Lenguajes de programación

Introducción al lenguaje Haskell

Haskell

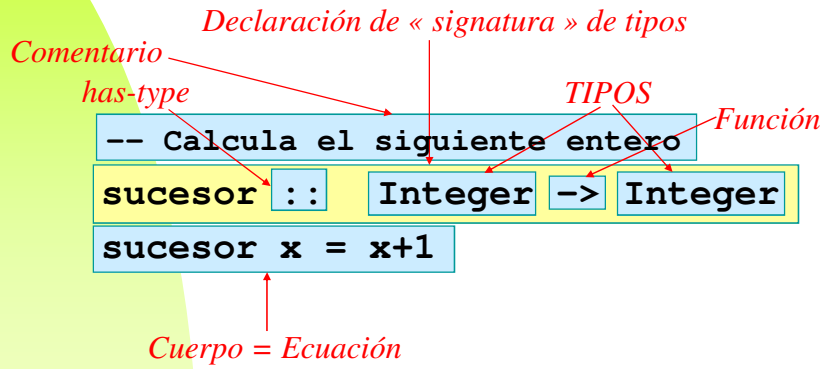
- Haskell es un lenguaje funcional puro, no estricto y fuertemente tipado
  - **Puro** = No efectos laterales, puras expresiones
  - **No estricto** = usa un orden no aplicativo (normal)
  - **Tipaje fuerte** = Todas las expresiones tienen un tipo de datos
- Propuesto por Paul Hudak (1987)
- Llamado así en honor a Haskell Curry (1900-1982)
  - Lógico y matemático
  - "Padre" de la lógica combinatoria
  - "Creador" de la "currificación"

Programas

- Un programa consiste en declaraciones y definiciones de funciones
  - **Declararla**: indicar el tipo
  - **Definirla**: dar el método de cómputo
- Ejemplo

```
-- Calcula el siguiente entero
sucesor :: Integer -> Integer
sucesor x = x + 1
```

Elementos de un programa



Implementación de Haskell

- The Hugs: http://haskell.org/hugs

  o
- The Haskell Platform: http://www.haskell.org/

Currificación

- Calcula la suma de los cuadrados de sus dos argumentos

Funciones currificadas!!

```
sumaCuadrados :: Int -> Int -> Int
sumaCuadrados x y = x * x + y * y
```

¿Qué regresa sumaCuadrados 2?

Recursión

- Ejemplo: Cálculo del factorial

```
factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n-1)
```

Caso Base: solución directa

Caso General: llamada recursiva

Listas

- Se representan como `[2,4,3,5]`
- Funciones básicas:
  - head y tail son como car y cdr de Scheme

```
head :: [a] -> a
head (x:lista) = x

tail :: [a] -> [a]
tail (_:xs) = xs
```

Patrón con el 1er elemento separado con ":"

Variable anónima

Polimorfismo

- Número de elementos de una lista

Polimorfismo de tipo
(listas de cualquier tipo)

```
length :: [a] -> Int
length [] = 0
length (x:xs) = 1 + length xs
```

Strings

- En Haskell los strings son listas de caracteres:

```
> ['h','o','l','a']
=> "hola"

> reverse "coche"
=> "ehcoc"
```

Condiciones

- Posición de un entero en una lista
  posint 3 [2,3,4,5,6] => 2

```
posint :: Int -> [Int] -> Int
posint _ [] = 0
posint x (y:lista) =
  if x == y then 1
    else 1 + posint x lista
```

Expresión Condicional:
if exp_b then x else y

Manejo de errores

- Encuentra la Posición de un entero en una lista

```
posint :: Int -> [Int] -> Int
posint _ [] = error "Entero no encontrado"
posint x (y:lista) =
  if x == y then 1 else 1 + posint x lista

> posint 3 [1,2,4,5]
Program error: Entero no encontrado.
```

Case

- Obten el primer elemento de una lista
  `car [2,3,4,5,6] => 2`

```
car :: [a] -> a
car xs = case xs of
   [] -> error "Lista vacia!"
   (x:_) -> x
```

Patrones