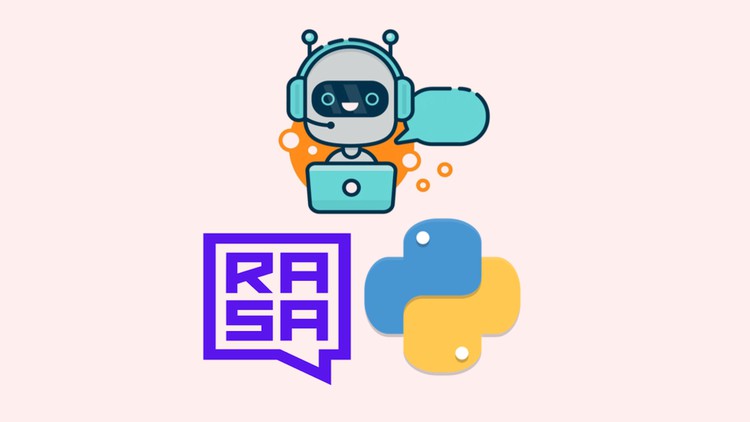
MI PRIMER AGENTE INTELIGENTE:  
FRAMEWORK RASA

Basado en Machine Learning y con interfaz en Telegram



Alumno: Juan Mateo Romero

Materia: Programación Exploratoria

Profesora: Analía Amandi

UNICEN – Tandil

20/10/2023

# **INTRODUCCION**

En los últimos años, la Inteligencia Artificial (IA) y el Machine Learning (ML) están transformando la forma en que los humanos interactúan con la tecnología. Los agentes inteligentes son programas de software que son capaces de comprender y responder de forma autónoma a las necesidades de los usuarios y se están convirtiendo en una parte esencial de la vida cotidiana. Por ejemplo: asistentes virtuales tales como Alexa, Siri, Google Assistant, el ahora descontinuado Cortana de Microsoft [1]; Chats de servicio al cliente; Sistemas de recomendación en plataformas de video/streaming/música, entre otros.

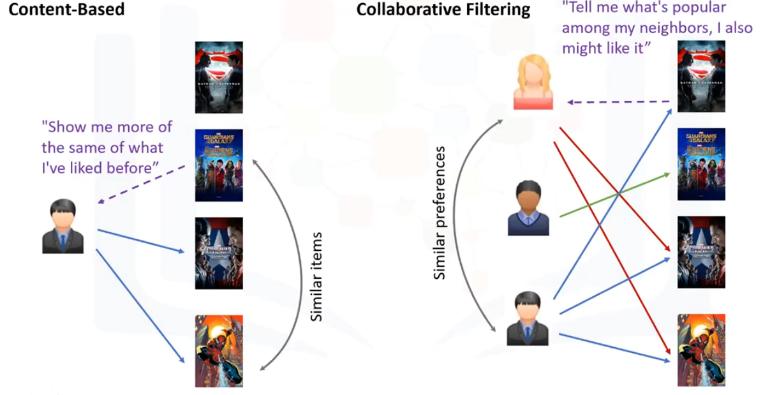
Este informe se centra en la creación y desarrollo de un agente inteligente basado en el framework de código abierto, Rasa. La cual es una poderosa plataforma para la construcción de agentes inteligentes conversacionales, donde se aprovecha el Machine Learning y comprensión de lenguaje natural (NLU) para construir conversaciones y experiencias de usuario más fluidas y personalizadas. Dicho agente, se ha llevado su representación visual a Telegram, una muy conocida app de mensajería, cuya API puede utilizarse de forma gratuita.

Este informe explora en detalle el proceso de desarrollo de un agente inteligente que ha sido creado y basado en un asistente virtual para turismo, el cual ayudará al usuario a encontrar un lugar para poder viajar e irse de vacaciones. Este tiene una base de conocimiento en Prolog. Tambien posee varias funcionalidades básicas, por ejemplo, es posible solicitar un pasaje de avión desde una ciudad a otra. Tambien, si el usuario es un cliente habitual, el asistente puede llegar a conocer sus gustos mediante una Red Neuronal estudiando que viaje podría recomendarle a partir de sus viajes anteriores. Existen tambien otras funcionalidades que luego serán vistas a lo largo del informe.

[1] Algunos asistentes virtuales que se pueden comprar en la actualidad

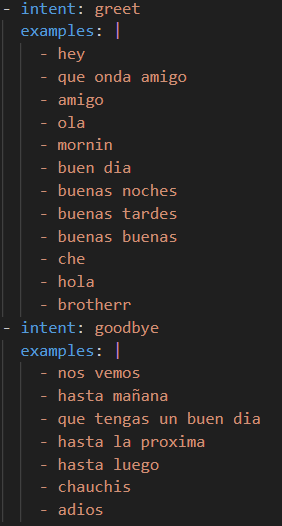


[2] funcionamiento del sistema de recomendación de las plataformas

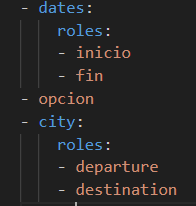


# **DESARROLLO:**

El primer paso en la creación de un agente inteligente radica en comprender su función principal y más básica. En esencia, un agente debe ser capaz de mantener una comunicación fluida y efectiva con los usuarios, ajustando sus respuestas y acciones de acuerdo al contexto en el que se encuentre. En Rasa la comprensión de lenguaje natural y el aprendizaje automático, se logra a través de la implementación de dos componentes esenciales: el NLU y el Core.

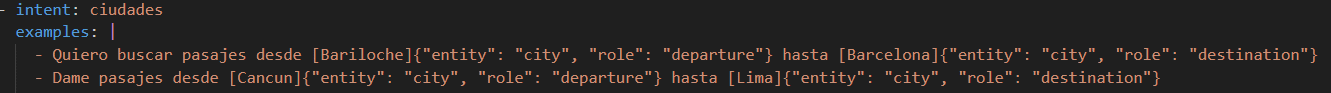
El primer componente se ayuda de los *intents* los cuales son literalmente ‘intentos’de comunicación que va a recibir el Bot, por lo que el está entrenado con Machine Learning para comprenderlos. Cada *intent* quiere dar a entender una idea. Para captar el *intent*, es necesario dar una variedad de ejemplos o distintas maneras de expresar la misma idea. Esto puede ser tanto formal como informal. Estos datos se añaden en un archivo de tipo YAML dentro de la carpeta donde se creó el agente como se muestra en la imagen a la derecha.

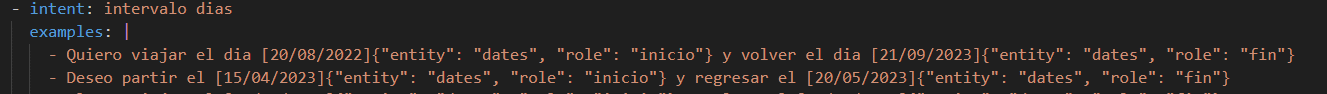
A su vez se puede enriquecer la captura de los intents, sumando entidades. Las entidades son fragmentos de información específica dentro de una expresión que el agente necesita identificar y procesar de manera particular. Por ejemplo, en una solicitud para reservar un vuelo, una entidad es el origen y otra el destino, asi tambien para fecha de ida y la fecha de vuelta. Al etiquetar y extraer estas entidades en los inputs del usuario, el agente puede comprender y actuar en consecuencia. En este caso, se utilizan entidades con roles, tal como se ve a continuación:



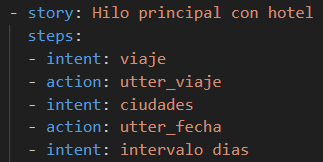
Fragmento de domain.yml para la declaración de las entidades y sus roles.

Un intent donde se capte una entidad con roles debería parecerse a lo siguiente:



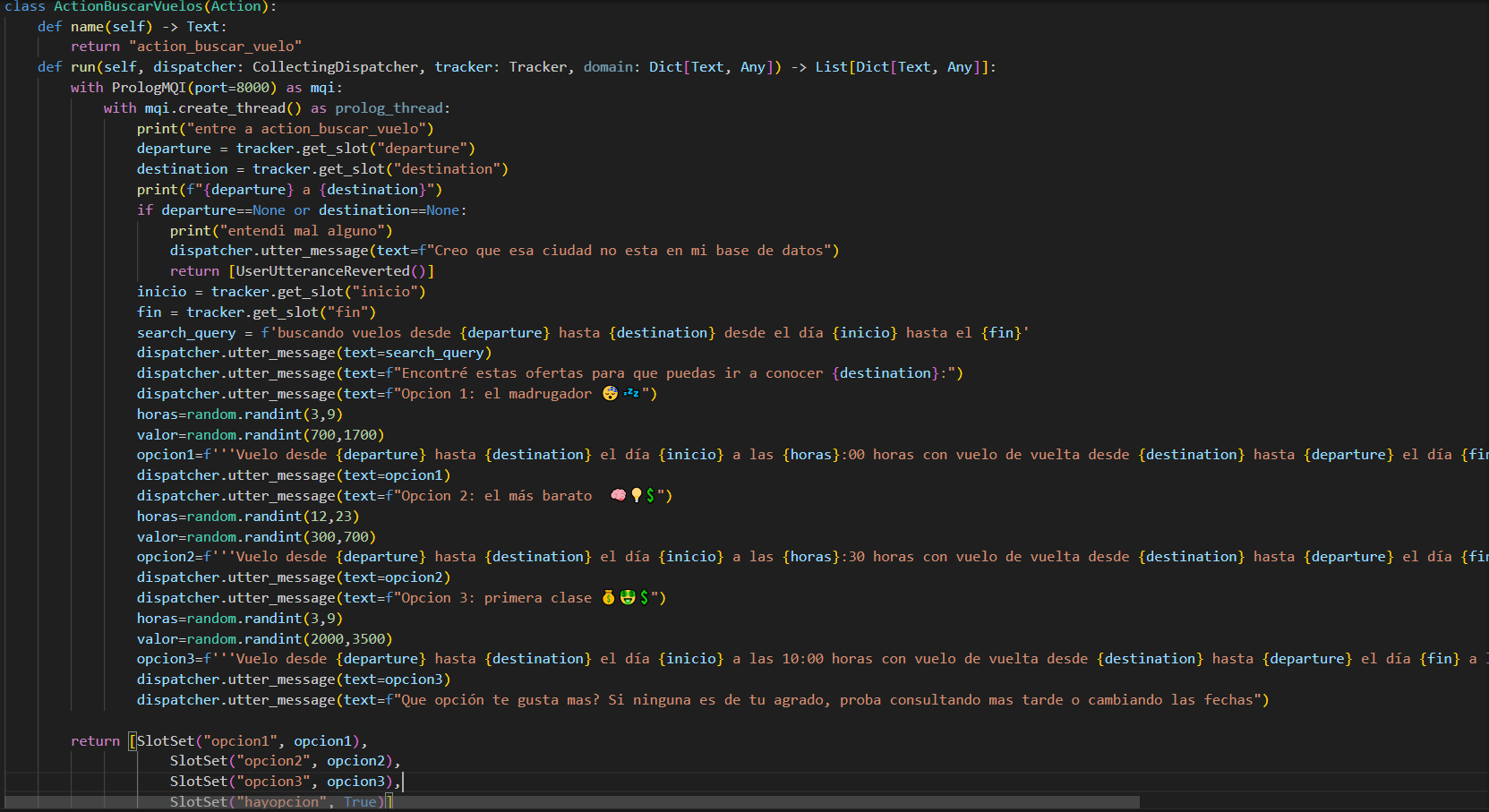


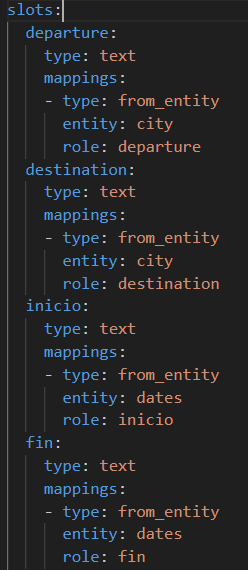
Para que el agente siga un hilo de conversación, es necesario que se le asignen *stories*, donde cada *story* es una serie de intents y acciones en consecuencia que se ejecutarán.



¿Qué es una action?

En Rasa, una action define el comportamiento del agente en respuesta a los intents del usuario. Las acciones representan las respuestas y comportamientos que el agente debe llevar a cabo en un diálogo. Estas pueden variar desde respuestas de texto simples declaradas en el domain hasta tareas más complejas de distintos tipos las cuales se programan en Python en actions.py.

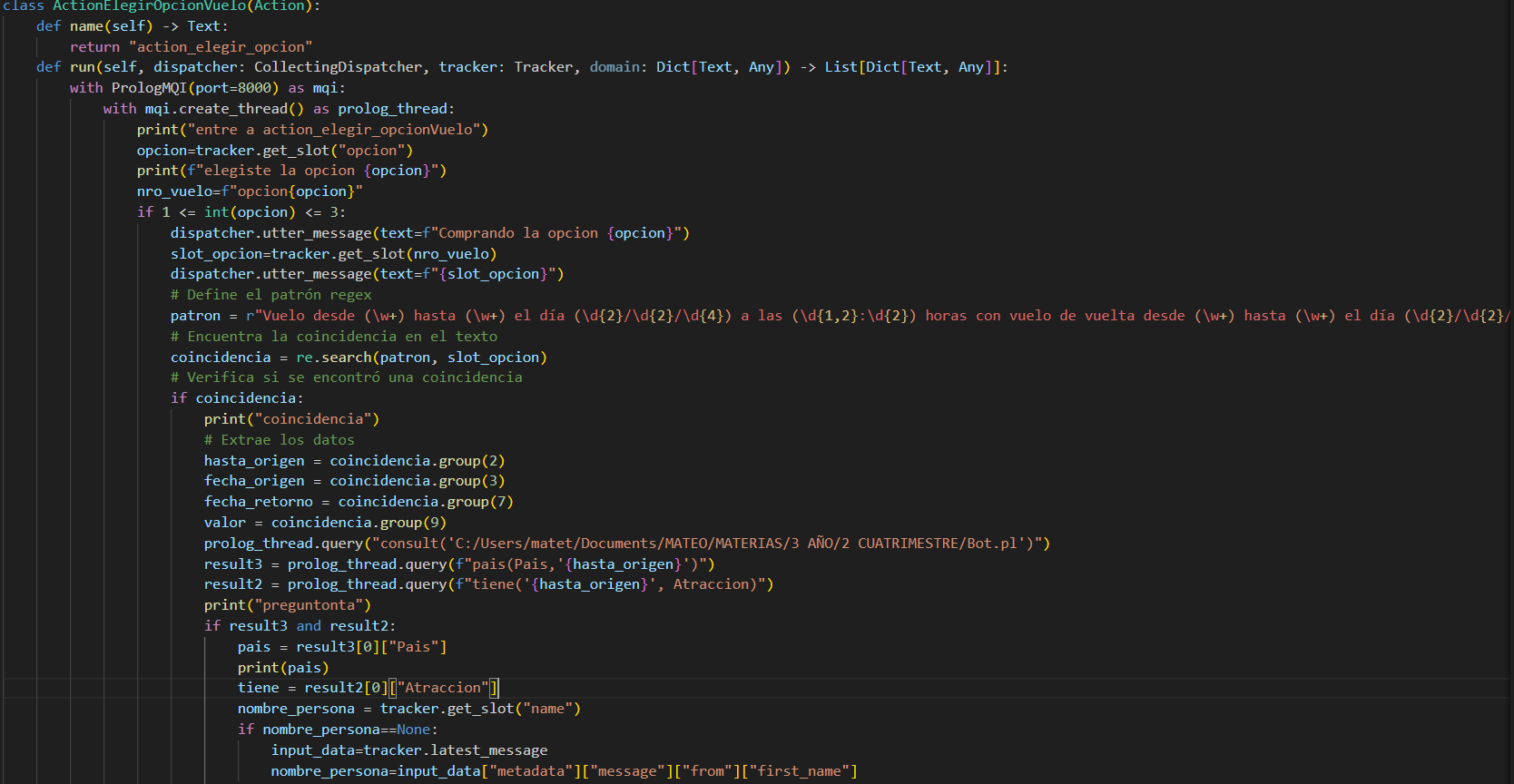


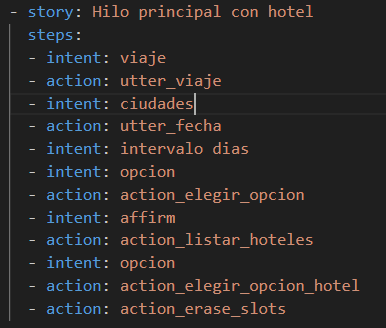
En este caso, la “action\_buscar\_vuelo” recupera el valor de las entidades los cuales se guardan en ubicaciones de memoria llamados “slots” en el momento en que son capturadas.

Cada slot tiene un tipo y un mapeo, el cual puede ser de algún tipo particular, from\_entity, o custom.

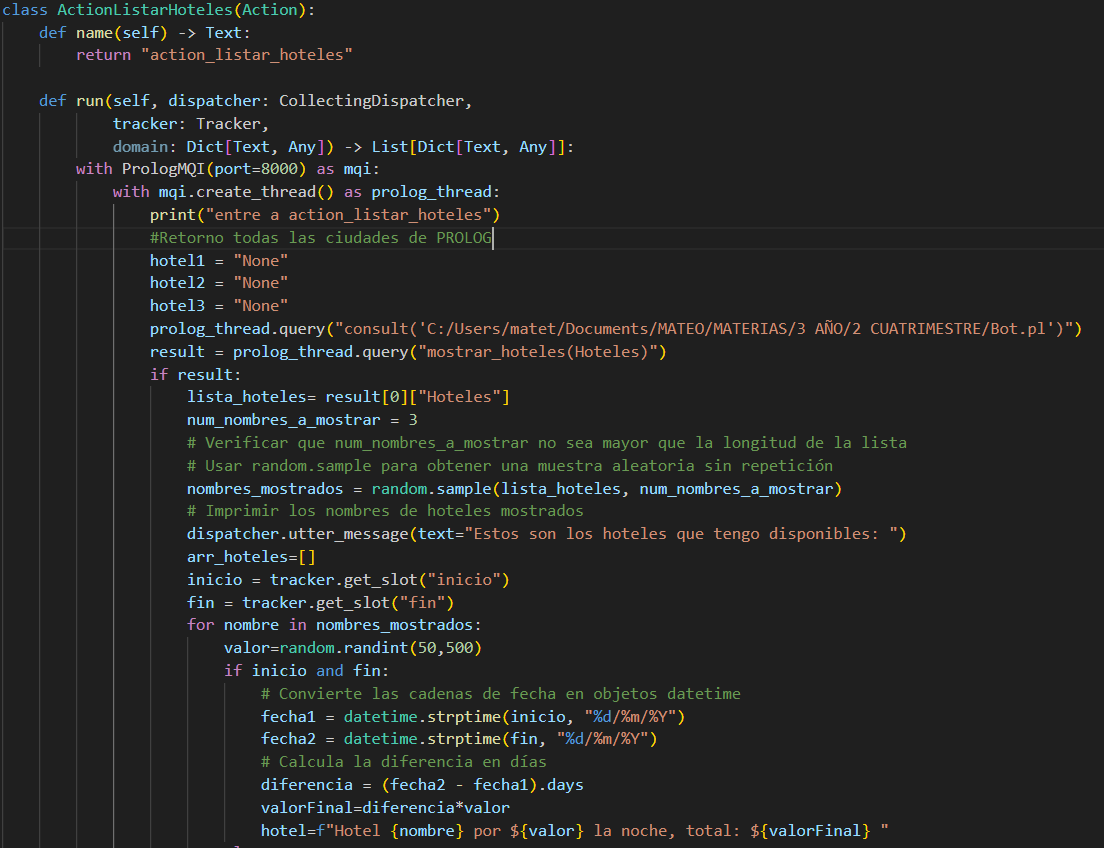
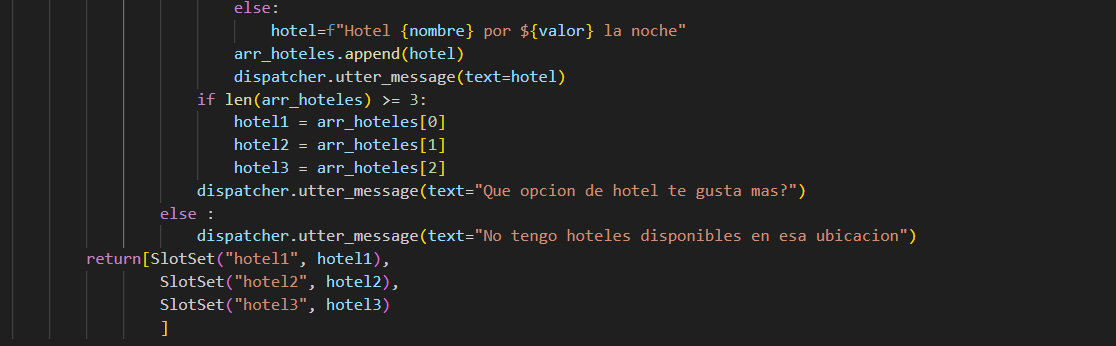
Volviendo al funcionamiento de *“action\_buscar\_vuelo”*, luego de recuperar los valores de los slots, se **simula una búsqueda de vuelos con los datos obtenidos**, mostrando por pantalla tres opciones de vuelo con diferentes precios y horarios y son guardadas en otros slots, para poder seguir el hilo de la conversación y volver a usarlos en cuanto el usuario elija la opcion que mas se adecue a sus gustos.

En la *“action\_elegir\_opcion”* se recupera el dato del slot de la opcion elegida y se utiliza un patrón para poder separar de nuevo los datos por partes. Se consulta a través de swiprolog a una base de datos en Prolog a que pais pertenece el destino y que atracción tiene. Por último, **guarda los datos en un archivo** de extensión “.txt”. Mas adelante se verá por qué.

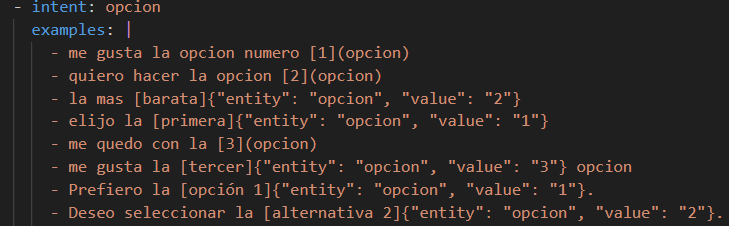


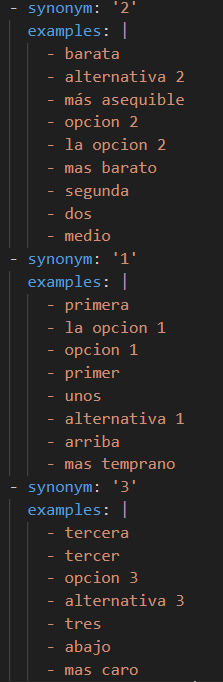


Siguiendo con la *story* principal, luego de haber comprado y guardado los datos de un viaje, se da la opcion al usuario de alojarse en un hotel. En caso de querer, se detecta el intent affirm e ingresa a *“action\_listar\_hoteles”* y se **listan los hoteles disponibles.** Los cuales se encuentran en la base de datos en Prolog. Le pone un precio y le muestra tres opciones al usuario.



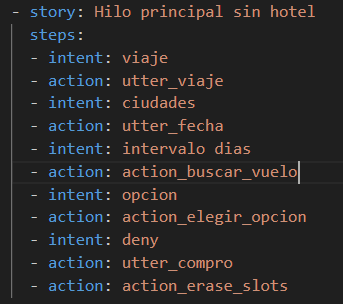
Al finalizar esta action, el usuario debe elegir el hotel que más le guste, de acuerdo a sus preferencias y la *“action\_elegir\_hotel”* **lo marca como comprado.** Acto seguido se **borran los datos almacenados en los slots** con *“action\_erase\_slots”*.

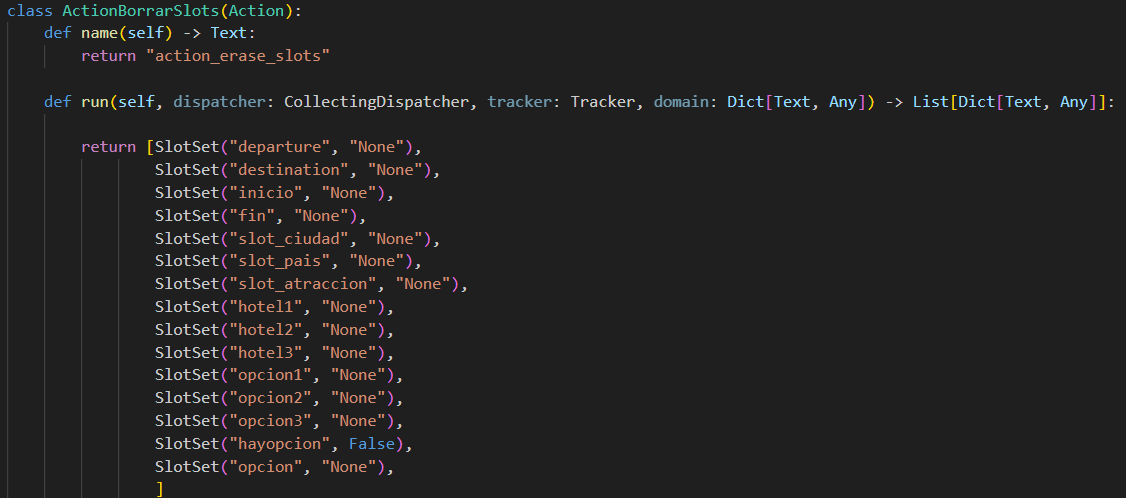




Para la opcion se debió implementar synonyms ya que es posible que el usuario no diga opcion 1/2/3 sino que podría decir: “primera opcion”

Si el usuario hubiese elegido que no quería sacar hotel, el Bot salta a la *story* “Hilo principal sin hotel” yrealiza la action utter\_compro la cual solo dice “gracias por tu compra”. Y luego procede a la *“action\_erase\_slots”* al igual que en la otra story.

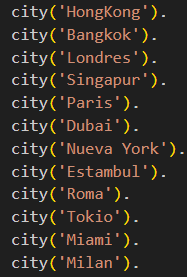


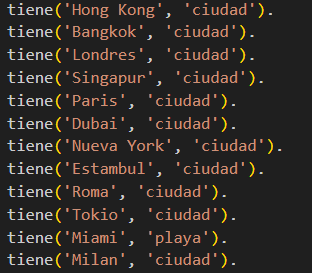
Por otro lado, es posible que el usuario no hubiese querido viajar en ninguno de las opciones dadas, por lo que se le da opcion de cambiar las fechas, y volvería a ingresar a “action\_buscar\_vuelo” con una nueva fecha que ingrese y el mismo origen y destino que ya tenía dado.

Anteriormente se mencionó que se consulta a una base de datos en **Prolog**. Esta es consultada desde las actions en Python a través de Swiprolog y permite acceder a sus datos. De este modo se pueden hacer *querys* según se necesite.

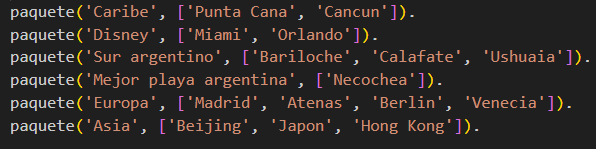
En esta base se puede encontrar datos como:

Nombres de ciudades Pais al que pertenece una ciudad:



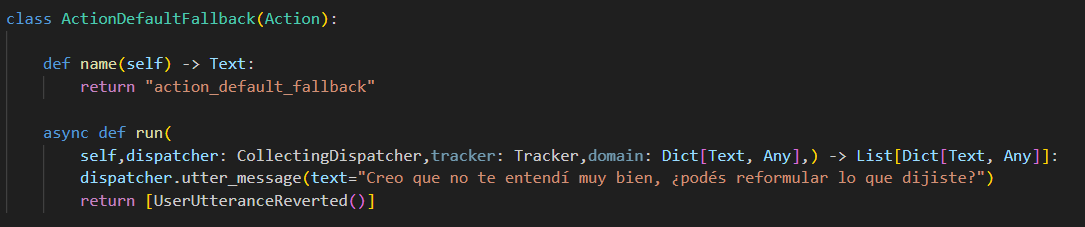
La atracción que tenga un destino: Lista de hoteles:

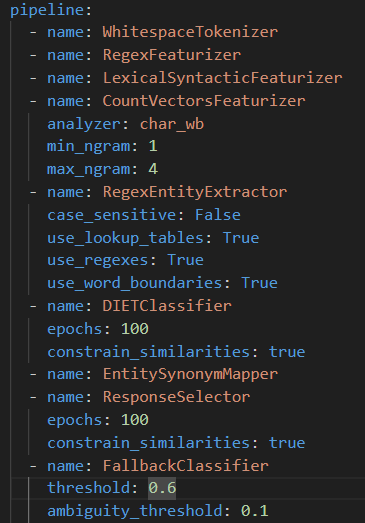
Los paquetes:



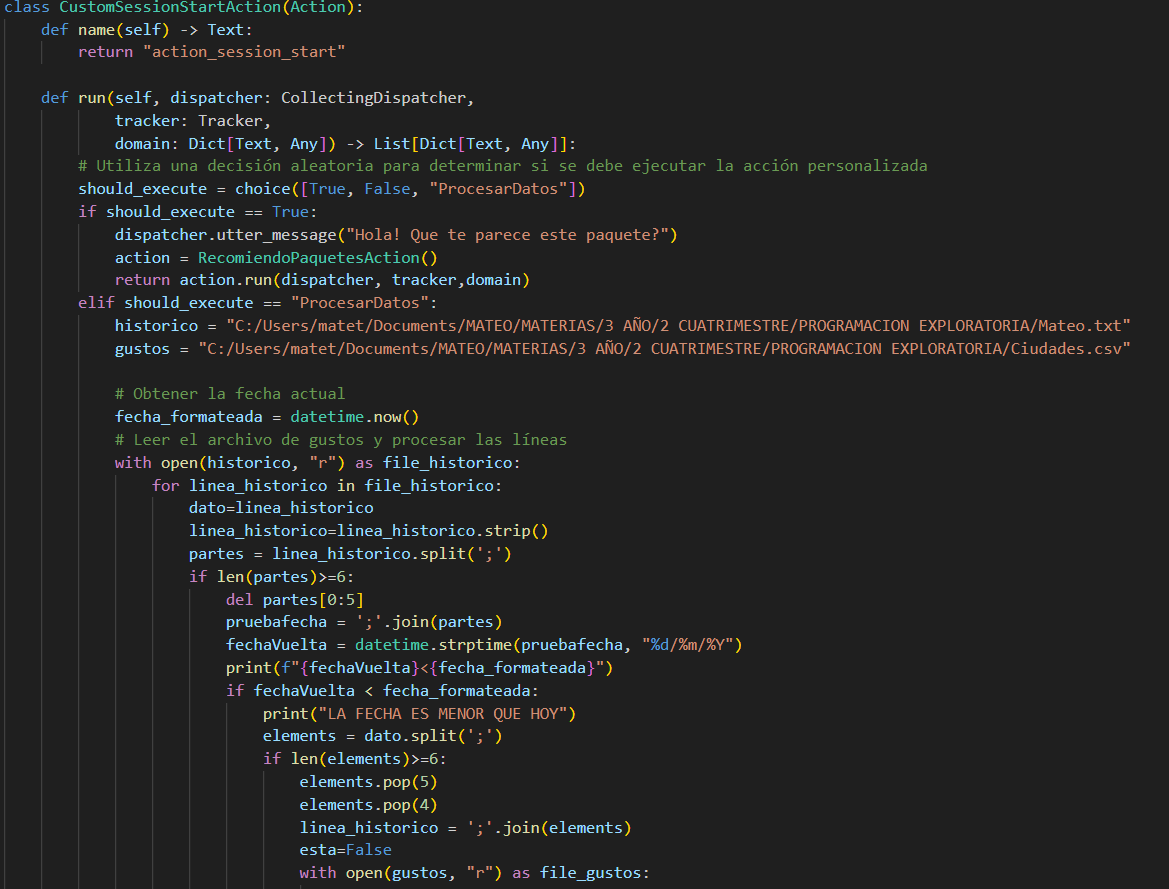
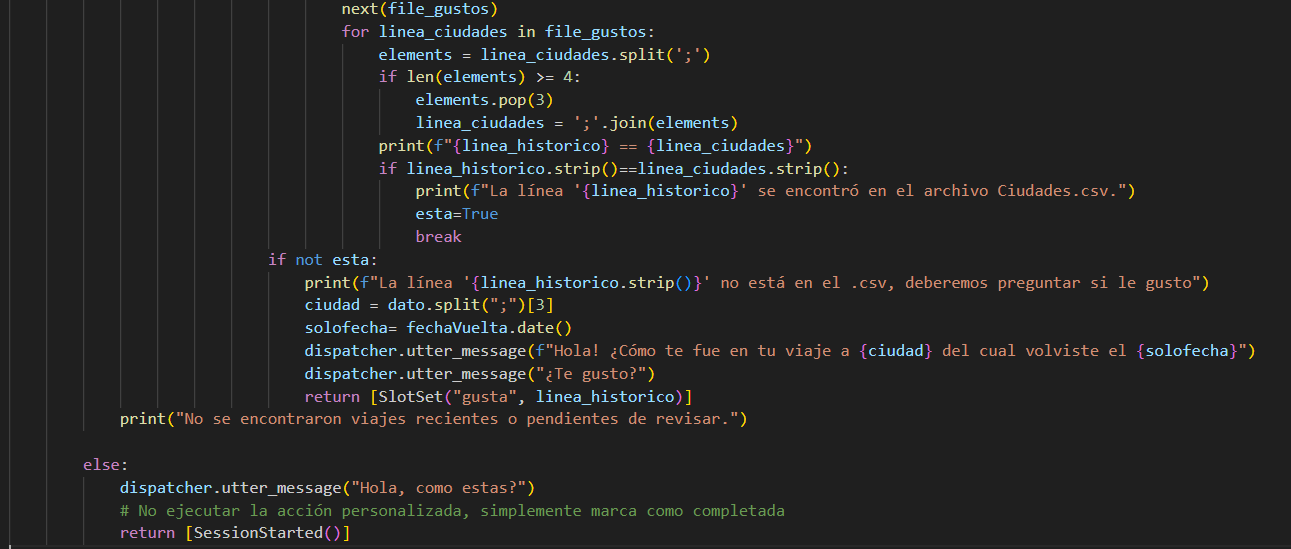
Cada parte de la base de datos en Prolog se consulta con su función correspondiente.

En caso de que (en cualquier momento del programa) el usuario ingresase algo por teclado y el Bot no es capaz de identificar un intent con una certeza mayor que 60%, se llama a la action *"action\_default\_fallback"*, la cual le **dice al usuario que lo que dijo no lo entendió** y retorna UteranceReverted la cual simula que el ultimo intent no fue captado. De este modo el usuario deberá reescribir lo dicho anteriormente o puede decir otra cosa. Esta action ejecutada, ya existe en Rasa por defecto, pero es overrideada para que ademas de que el usuario pueda volver a escribir, le informe antes que no se entendió lo que se quiso decir.

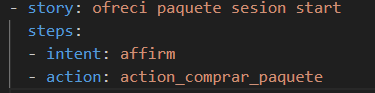


Fragmento de config.yml donde se ve el FallbackClassifier

Es muy importante resaltar el rol que tiene la *“action\_session\_start”* en el programa. Esta tambien es una action por default de Rasa, la cual solo inicia la conversación. En este caso tiene el poder de que al iniciar la conversación se hagan una de tres cosas (mediante un random): solo saludar; recomendar un paquete de viaje al usuario; preguntarle al usuario como le fué en un viaje anterior. Debajo se encuentra su implementación:

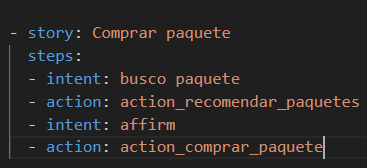


En el segundo caso (should\_execute=true), el Bot consulta a Prolog el cual le retorna todos los paquetes y **le recomienda un paquete al azar**, preguntándole al usuario si este lo quiere comprar o no. Son paquetes en promoción que no tienen en cuenta las preferencias del usuario para ser recomendado. Y se sigue la siguiente *story*

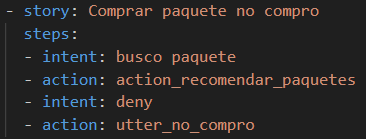


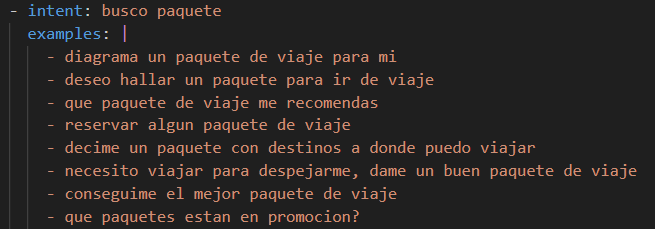
En caso de que al usuario no le gustase o interesase del todo el paquete, puede negarse y el Bot le dirá que otra cosa se le ofrece.

De todos modos, si no se ofrece un paquete al inicio de la conversación, el usuario tiene la posibilidad de pedirle uno al Bot, el cual hará la misma lógica que en la “action\_session\_start” mediante la story “comprar paquete”. En caso de gustarle, se almacenan los datos del paquete en el archivo.txt al igual que con los vuelos, a través de *“action\_comprar\_paquete”*



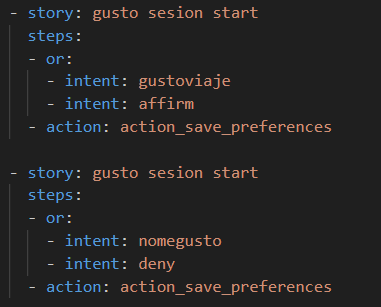
El usuario tambien es capaz de negarse a comprarlo si no le gustase.

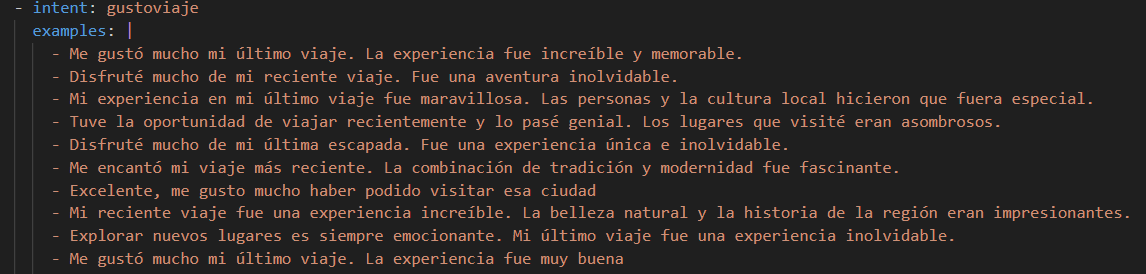


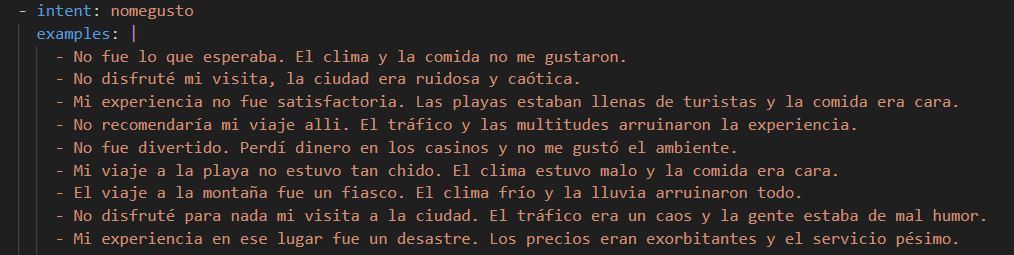


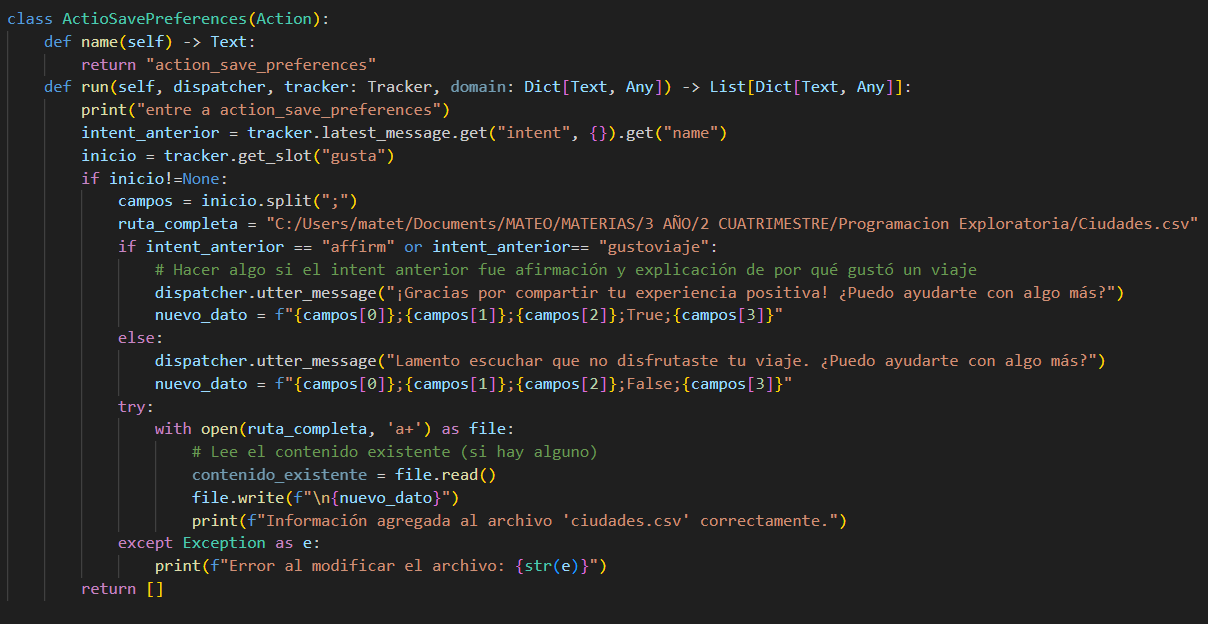
Utter\_no\_compro le dice al usuario que “no hay problema” y si se le ofrece algo más.

En caso que en la session start, el should\_execute= “ProcesarDatos” se chequea que los datos almacenados sobre el histórico de viajes comprados estén todos cargados en otro archivo del usuario de tipo .csv en el cual se sabe si al usuario le gusto o no le gusto su viaje. En caso de que una tupla Precio;Pais;Tiene;Ciudad;FechaIda;FechaVuelta no se encuentre en este archivo y se sepa que el viaje esta terminado, **se le solicita al usuario que dé su opinión acerca del mismo.**   
No sin antes guardar en un slot “gusta” la tupla de datos del viaje del que se consulta, sin las fechas para simplificar la siguiente action.

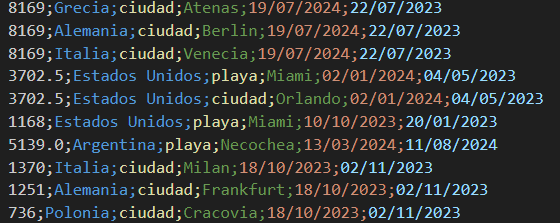
De este modo, existes dos stories alternativas a si al usuario le gusto o no le gusto el viaje. Cargando el dato en el csv desde la action *“action\_save\_preferences”.*



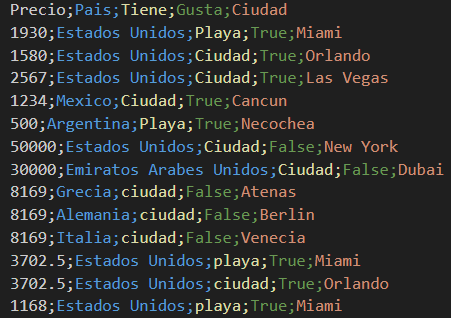




Esta action, agarra los datos guardados en el slot “gusta” anteriormente en la sesión\_start y dependiendo de por qué intent entró, **agrega una columna True o False correspondiente a si al usuario le gustó o no le gustó su viaje realizado.** Y guarda la tupla dentro del archivo .csv en el cual ya se encuentran todos los viajes realizados y finalizados que se sabe si al usuario le gustaron o no.



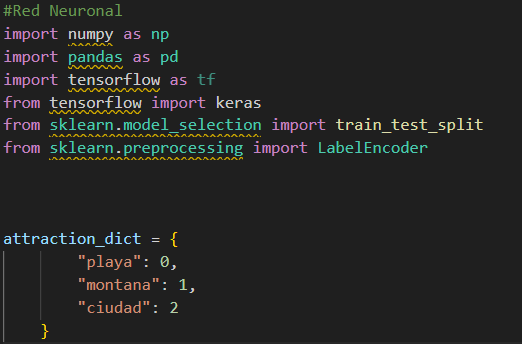
Ejemplo de datos en el archivo .txt donde se ven las columnas: precio, pais, atracción, ciudad, fechaIda, fechaVuelta.



Ejemplo de los datos almacenados en el archivo .csv que contiene las tuplas precio, pais, atracción, gusta, ciudad.

La pregunta que surge a partir de lo anterior probablemente sea, ¿con que fin? Esto se debe a que el Bot posee, como se mencionó anteriormente, una Red Neuronal. Para que esta pueda ser entrenada y funcione correctamente, debe basarse en una fuente solida de datos (normalmente suele ser del orden de miles o millones). En este caso, la fuente de entrenamiento es el archivo .csv.

De los datos almacenados en el mismo, solo se utilizan los datos “precio”, “pais” y “atracción” como datos de entrenamiento y “gusta” es el target u objetivo. Mientras que “ciudad” no entra debido a que puede llegar a dar confusiones ya que dos ciudades podrían tener el mismo nombre en distintos países.



Para esto, se importan las librerías correspondientes y fue necesario la utilización de dos diccionarios: uno con todos los países almacenados en Prolog y un valor numérico; otro con las atracciones y un valor numérico.

Es de gran importancia el uso de diccionarios ya que de este modo se quita la redundancia para datos a la hora de evaluar una tupla. De este modo, por ejemplo, el valor de “China” siempre se corresponderá con 0.

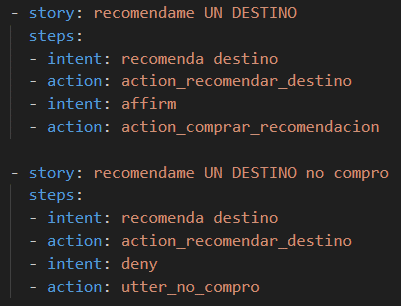


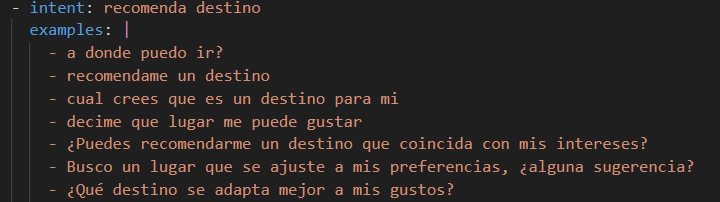
Aquí se muestra la implementación completa de la Red Neuronal, esta fue almacenada en una función para que pueda ser llamada hasta encontrar una certeza mayor a 0.6, como se ve en el while true de la imagen. Esta medida fue necesaria debido a la poca cantidad de datos que se utilizaron en el .csv. Esto se hace al iniciar el Bot. En la realidad esto no se puede hacer debido a lo costoso que es volver a reentrenar una Red Neuronal con miles o millones de datos.

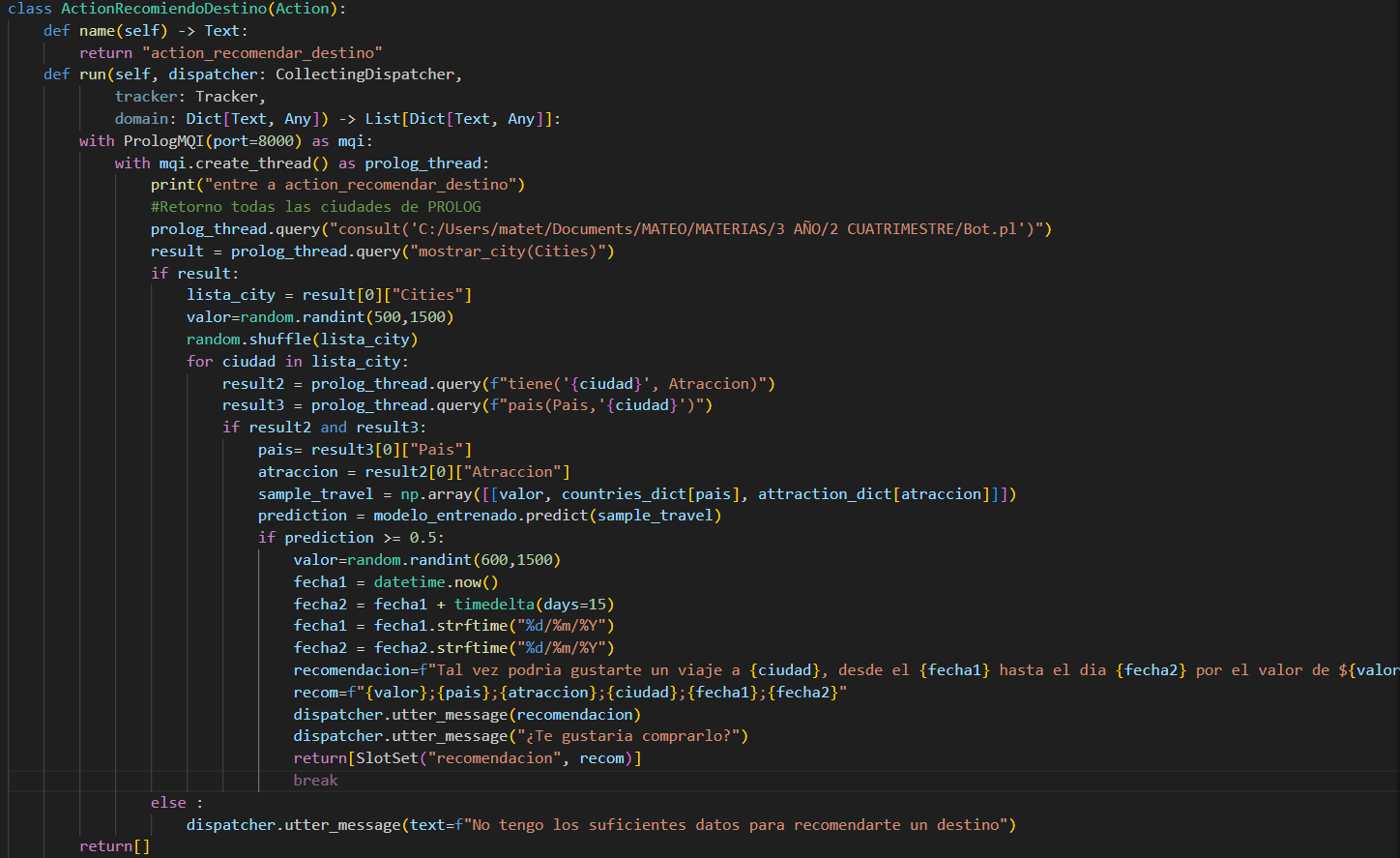
Se puede ver claramente como se utilizan los datos de los diccionarios para usar como entrenamiento. Tambien se crean las capas las cuales la primera siempre son los inputs, luego estan lo que se llaman “capas ocultas” las cuales son las que hacen todo el trabajo, en este caso se asignaron dos capas con 16 y 8 neuronas respectivamente. Por último, una única neurona para retornar el resultado de la predicción.

Luego de haber entrenado el modelo, se evalúa y retorna el modelo (es decir la Red Neuronal entrenada) y accuracy.

Una vez creada la Red Neuronal al inicializarse el Bot, el usuario es capaz de hacer otra de las grandes funcionalidades que el agente posee, pedirle que le recomiende un destino, puesto que no sabe a donde viajar. El mismo le responderá siguiendo las siguientes stories:





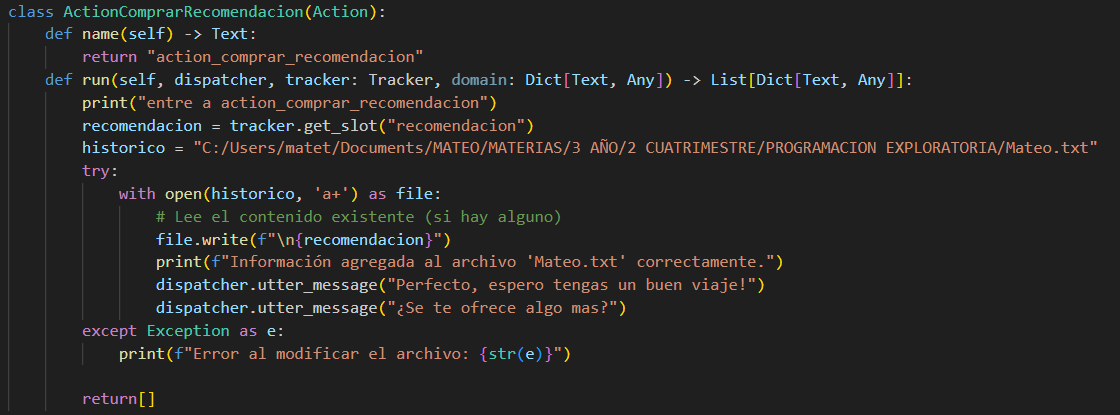


Una vez captado el intent, entra a “action\_recomendar\_destino” donde se hacen varias consultas a la base de conocimiento en Prolog tales como:

* Obtener el listado de todas las ciudades disponibles.
* Para cada ciudad obtener el pais al que pertenece.
* Para cada ciudad obtener la atracción que tiene.

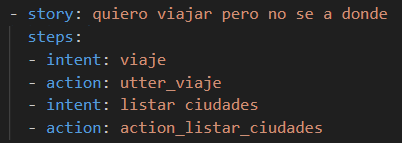
Una vez obtenidos estos datos, se genera una tupla precio;pais;atracción los cuales son los datos precisos con los que se deben consultar a la Red Neuronal a la hora de hacer un predict. Esta es una función propia de la misma y arroja 1 o 0, lo cual significa “Me gusta” o “No me gusta” un viaje. De este modo, el Bot es capaz de ver si al usuario podría gustarle un viaje y en ese caso se lo propone. Si el usuario no le convenciera la idea, puede negarse y el Bot salta a la story “recomendame DESTINO no compro” y le pregunta al usuario que otra cosa quiere.

Cuando el Asistente acierta con la predicción, y el usuario quiere el viaje, le expresa que lo quiere comprar y se accede a *“action\_comprar\_recomendacion”*. Esta guarda los datos en el archivo .txt donde se almacenan todos los viajes comprados por el usuario.

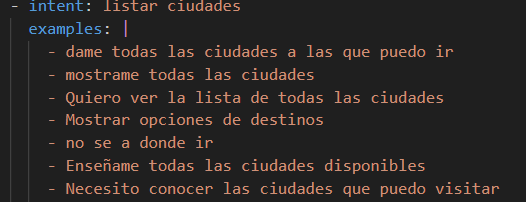


Además de las funcionalidades principales presentadas anteriormente, el asistente posee otras tantas que ayudan un poco al usuario a orientarlo en caso de que no sepa a donde quiere viajar. Entre ellas se puede encontrar:

1. **Listar todas las ciudades disponibles.**



En esta story, se demuestra la intención de querer viajar a algún lado, sin un destino. El Bot indicará que se le ingrese desde donde y a que lugar se quiere viajar, hasta ahora comienza igual que la story “Hilo principal con hotel” explicada anteriormente. Pero, ¿Qué pasa si el usuario no sabe a dónde ir? Este puede pedir que se le listen todas las ciudades a las que puede ir, lo cual se hace en *“action\_listar\_ciudades”.*

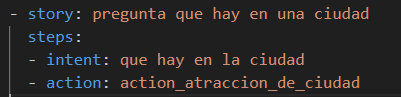
**

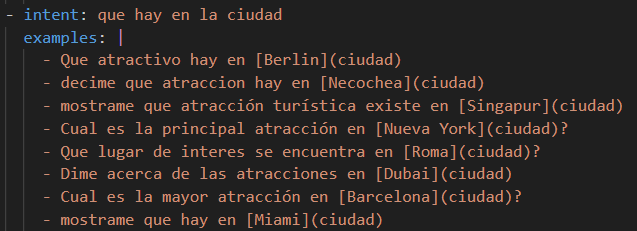


En esta action simplemente se consulta a la base de datos almacenada en Prolog. El cual retorna la lista completa de las ciudades y luego son mostradas al usuario.

1. **Ver qué es lo que tiene de interés una ciudad.**

Si al usuario le llamase la atención el nombre de una ciudad y quisiese averiguar qué se va a encontrar en ella, puede consultarle que tiene de interés, ingresando al siguiente hilo de conversación:



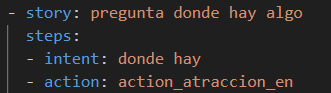


Aquí se llama a la action *“action\_atraccion\_de\_ciudad”* donde se utiliza el dato del slot capturado por el intent “que hay en la ciudad” el cual es el nombre de una ciudad. De este modo se consulta a Prolog acerca esa ciudad y dependiendo su salida, le explica al usuario que tiene de interés.

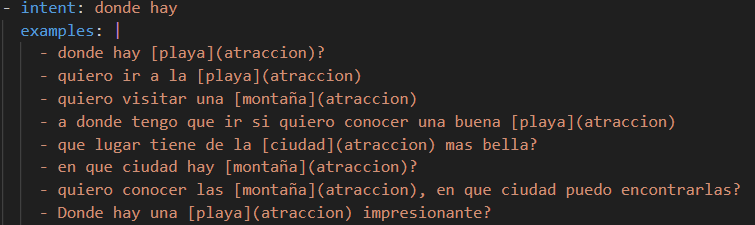


1. **Pedir que se muestre en que ciudad hay una atracción.**

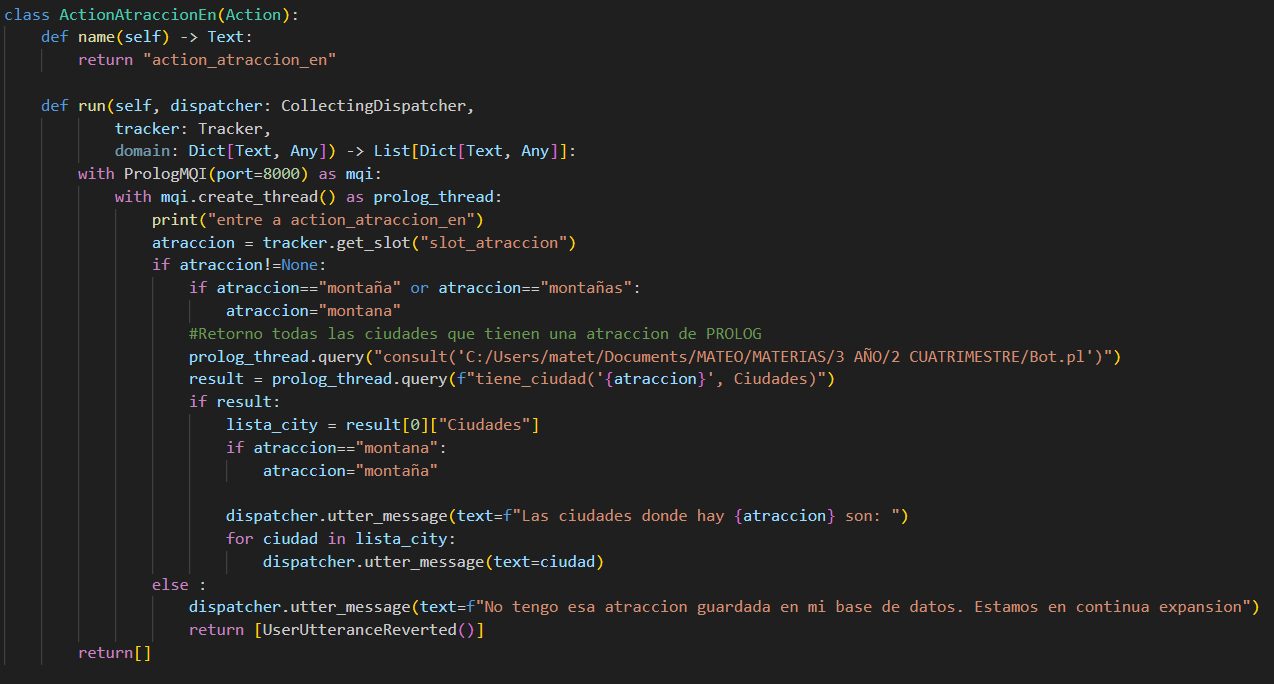
Si el usuario quisiese filtrar los resultados por la atracción que tengan (montaña/playa/ciudad) le puede pedir al asistente que lo haga y le retorne los resultados.



A través del intent “donde hay” el usuario le expresa esa inquietud y el asistente responderá según *“action\_atraccion\_en”*.

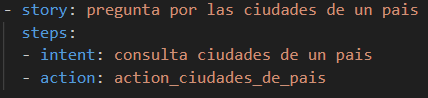


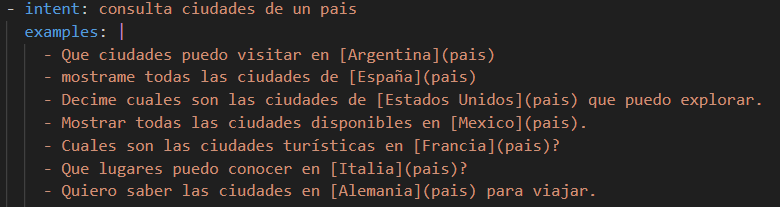
En esta action lo que se hace es obtener el valor del slot almacenado en el slot “slot\_atraccion” captado anteriormente en el intent y consultar a Prolog qué ciudades tienen esa atracción. De este modo se retorna el listado de ciudades que la poseen.



1. **Averiguar que ciudades estan en un pais particular.**

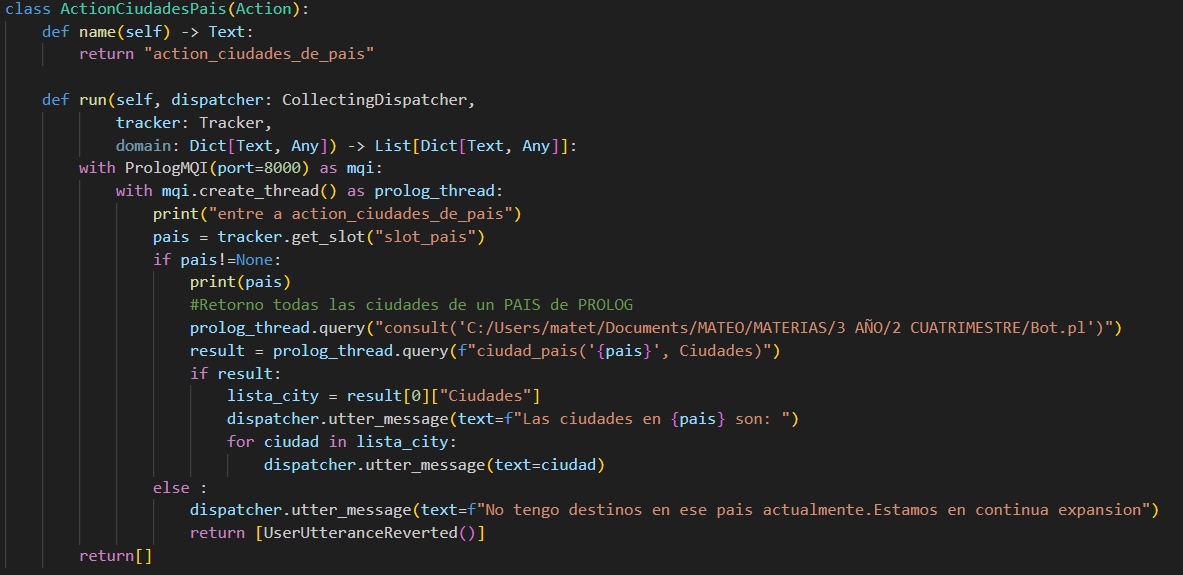
Si al usuario le llamase la atención algún pais en particular, podría pedirle al Bot que filtre la búsqueda mostrando las ciudades de ese pais.





De este modo se almacena el nombre del pais en un slot y se accede a la action *“action\_ciudades\_de\_pais”.*

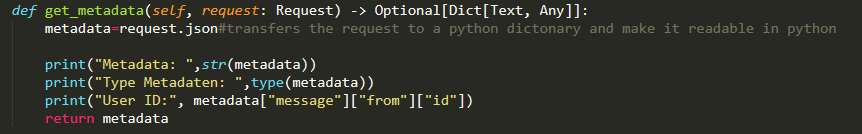
Aquí se consulta a partir del valor del slot “slot\_pais” el listado de ciudades que se encuentran dentro del territorio de ese pais.



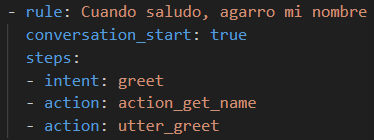
Un aspecto distintivo de este asistente es que es capaz de obtener el nombre del usuario de manera implícita, sin necesidad de que el usuario lo proporcione explícitamente. Esto se logra gracias a la integración con la API de Telegram, lo cual mejora la personalización de las interacciones. Tambien se proporciona acceso a información del perfil del usuario, lo que incluye datos como el nombre, la foto de perfil y otros detalles, pero no son utilizados. Cuando un usuario inicia una conversación con el asistente a través de Telegram, el asistente puede acceder a esta información y utilizarla para dirigirse al usuario por su nombre, lo que agrega un toque personal y amigable a la conversación.

Este enfoque es valioso, porque permite que el asistente se comunique de manera más natural y personalizada. Además, elimina la necesidad de que el usuario proporcione su nombre manualmente.

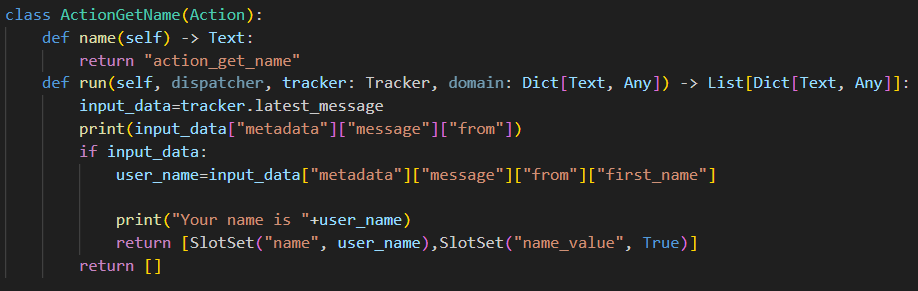
Para lograr esto, fue necesario cambiar el método getMetadata dentro del archivo Channels.py



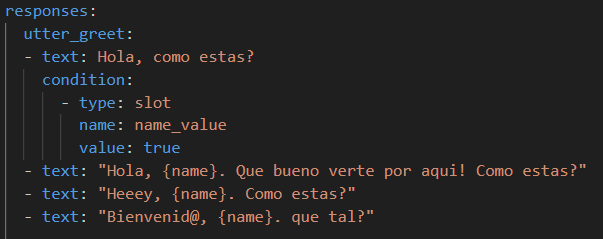
De este modo, es posible que cuando se lo requiera, se obtenga el nombre.



Con la ayuda de esta rule, se llama a la action *“action\_get\_name”*

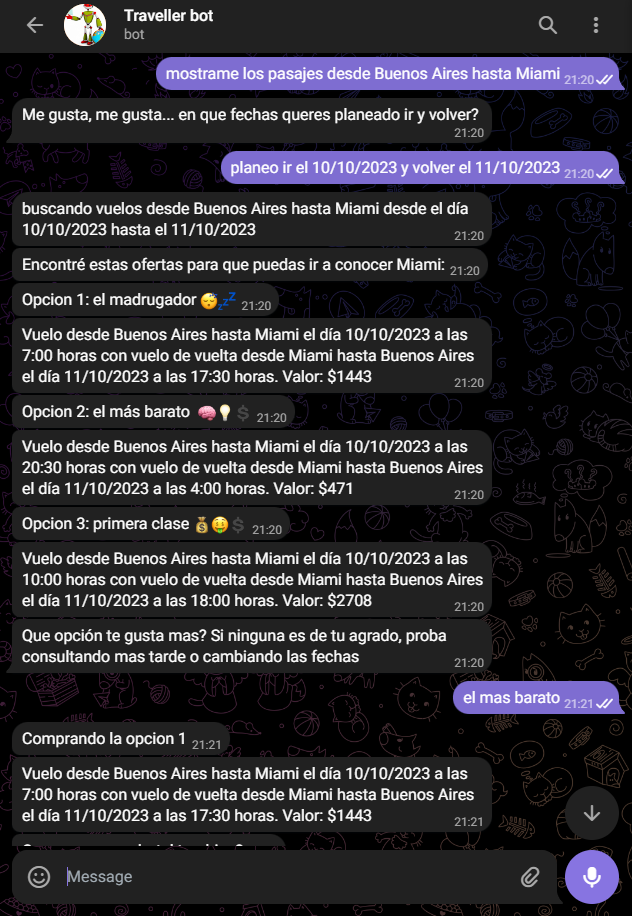


Dentro de la misma, se almacena el nombre del usuario en el slot “name” y se marca en true el slot “name\_value” para poder chequear si el nombre ya fue cargado para luego poder utilizarlo en cualquier momento.

 Por ejemplo, es posible que el Asistente te salude con tu nombre, sin habérselo dicho

Por último, como se ha comentado, se ha utilizado Telegram como interfaz gráfica mediante su integración al Bot. Esto se hace bastante sencillo siguiendo los pasos en la documentación de Rasa añadiendo unas líneas en creedentials y utilizando una consola extra llamada Ngrok la cual nos conecta nuestro Bot con Telegram. Se ha personalizado el perfil del Asistente en la red de mensajería y ahora cuenta con una foto de perfil, un nombre: TravellerBot, y una lista de cosas que puede hacer que se muestra si el chat esta vacío, es decir lo pueden ver nuevos usuarios para saber como se usa o que se le puede decir al nuevo asistente.

Foto de perfil de TravellerBot generada con Dalle-2



Ejemplo de como se ve el Bot funcionando en un chat de Telegram.

# **ASPECTOS A MEJORAR:**

* Aumentar la cantidad de acciones o funciones con computación afectiva:

Se podría guardar el valor de las emociones del usuario por ejemplo cuando se le pregunta como está y sumado a eso, recomendarle que si se siente triste podría irse de viaje pronto; si está estresado podría irse de viaje espiritual a algún lugar en particular, etc.

* Agregar una integración con Google Calendar:

Aquí se podría crear una función que cree un evento para el viaje comprado en las fechas y horarios estipulados. Aunque no se tiene la certeza de que tan factible sea debido a que se necesitaría tener acceso a la cuenta de Google de cada usuario.

* Añadir descripciones a las ciudades:

Es algo costoso para cada ciudad buscar su información y una imagen manualmente para tenerlo disponible almacenado en algún sitio en el momento que se necesite. Aunque podría ayudar al usuario a estar más o menos decidido a comprar un viaje, según corresponda.

# **CONCLUSION:**

En conclusión, Rasa es un sistema con un potencial significativo para el desarrollo de chatbots y asistentes virtuales. A diferencia de algunos chatbots comunes que a veces pueden no proporcionar respuestas precisas, Rasa ofrece una solución más sólida y versátil. Se pueden crear asistentes mejores que cualquiera de los que se pueden ver funcionando en algunos lugares del pais. Por ejemplo, el de Banco Galicia, tiene un menú y aun asi se cuelga y no responde.

Una de las principales ventajas de Rasa es su flexibilidad. Permite a los usuarios personalizar y mejorar sus chatbots de diversas maneras, lo que significa que no están limitados a un chatbot estándar.

El método de aprendizaje basado en proyectos, ofrece una gran oportunidad para comprender en profundidad la creación de asistentes inteligentes y ver en primera persona no solo como se crea un asistente inteligente, sino que les da a los alumnos la posibilidad de ver cómo puede llegar a ser el día a día en el futuro en un trabajo.

En la industria de la Ingeniería en Sistemas, donde la tecnología desempeña un papel central, la adquisición de conocimientos sobre la creación de asistentes inteligentes y el uso de tecnologías de IA es esencial. Estas habilidades proporcionan una base sólida para comprender y enfrentar los desafíos en constante evolución de un mundo impulsado por la tecnología.

Por último, este proyecto demuestra que el futuro de la IA y los asistentes inteligentes es prometedor. También destaca la importancia de prepararse y adquirir una base sólida en esta área para abrir puertas a futuros empleos.

# **BIBLIOGRAFIA:**

* <https://rasa.com/docs/rasa>
* <https://forum.rasa.com/t/where-can-i-access-metadata-from-tracker/46297>
* <https://rasa.com/docs/rasa/connectors/telegram/>