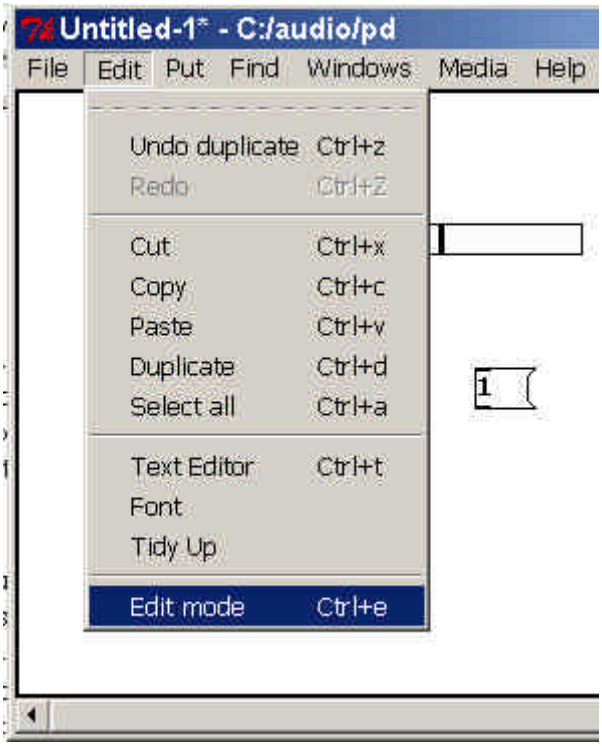
Índice

II. Principios básicos de funcionamiento

1. Edición, ejecución, ayudas, etc.

En PD existen dos estados o dos modos básicos de funcionamiento:
Edición y **ejecución**



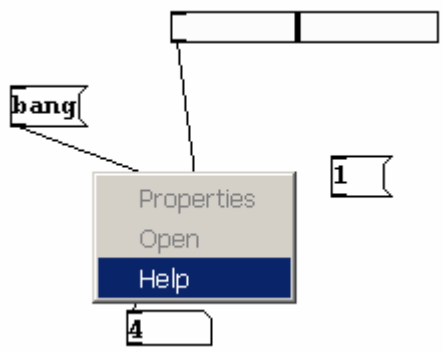
- En **edición** se pueden añadir nuevos objetos, conectarlos entre si, arrastrarlos y desplazarlos… En este modo, el cursor aparece como una mano.
- En **ejecución**, sólo se puede interactuar con los objetos gráficos (potenciómetros o sliders, casillas numéricas, botones, etc.). En este modo, el cursor aparece como una flecha.

Cada ventana está en un modo determinado, pero si tenemos varias ventanas abiertas, algunas pueden estar en uno y otras en otro.

Para pasar de un modo a otro se puede seleccionar el menú de la ventana en cuestión tal como se muestra en la imagen. También se puede cambiar, pulsando *CTRL+E*, que cambiará el modo de la ventana activa o seleccionada.

NB. En edición, PD también puede estar sonando o funcionando.

1.1. Help - Ayuda



Independientemente de cual sea el modo activo (ejecución o edición), al clicar sobre un objeto con el botón derecho, aparecen normalmente varias opciones, entre ellas la opción Help. Al seleccionarla, se abrirá una nueva ventana PD con un programa que orientará sobre el uso del objeto en cuestión.

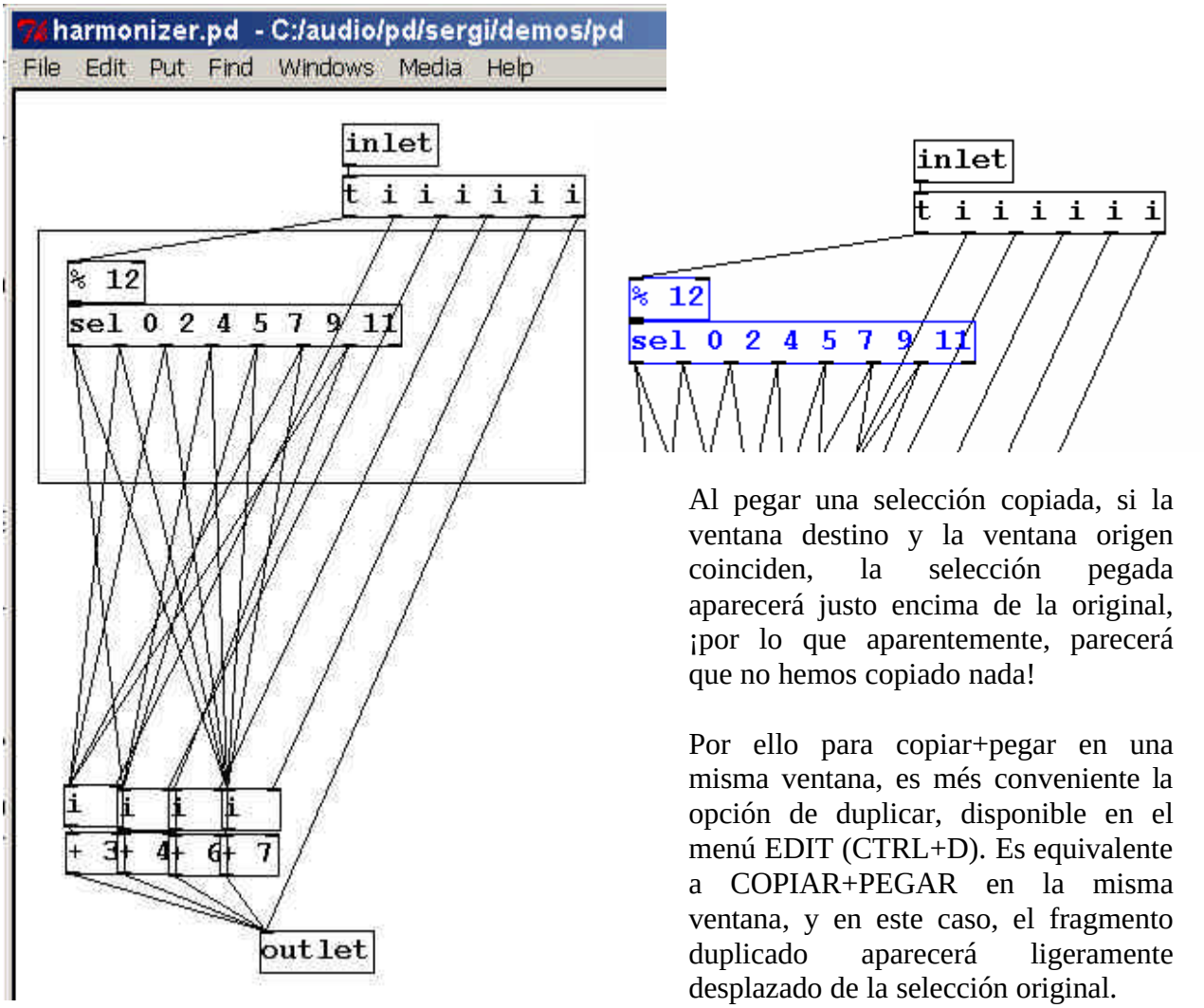
Esta nueva ventana contiene un programa (o patch) PD como cualquier otro, que es por tanto, editable, copiable, modificable, etc.

Si modificásemos un fichero de ayuda, convendría al menos salvarlo con un nombre diferente. Algo mucho más apropiado es copiar un trozo de este fichero para reaprovecharlo pegándolo en otra ventana.

NB. Al solicitar ayuda sobre el fondo de la ventana (i.e. sin estar encima de ningún objeto) obtendremos la lista total de objetos PD (cfg. Apéndice V.1 de este manual).

1.2. Copiar, Cortar, Pegar

En modo edición, es posible seleccionar una zona rectangular manteniendo el ratón pulsado (botón izquierdo). La zona seleccionada se vuelve azul y podemos copiarla o cortarla, para posteriormente pegarla, ya sea sobre la misma o sobre otra ventana.



Al pegar una selección copiada, si la ventana destino y la ventana origen coinciden, la selección pegada aparecerá justo encima de la original, ¡por lo que aparentemente, parecerá que no hemos copiado nada!

Por ello para copiar+pegar en una misma ventana, es més conveniente la opción de duplicar, disponible en el menú EDIT (CTRL+D). Es equivalente a COPIAR+PEGAR en la misma ventana, y en este caso, el fragmento duplicado aparecerá ligeramente desplazado de la selección original.

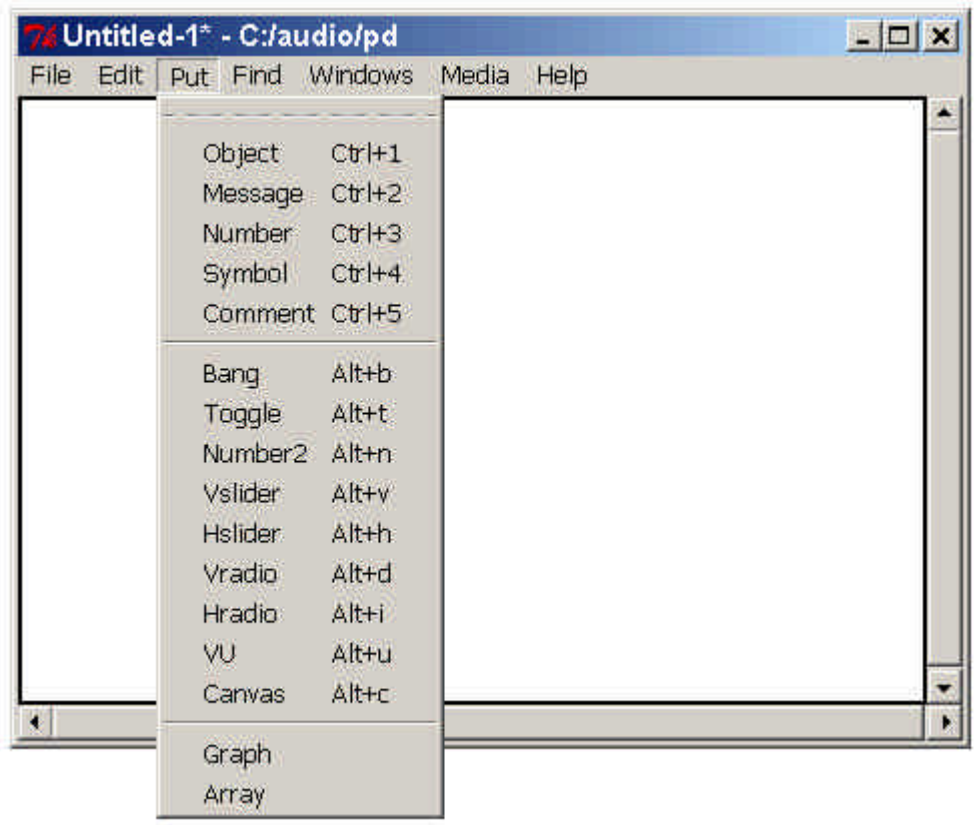
2. Objetos, mensajes y comentarios

2.1. Objetos y mensajes

- PD se compone de ventanas, denominadas *patches*.
- Cada ventana o *patch* es un programa PD (independiente o dependiente de los demás, como ya veremos).
- Independientemente de que sólo una ventana pueda estar activa en un momento dado (i.e. la ventana seleccionada), PD puede tener muchas ventanas abiertas, y todas ellas (i.e. todos los *patches*) pueden estar funcionando simultáneamente.

Si comenzamos con una ventana vacía como la de la imagen, podremos ir añadiendo elementos de varios tipos, mediante el menú, o mediante diversas combinaciones de las teclas CTRL y ALT. Cada una de estas opciones corresponde a un tipo especial de elemento. De momento mostraremos cuatro de ellos:

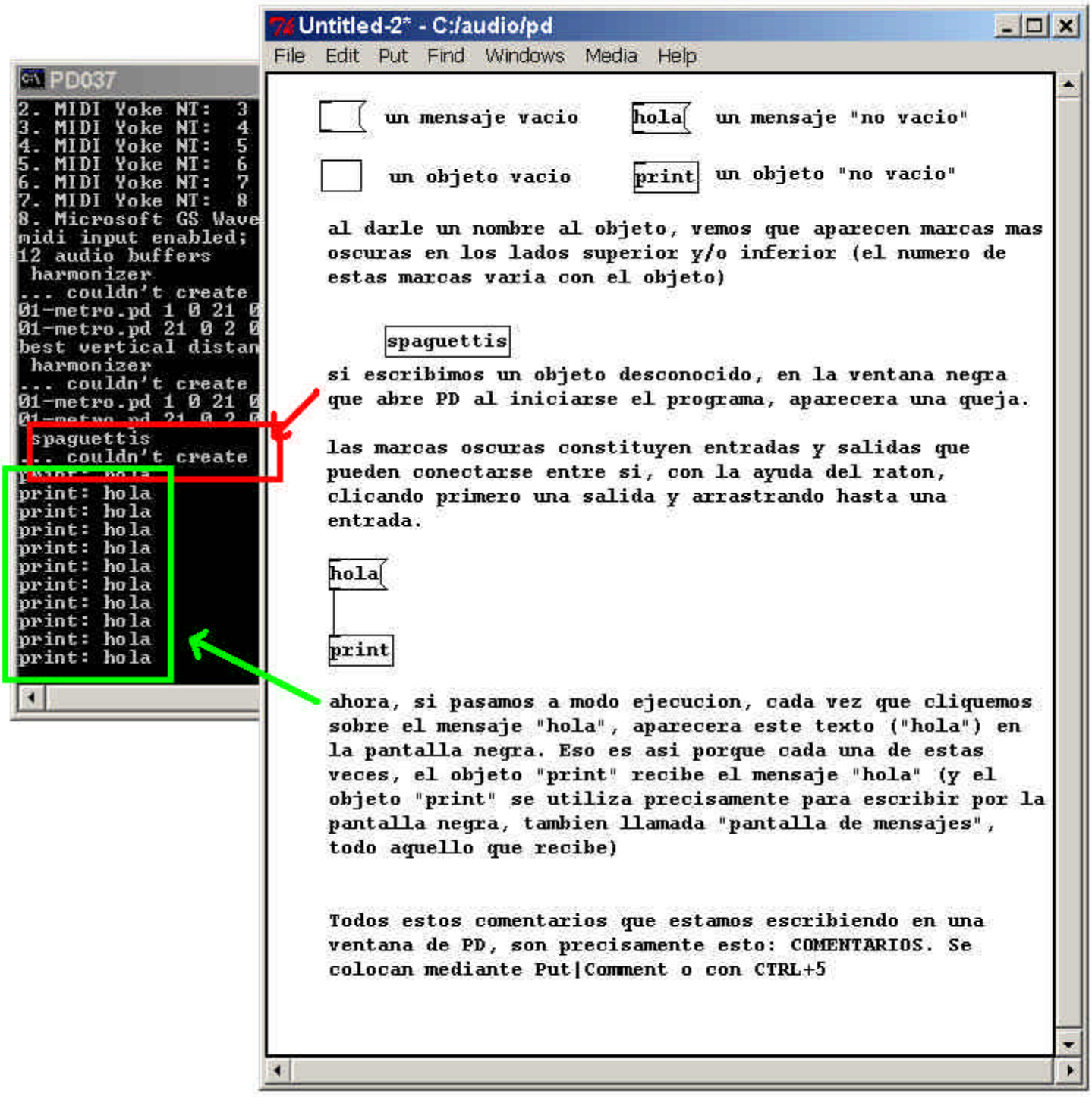
Objetos, mensajes, números y comentarios



- El elemento “objeto” tiene una forma rectangular.
- El elemento “mensaje” tiene también forma rectángulo pero con unas pequeñas muescas en las esquinas derechas.
- En ambo tipos de cajitas (objetos & mensajes) se pueden escribir nombres (o breves textos).

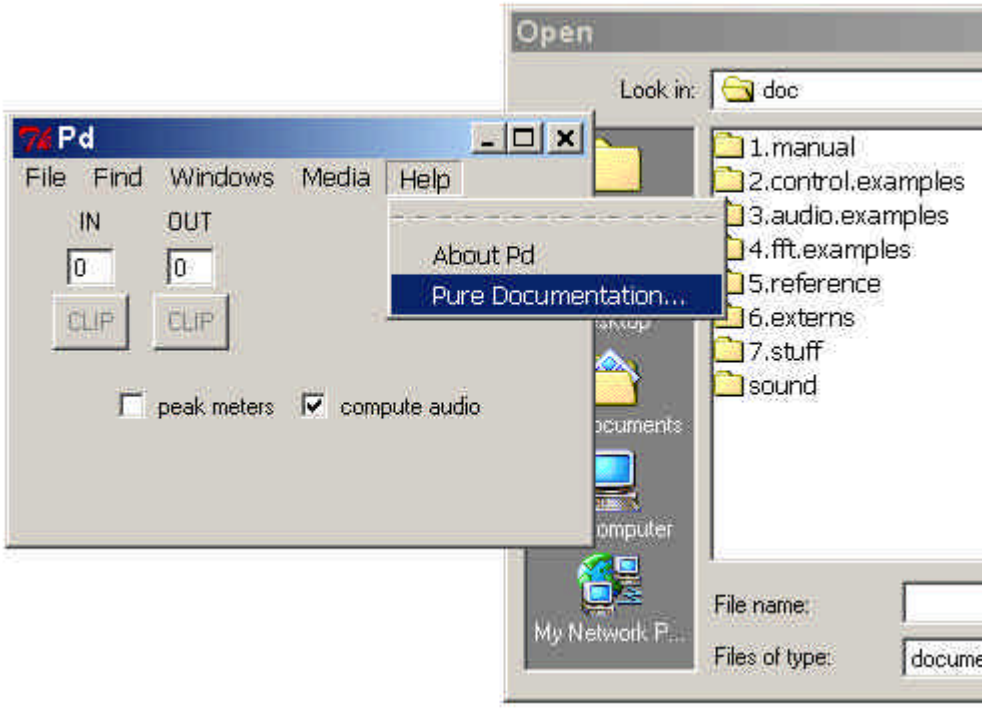
- Un elemento “mensaje” admite en principio cualquier texto.
- Un elemento “objeto”, solo admite palabras “comprensibles” para PD. Estas palabras constituyen el vocabulario de PD, y son los equivalente a lo que en cualquier otro lenguaje de programación serían las instrucciones básicas (e.g. IF, <, +…) y las funciones.

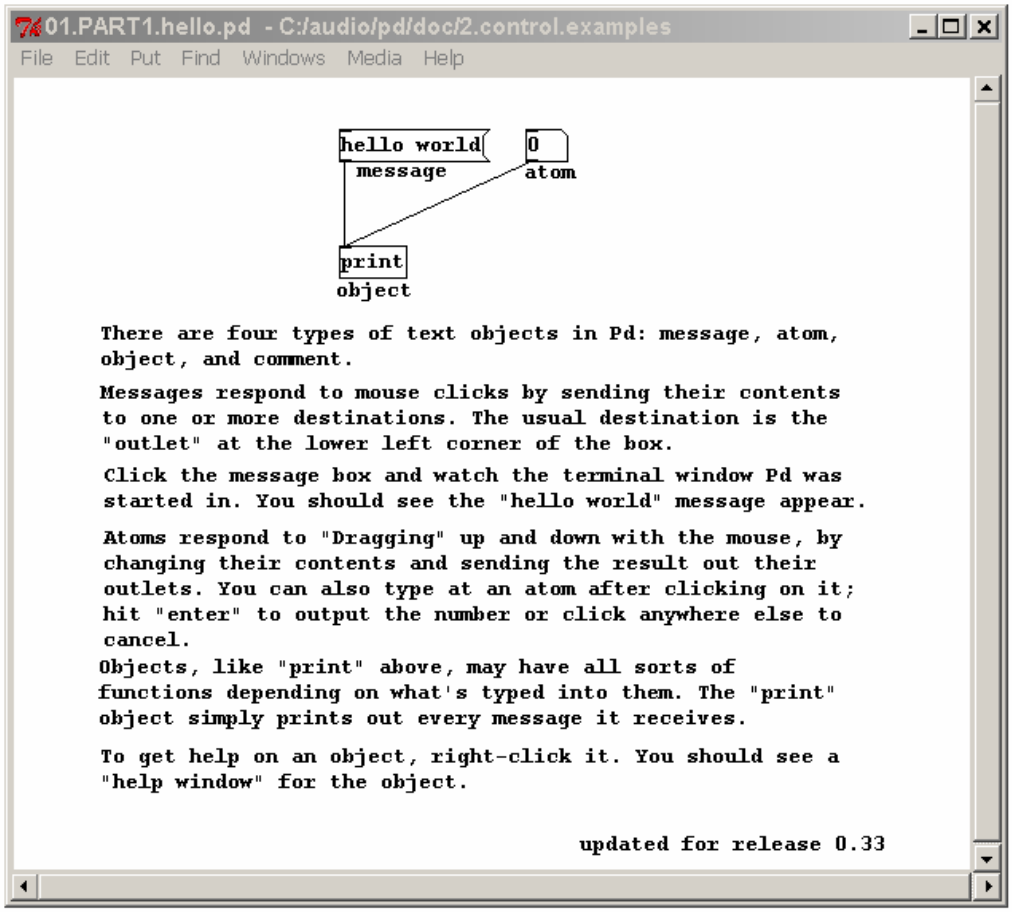
NB. Este último término (i.e. *funciones*) implica que, al igual que en los lenguajes de programación más convencionales, el usuario también podrá ir ampliando el vocabulario de PD, aunque de momento no lo vamos a considerar.



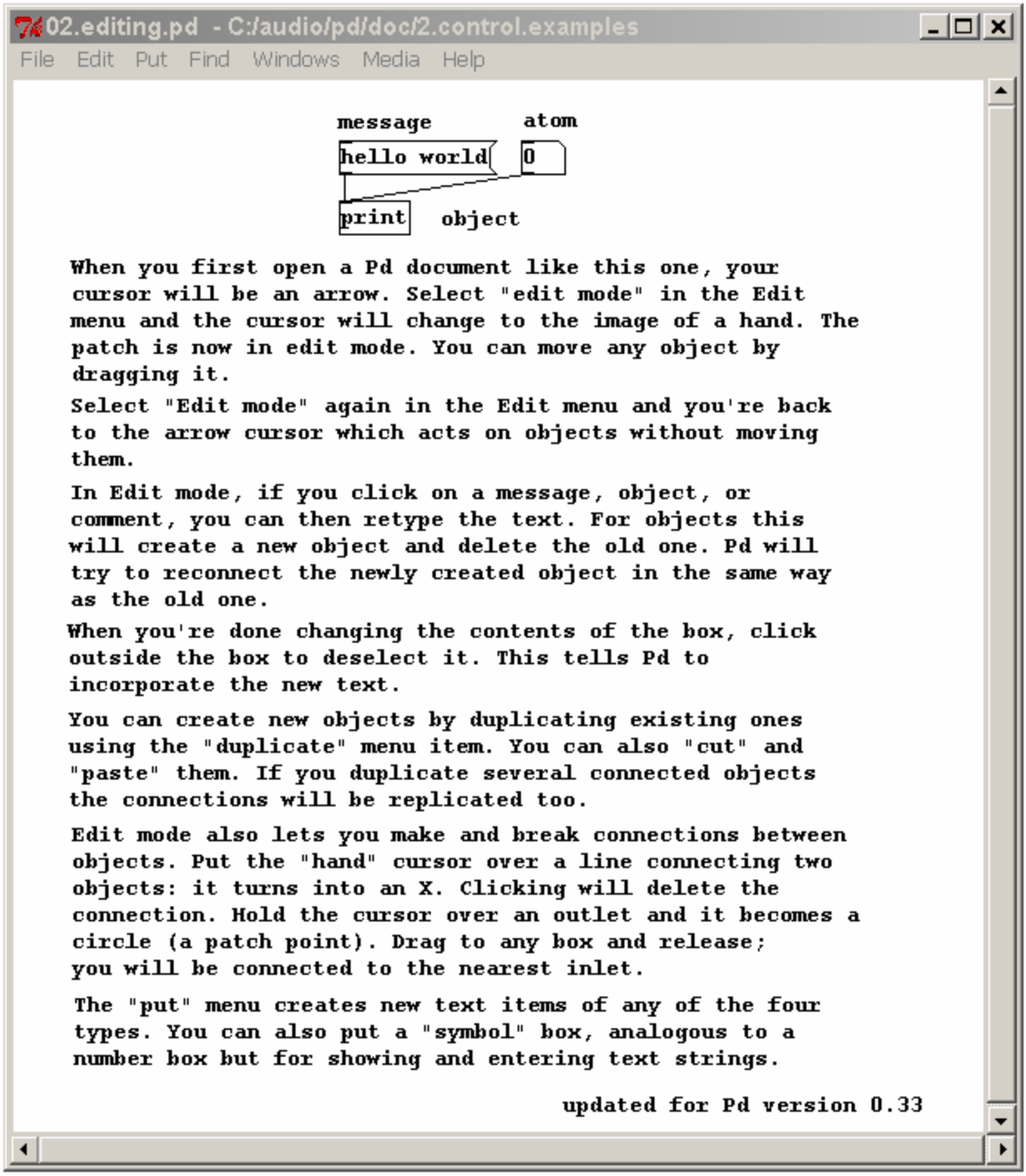
En el anterior programa PD (*01-hola.pd*), se ilustra el uso de los mensajes, de los objetos, como conectarlos entre sí, el uso de los comentarios, y la ventana negra de mensajes.

Para complementar este primer ejemplo, abrir desde PD los ficheros de ayuda del directorio 2.control.examples, 01.PART1.hello.pd y 02.editing.pd.





NB. La ayuda de PD son también patches PD, por lo que se pueden modificar para hacer pruebas, copiar, etc. Es obvio que en el caso de querer salvar las ayudas modificadas, mejor hacerlo en otro sitio y con otro nombre.



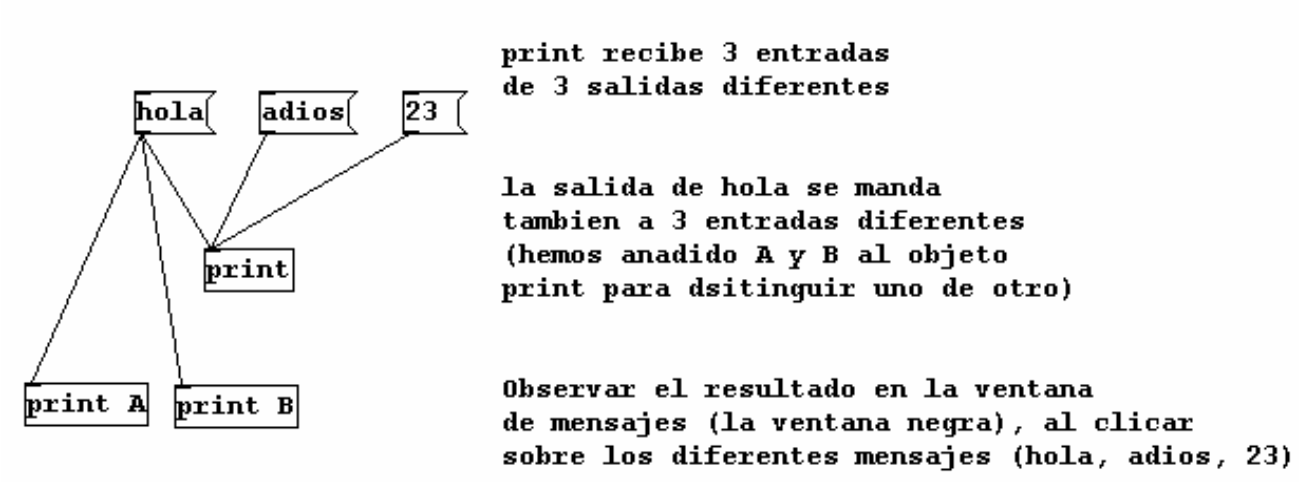
2.2. Entradas, salidas y conexión de objetos (y mensajes)

- Las marcas negras situadas en la parte superior de un objeto (o mensaje) son sus entradas
  El número de entradas de un objeto puede variar (de 0 a varias)

- Las marcas negras situadas en la parte inferior de un objeto (o mensaje) son sus salidas
  El número de salidas puede variar (de 0 a varias)

Como hemos visto en el ejemplo anterior, para conectar objetos y mensajes entre si, se debe clicar sobre la salida de un objeto (o mensaje) y arrastrar si soltar el ratón, hasta conectarlo con la entrada del objeto (o mensaje) que queramos conectar.

- Solo se puede conectar de salida (parte inferior de un objeto o mensaje) ➔ entrada (parte superior de un objeto o mensaje).
- Una salida puede conectarse a varias entradas
- Una entrada puede recibir de varias salidas



No todas las entradas y salidas son compatibles. Más adelante daremos más detalles. De momento, si intentamos conectar una salida con una entrada incompatible, aparecerá un mensaje de error en la ventana de mensajes, bien a la hora de conectarlos, o en otros casos, en ejecución, a la hora de activar el envío.

2.3. Desconexión, selección y eliminación de objetos

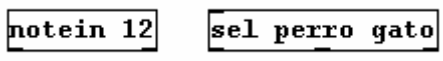
- Para eliminar una conexión, seleccionarla (en modo edición), y cuando esté seleccionada (azul), pulsar la tecla RETROCESO o BACKSPACE (En PD en Windows, la tecla SUPRIMIR o DEL no funciona). Lo mismo se aplica a la eliminación de objetos.

Selección de objetos

- Normalmente, al clicar sobre un objeto, estamos seleccionando su contenido (i.e. su nombre y el texto que aparece dentro). Al pulsar RETROCESO habremos borrado el nombre, no el objeto.
- Para borrar un objeto (y no el texto que contiene), hay que seleccionarlo enmarcando con el ratón. Después pulsar RETROCESO.

2.4. Sobre los nombres de los objetos

- Los nombres de los objetos no pueden contener espacios
- Si vemos un objeto en el que aparecen varios nombres (o números) separados por espacios, estaremos ante un objeto con argumentos. Más adelante (trataremos este tema



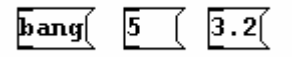
Ejemplo de 2 objetos (***notein*** y ***sel***) con argumentos.

- PD distingue entre mayúsculas y minúsculas. En este sentido, la mayoría de objetos PD utilizan sólo minúsculas en el nombre, pero en el caso de que alguno contuviera alguna mayúscula, deberíamos escribirlo tal cual

2.5. Mensajes especiales: bangs y constantes

Hemos afirmado que los mensajes -a diferencia de los objetos, que utilizan un vocabulario "selecto"- pueden contener cualquier texto. Esto es y no es cierto.

- Algunos objetos (*print* es un claro ejemplo) admiten cualquier mensaje.
- Otros admiten sólo algunos particulares, y pueden de hecho reaccionar de forma especial a cada uno de ellos. Todavía no entraremos en estos aspectos, pero tal vez pueda resultar intuitivo pensar que mensajes como "open" o "close" (por poner solo dos ejemplos) puedan tener efectos específicos en determinados objetos.



Existen algunos tipos de mensajes especiales, muy utilizados y válidos en muchos objetos. Describiremos "bang" y los valores numéricos constantes.

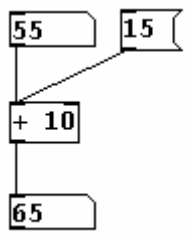
- *Bang* se utiliza para "disparar" cualquier tipo de evento. Significa algo así como "hazlo!". La mayoría de objetos que reciben un mensaje de tipo "bang" se ponen en marcha y realizan una acción (la acción dependerá del objeto en cuestión) y normalmente producen una salida.

- Los valores numéricos constantes se utilizan de la misma forma que en la mayoría de lenguajes de programación. Siempre que queramos guardar un valor numérico con un significado especial, o un valor por defecto.

En el próximo capítulo hablaremos más a fondo de entradas y salidas, y encontraremos aplicaciones claras de sendos tipos de mensajes.

2.6. Objetos numéricos (GUI)

Aunque más adelante veremos los objetos de interfaz gráfica, de momento adelantaremos uno: el objeto ***Number***, que se puede seleccionar desde el menú o con CTRL+3.

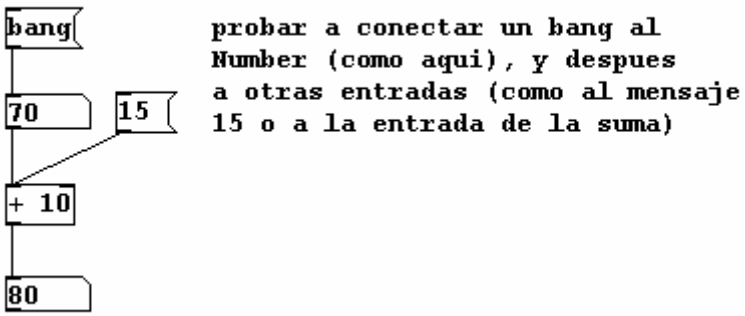


Este patch incluye dos objetos Number, un objeto + y un mensaje con un valor numérico constante (15).

NB. Observar que para que no de error debe existir al menos un espacio entre el signo + y el valor 10. Si no hubiese espacio, PD buscaría el objeto +10, que no existe. Al haber el espacio, lo que tenemos es el objeto + con un argumento, 10.

El objeto ***Number*** se comporta tanto como entrada como salida. Si (en modo ejecución) arrastramos el ratón por encima del objeto, modificaremos su valor.

1. Experimentar un poco con el patch para entender el funcionamiento.
2. Añadir un mensaje bang e irlo colocando sucesivamente en diferentes entradas para estudiar su funcionamiento.
3. Tratar de describir lo que sucede y de descubrir las reglas que rigen este comportamiento



3. Entradas, salida, prioridades (y el objeto *trigger*)

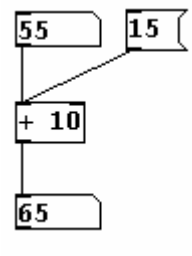
3.1. Introducción

PD es en muchos aspectos un lenguaje poco convencional. Una de sus peculiaridades es que al ser un lenguaje totalmente gráfico, el flujo se representa bidimensionalmente (en un plano), en lugar de linealmente, que es como se escribe texto en el papel (una línea después de la otra).

Dado que de un punto pueden producirse varias salidas y que un objeto puede tener varias entradas (y que en un ordenador no existe realmente el concepto de simultaneidad), la pregunta es ¿En que orden se ejecutan los patches PD?

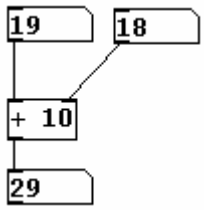
Las consideraciones de este capítulo son esenciales para entender bien el funcionamiento de PD.

3.1. Entradas activas



Retomemos el ejemplo anterior y hagamos algunos cambios.

1. Eliminemos el mensaje (15)
2. Añadamos un nuevo objeto ***Number***, pero conectándolo ahora a la entrada derecha del objeto +.



En el nuevo objeto se observa que:

1. El argumento situado en el objeto + (10) deja de valer en cuanto se activa un nuevo número por la derecha. Aunque sigamos viendo un 10, estaremos ahora sumando las dos entradas (e.g. 19 y 18)
2. Las dos entradas (izquierda, derecha) NO se comportan igual. El objeto + sólo produce una salida cuando recibe una entrada del ***Number*** de la izquierda.

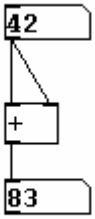
Una primera regla fundamental de PD es que la mayoría de objetos, sólo disparan una salida cuando reciben algo por la entrada más a la izquierda.

La entrada izquierda es la entrada activa

Normalmente, las entradas que va recibiendo un objeto, se van almacenando en su interior, hasta que la entrada izquierda active una salida. Esta salida se calculará con los valores que se habrán ido almacenando.

El patch de suma que acabamos de realizar es una muestra de este comportamiento: cuando se produce una entrada por la izquierda, se calcula la suma, utilizando el último valor que se hubiese recibido por la derecha. Este comportamiento se podría generalizar si el objeto tuviese más entradas (en este caso sólo tiene dos).

3.2. Orden de salidas



Escribimos un patch como el de la imagen.

Intuitivamente, la salida debería ser el doble de la entrada, ya que un mismo número entra por los dos lados, con lo cual se suma a si mismo. Sin embargo, el resultado no siempre es correcto[1].

Experimentar con este patch e intentar deducir lo que sucede.

Explicación:

El patch funcionará bien, si la primera conexión que hacemos a la hora de diseñarlo es la de la derecha.
La razón es que cuando en PD una salida se dirige a varias entradas, la que primero sale es la primera que se conectó.

Si la primera en conectarse fue la de la derecha, cuando se modifique un valor en ***Number***:

1. El valor saldrá primero por la derecha y se almacenará en la derecha del objeto +
2. El valor saldrá a continuación por la izquierda y al llegar a +, disparará una salida con el valor almacenado en la derecha (el mismo valor)

Si por el contrario la primera en conectarse fue la de la izquierda, cuando se modifique un valor en ***Number***:

1. El valor llegará por la izquierda de + y disparará una salida
2. + utilizará como valor almacenado el que hubiera llegado anteriormente (el anterior valor de ***Number***)

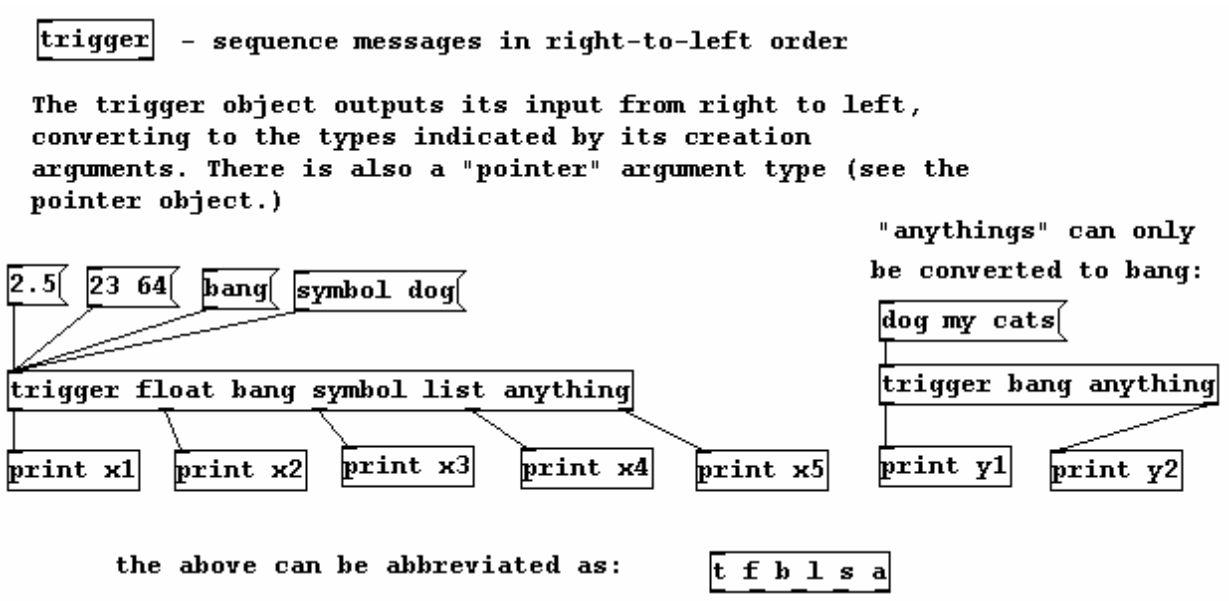
- Por esta razón, si hacemos las conexiones en el orden incorrecto y vamos incrementado lentamente ***Number***, el resultado será siempre del doble menos uno, ya que sumará el valor que dispara la salida con el valor anterior.
- Si en lugar de incrementar ***Number***, vamos decrementándolo lentamente, el resultado será del doble más uno.
- Si variamos ***Number*** más rápidamente, el resultado se vuelve impredecible, ya que el valor almacenado puede no ser el inmediatamente anterior.

[1] Si funciona bien, lo hará siempre. Si funciona mal, seguirá funcionando mal. Si lo borramos y lo volvemos a construir, aunque aparentemente sea siempre igual, en algunos casos funcionará bien, en otros no.

3.3. Soluciones : trigger

El que un patch funcione bien o mal, no puede depender del orden en que hayamos hecho las conexiones, ya que este orden no se percibe al estudiar el patch, y tendríamos por lo tanto comportamientos diferentes para patches aparentemente iguales.

Para solucionar este problema, existe un objeto, ***trigger***, que se debe usar SIEMPRE que queramos que una misma salida se distribuya por varias conexiones.



Esta era la ayuda PD del objeto ***trigger***. Lo que viene a decir es lo siguiente:

1. ***trigger*** admite una entrada y varios argumentos
2. Estos argumentos pueden ser los siguientes: *float*, *bang*, *symbol*, *list* y *anything*
3. El número de salidas de ***trigger*** será igual al número de argumentos.

   De momento nosotros nos centraremos sólo en los argumentos *bang* y *float*

   > **NB**. La palabra **float** significa flotante, y es la palabra que se suele utilizar en programación para designar valores numéricos no enteros, ya que se almacenen en una forma denominada *coma flotante*.

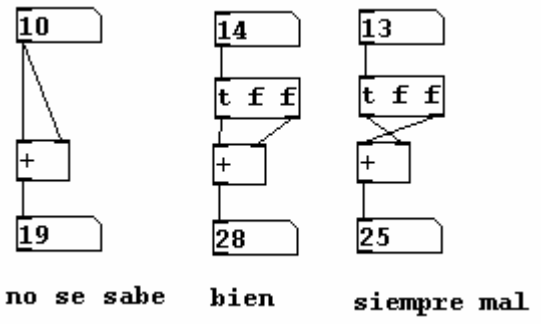
   float se utiliza en PD para designar y/o almacenar cualquier tipo de valor numérico, ya sea entero o con decimales. Por ello, si no queremos preocuparnos demasiado, podemos pensar en la palabra *float* como sinónimo de *número*.

La línea inferior de la ayuda, nos cuenta también que en el caso de ***trigger***, esta notación un tanto farragosa puede simplificarse, sustituyendo

- `trigger → t`
- `bang → b`
- `float → f`

- trigger permite redistribuir la entrada por varias salidas,
- garantizando el orden de salida (de derecha a izquierda)
- y dando la opción de convertir la salida a otro tipo de dato, por ejemplo, convirtiendo un valor numérico en un mensaje de tipo bang

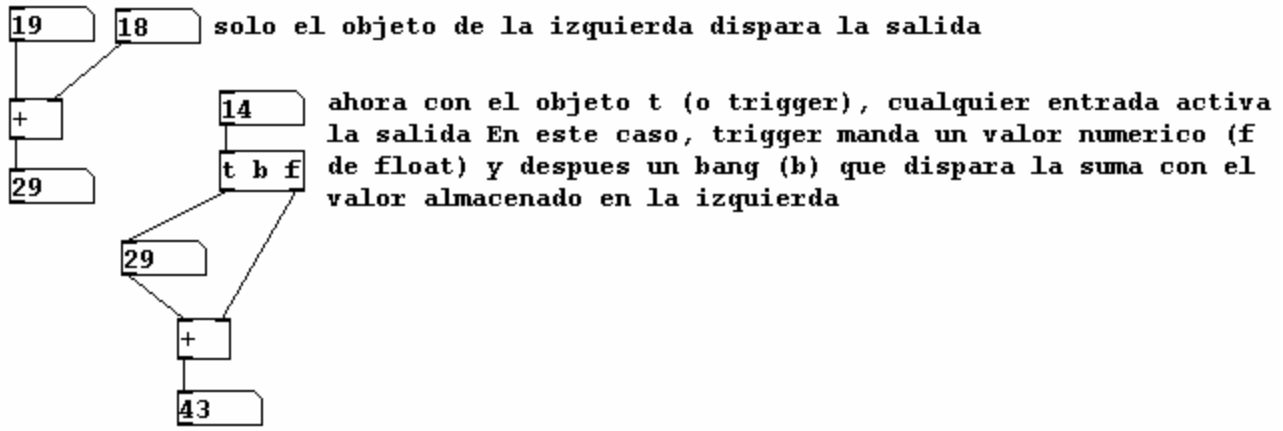
Si modificamos nuestra patch que pretendía sumar un número consigo mismo de la siguiente manera, estaremos garantizando que funcione siempre bien.

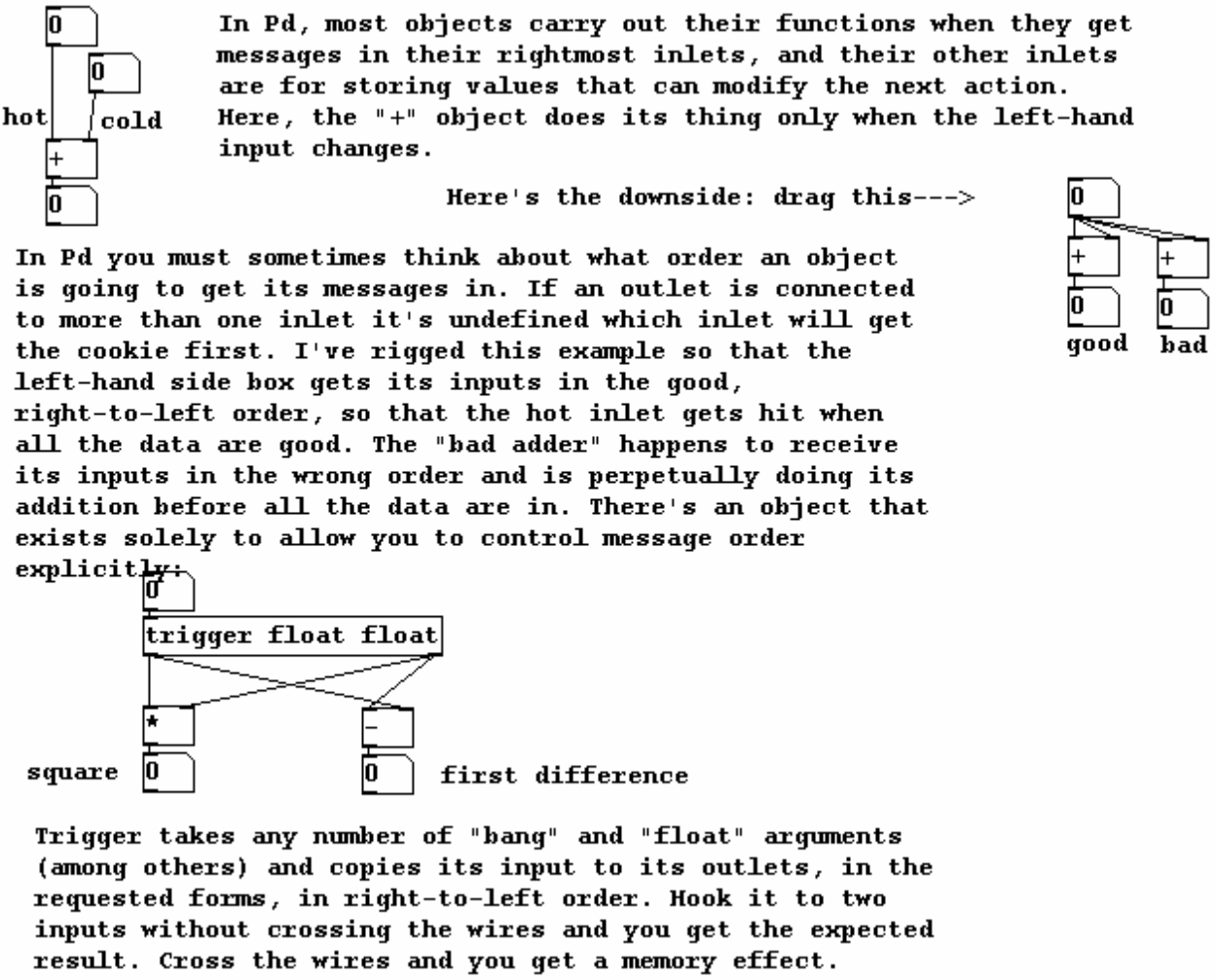


El comportamiento de *trigger*, de garantizar que sus varias salidas se realizarán en orden de derecha a izquierda es otra de las reglas de PD extensibles a casi todos los objetos

3.4. trigger y conversión a bang

En el siguiente ejemplo vemos como se puede utilizar la funcionalidad de ***trigger*** de reconvertir las entradas.

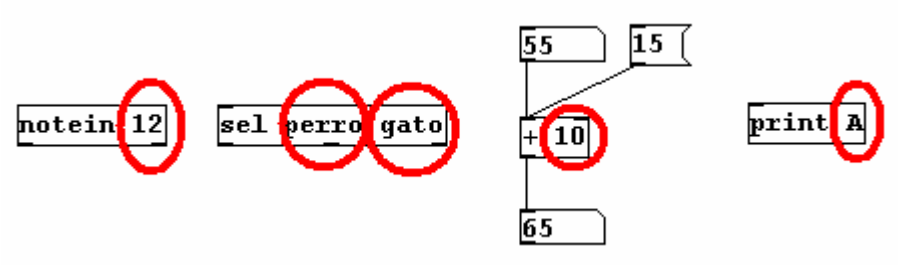




El ejemplo *02.control.examples/03.connections.pd* muestra también estos conceptos

3.5. Argumentos

Hemos visto ya varios objetos que aceptan argumentos:



El **12** de notein, **perro** y **gato** en sel, el **10** del operador +, el **A** de print…son todos argumentos.

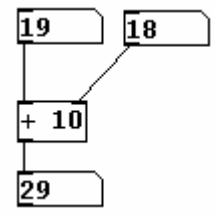
Los argumento son símbolos (textuales o numéricos) que se sitúan a continuación del nombre de un objeto (separados de éste por al menos un espacio).

Número y tipo de los argumentos

- Algunos objetos no aceptan argumentos
- Otros aceptan un número fijo (e.g. +)
- Otros un número variable (e.g. trigger)
- Algunos aceptan cualquier símbolo (e.g. print)
- Otros sólo símbolos específicos (e.g. valor númerico para los operadores como suma, o los símbolos específicos de select)

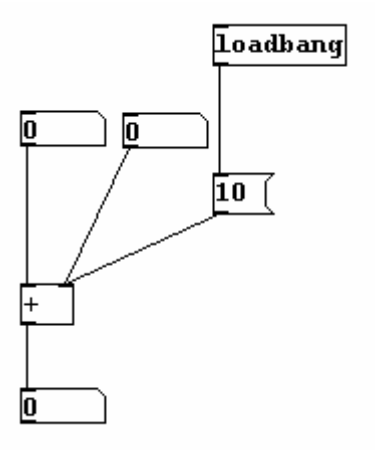
Argumentos como valor defecto

En algunos casos, los argumentos determinan un valor inicial por defecto.
En estos casos, las entradas de la derecha suelen modificar este valor (aunque visualmente no se modifique este valor en el patch), como es el caso de la suma:



- Inicialmente, todo valor que entra por la izquierda, sale sumado en 10.
- Después de que hayamos introducido un nuevo valor por la izquierda (e.g. 18), el sumando toma un nuevo valor (e.g. 18), aunque lamentablemente seguimos visualizando un 10, lo cual puede prestar a confusión.

Una alternativa a este tipo de argumentos, sería utilizar un ***loadbang***, que manda un bang cuando se abre el patch. Aunque algo más farragoso, el siguiente patch cumple la misma funcionalidad que con el argumento dentro de la suma.



Ya desde el inicio, el sumando de la derecha toma el valor 10. Pero el hecho de que no se visualice, hace que en cuanto se modifique el 10, este patch sea menos confuso que el anterior.

Argumentos que modifican el comportamiento de un objeto

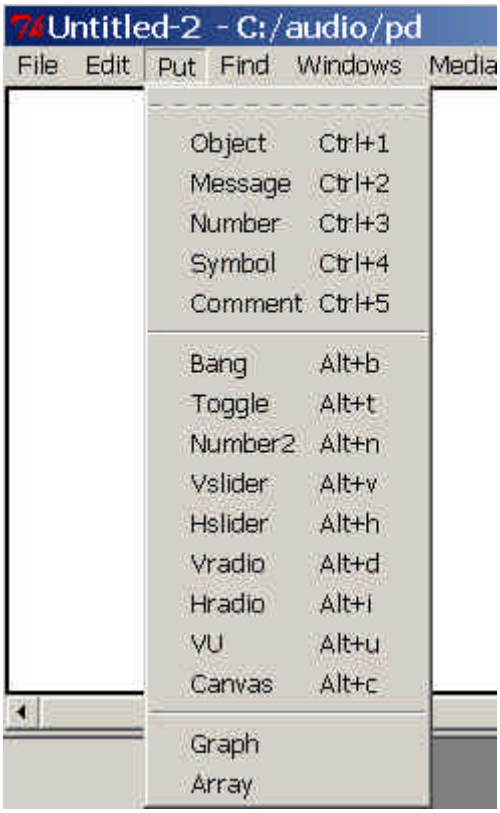
En algunos casos, los argumentos alteran el comportamiento de un objeto, como en el caso de ***trigger*** que acabamos de ver, en que el número de argumento determina incluso el número de salidas del objeto. En estos casos, estos argumentos no suelen poderse modificar en ejecución.

3.6. Resumen

1. Los objetos PD disparan una salida cuando reciben algo por la entrada izquierda
2. Cuando un objeto PD tiene varias salidas el orden de salida de éstas es de derecha a izquierda
3. Cuando una misma salida se manda a varias entradas diferentes es difícil garantizar el orden de llegada, a no ser que se utilice el objeto ***trigger***
4. El objeto ***trigger*** se utiliza para garantizar el orden de salida cuando queramos mandar un mismo valor o mensaje a varias entradas diferentes
5. Los argumentos se utilizan para inicializar objetos y para modificar su comportamiento.

4. Objetos de interfaz de usuario (GUI objects)

Hemos visto el objeto Number que permite introducir valores numéricos mendiante el ratón. En PD existen otros objetos de interfaz gráfico (GUI). Estos objetos permiten normalmente modificar su aspecto (tamaño, color, etc) y algunas otras opciones de configuración.



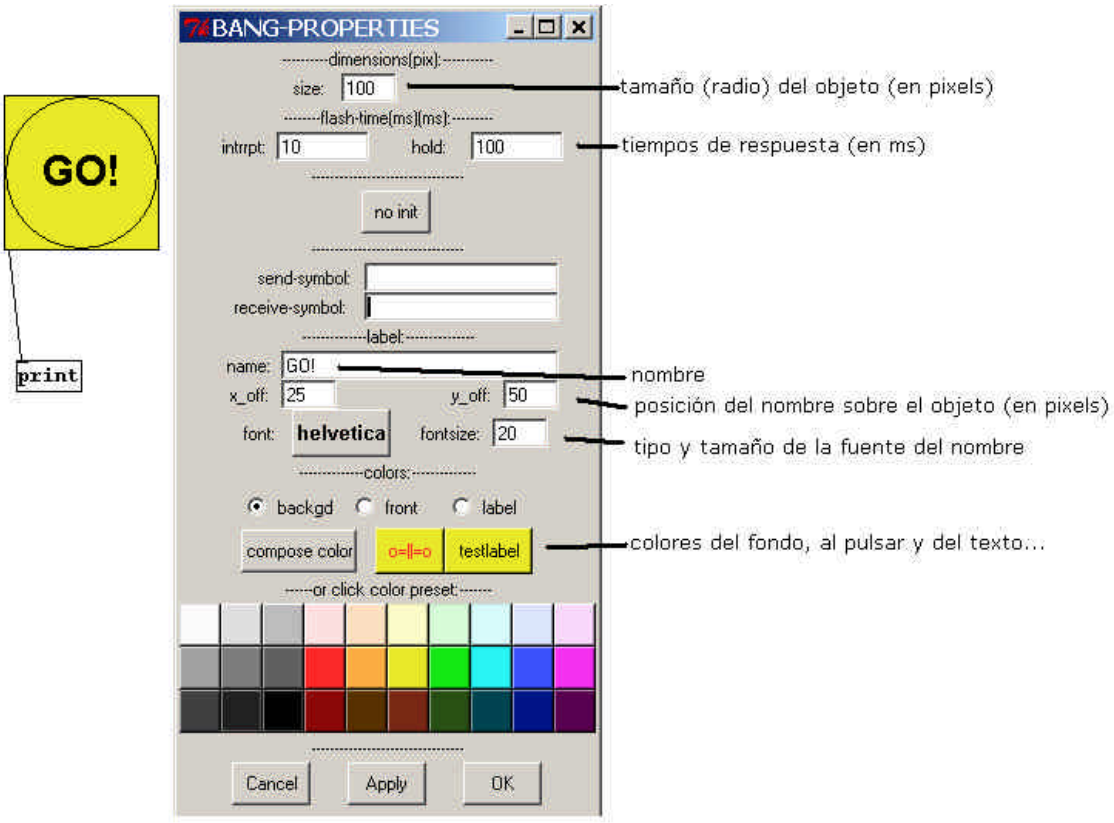
1. Bang
2. Toggle
3. Number2
4. Vslider
5. Hslider
6. Vradio
7. Hradio
8. VU
9. Canvas

4.1. Bang

- Se comporta exactamente igual que el mensaje bang
- Su aspecto es el de un botón pulsador de tamaño y color configurable por el usuario que manda un bang cada vez que se pulsa con el ratón (o cada vez que recibe cualquier entrada)

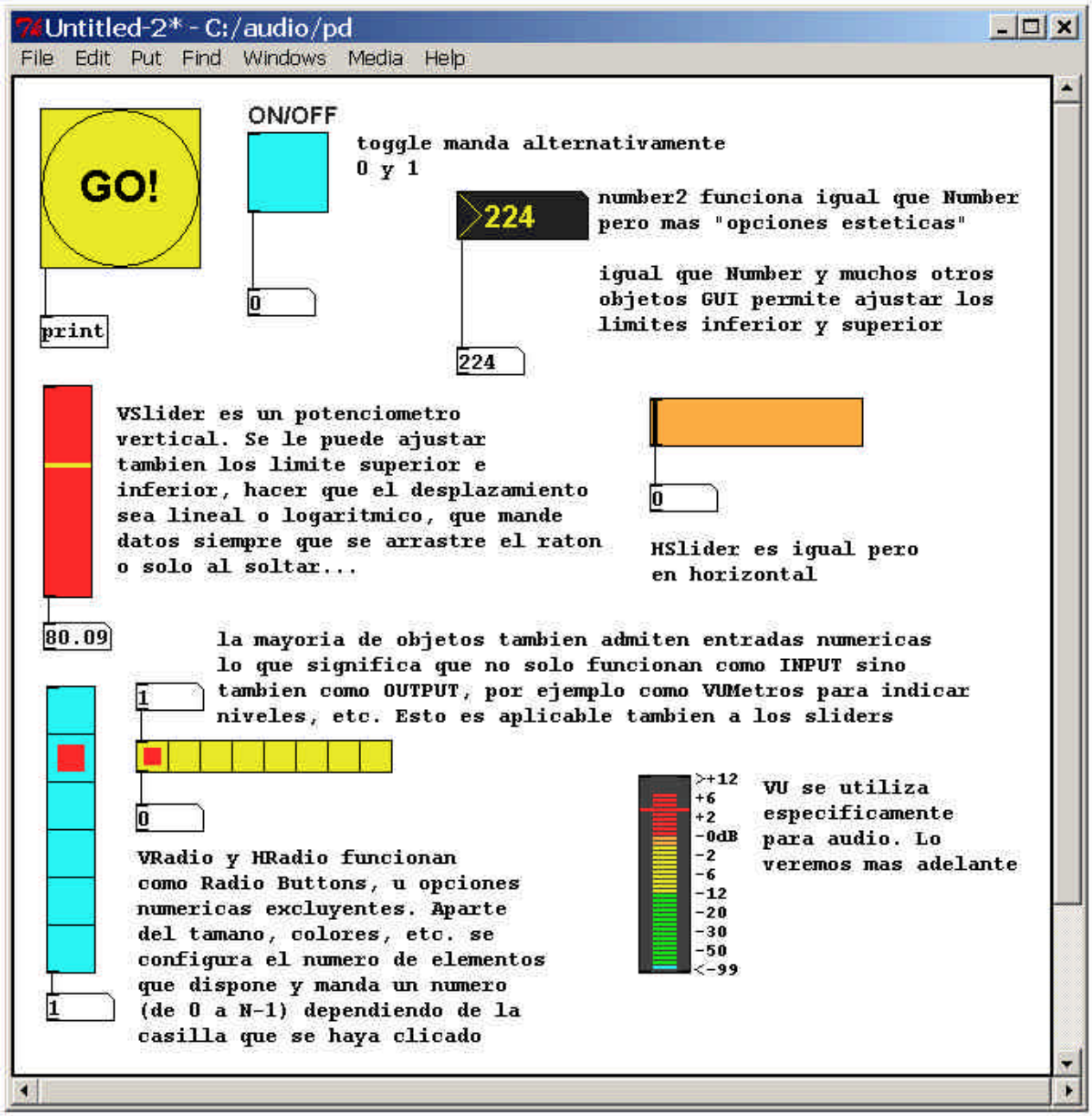
Mostraremos las posibilidades cosméticas de este objeto, en su mayoría aplicables también a cualquiera de los objetos GUI restantes.

Para activar el menú de configuración de cualquier objeto gráfico, basta pulsarlo con el botón derecho y seleccionar **Properties**.



Más adelante describiremos los campos **send-symbol** y **receive symbol** (presentes también en los otros objetos GUI)

4.2. Descripción de los objetos restantes



4.3. Resumen

- Existen varios objetos de tipo GUI
- Todos ellos permiten configurar y modificar su apariencia (tamaños, colores, etc.)
- En aquellos que manejan datos numéricos (todos salvo bang) se pueden configurar los límites inferior y superior
- Todos estos objetos disponen de salidas pero también de entradas, por lo que pueden funcionar tanto como interfaz de control (tomando INPUT del ratón) como para monitorizar datos, o ambos (IN & OUT) a la vez. Cuando reciben un valor en entrada modifican su aspecto y reenvían el valor a la salida

5. Objetos MIDI

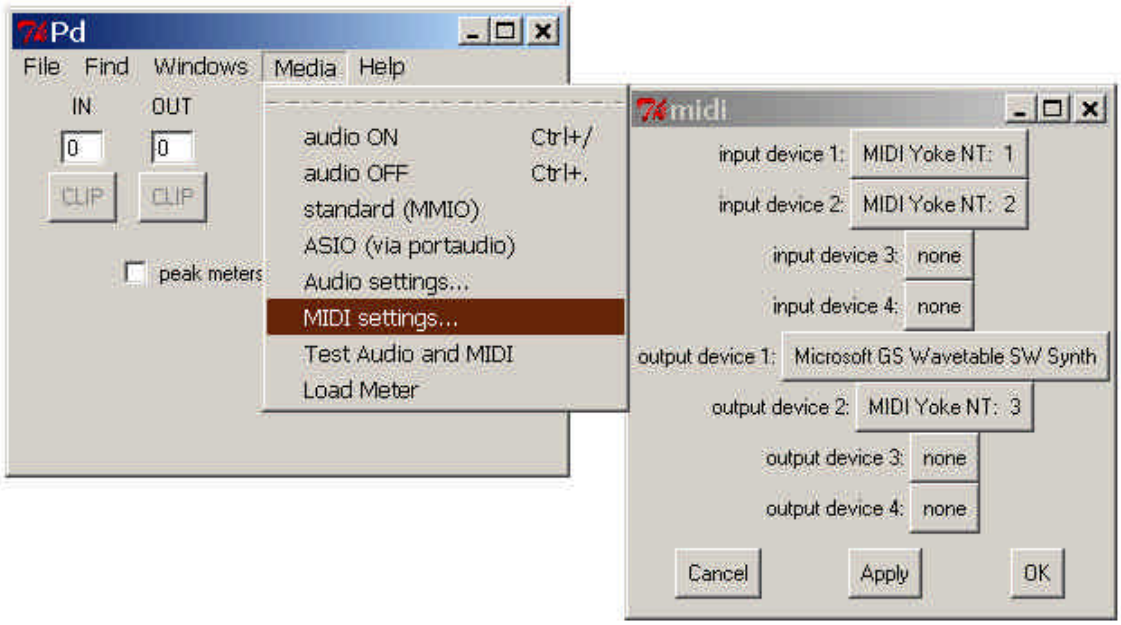
Aviso Preliminar

Este capítulo no es una introducción al MIDI; se supone que el lector sabe lo que es el MIDI, los canales, los diferentes tipos de mensajes, etc. Existen muchas publicaciones y webs que incluyen información para los principiantes que no posean estos conocimientos.

Para una introducción al MIDI el lector puede consultar el libro de este mismo autor, Sergi Jordà, *Guía Monográfica del Audio Digital y el MIDI*, Anaya, Madrid 1997, disponible en versión PDF y HTML en las siguientes direcciones:

XXXXXXXXXXXXX

5.1. Configuración MIDI en PD

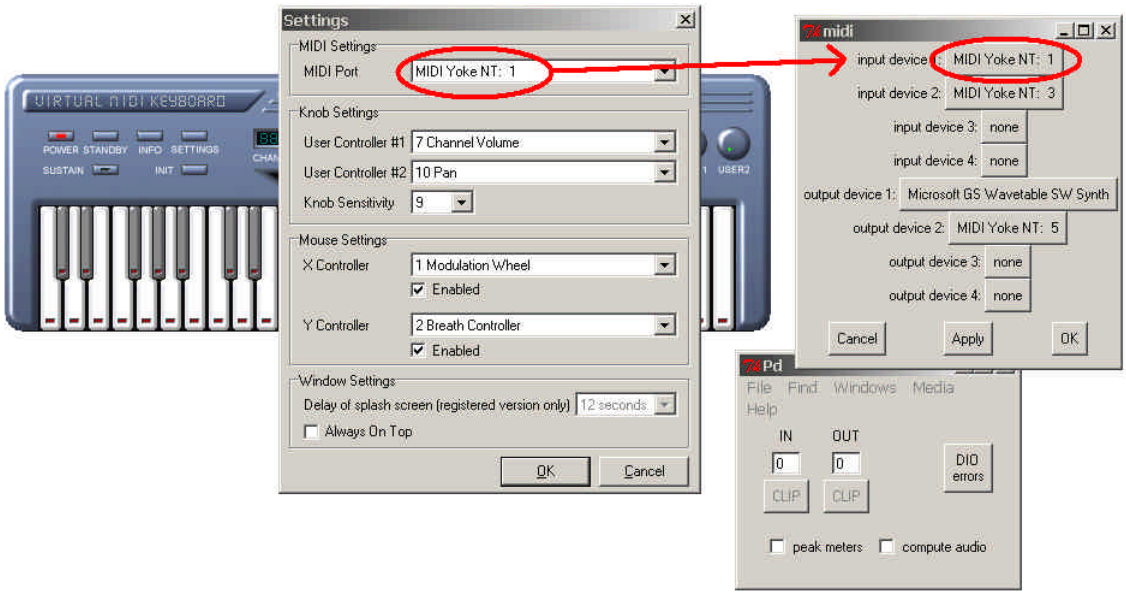


- PD puede manejar hasta 4 puertos MIDI de entrada y 4 de salida (siempre que estén disponibles en el ordenador)
- Estos puertos pueden ser reales (hardware) o virtuales
- La selección de puertos se realiza mediante menú, tal como se muestra en la figura.

El uso de puertos virtuales permite interconectar varias aplicaciones MIDI de un mismo ordenador. Para ello es necesario instalar drivers MIDI virtuales como MIDI Yoke, que se puede bajar gratuitamente de Internet.

A partir de ahora, conviene configurar el MIDI en PD.

- Si disponemos de un teclado o de otro dispositivo controlador externo, seleccionar en PD la entrada MIDI IN a la que esté conectado el teclado.
- Si disponemos de una tarjeta de sonido, seleccionar ésta como puerto de salida.
- Si no disponemos de ningún teclado externo, podremos realizar las pruebas utilizando un teclado virtual (como por ejemplo el Virtual MIDI Keyboard de Wouter van Beek, disponible en http://www.geocities.com/SiliconValley/Campus/6501/). En este último caso, deberemos tener instalado unos puertos virtuales como MIDI Yoke, y conectar mediante él, la salida del teclado virtual a la entrada de PD.



5.2. Introducción a los objetos

PD incorpora unos cuantos objetos que facilitan enormemente la entrada, salida, procesado y generación de datos MIDI.

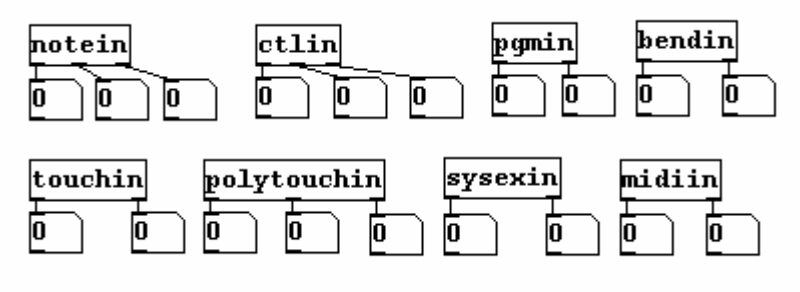
MIDI in *notein ctlin pgmin bendin touchin polytouchin midiin sysexin*
MIDI out *noteout ctlout pgmout bendout touchout polytouchout midiout*
Generación *makenote*
Procesado *stripnote*

En los casos de IN y OUT, hay sendos objetos de entrada genéricos (midiin y midiout) y un objeto para cada tipo de mensaje MIDI

Mensaje MIDI	Objeto IN	Objeto OUT
Note ON	*notein*	*noteout*
Control Change	*ctlin*	*ctlout*
Program Change	*pgmin*	*pgmout*
Pitch Bend	*bendin*	*bendout*
Aftertouch	*touchin*	*touchout*
Poly Aftertouch	*polytouchin*	*polytouchout*
SysEx	*sysexin*	
Genérico	*midiin*	*midiout*

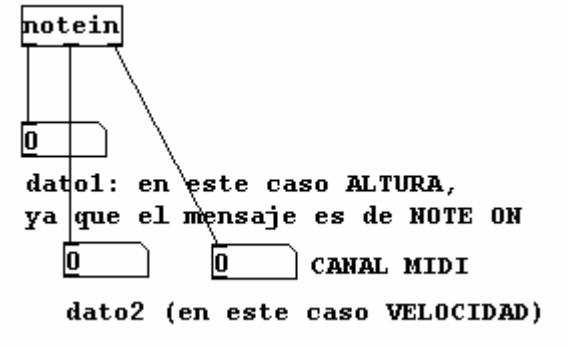
5.3. MIDI IN

Se observa que estos objetos tienen un número variable de salidas, pero ninguna entrada. Al principio, tal vez pueda parecer extraño que objetos que se denominan "de entrada" carezcan precisamente de entradas en PD, pero eso es así porque precisamente estos objetos no reciben INPUTS desde PD sino desde los puertos MIDI. Y es esa información recibida de los puertos, la que después ofrecen a PD por medio de sus salidas.



El número de salidas de cada objeto depende del número de datos que maneja el mensaje. Así, mensajes MIDI como los de Note On o Control Change que tienen 3 datos (canal, altura y velocidad en el caso de la nota, canal, número de control y valor de control, en el caso del control change), disponen aquí de 3 salidas, mientras que aquellos que como Program Change tan sólo tienen dos datos (canal y programa), sólo tienen dos salidas[2].

Cuando el objeto tiene todas sus salidas, el canal sale por la de más a la derecha, mientras que el dato 1 sale por la izquierda (y por consiguiente, dato2, por la del medio).



Orden de salida:

Aunque el orden de salida en PD es casi siempre el mismo (i.e. de derecha a izquierda, cfg. 3.2), no está de más recordarlo de vez en cuando. Esto significa que cuando se recibe un mensaje MIDI, lo primero que llega es el canal, después el dato2 y después el dato1.

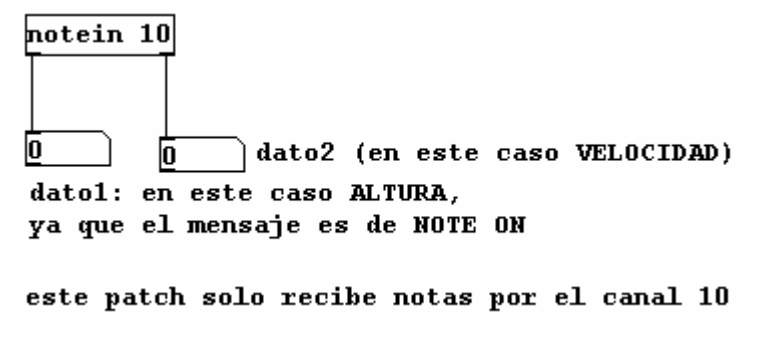
[2] En PD no existe el mensaje de NOTE OFF. Se recibe siempre mediante un NOTE ON con velocidad 0.

Objetos MIDI y argumentos

Los argumentos son muy importantes en este grupo de objetos.

Cuando hay un solo argumento, éste indica un canal.

En este caso, el objeto filtra las entradas, dejando pasar únicamente las del canal en cuestión. Entonces desaparece también una salida (ya que el dato canal es innecesario)

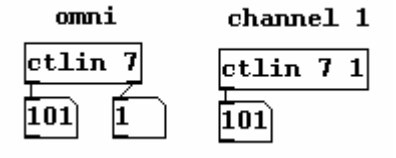


Dado que no hay ningún problema en tener tantos objetos de entrada MIDI simultáneos como se deseen, esta sería una primera forma de separar las entradas por canales.

Este principio se aplica a todos los objetos MIDI, menos a los de Control Change (***ctlin*** y **ctlout**) que presentan una inconsistencia que puede prestar a confusión.

Argumento en los objetos de Control Change

Cuando los objetos de control tienen un único argumento, éste indica el número de control y no el canal! Para filtrar por canal se debe añadir un segundo argumento.

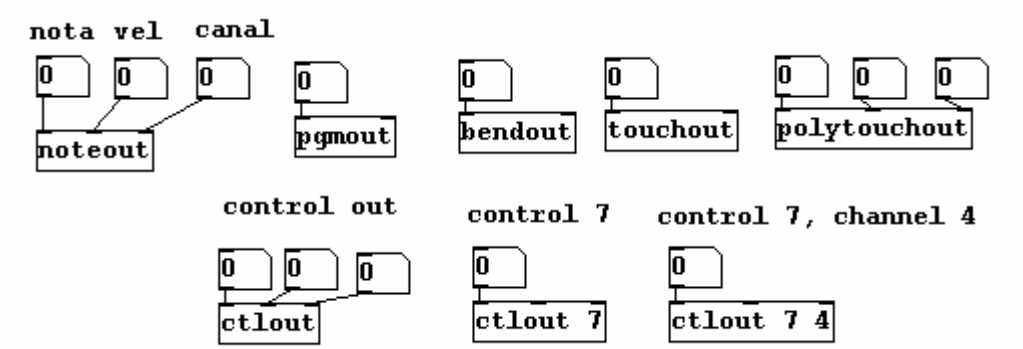


- El patch de la izquierda recibe mensajes de control 7 (i.e. volumen) por cualquier canal.
- El patch de la derecha recibe sólo mensajes de control 7 por el canal 1.

5.4. MIDI OUT

Los objetos MIDI de salida son equivalentes a los de entrada.

- Tienen los mismos nombres pero con la terminación cambiada
- No tienen salidas, ya que no salen a PD sino a un puerto MIDI.
- Tienen entradas y su número y orden coincide con el de los objetos de entrada
- Admiten argumentos y su comportamiento también es similar al de los objetos de entrada, de forma que el argumento indica canal de salida, salvo en el caso de los mensajes de control en donde un solo argumento indica tipo de control.
- La única diferencia es que al incluir argumentos, no disminuye el número de entradas.

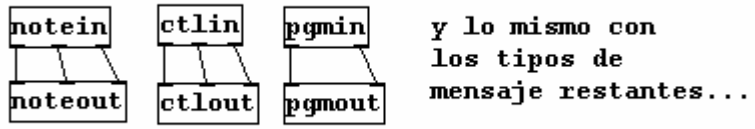


NB: Inicialmente, si no indicamos ningún canal de salida, ésta se mandará por el canal 1

5.5. Ejemplos MIDI sencillos

MIDI THRU

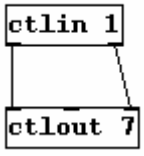
PD no está en THRU por defecto. Para ello, bastaría con el siguiente patch:



Redireccionamiento de canal

(a) Los mensajes de nota recibidos por el canal 5 se envían por el canal 3
(b) Los mensajes de nota recibidos por cualquier canal se envían por el canal 4

Cambio de tipo de mensaje



Los mensajes de control de tipo 1 (típico mensaje defecto mandado por la mayoría de ruedas de modulación de cualquier teclado MIDI) se convierte en mensaje de control 7 (volumen). Este patch funciona en modo OMNI (es decir para todos los canales).

Ejercicios propuestos

1. Hacer el patch anterior con entrada en canal 5 y salida en canal 6, pero configurable por el usuario de forma interactiva.
2. Transponer el teclado en una octava más
3. Convertir cada nota en un acorde triada, de forma que cada vez que introduzcamos una nota, suene un acorde (de momento sin tener en cuenta la tonalidad). Más adelante se propondrá un armonizador más sofisticado.
4. Crear un teclado "espejo" (de forma que las notas vayan del agudo al grave). Mirar para ello la documentación del objeto ***abs*** (que calcula el valor absoluto)
5. Cambiar programas (Program Change) mediante mensajes de control número 3, preservando el canal de mensaje.
6. Hacer que el usuario pueda elegir lo más cómodamente posible entre un sonido de trompeta y uno de guitarra, indicando también el canal MIDI en el que se

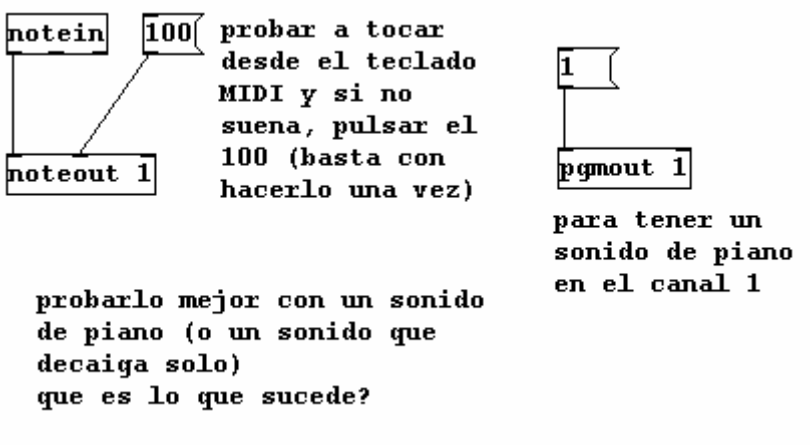
desea hacer el cambio (consultar la lista del Apéndice D2 para la lista numérica de los programas GM[3]).

7. Utilizar sliders (objetos GUI) para crear un mezclador MIDI gráfico que mediante 16 sliders verticales y 16 sliders horizontales, controle respectivamente el volumen (Control Change 7) y la panorámica de los 16 canales MIDI.

Resultados en Apéndice A, página XX.

Experimentar!!

PD es la herramienta idónea para experimentar; para comenzar a realizar pequeños programas sin necesidad de tener claro el resultado final; para coger patches encontrados y modificarlos…



- Probar por ejemplo este pequeño patch desde un teclado MIDI
- ¿Que sucede? ¿Porqué?

Descripción

En principio deberíamos oír lo que se toque al piano, duplicado, con retardo, y las notas se deberían quedar sonando un rato como si el piano tuviese pedal.

La explicación es que hemos eliminado el mensaje de velocidad, de forma que cada vez que pulsemos una tecla, se mandará un nuevo mensaje, pero también cuando la soltemos. Dado que no se manda nunca la velocidad, ***noteout*** *utilizará el valor que tenga guardado. Si al principio no sonase, es que tiene guardado un valor 0, por eso hemos incluido el valor constante 100. A partir de este momento, cuando pulsemos una tecla del piano, mandaremos pues la altura, y PD la hará sonar con una velocidad 100 (bastante intensa, teniendo en cuenta que el máximo es 127). Cuando soltemos la tecla, el teclado estará mandando un nuevo mensaje con la misma altura pero ahora con velocidad 0. Dado que nuestro patch ha filtrado la velocidad, lo que recibiremos será otra vez la misma nota, y PD le asignará de nuevo una velocidad 100, y no 0 (fin de nota) como debería ser. En lugar de apagar la nota anterior, esta sonará de nuevo con velocidad 100…*

[3] La lista de D2 indica unos valores de 0 a 127, mientras que PD utiliza un rango de 1 a 128. Esto significa que hay que sumar siempre 1 a los valores de la tabla.

5.6. Uso de varios puertos MIDI simultáneos

Hemos visto que PD permite utilizar simultáneamente hasta 4 puertos MIDI de entrada y 4 de salida (lógicamente, en el caso de que el ordenador disponga de este número de puertos). Ello hace que aunque dispongamos de dispositivos hardware MIDI tanto de entrada como de salida, el uso de puertos virtuales, que permiten interconectar aplicaciones MIDI entre si, sea muy recomendable.

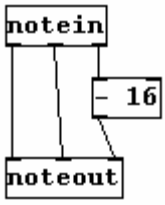
Por ejemplo, podremos tener a PD recibiendo datos MIDI de uno o varios dispositivos externos (si nuestra interfaz MIDI soporta varias entradas) y simultáneamente de un secuenciador interno como Cubase o Cakewalk, y estar mandando los datos de salida a un dispositivo hardware externo, a la tarjeta de sonido, a un sintetizador por software como Reaktor o Audiomulch y al mismo secuenciador de antes.

Las posibilidades de este tipo de configuraciones se dejan para el lector. Aquí explicaremos tan sólo como utilizar estos varios puertos desde PD.

- El primer dispositivo de entrada seleccionado en la lista manda por los canales 1-16.
- El segundo dispositivo de entrada lo hace por los canales 17-32.
- El tercer dispositivo de entrada por los canales 33-48.
- El cuarto dispositivo de entrada por los canales 49-64.

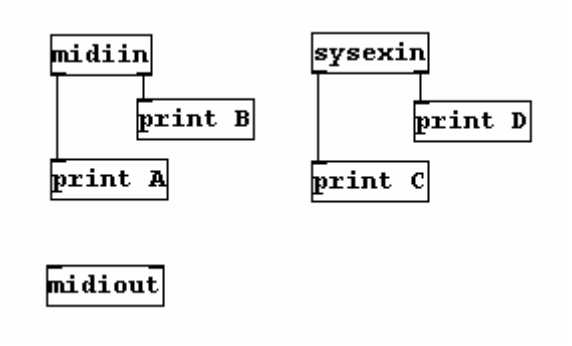
Lo mismo sucede con los dispositivos de salida.

- Si mandamos algo por los canales de salida 1-16, lo recibirá el primer dispositivo de salida de la lista.
- Si mandamos algo por los canales 17-32, lo recibirá el segundo dispositivo.
- Si mandamos algo por los canales 33-48, lo recibirá el tercero.
- Si mandamos algo por los canales 49-64, lo recibirá el cuarto.



En este ejemplo queremos que las notas que entren procedentes del dispositivo IN2, se manden tal cual al dispositivo OUT1. Dado que el de entrada maneja los canales 17-32 y el de salida del 1-16, lo que haremos será restar 16 al valor del canal de cada mensaje que recibamos.

5.7. midiin, midiout & sysexin



Estos objetos son más complejos que los anteriores. Se incluyen aquí su descripción, pero no se detallará su uso, dejando su estudio para los lectores más avanzados (que sabrán que hacer con ello).

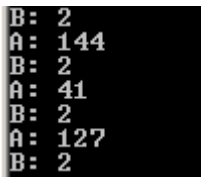
- ***midiin*** recibe cualquier tipo de mensaje MIDI sin filtrar
- ***sysexin*** recibe mensaje de sistema exclusivo
- ni ***midiin*** ni ***sysexin*** aceptan argumentos para filtrar el canal MIDI[4] ni el puerto
- ***midiout*** permite enviar cualquier tipo de mensaje por cualquier canal y puerto

midiin

midiin y ***sysexin*** descomponen byte a byte todos los mensajes MIDI recibidos

- la salida derecha es un valor numérico que indica el puerto
- la salida izquierda es el valor numérico del byte correspondiente.

Así, cuando el ***midiin*** del patch anterior recibe un mensaje de NOTE ON (3 bytes) e imprima en la ventana de mensajes:



Esto indicará:

- NOTE ON por canal 1 (144) y puerto 2
- la nota es un LA (41) (y puerto 2)
- con una velocidad de 127 (y puerto 2)

midiout funciona de forma equivalente.

5.8. Resumen

- PD puede gestionar simultáneamente hasta 4 puertos de entrada MIDI y 4 de salida
- Existen objetos PD para cada uno de los mensajes MIDI típicos, en sendas versiones de entrada y salida
- Los objetos de entrada admiten argumentos (e.g. canal) que permiten filtrar la información que se recibe

[4] De hecho, los mensajes de Sysex no están asociados a ningún canal MIDI

6. MIDI (2) y objetos de control de tiempo

6.1. Introducción

Los objetos que hemos visto hasta ahora permiten procesar MIDI produciendo unas salidas en función de las entradas recibidas.

En este capítulo veremos otras formas de generar salidas MIDI. Para ello utilizaremos:

- objeto de reloj con repeticiones (***metro***)
- objeto de generación de valores aleatorios (***random***)
- objeto de formateo de notas MIDI (***makenote***)

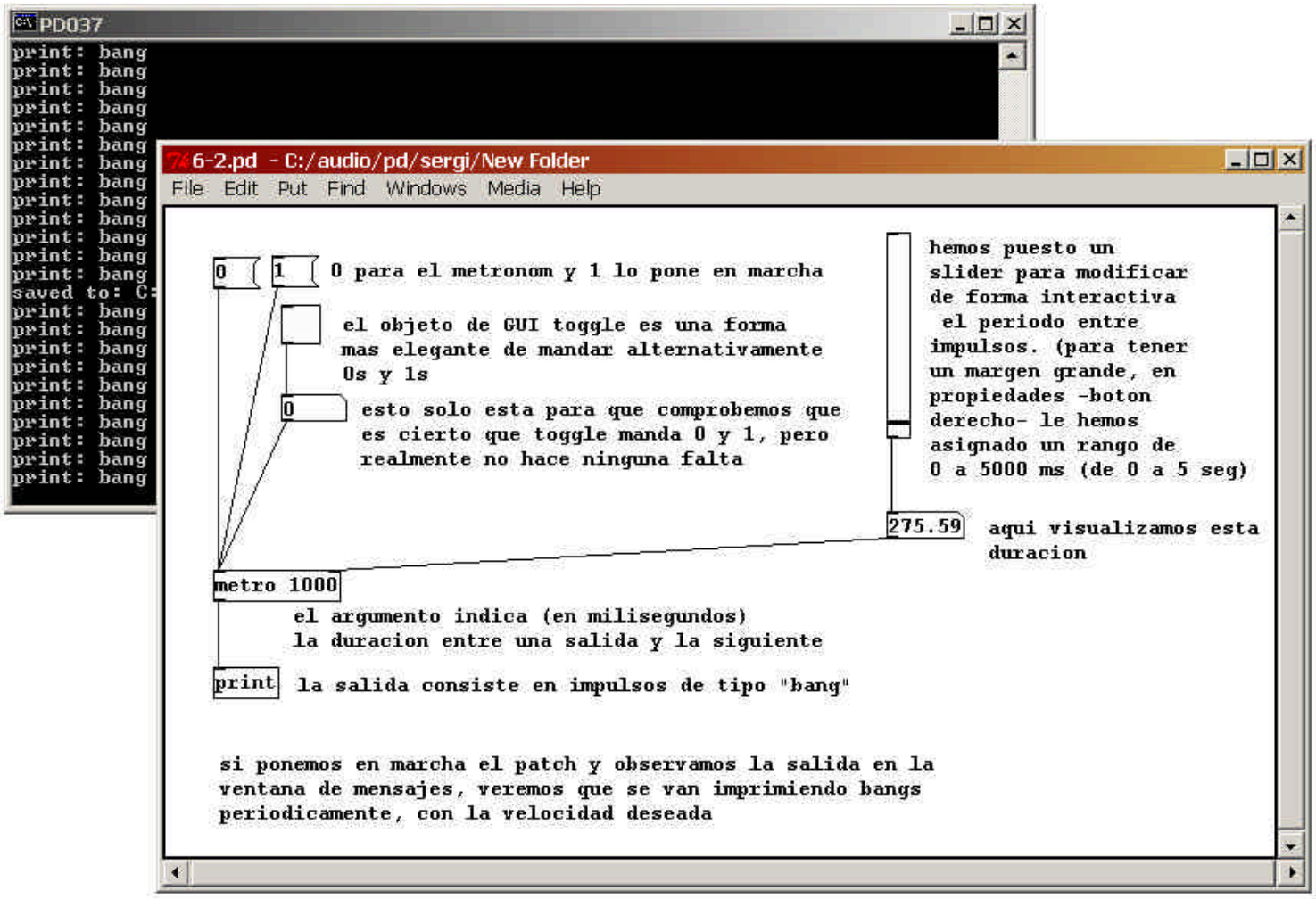
6.2. Control de repeticiones con el objeto metro

En muchas ocasiones, musicales o no, deseamos

- que una acción se realice de forma periódica
- que determinados procesos avancen solos sin necesidad de que les “empujemos” con inputs
- ….

En todos estos casos puede sernos útil el objeto ***metro***, que funciona como un metrónomo.

Si en PD bang es la señal que se utiliza para indicar “haz algo”, es lógico que el objeto metro sea un “lanzador periódico de bangs”



Primera propuesta:

Intentar convertir esta repetición de bangs impresos, en un verdadero metrónomo.

Ayuda:

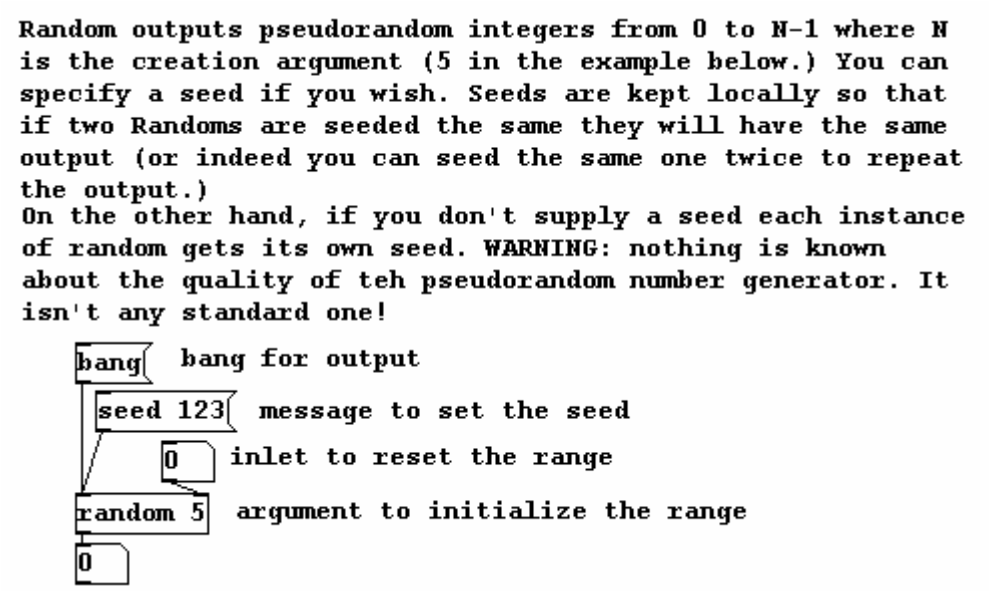
Los sonidos MIDI de percusión son más sencillos de gestionar porqué (de acuerdo con la normativa General MIDI) no utilizan NOTE OFF. Es decir, se activan con un NOTE ON y no necesitan ni esperan NOTE OFF. Con otro tipo de sonidos, incluidos los sonidos que no se sostienen (como los del piano), el sintetizador sí que espera normalmente un mensaje de NOTE OFF (o lo que es lo mismo, de NOTE ON con velocidad nula). Conviene recordar que la percusión General MIDI se produce siempre en el canal 10. Por ello, el metrónomo no será más que ir disparando un mensaje de NOTE ON por el canal 10.

El valor de la altura determinará el sonido de percusión que sonará, para lo que se puede consultar una tabla con las baterías General MIDI (cfg. Apéndice D3), mientras que la velocidad controlará la intensidad de este metrónomo. La solución se incluye en el apéndice A p. XX.

6.3. Modificación con el objeto random

El paso siguiente podría ser la incorporación de varios sonidos de percusión diferentes. Para ello, utilizaremos el objeto ***random***, que devuelve un valor aleatorio cada vez que recibe un bang.

A continuación se incluye el patch con la ayuda "oficial"



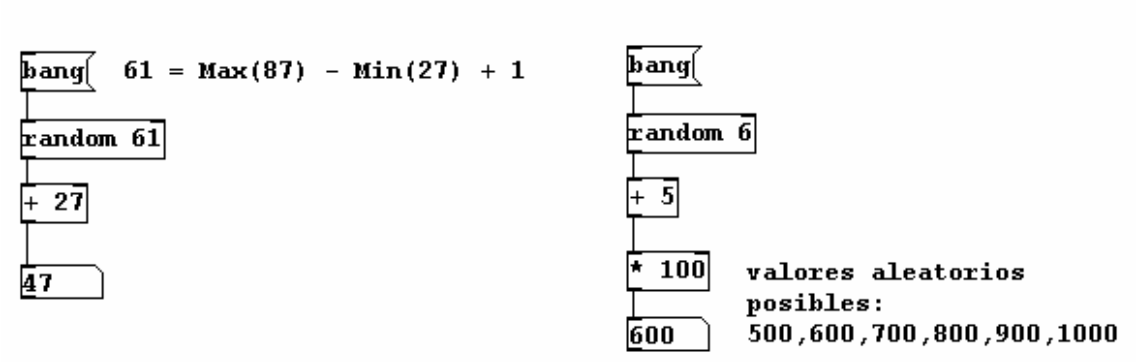
Algunos comentarios:

- El rango (de 0 a N-1) se puede indicar como argumento y se puede modificar mediante la entrada derecha.
- La semilla (seed) se utiliza para tener "mayores garantías" sobre la aleatoriedad de las salidas. Si no introducimos un valor de seed diferente cada vez, en cada ocasión que abramos PD, la secuencia de números aleatorios se repetirá. Más adelante veremos formas de generar semillas diferentes cada vez. De momento, "sobreviviremos" con esto.

Valores Acotados

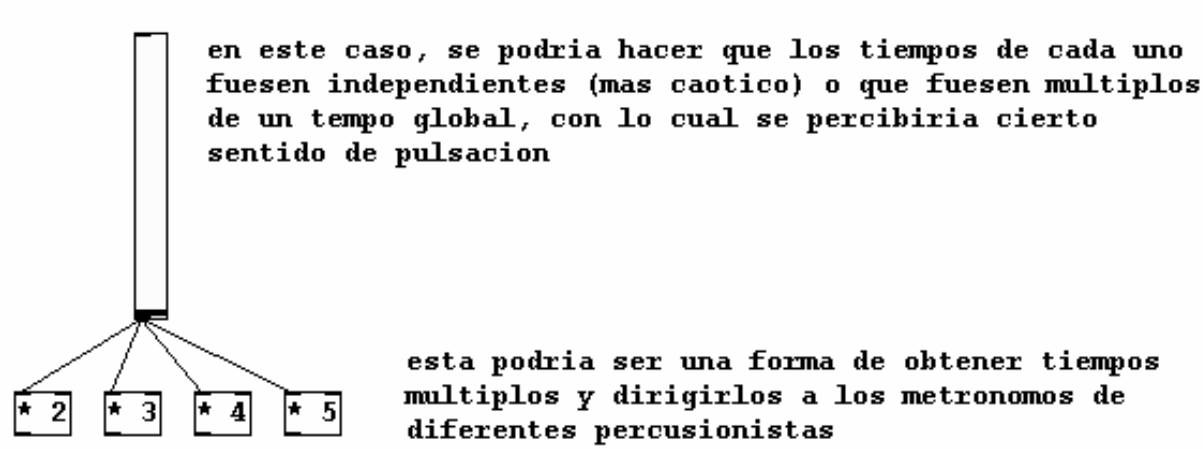
A menudo nos interesa obtener valores aleatorios acotados, no entre 0 y N, sinó entre M y N. Por ejemplo, los sonidos de batería General MIDI, varían entre 27 y 87. ¿Cómo obtener valores en este rango? El siguiente patch lo muestra. Estudiarlo bien, hasta estar seguros de entenderlo, ya que posiblemente tengamos que estudiar esta estrategia muchas más veces.

En otros casos, tal vez no deseemos cualquier valor sino por ejemplo valores múltiplos de 100, como 100, 200, 300…. 1000, 1100… El segundo patch genera valores aleatorios entre 500 y 1000.



Con estas ideas es fácil adaptar un generador aleatorio de sonidos de percusión. Se podría hacer que la velocidad MIDI (i.e. la intensidad del sonido), también fuese variable, con un rango de por ejemplo entre 64 y 127. La solución se incluye en el apéndice A.

Una vez conseguido el patch, probablemente tenga más gracia si lo copiamos varias veces, de forma a tener una orquesta de percusionistas aleatorios en lugar de uno solo.



En este ejemplo estamos haciendo que un único slider mande valores diferentes, pero múltiplos de un valor original, como lapso de tiempo a los diferentes objetos metro.

Otras mejoras posibles

Más adelante veremos:

- como acotar más el ámbito de los sonidos (e.g. para que suenen sólo unos pocos como bombo, caja , toms y charles)
- como hacer que unos (e.g. bombo y caja) sonidos se reproduzcan más frecuentemente que otros
- como incluir silencios aleatorios
- como controlar el tempo pulsando un dedo periódicamente
- ….

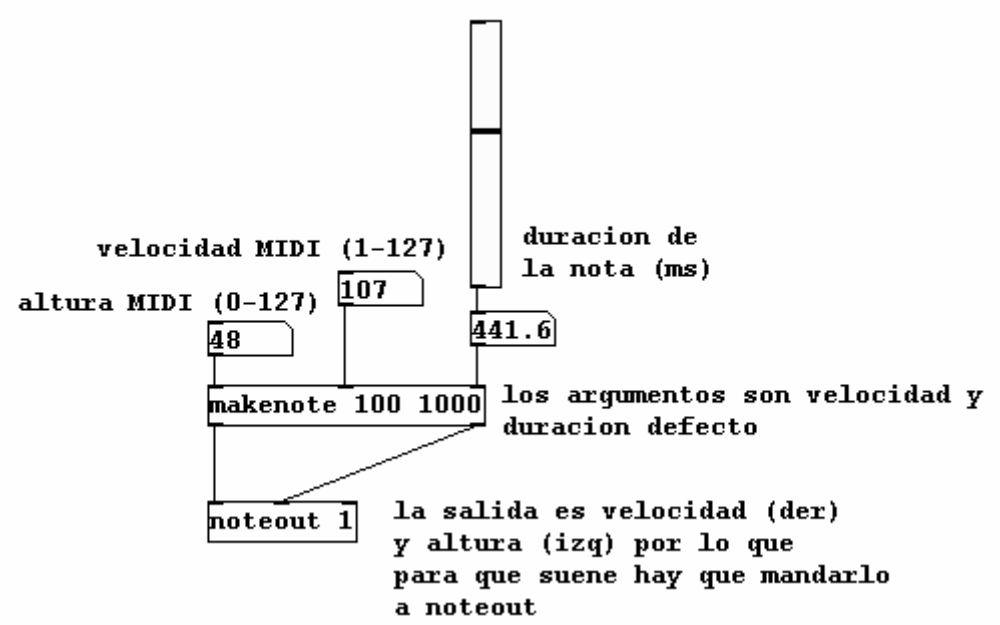
Todas estas pequeñas mejoras y experimentaciones no son demasiado complejas, pero necesitan objetos nuevos y estructuras lógicas que no hemos visto todavía. Si se te ocurren más ampliaciones partiendo de este mecanismo mínimo, apúntalas para ver si más adelante consigues implementarlas.

Ahora pasaremos a tratar sonidos que no sean de percusión para lo cual necesitaremos un nuevo objeto PD, ***makenote***.

6.4. Creación de notas con una duración determinada con makenote

Los sonidos MIDI de percusión (canal 10) tiene la peculiaridad de que no necesitan mensaje de final de nota. Los otros tipos de sonidos MIDI, aunque sean de piano o de instrumentos percutidos, esperan un mensaje de final de nota (mensaje de NOTE ON con velocidad 0). Para esto existe el objeto makenote que se utiliza para crear notas con una duración determinada. Eso es:

1. Generar un mensaje de NOTE ON con la altura y velocidad que le indiquemos
2. Generar otro mensaje con la misma altura pero velocidad 0, transcurridos los milisegundos indicados



Si nos ponemos a jugar con este patch tan sencillo, moviendo la casilla numérica de la izquierda (altura) empezaremos a mandar un montón de notas una detrás de otra. Funcionará mejor con un sonido de tipo piano, y será como recorrer rápidamente el teclado arriba y abajo con un dedo (a lo Chico Marx). Se puede ajustar la duración para que parezca más creíble (ni demasiado corta ni demasiado larga).

Ahora podemos integrar también esta estructura a los ejercicios anteriores de generación aleatoria con ***metro*** y ***random***.

A tener en cuenta:

- **La tesitura del instrumento**. Si queremos generar melodías aleatorias normalmente no querremos que sea en todo el rango MIDI (de 0 a 127) sino en un registro que suene bien con el instrumento elegido. En el apéndice D1 se incluye una tabla con las notas MIDI asociadas a cada nota así como su frecuencia en Herzios. En este sentido, es también importante recordar que todas las notas múltiplos de 12 corresponden a notas Do (e.g. 0,12,24,36,48,60,72….), siendo 60 el valor asociado con el Do central.

- Si el tempo viene indicado por el valor en ms que entra en ***metro***, podremos hacer que la duración de la nota dependa también de este valor.

En el apéndice se muestra el patch resuelto. Este patch se puede ir sofisticando con más controles interactivos, más voces simultaneas, cambiando los instrumentos mediante pgmout, etc. pero para seguir avanzando ha llegado el momento de ver nuevos objetos. Esto no lo haremos hasta el capítulo 8. En el próximo (7) vamos a estudiar varias formas de organizar mejor nuestros patches.

Stripnote y spigot

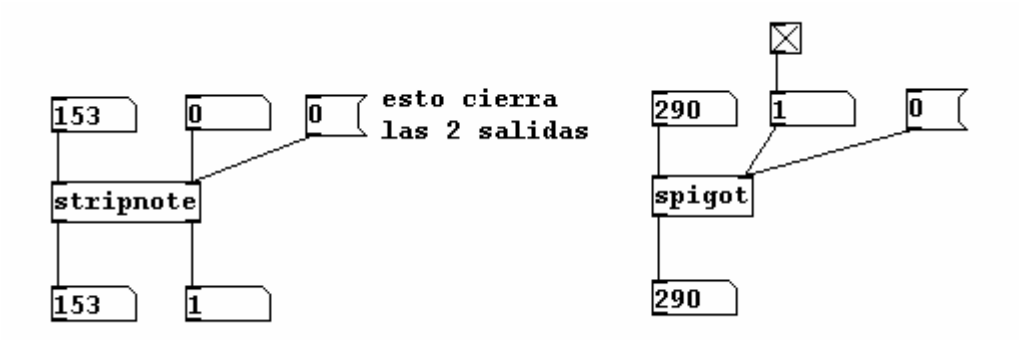
Si ***makenote*** se encarga de mandar mensajes de note off transcurrido un cierto tiempo, el objeto ***stripnote*** hace en cierta forma lo contrario: filtrar mensajes de note off, o lo que es lo mismo, no dejar pasar parejas de entradas cuyo valor derecho (velocidad) sea nulo.

Sus aplicaciones son menos esenciales que las de ***makenote***. Por ejemplo, si quisiésemos ir almacenando las alturas que utiliza determinado músico, mirando el ***notein***, está claro que los mensajes noteoffs no nos interesan. Con ***stripnote***, guardaríamos sólo los mensajes de inicio de nota.

NB. Dado que las parejas de altura velocidad pueden venir de cualquier sitio y no tienen porque estar sincronizadas, de una forma más general, podemos entender también este objeto, como uno que deja pasar dos valores, siempre que:

a) llegue un mensaje por la izquierda y
b) el último mensaje recibido por la derecha sea diferente de cero

Para comprenderlo mejor se recomienda jugar con un patch tan sencillo como el siguiente.



El objeto ***spigot*** es parecido pero sólo tiene una salida (la izquierda); la entrada derecha (0 o no nula) sólo se usa para cerrar/abrir la salida izquierda.

6.5. Resumen

- ***metro*** permite mandar bangs de forma periódica para realizar acciones repetidas
- ***random*** genera números aleatorios en un rango controlable por el usuario
- ***makenote*** formatea mensajes de nota teniendo en cuenta la duración
- la combinación de estos 3 objetos con los objetos MIDI estudiados en el capítulo anterior abre ya muchas posibilidades de composición algorítmica
- ***stripnote*** y ***spigot*** permiten filtrar mensajes de noteoff

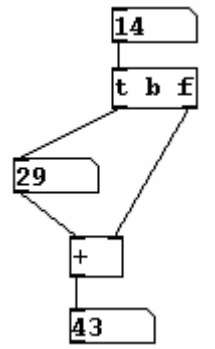
7. Cosmética y eficiencia: entradas, salidas, encapsulamiento de objetos, abstractions, *send* y *receive*…

7.1. Encapsulamiento y abstractions

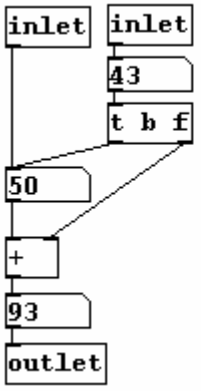
Conforme van creciendo nuestros patches, estos se hacen cada vez menos manejables. Por otra parte, en ocasiones puede ser interesante aprovechar varias veces un fragmento ya escrito, como por ejemplo copiar varias voces algorítmicas para obtener una mayor polifonía. Para estos casos, y para otros existe en PD el concepto de abstracción, similar al concepto de función en lenguajes más convencionales.

Inlets y outlets

En un lenguaje tradicional, una función es un fragmento de código que puede tener varias entradas (parámetros) y hasta una salida (retorno de la función). En PD, una abstracción puede tener varias entradas y también varias salidas.



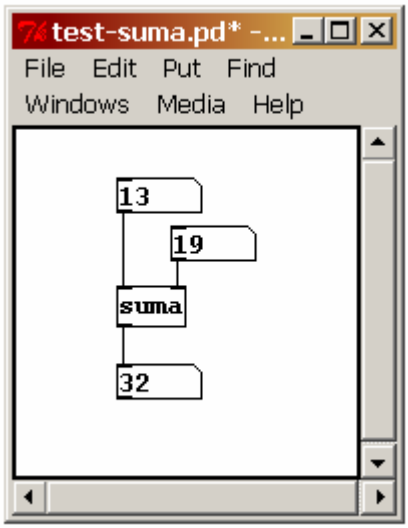
Este ejemplo, sacado de la sección 3.4, construía una suma en la que cualquiera de sus dos entradas producía una salida. Vamos a construir un nuevo objeto suma partiendo de este ejemplo, añadiendo los objetos ***inlet*** y ***outlet***.



anadimos los objetos inlet y outlet
colocamos inlet en cada una de las
entradas y outlet en cada una de las
salidas (aqui solo tenemos una)

1. Salvemos este patch con el nombre ***suma***
2. Creemos un nuevo patch vacío e incluyamos un nuevo objeto que llamaremos así, ***suma***. Habíamos visto (2.1) que cuando un objeto no existe, aparece un mensaje de error en la consola, pero este no es el caso ahora, y al escribir ***suma***, se crea un objeto con 2 entradas y una salida (que corresponden con el número de ***inlets*** y ***outlets*** que habíamos incorporado en nuestro objeto).
3. Si probamos su funcionamiento, comprobaremos que efectivamente es el objeto que habíamos creado.

4. Si clicamos sobre este objeto ***suma***, se abrirá el patch que hemos construido previamente.



En PD, estos objetos (como el suma del ejemplo) se denominan **abstractions**.

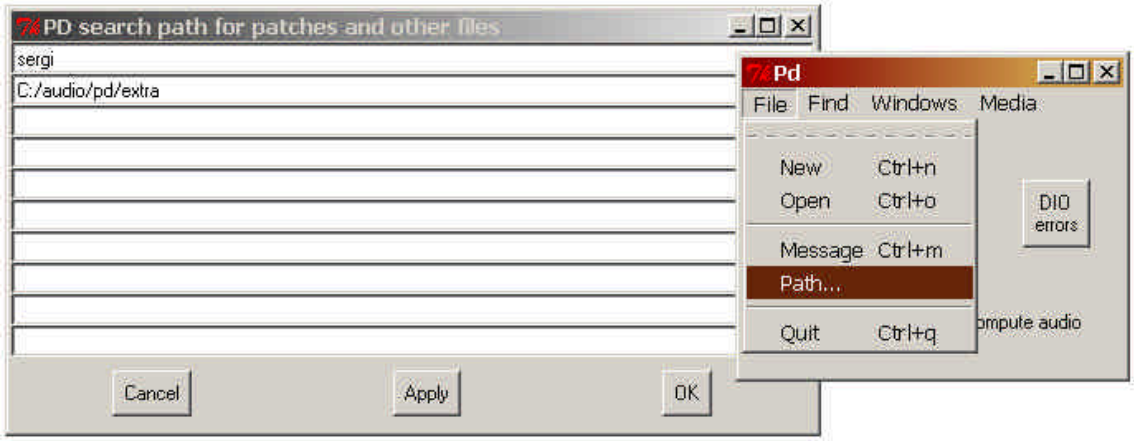
Las leyes básicas del encapsulamiento

Los objetos que creemos

- tendrán tantas entradas como objetos ***inlets***
- tendrán tantas salidas como objetos ***outlets***
- el orden de entradas (y de salidas) en el objeto se corresponderá con su posición espacial en el patch (es decir la entrada de la izquierda corresponderá con el objeto ***inlet*** situado más a la izquierda, etc.)

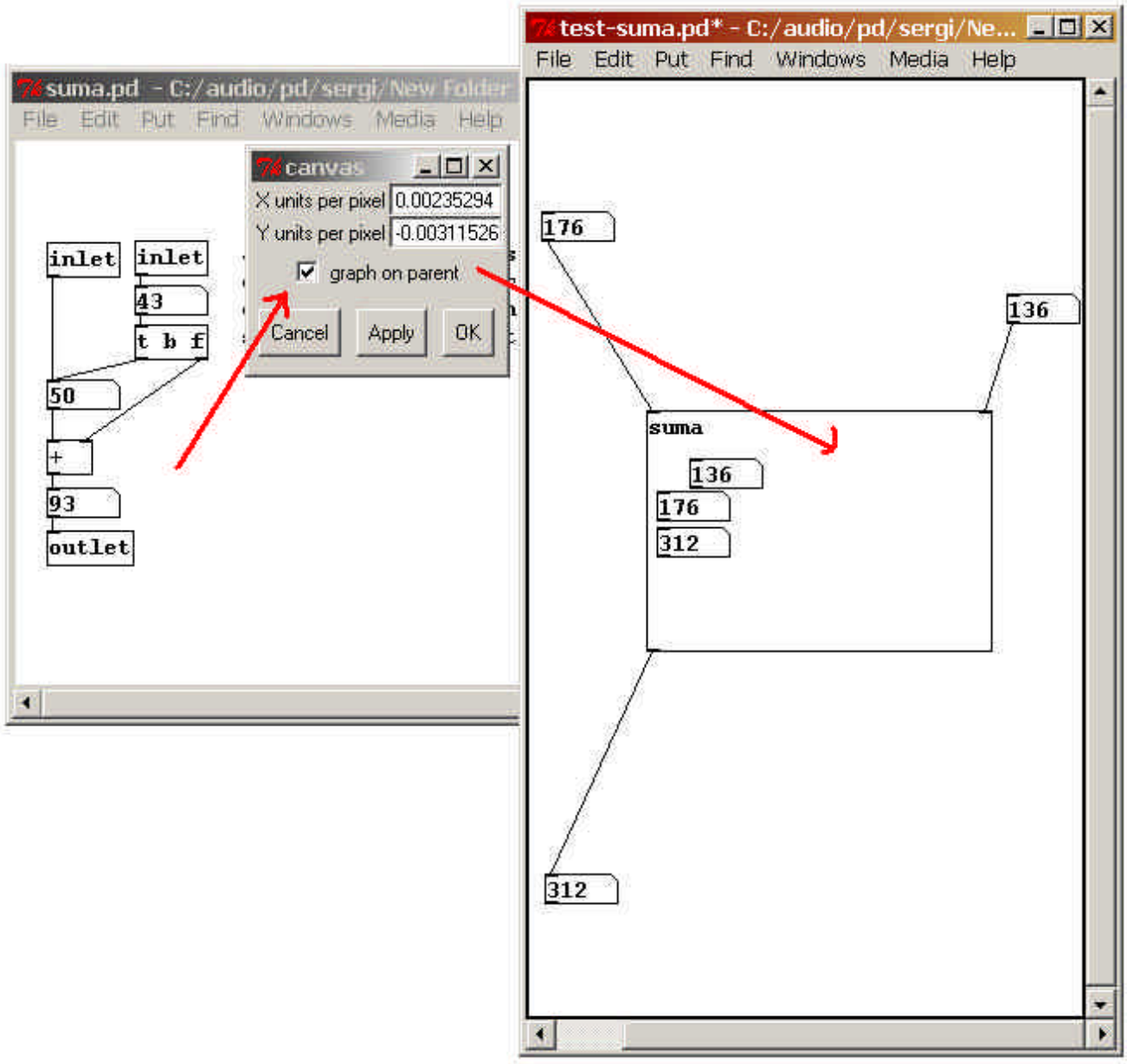
Para que cualquier patch encuentre los objetos que hemos creado nosotros, es necesario que se cumpla una de las dos condiciones siguientes:

- Que nuestro objeto esté salvado en unos de los directorios incluidos en el path de PD (ver figura siguiente)
- Que el objeto y el patch que lo debe contener se encuentren en el mismo directorio (lo cual solo podrá darse, si el patch que lo contiene ya ha sido bautizado y salvado, puesto que antes no está en ningún directorio).



Visualización de la interfaz en los objetos – *graph on parent*

Si cuando salvamos un objeto que va a ser reutilizado, seleccionamos en sus propiedades la opción “graph on parent”, cuando incluyamos este objeto, nos mostrará sus casillas numéricas, tal como se aprecia en la siguiente figura (aunque después puede resultar un poco “complicado” que estas casillas numéricas aparezcan donde nosotros queramos...).

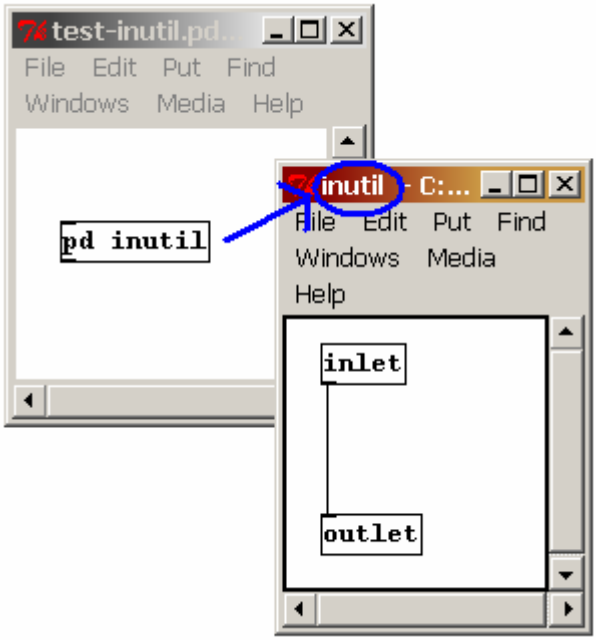


Rendimiento en las abstractions

La ventana de un subptach puede estar cerrada o abierta (al clicar sobre el objeto). Si no utilizamos *"graph on parent"* los números y otros objetos GUI de un subpatch que tiene la ventana cerrada no consumen recursos (i.e. el programa no debería ir más lento por tener casillas numéricas que no se visualizan).

Subpatches

Los subptaches son una versión reducida de las abstractions. Permiten hacer que nuestro programa sea más limpio pero no nos permite reutilizar objetos. Los subpatches son subventanas de la ventana principal.



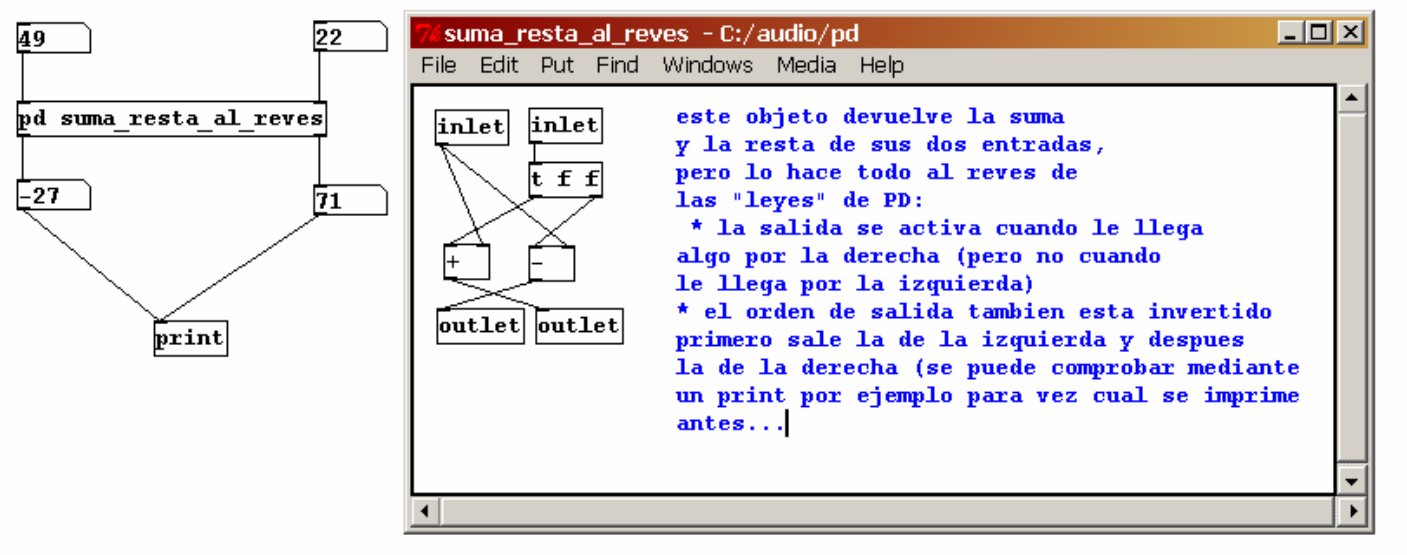
En una caja de objeto vacía escribamos un nombre de objeto que comience por el prefijo pd, seguido de una espacio y de un nombre de objeto que todavía no exista. Cuando terminemos de escribir el nombre, se abrirá una ventana vacía, en la podremos colocar los objetos que queramos, con sus inlets y sus outlets.

La diferencia entre un subptach y una abstracción es que el subpatch no se salva en ningún fichero independiente, sino que está incluido (incrustado) en el fichero que lo contiene. Por esta misma razón, no se puede reutilizar, ni siquiera en la misma ventana.
La ventaja es que el usuario no debe preocuparse de donde dejar el subptach (i.e. en que directorio, no borrarlo por descuido, etc.) para que el programa principal lo encuentre

Notas sobre órdenes y prioridades en las abstractions y los subptaches

- Hemos dicho varias veces que la mayoría de objetos PD se activan (disparan) cuando reciben un input por la izquierda.
- Hemos dicho también que en la mayoría de ellos, las salidas se producen ordenadamente de derecha a izquierda.

Ambas son normas de comportamiento de PD que una vez comprendidas y dominadas, facilitan enormemente la tarea de programación. Con los subpatches y las abstracciones es posible violar estas reglas; pero que sea posible no significa que sea recomendable.

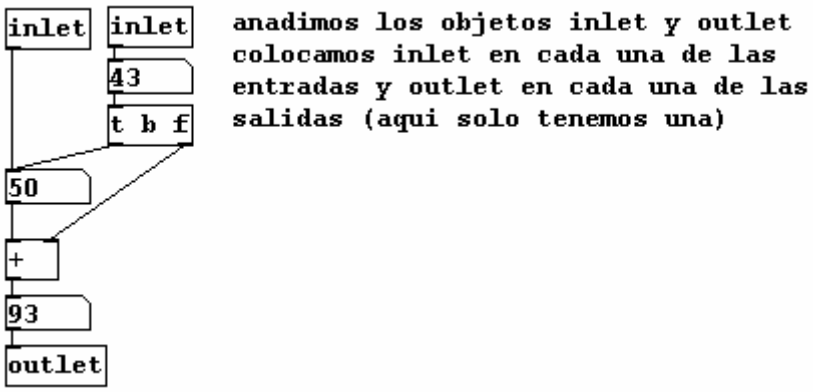


En este ejemplo todo funciona al revés de lo previsto, lo que hace que su uso sea confuso. En el apéndice se ofrece la versión corregida.

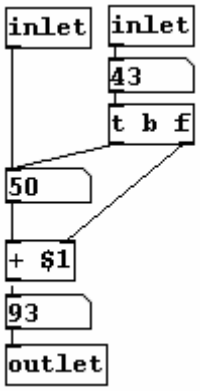
Argumentos en nuestros subpatches

A menudo es útil poder hacer que nuestros patches puedan utilizar argumentos, igual que la mayoría de objetos PD. Esto se consigue mediante el signo $ seguido de un valor numérico.

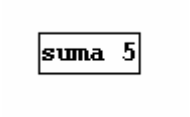
Retomemos el objeto suma que hemos creado al inicio de este capítulo.



Si queremos que nuestro objeto pueda tener un argumento de forma que, al igual que el operador + de PD, pueda tener un valor sumando por defecto, bastará con poner $1 en el lugar donde deberá ir este argumento.



Al haber incluído $1 como argumento en el objeto +, podremos añadir un argumento a nuestro objeto suma, y éste pasara a ocupar el valor del $1.



Ahora, hasta que no introduzcamos ningún valor por la derecha, cuando entre un valor por la izquierda, saldrá sumando en 5.

NB. Cuando introduzcamos un valor por la derecha, el 5 dejará de ser cierto, pero lo seguiremos viendo…

Ejercicio

Partiendo del ejemplo de random con valores acotados, desarrollado en 6.3, escribir dos patches variaciones de random de forma que:

- El primero admita los 2 argumentos, valor mínimo y valor máximo
- El segundo admita los 2 argumentos, valor central y rango, de forma que si estos valiesen por ejemplo 60 y 5, sacaría valores comprendidos entre 55 y 65.

Ayuda: será conveniente utilizar también el objeto ***loadbang***.

Es útil que estos argumentos se puedan modificar mediante entradas, tal como sucede con la mayoría de objetos PD. De esta forma, nuestros patches random podrían tener 3 entradas, de izquierda a derecha: un bang para disparar una salida, el valor mínimo (o central) y el valor máximo (o rango).

7.2. Comunicación sin cables: send y receive

Los objetos send y receive

En algunas ocasiones, las posiciones de los objetos en una ventana hacen que sea difícil mantener un cierto orden y evitar un confuso amasijo de cables. En estas situaciones, puede resultar muy conveniente utilizar los objetos ***send*** y ***receive***, que se pueden escribir de forma abreviada como s y r respectivamente.

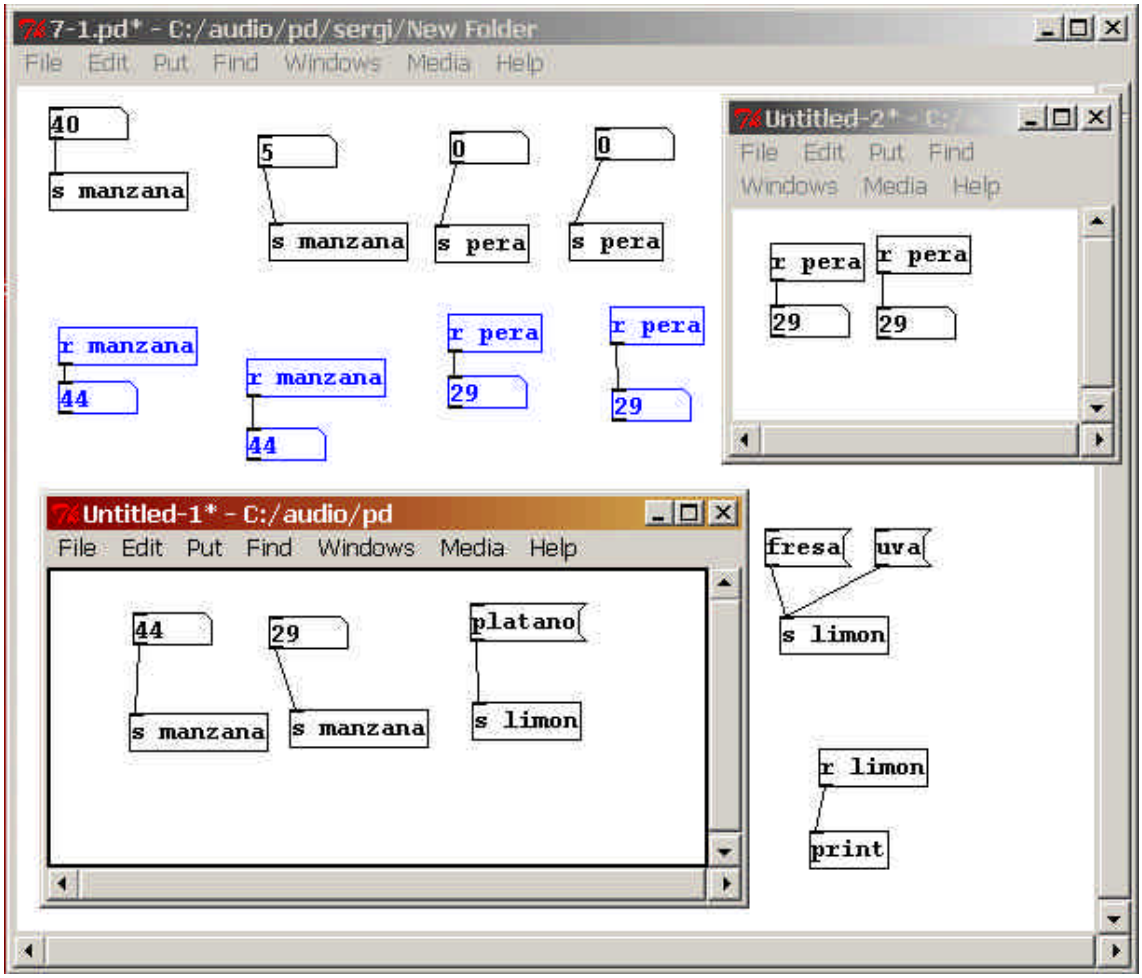
- Ambos objetos necesitan un parámetro, que se podría asimilar al nombre de una variable en un lenguaje de programación convencional[5]
- El objeto ***send*** tiene una entrada, y el objeto ***receive*** tiene una salida

[5] En realidad es más el nombre de un canal de comunicación que una variable…

- Cuando un ***send*** recibe algo (e.g. un valor numérico, un mensaje…) inmediatamente lo manda a todos los ***receives*** que tengan el mismo argumento
- Puede haber un número cualquiera de ***sends*** con el mismo argumento (obviamente, también con argumentos diferentes)
- Puede haber un número cualquiera de ***sends*** con el mismo argumento (obviamente, también con argumentos diferentes)

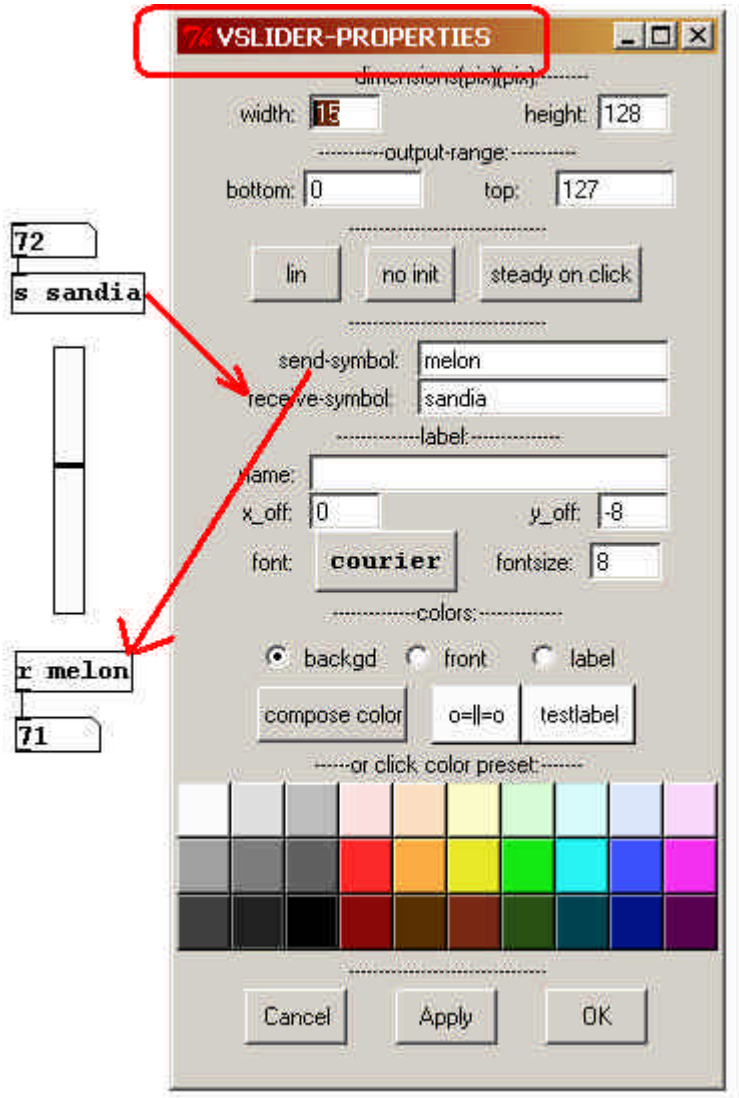
Esta comunicación se realiza incluso entre objetos de ventanas PD diferentes!!!

El siguiente ejemplo muestra tres ventanas PD totalmente independientes, varios objetos ***send***, varios ***receive*** y tres argumentos diferentes (manzana, pera, limon). Podemos pensar en estos tres argumentos como tres variables globales, o tres canales de comunicación. Cuando cualquier send "manzana" manda un valor, lo reciben inmediatamente todos los receive "manzana".



Send y *receive* en objetos GUI

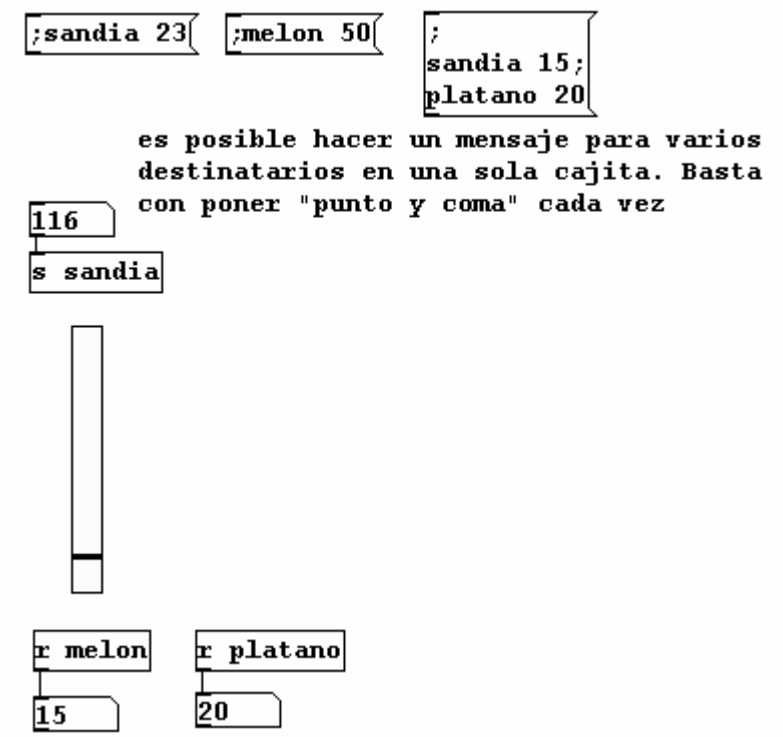
La mayoría de objetos de tipo GUI incorporan los objetos ***send*** y ***receive***, lo que permite diseñar interfaces GUI sin cables[6].



send y *receive* con mensajes

En 2.1 vimos los mensajes. Los mensajes también pueden incluir destinatarios (nombres de objetos receive) lo que permite su uso sin cables. Para indicar el destinatario de un mensaje, basta con anteponer ; (punto y coma) al nombre del destinatario. Así, cuando un mensaje comienza por ; la primera palabra que le sigue no es el mensaje en si, sino su destinatario. La palabra siguiente sí que es el mensaje.

[6] En MAX la cosmética en los GUI se basa sobretodo en la posibilidad de hacer invisibles las conexiones que se desee. Aunque esto no es posible en PD, la integración de ***send*** y ***receive*** en los GUI cumple esta misma función con mayor flexibilidad.

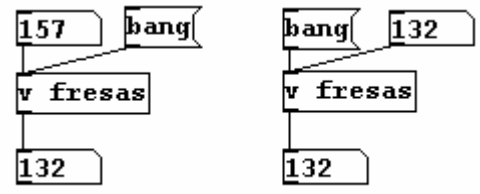


7.3. Integers, floats y value

value

Una variante del ***send*** y ***receive*** es el objeto ***value*** (o ***v*** en abreviado). Igual que ***send*** y ***receive*** utiliza un nombre como argumento. A diferencia de los anteriores, la entrada de ***value*** no sale automáticamente por la salida. Para que la salida se produzca se necesita un bang en la entrada.

Para captar bien su funcionamiento, conviene ejecutar el siguiente ejemplo.

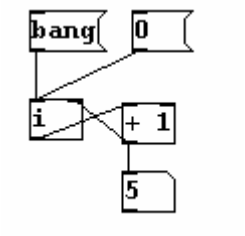


NB. Al igual que con ***send*** y ***receive***, los objetos ***value*** se comunican aunque estén en ventanas diferentes.

integer y float

- ***integer*** y ***float*** se utilizan para almacenar valores.
- No utilizan argumentos, por lo que NO funcionan como variables compartidas (a diferencia de ***send/receive*** o ***value***)
- Un valor numérico por la entrada derecha almacena este valor (y no lo dispara)
- Un valor numérico por la entrada izquierda almacena este valor y lo dispara inmediatamente
- Un bang por la entrada izquierda, dispara el valor almacenado

El siguiente patch utiliza un objeto integer (abreviado como i) para construir un contador. Este patch es breve y aparentemente sencillo, pero no trivial. Es además de suma importancia, ya que los contadores se utilizan en multitud de casos.



Cada vez que se pulsa un bang, el contador se incrementa en uno. El mensaje 0 inicializa el contador.

¿Como funciona?

1. Si inicialmente mandamos un 0 a i, dado que esta entrada se produce por la izquierda, inmediatamente sale el mismo 0.
2. Este 0 llega al objeto + 1, con lo cual, la salida es la entrada incrementada en 1, es decir 1
3. Este valor entra de nuevo en el objeto i, pero esta vez por la derecha, con lo cual no sale, pero se almacena
4. Cuando i recibe un bang, este objeto dispara el valor almacenado, es decir 1, que tras pasar por + 1, entra de nuevo en i con un valor 2
5. Este valor no sale y se almacena hasta que volvamos a mandar un bang
6. Etc, etc…

Con una estructura de este tipo es fácil construir por ejemplo un reloj. Se propone como ejercicio un cronómetro sencillo (que cuente segundos).

8. Control de Flujo

8.1. Introducción

Hasta el momento, no hemos visto ninguno de los elementos que hacen que un entorno más o menos flexible se convierta en un verdadero lenguaje de programación. Nos referimos a las estructuras de control que permiten tomar decisiones, hacer que una determinada parte de un programa se ejecute sólo cuando se cumplan las condiciones propicias.

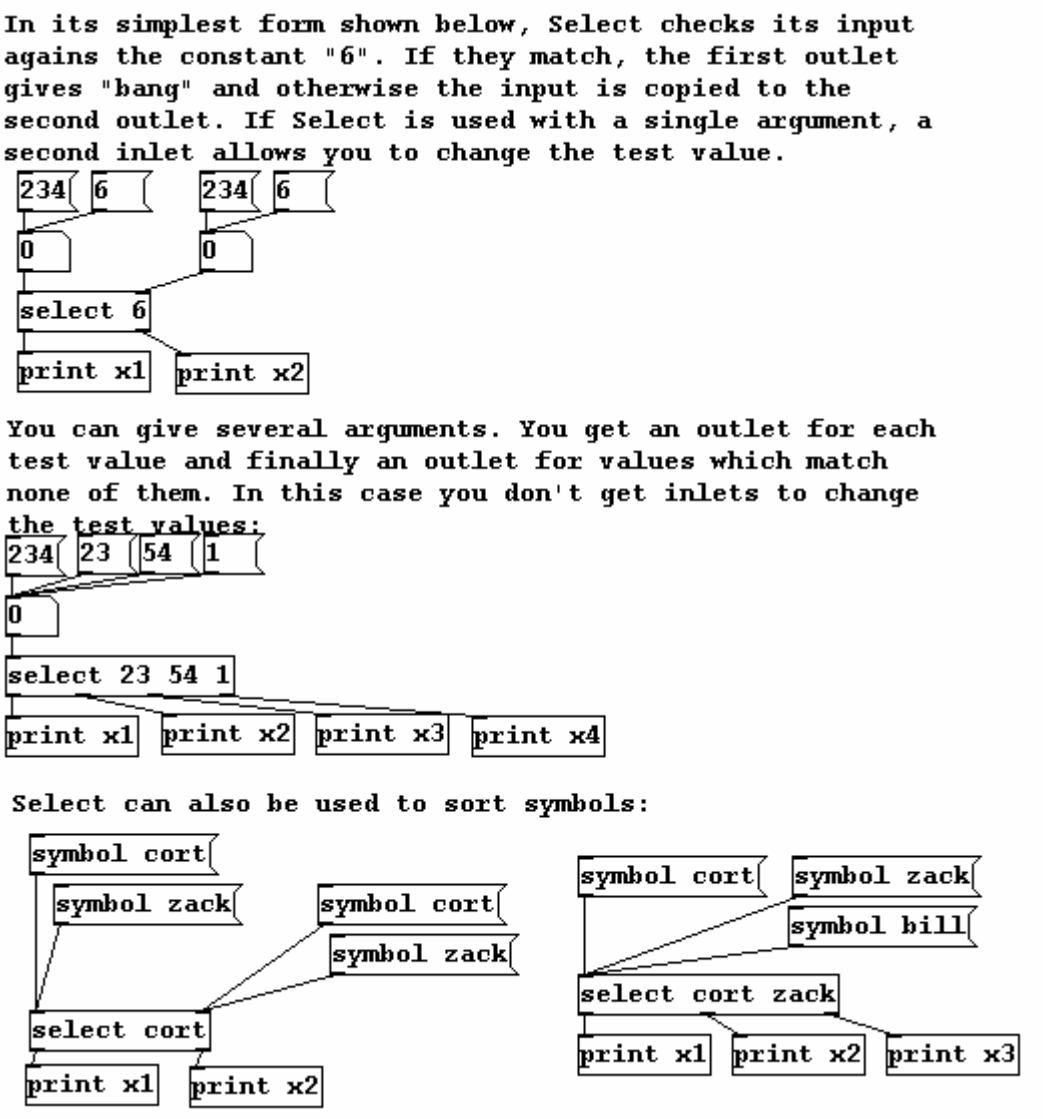
En PD no existen el if o el while, lo que puede despistar inicialmente a aquellos que estén acostumbrados a programas en otros lenguajes más tradicionales. En este capítulo veremos los objetos *select*, *moses*, *route*, las listas, los objetos *pack*, *unpack* y *swap*, así como los operadores lógicos y relacionales, junto con un repaso de los objetos operadores aritméticos.

8.2. sel

- el objeto sel (o select) tiene una única entrada (en algunos casos dos, tal como se indica más adelante)
- admite un nuevo variable de argumentos
- presenta tantas salidas como argumentos + 1
- si una entrada coincide con el argumento número N, select manda un bang por la salida número N
- si una entrada no coincide con ninguno de los argumentos, select saca esta entrada inalterada por la salida N+1 (es decir por la de más a la derecha)

- los argumentos de sel pueden ser de cualquier tipo (valores numéricos, palabras, etc.), por lo que sus también podrán serlo sus entradas.
- cuando sel tiene un único argumento, presenta además una entrada adicional (por la derecha), que permite modificar el valor a comparar.

Incluimos a continuación el help de PD correspondiente a este objeto, ya que ofrece todas las posibilidades simples que hemos comentado. Se recomienda abrir este mismo patch (seleccionando help desde el propio objeto sel) y probar todas estas configuraciones. Una vez entendido su funcionamiento pasaremos a explicar su uso y posibilidades en casos concretos.



sel como switch

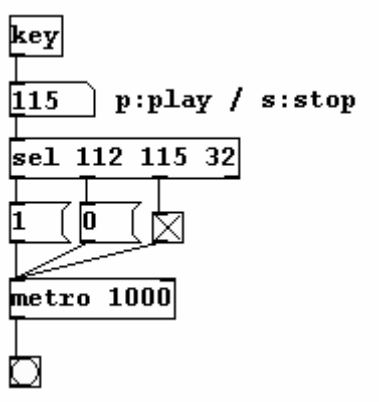
Este objeto funciona como la estructura *switch* disponible en la mayoría de lenguajes de programación tradicionales (aunque su nombre varía dependiendo del lenguaje). A continuación se muestra un ejemplo que podría corresponder a C, C++, Java, Javascript, etc.

```
switch (iValue)
{
     case 0:    wPeriod2 = 1; break;
     case 127:  wPeriod2 = wPeriod-1; break;
     default:   wPeriod2 = wPeriod * iValue / 127; break;
}
```

- iValue sería la entrada de sel
- 0 y 127 sus argumentos

8.3. Ejemplos de uso de sel

Utilizar el teclado ASCII para ejecutar acciones diversas (objeto key)

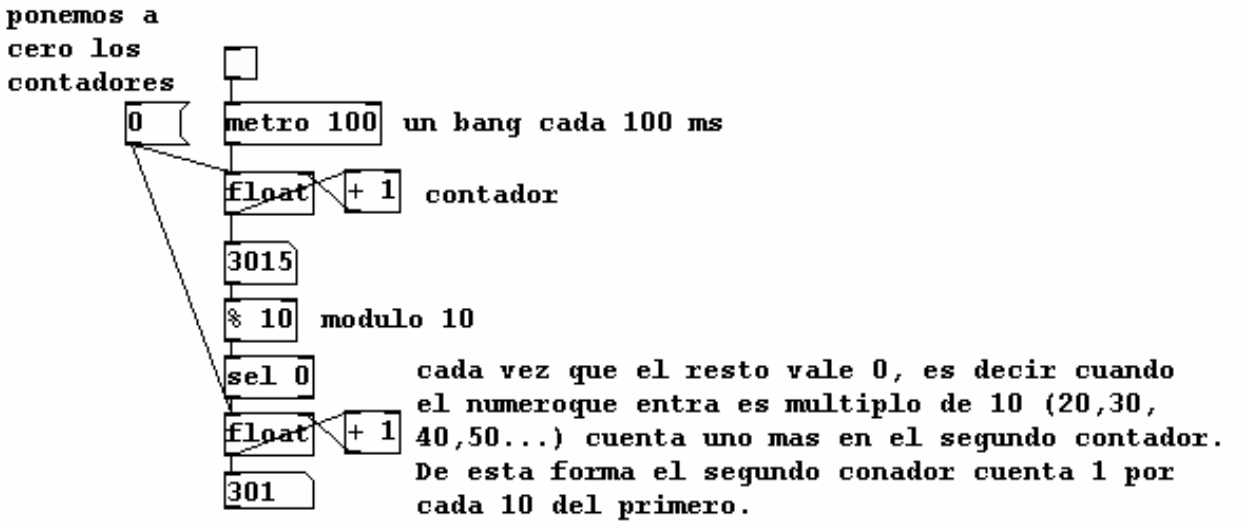


Cuando se pulsa una tecla del ordenador, el objeto key devuelve el código numérico (ASCII) de esta tecla.
En este ejemplo, la letra p (código 112) pone en marcha el metrónomo, la tecla s (115) lo para, y la tecla PAUSA (32) lo conmuta.

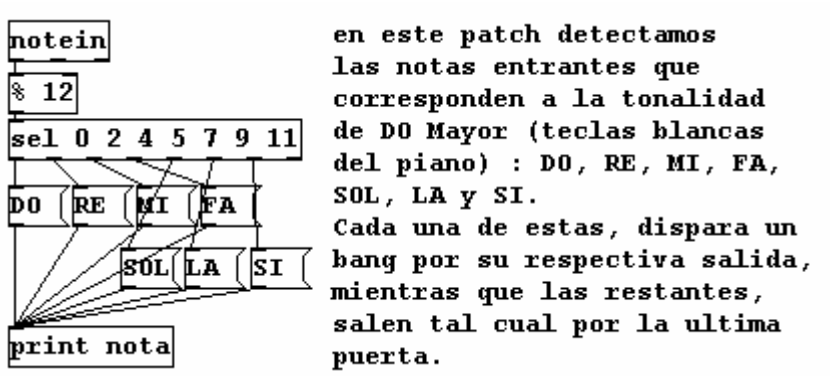
Sel y el operador módulo (%)

El operador módulo, presente en muchos lenguajes de programación, devuelve el resto de la división entera. Esto, que puede sonar raro para los que no lo hayan visto nunca, es en realidad muy sencillo. Por ejemplo el resto de la división entera por 10, podrá ser 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ó 10. Esto significa que cualquier entrada numérica, módulo 10, dará un resultado entre 0 y 9.

El siguiente ejemplo, módulo 10 se usa para contar los segundos. Dado que el metrónomo del ejemplo va a 100 ms, cada 10 valores corresponderán a un segundo.



En el siguiente ejemplo, se utiliza el operador módulo, en este caso con un valor 12, para distinguir entre los doce semitonos.



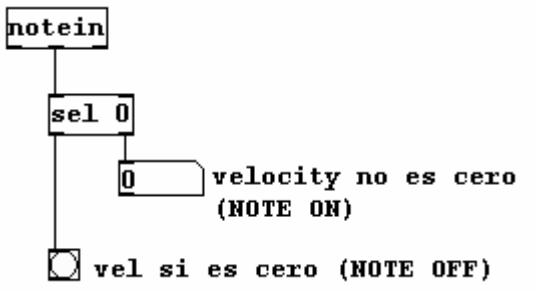
- Las notas DO en notación MIDI corresponden a los valores 0, 12, 24, etc. Su resto de dividir por 12 es siempre 0
- Las notas DO#, a 1, 13, 25,… Su resto de dividir por 12 es siempre 1
- Las notas RE a 2, 14, 26… Su resto es siempre 2
- …
- Las notas SI a 11, 23, 35… Su resto es siempre 11

Dicho al revés:
Si el resto, al dividir por 12 un nota MIDI cualquiera, vale

- 0: la nota es un DO
- 1: la nota es un DO# (o REb)
- 2: la nota es un RE
- ….
- 11: la nota es un SI

Este ejemplo será el punto de partida para más adelante realizar un armonizador que cree acordes a partir de notas entradas desde teclado.

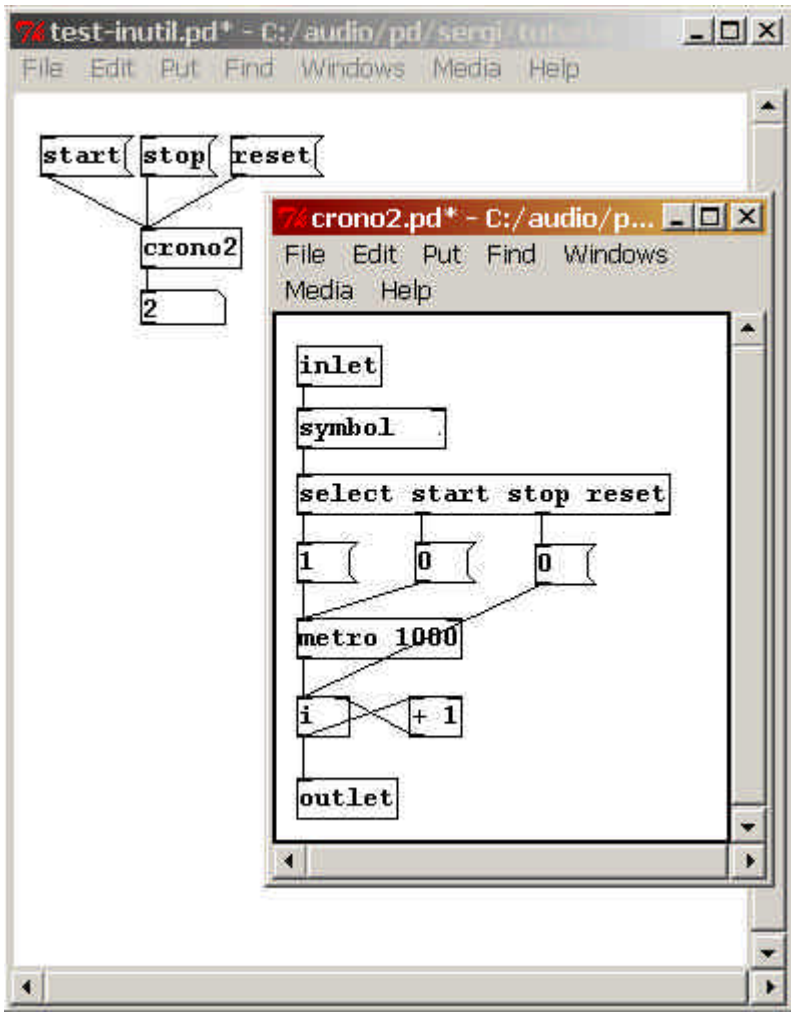
Distinguir entre note on y note off



Select con argumentos de texto (el objeto *symbol*)

Utilizar ***select*** con argumentos de texto, permite hacer que nuestros objetos propios admitan mensajes de tipo comando. En el siguiente ejemplo se modifica el objeto crono que ya creamos en 7.3, para que ahora funcione mediante los mensajes start, stop y reset.
Para poder hacerlo, debemos introducir un nuevo concepto que a partir de ahora, iremos viendo de vez en cuando, el *símbolo*.

Hasta ahora hemos tratado el texto, sólo dentro de mensajes. Esto funciona aproximadamente como si fueran constantes. Hay veces en las que será necesario convertir el texto en variables. Para ello, deberemos convertir los mensajes en símbolos, lo que se consigue simplemente pasando el mensaje a través de un objeto ***symbol***. En este caso, antes de mandar un texto a select, debemos hacerlo pasar por ***symbol***, para convertirlo.



8.4. Ejemplos más complejos

Con ***sel*** podemos ya plantearnos ejercicios bastante más complejos. Mostramos algunos para que puedan ser estudiados con calma.

Generador de notas en una tonalidad

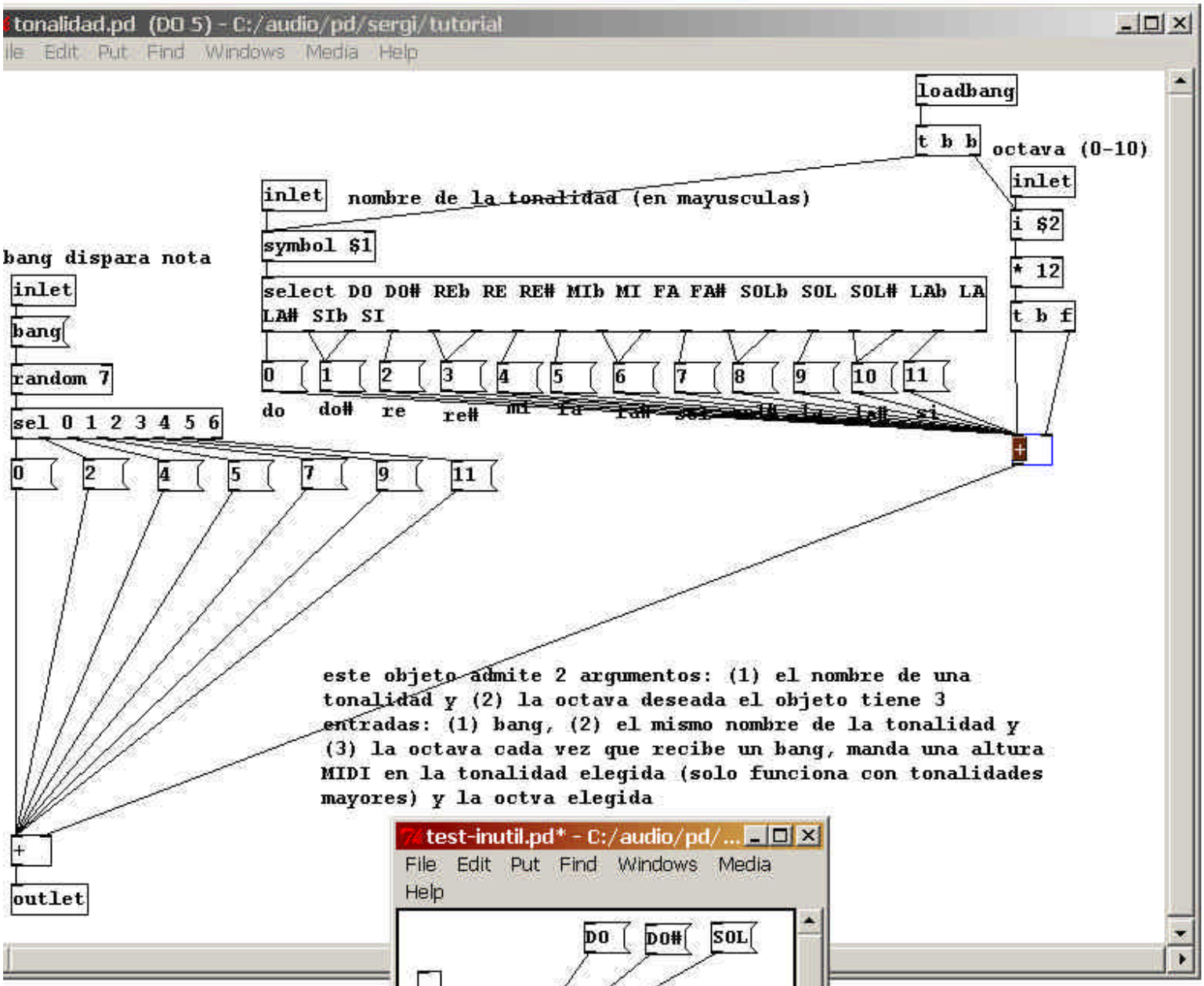
El siguiente ejemplo es más complejo. Muestra dos usos de ***select*** para conseguir un patch que tiene ya cierta utilidad práctica y cierta dificultad. Se trata de crear un patch que genere notas cada vez que recibe un bang, pero a diferencia de los anteriores, que generaban notas totalmente aleatorias, este patch generará notas en una tonalidad (mayor) y octava determinadas, que podrán indicarse tanto en la entrada, como como argumentos.

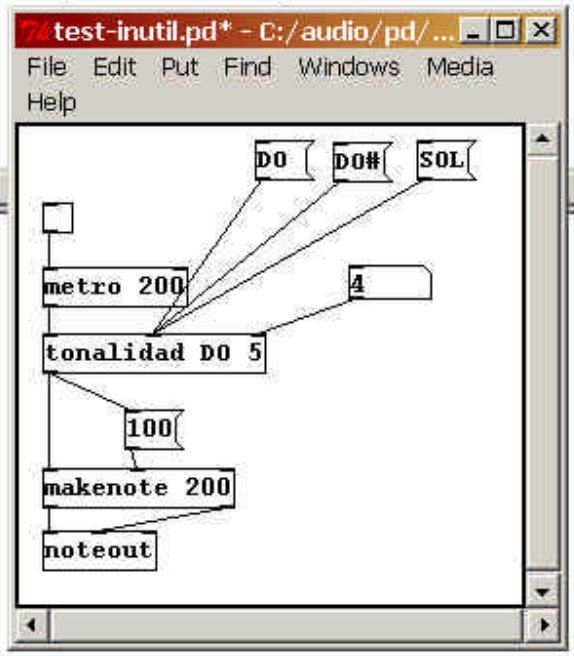
- La tonalidad se introducirá mediante texto (DO, DO#, REb…, etc).

  En este sentido, conviene tener en cuenta que PD, a diferencia de otros lenguajes, distingue entre mayúsculas y minúsculas, por lo que tonalidades como **do** o **Do** serán incorrectas en nuestro ejemplo (para que fuesen correctas, deberíamos tener en cuenta muchos más casos en el ***select*** que trata estos nombres).

- La octava se introducirá mediante un número entre 0 (notas MIDI 0 a 11) y 10, notas MIDI (120 a 127).

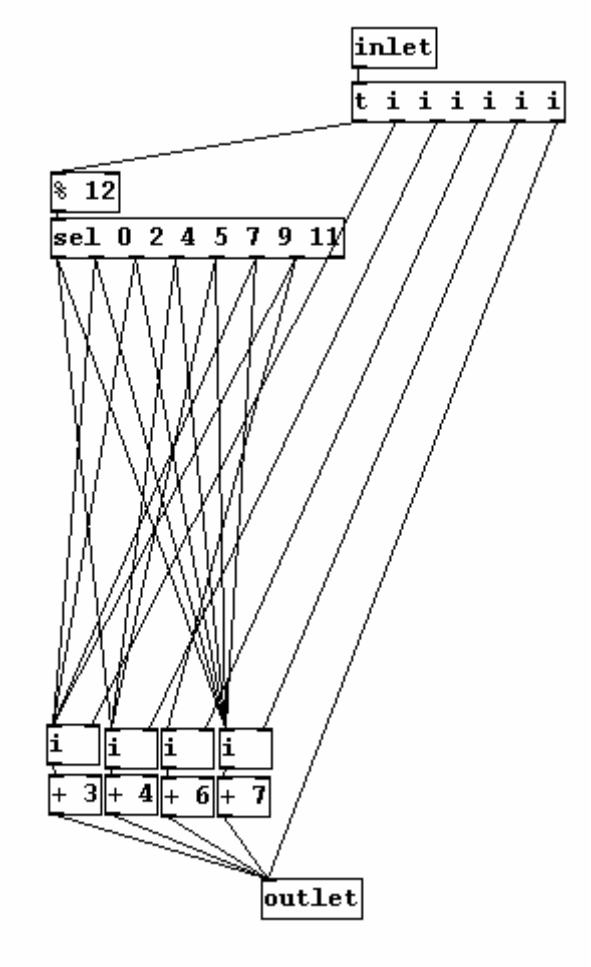
- Cada vez que el objeto reciba un bang por su entrada izquierda, saldrá la altura de una nota de esta octava y esta tonalidad (mayor).


tonalidad.pd (DO 5) - C:/audio/pd/sergi/tutorial
File Edit Put Find Windows Media Help
loadbang
t b b
octava (0-10)
inlet
nombre de la tonalidad (en mayusculas)
inlet
symbol $1
i $2
* 12
t b f
bang dispara nota
inlet
bang
random 7
select DO DO# REb RE RE# MIb MI FA FA# SOLb SOL SOL# LAb LA LA# SIb SI
sel 0 1 2 3 4 5 6
0 1 2 3 4 5 6 7 8 9 10 11
do do# re re# mi fa fa# sol sol# la la# si
0 2 4 5 7 9 11
este objeto admite 2 argumentos: (1) el nombre de una tonalidad y (2) la octava deseada el objeto tiene 3 entradas: (1) bang, (2) el mismo nombre de la tonalidad y (3) la octava cada vez que recibe un bang, manda una altura MIDI en la tonalidad elegida (solo funciona con tonalidades mayores) y la octva elegida
+
outlet


test-inutil.pd* - C:/audio/pd/...
File Edit Put Find Windows Media
Help
DO
DO#
SOL
metro 200
4
tonalidad DO 5
100
makenote 200
noteout

Armonizador

Este patch recibe una nota, y si esta corresponde a la tonalidad de DO Mayor, genera el acorde de cuatro notas correspondiente (que será mayor, menor o disminuido según la nota que le entre).

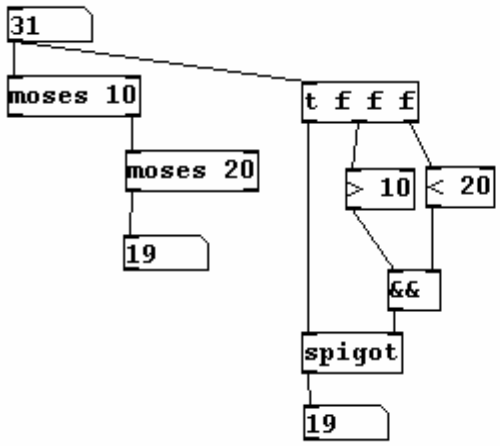


……

……..

condiciones y spigot (abrir/cerrar puerta)

2 formas de dejar pasar valores entre 10 y 20:

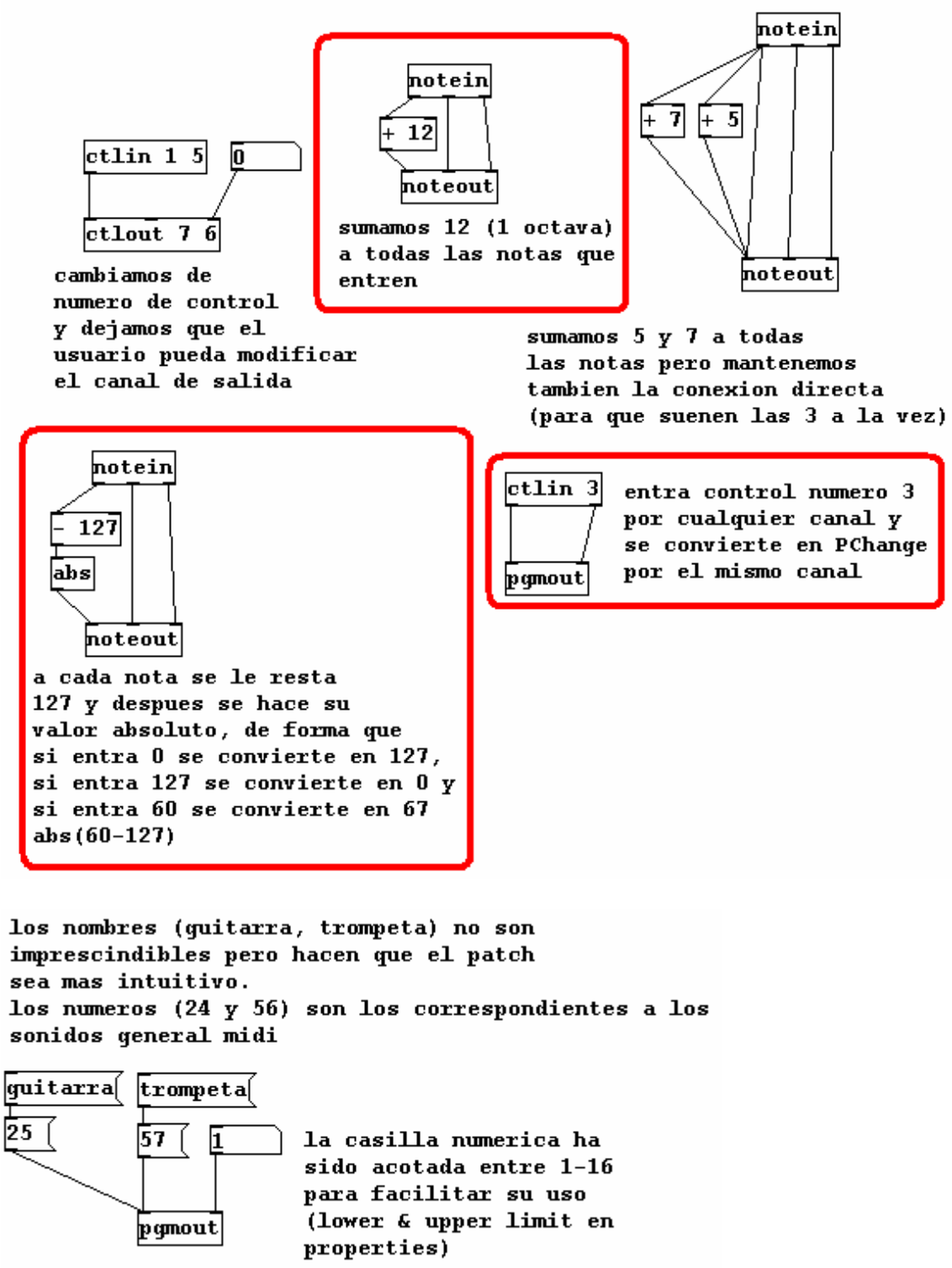


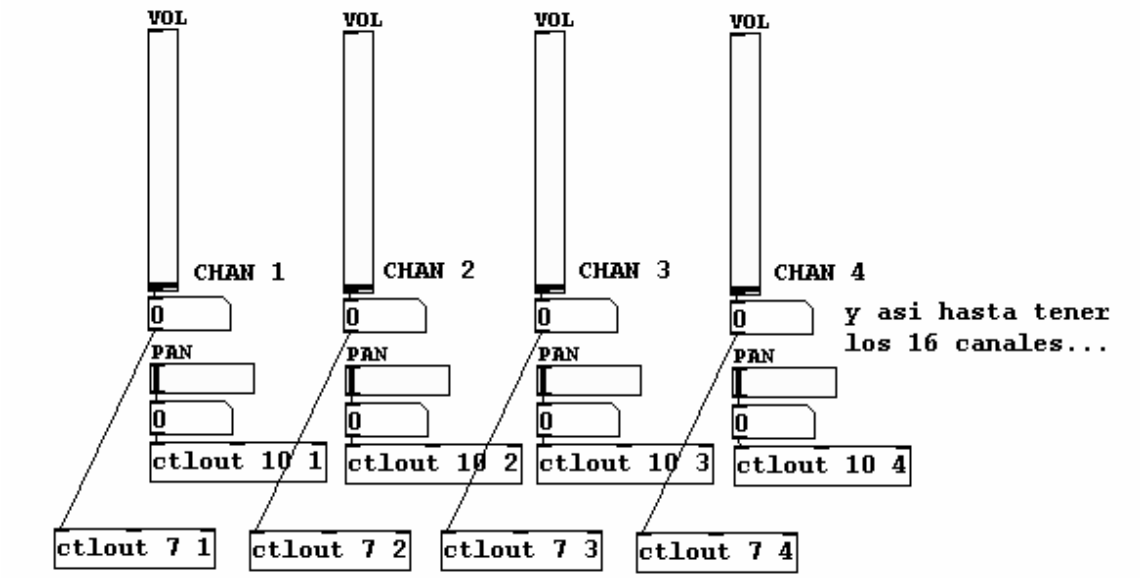
moses es mas sencillo, y mas flexible a la hora de trabajar con rangos, ya que seguimos teniendo disponibles las otras salidas (menor que 10 y mayor que 20)

spigot en combinacion con todo tipo de operadores logicos y relacionales es mas potente ya que permite evaluar cualquier tipo de condiciones (no solo de rango)

A. Corrección de ejercicios

5.5

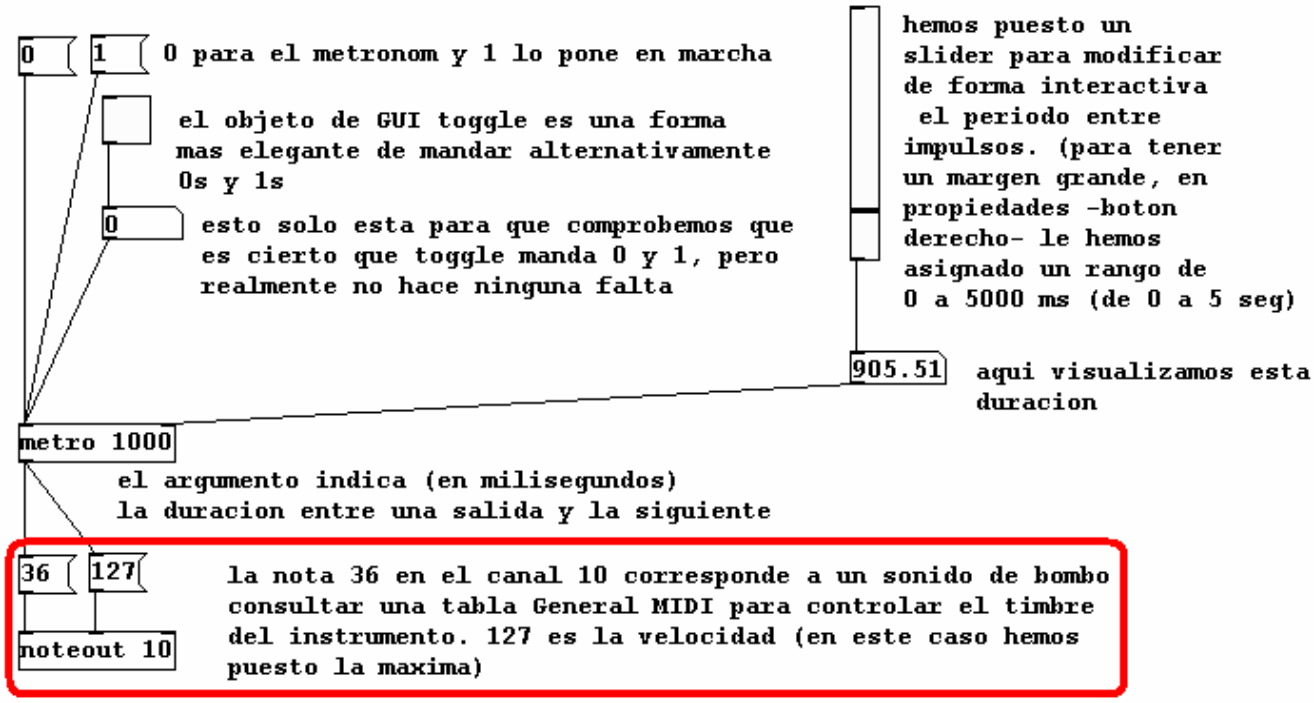




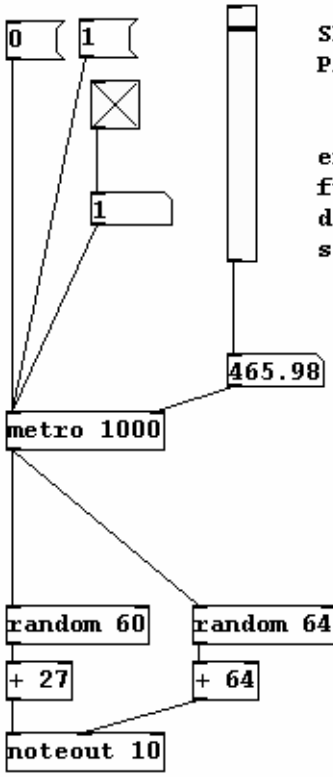
NB. esto se podria escribir de muchas formas, algunas mas complejas pero tambien mas elegantes. De momento nos quedamos con la mas sencilla.

Mas adelante, veremos tambien como hacer que los objetos que "podrian no verse en el GUI" (e.g. los objetos ctlout), realmente no aparezcan y el usuario vea solamente el GUI.

6.2


0
1
0 para el metronom y 1 lo pone en marcha
el objeto de GUI toggle es una forma mas elegante de mandar alternativamente 0s y 1s
0
esto solo esta para que comprobemos que es cierto que toggle manda 0 y 1, pero realmente no hace ninguna falta
hemos puesto un slider para modificar de forma interactiva el periodo entre impulsos. (para tener un margen grande, en propiedades -boton derecho- le hemos asignado un rango de 0 a 5000 ms (de 0 a 5 seg)
905.51
aqui visualizamos esta duracion
metro 1000
el argumento indica (en milisegundos) la duracion entre una salida y la siguiente
36
127
noteout 10
la nota 36 en el canal 10 corresponde a un sonido de bombo consultar una tabla General MIDI para controlar el timbre del instrumento. 127 es la velocidad (en este caso hemos puesto la maxima)

6.3

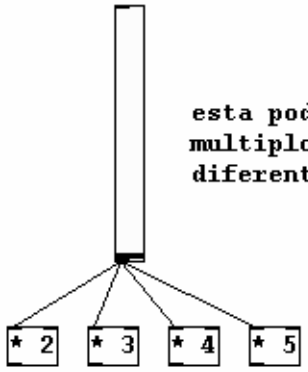


SE RECOMIENDA HACER VARIAS COPIAS DEL PATCH
PARA HACER SONAR A MAS PERCUSIONISTAS SIMULTANEOS!!!

en este caso, se podria hacer que los tiempos de cada uno fuesen independientes (mas caotico) o que fuesen multiplos de un tempo global, con lo cual se percibiria cierto sentido de pulsacion

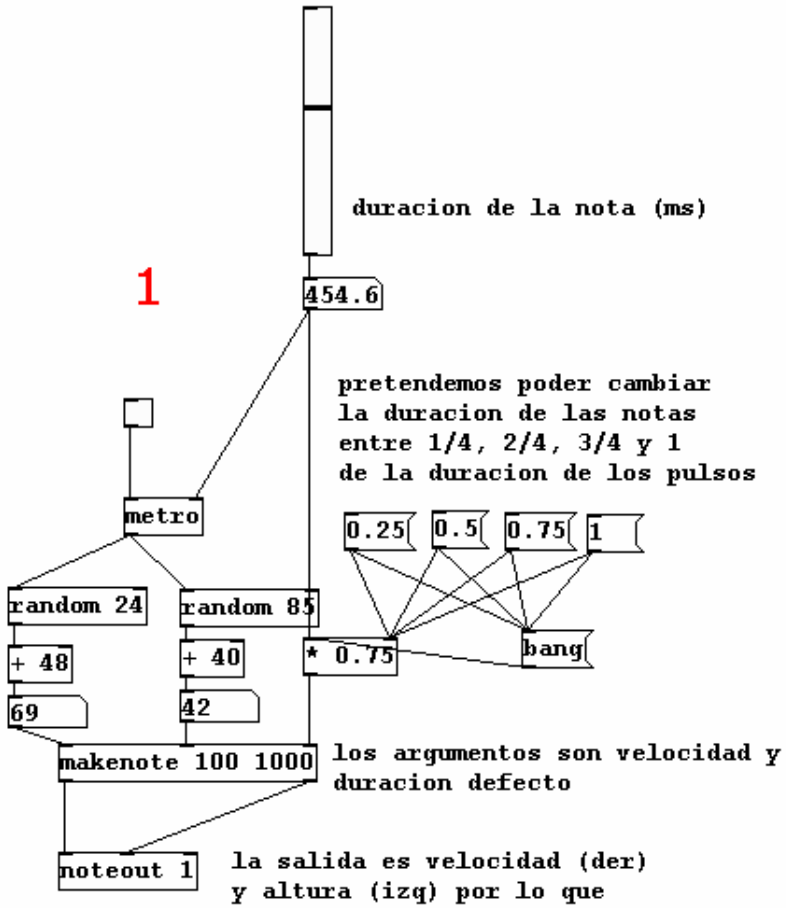
version modificada para bateria aleatoria

NOTA!!!
segun lo que habiamos comentado antes, no podemos garantizar que es lo que recibira primero noteout, si la velocidad o la nota. para garantizar un orden (normalmente deberiamos estar seguros de que la velocidad llega antes) deberiamos utilizar trigger [t b b] a la salida de metro. En este caso, no nos importa demasiado ya que los valores son aleatorios. Esto significa que si lo que llega primero es la nota, sonara utilizando la velocidad generada en el pulso anterior. Dado que todos los valores son aleatorios dificilmente producira una diferencia en la salida!!



esta podria ser una forma de obtener tiempos multiplos y dirigirlos a los metronomos de diferentes percusionistas

6.4



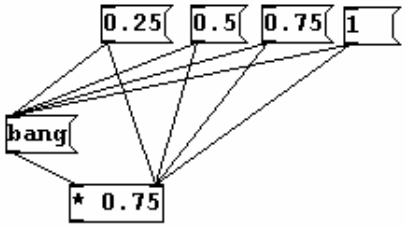
2

este patch tiene un problema y es que la duracion de las notas solo cambia cuando modificamos tambien el tempo. Esto es un problema tipico de PD debido a que los objetos solo producen salida cuando reciben la entrada de la izquierda

3

igual que sucedia con la suma (que solo producia un resultado cuando entraba el sumando izquierdo) sucede ahora con la multiplicacion que debe alimentar la entrada derecha de makenote.

esto se puede resolver, haciendo que cada vez que se pulse uno de los valores (0.25, 0.5, 0.75 y 1) se mande un bang a la multiplicacion

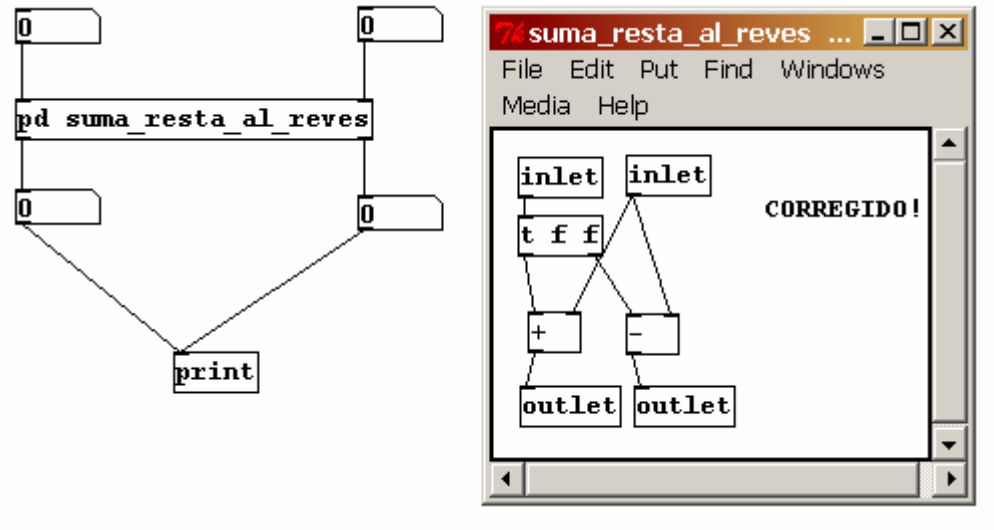


4

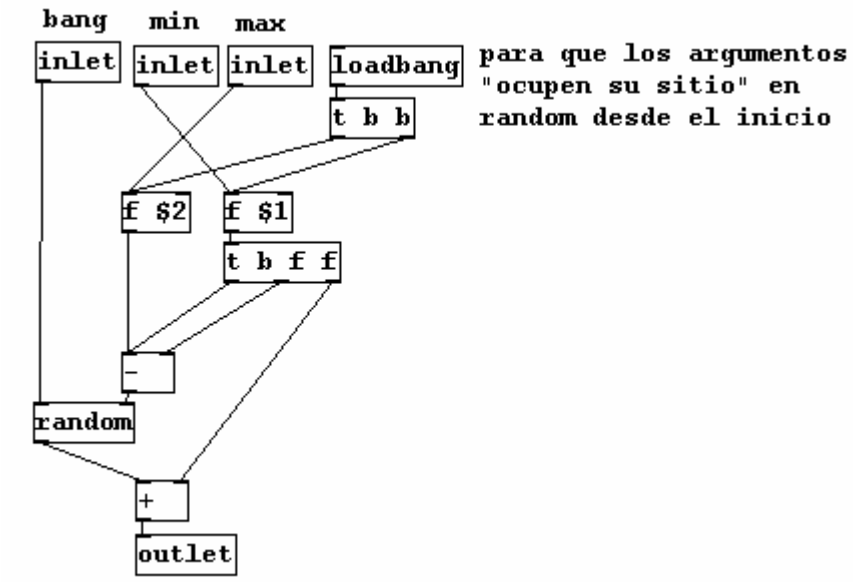
otra forma de completar este patch es anadiendo varios sliders adicionales con los que controlar:
-el rango aleatorio de las notas (ahora 24)
-la altura minima o tesitura (ahora 48)
-la velocidad o intensidad minima (ahora 40)

recordar que modificando las propiedades de los sliders se ajustan sus valores minimos y maximos

7.1

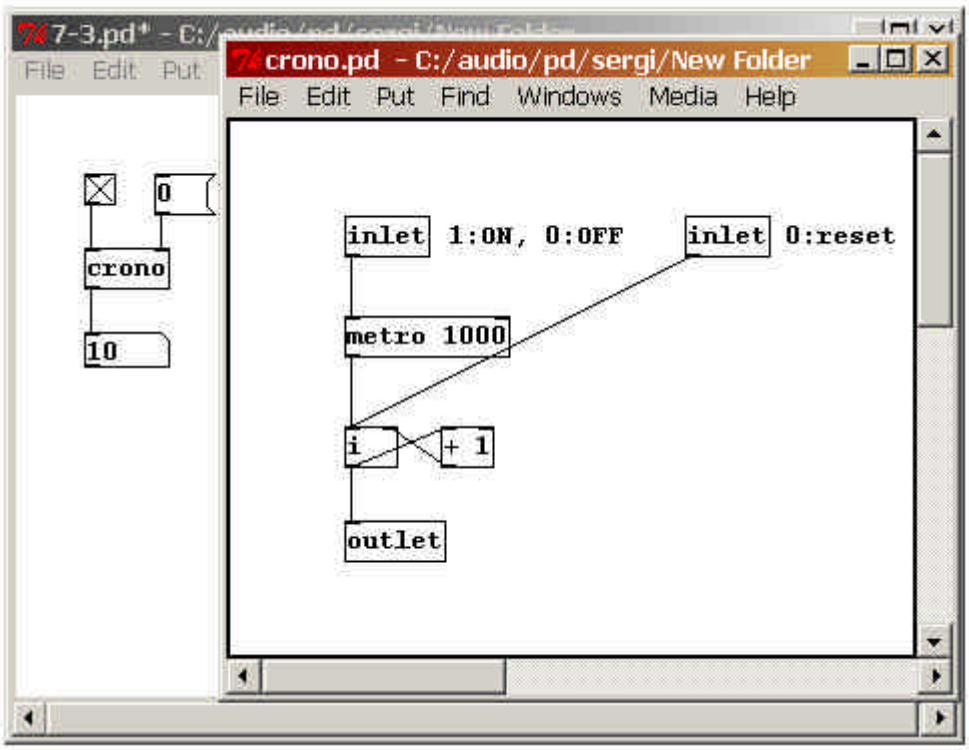


Random acotado entre mínimo y máximo.



NB. Este ejercicio no era nada trivial. El ***loadbang*** y los dos ***triggers*** son necesarios para que el objeto funcione correctamente tanto con argumentos iniciales, como cuando se modifican las cotas mediante las entradas 2 (min) y 3 (max). A pesar de esto, el objeto no funcionará cuando por error, la primera cota sea superior a la segunda. Para ello habría que modificarlo ligeramente.

7.3


7-3.pd* - C:/
File Edit Put
crono.pd - C:/audio/pd/sergi/New Folder
File Edit Put Find Windows Media Help
0
crono
10
inlet 1:ON, 0:OFF
inlet 0:reset
metro 1000
i
+ 1
outlet

B. Lista completa de objetos PD (v0.29)

Esta lista se obtiene también seleccionando HELP sobre el fondo de una ventana PD.

The "reference" section of the documentation should contain a patch demonstrating how to use each of Pd's classes. As of version 0.29, a complete list of "object" classes follows. Not included in this list are messages, atoms, graphs, etc. which aren't typed into object boxes but come straight off the "add" menu.

GLUE	
bang	output a bang message
float	store and recall a number
symbol	store and recall a symbol
int	store and recall an integer
send	send a message to a named object
receive	catch "sent" messages
select	test for matching numbers or symbols
route	route messages according to first element
pack	make compound messages
unpack	get elements of compound messages
trigger	sequence and convert messagess
spigot	interruptible message connection
moses	part a numeric stream
until	looping mechanism
print	print out messages
makefilename	format a symbol with a variable field
change	remove repeated numbers from a stream
swap	swap two numbers
value	shared numeric value

TIME	
delay	send a message after a time delay
metro	send a message periodically
line	send a series of linearly stepped numbers
timer	measure time intervals
cputime	measure CPU time
realtime	measure real time
pipe	dynamically growable delay line for numbers

MATH

+ - * / pow	arithmetic
== != > < >= <=	relational tests
& && | || %	bit twiddling
mtof ftom powtodb rmstodb dbtopow dbtorms	convert acoustical units
mod div sin cos tan atan atan2 sqrt log exp abs	higher math
random	Generates random number
max min	greater or lesser of 2 numbers
clip	force a number into a range

MIDI

notein ctlin pgmin bendin touchin polytouchin midiin sysexin	MIDI input
noteout ctlout pgmout bendout touchout polytouchout midiout	MIDI output
makenote	schedule a delayed "note off" message corresponding to a note-on
stripnote	strip "note off" messages

TABLES

tabread	read a number from a table
tabread4	read a number from a table, with 4 point interpolation
tabwrite	write a number to a table
soundfiler	read and write tables to soundfiles

MISC

loadbang	bang on load
serial	serial device control for NT only
netsend	send messages over the internet
netreceive	receive them
qlist	message sequencer
textfile	file to message converter
openpanel	"Open" dialog
savepanel	"Save as" dialog
bag	set of numbers
poly	polyphonic voice allocation
key, keyup	numeric key values from keyboard
keyname	symbolic key name

AUDIO MATH

+~ -~ *~ /~	arithmetic on audio signals
max~ min~	maximum or minimum of 2 inputs
clip~	constrict signal to lie between two bounds
q8_rsqrt~	cheap reciprocal square root (beware -- 8 bits!)
q8_sqrt~	cheap square root (beware -- 8 bits!)
wrap~	wraparound (fractional part, sort of)
fft~	complex forward discrete Fourier transform
ifft~	complex inverse discrete Fourier transform
rfft~	real forward discrete Fourier transform
rifft~	real inverse discrete Fourier transform
framp~	output a ramp for each block
mtof~, ftom~, rmstodb~, dbtorms~, rmstopow~, powtorms~	acoustic conversions

AUDIO GLUE

dac~	audio output
adc~	audio input
sig~	convert numbers to audio signals
line~	generate audio ramps
threshold~	detect signal thresholds
snapshot~	sample a signal (convert it back to a number)
bang~	send a bang message after each DSP block
samplerate~	get the sample rate
send~	nonlocal signal connection with fanout
receive~	get signal from send~
throw~	add to a summing bus
catch~	define and read a summing bus
block~	specify block size and overlap
switch~	switch DSP computation on and off
readsf~	soundfile playback from disk
writesf~	record sound to disk

AUDIO OSCILLATORS AND TABLES

phasor~	sawtooth oscillator
cos~	cosine
osc~	cosine oscillator
tabwrite~	write to a table
tabplay~	play back from a table (non-transposing)
tabread~	non-interpolating table read
tabread4~	four-point interpolating table read
tabsend~	write one block continuously to a table
tabreceive~	read one block continuously from a table

AUDIO FILTERS

vcf~	voltage controlled filter
noise~	white noise generator
env~	envelope follower
hip~	high pass filter
lop~	low pass filter
bp~	band pass filter
biquad~	raw filter
samphold~	sample and hold unit
print~	print out one or more "blocks"

AUDIO DELAY

delwrite~	write to a delay line
delread~	read from a delay line
vd~	read from a delay line at a variable delay time

SUBWINDOWS

pd	define a subwindow
inlet	add an inlet to a pd
outlet	add an outlet to a pd
table	array of numbers in a subwindow

DATA TEMPLATES

template	define the fields in a template
drawcurve, filledcurve	draw a curve
drawpolygon, filledpolygon	draw a polygon
plot	plot an array field
drawnumber	print a numeric value

ACCESSING DATA

pointer	point to an object belonging to a template
get	get numeric fields
set	change numeric fields
element	get an array element
getsize	get the size of an array
setsize	change the size of an array
append	add an element to a list
sublist	get a pointer into a list which is an element of another scalar
scalar	draw a scalar on parent

C. PD en Internet

There is a new Pd community web site, pure-data.org, which aims to be the central resource for Pd, from documentation and downloads; to forums, member pages, a patch exchange.

There is a growing number of Pd-related projects hosted at SourceForge. This is open to all Pd developers, and all are encouraged to join. Send an email to the pd-dev list, pd-dev[AT]iem.kug.ac.at, to join.

Most of the interesting resources related to Pd show up on the Pd mailing list, maintained by Iohannes Zmoelnig. To subscribe or browse the archives visit: http://iem.kug.ac.at/mailinglists/pd-list/. . This is the best source of recent information regarding installation problems and bugs. It is perfectly reasonable to post "newbie" questions on this list; alternatively you can contact msp@ucsd.edu for help.

Many extensions to Pd are announced on the mailing list. In particular, for people interested in graphics, there is a A 3D graphics rendering package, named GEM, based on OpenGL, was written by Mark Danks, adapted to Linux by Guenter Geiger, and is now maintained by Iohannes Zmoelnig. GEM runs on Windows and Linux and probably will run with some coaxing on IRIX. You can get it from: http://iem.kug.ac.at/GEM .

A video processing package, Framestein, is by Juha Vehvilainen. This runs on Windows only: http://framestein.org .

Here are some more Pd links (in the order I found them):
Miller Puckette's home page
Guenter Geiger's home page
Mark Dank's home page
Pd page on Wonk (Klaus)
Johannes M Zmoelnig
Norbert Math's Pd page
Thomas Musil's IEMLIB
Nicolas Lhommet's WikiWikiWeb page for Pd
Norbert's searchable list of all known Pd objects
Krzysztof Czaja's MIDI file support
David Sabine's Pd Documentation Project: new, highly detailed help windows
Fernando Pablo Lopez's augmented Pd RPMs from Planet CCRMA
Cyclone - Krzysztof Czaja's Max compatibility library
On-line book project: *Theory and Techniques of Electronic Music*

D. Tablas MIDI varias

D1. Relación de notas MIDI y frecuencias (en Hz)

Nota	Freq. (Hz)	MIDI	Nota	Freq.(Hz)	MIDI	Nota	Freq.(Hz)	MIDI
La 1	27.500	21	La 4	220.000	57	La 7	1760.000	93
La# 1	29.135	22	La# 4	233.082	58	La# 7	1864.655	94
Si 1	30.868	23	Si 4	246.942	59	Si 7	1975.533	95
Do 2	32.703	24	Do 5	261.626	60	Do 8	2093.005	96
Do# 2	34.648	25	Do# 5	277.183	61	Do# 8	2217.461	97
Re 2	36.708	26	Re 5	293.665	62	Re 8	2349.318	98
Re# 2	38.891	27	Re# 5	311.127	63	Re# 8	2489.016	99
Mi 2	41.203	28	Mi 5	329.628	64	Mi 8	2637.021	100
Fa 2	43.654	29	Fa 5	349.228	65	Fa 8	2793.826	101
Fa# 2	46.249	30	Fa# 5	369.994	66	Fa# 8	2959.956	102
Sol 2	48.999	31	Sol 5	391.995	67	Sol 8	3135.964	103
Sol# 2	51.913	32	Sol# 5	415.305	68	Sol# 8	3322.438	104
La 2	55.000	33	La 5	440.000	69	La 8	3520.000	105
La# 2	58.270	34	La# 5	466.164	70	La# 8	3729.310	106
Si 2	61.735	35	Si 5	493.883	71	Si 8	3951.066	107
Do 3	65.406	36	Do 6	523.251	72	Do 9	4186.009	108
Do# 3	69.296	37	Do# 6	554.365	73	Do# 9	4434.922	109
Re 3	73.416	38	Re 6	587.330	74	Re 9	4698.637	110
Re# 3	77.782	39	Re# 6	622.254	75	Re# 9	4978.032	111
Mi 3	82.407	40	Mi 6	659.255	76	Mi 9	5274.042	112
Fa 3	87.307	41	Fa 6	698.457	77	Fa 9	5587.652	113
Fa# 3	92.499	42	Fa# 6	739.989	78	Fa# 9	5919.912	114
Sol 3	97.999	43	Sol 6	783.991	79	Sol 9	6271.928	115
Sol# 3	103.826	44	Sol# 6	830.609	80	Sol# 9	6644.876	116
La 3	110.000	45	La 6	880.000	81	La 9	7040.000	117
La# 3	116.541	46	La# 6	932.328	82	La# 9	7458.620	118
Si 3	123.471	47	Si 6	987.767	83	Si 9	7902.133	119
Do 4	130.813	48	Do 7	1046.502	84	Do 10	8372.019	120
Do# 4	138.591	49	Do# 7	1108.731	85	Do# 10	8869.845	121
Re 4	146.832	50	Re 7	1174.659	86	Re 10	9397.273	122
Re# 4	155.564	51	Re# 7	1244.508	87	Re# 10	9956.064	123
Mi 4	164.814	52	Mi 7	1318.510	88	Mi 10	10548.083	124
Fa 4	174.614	53	Fa 7	1396.913	89	Fa 10	11175.305	125
Fa# 4	184.997	54	Fa# 7	1479.978	90	Fa# 10	11839.823	126
Sol 4	195.998	55	Sol 7	1567.982	91	Sol 10	12543.855	127
Sol# 4	207.652	56	Sol# 7	1661.219	92	Sol# 10	13289.752	128

D2. Programas General MIDI

0	Piano de cola	1	Piano brillante	2	Piano de cola eléctrico	3	Piano de bar
4	Piano eléctrico Rhodes	5	Piano eléctrico con chorus	6	Clavicordio	7	Clavinet
8	Celesta	9	Glockenspiel	10	Caja de música	11	Vibráfono
12	Marimba	13	Xilófono	14	Campanas tubulares	15	Salterio
16	Organo Hammond	17	Organo percusivo	18	Organo de rock	19	Organo de iglesia
20	Armonio	21	Acordeón	22	Armónica	23	Bandoneón
24	Guitarra española	25	Guitarra acústica	26	Guitarra eléctrica de jazz	27	Guitarra eléctrica limpia
28	Guitarra eléctrica apagada	29	Guitarra eléctrica con overdrive	30	Guitarra eléctrica distorsionada	31	Armónicos de guitarra eléctrica
32	Bajo acústico	33	Bajo eléctrico (dedos)	34	Bajo eléctrico (púa)	35	Bajo eléctrico sin trastes
36	Bajo eléctrico golpeado 1	37	Bajo eléctrico golpeado 2	38	Bajo sintético1	39	Bajo sintético2
40	Violín	41	Viola	42	Violonchelo	43	Contrabajo
44	Violín trémolo	45	Violín pizzicato	46	Arpa	47	Timbal de orquesta
48	Sección de cuerda 1	49	Sección de cuerda 2	50	Sección de cuerda sintética 1	51	Sección de cuerda sintética 2
52	Coro Aahs	53	Coro Oohs	54	Voz sintética	55	Golpe de orquesta
56	Trompeta	57	Trombón	58	Tuba	59	Trompeta c sordina
60	Fiscorno	61	Sección de metal	62	Sección de metal sintética 1	63	Sección de metal sintética 2
64	Saxo soprano	65	Saxo alto	66	Saxo Tenor	67	Saxo Barítono
68	Oboe	69	Corno inglés	70	Fagot	71	Clarinete
72	Flautín	73	Flauta travesera	74	Flauta de pico	75	Flauta de Pan
76	Cuello de botella (soplo)	77	Shakuhachi (flauta japonesa)	78	Silbido	79	Ocarina
80	Sinte melodía 1 (onda cuadrada)	81	Sinte melodía 2 (diente sierra)	82	Sinte melodía 3 (órgano)	83	Sinte melodía 4 (siseo)
84	Sinte melodía 5 (charango)	85	Sinte melodía 6 (vocal)	86	Sinte melodía 7 (quintas)	87	Sinte melodía 8 (bajo)
88	Sinte armonía 1 (new age)	89	Sinte armonía 2 (cálido)	90	Sinte armonía 3 (polysynth)	91	Sinte armonía 4 (Coral)
92	Sinte armonía 5 (arco)	93	Sinte armonía 6 (metálico)	94	Sinte armonía 7 (celestial)	95	Sinte armonía 8 (filtro)
96	Sinte efecto 1 (lluvia)	97	Sinte efecto 2 (banda sonora)	98	Sinte efecto 3 (cristales)	99	Sinte efecto 4 (atmosférico)
100	Sinte efecto 5 (brillante)	101	Sinte efecto 6 (duendes)	102	Sinte efecto 7 (ecos)	103	Sinte efecto 8 (ciencia ficción)
104	Sitar	105	Banjo	106	Shamisen (laúd japonés)	107	Koto (cítara japonesa)
108	Kalimba	109	Gaita	110	Violín country	111	Shannai (dulzaina hindú)
112	Cascabeles	113	Agogó	114	Steel Drum	115	Caja de madera
116	Taiko (tambor japonés)	117	Timbal melódico	118	Caja sintética	119	Platillo invertido
120	Trasteo de guitarra	121	Respiración	122	Orilla del mar	123	Trino
124	Timbre de teléfono	125	Helicóptero	126	Aplausos	127	Disparo

D3. Mapa General MIDI de los sonidos de percusión (canal 10)

27	High Q	28	Slap	29	Scratch Push	30	Scratch Pull
31	Sticks	32	Square Click	33	Metronome Click	34	Metronome Bell
35	Bass Drum 2	36	Bass Drum 1	37	Side Stick	38	Acoustic Snare
39	Handclap	40	Electric Snare	41	Low Floor Tom	42	Closed High Hat
43	High Floor Tom	44	Pedal High Hat	45	Low Tom	46	High Hat
47	Low Mid Tom	48	High Mid Tom	49	Crash Cymbal 1	50	High Tom
51	Ride Cymbal 1	52	Chinese Cymbal	53	Ride Bell	54	Tambourine
55	Splash Cymbal	56	Cowbell	57	Crash Cymbal 2	58	Vibraslap
59	Ride Cymbal 2	60	High Bongo	61	Low Bongo	62	Mute High Conga
63	High Conga	64	Low Conga	65	High Timbale	66	Low Timbale
67	High Agogo	68	Low Agogo	69	Cabasa	70	Maracas
71	Short Whistle	72	Long Whistle	73	Short Guiro	74	Long Guiro
75	Claves	76	High Wood Block	77	Low Wood Block	78	Mute Cuica
79	Cuica	80	Mute Triangle	81	Triangle	82	Shaker
83	Jingle Bell	84	Belltree	85	Castanets	86	Mute Surdo
87	Open Surdo						