# Informe de Laboratorio I Paradigma Funcional: Scheme

Oscar Henríquez González

17 de Abril de 2023

**Paradigmas de programación  
Profesor:** Gonzalo Martínez

## Tabla de contenido:

[Informe de Laboratorio I Paradigma Funcional: Scheme 1](#_Toc133185792)

[Tabla de contenido: 2](#_Toc133185793)

[Introducción 4](#_Toc133185794)

[Descripción breve del problema 4](#_Toc133185795)

[Análisis del problema 5](#_Toc133185796)

[Diseño de la solución 5](#_Toc133185797)

[Aspectos de implementación 6](#_Toc133185798)

[Instrucciones de uso 6](#_Toc133185799)

[Resultados y autoevaluación: 7](#_Toc133185800)

[Conclusiones del trabajo 8](#_Toc133185801)

[Referencias 8](#_Toc133185805)

## Introducción

El siguiente laboratorio nos propone como desafío la implementación de un sistema de archivos, este tipo de software es fundamental en un sistema operativo para gestionar el almacenamiento de datos, se enfoca en la organización superficial de archivos y carpetas.

El sistema debe permitir múltiples usuarios, quienes pueden acceder a diferentes unidades y almacenar archivos de cualquier tipo y tamaño, considerando gran parte de las funciones de un sistema de archivos real como listar archivos, recorrer directorios, **e**tc.

## Descripción breve del problema

Considerando el problema desde el punto de vista de un paradigma funcional de programación tiene implicaciones importantes en la forma en que se aborda el desarrollo de este sistema, como:

Procurar la Inmutabilidad: En Scheme, las estructuras de datos son inmutables por defecto, lo que significa que cada modificación a una estructura de datos crea una nueva versión de la misma.

Usar funciones puras: En Scheme, las funciones puras no tienen efectos secundarios, lo que significa que no modifican el estado del sistema ni dependen del estado de otros componentes. Esto permite desarrollar funciones que realicen operaciones en el sistema de archivos sin modificar directamente su estado, lo que facilita la creación de código modular y la reutilización de funciones.

Usar recursividad: La recursividad es una característica fundamental del paradigma funcional de programación, y puede ser muy útil en el desarrollo de un sistema de archivos. Por ejemplo, se puede utilizar la recursividad para recorrer árboles de directorios y archivos, lo que permite implementar operaciones como listar, buscar y cambiar de directorio.

Abstracciones de alto nivel: Scheme ofrece varias abstracciones de alto nivel que pueden simplificar la implementación de un sistema de archivos. Por ejemplo, se pueden utilizar macros para crear funciones que implementen operaciones comunes en el sistema de archivos, como copiar, mover y eliminar archivos.

En resumen, el paradigma funcional de programación puede facilitar la implementación de un sistema de archivos en Scheme al proporcionar herramientas como la inmutabilidad, funciones puras, recursividad y abstracciones de alto nivel. Esto permite crear código modular, reutilizable y fácil de mantener

## Análisis del problema

Para analizar este problema nos vamos a enfocar en los principales desafíos que nos significara realizar este desarrollo bajo el paradigma funcional:

Estructuras de datos: Seleccionar estructuras de datos adecuadas que permitan representar y manipular archivos y directorios de manera eficiente, manteniendo al mismo tiempo la inmutabilidad y evitando efectos secundarios.

Operaciones en el sistema de archivos: Implementar funciones para realizar operaciones básicas en el sistema de archivos, como crear, listar y eliminar archivos y directorios. Estas funciones deben ser diseñadas de acuerdo con los principios funcionales, evitando mutaciones y efectos secundarios.

Navegación y búsqueda: Diseñar funciones que permitan la navegación y búsqueda dentro del sistema de archivos emulado. Esto incluye funciones para desplazarse entre directorios, encontrar archivos o directorios específicos, y listar el contenido de un directorio.

Gestión de errores: Implementar un manejo adecuado de errores para situaciones comunes, como intentar acceder a un archivo inexistente o intentar eliminar un directorio que no está vacío. Las funciones de manejo de errores también deben adherirse a los principios del paradigma funcional.

Persistencia: Si se requiere que el sistema de archivos persista los datos entre sesiones, es necesario diseñar una estrategia para guardar y recuperar el estado del sistema de archivos en memoria.

## Diseño de la solución

(presentar su enfoque de solución, describir, diagramar, descomposición de problemas, algoritmos o técnicas empleados para problemas particulares, recursos empleados) (max. 2.5 páginas. Puede incluir diagramas como anexos fuera del límite de las 10 páginas) (30%)

Para esta solución se ha definido la siguiente estructura de TDAs:

Sistema (system): Incluye las funciones propias al sistema de archivos que soportan las demás operaciones.

Unidad (unit): Realiza las operaciones donde se implican discos o unidades.

Usuario (user): Contiene las funciones de tratamiento de usuarios como registro, login, logout.

Archivo (file): Contiene la creación del archivo y las propiedades y operaciones que se realizan sobre estos.

Carpeta (folder): Contiene la creación de carpetas y las propiedades y operaciones que se realizan sobre estos.

## Aspectos de implementación

El proyecto está diseñado lógicamente en 6 módulos que contienen las funciones necesarias para cumplir con el objetivo del proyecto. De este modo se ordenan por TDA cada uno de sus módulos y se hace más fácil de mantener.

“Físicamente” está estructurado en 7 archivos de código fuente Scheme con extensión .rkt:

1. main\_157809385\_HenriquezGonzalez.rkt
2. file\_157809385\_HenriquezGonzalez.rkt
3. folder\_157809385\_HenriquezGonzalez.rkt
4. system\_157809385\_HenriquezGonzalez.rkt
5. unit\_157809385\_HenriquezGonzalez.rkt
6. user\_157809385\_HenriquezGonzalez.rkt
7. utils\_157809385\_HenriquezGonzalez.rkt

Cada uno de estos archivos representa un TDA que se ha considerado para resolver el problema planteado, excepto el archivo *main\_157809385\_HenriquezGonzalez.rkt* que contiene los casos de prueba y ejecución del sistema.

Para el desarrollo de este proyecto se ha utilizado el IDE DrRacket, version 8.7, english by PLT.

Sitio oficial y descarga Dr. Racket (https://racket-lang.org/)

## Instrucciones de uso

Las pruebas se deben realizar ejecutando el archivo:

*main\_157809385\_HenriquezGonzalez.rkt*

Ejecutándose en el IDE Dr. Racket.

Desde la consola de Dr. Racket (ver imagen 1)debe ingresar el nombre de la función a ejecutar y los parámetros que sean necesarios.

Ejemplo de ejecución:

|  |
| --- |
|  |
| Imagen 1: Consola de ejecución Dr. Racket. |

## Resultados y autoevaluación:

Resumen de resultados obtenidos. Esto puede abordarse listando todos los requerimientos del proyecto en una tabla indicando el grado de alcance. Indicar que tipos de pruebas se hicieron. Cuantas de las pruebas fueron exitosas, cuantas fracasaron, razones de fallos. Especificar que funciones no se completaron y por qué no se completaron. (máximo 1 página) (10%)

## Conclusiones del trabajo

## El dominio de los paradigmas de programación y sus conceptos es fundamental para explorar las distintas perspectivas y enfoques en el desarrollo de software. En particular, el paradigma funcional desempeña un papel clave en esta tarea. No es imprescindible conocer múltiples lenguajes de programación; en cambio, resulta más útil comprender los paradigmas de programación subyacentes que los envuelven. De esta manera, es factible abordar una amplia gama de lenguajes de programación sin dificultades.

## El cálculo lambda, por su parte, ha adquirido una gran relevancia en el desarrollo de este laboratorio. Esta herramienta permite la creación de funciones anónimas y el uso reiterativo de la currificación. En conjunto con la notación matemática y las funciones de primer orden proporcionadas por el paradigma funcional, se puede hablar de un enfoque potente y seguro que ha ido ganando importancia en los últimos años donde la implementación de soluciones se ha enfocado en el paradigma funcional, como servicios en la nube serverless de amazon o GCP, etc. Donde solo viven funciones que se ejecutan al ser llamadas y se pagan al no ser utilizadas.

## En cuanto a los resultados obtenidos, no se logró cumplir con todos los objetivos propuestos. Sin embargo, se presentó una representación del problema algo compleja que podría haber sido replanteada de forma más sencilla y vincularla más a una estructura funcional. En lugar de ello, se consiguió una estructura diferente, lo cual añadió cierta complejidad al trabajo. En futuras versiones, sería recomendable revisar y ajustar la formulación del problema para facilitar una implementación más eficiente y acorde con los principios del paradigma funcional.

## Referencias

* Dr. Racket (Ed.). (n.d.). *Racket Documentation*. Racket documentation. Retrieved April 23, 2023, from https://docs.racket-lang.org/index.html
* Kent Dybvig, R. (2009). The scheme programming language (4th ed.). MIT Press. Retrieved April 23, 2023, from https://www.scheme.com/tspl4/