# Composición algorítmica y síntesis de sonido en PureData

Dr. Alfonso Alba Cadena
Facultad de Ciencias, UASLP
fac@fc.uaslp.mx

# Contenido

- Sesión 1: Introducción a Puredata
- Sesión 2: Síntesis de sonido
- Sesión 3: Arreglos
- Sesión 4: Composición algorítmica I: Melodía
- Sesión 5: Composición algorítmica II: Ritmo

Sesión 1

# INTRODUCCIÓN A PUREDATA

# ¿Qué es PureData?

- PureData (Pd) es un entorno visual de programación orientado al procesamiento de audio en tiempo real.
- Desarrollado por Miller Puckette.
- Surge como una alternativa a lenguajes como Max/MSP, pero de libre distribución.
- Versión básica de Pd: http://crca.ucsd.edu/~msp/software.html

# Programación en Pd

- La programación en Pd se realiza de manera visual, conectando unos elementos con otros y estableciendo sus propiedades.

- Una vez abierto un documento de Pd (llamados *parches*), uno puede alternar entre el modo de edición y el modo de ejecución usando CTRL-E.

- En el modo de edición, se pueden colocar (menú **Put**), mover, e interconectar los elementos del parche libremente.

- En el modo de ejecución, se pueden utilizar los elementos de interfaz gráfica para manipular los parámetros del parche.

# Elementos de Pd

- **Objetos (CTRL+1):** representan instrucciones que Pd debe ejecutar. Los objetos pueden tener entradas (en la parte superior) y salidas (en la parte inferior).

- **Mensajes (CTRL+2):** contienen información que puede enviarse a uno o más objetos cuando se hace click en el mensaje.

- **Cajas de número (CTRL+3)**: permiten enviar y recibir mensajes numéricos de manera interactiva.

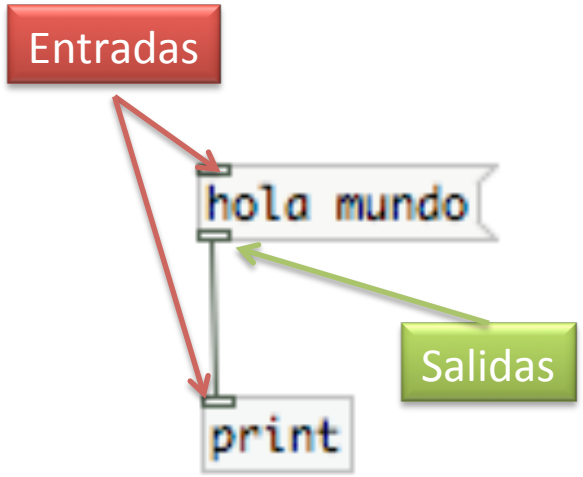
- **Objetos GUI:** permiten la interacción con el usuario de diversas maneras.

- **Arreglos:** almacenan una serie de números.

# Entradas y salidas

- Todos los objetos de Pd tienen cero o mas entradas en la parte superior, y cero o mas salidas en la parte inferior.

- Las salidas de un objeto pueden conectarse a las entradas de otro al arrastrar un “cable” desde la salida hasta la entrada.

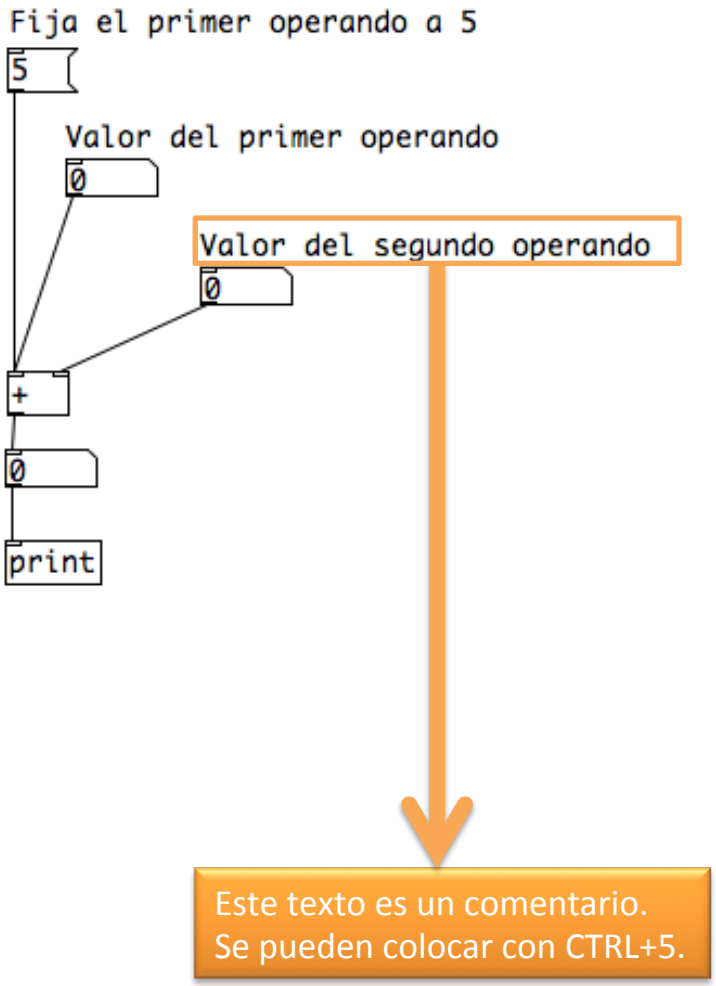


# Operadores

- Pd contiene objetos para realizar muchas de las operaciones que se encuentran en cualquier lenguaje de programación.

- Estos objetos reciben uno o mas mensajes numéricos de entrada, y envían el resultado por la salida cuando reciben un mensaje en su entrada activa.

- Se puede agregar un parámetro de inicialización para el segundo operando.

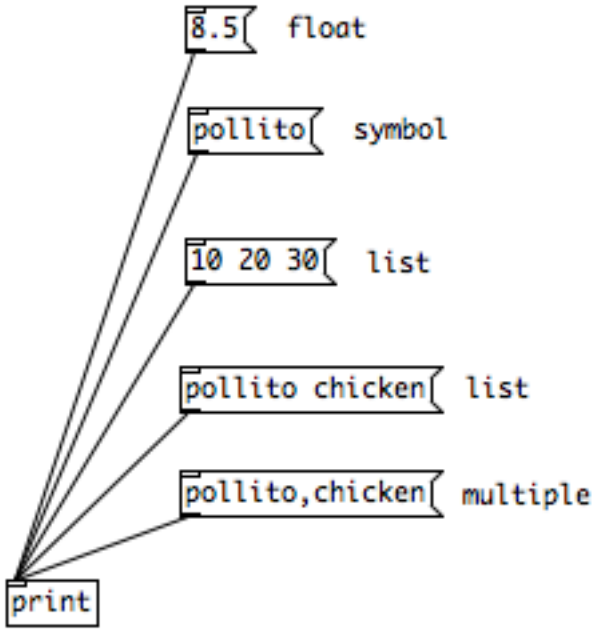


# Operadores y funciones matemáticas

- **[+], [-], [*], [/], [%], [pow]** – Aritmética con mensajes
- **[min], [max]** – Mínimo y máximo
- **[&], [|], [<<], [>>] –** Operadores binarios (bit-wise)
- **[&&], [||] –** Operadores lógicos
- **[sin], [cos], [tan], [atan], [atan2]** - Trigonometría
- **[sqrt], [log], [exp], [abs], [clip]** – Otras funciones
- **[expr]** – Evaluación de expresiones arbitrarias
- **[random]** – Generador de números aleatorios

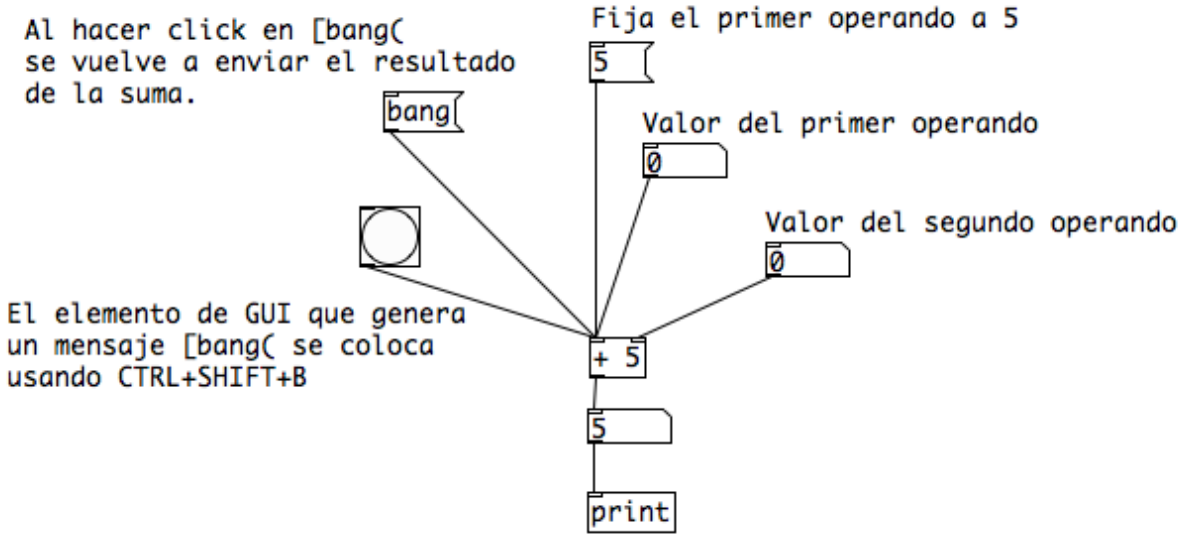
# Tipos de mensajes

- Numéricos (float): cualquier número de punto flotante
- Símbolos (symbol): cualquier cadena que no incluya espacio, coma o punto y coma, y que no sea un número.
- Listas (list): dos o mas símbolos o números separados por espacios.
- Múltiples: dos o mas de los anteriores separados por comas.



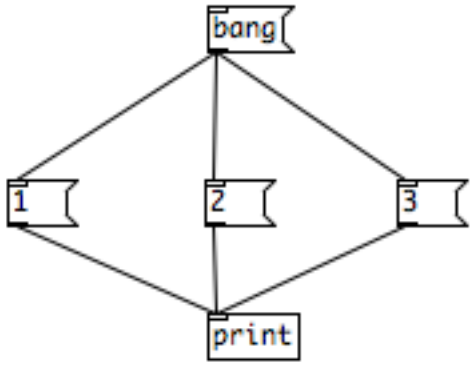
# [bang(

- El mensaje **[bang(** es muy importante en Pd.
- Se utiliza, entre otras cosas, para
  - Hacer que un objeto emita un mensaje de salida
  - Hacer que un objeto realice una tarea específica
  - Coordinar y sincronizar eventos

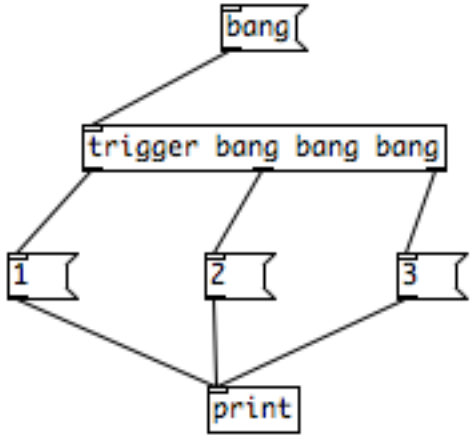


# Coordinación y ruteo de mensajes

- El objeto **[trigger]** (puede abreviarse **[t]**) permite enviar un mensaje a distintos objetos en un orden específico.




Cuando se envía un mensaje a múltiples objetos, los mensajes son enviados en el orden en que las conexiones fueron hechas.

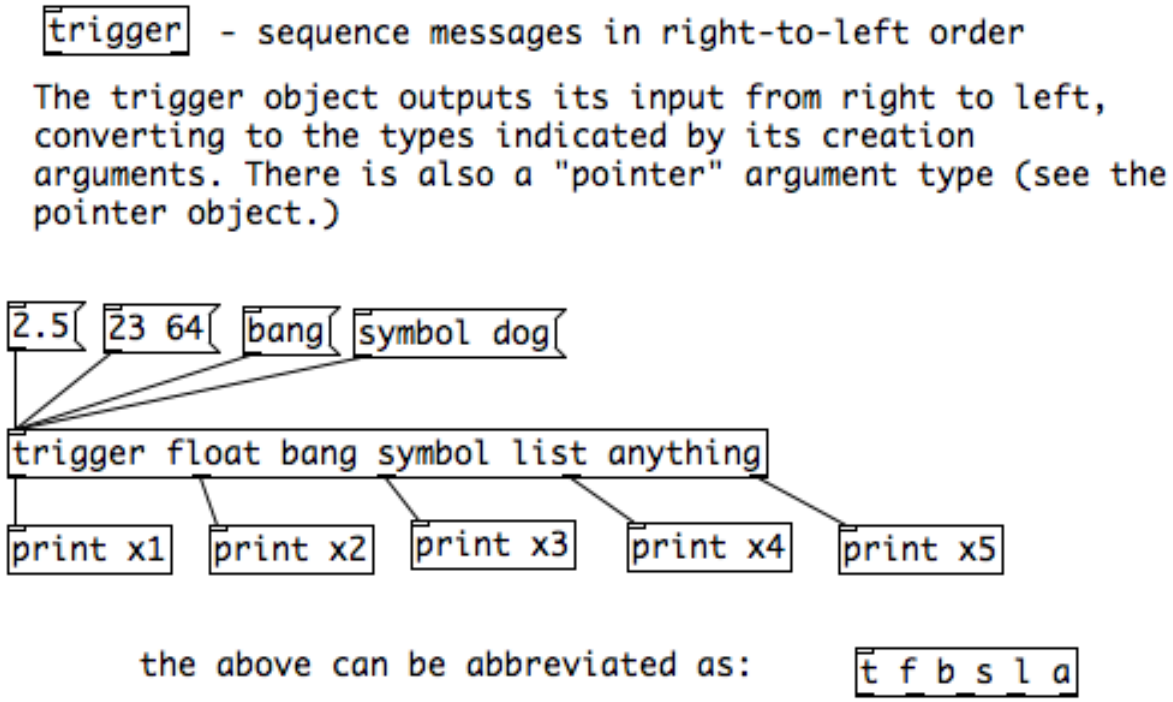



Mediante el objeto [trigger], uno puede enviar los mensajes en un orden específico sin importar el orden en que se hayan hecho las conexiones.

El objeto [trigger] siempre envía los mensajes de derecha a izquierda.

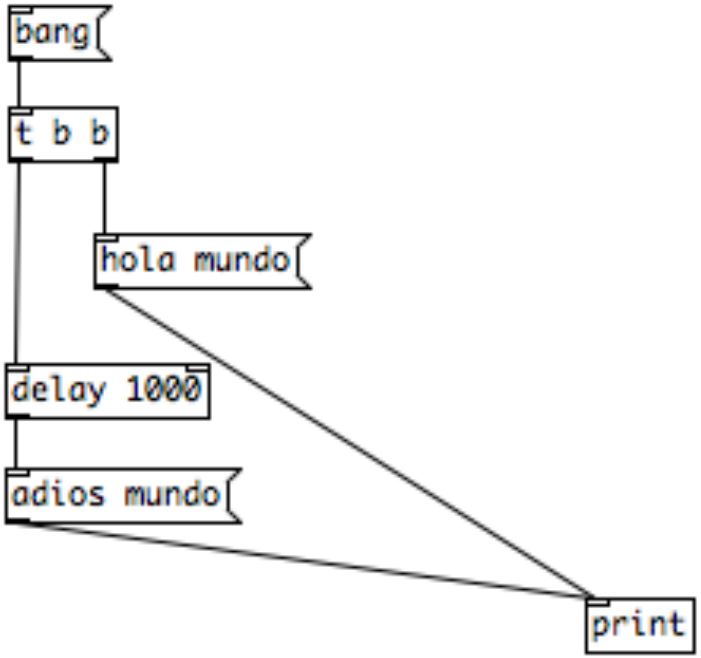
# Mas sobre **[trigger]**

- Los parámetros de inicialización de **[trigger]** indican el tipo de mensaje que se enviará en cada salida.



# Retardo de mensajes

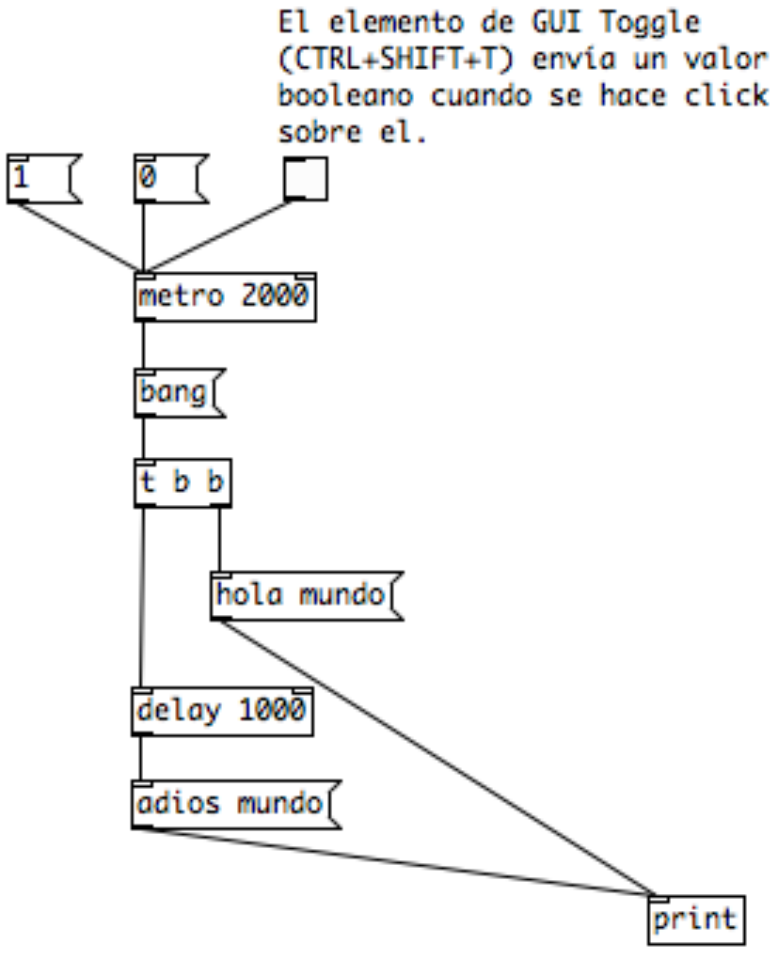
- El objeto **[delay]** recibe un mensaje y lo envía un cierto tiempo después (dado en ms).



# Metrónomos

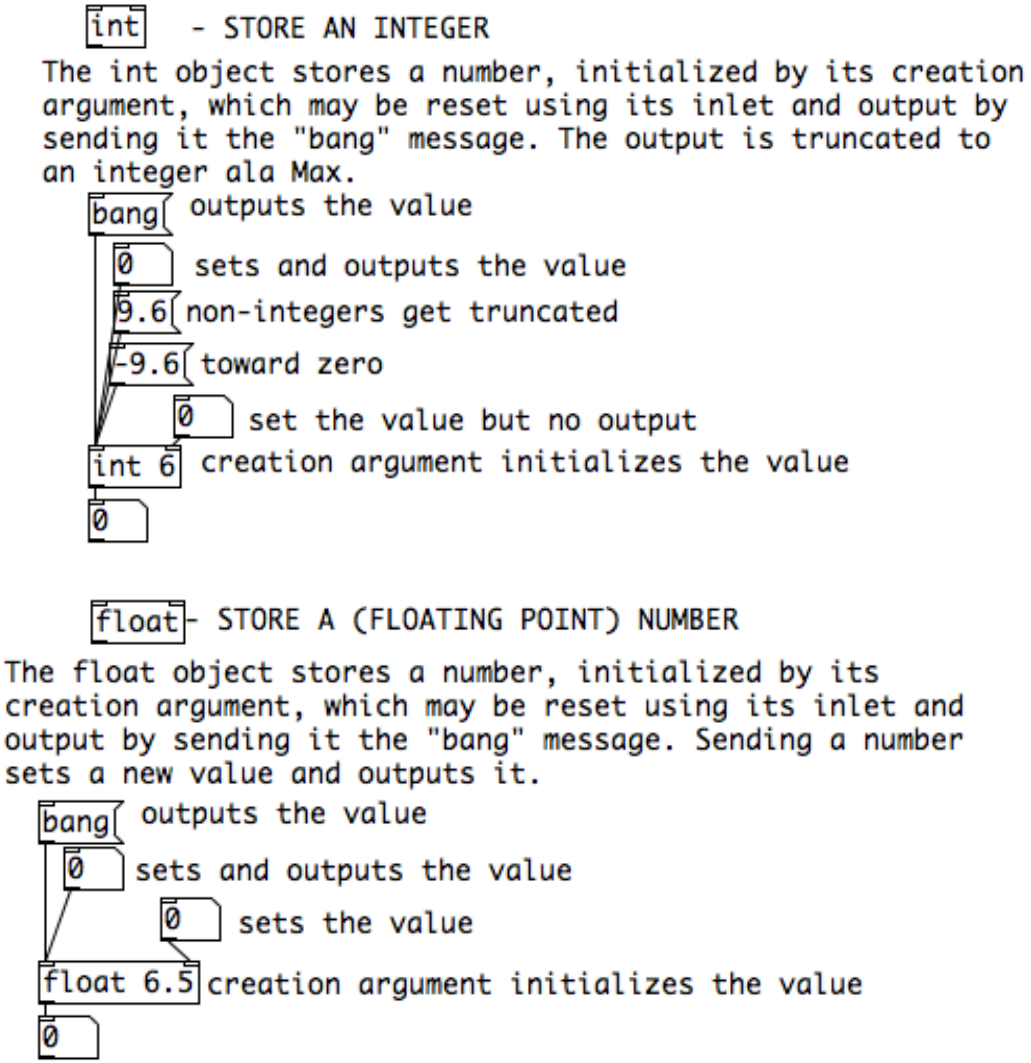
- El objeto **[metro]** emite un mensaje **[bang(** cada cierto intervalo de tiempo, dado en ms.

- La emisión de mensajes puede iniciarse o detenerse enviando un valor booleano (1 ó 0) a la primer entrada de **[metro]**.



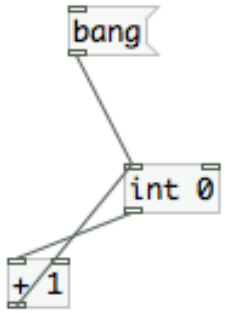
# Variables

- En PureData, las variables son objetos que mantienen su valor hasta que les es asignado uno nuevo.
- Estos objetos cuentan con dos entradas. Ambas sirven para asignar un valor a la variable, pero un mensaje en la entrada izquierda ocasiona que la variable emita su valor a la salida.

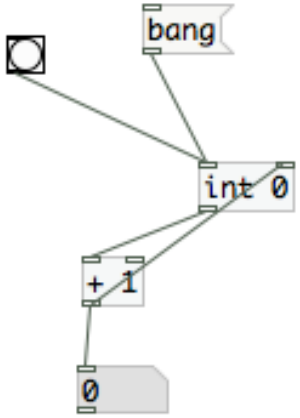


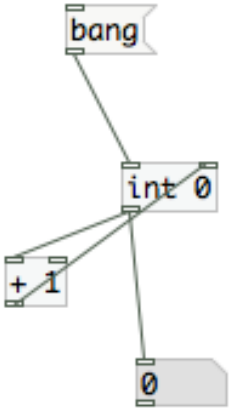
# Contadores

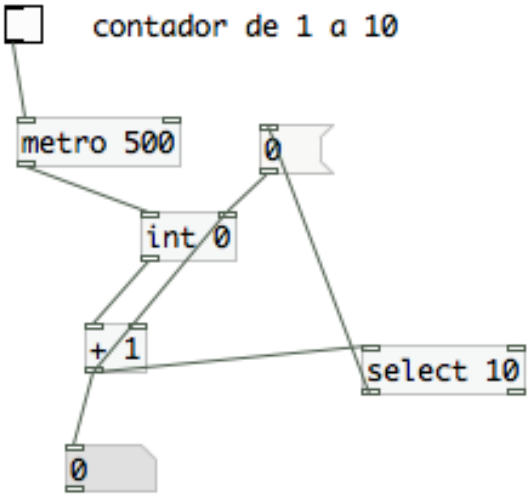
contador mal hecho

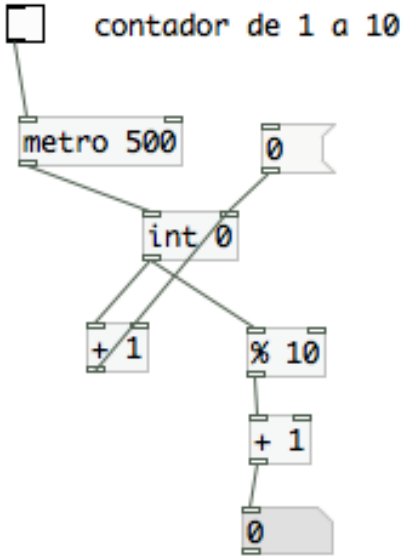



forma correcta de hacer un contador



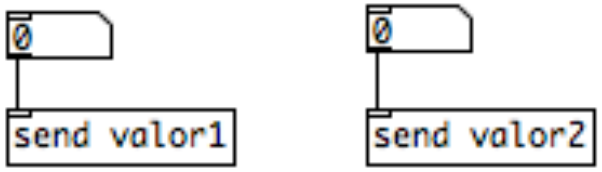



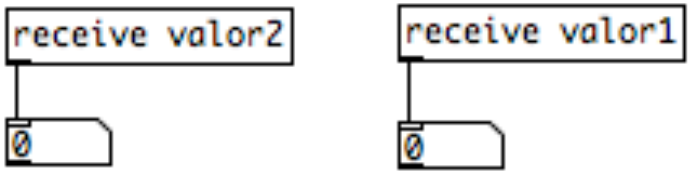
# Envío y recepción remota de mensajes

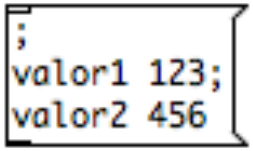
- Los objetos **[send]** y **[receive]** (abreviados **[s]** y **[r]**) permiten enviar mensajes sin usar cables.




[send] envía mensajes a través de un "canal" definido por un nombre.

[receive] recibe los mensajes que son enviados por el canal indicado.




También es posible enviar mensajes hacia un canal específico desde una caja de mensajes.

El canal se especifica al principio del mensaje, y se usa punto y coma para separar mensajes que van a distintos canales.

El primer mensaje siempre se envía a la salida de la caja de mensajes.

# Algunos objetos básicos de control

- **[metro]** – metrónomo
- **[trigger]** – Envío de mensajes en orden
- **[delay]** – Retardo de mensajes
- **[loadbang]** – Envía **[bang(** al cargar el parche

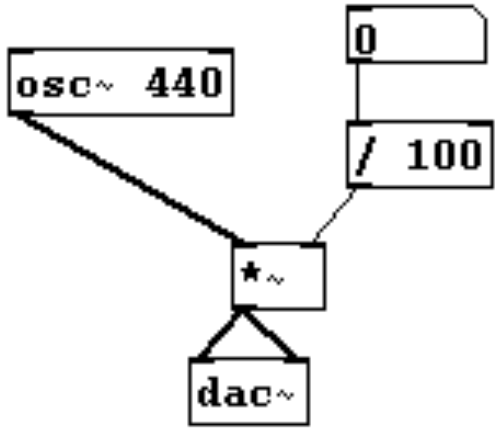
- **[select]** – Selecciona una salida dependiendo de la entrada
- **[send], [receive]** – Distribución de mensajes

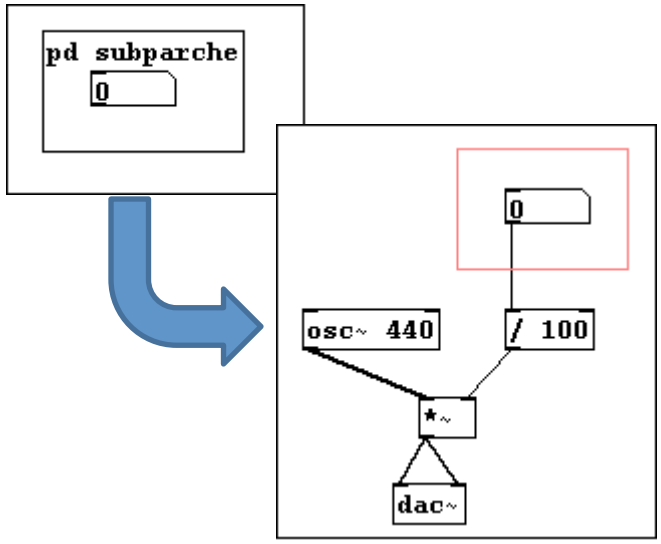
# Sub-parches y abstracción

- **Parche:** Conjunto de elementos de Pd interconectados entre sí, que realizan alguna tarea específica.

- **Sub-parche:** Un parche dentro de otro parche, que se almacena como parte del parche anfitrión.

- **Abstracción:** Un parche independiente (almacenado en un archivo separado) que puede utilizarse dentro de cualquier parche.



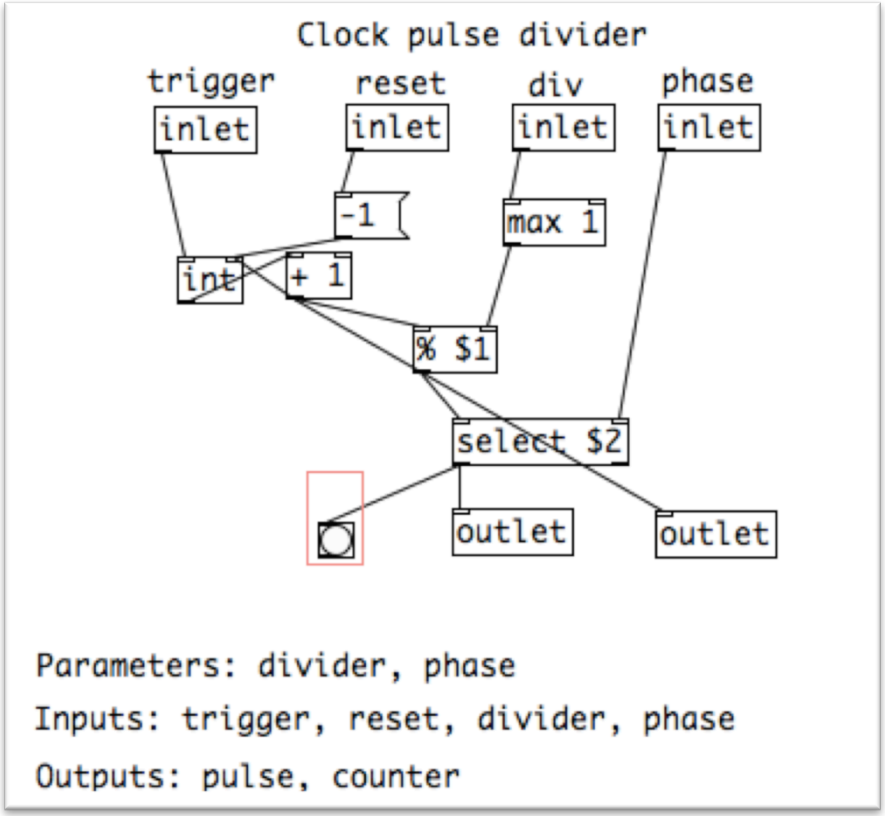
# Algunos objetos básicos (subparches)

- **[pd]** – Crea un subparche dentro de un parche
- **[inlet]** – Agrega una entrada de control
- **[inlet~]** – Agrega una entrada de audio
- **[outlet]** – Agrega una salida de control
- **[outlet~]** – Agrega una salida de audio

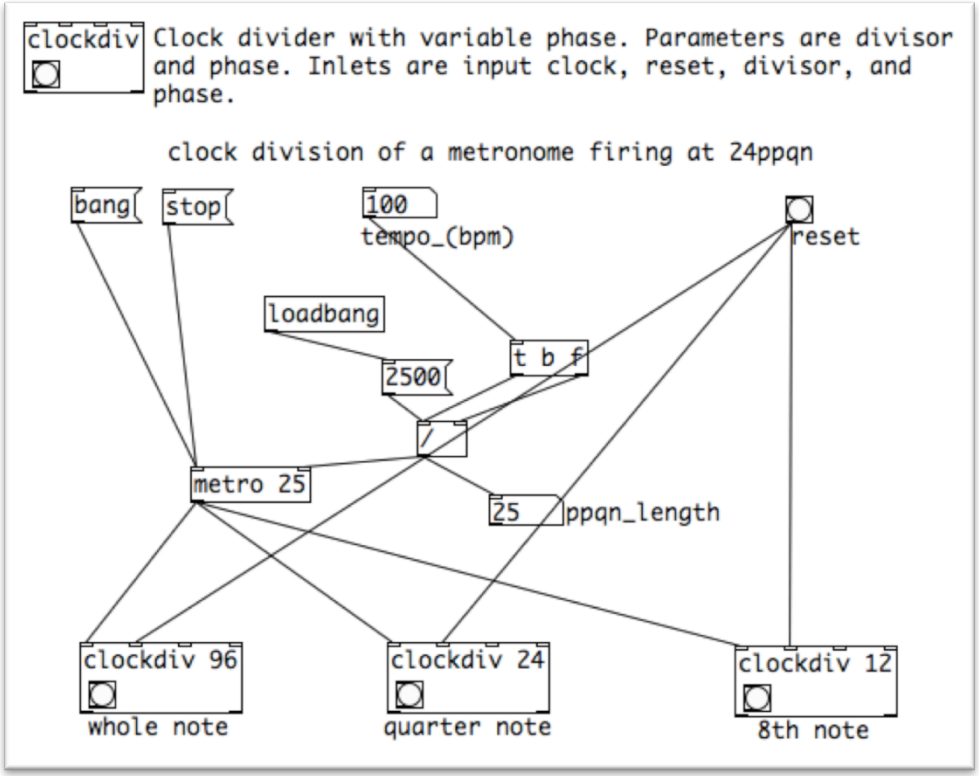
- Dentro de una abstracción, los parámetros de inicialización se denotan por $1, $2, etc.

# Ejemplo de abstracción: Divisor de reloj

clockdiv.pd




clockdiv-help.pd



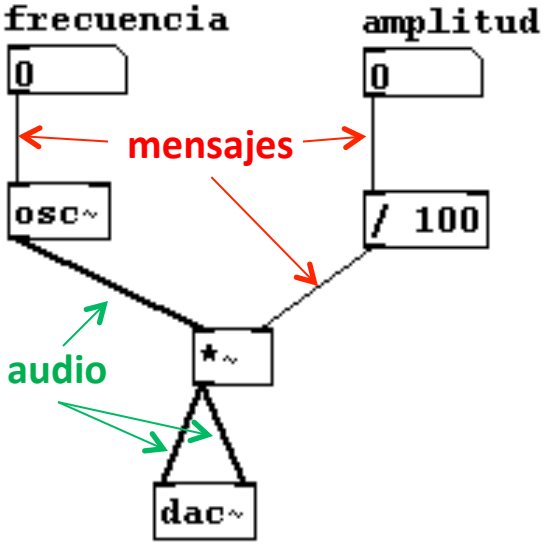
Sesión 2

# SÍNTESIS DE SONIDO

# Tipos de señales en Pd

- **Mensajes:** Se envían de manera esporádica, y se utilizan principalmente para control e interface.

  Muchos objetos de Pd envían un mensaje solamente cuando reciben otro en su entrada izquierda (también llamada *entrada activa*).

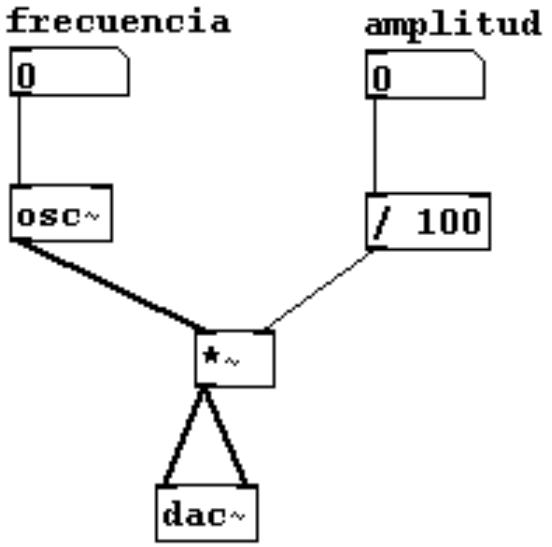


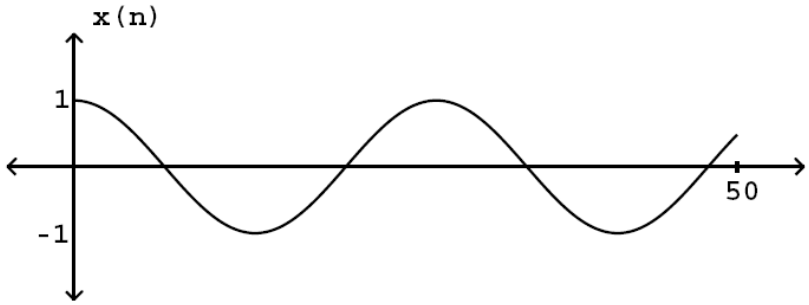

- **Señales de audio:** consisten en flujos continuos que se transmiten a la frecuencia de muestreo especificada en las opciones de audio.

  Los nombres de los objetos que generan señales de audio terminan en ~ (tilde).

# Ondas senoidales

- El sonido consiste de vibraciones (ondas) en un medio elástico como el aire.
- La forma más simple de vibración consiste en una onda senoidal.
- El objeto **[osc~]** produce una oscilación senoidal continua a una frecuencia dada en Hertz.
- El objeto **[dac~]** envía una señal digital a través de la tarjeta de audio de la computadora para que podamos escucharla.



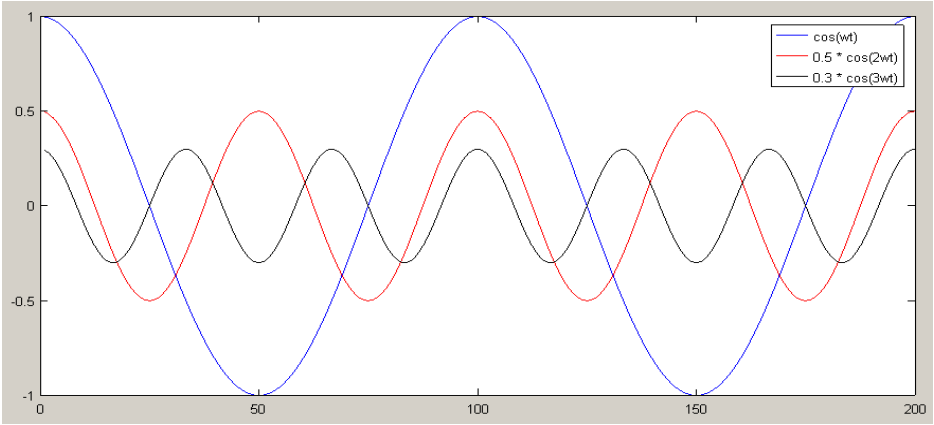
# Operadores y funciones para señales

- **[+~], [-~], [*~], [/~], [pow~]** – Aritmética con señales
- **[cos~]** – Coseno de 2*pi veces la entrada
- **[sqrt~], [rsqrt~]** – Raiz cuadrada y su recíproco
- **[max~], [min~]** – Máximo y mínimo
- **[clip~]** – Recorte
- **[expr~], [fexpr~]** – Expresiones arbitrarias
- **[abs~]** – Valor absoluto
- **[exp~], [log~]** – Exponencial y logaritmo
- **[noise~]** – Ruido blanco uniforme en [-1,1]

# Ondas periódicas

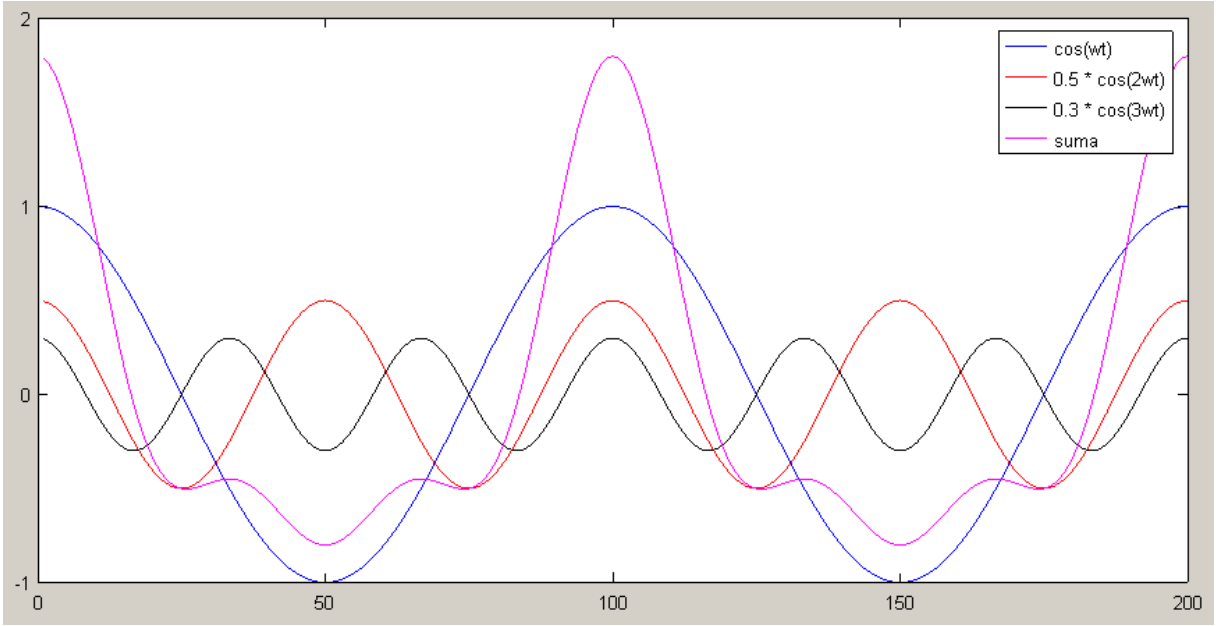
- Una senoidal con frecuencia ***f*** (en Hertz) tiene un periodo de ***1/f*** segundos.

- Cualquier senoidal con frecuencia ***kf***, donde ***k*** es entero positivo, se repite cada ***1/f*** segundos. Estas frecuencias se llaman *armónicos* de ***f.***



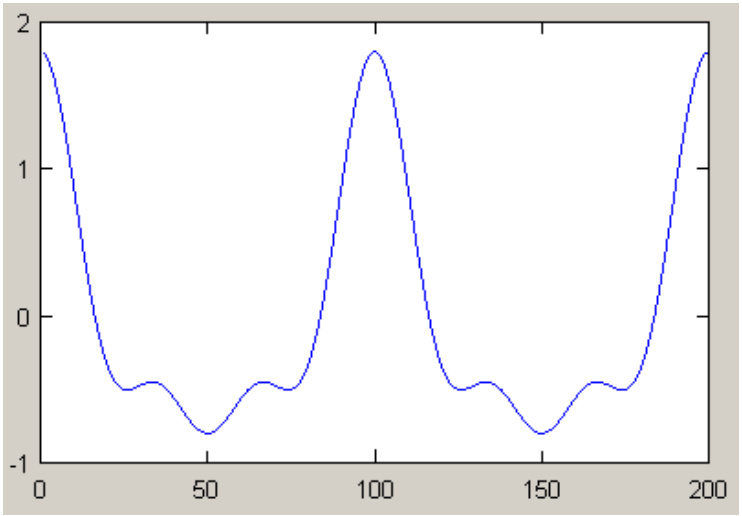
# Ondas periódicas

- En general, cualquier suma de senoidales con frecuencias $k_1 f$, $k_2 f$, …, $k_n f$ tendrá periodo $1/f$, donde a $f$ se le llama la *frecuencia fundamental* de la onda.
- Toda onda periódica puede descomponerse como la suma de senoidales armónicas con distintas frecuencias y fases (series de Fourier).



# Espectro de frecuencias

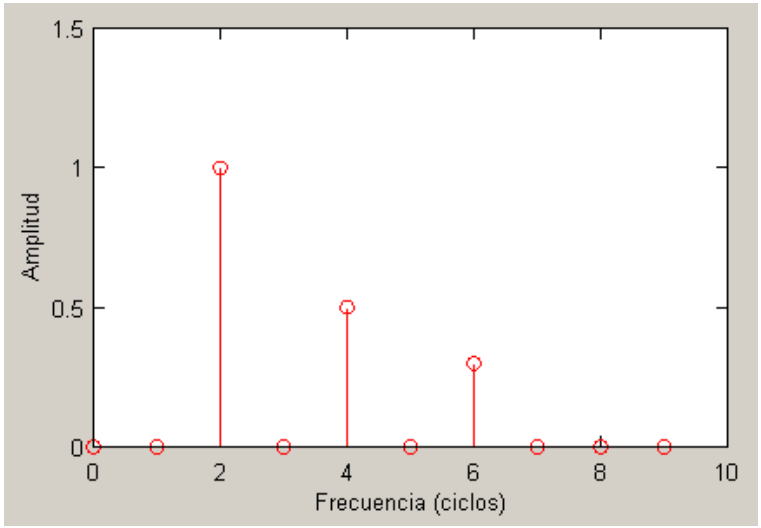
- El espectro de frecuencias de una onda indica la amplitud (y la fase) de cada una de las componentes senoidales presentes en la onda.




Señal

FFT

IFFT




Espectro

# Propiedades de un sonido

- Los sonidos que percibimos tienen principalmente cuatro atributos que los distinguen: volumen, tono, timbre, y duración.

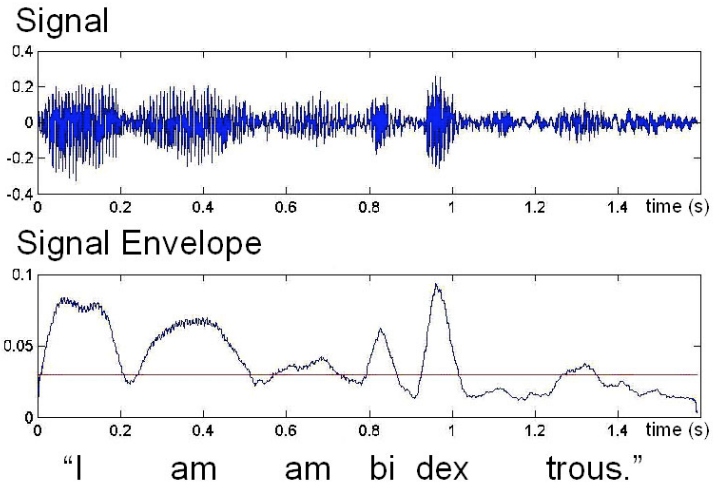
- Los primeros tres atributos están relacionados con las propiedades de las ondas periódicas de la siguiente manera:

| Atributo perceptual | Propiedad física |
|---|---|
| Volumen | Amplitud |
| Tono | Frecuencia fundamental |
| Timbre | Espectro |

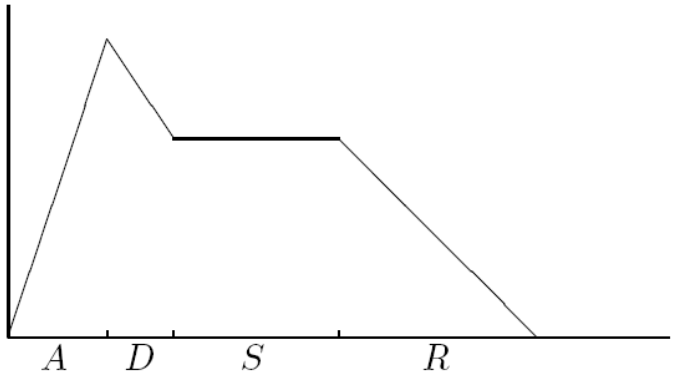
- Además, estos atributos pueden variar con el tiempo durante la duración del sonido.

# Volumen

- El volumen de un sonido puede modificarse simplemente multiplicando la señal por una función *envolvente*.

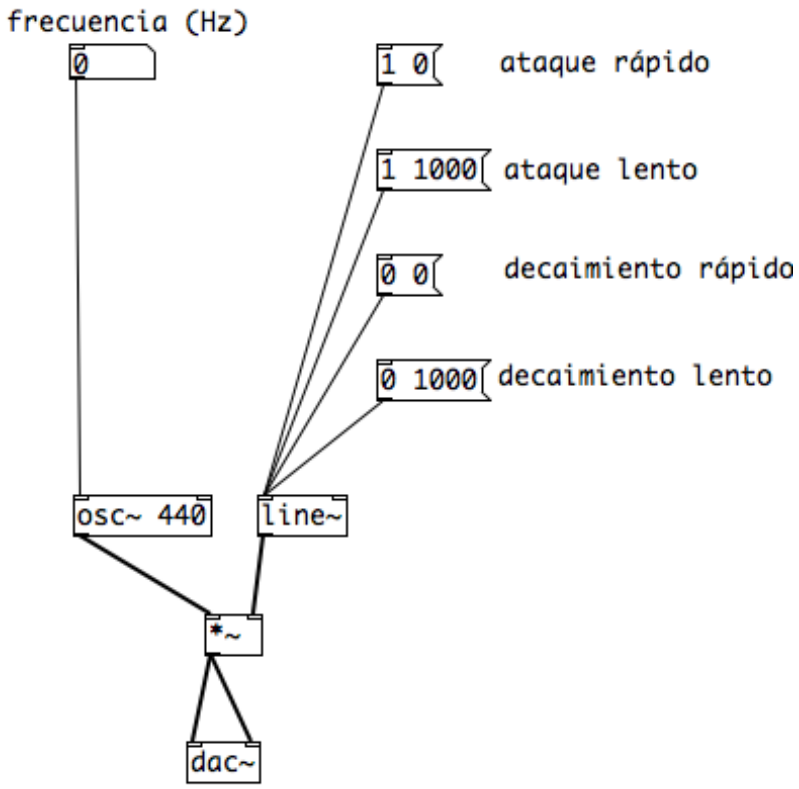



- Una manera simple de simular envolventes es mediante una función lineal a trozos.

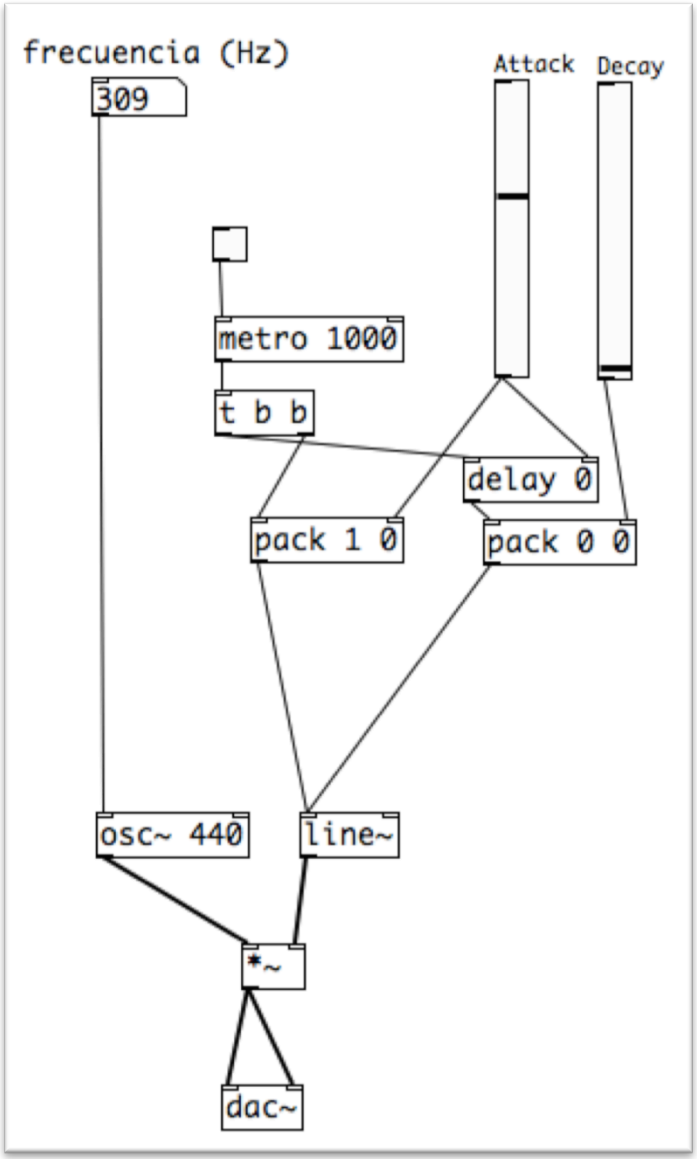


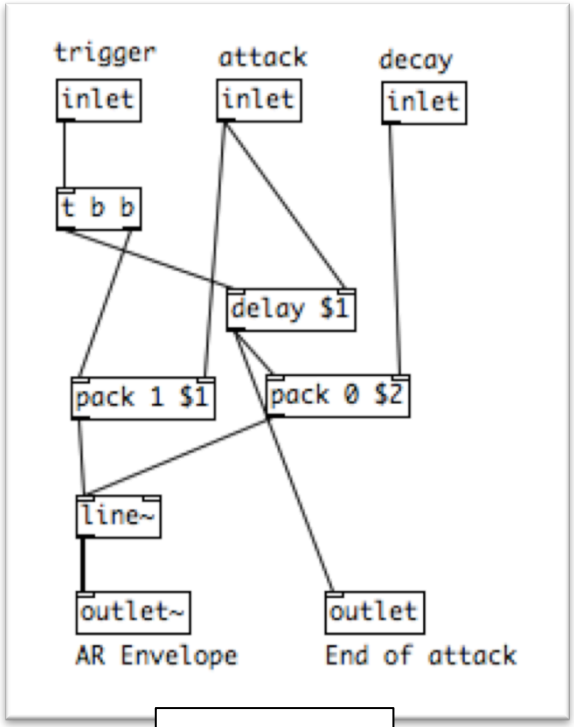
# Funciones lineales a trozos

- El objeto **[line~]** sirve para generar rampas cuya altura y duración están determinadas por los mensajes que recibe.
- Los mensajes consisten en dos números que especifican, respectivamente, la altura que la rampa debe alcanzar a partir del nivel actual, y el tiempo que debe tardar en alcanzar esa altura.
- Un mensaje con un solo número ocasiona un cambio inmediato en el nivel de salida.

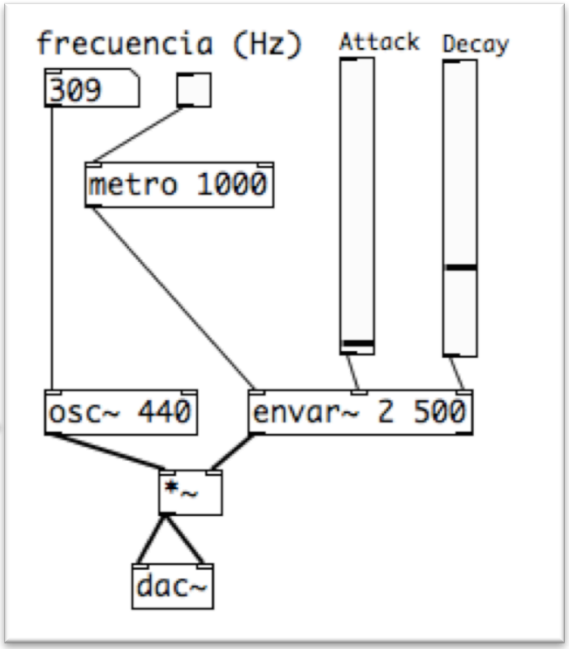


# Envolventes simples con [line~]




envar~.pd



# Timbre

- El timbre es el atributo más complejo de un sonido, y es el que nos hace diferenciar, por ejemplo, un violín de una flauta o un piano.

- Existen varias técnicas que nos permiten modelar el espectro de una onda, tales como:
  - Modulación de amplitud
  - Modulación de frecuencia
  - Síntesis substractiva y filtros

# Timbre y espectro

- De acuerdo al Teorema de Fourier, toda señal periódica se puede representar como la suma de señales oscilatorias simples (senos y cosenos) con distintas amplitudes y fases.

- Una señal armónica se compone de oscilaciones cuyas frecuencias son todas múltiplo de una frecuencia fundamental.

- En una señal inarmónica, no se puede percibir una frecuencia fundamental.

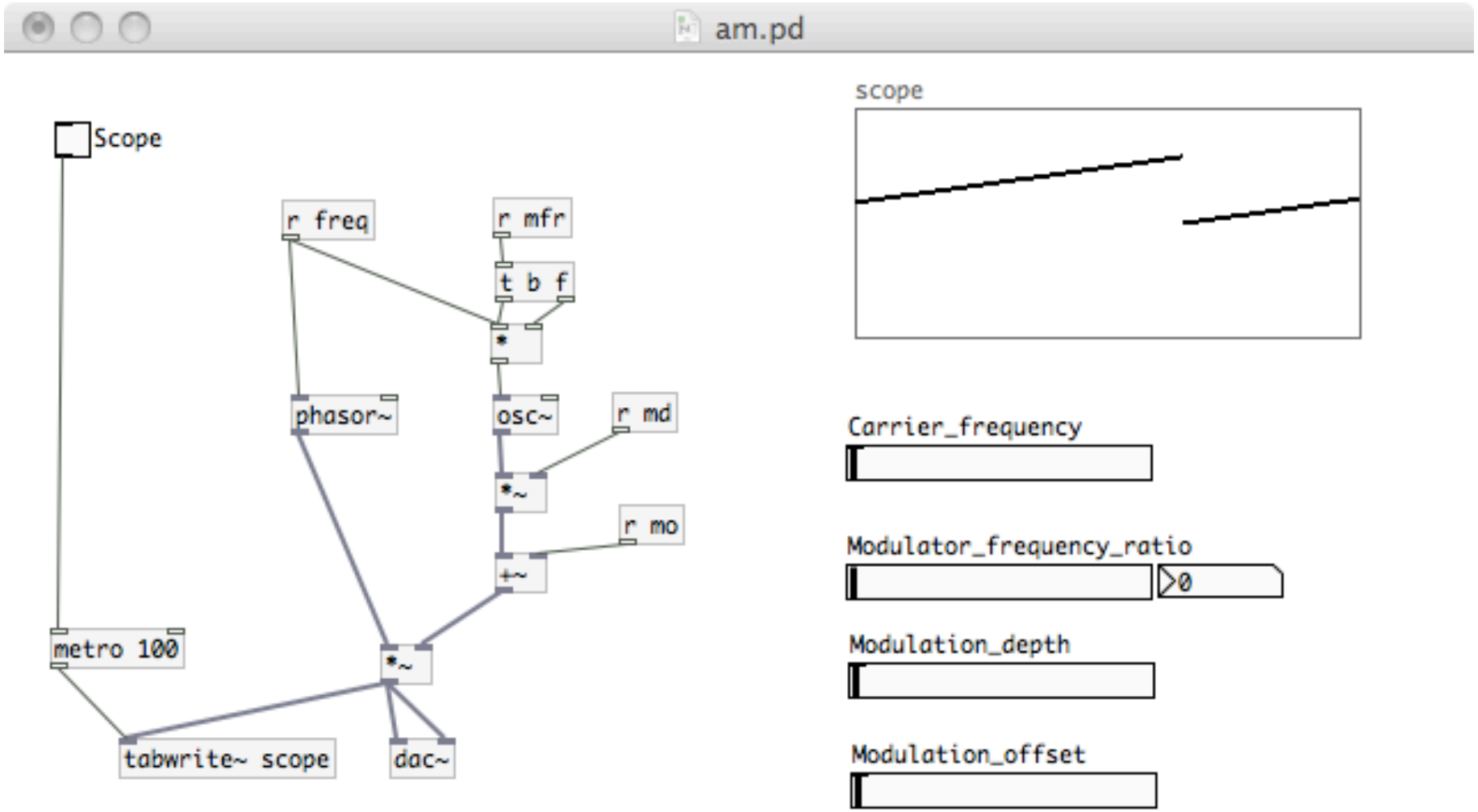
# Modulación

- Modular significa cambiar un parámetro de una señal (tono, amplitud, etc.) de manera continua, de acuerdo al valor de otra señal.

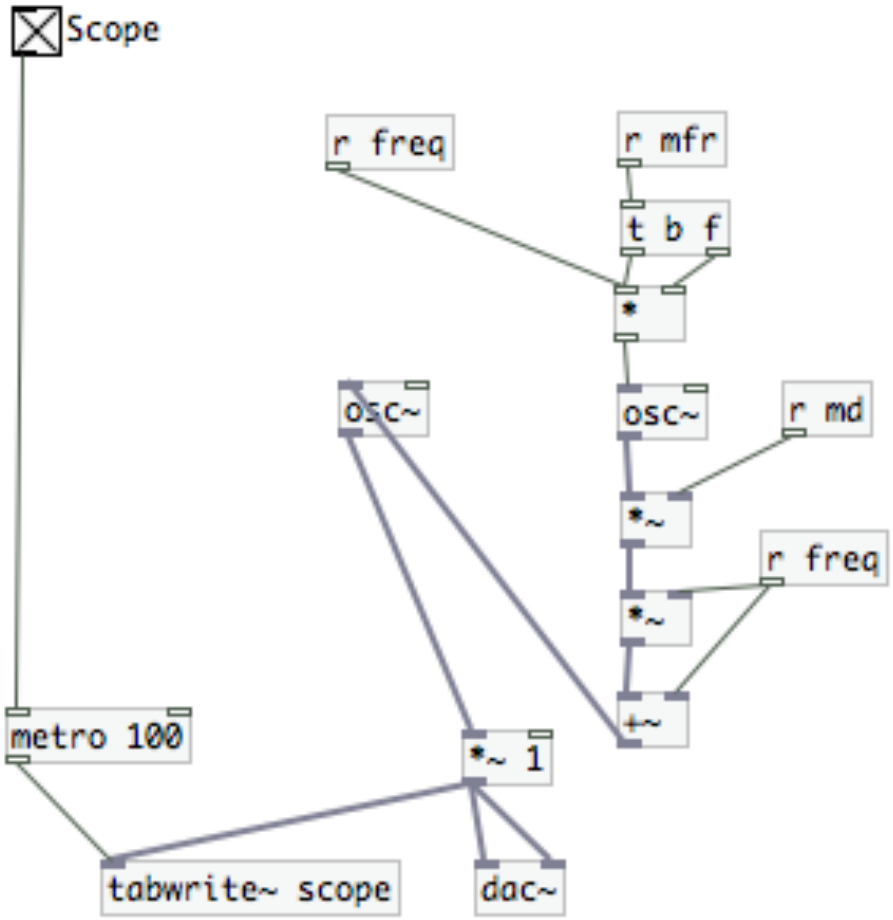
- Si la modulación es lenta, el oído puede percibir la “forma” de la señal moduladora.

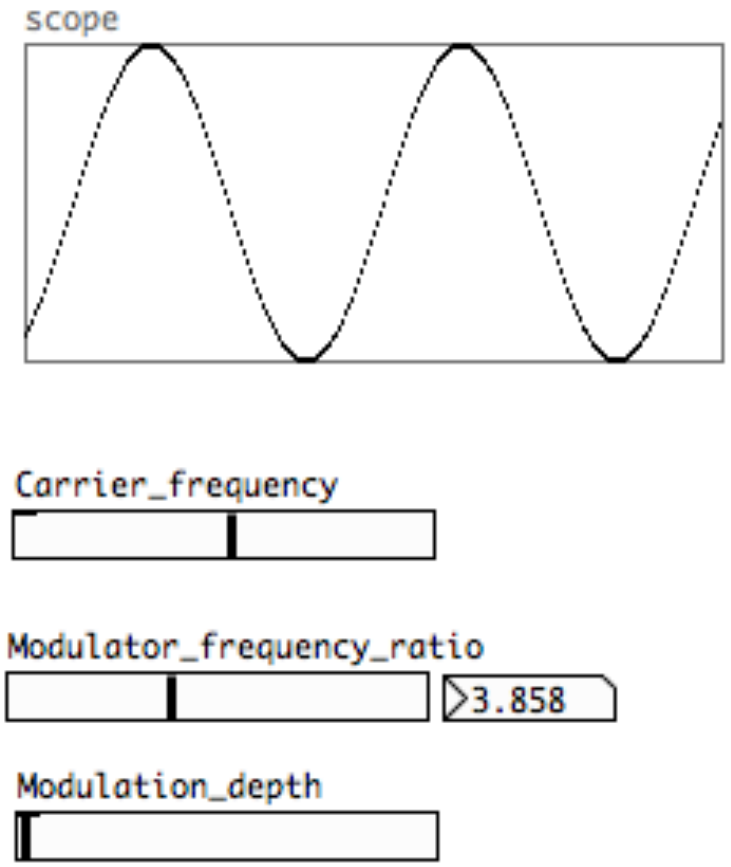
- Si la modulación es rápida (frecuencias audibles), se percibe un cambio de timbre en la señal portadora.

# Modulación de amplitud (AM)

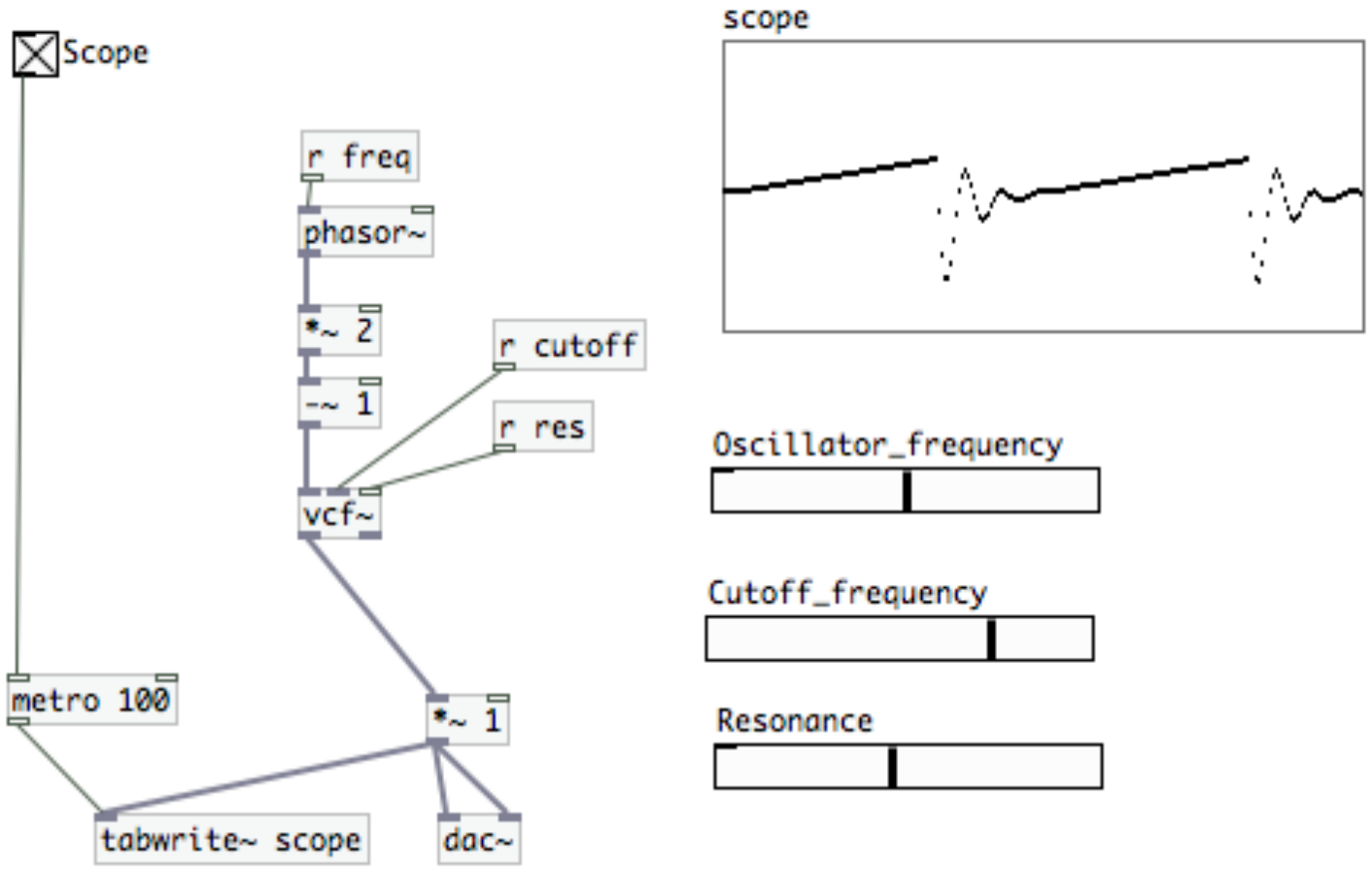


# Modulación de frecuencia (FM)



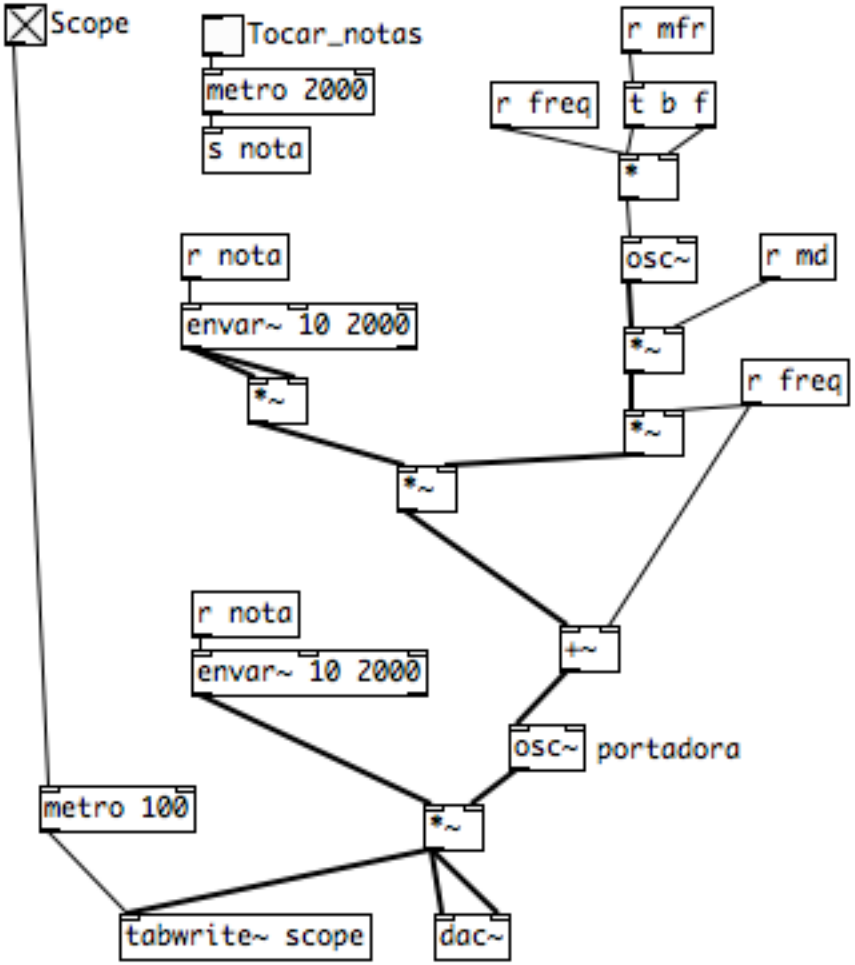



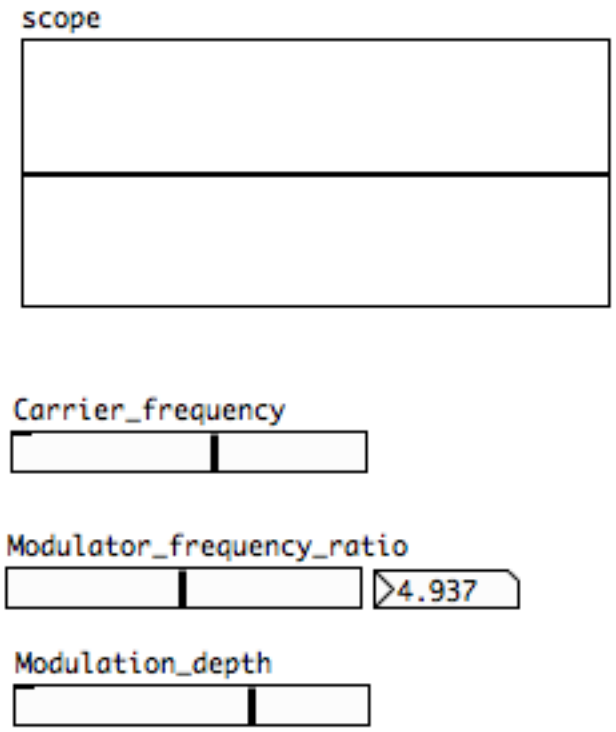
# Filtros y síntesis substractiva



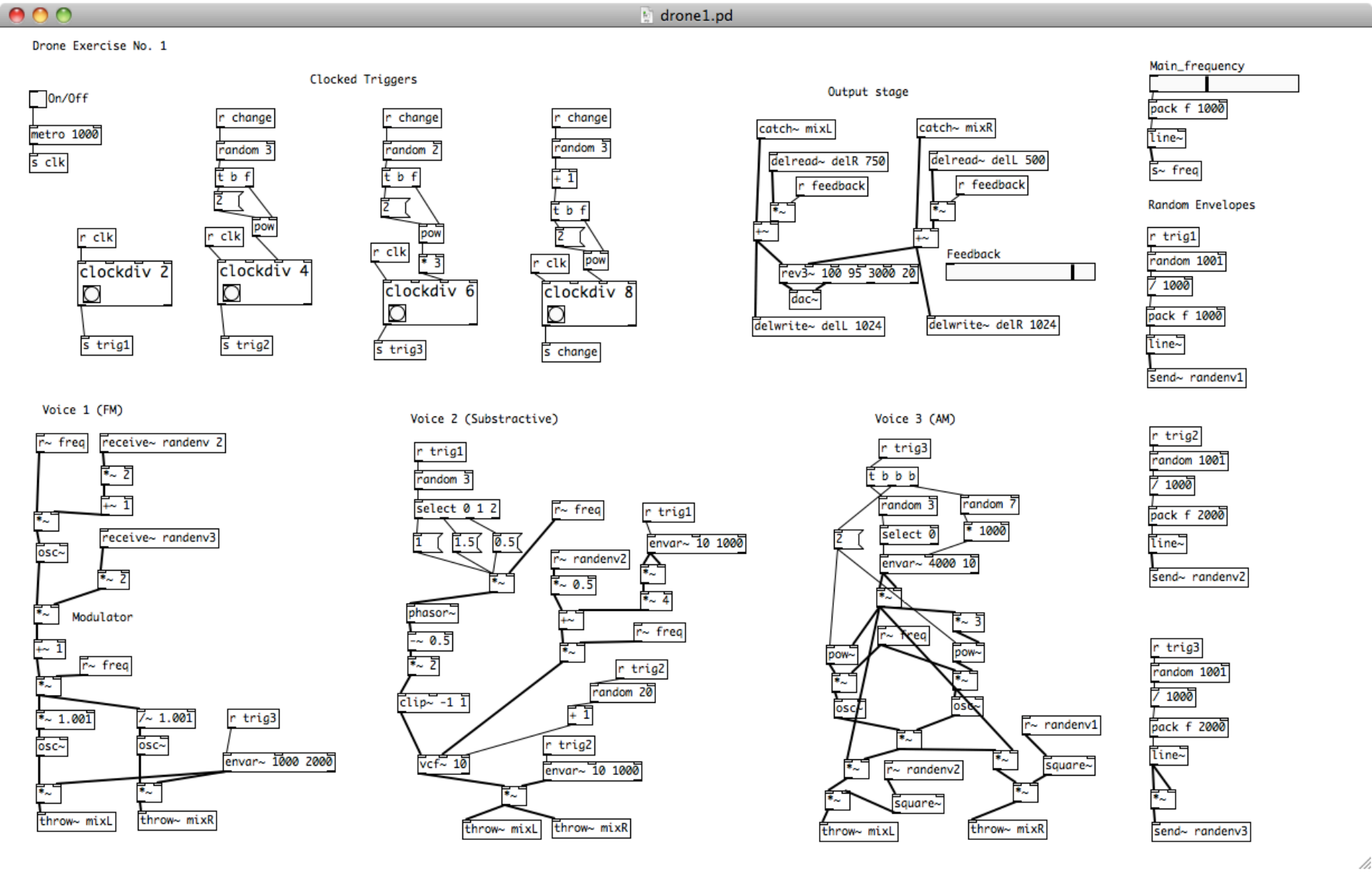
# Modulando los moduladores

Los sonidos naturales no suelen tener timbres estáticos, por lo que una manera de generar sonidos mas interesantes es modificando de manera continua los parámetros de síntesis. En otras palabras, hay que modular los moduladores.



# Combinando múltiples técnicas



# Algunos objetos básicos (audio)

- **[osc~]** – Oscilador senoidal
- **[adc~], [dac~]** – Entrada y salida de audio (estereo)
- **[line~]** – Generador de rampas
- **[lop~], [hip~]** – Filtros pasa bajas y pasa altas
- **[bp~], [vcf~]** – Filtros pasa-banda resonantes
- **[noise~]** – Ruido blanco uniforme
- **[phasor~]** – Generador de diente de sierra
- **[send~], [receive~]** – Distribución de señales
- **[throw~], [catch~]** – Buses aditivos
- **[delwrite~], [delread~], [vd~]** – Canales de retardo

Sesión 3

# ARREGLOS

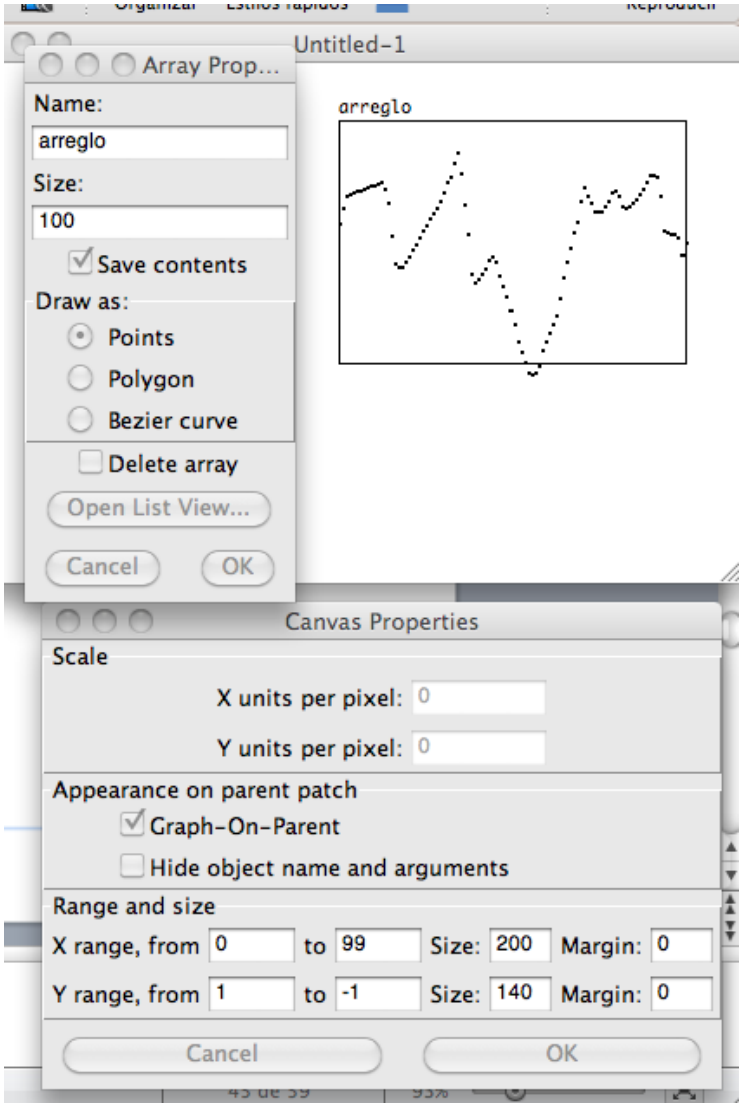
# Motivación

- Los arreglos son fundamentales en programación, ya que permiten almacenar y organizar datos.

- En Pd también pueden utilizarse arreglos, por ejemplo para:
  - Almacenar las propiedades de las notas de una melodía
  - Definir funciones que permitan transformar una secuencia de datos de distintas maneras
  - Implementar fuentes de modulación complejas
  - Almacenar y reproducir señales de audio (sonidos)

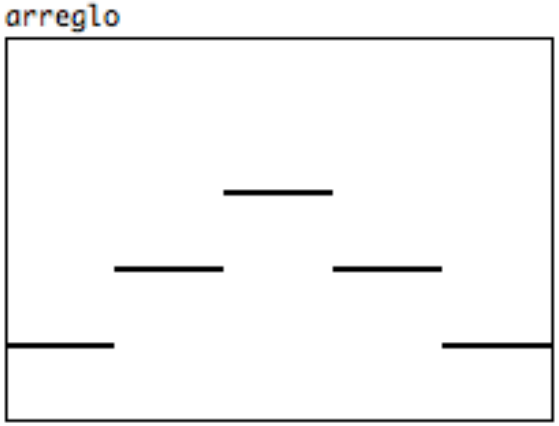
# Creación de un arreglo

- Para agregar un arreglo a un parche de Pd, se utiliza el menú Put->Array.
- Todo arreglo se identifica mediante un nombre, y debe tener un tamaño (número de elementos) definido.
- Las propiedades y el rango de despliegue de un arreglo se pueden modificar haciendo click-derecho sobre él.
- También es posible crear arreglos usando el objeto **[table]** y usando el nombre y tamaño del arreglo como argumentos: e.g., **[table arreglo 100]**



# Inicialización de un arreglo

- Para inicializar un arreglo, se le puede enviar un mensaje que consiste en una lista de números.




- El primer número indica el índice del arreglo a partir del cual se escribirán los datos. El resto de la lista son los datos a almacenar en el arreglo.

```
;
arreglo 0 1 2 3 4 5
```

```
;
arreglo 0 1 1 1 1 1
```

```
;
arreglo 2 3 2 1
```

# Edición de un arreglo

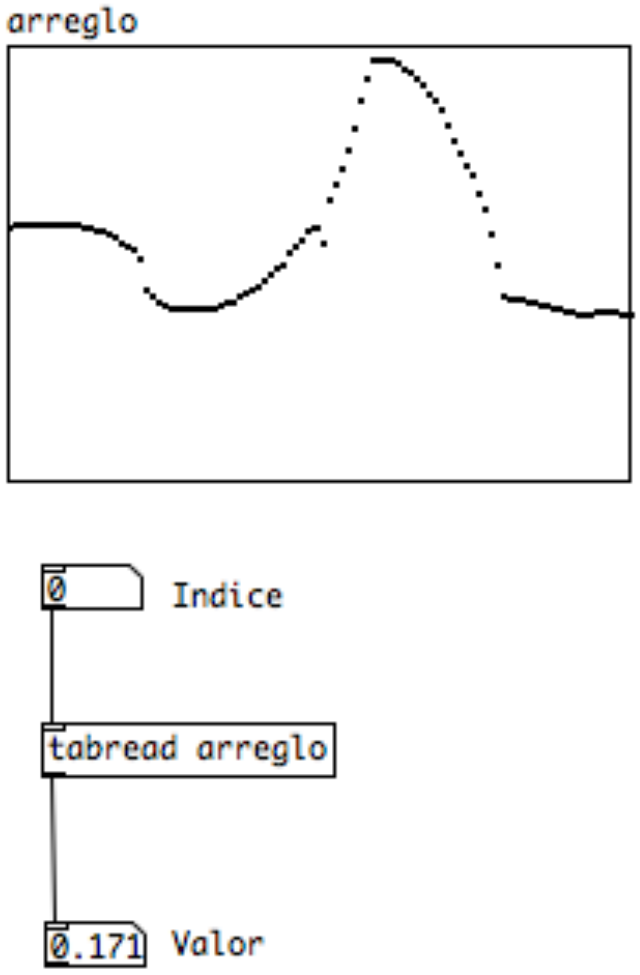
- En Pd los arreglos se muestran de forma gráfica.
- Para editar los datos, solamente hay que arrastrar el apuntador del mouse sobre la gráfica del arreglo en modo de Ejecución.
- Para una edición mas fina, hacer click derecho sobre el arreglo y elegir la opción Open.
- Para una edición aún mas fina, abrir las propiedades del arreglo y elegir “Open List View…”

# Lectura de un arreglo

- Para leer un arreglo se usa el objeto **[tabread]**, el cual lleva como argumento el nombre del arreglo.

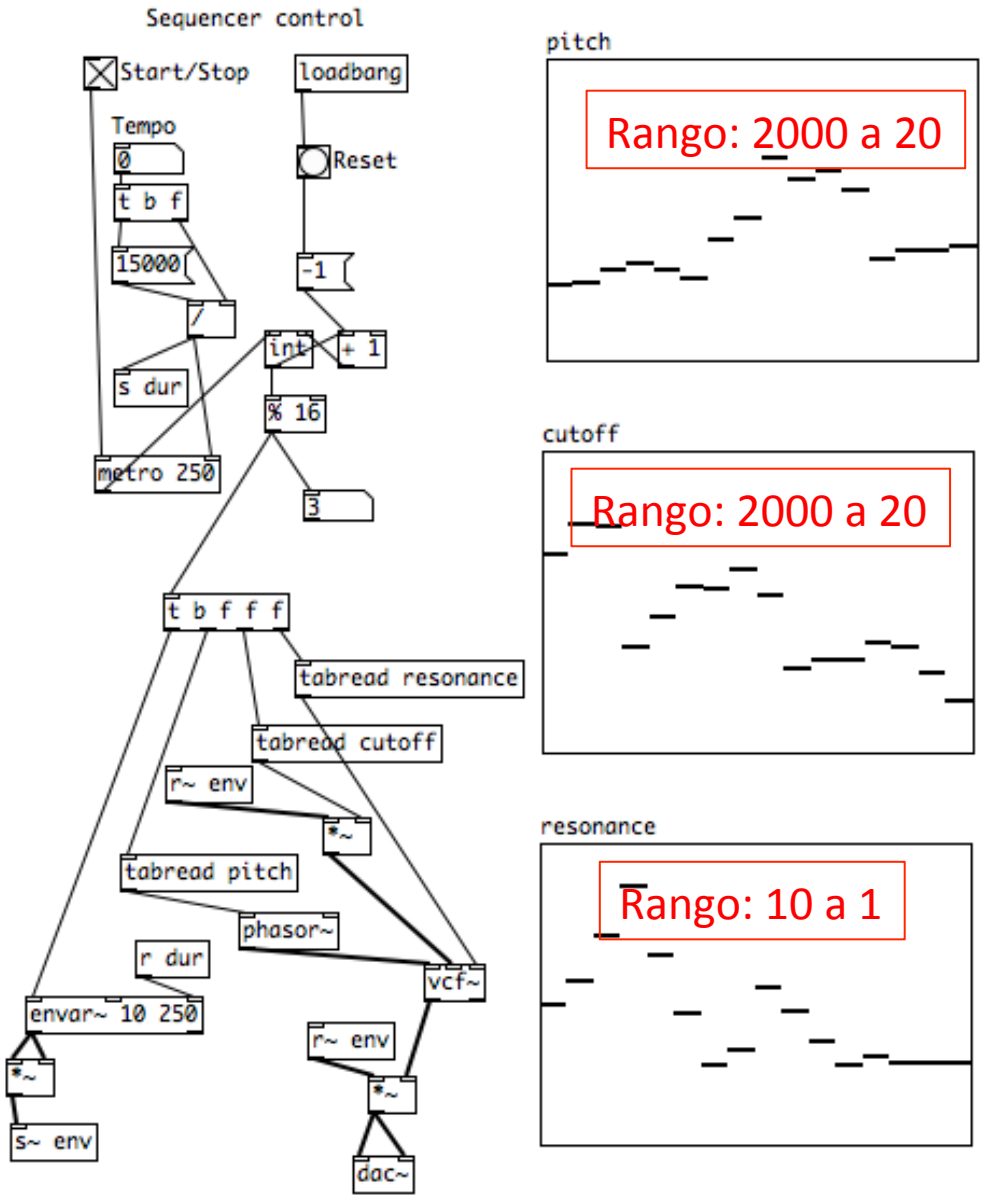
- **[tabread]** recibe como entrada el índice del elemento que se desea leer, y devuelve como salida el valor del elemento.

- Al igual que en C y C++, los arreglos en Pd se indexan a partir de cero.



# Secuenciación básica

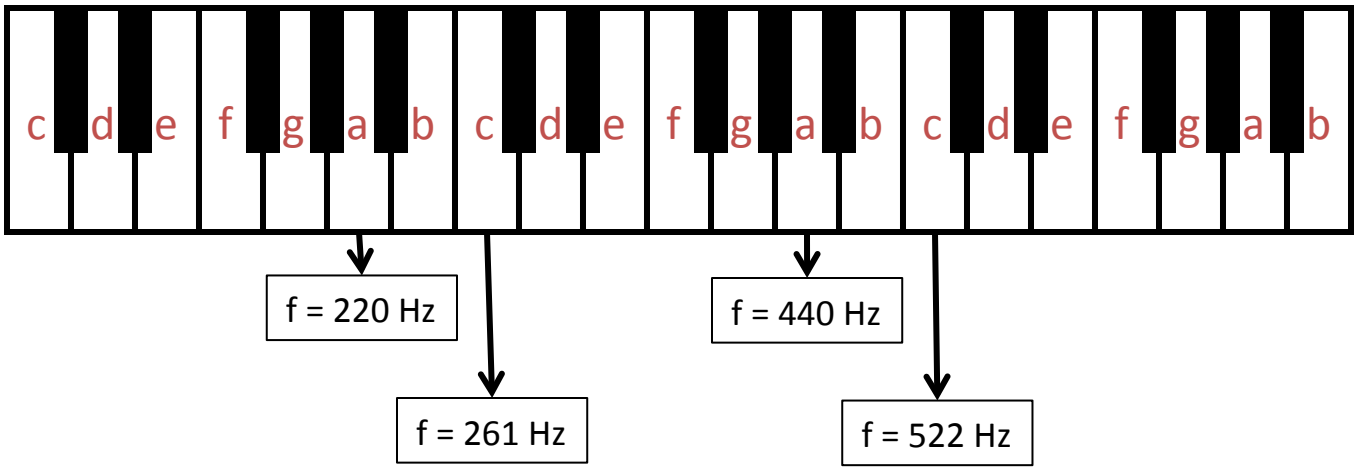
- Una de las principales aplicaciones musicales de los arreglos es la implementación de secuenciadores.
- La idea básica es leer cíclicamente un arreglo (o más) usando contadores módulo-N.



# Tono

- En el caso de un instrumento musical, el tono determina la frecuencia fundamental que se está tocando en un momento determinado.

- La relación entre el tono y la frecuencia es exponencial: cada octava se duplica la frecuencia.



# Entonaciones

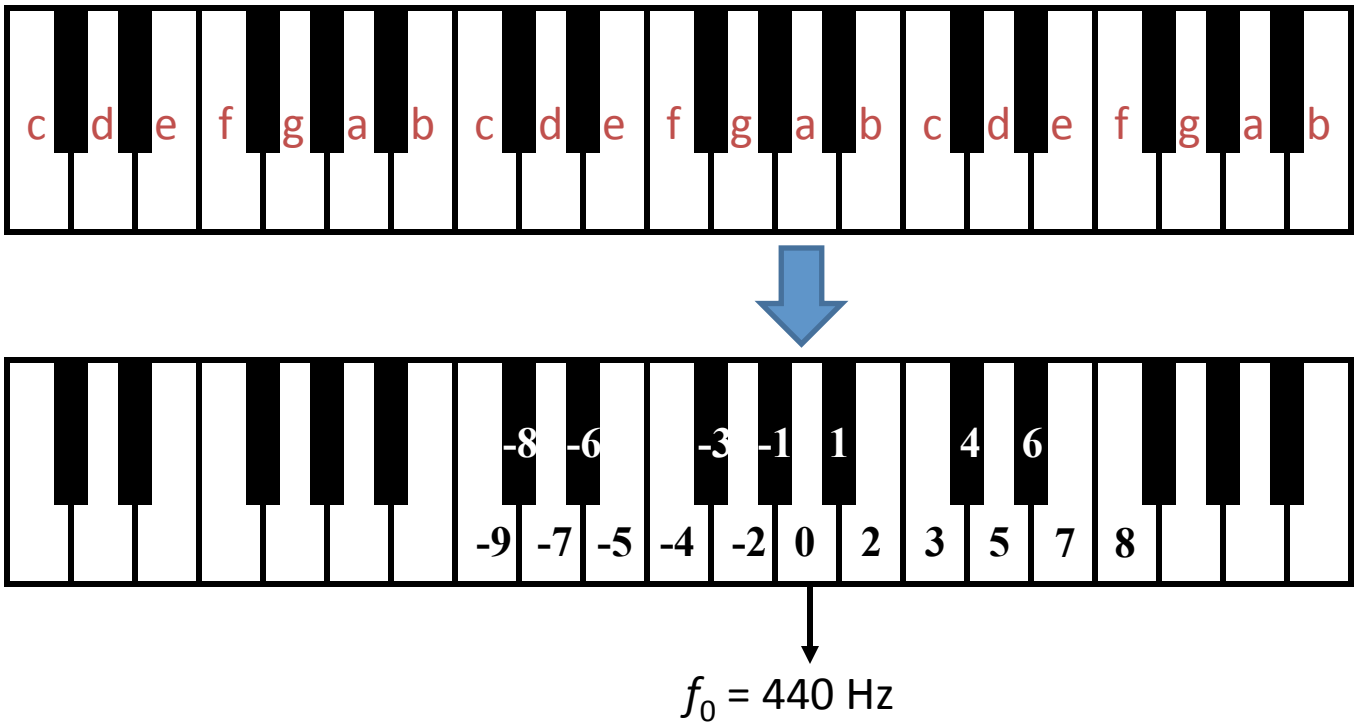
- Una entonación describe la relación entre los tonos y las frecuencias que les corresponden.
- En una entonación regular, se divide la octava en intervalos regulares (en escala logarítmica).
- Ya que la proporción de frecuencia de un tono a su octava superior es 2:1, entonces la distancia entre semitonos es siempre de $2^{1/12}$:1 en la música occidental.

# Numeración de las notas

- Podemos asociar un número entero ***n*** a cada nota y calcular su frecuencia como:

$$f = f_0 \cdot 2^{n/12},$$

donde $f_0$ es la frecuencia correspondiente a la nota 0.

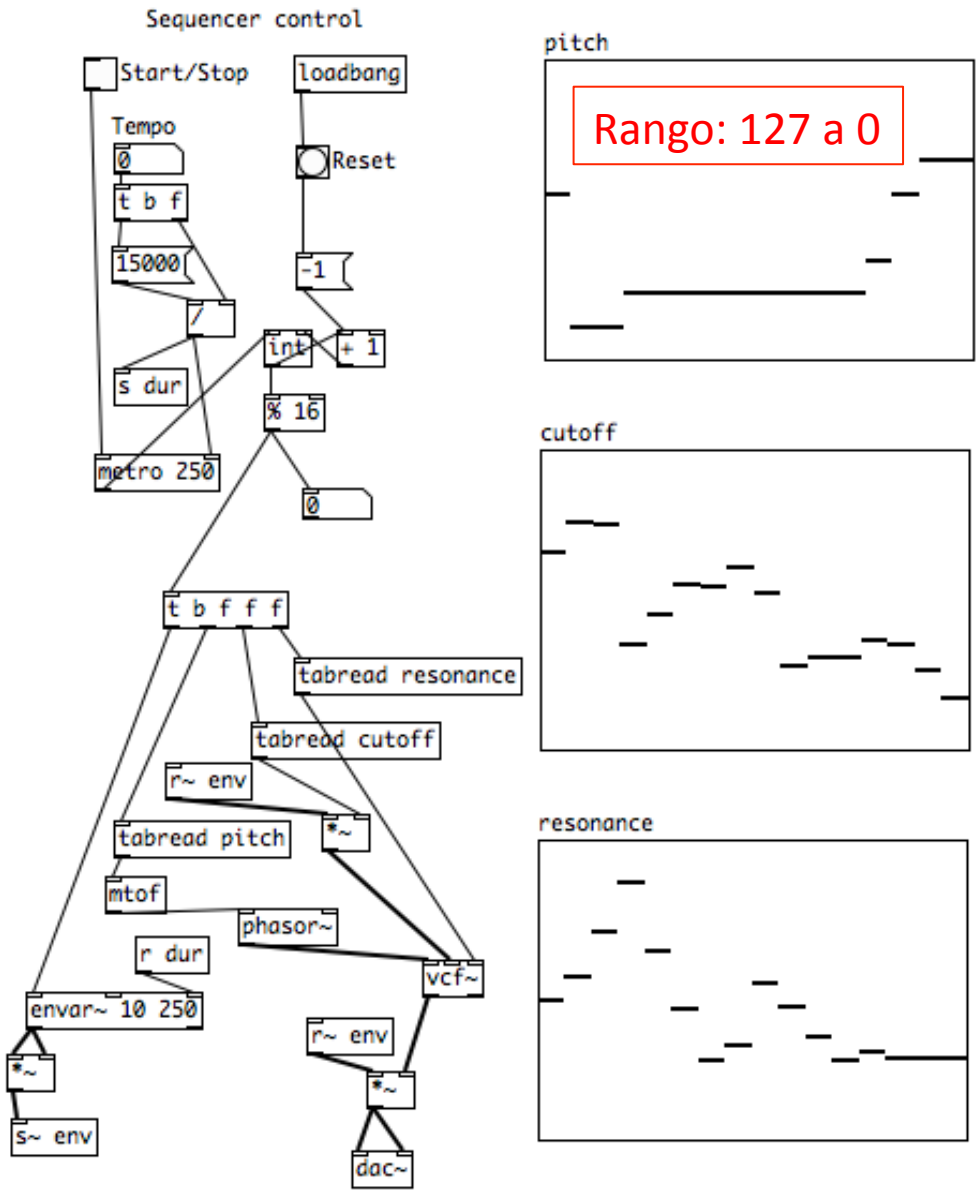


# Objeto [mtof]

- El objeto **[mtof]** (MIDI to frequency) realiza un mapeo similar, pero donde la nota base (440 Hz) corresponde al entero n=69.

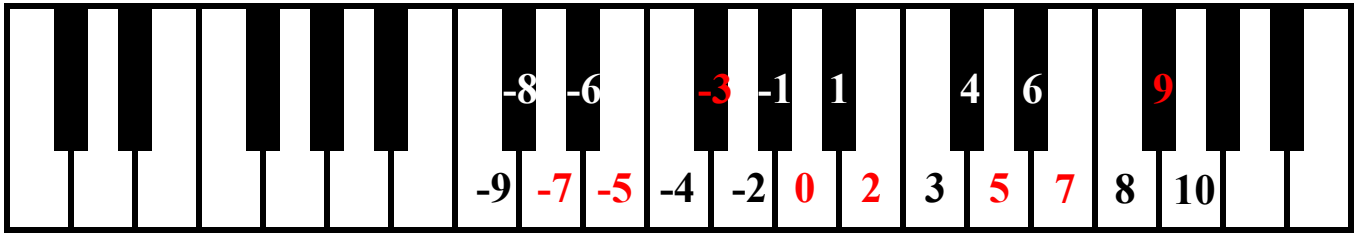
- En otras palabras, **[mtof]** realiza la siguiente operación:

$$f = f_0 \cdot 2^{(n-69)/12},$$



# Escalas

- Una escala puede verse como un subconjunto de notas que típicamente se repiten en cada octava.




Escala Pentatónica

- Algunas escalas comunes son: cromática, mayor, menor, pentatónica.

# Cuantización a una escala

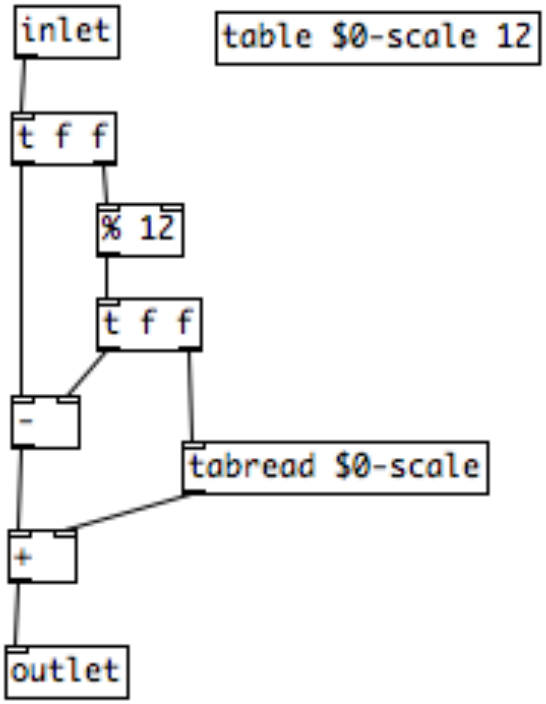
- La cuantización a una escala se realiza tomando como entrada una nota MIDI (un entero entre 0 y 127), y devolviendo la nota de la escala más cercana a la nota de entrada.

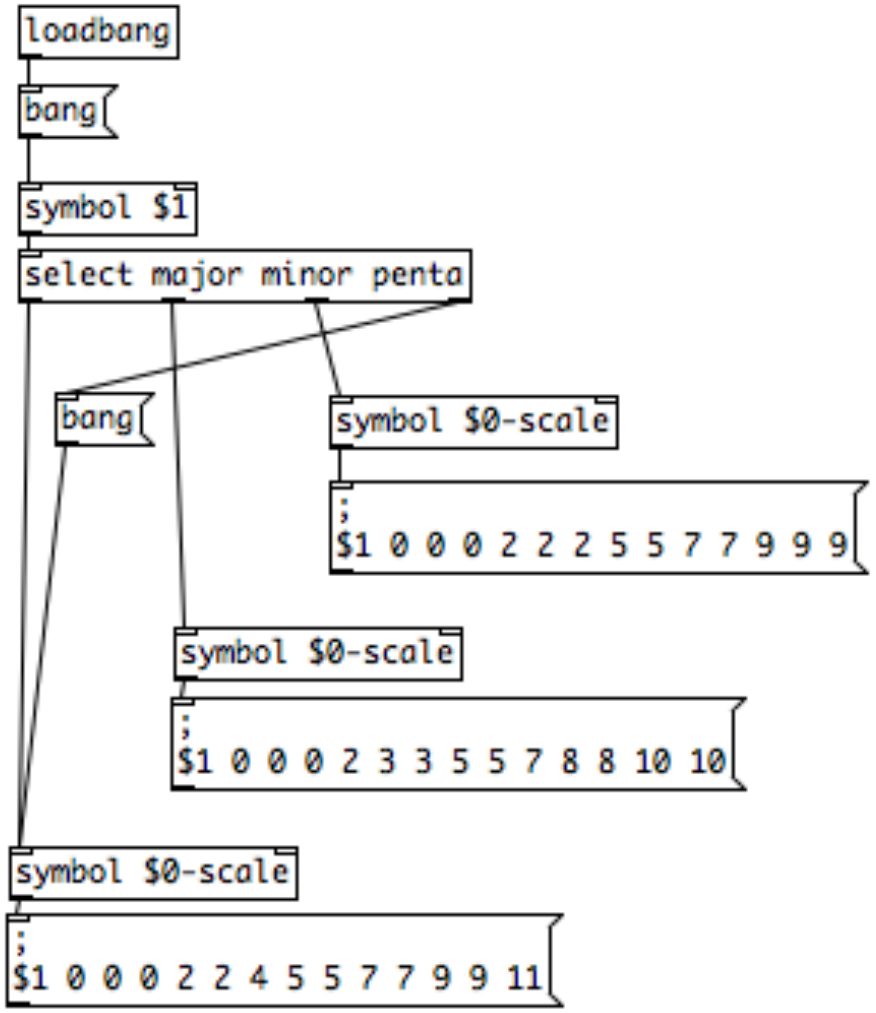
- Para lograr esto, podemos usar un arreglo que determine cómo se cuantiza cada nota en una sola octava.

- Por ejemplo, para cuantizar a una escala mayor, podemos usar la siguiente tabla (notar que no es la única):

| **In** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Out** | 0 | 0 | 2 | 2 | 4 | 5 | 5 | 7 | 7 | 9 | 9 | 11 |

# Cuantización a una escala

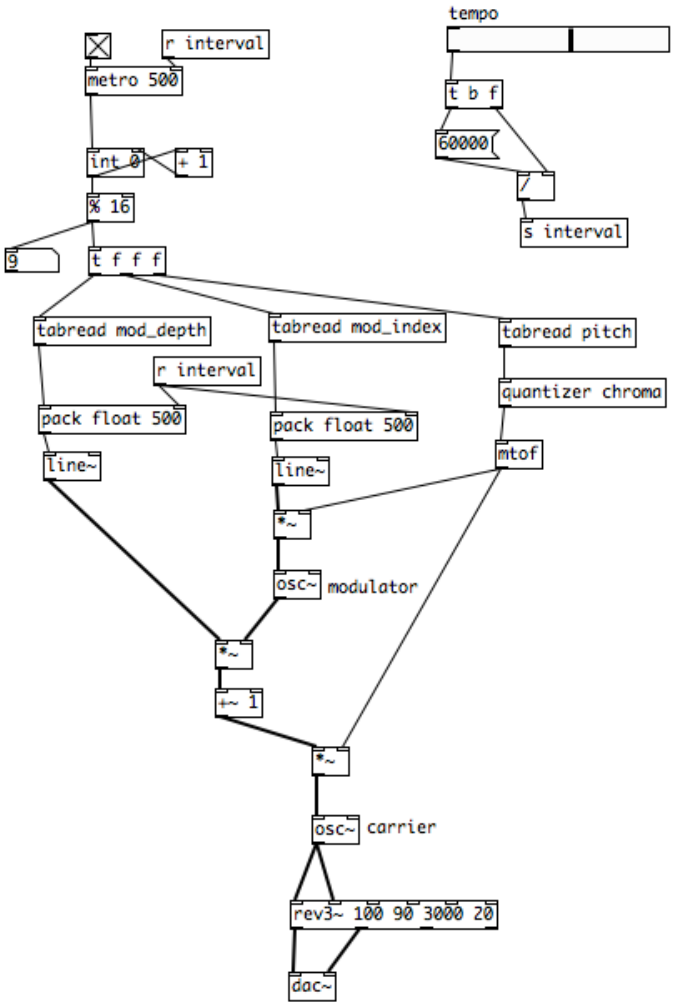


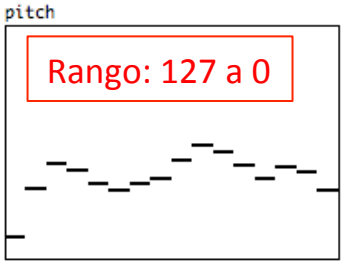



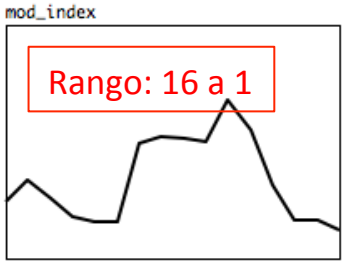
# Envolventes complejas

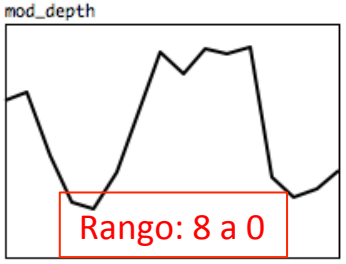
- Interpolando la salida de un “secuenciador” (por ejemplo, mediante **[line~]**) es posible crear formas complejas de envolventes.

- También puede utilizarse **[tabread4~]** para obtener la salida interpolada de un arreglo.



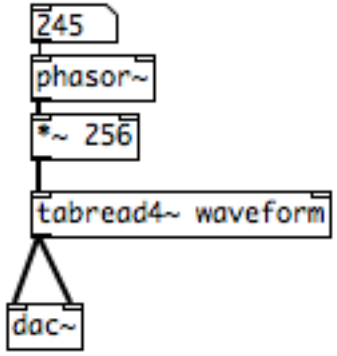



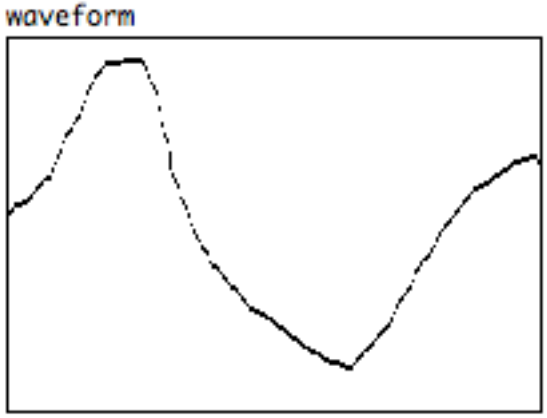



# Tablas de ondas

- Es también posible utilizar un arreglo para representar una forma de onda arbitraria.

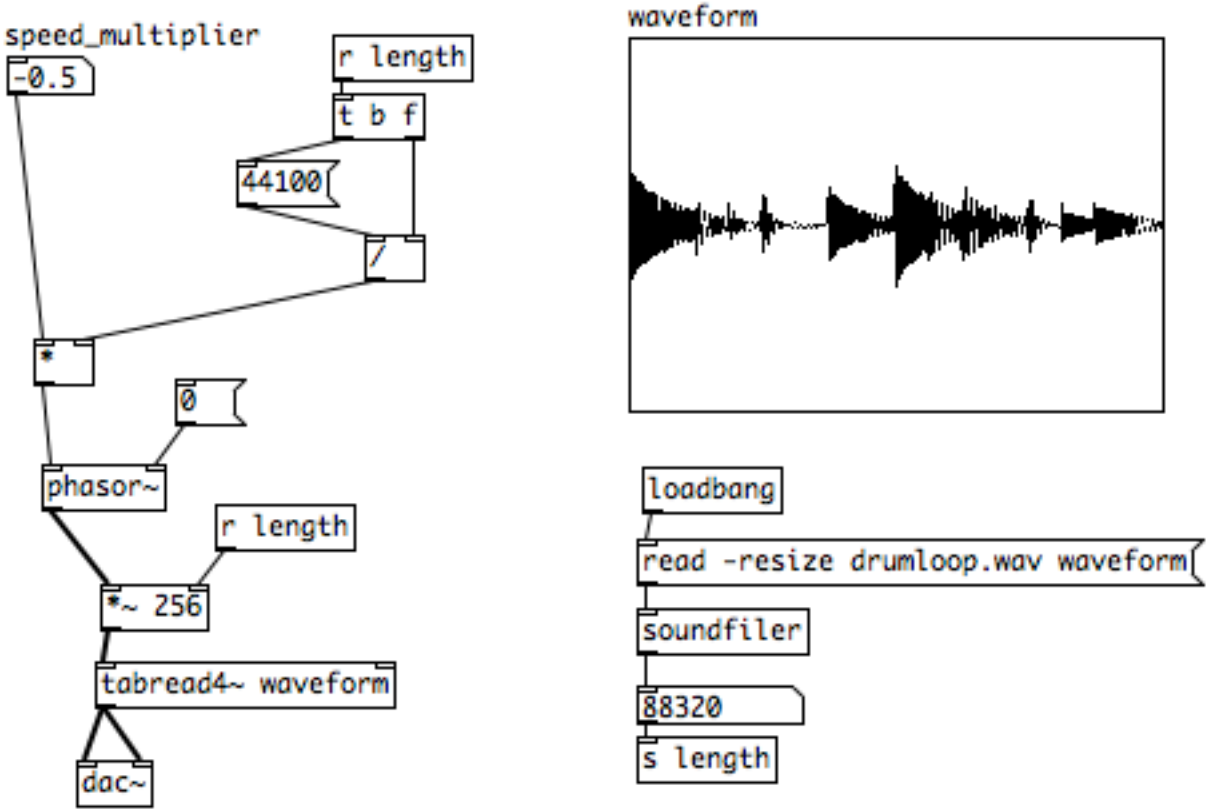
- En este caso, la lectura del arreglo debe realizarse a la tasa de muestreo del audio, para lo cual se utiliza el objeto **[tabread~]**, o bien, la versión con interpolación **[tabread4~]**.



# Lectura de archivos de audio

- También es posible leer un archivo de audio, mediante **[soundfiler]**, y almacenarlo en un arreglo para reproducirlo posteriormente.



# Algunos objetos básicos (arreglos)

- **[table]** – Define un sub-parche con un arreglo
- **[tabread]** – Lectura de un arreglo
- **[tabwrite]** – Escritura de un arreglo
- **[tabread~]** – Lectura contínua de un arreglo sin interpolación
- **[tabread4~]** – Lectura contínua e interpolada de un arreglo
- **[tabwrite~]** – Escritura contínua en un arreglo
- **[tabosc4~]** – Lectura contínua cíclica de un arreglo
- **[soundfiler]** – Lecture y escritura de archivos de audio

Sesión 4

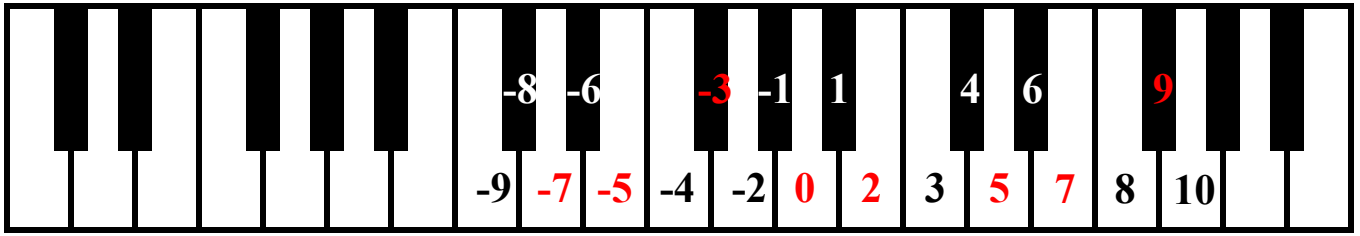
# COMPOSICIÓN ALGORÍTMICA I: MELODÍA

# Música algorítmica

- A muy grandes rasgos, la música algorítmica se fundamenta en tres etapas:
  1. Generar secuencias de números "interesantes" (equilibrio entre repetitividad y sorpresa)
  2. Transformar las secuencias de números en secuencias de notas musicalmente relacionadas (escalas, acordes, cánones, etc)
  3. Agregar una o más "capas superiores" que controlen (posiblemente a través de otras transformaciones) las dos etapas anteriores

# Escalas

- Una escala puede verse como un subconjunto de notas que típicamente se repiten en cada octava.

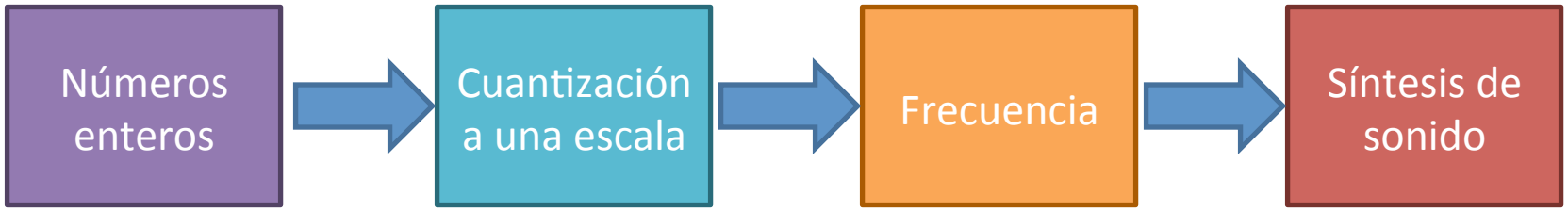



Escala Pentatónica

- Algunas escalas comunes son: cromática, mayor, menor, pentatónica.

# Ejemplo de transformación (etapa 2): Cuantización a una escala

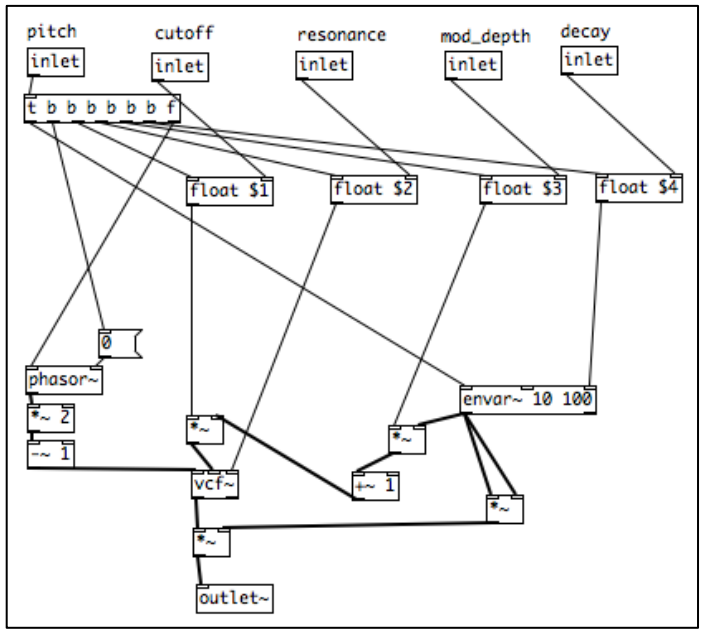
- Nuestro objeto cuantizador es un buen ejemplo de transformación.



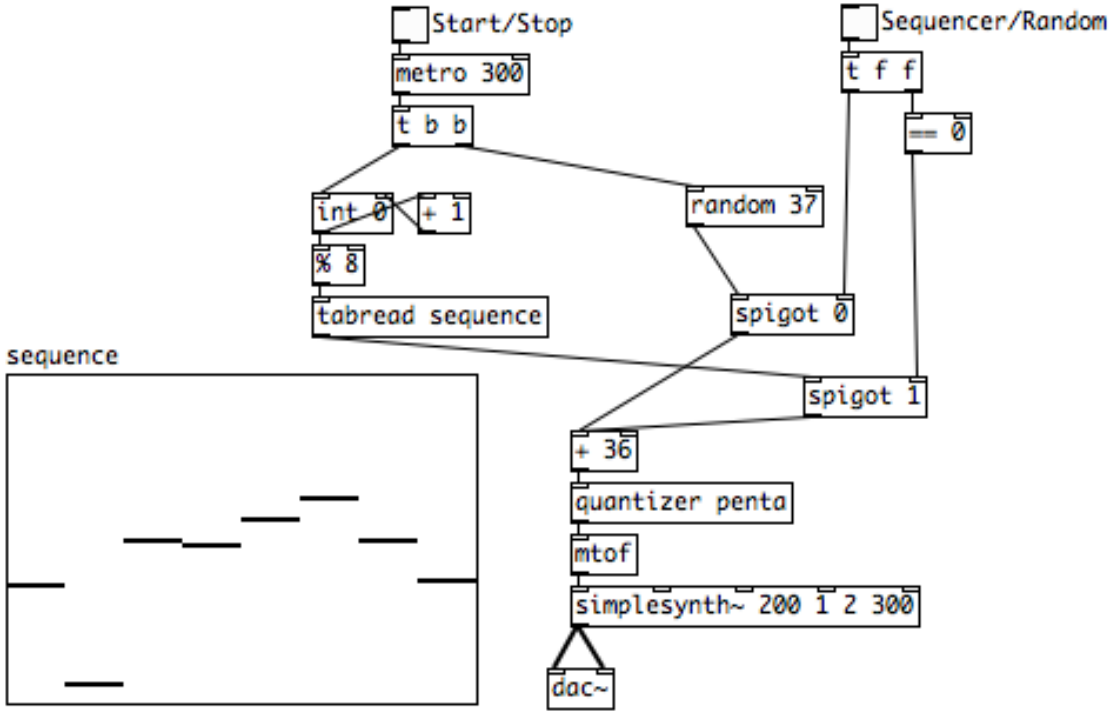

- Pero de dónde se pueden obtener secuencias de números “interesantes”?

# Predicción vs sorpresa

- Las secuencias “interesantes” son aquellas que muestran un balance entre lo predecible (e.g., un secuenciador) y lo aleatorio.

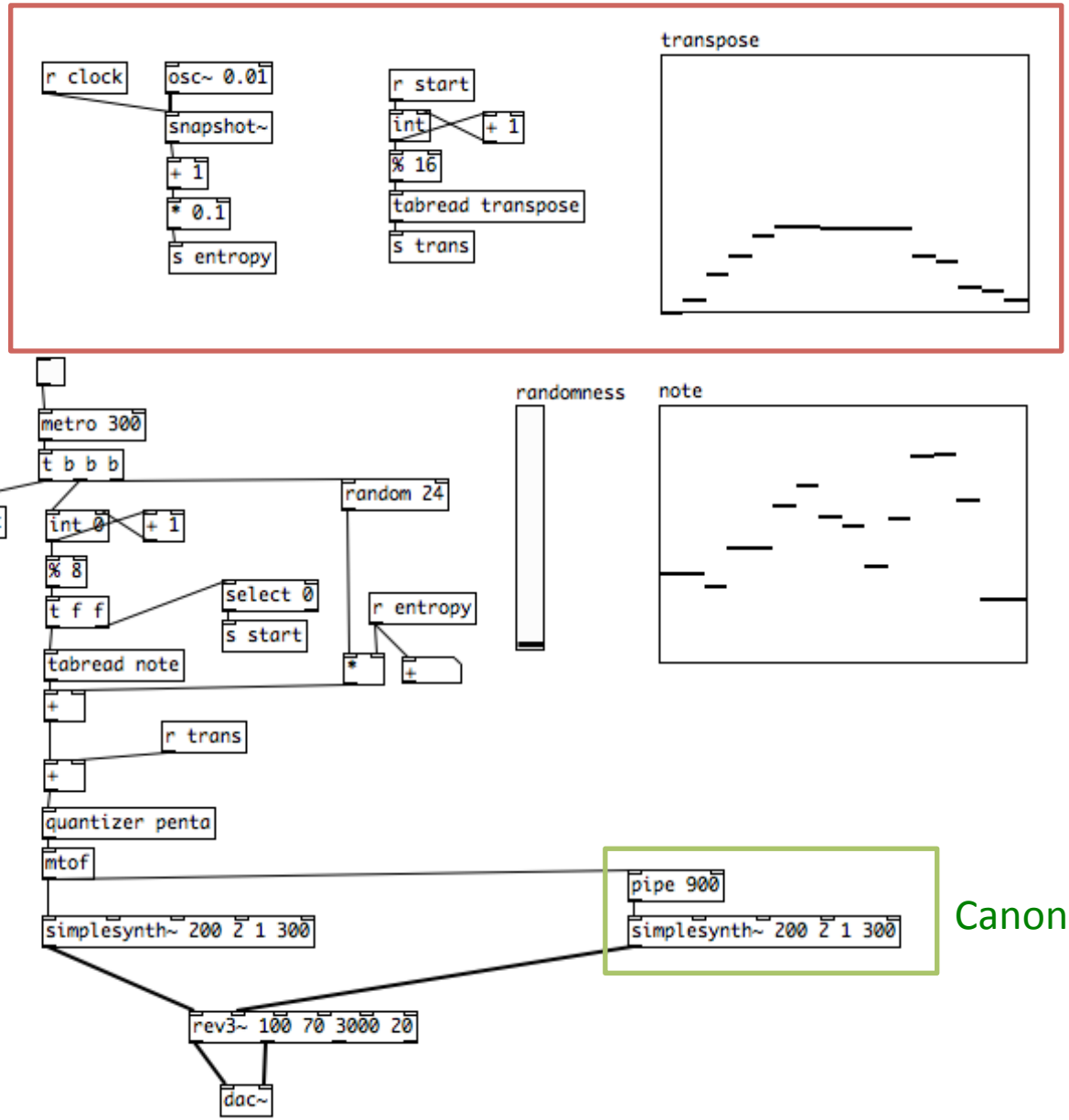



simplesynth~.pd



# Predicción vs. sorpresa

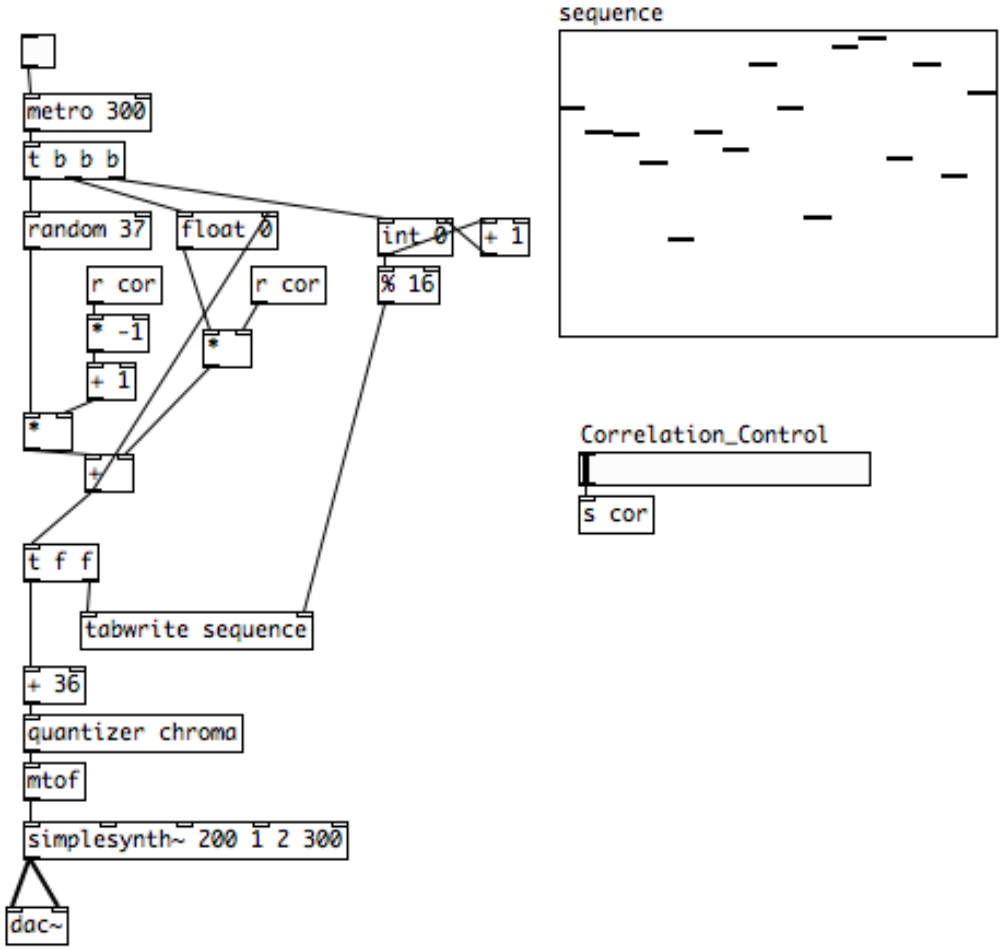
Capa de nivel superior




Canonizador

# Secuencias aleatorias con correlación

- Las secuencias aleatorias de ruido blanco (i.i.d.) no siempre producen buenos resultados.
- En algunos casos, es preferible utilizar ruido con color (correlación).
- Una forma de introducir correlación es tomar un promedio pesado entre una nueva muestra de ruido, y la muestra anterior (filtro pasa bajas de un polo).



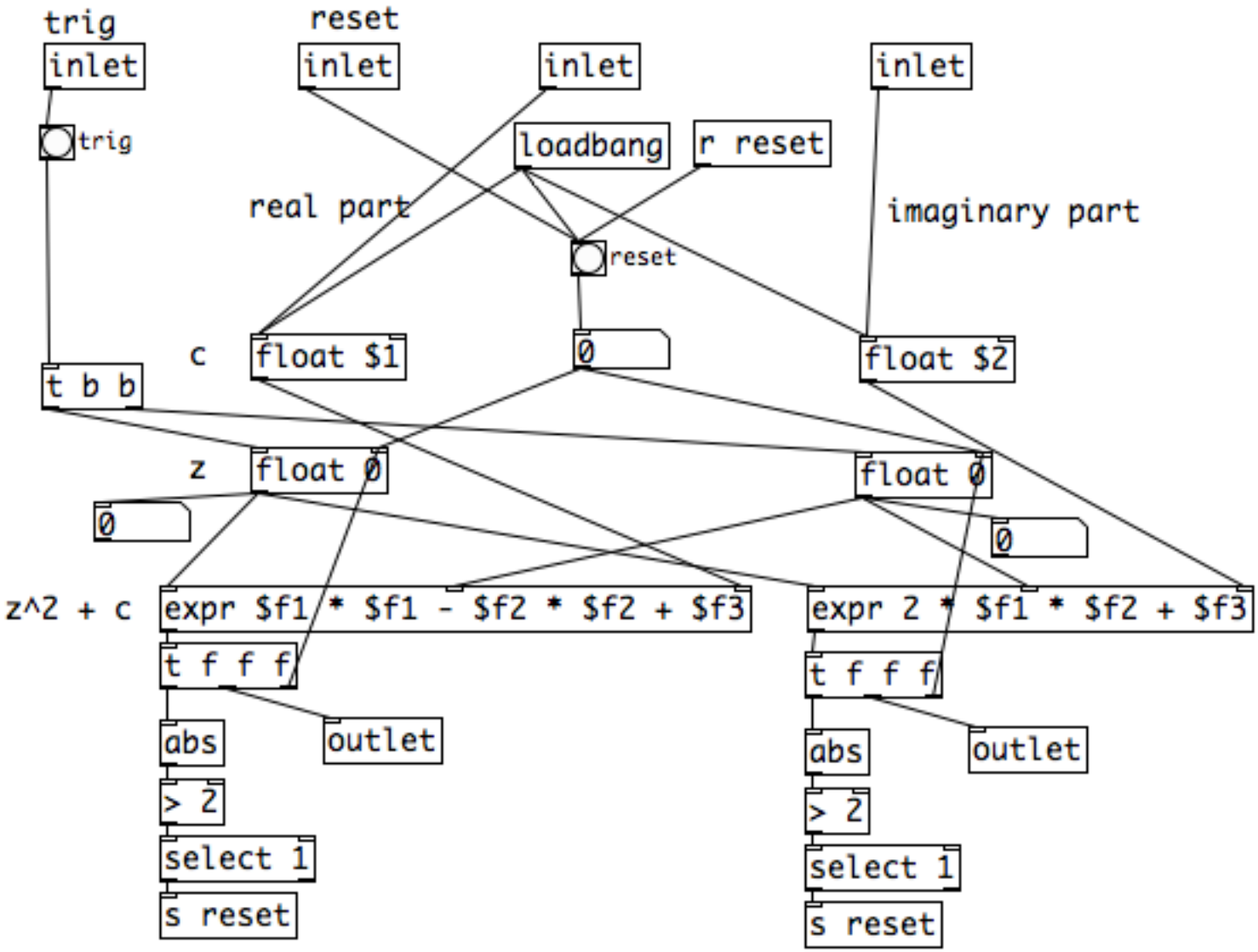
# Secuencias fractales

- Los fractales son objetos matemáticos que muestran un cierto grado de auto-similaridad, la cual puede dar lugar a secuencias interesantes.

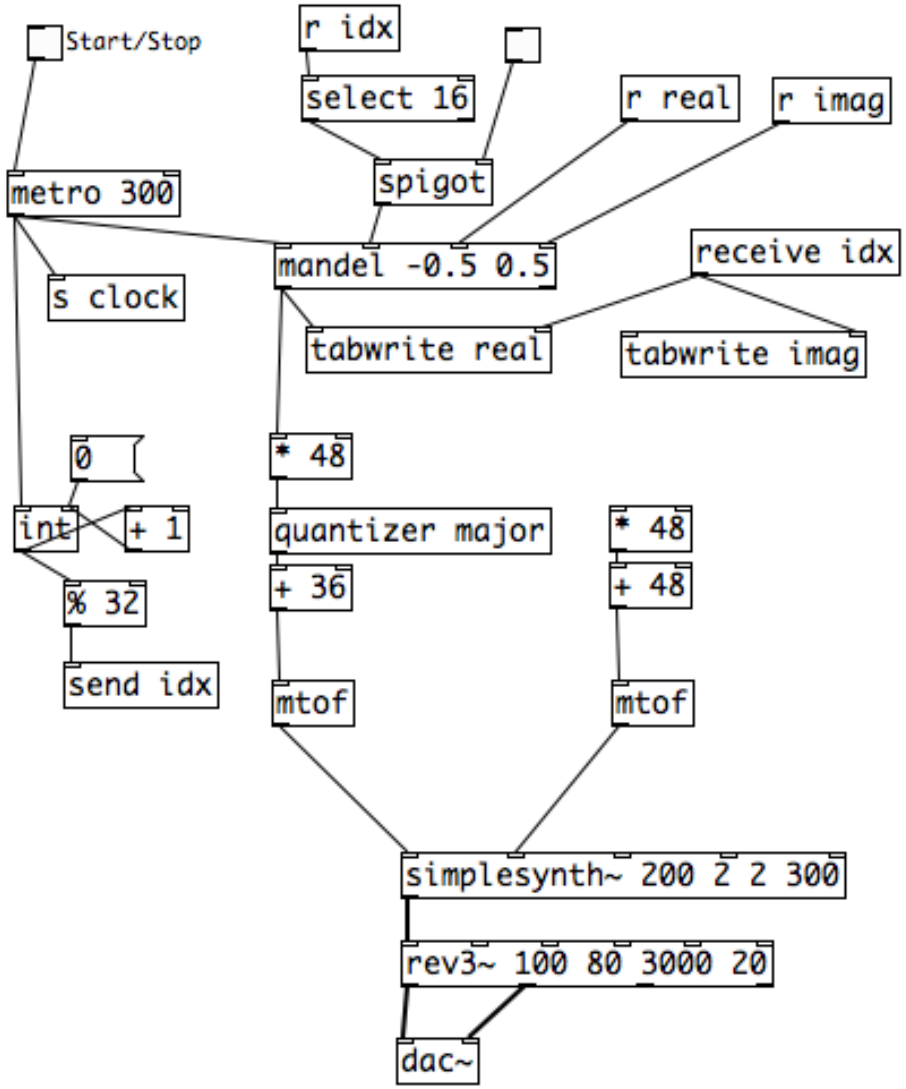
- Muchos fractales se obtienen a partir de fórmulas recursivas. Por ejemplo, el conjunto de Mandelbrot está formado por aquellos complejos ***c*** tales que la órbita de la fórmula recursiva $z=z^2+c$, iniciando con $z=0$, es acotada.

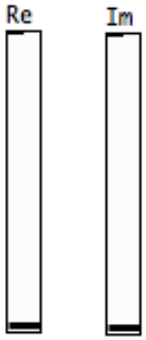
- Una forma simple de generar secuencias asociadas con fractales es generando la órbita para un punto.

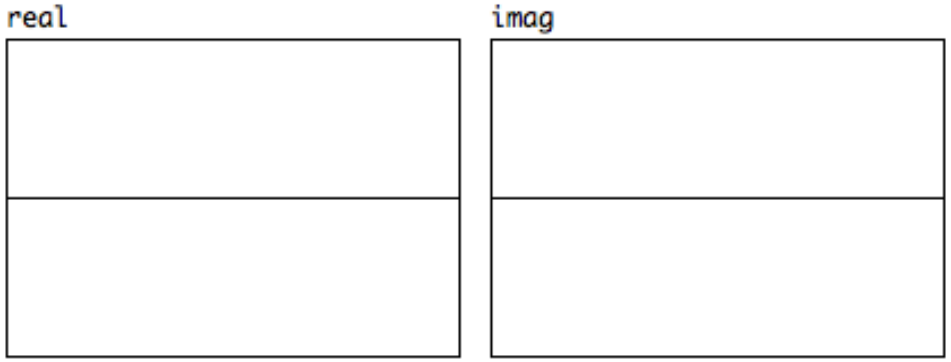
# Ejemplo de órbitas



# Ejemplo de órbitas



# Secuencias Morse-Thue

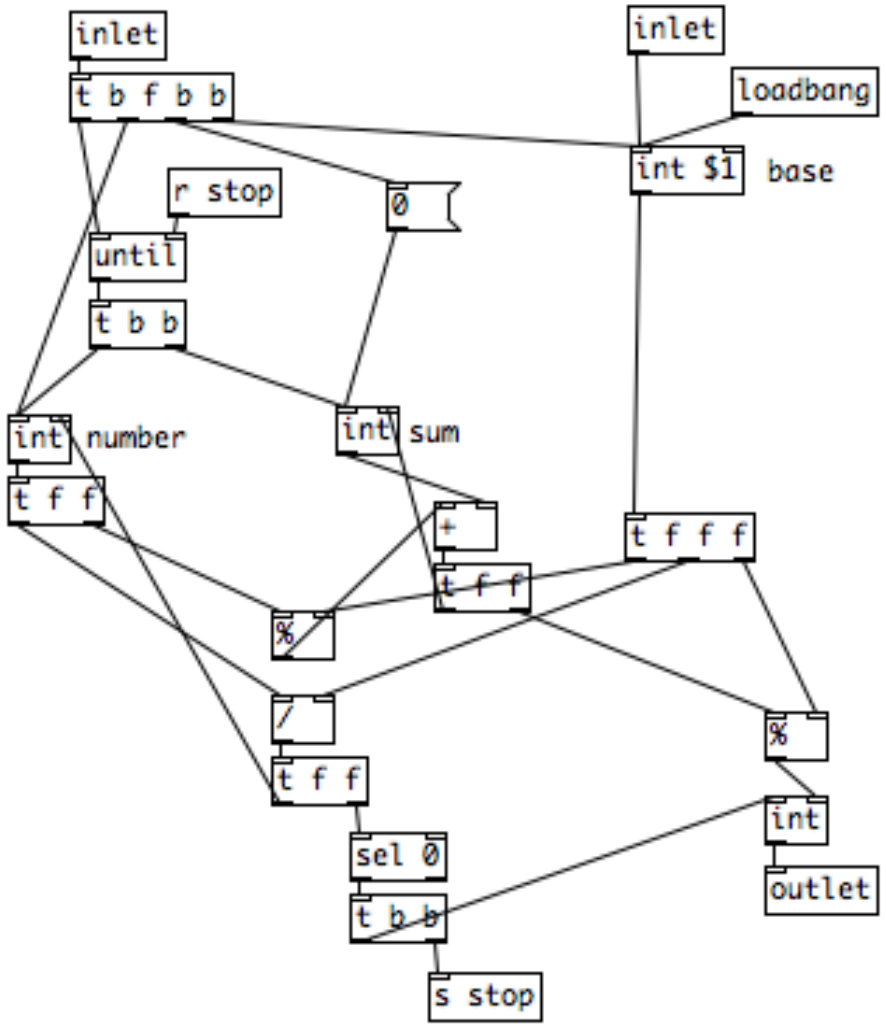
- La secuencia Morse-Thue es una secuencia binaria infinita que se obtiene calculando la paridad (la suma módulo-2 de los dígitos en la representación binaria) de los números naturales, e.g., 011010011001011O…

- La idea anterior se puede generalizar a una transformación donde se calcule la ***n***-aridad de cada número ***x*** en una cierta secuencia. Esta se puede calcular mediante el siguiente pseudo-algoritmo:

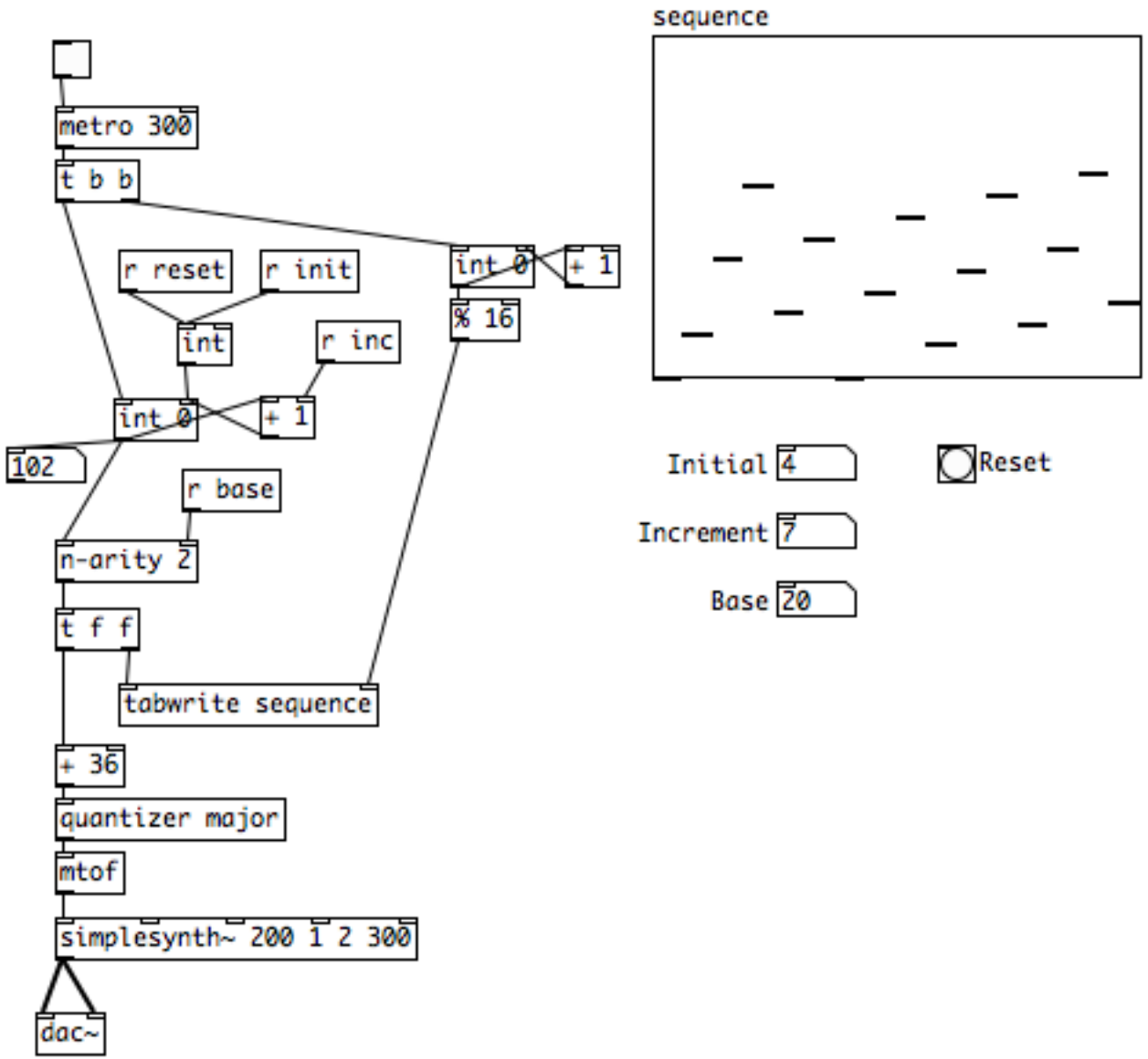
```
sum = 0;
while (x > 0) {
      sum = sum + (x % base);
      x = (int)(x / base);
}
return sum % base;
```

# Cálculo de la ***n***-aridad en Pd

n-arity.pd



# Ejemplo con secuencias Morse-Thue



Sesión 5

# COMPOSICIÓN ALGORÍTMICA II: RITMO

# Nociones sobre métrica y ritmo

- El ritmo determina el tiempo en el que ocurren los eventos y su duración.

- Usualmente, el ritmo consiste en un patrón mas o menos periódico, que puede representarse como una secuencia discreta de eventos equi-espaciados en el tiempo (e.g., un metrónomo).

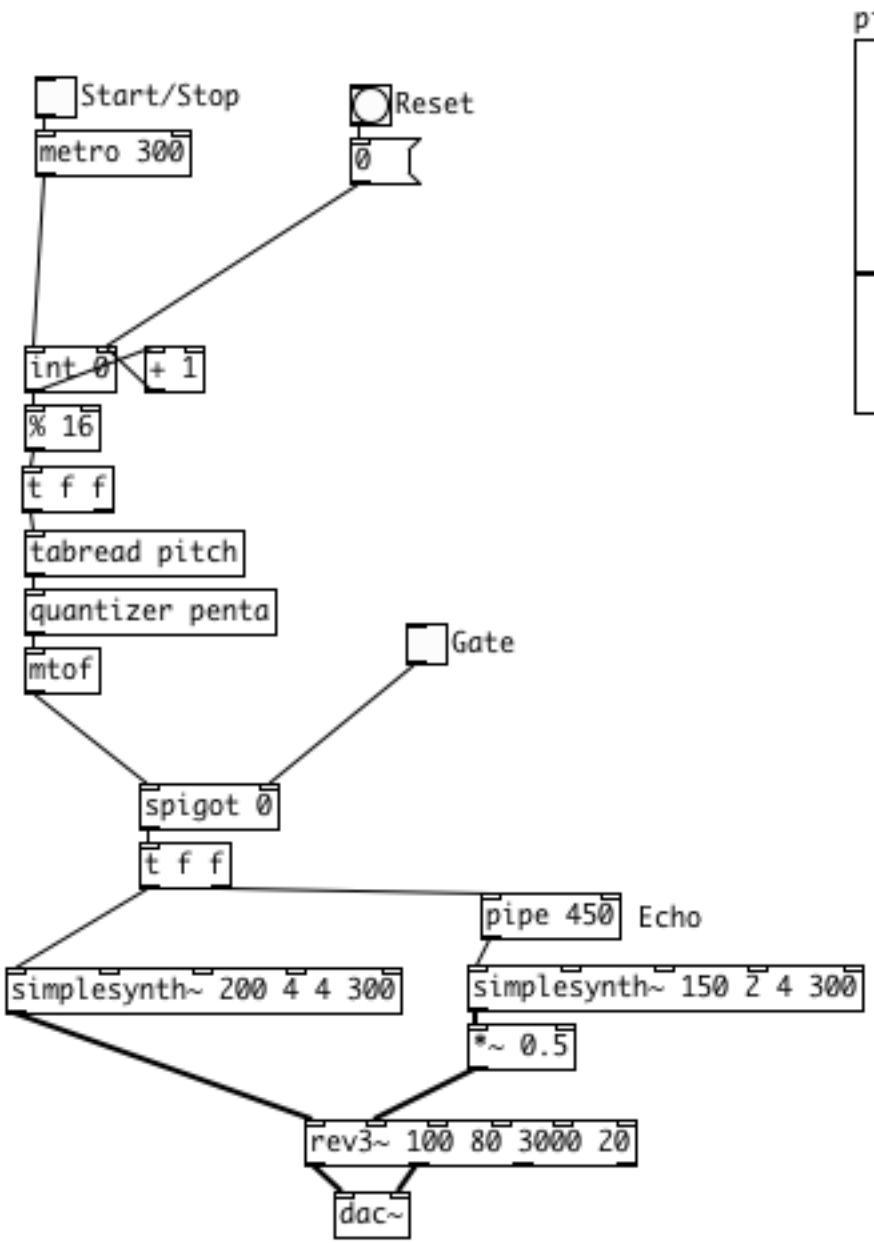
- La longitud y el número de eventos determinan la métrica de un patrón rítmico.

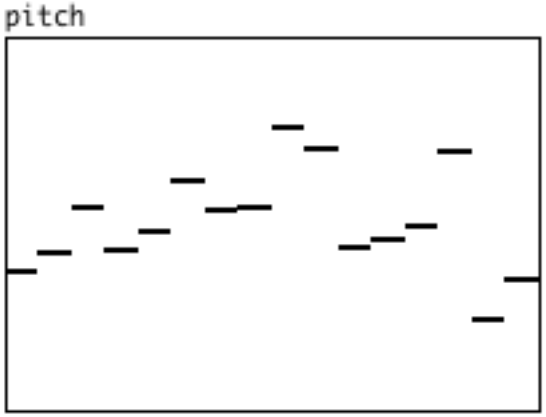
# Objeto **[spigot]**

- El objeto **[spigot]** cuenta con dos entradas y una salida. Los mensajes que recibe en la entrada activa se envían a la salida solamente cuando el valor recibido en la segunda entrada es distinto de cero.

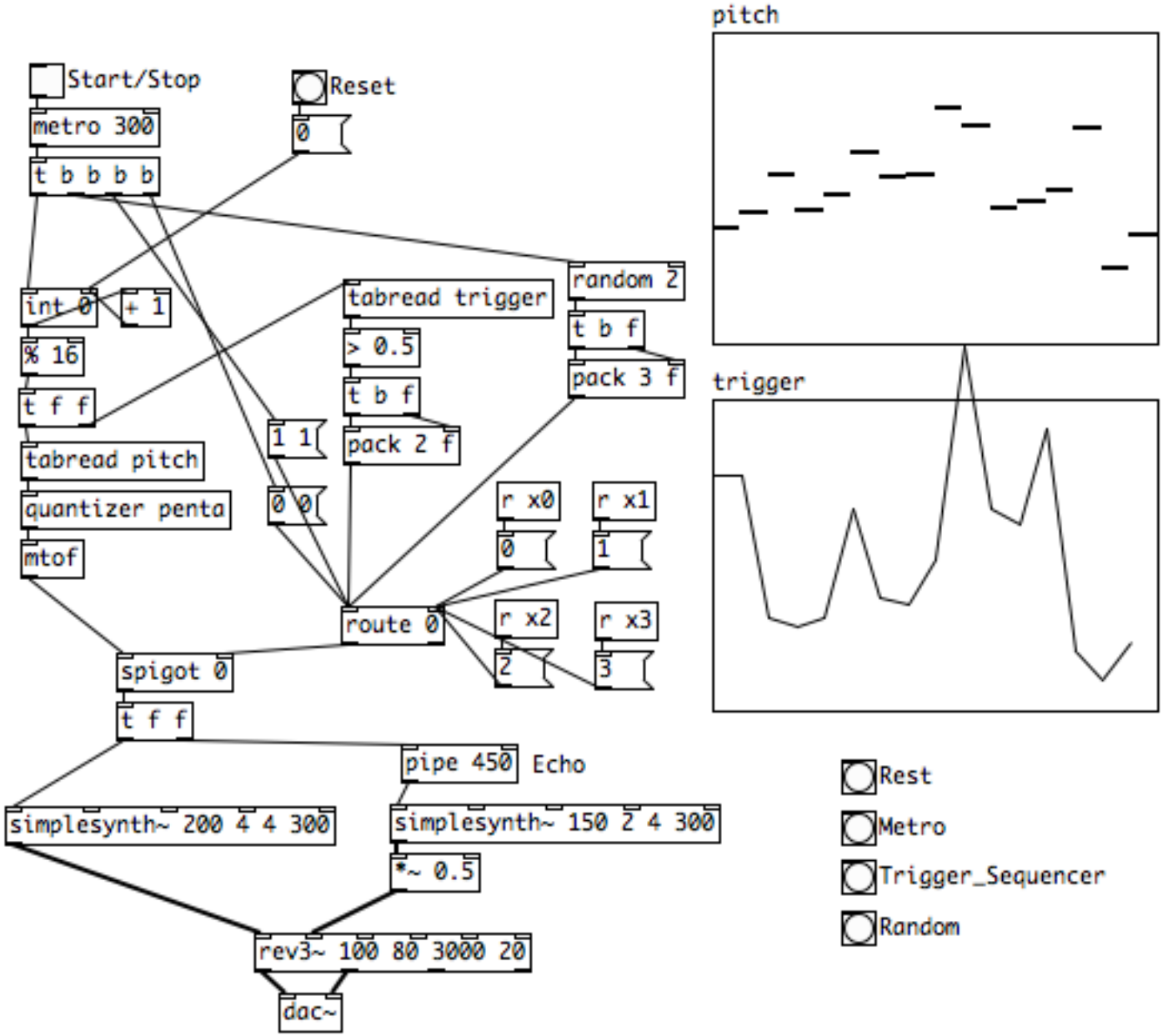
- Usando este objeto, podemos controlar el paso de mensajes desde un metrónomo hacia los generadores de notas, creando así distintos patrones rítmicos.

# Ejemplo con **[spigot]**

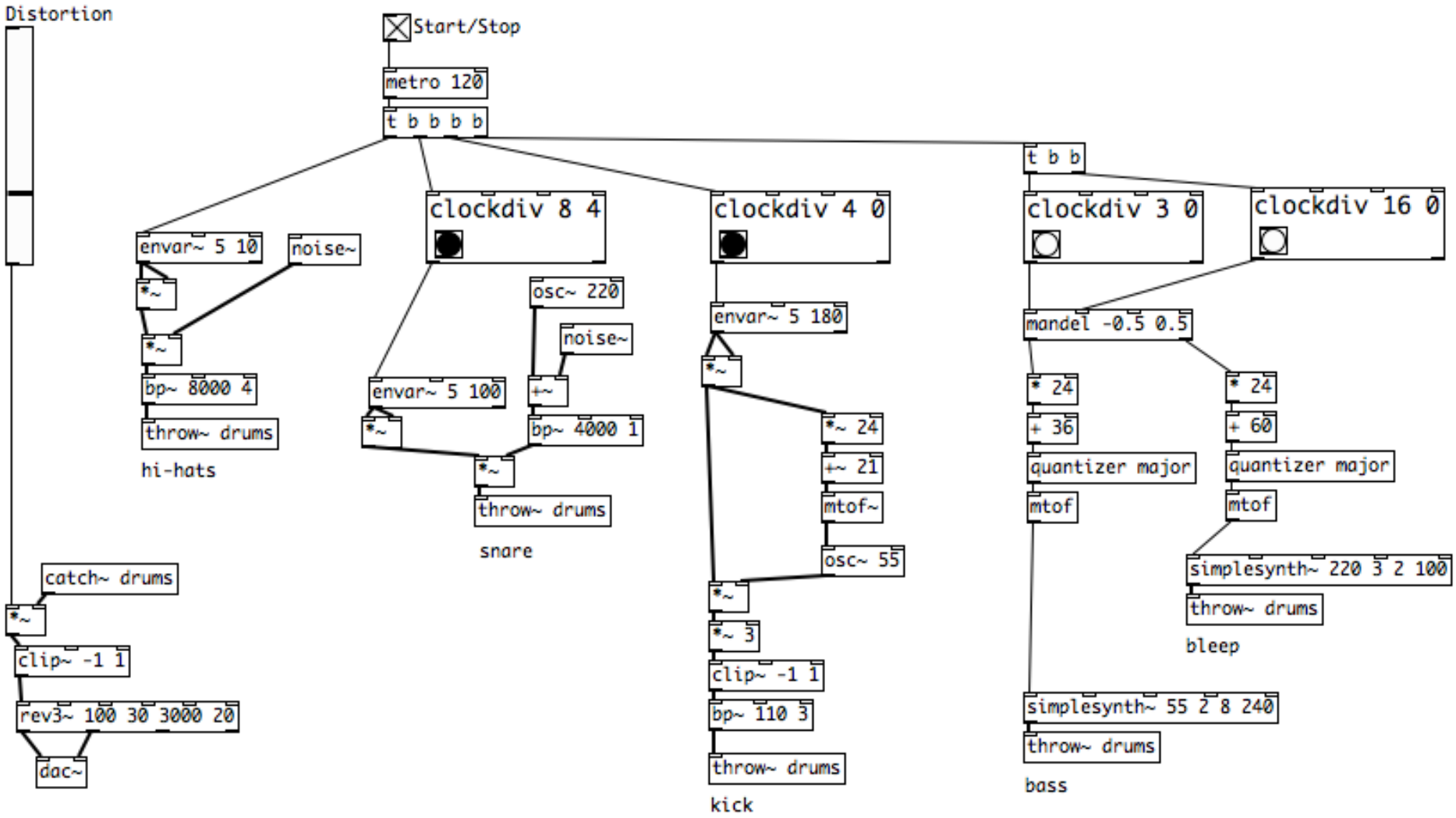


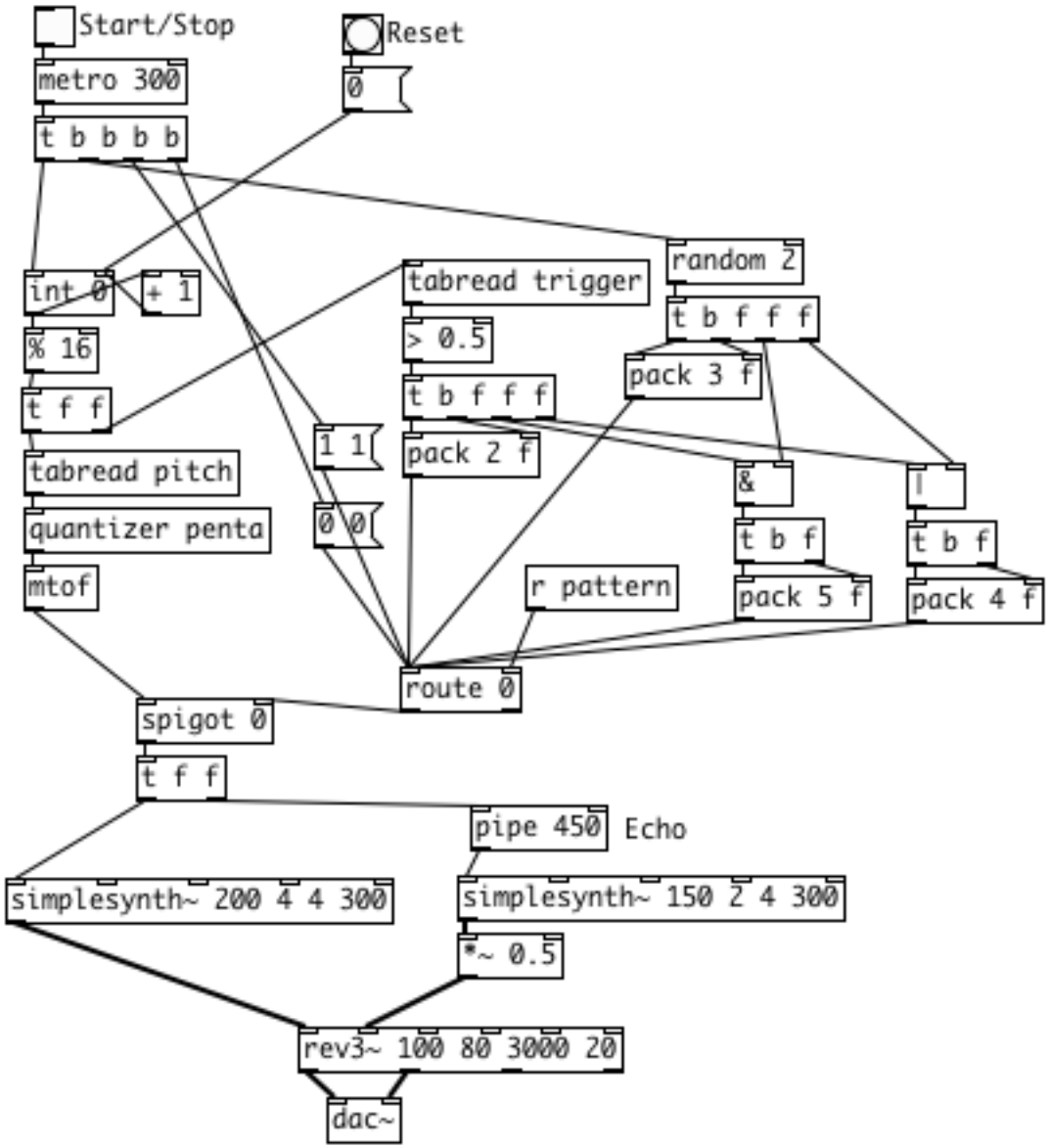
# Ejemplo: Patrones aleatorios

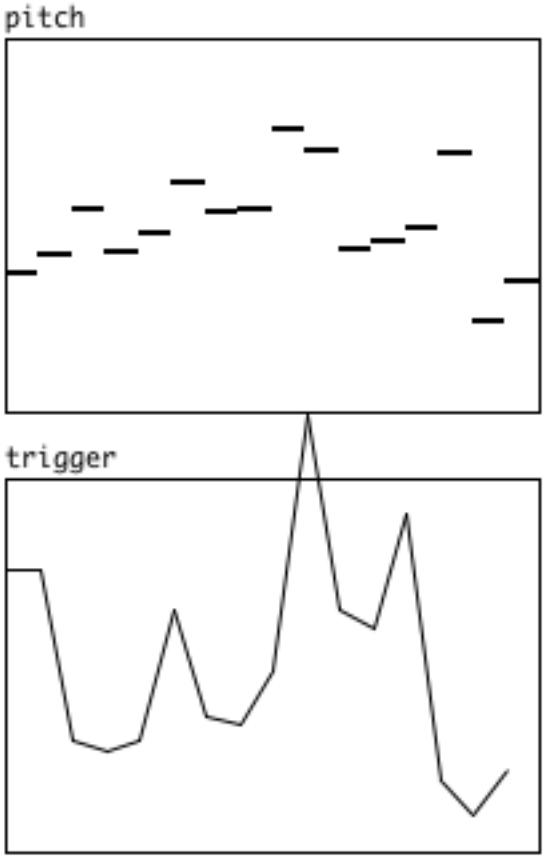


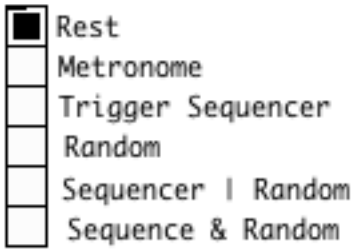
# Divisores de reloj



# Combinando patrones mediante operadores lógicos

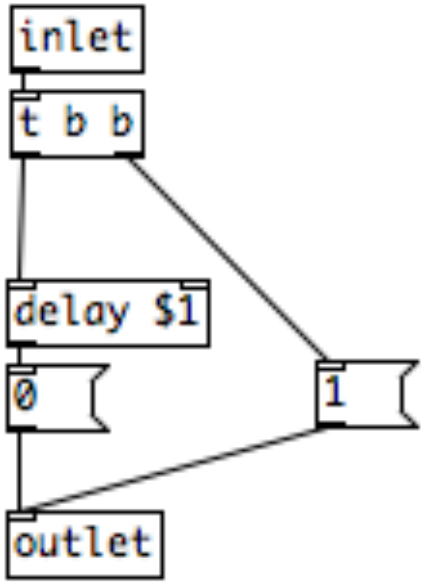


# Operaciones lógicas con **[bang(**

- No podemos operar directamente con los mensajes **[bang(** que envía **[metro]** o **[clockdiv]**, por lo que usaremos el objeto **[pulse]** para generar datos numéricos (binarios) con los que se pueda operar.




pulse.pd

# Autómatas celulares

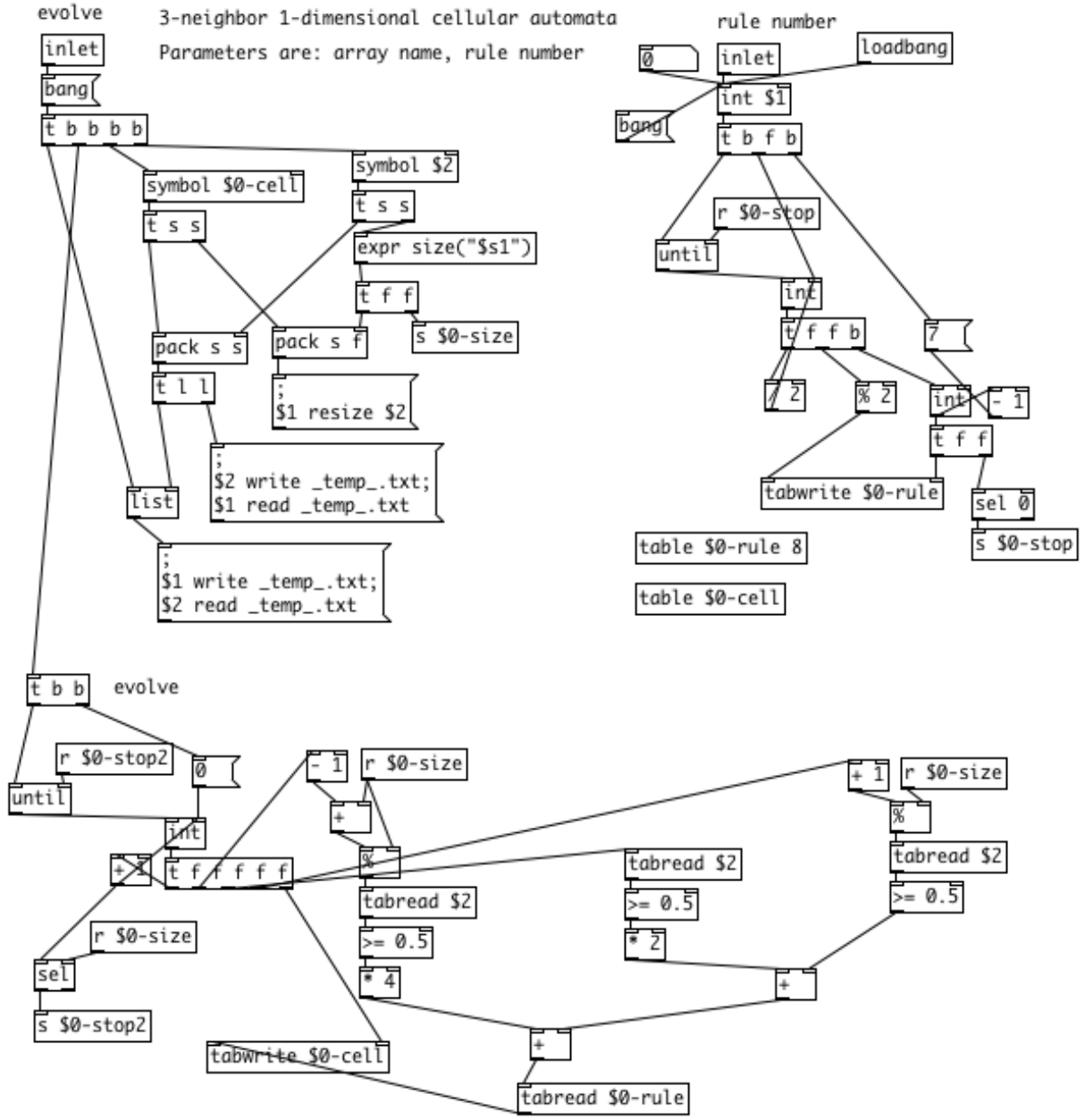
- Un autómata celular consiste en un arreglo de celdas, cada una de las cuales se encuentra en un estado (de un conjunto finito de estados).

- Un conjunto de reglas permite evolucionar el estado de cada celda, dependiendo del estado actual de esa celda y sus vecinas.

- Los autómatas celulares mas simples son unidimensionales, tienen un conjunto de dos estados (0 ó 1), y las reglas consideran únicamente las dos celdas adyacentes como vecinas (vecindades de tamaño 3).

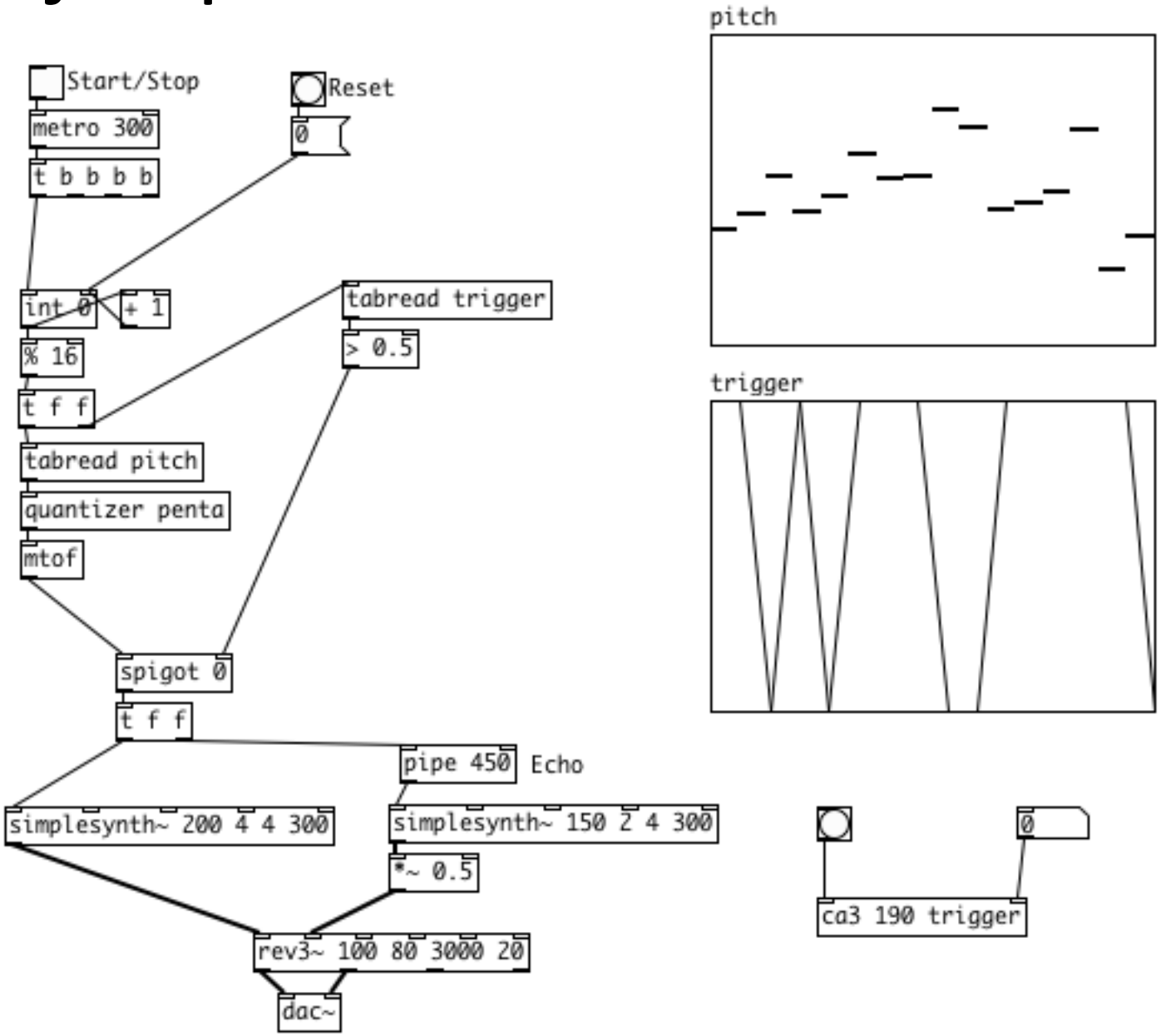
- En este caso, una vecindad puede tener una de 8 combinaciones de estados, por lo que existen $2^8 = 256$ maneras de definir la regla de evolución.

# Autómata celular




ca3.pd

# Ejemplo con autómata celular



# Acentuación

- Los patrones rítmicos también pueden utilizarse para modular parámetros de síntesis, de manera que algunas notas resulten “acentuadas”.