# Informática Musical

Procesamiento de audio digital: NumPy, SoundDevice, SciPy.

Los ejercicios marcados con **(Evaluable)** son prácticas de laboratorio evaluables. Deben seleccionarse **al menos 2**, desarrollarlos y subirlos al CV. Todos los archivos entregados tienen que contener el nombre y los apellidos de los miembros del grupo (uno por línea).

---

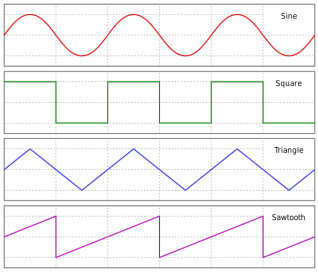
En los siguientes ejercicios utilizaremos las librerías de Python estudiadas en clase, así como la clase *KBHit* (basada en *pygame*) para captura no bloqueante de teclado (explicada en los notebooks de clase).

1. (**Evaluable**) Implementar una clase `Osc` que implemente un oscilador basado en chunks. El método constructor recibirá los parámetros de frecuencia, volumen y fase, que se guardarán como atributos de la clase. SE guardará también como atributo el *frame* actual (para evitar pops entre chunks según se ha visto en clase). El método *next* devolverá el siguiente chunk del generador cada vez que se invoque. Implementar métodos *get/set* para poder obtener y modificar dinámicamente el volumen y la frecuencia del oscilador.

   Probar este oscilador con método *testOsc* que cree un oscilador y reproduzca la señal, permita modificar la frecuencia (con las teclas `'F'/'f'`) y el volumen (con `'V'/'v'`).

   Se entregar un notebook **autocontenido** con la clase, el método *testOsc* y código para probar el funcionamiento.

2. Extenderlo el oscilador del ejercicio anterior para poder elegir formas de onda cuadrada, diente de sierra y triangular, utilizando la librería `SciPy`. Permitir cambiar dinámicamente la forma de onda.

   (Difícil) El cambio dinámico de parámetros en este oscilador puede producir pops o ruidos en la señal. Investigar formas de evitar/suavizar estas transiciones.

3. Implementar una clase `Moudulator` que extienda la clase `Osc` del ejercicio 1 y genere un modulador de señal con valores mínimo/máximo definidos en la cosntructora, además de la frecuencia.

4. **(Evaluable)** Investigar la representación de muestras estéreo en NumPy y hacer un efecto *modulador de balance o paneo* (con -1 se envía toda la señal a la pista izquierda; con 1 a la derecha). Puede utilzarse la clase `Modulator` del ejercicio anterior o implementar ad-hoc dicho modulador el balance de la pistas.

5. **(Evaluable)** Implementar un programa que permita grabar y reproducir a la vez utilizando la clase *Stream*, con callbacks (en el mismo método callback puede gestionarse la reproducción y la grabación). Para probarlo, tomar un wav de entrada para la reproducción y a la vez grabar otro, que se guardará en archivo.

6. Implementar un reproductor de *wav* que *normalice* el volumen de la salida, procesando por chunks. Para cada chunk calculará el valor máximo de las muestras en valor absoluto. Con ese coeficiente es fácil calcular el máximo valor por el que puede incrementar el volumen sin saturar la señal (todas las muestras deben quedar en el rango $[-1, 1]$).

   Como mejora, el coeficiente multiplicador puede modificarse *suavemente* entre chunks para no generar cambios abruptos de dinámica en el sonido de salida.

7. **(Evaluable)** Implementar una clase `Delay` para hacer una (línea de retardo): recibe una señal de entrada y devuelve esa misma señal, pero retardada en el tiempo una cantidad prefijada de tiempo. La constructora recibirá como argumento el tiempo de retardo en segundos. Tanto la entrada como la salida serán chunks de audio e internamente deberá gestionarse un buffer para producir el retardo.

   Para probar el funcionamiento vamos a implementar un *idiotizador*[1]. Para ello utilizaremos un Stream de entrada/salida y reenviaremos la señal de entrada a la salida, con un retardo temporal utilizando la línea de retardo del ejercicio anterior.

[1] Véase `https://www.youtube.com/watch?v=zhUDQGWM8kM`