|  |  |
| --- | --- |
| **UNIVERSIDAD DE SANTIAGO DE CHILE**  **FACULTAD DE INGENIERÍA**  **DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA** |  |



**Manual de Usuario**

**Chatbot**

**Programado en Scheme**

Gabriel Gaete L.

Santiago de Chile

1 - 2018

# TABLA DE CONTENIDO

[CAPÍTULO 1. INTRODUCCIÓN 4](#_Toc512107751)

[CAPÍTULO 2. MARCO TEÓRICO 5](#_Toc512107752)

[2.1 PARADIGMA FUNCIONAL 5](#_Toc512107753)

[2.2 TIPO DE DATO ABSTRACTO (TDA) 6](#_Toc512107754)

[CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA 6](#_Toc512107756)

[3.1 ANÁLISIS 7](#_Toc512107757)

[CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN 7](#_Toc512107758)

[CAPÍTULO 5. RESULTADOS 9](#_Toc512107759)

[CAPÍTULO 6. CONCLUSIONES 11](#_Toc512107760)

[CAPÍTULO 7. REFERENCIAS 12](#_Toc512107761)

# TABLA DE FIGURAS

[Figura 1 Representación de un mensaje 6](#_Toc512032646)

[Figura 2 Ejemplo de conversación simulada a través de la función test 7](#_Toc512032647)

[Figura 3 Estructura chatbot1 8](#_Toc512032648)

[Figura 4 Ejemplo de composición de funciones para generar un chatbot 9](#_Toc512032649)

# CAPÍTULO 1. INTRODUCCIÓN AL CHATBOT

Un chatbot es un programa que permite simular una conversación con una persona, entregando respuestas automáticas a entradas hechas por un usuario. Estos son utilizados por diferentes marcas y compañías para obtener información, reservar algo, o comprar un producto, entre muchas otras aplicaciones. El chatbot del presente informe permite hacer una compra de pasajes hacia capitales regionales de Chile, asumiendo que el usuario se encuentra en Santiago.

El presente manual pretende explicar cómo utilizar de manera efectiva el chatbot, a través de ejemplos de llamadas a las funciones del programa.

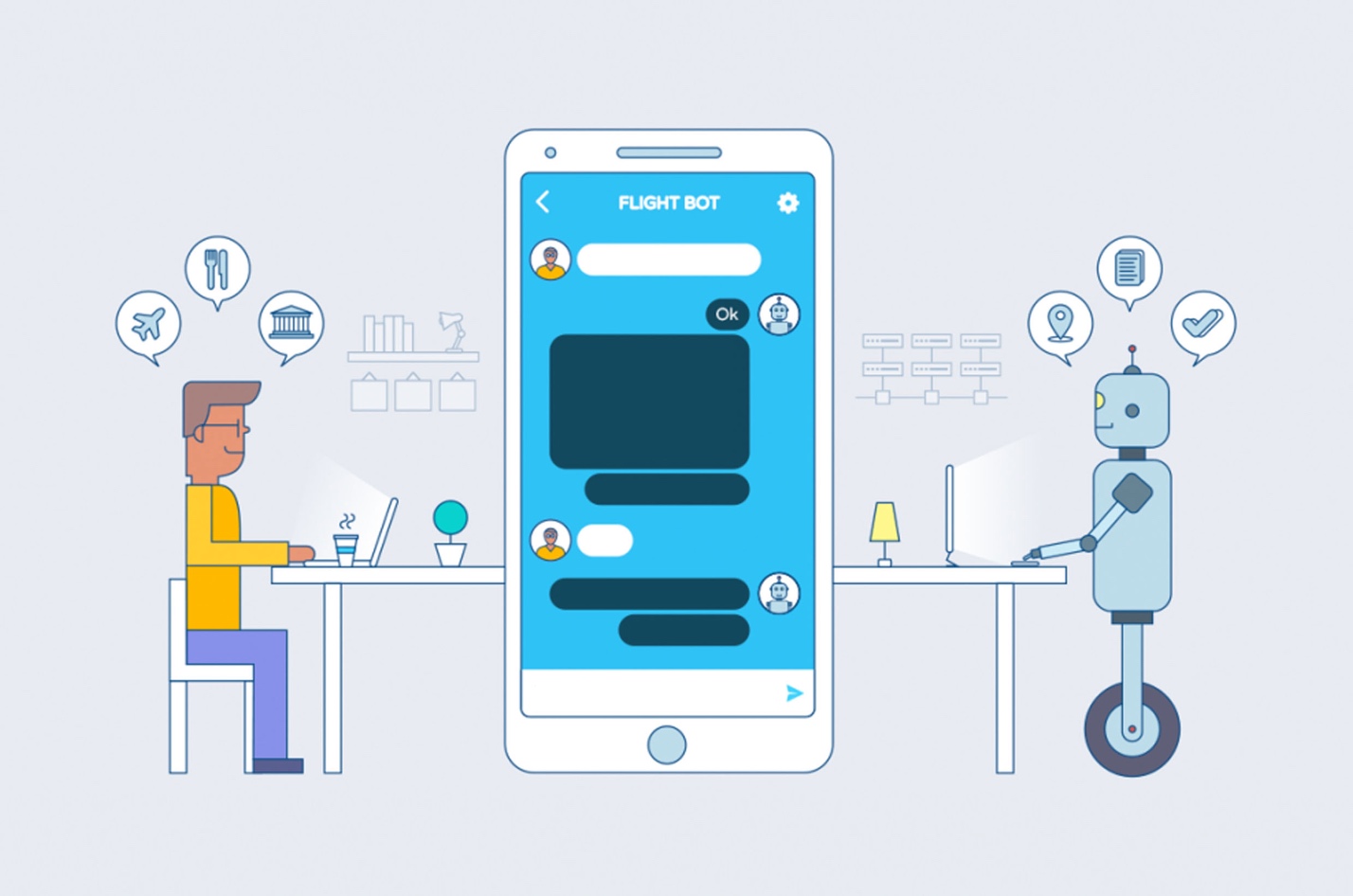


Figura 1 Relación usuario/chatbot (Fuente: http://www.tci.net.pe/sera-2018-ano-los-chatbots/)

Para la correcta utilización de este programa, se debe conocer cómo es el flujo conversacional esperado por el bot frente al usuario, por lo que se recomienda guiarse por el siguiente diagrama para evitar posibles errores en el flujo de la conversación.

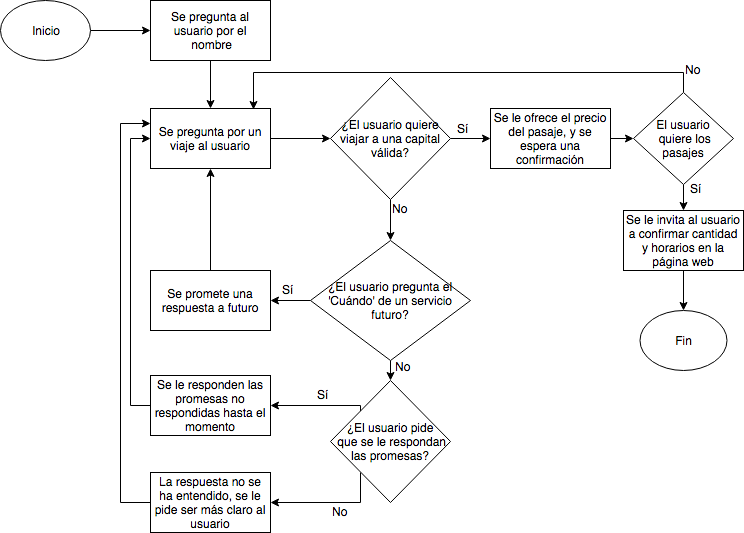


Figura 2 Flujo de conversación esperado por el chatbot

# CAPÍTULO 2. COMPILACIÓN Y EJECUCIÓN

Para ejecutar el programa, solo basta con abrir el archivo “chatbot\_19753546\_GaeteLucero.rkt” con DrRacket, y dar click al botón “Ejecutar”. Una vez realizado este proceso, es posible acceder a las funciones del código.

# CAPÍTULO 3. FUNCIONALIDADES Y MODOS DE USO

Una vez abierto y ejecutado el programa, se pueden utilizar las siguentes funciones para mantener una conversación con el chatbot.

## 3.1 beginDialog

La función beginDialog permite iniciar un nuevo diálogo con el chatbot, donde siempre el mensaje iniciales ofrecido por el bot, consultando de alguna forma, el nombre del usuario. Esta función, retorna un log, el cual debe ser utilizado como argumento de la siguiente función. La siguiente figura, muestra un ejemplo de su retorno.

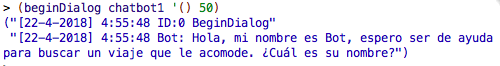


Figura 3 ejemplo de uso beginDialog.

En la figura 2, se utiliza beginDialog con una estructura chatbot predefinida, llamada chatbot1, un log inicial vacio (representado por ‘() ) y una semilla, la cual puede ser cualquier número entero. En este ejemplo, se utilizó el 50.

## 3.2 sendMessage

La función sendMessage permite el envío de un mensaje por parte del usuario al chatbot. Este mensaje parte del supuesto que se conoce el flujo de conversación que hay detrás, por lo que se recomienda tener en cuenta el diagrama presentado en la Figura 2. Esta función recibe los mismos argumentos que la anterior (**beginDialog**), salvo por el mensaje. La siguiente figura, muestra cómo sería una composición de dos mensajes, más la función anterior para iniciar el chat. Nótese cómo el log retornado por una función, es usado como argumento en la siguiente. Cabe destacar que si se ingresa la pregunta “¿Cuándo …”, el bot responderá con una promesa, la cual puede ser forzada a ser respondida mediante un mensaje que contenga las palabras “Respóndeme” o “responderme”. Considerar ortografía y capitalizado.

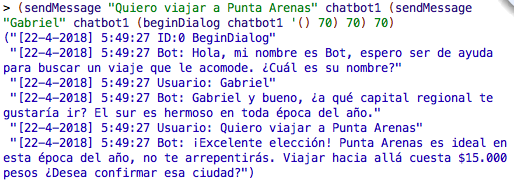


Figura 4 Composición de funciones para producir una conversación

## 3.3 endDialog

Al igual que la función **beginDialog**, se reciben los mismo tres argumentos de entrada. El retorno de esta función es un log con una despedida por parte del bot, y una etiqueta de cierre, como muestra la siguiente figura.

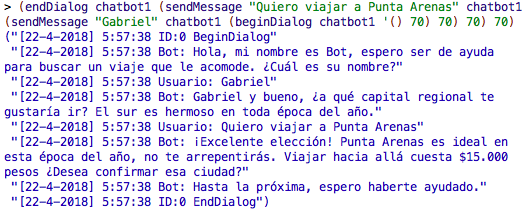


Figura 5 Uso de endDialog

Nótese cómo se le da un cierre adecuado a la conversación anterior mediante esta función.

## 3.4 rate

Esta función recibe como argumentos un chatbot, un puntaje, una función “f” y un log, con los cuales se le permite dar una calificación al bot tras haber finalizado una conversación. Tras esta función, se retorna un chatbot con una nueva calificación, y un nuevo ID. El puntaje del chatbot debe ser un número entero entre un rango de 0 a 5, incluyendo estos valores. El 0 corresponde a la nota en caso de que no se haya podido evaluar, mientras que del 1 al 5 se tiene la calificación más baja a la más alta. Por otro lado, la función f permite que el bot se autoevalúe en base a la última conversación escrita en el log. Esta función se llama “autoRate”. Para mayor claridad, se adjunta la siguiente figura.



Figura 6 Ejemplo función rate

Como se puede ver en la figura 6, la función rate retorna una estructura chatbot, la cual es diferente a chatbot1. Esta función solo puede ser aplicada cuando la conversación ha terminado, de lo contrario, el chatbot no se puede actualizar.

## 3.5 test

Esta función permite ilustrar/simular el desarrollo de una conversación entre un usuario “user” y un chatbot. El argumento user corresponde a una lista de strings, en las que en cada lista se encuentra el mínimo de mensajes para que la conversación tenga un inicio y un término adecuado. Para probar esta función, se brindan tres ejemplos dentro del código fuente, con el nombre user1, user2 y user3

# CAPÍTULO 2. MARCO TEÓRICO

## 2.1 PARADIGMA FUNCIONAL

Como se dijo anteriormente, el paradigma funcional tiene sus bases en el concepto matemático de función. Dado que el presente proyecto debe ser abordado bajo este modelo, se deben tener claras las herramientas proporcionadas, como también el lenguaje utilizado *(Scheme)*. El paradigma funcional brinda la posibilidad de utilizar funciones anónimas, las cuales se caracterizan por no estár ligadas a un identificador; son *“creadas”* en tiempo de ejecución. Otra de las características que brinda el este paradigma, es el concepto de evaluación perezosa, el cual retrasa la carga de un recurso hasta el momento mismo de su utilización. Por último, otro concepto “fuerte” que debe conocerse para la implementación de la solución a este laboratorio, es el concepto de función como ciudadano de primera clase, el cual permite la existencia de funciones de orden superior. Esto brinda la posibilidad de que una función “retorne” a otra, o la reciba como uno de sus argumentos. Estas herramientas, propias del paradigma funcional, son aplicadas en este laboratorio. Se omite la explicación de las restantes, como el concepto de encapsulación, recursión, etc.

## 2.2 TIPO DE DATO ABSTRACTO (TDA)

## Corresponde a un método de abstracción para tipos de datos de manera tal que se puedan definir operaciones sobre éste sin preocuparse de la implementación. Un TDA está compuesto por funciones constructoras encargadas de definir la estructura del nuevo tipo de dato basado en los tipos de datos atómicos (nativos de cada lenguaje), funciones selectoras que permiten obtener información desde una estructura o “variable” del tipo de dato entre otras más, funciones modificadoras que, como su nombre indica, permiten modificar el valor de alguno de los identificadores del TDA. Cabe destacar, que como en el caso de Scheme, no existen variables como tal, las funciones modificadoras consisten en crear copias del elemento original con la modificación pertinente.

# CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA

Se solicita crear un algoritmo en el lenguaje de programación *Scheme* que simule un bot conversacional (chatbot). Para este laboratorio, el tema del chatbot será una venta de pasajes a capitales regionales de Chile. Este debe funcionar en base a una estructura chatbot, un *log* o historial de conversaciones, y una semilla (seed), para utilizar funciones pseudo-aleatorias. El algoritmo debe ser capaz de realizar las siguientes funcionalidades:

1. **Implementación de un TDA:** Haciendo uso de la estructura TDA de 6 capas, implementar abstracciones adecuadas al problema.
2. **beginDialog:** Función que retorna un log modificado, agregando una etiqueta que indica el inicio de una convesación con un chatbot, un *ID* o identificador del chatbot, y un primer mensaje de bienvenida al usuario.
3. **sendMessage:** Función que retorna un log modificado, representando el envío de mensajes por parte del usuario hacia el chatbot y su respectiva respuesta.
4. **endDialog**: Función que retorna un log modificado, agregando una despedida por parte del chatbot, además de una etiqueta que indica el fin de una conversación.
5. **rate:** Función que permite evaluar el desempeño del chatbot en la conversación, a partir de dos calificaciones, una entregada por el usuario, y otra calificación dada por el mismo programa. Esta función retorna un chatbot con un registro actualizado de sus calificaciones.
6. **test:** Función que permite simular una conversación entre un usuario y un chatbot. Retorna un log actualizado con el resultado de la conversación simulada.

Estas funciones tienen un formato definido para su implementación y desarrollo, trabajan con estructuras específicas, por lo mismo, se han definido funciones auxiliares para trabajar con los formatos requeridos.

## 3.1 ANÁLISIS

El principal problema que se presenta en este laboratorio es la de definir funciones y estructuras con las que se pueda trabajar, dado que no se puede modificar la firma de las funciones requeridas, la implementación de cada uno de estos componentes es escencial para garantizar un correcto funcionamiento del chatbot.

### 

### 

# CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN

Como se mencionó en el capítulo anterior, el problema principal recae en la representación de las estructuras a utilizar, tanto del *log* como de los mensajes en sí, por la complejidad que le agrega o le quita a las funciones con las que trabaja.

Así, definiendo una implementación en *Scheme* para los mensajes, sería una lista de tres elementos, donde el primero será una estructura de tipo fecha, el segundo elemento será un string que representará al emisor del mensaje, mientras que el último elemento de la lista corresponde al contenido del mensaje. A continuación, se presenta un ejemplo de la representación en *Scheme* de un mensaje.

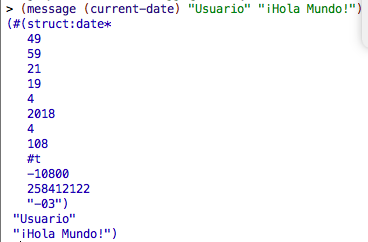


Figura 7 Representación de un mensaje

De esta forma, al momento de crear una representación para el log, se puede acceder a toda la información de los mensajes a través de los selectores para este TDA, logrando que las funciones que deben retornar un log actualizado (**beginDialog, sendMessage, endDialog**) tengan una manera directa de trabajar con este. Por otro lado, para lograr interpretar los mensajes que entrega el usuario, con el fin de entregar una respuesta que se acomode al flujo de la conversación, se trata al contenido del mensaje del usuario como una lista de strings; luego estos son intersectados con una series de palabras para interpretar el mensaje. Una vez interpretado el mensaje, se determina la respuesta que debe entregar el chatbot.

Por último, para la función **rate**, específicamente para la autoevaluación, se utiliza un criterio basado en el largo de la última conversación (sólo se puede evaluar la última conversación finalizada). El criterio utilizado para evaluar, se basa en que si la conversación es de un largo mínimo para vender un pasaje, la evaluación es la máxima. Por otro lado, si la conversación es extremadamente larga para vender un pasaje, la calificación es la mínima. Por último, si la conversación es lo suficientemente corta como para lograr vender un pasaje, la evaluación no se puede determinar.

Teniendo todas estas funciones, la función **test** ya puede ser implementada. La siguiente figura, muestra el log entregado por la función test.

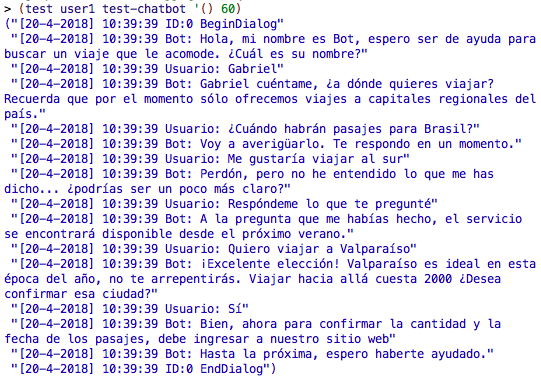


Figura 8 Ejemplo de conversación simulada a través de la función test

En la figura anterior, se tiene un ejemplo de cómo es la representación del log en una conversación con un chatbot. Por otro lado, se puede ver la aplicación del concepto *evaluación perezosa*, al momento de preguntarle por un viaje hacia Brasil.

# CAPÍTULO 5. RESULTADOS

A pesar de las llamadas recursivas, y tomando como referencia el IDE DrRacket, el uso de memoria se mantiene en un nivel bajo. Por otro lado, el tiempo de ejecución de las funciones requeridas es aceptable, donde no existe un tiempo de respuesta notable para el usuario, siendo la impresión por pantalla del log lo que más tiempo toma.

Se ha hecho uso de recursión, tanto NATURAL como de COLA, siendo esta última la más utilizada a lo largo del programa debido al mejor rendimiento de esta. La recursividad NATURAL sólo fue utilizada por cumplir requerimientos exigidos.

Para utilizar cualquier función del programa, se requiere crear una estructura chatbot, la cual, para efectos de evitar su creación en las funciones, se ha asociado una al identificador *chatbot1*.

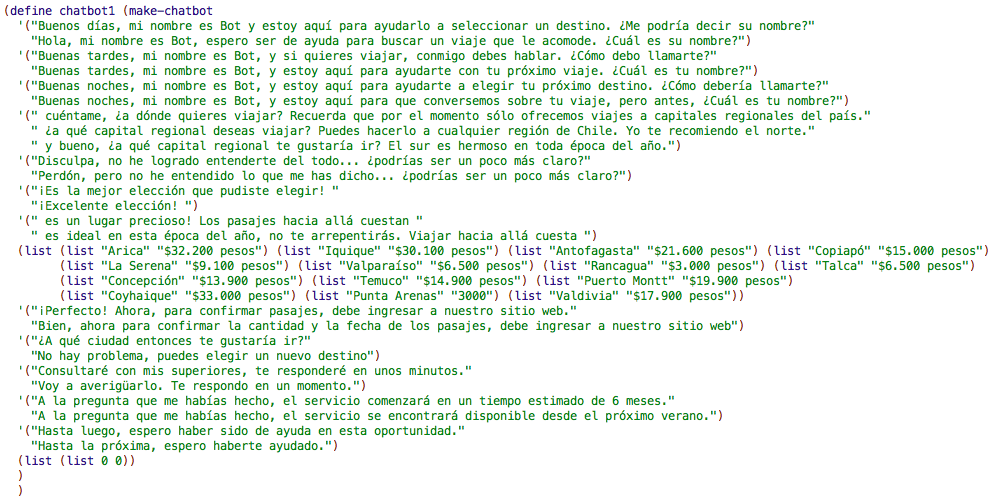


Figura 9 Estructura chatbot1

Por otro lado, también se necesita un log, el cual sólo es una lista inicialmente vacía (o el retorno de las funciones **beginDialog**, **sendMessage** y **endDialog**). La siguiente figura muestra un ejemplo de llamado a las funciones.

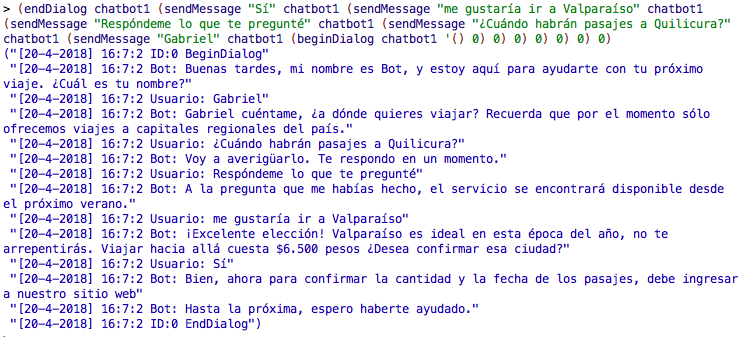


Figura 10 Ejemplo de composición de funciones para generar un chatbot

Tras la finalización de cada función existe memoria utilizada por DrRacket, por lo que se recomienda liberar esta memoria tras cada uso.

Por último, tras evaluar el rendimiento y el funcionamiento del programa, se puede afirmar que Scheme no es un buen lenguaje para la realización de este proyecto, dado que realizar operaciones sobre el log no son sencillas.

# CAPÍTULO 6. CONCLUSIONES

El paradigma funcional permite expresar la solución al laboratorio mediante un lenguaje expresivo y matemáticamente elegante, además de evitar el concepto de cambios de estado, por lo que la modificiación de valores no es permitida. Y esto último es una de las armas de doble filo del paradigma funcional. Por un lado, la resolución del laboratorio se hace complicada sin el uso de variables, dado que no se puede “actualizar” el historial de mensajes en si. Sin embargo, el paradigma funcional sí entrega las herramientas para lograr esto, como puede ser la composición de funciones (estrategia utilizada, ver Figura 4). Esto permite predecir el comportamiento del programa de manera más fácil.

Por otro lado, por la misma razón no existe el concepto de “iteración” como tal. Esto permite que en particular, el paradigma funcional permita comprender de mejor manera el concepto de recursividad. A pesar de esto, para problemas en los que se deban procesar mayores cantidades de información, este paradigma podría no ser el más eficiente, dado que se ocupan grandes cantidades de memoria, sobretodo en recursiones de tipo NATURAL.

Concluyendo, puesto que se ha logrado desarrollar un bot conversacional como se ha requerido (usando el paradigma funcional con el lenguaje de programación Scheme), es posible concluir que se ha logrado el objetivo principal del laboratorio.

# CAPÍTULO 7. REFERENCIAS

# Herrero, C. (s.f.). No son mis cookies. Recuperado el 21 de Abril de 2018, de No son mis cookies: http://nosinmiscookies.com/que-es-un-chatbot/