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

Paradigmas de Programación

Proyecto semestral de laboratorio

Versión Preliminar - actualizada al 20/09/2023

Laboratorio 1 (Paradigma Funcional - Lenguaje Scheme)

Versión 2.1 : Ajuste de puntajes. Incorporación de nuevas funciones. Modificación de dominios de algunas funciones. Actualización de descripciones y requisitos de implementación de funciones (ej. se añade initialFlowCode en constructor de chatbot)

(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: Programa para realizar una simulación de un chatbot.

Profesor responsable: Roberto González (al hacer consultas en este documento, procurar hacer la mención a **@Roberto Gonzalez Ibanez** (roberto.gonzalez.i@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. Algunos son ineludibles/obligatorios, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

1. **(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.

2. **(obligatorio) Lenguaje:** La implementación debe ser en el lenguaje de programación Scheme/Racket en base a una programación principalmente declarativa - funcional.

3. **(obligatorio) Versión:** Usar DrRacket versión 6.11 o superior

4. **(obligatorio) Standard:** Se deben utilizar funciones estándar del lenguaje. No emplear bibliotecas externas.

5. **(obligatorio) No variables:** No hacer uso de función *define* **en combinación** con otras como set! (o similares) para emular el trabajo con variables.

6. **(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.

7. **(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. **(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.

9. **(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. **(obligatorio) Script de pruebas (pruebas_RUT_Apellidos.rkt):** Incluir como parte de su entregable un archivo independiente al código donde muestre de forma completa, con la documentación correspondiente, el funcionamiento de su programa. 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.**

11. **(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. **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. **(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 “system”, “chatbot”, “flow”, “option”, “user” y “chatHistory” que 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).

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. **(0.2 pts) TDA Option - constructor**. Función constructora de una opción para flujo de un chatbot. Cada opción se enlaza a un chatbot y flujo especificados por sus respectivos códigos.

Nombre función	option
Prerrequisitos para evaluación (req. funcionales)	1
Requisitos de implementación	Usar estructuras basadas en listas y/o pares. El código de un option es único. Su unicidad se verifica al momento de agregarlo a un flow a fin de evitar duplicidad.
Dominio	code (Int) X message (String) X ChatbotCodeLink (Int) X InitialFlowCodeLink (Int) X Keyword* (en referencia a 0 o más palabras claves)
Recorrido	*option*
Ejemplo de uso	;creando opciones ;opción 1 vinculada al chatbot 2 con su flujo 4 (asumiendo su existencia) en sistema (define op1 (option 1 “1) Viajar” 2 4 “viajar” “turistear” “conocer”)) ;opción 1 vinculada al chatbot 4 con su flujo 3 (asumiendo su existencia) en sistema (define op2 (option 2 “2) Estudiar” 4 3 “aprender” “perfeccionarme”))

3. **(0.4 pts) TDA Flow - constructor**. Función constructora de un flujo de un chatbot.

Nombre función	flow
Prerrequisitos para evaluación (req. funcionales)	2
Requisitos de implementación	Usar estructuras basadas en listas y/o pares Flujos quedan identificados por un ID único. Su unicidad se verifica al momento de agregarlo a un chatbot a fin de evitar duplicidad. La función también verifica que las opciones añadidas no se repitan en base al id de éstos.
Dominio	id (int) X name-msg (String) X Option* (Indica que puede recibir cero o más opciones)
Recorrido	*flow*
Ejemplo de uso	;creando un nuevo flow (define f10 (flow 1 “Flujo1: mensaje de prueba”)) ;alternativamente podría usarse: (define f12 (flow 1 “Flujo1: mensaje de prueba” op1 op2))

4. **(0.2 pts) TDA Flow - modificador**. Función modificadora para añadir opciones a un flujo.

Nombre función	flow-add-option
Prerrequisitos para evaluación (req. funcionales)	3
Requisitos de implementación	No usar recursión La función también verifica que las opciones añadidas no se repitan en base al id de éstos.
Dominio	flow X option
Recorrido	*flow*
Ejemplo de uso	;añadiendo opciones 1 y 2 al flujo f10 ;el resultado alcanzado en f12 es equivalente al ilustrado en f12 de la función 3. (define f11 (flow-add-option f10 op1)) (define f12 (flow-add-option f11 op2))

5. **(0.4 pts) TDA chatbot - constructor**. Función constructora de un chatbot.

Nombre función	chatbot
Prerrequisitos para evaluación (req. funcionales)	4
Requisitos de implementación	Usar estructuras basadas en listas y/o pares Chatbots quedan identificados por un ID único. Su unicidad se verifica al momento de agregarlo a un sistema a fin de evitar duplicidad. La función también verifica que los flujos añadidos no se repitan en base al id de éstos.
Dominio	chatbotID (int) X name (String) X welcomeMessage (String) X startFlowId(int) X flows* (indicando que puede recibir 0 o más flujos)
Recorrido	*chatbot*
Ejemplo de uso	;creando un nuevo chatbot (define cb10 (chatbot 0 “Asistente” “Bienvenido\n¿Qué te gustaría hacer?” 1)) ;alternativamente podría usarse: (define cb11 (chatbot 0 “Asistente” “Bienvenido\n¿Qué te gustaría hacer?” 1 f12))

6. **(0.2 pts) TDA chatbot - modificador**. Función modificadora para añadir flujos a un chatbot.

Nombre función	chatbot-add-flow
Prerrequisitos para evaluación (req. funcionales)	5
Requisitos de implementación	Usar recursión de cola o natural para añadir flujos al final de la lista de flujos. La función también verifica que los flujos añadidos no se repitan en base al id de éstos.
Dominio	chatbot X flow
Recorrido	*chatbot*
Ejemplo de uso	;añadiendo flujo a un chatbot ;el resultado alcanzado en cb11 es equivalente al ilustrado en cb11 de la función 5. (define cb11 (chatbot-add-flow cb10 f12))

7. **(0.4 pts) TDA system - constructor**. Función constructora de un sistema de chatbots. Deja registro de la fecha de creación.

Nombre función	system
Prerrequisitos para evaluación (req. funcionales)	6
Requisitos de implementación	Usar estructuras basadas en listas y/o pares El sistema además de contener los distintos chatbots, también contiene el chatHistory de cada usuario que interactúa con el sistema. Sobre el chatHistory, éste corresponde al registro completo del diálogo entre usuario y cada uno de los chatbots con los que interactúa. El historial se mantiene para cada usuario y debe tener el String formateado de cada mensaje del usuario y chatbot (para luego ser visualizado con la función display), fecha, hora y emisor (usuario o sistema).
Dominio	name (string) X InitialChatbotCodeLink (Int) X chatbot* (indicando que pueden recibir 0 o más chatbots)
Recorrido	*system*
Ejemplo de uso	;creando la un nuevo sistema de chatbots con nombre “NewSystem” (define s0 (system “NewSystem” 0)) ;alternativamente podría usarse: (define s1 (system “NewSystem” cb11))

8. **(0.2 pts) TDA system - modificador**. Función modificadora para añadir chatbots a un sistema.

Nombre función	system-add-chatbot
Prerrequisitos para evaluación (req. funcionales)	7
Requisitos de implementación	No usar recursividad Debe verificar que el chatbot no exista en el sistema a partir del id de éste.
Dominio	system X chatbot
Recorrido	*system*
Ejemplo de uso	;añadiendo un chatbot al sistema. ;el resultado alcanzado en s1 es equivalente al ilustrado en s1 de la función 7. (define s1 (system-add-chatbot s0 cb11))

9. **(0.2 pts) TDA system - modificador**. Función modificadora para añadir usuarios a un sistema.

Nombre función	system-add-user
Prerrequisitos para evaluación (req. funcionales)	8
Requisitos de implementación	Debe verificar que el usuario no exista en el sistema a partir del id de éste, que está dado por su nombre de usuario (String).
Dominio	system X user (string)
Recorrido	*system*
Ejemplo de uso	;añadiendo dos usuarios al sistema (define s2 (system-add-user s1 “user0”)) (define s3 (system-add-user s2 “user1”))

10. **(0.2 pts) TDA system**. Función que permite iniciar una sesión en el sistema.

Nombre función	system-login
Prerrequisitos para evaluación (req. funcionales)	9
Requisitos de implementación	Solo pueden iniciar sesión usuarios registrados mediante la función register system-add-user. No se puede iniciar sesión si ya existe una sesión iniciada por otro usuario.
Dominio	system X user (string)
Recorrido	*system*
Ejemplo de uso	;iniciando una sesión de usuario (define s4 (system-login s3 “user0”))

11. **(0.2 pts) TDA system**. Función que permite cerrar una sesión abierta.

Nombre función	system-logout
Prerrequisitos para evaluación (req. funcionales)	10
Requisitos de implementación	Sin requisitos especiales
Dominio	system
Recorrido	*system*
Ejemplo de uso	;cerrando una sesión de usuario (define s1000 (system-logout s4))

12. **(1 pto) system-talk-rec**. Función que permite interactuar con un chatbot.

Nombre función	system-talk-rec
Prerrequisitos para evaluación (req. funcionales)	11
Requisitos de implementación	Usar recursividad Solo se puede conversar si se ha iniciado una sesión con un usuario previamente registrado.
Dominio	system X message (string) ;message puede ser la opción o palabra clave
Recorrido	*system*
Ejemplo de uso	;añadiendo un chatbot al sistema. (define s5 (system-talk-rec s4 “hola”)) (define s6 (system-talk-rec s5 “viajar”))

13. **(1 pto) system-talk-norec**. Función que permite interactuar con un chatbot. Mismo propósito de la función anterior pero con una implementación declarativa.

Nombre función	system-talk-norec
Prerrequisitos para evaluación (req. funcionales)	12
Requisitos de implementación	No usar recursividad. Procurar que la implementación de la función sea principalmente declarativa.
Dominio	system X message (string) ;message puede ser la opción o palabra clave
Recorrido	*system*
Ejemplo de uso	;añadiendo un chatbot al sistema. (define s7 (system-talk-rec s6 “placer”)) (define s8 (system-talk-rec s7 “solo”))

14. **(0,4 pto) system-synthesis**. Función que ofrece una síntesis del chatbot para un usuario particular a partir de chatHistory contenido dentro del sistema

Nombre función	system-synthesis
Prerrequisitos para evaluación (req. funcionales)	13
Requisitos de implementación	No usar recursividad. Debe ser una implementación declarativa.
Dominio	system X usuario (String)
Recorrido	*string (formateado para poder visualizarlo con display)*
Ejemplo de uso	;solicitando una síntesis del sistema. (system-synthesis s8 “user0”)

15. **(0,5 pto) system-simulate**. Permite simular un diálogo entre dos chatbots del sistema.

Nombre función	system-simulate
Prerrequisitos para evaluación (req. funcionales)	14
Requisitos de implementación	El registro de las interacciones queda en chatHistory. La semilla (seed) se establece para generar números aleatorios respetando principio de transparencia referencial. Usar la función myRandom provista al final del enunciado. El número máximo de interacciones es una cota superior. Si no se pueden generar más interacciones, la simulación termina.
Dominio	system X maxInteractions (int) X seed (int)
Recorrido	*system*
Ejemplo de uso	;solicitando una simulación al sistema. (system-simulate s8 5 3212312)

Función para generar números pseudoaleatorios:

```
(define (myRandom Xn)
  (modulo (+ (* 1103515245 Xn) 12345) 2147483648)
)
```

Script de Pruebas N°1

```
(define op1 (option  1 "1) Viajar" 2 1 "viajar" "turistear" "conocer"))

(define op2 (option  2 "2) Estudiar" 3 1 "estudiar" "aprender" "perfeccionarme"))

(define f10 (flow 1 "flujo1" op1 op2 op2 op2 op2 op1)) ;solo añade una ocurrencia de op2

(define f11 (flow-add-option f10 op1)) ;se intenta añadir opción duplicada

(define cb0 (chatbot 0 "Inicial" "Bienvenido\n¿Qué te gustaría hacer?" 1 f10 f10 f10 f10))  ;solo añade una ocurrencia de f10

(define s0 (system "Chatbots Paradigmas" 0 cb0 cb0 cb0))

(define s1 (system-add-chatbot s0 cb0)) ;igual a s0

(define s2 (system-add-user s1 "user1"))

(define s3 (system-add-user s2 "user2"))

(define s4 (system-add-user s3 "user2")) ;solo añade un ocurrencia de user2

(define s5 (system-add-user s4 "user3"))

(define s6 (system-login s5 "user8")) ;user8 no existe. No inicia sesión

(define s7 (system-login s6 "user1"))

(define s8 (system-login s7 "user2"))  ;no permite iniciar sesión a user2, pues user1 ya inició sesión

(define s9 (system-logout s8))

(define s10 (system-login s9 "user2"))
```

Script de Pruebas N°2

;Ejemplo de un sistema de chatbots basado en el esquema del enunciado general

;Chabot0

(define op1 (option 1 "1) Viajar" 1 1 "viajar" "turistear" "conocer"))

(define op2 (option 2 "2) Estudiar" 2 1 "estudiar" "aprender" "perfeccionarme"))

(define f10 (flow 1 "Flujo Principal Chatbot 1\nBienvenido\n¿Qué te gustaría hacer?" op1 op2 op2 op2 op2 op1)) ;solo añade una ocurrencia de op2 y op1

(define f11 (flow-add-option f10 op1)) ;se intenta añadir opción duplicada

(define cb0 (chatbot 0 "Inicial" "Bienvenido\n¿Qué te gustaría hacer?" 1 f10 f10 f10 f10)) ;solo añade una ocurrencia de f10

;Chatbot1

(define op3 (option 1 "1) New York, USA" 1 2 "USA" "Estados Unidos" "New York"))

(define op4 (option 2 "2) París, Francia" 1 1 "Paris" "Eiffel"))

(define op5 (option 3 "3) Torres del Paine, Chile" 1 1 "Chile" "Torres" "Paine" "Torres Paine" "Torres del Paine"))

(define op6 (option 4 "4) Volver" 0 1 "Regresar" "Salir" "Volver"))

;Opciones segundo flujo Chatbot1

(define op7 (option 1 "1) Central Park" 1 2 "Central" "Park" "Central Park"))

(define op8 (option 2 "2) Museos" 1 2 "Museo"))

(define op9 (option 3 "3) Ningún otro atractivo" 1 3 "Museo"))

(define op10 (option 4 "4) Cambiar destino" 1 1 "Cambiar" "Volver" "Salir"))

(define op11 (option 1 "1) Solo" 1 3 "Solo"))

(define op12 (option 2 "2) En pareja" 1 3 "Pareja"))

(define op13 (option 3 "3) En familia" 1 3 "Familia"))

(define op14 (option 4 "4) Agregar más atractivos" 1 2 "Volver" "Atractivos"))

(define op15 (option 5 "5) En realidad quiero otro destino" 1 1 "Cambiar destino"))

(define f20 (flow 1 "Flujo 1 Chatbot1\n¿Dónde te Gustaría ir?" op3 op4 op5 op6))

(define f21 (flow 2 "Flujo 2 Chatbot1\n¿Qué atractivos te gustaría visitar?" op7 op8 op9 op10))

(define f22 (flow 3 "Flujo 3 Chatbot1\n¿Vas solo o acompañado?" op11 op12 op13 op14 op15))

(define cb1 (chatbot 1 "Agencia Viajes" "Bienvenido\n¿Dónde quieres viajar?" 1 f20 f21 f22))

;Chatbot2

(define op16 (option 1 "1) Carrera Técnica" 2 1 "Técnica"))

(define op17 (option 2 "2) Postgrado" 2 1 "Doctorado" "Magister" "Postgrado"))

(define op18 (option 3 "3) Volver" 0 1 "Volver" "Salir" "Regresar"))

```
(define f30 (flow 1 "Flujo 1 Chatbot2\n¿Qué te gustaría estudiar?" op16 op17 op18))
(define cb2 (chatbot 2 "Orientador Académico"  "Bienvenido\n¿Qué te gustaría estudiar?" 1 f30))
```

;Sistema

```
(define s0 (system "Chatbots Paradigmas" 0 cb0 cb0 cb0 cb1 cb2))
(define s1 (system-add-chatbot s0 cb0)) ;igual a s0
(define s2 (system-add-user s1 "user1"))
(define s3 (system-add-user s2 "user2"))
(define s4 (system-add-user s3 "user2"))
(define s5 (system-add-user s4 "user3"))
(define s6 (system-login s5 "user8"))
(define s7 (system-login s6 "user1"))
(define s8 (system-login s7 "user2"))
(define s9 (system-logout s8))
(define s10 (system-login s9 "user2"))
(define s1 (system-add-chatbot s0 cb0)) ;igual a s0
(define s2 (system-add-user s1 "user1"))
(define s3 (system-add-user s2 "user2"))
(define s4 (system-add-user s3 "user2"))
(define s5 (system-add-user s4 "user3"))
(define s6 (system-login s5 "user8"))
(define s7 (system-login s6 "user1"))
(define s8 (system-login s7 "user2"))
(define s9 (system-logout s8))
(define s10 (system-login s9 "user2"))
;las siguientes interacciones deben funcionar de igual manera con system-talk-rec  o system-talk-norec
(define s11 (system-talk-rec s10 "hola"))
(define s12 (system-talk-rec s11 "1"))
(define s13 (system-talk-rec s12 "1"))
(define s14 (system-talk-rec s13 "Museo"))
(define s15 (system-talk-rec s14 "1"))
(define s16 (system-talk-rec s15 "3"))
(define s17 (system-talk-rec s16 "5"))
(display (system-synthesis s17 "user2"))
(system-simulate s0 5 32131)
```

Ejemplo de Output formateador para (display (system-synthesis s17 "user2"))

1695355052 - user2: hola

1695355052 - Inicial: Flujo Principal Chatbot 1

Bienvenido

¿Qué te gustaría hacer?

1) Viajar

2) Estudiar

1695355052 - user2: 1

1695355052 - Agencia Viajes: Flujo 1 Chatbot1

¿Dónde te Gustaría ir?

1) New York, USA

2) París, Francia

3) Torres del Paine, Chile

4) Volver

1695355052 - user2: 1

1695355052 - Agencia Viajes: Flujo 2 Chatbot1

¿Qué atractivos te gustaría visitar?

1) Central Park

2) Museos

3) Ningun otro atractivo

4) Cambiar destino

1695355052 - user2: Museo

1695355052 - Agencia Viajes: Flujo 2 Chatbot1

¿Qué atractivos te gustaría visitar?

1) Central Park

2) Museos

3) Ningun otro atractivo

4) Cambiar destino

1695355052 - user2: 1

1695355052 - Agencia Viajes: Flujo 2 Chatbot1

¿Qué atractivos te gustaría visitar?

1) Central Park

2) Museos

3) Ningun otro atractivo

4) Cambiar destino

1695355052 - user2: 3

1695355052 - Agencia Viajes: Flujo 3 Chatbot1

¿Vas solo o acompañado?

1) Solo

2) En pareja

3) En familia

4) Agregar mas atractivos

5) En realidad quiero otro destino

1695355052 - user2: 5

1695355052 - Agencia Viajes: Flujo 1 Chatbot1

¿Dónde te Gustaría ir?

1) New York, USA

2) París, Francia

3) Torres del Paine, Chile

4) Volver

Ejemplo de Output para:

(display (system-synthesis (system-simulate s0 5 312321) "user312321"))

1695486245 - user312321: Hola

1695486245 - Inicial: Flujo Principal Chatbot 1

Bienvenido

¿Qué te gustaría hacer?

1) Viajar

2) Estudiar

1695486245 - user312321: 1

1695486245 - Agencia Viajes: Flujo 1 Chatbot1

¿Dónde te Gustaría ir?

1) New York, USA

2) París, Francia

3) Torres del Paine, Chile

4) Volver

1695486245 - user312321: 4

1695486245 - Inicial: Flujo Principal Chatbot 1

Bienvenido

¿Qué te gustaría hacer?

1) Viajar

2) Estudiar

1695486245 - user312321: 1

1695486245 - Agencia Viajes: Flujo 1 Chatbot1

¿Dónde te Gustaría ir?

1) New York, USA

2) París, Francia

3) Torres del Paine, Chile

4) Volver

1695486245 - user312321: 1

1695486245 - Agencia Viajes: Flujo 2 Chatbot1

¿Qué atractivos te gustaría visitar?

1) Central Park

2) Museos

3) Ningun otro atractivo

4) Cambiar destino

1695486245 - user312321: 3

1695486245 - Agencia Viajes: Flujo 3 Chatbot1

¿Vas solo o acompañado?

1) Solo

2) En pareja

3) En familia

4) Agregar mas atractivos

5) En realidad quiero otro destino