



**UNIVERSIDAD DE SANTIAGO DE CHILE**  
FACULTAD DE INGENIERÍA  
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

**Paradigmas de programación**

Laboratorio: Paradigma funcional (Stackoverflow)

Profesores: Roberto González  
Camila Marquez

Alumna: Javiera Tapia

Santiago, Chile  
09 de noviembre de 2020

# Índice

<b>Tabla de contenidos</b>	<b>3</b>
<b>Introducción</b>	<b>4</b>
Descripción breve del problema	6
<b>Descripción breve del paradigma</b>	<b>7</b>
Análisis del problema (requisitos específicos)	8
Requerimientos funcionales	8
<b>Diseño de la solución</b>	<b>10</b>
<b>Instrucciones de uso</b>	<b>13</b>
Instrucciones generales	13
Resultados esperados	13
Posibles errores	13
<b>Resultados y autoevaluación</b>	<b>14</b>
<b>Conclusiones del trabajo</b>	<b>15</b>
<b>Referencias</b>	<b>16</b>
<b>Anexos</b>	<b>17</b>
Anexo 1. Tabla de construcción de TDAs	17
Anexo 2. Estructura de directorios principales	18
Anexo 3. Entorno DrRacket	19
Anexo 4. Resultado de pruebas	20
Función register	20
Funciones ask y login	20
Función reward	20
Función answer	21
Función accept	21
Función stack->string	22
Función search	23
Anexo 5: Implementación simbólica de TDAs	23

# Tabla de contenidos

## ❑ **Introducción**

Sección que introduce brevemente el laboratorio, objetivo principal y metas del mismo.

## ❑ **Descripción breve del problema**

Sección que plantea la problemática principal del laboratorio y sus componentes principales.

## ❑ **Análisis del problema (requisitos específicos)**

Sección que presenta los requisitos específicos y funcionales del laboratorio, junto con las restricciones del mismo.

## ❑ **Diseño de la solución**

Sección que presenta el enfoque de solución, descripciones, diagramas, descomposición de problemas, algoritmos o técnicas empleados para problemas particulares, recursos empleados, etc.

## ❑ **Aspectos de implementación**

Sección que presenta estructura del proyecto, bibliotecas empleadas, razones de su elección, compilador o intérprete usado, etc.

## ❑ **Instrucciones de uso**

Sección que presenta ejemplos, resultados esperados, posibles errores, etc.

## ❑ **Resultados y autoevaluación**

Sección que presenta un resumen de resultados obtenidos.

## ❑ **Conclusiones del trabajo**

Sección que presenta los alcances, limitaciones, dificultades de usar el paradigma para abordar el problema.

## ❑ **Referencias**

## ❑ **Anexos**

# Introducción

Los paradigmas de programación son complejos dado que influyen el diseño del lenguaje a través de un set de conceptos que nos ayudan a resolver problemas por medio de un *approach* basado en la teoría matemática.

En este laboratorio se hablará específicamente sobre el *paradigma funcional*, que concibe el mundo a través de las funciones, siendo éstas la unidades de abstracción elemental, considerados a su vez *ciudadanos de primera categoría*.

Se presentará a continuación la creación de un clon de la aplicación **Stackoverflow** --sitio de preguntas y respuestas para programadores profesionales y entusiastas de la programación--, haciendo uso del lenguaje de programación Scheme (perteneciente al marco de referencia funcional) y sus lineamientos, que permitirá la construcción pertinente de Tipos de Dato Abstracto (TDA) y su implementación.

## Descripción breve del problema

El stack tecnológico del foro Stackoverflow está, originalmente, basado en un lenguaje de programación orientado a objetos<sup>1</sup> (*server side*), sin embargo, es posible simular y reconstruir gran parte de sus componentes bajo otros lineamientos, por ejemplo, los que plantea el paradigma funcional. En esta ocasión, la problemática principal se centra en cubrir elementos claves como el manejo de usuarios registrados, preguntas, respuestas, etiquetas, etc y algunos de sus comportamientos (login, register, ask, vote, etc) que provee la plataforma real, haciendo uso del lenguaje *Scheme* (dialecto del lenguaje de programación Lisp) y sus funciones standard, donde nos encontramos enfrentados a problemáticas como la dificultad de la programación sin estado e inmutabilidad (para adoptar el approach matemático) y la elección adecuada y propicia de TDAs.

Por otra parte, la implementación estará atada a requerimientos específicos como el uso de algunas técnicas y herramientas que reconstruirán estos componentes y comportamientos de la plataforma, tales como: recursión (natural y de cola), currificación, descomposición recursiva, integración de funciones de orden superior, etc.

---

<sup>1</sup>Jeff Atwood, "What was Stackoverflow built with?", *The Overflow Blog*, <https://stackoverflow.blog/2008/09/21/what-was-stack-overflow-built-with/>

# Descripción breve del paradigma

Para pasar a describir las problemáticas y soluciones propuestas para cada uno de los conceptos requeridos por este laboratorio, es importante definir algunos elementos clave que le darán sentido y propósito a las distintas implementaciones y diseños escogidos:

1. **TDA:** los *Tipo de Datos Abstractos* (TDA) se caracterizan por sus operaciones que pueden ser clasificadas en: *constructores, funciones de pertenencia, modificadores y otras implementaciones*. Se trata de un set de operaciones y especificaciones. Un buen TDA es simple, coherente, adecuado, donde su representación es independiente<sup>2</sup>.
2. **Paradigma:** se trata de un marco de referencia, un modelo que nos permite enfrentar un problema tomando una serie de decisiones apropiadas y convenientes.
3. **Paradigma funcional:** paradigma que concibe el mundo a través de funciones.
4. **Función:** procedimiento o rutina.

El paradigma funcional busca ligar todo a la matemática pura, su estilo es declarativo donde el foco principal es *qué resolver* en contraste con el estilo imperativo donde el foco se centra en *cómo resolver*. Este paradigma usa expresiones en vez de sentencias. Una expresión es evaluada para producir un valor, mientras que la sentencia es ejecutada para asignar variables.

Como futuros ingenieros debemos conocer estos términos, dado que se utilizan de forma frecuente en la industria. El propósito fundamental de todas estas ideas es ayudar a alcanzar tres propiedades que nos importan: seguridad frente a los bugs, facilidad de comprensión y lectura para cambios posteriores.

---

<sup>2</sup> Amarasinghe Saman, Chlipala Adam, "Abstract Data Types", *Massachusetts Institute of Technology*, <http://web.mit.edu/6.005/www/fa14/classes/08-abstract-data-types/>

# Análisis del problema (requisitos específicos)

El laboratorio requiere de la implementación de diversas funciones obligatorias que cumplen con la simulación del software Stackoverflow bajo el alero del paradigma funcional. El diseño de abstracciones y las características de éstas son fundamentales en primera instancia, puesto que las funciones requeridas deben combinarse de forma apropiada y actuar de forma consecuente al diseño planteado.

## Requerimientos funcionales

- 1) **Register:** función que registra un nuevo usuario con sus respectivas credenciales (username y password), validación y verificación que evitarán duplicidad de usuarios registrados (usando username como identificador único) y uso implícito de recursión natural. Esta función se implementa como parte del TDA Stack, donde los usuarios registrados serán representados como la primera lista dentro de la lista Stack.
- 2) **Login:** función que autentica a un usuario previamente registrado (dentro del stack), para lo anterior, se verifica la existencia del usuario en la lista Stack haciendo uso implícito de recursión natural. Si la autenticación es válida (username y password son correctos), se procede a retornar una función currificada de la **operación** (ask, accept, reward, etc) parcialmente evaluada, de lo contrario, es la operación misma (procedure) la que es retornada. El usuario autenticado se agrega a la posición inicial de la lista Stack (string), puesto que será un elemento temporal y permitirá la fácil manipulación del mismo en el resultados de las operaciones.
- 3) **Ask:** función currificada que validará al usuario correctamente logueado (que se reconocerá fácilmente como el primer elemento del tipo string en la lista Stack). Esta función retornará el stack actualizado que incluirá esta pregunta y sus características, eliminando la sesión del usuario en su ejecución.

- 4) **Reward:** función que debe implementar currificación para permitir a un usuario logueado ofrecer una recompensa a una determinada pregunta. Se debe aplicar la lógica de puntaje asociada a esta acción. El retorno es un stack actualizado sin el usuario logueado.
- 5) **Answer:** función que debe implementar currificación para permitir a un usuario logueado responder a una pregunta y vincularla a ella. El retorno es un stack actualizado que incluirá la respuesta sin el usuario logueado.
- 6) **Accept:** función que debe implementar currificación y permitir a un usuario logueado aceptar una respuesta a una de sus preguntas, se debe validar que la pregunta pertenezca a él. El retorno será un stack actualizado que actualizará los puntajes que correspondan a esta acción y que no incluirá al usuario logueado.
- 7) **Stack -> string:** función que recibirá un stack y entregará su representación como un string que será comprensible a un usuario dada su estructura. Se integran saltos de línea “\n” para mejorar su visualización.
- 8) **Search:** función que permitirá buscar preguntas y respuestas en base a una coincidencia parcial de texto en su contenido o etiquetas (si es requerido). El scope de búsqueda será definido a partir de la función llamada **all** o **labels**, a partir de esto, se deberá filtrar el stack acorde.



## Diseño de la solución

Respecto al diseño de la solución planteada para este laboratorio, se tomó como idea principal el desarrollo correcto y apropiado de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software. Idear y conceptualizar los requerimientos *a priori*, siguiendo un orden claro nos facilita la implementación de código, sin lugar a dudas. Antes de programar, es necesario analizar y aplicar una arquitectura correcta para evitar refactorizaciones innecesarias, documentar nuestro código (cabeceras de funciones, dominio, recorrido y descripción), aplicando buenas prácticas que no sólo nos favorece a quienes desarrollamos, sino también a nuestros colegas.

De acuerdo a la idea anterior, se esquematiza la abstracción del Stack como una **lista de listas** (ver Anexo 5: *implementación simbólica de TDAs*) o contenedor de data. Sin embargo, eso no fue suficiente, también se requería reservar listas vacías que aseguran la posición de cada estructura dentro de este contenedor, esta no puede ser arbitraria ya que el acceso a cada una de ellas debe estar definido, evitando efectos secundarios y errores de compilación.

Respecto a la descomposición de los problemas, cada requerimiento funcional se manejó como un subcomponente y se trabajó de forma aislada, sin embargo, se reconoció que la mayoría de las acciones debían ser una consecuencia del login correcto de un usuario. Por otra parte, se integró la técnica de *lazy loading* (evaluación perezosa) para operaciones costosas y funciones repetitivas.

Por otra parte, se aplicaron técnicas de recursión de cola y natural (vistas en clases) para las funciones auxiliares y, también, cuando fue parte de los requerimientos funcionales.

Finalmente, los recursos empleados fueron principalmente los propios de la programación funcional, algunas funciones de orden superior y la documentación tanto de los lenguajes Scheme como Racket.

# Aspectos de implementación

## Estructura del proyecto

La estructura del proyecto se basa en la modularización de cada TDA elegido, para ello, cada archivo maneja su propia lógica, siguiendo los principios de encapsulación. Adicionalmente, se provee de un archivo `utils.rkt`, donde se definen funciones auxiliares que se utilizarán en uno o más TDAs. (Ver Anexo 2: *Estructura de directorios principales*). Es importante señalar que se provee de un archivo `main.rkt` que será de uso exclusivo del usuario, donde se exponen los ejemplos y casos de uso para cada requerimiento funcional.

## Bibliotecas empleadas

Racket y Scheme proveen de muchas utilidades (tanto imperativas como funcionales) que están disponibles en su documentación, sin embargo, se limitará (según requerimiento) el uso sólo a aquellas de naturaleza funcional, específicamente las que se detallan a continuación:

- Librería *Standard* del lenguaje Scheme (R6RS): específicamente, *base library* (*rnrs base*).
- Librería *Standard* del lenguaje Scheme (R6RS): algunos *procedures/utilidades* de listas permitidos por el laboratorio como: **filter** y **append** (*rnrs list*).

Algunas de los *procedures* disponibles se recrearon *from scratch* como funciones auxiliares dentro del archivo `utils.rkt`

## Detalles del compilador e intérprete

Racket JIT compiler y DrRacket v7.8 como Integrated Development Environment (IDE).

# Instrucciones de uso

## Instrucciones generales

- Instalación de DrRacket (v7.8) (<https://download.racket-lang.org/>) en máquina local.
- Correr la aplicación y abrir archivo `lab_git_18005106_TapiaBobadilla_main.rkt`, donde se ubican los casos pruebas.
- Clickear en botón RUN, esquina superior derecha del IDE (ver *Anexo 3: Entorno DrRacket*).

## Ejemplos

Se definen 3 ejemplos para cada uno de los requerimientos funcionales y, en algunos casos, la implementación de ejemplos de casos específicos (usuario no logueado intentando preguntar, intento de registro de un usuario previamente registrado, etc.) Los resultados de cada ejemplo funcionarán como entrada de los siguientes, esto para demostrar el correcto funcionamiento de creación múltiple y, además, para ejecutar acciones como las de recompensa con un stack que ya posea preguntas previamente creadas. (ver *Anexo 4: Resultado de pruebas*).

## Resultados esperados

Se espera que cada uno de los ejemplos cumpla las expectativas mencionadas en los requerimientos funcionales, exponiendo una estructura coherente y clara.

## Posibles errores

No se reconocen posibles errores, puesto que los ejemplos se ajustan a los requerimientos. Se recomienda mantener la estructura de los ejemplos para evitar efectos secundarios en la ejecución de las funciones.

## Resultados y autoevaluación

Se presenta a continuación un resumen de los resultados obtenidos a través de este laboratorio, los cuales serán plasmados y presentados en la siguiente tabla:

Requerimientos	Grado de alcance (%)	Pruebas realizadas	% de pruebas exitosas	% de pruebas fracasadas (*3)
Register	100%	(3) casos de registros exitosos y (1) caso particular (validación existencia).	100%	0%
Login	100%	(3) casos de login exitosos y (1) caso particular (validación de usuario registrado).	100%	0%
Ask	100%	(3) casos de creación de preguntas exitosas y (1) caso particular (creación de preguntas múltiples).	100%	0%
Reward	100%	(3) casos de creación de recompensas exitosas y (1) caso particular (recompensa excede reputación del usuario).	100%	0%
Answer	100%	(3) casos de creación de preguntas exitosas y (2) casos particulares (creación múltiple de respuestas y validación de respuesta con referencia a pregunta no existente).	100%	0%
Accept	100%	(2) casos de aceptación de respuestas caso particular y (1) caso que valida la aceptación de una respuesta perteneciente al usuario.	75%	15%
Stack->string	100%	(2) casos expuestos con <b>display</b> exitosamente y (1) caso particular (usuario logueado).	100%	0%
Search	100%	(3) casos de búsquedas exitosas con y sin login de usuario. Se incluye la búsqueda <b>all</b> y <b>labels</b> requeridas usando pattern matching.	100%	0%

En la tabla anterior, no se presentan las funciones que no se completaron dado que en el laboratorio se trabajaron todos los requerimientos obligatorios y **uno** de aquellos opcionales (Search) en su totalidad.

<sup>3</sup> Durante el desarrollo del laboratorio, muchas pruebas fueron un fracaso, sin embargo, se trabajó en reducir al máximo los bugs y efectos secundarios hasta lograr los resultados esperados.

## Conclusiones del trabajo

Este trabajo ha permitido conocer en detalle el alcance del paradigma funcional, el que podemos ver se basa más en el “*qué hacer*” y no en “*cómo se debe hacer*”, lo que permite que cualquier persona tenga la posibilidad de comprender, fácilmente, un código de estas características. Cabe mencionar que también hay algunas ventajas del paradigma, en las que se pueden mencionar: código libre de errores y fallas, gran rendimiento, admisión de funciones anidadas, reutilización –debido a que tiene unidades independientes– y otras tantas. Sin embargo, y, como todo, hay una desventaja, siendo la demanda de una gran cantidad de memoria, uno de sus más grandes punto de inflexión.

Asimismo, cabe resaltar que en el paradigma hay una interesante característica de recursión y currificación, siendo la primera bastante elemental, pues permite hacer las iteraciones a través de esa función. Mientras que el segundo, nos retorna una función. Respecto a las dificultades, se puede mencionar que la carencia de shortcuts y tooling del IDE *DrRacket* realmente ralentizaron el desarrollo del trabajo. También, conviene decir que hubo dificultades al momento de trabajar con la inmutabilidad y la programación sin estados, ni variables.

A pesar de todo, hay cosas positivas que reconocer, como el haber trabajado con un lenguaje fuertemente orientado a la matemática, lo que permitió conocer más profundamente el paradigma funcional y lenguajes no tan populares como Scheme y Racket. En ese sentido, mencionar la importancia del uso de Git en las tareas diarias que tiene cualquier desarrollador, pues permite organizar mejor nuestras ideas y trabajos.

Finalmente, no hay que dejar de mencionar que los informáticos y desarrolladores debemos tener buenas prácticas al momento de trabajar, pues no solo nos beneficia a nosotros sino al resto de personas que acompañan en la labor. Asimismo, hacer un diseño apropiado *a priori* hace que el desarrollo y la implementación final sea sencilla y elegante.

# Referencias

Atwood, Jeff. (21 de septiembre de 2008). *What was Stackoverflow built with?*. The Overflow Blog.

<https://stackoverflow.blog/2008/09/21/what-was-stack-overflow-built-with/>

Amarasinghe, Saman., Chlipala, Adam., Devadas, Srinii., Ernst, Michael., Goldman, Max., Guttag, John., Jackson, Daniel., Miller, Rob., Solar-Lezama, Armando. (s.f.). *Abstract Data Types*. MIT.

<http://web.mit.edu/6.005/www/fa14/classes/08-abstract-data-types/>

Algunos aspectos de este trabajo fueron vistos de manera general en (<https://stackoverflow.blog>)

Documentación Racket extraída desde (<https://docs.racket-lang.org>)

Documentación Scheme extraída desde (<https://www.gnu.org/software/mit-scheme/>)

JournalDev. (s.f.). *Compare Functional Programming, Imperative Programming and Object Oriented Programming*.

<https://www.journaldev.com/8693/functional-imperative-object-oriented-programming-comparison>

Van Roy, Peter. (Abril de 2012). *Programming Paradigms for Dummies: What Every Programmer Should Know*. Researchgate.

[https://www.researchgate.net/publication/241111987\\_Programming\\_Paradigms\\_for\\_Dummies\\_What\\_Every\\_Programmer\\_Should\\_Know](https://www.researchgate.net/publication/241111987_Programming_Paradigms_for_Dummies_What_Every_Programmer_Should_Know)

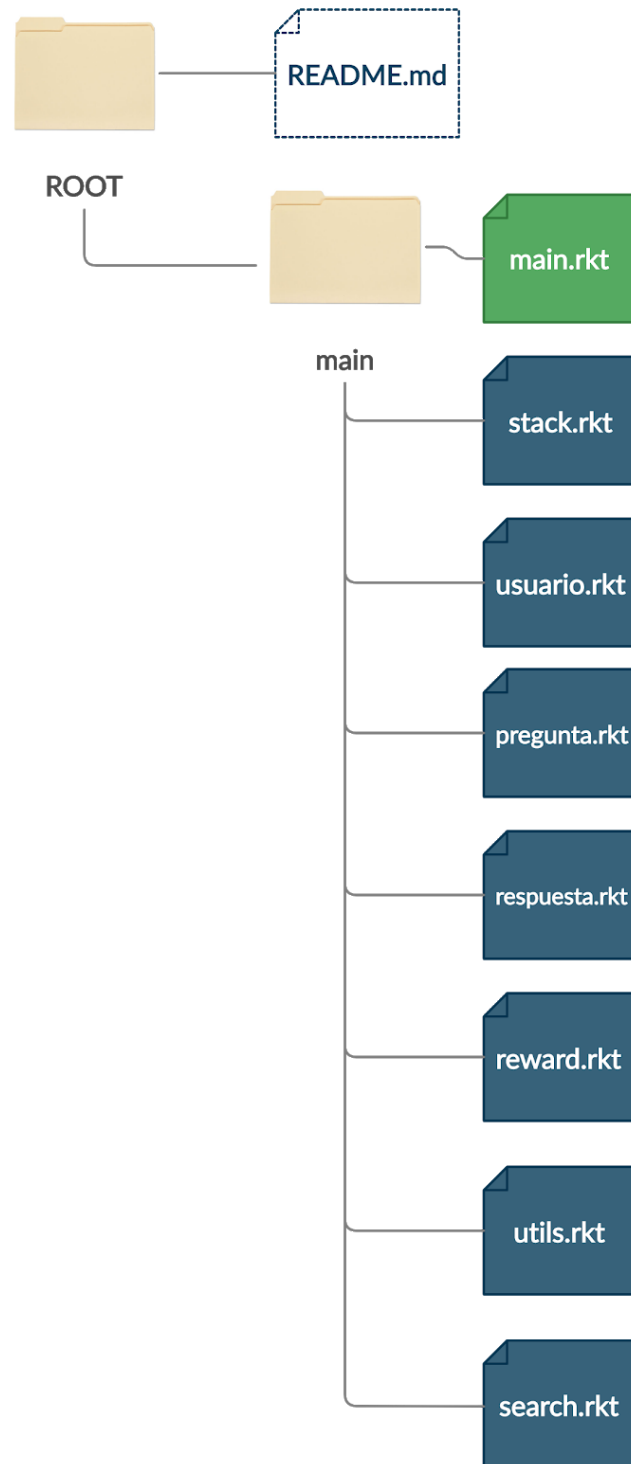
# Anexos

## Anexo 1. Tabla de construcción de TDAs

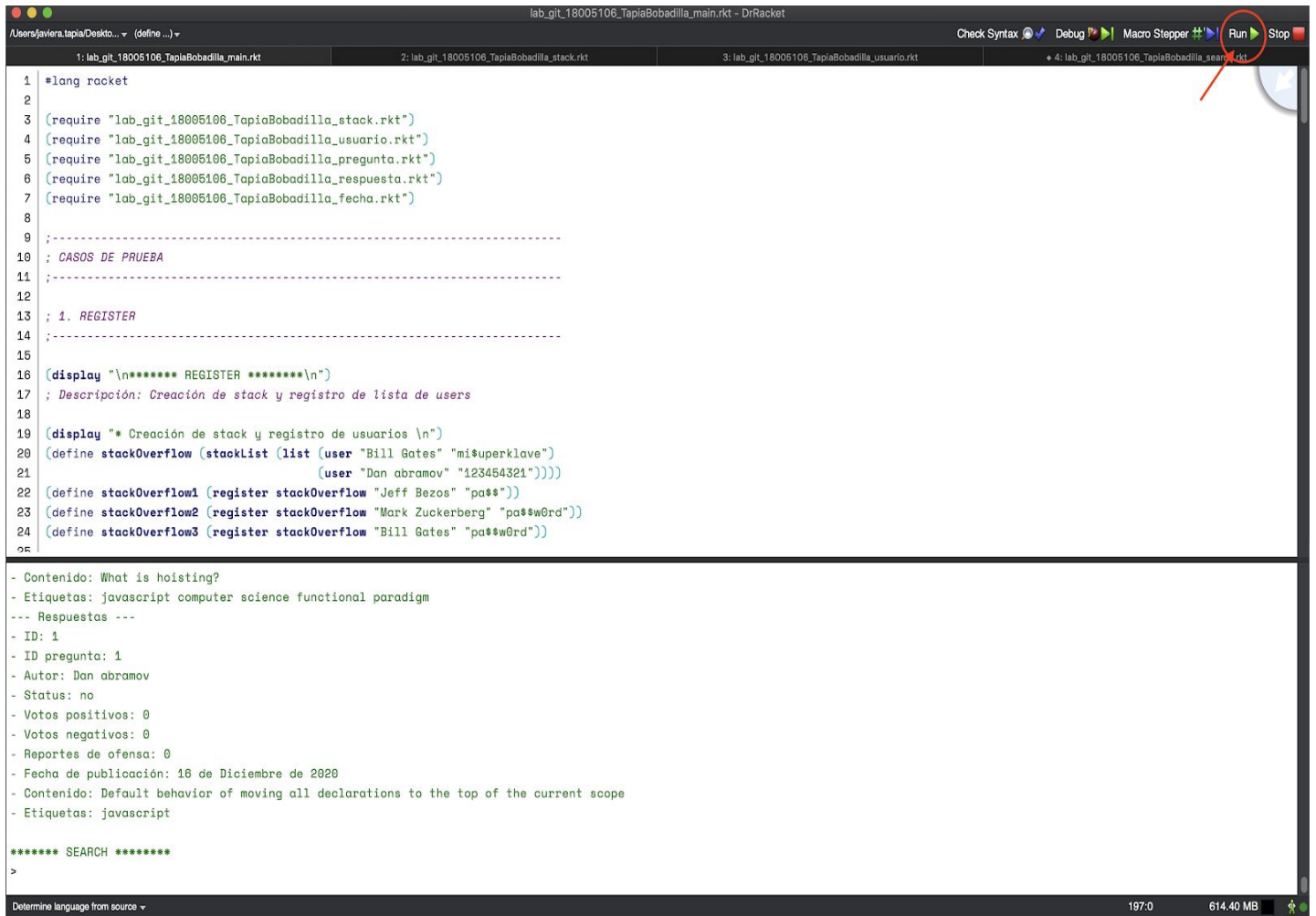
Abstracción	Contexto	Características (Qué)
Stack	Estructura base que funciona como contenedor de usuarios, preguntas, recompensas, respuestas y búsqueda.	Lista de listas que se encarga de contener una serie de estructuras (del tipo lista), tales como: usuarios, preguntas, recompensas, respuestas y búsqueda dentro de la plataforma.
Usuario (User)	Persona que publica, edita y elimina tanto preguntas como respuestas, ofrece recompensas, vota, reporta, entre otros.	<ul style="list-style-type: none"> <li>- Lista que posee credenciales de acceso y reputación</li> <li>- Ofrece reputación, pregunta y responde, da ranking, etc. dentro de la plataforma.</li> </ul>
Pregunta (Question)	Enunciado interrogativo realizado por un usuario sobre diversas tecnologías relacionadas a la ingeniería de software en la plataforma.	<ul style="list-style-type: none"> <li>- Lista que puede ser editada, modificada y eliminada.</li> <li>- Es creada por un usuario y recibe ranking por medio del mismo.</li> <li>- Posee textos con formato (negrita, cursiva) imágenes, enlaces y código</li> <li>- Posee un ID, votaciones (positivas y negativas), visualizaciones, etiquetas, título, contenido, fecha de publicación, fecha de última modificación , etc.</li> <li>- Es referenciada por una respuesta a través de su ID.</li> </ul>
Respuesta (Answer)	Enunciado que responde a una pregunta realizada por un usuario sobre diversas tecnologías relacionadas a la ingeniería de software en la plataforma	<ul style="list-style-type: none"> <li>- Lista que recibe ranking y referencia una pregunta.</li> <li>- Es creada por un usuario, quien puede recibir una bonificación (reputación) si esta es aceptada.</li> <li>- Posee un autor (usuario), fecha de publicación, votos (favor y contra), estado de aceptación (sí/no), reportes de ofensa y categorías (mejor respuesta, peor respuesta, etc).</li> </ul>
Recompensa (Reward)	Bonificación que genera un usuario y que compensa la utilidad de una pregunta dentro de la plataforma.	<ul style="list-style-type: none"> <li>- Lista que posee una descripción, id de pregunta y usuario que la otorgó.</li> <li>- Bonifica una pregunta y posee diversas reglas en su aplicación.</li> </ul>
Búsqueda (Search)	Motor de búsqueda dentro de la plataforma que resulta de utilidad para encontrar información rápidamente por medio de una coincidencia parcial de texto.	<ul style="list-style-type: none"> <li>- Búsqueda por etiqueta (entre corchetes)</li> <li>- Búsqueda palabra exacta (entre comillas)</li> <li>- Búsqueda por cantidad de respuestas (answers: numero_x)</li> <li>- Búsqueda por puntaje (score: numero_x)</li> </ul>



## Anexo 2. Estructura de directorios principales



### Anexo 3. Entorno DrRacket



The screenshot shows the DrRacket IDE interface. The top toolbar includes buttons for 'Check Syntax', 'Debug', 'Macro Stepper', 'Run' (highlighted with a red circle and arrow), and 'Stop'. Below the toolbar, four source files are listed: '1: lab\_git\_18005106\_TapiaBobadilla\_main.rkt', '2: lab\_git\_18005106\_TapiaBobadilla\_stack.rkt', '3: lab\_git\_18005106\_TapiaBobadilla\_usuario.rkt', and '4: lab\_git\_18005106\_TapiaBobadilla\_search.rkt'. The main editor displays the content of 'lab\_git\_18005106\_TapiaBobadilla\_main.rkt', which includes Racket code for a 'register' function and a 'search' function. The output window at the bottom shows the results of running the code, including a message about 'What is hoisting?' and a search result for 'SEARCH'.

```
1 #lang racket
2
3 (require "lab_git_18005106_TapiaBobadilla_stack.rkt")
4 (require "lab_git_18005106_TapiaBobadilla_usuario.rkt")
5 (require "lab_git_18005106_TapiaBobadilla_pregunta.rkt")
6 (require "lab_git_18005106_TapiaBobadilla_respuesta.rkt")
7 (require "lab_git_18005106_TapiaBobadilla_fecha.rkt")
8
9 ;-----
10 ; CASOS DE PRUEBA
11 ;-----
12
13 ; 1. REGISTER
14 ;-----
15
16 (display "\n***** REGISTER *****\n")
17 ; Descripción: Creación de stack y registro de lista de users
18
19 (display "* Creación de stack y registro de usuarios \n")
20 (define stackOverflow (stackList (list (user "Bill Gates" "mi$uperklave")
21                                         (user "Dan abramov" "123454321"))))
22 (define stackOverflow1 (register stackOverflow "Jeff Bezos" "pa$$"))
23 (define stackOverflow2 (register stackOverflow "Mark Zuckerberg" "pa$$w0rd"))
24 (define stackOverflow3 (register stackOverflow "Bill Gates" "pa$$w0rd"))
25
```

- Contenido: What is hoisting?  
- Etiquetas: javascript computer science functional paradigm  
--- Respuestas ---  
- ID: 1  
- ID pregunta: 1  
- Autor: Dan abramov  
- Status: no  
- Votos positivos: 0  
- Votos negativos: 0  
- Reportes de ofensa: 0  
- Fecha de publicación: 16 de Diciembre de 2020  
- Contenido: Default behavior of moving all declarations to the top of the current scope  
- Etiquetas: javascript

\*\*\*\*\* SEARCH \*\*\*\*\*  
>

## Anexo 4. Resultado de pruebas

### Función register

```
***** REGISTER *****
* Creación de stack y registro de usuarios
1)'(((("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0)) () () ())
2)'(((("Jeff Bezos" "pa$$" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0)) () () ())
3)'(((("Mark Zuckerberg" "pa$$w0rd" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0)) () () ())
* Caso específico: Usuario sin poder registrarse (validación de username)
4)'(((("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0)) () () ())
```

```
***** ASK Y LOGIN *****
* Login y creación de preguntas con usuario logueado
1)'(((("Jeff Bezos" "pa$$" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0))
  ((1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ()
  ())
2)'(((("Mark Zuckerberg" "pa$$w0rd" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0))
  ((1 "abierta" 0 0 0 "" (30 10 2020) "Mark Zuckerberg" (30 10 2020) "How to create a web app using php?" ("php" "web app"))))
  ()
  ())
3)'(((("Jeff Bezos" "pa$$" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0))
  ((1 "abierta" 0 0 0 "" (30 10 2020) "Dan abramov" (30 10 2020) "Is it better to use React hooks than life?" ("react" "javascript" "hooks"))))
  ()
  ())
* Caso específico: Creación de pregunta sin usuario logueado (retorna stack)
4)'(((("Jeff Bezos" "pa$$" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0)) () () ())
* Extra: Creación de multiples preguntas conservando stack
5)'(((("Jeff Bezos" "pa$$" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
   (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ()
  ())
```

### Funciones ask y login

### Función reward

```
***** REWARD *****
* Login y creación de recompensas con usuario logueado
1)'(((("Jeff Bezos" "pa$$" 100 0) ("Dan abramov" "123454321" 100 0) ("Bill Gates" "mi$uperklave" 100 -50))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
   (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  (("Bill Gates" 2 50))
  ()
  ())
2)'(((("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0) ("Jeff Bezos" "pa$$" 100 -75))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
   (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  (("Jeff Bezos" 2 75))
  ()
  ())
3)'(((("Jeff Bezos" "pa$$" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 -55))
  ((3 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What is dijkstra algorithm?" ("algorithms" "computer science"))
   (2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
   (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  (("Dan abramov" 2 55))
  ()
  ())
* Caso específico: Recompensa excede reputación de usuario (retorna stack)
4)'(((("Jeff Bezos" "pa$$" 100 0) ("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
   (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ()
  ())
```

## Función answer

```
***** ANSWER *****
* Login y creación de respuestas con usuario logueado
1)'(((("Bill Gates" "mi$uperklave" 100 0) ("Jeff Bezos" "pa$$" 100 -75) ("Dan abramov" "123454321" 100 -80))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
    (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ((("Dan abramov" 3 80) ("Jeff Bezos" 2 75))
    ((1 0 0 "no" 0 "Jeff Bezos" 2 (30 10 2020) "Refers to the current context of code, which determines the accessibility" ("javascript")))))
2)'(((("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0) ("Jeff Bezos" "pa$$" 100 -97))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
    (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ((("Jeff Bezos" 1 22) ("Jeff Bezos" 2 75))
    ((1 0 0 "no" 0 "Jeff Bezos" 1 (15 11 2020) "I don't know what it means but i think is cool" ("javascript")))))
3)'(((("Bill Gates" "mi$uperklave" 100 0) ("Jeff Bezos" "pa$$" 100 -75) ("Dan abramov" "123454321" 100 -80))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
    (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ((("Dan abramov" 3 80) ("Jeff Bezos" 2 75))
    ((1 0 0 "no" 0 "Dan abramov" 1 (16 12 2020) "Default behavior of moving all declarations to the top of the current scope" ("javascript")))))
* Caso específico: Creación de respuesta inválido con una pregunta inexistente (retorna stack)
4)'(((("Bill Gates" "mi$uperklave" 100 0) ("Dan abramov" "123454321" 100 0) ("Jeff Bezos" "pa$$" 100 -97))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
    (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ((("Jeff Bezos" 1 22) ("Jeff Bezos" 2 75))
    ((2 0 0 "no" 0 "Dan abramov" 2 (13 10 2020) "Scope in JavaScript refers to the current context of code" ("computer science" "javascript"))
      (1 0 0 "no" 0 "Jeff Bezos" 1 (15 11 2020) "I don't know what it means but i think is cool" ("javascript")))))
* Extra: Creación de multiples respuestas conservando stack
5)'(((("Bill Gates" "mi$uperklave" 100 0) ("Jeff Bezos" "pa$$" 100 -75) ("Dan abramov" "123454321" 100 -80))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
    (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ((("Dan abramov" 3 80) ("Jeff Bezos" 2 75))
    ((2 0 0 "no" 0 "Bill Gates" 2 (13 10 2020) "Scope in JavaScript refers to the current context of code" ("javascript"))
      (1 0 0 "no" 0 "Dan abramov" 1 (16 12 2020) "Default behavior of moving all declarations to the top of the current scope" ("javascript")))))
```

## Función accept

```
***** ACCEPT *****
* Login y aceptación con usuario logueado
1)'(((("Dan abramov" "123454321" 100 -80) ("Jeff Bezos" "pa$$" 27 0) ("Bill Gates" "mi$uperklave" 115 0))
  ((2 "cerrada" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
    (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ((("Dan abramov" 3 80))
    ((2 0 0 "si" 0 "Bill Gates" 2 (13 10 2020) "Scope in JavaScript refers to the current context of code" ("javascript"))
      (1 0 0 "no" 0 "Dan abramov" 1 (16 12 2020) "Default behavior of moving all declarations to the top of the current scope" ("javascript")))))
2)'(((("Dan abramov" "123454321" 100 0) ("Bill Gates" "mi$uperklave" 102 0) ("Jeff Bezos" "pa$$" 93 -75))
  ((1 "cerrada" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))
    (2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))))
  ((("Jeff Bezos" 2 75))
    ((1 0 0 "si" 0 "Jeff Bezos" 1 (15 11 2020) "I don't know what it means but i think is cool" ("javascript"))
      (2 0 0 "no" 0 "Dan abramov" 2 (13 10 2020) "Scope in JavaScript refers to the current context of code" ("computer science" "javascript")))))
* Caso específico: Usuario intenta aceptar pregunta que no lo pertenece. Retorna stack
3)'(((("Bill Gates" "mi$uperklave" 100 0) ("Jeff Bezos" "pa$$" 100 -75) ("Dan abramov" "123454321" 100 -80))
  ((2 "abierta" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
    (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))))
  ((("Dan abramov" 3 80) ("Jeff Bezos" 2 75))
    ((2 0 0 "no" 0 "Bill Gates" 2 (13 10 2020) "Scope in JavaScript refers to the current context of code" ("javascript"))
      (1 0 0 "no" 0 "Dan abramov" 1 (16 12 2020) "Default behavior of moving all declarations to the top of the current scope" ("javascript")))))
```



## Función **stack->string**

\*\*\*\*\* STACK->STRING \*\*\*\*\*

1)

--- Usuarios ---

- Username: Dan abramov
- Password: 123454321
- Username: Jeff Bezos
- Password: pa\$\$
- Username: Bill Gates
- Password: mi\$uperklave

--- Preguntas ---

- ID: 2
- Estado: cerrada
- Votos positivos: 0
- Votos negativos: 0
- Visualizaciones: 0
- Título:
- Fecha última modificación: 30 de Septiembre de 2020
- Autor: Jeff Bezos
- Fecha de publicación: 30 de Septiembre de 2020
- Contenido: What does scope mean?
- Etiquetas: javascript
- ID: 1
- Estado: abierta
- Votos positivos: 0
- Votos negativos: 0
- Visualizaciones: 0
- Título:
- Fecha última modificación: 30 de Octubre de 2020
- Autor: Bill Gates
- Fecha de publicación: 30 de Octubre de 2020
- Contenido: What is hoisting?
- Etiquetas: javascript computer science functional paradigm

- Recompensas ---

Usuario: Dan abramov

ID pregunta: 3

Cantidad recompensa: 80

- Respuestas ---

ID: 2

ID pregunta: 2

Autor: Bill Gates

Status: si

Votos positivos: 0

Votos negativos: 0

Reportes de ofensa: 0

Fecha de publicación: 13 de Octubre de 2020

Contenido: Scope in JavaScript refers to the current context of code

Etiquetas: javascript

ID: 1

ID pregunta: 1

Autor: Dan abramov

Status: no

Votos positivos: 0

Votos negativos: 0

Reportes de ofensa: 0

Fecha de publicación: 16 de Diciembre de 2020

Contenido: Default behavior of moving all declarations to the top of the current scope

Etiquetas: javascript

3)

\* Caso específico: Antecedentes de usuario logueado.

\*\*\* Información Usuario logueado: Dan abramov \*\*\*

--- Usuarios ---

- Username: Dan abramov
- Password: 123454321

--- Preguntas ---

- ID: 2
- Estado: abierta
- Votos positivos: 0
- Votos negativos: 0
- Visualizaciones: 0
- Título:
- Fecha última modificación: 30 de Septiembre de 2020
- Autor: Jeff Bezos
- Fecha de publicación: 30 de Septiembre de 2020
- Contenido: What does scope mean?
- Etiquetas: javascript
- ID: 1
- Estado: abierta
- Votos positivos: 0
- Votos negativos: 0
- Visualizaciones: 0
- Título:
- Fecha última modificación: 30 de Octubre de 2020
- Autor: Bill Gates
- Fecha de publicación: 30 de Octubre de 2020
- Contenido: What is hoisting?
- Etiquetas: javascript computer science functional paradigm

--- Respuestas ---

ID pregunta: 1

Autor: Dan abramov

Status: no

Votos positivos: 0

Votos negativos: 0

Reportes de ofensa: 0

Fecha de publicación: 16 de Diciembre de 2020

Contenido: Default behavior of moving all declarations to the top of the current scope

Etiquetas: javascript

## Función search

```
***** SEARCH *****
1) Búsqueda en preguntas, respuestas y etiquetas por la palabra 'javascript'
'((2 "cerrada" 0 0 0 "" (30 9 2020) "Jeff Bezos" (30 9 2020) "What does scope mean?" ("javascript"))
  (1 "abierta" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))
  (2 0 0 "si" 0 "Bill Gates" 2 (13 10 2020) "Scope in JavaScript refers to the current context of code" ("javascript"))
  (1 0 0 "no" 0 "Dan abramov" 1 (16 12 2020) "Default behavior of moving all declarations to the top of the current scope" ("javascript"))))
2) Búsqueda en etiquetas por la palabra 'computer science'
'((1 "cerrada" 0 0 0 "" (30 10 2020) "Bill Gates" (30 10 2020) "What is hoisting?" ("javascript" "computer science" "functional paradigm"))
  (2 0 0 "no" 0 "Dan abramov" 2 (13 10 2020) "Scope in JavaScript refers to the current context of code" ("computer science" "javascript"))))
3) Búsqueda en preguntas, respuestas y etiquetas por la palabra 'current' con login
'((2 0 0 "no" 0 "Dan abramov" 2 (13 10 2020) "Scope in JavaScript refers to the current context of code" ("computer science" "javascript"))))
```

## Anexo 5: Implementación simbólica de TDAs

