

Ahorcado

Programación funcional imperativa

Juan Pedro Villa Isaza

Stack Builders

24 de abril de 2015



Haskell es:

- ▶ Funcional
- ▶ *Puro*
- ▶ ...



Haskell es:

- ▶ Funcional
- ▶ *Puro*
- ▶ ...
- ▶ *Interesante*



Haskell es:

- ▶ Funcional
- ▶ *Puro*
- ▶ ...
- ▶ *Interesante*
- ▶ ¿Inútil?

▶ `reverse :: [a] -> [a]`
`reverse = ...`

▶ `> reverse "haskell"`
`"lleksah"`

▶ `main :: IO ()`
`main = putStrLn (reverse "haskell")`

▶ `> main`
`lleksah`

```
▶ main :: IO ()  
main = do  
    line <- getLine  
    putStrLn (reverse line)
```

```
▶ > main  
haskell  
lleksah
```

```
type IO a = Mundo -> (a,Mundo)
```



Haskell es el mejor lenguaje de programación imperativa del mundo.

—Simon Peyton Jones

| |
|
|
|
|

Vidas: 6

| |
|
|
|
|

a

Vidas: 6

| |
|
|
|
|

-a-----

Vidas: 6

```

-----
|  |
|
|
|
|
-----

```

-a-----

e

Vidas: 6

```

-----
|  |
|
|
|
|
-----

```

-a-----

e

Vidas: 6

```

-----
|  |
|
|
|
|
-----

```

-a-----

e

Vidas: 6

```

-----
|  |
|
|
|
|
-----

```

-a-----

e

Vidas: 6

--
| |
|
|
|
|
--

-a--e--

Vidas: 6

--
| |
|
|
|
|
--

-a--e--

Vidas: 6

--
| |
|
|
|
|
--

-a--e--

Vidas: 6

|
|
|
|
|

-a--e--

i

Vidas: 6

```

-----
| |
| 0
|
|
|
|
-----

```

-a--e--

Vidas: 5

```

-----
|  |
|  0
|
|
|
|
-----

```

-a--e--

Vidas: 5

```

-----
|  |
|  0
|
|
|
|
-----

```

-a--e--

Vidas: 5

```
  ---
  |  |
  | 0
  |  |
  |
  |
  ---
```

-a--e--

Vidas: 4

```
  ---
  |  |
  | 0
  |  |
  |
  |
  ---
```

-a--e--

Vidas: 4

u

```
  ---  
  |  |  
  | 0  
  | /|  
  |  
  |  
  ---
```

-a--e--

Vidas: 3

```
  ---  
  |  |  
  | 0  
  | /|  
  |  
  |  
  ---
```

-a--e--

Vidas: 3

1

| |

| 0

| /|

|

|

|

-a--ell

Vidas: 3

```
  ---  
  |  |  
  | 0  
  | /|  
  |  
  |  
  ---
```

-a--ell

Vidas: 3

y

```
  ---  
  |  |  
  | 0  
  | /|\  
  |  
  |  
  ---
```

-a--ell

Vidas: 2

```
  ---
  |  |
  | 0
  | /\
  |
  |
  ---
```

-a--ell

Vidas: 2

s

```
  ---  
  |  |  
  | 0  
  | /|\  
  |  
  |  
  ---
```

-as-ell

Vidas: 2

| |

| 0

| /|\

|

|

|

-as-ell

Vidas: 2

haskell

\0/
|
/\

;haskell!

```
▶ data HangmanGame = HangmanGame
    { hangmanGameStatus :: HangmanGameStatus
    , hangmanGameWord   :: HangmanWord
    }
```

```
▶ data HangmanGameStatus
    = Lost
    | Playing Int
    | Won
```

```
type HangmanWord = [(Char,Bool)]
```

▶ `nextHangmanGame :: HangmanGame`
 `-> [Char]`
 `-> HangmanGame`

▶ `nextHangmanWord :: HangmanWord`
 `-> Char`
 `-> Maybe HangmanWord`

```
playHangmanGame :: HangmanGame -> IO ()
```

```
main :: IO ()
main = do
    ...
    playHangmanGame (HangmanGame ...)
    ...
```

`https://github.com/stackbuilders/hangman`

Gracias al sistema de tipos, distinción clara entre:

- ▶ IO y no IO
- ▶ Acciones y funciones
- ▶ Código impuro y código puro
- ▶ Programación imperativa y programación no imperativa

- ▶ Programación funcional:
 - ▶ Razonamiento ecuacional
 - ▶ ...
- ▶ Programación funcional imperativa:
 - ▶ Menos dolores de cabeza
 - ▶ ...

Haskell es el mejor lenguaje de programación imperativa del mundo.

—Simon Peyton Jones

Según las respuestas de más de 368000 personas:

- ▶ “El aprendizaje de este lenguaje mejoró mi habilidad como programador”:

1. Haskell

2. Standard ML

3. Scheme

4. Coq

5. Common Lisp

45. Shell

46. MATLAB

47. PHP

48. Visual Basic

49. COBOL




<http://hmrp.pl/zdDNYP>

Según las respuestas de más de 368000 personas:

- “Recomendaría a la mayoría de programadores aprender este lenguaje, sin importar si tienen o no una necesidad específica de hacerlo”:


- | | |
|------------|------------------|
| 1. Haskell | 44. ActionScript |
| 2. Scheme | 45. Pascal |
| 3. Coq | 46. Fortran |
| 4. Clojure | 47. Visual Basic |
| 5. Erlang | 48. COBOL |

 <http://hmrp.pl/zuMkvG>

Según las respuestas de más de 368000 personas:

- ▶ “El aprendizaje de este lenguaje cambió significativamente la manera en que uso otros lenguajes”:

- | | |
|----------------|------------------|
| 1. Haskell | 43. Fortran |
| 2. Scheme | 44. R |
| 3. Coq | 45. AWK |
| 4. Common Lisp | 46. Visual Basic |
| 5. Erlang | 47. COBOL |

 <http://hmrp.pl/y0tN9k>

► `reverse :: [a] -> [a]`
`reverse = foldl (flip (:)) []`

► `reverse :: [a] -> [a]`
`reverse l = rev l []`
 where
 `rev [] a = a`
 `rev (x:xs) a = rev xs (x:a)`
