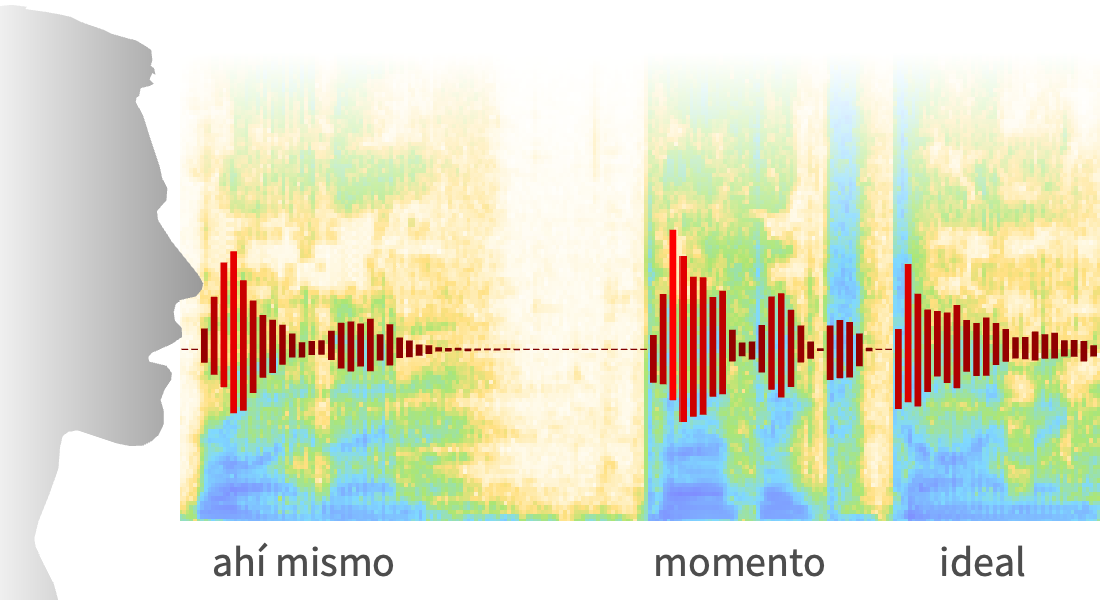


**Análisis de Audio**

Alumno: Marco Medina

Fechas: 21/06

Institución: Instituto Tecnologico Beltran

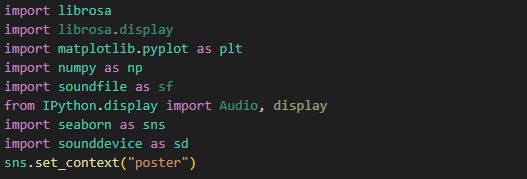


En esta práctica analizamos con scripts de Python señales de audio. El código realiza operaciones de carga, visualización, manipulación y reproducción de archivos WAV, implementando diversas bibliotecas para modificar características sonoras.

**Para llevar a cabo esto, dividimos el código en etapas para visualizar facilmente los puntos a analizar:**

1. Configuración inicial e importación de bibliotecas
2. Carga y análisis básico del archivo de audio
3. Visualización de la forma de onda
4. Manipulación de la señal mediante cuatro técnicas diferentes
5. Reproducción y almacenamiento de resultados

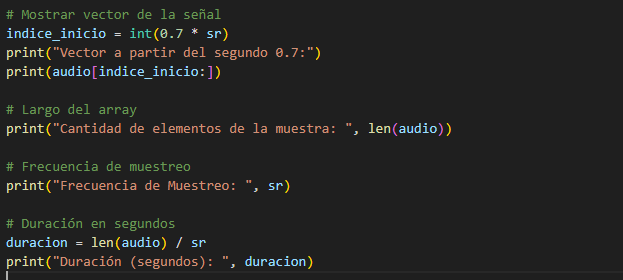
**1- En primera instancia importamos las bibliotecas que usaremos para manipular el audio.**



* **De estas las más importantes son**:
  + **librosa**: procesamiento avanzado del audio
  + **sounddevice**: reproducción en tiempo real
  + **soundfile**: manejo de archivos de audio
  + **matplotlib/seaborn**: visualización profesional
* **Cabe destacar el estilo visual**: “poster” optimiza los gráficos para presentaciones

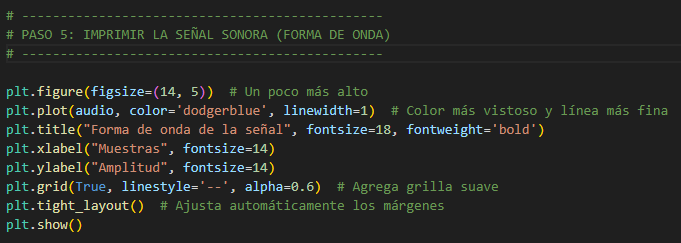
**2- Luego cargamos el archivo de audio en .WAV y analizamos las partes importantes del mismo.**

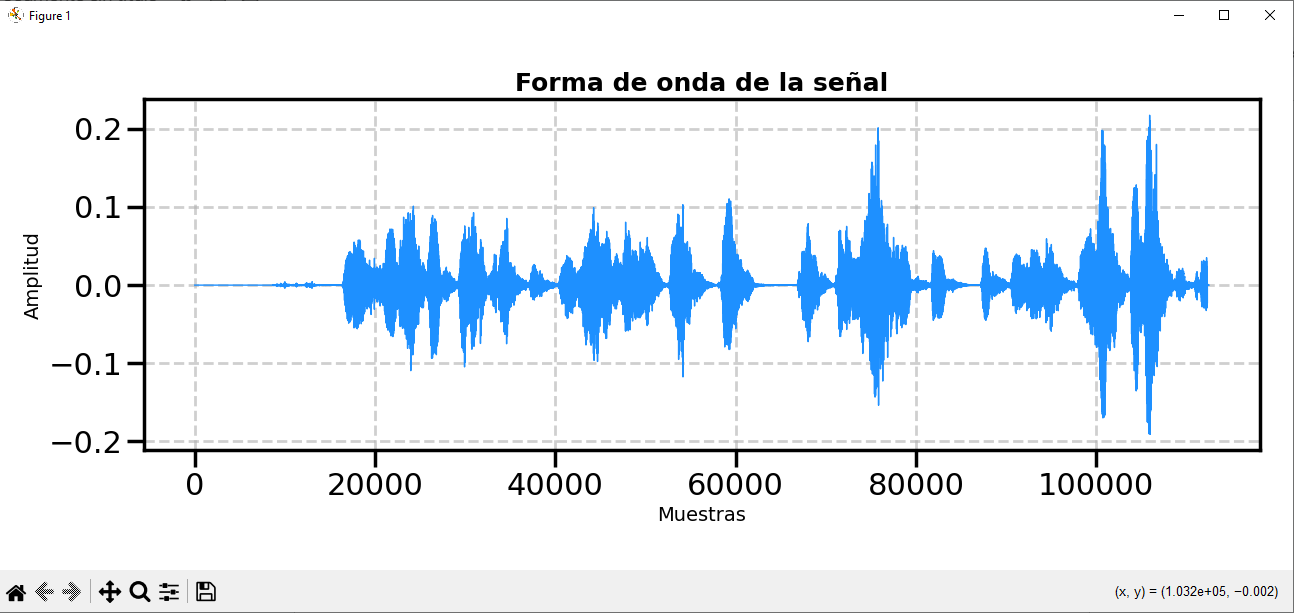
* **Se aclara las variables**:
  + **Audio**: Array Numpy con amplitudes normalizadas.
  + **sr**: Frecuencia de muestreo en Hz



* **Con el audio ya cargado, visualizamos los datos que necesitamos. Por ejemplo**:
  + indice\_inicio: que arranca en 0.7 , ya que, al haber huecos vacíos de audio se necesita vectorizar a partir que se empieza a captar el sonido.
* **Luego generamos**:
  + largo del array utilizando la función “len()”.
  + frecuencia llamando a “sr”.
  + Duración del audio en segundos, dividiendo el largo del array por la frecuencia.

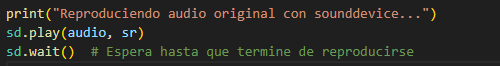
**3- Visualizamos el gráfico de las ondas de audio (Personalizando la las muestras y la amplitud con colores visibles sin ser invasivos):**

****

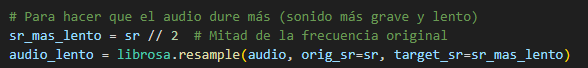
****

**4- Ahora reproducimos el audio original y luego lo manipulamos el audio para que se escuche de otras formas:**

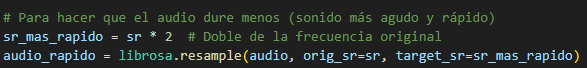
* **Audio original:**

****

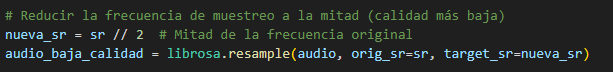
* **Audio mas largo (Sonido mas grave y lento):**

****

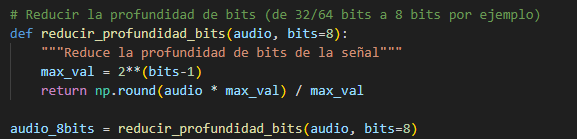
* **Audio mas corto (Sonido más agudo y rápido):**

****

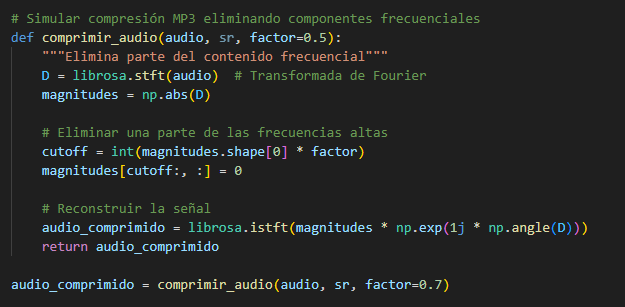
* **Audio con menos calidad (se reduce a la mitad de la frecuencia original)**

****

* **Audio con menor profundidad de bits (de 32/64 bits a 8 bits):**

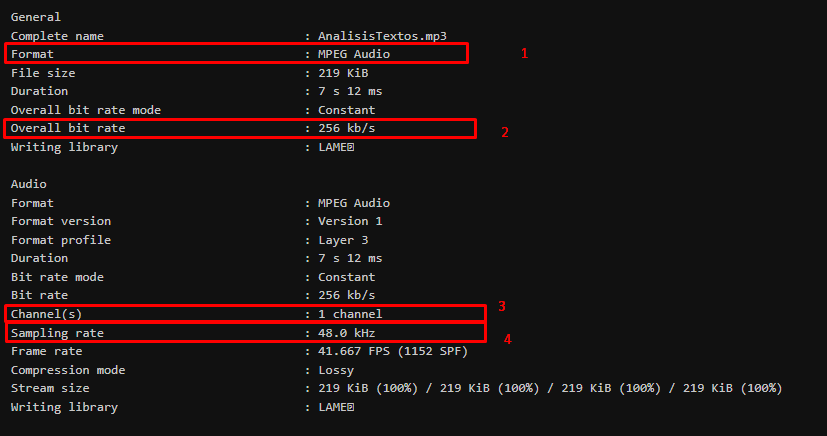
****

* **Audio que simula la compresión que tiene los audios.MP3:**

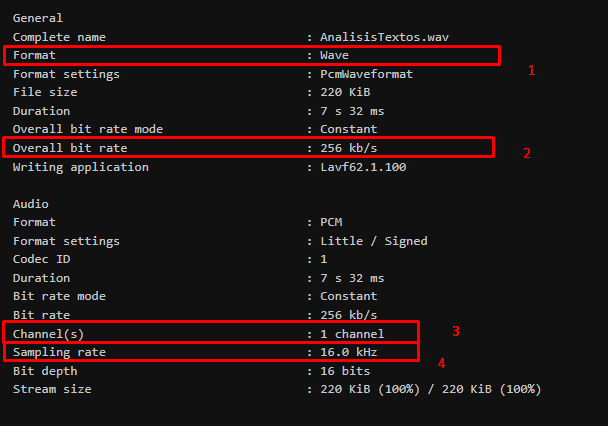
****

**Análisis del audio.MP3 Y .WAV utilizando MediaInfo**

* **Audio.MP3:**



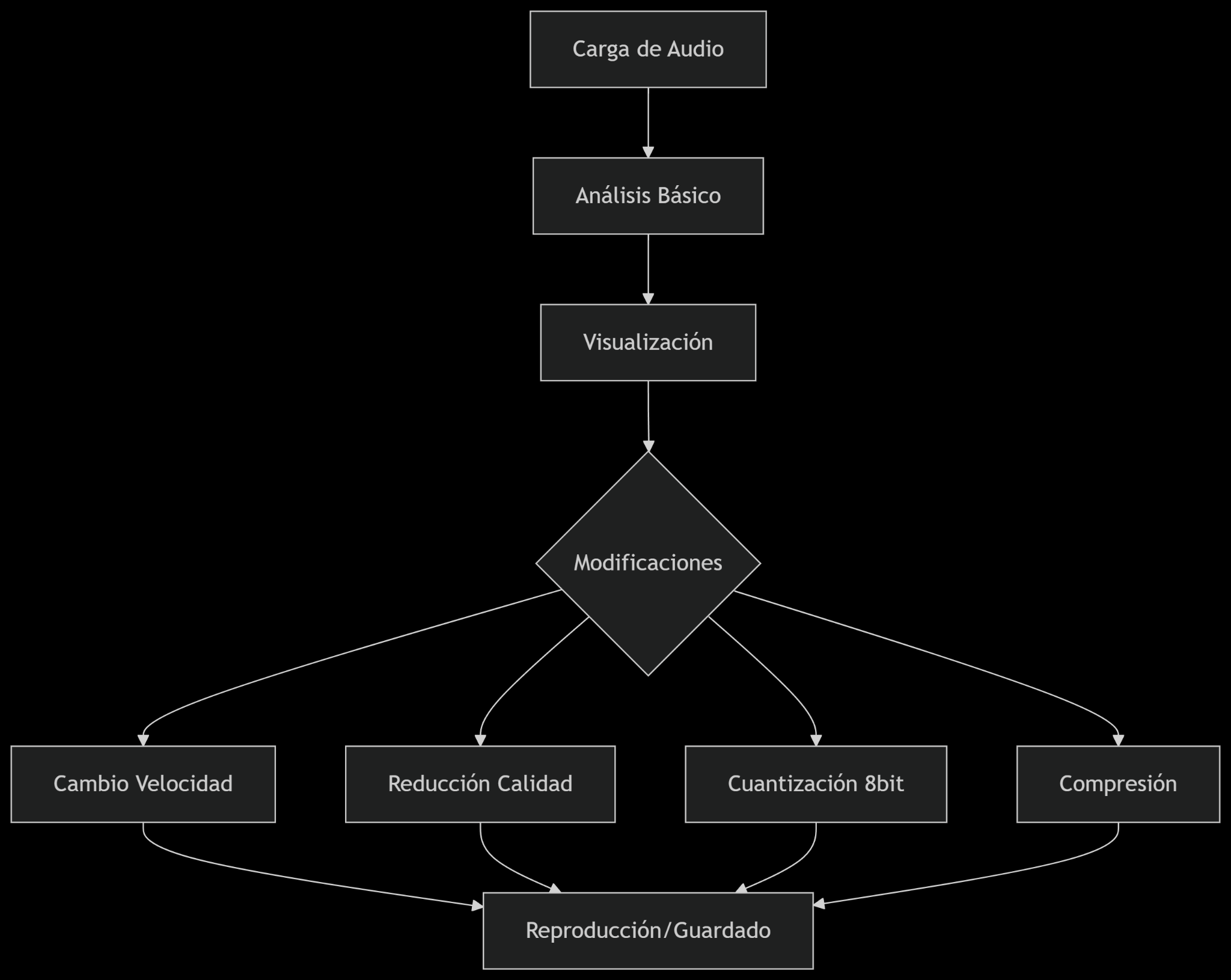
* **Audio.WAV:**



* En las anteriores imágenes se visualizamos los datos que nos brinda MediaInfo, de los cuales remarcamos: el formato(1), la tasa de bits(2), canales(3) y formato de muestreo(4).

**Conclusión**

* El código implementa un pipeline completo de procesamiento de audio digital con:
  + 4 técnicas distintas de transformación
  + Visualización profesional
  + Manejo adecuado de formatos WAV
  + Modularidad para extensiones futuras



* Las operaciones realizadas son fundamentales en aplicaciones de:
  + Producción musical digital
  + Procesamiento de voz (ASR, TTS)
  + Análisis de señales acústicas
  + Desarrollo de efectos de audio