Práctica 3: Haskell

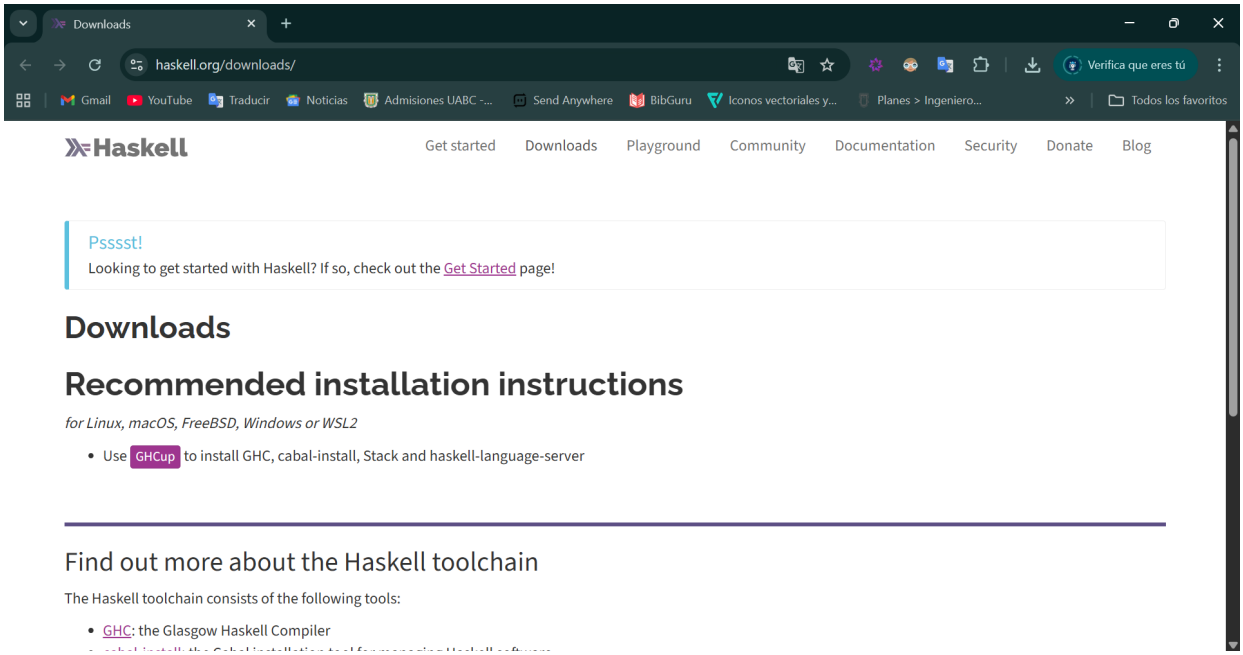
Isidro Francisco Pérez Paz - 377806

Instalación del entorno de Haskell

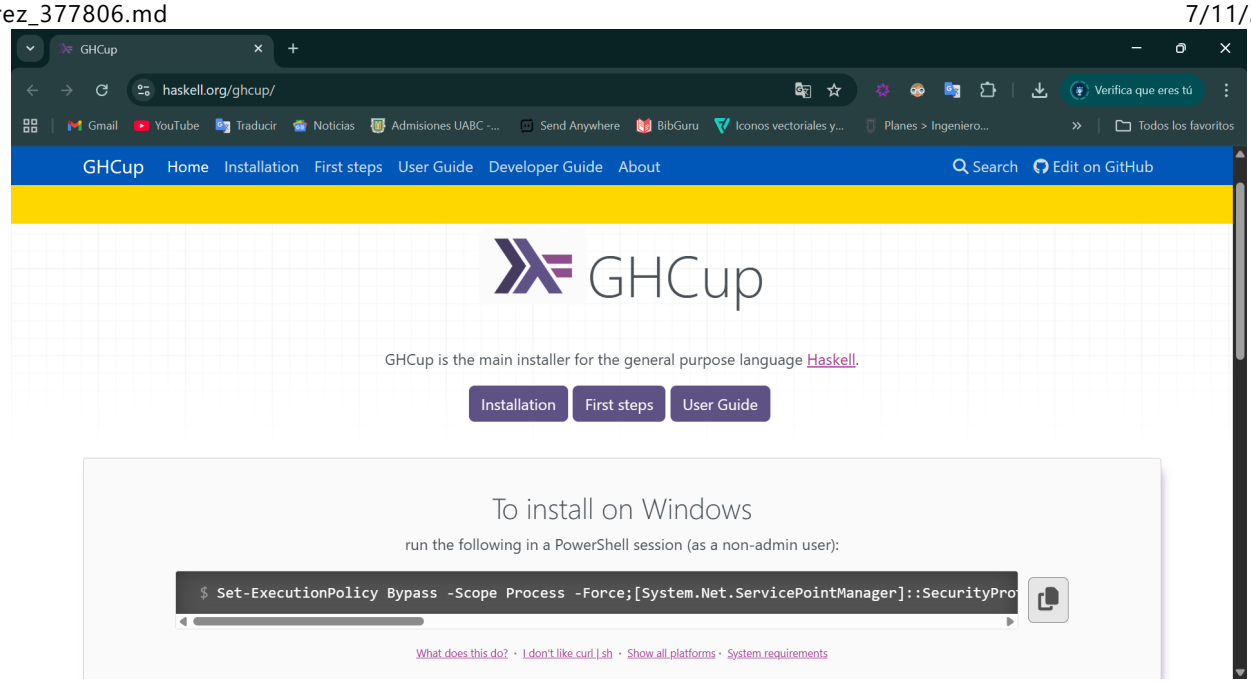
Descarga entorno

Para poder utilizar *Haskell*:

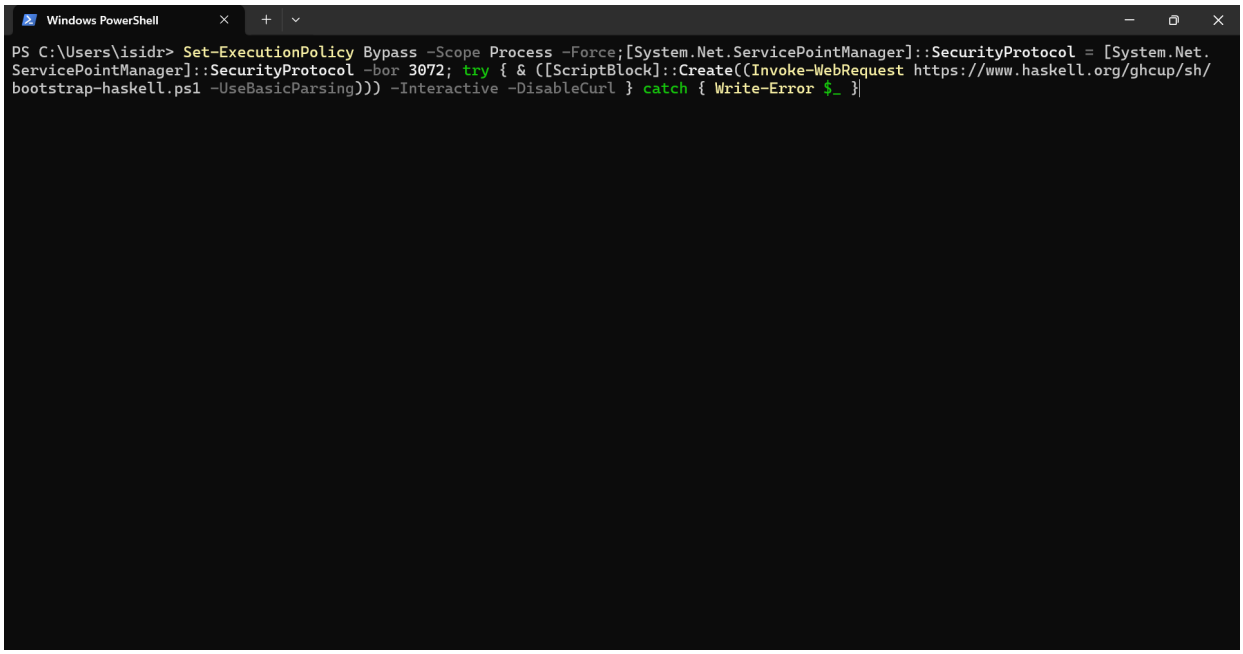
1. **Dirigete a la pagina de *Haskell* y ve a la pestaña de Descargas**
   Dentro te va aparecer esto:



2. **Pulsa en el recuadro morado "GHCup"**
   Ya dentro copia el comando que aparece aqui:



3. **Dirigete a *PowerShell* pegalo, ejecutalo y instala todo**
   Nota: no lo ejecutes como administrador.



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:

```haskell
main :: IO ()
main = do
  loadFile defaultConfig
  website <- lookupEnv "WEBSITE"

  case website of
    Nothing -> error "You should set WEBSITE at .env file."
    Just s -> do
      result <- openBrowser s
      if result
        then print ("Could open " ++ s)
        else print ("Couldn't open " ++ s)

      putStrLn "Commands:"
      putStrLn "+ <String> - Add a TODO entry"
      putStrLn "- <Int>    - Delete the numbered entry"
      putStrLn "s <Int>    - Show the numbered entry"
      putStrLn "e <Int>    - Edit te numbered entry"
      putStrLn "l          - List todo"
      putStrLn "r          - Reverse todo"
      putStrLn "c          - Clear todo"
      putStrLn "q          - Quit"
      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.

```haskell
prompt :: [String] -> IO ()
prompt todos = do
  putStrLn ""
  putStrLn "Test todo with Haskell. You can use +(create), -(delete), s(show), e(dit), l(ist), r(everse), c(lear),
  command <- getLine
  if "e" `isPrefixOf` command
    then do
      print "What is the new todo for that?"
      newTodo <- getLine
      editTodo command todos newTodo
    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_ putTodo (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.