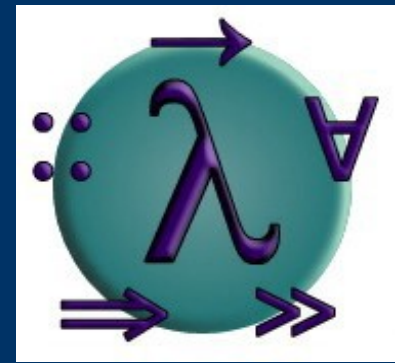


# El lenguaje Haskell 98



*Un lenguaje perezoso  
completamente Curry  
con sobrecarga.*



Juan Pedro Villa <jvillais+haskell@gmail.com>

---

---

# *Introducción*

“Las razones agudas son ronquidos para los oídos tontos”.

William Shakespeare



# Objetivos

## *Objetivo general*

- Investigar y aprender las características de la programación funcional a partir del lenguaje puramente funcional Haskell.

## *Objetivos específicos*

- Responder a la pregunta por la importancia del aprendizaje de un lenguaje de programación funcional y de su uso en la informática.
  - Comparar el paradigma funcional con otros paradigmas de programación, identificando las ventajas y desventajas de cada uno, especialmente en términos de modularización.
- 
-

# Objetivos (2)

## *Objetivos específicos (2)*

- Interpretar los resultados obtenidos del proceso de investigación y de elaboración de programas en el lenguaje de programación Haskell para la determinación de los aspectos fundamentales de un lenguaje funcional.
  - Participar en una comunidad de escritura colaborativa (wiki) mediante el uso y la publicación de contenidos relacionados con Haskell y la programación funcional.
  - Conocer la estructura básica de un proyecto de investigación que puede aplicarse o servir como base para otros.
- 
-

# Contenidos

- Introducción
- Programación Funcional
- Programación Funcional con Haskell 98
  - Programación funcional básica con Haskell 98
  - Comunidad Haskell
- Conclusiones



# *Programación funcional*



# *Programación funcional*

- La programación funcional es un paradigma de programación declarativa basado en el uso de funciones matemáticas.
  - Características:
    - Modo de evaluación: Impaciente o perezosa.
    - Funciones de primer orden o de orden superior.
    - Inferencia de tipos y polimorfismo o lenguajes libres de tipos (no tipificados) o con tipificación dinámica.
- 
-

# *Cualidades de los lenguajes de programación funcional*

- Todos los procedimientos son funciones y distinguen claramente los valores de entrada (parámetros) de los de salida (resultados).
  - No existen variables ni asignaciones – las variables han sido reemplazadas por los parámetros.
  - No existen ciclos – éstos han sido reemplazados por las llamadas recursivas.
- 
-

# *Cualidades de los lenguajes de programación funcional (2)*

- El valor de una función depende sólo del valor de sus parámetros y no del orden de evaluación o de la trayectoria de ejecución que llevó a la llamada.
- Las funciones son valores de primera clase.



# *Ventajas de la programación funcional (con Haskell)*

- Brevedad
  - Facilidad para comprender
  - Manejo de los tipos de datos
  - Reutilización de código y polimorfismo
  - Evaluación perezosa y programas modulares
  - Abstracciones poderosas y funciones como valores de primera clase
  - Recolección de basura
- 
-

# *Desventajas de la programación funcional*

- El inconveniente principal típicamente ha sido la ineficiencia en la ejecución de los lenguajes funcionales. Debido a su naturaleza dinámica, estos lenguajes siempre han sido interpretados más que compilados, resultando en una pérdida sustancial en velocidad de ejecución.
- 
-

# *Lenguajes de programación funcional*

- Lisp
  - Scheme
  - ML
  - Haskell
  - Hope
  - Miranda
  - Orwell
- 
-

# *Programación funcional en un lenguaje imperativo*

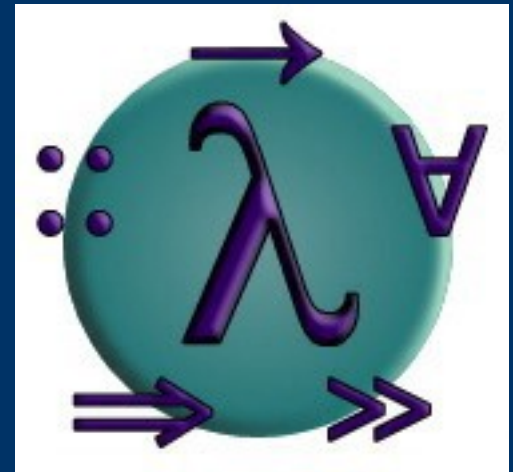
- El requisito básico para la programación funcional en cualquier lenguaje es la disponibilidad de la recursión y un mecanismo general de funciones adecuado.
  - Un problema típico en la programación de estilo funcional es el costo de llevar a cabo todos los ciclos mediante la recursión.
  - Pero existe una forma de recursión que puede convertirse fácilmente a una estructura estándar de ciclo, y se llama recursión de cola, donde la última operación de un procedimiento es llamarse a sí mismo.
- 
-

# *Las matemáticas en la programación funcional*

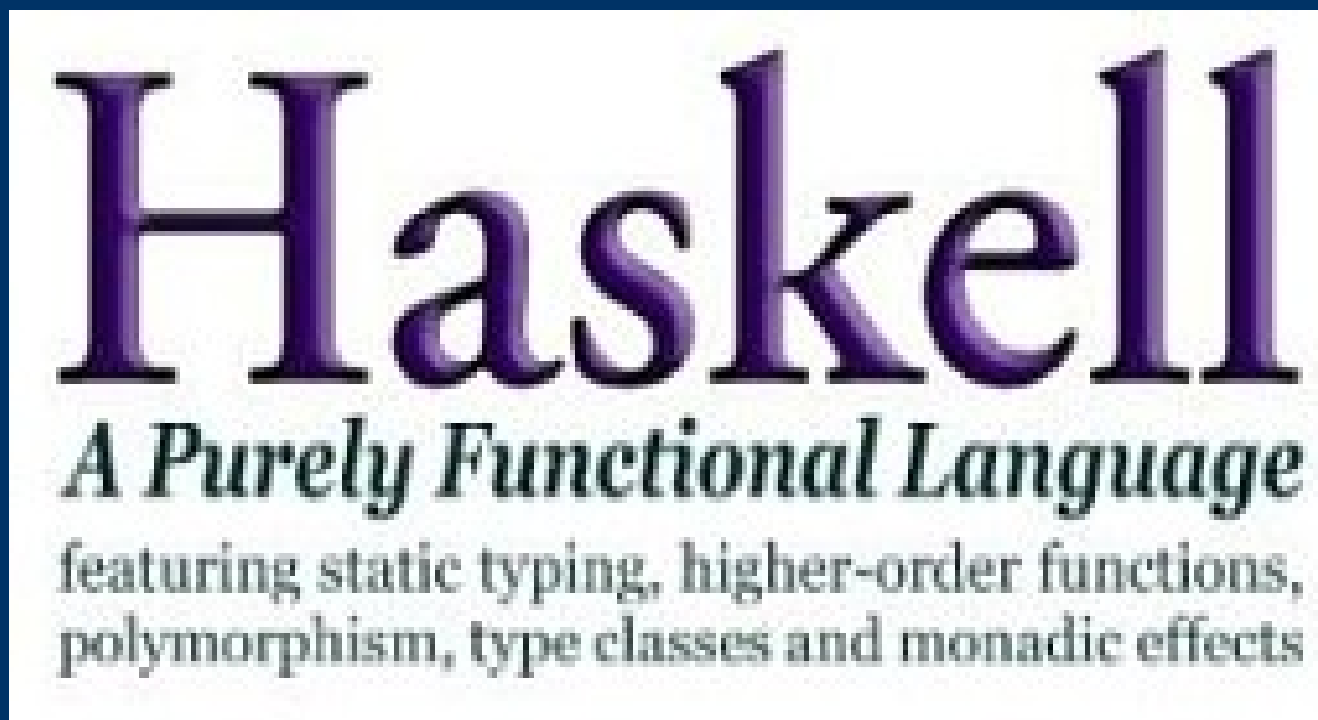
- Funciones recursivas
  - Se habla de definiciones recursivas.
- Cálculo lambda
  - Inventado por Alonzo Church como un formalismo matemático para expresar el cómputo por medio de funciones.



# *El lenguaje Haskell 98*



# Haskell



Un lenguaje puramente funcional fuertemente tipificado, con funciones de orden superior, polimorfismo, sistema de clases de tipos, mónadas, evaluación perezosa, ...

(Imagen tomada de Haskell - HaskellWiki)

---

# Haskell

<b>Lenguaje de programación Haskell</b>	
<b>Paradigma de computación</b>	Funcional, no estricto, modular
<b>Año de aparición</b>	1990
<b>Tipos de datos</b>	Estáticos y débiles
<b>Implementaciones</b>	GHC, Hugs, NHC, JHC, Yhc
<b>Estándar</b>	Haskell 98
<b>Influencias de</b>	Miranda, ML, Gofer
<b>Influencias en</b>	Python, Perl

# *Historia*

- Functional Programming and Computer Architecture (FPCA '87)
  - Haskell Report
  - [www.haskell.org](http://www.haskell.org)
  - Haskell 98
  - Haskell Wiki
  - Timeline (pdf)
  - Haskell 2008
- 
-

# *Haskell Brooks Curry (1900 - 1982)*



- Lógico y matemático norteamericano.
- Pionero de la lógica matemática moderna.
- Desarrolló la lógica combinatoria, que es la base de un estilo de programación funcional.
- Su trabajo ha sido útil en ciencias de la computación y en el diseño de lenguajes.

# *Presente y futuro del lenguaje Haskell*

- Haskell en educación
- Haskell en investigación
- Haskell en informática



# *Haskell en cinco pasos*

## 1. Instalar Haskell

1. GHC

2. Hugs

## 2. Iniciar Haskell

## 3. Escribir el primer programa en Haskell

1. Hello World!

2. Factorial

## 4. Haskell como calculadora

## 5. Pasos posteriores

---

---

# *Tipos simples predefinidos*

- El tipo *Bool*
  - El tipo *Int*
  - El tipo *Integer*
  - El tipo *Float*
  - El tipo *Double*
  - El tipo *Char*
- 
-

# El tipo *Bool*

- Los valores de este tipo representan expresiones lógicas, cuyo resultado puede ser `True` o `False`.
  - Funciones y operadores:
    - `(&&)` :: `Bool -> Bool -> Bool`. Conjunción lógica.
    - `(||)` :: `Bool -> Bool -> Bool`. Disyunción lógica.
    - `not` :: `Bool -> Bool`. Negación lógica.
    - `otherwise` :: `Bool`. Función constante que devuelve el valor `True`.
- 
-

# Los tipos *Int* e *Integer*

- Los valores del tipo *Int* son números enteros de precisión limitada que cubren al menos el intervalo  $[-2^{29}, 2^{29} - 1]$ .
  - Los valores del tipo *Integer* son números enteros de precisión ilimitada.
  - Funciones y operadores:
- 
-

# Los tipos *Int* e *Integer* (Funciones y operadores)

- $(+), (-), (*) :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$ . Suma, resta y producto de enteros.
  - $(^) :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$ . Operador potencia.
  - $\text{div}, \text{mod} :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$ . Cociente y residuo.
  - $\text{abs} :: \text{Int} \rightarrow \text{Int}$ . Valor absoluto.
  - $\text{signum} :: \text{Int} \rightarrow \text{Int}$ . Devuelve -1, 0 ó 1 según el signo.
  - $\text{negate} :: \text{Int} \rightarrow \text{Int}$ . Invierte el signo (uso prefijo con  $(-)$ ).
  - $\text{even}, \text{odd} :: \text{Int} \rightarrow \text{Bool}$ . Comprueban la naturaleza par o impar de un entero.
- 
-

# Los tipos *Float* y *Double*

- Representan números reales.
  - Notación:
    - Habitual: `1.35`, `-15.345`, `1.0` ó `1`.
    - Científica: `1.5e7` ó `1.5e-17`.
  - Funciones y operadores:
    - `(+)`, `(*)`, `(-)`, `(/)` :: `Float -> Float -> Float`.  
Suma, producto, resta y división real.
    - `(^)` :: `Float -> Int -> Float`. Operador potencia.
    - `(**)` :: `Float -> Float -> Float`. Operador potencia.
- 
-

# Los tipos *Float* y *Double* (Funciones y operadores)

- `abs :: Float -> Float`. Valor absoluto.
  - `signum :: Float -> Float`. -1.0, 0.0 ó 1.0 según el signo.
  - `negate :: Float -> Float`. Valor negado. Uso prefijo `-`.
  - `sin, asin, cos, acos, tan, atan :: Float -> Float`. Funciones trigonométricas (radianes).
  - `atan2 :: Float -> Float -> Float`. Arcotangente.
  - `log, exp :: Float -> Float`. Funciones y logarítmica y exponencial.
  - `sqrt :: Float -> Float`. Raíz cuadrada.
- 
-

# Los tipos *Float* y *Double* (*Funciones y operadores*) (2)

- `pi :: Float`. El valor del número pi.
  - `truncate, round, floor, ceiling :: Float -> Integer` ó `Float -> Int`.
    - `truncate` elimina la parte decimal.
    - `round` redondea al entero más próximo.
    - `floor` devuelve el entero inferior.
    - `ceiling` devuelve el entero superior.
  - `fromInt :: Int -> Float` y `fromInteger :: Integer -> Float`. Conversión de tipo.
- 
-

# El tipo Char

- Representa un carácter (una letra, un dígito, un signo, etc.).
  - Funciones y operadores:
    - `ord :: Char -> Int`. Devuelve el código ASCII.
    - `chr :: Int -> Char`. Inversa a `ord`.
    - `isUpper, isLower, isDigit, isAlpha :: Char -> Bool`.
    - `toUpper, toLower :: Char -> Char`.
- 
-

# Operadores de igualdad y orden

- Para todos los tipos básicos.
    - $>$  Mayor que
    - $<$  Menor que
    - $==$  Igual a
    - $>=$  Mayor o igual que
    - $<=$  Menor o igual que
    - $\neq$  Distinto de
- 
-

# *Constructores de tipos predefinidos*

- Tipos estructurados que permiten representar colecciones de objetos.
  - Tuplas
  - Listas
  - El constructor de tipo

# Tuplas

- Una tupla es un dato compuesto donde el tipo de cada componente puede ser distinto.
  - Si  $v_1, v_2, \dots, v_n$  son valores con tipos  $t_1, t_2, \dots, t_n$  entonces  $(v_1, v_2, \dots, v_n)$  es una tupla con tipo  $(t_1, t_2, \dots, t_n)$ .
  - Ejemplos:
    - (Integer, Bool)
    - (Integer, Integer, Integer)
- 
-

# Listas

- Es una colección de 0 o más elementos del mismo tipo. Hay dos constructores:
    - `[]` representa la lista vacía.
    - `(:)` permite añadir un elemento al principio de la lista.
  - Si  $v_1, v_2, \dots, v_n$  son valores con tipo  $t$ , entonces  $v_1 : (v_2 : (\dots (v_{n-1} : (v_n : [])))$  es una lista con tipo  $[t]$ .
  - Ejemplos:
    - `[1..3] = 1:(2:(3:[])) = ...`
    - Cadenas de caracteres: `[Char]` o `String`.
- 
-

# El constructor de tipo ->

- Tipos funcionales:
  - Si  $t_1, t_2, \dots, t_n, t_r$  son tipos válidos entonces
  - $t_1 \rightarrow t_2 \rightarrow \dots \rightarrow t_n \rightarrow t_r$  es el tipo de una función con n argumentos.
- Ejemplos:
  - `Factorial :: Integer -> Integer.`

# Comentarios

- Un comentario es un texto que acompaña al programa pero que es ignorado por el evaluador.
  - Hay dos modos de incluir comentarios en un programa:
    - Comentarios de una sola línea: Comienzan por dos guiones y abarcan hasta el final de la línea.
    - Comentarios de más de una línea: Comienzan por los caracteres { – y acaban en – }. Pueden abarcar varias líneas y anidarse.
- 
-

# Operadores

- Se trata de funciones con dos argumentos cuyo nombre es simbólico (cadena de signos) y que pueden ser invocados de forma infija.
  - Al definir un operador, puede definirse su prioridad (entero entre 0 y 9) y su asociatividad.
    - `infix` prioridad identificador
    - `infixl` prioridad identificador
    - `infixr` prioridad identificador
- 
-

# Asociatividad y prioridad de los operadores predefinidos

- infixr 9 .
  - infixl 9 !!
  - infixr 8 ^, ^^, \*\*
  - infixl 7 \*, /, `quot`, `rem`, `div`, `mod`
  - infixl 6 +, -
  - infixr 5 :, ++
  - infix 4 ==, /=, <, <=, >, >=, `elem`, `notElem`
  - infixr 3 &&
  - infixr 2 ||
  - infixl 1 >>, >>=
  - infixr 1 =<<
  - infixr 0 \$, \$!, `seq`
- 
-

# Operadores frente a funciones

- Los operadores se usan de modo infijo, mientras que las funciones se usan de modo prefijo.
- Los dos conceptos son tan similares que Haskell permite convertir cualquier operador en una función de dos argumentos, y una función de dos argumentos puede convertirse en un operador.
- Ejemplos:

- `div 4 2 = 4 `div` 2`

- `4 + 2 = (+) 4 2`

---

---

# Comparación de patrones

- En Haskell, un argumento puede ser un patrón. Es posible definir una función dando más de una ecuación para ésta. Cada ecuación define el comportamiento de la función para distintas formas del argumento, y los patrones permiten capturar dicha forma. Al aplicar la función a un parámetro concreto, la comparación de patrones determina la ecuación a utilizar.
  - El uso de patrones permite modelar cómo se evaluará una llamada a una función a partir de sus argumentos.
- 
-

# *Comparación de patrones*

- Patrones constantes
  - Patrones para listas
  - Patrones para tuplas
  - Patrones aritméticos
  - Patrones nombrados o seudónimos
  - El patrón subrayado
  - Patrones y evaluación perezosa
- 
-

# Expresiones case

- Es posible realizar una comprobación de patrones en cualquier punto de una expresión. Para ello debe usarse la construcción case.
- Sintaxis:

```
case expresión of
  patrón1 -> resultado1
  patrón2 -> resultado2
  ...     -> ...
  patrónn -> resultadoon
```

# La función error

- Es una función predefinida que permite al programador terminar la evaluación de una expresión y mostrar un mensaje por pantalla.

- Sintaxis:

```
error "mensaje"
```

- Ejemplo:

```
raizCuadrada n = if n >= 0 then sqrt  
else error "Raíz compleja"
```



# Funciones a trozos

- En las funciones a trozos o por partes, el resultado depende de cierta condición.
- Las condiciones deben aparecer después de una guarda (|) y son operaciones booleanas.
- Ejemplo:

absoluto :: Integer -> Integer

$$| x \geq 0 = x$$

$$| x < 0 = -x$$

# *Expresiones condicionales*

- Otro modo de escribir expresiones cuyo resultado dependa de cierta condición es:

```
if expresiónBool then expresiónSí else  
  expresiónNo
```

```
if 5 > 2 then 10.0 else (10.0 / 0.0)
```



# *Definiciones locales*

- En algunas ocasiones, es conveniente dar un nombre a una subexpresión que aparece varias veces en otra y utilizar este nombre en lugar de la expresión.
  - Son el mecanismo más básico que proporciona Haskell para conseguir encapsulamiento.
  - Con las definiciones locales se gana legibilidad y eficiencia.
- 
-

# *Expresiones lambda*

- Haskell permite definir funciones anónimas usando expresiones lambda.
- Son útiles al manejar funciones de orden superior (que toman como argumento otras funciones).



# *Sangrado*

- La disposición del texto del programa delimita las definiciones mediante la regla del fuera de juego (offside rule o layout).
- Una definición acaba con el primer trozo de código con un margen izquierdo menor que el del comienzo de la definición actual.



# *Lo verdaderamente importante en Haskell*

- Evaluación perezosa
- Sistema de clases
- Mónadas para Entrada-Salida
- Puramente funcional
- Currificación

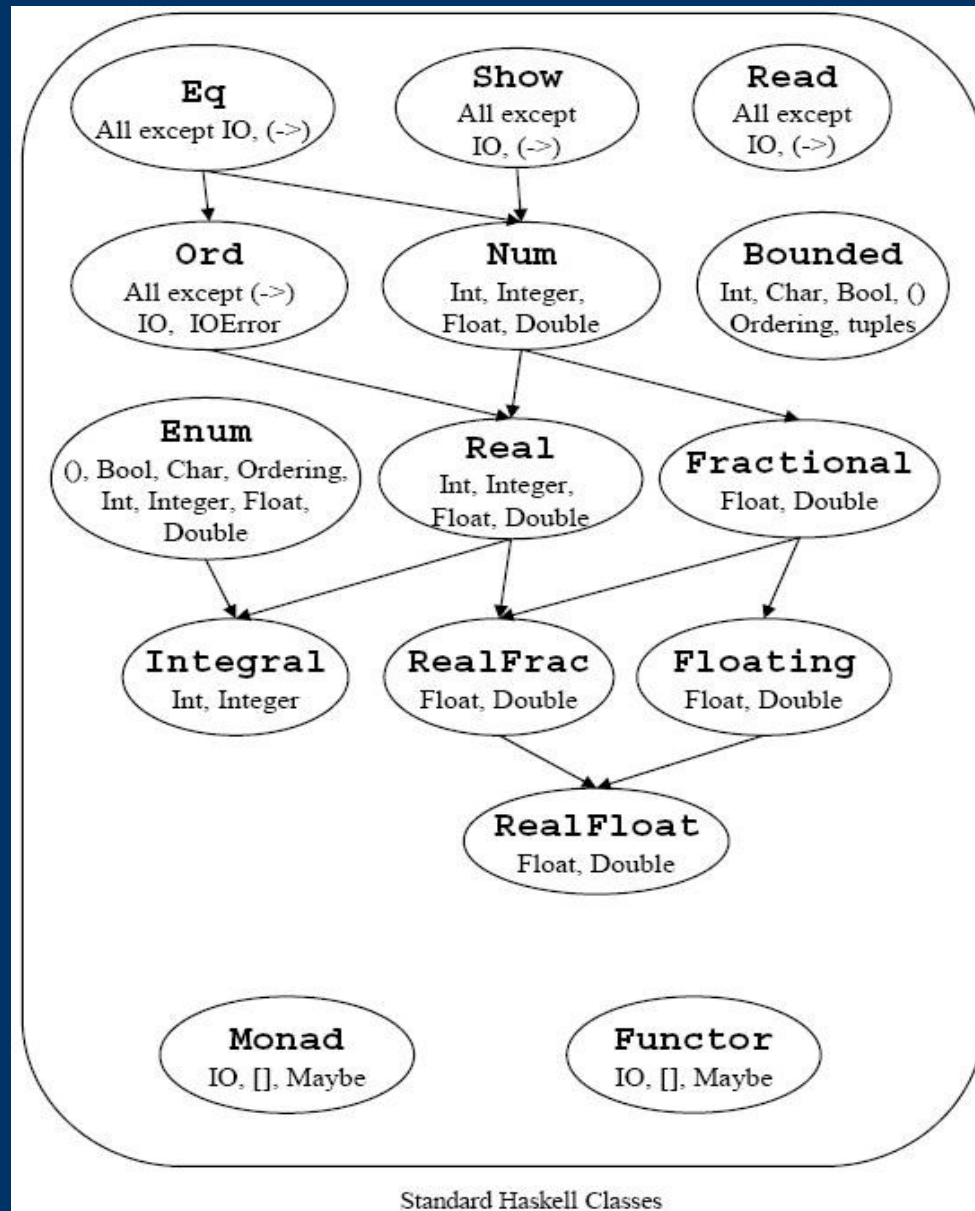


# *El sistema de clases de Haskell*

- El sistema de clases de Haskell permite restringir el tipo de ciertas funciones polimórficas imponiendo condiciones a los tipos usados en su declaración. Estas condiciones vienen dadas en forma de predicados que los tipos deben verificar.



# Standard Haskell Classes



# *El sistema de clases de Haskell*

- La clase `Eq` de `Prelude`: Clase cuyas instancias pueden compararse a través del operador de igualdad.
  - La clase `Ord` de `Prelude`: Sus miembros son `(<)`, `(<=)`, `(>)`, `(>=)`, `max`, `min` y `compare`.
  - `Read` y `Show`: Los resultados deben presentarse en la pantalla en forma de cadenas de caracteres (`Show`) o construirse a partir de éstas (`Read`).
- 
-

# *El sistema de clases de Haskell*

- Num, Integral y Fractional: Definidas para sobrecargar operaciones aritméticas para que todos los tipos numéricos puedan usarlas.
- Ratio: Proporciona tipos racionales sobre varios tipos numéricos.



# *Comunidad Haskell*

- Haskell Wiki
- Listas de correo, canal IRC, blogs, noticias, etc.
- Reconocimientos
- Programadores e investigadores en Haskell y en programación funcional



# Monadius



# Frag



***Conclusiones, enlaces y referencias bibliográficas***



# Enlaces

- Colaboradores De Wikipedia. "Haskell." Wikipedia, La Enciclopedia Libre. <<http://es.wikipedia.org/wiki/Haskell>>.
  - Haskell. Haskell community. <<http://haskell.org/>>.
  - Haskell PBwiki. <<http://haskell.pbwiki.com/>>.
  - "Haskell Research Project." The Black Cat. <<http://jpvitbc.com/haskell.aspx>>.
- 
-

# *Referencias bibliográficas*

- Hudak, Paul, John Peterson, and Joseph H. Fasel. A Gentle Introduction to Haskell 98. (pdf)
  - Hughes, John. Why Functional Programming Matters. Institutionen for Datavetenskap. (pdf)
  - Jones, Simon Peyton, ed. Haskell 98 Language and Libraries the Revised Report. Haskell Community. (pdf)
  - Jones, Simon Peyton. Wearing the Hair Shirt a Retrospective on Haskell. Microsoft Research. (pdf)
  - Ruiz, Blas, Francisco Gutiérrez, Pablo Guerrero, and José Gallardo. Razonando Con Haskell Un Curso Sobre Programación Funcional. Madrid: Thomson, 2004. (web)
- 
-