Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática
Paradigmas de Programación

Paradigmas de Programación

Proyecto semestral de laboratorio

Versión 1.1 - actualizada al 21/09/2024

Laboratorio 1 (Paradigma Funcional - Lenguaje Scheme)

Versión 1.1 22/09/24: Se ajustan puntajes de cada RF

Versión 1.0 17/09/24: Versión final v1

Versión 0.9 7/09/24 : Versión preliminar

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

Enunciado General: Procure consultar los aspectos generales del proyecto de laboratorio en el documento general.

Fecha de Entrega: Ver calendario clase a clase donde se señala el hito

Objetivo del laboratorio: Aplicar conceptos del paradigma de programación funcional usando el lenguaje de programación Scheme en la resolución de un problema acotado.

Resultado esperado: Juego Conecta4

Profesor responsable: Roberto González (al hacer consultas en este documento, procurar hacer la mención a **@Roberto Gonzalez Ibanez** (roberto.gonzalez.i@usach.cl) y **@ Gonzalo Martinez** (gonzalo.martinez@usach.cl) para que las notificaciones de sus consultas lleguen al profesor correspondiente).

Recomendaciones: El laboratorio está diseñado como un conjunto de ejercicios a abordar bajo cada paradigma. En este sentido, el desarrollo del laboratorio constituye un espacio para practicar y prepararse además para la evaluación de cátedra del correspondiente paradigma. Por tanto, se recomienda incorporar en sus hábitos de estudio/trabajo el desarrollo de las funcionalidades de forma diaria. En total el laboratorio 1 lista N funcionalidades a cubrir. Para alcanzar la nota de aprobación, se deben cubrir las M primeras y para la nota máxima se pueden cubrir Q más. Procure destinar tiempo para analizar y hacer una propuesta de diseño para el laboratorio completo antes de proceder a la implementación de la solución. No es necesario que sus funciones implementen comprobación de tipo, esto es opcional.

Requerimientos No Funcionales (RNF)

Algunos son ineludibles/obligatorios, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

1. **RNF1. (obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los **requerimientos funcionales solicitados**. El objetivo de esta autoevaluación es que usted pueda anticiparse a una nota tentativa al momento de la entrega del laboratorio para posteriormente, en caso de que su calificación sea inferior a 4.0, pueda proceder oportunamente a realizar mejoras en su laboratorio a través del comodín correspondiente.
    a. El formato de la autoevaluación es:
        i. archivo **AUTOEVALUACION_RUT_NOMBRE_APELLIDOS.txt**
        ii. **Evaluar cada Requerimiento Funcional (RF) y no funcional (RNF)) en la escala de 0, 0,25, 05, 0,75, 1. Formato a seguir (puntajes de ejemplo):**
            1. **RF1 - Nombre RF1: 0,25**
            2. **RF2 - Nombre RF2: 0,5**
            3. **RFN - Nombre RFN: 1**
        iii. **Sólo debe agregar puntaje (0 a 1), usted no debe calcular nota ni agregar justificación.**
2. **RNF2. (obligatorio) Lenguaje:** La implementación debe ser en el lenguaje de programación Scheme/Racket en base a una programación principalmente declarativa - funcional.
3. **RNF3. (obligatorio) Versión:** Usar DrRacket versión 6.11 o superior
4. **RNF4. (obligatorio) Standard:** Se deben utilizar funciones estándar del lenguaje. No emplear bibliotecas externas.
5. **RNF5. (obligatorio) No variables:** No hacer uso de función *define* **en combinación** con otras como set! (o similares) para emular el trabajo con variables.
6. **RNF6. (1 pts) Documentación:** Todas las funciones deben estar debidamente comentadas. Indicando descripción de la función, tipo de algoritmo/estrategia empleado (ej: fuerza bruta, backtracking, si aplica) argumentos de entrada (dominio) y retorno (recorrido). En caso de que la función sea recursiva, indicar el tipo de recursión utilizada y el porqué de esta decisión.
    a. Cada función debe ser comentada en el mismo código de la siguiente forma:

```
; Descripción: insertar descripción de la función.

; Dom: valor1 (tipo dato) X arg2 (tipo dato) ...
```

```
; Rec: valor1 (tipo dato) X ...

; Tipo recursión: No aplica/Natural/Cola

(define mi-funcion ....
```

7. **RNF7. (obligatorio) Dom->Rec:** Respetar la definición de función en términos de conjunto de salida (dominio - tipo de entrada de la función) y llegada (recorrido - tipo de retorno de la función) sin efectos colaterales, además del nombre de las mismas (respetar mayúsculas y minúsculas).

8. **RNF8. (1 pts) Organización:** Estructurar su código en archivos independientes. Un archivo para cada TDA implementado y uno para el programa principal donde se dispongan sólo las funciones requeridas en el apartado de requerimientos funcionales. Debe usar la función require/provide. Respecto de este último punto, puede optar por dejar cada función señalada en este enunciado en los respectivos archivos de cada TDA, procurando en el main hacer el require/provide a todos los archivos que permitan usar dichas funciones directamente desde el main.

9. **RNF9. (2.5 pts) Historial:** Historial de trabajo en Github tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 2 semanas (no espere a terminar la materia para empezar a trabajar en el laboratorio. Puede hacer pequeños incrementos conforme avance el curso)**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.375 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

10. **RNF10. (obligatorio) Script de pruebas (pruebas_RUT_Apellidos.rkt) -Leer instrucciones al final de este enunciado-:** Incluir como parte de su entregable debe entregar una cantidad definida al final de este enunciado de archivos independiente al código donde muestre de forma completa, con la documentación correspondiente, el funcionamiento de su programa. La siguiente instrucción se aplica a cada archivo indicado al final de este enunciado: este archivo será similar al script de prueba proporcionado al final de este documento. Este archivo debe incluir los ejemplos provistos en el script de prueba de este enunciado además de **3 ejemplos** por cada una de las funciones requeridas. **Solo se revisarán proyectos que incluyan este archivo.**
PARA MAYOR CLARIDAD VER INSTRUCCIONES AL FINAL DE ESTE ENUNCIADO.

11. **RNF11. (obligatorio) Prerrequisitos:** Para cada función se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la función implementada. Ej: Para evaluar la función login, debe estar implementada la función register.

Requerimientos Funcionales (RF)

La nota correspondiente al apartado de RF comienza en 1.0 y por cada RF correcto dicho puntaje escrito en el enunciado se suma a la nota base.

Para que el requerimiento sea evaluado, DEBE cumplir con el prerrequisito de evaluación y requisito de implementación. En caso contrario la función no será evaluada. El total de requerimientos permiten alcanzar una nota mayor que 7.0, por lo que procura realizar las funciones que consideres necesarias para alcanzar un 7.0. Si realizas todas las funciones y obtienes el puntaje máximo, la nota asignada será igualmente un 7.0. El puntaje de desborde se descarta.

1. **RF01. (0.5 pts TDAs)**. Especificar e implementar abstracciones apropiadas para el problema. Recomendamos leer el enunciado completo (el general y el presentado en este documento) con el fin de que analice el problema y determine el o los TDAs y representaciones apropiadas para la implementación de cada uno. Luego, planifique bien su enfoque de solución de manera que los TDAs y representaciones escogidos sean aplicables y pertinentes para abordar el problema bajo el paradigma funcional.

   Para la implementación debe regirse por la estructura de especificación e implementación de TDA vista en clases: Representación, Constructores, Funciones de Pertenencia, Selectores, Modificadores y Otras Funciones. Procurar hacer un uso adecuado de esta estructura a fin de no afectar la eficiencia de sus funciones. En el resto de las funciones se debe hacer un uso adecuado de la implementación del TDA (ej: usar selectores, modificadores, constructores, según sea el caso. No basta con implementar un TDA y luego NO hacer uso del mismo). **Solo implementar las funciones estrictamente necesarias dentro de esta estructura**.

   A modo de ejemplo, si usa una representación basada en listas para implementar un TDA, procure especificar e implementar funciones específicas para selectores **(ej: en lugar de usar car, cdr y otras funciones propias del TDA lista, realice implementaciones o establezca sinónimos con nombres que resulten apropiados para el TDA. Por ejemplo (define miselector car) o (define miselector (lambda (param) …… car …….)).**

   Dejar claramente documentado con comentarios en el código aquello que corresponde a la estructura base del TDA. Estructura bases mínimas que deberá considerar para el resto de las funciones corresponden “board”, “player”, “piece”, "game" y todas las funciones que se indican como constructor (como mínimo), las cuales constituyen elementos centrales sobre el que se aplicaran las distintas funciones implementadas.

   Debe contar además con representaciones complementarias para otros elementos que considere relevantes para abordar el problema.

   Especificar representación de manera clara para cada TDA implementado (en el informe y en el código a través de comentarios). Luego implementar constructores y según se requiera, implemente funciones de pertenencia, selectores, modificadores

y otras funciones que pueda requerir para las otras funciones listadas a continuación.

Las funciones especificadas e implementadas en este apartado son complementarias (de apoyo) a las funciones específicas de los dos TDAs que se señalan a continuación. Su desarrollo puede involucrar otros TDAs y tantas funciones como sean necesarias para abordar los requerimientos.

Procurar que todas las palabras claves creadas se registren en minúsculas o mayúsculas de manera consistente con el objetivo de lograr el match exacto en las interacciones con el usuario. Para tales efectos puede usar funciones como string-downcase, string-upcase o string-ci=? (para comparar case insensitive).

Como convención, los nombres de funciones serán en minúsculas y en caso de ser nombres compuestos, cada componente se debe separar por guiones (ej: string-downcase).

Para cada uno de los ejemplos expresados para las funciones indicadas a continuación, se realizan definiciones con el fin de simplificar las expresiones. Recordar que estas definiciones no son variables, sino que por el contrario se pueden entender como funciones constantes.

2. **RF02. (0.1 pts) TDA Player - constructor**. Función que permite crear un jugador.

Nombre función	**player**
Prerrequisitos para evaluación (req. funcionales)	1
Requisitos de implementación	Usar estructuras basadas en listas y/o pares
Dominio	id (int) X name (string) X color (string) X wins (int) X losses (int) X draws (int) X remaining-pieces (int) donde: ● wins: número de veces que ha ganado ● losses: número de veces que ha ganado ● draws: número de veces que ha estado en empate ● remaining-pieces: cantidad de fichas actuales ○ La regla del juego es que se comienza con 21 fichas, pero ustedes pueden crear juegos más cortos con una cantidad de fichas mayor a 4 y menor a 21. ○ La cantidad de fichas por jugador no afecta el desarrollo del laboratorio ni las jugadas a realizar.
Recorrido	*player*

Ejemplo de uso	`(define p1 (player 1 "Juan" "red" 0 0 0 21))`
	
	`(define p2 (player 2 "Juan" "yellow" 0 0 0 21))`

3. **RF03. (0.1 pts) TDA Piece - constructor**. Función que crea una ficha de Conecta4.

Nombre función	**piece**
Prerrequisitos para evaluación (req. funcionales)	2
Requisitos de implementación	Usar estructuras basadas en listas y/o pares
Dominio	color (string)
Recorrido	*piece*

Ejemplo de uso	`(define red-piece (piece "red"))`
	
	`(define yellow-piece (piece "red"))`

4. **RF04. (0.2 pts) TDA Board - constructor**. Crear un tablero de Conecta4.

Nombre función	**board**
Prerrequisitos para evaluación (req. funcionales)	3
Requisitos de implementación	Usar estructuras basadas en listas y/o pares. El tablero debe ser de 7x6 (7 columnas, 6 filas) y debe estar vacío puesto que nadie ha jugado de momento, es un contructor.
Dominio	No recibe parámetros de entrada
Recorrido	*board*

Ejemplo de uso	`(define empty-board (board))`
	
	`(define b0(board))`

5. **RF05. (0.1 pts) TDA Board - otros - sePuedeJugar?**. Función que permite verificar si se puede realizar más jugadas en el tablero.

Nombre función	**board-can-play?**
Prerrequisitos para evaluación (req. funcionales)	4
Requisitos de implementación	Verifica si en el tablero se puede realizar una jugada. Para realizar una jugada en el tablero debe existir al menos una posición disponible (cómo mínimo) Aclaración (30 octubre): dado que el contexto es un juego basado en turnos no se pueden realizar jugadas si ya existe un vencedor, puesto que la partida se dio por terminada.
Dominio	board (board)
Recorrido	*boolean (#t si se puede jugar, #f si no)*

Ejemplo de uso	`(board-can-play? empty-board) # Un tablero vacío retornará siempre #t`
	
	`(board-can-play? updated-board)`

6. **RF06. (0.1 pts) TDA Board - modificador - jugar ficha**. Jugar una ficha en el tablero

Nombre función	**board-set-play-piece**
Prerrequisitos para evaluación (req. funcionales)	5

Requisitos de implementación	Resolver de manera declarativa. La función debe colocar la ficha en la posición más baja disponible de la columna seleccionada.
Dominio	board (board) X column (int) X piece (piece)
Recorrido	*board*
Ejemplo de uso	`(define updated-board (board-set-play-piece empty-board 3 red-piece))`

7. **RF07. (0.2 pts) TDA Board - otros - verificar victoria vertical**. Función que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma vertical.

Nombre función	**board-check-vertical-win**
Prerrequisitos para evaluación (req. funcionales)	6
Requisitos de implementación	Resolver con recursividad natural. La función debe verificar si hay 4 fichas consecutivas del mismo color en cualquier columna.
Dominio	board (board)
Recorrido	*int (1 si gana jugador 1, 2 si gana jugador 2, 0 si no hay ganador vertical)*
Ejemplo de uso	`(board-check-vertical-win updated-board)`

8. **RF08 (0.1 pts) TDA Board - otros - verificar victoria horizontal**. Función que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma horizontal.

Nombre función	**board-check-horizontal-win**

Prerrequisitos para evaluación (req. funcionales)	7
Requisitos de implementación	Resolver con recursividad natural. La función debe verificar si hay 4 fichas consecutivas del mismo color en cualquier fila.
Dominio	board (board)
Recorrido	*int (1 si gana jugador 1, 2 si gana jugador 2, 0 si no hay ganador vertical)*
Ejemplo de uso	`(board-check-horizontal-win updated-board)`

9. **RF09. (0.3 pts) TDA Board - otros - verificar victoria diagonal**. Función que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma diagonal.

Nombre función	**board-check-diagonal-win**
Prerrequisitos para evaluación (req. funcionales)	8
Requisitos de implementación	Resolver con algún tipo de recursión (natural o cola). La función debe verificar si hay 4 fichas consecutivas del mismo color en cualquier diagonal (ascendente o descendente).
Dominio	board (board)
Recorrido	*int (1 si gana jugador 1, 2 si gana jugador 2, 0 si no hay ganador vertical)*
Ejemplo de uso	`(board-check-diagonal-win updated-board)`

10. **RF10. (0.1 pts) TDA Board - otros - entregarGanador**. Función que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma diagonal.

Nombre función	**board-who-is-winner**
Prerrequisitos para evaluación (req. funcionales)	7,8,9
Requisitos de implementación	Utilizar las funciones de verificación vertical, horizontal y diagonal para determinar si hay un ganador. Un ganador puede ser vertical, horizontal o diagonal. Realizar de forma declarativa.
Dominio	board (board)
Recorrido	*int (1 si gana jugador 1, 2 si gana jugador 2, 0 si no hay ganador vertical)*
Ejemplo de uso	`(board-who-is-winner updated-board)`

11. **RF11. (0.1 pts) TDA Game - constructor**. Función que permite crear una nueva partida.

Nombre función	**game**
Prerrequisitos para evaluación (req. funcionales)	10
Requisitos de implementación	Usar estructuras basadas en listas y/o pares
Dominio	player1 (player) X player2 (player) X board (board) X current-turn (int)
Recorrido	*game*

Ejemplo de uso	`(define new-game (game p1 p2 empty-board 1))`

12. **RF12. (0.1 pts) TDA Game - otros - history**. Función que genera un historial de movimientos de la partida.

Nombre función	**game-history**
Prerrequisitos para evaluación (req. funcionales)	11
Requisitos de implementación	Resolver de manera declarativa.
Dominio	game (game)
Recorrido	*History*
Ejemplo de uso	`(game-history updated-game)`

13. **RF13. (0.1 pts) TDA Game - otros - esEmpate?**. Función que verifica si el estado actual del juego es empate.

Nombre función	**game-is-draw?**
Prerrequisitos para evaluación (req. funcionales)	12
Requisitos de implementación	Verificar si el tablero está lleno o si ambos jugadores se han quedado sin fichas. Tablero lleno significa que todas las posiciones del mismo se encuentran ocupadas.
Dominio	game (game)

Recorrido	*boolean (#t si es empate, #f si no)*
Ejemplo de uso	`(game-is-draw? current-game)`

14. **RF14. (0.1 pts) TDA Player - otros - actualizarEstadisticas**. Función que actualiza las estadísticas del jugador, ya sea victoria, derrotas o empates.

Nombre función	**player-update-stats**
Prerrequisitos para evaluación (req. funcionales)	13
Requisitos de implementación	Actualizar las estadísticas del jugador (victorias, derrotas o empates) después de finalizar un juego.
Dominio	player (player) X result (string: "win", "loss", o "draw") En caso de que result sea "win" actualiza contador de victorias, "loss" derrotas y "draw" empate.
Recorrido	*Player*
Ejemplo de uso	`(define updated-player (player-update-stats p1 "win"))`

15. **RF15. (0.1 pts) TDA Game - selector - getCurrentPlayer**. Función que obtiene el jugador cuyo turno está en curso.

Nombre función	**game-get-current-player**
Prerrequisitos para evaluación (req. funcionales)	14
Requisitos de implementación	Obtener el jugador cuyo turno está en curso.

Dominio	game (game)
Recorrido	*player*
Ejemplo de uso	`(game-get-current-player current-game)`

16. **RF16. (0.1 pts) TDA Game - selector - board-get-state**. Función que entrega el estado actual del tablero en el juego.

Nombre función	**game-get-board**
Prerrequisitos para evaluación (req. funcionales)	15
Requisitos de implementación	Obtener una representación del estado actual del tablero.
Dominio	Game
Recorrido	*Board*
Ejemplo de uso	`(game-get-board current-game)`

17. **RF17. (0.1 pts) TDA Game - modificador - game-set-end**. Función finaliza el juego actualizando las estadísticas de los jugadores según el resultado.

Nombre función	**game-set-end**
Prerrequisitos para evaluación (req. funcionales)	15
Requisitos de implementación	Finalizar el juego, actualizando las estadísticas de los jugadores según el resultado.

Dominio	Game (game)
Recorrido	*game*
Ejemplo de uso	`(define ended-game (game-set-end current-game))`

18. **RF18. (3,5 pts) TDA Game - modificador - realizarMovimiento**. Función que realiza un movimiento.

Nombre función	**game-player-set-move**
Prerrequisitos para evaluación (req. funcionales)	7
Requisitos de implementación	Resolver de manera declarativa. - Verificar que sea el turno del jugador correcto. - Actualizar el estado del juego después de realizar un movimiento: ● Colocar la pieza en el tablero. ● Cambiar el turno al otro jugador. ● Disminuir la cantidad de fichas del jugador que realizó el movimiento. ● Actualizar el historial de movimientos. - Verificar si hay un ganador o si el juego ha terminado en empate después del movimiento. - Si el juego ha terminado, actualizar las estadísticas de ambos jugadores.
Dominio	game (game) X player (player) X column (int) (define g1 (game-player-set-move blabla) (define g2 (game-player-set-move g1 blabla)
Recorrido	game
Ejemplo de uso	`(define updated-game (game-player-set-move current-game p1 3))`

19. **(BONUS) RF19. (2,5 pts) TDA Game - modificador - `game-ai-set-move`** Función que se encarga de que Google AI Studio (Google Gemini) pueda jugar Conecta4 con el jugador.

Nombre función	**`game-ai-set-move`**
Prerrequisitos para evaluación (req. funcionales)	17

Requisitos de implementación	• Utilizar la API de Google AI Studio (Google Gemini) para obtener la decisión de la jugada.
	• Pasar el estado actual del juego a Google AI Studio (Google Gemini) de manera que pueda tomar una decisión informada.
	• Interpretar la respuesta de Google AI Studio (Google Gemini) y convertirla en un movimiento válido.
	• Realizar el movimiento en el tablero.
	• Actualizar el estado del juego después de realizar el movimiento:
	  ◦ Colocar la pieza en el tablero.
	  ◦ Cambiar el turno al jugador humano.
	  ◦ Disminuir la cantidad de fichas de la IA.
	  ◦ Actualizar el historial de movimientos.
	• Verificar si hay un ganador o si el juego ha terminado en empate después del movimiento.
	• Si el juego ha terminado, actualizar las estadísticas de ambos jugadores.

Dominio	game (game)
Recorrido	game
Ejemplo de uso	`(define updated-game (game-ai-set-move current-game))`

Ejemplo de prompt para Google AI Studio (Google Gemini) API:

Usted debe preocuparse de cómo alimentar su modelo de GenAI para obtener resultados precisos. A continuación sólo se presenta un ejemplo. Cabe destacar que mientras más información de la partida se entregue al servicio de Google AI Studio mejores resultados obtendrá en términos de la respuesta hacia su programa. No es necesario que usted pregunte en este documento cómo realizar mejores consultas hacia el servicio (prompt), puesto que es responsabilidad del estudiante la investigación de este apartado.

"Este es el tablero actual **[representación del tablero]**. Es tu turno para jugar. Dame el número de la columna (0-6) donde deseas colocar tu próxima ficha."

20. **(BONUS) RF20. (1 pts) TDA Game - otros - game-ai-explain-history** Función que se encarga de que Google AI Studio (Google Gemini) explique el historial de las jugadas (independiente si es contra un player humano o IA) en español.

Nombre función	**game-ai-explain-history**
Prerrequisitos para evaluación (req. funcionales)	17
Requisitos de implementación	• Utilizar la API de Google AI Studio (Google Gemini) para que explique en español el historial de jugadas actualmente almacenado. • Pasar el estado actual del juego a Google AI Studio (Google Gemini) de manera que pueda tomar una decisión informada. • Interpretar la respuesta de Google AI Studio (Google Gemini). • Realizar el movimiento en el tablero.
Dominio	game (game)
Recorrido	string

Ejemplo de uso	`(define current-game-explanation (game-ai-explain-history current-game))`
	
	`;; Mostrar en pantalla la explicación provista por Google AI Studio`
	
	`(display current-game-explanation)`

Dado que la nota base es 1.0 realizando de forma correcta y sin errores hasta el RF20, cada puntaje se suma y la nota final es 7.3, por lo que se aproxima al máximo (7).

Integración con API Google AI Studio (Google Gemini):

El primer paso para realizar la integración con Google AI Studio es realizar lo mencionado en el enunciado general sección Integración con :

Ver Laboratorio enunciado general.

Para lograr la integración desde Scheme hacia Google Gemini (Google AI Studio) debe usar las funciones provistas por el lenguaje. En particular, Racket provee las funciones "json" y "net/url" las cuales ayudarán con la integración pedida. En particular se debe construir funciones que realicen una llamada POST al servicio de Google Gemini. Leer el enunciado general del laboratorio para más información. A continuación un ejemplo que les puede orientar cómo realizar dicha integración. Esto es sólo un ejemplo, no es que usted sea obligado a utilizar estas librerías, si encuentra otras provistas por el lenguaje puede usarlo, no hay restricciones.

```
#lang racket

(require net/url)

(require json)
```

En este laboratorio puede usar todos los recursos y funciones provistas por el lenguaje (Racket y Scheme). No es necesario preguntar cómo realizar o qué funciones pueden utilizar para realizar la integración puede usar una cierta función puesto que no hay límites en el uso del lenguaje en estos requerimientos de integración con Google AI Studio (Google Gemini).

Es responsabilidad del estudiante investigar e implementar la integración con Google AI Studio. Estos aspectos no serán revisados en clases ni tampoco responderemos preguntas sobre cómo usar el servicio de Google AI Studio o la integración de Racket/Scheme con dicho servicio.

Cualquier uso de documentación del lenguaje, otros medios online o herramientas GenAI debe estar declarado en su informe y código a través de un comentario. Es importante que usted declaré específicamente que uso y obtuvo de herramientas del tipo GenAI, caso contrario se tomará como plagio y se asignará la nota mínima 1.0 al no tener evidencia de que es trabajo propio.

Documentación de ayuda:

https://ai.google.dev/gemini-api/docs/quickstart?hl=es-419&lang=rest
JSON - Racket Documentation

How do I parse JSON in Racket? - Stack Overflow

2 URLs and HTTP - Racket Documentation

Understanding HTTP methods in REST API development

Tips para búsqueda: "Realizar petición POST en Racket/Scheme"

Script de ejecución

SCRIPT DE EJECUCIÓN (script_base_RUT_NOMBRE_APELLIDOS.rkt)

Usted también debe entregar 2 scripts más escritos de la siguiente manera:

SCRIPT DE EJECUCIÓN PROPIO 1 (script1_RUT_NOMBRE_APELLIDOS.rkt)

SCRIPT DE EJECUCIÓN PROPIO 1 (script2_RUT_NOMBRE_APELLIDOS.rkt)

Los scripts no deben tener código comentado. La revisión tomará cada script y lo ejecutará, usted debe asegurar que el script funciona sin que nosotros los revisores tengamos que realizar modificación alguna o tengamos que copiar y pegar código en algún otro archivo.

IMPORTANTE: NO ELIMINEN ESTE SCRIPT NI SUGIERAN CAMBIOS

Script de ejecución base (script_base_RUT_NOMBRE_APELLIDOS.rkt)

```
; Script de prueba para Conecta4


; 1. Creación de jugadores (10 fichas cada uno para un juego corto)
(define p1 (player 1 "Juan" "red" 0 0 0 10))
(define p2 (player 2 "Mauricio" "yellow" 0 0 0 10))


; 2. Creación de piezas
(define red-piece (piece "red"))
(define yellow-piece (piece "yellow"))


; 3. Creación del tablero inicial
(define empty-board (board))


; 4. Creación de un nuevo juego
(define g0 (game p1 p2 empty-board 1))


; 5. Realizando movimientos para crear una victoria diagonal
```

```
(define g1 (game-player-set-move g0 p1 0))  ; Juan coloca en columna 0
(define g2 (game-player-set-move g1 p2 1))  ; Mauricio coloca en columna 1
(define g3 (game-player-set-move g2 p1 1))  ; Juan coloca en columna 1
(define g4 (game-player-set-move g3 p2 2))  ; Mauricio coloca en columna 2
(define g5 (game-player-set-move g4 p1 2))  ; Juan coloca en columna 2
(define g6 (game-player-set-move g5 p2 3))  ; Mauricio coloca en columna 3
(define g7 (game-player-set-move g6 p1 2))  ; Juan coloca en columna 2
(define g8 (game-player-set-move g7 p2 3))  ; Mauricio coloca en columna 3
(define g9 (game-player-set-move g8 p1 3))  ; Juan coloca en columna 3
(define g10 (game-player-set-move g9 p2 0)) ; Mauricio coloca en columna 0
(define g11 (game-player-set-move g10 p1 3)) ; Juan coloca en columna 3
(victoria diagonal)

; 6. Verificaciones durante el juego
(display "¿Se puede jugar en el tablero vacío? ")
(board-can-play? empty-board)

(display "¿Se puede jugar después de 11 movimientos? ")
(board-can-play? (game-get-board g11))

(display "Jugador actual después de 11 movimientos: ")
(game-get-current-player g11)

; 7. Verificación de victoria
(display "Verificación de victoria vertical: ")
(board-check-vertical-win (game-get-board g11))

(display "Verificación de victoria horizontal: ")
(board-check-horizontal-win (game-get-board g11))
```

```
(display "Verificación de victoria diagonal: ")
(board-check-diagonal-win (game-get-board g11))

(display "Verificación de ganador: ")
(board-who-is-winner (game-get-board g11))

; 8. Verificación de empate
(display "¿Es empate? ")
(game-is-draw? g11)

; 9. Finalizar el juego y actualizar estadísticas
(define ended-game (game-set-end g11))

; 10. Actualizar estadísticas de los jugadores
(define updated-p1 (player-update-stats p1 "win"))
(define updated-p2 (player-update-stats p2 "loss"))

; 10. Mostrar historial de movimientos
(display "Historial de movimientos: ")
(game-history ended-game)

; 11. Mostrar estado final del tablero
(display "Estado final del tablero: ")
(game-get-board ended-game)
```