

INTÉRPRETE CON SENTIDO

EDITOR DE AUDIO USANDO PYDUB

Stiven Alejandro Carvajal

Victor Manuel Garcia

Natalia Velez Orjuela

Universidad Autónoma de Manizales

Facultad de Ingeniería de Sistemas

Paradigmas de Lenguajes de Programación

2025

Abstract

El estudio de los lenguajes de programación a nivel conceptual permite conocer mejor los lenguajes que usan los programadores, así como también las características comunes a diferentes lenguajes; los programadores pueden utilizarlos mejor y aprovechar sus capacidades al máximo conociendo las limitaciones y características. Por otra parte, les permite acercarse a los lenguajes de programación que puedan conocer en un futuro.

Este documento describe el proyecto del curso de Paradigmas de Lenguajes de Programación, realizado en ANTLR4, Python y haciendo uso de la librería de *Python Pydub*. El trabajo se enfoca en la implementación de un intérprete para un lenguaje de programación. Este intérprete realizará un análisis sintáctico para validar que la entrada cumpla con la estructura de la gramática y generará un árbol de sintaxis abstracta (AST), donde cada nodo representará una operación.

En adelante llamaremos a este intérprete, Musicologo, porque además de interpretar el programa, permitirá manipular, editar y transformar archivos de audio mediante una sintaxis estructurada. Musicologo podrá cargar un archivo de audio y ejecutar una serie de operaciones definidas en el lenguaje. Con esta información, el programa generará un árbol de sintaxis abstracta (AST) que representará las operaciones según la gramática definida pero no se va a mostrar el árbol.

El intérprete está compuesto por los siguientes componentes principales:

- Lexer: se encarga de analizar la entrada y convertirla en una secuencia de tokens, según las reglas definidas en la gramática.
- Parser: Recibe los tokens del lexer y verifica que la estructura del código cumpla con las reglas sintácticas establecidas, generando el AST.
- Evaluador (Visitor): implementa el patrón Visitor, recorriendo el AST y aplicando las transformaciones de audio definidas.

El evaluador visitará el AST y ejecutará las operaciones definidas en los nodos de árbol. Algunas de las funciones principales que realizará el evaluador son:

- ☐ Cargar archivos de audio: Permite cargar los archivos de audio en formato MP3
- ☐ Editar audio: Realiza transformaciones como:
 - ☐ **Recorte de fragmentos específicos.**
 - ☐ **Ajuste del volumen.**
 - ☐ **Unión de múltiples archivos de audio en uno solo.**
 - ☐ **Repetición de fragmentos de audio.**
 - ☐ **Silenciamiento de segmentos específicos.**
 - ☐ **División de un archivo de audio en partes separadas.**

Tabla de Contenidos

| | |
|--|-----------|
| 1. Planteamiento del Problema..... | 10 |
| 1.1 FUNCIONES PRINCIPALES..... | 11 |
| 1.2. ELEMENTOS PARA CONDICIONES..... | 13 |
| 1. 3. REGLAS SINTÁCTICAS..... | 14 |
| 2. Análisis del Problema..... | 15 |
| 2.1 Casos de uso..... | 15 |
| 2.1.1 Caso de uso 1: Cargar un archivo de audio..... | 15 |
| 2.1.2 Caso de uso 2: Analizar la sintaxis de una operación de audio..... | 16 |
| 2.1.3 Caso de uso 3: Tokenizar la instrucción..... | 16 |
| Actor: Musicologo..... | 16 |
| 2.1.4 Caso de uso 5: Aplicar transformaciones de audio..... | 16 |
| 2.1.5 Caso de uso 6: Recortar fragmentos específicos..... | 17 |
| 2.1.6 Caso de uso 7: Aumentar o disminuir el volumen..... | 17 |
| 2.1.7 Caso de uso 8: Repetir fragmentos de audio..... | 17 |
| 2.1.8 Caso de uso 9: Concatenar varios archivos de audio..... | 17 |
| 2.1.9 Caso de uso 10: Guardar el resultado en un nuevo archivo..... | 17 |
| 2.1.10 Caso de uso 11: Mostrar información del nuevo archivo..... | 18 |
| 2.1.11 Caso de uso 12: Exportar nuevo audio..... | 18 |
| 3. Diseño de la solución propuesta..... | 18 |
| 3.1 Diagrama de casos de usos..... | 20 |
| 3.2 Diagrama de clases..... | 21 |
| 4. Documentación de la implementación..... | 24 |
| 4.1 Gramática del lenguaje..... | 24 |
| 4.1.1 Regla de inicio..... | 25 |
| 4.1.2 Expresiones..... | 25 |
| 4.1.3 Bloque de expresiones..... | 29 |
| 4.1.4 condiciones..... | 30 |
| 4.1.6 Elementos condicionales..... | 31 |
| 4.1.7. REGLAS SINTÁCTICAS..... | 32 |
| 4.1.7.1 Volumen:..... | 32 |
| 4.1.7.2 Archivo_MP3..... | 33 |
| 4.1.7.3 ID..... | 33 |
| 4.1.7.4 RUTA..... | 34 |
| 4.1.7.5 TIEMPO..... | 34 |

| | |
|---|----|
| 4.1.7.6 NUMERO: | 35 |
| Figura 25. Manejo de espacios en blanco | 35 |
| 4.2 Evaluador al visitador | 36 |
| 4.3 Manual de usuario | 36 |
| 4.3.1 Cargar | 36 |
| 4.3.1.1 Gramatica | 36 |
| 4.3.1.2 Ejemplo de uso | 36 |
| 4.4.2 Recortar | 36 |
| 4.4.2.1 Gramatica | 36 |
| 4.4.2.2 Ejemplo de uso | 36 |
| 4.4.3 Exportar | 37 |
| 4.4.3.1 Gramatica | 37 |
| 4.4.4 Subir volumen | 37 |
| 4.4.4.1 Gramatica | 37 |
| 4.4.5 Condicional | 37 |
| 4.4.5.1 Gramatica | 37 |
| 4.4.5.2 Ejemplo de uso | 38 |
| 4.4.6 Dividir | 38 |
| 4.4.6.1 Gramatica | 38 |
| 4.4.6.2 Ejemplo de uso | 38 |
| 4.4.7 Combinar | 38 |
| 4.4.7.1 Gramatica | 38 |
| 4.4.7.2 Ejemplo de uso | 39 |
| 4.4.8 Silenciar | 39 |
| 4.4.8.1 Gramatica | 39 |
| 4.4.8.2 Ejemplo de uso | 39 |
| 4.4.9 Concatenar | 39 |
| 4.4.9.1 Gramatica | 39 |
| 4.4.9.2 Ejemplo de uso | 39 |
| 4.4.10 Repetir | 40 |
| 4.4.10.1 Gramatica | 40 |
| 4.4.10.2 Ejemplo de uso | 40 |
| 4.4 Manual técnico | 40 |
| 4.4.1 Descripción | 40 |
| 4.4.2 Requisitos del sistema | 40 |
| 4.4.3 Instalación | 40 |
| 4.4.4 Configurar FFmpeg | 41 |
| 4.4.5 Estructura del proyecto | 41 |
| 4.4.6 Uso del intérprete | 42 |

| | |
|--|-----------|
| 4.4.7 Comandos principales..... | 43 |
| 4.4.8 Estructura interna..... | 43 |
| 4.4.9 Componentes Principales..... | 43 |
| 4.4.10 Errores comunes y soluciones..... | 44 |
| 5. Conclusiones..... | 44 |

Lista de tablas

| | |
|---|---|
| Tabla 1. Ejemplo de especificaciones del lenguaje | 7 |
| Tabla 2. Alcance de validación de errores | 3 |
| Tabla 3. Alcance de validación de errores | 4 |

Lista de figuras

| | |
|---|----|
| Figura 1. Ejemplo de sintaxis | 8 |
| Figura 2. Ejemplo de invocación del intérprete | 13 |
| Figura 3. Diagrama de casos de uso | 19 |
| Figura 4. Diagrama de clases | 20 |
| Figura 5. Regla de inicio de la gramática | 23 |
| Figura 6. Cargar un archivo | 24 |
| Figura 7. Recortar un audio | 24 |
| Figura 8. Exportación de audio | 25 |
| Figura 9. Modificación de volumen | 25 |
| Figura 10. Uso de condicionales | 26 |
| Figura 11. División de archivos | 26 |
| Figura 12. Combinación de archivos | 27 |
| Figura 13. Silenciamiento de partes | 27 |
| Figura 14. Concatenación de archivos | 27 |
| Figura 15. Repetición de un bloque de instrucciones | 28 |
| Figura 16. Definición de bloque de expresiones | 28 |
| Figura 17. Condición | 28 |
| Figura 18. Funciones principales | 29 |
| Figura 19. Elementos para condiciones | 30 |
| Figura 20. Archivo mp3 | 32 |

| | |
|---|----|
| Figura 21. Identificadores | 32 |
| Figura 22. Ruta | 32 |
| Figura 23. Tiempo | 33 |
| Figura 24. Número | 33 |
| Figura 25. Manejo de espacios en blanco | 33 |
| Figura 26. Clonación de repositorio | 40 |
| Figura 27. Instalacion de dependencias | 40 |
| Figura 28. Estructura del proyecto | 45 |

1. Planteamiento del Problema

El intérprete debe estar orientado a la composición de algoritmos para la manipulación de archivos de audio. Este intérprete debe reconocer una sintaxis específica que permita realizar operaciones sobre los archivos de audio de manera eficiente, ejemplos de expresiones válidas dentro del lenguaje:

```
PS C:\Users\usuario\Pictures\PROYECTOPOO\Musicologo> python src/main.py
Bienvenido a Musicologo. Ingrese los comandos que desee.

>>> cargar archivos/musicalaura.mp3 como miAudio
Cargando archivo: archivos/musicalaura.mp3
```

Figura 1. Ejemplo de sintaxis

Para ejecutar el intérprete Musicólogo, es necesario abrir una terminal y navegar hasta el directorio donde se encuentra el proyecto. Una vez en la ubicación correcta, el programa se ejecuta con el siguiente comando:

```
python src/main.py
```

Al iniciar, Musicólogo da la bienvenida al usuario e indica que puede ingresar los comandos correspondientes. Los comandos siguen una sintaxis específica para cargar, editar y manipular archivos de audio.

Por ejemplo, el siguiente comando carga un archivo de audio ubicado en el directorio `archivos/` y lo almacena en una variable llamada `miAudio`:

```
>>> cargar archivos/musicalaura.mp3 como miAudio
```

Tras ejecutar este comando, Musicólogo confirma la carga del archivo con un mensaje en la pantalla

```
Cargando archivo: archivos/musicalaura.mp3
```

A continuación se presenta una tabla para conocer más a detalle las especificaciones del lenguaje

Tabla 1. Ejemplo de especificaciones del lenguaje

1.1 FUNCIONES PRINCIPALES

| Instrucción | Símbolo | Ejemplo | Explicación |
|-------------------------|------------------|---|--|
| Cargar archivo de audio | COMANDO_CARGAR | cargar ruta/cancion.mp3 como miCancion; | Importa un archivo de audio en formato MP3 y lo asigna a un identificador. |
| Recortar audio | COMANDO_RECORTAR | recortar 00:10 00:20 miCancion como miCancionRecortada ; | Recorta un segmento de un archivo de audio desde el segundo 10 hasta el segundo 20 y lo guarda con |

| | | | |
|------------------------------|-------------------------|--|--|
| | | | un nuevo identificador. |
| Incrementar volumen audio | COMANDO_INCREMENTAR_VOL | subirVol miCancion 50dB 00:10 00:15 | Aumenta el volumen de la pista de audio en 50 dB dentro del intervalo de tiempo especificado (del segundo 10 al 15). |
| Dividir audio | COMANDO_DIVIDIR | dividir 00:30 miCancion como p | Divide el audio en dos partes en el segundo 30, generando los archivos p1 y p2. |
| Combinar audios | COMANDO_COMBINAR | combinar cancion1 cancion2 como cancionCombinada | Fusiona dos archivos de audio en uno nuevo con el nombre cancionCombinada. |
| Silenciar porciones de audio | COMANDO_SILENCIAR | silenciar 00:30 00:35 miCancion | Silencia el audio dentro del intervalo de tiempo indicado. |
| Concatenar audio | COMANDO_CONCATENAR | concatenar miCancion 3 como miCancionConcatenada | Repite el archivo miCancion tres veces y lo guarda con un nuevo identificador. |
| Exportar audio | COMANDO_EXPORTAR | exportar miCancionRecortada; | Guarda el archivo de audio modificado en el sistema. |

| | | | |
|---------------|----|------------------------------|---|
| Identificador | ID | miCancion, cancionEditada | Nombre asignado a un archivo de audio para su manipulación. |
|---------------|----|------------------------------|---|

1.2. ELEMENTOS PARA CONDICIONES

El lenguaje permite definir condiciones y comparaciones:

| Elemento | Símbolos Disponibles | Descripción |
|------------------------|--|--|
| Condicional | <code>si if</code> | Evalúa una condición antes de ejecutar una acción. |
| Acción | <code>entonces then</code> | Indica la ejecución de una acción si se cumple la condición. |
| Alternativa | <code>sino else</code> | Define una acción alternativa si la condición no se cumple. |
| Operadores | <code>> < >= <= == !=</code> | Comparadores lógicos usados en condiciones. |
| Características | <code>duración dur volumen vol</code> | Propiedades del audio que pueden evaluarse en una condición. |

1. 3. REGLAS SINTÁCTICAS

| Elemento | Expresión Regular | Descripción |
|---------------------------|---|---|
| Volumen | <code>('-'?)?[0-9]+ ('dB' 'db' 'DB' 'Db');</code> | Identifica valores de volumen expresados en decibeles (dB). Puede incluir un número positivo o negativo, seguido de la unidad de medida en diferentes combinaciones de mayúsculas y minúsculas, como "dB", "db", "DB" o "Db". Esto permite representar aumentos o reducciones de volumen en la manipulación de archivos de audio. |
| Archivo MP3 | <code>[a-zA-Z0-9]+ '.' ('mp3' [a-zA-Z0-9]+);</code> | Define el formato de los nombres de archivos de audio en formato MP3. Un archivo válido debe comenzar con una secuencia de letras o números, seguida de un punto (.) y la extensión ".mp3". Aunque la expresión parece permitir otras extensiones, el objetivo principal es identificar archivos de audio en este formato. |
| Identificador | <code>[a-zA-Z][a-zA-Z0-9]*</code> | Nombre asignado a un archivo de audio para su manipulación. Inicia obligatoriamente con una letra |
| Ruta | <code>([a-zA-Z0-9]+'/')+</code> | Especifica la ubicación de un archivo dentro del sistema. |
| Tiempo | <code>[0-9]+ ':' [0-9]+ (':' [0-9]+)?</code> | Representa un instante de tiempo en un archivo de audio. |
| Número | <code>[0-9]+</code> | Valor numérico usado en repeticiones y condiciones. |
| Espacios en Blanco | <code>[\t\r\n]+ -> skip</code> | Los espacios y saltos de línea son ignorados por el intérprete. |

Tabla 2. Alcance de validación de errores

| | |
|---------|--|
| Error 1 | Sintaxis incorrectas en funciones de manipulación de audio |
| Error 2 | Operación de edición de audio no permitida |

El intérprete deberá poder ser invocado de las siguientes formas:



```
PS C:\Users\usuario\Pictures\PROYECTOPOO\Musicologo> python src/main.py
```

Figura 2. Ejemplo de invocación del intérprete

La librería utilizada para este proyecto es **Pydub** la cual es utilizada para la manipulación y procesamiento de archivos de audio. La cual permite realizar operaciones como carga, edición y exportación de archivos en distintos formatos. Nosotros realizamos un intérprete específicamente para archivos .MP3.

2. Análisis del Problema

Recopilar requisitos de usuarios. Crear un diagrama de casos de uso que muestra las interacciones y describir los casos de uso.

2.1 Casos de uso

2.1.1 Caso de uso 1: Cargar un archivo de audio

Actor: Usuario

Descripción: El usuario selecciona un archivo de audio desde su dispositivo y lo carga en el sistema para su procesamiento. El sistema de archivos valida que el archivo de audio sea compatible (por ejemplo, MP3) antes de permitir la carga. Este servirá como entrada para las siguientes operaciones.

2.1.2 Caso de uso 2: Analizar la sintaxis de una operación de audio

Actor: Musicologo

Descripción: Musicologo analiza la sintaxis de una instrucción proporcionada por el usuario para asegurarse de que es válida y pueda ser procesada correctamente, el sistema analiza la estructura de la instrucción y si es correcta su sintaxis permite que el proceso continúe.

2.1.3 Caso de uso 3: Tokenizar la instrucción

Actor: Musicologo

Descripción: Una vez analizada y validada la sintaxis de la instrucción de audio, el sistema la divide en partes más pequeñas, es decir, tokens. Cada token representa un elemento dentro de la instrucción, lo que facilita su posterior procesamiento.

Caso de uso 4: Generar el árbol de sintaxis abstracta (AST)

Actor: Musicologo

Descripción: Luego de la tokenización, los elementos de la instrucción se organizan en un árbol de sintaxis abstracta (AST). Este árbol representa la jerarquía y estructura de la operación, permitiendo su transformación en acciones aplicables al audio.

2.1.4 Caso de uso 5: Aplicar transformaciones de audio

Actor: Musicologo

Descripción: Con base en el AST generado, se aplican las transformaciones correspondientes al archivo de audio. Estas pueden incluir modificaciones en volumen, recorte de fragmentos, repetición de segmentos o concatenación de varios archivos.

2.1.5 Caso de uso 6: Recortar fragmentos específicos

Actor: Musicologo

Descripción: Se identifican y seleccionan fragmentos específicos del archivo de audio para ser recortados según las indicaciones del usuario.

2.1.6 Caso de uso 7: Aumentar o disminuir el volumen

Actor: Musicologo

Descripción: Se ajusta la intensidad del sonido del archivo según los parámetros especificados, ya sea aumentando o disminuyendo el volumen.

2.1.7 Caso de uso 8: Repetir fragmentos de audio

Actor: Musicologo

Descripción: Se seleccionan fragmentos específicos dentro del audio y se duplican para generar repeticiones, según las necesidades del usuario.

2.1.8 Caso de uso 9: Concatenar varios archivos de audio

Actor: Musicologo

Descripción: Se combinan múltiples archivos de audio en un solo archivo, siguiendo el orden definido por el usuario.

2.1.9 Caso de uso 10: Guardar el resultado en un nuevo archivo

Actor: Sistema de archivos

Descripción: El sistema de archivos almacena el archivo de audio procesado, asegurándose que los cambios realizados se conserven para su uso posterior. siguiendo el orden definido por el usuario.

2.1.10 Caso de uso 11: Mostrar información del nuevo archivo

Actor: Musicologo

Descripción: Se muestra al usuario información relevante sobre el archivo de audio procesado, como su duración, formato, tamaño y cualquier otra modificación realizada.

2.1.11 Caso de uso 12: Exportar nuevo audio

Actor: Sistema de archivos

Descripción: El archivo de audio finalizado se exporta en un formato compatible para que el usuario pueda utilizarlo según lo requiera.

3. Diseño de la solución propuesta

Tabla 3. Alcance de validación de errores

| Ejemplo de expresión | Ejemplo de salida |
|----------------------|-------------------|
|----------------------|-------------------|

| | |
|---|--|
| <p>Error 1.</p> <p>Formato de audio no soportado</p> | <pre>Bienvenido a Musicologo. Ingrese los comandos que desee. >>> cargar archivos/musicalaura.wav como miAudioNoSoportado Error: Solo se soportan archivos .mp3.</pre> |
| <p>Error 2</p> <p>Intento de uso de un audio no cargado aún</p> | <pre>>>> si duracion miAudioNoCargado > 05:00 entonces { dividir 02:30 miAudioNoCargado como subParte } Error: El ID no existe.</pre> |

3.1 Diagrama de casos de usos



Figura 3. Diagrama de casos de uso

El sistema permite a los usuarios cargar y procesar archivos de audio mediante diferentes operaciones. El usuario principal es el **musicólogo**, quien interactúa con el sistema para analizar, transformar y exportar archivos de audio.

1. **Carga de archivos:** El usuario selecciona y carga un archivo de audio al sistema, el cual valida su compatibilidad antes de permitir su procesamiento.
2. **Análisis y procesamiento:** Se analiza la sintaxis de las instrucciones de audio, se tokenizan y se genera un árbol de sintaxis abstracta (AST) para estructurar las operaciones a realizar.
3. **Transformaciones de audio:** Con base en el AST, se aplican modificaciones como recortes, ajustes de volumen, repetición, concatenación y silenciamiento de fragmentos.
4. **Gestión del almacenamiento y exportación:**
 - El **sistema de archivos** se encarga de **guardar** el archivo procesado, asegurando que los cambios se conserven.
 - También permite **mostrar información** relevante sobre el nuevo archivo, como duración, formato y tamaño.
 - Finalmente, posibilita la **exportación** del audio en un formato compatible para su uso posterior.

3.2 Diagrama de clases

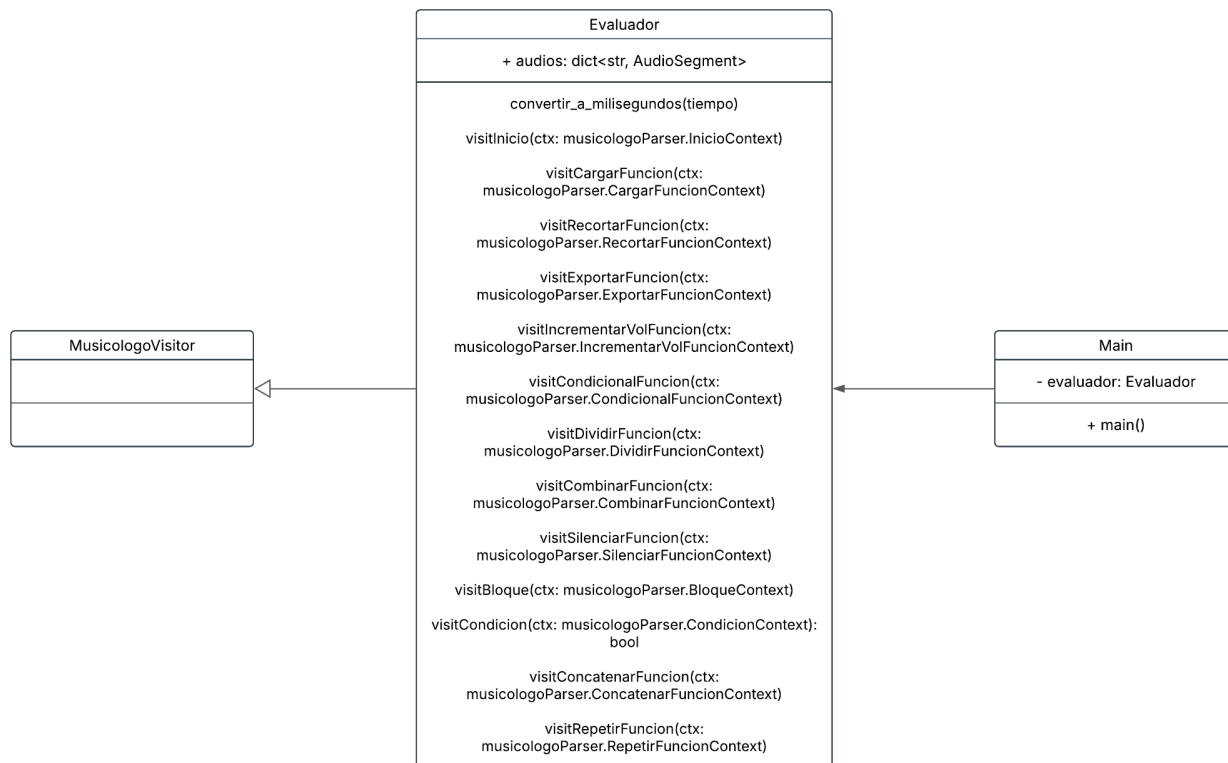


Figura 4. Diagrama de clases

Descripción de las Clases

1. Clase Evaluador

La clase **Evaluador** es el núcleo del sistema y se encarga de interpretar y procesar las instrucciones del usuario relacionadas con la manipulación de archivos de audio. Esta clase:

- Hereda de **MusicologoVisitor**, una clase generada por ANTLR4, permitiendo implementar el patrón Visitor para recorrer y evaluar el árbol sintáctico abstracto (AST).
- Sobrescribe distintos métodos como, **visitCargarFuncion**, **visitExportarFuncion**, entre otros, para manejar operaciones específicas sobre los archivos de audio.
- Mantiene un diccionario **audios: dict[str, AudioSegment]** donde almacena los archivos procesados, permitiendo su posterior manipulación.
- Incluye el método **convertir_a_milisegundos(tiempo)**, que se encarga de convertir valores de tiempo a milisegundos para facilitar el procesamiento de audio.

2. Clase Main

La clase **Main** actúa como clase de control y representa el punto de entrada del sistema. Su principal función es instanciar **Evaluador** y enviarle las instrucciones del usuario para su procesamiento.

Este diagrama de clases. Se relaciona de la siguiente manera:

La clase Evaluador hereda del visitor generado por Antlr4 y sobrescribe sus funciones con el objetivo de implementar sus funcionalidades.

La clase Main funciona como clase control y llama a su instancia de Evaluador para que evalúe las entradas del usuario.

4. Documentación de la implementación

4.1 Gramática del lenguaje

El lenguaje definido por esta gramática permite la evaluación y manipulación de archivos de audio mediante una serie de comandos específicos. Soporta operaciones básicas como carga, recorte, exportación y manipulación del volumen, además de estructuras de control condicionales y de repetición. Las expresiones en este lenguaje siguen una sintaxis clara y estructurada, basada en comandos que pueden incluir parámetros como identificadores de archivos, tiempos y valores de volumen. Dentro de las operaciones que soporta nuestra gramática son:

- **Carga y exportación:** Importa y guarda archivos MP3.
- **Edición de audio:** Recorta, silencia, concatena y combina fragmentos.
- **Ajustes de volumen:** Incrementa el volumen en un archivo completo o en un rango específico.
- **Estructuras de control:** Permite condicionales (**si-entonces**) y ciclos (**repetir**)

4.1.1 Regla de inicio

La evaluación de la gramática comienza con la regla de inicio, la cual establece que una secuencia de expresiones debe ser procesada hasta el final del archivo (EOF). Esto garantiza que el análisis sintáctico recorrerá todas las expresiones hasta que se encuentre con el final del archivo, asegurando que toda la entrada sea evaluada correctamente.

```
inicio: expresion* EOF;
```

Figura 5. Regla de inicio de la gramática

4.1.2 Expresiones

Las expresiones de esta gramática representan comandos específicos para manipular archivos de audio. Cada expresión se compone de una combinación de comandos y parámetros, lo que permite definir distintas operaciones sobre archivos MP3. Las expresiones posibles incluyen:

Carga de archivos de audio: Permite cargar un archivo en formato MP3 y asignarlo a un identificador. En esta expresión, COMANDO_CARGAR representa el comando de carga de un archivo de audio (cargar, load, crg, ld), RUTA? indica que la ruta donde se encuentra el archivo es opcional, ARCHIVO_MP3 es el nombre del archivo MP3, y COMANDO-ASIGNAR es una palabra clave (como, as) que indica que el archivo se almacenará con un identificador ID el cual es el identificador del archivo cargado.

```
expresion: COMANDO_CARGAR ' ' RUTA? ARCHIVO_MP3 ' ' COMANDO_ASIGNAR ' ' ID #cargarFuncion
```

Figura 6. Cargar un archivo

Recorte de audio: Extrae un fragmento específico de un archivo en función de un intervalo de tiempo. En esta expresión, COMANDO_RECORTAR representa el comando de recorte (recortar, cut, rct, ct), TIEMPO indica el inicio y el fin del fragmento a extraer, ID corresponde al archivo de audio sobre el cual se aplicará el recorte, y COMANDO_ASIGNAR junto con un ID permiten almacenar el fragmento resultante con un nuevo nombre.

```
| COMANDO_RECORTAR ' ' TIEMPO ' ' TIEMPO ' ' ID ' ' COMANDO_ASIGNAR ' ' ID #recortarFuncion
```

Figura 7. Recortar un audio

Exportación de audio: Permite guardar un archivo de audio procesado. En esta expresión, COMANDO_EXPORTAR representa el comando de exportación (exportar, export, exp, ex) e ID corresponde al archivo de audio que será exportado.

```
| COMANDO_EXPORTAR ' ' ID #exportarFuncion
```

Figura 8. Exportación de audio

Modificación del volumen: incrementa o disminuye el volumen de un archivo de audio, con la opción de aplicar el cambio en un intervalo de tiempo determinado. En esta expresión, COMANDO_INCREMENTAR_VOL es el comando que permite subir el volumen (subirVol), ID representa el archivo de audio a modificar, VOLUMEN indica la

cantidad de volumen a aumentar o disminuir (por ejemplo, 5dB), TIEMPO (opcional) permite aplicar el cambio solo en un intervalo de tiempo específico, y COMANDO_ASIGNAR con ID (opcional) almacena el resultado con un nuevo nombre.

```
| COMANDO_INCREMENTAR_VOL ' ' ID ' ' VOLUMEN ( ' ' TIEMPO ' ' TIEMPO)? ( ' ' COMANDO_ASIGNAR ' ' ID)? #incrementarVolFuncion
```

Figura 9. Modificación de volumen

Uso de condicionales: Permite ejecutar expresiones según características del audio, como su duración o volumen. En esta expresión, CONDICIONAL representa el comando condicional (si, if), condicion es una expresión lógica basada en características del audio (duración y volumen), HACER es una palabra clave (entonces, then) que indica el inicio del bloque de ejecución, bloque es un conjunto de instrucciones que se ejecutarán si la condición es verdadera y ELSE bloque (opcional) representa un bloque alternativo a ejecutar si la condición no se cumple.

```
| CONDICIONAL ' ' condicion ' ' HACER ' ' bloque ( ' ' ELSE ' ' bloque)? #condicionalFuncion
```

Figura 10. Uso de condicionales

División de archivos: Separa un archivo de audio en partes según un tiempo específico. En esta expresión, COMANDO_DIVIDIR es el comando para dividir un archivo de audio (dividir, split, div), TIEMPO indica el punto de tiempo en el que se realizará la división, ID corresponde al archivo de audio a dividir y COMANDO_ASIGNAR junto con ID asigna las partes resultantes a un nuevo identificador.

```
| COMANDO_DIVIDIR ' ' TIEMPO ' ' ID ' ' COMANDO_ASIGNAR ' ' ID #dividirFuncion
```

Figura 11. División de archivos

Combinación de archivos: Permite unir dos archivos de audio en uno solo. En esta expresión, COMANDO_COMBINAR representa el comando que permite unir dos archivos de audio (combinar, mix, comb), ID es el primer archivo a combinar, ID es el segundo archivo de audio a combinar, y COMANDO_ASIGNAR junto con ID asignan un identificador al archivo resultante.

```
| COMANDO_COMBINAR ' ' ID ' ' ID ' ' COMANDO_ASIGNAR ' ' ID #combinarFuncion
```

Figura 12. Combinación de archivos

Silenciamiento de partes: Aplica silencio en un intervalo específico del audio. En esta expresión, COMANDO_SILENCIAR representa el comando utilizado para aplicar silencio en un intervalo de tiempo (silenciar, mute, sil), TIEMPO indica el tiempo de inicio del silenciamiento, TIEMPO indica el tiempo de finalización del silenciamiento e ID es el archivo de audio en el que se aplicará el silencio.

```
| COMANDO_SILENCIAR ' ' TIEMPO ' ' TIEMPO ' ' ID #silenciarFuncion
```

Figura 13. Silenciamiento de partes

Concatenación de archivos: Permite repetir y unir un archivo varias veces. En esta expresión, COMANDO_CONCATENAR representa el comando para concatenar un

archivo de audio (concatenar, concat), ID es el archivo de audio que será concatenado, NUMERO indica la cantidad de veces que se repetirá el audio y COMANDO_ASIGNAR junto con ID asignan un identificador al archivo resultante.

```
| COMANDO_CONCATENAR ' ' ID ' ' NUMERO ' ' COMANDO_ASIGNAR ' ' ID #concatenarFuncion
```

Figura 14. Concatenación de archivos

Repetir: La repetición de bloques permite ejecutar un bloque de expresiones varias veces. En esta expresión, COMANDO_REPETIR representa el comando que repite un bloque de expresiones (repetir, repeat), NUMERO indica el número de veces que se ejecutará el bloque, HACER es la palabra clave (entonces, then) que señala el inicio del bloque, y bloque es el conjunto de expresiones que se ejecutarán repetidamente.

```
| COMANDO_REPETIR ' ' NUMERO ' ' HACER ' ' bloque #repetirFuncion  
;
```

Figura 15. Repetición de un bloque de instrucciones

4.1.3 Bloque de expresiones

Los bloques de expresiones permiten agrupar múltiples expresiones dentro de llaves {} o ejecutar una sola expresión directamente. Las expresiones dentro de un bloque pueden

estar separadas por '&&', lo que indica una ejecución secuencial.

```
bloque: '{ ' '* expresion ( ' && ' expresion)* ' '* }' | expresion;
```

Figura 16. Definición de bloque de expresiones

4.1.4 condiciones

Las condiciones permiten evaluar propiedades de un archivo de audio, como su duración o volumen. Estas condiciones se expresan mediante operadores de comparación (>, <, >=, etc.), lo que permite ejecutar comandos de manera condicional. Además, cada condición representa una propiedad del archivo de audio, como duración o volumen, y utiliza un ID que corresponde al nombre del archivo de audio al que se evaluará la característica.

```
condicion: CARACTERISTICA ' ' ID ' ' OPERADOR ( ' ' TIEMPO | ' ' VOLUMEN);
```

Figura 17. Condición

4.1.5 Funciones principales: los comandos principales de la gramática definen las operaciones que pueden realizarse sobre los archivos de audio. COMANDO_CARGAR permite cargar un archivo MP3 en memoria, utilizando palabras clave como ‘cargar’, ‘load’, ‘crg’ o ‘ld’. COMANDO_RECORTAR facilita la extracción de un fragmento de audio, empezando términos como ‘recortar’, ‘cut’, ‘rct’ o ‘ct’. Para exportar un archivo de audio procesado, se usa COMANDO_EXPORTAR, con palabras clave como

‘exportar’, ‘export’, ‘exp’ o ‘ex’. COMANDO_INCREMENTAR_VOL se encarga de modificar el volumen de un archivo, utilizando la palabra clave ‘subirVol’. COMANDO_ASIGNAR (‘como’ o ‘as’) permite asignar un identificador a los archivos o fragmentos generados. COMANDO_DIVIDIR, con opciones como ‘dividir’, ‘split’ o ‘div’, separa un archivo en partes según un tiempo determinado. Para fusionar dos archivos se emplea COMANDO_COMBINAR, con términos como ‘combinar’, ‘mix’ o ‘comb’. COMANDO_SILENCIAR aplica silencio a un segmento de audio, con opciones como ‘silenciar’, ‘mute’ o ‘sil’. COMANDO_CONCATENAR, con ‘concatenar’ o ‘concat’, une un archivo de audio consigo mismo varias veces.

```
// Funciones principales
COMANDO_CARGAR: 'cargar' | 'load' | 'crg' | 'ld';
COMANDO_RECORTAR: 'recortar' | 'cut' | 'rct' | 'ct';
COMANDO_EXPORTAR: 'exportar' | 'export' | 'exp' | 'ex';
COMANDO_INCREMENTAR_VOL: 'subirVol';
COMANDO_ASIGNAR: 'como' | 'as';
COMANDO_DIVIDIR: 'dividir' | 'split' | 'div';
COMANDO_COMBINAR: 'combinar' | 'mix' | 'comb';
COMANDO_SILENCIAR: 'silenciar' | 'mute' | 'sil';
COMANDO_REPETIR: 'repetir' | 'repeat';
COMANDO_CONCATENAR: 'concatenar' | 'concat';
```

Figura 18. Funciones principales

4.1.6 Elementos condicionales

Las condiciones en la gramática permiten evaluar propiedades de los archivos de audio y ejecutar comandos en función de su duración o volumen. CONDICIONAL, representado por 'si' o 'if', indica el inicio de una estructura condicional. HACER, con las palabras clave 'entonces' o 'then', marca el inicio del bloque de instrucciones que

se ejecutará si la condición es verdadera. Si la condición no se cumple, se puede definir un bloque alternativo con ELSE, usando 'sino' o 'else'. Para comparar valores, se emplean distintos OPERADORES, como '>', '<', '>=', '<=', '==' (igualdad) y '!=' (diferente). Finalmente, CARACTERISTICA permite especificar qué propiedad del archivo de audio será evaluada, ya sea su 'duración' ('duracion', 'dur', 'd') o su 'volumen' ('vol', 'v').

```
// Elementos para condiciones
CONDICIONAL: 'si' | 'if';
HACER: 'entonces' | 'then';
ELSE: 'sino' | 'else';
OPERADOR: '>' | '<' | '>=' | '<=' | '==' | '!=';
CARACTERISTICA: 'duración' | 'duracion' | 'dur' | 'd' | 'volumen' | 'vol' | 'v';
```

Figura 19. Elementos para condiciones

4.1.7. REGLAS SINTÁCTICAS

4.1.7.1 Volumen:

Este elemento representa los valores de volumen que pueden ser utilizados en los comandos de la gramática. Un valor de volumen puede ser un número positivo o negativo, lo que permite tanto aumentar como disminuir el nivel del audio. Además, debe ir acompañado de la unidad de medida 'dB', la cual puede escribirse en diferentes combinaciones de mayúsculas y minúsculas como 'dB', 'db', 'DB' o 'Db'.


```
VOLUMEN: ('-')?[0-9]+ ('dB' | 'db' | 'DB' | 'Db');
```

Figura 19. volumen

4.1.7.2 Archivo_MP3

Este elemento define el formato para los nombres de archivos de audio en la gramática. Un nombre de archivo MP3 puede estar compuesto por letras y números, pero debe terminar obligatoriamente con la extensión ' .mp3 '. Esto garantiza que solo se trabajen archivos de audio compatibles dentro del sistema.

```
ARCHIVO_MP3: [a-zA-Z0-9]+ '.mp3';
```

Figura 20. Archivo mp3

4.1.7.3 ID

Los identificadores (ID) se utilizan para asignar nombres a archivos cargados y a resultados de operaciones dentro del sistema. Un identificador válido debe comenzar con una letra y puede contener tanto letras como números. Sin embargo, no se permiten caracteres especiales ni espacios, lo que garantiza una nomenclatura uniforme y sin errores en la ejecución de comandos.

```
ID: [a-zA-Z][a-zA-Z0-9]*; // Empieza con una letra y puede contener números
```

Figura 21. identificadores

4.1.7.4 RUTA

El elemento RUTA representa una dirección en la que se encuentra un archivo dentro del sistema de archivos. Está compuesta por segmentos de letras y números, separados por ' / '. Esto permite especificar correctamente ubicaciones dentro de estructuras de carpetas.

```
RUTA: ([a-zA-Z0-9]+ '/' )+;
```

Figura 22 . Ruta

4.1.7.5 TIEMPO

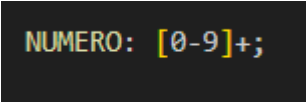
: El formato de tiempo se define para indicar marcas de tiempo dentro de los archivos de audio. Se utiliza en operaciones como el recorte, división o aplicación de efectos. La estructura permitida es 'MM:SS' (minutos y segundos), aunque opcionalmente se puede incluir el formato de tres niveles 'HH:MM:SS' (horas, minutos y segundos).

```
TIEMPO: [0-9]+ ':' [0-9]+ (':' [0-9]+)?;
```

Figura 23. Tiempo

4.1.7.6 NUMERO:

Este elemento representa cualquier número entero positivo. Se utiliza en varios comandos, como la cantidad de repeticiones en la concatenación de archivos o la especificación de volúmenes en dB cuando no incluyen signo negativo.

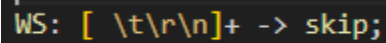


```
NUMERO: [0-9]+;
```

Figura 24. Numero

4.1.7.7 WS (Espacios en blanco):

Este elemento define los espacios en blanco, tabulaciones y saltos de línea dentro de la gramática. Su función es ser ignorado durante el análisis sintáctico, lo que permite escribir los comandos con mayor claridad sin afectar su interpretación.



```
WS: [ \t\r\n]+ -> skip;
```

Figura 25. Manejo de espacios en blanco

4.2 Evaluador al visitador

4.3 Manual de usuario

A continuación se muestran las funciones que se pueden utilizar con nuestro intérprete y cómo usarlas.

4.3.1 Cargar

4.3.1.1 Gramatica

```
expresion: COMANDO_CARGAR ' ' RUTA? ARCHIVO_MP3 ' ' COMANDO_ASIGNAR ' ' ID
```

4.3.1.2 Ejemplo de uso

```
cargar "musica/cancion.mp3" como miAudio
```

4.4.2 Recortar

4.4.2.1 Gramatica

```
expresion: COMANDO_RECORTAR ' ' TIEMPO ' ' TIEMPO ' ' ID ' ' COMANDO_ASIGNAR ' ' ID;
```

4.4.2.2 Ejemplo de uso

```
recortar 00:30 01:15 miAudio como fragmento1
```

4.4.3 Exportar

4.4.3.1 Gramatica

```
expresion: COMANDO_EXPORTAR ' ' ID;
```

4.4.3.2 Ejemplo de uso

```
exportar fragmento1
```

4.4.4 Subir volumen

4.4.4.1 Gramatica

```
expresion: COMANDO_INCREMENTAR_VOL ' ' ID ' ' VOLUMEN ( ' ' TIEMPO ' ' TIEMPO)? ( ' ' COMANDO_ASIGNAR ' ' ID)?;
```

4.4.4.2 Ejemplo de uso

```
subirVol miAudio 5dB 00:10 00:50 como audioAmplificado
```

4.4.5 Condicional

4.4.5.1 Gramatica

```
expresion: CONDICIONAL ' ' condicion ' ' HACER ' ' bloque ( ' ' ELSE ' ' bloque)?;
```

4.4.5.2 Ejemplo de uso

```
si duracion miAudio > 03:00 entonces {
    recortar 00:00 03:00 miAudio como miAudioCorto
} sino {
    exportar miAudio
}
```

4.4.6 Dividir

4.4.6.1 Gramatica

```
expresion: COMANDO_DIVIDIR ' ' TIEMPO ' ' ID ' ' COMANDO_ASIGNAR ' '
ID;
```

4.4.6.2 Ejemplo de uso

```
dividir 02:00 miAudio como segmento1
```

4.4.7 Combinar

4.4.7.1 Gramatica

```
expresion: COMANDO_COMBINAR ' ' ID ' ' ID ' ' COMANDO_ASIGNAR ' ' ID;
```

4.4.7.2 Ejemplo de uso

```
combinar fragmento1 fragmento2 como cancionFinal
```

4.4.8 Silenciar

4.4.8.1 Gramatica

```
expresion: COMANDO_SILENCIAR ' ' TIEMPO ' ' TIEMPO ' ' ID;
```

4.4.8.2 Ejemplo de uso

```
silenciar 01:30 02:00 miAudio
```

4.4.9 Concatenar

4.4.9.1 Gramatica

```
COMANDO_CONCATENAR ' ' ID ' ' NUMERO ' ' COMANDO_ASIGNAR ' ' ID
```

4.4.9.2 Ejemplo de uso

```
concatenar miCancion 4 como miCancionExtendida
```

4.4.10 Repetir

4.4.10.1 Gramatica

```
COMANDO_REPETIR ' ' NUMERO ' ' HACER ' ' bloque
```

4.4.10.2 Ejemplo de uso

```
repetir 3 entonces { cargar archivos/laura.mp3 como audio3 &&  
exportar audio3 }
```

4.4 Manual técnico

4.4.1 Descripción

Este apartado proporcionará un guía técnica para la instalación, configuración y uso del intérprete y uso del intérprete Musicólogo, desarrollado en Python y basado en la librería PyDub para la manipulación de archivos de audio en formato MP3.

4.4.2 Requisitos del sistema

- Python 3.8 o superior.
- Librerías requeridas: ANTLR4, PyDub.
- Sistema operativo: Windows, macOS o Linux.
- FFmpeg instalado y configurado.

4.4.3 Instalación

- **Clonar el repositorio:**

El link del repositorio es el siguiente <https://github.com/CokeinsZ/Musicologo>

```
git clone <URL_DEL_REPOSITORIO>
cd <NOMBRE_DEL_PROYECTO>
```

Figura 26. Clonación de repositorio

- Instalar dependencias

```
pip install pydub
pip install ffmpeg-python
```





Figura 27. instalación de dependencias


4.4.4 Configurar FFmpeg


- Descargar e instalar FFmpeg desde <https://ffmpeg.org/download.html>.
- Asegurar que la variable de entorno PATH contenga la ruta ffmpeg.

4.4.5 Estructura del proyecto

MUSICOLOGO-MAIN

- | —  **archivos** → Contiene archivos de audio .
- | —  **src** → Carpeta principal del código fuente.
- | | —  **grammar** → Contiene archivos de gramática para ANTLR4, usados para definir la sintaxis del lenguaje.
- | | —  **evaluador.py** → Módulo que evalúa las instrucciones del lenguaje Musicólogo y ejecuta las acciones correspondientes.

| | —  **main.py** → Archivo principal del proyecto, donde se inicializa el intérprete o la aplicación.

| —  **dockerfile** → Archivo de configuración para Docker, usado para contenerizar el proyecto.

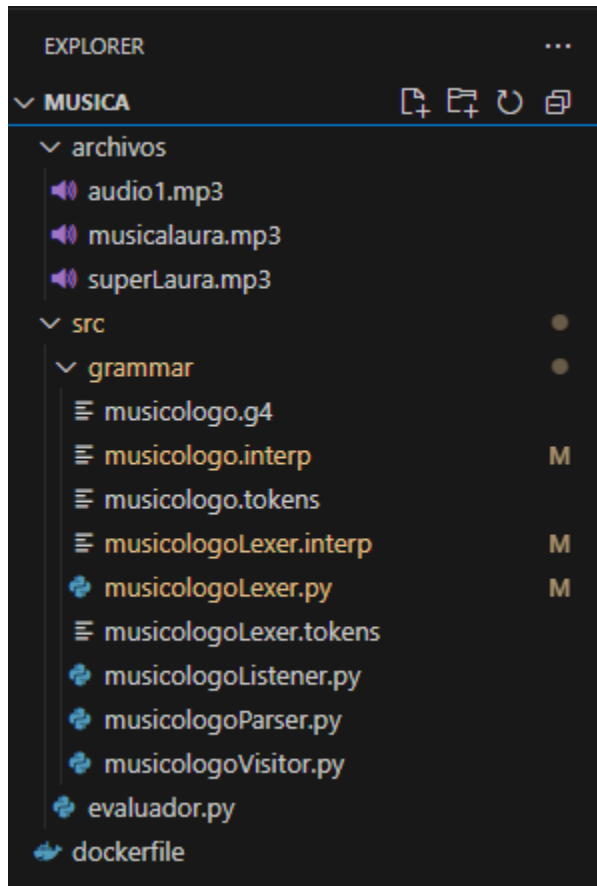


Figura 28. Estructura del proyecto

4.4.6 Uso del intérprete

Para ejecutar el intérprete, abrir una terminal en la carpeta del proyecto y ejecutar:

```
python src/main.py
```

Figura 28. Uso del interprete

El mostrará un mensaje de bienvenida y además un mensaje para que puedas ingresar los comandos para la manipulación de audio.

4.4.7 Comandos principales

| COMANDO | DESCRIPCIÓN | EJEMPLO |
|------------|-------------------------|--|
| cargar | Carga un archivo MP3 | cargar “musica/cancion.mp3” como miAudio |
| recortar | Recorta un fragmento | recortar 00:10 00:20 miAudio como miCorte |
| exportar | Guarda el audio editado | exportar miCorte |
| subirVol | Aumenta el volumen | subirVol miAudio 5dB 00:10 00:50 como audioAmplificado |
| concatenar | Une audios | concatenar miCancion 3 como miCancionExtendida |

4.4.8 Estructura interna

El intérprete se basa en ANTLR4 para la tokenización y generación del árbol de sintaxis abstracta (AST), y en PyDub para aplicar transformaciones de audio.

4.4.9 Componentes Principales

- **Lexer:** Genera tokens a partir del código fuente.
- **Parser:** Construye el AST.
- **Evaluador:** Recorre el AST y ejecuta las operaciones de audio.

4.4.10 Errores comunes y soluciones

| ERROR | POSIBLE CAUSA | SOLUCIÓN |
|-------------------------------|--------------------------------|--------------------------------------|
| formato de audio no soportado | Archivo no es MP3 | Convertir a MP3 antes de cargar |
| No se reconoce el comando | Sintaxis incorrecta | Revisar la gramática de los comandos |
| FFmpeg no encontrado | No instalado o mal configurado | verificar variable de entorno PATH |

5. Conclusiones

- Se realiza la implementación de un lenguaje específico para la manipulación de audio. La creación de Musicólogo permitió desarrollar un lenguaje específico para la manipulación de archivos de audio, facilitando operaciones como carga, edición, exportación y transformación de sonidos mediante una sintaxis estructurada y clara.
- Se hace uso de ANTLR4 para el análisis Sintáctico ya que este nos permitió definición de la gramática del lenguaje, permitiendo la identificación de tokens, la construcción del árbol de sintaxis abstracta (AST) y la validación de la estructura de las instrucciones dadas por el usuario.
- Se hace uso de la librería Pydub para hacer modificaciones con el archivo de audio como subir el volumen, silenciarlo, unificarlo, recortarlo, repetirlo.