¡Hola, Haskell!

Fernanda Andrade

Stack Builders

28 de abril de 2016

¡Hola, mundo!

```
1 holaHaskell :: IO ()
2 holaHaskell = putStrLn "Hola, Haskell"
```

Haskell

`https://www.haskell.org/`

- Funcional
- Puro
  - Inmutabilidad
  - Sin efectos secundarios
  - Transparencia referencial
- Evaluación perezosa
- Tipificación estática
- Funciones de orden superior

Haskell es funcional

Factorial en C:

```
1 int  factorial (int n) {
2     int result = 1;
3     for (int i = 1; i <= n; ++i)
4         result *= i;
5     return result ;
6 }
```

Haskell es funcional

Factorial en Haskell:

```
1 factorial :: Integral -> Integral
2 factorial 0 = 1
3 factorial n = n * factorial (n-1)
```

- No es imperativo.
- Evaluación de expresiones.

Haskell es puro

Inmutabilidad

Secuencia Fibonacci en C:

```
1 long long int fibb(int n) {
2     int fnow = 0, fnext = 1, tempf;
3     while(--n>0){
4         tempf = fnow + fnext;
5         fnow = fnext;
6         fnext = tempf;
7     }
8     return fnext;
9 }
```

Haskell es puro

Inmutabilidad

Secuencia Fibonacci en Haskell:

```
1 fibonacci :: Integer -> Integer
2 fibonacci 0 = 0
3 fibonacci 1 = 1
4 fibonacci n = fibonacci (n - 1) + fibonacci (n - 2)
```

- Variables y estructuras de datos son inmutables.

Haskell no tiene efectos secundarios

En Haskell:

```
1 count :: List -> Int
```

En C:

```
1 int i = 0;
2 int count (List ls) {
3   i = 5;
4   ...
5 }
```

- Funciones sólo pueden calcular y retornar valores.
- Funciones garantizan integridad.

Haskell es puro y funcional

Referencia transparencial

```
1 reverse :: [a] -> [a]
2 reverse []     = []
3 reverse (x:xs) = reverse xs ++ [x]
```

```
> reverse [0,1,2,3,4,5]
 [5,4,3,2,1,0]
```

Haskell es puro y funcional

Referencia transparencial

```
1 propReverse :: [Integer] -> [Integer]
2 propReverse xs = reverse (reverse xs)
```

```
> propReverse [0,1,2,3,4,5]
 [0,1,2,3,4,5]
```

- Una función es llamada dos veces con los mismos parámetros, obtendremos siempre el mismo resultado.

Haskell tiene evaluación perezosa

```haskell
square :: Int -> Int
square x = x * x
```

```
> square (1 + 2)
=> (1 + 2) * (1 + 2)
=> 3 * (1+2)
=> 3 * 3
=> 9
```

- Haskell no calculará resultados hasta que se vea realmente forzado a hacerlo.

Haskell tiene evaluación perezosa

Primeros 5 números de una lista infinita

```
> take 5  [1..]
 [1,2,3,4,5]
```

- Es posible trabajar con estructura de datos infinitos.

Haskell es un lenguaje tipificado estáticamente

```
1 printString :: String -> IO ()
2 printString word = putStrLn word
```

```
> printString 5
  <interactive>:18:13:
    No instance for (Num String) arising from the literal
    In the first argument of ‘printString’, namely ‘5’
    In the expression: printString 5
    In an equation for ‘it’: it = printString 5
```

- Haskell verifica que el tipo de dato declarado coincide con el tipo inferido (en tiempo de compilación).

Funciones de orden superior

map

```
1 map :: (a -> b) -> [a] -> [b]
2 map _ [] = []
3 map f (x:xs) = f x : map f xs
```

```
> map even [0,1,2,3,4,5]
[True, False, True, False, True, False]
```

```
> map (+3) [0,1,2,3,4,5]
 [3,4,5,6,7,8]
```

- Funciones pueden tomar funciones como parámetros y devolver funciones como resultado.

Funciones de orden superior

filter

```
1  filter  :: (a -> Bool) -> [a] -> [a]
2  filter  _ []  = []
3  filter  p (x:xs)
4      | p x         = x :  filter  p xs
5      | otherwise =  filter  p xs
```

```
> filter  even  [1,2,3,4,5,6]
[2,4,6]
```

```
> filter  (>3)  [1,2,3,4,5,6]
[4,5,6]
```