Haskell en ejemplos

Juan Pedro Villa Isaza

Stack Builders

8 de abril de 2016

¡Hola, Haskell!

```
main :: IO ()
main = putStrLn "¡Hola, Haskell!"
```

Haskell

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

Haskell es...

- Funcional
- Puro
  - Inmutabilidad
  - Sin efectos secundarios
  - Transparencia referencial
- Multipropósito
- ...

Haskell es funcional

Fibonacci

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

```
> fibonacci 5
5
```

Haskell es funcional

Fibonacci

```
int fibonacci(int n) {
  if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return fibonacci (n - 1) + fibonacci (n - 2);
}
```

Haskell es puro

Factorial

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

```
> factorial 5
120
```

Haskell es puro

Factorial

```
int factorial(int n) {
  int factorial = 1;
  while (n > 0) {
    factorial = factorial * n;
    n = n - 1;
  }
  return factorial;
}
```

Haskell es puro

Factorial

```
int factorial(int n) {
  if (n == 0)
    return 1;
  else
    return n * factorial(n - 1);
}
```

Haskell es puro

Factorial

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

```
> factorial 5
120
```

Haskell es puro y funcional

reverse

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

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

Haskell es puro y funcional

reverse

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

```
> quickCheck propReverse
+++ OK, passed 100 tests.
```

Haskell tiene...

- Funciones de orden superior
- Evaluación perezosa
- ...

Haskell tiene funciones de orden superior

`map`

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

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

Haskell tiene funciones de orden superior

`map`

Los diez primeros números de Fibonacci (en C):

```
for (int i = 1; i <= 10; i++) {
  printf("%d\n", fibonacci(i - 1));
}
```

Haskell tiene funciones de orden superior

`map`

Los diez primeros números de Fibonacci (en Haskell):

```
> map fibonacci [0..9]
[0,1,1,2,3,5,8,13,21,34]
```

Haskell tiene evaluación perezosa

`map`

Los diez primeros números de Fibonacci (en Haskell):

```
> let fibonaccis = map fibonacci [0..]
> take 10 fibonaccis
[0,1,1,2,3,5,8,13,21,34]
```

Haskell tiene funciones de orden superior

filter

```
filter :: (a -> Bool) -> [a] -> [a]
filter _ []       = []
filter p (x:xs)
  | p x           = x : filter p xs
  | otherwise     = filter p xs
```

```
> filter odd [0..9]
[1,3,5,7,9]
```

Haskell tiene funciones de orden superior

`filter`

Los diez primeros números pares de Fibonacci (en C):

```
for (int i = 1, j = 0; i <= 10; j++) {
  if (fibonacci(j) % 2 == 0) {
    printf("%d\n", fibonacci(j));
    i = i + 1;
  }
}
```

Haskell tiene funciones de orden superior

filter

Los diez primeros números pares de Fibonacci (en Haskell):

```
> take 10 (filter even fibonaccis)
[0,2,8,34,144,610,2584,10946,46368,196418]
```

Haskell tiene funciones de orden superior

foldr

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr _ n []     = n
foldr c n (x:xs) = c x (foldr c n xs)
```

```
sum :: [Integer] -> Integer
sum ns = foldr (+) 0 ns
```

Haskell tiene funciones de orden superior

foldr

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr _ n []     = n
foldr c n (x:xs) = c x (foldr c n xs)
```

```
sum :: [Integer] -> Integer
sum = foldr (+) 0
```

Haskell tiene funciones de orden superior

foldr

La suma de los diez primeros números pares de Fibonacci (en C):

```
int s = 0;
for (int i = 1, j = 0; i <= 10; j++) {
  if (fibonacci(j) % 2 == 0) {
    s = s + fibonacci(j);
    i = i + 1;
  }
}
printf("%d\n", s);
```

Haskell tiene funciones de orden superior

`foldr`

La suma de los diez primeros números pares de Fibonacci (en Haskell):

```
> sum (take 10 (filter even fibonaccis))
257114
```

Haskell en ejemplos

https://github.com/stackbuilders/haskell-en-ejemplos