

Práctica 3: Haskell

Isidro Francisco Pérez Paz - 377806

Instalación del entorno de Haskell

Descarga entorno

Para poder utilizar *Haskell*:

1. Dirigete a la pagina de *Haskell* y ve a la pestaña de Descargas

Dentro te va aparecer esto:

The screenshot shows a web browser window with the URL 'haskell.org/downloads/' in the address bar. The page header includes the Haskell logo and navigation links like 'Get started', 'Downloads', 'Playground', 'Community', etc. A prominent blue sidebar on the left contains the text 'Psssst!' and 'Looking to get started with Haskell? If so, check out the [Get Started](#) page!'. Below this, there's a section titled 'Downloads' with the heading 'Recommended installation instructions' and a note for 'Linux, macOS, FreeBSD, Windows or WSL'. A bullet point lists 'Use [GHCup](#) to install GHC, cabal-install, Stack and haskell-language-server'. Further down, another section discusses the Haskell toolchain with a bullet point about 'GHC: the Glasgow Haskell Compiler'.

2. Pulsa en el recuadro morado "GHCup"

Ya dentro copia el comando que aparece aqui:

The screenshot shows the GHcup website's "Installation" section. The page has a yellow header with the GHcup logo and navigation links for Home, Installation, First steps, User Guide, Developer Guide, and About. Below the header is a large yellow banner with the GHcup logo. The main content area has a background of lined paper. It starts with the text "To install on Windows" and "run the following in a PowerShell session (as a non-admin user):". A code block shows the PowerShell command: `$ Set-ExecutionPolicy Bypass -Scope Process -Force;[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; try { & ([ScriptBlock]::Create([Invoke-WebRequest https://www.haskell.org/ghcup/sh/bootstrap-haskell.ps1 -UseBasicParsing])) -Interactive -DisableCurl } catch { Write-Error $_ }`. Below the command are links for "What does this do?", "I don't like curl.sh", "Show all platforms", and "System requirements".

3. Dirígete a **PowerShell** pegalo, ejecutalo y instala todo

Nota: no lo ejecutes como administrador.

A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command shown is: `PS C:\Users\isidr> Set-ExecutionPolicy Bypass -Scope Process -Force;[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; try { & ([ScriptBlock]::Create([Invoke-WebRequest https://www.haskell.org/ghcup/sh/bootstrap-haskell.ps1 -UseBasicParsing])) -Interactive -DisableCurl } catch { Write-Error $_ }`.

Funcionamiento de la aplicación TODO escrita en Haskell

1. Inicialización del proyecto

- Se crea un nuevo proyecto usando Stack como plantilla.
- Se establece el archivo **package.yaml** o **.cabal**, se configuran dependencias mínimas y se pueden añadir otras como **dotenv**, **open-browser**.
- Se compila/testea con stack build, stack test, stack run.

2. Ejecución de la aplicación

- Al ejecutar `stack run`, se muestra un menú con comandos disponibles:

```

32 main :: IO ()
33 main = do
34   loadFile defaultConfig
35   website <- lookupEnv "WEBSITE"
36
37   case website of
38     Nothing -> error "You should set WEBSITE at .env file."
39     Just s -> do
40       result <- openBrowser s
41       if result
42         then print ("Could open " ++ s)
43       else print ("Couldn't open " ++ s)
44
45   putStrLn "Commands:"
46   putStrLn "+ <String>" -> addTodoEntry
47   putStrLn "- <Int>" -> deleteNumberedEntry
48   putStrLn "s <Int>" -> showNumberedEntry
49   putStrLn "e <Int>" -> editNumberedEntry
50   putStrLn "l" -> listTodos
51   putStrLn "r" -> reverseTodos
52   putStrLn "c" -> clearTodos
53   putStrLn "q" -> quit
54   prompt [] -- Start with the empty todo list.

```

- Luego se invoca la función `prompt []` con la lista de todos inicialmente vacía.

3. Bucle interactivo

- Dentro de `prompt todos`, se imprime un mensaje para el usuario pidiendo un comando.
- Se lee la línea de entrada del usuario.
- Si el comando comienza con '`e`' (editar), se primero pide "What is the new todo for that?" y se lee la nueva descripción, luego llama `editTodo`.
- En otro caso, se llama a `interpret command todos` que analiza el comando y actúa según el prefijo.

```

14 prompt :: [String] -> IO ()
15 prompt todos = do
16   putStrLn ""
17   putStrLn "Test todo with Haskell. You can use +(create), -(delete), s(show), e(edit), l(list), r(reverse), c(lear),"
18   command <- getLine
19   if "e" `isPrefixOf` command
20     then do
21       putStrLn "What is the new todo for that?"
22       newTodo <- getLine
23       editTodo command todos newTodo
24     else interpret command todos

```

4. Interpretación de comandos

Dependiendo del comando dado:

- `('+': '' : todo)`: agrega un nuevo todo al frente de la lista: construye la nueva lista `todo : todos`, y vuelve a `prompt` con los nuevos todos.
- `('-': '' : num)`: lee `num` como índice, llama `deleteOne (read num) todos`. Si devuelve `Nothing`, imprime "No TODO entry matches the given number" y vuelve al `prompt` con la lista sin cambios; si `Just todos'`, entonces pasa `todos'` al `prompt`.
- `('s': '' : num)`: muestra el elemento número `num` si existe, usando `showOne`. Si no, muestra mensaje de error. Luego vuelve al `prompt` con la misma lista.
- `"l"`: lista todos los elementos actuales: calcula `length todos`, imprime cuántos en total; hacen `mapM_ putStrLn (zip [0..] todos)` (imprime índice y el todo) luego vuelve al `prompt`.
- `"r"`: muestra la lista invertida: calcula `reversedTodos = reverseTodos todos`, luego similar a `l`, imprime la invertida, luego vuelve al `prompt` con la lista original aún intacta (importante: no la sustituye).
- `"c"`: limpia la lista: imprime "Clear todo list." y llama `prompt []` (lista vacía).
- `"q"`: termina el bucle: `interpret "q" todos = return ()`.
- Cualquier otro comando — imprime `Invalid command: \``` y vuelve al `prompt` con la lista sin cambios.

5. Funciones auxiliares

- **deleteOne :: Int -> [a] -> Maybe [a]**: elimina el elemento en índice dado, si existe; de lo contrario **Nothing**.
 - **showOne :: Int -> [a] -> Maybe a**: devuelve el elemento en índice dado, si válido; de lo contrario **Nothing**.
 - **editIndex :: Int -> a -> [a] -> [a]**: cambia el elemento en índice **i** a **x**, construyendo nueva lista (inmutabilidad).
 - **editTodo :: String -> [String] -> String -> IO ()**: se encarga del flujo cuando se edita: parsea el índice desde comando (`'e': ' ' : num`), llama **editOne**, imprime "Old todo is ...", "New todo is ...", luego imprime el nuevo estado (número de elementos y lista), y llama **prompt newTodos**.
 - **reverseTodos :: [a] -> [a]**: invierte la lista implementando recursivamente **go xs []**.
-

Estructura del código

- **app/Main.hs**: donde está la función **main**, imprime los comandos y llama **prompt []**.
- **src/Lib.hs**: contiene la lógica del prompt, la interpretación de comandos, y las funciones auxiliares como **deleteOne**, **showOne**, **editIndex**, etc.
- **test/Spec.hs**: contiene al menos una prueba: se explica en el artículo que prueban **editIndex**, por ejemplo.
- **package.yaml / stack.yaml**: configuración del proyecto con dependencias. El tutorial menciona añadir **dotenv** y **open-browser** aunque para la funcionalidad básica del TODO no son estrictamente necesarias.