

Reconocimiento de voz con Python

Moisés Noguera Carrillo

Índice

<i>Introducción.....</i>	<i>2</i>
<i>Instalación de los requisitos necesarios.....</i>	<i>2</i>
<i>El paquete SpeechRecognition.....</i>	<i>3</i>
Instalación de SpeechRecognition	3
La clase Recognizer	4
Reconociendo voz desde un archivo de audio	5
Instalación del paquete PyAudio	5
La clase Microphone.....	6
Reconociendo voz desde un micrófono	6
Asistente de voz sencillo	7
Instalación de paquetes adicionales	7
Desarrollando el asistente de voz	8
Función record_audio(ask = False)	8
Función jarvis_speak(audio_string).....	10
Función respond(voice_data).....	11
Bibliografía	13

Introducción

A lo largo del documento se van a explicar los fundamentos básicos necesarios para poder desarrollar un asistente de voz sencillo usando el lenguaje de programación Python y los paquetes que ofrece.

Hoy en día la existencia de asistentes virtuales con los que interactuamos con nuestra voz es algo a lo que estamos acostumbrados y gracias a ellos se pueden realizar tareas de forma muy cómoda dando una simple orden. Ejemplos de este tipo de programas son: Siri, Google Assistant, Alexa... El desarrollo detrás de todos estos asistentes es bastante complejo ya que son capaces de reconocer e interpretar las mismas órdenes expresadas de forma distinta y realizar tareas muy variadas y completas.

Sin embargo, gracias al rápido avance de la tecnología en las últimas décadas el desarrollar un programa que responda a determinados comandos de voz es bastante sencillo y se puede conseguir en unas pocas líneas de código.

Instalación de los requisitos necesarios.

El requisito esencial para comenzar es tener instalado Python (versión 3.3 o superior) en el sistema operativo. El proceso de instalación dependerá del SO utilizado:

- En macOS: Normalmente macOS viene con Python 2 instalado por defecto. En el caso de querer instalar Python 3 hay que seguir los siguientes pasos.
 - Instalar GCC: Esto se consigue instalando [Xcode](#), [Command Line Tools](#) o el paquete [OSX-GCC-Installer](#).
 - Instalar Homebrew: Es el gestor de paquetes más popular para macOS. Para instalarlo ejecutar en el terminal: `$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"`
 - Instalar Python 3: `brew install python`
- En Linux (Ubuntu): Para instalar Python 3 en Linux con la distribución Ubuntu (en este caso la versión 3.6), basta con ejecutar las siguientes órdenes:
 - `sudo apt-get update`
 - `sudo apt-get install python3.6`
- En Windows: Basta con acceder a la página oficial de [Python](#) y descargar la última versión de Python disponible. Una vez hecho esto hay que seguir los pasos del instalador y Python quedará instalado en el sistema.

Una vez que se ha instalado Python, hay que instalar el gestor de paquetes de Python llamado *pip* a través del cual se puede instalar cualquier paquete disponible para este lenguaje. Para instalarlo hay que seguir los siguientes pasos:

- Descargar el script [*get-pip.py*](#).
- En un terminal situado en el mismo lugar donde se ha descargado el script ejecutar:
 - `python get-pip.py` (en macOS y Linux).
 - `py get-pip.py` (en Windows).

El paquete SpeechRecognition

Actualmente existen una gran variedad de paquetes en Python que proporcionan herramientas para reconocimiento de voz. Algunos de ellos son:

- [Apiai](#)
- [Assemblyai](#)
- [Google-cloud-speech](#)
- [Pocketsphinx](#)
- [SpeechRecognition](#)
- [Watson-developer-cloud](#)
- [Wit](#)

De entre todos los paquetes anteriores, el que destaca por su facilidad de uso es SpeechRecognition. Este paquete evita a los desarrolladores tener que crear scripts para acceder a los micrófonos y procesar archivos de audio desde cero de modo que con unas pocas líneas de código se puede reconocer voz muy fácilmente.

Además, SpeechRecognition ofrece el acceso a varias APIs populares de reconocimiento de voz, lo que lo hace muy flexible.

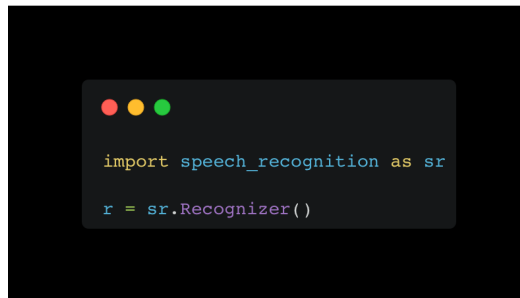
Instalación de SpeechRecognition

La instalación de SpeechRecognition es extremadamente sencilla una vez que se ha instalado pip. Las versiones de Python compatibles con este paquete son: Python 2.7 , 2.7 y 3.3+. Para instalarlo basta con ejecutar en el terminal:

- `pip install SpeechRecognition`

La clase Recognizer

De entre todo el conjunto de clases de las que está compuesto el paquete SpeechRecognition, la más importante y la que va a proporcionar las funcionalidades necesarias para el reconocimiento de voz es la clase *Recognizer*. Cada instancia que se cree de esta clase va a proporcionar un conjunto de métodos para reconocer voz desde una fuente de audio. Un ejemplo de creación de una instancia de esta clase es:

A screenshot of a code editor with a dark background. It shows two lines of Python code: `import speech_recognition as sr` and `r = sr.Recognizer()`. The code is highlighted with a light blue background.

```
import speech_recognition as sr
r = sr.Recognizer()
```

Ilustración 1: Instancia de la clase Recognizer


Como se ha comentado anteriormente, SpeechRecognition pone a disposición de los desarrolladores varias APIs para realizar el reconocimiento de voz. Una instancia de Recognizer puede acceder a siete métodos diferentes para conectarse a cada una de estas APIs. Los métodos son los siguientes:

- `recognize_bing()`: [Microsoft Bing Speech](#).
- `recognize_google()`: [Google Web Speech API](#).
- `recognize_google_cloud()`: [Google Cloud Speech](#). Requiere instalar el paquete `google_cloud_speech`.
- `recognize_houndify()`: [Houndify](#) de SoundHound.
- `recognize_ibm()`: [IBM Speech to Text](#).
- `recognize_sphinx()`: [CMU Sphinx](#). Requiere la instalación de PocketSphinx.
- `recognize_wit()`: [Wit.ai](#)

Puesto que SpeechRecognition viene de serie con una clave por defecto para la Google Web Search API para que se pueda acceder a ella directamente, es la API que se va a usar para el desarrollo del asistente. Cada uno de los métodos mencionados arriba recibe como parámetro una instancia de la clase *AudioData* que representará los datos captados de una fuente de audio que puede ser obtenida desde un archivo de audio (WAV, AIFF, AIFF-C y FLAC) o directamente desde un micrófono.

Reconociendo voz desde un archivo de audio

En una primera aproximación se va a realizar reconocimiento de voz desde un archivo de audio de tipo WAV. El archivo concreto recibe el nombre “harvard.wav”. A continuación, se muestra un ejemplo de código para realizar esta tarea:



```
import speech_recognition as sr

# Crear una instancia de la clase Recognizer.
r = sr.Recognizer()

# Instancia de la clase AudioFile
harvard = sr.AudioFile("harvard.wav")
with harvard as source:
    # Capta los datos del archivo de audio en una variable de tipo AudioData
    audio = r.record(source)

# Mostrar por pantalla el audio transcrito
print(r.recognize_google(audio))
```

Ilustración 2: Reconocimiento de voz desde archivo de audio

En este programa, tras crear la instancia *r* de la clase *Recognizer*, se crea una instancia de la clase *AudioFile* denominada *harvard* que accede al archivo de audio del mismo nombre pasado como parámetro en el constructor. Una vez hecho esto, el gestor de contexto de Python abre el archivo y lee su contenido almacenándolo en una instancia de *AudioFile* llamada *source*. Por último, va a ser el método *record()* el que se encargue de captar la información del archivo de audio en *source* para almacenarla en *audio* que es una instancia de tipo *AudioData*. Será esta instancia de *AudioData* la que se pase como parámetro al método de reconocimiento de texto *recognize_google()* para finalmente realizar el reconocimiento de la voz del archivo de audio. Al mostrar por pantalla el contenido del archivo se obtiene la siguiente salida: “*the stale smell of old beer lingers it takes heat to bring out the odor a cold.dip restores health and zest a salt pickle taste fine with ham tacos al Pastore are my favorite he's zestful food is bahat cross bun*”. Si se desea reproducir este ejemplo el archivo *harvard.wav* se puede descargar del siguiente [repositorio](#).

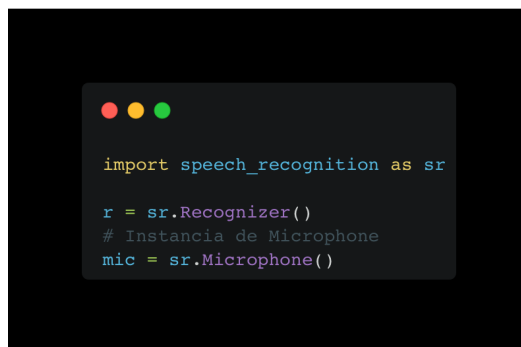
Instalación del paquete PyAudio

Si se quiere captar audio a través de un micrófono para realizar reconocimiento de voz es necesario instalar [PyAudio](#). Para instalarlo hay que seguir los siguientes pasos:

- En Linux (basado en Debian): Instalar usando apt.
 - `sudo apt-get install python-pyaudio python3-pyaudio`
- En macOS: En primer lugar hay que instalar PortAudio con HomeBrew y luego PyAudio con pip.
 - `brew install portaudio`
 - `pip install pyaudio`
- En Windows: Simplemente instalar PyAudio con pip.
 - `pip install pyaudio`

La clase Microphone

La otra opción, y quizá la más interesante, mediante la cual se puede realizar reconocimiento de voz es captando audio directamente desde un micrófono. Esto se va a poder realizar de forma muy sencilla e intuitiva gracias a la clase *Microphone* disponible en SpeechRecognition una vez instalado PyAudio. Para ello hay que crear una instancia de *Microphone* que accederá al micrófono establecido por defecto en el sistema.

A screenshot of a code editor with a dark background. The code is written in Python and shows the import of the speech_recognition module as 'sr', the creation of a Recognizer object 'r', and the creation of a Microphone object 'mic'.

```
import speech_recognition as sr

r = sr.Recognizer()
# Instancia de Microphone
mic = sr.Microphone()
```

Ilustración 3: Instancia de la clase Microphone

Reconociendo voz desde un micrófono

La estructura de un programa sencillo para captar audio desde un micrófono y realizar un reconocimiento de ese audio es muy similar a la vista en el ejemplo del archivo de audio, con la salvedad de que ahora la fuente desde la cual se proporcionan los datos de audio es un micrófono.



```
import speech_recognition as sr

# Crear una instancia de la clase Recognizer y Microphone.
r = sr.Recognizer()
mic = sr.Microphone()

with mic as source:
    audio = r.listen(source)

print(r.recognize_google(audio))
```

Ilustración 4: Reconocimiento de voz desde un micrófono

Del mismo modo que ocurría con la clase *AudioFile*, la clase *Microphone* es un gestor de contexto. Para captar el audio de entrada del micrófono se usa el método *listen()*. Este método recibe como argumento una fuente de audio y recoge audio de ella hasta que detecte silencio. La información captada por *listen()* es guardada en una instancia de tipo *AudioData* que se pasa como argumento a *recognizer_google()* al igual que en el ejemplo anterior para reconocer la voz. Se mostrará por pantalla aquello que el usuario haya dicho al micrófono.

Asistente de voz sencillo

En este apartado se muestra como a partir de todo lo explicado hasta ahora, se puede desarrollar un asistente que responda a comandos de voz predefinidos para realizar tareas sencillas.

Instalación de paquetes adicionales

A parte de los paquetes ya instalados hasta ahora, hay que instalar dos paquetes adicionales para el asistente. Los paquetes son los siguientes:

- El paquete [gTTS](#) (Google Text-To-Speech): Este es el paquete que va a hacer que el asistente “hable”. A través de él, se proporciona acceso a la API de Google Translate Text-To-Speech para poder generar archivos de audio en formato MP3 con el texto que se le indique. Para instalarlo:
 - `pip install gTTS`
- El paquete [Playsound](#): Este paquete se va a instalar por cuestiones prácticas. Cada archivo de audio que se genere para que el asistente hable se reproducirá de forma automática en el reproductor por defecto del sistema (iTunes en el caso de macOS).

Para evitar que cada vez que el asistente hable se abra el reproductor se va a hacer uso de playsound que va a permitir que el archivo sea reproducido directamente sin que se abra el reproductor por defecto. Playsound tiene una dependencia denominada Appkit que se encuentra en otro paquete llamado PyObjC que también hay que instalar. Para instalarlo:

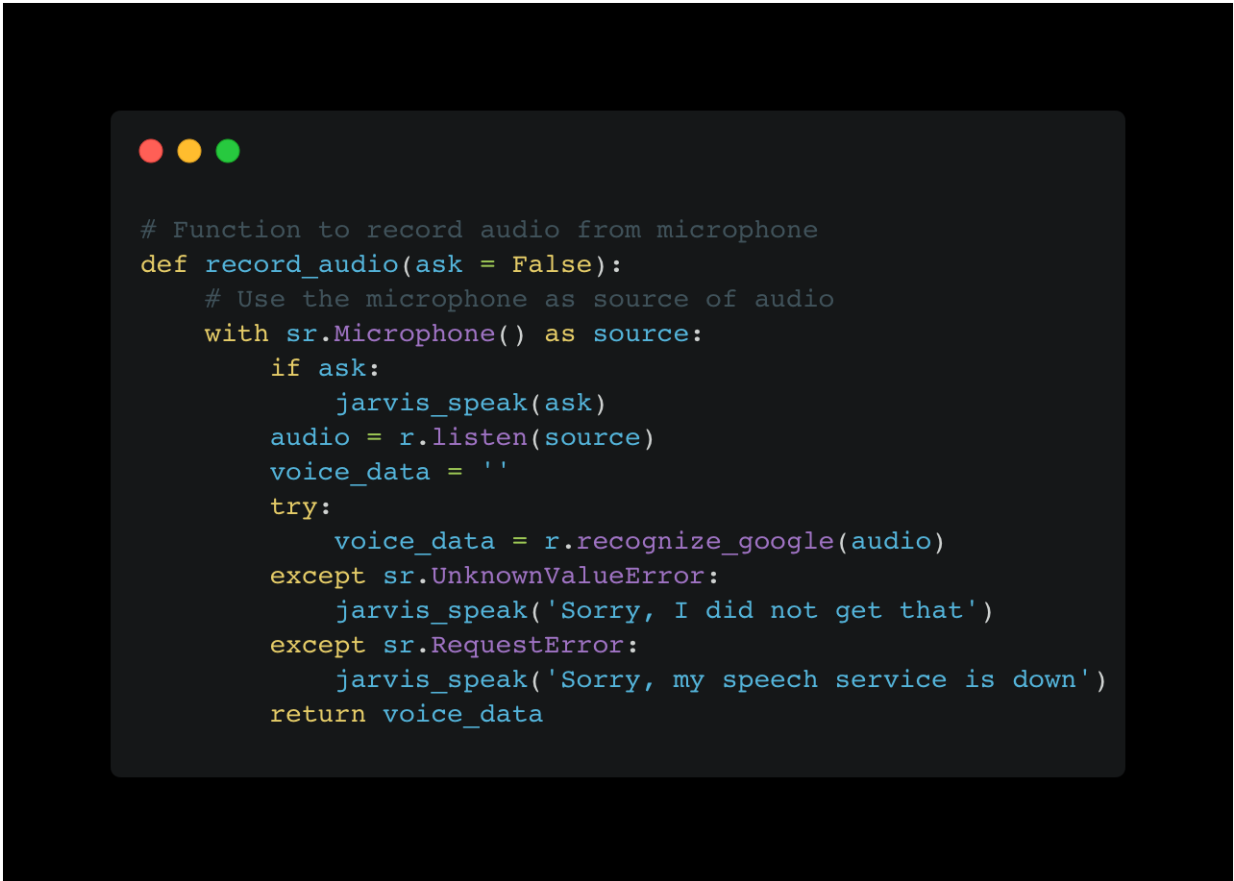
- `pip install playsound`
- `pip install PyObjC`

Desarrollando el asistente de voz

El asistente se va a estar formado por tres funciones principales:

- `record_audio(ask = False)`
- `jarvis_speak(audio_string)`
- `respond(voice_data)`

Función `record_audio(ask = False)`

A screenshot of a code editor with a dark background. At the top left, there are three colored circles (red, yellow, green) representing window controls. The code is written in a light-colored font. It defines a function `record_audio` with a parameter `ask = False`. The function uses `sr.Microphone` to capture audio, then `r.listen` to wait for audio. If `ask` is True, it calls `jarvis_speak` before listening. It then uses `r.recognize_google` to transcribe the audio. There are two `except` blocks: one for `sr.UnknownValueError` which calls `jarvis_speak` with the message "Sorry, I did not get that", and another for `sr.RequestError` which calls `jarvis_speak` with the message "Sorry, my speech service is down". Finally, it returns `voice_data`.

```
# Function to record audio from microphone
def record_audio(ask = False):
    # Use the microphone as source of audio
    with sr.Microphone() as source:
        if ask:
            jarvis_speak(ask)
        audio = r.listen(source)
        voice_data = ''
        try:
            voice_data = r.recognize_google(audio)
        except sr.UnknownValueError:
            jarvis_speak('Sorry, I did not get that')
        except sr.RequestError:
            jarvis_speak('Sorry, my speech service is down')
    return voice_data
```

Ilustración 5: Función `record_audio(...)` para reconocer voz

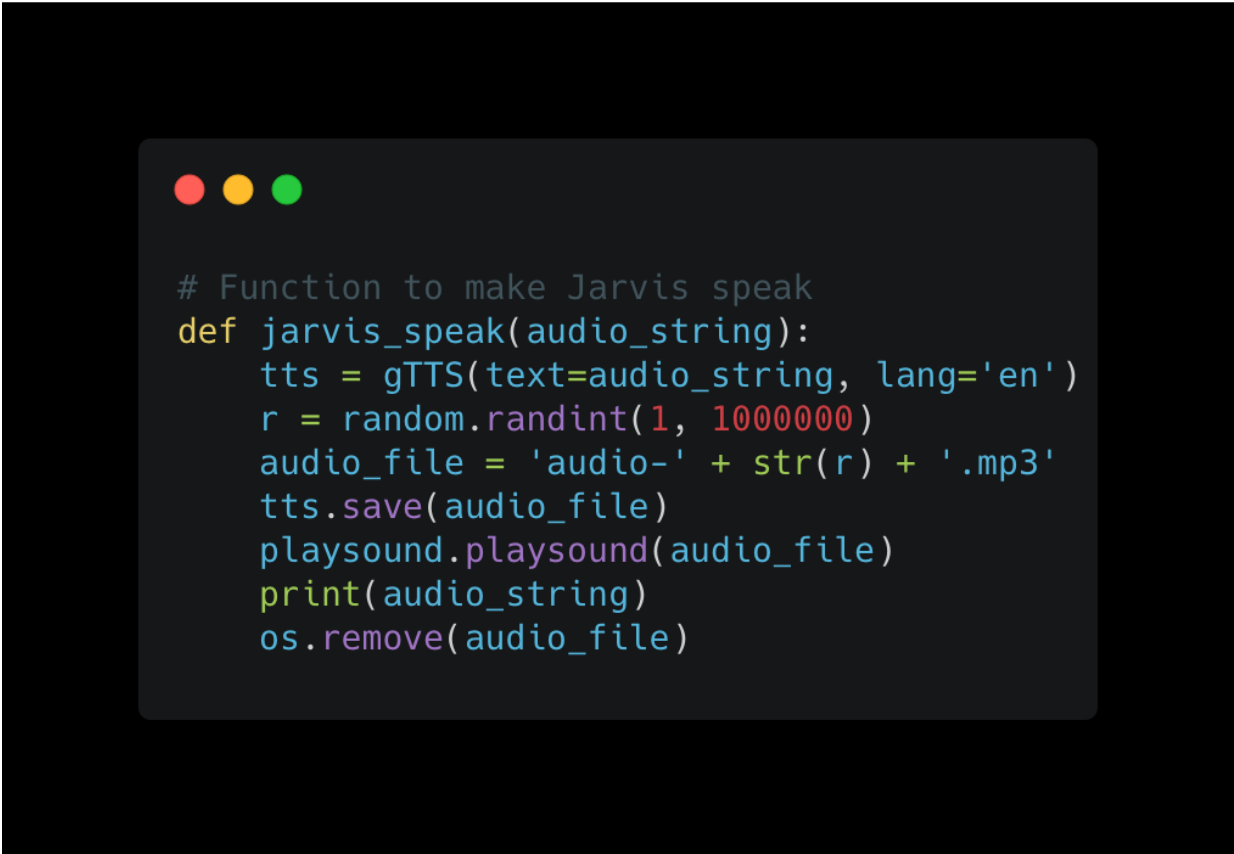
Esta función tiene como cometido reconocer el audio captado desde el micrófono del sistema haciendo uso del método *listen()* y usando después *recognize_google()* tal y como se ha mencionado en el ejemplo anterior. Sin embargo ahora se deben tener en cuenta las posibles excepciones que pueden ocurrir a la hora de reconocer la voz que tiene en cuenta SpeechRecognition. Son las siguientes:

- *UnknownValueError*: Ocurre cuando no se puede captar correctamente el audio debido a sonidos de fondo, sonidos que no son palabras (tos, chasquidos, palmas...). En ese caso el asistente informará al usuario de que no ha podido comprender lo que le ha dicho y quedará a la espera de un nuevo intento.
- *RequestError*: Ocurre cuando el acceso a la API de reconocimiento de voz no se puede realizar por un fallo en la conexión a Internet, que la API haya sufrido una caída del servicio.... El asistente informará al usuario de que su servicio no está disponible en ese momento.

Si el reconocimiento de voz se hace con éxito y no ocurren ninguna de las excepciones mencionadas la función devuelve una cadena de texto llamada *voice_data* con la oración obtenida.

Adicionalmente, *record_audio()* tiene un parámetro opcional *ask* que es una cadena de texto que el asistente reproducirá en para atender algunos de los comandos de voz que se verán más adelante.

Función `jarvis_speak(audio_string)`



```
# Function to make Jarvis speak
def jarvis_speak(audio_string):
    tts = gTTS(text=audio_string, lang='en')
    r = random.randint(1, 1000000)
    audio_file = 'audio-' + str(r) + '.mp3'
    tts.save(audio_file)
    playsound.playsound(audio_file)
    print(audio_string)
    os.remove(audio_file)
```

Ilustración 6: Función `jarvis_speak(...)` para que el asistente pueda "hablar".

A través de esta función se va a conseguir simular que el asistente está hablando. Para ello hay que hacer uso de la API Google Translate Text-To-Speech importante del paquete `gtts` la clase `gTTS`.

En primer lugar, se crea una instancia de `gTTS` llamada `tts` cuyo constructor recibe como parámetros el texto que se va a transformar en audio y el idioma de este. La instancia `tts` generará un archivo de audio en formato MP3 al que se le deberá asignar un nombre. Este nombre es asignado de forma aleatoria con la siguiente estructura: “audio-(r).mp3” siendo `r` un número entero aleatorio entre 1 y 10^6 convertido a string. Para crear el archivo de audio se usa el método `save()` al que se le pasa como parámetro el nombre generado anteriormente.

En segundo lugar, se usa la función `playsound` del paquete `playsound` con el archivo de audio creado para que se reproduzca directamente sin abrir el reproductor por defecto del sistema. Adicionalmente se muestra por pantalla la respuesta del asistente en modo texto.

Finalmente, empleando la funcionalidad `remove()` del paquete `os` presente por defecto en Python se elimina el archivo de audio una vez reproducido para evitar que el espacio de trabajo se llene de archivos de audio innecesarios cada vez que el asistente responda.

Función `respond(voice_data)`

```
# Voice commands
def respond(voice_data):
    if 'what is your name' in voice_data:
        jarvis_speak('My name is Sara')

    if 'what time is it' in voice_data:
        now = datetime.datetime.now()
        hour = '{:02d}'.format(now.hour)
        minute = '{:02d}'.format(now.minute)
        jarvis_speak('It's ' + hour + ':' + minute)

    if 'what is the date today' in voice_data:
        now = datetime.datetime.now()
        month_day = '{:02d}'.format(now.day)
        month_name = now.strftime("%B")
        year = '{:02d}'.format(now.year)
        week_day = now.strftime("%A")
        jarvis_speak('It's ' + week_day + ', ' + month_name + ' ' + month_day + ', ' + year)

    if 'search' in voice_data:
        search = record_audio('What do you want to search for?')
        url = 'https://google.com/search?q=' + search
        webbrowser.get().open(url)
        jarvis_speak('Here is what I found for ' + search)

    if 'find location' in voice_data:
        location = record_audio('What is the location?')
        url = 'https://google.es/maps/place/' + location + '/&'
        webbrowser.get().open(url)
        jarvis_speak('Here is the location of ' + location)

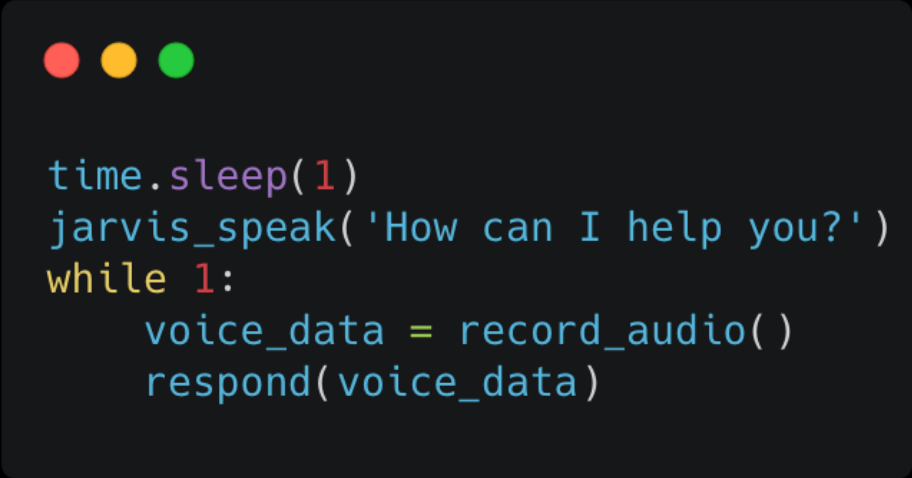
    if 'bye' in voice_data:
        jarvis_speak('Good bye!')
        exit()
```

Ilustración 7: Función `respond(...)` con los comandos de voz disponibles

En esta función se gestionan los seis comandos de voz ante los que responde el asistente. Recibe como parámetro el audio transformado en texto que ha sido captado por el micrófono cada vez que el usuario habla (*voice_data*):

- ‘what is your name’: Si esta cadena está presente en el texto captado del usuario el asistente responde (usando *jarvis_speak()*) diciendo que su nombre es Sara.
- ‘what time is it’: En el caso de que el usuario pregunte la hora el asistente la proporciona. La hora se obtiene gracias al paquete *datetime* que viene por defecto con Python.
- ‘what is the date today’: El asistente proporcionará la fecha de hoy indicando el día de la semana, el nombre y día del mes y el año. Cada uno de estos datos se obtiene de nuevo haciendo uso de *datetime*.

- ‘search’: Mediante este comando de voz se puede buscar cualquier cosa en Internet. En este caso se utiliza la función *record_audio()* pasando el parámetro opcional con el string ‘What do you want to search for?’ que será lo que el asistente pregunte ante este comando de voz. En ese momento el usuario responde aquello que desea buscar y *record_audio* lo reconoce y lo devuelve en la variable *search*. Una vez que se obtiene lo que se va a buscar se construye la URL correspondiente y se abre en el navegador por defecto gracias al paquete *webbrowser* que también viene por defecto en Python. Cuando se realiza la búsqueda el asistente informa de ello al usuario.
- ‘find location’: A través de este comando de voz se puede buscar una localización accediendo a Google Maps. El funcionamiento es exactamente el mismo que en el comando anterior: el asistente pregunta la localización a buscar, el usuario la indica y se genera la URL para abrirla en el navegador con *webbrowser*.
- ‘exit’: Este comando de voz se usa para salir del asistente. Ante él, el asistente responde diciendo adiós y se cierra el programa.



```
time.sleep(1)
jarvis_speak('How can I help you?')
while 1:
    voice_data = record_audio()
    respond(voice_data)
```

Ilustración 8: Inicio del asistente

Cuando se inicia el asistente pasa un segundo de espera y pregunta al usuario en qué le puede ayudar. A partir de ahí se entra en un bucle infinito en el que se va captando lo que usuario pregunta con *record_audio()* y lo que el asistente responde según los comandos de voz con *respond(...)*.

Bibliografía

- Tutorial de uso de SpeechRecognition: [https://realpython.com/python-speech-recognition/ - appendix-recognizing-speech-in-languages-other-than-english](https://realpython.com/python-speech-recognition/-appendix-recognizing-speech-in-languages-other-than-english)
- Tutorial de creación de un asistente de voz con SpeechRecognition: <https://www.youtube.com/watch?v=x8xjj6cR9Nc>
- Sitio web oficial de Python: <https://www.python.org/>
- Sitio web oficial de Pip: <https://pip.pypa.io/en/stable/>