Introducción a Haskell

Y a la programación funcional

Pablo Baeyens
@pbaeyens

Mario Román
@M42

OSL 2015

Índice

¡Contribuye!

El código fuente de estas diapositivas está disponible en:

github.com/M42/osl-talk-haskell

Erratas, correcciones y aportaciones son bienvenidas.

Instalando Haskell Platform

El paquete `haskell-platform` contiene el compilador, depurador, gestor de librerías y otras utilidades para programar en Haskell. En otras distribuciones puede instalarse directamente `ghc` (Glasgow Haskell Compiler):

```
sudo apt-get install haskell-platform
```

Sin efectos secundarios

En los lenguajes de programación funcional, una función toma un argumento y devuelve una salida. No altera el mundo alrededor ni cambia el valor de los argumentos.

En lenguajes imperativos, cuando llamamos a una función puede cambiar nuestras variables o escribir por pantalla. Eso hace que el orden de llamada de las funciones importe.

```
int n = 0;
int next_n() { return n++; }
next_n(); // n = 1
```

El intérprete: GHCi

GHC es un compilador de Haskell con GHCi como intérprete asociado. El intérprete permite los siguientes comandos:

- `:q` Quitar
- `:l` Cargar módulo
- `:r` Recargar módulos
- `:t` Consultar tipos

Una vez cargado el intérprete podemos utilizarlo para probar el lenguaje. Haskell permite operaciones aritméticas básicas, y operaciones con cadenas, listas o booleanos.

El intérprete: GHCi

Podemos probar el uso de un puñado de funciones simples. Las funciones se escriben dejando sus argumentos a su lado y separados por espacios. ¡Estamos usando **notación polaca**!

```
ghci> 3 + 4
7
ghci> (+) 2 9
11
ghci> succ 27
28
ghci> max 23 34
34
```

Tipos

Haskell tiene los tipos básicos ya construidos. Existen `Int`, `Bool`, `Char`, ...

```
ghci> :t True
True :: Bool
ghci> :t 'a'
'a' :: Char
ghci> :t "a string!"
"a string!" :: [Char]
ghci> :t 2
2 :: (Num a) => a
```

Clases de tipos

También están definidas algunas clases de tipos, que agrupan a tipos con la misma interfaz. Por ejemplo, la mayoría de los tipos son instancias de la clase `Eq`, porque disponen de una función ==.

```
ghci> :t 2
2 :: Num a => a
ghci> :t pi
pi :: Floating a => a
ghci> :t (==)
(==) :: Eq a => a -> a -> Bool
```

Las instancias de la clase `Num` pueden sumarse y multiplicarse, las instancias de `Show` pueden convertirse a texto (String), y las instancias de `Integral` permiten calcular restos modulares sobre ellas.

Variables de tipo

Vemos que Haskell infiere siempre el tipo más general posible.

Constructores de tipos

Podemos definir funciones sobre tipos. Son constructores de tipos, que toman un tipo y nos dan otro. Por ejemplo, el constructor `[]`

construye listas de un tipo, y el constructor `Maybe` construye un tipo que puede tener o no un valor dado.

```
ghci> :t "Haskell!"
"Haskell!" :: [Char]
ghci> :t [1,2,3,4]
[1,2,3,4] :: (Num a) => [a]
ghci> :t [True, False, False]
[True, False, False] :: [Bool]
ghci> :t []
[] :: [a]
ghci> :t Just True
Just True :: Maybe Bool
```

Quicksort

Implementación del algoritmo Quicksort

```
qsort []     = []
qsort (x:xs) = qsort [y | y<-xs, y<=x]
               ++ [x]
               ++ qsort [y | y<-xs, y>x]
```