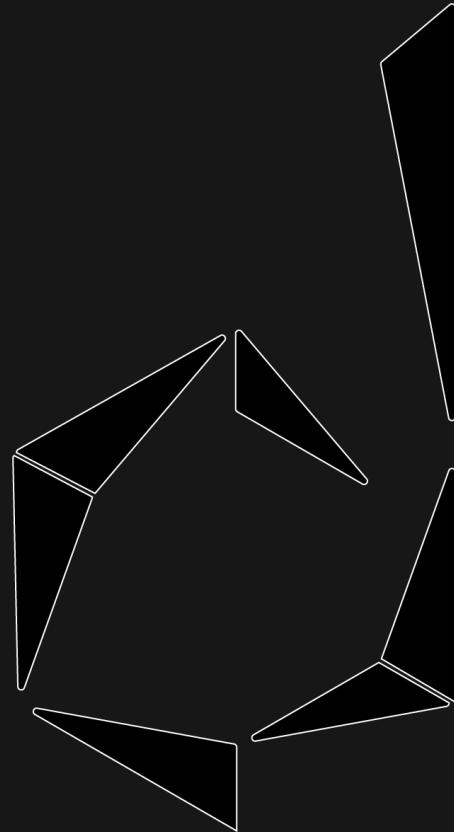


# INGENIERÍA MECATRÓNICA



## DI\_CERO

DIEGO CERVANTES RODRÍGUEZ

INTELIGENCIA ARTIFICIAL

PYTHON 3.9.7

Prompt Engineering,  
ChatGPT, Bard y LangChain  
12/09/2024

## Contenido

<b>Prompt Engineering.....</b>	2
<b>ChatGPT API.....</b>	3
Código Python: API ChatGPT.....	7
<b>Google Bard API.....</b>	11
Código Python: API Google Bard - El código todavía no sirve en México.....	12
<b>LangChain 🦜🔗.....</b>	13
Instalaciones: .....	20
Código Python: LangChain - Modelos y Prompts.....	25
Resultado del Código Python .....	31
Código Python: LangChain - Memoria .....	31
Resultado del Código Python .....	38
Código Python: LangChain - Cadenas.....	39
Resultado del Código Python .....	44
Código Python: LangChain - Índices.....	45
Resultado del Código Python .....	53
Código Python: LangChain - Agentes con Herramientas Pre-hechas y Personalizadas.....	54
Resultado del Código Python .....	64
Referencias.....	65



# Prompt Engineering

Prompt engineering se refiere a la práctica de diseñar o manipular las indicaciones o preguntas que se le presentan a un modelo de lenguaje (LLM), como lo puede ser ChatGPT de Open AI o Bard de Google, con el fin de obtener respuestas más precisas. Esto implica comprender cómo estructurar y formular las instrucciones de manera efectiva para obtener los resultados deseados del modelo. Para ello se utilizan las siguientes técnicas:

- **Zero-Shot Prompting:** Al modelo de lenguaje se le hace una pregunta de forma directa sin proporcionar un ejemplo de la respuesta que se busca obtener. Esta técnica es la menos recomendada ya que puede llevar a formatos de respuestas no deseadas.
  - **Pregunta.**
- **One-Shot Prompting:** Al Large Language Model (LLM) se le hace una pregunta proporcionando un ejemplo de la respuesta que se busca obtener a través de la siguiente sintaxis:
  - **Pregunta. Ejemplo = “Formato de la respuesta que se busca obtener”.**
- **Few-Shot Prompting:** Al modelo de lenguaje se le hace una pregunta proporcionando múltiples ejemplos de la respuesta que se busca obtener a través de la siguiente sintaxis:
  - **Pregunta. Ejemplo 1 = “Formato de la respuesta que se busca obtener”, Ejemplo 2 = “Formato de la respuesta que se busca obtener”, ..., etc.**
- **Role Play:** Un modelo de lenguaje no puede tener sentimientos, opiniones o conciencia, por lo que cuando se busca que haga algo del estilo, se le puede indicar que simule un rol y para ello se utiliza la siguiente sintaxis:
  - **Actúa como un rol que se quiere interpretar y quiero saber tu opinión acerca de tema que se quiere abordar.**
- **Knowledge Integration:** La base de datos de ChatGPT en la fecha actual de 04/09/23 está actualizada solamente hasta septiembre del 2021, por lo que, si se le realiza una pregunta que tenga que ver con un tema reciente, no tendrá conocimiento de ello, a diferencia de Google Bard, que está conectado a internet, por lo que conoce eventos actuales. Debido a esta situación, si se quiere que ChatGPT o Bard den una respuesta acerca de un tema que no conoce, se le puede proporcionar un artículo o texto que le dé un contexto del tema a través de la siguiente sintaxis:
  - **Integra la siguiente información con tu conocimiento en temas que se abordan en el artículo o texto de interés: Información = “Artículo o texto de contexto”.**
- **Chain of Thought Prompting:** Al presentar una **pregunta** al modelo de lenguaje **indicando que la respuesta sea proporcionada paso a paso**, se obtendrá una respuesta mucho más estructurada y detallada, para ello se utiliza la siguiente sintaxis:
  - **Pregunta. Dime opciones cortas y concretas, pensando cada paso, paso a paso.**

Bard y ChatGPT son modelos de lenguaje, no modelos de inteligencia artificial, su función es comprender y sintetizar preguntas hechas con un lenguaje humano natural, a diferencia de los buscadores convencionales como Google o Edge que se les debe proporcionar palabras clave para obtener mejores resultados. ChatGPT utiliza por debajo el modelo GPT-3 o 4 (Generative Pre-Training Transformer), mientras que Bard utiliza el modelo LaMDA (Language Model for Dialogue Applications).

# ChatGPT API

Las siglas de API significan Application Programming Interface (Interfaz de Programación de Aplicaciones), y es básicamente una forma controlada en la que a dos programas se les permite interactuar, intercambiando funciones o datos de programas previamente hechos para que cualquiera los pueda utilizar. Las APIs normalmente no muestran cómo realizan sus funciones, simplemente se pueden utilizar, pero no ver su código fuente.

Para poder utilizar la API de ChatGPT se debe crear una cuenta y luego seguir los pasos descritos a continuación en el siguiente enlace:

<https://platform.openai.com/apps>

The screenshot shows the OpenAI platform interface. At the top, there's a navigation bar with links like Overview, Documentation, API reference, Examples, and Playground. On the right, a user profile menu is open, showing options such as Personal, Manage account, View API keys (which is highlighted), Invite team, Visit ChatGPT, Visit DALL-E, Help, Pricing, Terms & policies, and Log out. Below the navigation, the main content area has a title "Welcome to the OpenAI platform" and a section titled "Start with the basics" with two buttons: "Quickstart tutorial" and "Examples". The left sidebar has sections for ORGANIZATION (Personal, Settings, Usage, Rate limits, Members, Billing) and USER (API keys, which is also highlighted). The main content area under "API keys" displays a message: "You currently do not have any API keys. Create one using the button below to get started." A "Create new secret key" button is visible. The background features a large, abstract geometric shape.

The screenshot shows the OpenAI platform's API keys management interface. On the left, there's a sidebar with sections for Organization (Personal, Settings, Usage, Rate limits, Members, Billing), User (Personal, API keys), and API keys (which is currently selected). The main area is titled 'API keys' and contains instructions about API key security. A modal window titled 'Create new secret key' is centered, prompting the user to enter a name for the new key ('dicer0') and providing options to cancel or create the key.

This screenshot shows the same OpenAI API keys page after a new API key has been successfully created. A green notification bar at the top right indicates that the 'API key copied!'. The modal window now displays a 'Done' button instead of the previous 'Create secret key' button, and the background page shows the newly created key listed in the main table.

Luego para usar la API en programas de Python se debe ejecutar el siguiente comando en la consola CMD de Windows, realizando así la instalación de su librería.

This screenshot shows the OpenAI API reference documentation for Python. The 'API REFERENCE' section is active, and the 'Introduction' page is displayed. It provides instructions for interacting with the API via HTTP requests and lists several Python libraries available for use. A code block at the bottom shows the command 'pip install openai' for installing the official Python bindings.

```

Símbolo del sistema - python  +  -
Microsoft Windows [Versión 10.0.22621.2134]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\diego>pip install openai
Collecting openai
  Downloading openai-0.27.8-py3-none-any.whl (73 kB)
    73.6/73.6 kB 405.9 kB/s eta 0:00:00
Collecting requests<=2.20 (from openai)
  Using cached requests-2.31.0-py3-none-any.whl (62 kB)
Collecting tqdm (from openai)
  Downloading tqdm-4.66.1-py3-none-any.whl (78 kB)
    78.3/78.3 kB 4.3 MB/s eta 0:00:00
Collecting aiohttp (from openai)
  Downloading aiohttp-3.8.5-cp39-cp39-win_amd64.whl (327 kB)
    327.1/327.1 kB 2.0 MB/s eta 0:00:00
Collecting charset-normalizer<4,>=2 (from requests<=2.20->openai)
  Downloading charset_normalizer-3.2.0-cp39-cp39-win_amd64.whl (96 kB)
    96.9/96.9 kB 370.3 kB/s eta 0:00:00
Collecting idna<4,>=2.5 (from requests<=2.20->openai)
  Using cached idna-3.4-py3-none-any.whl (61 kB)
Collecting urllib3<3,>=1.21.1 (from requests<=2.20->openai)
  Downloading urllib3-2.0.4-py3-none-any.whl (123 kB)
    123.9/123.9 kB 519.0 kB/s eta 0:00:00
Collecting certifi>=2017.4.17 (from requests<=2.20->openai)
  Downloading certifi-2023.7.22-py3-none-any.whl (158 kB)
    158.3/158.3 kB 949.3 kB/s eta 0:00:00
Collecting attrs>=17.3.0 (from aiohttp->openai)
  Downloading attrs-23.1.0-py3-none-any.whl (61 kB)
    61.2/61.2 kB 1.1 MB/s eta 0:00:00
Collecting multidict<7.0,>=4.5 (from aiohttp->openai)

```

Cabe mencionar que, al utilizar la API en su modo gratuito, solo se le podrán realizar 100 llamadas por día, si se excede ese límite, se recibirá el error `RateLimitError` al intentar ejecutar el programa de Python. Pero **si se decide comprar el servicio de la API, esta se cobrará por medio de Tokens**, que representan pedazos de palabras, **el máximo de tokens que se pueden mandar o recibir es de 4096**. Si se quiere medir los tokens que tiene una palabra o frase completa se puede utilizar la herramienta Tokenizer de OpenAI: <https://platform.openai.com/tokenizer>

The GPT family of models process text using **tokens**, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

TEXT	TOKENS
Olis te amo	5

A helpful rule of thumb is that one token generally corresponds to ~4 characters of text for common English text. This translates to roughly 1% of a word (so 100 tokens ≈ 75 words).

If you need a programmatic interface for tokenizing text, check out our `tiktoken` package for Python. For JavaScript, the `gpt-3-encoder` package for node.js works for most GPT-3 models.

El cobro de la API se hace en función del uso de la librería y es configurable, ya que se puede indicar un límite de cobro mensual y también se puede observar el historial de uso por medio de nuestra cuenta de OpenAI, para ello se debe ingresar a la opción de Account → Manage Account→ Usage:

The screenshot shows the OpenAI API usage dashboard. On the left, there's a sidebar with sections for Organization (dicer0), User (API keys), and Usage (selected). The main area is titled "Usage" and displays a bar chart for August 15th, showing a single green bar at \$0.01. Below the chart, a progress bar indicates usage for the month, showing \$0.01 / \$35.00. A sidebar on the right shows account details for CERVANTES RODRÍGUEZ DIEGO (diegor06@gmail.com) and various account management options.

Para evitar recibir el error `RateLimitError` al intentar ejecutar el programa, también debo asegurarme de que mis créditos se encuentren arriba de 0, pero si es así, debo comprar más.

The screenshot shows the organization settings page. The sidebar has sections for Your profile, Organization (General, Members, Billing selected), Project (General, Members, Limits), Forum, and Help. The main content is the "Billing" section, which includes tabs for Overview (selected), Payment methods, Billing history, and Preferences. Under Overview, it shows a credit balance of \$0.00 and a note about auto recharge being off. It also has buttons for Add to credit balance and Cancel plan. Below this are links for Payment methods, Billing history, Preferences, and Usage limits.

## Código Python: API ChatGPT

La parte interesante de la librería de openai que integra la herramienta de ChatGPT en nuestros programas de Python es que permite asignarle un rol para que conteste nuestras preguntas de forma personalizada y se puede configurar para que guarde el historial del chat, creando así una conversación entre el LLM y el usuario, para ello se tienen los siguientes 3 roles:

- **System:** Permite asignar un rol a ChatGPT para que conteste nuestras preguntas de forma personalizada, utilizando así la técnica **Role Play** al mandar el prompt.
- **User:** Con este rol se mandan preguntas del usuario a ChatGPT.
- **Assistant:** Este rol es adoptado por ChatGPT siempre que responda preguntas, al utilizarse en el código se permite crear un historial en el chat que guarde nuestras preguntas y sus respuestas para que se simule una conversación entre el LLM y el usuario.

Además, cabe mencionar que por buenas prácticas es recomendable declarar la API key en un archivo separado al programa principal y luego importarla en donde se quiera utilizar. Es importante mencionar que el nombre de la carpeta donde se encuentra el archivo Python que contiene las claves no debe empezar con un número ni contener espacios para que el archivo pueda ser importado.

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.

#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.

#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior derecho de Play se corre el programa.

#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#IMPORTACIÓN DE LIBRERÍAS:
import openai #openai: Librería que permite utilizar el LLM (Large Language Model) de ChatGPT con Python.

#Cabe mencionar que, al utilizar la API en su modo gratuito, solo se podrán realizar 100 llamadas a la API por día,
#si se excede ese límite, se recibirá el error RateLimitError al intentar ejecutar el programa de Python.

#IMPORTACIÓN DE LLAVE: Cuando se quiera utilizar una API que utiliza un key, por seguridad es de buenas prácticas
#declararla en un archivo externo, además cabe mencionar que el nombre de dicho archivo y constante no pueden empezar
#con un número, sino cuando la quiera importar obtendrá un error y se va accediendo a las carpetas por medio de puntos:
# - Directorio normal:      carpeta1/carpeta2/carpeta3
# - Directorio paquetes:    carpeta1.carpeta2.carpeta3
#La parte del directorio se coloca después de la palabra reservada from y la llave a importar después de import.
from API_Keys.Llaves_ChatGPT_Bard import LlaveChatGPT

#openai.api_key: A través de este atributo perteneciente a la librería openai, se declara la API key, que previamente
#debió ser creada y extraída de la página oficial de OpenAI, asociada a nuestro usuario:
#https://platform.openai.com/account/api-keys
```

```

openai.api_key = LlaveChatGPT

#INTRODUCIR POR MEDIO DE CÓDIGO UNA SOLA PREGUNTA QUE QUIERO QUE RESPONDA CHATGPT:
#openai.ChatCompletion.create(): El método create() aplicado al objeto ChatCompletion perteneciente a la librería
#openai se encarga de crear chats que se puedan mandar a ChatGPT. Este recibe los siguientes parámetros:
# - model: Describe el modelo de lenguaje que se utilizará para generar la salida. Los modelos disponibles son
#   gpt-3, gpt-4 y el más reciente es gpt-3.5-turbo. Todos se encuentran mencionados en la
#   documentación de OpenAI: https://platform.openai.com/docs/models/overview
# - messages: Representa la lista de mensajes que se utilizarán para generar la salida del chat. Cada mensaje es un
#   objeto con los siguientes campos:
#     - role: El rol del mensaje ayuda al modelo de lenguaje a entender el contexto de la conversación. Los roles
#       posibles son system, user y assistant, indicándole así de forma separada a quién está interpretando ChatGPT
#       para que de esta manera pueda dar respuestas de forma específica:
#         - system: Por medio de este rol se le indica a ChatGPT a quién está interpretando cuando responda las
#           preguntas del usuario.
#         - user: En este rol se está indicando las preguntas que está realizando el usuario a ChatGPT.
#         - assistant: Este rol es adoptado por ChatGPT siempre que responda a la pregunta de un usuario y se
#           observa en el resultado retornado por el objeto ChatCompletion después de usar el método create().
#           Su mayor uso es el de permitir que el chat recuerde entradas y salidas anteriores.
#     - content: Indica el contenido del mensaje mandado a ChatGPT.
# - max_tokens: Limita el número máximo de tokens que se devolverán en la salida, los tokens son considerados como
#   trozos de palabras, donde 1.000 tokens corresponden a unas 750 palabras. Por default el límite es de 4096, por
#   lo tanto, se pueden recibir y/o devolver como máximo más o menos 3,072 palabras, pero esto varía porque un token
#   no es igual a una palabra, sino a trozos de ellas.
# - temperature: A través de un valor entre 0.0 y 2.0 se controla la creatividad de la respuesta dada. Cuanto más
#   alto sea el valor, más creativa será la salida, pero si es muy alto la respuesta puede ser muy aleatoria y no
#   tener sentido. Esto sucede con temperaturas arriba de 1.
# - n: El parámetro n indica el número de respuestas que queremos obtener por cada pregunta.

#Todos los parámetros se pueden consultar en este enlace: https://platform.openai.com/docs/api-reference/chat/create

completion = openai.ChatCompletion.create(
    model = "gpt-3.5-turbo",
    messages = [
        {"role": "user", "content": "Cuéntame un chiste muy gracioso"}
    ],
    max_tokens = 2000,
    temperature = 0.5,
    n = 1
)

#openai.ChatCompletion: El objeto ChatCompletion recibe una lista de mensajes como entrada a través del método
#create() y devuelve un diccionario generado por el LLM como salida, que se almacena en la posición 0 del parámetro
#choices, luego dentro del parámetro message existirá el key content y role, el primero describe el mensaje devuelto
#por ChatGPT, mientras que el segundo describe el rol con el que contestó (que siempre será assistant).
print(completion.choices[0].message.content)

```

```

#INTRODUCIR POR MEDIO DE CONSOLA EL ROL DE CHATPGPT Y EL TEXTO QUE QUIERO QUE RESPONDA, ADEMÁS DE QUE ALMACENA
#SU RESULTADO PARA CREAR UNA CONVERSACIÓN:
#LISTAS: Las listas en Python son tipos de datos estructurados, parecido a lo que son los arrays en otros
#lenguajes de programación, aunque no es el único tipo de dato agrupado que existe en Python, existen además las
#tuplas, diccionarios y numpy arrays.
rolMensajesRespuestaChat = [] #Lista que almacenará el rol, mensajes y respuesta de ChatGPT.
#input(): Método que sirve para imprimir en consola un mensaje y que luego se permita al usuario ingresar un valor
#por consola, que será de tipo String y podrá ser almacenado en una variable.
rolChatGPT = input("Indica a quién quieres que interprete ChatGPT cuando conteste tus preguntas:")
#append(): Método que sirve para agregar valores a una lista, array o diccionario.
#Al indicarle a ChatGPT el rol que está interpretando no basta con solo decir el nombre de quién es, se debe
#especificar lo más posible y a mayor detalle a quién está representando y/o qué función va a llevar a cabo durante
#el chat a través de la siguiente sintaxis:
#Eres un entrevistador de TI que responde de forma muy dura mis contestaciones, Eres el famoso personaje de Marvel
#Iron Man, Eres la guapísima Alexandra Dadario y estamos en una cita, etc.
rolMensajesRespuestaChat.append({"role": "system", "content": rolChatGPT}) #Rol de ChatGPT al responder.
#print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter), además si se quiere
#concatenar un mensaje (mostrar resultados de variables junto con texto estático), este se debe separar entre
#comillas o signos de +, declarando los mensajes estáticos entre comillas y los nombres de variables sin comillas.
print("Introduce el mensaje que le quieras hacer al rol de ChatGPT que ingresaste: ")
#Bucle indeterminado while que se ejecuta hasta que lo introducido en consola sea el mensaje Adiosito para cerrar
#el chat.
preguntaChat = "" #Variable vacía tipo String.
while(preguntaChat != "Bye"):
    preguntaChat = input() #Variable que almacena la pregunta introducida en consola.
    rolMensajesRespuestaChat.append({"role": "user", "content": preguntaChat}) #Pregunta del usuario hecha a ChatGPT.
#openai.ChatCompletion.create(): El método create() aplicado al objeto ChatCompletion perteneciente a la librería
#openai se encarga de crear chats que se puedan mandar a ChatGPT. Este recibe los siguientes parámetros:
# - model: Describe el modelo de lenguaje que se utilizará para generar la salida. Los modelos disponibles son
#   gpt-3.5, gpt-4 y el más reciente es gpt-3.5-turbo.
# - messages: Representa la lista de mensajes que se utilizarán para generar la salida del chat. Cada mensaje es
#   un objeto con los siguientes campos:
#     - role: El rol del mensaje ayuda al modelo de lenguaje a entender el contexto de la conversación. Los
#       roles posibles son system, user y assistant, indicándole así de forma separada a quién está
#       interpretando ChatGPT para que de esta manera pueda dar respuestas de forma específica:
#         - system: Por medio de este rol se le indica a ChatGPT a quién está interpretando cuando responda las
#           preguntas del usuario.
#         - user: En este rol se está indicando las preguntas que está realizando el usuario a ChatGPT.
#         - assistant: Este rol es adoptado por ChatGPT siempre que responda a la pregunta de un usuario y se
#           observa en el resultado retornado por el objeto ChatCompletion después de usar el método create().
#           Su mayor uso es el de permitir que el chat recuerde entradas y salidas anteriores.
#     - content: Indica el contenido del mensaje mandado a ChatGPT.

```

```

contestacion = openai.ChatCompletion.create(
    model = "gpt-3.5-turbo",
    messages = rolMensajesRespuestaChat
)

#openai.ChatCompletion: El objeto ChatCompletion recibe una lista de mensajes como entrada a través del método
#create() que pueden indicar el rol del Chat y las preguntas que le hace el usuario al modificar su parámetro
#role; posteriormente devuelve un mensaje generado por el LLM como salida, que se almacena en la posición 0 del
#parámetro choices, dentro de este existirá un key message que a su vez contiene un key content y role, el
#primero describe el mensaje devuelto por ChatGPT, mientras que el segundo describe el rol con el que contestó,
#que siempre será el de assistant. Esto se puede realizar con alguna de las siguientes sintaxis:
# - openai.ChatCompletion.choices[0].message.content.
# - openai.ChatCompletion["choices"][0]["message"]["content"]
#Ambas sintaxis realizan la misma función.

respuestaChatGPT = contestacion["choices"][0]["message"]["content"]
#respuestaChatGPT = contestacion.choices[0].message.content
rolMensajesRespuestaChat.append({"role": "assistant", "content": respuestaChatGPT})
print("\n" + respuestaChatGPT + "\n")
"""La forma en la que el objeto openai.ChatCompletion devuelve la respuesta del chat es la siguiente:
{
    #Siempre se usa choices[0], ya que solo tiene una posición y ahí es donde se encuentra message y role.
    "choices": [
        {
            "finish_reason": "stop",
            "index": 0,
            "message": {
                "content": "The 2020 World Series was played in Texas at Globe Life Field in Arlington.",
                "role": "assistant"
            }
        }
    ],
    "created": 1677664795,
    "id": "chatcmpl-7QyqpwdfhqwjicIEznoc6Q47XAyW",
    "model": "gpt-3.5-turbo-0613",
    "object": "chat.completion",
}

#La forma en la que se cobra al utilizar la API de ChatGPT es a través de Tokens, que son considerados como
#trozos de palabras, donde 1.000 tokens corresponden a unas 750 palabras. Esta información también es
#devuelta en el resultado del objeto openai.ChatCompletion al utilizar el método create(). Para contar los
#tokens de un texto se puede utilizar el siguiente enlace: https://platform.openai.com/tokenizer
#Y debemos tomar en cuenta que el máximo número de Tokens que se pueden mandar son de 4096, si esto se excede
#se nos lanzará una excepción.

"usage": {
    "completion_tokens": 17,
    "prompt_tokens": 57,
    "total_tokens": 74
}

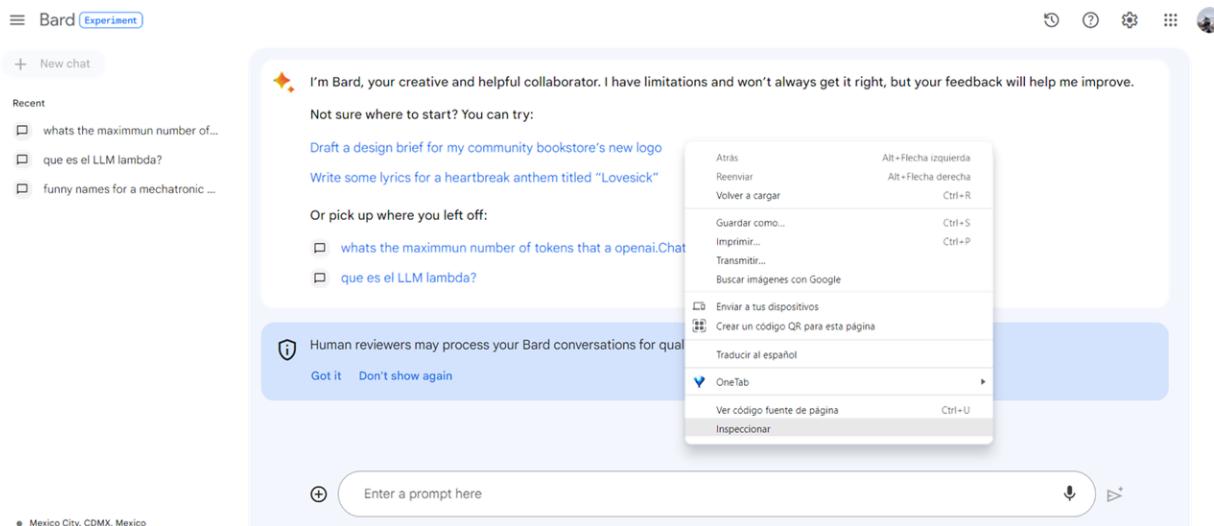
```

```
}
```

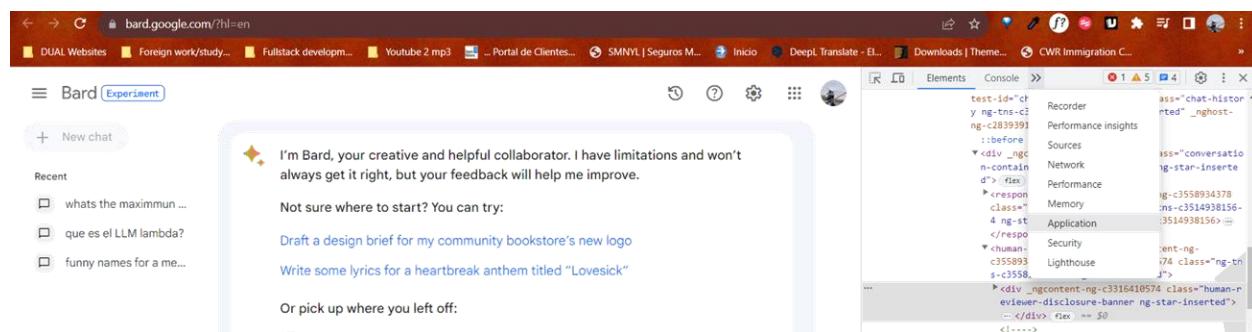
## Google Bard API

Para usar la API de Google Bard se deben seguir los pasos descritos a continuación, **aunque cabe mencionar que todavía no se puede utilizar con cuentas mexicanas**, ya que por el momento solo se encuentra disponible dentro de USA y UK:

1. Primero nos debemos introducir en el chat de Google Bard dentro del navegador de nuestra preferencia: <https://bard.google.com/>
2. Luego debemos dar clic derecho y seleccionar la opción de Inspeccionar.



3. Dentro del inspeccionador del sitio nos introduciremos en la opción de Application.



4. Posteriormente se dará clic en Application → Cookies → <https://bard.google.com/> para encontrar y copiar una llave llamada **\_Secure-1PSID**. Al hacerlo, en la parte inferior derecha, justo debajo de donde dice **Cookie Value** aparecerá el texto de la API key que se debe copiar y pegar en el código de Python:

I'm Bard, your creative and helpful collaborator. I have limitations and won't always get it right, but your feedback will help me improve.

Not sure where to start? You can try:

- Draft a design brief for my community bookstore's new logo
- Write some lyrics for a heartbreak anthem titled "Lovesick"
- Or pick up where you left off:
- what's the maximum number of tokens that a openai.ChatCompletion...
- que es el LLM lambda?

Human reviewers may process your Bard conversations for quality purposes. Don't enter sensitive info. [Learn more](#)

Got it Don't show again

Name	V...	D...	P...	E...	S...	H...	S...	P...	P...
_Secure-3PSIDCC	Z...	/	2...	✓	✓	N...	H...		
_Secure-1PSIDTS	S...	/	2...	9...	✓	✓	N...	H...	
_Secure-3PSIDTS	Z...	/	2...	9...	✓	✓	N...	H...	
<u>_Secure-1PSID</u>	Z...	/	2...	8...	✓	✓	H...		
SSID	Z...	/	2...	7...			H...		
_Secure-1PSIDCC	A...	/	2...	9...	✓	✓	H...		
SIDCC	A...	/	2...	8...			H...		
_Secure-3PAPISID	Z...	/	2...	5...	✓	✓	N...	H...	
SAPISID	Z...	/	2...	4...			H...		
APISID	Z...	/	2...	4...			H...		
SSID	A...	/	2...	2...	✓	✓	H...		
_Secure-3PSID	Z...	/	2...	8...	✓	✓	N...	H...	
<u>_Secure-1PAPISID</u>	Z...	/	2...	5...	✓	✓	H...		
HSID	A...	/	2...	2...	✓	✓	H...		
_ga_WC57K50ZZ	G...	/	2...	5...			M...		
NID	S...	/	2...	4...	✓	✓	N...	M...	
1P_JAR	Z...	/	2...	1...	✓	✓	N...	M...	
AEC	A...	/	2...	6...	✓	✓	L...	M...	
_ga_H68ZH5W8EE	G...	/	2...	5...			M...		
SEARCH__SAME SITE	C...	/	2...	2...			S...	M...	
_ga	G...	/	2...	3...			M...		
_oobr	*	/	2...	1...			M...		

Después para realizar la instalación de la librería bardapi que permite utilizar la API de Bard en programas de Python se debe ejecutar el siguiente comando en la consola CMD de Windows.

```
pip install bardapi
```

```
C:\Users\diego>pip install bardapi
Collecting bardapi
  Obtaining dependency information for bardapi from https://files.pythonhosted.org/packages/d3/25/94a8427be37b77fad0a1a0229376d46f7cd511d134bbf2347a1cb7180b5f/bardapi-0.1.33-py3-none-any.whl.metadata
    Downloading bardapi-0.1.33-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: requests in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from bardapi) (2.31.0)
Collecting deep-translator (from bardapi)
  Obtaining dependency information for deep-translator from https://files.pythonhosted.org/packages/38/3f/61a8ef73236dbea83a1a063a8af2f8e1e41a0df64f122233938391d6f175/deep_translator-1.11.4-py3-none-any.whl.metadata
    Downloading deep_translator-1.11.4-py3-none-any.whl.metadata (30 kB)
Requirement already satisfied: colorama in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from bardapi) (0.4.6)
Collecting httpx[http2]>=0.20.0 (from bardapi)
  Obtaining dependency information for httpx[http2]>=0.20.0 from https://files.pythonhosted.org/packages/ec/91/e41f64f08d2a13aae7e8c819d82ee3aa7cdc484d18c0ae859742597d5aa0/httpx-0.24.1-py3-none-any.whl.metadata
    Downloading httpx-0.24.1-py3-none-any.whl.metadata (7.4 kB)
Collecting google-cloud-translate (from bardapi)
  Obtaining dependency information for google-cloud-translate from https://files.pythonhosted.org/packages/84/8f/da05f953868e4f5cfe056da26db0e7917e86227de4821c378d199211c81f/google_cloud_translate-3.12.0-py2.py3-none-any.whl.metadata
    Downloading google_cloud_translate-3.12.0-py2.py3-none-any.whl.metadata (5.2 kB)
Collecting browser-cookie3 (from bardapi)
  Obtaining dependency information for browser-cookie3 from https://files.pythonhosted.org/packages/8a/0d/0fa3b3a13870a9beb4739449e2fc0071d6671b5c300783966541867457c/browser_cookie3-0.19.1-py3-none-any.whl.metadata
    Downloading browser_cookie3-0.19.1-py3-none-any.whl.metadata (632 bytes)
Collecting langdetect (from bardapi)
  Downloading langdetect-1.0.9.tar.gz (981 kB)
    981.5/981.5 kB 485.6 kB/s eta 0:00:00
Preparing metadata (setup.py) ... done
```

## Código Python: API Google Bard - El código todavía no sirve en México

Para activar el código de Google Bard se necesita usar una VPN, ya que si la cuenta proviene de fuera de USA o Reino Unido, al ejecutar el programa la API key se mostrará como inválida.

```
#ESTE CÓDIGO POR EL MOMENTO NO SIRVE YA QUE USA UNA API KEY DE UNA CUENTA MEXICANA, Y POR EL MOMENTO LA API
#SOLO ESTÁ DISPONIBLE PARA CUENTAS DE USA O REINO UNIDO, POR LO QUE SE ARROJA UN ERROR INDICANDO QUE ÉSTA ES
#INVÁLIDA.

# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.

#La primera linea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
```

```

#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior derecho de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#IMPORTACIÓN DE LIBRERÍAS:
import os #os: Librería que permite acceder a funciones y métodos relacionados con el sistema operativo.
from bardapi import Bard #bardapi: Librería que permite utilizar el LLM (Large Language Model) de Google Bard.

#IMPORTACIÓN DE LLAVE: Cuando se quiera utilizar una API que utiliza un key, por seguridad es de buenas prácticas
#declararla en un archivo externo, además cabe mencionar que el nombre de dicho archivo y constante no pueden empezar
#con un número, sino cuando la quiera importar obtendré un error y se va accediendo a las carpetas por medio de puntos:
# - Directorio normal:      carpeta1/carpeta2/carpeta3
# - Directorio paquetes:    carpeta1.carpeta2.carpeta3
#La parte del directorio se coloca después de la palabra reservada from y la llave a importar después de import.
from API_Keys.Llaves_ChatGPT_Bard import LlaveBard

#os.environ: El método environ proveniente de la librería os permite acceder a las variables de entorno del sistema
#operativo, estas están organizadas en una forma de key:value, por lo que serán datos tipo diccionario o JSON, y su
#objetivo es proveer cierta información de configuración que afecte a múltiples programas o aplicaciones, evitando
#así que cada uno deba ser configurado por separado, pudiendo adaptarse automáticamente a diferentes entornos al leer
#las variables de entorno relevantes, que usualmente almacenan información sensible, como contraseñas o API keys.
#Para crear una nueva variable de entorno se utiliza la siguiente sintaxis, indicando su nombre en mayúsculas:
#   os.environ['NOMBRE_VARIABLE'] = 'valorVariableDeEntorno'

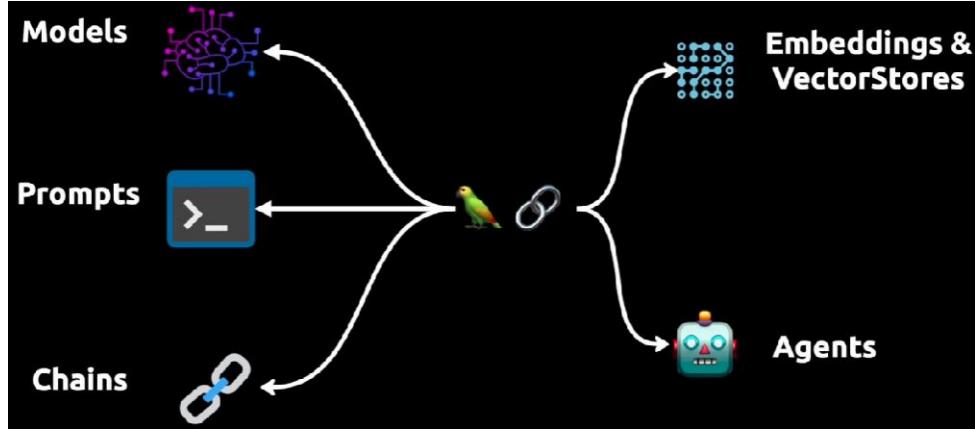
#Para almacenar una variable de entorno en una variable se utiliza la siguiente sintaxis:
#   variable = os.environ.get('NOMBRE_VARIABLE')
os.environ["BARD_API_KEY"] = LlaveBard
APIkey = os.environ.get("BARD_API_KEY")
#bardapi.Bard(): El constructor de Bard como mínimo recibe la API key.
googleBard = Bard(token = APIkey)
#bardapi.Bard().get_answer(): El método get_answer() sirve para mandar una pregunta de forma directa a Google Bard.
result = googleBard.get_answer("Cuéntame un chiste muy gracioso")
print(result)

```

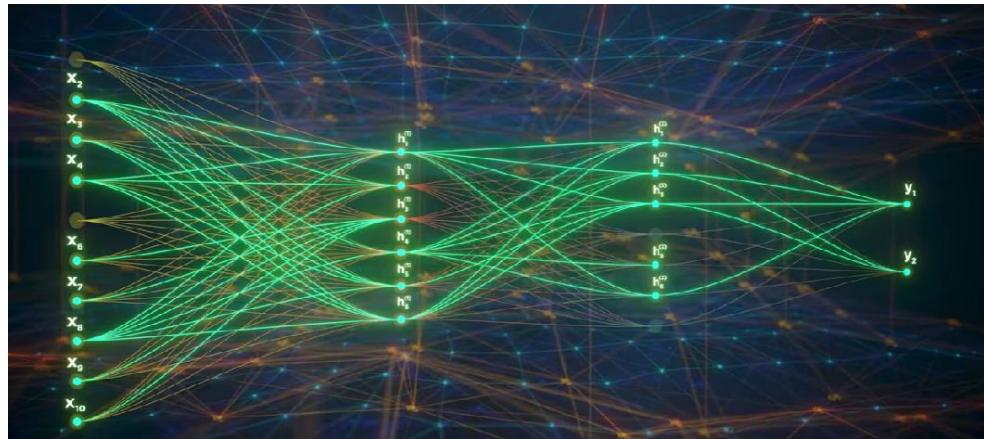
## LangChain

A las APIs de Google Bard o ChatGPT se les envía un texto y me retornan otro que ya haya sido procesado por el LLM, pero si se busca que estos modelos de lenguaje realicen acciones que no pueden hacer bien por sí solas como resolver ecuaciones matemáticas o que tomen información de archivos propios, bases de datos privadas, páginas web específicas o inclusive de otros modelos, ahí es donde entra en juego la herramienta open source de LangChain. Cabe mencionar que la frase “realizar acciones” no se refiere a que el modelo pueda ejecutar funciones externas al código como prender luces o cosas

del estilo, sino que se refiere a que pueda contestar preguntas de mejor forma a través de distintas herramientas o fuentes de información. Para ello se hace uso de las siguientes 6 herramientas:



1. **Modelos (Models):** El modelo se refiere a la red neuronal que se va a utilizar para procesar el texto de entrada y generar una respuesta. Existen varios hasta dentro de una misma compañía, por ejemplo, OpenAI cuenta con gpt3, gpt4, gpt3.5 turbo, etc.



2. **Prompt:** Es el texto que se le envía al modelo para generar una respuesta y en este es donde se utilizan las técnicas de **Prompt Engineering** previamente explicadas, para ello la librería LangChain cuenta con diferentes clases que permiten utilizar dichas técnicas:
  - **Prompt Template:** Esta clase permite crear una plantilla, utilizada cuando se quiera mandar una instrucción con algunas partes que puedan ser variables a una LLM, como por ejemplo cuando se quiere traducir cierto documento; la instrucción será la misma, pero el contenido del Prompt será distinto.
    - **FewShotTemplate:** Esta clase permite complementar la plantilla creada con ejemplos que le den contexto al LLM para que entienda el formato y tipo de respuesta que queremos que entregue, utilizando así la técnica **Few-Shot Prompting de Prompt Engineering**.
  - **Chat Prompt Template:** Esta clase permite armar una plantilla que contenga instrucciones constantes y contenidos variables para mandársela a un chat, que a diferencia de la clase **Prompt Template**, guardará el historial de la conversación. Para ello hace uso de 3 clases con el fin de referirse al rol que adopta el chat al responder, el

mensaje mandado por el usuario y la respuesta que devuelve, de la misma forma como se hace con la API de ChatGPT.

- **System Message Prompt Template:** Con esta clase se representa el rol que indica a ChatGPT a quién está interpretando cuando responda las preguntas del usuario.
- **Human Message Prompt Template:** Con esta clase se representa el rol del usuario que manda preguntas a ChatGPT.
- **AI Message Prompt Template:** Con esta clase se representa el rol que es adoptado por ChatGPT siempre que responda la pregunta de un usuario. Su mayor uso es el de permitir que el chat recuerde entradas y salidas anteriores.

- **Output Parser:** Esta clase permite dar cierto formato a la respuesta recibida de un LLM, ya sea para ordenarla en forma de lista, diccionario, XML, JSON, etc.

3. **Memoria (Memory):** Permite almacenar las preguntas y respuestas hechas entre el LLM y el usuario, permitiendo así que se simule una conversación entre ambos, para ello la librería LangChain cuenta con diferentes clases que permiten almacenar la información del chat de distintas formas:

- **ConversationBufferMemory:** Clase que guarda todo el historial, osea cada cosa que haya dicho el o los usuarios del chat y todo lo que haya contestado el modelo.
- **ConversationBufferWindowMemory:** Clase que guarda solo los últimos mensajes del historial, esto se hace para que el costo del uso de la API baje, ya que utiliza menos tokens, pero se corre el riesgo de perder información importante del chat.
- **ConversationSummaryMemory:** Clase que a través de una conexión con un segundo modelo (Cadena) crea y guarda un resumen en inglés de todo el historial.
- **ConversationKGMemory:** Clase que a través de una conexión con un segundo modelo (Cadena) crea y guarda una lista de palabras clave del chat llamada **Knowledge Graph** para darle contexto al modelo cuando responda preguntas posteriores, el problema de usar esta herramienta es que tiene problemas de traducción y si se realizan las preguntas en un lenguaje que no sea inglés, se puede llegar a tener errores al interpretar el historial.

4. **Cadenas (Chains):** Este es de los conceptos más importantes de la librería, ya que permite conectar varios modelos entre sí, hasta cuando son de distintos tipos, permitiéndonos así realizar varias iteraciones entre modelos durante una consulta para obtener un mejor procesamiento final de los datos cuando este se busca aplicar a tareas muy complejas, existen dos tipos globales de cadenas:

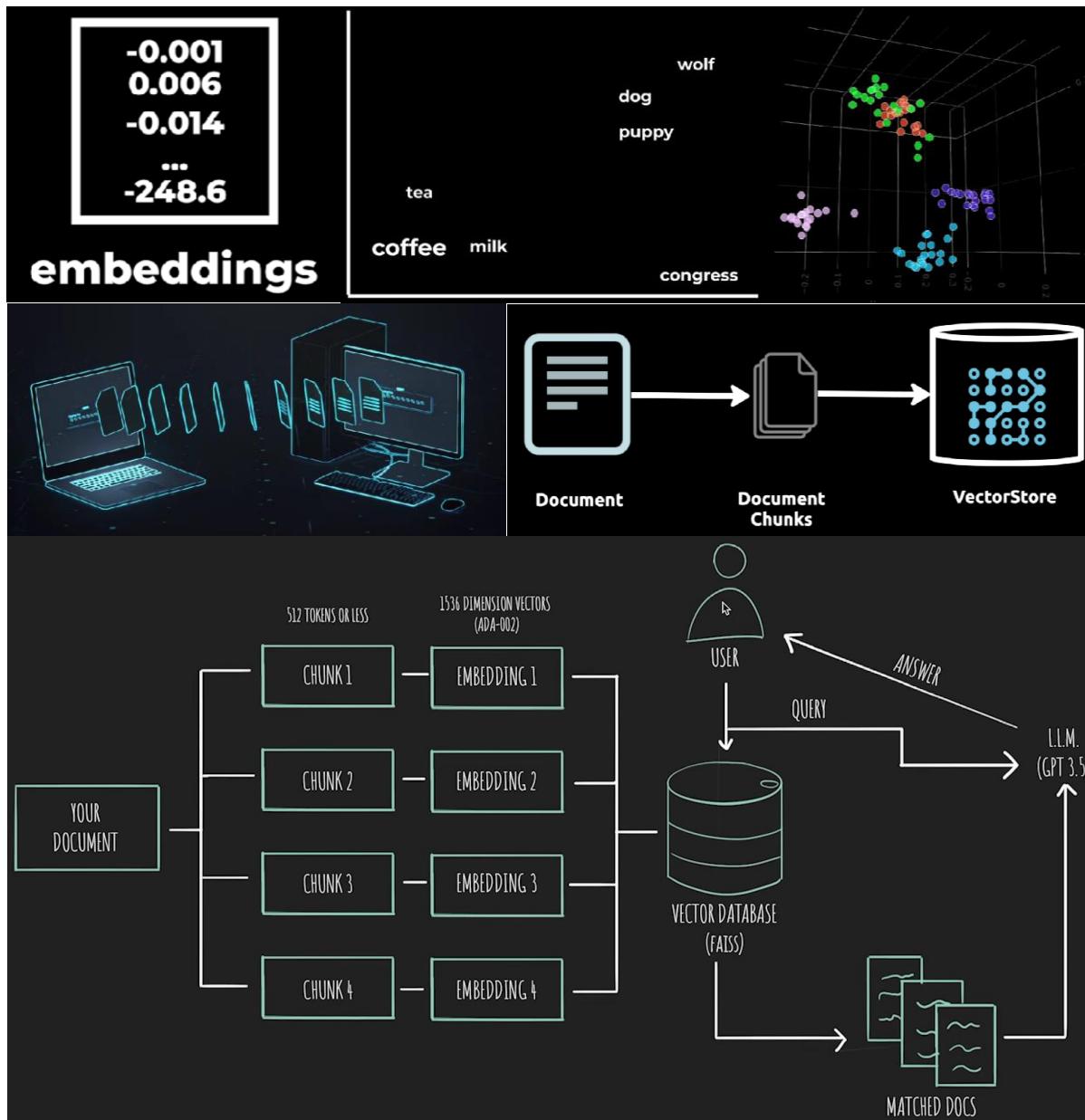
- **LLMChain:** Con esta clase se conecta un prompt con un modelo de lenguaje, creando así una cadena individual.
- **SequentialChain:** Con esta clase se pueden conectar dos o más cadenas individuales (osea modelos de lenguaje que se encuentran enlazados con un prompt), pudiendo recibir así múltiples entradas y generar múltiples salidas, ya que la salida de una cadena puede ser la entrada de otra cadena de forma secuencial.
- **SimpleSequentialChain:** Con esta clase que se realiza lo mismo que con la cadena **SequentialChain** pero con la condición de que solo puede recibir 1 entrada y proporcionar 1 salida.



5. **Índices (Retrieval o Data connection)**: La forma en la que más se aprovechan los modelos de lenguaje es cuando se les da acceso a distintas fuentes de información, como lo puede ser un archivo PDF, Word, Excel, PowerPoint, etc. Los índices en LangChain son los que nos van a permitir enlazar un gran número de documentos para que sean procesados por el modelo, para ello la librería cuenta con diferentes clases que permiten realizar el enlace:

- **DirectoryLoader**: Esta clase permite abrir, cargar y procesar todos los archivos de un mismo tipo que se encuentren en una carpeta para después pasárselos a los LLM y que sean procesados, ya sea que tengan extensión PDF, txt, Word, etc.
- **CharacterTextSplitter**: Como los modelos solo aceptan un número finito de Tokens (pedazos de palabras), lo que se hace para que el LLM pueda procesar toda esa información es dividirla en cachos llamados Chunks, lo cual es realizado por esta clase.
  - **PyPDFLoader**: Clase de la librería langchain que permite ingresar un archivo pdf al programa y dividirlo en función de su número de páginas.
  - **PdfReader**: Clase de la librería PyPDF2 que permite ingresar un archivo pdf al programa y dividirlo en función de su número de páginas y luego el contenido de sus páginas lo divide en cachos a través de la función **CharacterTextSplitter**.
- **OpenAIEmbeddings**: Los LLM asocian las palabras que reciben a través de un vector llamado **Embedding**, el cual es un simple array de varias dimensiones que se encuentra en un espacio vectorial, cuya función es asociar de forma gráfica una palabra con otras parecidas y/o alejarla de otras que sean muy distintas, de esta manera es como el modelo entiende el lenguaje humano. Si este proceso de conversión de un chunk de palabras a un embedding se realiza a través de OpenAI, es cobrado por medio de la API Key en función del número de Tokens que son convertidos a vectores numéricos.
  - **FAISS y Chroma**: Las clases FAISS y Chroma representan dos tipos de **Vector Stores** de OpenAI, las cuales son bases de datos optimizadas para almacenar los vectores obtenidos después de procesar Chunks de información.
    - **FAISS**: Vector store de rápida reacción, poco flexible y difícil de usar.
    - **Chroma**: Vector store de lenta reacción, flexible y fácil de usar.
- **Retriever**: Una vez que los datos de una fuente externa a OpenAI hayan sido convertidos a vectores y luego almacenados en una base de datos, se podrá realizar consultas al modelo y este contestará en función de la información contenida en los documentos anexados.
  - **RetrievalQA**: Esta clase permite traer información de alguna fuente en específico para contestar una pregunta hecha a un modelo LLM, sabiendo a dónde tiene que ir a buscar para obtener la respuesta solicitada.
  - **ConversationalRetrievalChain**: Esta clase permite traer información de alguna fuente en específico para contestar una pregunta hecha a un modelo de Chat, sabiendo a dónde tiene que ir a buscar para obtener la respuesta solicitada,

además de traer de vuelta de qué documento y chunk obtuvo la información requerida para contestar la pregunta hecha por el usuario.



6. **Agentes (Agents):** Estos son modelos o cadenas a las cuales se les da acceso a una fuente o API para que puedan realizar alguna acción que no pueda ser bien ejecutada por el LLM, solucionando así una tarea específica, como obtener datos meteorológicos, resolver ecuaciones, matemáticas, etc. Esto se realiza a través de las siguientes herramientas externas:

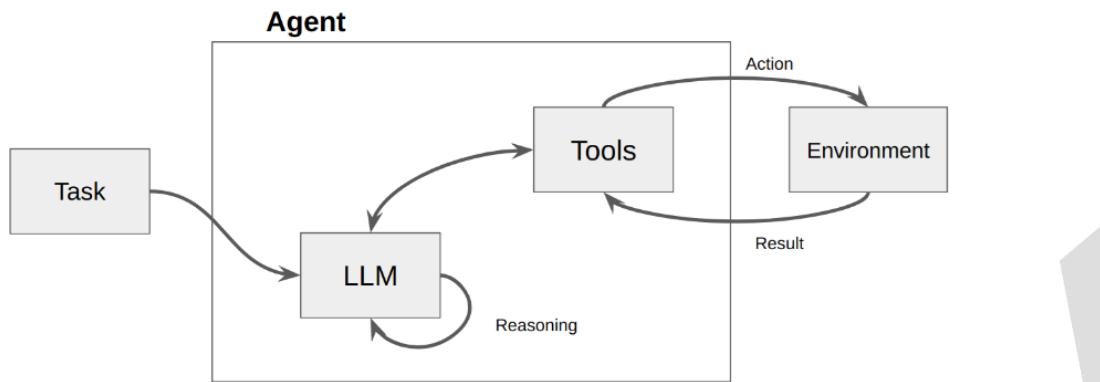
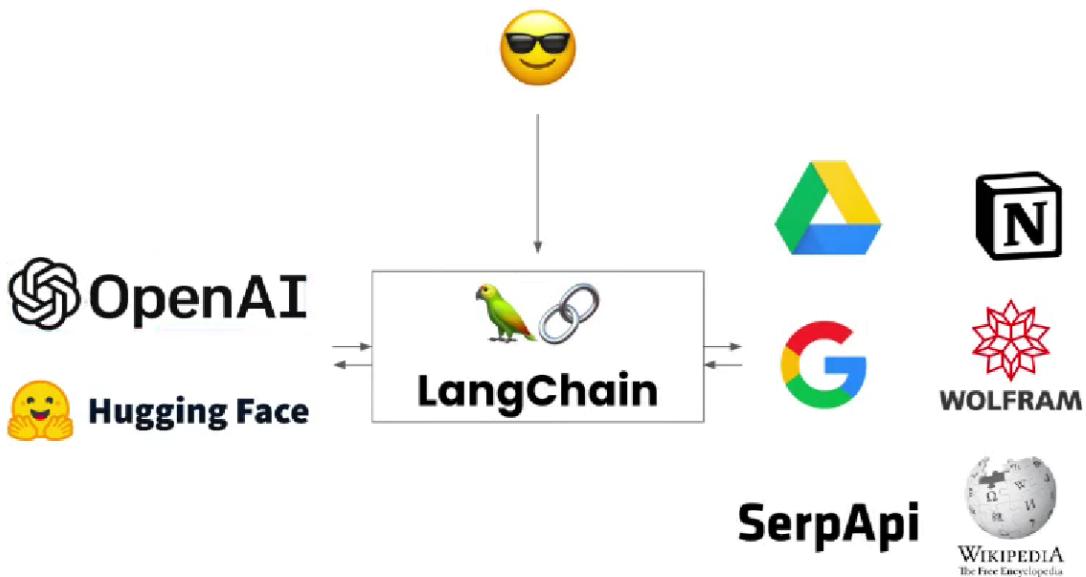
- **Tools:** Son las herramientas con las que ya cuenta langchain para realizar una acción.
  - **Ejecutar comandos en consola:**
    - **terminal:** Herramienta que permite ejecutar comandos en consola.
    - **python\_repl:** Esta herramienta permite ejecutar solamente scripts (programas) de Python a través de la consola del sistema.

- **Realizar búsquedas en internet:**
  - **serpapi:** Tool que permite extraer información de internet para responder una realizada por el usuario.
  - **google-search:** Esta herramienta de langchain permite utilizar específicamente el buscador de Google para obtener la información que responde una pregunta realizada al agente.
  - **wikipedia:** Tool que permite buscar información en Wikipedia.
  - **requests:** Esta herramienta permite extraer información de la URL de un sitio web en específico para responder una pregunta.
- **Resolver o contestar preguntas acerca de operaciones matemáticas:** Cuando se usen estas tools es recomendable declarar que la temperatura del modelo sea de 0, para que siempre dé el mismo resultado.
  - **wolfram-alpha:** Esta herramienta permite resolver problemas o contestar preguntas que tengan que ver con matemáticas, ciencia, tecnología, etc.
  - **pal-math:** Esta herramienta permite resolver problemas matemáticos a través de una instrucción, como crear ecuaciones a través de un problema de la vida real y cosas por el estilo.
  - **llm-math:** Esta herramienta permite resolver problemas matemáticos.
- **Obtener información meteorológica:**
  - **open-meteo-api:** Permite obtener información meteorológica a través de la herramienta OpenMeteo.
- **Obtener información de noticias recientes o películas:**
  - **news-api:** Obtiene información acerca de noticias actuales.
  - **tmdb-api:** Obtiene información acerca de películas.
- **Herramientas personalizadas:** Para ello se debe crear una clase donde se declare un nombre, descripción (que indica cuando esta se ejecuta) y una función propia que describe que la acción a realizar por medio de código.
  - **BaseTool:** Para declarar una herramienta personalizada se crea una clase propia que herede de BaseTool, dentro de ella se declaran los valores de los parámetros name, description y las funciones que ejecutan la acción personalizada.
- **Agente:** Existen los diferentes tipos de agentes descritos a continuación:
  - **zero-shot-react-description:** Este tipo de agente es el más utilizado y funciona con un modelo LLM, por lo que no tendrá memoria. Para ello primero razona sobre la pregunta que se le hizo, luego recopila información de las herramientas que tenga disponibles y finalmente contesta algo.
  - **conversational-react-description:** Este tipo de agente funciona con un modelo de Chat, por lo que en este caso sí se guardará el historial de la conversación a través de una variable de memoria. Para ello primero razona sobre la pregunta que se le hizo, luego recopila información de las herramientas que tenga disponibles y finalmente contesta algo.
  - **react-docstore:** Este tipo de agente está hecho para interactuar con mucha información extraída de documentos o artículos extraídos de buscadores como

Wikipedia o Google que ya deben estar anexados al modelo por medio de índices, para que a través de ellos conteste las preguntas hechas por el usuario.

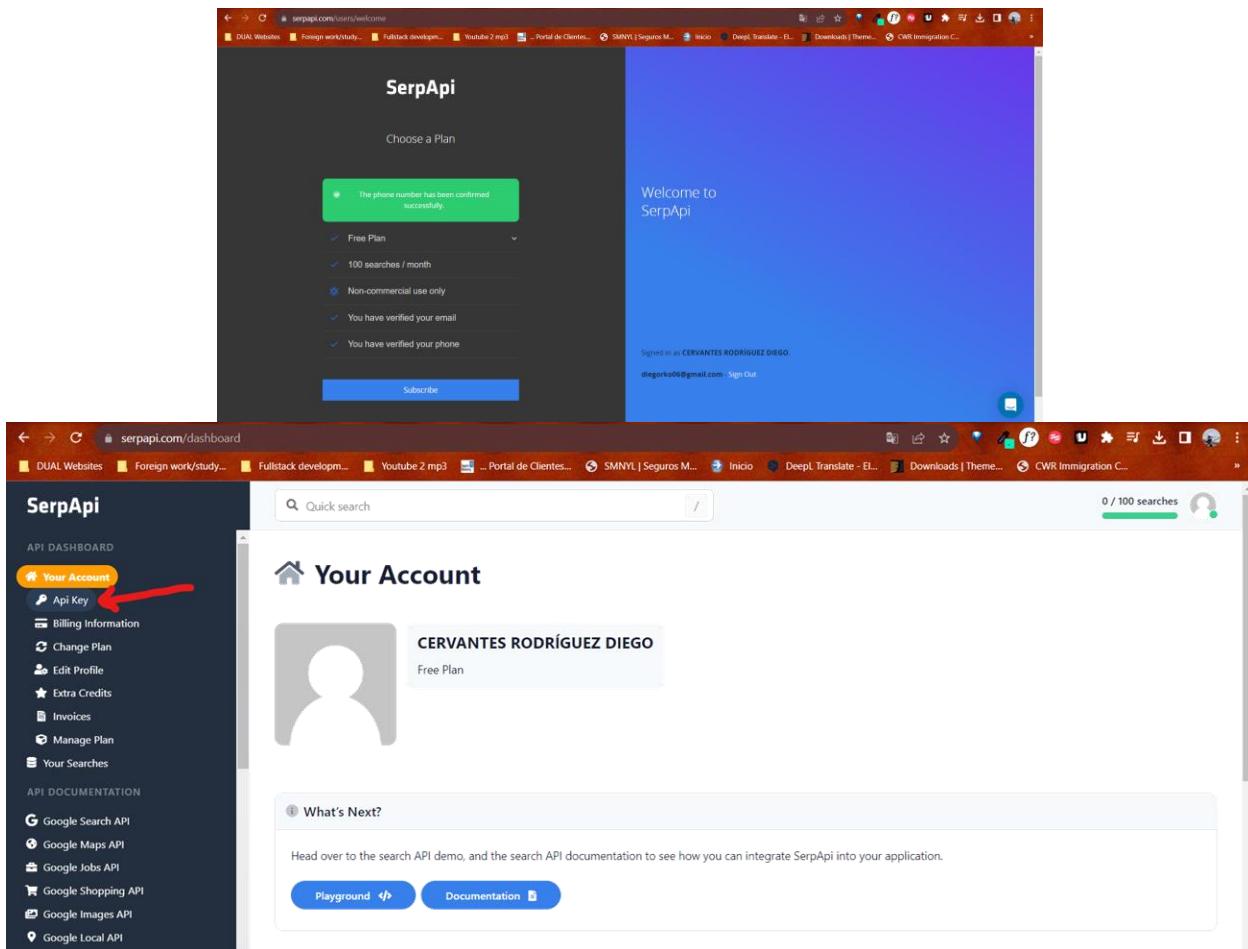
- **self-ask-with-search:** Este tipo de agente lo que hace es realizarse preguntas intermedias a sí mismo que tengan que ver con la pregunta hecha por el usuario, luego investiga la respuesta de dichas preguntas de forma individual en un buscador y utiliza las respuestas encontradas para responder la pregunta principal. Debido a su funcionamiento, forzosamente debe tener integrada una herramienta que le permita realizar búsquedas en internet.

El mejor tipo de agente a elegir dependerá de las necesidades específicas del proyecto. Los agentes **zero-shot-react-description** y **conversational-react-description** son buenos al realizar tareas generales, el agente **react-docstore** es mejor utilizarlo para interactuar con un almacén de documentos y el agente **self-ask-with-search** es una buena opción para realizar búsquedas en la web.



Para poder utilizar la Tool **serpapi** que permite realizar búsquedas en Google se debe activar la SerpApi, para ello ingresamos en el siguiente enlace, creamos nuestra cuenta y damos clic en la opción de Api Key para que esta la copiemos y peguemos en el código:

<https://serpapi.com/users/welcome>



## Instalaciones:

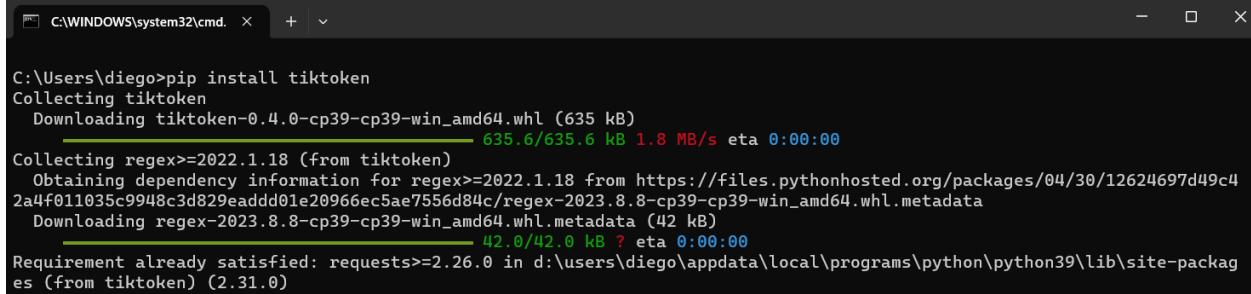
- **LangChain:** Para poder utilizar la librería LangChain en programas de Python se debe ejecutar el siguiente comando en la consola CMD de Windows.

```
pip install langchain
```

```
C:\Users\diego>pip install langchain
Collecting langchain
  Obtaining dependency information for langchain from https://files.pythonhosted.org/packages/64/f0/073d8033fb0aa47966d829b673ee4652284aa58ff38864bb2f2978
  f0d3\langchain-0.0.68-py3-none-any.whl.metadata
    Downloading langchain-0.0.68-py3-none-any.whl.metadata (14 kB)
Collecting PyYAML<5.3, >=3.11 (from langchain)
  Obtaining dependency information for PyYAML<5.3, >=3 from https://files.pythonhosted.org/packages/84/4d/82704d1ab9290b03da94e6425f5e87396b999fd7eb8e08f3a92c15
  8402fbf7\PyYAML-6.0.1-cp39-cp39-win_amd64.whl.metadata
    Downloading PyYAML-6.0.1-cp39-cp39-win_amd64.whl.metadata (2.1 kB)
Collecting SQLAlchemy<3.1, >=1.4 (from langchain)
  Obtaining dependency information for SQLAlchemy<3.1, >=1.4 from https://files.pythonhosted.org/packages/01/fb/becdf158acfec58b7a3d04affe79df9f8e5f95aa2114b5d
  acf59a8752f28\SQLAlchemy-2.0.28-cp39-cp39-win_amd64.whl.metadata
    Downloading SQLAlchemy-2.0.28-cp39-cp39-win_amd64.whl.metadata (9.7 kB)
Requirement already satisfied: six<2.0.0, >=1.11.0 (in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from langchain)) (1.14.0)
Requirement already satisfied: async-timeout<5.0.0, >=4.0.0 (in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from langchain)) (4.0.3)
Collecting dataclasses<0.6.0, >=0.5.7 (from langchain)
  Obtaining dependency information for dataclasses<0.6.0, >=0.5.7 from https://files.pythonhosted.org/packages/97/f5/e7cc9bf36152810cab88b6c9c1125e8bc9
  df9fb883\dataclasses-0.6.0.1-py3-none-any.whl.metadata
    Downloading dataclasses-0.6.0.1-py3-none-any.whl.metadata (22 kB)
Collecting langsmith<1.0, >=0.21 (from langchain)
  Obtaining dependency information for langsmith<1.0, >=0.21 from https://files.pythonhosted.org/packages/d7/f7/0b36e22cbdb52fdfdf75f3216c3a8ea259f30eef1
  e2abfd56c215634829d\langsmith-0.0.25-py3-none-any.whl.metadata
    Downloading langsmith-0.0.25-py3-none-any.whl.metadata (10 kB)
Collecting numpy<2.0, >=1.20 (from langchain)
  Obtaining dependency information for numpy<2.0, >=1.20 from https://files.pythonhosted.org/packages/41/17/22c110d5935e7201bdb33e9b96f50336cf2d06773d6c
  d89fd1b1214910\numpy-2.0.5-cp39-cp39-win_amd64.whl.metadata
    Downloading numpy-2.0.5-cp39-cp39-win_amd64.whl.metadata (2.2 kB)
Requirement already satisfied: numpy<2,>=1 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from langchain) (1.21.2)
Collecting pydantic<3,>=1 (from langchain)
  Obtaining dependency information for pydantic<3,>=1 from https://files.pythonhosted.org/packages/fd/35/86b61e7571e695587d7dd2937100436dcceaa277d2f016d4e
  4f7d3791a\pydantic-2.2.1-py3-none-any.whl.metadata
    Downloading pydantic-2.2.1-py3-none-any.whl.metadata (145 kB)
Requirement already satisfied: requests<3,>=2 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from langchain) (2.31.0)
Collecting tenacity<9.0.0, >=8.1.0 (from langchain)
  Obtaining dependency information for tenacity<9.0.0, >=8.1.0 from https://files.pythonhosted.org/packages/f4/f1/990741d5bb2487d529d20a433210ffa136a367751e4
  505\tenacity-8.2.3-py3-none-any.whl.metadata (1.0 kB)
Requirement already satisfied: attrs<17.3.0, >=17.3.0 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from aiohttp<4.0.0, >=3.8.3>langcha
  165.6/145.6 kB 222.3 kB/s eta 0:00:08
```

- **Prompt - TikToken:** Si se quiere calcular el número de Tokens mandados en un Prompt de LangChain se debe ejecutar el siguiente comando en la consola CMD de Windows para realizar la instalación de la librería tiktoken.

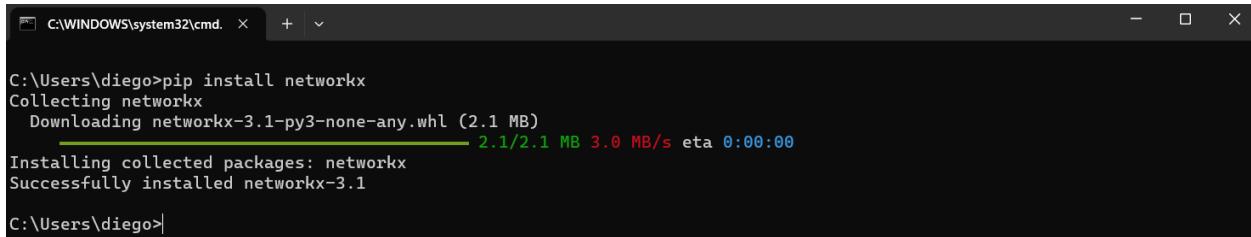
```
pip install tiktoken
```



```
C:\Users\diego>pip install tiktoken
Collecting tiktoken
  Downloading tiktoken-0.4.0-cp39-cp39-win_amd64.whl (635 kB)
    635.6/635.6 kB 1.8 MB/s eta 0:00:00
Collecting regex>=2022.1.18 (from tiktoken)
  Obtaining dependency information for regex>=2022.1.18 from https://files.pythonhosted.org/packages/04/30/12624697d49c42a4f011035c9948c3d829eadd01e20966ec5ae7556d84c/regex-2023.8.8-cp39-cp39-win_amd64.whl.metadata
  Downloading regex-2023.8.8-cp39-cp39-win_amd64.whl.metadata (42 kB)
    42.0/42.0 kB ? eta 0:00:00
Requirement already satisfied: requests>=2.26.0 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from tiktoken) (2.31.0)
```

- **Memoria - Knowledge Graph:** Si se quiere conectar distintos modelos para guardar en memoria las palabras clave del historial de una conversación con la clase **ConversationKGMemory** de la librería LangChain se debe ejecutar el siguiente comando en la consola CMD de Windows para realizar la instalación de la librería networkx.

```
pip install networkx
```



```
C:\Users\diego>pip install networkx
Collecting networkx
  Downloading networkx-3.1-py3-none-any.whl (2.1 MB)
    2.1/2.1 MB 3.0 MB/s eta 0:00:00
Installing collected packages: networkx
Successfully installed networkx-3.1
C:\Users\diego>
```

- **Índices - PdfReader - PyPDF2:** Si se quiere leer el contenido de un pdf con una herramienta fuera de la librería LangChain se debe ejecutar el siguiente comando en la consola CMD de Windows para realizar la instalación de la librería PyPDF2.

```
pip install PyPDF2
```



```
C:\Users\diego>pip install PyPDF2
Collecting PyPDF2
  Using cached pypdf2-3.0.1-py3-none-any.whl (232 kB)
Requirement already satisfied: typing_extensions>=3.10.0.0 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from PyPDF2) (4.7.1)
Installing collected packages: PyPDF2
Successfully installed PyPDF2-3.0.1
C:\Users\diego>
```

- **Índices - PyPDFLoader - LangChain:** Si se quiere leer el contenido de un pdf a través de una herramienta de índices perteneciente a la biblioteca LangChain se debe ejecutar el siguiente comando en la consola CMD de Windows para realizar la instalación de la librería pypdf.

```
pip install pypdf
```

```
C:\Users\diego>pip install pypdf
Collecting pypdf
  Obtaining dependency information for pypdf from https://files.pythonhosted.org/packages/7e/19/d90c9a6b187df41a0136c8ef
  fdede51a63ed23c0be80f9792f042c592df8/pypdf-3.15.2-py3-none-any.whl.metadata
    Using cached pypdf-3.15.2-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: typing_extensions>=3.10.0.0 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from pypdf) (4.7.1)
Using cached pypdf-3.15.2-py3-none-any.whl (271 kB)
Installing collected packages: pypdf
Successfully installed pypdf-3.15.2

C:\Users\diego>
```

- **Índices - DirectoryLoader (.txt) - LangChain:** Para poder leer todos los documentos con extensión txt de un directorio en específico se debe ejecutar el siguiente comando en la consola CMD de Windows para realizar la instalación de la librería unstructured.

```
pip install unstructured
```

```
C:\Users\diego>pip install unstructured
Collecting unstructured
  Obtaining dependency information for unstructured from https://files.pythonhosted.org/packages/f9/a9/4069cd659fb920bf771d3482a3a56846731d4e9abc5efa1223d65df1605/unstructured-0.10.5-py3-none-any.whl.metadata
    Downloading unstructured-0.10.5-py3-none-any.whl.metadata (22 kB)
Collecting chardet (from unstructured)
  Obtaining dependency information for chardet from https://files.pythonhosted.org/packages/38/6f/f5fbc992a329ee4e0f288c1fe0e2ad9485ed064cac731ed2fe47dcc38cbf/chardet-5.2.0-py3-none-any.whl.metadata
    Downloading chardet-5.2.0-py3-none-any.whl.metadata (3.4 kB)
Collecting filetype (from unstructured)
  Downloading filetype-1.2.0-py2.py3-none-any.whl (19 kB)
Collecting python-magic (from unstructured)
  Downloading python_magic-0.4.27-py2.py3-none-any.whl (13 kB)
Collecting lxml (from unstructured)
  Obtaining dependency information for lxml from https://files.pythonhosted.org/packages/80/2e/49751104148b03ad880aaaf381cc24d67b7d8f401f7d074ad7db4f6d95597/lxml-4.9.3-cp39-cp39-win_amd64.whl.metadata
    Downloading lxml-4.9.3-cp39-cp39-win_amd64.whl.metadata (3.9 kB)
Collecting nltk (from unstructured)
  Downloading nltk-3.8.1-py3-none-any.whl (1.5 MB)
    1.5/1.5 MB 4.2 MB/s eta 0:00:00
Requirement already satisfied: tabulate in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured) (0.9.0)
```

- **Índices - DirectoryLoader (.pdf) - LangChain:** Para poder leer todos los documentos con extensión pdf de un directorio en específico se debe ejecutar el siguiente comando en la consola CMD de Windows para realizar la instalación de la librería unstructured [pdf].

```
pip install unstructured[pdf]
```

```
C:\Users\diego>pip install unstructured[pdf]
Requirement already satisfied: unstructured[pdf] in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (0.10.5)
Requirement already satisfied: chardet in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (5.2.0)
Requirement already satisfied: filetype in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (1.2.0)
Requirement already satisfied: python-magic in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (0.4.27)
Requirement already satisfied: lxml in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (4.9.3)
Requirement already satisfied: nltk in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (3.8.1)
Requirement already satisfied: tabulate in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (0.9.0)
Requirement already satisfied: requests in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (2.31.0)
Requirement already satisfied: beautifulsoup4 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (4.12.2)
Requirement already satisfied: emoji in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (2.8.0)
Collecting pdf2image (from unstructured[pdf])
  Downloading pdf2image-1.16.3-py3-none-any.whl (11 kB)
Collecting pdfminer.six (from unstructured[pdf])
  Downloading pdfminer.six-20221105-py3-none-any.whl (5.6 MB)
    5.6/5.6 MB 3.8 MB/s eta 0:00:00
Requirement already satisfied: Pillow<10 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[pdf]) (8.3.2)
```

- **Índices - DirectoryLoader (.docx) - LangChain:** Para poder leer todos los documentos con extensión docx (Word) de un directorio en específico se debe ejecutar el siguiente comando en la consola CMD de Windows para realizar la instalación de la librería unstructured [docx].

```
pip install unstructured[docx]
```

```
C:\Users\diego>pip install unstructured[docx]
Requirement already satisfied: unstructured[docx] in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (0.10.5)
Requirement already satisfied: chardet in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (5.2.0)
Requirement already satisfied: filetype in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (1.2.0)
Requirement already satisfied: python-magic in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (0.4.27)
Requirement already satisfied: lxml in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (4.9.3)
Requirement already satisfied: nltk in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (3.8.1)
Requirement already satisfied: tabulate in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (0.9.0)
Requirement already satisfied: requests in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (2.31.0)
Requirement already satisfied: beautifulsoup4 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (4.12.2)
Requirement already satisfied: emoji in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from unstructured[docx]) (2.8.0)
Collecting python-docx (from unstructured[docx])
  Downloading python-docx-0.8.11.tar.gz (5.6 MB)
    Preparing metadata (setup.py) ... done
Requirement already satisfied: soupsieve>1.2 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from beautifulsoup4->unstructured[docx]) (2.4.1)
Requirement already satisfied: click in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from nltk->unstructured[docx]) (8.1.7)
```

- **Índices - FAISS - Vector Stores:** Para almacenar los embeddings (vectores numéricos) que representan los datos pertenecientes a los archivos anexados al programa en un **Vector Store** de tipo **FAISS** se debe ejecutar el siguiente comando en la consola CMD de Windows.

```
pip install faiss-cpu
```

```
C:\Users\diego>pip install faiss-cpu
Collecting faiss-cpu
  Downloading faiss_cpu-1.7.4-cp39-cp39-win_amd64.whl (10.8 MB)
    10.8/10.8 MB 3.1 MB/s eta 0:00:00
Installing collected packages: faiss-cpu
Successfully installed faiss-cpu-1.7.4
C:\Users\diego>
```

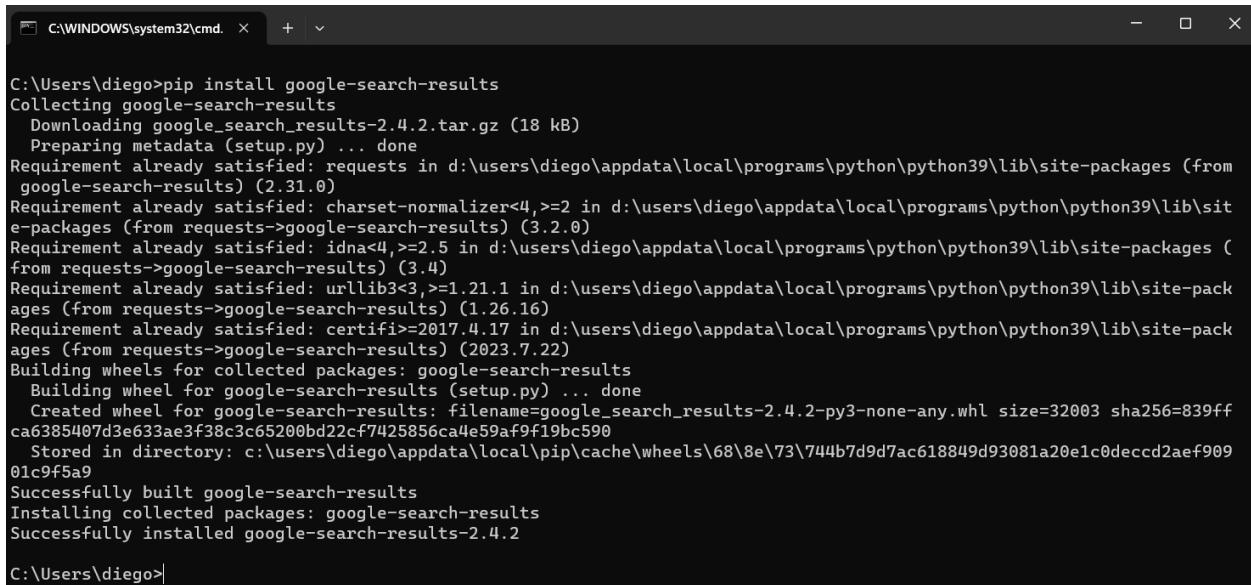
- **Índices - Chroma - Vector Stores:** Para almacenar los embeddings (vectores numéricos) que representan los datos pertenecientes a los archivos anexados al programa en un **Vector Store** de tipo **Chroma** se debe ejecutar el siguiente comando en la consola CMD de Windows.

```
pip install chromadb
```

```
C:\Users\diego>pip install chromadb
Collecting chromadb
  Obtaining dependency information for chromadb from https://files.pythonhosted.org/packages/f1/c2/d882d8650ba7cdb23f69d3abfb6bf11104cd408ef4ufbec74b0ec0f36b9ea/chromadb-0.4.6-py3-none-any.whl.metadata
  Downloading chromadb-0.4.6-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: requests>=2.28 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from chromadb) (2.31.0)
```

- **Agentes - SerpApi - Google Search Results:** Si se quiere proporcionar a un agente la habilidad de realizar búsquedas en internet a través de la herramienta SERPAPI se debe ejecutar el siguiente comando en la consola CMD de Windows para realizar la instalación del buscador a través de la librería Google Search Results.

```
pip install google-search-results
```

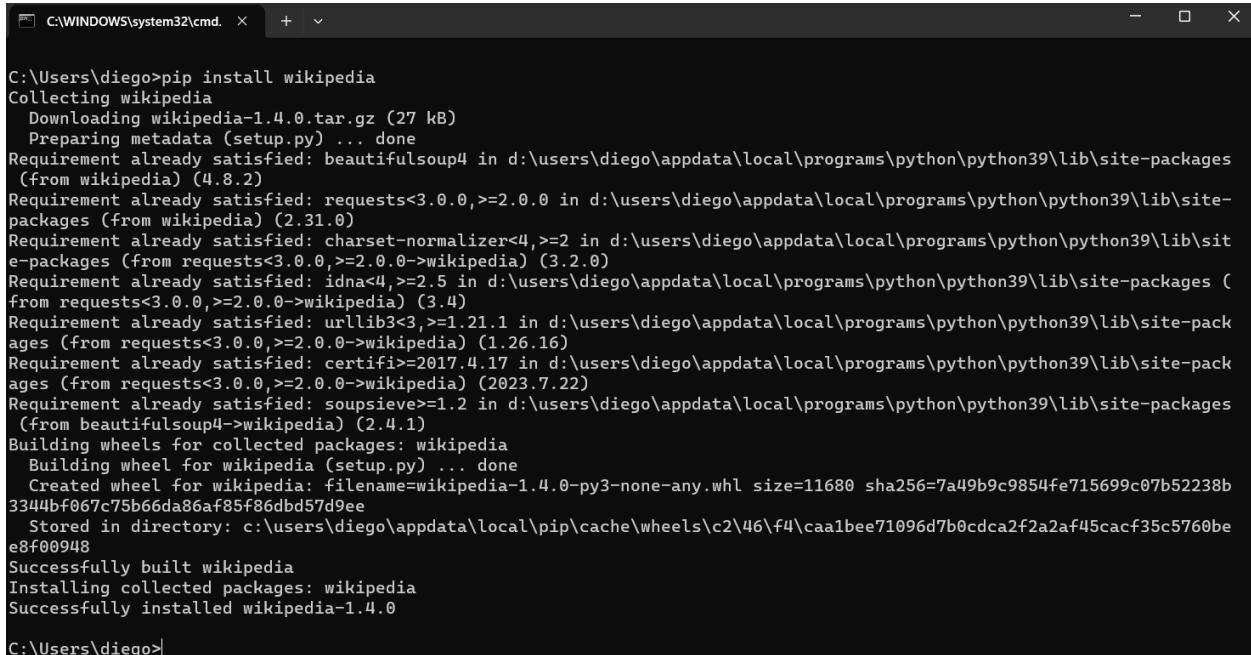


```
C:\Users\diego>pip install google-search-results
Collecting google-search-results
  Downloading google_search_results-2.4.2.tar.gz (18 kB)
    Preparing metadata (setup.py) ... done
Requirement already satisfied: requests in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from google-search-results) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from requests->google-search-results) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from requests->google-search-results) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from requests->google-search-results) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from requests->google-search-results) (2023.7.22)
Building wheels for collected packages: google-search-results
  Building wheel for google-search-results (setup.py) ... done
    Created wheel for google-search-results: filename=google_search_results-2.4.2-py3-none-any.whl size=32003 sha256=839ffca6385407d3e633ae3f38c3c65200bd22c7f7425856ca4e59af9f19bc590
    Stored in directory: c:\users\diego\appdata\local\pip\cache\wheels\68\8e\73\744b7d9d7ac618849d93081a20e1c0decccd2aef90901c9f5a9
Successfully built google-search-results
Installing collected packages: google-search-results
Successfully installed google-search-results-2.4.2

C:\Users\diego>
```

- **Agentes - SerpApi - Wikipedia:** Si se quiere proporcionar a un agente la habilidad de realizar búsquedas en del sitio de Wikipedia a través de la herramienta SERPAPI se debe ejecutar el siguiente comando en la consola CMD de Windows.

```
pip install wikipedia
```



```
C:\Users\diego>pip install wikipedia
Collecting wikipedia
  Downloading wikipedia-1.4.0.tar.gz (27 kB)
    Preparing metadata (setup.py) ... done
Requirement already satisfied: beautifulsoup4 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from wikipedia) (4.8.2)
Requirement already satisfied: requests<3.0.0,>=2.0.0 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from wikipedia) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from requests<3.0.0,>=2.0.0->wikipedia) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from requests<3.0.0,>=2.0.0->wikipedia) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from requests<3.0.0,>=2.0.0->wikipedia) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from requests<3.0.0,>=2.0.0->wikipedia) (2023.7.22)
Requirement already satisfied: soupsieve>=1.2 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from beautifulsoup4->wikipedia) (2.4.1)
Building wheels for collected packages: wikipedia
  Building wheel for wikipedia (setup.py) ... done
    Created wheel for wikipedia: filename=wikipedia-1.4.0-py3-none-any.whl size=11680 sha256=7a49b9c9854fe715699c07b52238b3344bf067c75b66da86af85f86dbd57d9ee
    Stored in directory: c:\users\diego\appdata\local\pip\cache\wheels\c2\46\f4\caa1bee71096d7b0cdca2f2a2af45cacf35c5760be8f00948
Successfully built wikipedia
Installing collected packages: wikipedia
Successfully installed wikipedia-1.4.0

C:\Users\diego>
```

- **Visualización de Datos - DataFrames - Pandas:** La herramienta llamada **pandas** se puede utilizar de forma opcional si se quiere observar en forma de tablas organizadas los datos devueltos en forma de diccionarios por los modelos al hacer consultas, esta permite integrar un tipo de dato llamado **DataFrame** que permite la organización, análisis y mejor visualización de datos en consola, para ello se debe ejecutar el siguiente comando en la consola CMD de Windows.

```
pip install pandas
```

```
C:\Users\diego>pip install pandas
Requirement already satisfied: pandas in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (1.3.3)
Requirement already satisfied: numpy>=1.17.3 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages
  (from pandas) (1.22.4)
Requirement already satisfied: python-dateutil>=2.7.3 in d:\users\diego\appdata\local\programs\python\python39\lib\site-
packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages
  (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in d:\users\diego\appdata\local\programs\python\python39\lib\site-packages (from
  python-dateutil>=2.7.3->pandas) (1.12.0)

C:\Users\diego>
```

## Código Python: LangChain - Modelos y Prompts

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera linea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior derecho de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#IMPORTACIÓN DE LIBRERÍAS:
#IMPORTACIÓN DE LLAVE: Cuando se quiera utilizar una API que utiliza un key, por seguridad es de buenas prácticas
#declararla en un archivo externo, además cabe mencionar que el nombre de dicho archivo y constante no pueden empezar
#con un número, sino cuando la quiera importar obtendré un error y se va accediendo a sus carpetas por medio de puntos:
# - Directorio normal:      carpeta1/carpeta2/carpeta3
# - Directorio paquetes:    carpeta1.carpeta2.carpeta3
#La parte del directorio se coloca después de la palabra reservada import y posteriormente se manda a llamar sus
#variables o constantes de igual manera a través de un punto.
import API_Keys.Llaves_ChatGPT_Bard
#ChatGPT API key
```

```

ApiKey = API_Keys.Llaves_ChatGPT_Bard.LlaveChatGPT

#1.-MODELOS (Models): El modelo se refiere a la red neuronal que se va a utilizar para procesar el texto de entrada y
#generar una respuesta, los Large Language Model (LLM) responden preguntas sin guardar un historial, mientras que los
#Chats si guardan las preguntas y respuestas realizadas para crear una conversación. Existen varios modelos dentro de
#una misma compañía, por ejemplo, OpenAI cuenta con gpt3, gpt4, gpt3.5 turbo, etc.
print("\n\n-----1.-MODELOS-----")

#OpenAI: Clase de la librería langchain que permite utilizar el LLM (Large Language Model) de OpenAI con Python, este
#puede resolver tareas sencillas, pero no se le proporciona roles y no guarda un historial de conversación.
from langchain.llms import OpenAI           #OpenAI: Modelo LLM.

#Cabe mencionar que, al utilizar la API en su modo gratuito, solo se podrán realizar 100 llamadas a la API por día,
#si se excede ese límite, se recibirá el error RateLimitError al intentar ejecutar el programa de Python, pero si se
#compra el servicio de la API, se cobrará a través de Tokens, que representan pedazos de palabras. Como máximo se
#pueden recibir o mandar a la vez 4096 tokens, que aproximadamente son 3,072 palabras.

#OpenAI(): En el constructor de la clase OpenAI perteneciente al paquete llms de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará text-davinci-003 que
#   pertenece a GPT-3.5.
# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.
# - prompt_length: La longitud del prompt.
# - max_tokens: El número máximo de tokens que se pueden generar.
# - stop_token: El token de parada.
# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#   demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#   función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.

#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:
#https://platform.openai.com/docs/models

openaiLLM = OpenAI(model_name = "text-davinci-003", openai_api_key = ApiKey, temperature = 0.5)           #LLM.

#A través de la instancia del objeto OpenAI se le puede mandar texto directamente al LLM convocado.
respuestaLLM = openaiLLM("Cuentame un chiste muy gracioso")

#print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
print("Respuesta LLM: ", respuestaLLM + "\n\n")

#ChatOpenAI: Clase de la librería langchain que permite utilizar el modelo de chat (ChatGPT) de OpenAI con Python, este
#puede contestar preguntas adoptando un rol y guardar un historial durante la conversación.
from langchain.chat_models import ChatOpenAI    #ChatOpenAI: Modelo de Chat.

from langchain.schema import HumanMessage      #HumanMessage: Clase para mandar una pregunta del usuario al Chat.

#ChatOpenAI(): En el constructor de la clase ChatOpenAI del paquete chat_models de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará gpt-3.5-turbo que
#   pertenece a GPT-3.5.

```

```

# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.
# - prompt_length: Longitud del prompt.
# - max_tokens: Número máximo de tokens que se pueden generar.
# - stop_token: El token de parada.
# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#   demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#   función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.
#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:
#https://platform.openai.com/docs/models
openaiChatGPT = ChatOpenAI(model_name = "gpt-3.5-turbo", openai_api_key = ApiKey, temperature = 0.7)      #Chat.

#HumanMessage: A través de un objeto de la clase ChatOpenAI se le mandará al modelo una lista que indique el rol y
#pregunta mandada al chat, de forma muy parecida a como se realiza con el método ChatCompletion.create() de la API
#openai; esto dentro de la librería langchain se realiza a través del constructor de un objeto
#HumanMessage(role = "", content = "") y el resultado de igual manera será una lista, por lo que se deberá transformar
#a un string con el método str() para poder imprimirla en consola.
respuestaChatGPT = openaiChatGPT([HumanMessage(role = "user", content="Hola como estás?")])
print("Respuesta Chat: " + str(respuestaChatGPT) + "\n\n")

#2.-PROMPTS: Es el texto que se le envía al modelo para generar una respuesta y en este es donde se utilizan las
#técnicas de Prompt Engineering, para ello la librería LangChain cuenta con diferentes clases que permiten utilizar
#dichas técnicas, dependiendo de si se está mandando el Prompt a un LLM o a un Chat.
print("\n\n-----2.-PROMPTS-----")
#PromptTemplate: Clase de la librería langchain que permite mandar instrucciones o preguntas personalizadas a un modelo
#LLM (Large Language Model) previamente invocado con Python, que no guarda un historial.
from langchain import PromptTemplate          #PromptTemplate: Pregunta mandada a un modelo LLM.
#El template se declara como un String que se encuentre entre dos comillas triples """Instrucción Prompt""", el punto
#de esto es declarar una instrucción que se puede aplicar a varias preguntas distintas y las variables de dicha
#pregunta se declaran dentro de dos llaves {variablePrompt}.
templateTech = """Eres un asistente virtual de {rolAsistenteVirtual} que proporciona un camino de aprendizaje dando
opciones cortas y concretas, pensando cada paso, paso a paso de los temas individuales que se deben aprender para
convertirse en un conocedor de un tema en específico.
Pregunta: Cuales son los pasos para aprender sobre {aprenderTema}.
Respuesta:"""
#PromptTemplate(): En el constructor de la clase PromptTemplate perteneciente a la librería langchain se indica:
# - template: Parámetro que indica la pregunta del prompt, esta se pudo haber guardado previamente en una variable.
# - input_variables: Indica a través de una lista todos los nombres de las variables incluidas en la plantilla del
#   prompt, que se declararon dentro del template entre llaves {}.
plantillaPrompt = PromptTemplate(template = templateTech, input_variables = ["rolAsistenteVirtual", "aprenderTema"])
#PromptTemplate().format(): Método que rellena las variables del template con valores de entrada.

```



```

promptMandadoLLM = plantillaPrompt.format(rolAsistenteVirtual = "Tecnología", aprenderTema = "IoT")
print("Prompt LLM: " + promptMandadoLLM + "\n\n")
#OpenAI(PromptTemplate().format()): Prompt mandado al modelo LLM.
respuestaPromptLLM = openaiLLM(promptMandadoLLM)
print("Respuesta LLM con Prompt: ", respuestaPromptLLM + "\n\n")
#OpenAI().get_num_tokens(PromptTemplate().format()): El método get_num_tokens() se aplica al objeto del Modelo
#utilizado y sirve para calcular el número de tokens enviados en un Prompt.
print("Número de Tokens del Promt =", openaiLLM.get_num_tokens(promptMandadoLLM), "del máximo que son 4096. \n\n")

#ChatPromptTemplate: Clase de la librería langchain que permite mandar instrucciones o preguntas personalizadas a un
#modelo de Chat, este puede contestar preguntas adoptando un rol a través de las siguientes clases:
# - SystemMessagePromptTemplate: Con esta clase se indica el rol que interpretará ChatGPT al responder las preguntas
#   del usuario.
# - HumanMessagePromptTemplate: Con esta clase se representa el rol del usuario que manda preguntas a ChatGPT.
# - AIMessagePromptTemplate: Con esta clase se representa el rol que es adoptado por ChatGPT siempre que responda la
#   pregunta de un usuario. Su mayor uso es el de permitir que el chat recuerde entradas y salidas anteriores.
from langchain.prompts import ChatPromptTemplate #ChatPromptTemplate: Instrucciones mandadas a un modelo de chat.
from langchain.prompts import SystemMessagePromptTemplate, HumanMessagePromptTemplate, AIMessagePromptTemplate

#SYSTEM - ROL DEL CHAT AL RESPONDER PREGUNTAS DEL USUARIO: Para ello se utiliza un objeto PromptTemplate.
#PromptTemplate(): En el constructor de la clase PromptTemplate perteneciente a la librería langchain se indica:
# - template: Parámetro que indica la pregunta del prompt.
# - input_variables: Indica a través de una lista todos los nombres de las variables incluidas en la plantilla del
#   prompt, que se declararon dentro del template entre llaves {}.
plantillaPromptSistema = PromptTemplate(
    template = "Eres un asistente virtual de viajes que me recomienda alternativas interesantes para viajar por {paisViaje}.",
    input_variables = ["paisViaje"]
)
#SystemMessagePromptTemplate(): Esta clase recibe como parámetro un objeto PromptTemplate, que previamente ya tiene
#diseñado el template que se mandará en el Prompt, indicándole al Chat el rol que está interpretando al responder.
promptSistema = SystemMessagePromptTemplate(prompt = plantillaPromptSistema)

#HUMAN - PREGUNTAS QUE EL USUARIO LE HACE AL MODELO: Para ello se utiliza un objeto PromptTemplate.
plantillaPromptHumano = PromptTemplate(
    template = "Mi viaje empieza el {fechaInicio} y termina el {fechaFin}. El vuelo es redondo, llegando y saliendo de
{ciudadVuelo}",
    input_variables = ["fechaInicio", "fechaFin", "ciudadVuelo"]
)
#HumanMessagePromptTemplate(): Esta clase recibe como parámetro un objeto PromptTemplate, que previamente ya tiene
#diseñado el template de la pregunta que hace el usuario al chat.
promptHumano = HumanMessagePromptTemplate(prompt = plantillaPromptHumano)

#ChatPromptTemplate.from_messages(): Método que sirve para unificar los templates previamente creados para el sistema
#(que le dice al modelo el rol que debe interpretar al responder mis preguntas), para el humano (que indica tal cual
#la pregunta realizada por el usuario) y de la AI (que es un rol adoptado por el modelo para guardar las preguntas y

```

```

#respuestas realizadas en un historial), creando así una conversación. El parámetro que recibe el método es una lista
#que incluye todas las plantillas de Prompt mencionadas previamente.

plantillaChatPrompt = ChatPromptTemplate.from_messages([promptSistema, promptHumano])

#ChatPromptTemplate().format_prompt().to_messages(): Método que rellena las variables del template mandado al Chat con
#valores de entrada para el prompt del sistema, del humano y de la AI, retornando una lista.

promptMandadoChat = plantillaChatPrompt.format_prompt(
    paisViaje = "Francia",
    fechaInicio = "30/11/2023",
    fechaFin = "30/11/2023",
    ciudadVuelo = "Madrid").to_messages()

#str(): Método que convierte un número, lista, diccionario, etc. en un string para que pueda ser impreso en consola.

print("Prompt Chat: " + str(promptMandadoChat) + "\n\n")

#ChatOpenAI(ChatPromptTemplate().format().to_messages()): Prompt mandado al modelo de Chat.

respuestaChat = openaiChatGPT(promptMandadoChat)

#Del diccionario retornado, el key de content es el que contiene la respuesta de la pregunta.

print("Respuesta de Chat con Prompt: ", respuestaChat.content + "\n\n")

#FewShotPromptSelector: Clase de la librería langchain que permite mandar ejemplos con el formato de respuesta que se
#espera obtener al mandar un Prompt, utilizando así la técnica Few-Shot Prompting, también llamada Example Selector.

from langchain import FewShotPromptTemplate      #FewShotPromptTemplate: Ejemplos de respuesta mandados al modelo.

#Primero se declara una lista con diccionarios anidados de preguntas y respuestas con el formato que se busca obtener.

ejemplos = [
    {"pregunta": "¿Cuales son los lugares más interesantes de la ciudad de México?", "respuesta": "El paseo en globo aerostático
sobre las pirámides de Teotihuacán"},

    {"pregunta": "¿Cuales son los lugares más interesantes de Puerto Vallarta?", "respuesta": "La cascada El Salto"},

    {"pregunta": "¿Cuales son los lugares más interesantes de Toluca?", "respuesta": "El nevado de Toluca"}]

]

#PromptTemplate(): En el constructor de la clase PromptTemplate perteneciente a la libreria langchain se indica:

# - template: Parámetro que indica la pregunta del prompt.

# - input_variables: Indica a través de una lista todos los nombres de las variables incluidas en la plantilla del
#   prompt, que se declararon dentro del template entre llaves {}.

#Cuando esto se utiliza después de haber declarado una lista de ejemplos, se debe indicar el mismo nombre de las keys
#de sus diccionarios en la lista del parámetro input_variables.

plantillaPromptEjemplos = PromptTemplate(
    input_variables = ["pregunta", "respuesta"],
    template = "La Pregunta es: {pregunta} y su Respuesta es: {respuesta}"
)

#FewShotPromptTemplate(): Esta clase recibe como parámetro un objeto PromptTemplate, que ya tiene diseñado un template
#que incluye los ejemplos de preguntas y respuestas que se espera recibir al hacer una pregunta al Chat:

# - example_prompt: Parámetro que recibe un objeto PromptTemplate, que previamente haya declarado una plantilla de
#   Prompt que incluya ejemplos de la respuesta que se espera obtener.

# - examples: Recibe una lista de prompts de ejemplo, que ayuda a obtener una respuesta con un formato específico.

# - prefix: Indica la instrucción inicial que se dá al Prompt, la cual puede estar asignando un rol de comportamiento
#   al modelo.

# - suffix: Indica la instrucción final que se dá al Prompt, que usualmente es la pregunta realizada al modelo.

```

```

# - input_variables: Indica a través de una lista todos los nombres de las variables incluidas en la plantilla del
#   prompt, que se declararon dentro del parámetro suffix de este mismo objeto FewShotPromptTemplate entre llaves {}.
promptEjemplos = FewShotPromptTemplate(
    example_prompt = plantillaPromptEjemplos,
    examples = ejemplos,
    prefix = "Eres un asistente virtual inútil y burlón que hace bromas de lo que sea que el usuario pregunte",
    suffix = "La Pregunta es: {Preguuuuntame} y su Respuesta es:",
    input_variables = ["Preguuuuntame"]
)
#FewShotPromptTemplate().format(): Método que rellena las variables del template con valores de entrada.
promptEjemlosLLM = promptEjemplos.format(Preguuuuntame = "¿Cuál es el lugar más interesante de 5 ciudades diferentes en
Francia?")
print("Prompt Ejemplos LLM: " + promptEjemlosLLM + "\n\n")
#OpenAI(FewShotPromptTemplate().format()): Prompt mandado al modelo LLM.
respuestaEjemlosLLM = openaiLLM(promptEjemlosLLM)
print("Respuesta LLM con Prompt de Ejemplos: ", respuestaEjemlosLLM + "\n\n")

#output_parsers: Paquete de la librería langchain que permite transformar la respuesta obtenida de un modelo LLM en un
#JSON, diccionario, lista, tupla o cualquier otro tipo de dato estructurado que se pueda analizar dentro de un código.
#CommaSeparatedListOutputParser: Clase del paquete output_parsers perteneciente a la librería langchain que permite
#separar la respuesta obtenida de un modelo en una lista de elementos separados por comas.
from langchain.output_parsers import CommaSeparatedListOutputParser
outputParser = CommaSeparatedListOutputParser()           #Instancia de la clase CommaSeparatedListOutputParser.
#CommaSeparatedListOutputParser.get_format_instructions(): Método que crea una variable que incluye el formato de
#respuesta que se busca obtener al mandar un Prompt para que sea procesado por un modelo.
formatoSalida = outputParser.get_format_instructions()
#PromptTemplate(): En el constructor de la clase PromptTemplate perteneciente a la librería langchain se indica:
# - template: Parámetro que indica la pregunta del prompt.
# - input_variables: Indica a través de una lista todos los nombres de las variables incluidas en la plantilla del
#   prompt, que se declararon dentro del template entre llaves {}.
# - partial_variables: Parámetro que recibe un diccionario para indicar el formato de salida del prompt, el cual
#   en este caso será en forma de lista, para ello se declara una key que indique el nombre del formato declarado
#   como variable {} en el parámetro template del prompt y como su value se utiliza la variable que utilizó el método
#   CommaSeparatedListOutputParser.get_format_instructions().
plantillaFormato = PromptTemplate(
    template = "Cuales son los ingredientes para preparar {platillo}\n{variableFormato}",
    input_variables = ["platillo"],
    partial_variables = {"variableFormato" : formatoSalida}
)
#PromptTemplate().format(): Método que rellena las variables del template con valores de entrada.
promptFormatoLLM = plantillaFormato.format(platillo = "un brownie Keto")
print("Prompt Formato Lista LLM: " + promptFormatoLLM + "\n\n")
#OpenAI(PromptTemplate().format()): Prompt mandado al modelo LLM.
respuestaFormatoLLM = openaiLLM(promptFormatoLLM)
print("Respuesta de LLM con Formato de Prompt: ", respuestaFormatoLLM + "\n\n")

```

```
#CommaSeparatedListOutputParser().parse(PromptTemplate().format()): El método .parse() permite utilizar el formato que
#instancia la clase CommaSeparatedListOutputParser aplicado a la plantilla creada con el objeto PromptTemplate después
#de haber sido mandada al modelo de lenguaje.

respuestaFormatizada = outputParser.parse(respuestaFormatoLLM)

print("Respuesta en forma de lista de un Prompt: ", str(respuestaFormatizada) + "\n\n")
```

## Resultado del Código Python

```
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/5.-Inteligencia Artificial/3.-Langchain - Modelos y Prompts.py"

-----1. -MODELOS-----
Resposta LLM:
¿Cómo se llama el pez que nunca dice la verdad?
Mentiroso

Resposta Chat: content='¡Hola! Estoy bien, ¿y tú?' additional_kwargs={}

-----2. -PROMPTS-----
Prompt LLM: Eres un asistente virtual de Tecnología que proporciona un camino de aprendizaje dando
opciones cortas y concretas, pensando cada paso, paso a paso de los temas individuales que se deben aprender para
convertirse en un conocedor de un tema en específico.
Pregunta: Cuales son los pasos para aprender sobre IoT.
Resposta:

Resposta LLM con Prompt:
Los pasos para aprender sobre IoT son los siguientes:
1. Comprender los conceptos básicos de IoT, como la conectividad, los sensores, la computación en la nube y el análisis de datos.
2. Aprender sobre los protocolos y tecnologías de IoT, como Bluetooth, Wi-Fi, Zigbee, MQTT y CoAP.
3. Comprender los conceptos de seguridad y privacidad relacionados con IoT.
4. Aprender sobre la arquitectura de IoT, los dispositivos y los protocolos de comunicación.
5. Aprender sobre las plataformas de IoT, como AWS IoT, Microsoft Azure IoT y Google Cloud IoT.
6. Desarrollar habilidades prácticas para diseñar, implementar y administrar sistemas de IoT.
7. Comprender cómo se desarrollan aplicaciones para IoT.
8. Aprender sobre las tendencias.

Número de Tokens del Prompt = 113 del máximo que son 4096.
```

## Código Python: LangChain - Memoria

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.

#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.

#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior derecho de Play se corre el programa.

#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#IMPORTACIÓN DE LIBRERÍAS:
#IMPORTACIÓN DE LLAVE: Cuando se quiera utilizar una API que utiliza un key, por seguridad es de buenas prácticas
#declararla en un archivo externo, además cabe mencionar que el nombre de dicho archivo y constante no pueden empezar
#con un número, sino cuando la quiera importar obtendrá un error y se va accediendo a sus carpetas por medio de puntos:
# - Directorio normal:      carpeta1/carpeta2/carpeta3
# - Directorio paquetes:   carpeta1.carpeta2.carpeta3
#La parte del directorio se coloca después de la palabra reservada import y posteriormente se manda a llamar sus
#variables o constantes de igual manera a través de un punto.

import API_Keys.Llaves_ChatGPT_Bard
#ChatGPT API key
```

```

ApiKey = API_Keys.Llaves_ChatGPT_Bard.LlaveChatGPT

#1.-MODELOS (Models): El modelo se refiere a la red neuronal que se va a utilizar para procesar el texto de entrada y
#generar una respuesta, los Large Language Model (LLM) responden preguntas sin guardar un historial, mientras que los
#Chats si guardan las preguntas y respuestas realizadas para crear una conversación. Existen varios modelos dentro de
#una misma compañía, por ejemplo, OpenAI cuenta con gpt3, gpt4, gpt3.5 turbo, etc.
print("\n\n-----1.-MODELOS-----")

#OpenAI: Clase de la librería langchain que permite utilizar el LLM (Large Language Model) de OpenAI con Python, este
#puede resolver tareas sencillas, pero no se le proporciona roles y no guarda un historial de conversación.
from langchain.llms import OpenAI          #OpenAI: Modelo LLM.

#Cabe mencionar que, al utilizar la API en su modo gratuito, solo se podrán realizar 100 llamadas a la API por día,
#si se excede ese límite, se recibirá el error RateLimitError al intentar ejecutar el programa de Python, pero si se
#compra el servicio de la API, se cobrará a través de Tokens, que representan pedazos de palabras; como máximo se
#pueden recibir o mandar a la vez 4096 tokens, que aproximadamente son 3,072 palabras.

#OpenAI(): En el constructor de la clase OpenAI perteneciente al paquete llms de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará text-davinci-003 que
#   pertenece a GPT-3.5.
# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.
# - prompt_length: La longitud del prompt.
# - max_tokens: El número máximo de tokens que se pueden generar.
# - stop_token: El token de parada.
# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#   demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#   función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.

#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:
#https://platform.openai.com/docs/models

openaiLLM = OpenAI(model_name = "text-davinci-003", openai_api_key = ApiKey, temperature = 0.5)           #LLM.

#ChatOpenAI: Clase de la librería langchain que permite utilizar el modelo de chat (ChatGPT) de OpenAI con Python, este
#puede contestar preguntas adoptando un rol y guardar un historial durante la conversación.
from langchain.chat_models import ChatOpenAI    #ChatOpenAI: Modelo de Chat.

#ChatOpenAI(): En el constructor de la clase ChatOpenAI del paquete chat_models de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará gpt-3.5-turbo que
#   pertenece a GPT-3.5.
# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.
# - prompt_length: Longitud del prompt.
# - max_tokens: Número máximo de tokens que se pueden generar.
# - stop_token: El token de parada.

```

```

# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
# demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
# función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.

#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:
#https://platform.openai.com/docs/models

openaiChatGPT = ChatOpenAI(model_name = "gpt-3.5-turbo", openai_api_key = ApiKey, temperature = 0.7)      #Chat.

#3.-MEMORIA (Memory): Esta clase permite almacenar las preguntas y respuestas hechas entre el LLM y el usuario,
#permitiendo así que se simule una conversación entre ambos.

print("\n-----3.-MEMORIA-----")

#MEMORIA DE HISTORIAL COMPLETO: Guarda todos los mensajes enviados y recibidos del chat.

# - ConversationBufferMemory: Con esta clase se crea una de las memorias más básicas para guardar todo el historial
# de preguntas y respuestas mandadas y recibidas de un modelo de chat creado con la clase ChatOpenAI.

# - ConversationChain: Clase para generar conversaciones entre dos o más participantes, indicando el rol de cada
# uno, ya sea el usuario (Human) o el modelo (AI). Esta clase se apoya de alguna otra que almacene el historial
# de la conversación y puede ser configurada para conectar diferentes modelos de lenguaje entre sí, resolviendo
# así tareas más complejas.

from langchain.memory import ConversationBufferMemory #ConversationBufferMemory: Memoria de historial de chat.
from langchain.chains import ConversationChain      #ConversationChain: Cadena de memoria del chat.

#ConversationBufferMemory(): Esta clase nos ayuda a gestionar todo el histórico de la conversación en un modelo de
#chat, no recibe nada como parámetro, solamente se utiliza para crear una instancia de la clase.

memoriaHistorial = ConversationBufferMemory()           #Instancia de la clase ConversationBufferMemory.

#ConversationChain(): La clase ConversationChain se utiliza para generar conversaciones de texto entre dos o más
#participantes y puede ser configurada para conectar diferentes modelos de lenguaje (LLM) o modelos de chat entre sí.
#Además, cabe mencionar que durante la conversación se estará indicando quién es el que está realizando cada
#interacción, ya sea el usuario (Human) o el modelo (AI).

# - llm: Indica el modelo de lenguaje o chat a utilizar.

# - memory: Recibe un objeto de memoria que gestione el historial de la conversación.

# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto
# ConversationChain imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt
# y el rol del usuario que está contestando cada cosa, pero cuando es False, no se imprimirá ninguna información.

chatbotHistorial = ConversationChain(llm = openaiChatGPT, memory = memoriaHistorial, verbose = True)

#ConversationChain.predict(): El método predict() genera una conversación de texto utilizando el objeto
#ConversationChain y a través de su parámetro input se introduce el Prompt mandado al chat.

chatbotHistorial.predict(input = "Hola como estás? Me llamo di_cero y soy la mente maestra detrás de la máquina.")

#Si se imprime en consola el resultado del objeto ConversationChain podremos observar que lo que retorna es lo que
#está almacenado en la instancia de la clase ConversationBufferMemory, ya que esta representa el historial guardado
#del chat.

print("1.-Respuesta de Chat con Memoria Buffer:\n" + str(chatbotHistorial) + "\n\n")
#ConversationBufferMemory.chat_memory.messages: Dentro de la variable de memoria del chat se encuentra el valor

```

```

#chat_memory, este almacena todos los roles del chat, ya sea el del usuario (Human) o el del modelo (AI), todo el
#historial es guardado dentro de una lista interna llamada messages.

print("\tHistorial del chat guardado en el objeto ConversationBufferMemory:\n" + str(memoriaHistorial.chat_memory.messages)
+ "\n\n")

#Debido al historial creado, esta nueva instrucción la contestará en función de lo que previamente le dije.

chatbotHistorial.predict(input = "Como me llamo?")

print("\tHistorial del chat con memoria Buffer:\n" + str(memoriaHistorial.chat_memory.messages) + "\n\n")

#MEMORIA DE VENTANA: Guarda solo los últimos mensajes enviados y recibidos del chat.

# - ConversationBufferWindowMemory: Con esta clase se crea un tipo de memoria que en vez de guardar todo el historial
# de preguntas y respuestas mandadas y recibidas de un modelo de chat creado con la clase ChatOpenAI, solo guarda
# los últimos mensajes mandados, a esto se le llama ventana de mensajes.

from langchain.memory import ConversationBufferWindowMemory #ConversationBufferWindowMemory: Memoria de mensajes.

#ConversationBufferMemory(): Esta clase nos ayuda a gestionar la ventana de mensajes que guarda parte de la conversación
#realizada sobre un modelo de chat. Por medio del parámetro k se indica el número de los últimos mensajes a guardar.

memoriaMensajes = ConversationBufferWindowMemory(k = 2)           #Instancia de la clase ConversationBufferWindowMemory.

#ConversationChain(): La clase ConversationChain se utiliza para generar conversaciones de texto entre dos o más
#participantes y puede ser configurada para conectar diferentes modelos de lenguaje (LLM) o modelos de chat entre sí.

#Además, cabe mencionar que durante la conversación se estará indicando quién es el que está realizando cada
#interacción, ya sea el usuario (Human) o el modelo (AI).

# - llm: Indica el modelo de lenguaje o chat a utilizar.

# - memory: Recibe un objeto de memoria que gestione el historial de la conversación.

# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto
# ConversationChain imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt
# y el rol del usuario que está contestando cada cosa, pero cuando es False, no se imprimirá ninguna información.

chatbotMensajes = ConversationChain(llm = openaiChatGPT, memory = memoriaMensajes, verbose = True)

#ConversationChain.predict(): El método predict() genera una conversación de texto utilizando el objeto
#ConversationChain y a través de su parámetro input se introduce el Prompt mandado al chat.

chatbotMensajes.predict(input = "Hello... I like trains, chu chu.")

#Si se imprime en consola el resultado del objeto ConversationChain podremos observar que lo que retorna es lo que
#está almacenado en la instancia de la clase ConversationBufferMemory, ya que esta representa el historial guardado
#del chat.

print("2.-Respuesta de Chat con Memoria de Ventana:\n" + str(chatbotMensajes) + "\n\n")

#ConversationBufferWindowMemory.chat_memory.messages: Dentro de la variable de memoria del chat se encuentra el valor
#chat_memory, este almacena todos los roles del chat, ya sea el del usuario (Human) o el del modelo (AI) y también las
#partes del historial de la conversación incluidas en la ventana de memoria en una lista interna llamada messages.

print("\tHistorial del chat guardado en el objeto ConversationBufferWindowMemory:\n"
+ str(memoriaMensajes.chat_memory.messages) + "\n\n")

chatbotMensajes.predict(input = "What do I like?")

print("\tHistorial del chat con memoria de ventana:\n" + str(memoriaMensajes.chat_memory.messages) + "\n\n")

#Debido al historial creado, esta nueva instrucción la contestará en función de lo que previamente le dije, pero
#ya con esto último se borrará el primer mensaje en el historial, porque solo indicamos que guarde k = 2, por lo
#que almacenará solo los últimos 2 mensajes, incluyendo la respuesta del modelo, debido a esta situación no sabrá
#como responder la última pregunta que le hice, entonces hay que tener cuidado porque al utilizar esta memoria,
#el chat empezará a olvidar información.

```

```

chatbotMensajes.predict(input = "What do I like?")

#Aunque la variable de la memoria si guarda todo el historial de la conversación, solamente manda los últimos dos
#mensajes de este al modelo, por eso es que olvida cosas.

print("\tHistorial del chat con memoria de ventana ya cuando olvidó información:\n" + str(memoriaMensajes.chat_memory.messages)
+ "\n\n")

#RESUMEN DE CONVERSACIÓN: Utiliza un segundo modelo para crear un resumen en inglés de la conversación entre el
#usuario y el modelo, reduciendo así el número de tokens utilizados y bajando el costo de la API.

# - ConversationSummaryMemory: Con esta clase se crea una cadena de modelos, donde en vez de guardar todo el
#   historial de la conversación de forma literal, cada vez que responda un prompt el chat, el historial será
#   mandado a otro modelo que realice un resumen de la conversación, con el peligro de que se borren algunos
#   datos importantes o que se haga un mal resumen, pero de esta forma se optimiza el uso de recursos y además
#   el costo de la API baja porque se reduce el número de tokens en uso.

from langchain.memory import ConversationSummaryMemory #ConversationSummaryMemory: Memoria de resumen de historial.

#ConversationSummaryMemory(): Esta clase crea un resumen en inglés de todo el historial de la conversación a través
#de un segundo modelo para seguir teniendo un contexto del tema tratado en el chat, reduciendo así el número de tokens
#utilizados para bajar el costo de uso de la API, para ello se le debe pasar el modelo auxiliar de chat que utiliza
#en su parámetro llm, que puede ser tanto de tipo Chat (ChatOpenAI) como de tipo LLM (OpenAI).

#ChatOpenAI(): En el constructor de la clase OpenAI perteneciente al paquete llms de la librería langchain se indica:

# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará gpt-3.5-turbo que
#   pertenece a GPT-3.5.

# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.

# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#   demasiado grande, puede responder con algo totalmente aleatorio.

#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:

#https://platform.openai.com/docs/models

modeloResumen = ChatOpenAI(model_name = "gpt-3.5-turbo", openai_api_key = ApiKey, temperature = 0.7)    #Chat.

memoriaResumen = ConversationSummaryMemory(llm = modeloResumen) #Instancia de la clase ConversationSummaryMemory.

#participantes y puede ser configurada para conectar diferentes modelos de lenguaje (LLM) o modelos de chat entre sí.
#Además, cabe mencionar que durante la conversación se estará indicando quién es el que está realizando cada
#interacción, ya sea el usuario (Human) o el modelo (AI).

# - llm: Indica el modelo de lenguaje o chat a utilizar.

# - memory: Recibe un objeto de memoria que gestione el historial de la conversación.

# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto
#   ConversationChain imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt
#   y el rol del usuario que está contestando cada cosa, pero cuando es False, no se imprimirá ninguna información.

chatbotResumen = ConversationChain(llm = openaiChatGPT, memory = memoriaResumen, verbose = True)

#ConversationChain.predict(): El método predict() genera una conversación de texto utilizando el objeto
#ConversationChain y a través de su parámetro input se introduce el Prompt mandado al chat.

chatbotResumen.predict(input = "Oye ChatGpt si quiero crear un asistente virtual como Jarvis de Ironman como le hago?")

#Si se imprime en consola el resultado del objeto ConversationChain podremos observar que lo que retorna es lo que
#está almacenado en la instancia de la clase ConversationBufferMemory, ya que esta representa el historial guardado
#del chat.

```

```

print("3.-Respuesta de Chat con Memoria de Resumen:\n" + str(chatbotResumen) + "\n\n")
#ConversationSummaryMemory.chat_memory.messages: Dentro de la variable de memoria del chat se encuentra el valor
#chat_memory, este almacena todos los roles del chat, ya sea el del usuario (Human) o el del modelo (AI), y un resumen
#de todo el historial, que se encuentra guardado dentro de una lista interna llamada messages.
chatbotResumen.predict(input = "Pero cuales son las mejores herramientas que puedo utilizar?")
print("\tHistorial del chat con memoria de resumen:\n" + str(memoriaResumen.chat_memory.messages) + "\n\n")
chatbotResumen.predict(input = "De donde puedo obtener un sintetizador de voz para que me pueda responder?")
print("\tHistorial del chat con memoria de resumen:\n" + str(memoriaResumen.chat_memory.messages) + "\n\n")

#PALABRAS CLAVE DEL HISTORIAL: Utiliza un segundo modelo para crear listas con las palabras clave de la conversación.
# - ConversationKGMemory: Con esta clase se crea una cadena de modelos, donde en vez de guardar todo el historial
# de la conversación de forma literal, cada vez que responda un prompt el chat, el historial será mandado a otro
# modelo que extraiga palabras clave de la conversación, creando así un gráfico de conocimiento en forma de lista
# llamado Knowledge Graph para después poder contestar con ese contexto.
#Esta clase a veces llega a tener problemas al acceder los elementos en memoria ya que tiene problema con la traducción
#de las instrucciones del prompt, que por default están en inglés.
from langchain.memory import ConversationKGMemory #ConversationKGMemory: Memoria de palabras clave del historial.
#ConversationKGMemory(): Esta clase crea una lista con palabras clave de todo el historial de la conversación a través
#de un segundo modelo para seguir teniendo un contexto del tema tratado en el chat, reduciendo así el número de tokens
#utilizados para bajar el costo de uso de la API, para ello se le debe pasar el modelo auxiliar de chat que utiliza
#en su parámetro llm, que puede ser tanto de tipo Chat (ChatOpenAI) como de tipo LLM (OpenAI).
#ChatOpenAI(): En el constructor de la clase OpenAI perteneciente al paquete llms de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará gpt-3.5-turbo que
#   pertenece a GPT-3.5.
# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.
# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#   demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#   función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.
#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:
#https://platform.openai.com/docs/models
modeloPalabrasClave = ChatOpenAI(model_name = "gpt-3.5-turbo", openai_api_key = ApiKey, temperature = 1)      #Chat.
memoriaGraph = ConversationKGMemory(llm = modeloPalabrasClave) #Instancia de la clase ConversationKGMemory.
#ConversationChain(): La clase ConversationChain se utiliza para generar conversaciones de texto entre dos o más
#participantes y puede ser configurada para conectar diferentes modelos de lenguaje (LLM) o modelos de chat entre sí.
#Además, cabe mencionar que durante la conversación se estará indicando quién es el que está realizando cada
#interacción, ya sea el usuario (Human) o el modelo (AI).
# - llm: Indica el modelo de lenguaje o chat a utilizar.
# - memory: Recibe un objeto de memoria que gestione el historial de la conversación.
# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto
#   ConversationChain imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt
#   y el rol del usuario que está contestando cada cosa, pero cuando es False, no se imprimirá ninguna información.
chatbotPalabrasClave = ConversationChain(llm = openaiChatGPT, memory = memoriaGraph, verbose = True)
#ConversationChain.predict(): El método predict() genera una conversación de texto utilizando el objeto

```

```

#ConversationChain y a través de su parámetro input se introduce el Prompt mandado al chat.
chatbotPalabrasClave.predict(input = "Holis ChatGpt mi nombre es Diego Cervantes y soy mecatrónico.")

#Si se imprime en consola el resultado del objeto ConversationChain podremos observar que lo que retorna es lo que
#está almacenado en la instancia de la clase ConversationBufferMemory, ya que esta representa el historial guardado
#del chat.

print("4.-Respuesta de Chat con Palabras Clave de Knowledge Graph:\n" + str(chatbotPalabrasClave) + "\n\n")

#ConversationKGMemory.memory.kg.get_triples(): Para obtener el Knowledge Graph que representa las palabras clave de
#la conversación se debe acceder a su valor memory.kg dentro del objeto ConversationChain, para ahí aplicar el
#método get_triples() y así obtener la lista de palabras clave que le dan contexto a la memoria de la conversación.
#Pero hay que tener muy en cuenta que esto puede tener errores cuando se utiliza un lenguaje que no sea el inglés.

chatbotPalabrasClave.predict(input = "Mi película favorita es Ironman, mis series favoritas son Daredevil y How I met your
mother")

print("\tKnowledge Graph del chat:\n" + str(chatbotPalabrasClave.memory.kg.get_triples()) + "\n\n")

chatbotPalabrasClave.predict(input = "Mis habilidades técnicas son de desarrollador web, móvil, sistemas embebidos, robótica,
etc.")

print("\tKnowledge Graph del chat:\n" + str(chatbotPalabrasClave.memory.kg.get_triples()) + "\n\n")

#ConversationKGMemory.chat_memory.messages: Dentro de la variable de memoria del chat se encuentra el valor
#chat_memory, este almacena todos los roles del chat, ya sea el del usuario (Human) o el del modelo (AI) y también la
#respuesta del chat en función del Knowledge Graph de la conversación guardada en la lista interna llamada messages.

chatbotPalabrasClave.predict(input = "Cuál es mi nombre, a que me dedico y cual es mi serie favorita?")

print("\tHistorial del chat con memoria de Knowledge Graph:\n" + str(memoriaGraph.chat_memory.messages) + "\n\n")

```

## Resultado del Código Python

```
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/5.-Inteligencia Artificial/4.-LangChain - Memoria.py"

-----1. MODELOS-----
-----3. MEMORIA-----

> Entering new ConversationChain chain...
Prompt after formatting:
The following is a friendly conversation between a human and an AI. The AI is talkative and provides lots of specific details from its context. If the AI does not know the answer to a question, it truthfully says it does not know.

Current conversation:

Human: Hola como estás? Me llamo di_cero y soy la mente maestra detrás de la máquina.
AI:

> Finished chain.
1. Respuesta de Chat con Memoria Buffer:
memory=ConversationBufferMemory(chat_memory=ChatMessageHistory(messages=[HumanMessage(content='Hola como estás? Me llamo di_cero y soy la mente maestra detrás de la máquina.', additional_kwargs={}), AIMessage(content='¡Hola di_cero! ¡Es un placer conocerte! Estoy programado para procesar información y responder preguntas, así que estoy aquí para ayudarte en lo que necesites. ¿En qué puedo ayudarte hoy?', additional_kwargs={})]), output_key=None, input_key=None, return_messages=False, human_prefix='Human', ai_prefix='AI', memory_key='history') callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x00000174FB45C820>, verbose=True, prompt=PromptTemplate(input_variables=['history', 'input']), output_parser=None, partial_variables={}, template='The following is a friendly conversation between a human and an AI. The AI is talkative and provides lots of specific details from its context. If the AI does not know the answer to a question, it truthfully says it does not know.\n\nCurrent conversation:{\n    "Human": {\n        "input": {\n            "history": {\n                "Human": "Hola como estás? Me llamo di_cero y soy la mente maestra detrás de la máquina.",\n                "AI": "¡Hola di_cero! ¡Es un placer conocerte! Estoy programado para procesar información y responder preguntas, así que estoy aquí para ayudarte en lo que necesites. ¿En qué puedo ayudarte hoy?"\n            }\n        }\n    }\n}', validate_template=True)
llm=ChatOpenAI( verbose=False, callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x00000174FB45C820>, client=<class 'openai.api_resources.chat_completion.ChatCompletion'>, model_name='gpt-3.5-turbo', temperature=0.7, model_kwargs={}, openai_api_key='sk-wYs5u8t8K4430xgd7LwqT3B1bkFJ8HcmTQBCiynk3V3liOE', openai_organization=None, request_timeout=60, max_retries=6, streaming=False, n=1, max_tokens=None) output_key='response' input_key='input'

Historial del chat guardado en el objeto ConversationBufferMemory:
[HumanMessage(content='Hola como estás? Me llamo di_cero y soy la mente maestra detrás de la máquina.', additional_kwargs={}), AIMessage(content='¡Hola di_cero! ¡Es un placer conocerte! Estoy programado para procesar información y responder preguntas, así que estoy aquí para ayudarte en lo que necesites. ¿En qué puedo ayudarte hoy?', additional_kwargs={})]

On How I met your mother: How I met your mother es una comedia muy divertida.
Human: Mis habilidades técnicas son de desarrollador web, móvil, sistemas embebidos, robótica, etc.
AI:

> Finished chain.
Knowledge Graph del chat:
[('Diego Cervantes', 'mecatrónico', 'es'), ('Ironman', 'mi película favorita', 'es'), ('Daredevil', 'mis series favoritas', 'son'), ('How I met your mother', 'una comedia muy divertida', 'es'), ('Mis habilidades técnicas', 'desarrollador web', 'son de'), ('Mis habilidades técnicas', 'desarrollador móvil', 'son de'), ('Mis habilidades técnicas', 'sistemas embebidos', 'son de'), ('Mis habilidades técnicas', 'robótica', 'son de')]

> Entering new ConversationChain chain...
Prompt after formatting:
The following is a friendly conversation between a human and an AI. The AI is talkative and provides lots of specific details from its context. If the AI does not know the answer to a question, it truthfully says it does not know.

Current conversation:

Human: ¿Cuál es mi nombre, a que me dedico y cual es mi serie favorita?
AI:

> Finished chain.
Historial del chat con memoria de Knowledge Graph:
[HumanMessage(content='Holí ChatGpt mi nombre es Diego Cervantes y soy mecatrónico.', additional_kwargs={}), AIMessage(content='¡Hola Diego!', additional_kwargs={}), HumanMessage(content='Mi película favorita es Ironman, mis series favoritas son Daredevil y How I met your mother', additional_kwargs={}), AIMessage(content='¡Genial! Me encanta Ironman, es una película muy emocionante. ¿Has visto todas las películas de la saga? También me gusta Daredevil, es una serie muy interesante con un enfoque más oscuro y realista de los superhéroes. How I Met Your Mother es una comedia muy divertida, ¿te interesa con alguno de los personajes?', additional_kwargs={}), HumanMessage(content='Mis habilidades técnicas son de desarrollador web, móvil, sistemas embebidos, robótica, etc.', additional_kwargs={}), AIMessage(content='Eso suena muy impresionante. Como mecatrónico, tengo conocimientos en sistemas embebidos y robótica, por lo que podríamos tener intereses similares en esas áreas. ¿Hay algún proyecto o tecnología específica en la que estés trabajando o te gustaría aprender más?', additional_kwargs={}), HumanMessage(content='¿Cuál es mi nombre, a que me dedico y cual es mi serie favorita?', additional_kwargs={}), AIMessage(content='Tu nombre es [nombre del usuario], te dedicas a [profesión del usuario] y tu serie favorita es [nombre de la serie favorita del usuario].', additional_kwargs={})]

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python> []
```

## Código Python: LangChain - Cadenas

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.

#La primera linea de código incluida en este programa se conoce como declaración de codificación o codificación de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando que Python interprete correctamente esos caracteres y evite posibles errores de codificación.

#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su historial y en Visual Studio Code con el botón superior derecho de Play se corre el programa.

#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:

#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#IMPORTACIÓN DE LIBRERÍAS:

#IMPORTACIÓN DE LLAVE: Cuando se quiera utilizar una API que utiliza un key, por seguridad es de buenas prácticas declararla en un archivo externo, además cabe mencionar que el nombre de dicho archivo y constante no pueden empezar con un número, sino cuando la quiera importar obtendré un error y se va accediendo a sus carpetas por medio de puntos:

# - Directorio normal:     carpeta1/carpeta2/carpeta3
# - Directorio paquetes:   carpeta1.carpeta2.carpeta3

#La parte del directorio se coloca después de la palabra reservada import y posteriormente se manda a llamar sus variables o constantes de igual manera a través de un punto.

import API_Keys.Llaves_ChatGPT_Bard

#ChatGPT API key

ApiKey = API_Keys.Llaves_ChatGPT_Bard.LlaveChatGPT

#1.-MODELOS (Models): El modelo se refiere a la red neuronal que se va a utilizar para procesar el texto de entrada y generar una respuesta, los Large Language Model (LLM) responden preguntas sin guardar un historial, mientras que los Chats si guardan las preguntas y respuestas realizadas para crear una conversación. Existen varios modelos dentro de una misma compañía, por ejemplo, OpenAI cuenta con gpt3, gpt4, gpt3.5 turbo, etc.

print("\n\n-----1.-MODELOS-----")

#OpenAI: Clase de la librería langchain que permite utilizar el LLM (Large Language Model) de OpenAI con Python, este puede resolver tareas sencillas, pero no se le proporciona roles y no guarda un historial de conversación.

from langchain.llms import OpenAI          #OpenAI: Modelo LLM.

#Cabe mencionar que, al utilizar la API en su modo gratuito, solo se podrán realizar 100 llamadas a la API por día, si se excede ese límite, se recibirá el error RateLimitError al intentar ejecutar el programa de Python, pero si se compra el servicio de la API, se cobrará a través de Tokens, que representan pedazos de palabras; como máximo se pueden recibir o mandar a la vez 4096 tokens, que aproximadamente son 3,072 palabras.

#OpenAI(): En el constructor de la clase OpenAI perteneciente al paquete llms de la librería langchain se indica:

# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará text-davinci-003 que pertenece a GPT-3.5.

# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
```

```

#     archivo.

# - prompt_length: La longitud del prompt.

# - max_tokens: El número máximo de tokens que se pueden generar.

# - stop_token: El token de parada.

# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#     demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#     función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.

#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:

#https://platform.openai.com/docs/models

openaiLLM = OpenAI(model_name = "text-davinci-003", openai_api_key = ApiKey, temperature = 0.5)           #LLM.

#ChatOpenAI: Clase de la librería langchain que permite utilizar el modelo de chat (ChatGPT) de OpenAI con Python, este
#puede contestar preguntas adoptando un rol y guardar un historial durante la conversación.

from langchain.chat_models import ChatOpenAI      #ChatOpenAI: Modelo de Chat.

#ChatOpenAI(): En el constructor de la clase ChatOpenAI del paquete chat_models de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará gpt-3.5-turbo que
#     pertenece a GPT-3.5.

# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#     archivo.

# - prompt_length: Longitud del prompt.

# - max_tokens: Número máximo de tokens que se pueden generar.

# - stop_token: El token de parada.

# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#     demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#     función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.

#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:

#https://platform.openai.com/docs/models

openaiChatGPT = ChatOpenAI(model_name = "gpt-3.5-turbo", openai_api_key = ApiKey, temperature = 0.7)      #Chat.

#2.-PROMPTS: Es el texto que se le envía al modelo para generar una respuesta y en este es donde se utilizan las
#técnicas de Prompt Engineering, para ello la librería LangChain cuenta con diferentes clases que permiten utilizar
#dichas técnicas, dependiendo de si se está mandando el Prompt a un LLM o a un Chat.

print("\n-----2.-PROMPTS-----")

#PromptTemplate: Clase de la librería langchain que permite mandar instrucciones o preguntas personalizadas a un modelo
#LLM (Large Language Model) previamente invocado con Python, que no guarda un historial.

from langchain import PromptTemplate      #PromptTemplate: Pregunta mandada a un modelo LLM.

```



```

#4.-CADENAS (Chains): Con esta herramienta se permite enlazar un modelo con un Prompt, también con ella se pueden
#conectar varios modelos entre sí, hasta cuando son de distintos tipos, permitiéndonos así realizar varias iteraciones
#entre modelos durante una consulta para obtener un mejor procesamiento final de los datos cuando este se busca aplicar
#a tareas muy complejas.

print("\n-----4.-CADENAS-----")

#CADENA SIMPLE: Permite encadenar un prompt con un modelo.

# - LLMChain: Con esta clase se conecta un prompt con un modelo de lenguaje, creando así una cadena individual.

from langchain import LLMChain      #LLMChain: Librería que crea una cadena, la cual incluye un prompt y un modelo.

#PromptTemplate(): En el constructor de la clase PromptTemplate perteneciente a la librería langchain se indica:

# - template: Parámetro que indica la plantilla del prompt previamente creada y almacenada en una variable.

# - input_variables: Indica a través de una lista todos los nombres de las variables incluidas en la plantilla del
#   prompt, que se declararon dentro de la variable template entre llaves {}.

plantillaPromptCadenaLLM = PromptTemplate(
    template = "Eres un asistente virtual experto en {tema} y respondes con una lista de 3 conceptos clave sobre el mismo.",
    input_variables = ["tema"]
)

#LLMChain(): Crea una cadena que unifica una plantilla de prompt con un modelo para que sea procesado.

# - llm: Indica el modelo de lenguaje o chat a utilizar.

# - prompt: Recibe un objeto tipo PromptTemplate que representa la plantilla del prompt mandada a la cadena LLM.

cadenaLLM = LLMChain(llm = openaiLLM, prompt = plantillaPromptCadenaLLM)

#LLMChain().predict(): El método predict lo que hace es rellenar las variables de la plantilla del prompt con valores
#de entrada. La cadena es una alternativa diferente a utilizar el objeto OpenAI(PromptTemplate().format()), pero
#básicamente hacen lo mismo.

respuestaChainLLM = cadenaLLM.predict(tema = "inteligencia artificial")

print("Cadena LLM con plantilla de Prompt: ", str(cadenaLLM), "\n\n")
print("Respuesta de LLMChain con plantilla de Prompt: ", str(respuestaChainLLM), "\n\n")

#CADENA SECUENCIAL: Permite encadenar un prompt con varios modelos de forma secuencial, uniendo así varias cadenas.

# - SequentialChain: Con esta clase se pueden conectar dos o más cadenas, osea modelos de lenguaje que se encuentran
#   enlazados con un prompt, pudiendo recibir así múltiples entradas y generar múltiples salidas, ya que la salida de
#   una cadena puede ser la entrada de otra cadena de forma secuencial. Además existe la clase SimpleSequentialChain
#   que hace lo mismo pero con la condición de que solo puede recibir 1 entrada y proporcionar 1 salida.

#Para que una cadena SequentialChain funcione, se deben declarar varias cadenas individuales LLMChain, cada una con su
#propio PromptTemplate.

from langchain.chains import SequentialChain      #SequentialChain: Cadena de cadenas con muchas entradas y salidas.

from langchain.chains import SimpleSequentialChain #SimpleSequentialChain: Cadena de cadenas con 1 entrada y 1 salida.

#PromptTemplate.from_template(): Método que crea un objeto PromptTemplate a partir de una plantilla de texto, la gran
#diferencia de usar este método en vez del constructor de la clase PromptTemplate() es que no se indica de forma
#explicativa las variables del prompt, solo se crea el objeto. La asignación de valor de las variables será realizada a
#través de su nombre con el método LLMChain().predict() después de haber creado la cadena individual.

promptTemplate_1 = """Eres un asistente virtual de viajes que enumera 3 recomendaciones de ciudades interesantes para
viajar por {paisViaje}. Si el país es Francia, no se debe incluir la ciudad de París."""

```

```

plantillaPromptCadenaLLM_1 = PromptTemplate.from_template(promptTemplate_1)
#LLMChain(): Crea una cadena que unifica una plantilla de prompt con un modelo para que sea procesado.
# - llm: Indica el modelo de lenguaje o chat a utilizar.
# - prompt: Recibe un objeto tipo PromptTemplate que representa la plantilla del prompt mandada a la cadena LLM.
# - output_key: Este parámetro representa el nombre que se le asigna a la salida de esta cadena individualmente, para
#   que así cuando se unifiquen varias a través del objeto SequentialChain, se pueda elegir cuales salidas se reciben
#   como entrada de otras cadenas o cuales hasta se consideran como la salida del mismo objeto SequentialChain.
cadenaLLM_1 = LLMChain(llm = openaiLLM, prompt = plantillaPromptCadenaLLM_1, output_key = "listaCiudades", verbose = True)
#PromptTemplate.from_template(): Método que crea un objeto PromptTemplate a partir de una plantilla de texto.
promptTemplate_2 = """Eres un asistente virtual de viajes que recibe una lista de 3 ciudades interesantes para
viajar por un país y debe devolver 5 lugares interesantes para visitar en cada ciudad.
La lista de ciudades es {listaCiudades}"""

plantillaPromptCadenaLLM_2 = PromptTemplate.from_template(promptTemplate_2)
#LLMChain(): Crea una cadena individual que unifica una plantilla de prompt con un modelo para que sea procesado.
cadenaLLM_2 = LLMChain(llm = openaiLLM, prompt = plantillaPromptCadenaLLM_2, output_key = "atraccionesCiudad", verbose = True)
#SequentialChain(): Objeto que crea una cadena de cadenas LLMChain, pudiendo recibir múltiples entradas y generar
#múltiples salidas, ya que la salida de una cadena puede ser la entrada de otra cadena de forma secuencial.
# - chains: Parámetro que recibe una lista con todas las cadenas individuales LLMChain que se conectarán, declarándolas
#   en el orden en el que se ejecutarán de forma secuencial.
# - input_variables: Indica a través de una lista todos los nombres de las variables de entrada incluidas en las
#   plantillas de los prompts pertenecientes a cada cadena, que se declararon entre llaves {}.
# - output_variables: Indica a través de una lista los nombres de las output_key que se quieren considerar como salida
#   de la cadena.
# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto
#   ConversationChain imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt
#   y el rol del usuario que está contestando cada cosa, pero cuando es False, no se imprimirá ninguna información.
cadenaSecuencial = SequentialChain(chains = [cadenaLLM_1, cadenaLLM_2],
                                      input_variables = ["paisViaje"],
                                      output_variables = ["listaCiudades", "atraccionesCiudad"],
                                      verbose = True)

#Para asignar valores a las entradas de una cadena se debe utilizar un diccionario, donde la key representa el nombre
#de la variable y el value su valor: {key: value} = {"nombreVariable": "Valor"}
respuestaSequentialChain = cadenaSecuencial({"paisViaje" : "Alemania"})

print("Cadena Secuencial LLM con plantilla de Prompt: ", str(cadenaSecuencial), "\n\n")
print("Respuesta de SequentialChain con plantillas de Prompt: ", str(respuestaSequentialChain), "\n\n")
print("Respuesta 1 de SequentialChain con plantillas de Prompt: ", str(respuestaSequentialChain["listaCiudades"]), "\n\n")
print("Respuesta 2 de SequentialChain con plantillas de Prompt: ", str(respuestaSequentialChain["atraccionesCiudad"]), "\n\n")

#SimpleSequentialChain(): Objeto que crea una cadena de cadenas LLMChain, pudiendo recibir una entrada y generar
#una salida.
# - chains: Parámetro que recibe una lista con todas las cadenas individuales LLMChain que se conectarán, declarándolas
#   en el orden en el que se ejecutarán de forma secuencial.
# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto
#   ConversationChain imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt
#   y el rol del usuario que está contestando cada cosa, pero cuando es False, no se imprimirá ninguna información.

```

```

cadenaSecuencialSimple = SimpleSequentialChain(chains = [cadenaLLM_1, cadenaLLM_2],
                                                verbose = True)

#SimpleSequentialChain().run(): El método run() proporciona el valor de la única entrada que puede tener la cadena de
#tipo SimpleSequentialChain y luego la ejecuta para que podamos ver su resultado.
respuestaSimpleSequentialChain = cadenaSecuencialSimple.run("Francia")
print("Cadena Secuencial Simple de LLM con plantilla de Prompt: ", str(cadenaSecuencialSimple), "\n\n")
print("Respuesta de SimpleSequentialChain con plantillas de Prompt: ", str(respuestaSimpleSequentialChain), "\n\n")

#PROCESAMIENTO DE LLM: Permite encadenar un prompt con varios modelos de forma secuencial, uniendo así varias cadenas.
# - TransformChain: Con esta clase se implementa una cadena de transformación, que se aplica a una entrada para
#   producir una salida con un formato personalizado. Las transformaciones pueden ser representadas por cualquier
#   función que tome una secuencia como entrada y devuelva una secuencia como salida.
#Por lo tanto, para que una cadena TransformChain funcione, se debe declarar una función propia que cambie el formato
#de la salida de otra cadena.
from langchain.chains import TransformChain    #TransformChain: Librería que crea una cadena de cadenas.
#Función propia que cambia el formato de cualquier salida proporcionada por un modelo de Chat o LLM.
def eliminarSaltosDeLinea(entrada):
    #Función que intercambia los saltos de línea por espacios.
    texto = entrada["texto"]      #Recibe una lista con un diccionario interno de key = texto.
    #lista.replace(): Método que reemplaza dentro de una lista un string por otro.
    return {"texto_limpio" : texto.replace("\n", "")}

#TransformChain(): Objeto que recibe un prompt, cambia su formato de una forma personalizada y lo retorna en una
#variable nueva.
# - input_variables: Indica a través de una lista los prompts de entrada.
# - output_variables: Indica a través de una lista el nombre de la variable de salida ya con el formato deseado.
# - transform: Recibe el nombre de la función propia que transforma el formato de la variable de entrada.
cadenaTransformarFormato = TransformChain(input_variables = ["texto"],
                                           output_variables = ["texto_limpio"],
                                           transform = eliminarSaltosDeLinea)

prompt_transform = """\n Este es un texto \ncon brincos \nde linea innecesarios\n."""
#TransformChain().run(): El método run() proporciona el valor de entrada del prompt y luego la ejecuta para que
#podamos ver su resultado.
respuestaTransformChain = cadenaTransformarFormato.run(prompt_transform)
print("Cadena TransformChain de LLM con plantilla de Prompt: ", str(cadenaTransformarFormato), "\n\n")
print("Respuesta de TransformChain con plantillas de Prompt: ", str(respuestaTransformChain), "\n\n")

```



## Resultado del Código Python

```
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/5.-Inteligencia Artificial/5.-LangChain - Cadenas.py"

-----1.-MODELOS-----
-----2.-PROMPTS-----
-----4.-CADENAS-----
Cadena LLM con plantilla de Prompt: memory=None callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x000000116913201C0> verbose=False prompt=PromptTemplate(input_variables=['tema'], output_parser=None, partial_variables={}, template='Eres un asistente virtual experto en {tema} y respondes con una lista de 3 conceptos clave sobre el mismo.', template_format='f-string', validate_template=True) llm=OpenAI(cache=None, verbose=False, callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x000000116913201C0>, client=<class 'openai.api_resources.completion.Completion'>, model_name='text-davinci-003', temperature=0.5, max_tokens=256, top_p=1, frequency_penalty=0, presence_penalty=0, n=1, best_of=1, model_kwargs={}, openai_api_key='sk-w5s5u8t8K4430xg07LwqT3B1bkF38HcMTQBCtjynk3V31i0E', openai_api_base=None, openai_organization=None, batch_size=20, request_timeout=None, logit_bias={}, max_retries=6, streaming=False, allowed_special=set(), disallowed_special='all') output_key='text'

Respuesta de LLMChain con plantilla de Prompt:

1. Machine Learning: es una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos que permiten a las computadoras "aprender" y mejorar su rendimiento sin ser explícitamente programadas.

2. Redes Neuronales: una red neuronal es un modelo computacional inspirado en la forma en que funciona el cerebro humano. Está compuesto por una gran cantidad de nodos interconectados que se comunican entre sí para procesar información.

3. Procesamiento del Lenguaje Natural: es una rama de la Inteligencia Artificial que se centra en el análisis automático de textos para extraer información útil. Esto incluye la comprensión del lenguaje, la interpretación de intenciones, la detección de patrones, etc.

> Entering new SequentialChain chain...

> Entering new LLMChain chain...
Prompt after formatting:
Eres un asistente virtual de viajes que enumera 3 recomendaciones de ciudades interesantes para

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

> Entering new LLMChain chain...
Prompt after formatting:
Eres un asistente virtual de viajes que recibe una lista de 3 ciudades interesantes para
viajar por un país y debe devolver 5 lugares interesantes para visitar en cada ciudad.
La lista de ciudades es

1. Burdeos: es una ciudad histórica con una cultura vibrante y una increíble gastronomía. Está rodeada de viñedos y ofrece una gran variedad de actividades al aire libre.

2. Marsella: una ciudad costera con una increíble vida nocturna y una cultura vibrante. Ofrece una gran variedad de restaurantes, bares y discotecas.

3. Niza: una ciudad con una vida nocturna animada, una cultura única y una increíble arquitectura. Es un destino ideal para los amantes de la playa y el sol.

> Finished chain.

Burdeos:
1. Catedral de Burdeos
2. Palacio Real de Burdeos
3. Jardín botánico de Burdeos
4. Barrio de La Victoire
5. La Cúpula de Burdeos

Marsella:
1. Vieux Port de Marsella
2. Basílica Notre-Dame de La Garde
3. Palais Longchamp
4. Parque Borély
5. Museo de La Marine

Niza:
1. Promenade des Anglais
2. Castillo de Niza
3. Musée Matisse
4. Parc Phoenix
5. Mercado de Los Flores
```

## Código Python: LangChain - Índices

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.

#La primera linea de código incluida en este programa se conoce como declaración de codificación o codificación de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando que Python interprete correctamente esos caracteres y evite posibles errores de codificación.

#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su historial y en Visual Studio Code con el botón superior derecho de Play se corre el programa.

#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:

#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#IMPORTACIÓN DE LIBRERÍAS:

#IMPORTACIÓN DE LLAVE: Cuando se quiera utilizar una API que utiliza un key, por seguridad es de buenas prácticas declararla en un archivo externo, además cabe mencionar que el nombre de dicho archivo y constante no pueden empezar con un número, sino cuando la quiera importar obtendré un error y se va accediendo a sus carpetas por medio de puntos:

# - Directorio normal:     carpeta1/carpeta2/carpeta3
# - Directorio paquetes:   carpeta1.carpeta2.carpeta3

#La parte del directorio se coloca después de la palabra reservada import y posteriormente se manda a llamar sus variables o constantes de igual manera a través de un punto.

import API_Keys.Llaves_ChatGPT_Bard

#ChatGPT API key

ApiKey = API_Keys.Llaves_ChatGPT_Bard.LlaveChatGPT

#1.-MODELOS (Models): El modelo se refiere a la red neuronal que se va a utilizar para procesar el texto de entrada y generar una respuesta, los Large Language Model (LLM) responden preguntas sin guardar un historial, mientras que los Chats si guardan las preguntas y respuestas realizadas para crear una conversación. Existen varios modelos dentro de una misma compañía, por ejemplo, OpenAI cuenta con gpt3, gpt4, gpt3.5 turbo, etc.

print("\n\n-----1.-MODELOS-----")

#OpenAI: Clase de la librería langchain que permite utilizar el LLM (Large Language Model) de OpenAI con Python, este puede resolver tareas sencillas, pero no se le proporciona roles y no guarda un historial de conversación.

from langchain.llms import OpenAI          #OpenAI: Modelo LLM.

#Cabe mencionar que, al utilizar la API en su modo gratuito, solo se podrán realizar 100 llamadas a la API por día, si se excede ese límite, se recibirá el error RateLimitError al intentar ejecutar el programa de Python, pero si se compra el servicio de la API, se cobrará a través de Tokens, que representan pedazos de palabras; como máximo se pueden recibir o mandar a la vez 4096 tokens, que aproximadamente son 3,072 palabras.

#OpenAI(): En el constructor de la clase OpenAI perteneciente al paquete llms de la librería langchain se indica:

# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará text-davinci-003 que pertenece a GPT-3.5.

# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
```

```

#     archivo.

# - prompt_length: La longitud del prompt.

# - max_tokens: El número máximo de tokens que se pueden generar.

# - stop_token: El token de parada.

# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#     demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#     función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.

#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:

#https://platform.openai.com/docs/models

openaiLLM = OpenAI(model_name = "text-davinci-003", openai_api_key = ApiKey, temperature = 0.5)           #LLM.

#ChatOpenAI: Clase de la librería langchain que permite utilizar el modelo de chat (ChatGPT) de OpenAI con Python, este
#puede contestar preguntas adoptando un rol y guardar un historial durante la conversación.

from langchain.chat_models import ChatOpenAI      #ChatOpenAI: Modelo de Chat.

#ChatOpenAI(): En el constructor de la clase ChatOpenAI del paquete chat_models de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará gpt-3.5-turbo que
#     pertenece a GPT-3.5.

# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#     archivo.

# - prompt_length: Longitud del prompt.

# - max_tokens: Número máximo de tokens que se pueden generar.

# - stop_token: El token de parada.

# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#     demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#     función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.

#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:

#https://platform.openai.com/docs/models

openaiChatGPT = ChatOpenAI(model_name = "gpt-3.5-turbo", openai_api_key = ApiKey, temperature = 0.7)      #Chat.

```

```

#5.0.-EMBEDDINGS: Los LLM convierten y asocian palabras a través de un vector llamado Embedding, el cual es un simple
#array de varias dimensiones que se encuentra en un espacio vectorial, cuya función es asociar de forma gráfica una
#palabra con otras parecidas y/o alejarla de otras que sean muy distintas, de esta manera es como el modelo entiende
#el lenguaje humano para realizar búsquedas, agrupaciones, clasificaciones, recomendaciones, etc.

print("\n-----5.-ÍNDICES-----")

# - OpenAIEMBEDDINGS: Clase que convierte cualquier texto que se le mande en un vector numérico.

from langchain.embeddings import OpenAIEMBEDDINGS

#OpenAIEMBEDDINGS(): El constructor de la clase OpenAIEMBEDDINGS recibe los siguientes parámetros de OpenAI:
# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#     archivo.

```

```

# - model: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará el más recomendado, que es
#   el text-embedding-ada-002, que puede recibir como máximo 8191 y da como salida un vector con tamaño de 1536.
#https://platform.openai.com/docs/guides/embeddings/what-are-embeddings
modeloEmbedding = OpenAIEmbeddings(openai_api_key = ApiKey, model = "text-embedding-ada-002")
promptEmbedding = "Soy di_cero!!!"

#OpenAIEmbeddings().embed_query(): El método .embed_query() toma una cadena de texto como entrada y devuelve un vector
#(o sea una lista) de números que representa su embedding.
respuestaEmbedding = modeloEmbedding.embed_query(promptEmbedding)

#len(lista): Devuelve el tamaño de la lista a la que se le aplique, en este caso devuelve el tamaño del embedding.
print("Embedding obtenido del Prompt: ", promptEmbedding, "=", str(respuestaEmbedding), "con tamaño de",
      str(len(respuestaEmbedding)), "\n\n")

#5.1.-ÍNDICES (Retrieval o Data connection): La forma en la que más se aprovechan los modelos de lenguaje es cuando se
#les da acceso a distintas fuentes de información, como lo puede ser un archivo PDF, Word, Excel, PowerPoint, etc.
#Los índices en LangChain son los que nos van a permitir enlazar un gran número de documentos para que sean procesados
#por el modelo, para ello la librería cuenta con diferentes clases que permiten realizar el enlace.

#CARGAR VARIOS DOCUMENTOS .TXT, .DOCX O .PDF A LA VEZ DE UNA CARPETA CON LA CLASE DirectoryLoader: La gran
#funcionalidad de esto radica cuando se quiere crear una base de datos de nuestros propios archivos, para hacerle
#preguntas sobre ellos.

#   - DirectoryLoader: Clase perteneciente al paquete document_loaders que permite cargar en el programa el contenido
#     de todos los archivos incluidos en una carpeta, para ello se debe proporcionar el path completo del directorio e
#     indicar el tipo de documento que se quiere importar.
#     Cabe mencionar que esta Clase se apoya en la librería unstructured al ejecutarse, por lo que para cada tipo de
#     documento que sea distinto a archivos con extensión .txt se deberá hacer una instalación adicional.
#       LEER DIRECTORIO CON ARCHIVOS .TXT:           Instalar con comando: pip install unstructured
#       LEER DIRECTORIO CON ARCHIVOS .PDF:            Instalar con comando: pip install unstructured[pdf]
#       LEER DIRECTORIO CON ARCHIVOS DE WORD .DOCX:   Instalar con comando: pip install unstructured[docx]
#   - CharacterTextSplitter: Esta clase perteneciente al paquete text_splitter de la librería langchain permite
#     dividir un texto muy grande en cachos limitados por cierto número de caracteres, ya que recordemos que el máximo
#     de tokens que admiten los modelos de OpenAI son de 4096 tokens, que aproximadamente son 3,072 palabras.

from langchain.document_loaders import DirectoryLoader          #DirectoryLoader: Carga varios archivos a la vez.
from langchain.text_splitter import CharacterTextSplitter       #CharacterTextSplitter: División por caracteres.
#DirectoryLoader(): El constructor recibe dos parámetros, el path global del directorio al que se quiere acceder y
#el parámetro glob indica el tipo de archivos que se recibirá de la carpeta indicada: glob = "**/*.extensiónArchivo".
cargarDirectorio = DirectoryLoader("C:/Users/diego/OneDrive/Documents/Aprendiendo/Python/5.-Inteligencia
Artificial/0.-Archivos_Ejercicios_Python/", glob = "**/*.txt")

#DirectoryLoader().load(): El método .load() carga en una variable todos los archivos contenidos en la carpeta
#indicada dentro del constructor del objeto DirectoryLoader.

documentosTxt = cargarDirectorio.load() #Carga todos los archivos txt de una carpeta.

print("Documentos txt originales extraídos de un directorio:\n", documentosTxt, "\n")

#DirectoryLoader().load()[0].page_content: El método page_content devuelve el contenido del documento.

#len(lista): Devuelve el tamaño de la lista a la que se le aplique, en este caso devuelve el número de palabras del
#documento.

print("El número de palabras del documento es de:\n", len(documentosTxt[0].page_content), "\n")

```

```

#CharacterTextSplitter(): El constructor del objeto lo que hace es indicar las características con las que se dividirá
#un texto grande que se quiere procesar a través de algún modelo de lenguaje, ya que estos están limitados en el
#número de tokens que pueden recibir, por ejemplo OpenAI solo admite 4096 tokens, que son aproximadamente 3,072
#palabras. Esta clase se utiliza cuando se busca que los trozos sean grandes.
# - chunk_size: Con este parámetro se indica el número de caracteres de cada cacho de texto, llamado chunk, este
#   número usualmente se encuentra entre 400 y 1000 caracteres. Pero es importante mencionar que, si el modelo
#   considera que al cortar cierta parte del texto con el chunk_size indicado hace que se pierda contexto, este número
#   será cambiado automáticamente por el método .split_documents().
# - chunk_overlap: Con este parámetro se indica los caracteres que se entrelazan con el cacho que tiene alado, para
#   que de esta forma no se pierda ninguna palabra.

dividirTexto = CharacterTextSplitter(chunk_size = 40, chunk_overlap = 0)      #División de texto por caracteres.

#CharacterTextSplitter().split_documents(): Método que divide el texto grande que recibe como parámetro en cachos
#cuyas características fueron descritas en el constructor de la clase CharacterTextSplitter.

documentosDivididos = dividirTexto.split_documents(documentosTxt)

print("\n\nDocumentos txt divididos con la clase CharacterTextSplitter:\n", documentosDivididos, "\n")
print("Cacho de documento txt dividido:\n", documentosDivididos[5], "\n\n\n")

#CARGAR UN DOCUMENTO PDF A LA VEZ Y DIVIDIRLO POR PÁGINAS CON LA CLASE PyPDFLoader DE LANGCHAIN:

# - PyPDFLoader: Clase del paquete document_loaders que permite leer documentos PDF con la librería langchain. Su
#   mayor ventaja es que de forma muy sencilla permite separar su contenido por página.
# - OnlinePDFLoader: Clase del paquete document_loaders que permite leer documentos PDF a través de un enlace. Se
#   carga su contenido con el método .load() como se hace con el objeto DirectoryLoader.

from langchain.document_loaders import PyPDFLoader          #PyPDFLoader: Carga 1 documento PDF a la vez.

#PyPDFLoader(): El constructor recibe como único parámetro el path global del archivo PDF al que se quiere acceder.

cargarDocumentoPDF = PyPDFLoader("C:/Users/diego/OneDrive/Documents/Aprendiendo/Python/5.-Inteligencia
Artificial/0.-Archivos_Ejercicios_Python/0.-Python - Conceptos Básicos.pdf")

#PyPDFLoader().load_and_split(): Método que divide el texto grande del archivo PDF recibido en el constructor de la
#clase PyPDFLoader por páginas, dentro un objeto document que incluye todo el contenido del PDF, esto después no
#podrá ser dividido de nuevo con alguna clase del paquete text_splitter perteneciente a la librería langchain, por
#lo que directamente con ella se creará el embedding de cada página.

paginasPDF = cargarDocumentoPDF.load_and_split()           #División del texto de un PDF por página.

print("Documento PDF dividido por páginas con la clase PyPDFLoader de langchain:\n", paginasPDF[2], "\n")
print("Número total de páginas en el documento:\n", len(paginasPDF), "\n")
print("Contenido de la segunda página del documento PyPDFLoader:\n", paginasPDF[2].page_content, "\n")
print("Número de caracteres de la segunda página del documento PyPDFLoader:\n", len(paginasPDF[2].page_content), "\n")
print("Tipo de dato del resultado obtenido con la clase PyPDFLoader:\n", type(paginasPDF), "\n\n\n")

#CARGAR UN DOCUMENTO PDF, DIVIDIRLO POR PÁGINAS Y LUEGO ESAS PÁGINAS PARTIRLAS EN CACHOS CON LA LIBRERÍA PyPDF2:

# - PdfReader: Esta clase de la librería open source PyPDF2 permite leer escribir y manipular archivos PDF con
#   python.
# - RecursiveCharacterTextSplitter: Esta clase permite dividir un texto grande en cachos para que pueda ser
#   procesado por un modelo de lenguaje, pero limita los cachos en los que se divide el texto en forma de tokens,
#   no de caracteres.

from PyPDF2 import PdfReader          #PdfReader: Carga 1 documento PDF a la vez.

from langchain.text_splitter import RecursiveCharacterTextSplitter #RecursiveCharacterTextSplitter: Cachos de tokens.

```

```

documentoPDF = PdfReader("C:/Users/diego/OneDrive/Documents/Aprendiendo/Python/5.-Inteligencia
Artificial/0.-Archivos_Ejercicios_Python/2.-Op-Amp Inversor con Filtro Pasa Altas.pdf")
#PdfReader().pages: Con el atributo .pages se accede al número de páginas del documento pdf leído con el constructor
#del objeto PdfReader.
#PdfReader().pages[i].extract_text(): Con el método .extract_text() se extrae todo el texto perteneciente a una
#página en específico del documento pdf ingresado a través del constructor del objeto PdfReader.
#len(): Método que devuelve el tamaño de la lista a la que se le aplique, en este caso devuelve el número de páginas
#del documento pdf.
textoPDF = "" #Variable textoPDF que después guardará todo el texto del pdf.
for i in range(len(documentoPDF.pages)): #Bucle for que lee todas las páginas del documento.
    pagina = documentoPDF.pages[i] #En la variable página se guarda el número de página.
    textoPagina = pagina.extract_text() #En la variable textoPagina se guarda el texto contenido en cada página.
    textoPDF += textoPagina #Concatenación del texto extraído de cada página.
print("Texto extraído de un documento PDF completo con la clase PdfReader de PyPDF2:\n", textoPDF, "\n\n")
print("Documento PDF dividido por páginas con la clase PdfReader de PyPDF2:\n", documentoPDF.pages[2].extract_text(), "\n")
print("Número total de páginas en el documento:\n", len(documentoPDF.pages), "\n")
print("Tipo de dato del resultado obtenido con la clase PyPDF2:\n", type(documentoPDF), "\n\n\n")
#RecursiveCharacterTextSplitter(): El constructor del objeto lo que hace es indicar las características con las que se
#separará en cachos un texto grande para que se pueda procesar con algún modelo de lenguaje, pero esto se hace en
#función de un número de tokens, no de caracteres. Esta clase se utiliza cuando se busca que los trozos sean pequeños.
# - chunk_size: Con este parámetro se indica el número de tokens de cada cache de texto, llamado chunk, este número
#   usualmente se encuentra entre 512 y 1000 tokens. Pero es importante mencionar que si el modelo considera que al
#   cortar cierta parte del texto con el chunk_size indicado hace que pierda contexto, este número será cambiado
#   automáticamente por el método .create_documents().
# - chunk_overlap: Con este parámetro se indica los caracteres que se entrelazan con el cache que tiene alado, para
#   que de esta forma no se pierda ninguna palabra.
# - length_function: Indica qué función se utilizará para contar el número de tokens de cada cache, puede ser la
#   función predefinida len(lista) o una función propia.
dividirPaginasPDF = RecursiveCharacterTextSplitter(chunk_size = 160, chunk_overlap = 10, length_function = len)
#CharacterTextSplitter().create_documents(): Método que divide el texto grande que recibe como parámetro en cachos
#cuyas características fueron descritas en el constructor de la clase RecursiveCharacterTextSplitter.
chunksPaginasPDF = dividirPaginasPDF.create_documents([textoPDF])
print("Documento pdf dividido con la clase RecursiveCharacterTextSplitter:\n", chunksPaginasPDF, "\n")
print("Cache de documento pdf dividido:\n", chunksPaginasPDF[10], "\n\n\n")
print("Cache de documento pdf dividido:\n", chunksPaginasPDF[11].page_content, "\n\n\n")

#5.2.-VECTOR STORES: Los LLM convierten y asocian palabras a través de un vector llamado Embedding, las clases FAISS
#y Chroma que representan Vector Stores ayudan a integrar bases de datos optimizadas para almacenar los vectores
#obtenidos después de procesar los Chunks de información.
#Embeddings: Es la herramienta que convierte los pedazos de palabras obtenidos de un documento (chunks) en vectores
#numéricos.
from langchain.embeddings import OpenAIEmbeddings
#OpenAIEmbeddings(): El constructor de la clase OpenAIEmbeddings recibe los siguientes parámetros de OpenAI:
# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.

```

```

# - model: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará el más recomendado, que es
#   el text-embedding-ada-002, que puede recibir como máximo 8191 y da como salida un vector con tamaño de 1536.
#https://platform.openai.com/docs/guides/embeddings/what-are-embeddings

modeloEmbedding = OpenAIEmbeddings(openai_api_key = ApiKey, model = "text-embedding-ada-002")

#Vectorstores: Representa una base de datos donde se asocian los chunks de palabras con sus embeddings
#correspondientes para que así se aliente a cualquier modelo de lenguaje con nuestra información, pudiendo así
#realizarle consultas sobre ella.

from langchain.vectorstores import FAISS    #FAISS: Vector store rápida, poco flexible y difícil de usar.
from langchain.vectorstores import Chroma   #Chroma: Vector store no tan rápida, flexible y fácil de usar.

#FAISS o Chroma.from_documents(): Método que crea la base de datos de vectores tipo FAISS o Chroma, esta recibe los
#chunks extraídos del texto y el modelo de embeddings declarado para su almacenamiento.

baseDeDatosFAISS_PyPDFLoader = Chroma.from_documents(paginasPDF, modeloEmbedding) #PyPDFLoader (langchain).
baseDeDatosFAISS_PdfReader = FAISS.from_documents(chunksPaginasPDF, modeloEmbedding) #PdfReader (PyPDF2).
baseDeDatosFAISS_DirectoryLoader = Chroma.from_documents(documentosDivididos, modeloEmbedding) #Dir..Load (langchain).

#5.3.-RETRIEVER: Este tipo de clases permiten extraer información de alguna fuente en específico, que suelen ser VECTOR
#STORES, sabiendo a dónde tiene que ir a buscar para obtener la respuesta solicitada a través de cadenas de búsqueda o
#algoritmos de búsqueda de proximidad.

#El concepto de cadena de búsqueda se refiere a un modelo de lenguaje que se ha entrenado con un conjunto de datos de
#preguntas y respuestas. Puede utilizarse para responder a preguntas sobre un documento personal ingresado a un VECTOR
#STORE. Para ello se puede utilizar alguna de las siguientes herramientas:

from langchain.chains.question_answering import load_qa_chain

# - load_qa_chain(): Método que carga una cadena de búsqueda de preguntas y respuestas preentrenada.
#   - llm: Indica el modelo de lenguaje o chat a utilizar.
#   - chain_type: Indica el tipo de cadena de búsqueda a utilizar:
#     - "stuff": Cadena de búsqueda predeterminada (LLM factual de Google AI) que es capaz de generar texto,
#       traducir idiomas, escribir diferentes tipos de contenido creativo y responder a sus preguntas de manera
#       informativa.

cadenaBusquedaPreentrenada = load_qa_chain(llm = openaiLLM, chain_type = "stuff")

#PREGUNTAS HECHAS A UN MODELO LLM:

from langchain.chains import RetrievalQA

# - RetrievalQA.from_chain_type(): Método que crea una cadena de búsqueda de preguntas y respuestas de un tipo
#   específico. Los tipos de cadenas de búsqueda disponibles son: "map_reduce", "refine" y "map_rerank".
#   - llm: Indica el modelo de lenguaje o chat a utilizar.
#   - chain_type: Indica el tipo de cadena de búsqueda a utilizar:
#     - "stuff": Cadena de búsqueda predeterminada (LLM factual de Google AI) que es capaz de generar texto,
#       traducir idiomas, escribir diferentes tipos de contenido creativo y responder a sus preguntas de manera
#       informativa.
#     - "map_reduce": Cadena de búsqueda que primero utiliza un retriever para recuperar documentos relevantes.
#     - "refine": Cadena de búsqueda que primero utiliza un retriever para recuperar documentos relevantes y
#       luego utiliza el LLM para refinar las respuestas.
#     - "map_rerank": Cadena de búsqueda que primero utiliza un retriever para recuperar documentos relevantes y
#       luego utiliza el LLM para reordenar las respuestas.
#   - retriever: Tipo de retriever que se utilizará para recuperar los documentos relevantes para contestar la
#     pregunta.

```

```

# - "faiss": Retriever predeterminado que utiliza el algoritmo de búsqueda vectorial faiss para recuperar los
# documentos relevantes.
#
# - "chroma": Retriever que utiliza el algoritmo de búsqueda vectorial chroma para recuperar los documentos
# relevantes.
#
# - "multi_query_retriever": Este retriever genera variantes de la pregunta de entrada y luego utiliza un
# algoritmo de búsqueda vectorial para recuperar los documentos relevantes para cada variante.
#
# - vectorStore.as_retriever(): El método as_retriever() convierte cualquier base de datos vectorial en un
# retriever para que de ahí se extraiga la información para responder la pregunta del usuario.
cadenaBusquedaPersonalizada = RetrievalQA.from_chain_type(llm = openaiLLM, chain_type = "map_reduce",
                                                               retriever = baseDeDatosFAISS_PdfReader.as_retriever())

#PREGUNTAS HECHAS A UN MODELO DE CHAT:
from langchain.chains import ConversationalRetrievalChain
# - ConversationalRetrievalChain.from_llm(): Método que crea una cadena de búsqueda de preguntas y respuestas a través
# de un modelo de lenguaje probabilístico. El modelo de lenguaje se puede utilizar para generar respuestas más
# naturales y conversacionales a través de la memoria de un historial de chat.
#
# - llm: Indica el modelo de lenguaje o chat a utilizar.
#
# - chain_type: Indica el tipo de cadena de búsqueda a utilizar:
#
#     - "stuff": Cadena de búsqueda predeterminada (LLM factual de Google AI) que es capaz de generar texto,
#       traducir idiomas, escribir diferentes tipos de contenido creativo y responder a sus preguntas de manera
#       informativa.
#
# - retriever: Tipo de retriever que se utilizará para recuperar los documentos relevantes para contestar la
# pregunta.
#
#     - "faiss": Retriever predeterminado que utiliza el algoritmo de búsqueda vectorial faiss para recuperar los
#       documentos relevantes.
#
#     - "chroma": Retriever que utiliza el algoritmo de búsqueda vectorial chroma para recuperar los documentos
#       relevantes.
#
#     - "multi_query_retriever": Este retriever genera variantes de la pregunta de entrada y luego utiliza un
#       algoritmo de búsqueda vectorial para recuperar los documentos relevantes para cada variante.
#
#     - vectorStore.as_retriever(): El método as_retriever() convierte cualquier base de datos vectorial en un
#       retriever para que de ahí se extraiga la información para responder la pregunta del usuario.
#
# - memory: Recibe un objeto de memoria que se utilizará para almacenar el historial de la cadena de búsqueda.
#
# - prompt: Plantilla de preguntas variables que se utilizará para generar respuestas.
#
# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto
#   ConversationChain imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt,
#   los tokens generados, y las puntuaciones de los tokens, pero cuando es False, no se imprimirá ninguna
#   información.
#
# - return_source_documents: Variable booleana que por default se encuentra con valor de False, pero cuando vale
#   True retorna la página de mis documentos en la cual se basó para contestar la pregunta.
cadenaBusquedaHistorialChat = ConversationalRetrievalChain.from_llm(llm = openaiLLM, chain_type = "map_reduce",
                                                               retriever =
baseDeDatosFAISS_DirectoryLoader.as_retriever(),
                                                               return_source_documents = True)

#El concepto de búsqueda de proximidad se refiere a un algoritmo de filtrado que se puede utilizar para encontrar los
#documentos más similares a una consulta realizada. Para ello se puede utilizar la siguiente herramienta:

```

```

# - FAISS o Chroma.from_documents().similarity_search(): El método similarity_search() utiliza el VECTOR STORE FAISS o
#   Chroma para realizar una búsqueda de proximidad.

#Pregunta hecha al documento ingresado al programa a través de la librería PyPDFLoader de langchain.
preguntaDocumento_PyPDFLoader = "Dime algunos usos del lenguaje de programación Python"
busqueda_PyPDFLoader = baseDeDatosFAISS_PyPDFLoader.similarity_search(preguntaDocumento_PyPDFLoader)

#Pregunta hecha al documento ingresado al programa a través de la librería PdfReader de PyPDF2.
preguntaDocumento_PdfReader = "Cómo decae la amplitud al modificar la frecuencia en un filtro pasa bajas?"
busqueda_PdfReader = baseDeDatosFAISS_PdfReader.similarity_search(preguntaDocumento_PdfReader)

#Pregunta hecha a los documentos ingresados al programa a través de la librería DirectoryLoader de langchain.
historialChat = []
preguntaDocumento_DirectoryLoader = "Cuales son las variables utilizadas para calcular la matriz de rigidez de un elemento?"
busquedaDirectoryLoader = baseDeDatosFAISS_DirectoryLoader.similarity_search(preguntaDocumento_DirectoryLoader)

#load_qa_chain().run(): El método run() se aplica a la cadena de búsqueda load_qa_chain, recibiendo como parámetros
#el resultado del método similarity_search() y la pregunta realizada para que así se busque en la VECTOR STORE y se
#obtenga el resultado de la pregunta realizada en base a la información del documento txt, word, pdf, etc.
resultado_PyPDFLoader = cadenaBusquedaPreentrenada.run(
    input_documents = busqueda_PyPDFLoader,
    question = preguntaDocumento_PyPDFLoader)

print("Cadena de Búsqueda load_qa_chain con base de datos vectorial FAISS al documento cargado con PyPDFLoader (langchain):\n",
      str(resultado_PyPDFLoader), "\n\n")

#RetrievalQA.from_chain_type()( {"query": pregunta}): A través de un diccionario con la key "query" se realiza una
#pregunta a una cadena de búsqueda RetrievalQA.
resultado_PdfReader = cadenaBusquedaPersonalizada( {"query" : preguntaDocumento_PdfReader})

print("Cadena de Búsqueda RetrievalQA con base de datos vectorial Chroma al documento cargado con PdfReader (PyPDF2):\n",
      str(resultado_PdfReader), "\n\n")

#ConversationalRetrievalChain.from_llm()( {"query": pregunta, "chat_history": listaHistorial}): A través de un
#diccionario con la key "question" y "chat_history" se realiza una pregunta a una cadena de búsqueda de chat
#ConversationalRetrievalChain y luego se almacena en una lista el historial de preguntas y respuestas.
resultado_DirectoryLoader = cadenaBusquedaHistorialChat( {"question" : preguntaDocumento_DirectoryLoader, "chat_history" :
historialChat})

print("Cadena de Búsqueda ConversationalRetrievalChain con base de datos vectorial FAISS al documento cargado con
DirectoryLoader (langchain):\n",
      str(resultado_DirectoryLoader), "\n\n")

print("Fuente de donde le cadena de búsqueda extrajo la respuesta de la pregunta hecha a los documentos DirectoryLoader:\n",
      str(resultado_DirectoryLoader['source_documents']), "\n\n",
      str(resultado_DirectoryLoader['source_documents'][0]), "\n\n",
      str(resultado_DirectoryLoader['source_documents'][0].page_content), "\n\n\n")

```



## Resultado del Código Python

```
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/5.-Inteligencia Artificial/6.-LangChain - Indices.py"

-----1.-MODELOS-----
-----5.-ÍNDICES-----
Embedding obtenido del Prompt: Soy di_cero!!! = [-0.0294668494835258, -0.006183130560992988, -0.0022461506449031866, -0.023088108229961764, -0.017345919352515605, -0.014269272696379934, -0.010098560369943266, -0.02161609134777245, -0.0044657786183207844, -0.03684018766821942, 0.016987861191984234, -0.02648302880656976, 0.001413168543283294, -0.0096609466016445, -0.00739109300236446, -0.0009863171136639684, 0.0076054162117810764, -0.004449201541188641, 0.023074846195726997, -0.0011893116958386, 0.010207959364414198, 0.0070285451965862965, 0.00622291480105203, -0.006027309109315516, -0.006073723900830626, 0.03071673160466974, 0.0072473585169110216, -0.015118002840531934, 0.018499662314227793, -0.0183117746581348, -0.014773206714235331, -0.02351247283676447, -0.008984602579612746, -0.002277646579226813, 0.01905663981240911, 0.006948976716468214, 0.007996627595254104, -0.015701505338505412, -0.007512586628750858, 0.02266374362354708, -0.014163182010437578, -0.0270134840989268, -0.018831196406289632, 0.006763317084746461, 0.012439198584986676, 0.0033650819369024014, -0.02997300240709623, -0.03201303444948595, -0.013990783947289726, 0.006869408236350132, -0.014242750905005566, 0.00341149672417511, 0.037715440951833, -0.013181838043196318, -0.02202719578602423, -0.010052146044089471, -0.02051539466035886, 0.023074846195726997, -0.0009979208115427458, -0.0258763196113307, 0.01225780194508553, 0.01384498088996525, 0.016576758619804316, 0.011922004395541771, -0.0012631489233825813, 0.0019892108387987656, 0.016178916219213905, -0.023711394036671652, 0.03264958229043061, -0.014163182010437578, -0.0070948521081309265, 0.01345369670649222, 0.015741289578564452, -0.02089951254239384, 0.0369728005598606, -0.01246949476107197, 0.0205220220707464019, 0.0245203487208724, 0.015608674824152563, 0.009568104146263596, 0.016709371511570946, 0.02092649909518404, -0.019069901846643877, -0.03440088853028414, -0.005576421365753927, 0.01704090746627804, 0.03447965733314649, 0.000658116881620449, -0.014693638234117248, 0.0386967830451143, -0.00040695490884205048, 0.02288918702966656, -0.00742638597176707, 0.002123482468998029, 0.0034148119981455456, -0.013765339609847167, -0.021960889336719105, -0.022358731173309515, 0.003925376334626353, -0.007293773308426132, 0.017836590404813606, 0.017345919352515605, 0.011192626971566895, -0.012173970938808155, -0.01237289213910336, 0.018260567837387548, 0.018446616039933986, -0.046070123911335914, 0.01339402034640366, -0.008089458109606953, 0.001359294131386821, 0.010343896827414897, -0.01239278425913288, -0.0375563039892816, 0.0026738309927027205, 0.003878961310280586, 0.017093951877926592, -0.014773206714235331, 0.0122213755178867197, -0.012963024722781593, -0.001575620820292528, -0.006285906666999283, -0.02206698002661463, 0.014644856348628, 0.0014769890972845983, 0.00037401308054361145, 0.009853224843794266, 0.0064549892242888925, 0.0036435713784850315, 0.013637497209256757, -0.01365924799258182, 0.011389417154744718, -0.004346425898143661, -0.003388289326659596, 0.01303596218572291, 0.020276689220004616, -0.010622386507828183, -0.0133343439863151, 0.013592941546698866, 0.02274331210366516, 0.007578893540295488, -0.015741289578564452, -0.02456012511214628, 0.021602831176187737, 0.011822543795394169, 0.001121914917739117, -0.006153923280948708, -0.0038723305259938602, -0.0007559001536681295, 0.03617711575880524, -0.006358844132699982, -0.007996627595254104, -0.010629016593622937, 0.029970778034885348, 0.0015631882452740776, -0.01864553724022919, 0.007804337412076282, -0.0479001989540518, -0.0051785789651635165, 0.0394659347386825615, 0.011835804898306306, -0.007406495477147186, -0.013805123849906209, -0.01892402692064248, 0.004117666517804175, 0.021907843062425297, -0.01887808046434867, 0.03063384880165428, 0.003878961310280586, 0.00858079442465842, 0.005483591317062393, -0.0016195492132192763, -0.02901595627346836, -0.026469766772334993, -0.00964171059730829, -0.005506798945650606, 0.026986961893012526, 0.02438772611767535, -0.04705146974122243, -0.030156437200945786, -0.0020986174989611286, 0.002211339434851526, 0.014415148553703962, -0.008096088195401707, 0.012094402458690073, 0.0361505916903357, 0.03694627649151652, -0.020860192649308724, -0.6416398660443243, -0.05195818771478347, 0.020197120739886533, 0.005927848509167913, 0.019825802407765655, 0.0299442
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

Cadena de Búsqueda load\_qa\_chain con base de datos vectorial FAISS al documento cargado con PyPDFLoader (langchain):

Python es un lenguaje de programación versátil y fácil de usar para inteligencia artificial, operaciones matemáticas complejas, visión artificial, análisis de datos, manejo de Backend, creación de APIs, creación de interfaces web con Jinja 2, manejo de contenedores con Docker, distribuciones de Linux, TDD con frameworks, y pruebas unitarias y de integración.

Cadena de Búsqueda RetrievalQA con base de datos vectorial Chroma al documento cargado con PdfReader (PyPDF2):

{'query': '¿Cómo decrea la amplitud al modificar la frecuencia en un filtro pasa bajas?', 'result': 'La amplitud decrea gradualmente a medida que se reduce la frecuencia del filtro pasa bajas desde la frecuencia de corte hasta una señal CD o muy parecida.'}

Cadena de Búsqueda ConversationalRetrievalChain con base de datos vectorial FAISS al documento cargado con DirectoryLoader (langchain):

```
{'question': '¿Cuáles son las variables utilizadas para calcular la matriz de rigidez de un elemento?', 'chat_history': [], 'answer': 'Las variables utilizadas para calcular la matriz de rigidez de un elemento son la longitud del elemento, el ángulo de giro, el módulo de elasticidad y la sección transversal del elemento.', 'source_documents': [Document(page_content='Elasticidad y/o Momentos de inercia.', metadata={'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\5.-Inteligencia Artificial\\\\0.-Archivos Ejercicios_Python\\\\a_MatrizRigidezViga2D.txt'}), Document(page_content='Módulo de elasticidad E, momento de inercia I y longitud L.', metadata={'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\5.-Inteligencia Artificial\\\\0.-Archivos Ejercicios_Python\\\\a_MatrizRigidezViga2D.txt'}), Document(page_content='Matriz de rigidez en estructura 2D', metadata={'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\5.-Inteligencia Artificial\\\\0.-Matriz de rigidez en estructura 2D', 'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\a_MatrizRigidezViga2D.txt'})]}
```

Fuente de donde le la cadena de búsqueda extrajo la respuesta de la pregunta hecha a los documentos DirectoryLoader:

```
[Document(page_content='Elasticidad y/o Momentos de inercia.', metadata={'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\5.-Inteligencia Artificial\\\\0.-Archivos Ejercicios_Python\\\\a_MatrizRigidezViga2D.txt'}), Document(page_content='Módulo de elasticidad E, momento de inercia I y longitud L.', metadata={'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\5.-Inteligencia Artificial\\\\0.-Archivos Ejercicios_Python\\\\a_MatrizRigidezViga2D.txt'}), Document(page_content='Matriz de rigidez en estructura 2D', metadata={'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\5.-Inteligencia Artificial\\\\0.-Matriz de rigidez en estructura 2D', 'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\a_MatrizRigidezViga2D.txt'})]
```

page\_content='Elasticidad y/o Momentos de inercia.' metadata={'source': 'C:\\\\Users\\\\diego\\\\OneDrive\\\\Documents\\\\Aprendiendo\\\\Python\\\\5.-Inteligencia Artificial\\\\0.-Archivos\_Ejercicios\_Python\\\\a\_MatrizRigidezViga2D.txt'}

%Elasticidad y/o Momentos de inercia.

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python>

## Código Python: LangChain - Agentes con Herramientas Pre-hechas y Personalizadas

Cuando se creen herramientas personalizadas que hereden de la clase BaseTool es muy útil utilizar la librería re, la cual permite utilizar expresiones regulares, estas son herramientas que sirven para buscar patrones de texto en strings. Hay muchas combinaciones posibles de caracteres y símbolos que se pueden usar en expresiones regulares para buscar patrones específicos:

- **Búsqueda de números en un string:**
  - `\d`: Expresión regular que busca un dígito del 0 al 9 en un string.
  - `\d+`: Expresión regular que coincide con uno o más dígitos del 0 al 9.
  - `\d{2,4}`: Expresión regular que coincide con una secuencia de dígitos de 2 al 4 específicamente, osea 234.
- **Búsqueda de letras en un string:**
  - `\w`: Expresión regular que busca una letra mayúscula o minúscula, un dígito del 0 al 9 o un guión bajo.
  - `\w+`: Expresión regular que coincide con uno o más caracteres de los descritos anteriormente.
  - `[A-Za-z]`: Expresión regular que busca solo una letra mayúscula o minúscula.
  - `\b`: Expresión regular que representa el inicio o final de una palabra en un string, usualmente se incluyen dos para indicar que se debe buscar un patrón en cada palabra. Por ejemplo, si se declara la expresión regular `\bHola\b`, esto extraerá todas las palabras Hola dentro de un string.
- **Búsqueda de palabras en un string (grupos de captura):**
  - `(patrón1|patrón2)`: Expresión regular que crea un grupo de captura para identificar uno o varios patrones específicos.
  - `|`: Compuerta OR utilizada para reconocer uno o varios patrones dentro de un grupo de captura.
- **Búsqueda de espacios en blanco, puntos y paréntesis en un string:**
  - `\s`: Expresión regular que coincide con un carácter de espacio vacío, como un espacio, tabulador (`\t`), salto de línea (`\n`), etc.
  - `\.`: Expresión regular que encuentra un punto en un string.
  - `\(`: Expresión regular que encuentra un paréntesis de apertura en un string.
  - `\)`: Expresión regular que encuentra un paréntesis de cierre en un string.
- **Búsqueda de repeticiones en un string:**
  - `*`: Encuentra cero o más repeticiones de la expresión regular que tenga a la izquierda.
  - `+`: Encuentra una o más repeticiones de la expresión regular que tenga a la izquierda.
  - `?`: Encuentra cero o una repetición de la expresión regular que tenga a la izquierda. Esto significa que la expresión regular que la precede es opcional y puede aparecer una vez o no aparecer en absoluto en la cadena que se está buscando.
  - `{2,4}`: Coincide con 2, 3 o 4 repeticiones de la expresión regular que tenga a la izquierda.
- **Anclaje de patrones:**
  - `^`: Cuando se coloca al principio de un patrón indica que la coincidencia debe encontrarse al comienzo de la línea de texto. Por ejemplo, si se declara la expresión regular `^Hola`, esto será cierto solo cuando la palabra Hola se encuentre al inicio del string.

- **\$:** Cuando se coloca al final de un patrón indica que la coincidencia debe encontrarse al final de la línea de texto. Por ejemplo, si se declara la expresión regular **\$mundo**, esto será cierto solo cuando la palabra mundo se encuentre al final del string.

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.

#La primera linea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.

#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior derecho de Play se corre el programa.

#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#IMPORTACIÓN DE LIBRERÍAS:

#IMPORTACIÓN DE LLAVE: Cuando se quiera utilizar una API que utiliza un key, por seguridad es de buenas prácticas
#declararla en un archivo externo, además cabe mencionar que el nombre de dicho archivo y constante no pueden empezar
#con un número, sino cuando la quiera importar obtendrá un error y se va accediendo a sus carpetas por medio de puntos:
# - Directorio normal:      carpeta1/carpeta2/carpeta3
# - Directorio paquetes:    carpeta1.carpeta2.carpeta3

#La parte del directorio se coloca después de la palabra reservada import y posteriormente se manda a llamar sus
#variables o constantes de igual manera a través de un punto.

import API_Keys.Llaves_ChatGPT_Bard

#ChatGPT API key

ApiKey = API_Keys.Llaves_ChatGPT_Bard.LlaveChatGPT

#1.-MODELOS (Models): El modelo se refiere a la red neuronal que se va a utilizar para procesar el texto de entrada y
#generar una respuesta, los Large Language Model (LLM) responden preguntas sin guardar un historial, mientras que los
#Chats si guardan las preguntas y respuestas realizadas para crear una conversación. Existen varios modelos dentro de
#una misma compañía, por ejemplo, OpenAI cuenta con gpt3, gpt4, gpt3.5 turbo, etc.

print("\n\n-----1.-MODELOS-----")

#OpenAI: Clase de la librería langchain que permite utilizar el LLM (Large Language Model) de OpenAI con Python, este
#puede resolver tareas sencillas, pero no se le proporciona roles y no guarda un historial de conversación.

from langchain.llms import OpenAI          #OpenAI: Modelo LLM.

#Cabe mencionar que, al utilizar la API en su modo gratuito, solo se podrán realizar 100 llamadas a la API por día,
#si se excede ese límite, se recibirá el error RateLimitError al intentar ejecutar el programa de Python, pero si se
#compra el servicio de la API, se cobrará a través de Tokens, que representan pedazos de palabras; como máximo se
#pueden recibir o mandar a la vez 4096 tokens, que aproximadamente son 3,072 palabras.
```

```

#OpenAI(): En el constructor de la clase OpenAI perteneciente al paquete llms de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará text-davinci-003 que
#   pertenece a GPT-3.5.
# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.
# - prompt_length: La longitud del prompt.
# - max_tokens: El número máximo de tokens que se pueden generar.
# - stop_token: El token de parada.
# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#   demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#   función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.
#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:
#https://platform.openai.com/docs/models
openaiLLM = OpenAI(model_name = "text-davinci-003", openai_api_key = ApiKey, temperature = 0)           #LLM.

#ChatOpenAI: Clase de la librería langchain que permite utilizar el modelo de chat (ChatGPT) de OpenAI con Python, este
#puede contestar preguntas adoptando un rol y guardar un historial durante la conversación.
from langchain.chat_models import ChatOpenAI    #ChatOpenAI: Modelo de Chat.
#ChatOpenAI(): En el constructor de la clase ChatOpenAI del paquete chat_models de la librería langchain se indica:
# - model_name: Parámetro que indica el modelo que se quiere utilizar, en este caso se utilizará gpt-3.5-turbo que
#   pertenece a GPT-3.5.
# - openai_api_key: Con este parámetro se proporciona la API key, que por buenas prácticas debe provenir de otro
#   archivo.
# - prompt_length: Longitud del prompt.
# - max_tokens: Número máximo de tokens que se pueden generar.
# - stop_token: El token de parada.
# - temperature: La temperatura es un valor entre 0 y 1 que indica la creatividad con la que contesta el LLM, si es
#   demasiado grande, puede responder con algo totalmente aleatorio y si es muy bajo responderá lo mismo siempre,
#   función que podría ser deseada cuando por ejemplo se contestan problemas matemáticos.
#Todos los modelos disponibles para usarse con OpenAI están enlistados en el siguiente enlace y cada uno es mejor
#en ciertas funciones que el otro:
#https://platform.openai.com/docs/models
openaiChatGPT = ChatOpenAI(model_name = "gpt-3.5-turbo", openai_api_key = ApiKey, temperature = 0)      #Chat.

#6.-AGENTES (Agents): Estos son modelos o cadenas a los cuales se les da acceso a una fuente o API para que puedan
#realizar alguna acción o proporcionar una respuesta que solucione una tarea en específico.
print("\n-----6.-AGENTES-----")
# - load_tools: Con este método perteneciente al paquete agents de la clase langchain se permite ingresar al
#   programa la herramienta que se busca integrar al agente a través de su nombre, que puede servir para darle la
#   habilidad de ejecutar comandos en consola, realizar búsquedas en internet, resolver o contestar preguntas acerca

```

```

# de operaciones matemáticas, obtener información meteorológica, de noticias, películas y/o crear una herramienta
# personalizada utilizando alguna otra API.

# - Ejecutar comandos en consola:
#     - terminal: Herramienta que permite ejecutar comandos en consola.
#     - python_repl: Esta herramienta permite ejecutar solamente scripts (programas) de Python a través de la
#         consola del sistema.

# - Realizar búsquedas en internet:
#     - serpapi: Tool que permite extraer información de internet para responder una realizada por el usuario.
#     - google-search: Esta herramienta de langchain permite utilizar específicamente el buscador de Google para
#         obtener la información que responde una pregunta realizada al agente.
#     - requests: Esta herramienta permite extraer información de la URL de un sitio web en específico para
#         responder una pregunta.

# - Resolver o contestar preguntas acerca de operaciones matemáticas: Cuando se usen estas tools es recomendable
# declarar que la temperatura del modelo sea de 0, para que siempre dé el mismo resultado.
#     - wolfram-alpha: Esta herramienta permite resolver problemas o contestar preguntas que tengan que ver con
#         matemáticas, ciencia, tecnología, etc.
#     - pal-math: Esta herramienta permite resolver problemas matemáticos a través de una instrucción, como
#         crear ecuaciones a través de un problema de la vida real y cosas por el estilo.
#     - llm-math: Esta herramienta permite resolver problemas matemáticos.

# - Obtener información meteorológica:
#     - open-meteo-api: Permite obtener información meteorológica a través de la herramienta OpenMeteo.

# - Obtener información de noticias recientes o películas:
#     - news-api: Obtiene información acerca de noticias actuales.
#     - tmdb-api: Obtiene información acerca de películas.

# - initialize_agent: Con esta función perteneciente al paquete agents de la clase langchain se indica el tipo de
# agente que se pretende crear.

from langchain.agents import load_tools          #load_tools: Método que permite anexar una Tool a un agente.
# - initialize_agent: Con este método perteneciente al paquete agents de la clase langchain se permite ingresar al
# programa el tipo de agente que se busca utilizar a través de su nombre. El mejor tipo de agente a elegir
# dependerá de las necesidades específicas del proyecto:
#     - Los agentes zero-shot-react-description y conversational-react-description son buenos al realizar tareas
#         generales, el primero utilizando un modelo LLM y el segundo un modelo de Chat.
#     - El agente react-docstore es mejor utilizarlo para interactuar con un almacén de documentos.
#     - El agente self-ask-with-search es una buena opción para realizar búsquedas en la web.

from langchain.agents import initialize_agent    #initialize_agent: Método que indica el tipo de agente que se creará.

#IMPORTACIÓN DE LIBRERÍAS:

#IMPORTACIÓN DE LLAVE: Cuando se quiera utilizar una API que utiliza un key, por seguridad es de buenas prácticas
#declararla en un archivo externo, además cabe mencionar que el nombre de dicho archivo y constante no pueden empezar
#con un número, sino cuando la quiera importar obtendré un error y se va accediendo a sus carpetas por medio de puntos:
# - Directorio normal:      carpeta1/carpeta2/carpeta3
# - Directorio paquetes:    carpeta1.carpeta2.carpeta3

#La parte del directorio se coloca después de la palabra reservada import y posteriormente se manda a llamar sus
#variables o constantes de igual manera a través de un punto.

import os #os: Librería que permite acceder a funciones y métodos relacionados con el sistema operativo.

import API_Keys.Llaves_ChatGPT_Bard

```

```

#SerpAPI API key
SerpApiKey = API_Keys.llaves_ChatGPT_Bard.LlaveSerpAPI

#os.environ: El método environ proveniente de la librería os permite acceder a las variables de entorno del sistema operativo, estas están organizadas en una forma de key:value, por lo que serán datos tipo diccionario o JSON, y su objetivo es proveer cierta información de configuración que afecte a múltiples programas o aplicaciones, evitando así que cada uno deba ser configurado por separado, pudiendo adaptarse automáticamente a diferentes entornos al leer las variables de entorno relevantes, que usualmente almacenan información sensible, como contraseñas o API keys.
#Para crear una nueva variable de entorno se utiliza la siguiente sintaxis, indicando su nombre en mayúsculas:
# os.environ['NOMBRE_VARIABLE'] = 'valorVariableDeEntorno'
os.environ["SERPAPI_API_KEY"] = SerpApiKey

#load_tools(): Método que sirve para cargar al programa las herramientas que se quiera integrar a un agente. El resultado se entrega en forma de diccionario y será recibido como parámetro del método initialize_agent(), además se debe indicar que modelo está utilizando, ya sea LLM o de Chat, dependiendo del tipo de agente al que se integre.
# - tool_names: A través de una lista se declaran los nombres de las tools que se quiera integrar al agente.
# - llm: Indica el modelo de lenguaje o chat a utilizar.

nombreHerramientas = ["serpapi", "llm-math", "wikipedia", "terminal"]
herramientasAgente = load_tools(tool_names = nombreHerramientas, llm = openaiLLM)
print("Las herramientas que se utilizaron son:\n", str(herramientasAgente), "\n")
print("Sus nombres y lo que hacen respectivamente es:\n", str(herramientasAgente[0].name), ":\t",
      str(herramientasAgente[0].description), "\n\n",
      str(herramientasAgente[1].name), ":\t",
      str(herramientasAgente[1].description), "\n\n",
      str(herramientasAgente[2].name), ":\t",
      str(herramientasAgente[2].description), "\n\n",
      str(herramientasAgente[3].name), ":\t",
      str(herramientasAgente[3].description), "\n")

#initialize_agent(): Método que sirve para anexar a un agente las herramientas previamente cargadas al programa con el método load_tools(), además en este se indica el LLM que se quiere utilizar y el tipo de agente que se pretende crear.
# - tools: Indica el diccionario que contiene el nombre de las herramientas que se quiere enlazar al agente.
# - llm: Indica el modelo de lenguaje o chat a utilizar.
# - agent: En este parámetro se indica el nombre del tipo de agente.
# - max_iterations: Este parámetro indica el máximo número de pensamientos (thoughts) que puede realizar el agente al responder una pregunta, esto es bueno declararlo porque así se evita que el agente se quede atorado en un bucle infinito buscando entre sus herramientas para contestar una pregunta que no puede responder.
# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto agente imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt, la herramienta que está utilizando para resolver cada parte del prompt, la respuesta dada por cada herramienta y la respuesta final del agente.

Agente = initialize_agent(tools = herramientasAgente, llm = openaiLLM,
                           agent = "zero-shot-react-description",
                           max_iterations = 3,
                           verbose = True)

#initialize_agent().run(): El método run() proporciona el valor de entrada del agente y luego lo ejecuta para que podamos ver su resultado.

```



```

resultadoAgente = Agente.run("Quien es más viejo, el presidente actual de USA o Robert Downey Jr? Toma la edad más grande y saca su raíz cuadrada.")

print("Respuesta de Agente de tipo Zero Shot enlazado con la herramienta serpapi: ", str(resultadoAgente), "\n\n\n")
resultadoAgente = Agente.run("En cual carpeta de mi ordenador se encuentra el archivo 17.-Vibracion en Estructuras.mph?")
print("Respuesta de Agente de tipo Zero Shot enlazado con la herramienta serpapi: ", str(resultadoAgente), "\n\n\n")

#UTILIZAR HERRAMIENTAS PERSONALIZADAS:

# - BaseTool: A través de una clase propia que herede de la clase Tool que pertenece al paquete tools de la librería
#   langchain se permite crear herramientas personalizadas, que pueden servir para darle al agente la habilidad de
#   ejecutar alguna acción en específico.

from langchain.tools import BaseTool          #BaseTool: Clase que permite crear herramientas personalizadas.
import math                                    #math: Librería que permite usar constantes o funciones matemáticas.
import random                                  #random: Librería que permite crear números aleatorios.

#BaseTool: La clase que herede de la clase BaseTool debe declarar los atributos name y description, además de indicar
#a través de su método _run() la acción específica que se realiza cuando el agente utilice la clase:
# - name: En este atributo se declara el nombre de la herramienta personalizada.
# - description: Este atributo es muy importante, ya que a través de él se le indicará al agente cuando es que debe
#   utilizar esta herramienta.
# - _run(): Este método es el que se va a ejecutar cuando el agente utilice esta herramienta.

#HERRAMIENTA PERSONALIZADA QUE CREA UN NÚMERO ALEATORIO ENTRE 0 Y 100:

class NumeroAleatorio(BaseTool):
    #name =           Nombre de la herramienta.
    name = "Número aleatorio"
    #description =  Instrucción que le indica al agente en qué situaciones deberá utilizar esta herramienta.
    description = "Usa esta herramienta cuando necesites un número aleatorio"
    #_run()       = Método que describe la acción que ejecuta esta herramienta en específico para contestar una
    #pregunta hecha al agente, ya sea acerca de matemáticas, búsquedas a internet, uso de APIs específicas, etc.
    #self: La instrucción self se utiliza para hacer referencia al objeto que se está manipulando cuando se instancia
    #la clase. Por eso es que a través de la misma nomenclatura que incluye un punto se accede a los distintos
    #atributos y/o métodos con un objeto desde fuera de la clase.
    def _run(self, operación: str):
        return "Número aleatorio = " + str(random.randint(0, 100))
    #_arun(): Este método se utiliza cuando se quiera que esta función se corra de forma asíncrona.
    async def _arun(self):
        raise NotImplementedError("Esta herramienta no tiene una ejecución asíncrona.")

#HERRAMIENTA PERSONALIZADA QUE CALCULA EL PERÍMETRO DE UN CÍRCULO A TRAVÉS DE UN SOLO PARÁMETRO QUE INDICA SU RADIO:

import re      #re: Librería que permite utilizar expresiones regulares para detectar patrones.
#Es de mucha utilidad utilizar la librería re, ya que esta permite identificar patrones en un string, y que lo que
#siempre recibirán este tipo de funciones es una instrucción en forma de string, y de esta se deben extraer los números
#que necesitamos para ejecutar la acción deseada y luego transformar eso a un formato de número entero o decimal.
class CircunferenciaCirculo(BaseTool):
    #name =           Nombre de la herramienta.
    name = "Calculadora de circunferencia"
    #description =  Instrucción que le indica al agente en qué situaciones deberá utilizar esta herramienta.

```

```

description = "Usa esta herramienta cuando necesites calcular la circunferencia de un círculo con su radio"
#_run()      = Método que describe la acción que ejecuta esta herramienta en específico para contestar una
#pregunta hecha al agente, ya sea acerca de matemáticas, búsquedas a internet, uso de APIs específicas, etc.
def _run(self, radio: float):
    #re.findall(r): El método findall() utilizado para encontrar todas las ocurrencias de una expresión regular en
    #una cadena de texto (string) recibe 3 parámetros, la expresión regular que se quiere encontrar, la cadena de
    #caracteres donde se buscará y modificadores opcionales que se pueden utilizar para personalizar el
    #comportamiento de la búsqueda.

    # - pattern: Parámetro que indica la expresión regular utilizada para extraer partes específicas del string.
    #   Las expresiones regulares más comunes son:
    #     - Búsqueda de números en un string:
    #       - \d: Expresión regular que busca un dígito del 0 al 9 en un string.
    #       - \d+: Expresión regular que coincide con uno o más dígitos del 0 al 9.
    #       - \d{2,4}: Expresión regular que coincide con una secuencia de dígitos de 2 al 4 específicamente,
    #                 osea 234.
    #     - Búsqueda de letras en un string:
    #       - \w: Expresión regular que busca una letra mayúscula o minúscula, un dígito del 0 al 9 o un guión
    #             bajo.
    #       - \w+: Expresión regular que coincide con uno o más caracteres de los descritos anteriormente.
    #       - [A-Za-z]: Expresión regular que busca solo una letra mayúscula o minúscula.
    #       - \b: Expresión regular que representa el inicio o final de una palabra en un string, usualmente se
    #             incluyen dos para indicar que se debe buscar un patrón en cada palabra. Por ejemplo, si se declara la
    #             expresión regular \bHola\b, esto extraerá todas las palabras Hola dentro de un string.
    #     - Búsqueda de palabras en un string (grupos de captura):
    #       - (patrón1|patrón2): Expresión regular que crea un grupo de captura para identificar uno o varios
    #             patrones específicos.
    #       - |: Compuerta OR utilizada para reconocer uno o varios patrones dentro de un grupo de captura.
    #     - Búsqueda de espacios en blanco, puntos y paréntesis en un string:
    #       - \s: Expresión regular que coincide con un carácter de espacio vacío, como un espacio, tabulador (\t),
    #             salto de línea (\n), etc.
    #       - \.: Expresión regular que encuentra un punto en un string.
    #       - \(: Expresión regular que encuentra un paréntesis de apertura en un string.
    #       - \): Expresión regular que encuentra un paréntesis de cierre en un string.
    #     - Búsqueda de repeticiones en un string:
    #       - *: Encuentra cero o más repeticiones de la expresión regular que tenga a la izquierda.
    #       - +: Encuentra una o más repeticiones de la expresión regular que tenga a la izquierda.
    #       - ?: Encuentra cero o una repetición de la expresión regular que tenga a la izquierda. Esto significa
    #             que la expresión regular que la precede es opcional y puede aparecer una vez o no aparecer en
    #             absoluto en la cadena que se está buscando.
    #       - {2,4}: Coincide con 2, 3 o 4 repeticiones de la expresión regular que tenga a la izquierda.
    #     - Anclaje de patrones:
    #       - ^: Cuando se coloca al principio de un patrón indica que la coincidencia debe encontrarse al comienzo
    #             de la línea de texto. Por ejemplo, si se declara la expresión regular ^Hola, esto será cierto solo
    #             cuando la palabra Hola se encuentre al inicio del string.
    #       - $: Cuando se coloca al final de un patrón indica que la coincidencia debe encontrarse al final de la

```

```

# línea de texto. Por ejemplo, si se declara la expresión regular $mundo, esto será cierto solo cuando
# la palabra mundo se encuentre al final del string.

# - string: Palabra en la que se desea buscar patrones a través de la expresión regular especificada.

# - flags: Modificadores opcionales que personalizan el comportamiento de la búsqueda.

# - re.IGNORECASE: Realiza la búsqueda sin distinción entre mayúsculas y minúsculas.

# - re.MULTILINE: Permite que el patrón coincida con múltiples líneas en la cadena.

# - re.DOTALL: Hace que el carácter . en el patrón coincida también con el carácter de nueva línea \n.

#Esta expresión regular busca en el string un número, un espacio opcional seguido de más números, otro
#espacio opcional y finalmente la palabra mm, cm o m, para así identificar el valor del radio y su unidad,
#regresando como resultado una lista de tuplas que contiene cada elemento encontrado.

unidades = re.findall(pattern = r"(\d+(\s?\d*)?)\s?(mm|cm|m)", string = radio, flags = re.IGNORECASE)

#Luego para extraer solamente la unidad se recorre solo la tercera tupla en la lista y se guarda en la
#variable unidades.

unidades = [unidad for _, _, unidad in unidades]

#Si existe un punto en la instrucción se juntará el resultado para que se considere como un número decimal.

if "." in radio:

    radio_operacion = re.findall(r"\d+", radio)

    #join(): Este método toma una lista de strings como argumento y los concatena, juntando cada string con el
    #carácter especificado.

    radio_operacion = ".".join(radio_operacion)

else:

    radio_operacion = re.findall(r"\d+", radio)

    radio_operacion = radio_operacion[0]

#float(): Método utilizado para convertir cualquier tipo de dato a un decimal de tipo float.

#math.pi: El atributo pi de la librería math representa la constante π.

return str(float(radio_operacion)*2.0*math.pi) + " " + unidades[0]

#_arun(): Este método se utiliza cuando se quiera que esta función se corra de forma asíncrona.

async def _arun(self):

    #raise: Palabra reservada que se utiliza para lanzar una excepción que interrumpe la ejecución de un programa.

    #NotImplementedError(): Tipo de excepción que se lanza cuando una función o método no implementa una
    #funcionalidad específica.

    raise NotImplementedError("Esta herramienta no tiene una ejecución asíncrona.")

#HERRAMIENTA PERSONALIZADA QUE CALCULA LA SUMA DE 2 NÚMEROS A TRAVÉS DE DOS PARÁMETROS:

class Suma(BaseTool):

    name = "Suma"

    description = "Usa esta herramienta para sumar dos números"

    def _run(self, operacion: str):

        #Esta expresión regular busca en el string un número y un punto decimal opcional seguido de más números
        #dentro de una misma palabra.

        numeros = re.findall(r'\b\d+(?:\.\d+)?\b', operacion)

        num1 = numeros[0]

        num2 = numeros[1]

        return float(num1) + float(num2)

    async def _arun(self):

```

```

raise NotImplementedError("Esta herramienta no tiene una ejecución asíncrona.")

#HERRAMIENTA PERSONALIZADA QUE CALCULA LA HIPOTENUSA DE UN TRIÁNGULO A TRAVÉS DE DOS POSIBLES VALORES RECIBIDOS A
#TRAVÉS DE TRES PARÁMETROS:

class HipotenusaTriangulo(BaseTool):
    name = "Hipotenusa de un triángulo"
    description = """Usa esta herramienta para calcular la longitud de una hipotenusa
        usando dos o más lados de un triángulo, y/o uno de sus ángulos en grados.
        Para usar esta herramienta se debe recibir al menos dos de los siguientes
        parámetros = ['cateto adyacente', 'cateto opuesto', 'angulo']"""
    parametros = ['cateto adyacente', 'cateto opuesto', 'angulo']

    def _run(self, operacion: str):
        cateto_adyacente = None #Iniciación de la variable del cateto_adyacente.
        cateto_opuesto = None #Iniciación de la variable del cateto_opuesto.
        angulo = None #Iniciación de la variable del angulo.
        hipotenusa = None #Iniciación de la variable de la hipotenusa.

        #Esta expresión regular busca en el string las palabras cateto adyacente, cateto opuesto, angulo o ángulo,
        #luego un espacio opcional, un signo de igual = y otro espacio opcional, eso lo mete en la primera tupla,
        #después busca en el string un número y un punto decimal opcional seguido de más números y eso lo mete en una
        #segunda tupla.

        triangulo = re.findall(r'(cateto adyacente|cateto opuesto|angulo|ángulo)\s*=\s*(\d+(:\.\d+)?)', operacion.lower(),
        re.IGNORECASE)

        #Bucle for que extrae el valor de las dos tuplas contenidas en la lista retornada por la expresión regular,
        #la cual contiene alguna de las palabras cateto adyacente, cateto opuesto, angulo o ángulo y en la otra tupla
        #su valor.

        for lados_angulo, valor in triangulo:
            if lados_angulo == "cateto adyacente":
                cateto_adyacente = float(valor) #Asignación de valor a la variable cateto_adyacente.
            if lados_angulo == "cateto opuesto":
                cateto_opuesto = float(valor) #Asignación de valor a la variable cateto_opuesto.
            if lados_angulo == "angulo":
                #math.radians(): Método que convierte ángulos a radianes.
                angulo = math.radians(float(valor)) #Asignación de valor a la variable angulo.

            if ((cateto_adyacente != None) and (cateto_opuesto != None)):
                #math.sqrt(): Método que saca la raíz cuadrada de un número.
                #**: Por medio de dos símbolos de multiplicación se obtiene el exponente de un número (^).
                hipotenusa = math.sqrt(cateto_adyacente**2 + cateto_adyacente**2)

            elif ((cateto_adyacente != None) and (angulo != None)):
                #math.cos(): Con este método se obtiene el coseno de un ángulo, que debe estar en radianes.
                hipotenusa = cateto_adyacente / math.cos(angulo)

            elif ((cateto_opuesto != None) and (angulo != None)):
                #math.sin(): Con este método se obtiene el seno de un ángulo, que debe estar en radianes.
                hipotenusa = cateto_opuesto / math.sin(angulo)

            else:
                hipotenusa = "No se han dado los datos necesarios para calcular la hipotenusa del triángulo."

        return hipotenusa

```

```

async def _arun(self):
    raise NotImplementedError("Esta herramienta no tiene una ejecución asíncrona.")

#Para declarar las herramientas personalizadas de un agente no se debe utilizar el método load_tools(), para ello
#simplemente se declara una lista que contenga una inicialización de las clases que hereden de la clase BaseTool.
nombreHerramientaPersonalizada = [NumeroAleatorio(), CircunferenciaCirculo(), Suma(), HipotenusaTriangulo()]

#initialize_agent(): Método que sirve para anexar a un agente las herramientas previamente cargadas al programa con el
#método load_tools() o de forma directa cuando se trata de herramientas personalizadas que heredan de la clase
#BaseTool, además en este se indica el LLM que se quiere utilizar y el tipo de agente que se pretende crear.
# - tools: Indica el diccionario que contiene el nombre de las herramientas que se quiere enlazar al agente.
# - llm: Indica el modelo de lenguaje o chat a utilizar.
# - agent: En este parámetro se indica el nombre del tipo de agente.
# - max_iterations: Este parámetro indica el máximo número de pensamientos (thoughts) que puede realizar el agente al
#   responder una pregunta, esto es bueno declararlo porque así se evita que el agente se quede atorado en un bucle
#   infinito buscando entre sus herramientas para contestar una pregunta que no puede responder.
# - verbose: Variable booleana que controla la información impresa en consola. Cuando verbose es True, el objeto
#   agente imprimirá información sobre el proceso de generación de la conversación, incluyendo el prompt, la
#   herramienta que está utilizando para resolver cada parte del prompt, la respuesta dada por cada herramienta y
#   la respuesta final del agente.
AgentePersonalizado = initialize_agent(
    agent = "zero-shot-react-description",
    tools = nombreHerramientaPersonalizada,
    llm = openaiLLM,
    verbose = True,
    max_iterations = 3
)
#initialize_agent().run(): El método run() proporciona el valor de entrada del agente y luego lo ejecuta para que
#podamos ver su resultado.
resultadoAgentePersonalizado = AgentePersonalizado.run("Calcula la circunferencia de un círculo con radio de 0.5m.")
print("Respuesta de Agente Personalizado: ", str(resultadoAgentePersonalizado), "\n\n\n")
resultadoAgentePersonalizado = AgentePersonalizado.run("Dame un número aleatorio.")
print("Respuesta de Agente Personalizado: ", str(resultadoAgentePersonalizado), "\n\n\n")
resultadoAgentePersonalizado = AgentePersonalizado.run("Suma los números 12.5 y 0.6")
print("Respuesta de Agente Personalizado: ", str(resultadoAgentePersonalizado), "\n\n\n")
resultadoAgentePersonalizado = AgentePersonalizado.run("Si tengo un triángulo con dos lados de longitud de 51cm y 34cm, cual
es el valor de su hipotenusa")
print("Respuesta de Agente Personalizado: ", str(resultadoAgentePersonalizado), "\n\n\n")
resultadoAgentePersonalizado = AgentePersonalizado.run("Si en un triángulo tengo un cateto opuesto de 51cm y un ángulo de 60
grados, cual es el valor de su hipotenusa?")
print("Respuesta de Agente Personalizado: ", str(resultadoAgentePersonalizado), "\n\n\n")

```



## Resultado del Código Python

```
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/5.-Inteligencia Artificial/7.-LangChain - Agentes.py"

-----1.- MODELOS-----
-----6.- AGENTES-----

Las herramientas que se utilizaron son:
[Tool(name='Search', description='A search engine. Useful for when you need to answer questions about current events. Input should be a search query.', args_schema=None, return_direct=False, verbose=False, callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x00000164DB890700>, func=<bound method SerpAPIWrapper.run of SerpAPIWrapper(<search_engine=<class \'serpapi.google_search.GoogleSearch\'>, params={\'engine\': \'google\', \'google_domain\': \'google.com\', \'gl\': \'us\', \'hl\': \'en\'}, serpapi_api_key='44e6e916da6b2c37e900f8cecd809e2c5f6a4467e9e1e4831d63e720cf824aff', aiosession=None)>, coroutine=<bound method SerpAPIWrapper.arun of SerpAPIWrapper(<search_engine=<class \'serpapi.google_search.GoogleSearch\'>, params={\'engine\': \'google\', \'google_domain\': \'google.com\', \'gl\': \'us\', \'hl\': \'en\'}, serpapi_api_key='44e6e916da6b2c37e900f8cecd809e2c5f6a4467e9e1e4831d63e720cf824aff', aiosession=None)>), Tool(name='Calculator', description='Useful for when you need to answer questions about math.', args_schema=None, return_direct=False, verbose=False, callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x00000164DB890700>, func=<bound method Chain.run of LLMMathChain(<memory=None, callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x00000164DB890700>, client=<class \'openai.api_resources.completion.Completion\'>, model_name='text-davinci-003', temperature=0.0, max_tokens=256, top_p=1, frequency_penalty=0, presence_penalty=0, n=1, best_of=1, model_kwarg={}, openai_api_key='sk-wys5u8t8k4430xg07lwqj38lB1kfJ8HcMTQBctiynk3V3li0E', openai_api_base=None, openai_organization=None, batch_size=20, request_timeout=None, logit_bias={}, max_retries=6, streaming=False, allowed_special=set(), disallowed_special='all'), prompt=PromptTemplate(input_variables=['question'], output_parser=None, partial_variables={}, template='Translate a math problem into an expression that can be executed using Python\'s numexpr library. Use the output of running this code to answer the question.\n\nQuestion: {{Question with math problem.}}\n``text{{single line mathematical expression that solves the problem}}\n``\nnumexpr.evaluate(text)...``output{{Output of running the code}}\n``\nAnswer: {{Answer}}\n\nQuestion: What is 37593 * 67?\n``text{{37593 * 67}}\n``\nnumexpr.evaluate("37593 * 67")...``output{{2518731}}\n``\nAnswer: 2518731\n\nQuestion: {question}\n, template_format='f-string', validate_template=True), input_key='question', output_key='answer')>, coroutine=<bound method Chain.arun of LLMMathChain(<memory=None, callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x00000164DB890700>, client=<class \'openai.api_resources.completion.Completion\'>, model_name='text-davinci-003', temperature=0.0, max_tokens=256, top_p=1, frequency_penalty=0, presence_penalty=0, n=1, best_of=1, model_kwarg={}, openai_api_key='sk-wys5u8t8k4430xg07lwqj38lB1kfJ8HcMTQBctiynk3V3li0E', openai_api_base=None, openai_organization=None, batch_size=20, request_timeout=None, logit_bias={}, max_retries=6, streaming=False, allowed_special=set(), disallowed_special='all'), prompt=PromptTemplate(input_variables=['question'], output_parser=None, partial_variables={}, template='translate a math problem into an expression that can be executed using Python\'s numexpr library. Use the output of running this code to answer the question.\n\nQuestion: {{Question with math problem.}}\n``text{{single line mathematical expression that solves the problem}}\n``\nnumexpr.evaluate(text)...``output{{Output of running the code}}\n``\nAnswer: {{Answer}}\n\nQuestion: What is 37593 * 67?\n``text{{37593 * 67}}\n``\nnumexpr.evaluate("37593 * 67")...``output{{2518731}}\n``\nAnswer: 2518731\n\nQuestion: {question}\n, template_format='f-string', validate_template=True), input_key='question', output_key='answer')>, WikipediaQueryRun(name='Wikipedia', description='A wrapper around Wikipedia. Useful for when you need to answer general questions about people, places, companies, historical events, or other subjects. Input should be a search query.', args_schema=None, return_direct=False, verbose=False, callback_manager=<langchain.callbacks.shared.SharedCallbackManager object at 0x00000164DB890700>, api_wrapper=WikipediaAPIWrapper(wiki_client=<modul
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

Action Input: ls
Observation: Command 'ls' returned non-zero exit status 1.
Thought: I need to find the directory the file is in.
Action: Terminal
Action Input: find / -name "17.-Vibracion en Estructuras.mph"
Observation: Command 'find / -name "17.-Vibracion en Estructuras.mph' returned non-zero exit status 2.
Thought: I need to find the exact path of the file.
Action: Terminal
Action Input: find / -name "17.-Vibracion en Estructuras.mph" -print
Observation: Command 'find / -name "17.-Vibracion en Estructuras.mph" -print' returned non-zero exit status 2.
Thought:

> Finished chain.
Respuesta de Agente de tipo Zero shot enlazado con la herramienta serpapi: Agent stopped due to iteration limit or time limit.

> Entering new AgentExecutor chain...
Necesito usar La herramienta Calculadora de circunferencia.
Action: calculadora de circunferencia
Action Input: 0.5m
Observation: 3.141592653589793 m
Thought: Ya tengo el resultado.
Final Answer: La circunferencia de un círculo con radio de 0.5m es 3.141592653589793 m.

> Finished chain.
Respuesta de Agente Personalizado: La circunferencia de un círculo con radio de 0.5m es 3.141592653589793 m.

> Entering new AgentExecutor chain...
Necesito un número aleatorio.
Action: Numero aleatorio
Action Input: N/A
Observation: Numero aleatorio = 24
Thought: Ya tengo mi número aleatorio.
Final Answer: 24
```

```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

> Entering new AgentExecutor chain...
Necesito sumar dos números
Action: Suma
Action Input: 12.5 y 0.6
Observation: 13.1
Thought: Ahora tengo la respuesta
Final Answer: La suma de 12.5 y 0.6 es 13.1

> Finished chain.
Respuesta de Agente Personalizado: La suma de 12.5 y 0.6 es 13.1


> Entering new AgentExecutor chain...
Necesito calcular la hipotenusa
Action: Hipotenusa de un triángulo
Action Input: cateto adyacente = 51cm, cateto opuesto = 34cm
Observation: 72.12489168102785
Thought: Ahora tengo el valor de la hipotenusa
Final Answer: La hipotenusa del triángulo es de 72.12489168102785 cm.

> Finished chain.
Respuesta de Agente Personalizado: La hipotenusa del triángulo es de 72.12489168102785 cm.


> Entering new AgentExecutor chain...
Necesito calcular la hipotenusa
Action: Hipotenusa de un triángulo
Action Input: cateto opuesto = 51cm, angulo = 60 grados
Observation: 58.889727457341834
Thought: Ahora tengo el valor de la hipotenusa
Final Answer: La hipotenusa es de 58.889727457341834 cm.

> Finished chain.
Respuesta de Agente Personalizado: La hipotenusa es de 58.889727457341834 cm.

```

## Referencias

Platzi, “3 errores que cometes al usar ChatGPT | APRENDE A USARLO MEJOR”, 2023 [Online], Available: <https://www.youtube.com/watch?v=JOJpO-q2dW8&t=91s>

Platzi, Carlos Alarcón, “Curso de Prompt Engineering con ChatGPT”, 2023 [Online], Available: <https://platzi.com/clases/7296-chatgpt/61970-chatgpt-para-prompt-engineering/>

OpenAI, “API Reference”, 2023 [Online], Available: <https://platform.openai.com/docs/api-reference>

AMP Tech, “Intro a LangChain: Construye sobre LLMs/GPT4”, 2023 [Online], Available: [https://www.youtube.com/watch?v=GoSbWL0\\_eGI](https://www.youtube.com/watch?v=GoSbWL0_eGI)

AMP Tech, “Langchain 1: Modelos y Prompts”, 2023 [Online], Available: <https://www.youtube.com/watch?v=qx3adFfbJRs>

AMP Tech, “LangChain: GPT4 ahora no olvidará nada”, 2023 [Online], Available: <https://www.youtube.com/watch?v=7xpykL0jAEA>

AMP Tech, “LangChain 3: Cadenas con GPT4”, 2023 [Online], Available: <https://www.youtube.com/watch?v=m1O6PJjEWnY>

Rabbitmetrics, “LangChain Explained in 13 Minutes | QuickStart Tutorial for Beginners”, 2023 [Online], Available: <https://www.youtube.com/watch?v=aywZrzNaKjs>

Greg Kamradt (Data Indy), “LangChain 101: Ask Questions On Your Custom (or Private) Files + Chat GPT”, 2023 [Online], Available: <https://www.youtube.com/watch?v=EnT-ZTrcPrg&list=PLqZXAkvF1bPNQER9mLmDbntNfSpzdDIU5&index=11>

Greg Kamradt (Data Indy), “LangChain 101: Agents Overview + Google Searches”, 2023 [Online], Available: <https://www.youtube.com/watch?v=Jq9Sf68ozk0&list=PLqZXAkvF1bPNQER9mLmDbntNfSpzdDIU5&index=5>

James Briggs, “LangChain Agents Deep Dive with GPT 3.5 — LangChain #7”, 2023 [Online], Available: <https://www.youtube.com/watch?v=jSP-gSEyVel>

