean Voisen

¿Por qué aprender Haskell?

¿Por qué aprender Haskell?

- Incorpora conceptos de programación de un nivel de abstracción superior a los existentes en otros lenguajes.

¿Por qué aprender Haskell?

- Incorpora conceptos de programación de un nivel de abstracción superior a los existentes en otros lenguajes.

Mayor abstracción => Mayor poder

¿Por qué aprender Haskell?

- Incorpora conceptos de programación de un nivel de abstracción superior a los existentes en otros lenguajes.

Mayor abstracción => Mayor poder

- Los conceptos y estilo de programación aprendidos, son aplicables a otros lenguajes y tecnologías.

¿Por qué aprender Haskell?

- Incorpora conceptos de programación de un nivel de abstracción superior a los existentes en otros lenguajes.

Mayor abstracción => Mayor poder

- Los conceptos y estilo de programación aprendidos, son aplicables a otros lenguajes y tecnologías.

Te convierte en mejor programador

¿Por qué aprender Haskell?

- Incorpora conceptos de programación de un nivel de abstracción superior a los existentes en otros lenguajes.

Mayor abstracción => Mayor poder

- Los conceptos y estilo de programación aprendidos, son aplicables a otros lenguajes y tecnologías.

Te convierte en mejor programador

- Es una plataforma madura para el desarrollo de aplicaciones en el mundo real.

¿Por qué aprender Haskell?

¿Por qué aprender Haskell?

- Presenciar la evolución de un lenguaje/plataforma ya madura y estable, pero con un ritmo de crecimiento muy interesante.

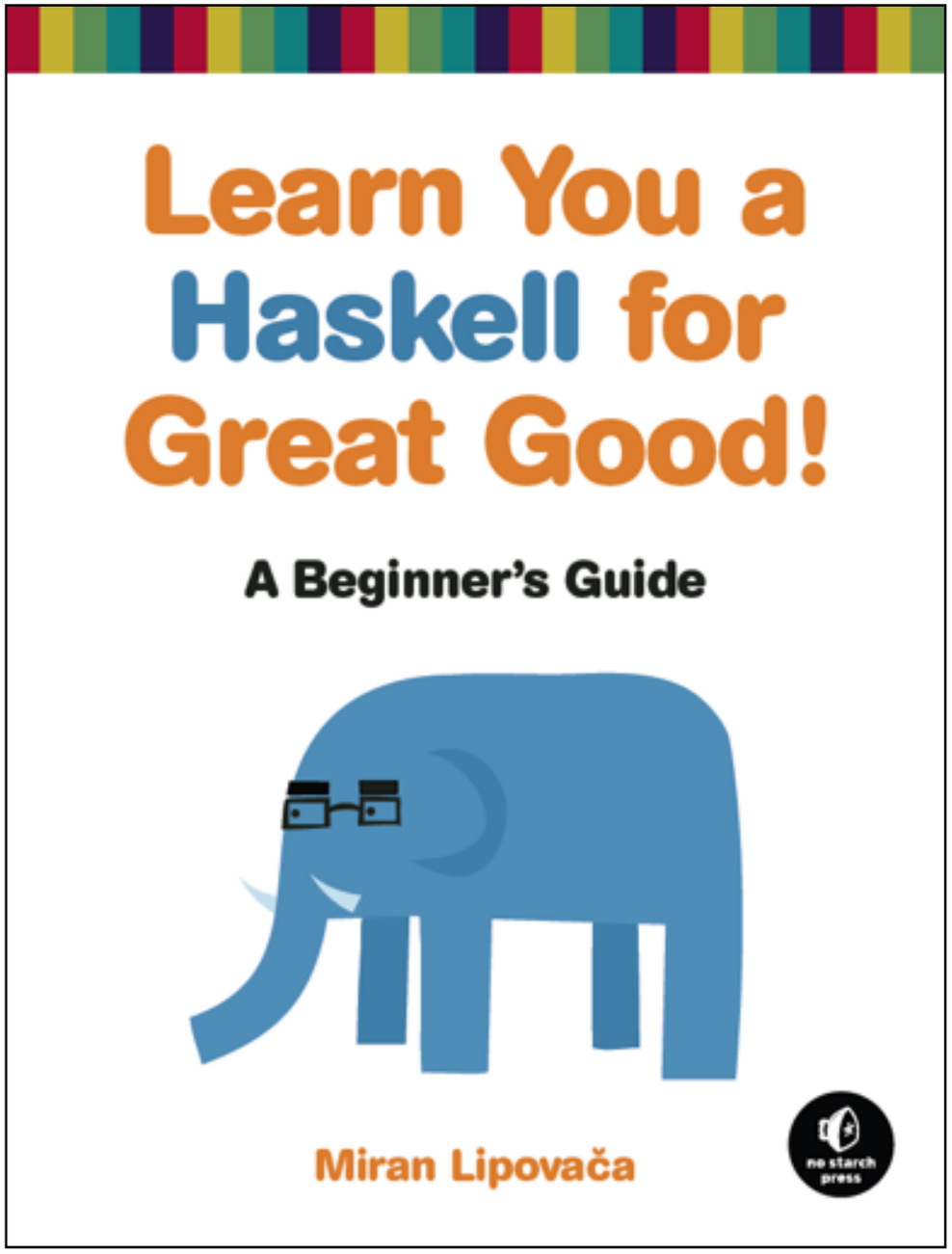
¿Por qué aprender Haskell?

- Presenciar la evolución de un lenguaje/plataforma ya madura y estable, pero con un ritmo de crecimiento muy interesante.
- La comunidad es muy dinámica y presta ayuda.

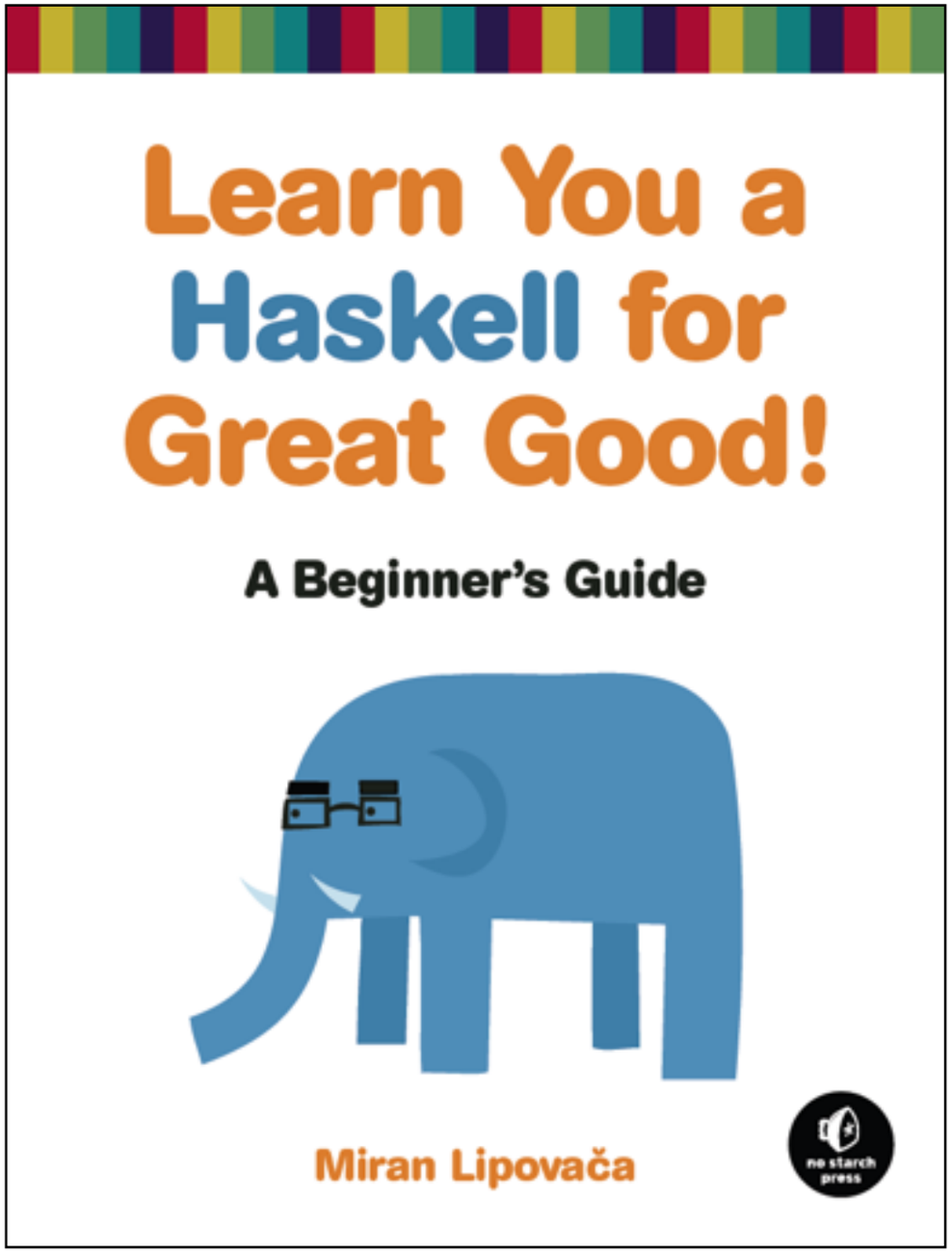
Porque... #SoyPunk

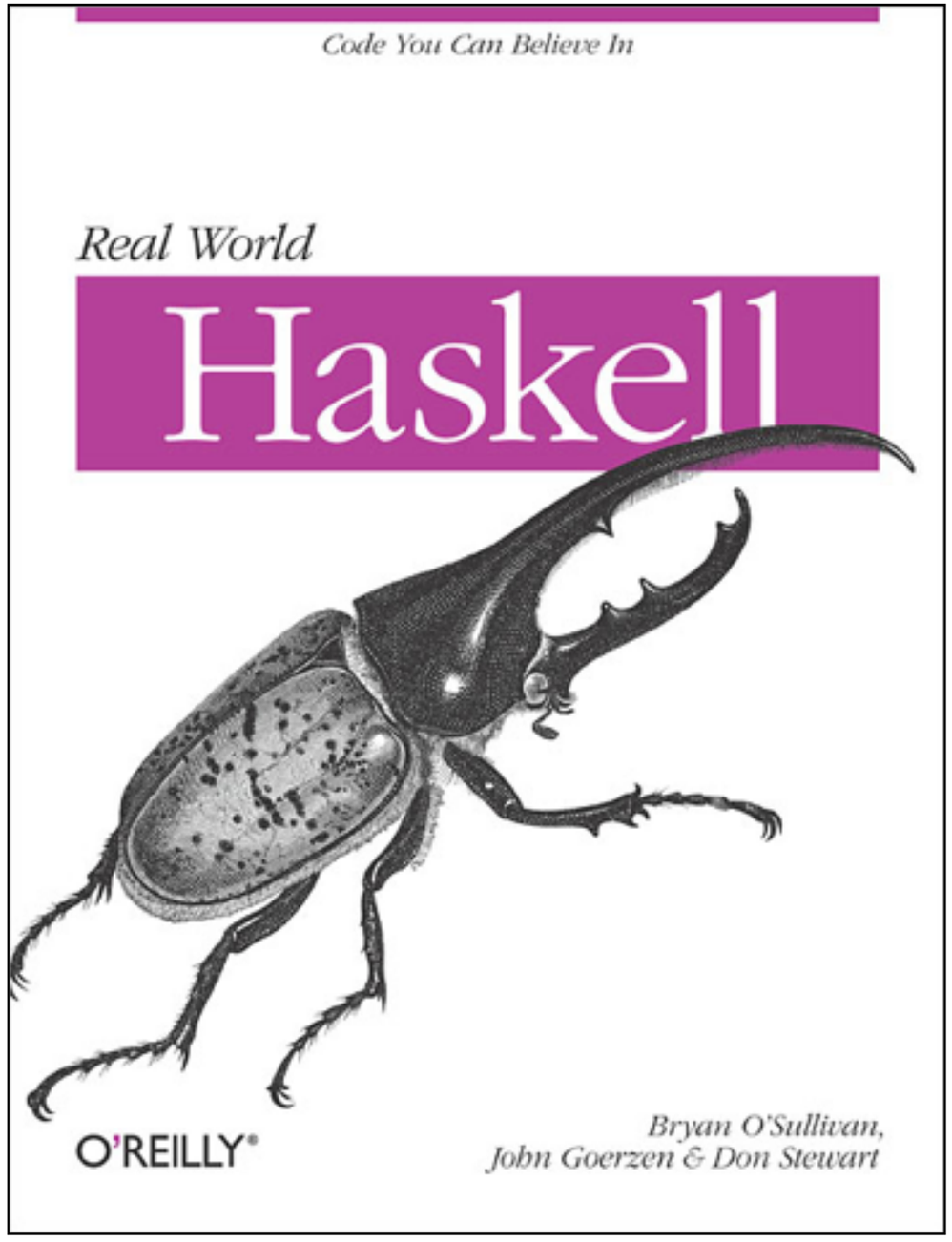
Para saber más... (1/3)

Para saber más... (1/3)

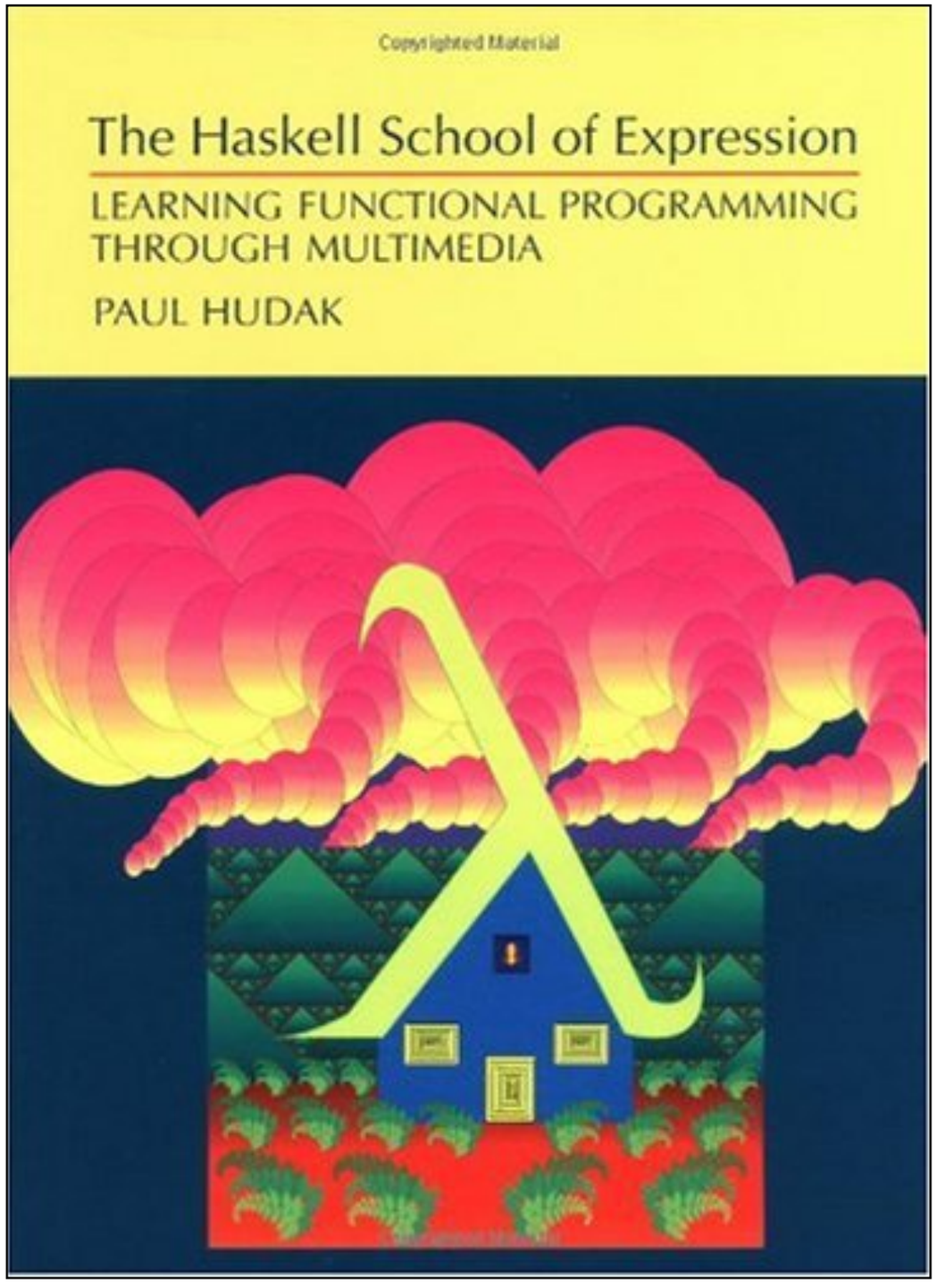


Para saber más... (1/3)





Para saber más... (2/3)



Para saber más... (3/3)

- http://tryhaskell.org/

  Tutorial interactivo en línea

- http://www.haskell.org

  Sitio oficial de Haskell

- http://j.mp/1cMCJr5

  Yet Another Haskell Tutorial

Para saber más...

Para saber más...

Para saber más...

@Lambda_Mx

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional
en México, D.F.

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional
en México, D.F.

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional
en México, D.F.

#HaskellDojoMx

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional
en México, D.F.

#HaskellDojoMx

Reunión quincenal para practicar Haskell

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional
en México, D.F.

#HaskellDojoMx

Reunión quincenal para practicar Haskell
y platicar sobre Programación Funcional

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional
en México, D.F.

#HaskellDojoMx

Reunión quincenal para practicar Haskell
y platicar sobre Programación Funcional

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional
en México, D.F.

#HaskellDojoMx

Reunión quincenal para practicar Haskell
y platicar sobre Programación Funcional

Para saber más...

@Lambda_Mx

Grupo de Interés en Programación Funcional
en México, D.F.

#HaskellDojoMx

Reunión quincenal para practicar Haskell
y platicar sobre Programación Funcional

¿Preguntas?

¡Gracias!

Agustín Ramos Fonseca
@MachinesAreUs