TRABAJO PRÁCTICO FINAL

Pensamiento Computacional

Matías Luciano Antezana
Pedro Jensen
pjensen@udesa.edu.ar
mantezana@udesa.edu.ar

Tabla de Contenidos

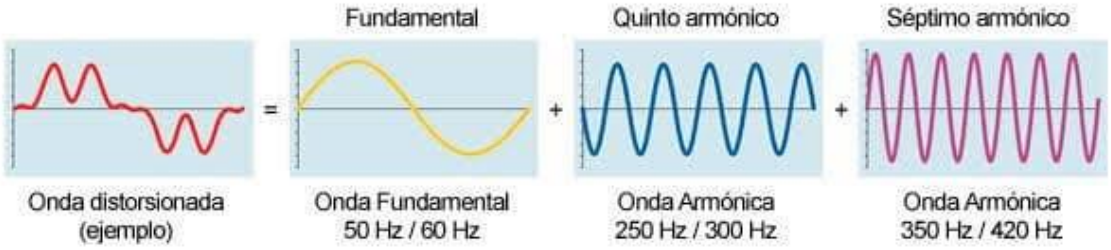
Objetivos

El trabajo Práctico tiene como objetivo explicar el desarrollo de un programa que permita la lectura de un archivo de texto donde se encuentras una partitura de una canción y a partir de esto poder realizar del desarrollo de un sintetizador y la creación de un archivo WAVE con las notas ya procesadas. En la segunda parte del trabajo se busca a partir de una partitura enviar las notas hacia un instrumento real, un metalófono e interactuar con el mismo de manera que se pueda escuchar la melodía de la canción.

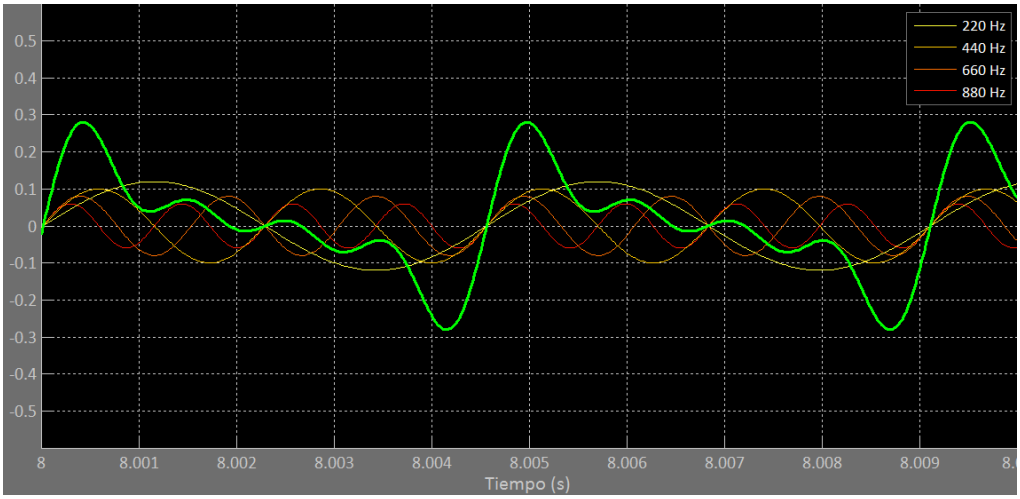
Desarrollo del Trabajo Práctico

El desarrollo del trabajo práctico viene dado la lectura de una partitura musical compuesta por notas y sus respectivos tiempos de inicio y duración. En principio mediante el uso de una terminal en donde se puedan introducir comandos y directorios de archivos se ejecuta el programa indicando el archivo de texto a leer, cuál será su frecuencia de muestreo o si se utilizara una por defecto, y el nombre del archivo de audio del tipo extensión WAVE.

Cada fragmento de dato de tiempo o nota es guardado en objetos del tipo diccionarios o listas según convenga para la funcionalidad que se requiera. Se pensó en la utilización de un objeto del tipo clase para el desarrollo de la representación del sintetizador de notas, el cual se encarga de componer las notas con la utilización del módulo numpy y la creación de arrays, los cuales permiten almacenar, similar a como lo hace una lista, múltiples valores de texto o número en una sola variable. Esto se puede observar en, por ejemplo, una gráfica del tipo senoidal con el uso de módulos que permitan graficar funciones matemáticas. Dependiendo la frecuencia de muestreo dada y el tiempo de la onda, el grafico de una onda pura se verá de una u otra forma. Las ondas generadas por un instrumento oscilarán a una frecuencia dada por su tono, y podrán generar resonancias en diferentes armónicos con diferentes amplitudes de onda. Estos armónicos sonondas periódicas, cualquiera de sus componentes sinusoidales cuya frecuencia sea un múltiplo entero de la frecuencia fundamental.



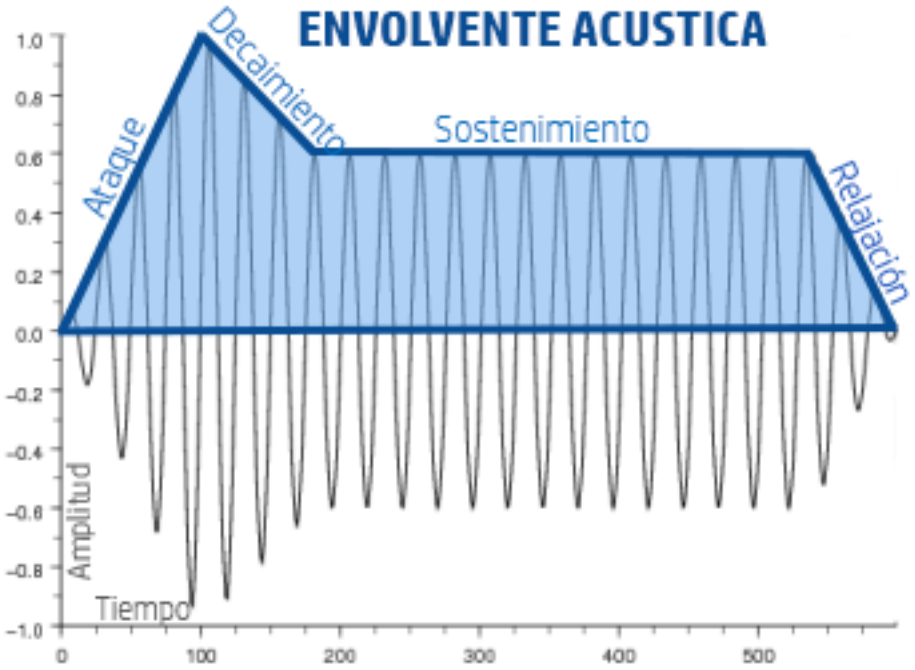
Forma de la vibración de un sonido armónico y de sus componentes:



Respecto a la intensidad del sonido, está viene dada por una serie de valores que distorsionan la forma de la onda, parte de la envolvente acústica. Estos son el ataque, el sostenimiento y el decaimiento. Además, se puede considerar el tiempo de relajación o desvanecimiento:

- Ataque: Es el tiempo de entrada, lo que tarda en escucharse el sonido después de haber sido ejecutado el instrumento.
- Decaimiento: Es el tiempo que tarda la amplitud en reducirse a la de sostenimiento, después de haber alcanzado la amplitud máxima, sin despegar la tecla o punto de inducción vibratoria.
- Sostenimiento: Después del decaimiento, es la amplitud que se mantiene constante hasta que se deja de inducir la vibración (por ejemplo, en el caso de los sintetizadores, hasta que se suelta una tecla o cable que controla este fin).
- Relajación o desvanecimiento: El tiempo que demora el sonido en perder toda su amplitud después de despegar la tecla o punto de inducción vibratoria.

Todo sonido conlleva estos periodos mencionados y definen precisamente como se escuchará el mismo. En muchos casos una mala elección de valores genera una distorsión del sonido o bien puede ocurrir la perdida de una parte de las notas.



La consolidación del sonido que se escuchará estará dada entonces por la suma de las notas, la modulación de la amplitud dada por la ADSR *(Attack, Decay, Sustain, Release*), el tipo de instrumento elegido y, por último, la frecuencia de muestreo elegida.

Una vez que se definió y estableció la forma de utilización de cada uno de los parámetros mencionados anteriormente, se pasa a la creación del archivo de audio de extensión WAVE. Esto se hace concatenando las notas lo cual resulta en un archivo de audio reproducible fácilmente en cualquier reproductor de música que admita formato WAVE (waveform audio format).

En cuanto a la segunda parte del desarrollo, en la cual se debe realizar la interacción con un instrumento real del tipo xilófono, comandado por una serie de servos que mueven un martillo e indicando con el uso de un código, que tecla debe sonar.

El planteo de esta parte del trabajo fue a partir de la lectura del repositorio provisto por los docentes para el desarrollo de la conexión con el servidor y la lectura de notas por parte del metalofono. Primero se planteó la posibilidad del desarrollo de la conversión de cualquier archivo de formato MID y poder convertirlo a formato txt. En base a esto y a que se requiere tener la partitura ordenada, se procedió al desarrollo de una función de ordenamiento según el tiempo de inicio. Luego para poder realizar la conexión remota, se deben inicializar los servidores, tanto el mock server como el server para la interacción real. Al pasarle las notas y la dirección IP de conexión respecto al host, el metalofono deberá reproducir la melodía indicada. Ya que solamente admite un cierto tipo de notas musicales, el resto de ellas serán omitidas.

Funciones:

- **MAIN:** Utiliza el módulo argparse permitiendo ejecutar el programa pasandole argumentos desde una terminal de comandos. Llama a la función menú.
- **MENU:** Luego de main, ejecuta el programa a partir de la lectura de un archivo de texto. Llama a la clase Sound y al método créate date. También a las funciones organization y create_wave_file.
- **ORGANIZATION:** Lee el archivo de texto que se le pase como argumento y separa las notas junto con sus tiempos en su objeto del tipo diccionario. Llama a la función org_dic
- **ORG_DIC:** Organiza el diccionario de acuerdo al tiempo de inicio de cada una de las notas del archivo de texto. Llama a la función separate_list.
- **SEPARATE_LIST:** Ordena en listas de forma cronológica la aparición de cada una de las notas del archivo dependiendo de tu tiempo de inicio y su duración.
- **CREATE_WAVE_FILE:** Función que crea el archivo WAVE de sonido.
- **ATTACK_SUST_DECAY_SEPARATOR:** Función que separa en listas dependiendo la función al ataque, el sostenimiento y el decaimiento.
- **HARMONICS_SEPARATOR:** Función que separa la cantidad de armónicos según la partitura leída.
- **READ_INSTRUC:** Función que lee el archivo del instrumento.
- **SEPARATE_TUPLE_VALUES_FOR3:** Función que devuelve en una lista el tipo de ataque, sostenimiento y decaimiento para los mismos que posean 3 parámetros.
- **SEPARATE_TUPLE_VALUES FOR2:** Función que devuelve en una lista el tipo de ataque, sostenimiento y decaimiento para los mismos que posean 2 parámetros.

Clases:

- **SOUND:** Clase principal que representa el sintetizador. En este caso no toma argumentos como definición.
    - **CREATE_DATA:** Define la utilización del sintetizador según los parámetros dados.
- **ATTACK_SUST_DECAY:** Clase principal que representa la envolvente de la nota.
    - **LINEAR:** Función que ejecuta el ataque.
    - **CONSTANT:** Función que ejecuta el sostenimiento devolviendo un array compuesto por unos.
    - **INVLINEAR:** Función que ejecuta el decaimiento devolviendo un array.
    - **TRI:** Función para ejecutar otro tipo de ataque.
    - **MINOR_VALUES:** Función que devuelve los valores de ataque menores o iguales dada una condición.
    - **MEDIUM_VALUES:** Función que devuelve los valores mayores que el ataque y menores que el decaimiento dada una condición.
    - **BIGGER_VALUES:** Función que devuelve los valores mayores o iguales de decaimiento dada una condición.
    - **MAIN_METHOD:** Función que llama a los otros métodos para ejecutar el ataque, sostenimiento y decaída para luego devolver la nota modificada.
    - **SIN:** Función seno.
    - **EXP:** Función exponencial.
    - **INVEXP:** Función inversa exponencial.
    - **QUARTCOS:** Función quartcos.
    - **QUARTSIN:** Función quartsin.
    - **HALFCOS:** Función halfcos.
    - **HALFSIN:** Función halfsin.
    - **LOGE:** Función logarítmica.
    - **INVLOG:** Función inversa logarítmica.
    - **TRI:** Función tri.
    - **PULSES:** Función de pulsos.

Alternativas consideradas- Estrategias adoptadas

El trabajo práctico presentó inicialmente un enfoque de armado y estructurado del mismo. Se pensó implementar el uso de funciones como método de definición de objetos, como por ejemplo la representación del sintetizador. Tuvimos en cuenta de qué forma se podía tomar las variables y manipularlas como dato de forma sencilla. En este caso la utilización de arrays o arreglos permitió realizar un manejo de esta información.

Así se pudo comenzar a utilizar funciones matemáticas y observar gráficos de ondas senoidales puras mediante la implementación de módulos. A medida que se modificaban los valores de frecuencia, frecuencia de muestreo, las ondas modificaban su aspecto. A partir de acá se comenzó a desarrollar las diferentes funciones referidas a la envolvente de los armónicos, lo cual se resolvió con el uso de condicionales y la evaluación de los parámetros en distintos intervalos de los array, lo cual determinaba ciertamente los momentos en los cuales la onda durante determinado tiempo tendría su momento de ataque, sostenimiento o decaimiento. Esto a su vez fue determinado en base a la modularización provista por los profesores.

En base a esto se realizo el procesado de datos/valores, los cuales se transformaron en un archivo de audio dependiendo la frecuencia de muestreo requerida. Alguna alternativa considerada fue, por ejemplo, la decisión por parte del usuario encargado de ejecutar el programa, la elección del tipo de función de modularización que quisiera realizar respecto al archivo leído. Otra fue la creación de clases con ciertos instrumentos musicales ya preestablecidos, y que, dado un parámetro fijo de timbre e intensidad, el sonido se reprodujera mas similar al mismo instrumento elegido.

Ya en la última parte del trabajo, se pensó la interacción entre un fragmento de código compuesto de una partitura la cual debió ser ordenada con el instrumento real, en este caos un metalofono. Lo primero que se realizo fue la conversión de cualquier archivo tipo MIDI hacia una partitura generada dentro de un archivo de texto el cual se procesa y de la misma forma que la generación del sintetizador, se guardan los datos en variables para mayor facilidad de acceso y ordenamiento. Una vez hecho esto el proceso es relativamente sencillo, se debe generar la conexión al mock server y al server para la interacción real utilizando la dirección IP propia del instrumento. Una alternativa que se tomo en cuenta para el desarrollo de esta parte fue la eliminación de las notas que no son soportadas por el instrumento, pero luego se descarto dicha idea ya que realizar esto generaría la distorsión plena de la canción que se quiera reproducir. Por lo tanto basto con ignorar las notas que no son posibles reproducir para el instrumento.

Por último, el enfoque respecto al testing de las funciones estuvo dado teniendo en cuenta lo que retorna las funciones creadas y en base a eso se pensó como y dependiendo de que parametros o variables o realizando que operación los mismos podrían arrojar un problema. Para ellos se desarrollo test de ValueError, TypeError, ZeroDivisionError, entre otros.

Problemas encontrados- Soluciones

El enfoque de problemas para el trabajo práctico estuvo dado mayormente a la hora de realizar importaciones de diferentes módulos los cuales requerían una previa instalación en el sistema. Esto se soluciono creando y ejecutando un archivo del tipo requirementes.txt el cual se encarga de instalar todos los paquetes requeridos para que el programa se ejecute correctamente. Otra de las soluciones fue instalar de forma manual cada uno de los paquetes que entraban en conflicto.

Otro de los problemas fue el llamado a directorios en los cuales archivos con diferentes funciones interactuaban entre sí, pero como no se encontraban al nivel requerido de paquete, o bien los subpaquetes no poseían un archivo del tipo __init__ el cual indica que dicho directorio corresponde a un paquete, se generaba una excepción en la cual el modulo especificado no fue encontrado. Este error fue solucionado modificando de raíz la estructura de carpeta de directorios para que así la importación de módulos se realice de forma correcta.

La ejecución de comandos también genero ciertos problemas respecto a la sintaxis de los mismos, así como también el no poder encontrar el archivo que se requiere leer ya que no se encontraba posicionado en la carpeta contenedora del mismo. Este tipo de inconveniente se soluciono entrando mediante el uso del comando cd mas la carpeta que se quiere ingresar.

Respecto a la generación del audio, se observo un inconveniente en el cual el audio generado se escuchaba muy distorsionado, esto se soluciono implementando otro modulo utilizado en archivos de audio. El manejo de arrays también genero inconvenientes cuando los mismos con coincidían en cantidad de elementos.

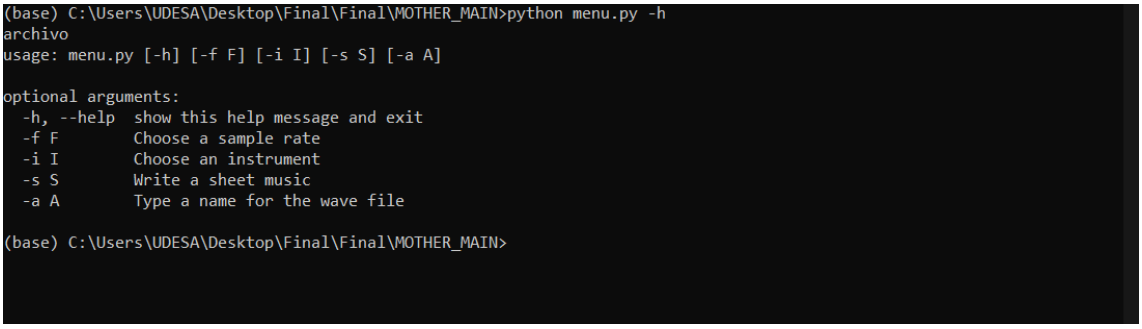
Por último, al realizar una prueba de instalación con el uso del setup, se arrojaron diversos errores en los cuales no permitía realizar la correcta instalación del paquete. Esto se corrigió con una modificación de sintaxis del mismo archivo y la adición de archivos __init__.py junto con la reestructuración de los directorios.

Indicaciones para ejecutar correctamente el programa

El programa inicia desde cualquier consola de comandos ejecutando la siguiente línea posicionado en la ruta del paquete:

Python menu.py -h

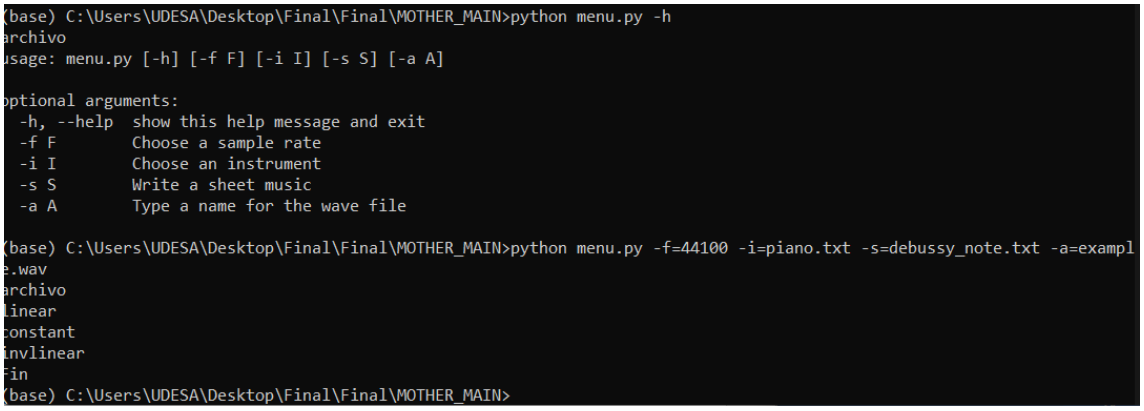
Se verán en pantalla los argumentos que se requieren de ingresar de la siguiente forma.



Para ingresar los argumentos, se debe escribir el comando de la siguiente manera:

python menu.py –f= <frecuencia de muestreo> -i=<instrumento> -s=<nombre archivo.txt> --a=<nombre del archivo WAV creado.wav>

Una vez hecho esto, se creará en la carpeta de nuestro paquete el archivo de audio.



Respecto al vínculo con el xilófono, basta con reproducir los archivos de conexión con los servers, el cargado de notas y como requerimiento se deberá pasar el host con la dirección IP propia del mismo. La ejecución de dicho archivo permitirá la reproducción del archivo de audio generado.

```
from xylophone.client import XyloClient
from xylophone.xylo import XyloNote


if __name__ == '__main__':
    notes = [
            XyloNote('A4', 1.5, 90),
            XyloNote('A4', 2.3, 90),
            XyloNote('G#6', 5.33333, 90),
            XyloNote('A4', 10.01, 90),
            ]

    client = XyloClient(host='localhost', port=8080)
    client.load(notes)
    client.play()
```

Resultado de Ejecuciones

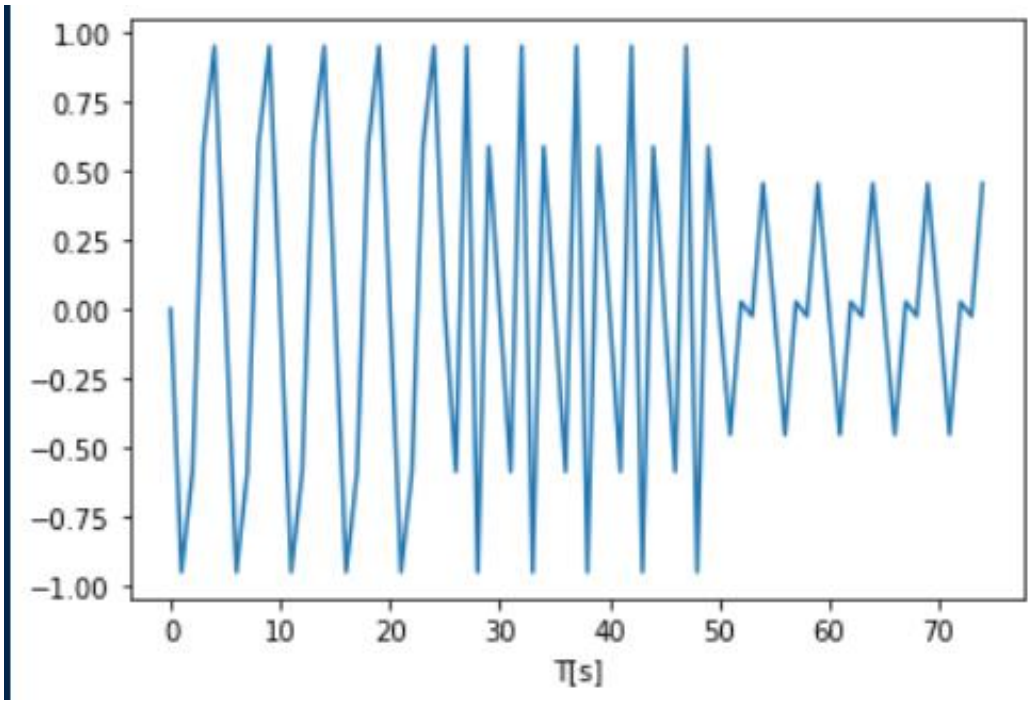
Ejecución setup.py

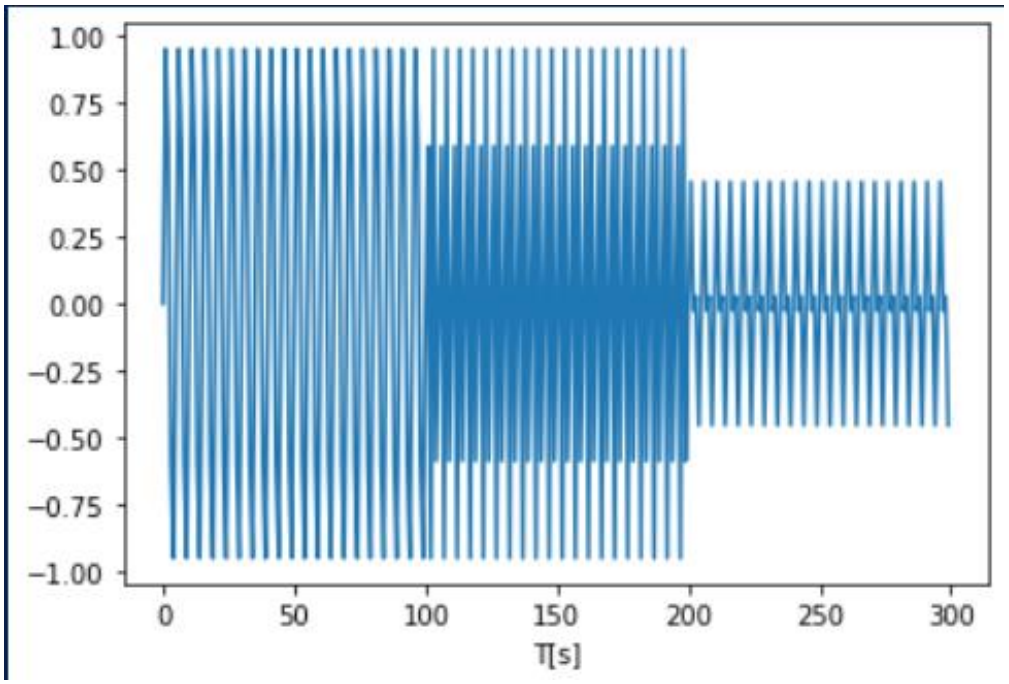
```
  Running setup.py clean for MOTHER-MAIN
Failed to build MOTHER-MAIN
Installing collected packages: MOTHER-MAIN
  Attempting uninstall: MOTHER-MAIN
    Found existing installation: MOTHER-MAIN 1.0.0
    Uninstalling MOTHER-MAIN-1.0.0:
      Successfully uninstalled MOTHER-MAIN-1.0.0
    Running setup.py install for MOTHER-MAIN ... done
  DEPRECATION: MOTHER-MAIN was installed using the legacy 'setup.py install' method, because a wheel could not be built
for it. A possible replacement is to fix the wheel build issue reported above. You can find discussion regarding this at
 https://github.com/pypa/pip/issues/8368.
Successfully installed MOTHER-MAIN-1.0.0

(base) C:\Users\UDESA\Desktop\Final\Final>
```

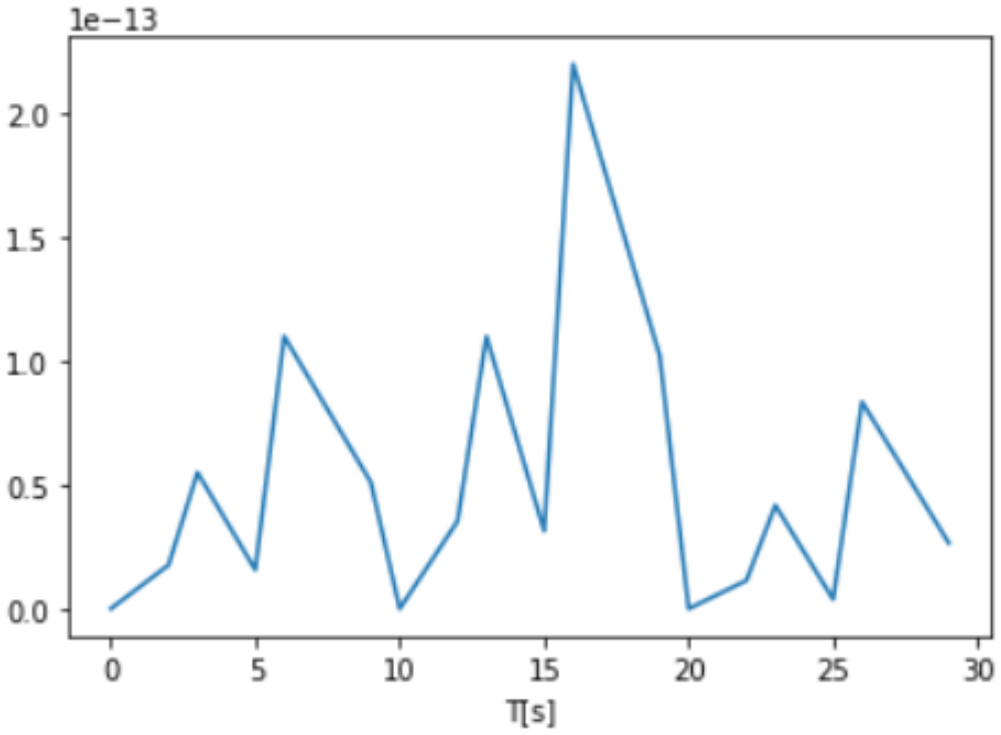
Ejecución sintetizador:

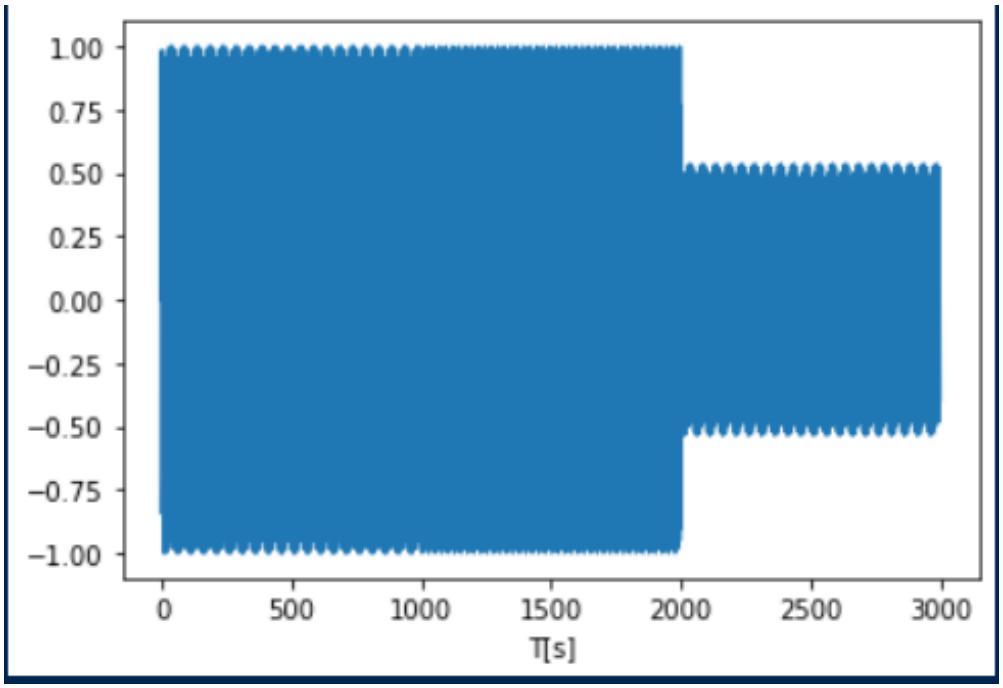
```
(base) C:\Users\UDESA\Desktop\Final\Final\MOTHER_MAIN>python menu.py -h
archivo
usage: menu.py [-h] [-f F] [-i I] [-s S] [-a A]

optional arguments:
  -h, --help  show this help message and exit
  -f F        Choose a sample rate
  -i I        Choose an instrument
  -s S        Write a sheet music
  -a A        Type a name for the wave file

(base) C:\Users\UDESA\Desktop\Final\Final\MOTHER_MAIN>python menu.py -f=44100 -i=piano.txt -s=debussy_note.txt -a=exampl
e.wav
archivo
linear
constant
invlinear
fin
(base) C:\Users\UDESA\Desktop\Final\Final\MOTHER_MAIN>
```

Gráficos de las ondas puras variando parámetro T[s]:






1e−13
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30
T[s]


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
500
1000
1500
2000
2500
3000
T[s]

```
----------------------------------------------------------------------
Ran 15 tests in 0.007s

OK
PS C:\Users\UDESA\Desktop\final 3-48am\final\MOTHER_MAIN> & C:/Users/UDESA/anaconda3/python.exe "c:/Users/UDESA/Desktop/final 3-48am/Final/MOTHER_MAIN/test_menu.py"
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
PS C:\Users\UDESA\Desktop\final 3-48am\final\MOTHER_MAIN>
```

```
Ran 3 tests in 0.001s

OK
PS C:\Users\UDESA\Desktop\final 3-48am\final\MOTHER_MAIN> & C:/Users/UDESA/anaconda3/python.exe "c:/Users/UDESA/Desktop/final 3-48am/Final/MOTHER_MAIN/test_wave_creator.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.008s

OK
PS C:\Users\UDESA\Desktop\final 3-48am\final\MOTHER_MAIN>
```

Error de archivo:

```
(base) C:\Users\UDESA\Desktop\final 3-48am\Final\MOTHER_MAIN>python menu.py
Traceback (most recent call last):
  File "C:\Users\UDESA\Desktop\final 3-48am\Final\MOTHER_MAIN\menu.py", line 60, in <module>
    main()
  File "C:\Users\UDESA\Desktop\final 3-48am\Final\MOTHER_MAIN\menu.py", line 25, in main
    sys.stdout.write(str(menu(args)))
  File "C:\Users\UDESA\Desktop\final 3-48am\Final\MOTHER_MAIN\menu.py", line 43, in menu
    list_org = organization(args.s)
  File "C:\Users\UDESA\Desktop\final 3-48am\Final\MOTHER_MAIN\read_score.py", line 10, in organization
    with open(filename,"r") as debussy:
FileNotFoundError: [Errno 2] No such file or directory: 'debussy_note.txt'

(base) C:\Users\UDESA\Desktop\final 3-48am\Final\MOTHER_MAIN>
```

Bibliografía

- Sitio web dedicado a la documentación del programa de Python.

  https://docs.python.org/3/tutorial/

- Repositorios GIT relacionados a la temática del trabajo práctico.

  https://github.com/yuma-m/synthesizer

  https://python.plainenglish.io/making-a-synth-with-python-oscillators-2cb8e68e9c3b

  https://python.plainenglish.io/build-your-own-python-synthesizer-part-2-66396f6dad81

  https://github.com/atheler/klang

  https://jythonmusic.me/envelope/

- Testing.

  https://www.clubdetecnologia.net/cursos/pruebas-con-python/escribir-y-ejecutar-un-test/

  https://geekflare.com/es/unit-testing-with-python-unittest/

  https://ellibrodepython.com/python-testing

  https://docs.python.org/es/3/library/test.html

- Esquema y formato de repositorios Git.

  https://www.delftstack.com/es/howto/python/python-setup.py/

https://github.com/datosgobar/estandares/blob/master/codigo/README.md

https://github.com/datosgobar/estandares/blob/master/codigo/python/Project-Example-1/package_name/test_project_euler.py

https://docs.python.org/3/howto/argparse.html

- Otros.

  https://es.stackoverflow.com/questions/305964/c%C3%B3mo-reproducir-notas-musicales-con-python

  https://crypto.stanford.edu/~blynn/sound/karplusstrong.html

  http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html

  https://numpy.org/doc/