Práctica 4: Nuestro primer patch sonoro

¿Qué elementos nuevos introduciremos en esta práctica?

https://gifer.com/embed/AFH7

Figura 1. Gatos con un metrónomo.

metro

Este objeto va a ser nuestro metrónomo en Pure Data y nos va a permitir **enviar** un **bang cada tantos milisegundos**. Por ejemplo, un metro con un argumento de 10.000 enviara un bang cada **10.000 milisegundos**, que es lo mismo que cada **10 segundos**. Este objeto nos va a permitir por ejemplo crear ritmos o secuenciar acciones.



Para que el **metro** funcione y comience a enviar bangs, tendremos que configurar sus parámetros y **activarlo** con un número distinto de 0. Para activarlo podemos enviar al inlet izquierdo un **bang**, un **toggle encendido**, o un mensaje con un valor numérico. Si queremos que deje de enviar bangs, tendremos que **pararlo** con un mensaje de **stop** o un valor igual a 0. En este caso, el toggle estará apagado y el mensaje contendrá el numero 0.

El **primer argumento** del metro (float) indica el **tiempo entre bangs** que por defecto está en milisegundos. El **segundo argumento** (float) nos permite **modificar el tempo** de la velocidad que ha configurado el primer argumento. El **tercer argumento** (symbol) nos permite **configurar**

Copyright 2026

las unidades, por ejemplo, en segundos con la palabra "second". También podremos modificar el tempo a través de un mensaje con los siguientes datos: la palabra "tempo", el valor del tempo y la unidad del tiempo.

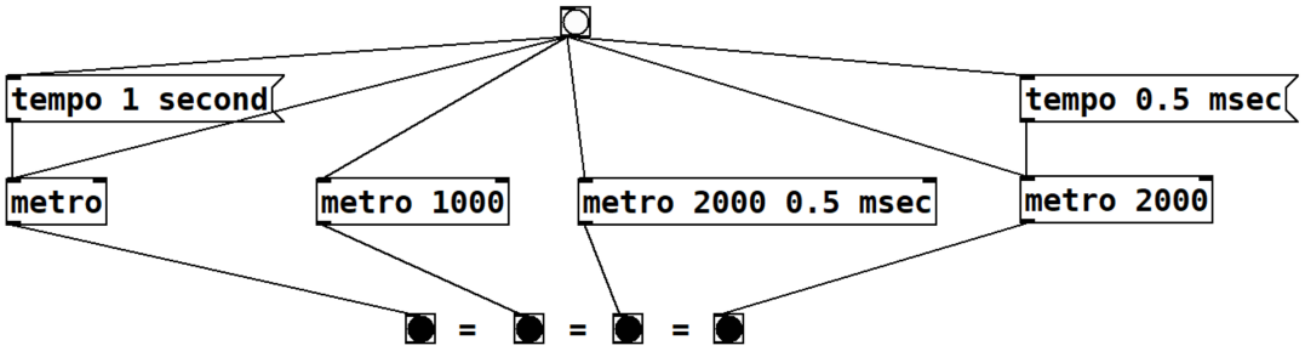


Figura 2. patch *metro-demo.pd.* Cuatro formas diferentes de configurar el objeto metro para que emita un bang por segundo.

A través del **inlet de la derecha** también podremos configurar el tiempo entre bangs una vez el metro esté corriendo. Para ello, utilizaremos una cajita de numero variable o un mensaje. Si en la figura 3 enviamos el mensaje "stop" al metro, la emisión de bangs se detendrá. Si enviamos 294 a traves del inlet derecho, en lugar de emir bangs cada 1000 milisegundos el metro, emitirá bangs ca

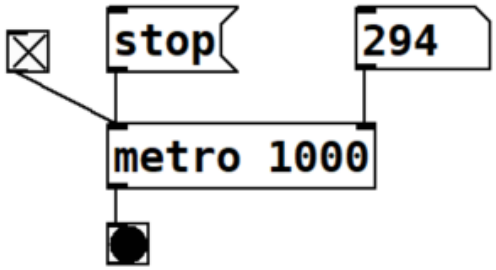


Figura 3. patch *metro-demo.pd. O*bjeto metro que emite

bangs cada 294 milisegundos.

line~

https://gifer.com/embed/cih

Figura 4. Tortuga deslizándose por una rampa.

El objeto "**line~**" genera **rampas lineales** entre dos valores en un tiempo determinado. Su outlet está en formato de **señal**. (Tambien podremos encontrar el objeto "line" que hace lo mismo, pero emitiendo una sucesión de números en lugar de una señal.)



Los parámetros de la rampa se determinan en los argumentos del objeto o por los mensajes recibidos en los inlets de line. Por ejemplo, generar una secuencia de valores entre 2 y 5 en 1000 milisegundos. Enviando un mensaje al **inlet izquierdo** podremos configurar los **valores de partida**, **destino** y el **tiempo de nuestra rampa**. Abre el patch *line-snapshot-demo.pd* para probar este objeto.

Copyright 2026

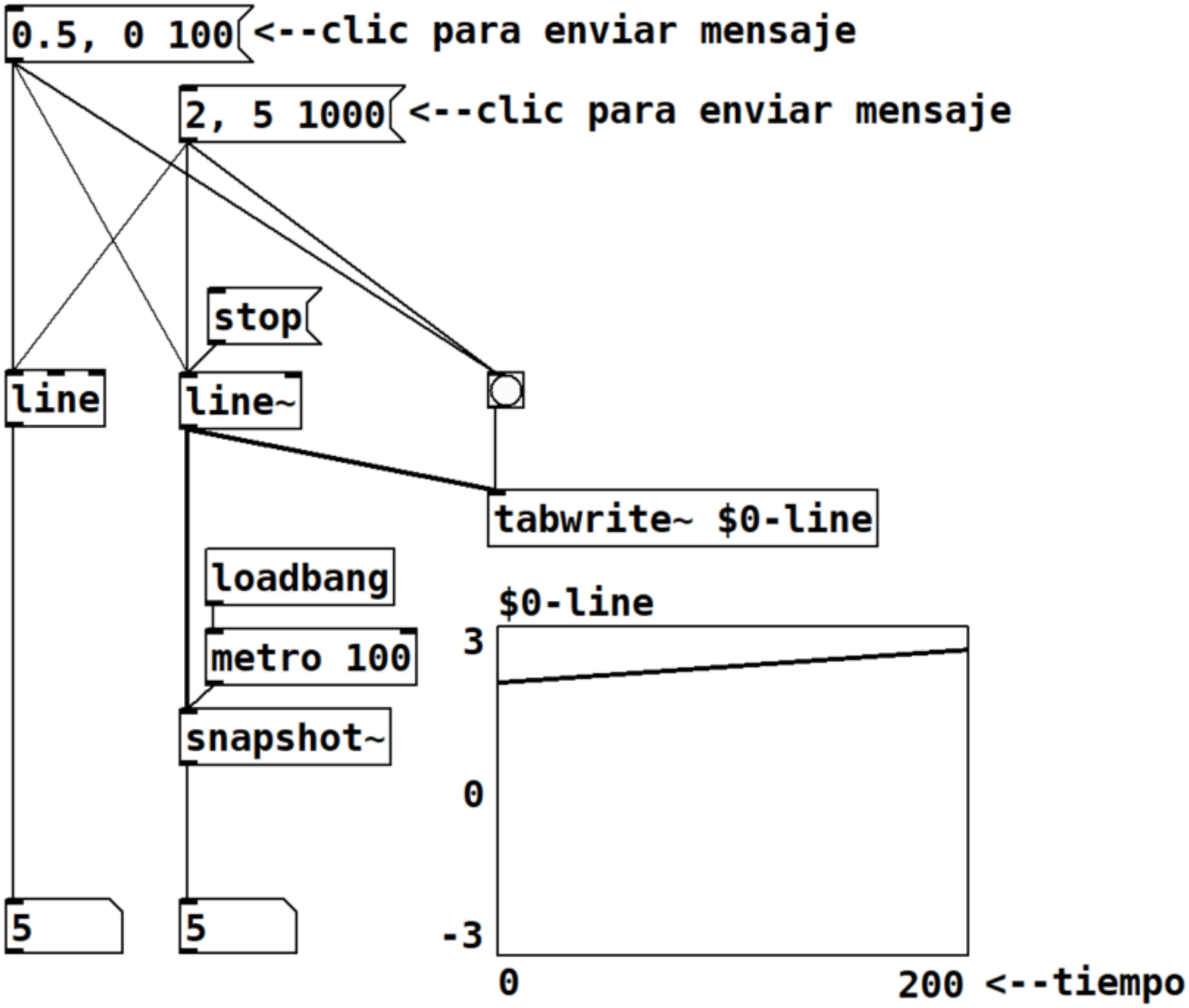


Figura 5. patch *line-snapshot-demo.pd.*

Veamos cómo tenemos que construir los **mensajes** para controlar nuestra rampa:

El **tiempo** también podremos configurarlo con el **inlet derecho.** Un mensaje con único valor indicará el valor del destino. El line de la imagen ira al valor 5 en 64 milisegundos.

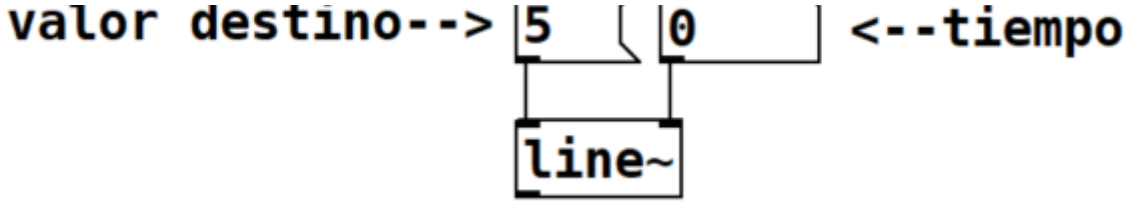


Figura 6. patch *line-snapshot-demo.pd*. objeto line~ que recibe un mensaje con la configuración: destino=5. Y el tiempo de la rampa se configura a traves del inlet derecho.

Cuando enviemos un mensaje con dos valores separados por un espacio, el primer valor del mensaje indicara el valor del destino y el segundo valor el tiempo que se tomara en llegar a ese val al valor 5 en 100 milisegundos.

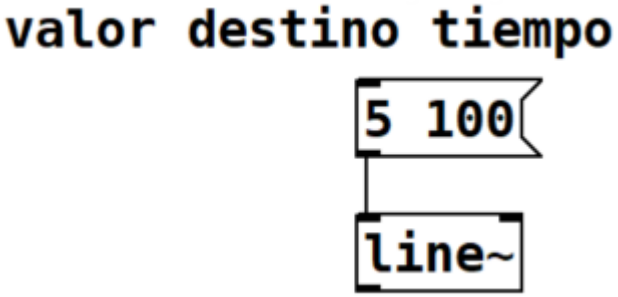


Figura 7. patch *line-snapshot-demo.pd*. objeto line~ que recibe un mensaje con la configuración: destino=5, tiempo=100.

Enviamos un mensaje con tres valores, los dos primeros separados por "coma" "espacio" y el segundo y el tercero por un "espacio"; el primer valor indicara el valor de partida, el segundo el valor de destino y el tercero el tiempo requerido para ir desde el valor de partida al valor de lor 2 al valor 5 en 100 milisegundos.

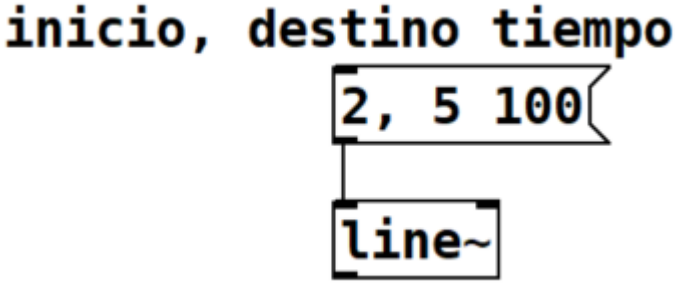


Figura 8. patch *line-snapshot-demo.pd*. objeto line~ que recibe un mensaje con la configuración: Inicio=2, destino=5, tiempo=100.

> Si no especificamos el tiempo, saltará directamente al valor de destino. Si queremos **detener la progresión** de la rampa enviaremos un mensaje con la palabra "**stop**". El

ultimo valor generado antes de parar la progresión, o el valor de destino de la rampa en caso de que la progresión no se detenga y llegue al final, quedará almacenado.
Cuando **no especifiquemos un valor de partida**, tomará como valor de partida ese valor almacenado en la **interacción anterior**. Cuando no haya habido una interacción anterior el **valor de partida por defecto sera 0.**

El **line** de la imagen siguiente irá desde el valor 2 al valor 5 en 100 milisegundos. Si recibe el mensaje stop, detendrá la progresión.

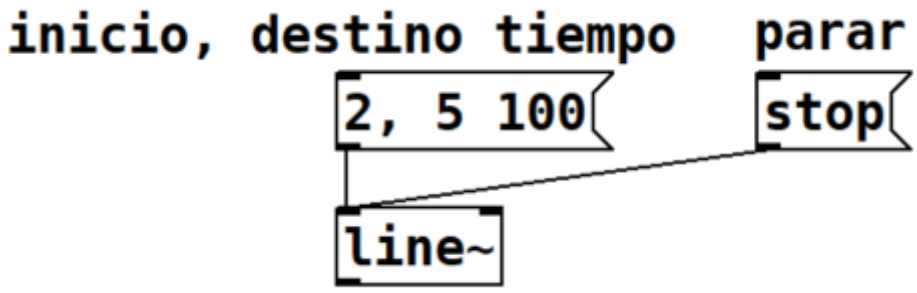


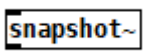
Figura 9. patch *line-snapshot-demo.pd*. objeto line~ que recibe un mensaje con la palabra "stop"

snapshot~

https://gifer.com/embed/73QW

Figura 10. El "snapshot~" haciendo una foto de la señal cada vez que le llega un bang.

El **snapshot~** permite convertir una señal en valores numéricos. Esto nos será muy útil cuando queramos enviar valores procedentes de una señal de audio a un objeto que no sea de audio.



> Recordad que los objetos que no son de audio no tienen la "~" y no aceptan señales deaudio como input.

Este objeto nos será útil para visualizar qué sucede con nuestras señales de audio. Cada vez que recib ue llega al inlet en ese momento, sería el equiv or una onda en un momento determinado.

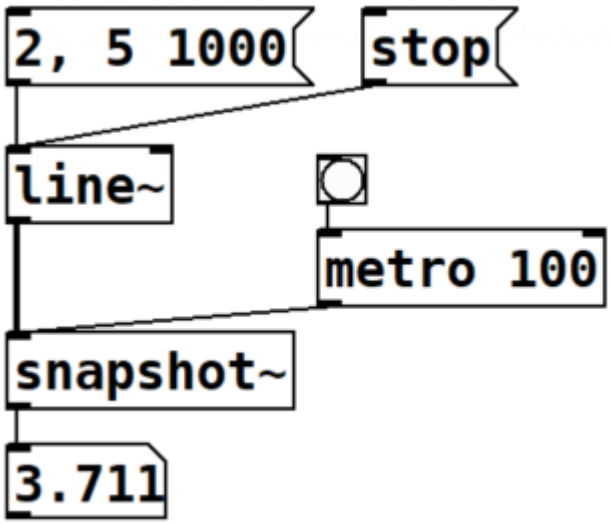


Figura 11. ejemplo del objeto "snapshot~"

> Ejercicio 6: Abre los patches de ejemplo: *line-snapshot-demo.pd* y *metro-demo.pd* y prueba el funcionamiento de los objetos que acabamos de ver. Encontrarás los patches en el material de este apartado ¿Tienes alguna duda?

env~

Similar al "snapshot~", el envelope follower: "**env~**" recibe una señal y saca por su salida la amplitud de esta señal, pero en dB, con una equivalencia de 1 igual a 100 dB.

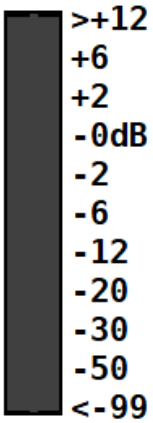


Este objeto recibe una señal y devuelve valores numéricos. Nos será muy útil para **conocer el volumen** de nuestra señal en **dB** y también para visualizar ese volumen en el **VU meter** que vamos a presentar a continuación.

Recordad que en el apartado "Las ondas y el sonido", habíamos visto que el volumen es la intensidad con la que percibimos un sonido. Es una magnitud que relaciona la amplitud de una onda sonora y la percepción humana de esta onda.

VU meter

Podremos crear este objeto desde el **menu>poner>VU Meter** o escribiendo "vu" en un objeto.



Este objeto es una **interfaz gráfica** que representa el **nivel de señal** como el que podemos encontrar en el **master de una mesa de mezclas**, o de un canal en programas como Ableton Live. Nos va a indicar el **volumen de la señal**.

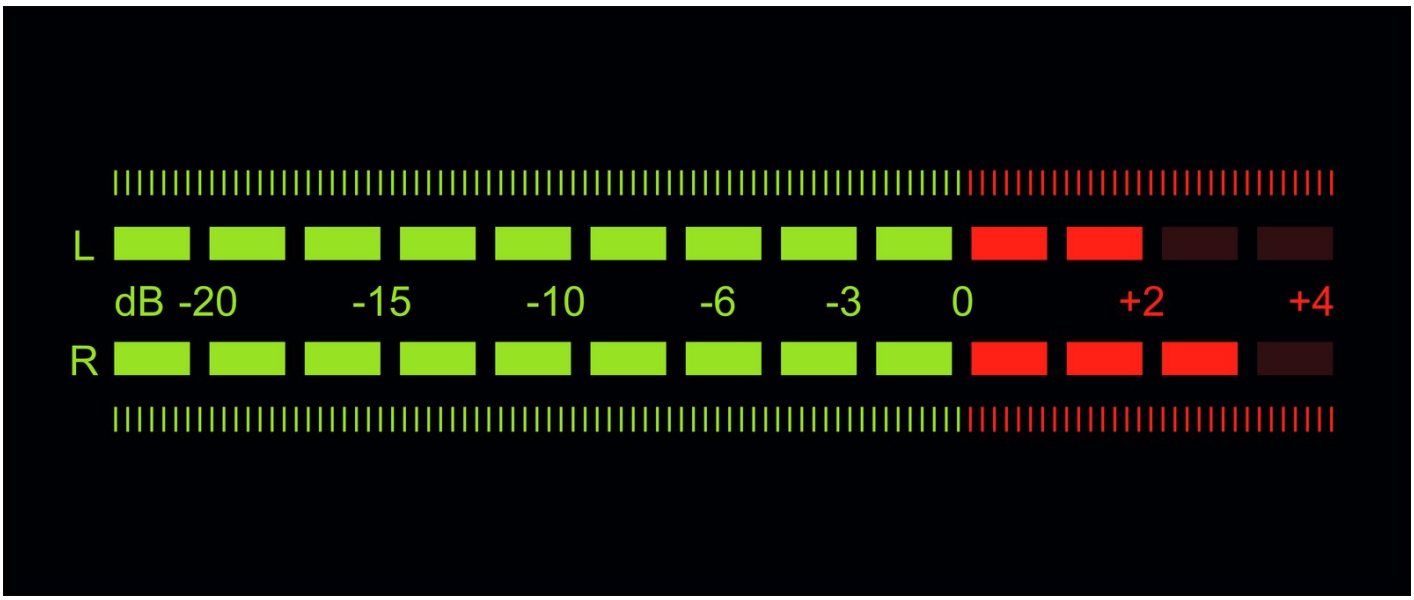


Figura 12. Representación de un VU meter.

El rango un VU meter va de valores negativos a valores positivos. Los valores mínimo y máximo varían en cada aparato o programa, sin embargo, el **0** sera el **valor de referencia** en todos ellos. Y un código de colores nos ayudara a leer esa escala. El **verde** nos indica un **nivel de señal adecuado**, el **amarillo** nos indica un **nivel de señal próximo al límite** entre un nivel adecuado y un nivel excesivo, y el **rojo** nos indica un **nivel de señal excesivo** que debemos evitar y a partir del cual consideraremos que nuestra señal esta "**picando**" y se va a distorsionar.

En Pure data el **VU-meter** tiene un rango visible de -100 a 12, y el 0 va a ser nuestro límite de pico entre señal adecuada y señal excesiva.

Ya que el **envelope** nos proporcionará valores con una igualdad de 100 dB a 1 unidad de amplitud, tendremos que restar 100 a la salida del envelope para enviar un valor adecuado al VU meter.

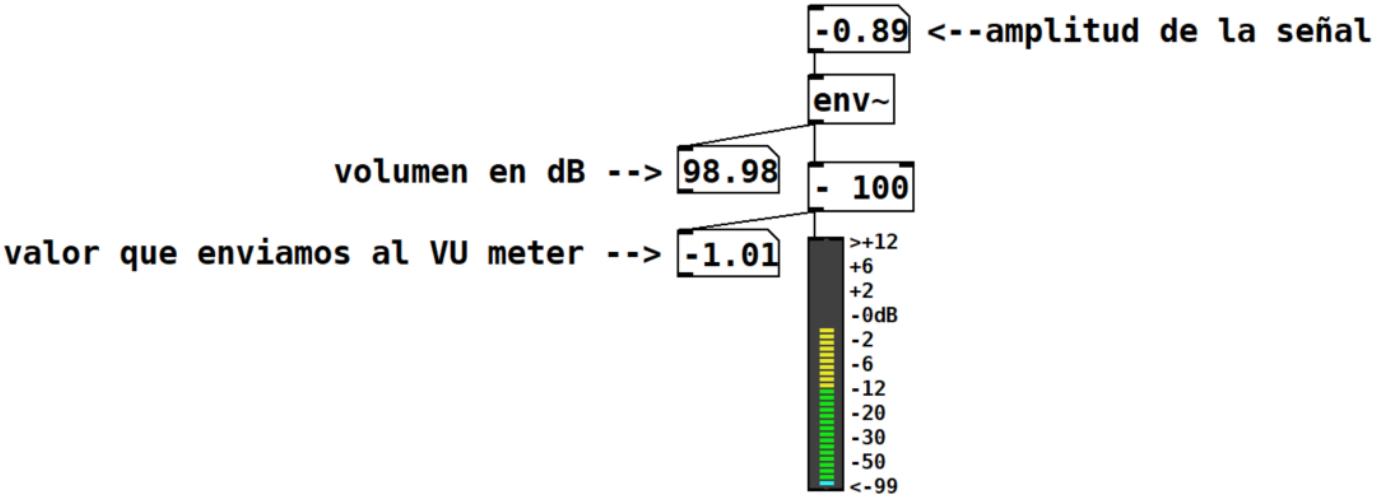


Figura 13. patch *Envelope-Vumeter-demo.pd*

Slider

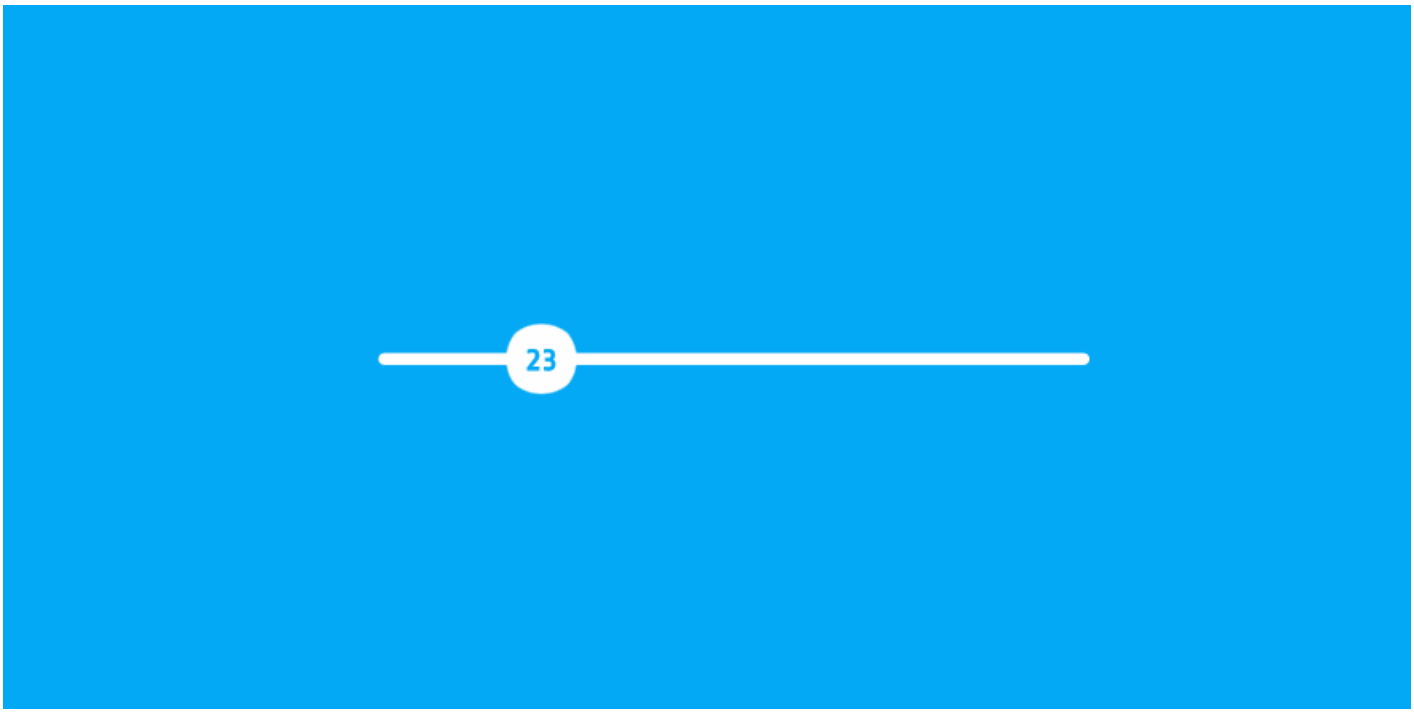
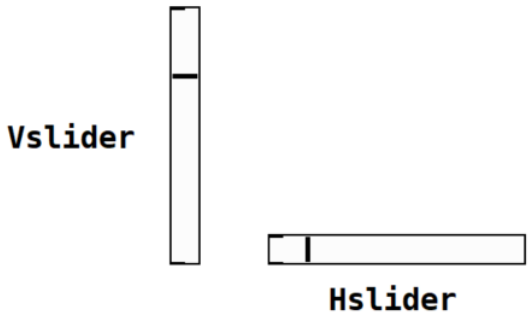


Figura 14. Slider en funcionamiento. (no es Pure Data)

El **slider** es una **interfaz gráfica** que nos permite **enviar valores repartidos en un rango determinado**, colocando y moviendo un indicador en cualquier posición de la representación de ese rango.



Para mover el indicador, clicaremos sobre el elemento y manteniendo el click pulsado moveremos el ratón. En caso de necesitar mas precisión a la hora de ajustar la posición del indicador, realizaremos la acción anteriormente mencionada mientras pulsamos la tecla **SHIFT** o **MAYUS**. Tambien podremos modificar la posición del indicador enviado un mensaje con el valor que queramos al inlet del slide.

Esta representación podrá ser vertical u horizontal y configuraremos los valores mínimo y máximo de su rango, haciendo **click derecho** sobre el **slider>Propiedades>Rango de Salida.**

En propiedades también podremos configurar los colores, etiqueta o tamaño del slider.

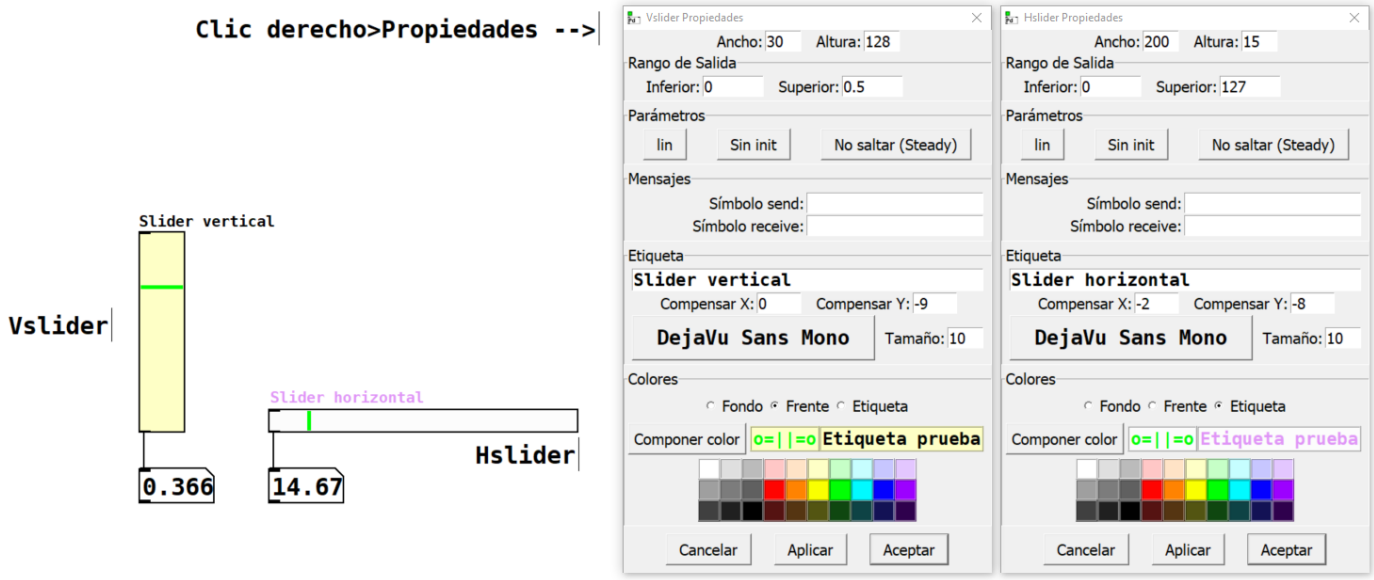


Figura 15. patch *Slider-demo.pd.* Slider vertical a la izquierda y Slider horizontal a la derecha con sus ventanas de propiedades.

El **slider por defecto** tiene un rango de **0 a 127**, rango que tendremos que ajustar a nuestras necesidades. Este objeto nos va a ser extremadamente útil para regular el **volumen**. Ajustaremos su rango de **0 a 0.5** o, como máximo, de 0 a 1.

Oscilador y *~



Figura 16. Personita generando una onda de 293.6 Hz.

El objeto **osc~** es un generador de ondas sinusoidales, también conocidas como ondas puras. Debe recibir en su **inlet izquierdo** la **frecuencia** de la onda que queramos generar en valores en **Hz.**



Este objeto va a generar una onda con una amplitud de 1, lo que quiere decir que generara valores correspondientes a esa onda sinusoidal de -1 a 1.

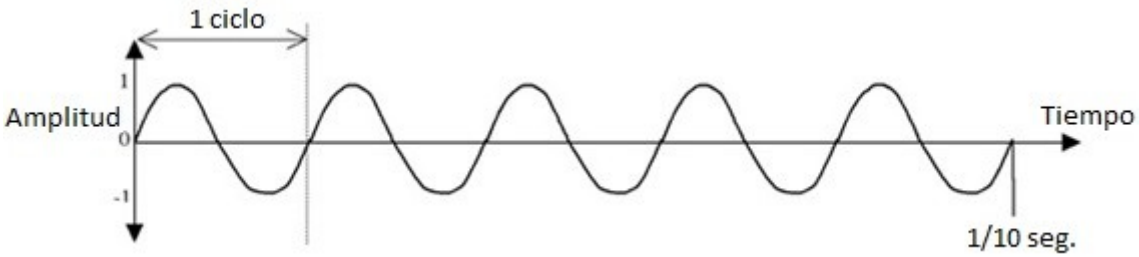


Figura 17. La onda Sinusoidal que genera el "osc~".

En **Pure Data** se parte de la equivalencia de **1 unidad de amplitud igual a 100 dB.**

> Recordad que los **dB** median la **intensidad sonora percibida** y que esta tiene una **relación logarítmica** a la amplitud, no lineal. Por lo tanto, **aunque 1 unidad de amplitud equivalga a 100dB, 0,5 unidades de amplitud NO equivalen a 50 dB**, **0,5** unidades de amplitud **equivalen a unos 90 dB**.

El objeto "*~" nos va a permitir controlar/modular la **amplitud de una señal** multiplicándola por un valor numérico o por otra señal. En el ejemplo del patch *osc-demo.pd* la multiplicamos por un valor numérico del tipo **float**. Cuando multipliquemos por valores situados entre **0 y 1** estaremos **disminuyendo la amplitud de la señal**. Valores **mayores de 1** estarán **aumentando la amplitud** de la señal.

Para el **control de volumen** siempre multiplicaremos por **valores ente 0 y 1**. Más adelante también utilizaremos este objeto para mezclar señales. Sus inlets esperan una señal o un valor numérico, y su outlet es una señal.

Abrid el patch *osc-demo.pd* que encontraré deis en el material que os damos para la Practica 2. Vamos a ver esto con un ejemplo en el que utilizaremos algunos de los objetos que hemos visto anteriormente.

Copyright 2026

- el **snapshot~**: para ver el valor de la señal en un momento determinado
- el **env~**: para transformar la amplitud de la señal en dB.
- el **VU**: visualizar el volumen en dB.
- el **metro** para hacer la foto de la señal con el "snapshot~"
- la **slider** para controlar el volumen.

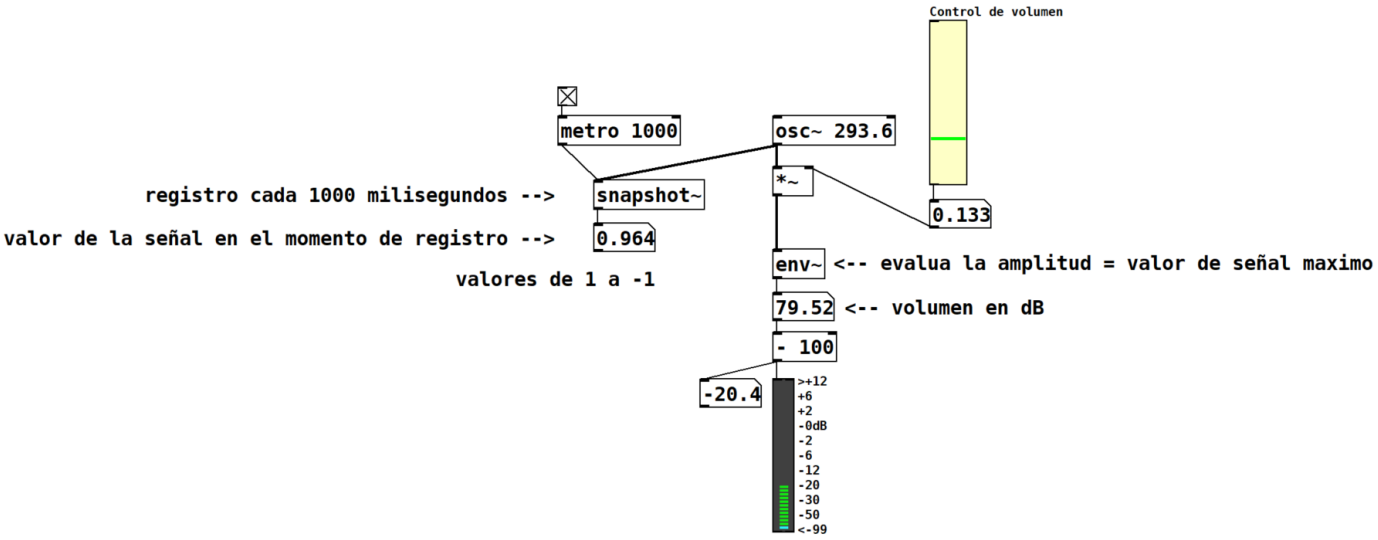
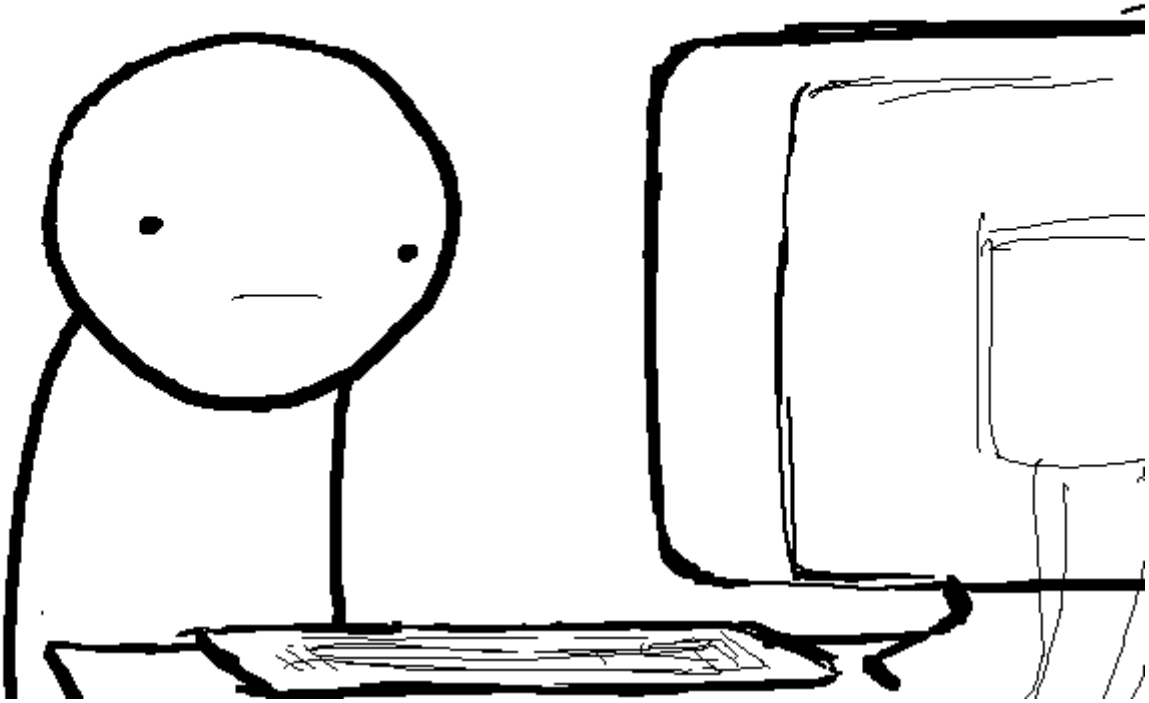


Figura 18. patch *osc-demo.pd*.

El "osc~" de la figura 17, genera una onda sinusoidal de 293,6 Hz de frecuencia. La amplitud de esa onda se reduce multiplicándola por 0,1333 en el objeto "*~". El "env~" registra la amplitud de la señal que le llega y la convierte a dB, con una equivalencia de 100 dB a 1 unidad de amplitud. Restamos 100 al volumen que proporciona el "env~" para poder visualizar el volumen en el VU meter.

¿Alguna duda de cómo funciona el patch *osc-demo.pd?*



MIDI to frequency: mtof

Esto ya lo hemos visto en "Configuración de Audio en Pure Data y notación MIDI". Como bien sabemos, el **oscilador** habla el idioma de los **Hz** y en este idioma interpretará los valores que reciba. El objeto "**mtof**" convierte **valores numéricos de MIDI a valores numéricos de Hz**. Nos será muy útil para trabajar con los valores de la escala musical diatónica (Do, Re, Mi ...). Este objeto **recibe valores numéricos y devuelve valores numéricos**. Inlet (float), outlet (float)

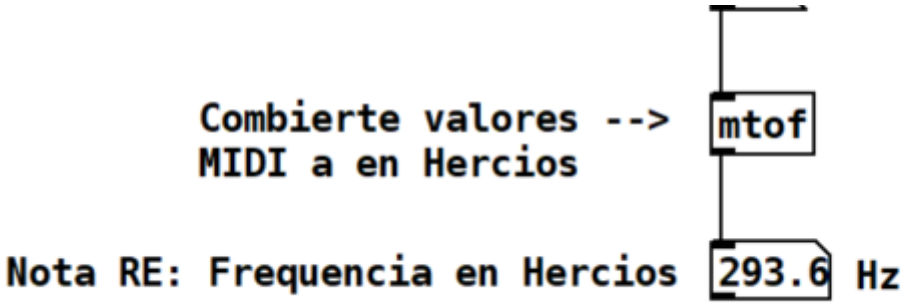


Figura 19.

patch *mtof-ftom-demo.pd.* Conversion de MIDI a Hz con el objeto mtof.

https://gifer.com/embed/8pIY

Figura 20. Transformer civil pasando de Hz a MIDI

Frequency to MIDI: ftom

El objeto "**ftom**" objeto realiza el **proceso inverso a** "**mtof**", convierte **valores numéricos de Hz a valores numéricos de MIDI**. Inlet (float), outlet (float).

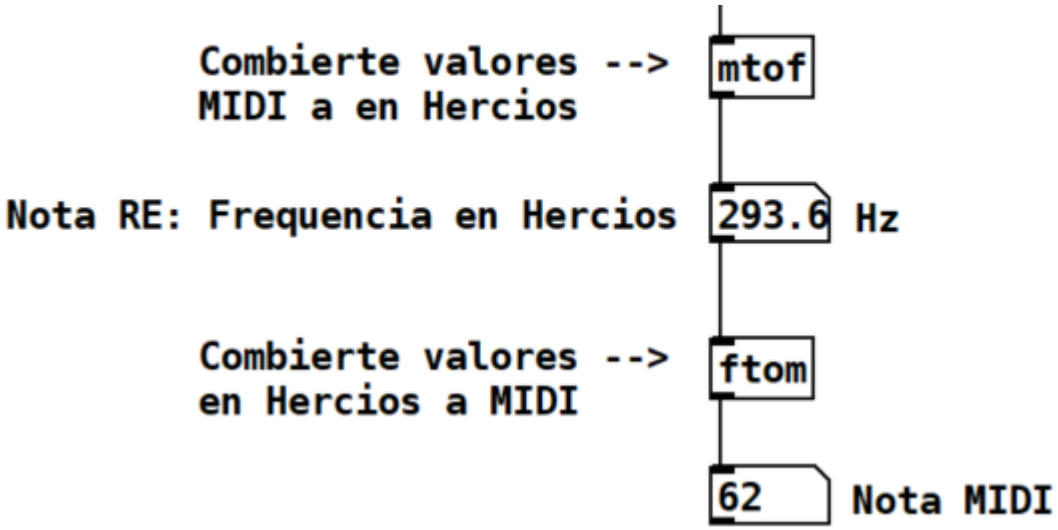
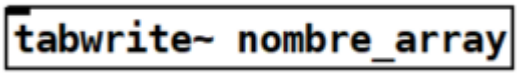


Figura 21. patch

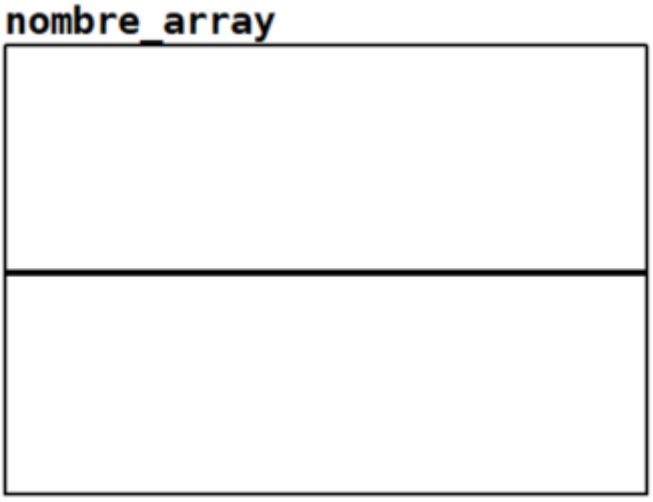
mtof-ftom-demo.pd. Conversion de MIDI a Hz con el objeto mtof, y de Hz a MIDI con el objeto ftom.

Tabwrite~ y Array (GUI)

El objeto "**tabwrite~**" nos permite **almacenar los valores de una señal de manera secuencial** en un vector.



Podremos visualizar este vector con la interfaz gráfica "**Array**", que es una especie de lienzo. Para colocar un array en nuestro patch: **menu horizontal>Poner>Array** o utilizando en comando **"Mayus/Shift+Ctrl+A"**



¿Pero c valores almacenados de manera secuen

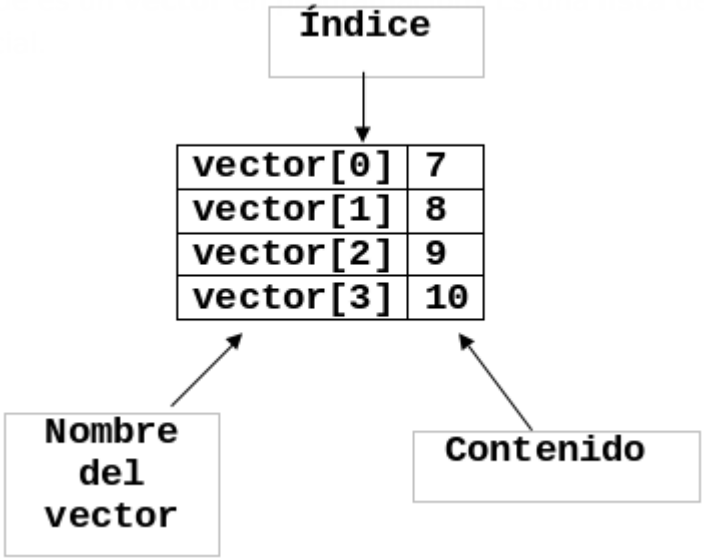


Figura 22. Diagrama de un vector.

El **argumento del "tabwrite~" indicará el nombre del vector** y para que empiece a registrar los valores que le llegan tendremos que enviar un **bang**.

Cuando el vector se llene dejará de registrar valores y si queremos que vuelva a registrar tendremos que enviar otro bang.
El vector **almacenará 100 valores por defecto.** Si queremos modificar su tamaño, haremos click derecho sobre **Lienzo>Propiedades>Propiedades del array>Tamaño**. En propiedades del array también podremos ver la lista de valores almacenados clicando en "**Ver Lista ...**".

Para modificar parámetros del lienzo utilizaremos la ventana "**Propiedades de Canvas**" que se ha abierto justo con la ventana de "**Propiedades del array**" La **interfaz gráfica array** nos va a

permitir **visualizar el valor almacenado en cada posición de un vector.** Tendremos que indicar en la configuración del array el nombre del vector que queramos visualizar. La posición en el vector, el **índice**, se representa en el **eje x** y el **valor almacenado**, el contenido, en el **eje y**. Podéis probar estos elementos en el patch *tabwrite-array-demo.pd* que encontrareis en el material de la Practica 2.

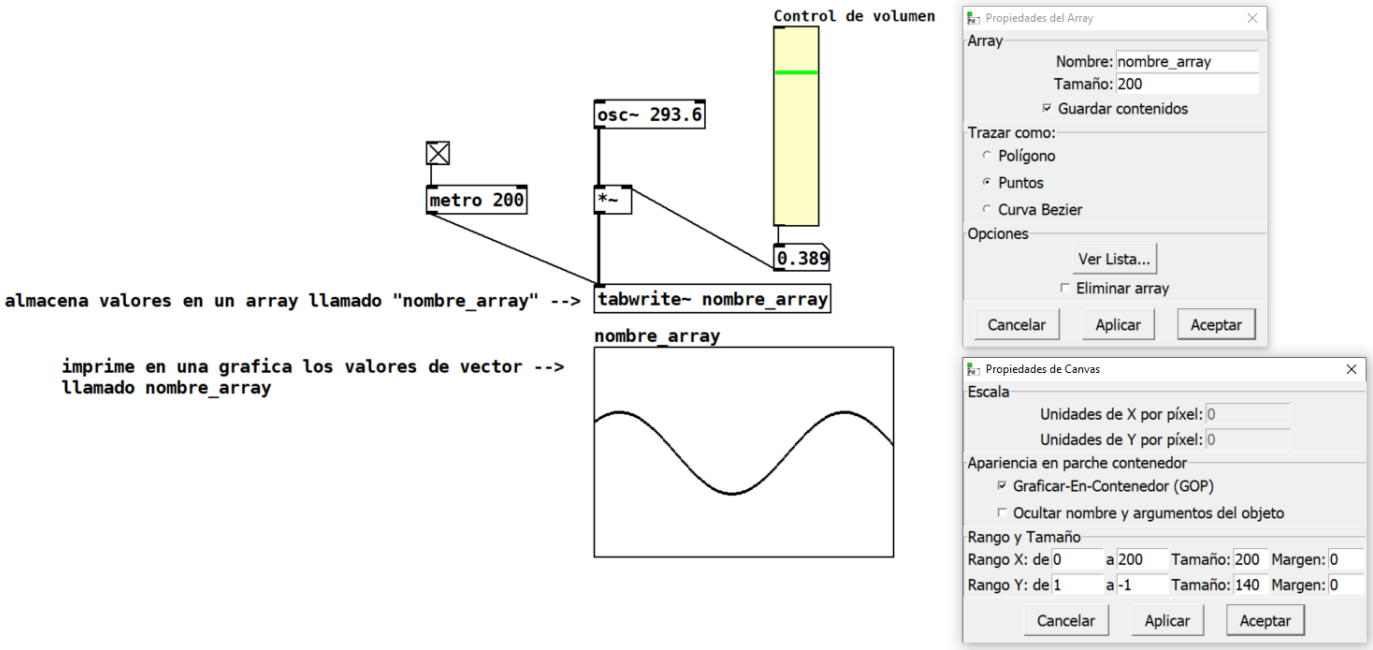


Figura 23. patch *tabwrite-array-demo.pd*.

Todo esto puede parecer un poco complicado, así que, para entender mejor cómo funciona, utilizaremos la representación de este vector a lo largo del tiempo. Vamos a utilizar la metáfora de una **fotografía**:



Figura 24. Una imagen de la animación en la figura 25.

Cada vez que el "**tabwrite~**" recibe un bang, toma una foto de los valores de la señal en un corto periodo de tiempo que medimos con un **cronómetro**. Cuando el cronómetro comienza a contar, el **primer valor registrado** ocupará la **primera posición del vector** y cuando el cronometro termina el **último valor registrado**, tomará la **última posición del vector**.

Para ver cómo una señal evoluciona en el tiempo, tendremos que tomar muchas fotos. Es lo mismo que sucede con un video o una película de dibujos animados, en la que por lo general vemos 24 imagen por segundo. Para que tome todas estas fotos utilizaremos el objeto **metro** que hemos visto anteriormente



Figura 25. Animación que veremos en el array del patch *tabwrite-array-demo.pd* si activamos el metro.

Tamaño del array:

Cuando queráis visualizar una señal utilizando el array, cambiar el tamaño del array os va a ayudar a ajustar la escala de lo que queráis visualizar, de forma que esta visualización os proporcione información comprensible. Podéis cambiar el tamaño del array en propiedades. Os dejo dos imágenes para que veáis la diferencia representando la misma señal con un array de tamaño 100 y otro de tamaño 2000:

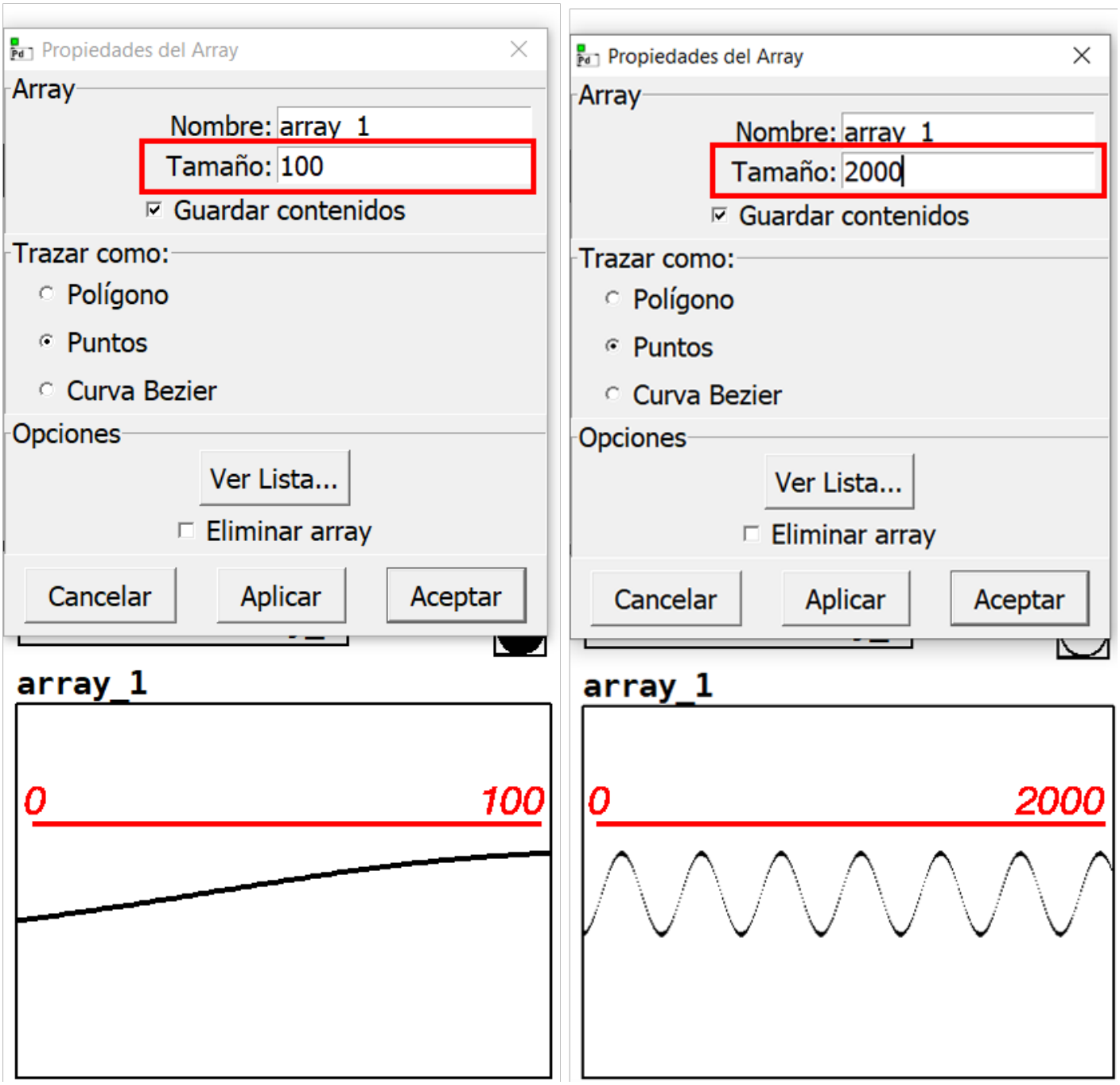


Figura 26. Misma señal representada con un array de tamaño de 100 (izquierda) y con un array de tamaño de 2000 (derecha).

send~ y receive~

Estos dos objetos nos van a permitir **enviar y recibir señales sin necesidad de unir cajitas con cables.** Son como un **emisor** y un **receptor inalámbricos** y nos van a ser muy útiles para mantener la claridad y comprensión de nuestros programas cuando tengamos muchos elementos y conexiones.

https://gifer.com/embed/HvQG

Figura 27. Criatura enviando datos.

Cuando queramos enviar datos que no tienen el formato de señal, utilizaremos los mismos objetos, pero sin la virgulilla "~". Esto nos va a permitir enviar mensajes (int, float y symbols) sin necesidad de conectar las cajitas con líneas.

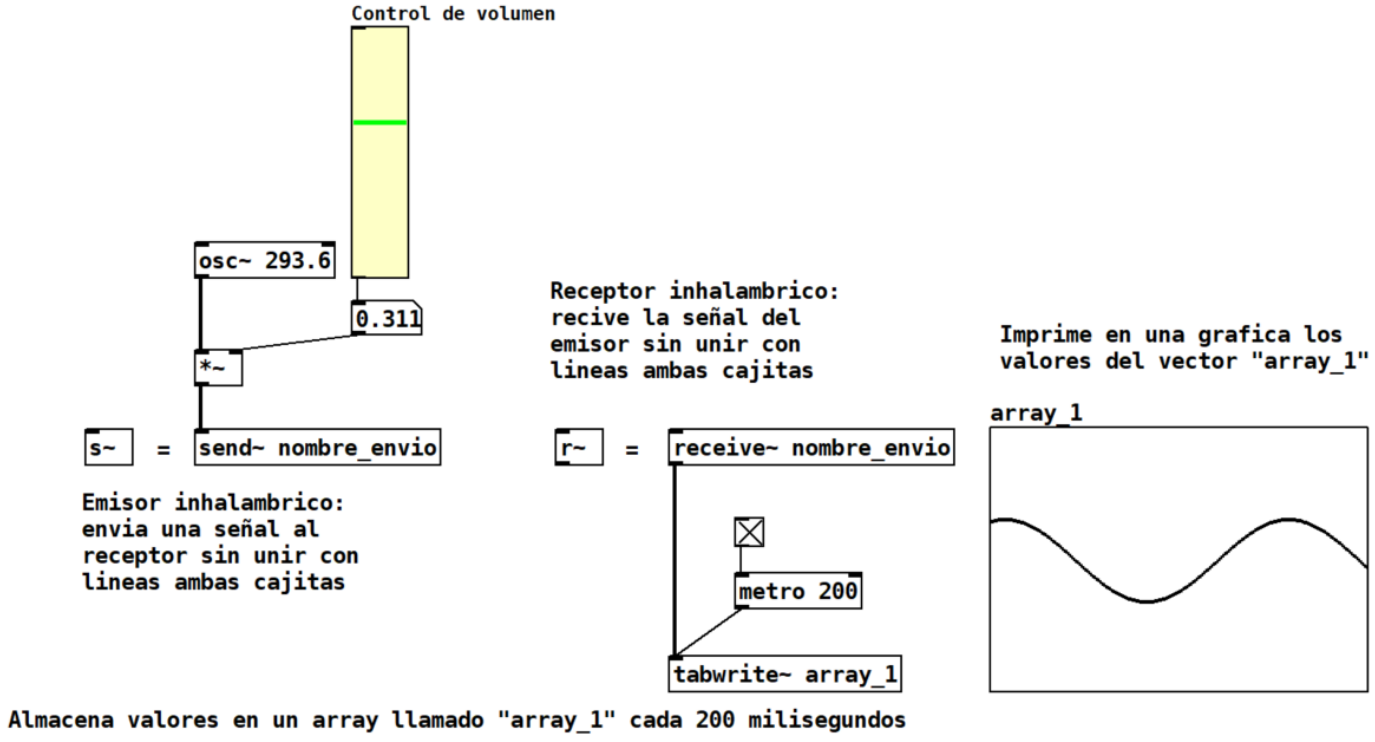


Figura 28. patch *send-receive-demo.pd*

dac~

¡¡Finalmente vamos a **conectarnos a los altavoces!!**

https://gifer.com/embed/763H

Figura 29. Altavoces recibiendo sonido desde Pure Data cuando no controlamos el volumen.

dac~ = "digital to analog converter" Este objeto representa la puerta de **conversión** entre los **números** que componen señales de audio y el **movimiento** de la membrana del altavoz que va a generar el sonido. Nuestra **tarjeta de sonido** va a convertir estos datos en **variaciones de corriente eléctrica** que van a hacer que nuestro **altavoz se mueva**.



Vemos que el objeto tiene dos inlets, cada uno corresponde a un canal de estéreo: **derecha** e **izquierda**.

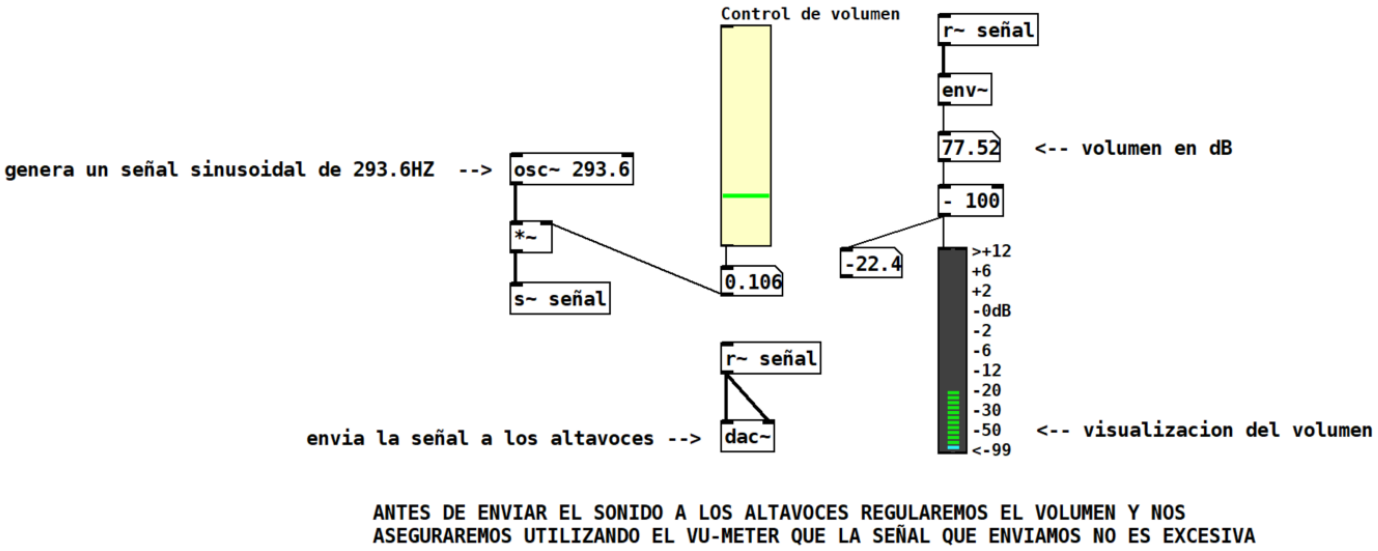


Figura 30. patch *dac-demo.pd*.

Como veis en el ejemplo del patch *dac-demo.pd*, puedo tener varios receptores del mismo envío y ambos reciben la misma señal.

¡Comencemos con la practica!

https://gifer.com/embed/2GU

Figura 31. Gato haciendo la Practica 2 de Pure Data.

Ahora vamos a crear un **patch** con los elementos que acabamos de ver. Podéis **copiar** y **pegar** elementos de un patch a otro, y recordad que tenéis acceso a todos los patches con todos los ejemplos que hemos visto en esta lección.

Para ello os voy a dar el algoritmo escrito de cómo ha de ser nuestro programa y vosotros lo construiréis en Pure Data. Recordad que hay diferentes formas de llegar al mismo resultado así que sentíos con libertad de combinar los objetos que acabamos de introducir de diferentes maneras para probar y conseguir resolver el problema de la manera que más cómoda os resulte.

> Ejercicio 7: Crear un patch que nos permita **reproducir las notas de la escala de una octava con nuestros altavoces.** En mi caso he elegido la octava tercera. A parte de reproducir las notas, nos gustaría **elegir la nota de la octava que se reproduce a través del ratón**.

Podéis consultar el numero MIDI de cada nota en la tabla que vimos en "Configuración de Audio en Pure Data y notación MIDI"

> Solo enviaremos sonido a los altavoces una vez estemos seguros de que el **nivel de volumen** es **adecuado**.

Algoritmo:

1. Voy a crear un nuevo patch.
2. Necesito tener el **DSP encendido** para que el programa trabaje con señales.
3. Necesito **hacer llegar a un oscilador los valores en Hz** de las frecuencias correspondientes a la octava.
4. Necesito **controlar el volumen**. Voy a **regular la amplitud de la señal** que sale del oscilador y voy a utilizar un rango de **0 a 0.5** para modularla.
5. Me aseguro de que el **volumen es adecuado**. Voy a **visualizar el nivel de volumen con el VU-meter** antes de enviar la señal al dac~.
6. Una vez el **volumen en Pure Data está bajo control** y con el **volumen de mis altavoces bajado** enviare la **señal al dac~** y poco a poco **subo el volumen de los altavoces**. El volumen de mis altavoces esta fuera de Pure Data, por ejemplo, en mi caso es un knob, una rueda que giro en mi tarjeta de sonido. Si no tenéis tarjeta de sonido externa sera el volumen de vuestro ordenador.
7. Una vez **escucho una de las notas** de la octava **probare a reproducir el resto** de las notas.
8. **Subo y bajo el volumen desde Pure data**. ¿Escuchas algo raro? clics y pums? Utiliza el objeto **line~** para suavizar los cambios de volumen.
9. Si quiero **visualizar la señal** utilizaré un **vector y la interfaz gráfica** que me permite visualizarlo.

Si tras intentarlo no habéis logrado finalizar la práctica, podéis mirar la página de pistas de la Practica 4

¿Qué tenemos que entregar?

El ejercicio 7. Sube a la carpeta del Moodle de la Practica 4 una captura de pantalla det patch y el patch que habéis creado. El patch deberá llamarse: **practica4_vuestronombre_vuestroapellido.pd** y la imagen **practica4-imagen_vuestronombre_vuestroapellido**.

Figuras:

Figura 1. Gatos con un metrónomo. https://gifer.com/en/AFH7

Figura 2. patch *metro-demo.pd.* Cuatro formas diferentes de configurar el objeto metro para que emita un bang por segundo.

Figura 3. patch *metro-demo.pd. O*bjeto metro que emite bangs cada 294 milisegundos.

Copyright 2026

Figura 4. Tortuga deslizándose por una rampa. https://gifer.com/en/cih

Figura 5. patch *line-snapshot-demo.pd.*

Figura 6. patch *line-snapshot-demo.pd*. objeto line~ que recibe un mensaje con la configuración: destino=5. Y el tiempo de la rampa se configura a traves del inlet derecho.

Figura 7. patch *line-snapshot-demo.pd*. objeto line~ que recibe un mensaje con la configuración: destino=5, tiempo=100.

Figura 8. patch *line-snapshot-demo.pd*. objeto line~ que recibe un mensaje con la configuración: Inicio=2, destino=5, tiempo=100.

Figura 9. patch *line-snapshot-demo.pd*. objeto line~ que recibe un mensaje con la palabra "stop".

Figura 10. El "snapshot~" haciendo una foto de la señal cada vez que le llega un bang. https://gifer.com/en/73QW

Figura 11. ejemplo del objeto "snapshot~"

Figura 12. Representación de un VU meter.https://www.freeimages.com/nl/photo/vu-meter-1242056

Figura 13. patch *Envelope-Vumeter-demo.pd*

Figura 14. Slider en funcionamiento. (no es Pure Data). https://www.pinterest.com.mx/pin/751186412851747700/

Figura 15. patch *Slider-demo.pd.* Slider vertical a la izquierda y Slider horizontal a la derecha con sus ventanas de propiedades.

Figura 16. Personita generando una onda de 293.6 Hz. https://gfycat.com/concretebetterbilby

Figura 17. La onda Sinusoidal que genera el "osc~". https://www.hsa.org.uk/electricidad/onda-y-frecuencia

Figura 18. patch *osc-demo.pd*.

Figura 19. patch *mtof-ftom-demo.pd.* Conversion de MIDI a Hz con el objeto mtof.

Figura 20. Transformer civil pasando de Hz a MIDI. https://gifer.com/en/8pIY

Figura 21. patch *mtof-ftom-demo.pd.* Conversion de MIDI a Hz con el objeto mtof, y de Hz a MIDI con el objeto ftom.

Figura 22. Diagrama de un vector. https://montealegreluis.com/logica-programacion/docs/arreglos.html

Figura 23. patch *tabwrite-array-demo.pd*.

Figura 24. Una imagen de la animación en la figura 25. https://tenor.com/view/human-wave-formation-soliders-gif-12353118

Figura 25. Animación que veremos en el array del patch *tabwrite-array-demo.pd* si activamos el metro. https://tenor.com/view/human-wave-formation-soliders-gif-12353118

Figura 26. Misma señal representada con un array de tamaño de 100 (izquierda) y con un array de tamaño de 2000 (derecha).

Figura 27. Criatura enviando datos. https://gifer.com/en/HvQG

Figura 28. patch *send-receive-demo.pd*

Figura 29. Altavoces recibiendo sonido desde Pure Data cuando no controlamos el volumen. https://gifer.com/en/763H

Figura 30. patch *dac-demo.pd*.

Figura 31. Gato haciendo la Practica 2 de Pure Data. https://gifer.com/en/2GU

Revision #63
Created 2022-09-21 10:34:12 CEST by Julia del Río
Updated 2023-12-29 14:30:28 CET by Julia del Río

Copyright 2026