9 DE NOVIEMBRE DE 2021

PROYECTO DE LABORATORIO PARADIGMAS DE PROGRAMACIÓN

PLATAFORMA QUE EMULA GOOGLE DOCS

CHRISTOPHER TORRES
21.024.191-4
Ingeniería civil en Informática

Índice

Introducción

El presente informe de Paradigmas de programación recopila toda la información complementaria al proyecto, presentando así la descripción del problema a resolver, su respectivo análisis, un diseño de solución junto al enfoque abordado, aspectos de implementación, la estructura del proyecto y finalmente, una conclusión con todos los resultados y ejemplos obtenidos.

La principal característica de este problema es la emulación de un programa similar a Google Docs abordado desde un paradigma funcional, esto quiere decir, que se trabajará con un lenguaje de programación que permita realizar sus tareas con funciones.

Descripción del problema

El problema que se ha presentado para el primer laboratorio de la asignatura fue la creación de una plataforma similar a Google Docs, que permitiera hacer varias de las funciones básicas que este permite a sus usuarios. Para ello, se debe crear una plataforma que funcione como una especie de "servidor" que almacena toda la información necesaria para funcionar, ya sean: información de la cuenta de cada usuario registrado (nombre, contraseña, fecha de creación), información de los documentos almacenados (autor, fecha de creación, modificaciones, historial de versiones, accesos, el contenido que tienen), fecha de creación y nombre de la plataforma.

Esta nueva plataforma, deberá permitir crear documentos colaborativos, esto quiere decir, que todos los usuarios que estén registrados en la plataforma pueden acceder a otros documentos mediante diferentes permisos otorgados por el dueño del documento si este lo desea. Esos permisos son: escritura, lectura y comentario. Estas modificaciones permiten que el documento tenga un historial de versiones los cuales se irán actualizando a medida que se generen cambios.

Descripción del paradigma funcional – Lenguaje utilizado

Mencionado anteriormente, el paradigma utilizado en este proyecto es el funcional, el cual permite a un programador tratar a las funciones como el principal recurso. En otras palabras, todos los elementos pueden entenderse como funciones y el código puede ejecutarse a través del llamado secuencial de estas funciones.

La versatilidad de la programación funcional radica en parte del uso de funciones de orden superior, ya que existe la posibilidad de que las funciones puedan tomar otras funciones como parámetro y devolver funciones como resultado.

Para poder aplicar este paradigma de manera práctica y comenzar a entender la relevancia que tiene sobre este proyecto, se decide la utilización del lenguaje **Racket** como lenguaje de programación principal.

Si bien, Racket es un lenguaje multiparadigma y de propósito general, este está basado en gran parte en lenguajes como **LISP** y **Scheme**, siendo este último un gran representante de la programación funcional.

Algunos conceptos de este paradigma aplicados al desarrollo de este programa son:

- El no uso de variables. Tampoco se emuló el trabajo de variables dentro de Racket (funciones como let, let*, set!, set!-values, etc).
- Uso de funciones como único recurso de implementación.
- Aplicación de recursión (debido a la no existencia de bucles) **(ver Anexo 1)**.
- Uso de listas como estructura de dato principal **(ver Anexo 2)**.
- Uso de expresiones Lambda **(ver Anexo 3)**.
- Uso de funciones de orden superior o composición de funciones **(ver Anexo 4)**.

Análisis del problema – Requisitos específicos

Algo muy relevante en este problema, es la identificación de cada parte requerida para el programa, ya que esto permitirá de manera más fácil abordar con diversas soluciones todo lo que se desee construir.

En primera instancia, lo primero que se debe definir, es la forma en que se debe almacenar toda la información. Dentro de la plataforma que se desea crear, se deben definir y respetar los siguientes puntos que se presentarán a continuación en forma esquematizada, entendiendo que el punto de partida es la creación de la plataforma.

Plataforma:

1. Nombre de la plataforma
2. Fecha de creación de la plataforma
3. Usuarios registrados
   3.1. Nombre de usuario
   3.2. Contraseña de usuario
   3.3. Fecha de creación de la cuenta
4. Usuario(s) activo(s)
5. Documentos
   5.1. Autor del documento
   5.2. Fecha de creación del documento
   5.3. Fecha de última modificación del documento
   5.4. Contenido
   5.5. Id del documento
   5.6. Accesos concedidos
      5.6.1. Permiso de lectura
      5.6.2. Permiso de escritura
      5.6.3. Permiso de comentario
   5.7. Historial de versiones
   5.8. Comentarios

De esta forma, queda de una manera abstracta, la forma en la que se debe trabajar dicho programa. Ahora bien, se definen algunos puntos de manera descriptiva para su mejor entendimiento, además de mencionar algunas consideraciones.

- Plataforma: Es el espacio virtual donde se almacena información. En este caso, almacena todo lo relevante para generar un espacio de trabajo colaborativo.
- Usuarios registrados: En este espacio, se almacenan todos los usuarios con un formato declarado: Usuario, contraseña y fecha de creación de la cuenta. Los usuarios registrados en la plataforma tendrán la posibilidad de interactuar con los diversos documentos a medida que los permisos correspondientes lo ameriten. Además, con este apartado, existe la posibilidad de realizar diversas acciones dentro de la plataforma, tales como crear documentos, modificar textos, leer documentos, compartir documentos, asignar permisos a otros usuarios ya registrados, etc. De una manera consistente, ya que de esta forma, no cualquier persona podrá hacerlo.
  También, se debe verificar que no existan usuarios repetidos para evitar conflictos de autenticidad.
- Usuario activo: Es el usuario que tiene una sesión activa en el documento, que le permite realizar las acciones pertinentes.
- Documentos: Es un contenedor de texto que tiene información relevante asociada. Este posee un nombre, una fecha de creación, un autor, un contenido, un historial y tiene almacenado los usuarios los cuales poseen diversos tipos de acceso al documento.
  El texto que contiene puede ser leído, comentado o modificado dependiendo de lo que deseen hacer sus usuarios con sus debidos accesos.
  El documento, posee una versión activa y un historial de versiones; la sesión activa del documento viene a ser la versión más actual de este o simplemente la que es visible de cara al usuario, mientras que el historial, guarda todas las versiones anteriores que hayan sido modificadas con diversas operaciones como: añadir texto al final, reemplazar palabras, eliminar palabras, etc. En este historial, cada contenido alterado de cualquier manera va acompañado de su respectiva fecha de modificación.
- Accesos: Existen tres tipos de acceso los cuales pueden recibir los usuarios registrados.
  - Acceso de lectura: Este permiso otorga la capacidad de leer el contenido de un documento o buscar texto específico sin la posibilidad de hacer modificaciones o cambios a este.
  - Acceso de escritura: Este permiso otorga la capacidad de modificar y gestionar el contenido de la versión activa del documento y también le permite indagar dentro de las versiones más antiguas de este mismo.
  - Acceso de comentario: Este permiso otorga la posibilidad de comentar parte del contenido de un documento.
- Id: Existe un identificador para cada documento, esto es esencial para su correcta búsqueda y organización de la plataforma.

Diseño de la solución

Una vez que se tiene una clara visión de los conceptos anteriormente definidos, se presentan los aspectos más relevantes para llegar a una solución concreta.

Como la estructura principal a utilizar dentro de este paradigma es la lista, se presenta un diagrama de cómo se almacenará la información **(ver Anexo 5)**.

Una vez se tienen los pilares de construcción de la solución, se comienza a descomponer desde lo macro a lo micro. Se dividirá en secciones la forma de enfocar el problema.

1. Consideración de lista principal (plataforma)

Se le atribuirá a la lista principal el nombre de plataforma para su mejor entendimiento. Además de mencionar sus índices desde 0 a N.

La posición 0 y 1, son fáciles de abordar, ya que el nombre solo será una String almacenada sin ninguna restricción, mientras que la fecha deberá tener ciertas restricciones para ser una fecha coherente. Esto conlleva la verificación de años, meses y días por cada mes. Para ello, se crea un TDA Fecha para incluir todos esos aspectos que se necesitan considerar. Posteriormente se necesitará una forma de transformar un formato de fecha (dd/mm/aa) a uno legible, esto para que se muestre una especie de interfaz del documento.

2. Usuarios registrados

La primera sub-lista que posee la plataforma, es la segunda posición. En esta, se almacenan todos los usuarios registrados. Algo para tener en cuenta, es que cada usuario es único e irrepetible, por lo que se debe verificar este punto (se puede verificar solo el nombre de usuario, ya que si se llegan a repetir contraseñas, no afecta a los usuarios).

3. Usuario activo

Debido al requerimiento del proyecto y ciertas limitaciones que se tienen, se decide que solo un usuario puede estar con la sesión activa y ejecutar acciones dentro de los documentos, por ende, en esta lista, solo se almacenará el nombre del usuario que interacciona con la plataforma (verificando esté registrado) en una lista que está en la tercera posición.

4. Lista de documentos

La cuarta posición es una sub-lista que guarda la información de los documentos desde su creación y almacenas versiones. Como se observa en el diagrama del **anexo 5**, la información en las primeras posiciones es simple de agregar, mientras que en las últimas dos, existen dos sub-listas más. La primera almacena los accesos compartidos con los usuarios, mientras que la lista de historial almacena las versiones de los documentos. En esta última, se debe generar una modificación en los documentos, ya solo se deben guardar el contenido anterior a la modificación y la fecha de esta, debido que todo lo otro sigue estando en la lista que almacena el documento.

5. Funciones de encriptado y desencriptado

En las últimas dos posiciones de la plataforma, se almacenan las funciones que sirven para encriptar y desencriptar el contenido de los documentos, esto se hace para tener seguridad dentro de los documentos y no se puedan leer fácilmente sin la llave de desencriptación.

Aspectos de implementación

El proyecto se desarrolló en el entorno que ofrece el mismo lenguaje, en este caso, DrRacket en su versión 8.2 y no se utilizó ningún IDE externo para la programación del código. La razón es que todo lo implementado en DrRacket (debug, terminal, check syntax, etc) permite llevar a cabo la elaboración de todo el programa y no se necesita emplear bibliotecas externas ni librerías que no son nativas del lenguaje.

Algo muy importante en la estructuración del proyecto es la división de archivos con la implementación de TDA's. Estos TDA's sirven para encapsular y modelar de mejor manera el código principal, ya que permiten abstraer gran parte de los problemas a abordar. En este caso en particular, se crearon cinco TDA's: TDA Usuario, TDA Acceso, TDA Documento, TDA ParadigmaDocs y TDA Fecha. Cada uno posee constructores, selectores, modificadores y funciones de pertinencia que hacen un gran trabajo para que el modelo sea mucho más concreto y estructurado, ya que cada TDA va en archivos aparte y genera un orden dentro de todo lo necesario para el programa.

Estos archivos se comunican entre si mediante la importación y exportación de archivos **(ver Anexo 6).**

Instrucciones de uso

Para la utilización de los ejemplos anidados al final del código principal, solo basta correr el programa y escribir el nombre de la plataforma almacenada en el ejemplo.

Los ejemplos se dividen en 3 para cada función y tienen una enumeración específica. Por ejemplo, Gdocs011, los primeros dos dígitos hacen referencia a que se está llamando a la primera función (número 01) y el tercer dígito hace mención del número del ejemplo, ya que existen 3 **(ver anexo 7)**.

Para todos los ejemplos se espera que no haya ningún error debido a que todos se testearon de la siguiente manera en la consola de DrRacket:

> Gdocs011

'("Gdocs" (25 10 2021) (("user2" "pass2" (22 4 2021)) ("user1" "pass1" (25 4 2021)) ("user3" "pass3" (25 4 2021))) () () #<procedure:encryptFn> #<procedure:decryptFn>)

Casos puntuales:

- Gdocs081 no funciona correctamente debido a que la función **Share** está incompleta, pero aun así no manda errores.

- Gdocs091, Gdocs092 y Gdocs093 deben ser escritos con la función (display) para que su contenido sea legible. Caso contrario, se mostrará una string sin formato.
  > (display Gdocs091)

Resultados y autoevaluación

Para la autoevaluación, se tomarán en cuenta las funciones con sus requerimientos y se evaluarán de acuerdo con la siguiente escala proveída por el documento oficial **(ver anexo 8)**.

Por otro lado, se comentará brevemente aspectos de la función, así mencionando errores, aciertos, complicaciones, etc. En la tabla anexada **(ver Anexo 9)**, está la parte de prerrequisitos para evaluación, los requisitos de implementación, las funciones mínimas requeridas, ejemplos y la evaluación correspondiente.

Requerimientos Funcionales
1. TDA's
Comentarios: Cada TDA está en un archivo aparte y cumple con todo lo necesario para ser un TDA (selectores, constructores, pertinencia, modificadores, funciones extra, etc).
2. Register
Comentarios: En un principio no se lograba realizar con recursión natural, pero casi al final del proyecto, se logró completar e implementar la recursión natural. Se falló un total de 42 veces y era por problemas en la recursión, una vez resuelto el problema no hubo más problemas.
3. Login
Comentarios: Los ejemplos de Login están hechos en base a la función Create y Share, además de demostrar que se devuelve un procedimiento en todos los casos (función de orden superior). Las veces que fallaba Login era por la currificación y el uso de operaciones, esto mandó aproximadamente 100 errores en el transcurso de la creación.
4. Create
Comentarios: Create no tiene ningún problema y cumple con lo pedido respecto a seleccionar las funciones de encriptación a través de la currificación (lambda). Esta función tuve solo 5 fallos y su creación fue bastante rápida.
5. Share
Comentarios: Share funciona perfectamente, aunque durante su desarrollo, existieron más de 150 errores por problemas de escritura y de funcionamiento. Generalmente fallaba porque no se lograba encontrar la forma de hacer un list-set y por verificar muchas cosas.
6. Add
Comentarios: Add fue muy complicada de crear por tener que crear una versión antigua y agregarla al historial. Tuvo aproximadamente 300 errores y gran mayoría venía de la actualización del documento actual. Finalmente, se masterizó el cambio y se logró el correcto funcionamiento de la función.
7. RestoreVersion
Comentarios: Restore versión fue muy similar a Add, por lo que no tomó más de 30 errores su realización. Generalmente daba errores por errores de escritura.

8. RevokeAllAccesses
Comentarios: Esta función costó por el aprendizaje de la función filter. Una vez se entendió su funcionamiento, revokeAllAccesses no tomó más de 10 intentos para que funcionara al 100%.
9. Search
Comentarios: Función no realizada por problemas en su funcionamiento, ya que no se logró hacer la búsqueda de textos tanto en las versiones activas como historial. Si se hubiese encontrado la forma de hacer esto, se pudo haber completado.
10. Paradigmadocs->String
Comentarios: Paradigmadocs->String solo tomó 50 intentos para lograr generar el string. Esto se logró a través de funciones del tipo String-append, String->list, List->String, etc. Hay que considerar que esta función necesita ser corrida con (display GdocsXXX).
Requerimientos no funcionales
11. Delete
Comentarios: Delete no tomó más de 20 intentos ya que era muy similar a Add pero a la inversa, por ende no hubo casi problemas.
12. SearchAndReplace
Comentarios: Nuevamente era muy similar a Delete, no tomó más de 20 intentos y fue muy sencilla de implementar gracias a la función String-Replace.
13. ApplyStyles
Comentarios: No realizada ya que era opcional y no se encontraba una forma sencilla de implementarla.
14. Comment
Comentarios: No se realizó porque para lograrlo, se debía hacer un cambio muy grande en la implementación de Documentos, y eso iba a generar un costo de tiempo y errores muy grandes.
15. Función de encriptación y desencriptación
Comentarios: Fallaba inicialmente por errores de cálculo al poner símbolos y letras extrañas, pero nada relevante. No costó más de 50 intentos cada una.
16. CtrlZ y CtrlY
Comentarios: No se realizó porque para lograrlo, se debía hacer un cambio muy grande ya que se necesitaba generar memoria para esto, además de que el tiempo invertido en esta función era exagerado.

Conclusiones

Concluyendo este proyecto, se logra presentar un programa que cumple de una manera bastante satisfactoria la problemática propuesta. Tiene sus limitaciones en 4 funciones que no se lograron completar (una funcional y tres opcionales) pero sin duda alguna, se puede utilizar perfectamente y emula bastante bien la plataforma de Google Docs.

El paradigma funcional en un principio era muy difícil de digerir, pero con el tiempo y los avances, se hizo todo más sencillo y aportó mucho al cambio de enfoque al que se estaba acostumbrado.

Bibliografía

Programación funcional: Ideal para algoritmos. (2021, 26 noviembre). IONOS Digitalguide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/programacion-funcional/

García, E. (s. f.). *¿Qué es la Programación Funcional?* CódigoFacilito. https://codigofacilito.com/articulos/programacion-funcional

Anexos

1. Recursión (de cola)

```
;descripción: Función que saca la fecha de creación de un usuario registrado
;dom: Lista Registro X String
;rec: Fecha
;recursión: de cola (elegida por su fácil implementación y no dejar estados pendientes)
(define (fechaDeCreacionUser? listaRegistrados usuario)
  (if (empty? listaRegistrados)
      null
      (if (eq? (car (car listaRegistrados)) usuario)
          (third (car listaRegistrados))
          (fechaDeCreacionUser? (cdr listaRegistrados) usuario)
          )
      )
  )
```

Fuente: Elaboración propia.

En esta función, se hace el uso de la recursión de manera tal que se recorre una lista y se va verificando cierta condición. Funciona mediante *car* y *cdr*, utilizando la cabecera y la cola de la lista hasta que esta quede vacía o hasta que la función retorne algo específico.

2. Estructura basada en listas

```
;---CONSTRUCTORES---
;descripción: Función que crea la plataforma con paradigmadocs
;dom: String X Date X EncryptFunction X DecryptFunction
;rec: Lista
(define (paradigmadocs nombrePlataforma fecha encryptFn decryptFn)
  (if (and (string? nombrePlataforma)
           (fecha? fecha))
      (list nombrePlataforma fecha (list) (list) (list) encryptFn decryptFn)
      null)
  )
```

Fuente: Elaboración propia.

En este constructor de ParadigmaDocs, se utiliza una lista para almacenar toda la información contenida en la plataforma, además de contener sub-listas que más adelante almacenan incluso más listas.

3. Uso de expresiones Lambda

```
(define (search paradigmadocs)
  (lambda (searchText)
    (if (null? (getLista2 paradigmadocs))
        null
        (if (null? (buscarDocsPropietario (getLista3 paradigmadocs) (car (getLista2 paradigmadocs))))
            null
            #f
            )
        )

    )
  )
```

Fuente: Elaboración propia.

En esta función se currifica utilizando la expresión Lambda, que permite pasar parámetros extra a las funciones. A lo largo del código principal, la mayoría de las funciones utilizan esta expresión.

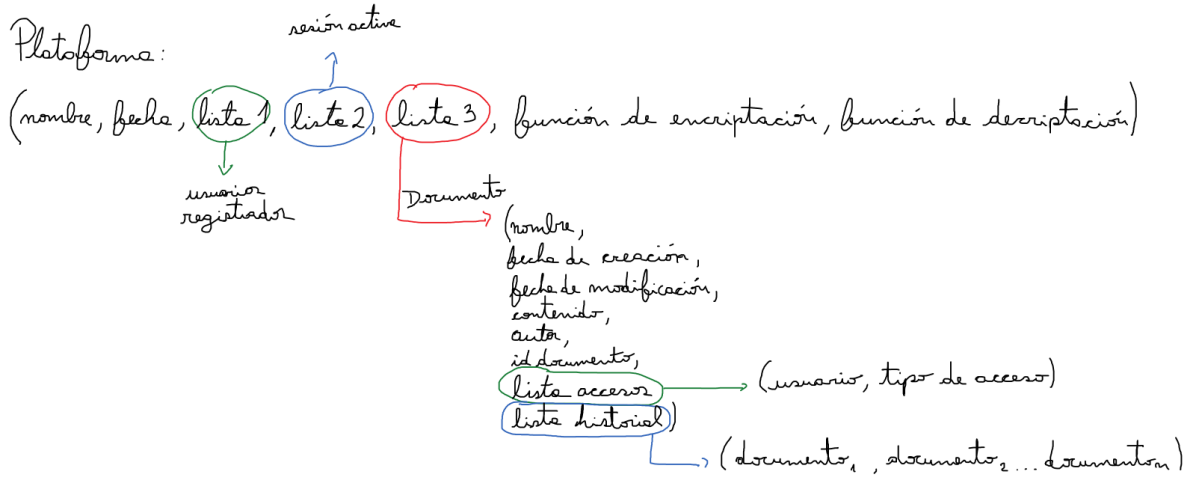
4. Funciones de orden superior

```
;-LOGIN-
;descripción: Función que autentica a un usuario registrado y le permite ejecutar un comando específico
;dom: paradigmadocs X string X string X function
;rec: Paradigmadocs
;recursión: de cola
(define (login paradigmadocs username password operation)
```

Fuente: Elaboración propia.

La función Login, recibe como parámetro “operation”, lo cual es un procedimiento aparte (otra función). Además, Login retorna un procedimiento dependiendo de la operación escogida.

5. Diagrama de solución



Fuente: Elaboración propia

6. Requires (importación) Provide (Exportación)

```
(require "TDAParadigmaDocs_21024191_TorresAceituno.rkt")
(require "TDAUser_21024191_TorresAceituno.rkt")
(require "TDADate_21024191_TorresAceituno.rkt")
(require "TDADocumento_21024191_TorresAceituno.rkt")
(require "TDAAcceso_21024191_TorresAceituno.rkt")
```

```
;To import
(provide (all-defined-out))
```

Fuente: Elaboración propia.

Funciones que tiene Racket para la intercomunicación entre archivos. Para poder utilizar las funciones de un archivo, se debe poner el **Provide** al final del archivo y para importar desde otros archivos, se debe utilizar el **Require** más el nombre del archivo. Tener en consideración que los archivos deben estar bajo el mismo directorio/carpeta.

7. Ejemplos Gdocs

```
; 1) REGISTER
(define Gdocs011 (register (register (register Gdocs000
(define Gdocs012 (register Gdocs000 (fecha 15 4 2021) "us
(define Gdocs013 (register (register Gdocs000 (fecha 25 3

; 2) LOGIN
(define Gdocs021 (login Gdocs011 "user3" "pass3" share))
(define Gdocs022 ((login Gdocs011 "user1" "pass1" create)
(define Gdocs023 ((login Gdocs022 "user2" "pass2" create)

; 3) CREATE
(define Gdocs031 ((login Gdocs011 "user3" "pass3" create)
(define Gdocs032 ((login Gdocs012 "user" "pass" create)
(define Gdocs033 ((login Gdocs032 "user" "pass" create)
```

Fuente: Elaboración propia.

Se definen **Gdocs** con la numeración correspondiente de cada función y cada ejemplo respectivo.

8. Escala autoevaluación

a. 0: No realizado.
b. 0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
c. 0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
d. 0.75: Implementación con problemas menores (funciona 75% de las veces)
e. 1: Implementación completa sin problemas (funciona 100% de las veces)

Fuente: Documento oficial de laboratorio.

9. Tabla autoevaluación de todas las funciones

1. TDA's
Prerrequisitos para evaluación: Ninguno.
Requisitos de implementación: Se cumple todo lo pedido.
Funciones mínimas requeridas: Se cumple con las funciones.
Evaluación: 1.
2. Register
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplido.
Ejemplos: Gdocs011, Gdocs012, Gdocs013.
Evaluación: 1.
3. Login
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Ejemplos: Gdocs021, Gdocs022, Gdocs023.
Evaluación: 1.
4. Create
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Ejemplos: Gdocs031, Gdocs032, Gdocs033.
Evaluación: 1.
5. Share
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Ejemplos: Godcs041, Gdocs042, Gdocs043.
Evaluación: 1.
6. Add
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Ejemplos: Gdocs051, Gdocs052, Gdocs053.
Evaluación: 1.
7. RestoreVersion
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Ejemplos: Gdocs061, Gdocs062, Gdocs063.

Evaluación: 1.
8. RevokeAllAccesses
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Ejemplos: Gdocs071, Gdocs072, Gdocs073.
Evaluación:
9. Search
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Se cumple el primer párrafo, el segundo parcialmente y el tercero se cumple parcialmente.
Ejemplos: Gdocs081.
Evaluación: 0.
10. Paradigmadocs->String
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Ejemplos: Gdocs091, Gdocs092, Gdocs093.
Evaluación: 1.
Requerimientos Opcionales
11. Delete
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Ejemplos: Gdocs101, Gdocs102, Gdocs103.
Evaluación: 1.
12. SearchAndReplace
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos Todos.
Ejemplos: Gdocs111, Gdocs112, Gdocs113.
Evaluación: 1.
13. ApplyStyles
Evaluación: 0.
14. Comment
Evaluación: 0.
15. Función de encriptación y desencriptación
Prerrequisitos para evaluación: Cumplido.
Requisitos de implementación: Cumplidos todos.
Evaluación: 1.
16. CtrlZ y CtrlY
Evaluación: 0.