

¡Hola, Haskell!

Fernanda Andrade

Stack Builders

28 y 29 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

NO ES IMPERATIVO

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

NO ES IMPERATIVO

Factorial en Haskell:

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



HASKELL ES FUNCIONAL

NO ES IMPERATIVO

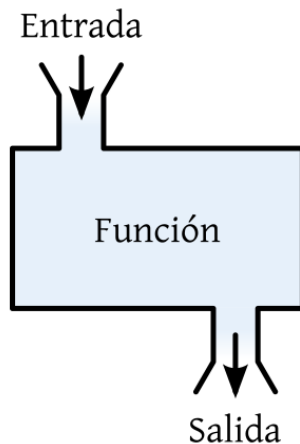
Factorial en Haskell:

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



HASKELL ES FUNCIONAL

EVALUACIÓN DE EXPRESIONES



HASKELL ES FUNCIONAL

EVALUACIÓN DE EXPRESIONES

$$f(x) = x + x$$



HASKELL ES FUNCIONAL

EVALUACIÓN DE EXPRESIONES

$$f(x) = x + x$$

$$f(2) = 2 + 2$$

$$f(2) = 2$$



HASKELL ES FUNCIONAL

EVALUACIÓN DE EXPRESIONES

$$f(x) = x + x$$

$$f(2) = 2 + 2$$

$$f(2) = 2$$

$$f(5) = 5 + 5$$

$$f(5) = 10$$



HASKELL ES FUNCIONAL

EVALUACIÓN DE EXPRESIONES

```
1  doble :: Int -> Int
2  doble n = n + n
```



HASKELL ES PURO

INMUTABILIDAD

Sumar números de 1 a n en C:

```
1 int sum(int n){  
2     int result = 0;  
3     for(int i=1; i<=n; i++)  
4         result += i  
5     return result  
6 }
```



HASKELL ES PURO

INMUTABILIDAD

En caso de $n = 3$

```
i = 1
```

```
total = 1
```



HASKELL ES PURO

INMUTABILIDAD

En caso de $n = 3$

```
i = 1
```

```
total = 1
```

```
i = 2
```

```
total = 3
```



HASKELL ES PURO

INMUTABILIDAD

En caso de $n = 3$

```
i = 1
```

```
total = 1
```

```
i = 2
```

```
total = 3
```

```
i = 3
```

```
total = 6
```



HASKELL ES PURO

INMUTABILIDAD

Sumar números de 1 a n en Haskell:

```
1 sum [1.. n]
```



HASKELL ES PURO

INMUTABILIDAD

Sumar números de 1 a n en Haskell:

```
1 sum :: Num a => [a] -> a
2 sum [] = 0
3 sum (x : xs) = x + sum xs
```

- Variables y estructuras de datos son inmutables.



HASKELL NO TIENE EFECTOS SECUNDARIOS

En C:

```
1 int main() {  
2     printf("Hola, mundo.");  
3     return (0);  
4 }
```



HASKELL NO TIENE EFECTOS SECUNDARIOS

En Haskell:

```
1 count :: [a] -> Int
```

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



HASKELL ES PURO

TRANSPARENCIA REFERENCIAL

En c:

```
1  int global = 5;
2
3  int suma (int n){
4      return (n + global);
5  }
6
7  int main(){
8      int resultado;
9
10     // resultado = 6
11     resultado = suma(1);
12
13     global = 0;
14
15     // resultado = 1
16     resultado = suma(1);
17
18 }
```



HASKELL ES PURO

TRANSPARENCIA REFERENCIAL

En Haskell:

```
1 x :: Int
```

```
2 x = 5
```

```
3
```

```
4 suma :: Int -> Int
```

```
5 suma n = n + x
```



HASKELL ES PURO

TRANSPARENCIA REFERENCIAL

```
> suma 1  
6  
> suma 1  
6
```

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



HASKELL ES PURO

TRANSPARENCIA REFERENCIAL

```
1 x :: Int
2 x = 5
3
4 suma :: Int -> Int
5 suma n = n + x
6
7 x = 0
```

Multiple declarations of 'x'

Declared at: suma.hs:2:1

suma.hs:7:1

Failed, modules loaded: none.



HASKELL TIENE EVALUACIÓN PEREZOSA

```
1 square :: Int -> Int
2 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
```



HASKELL ES UN LENGUAJE TIPIFICADO ESTÁTICAMENTE

```
> printString 5
```

```
<interactive>:18:13:
```

```
No instance for (Num String) arising from the  
literal '5'
```

```
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
```

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



FUNCIONES DE ORDEN SUPERIOR

MAP

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



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
```



FUNCIONES DE ORDEN SUPERIOR

FILTER

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

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

