~Pure Data~

Ninon Devis: ninon.devis@ircam.fr

Made with love for ATIAM

Normalizing the signal

Goal: Obtaining the best dynamic range by fitting the gain of the signal into certain ranges.

Two step-process:

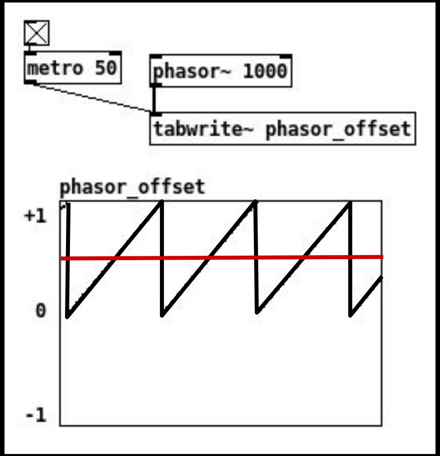
- Removing the DC Offset
- Normalizing the signal

What is DC Offset ?

Mean amplitude displacement from 0.

What is Normalization ?

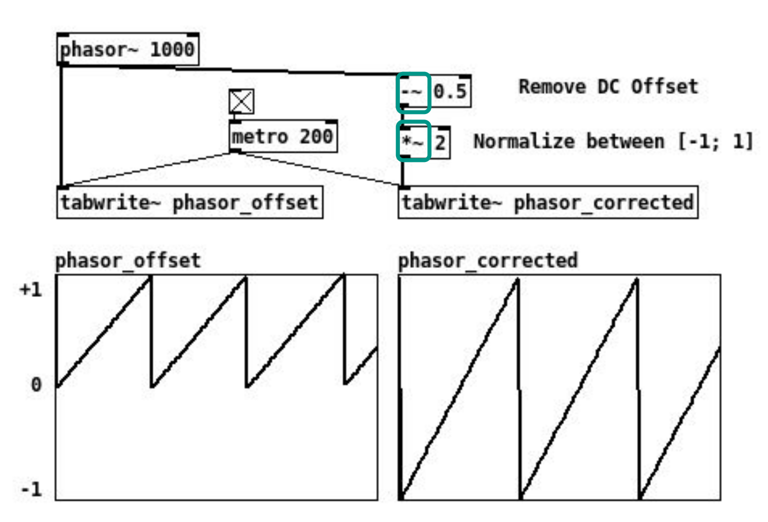
Adjust the gain to peak at the maximum the sound card allows before clipping: [-1, 1].



How to normalize this signal ?

DC Offset = 0.5

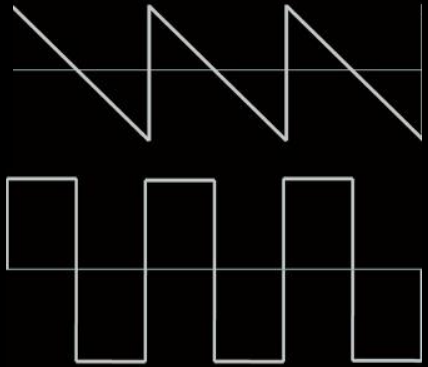
Don’t forget it’s a ~signal



Normalizing the signal

Example: from a Phasor to a Square

How to convert a sawtooth into a square?

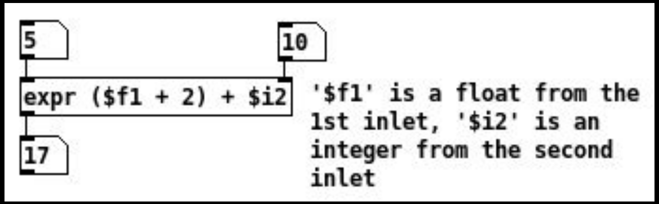


Sawtooth

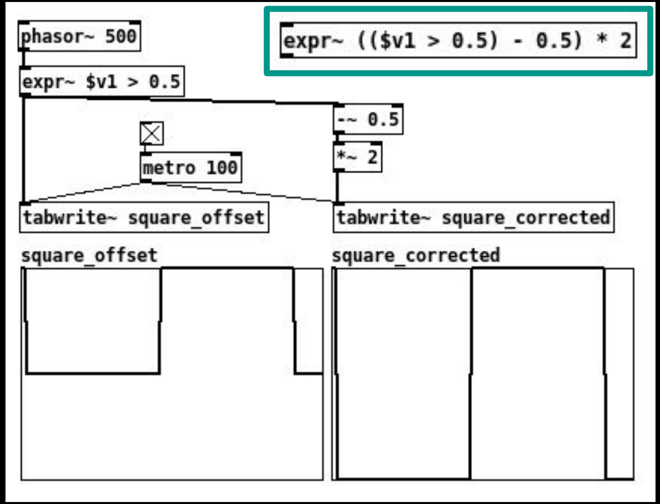
Square

$$\begin{cases} \text{output} = 1 \text{ if signal} > 0.5 \\ 0 \text{ otherwise} \end{cases}$$

➔ Introducing [expr] & [expr~]



The first inlet of [expr~] needs to be of type '$v1' for a signal



Select 'Polygon' in the properties of the array

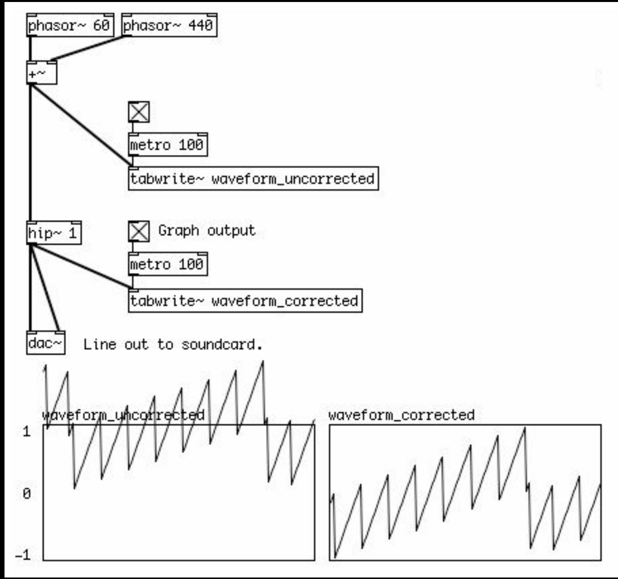
Normalizing the signal

Other way to get rid from the DC Offset

DC Offset = Direct Current Offset

- Continuous component that shifts the signal toward + or -
- As it is continuous, it has a 0 Hz frequency

➔ A Hi-Pass filter with a very low cutoff frequency (below the perception) can remove it without altering the signal.



➔ *Put a Hi-Pass before each channel of your dac~*

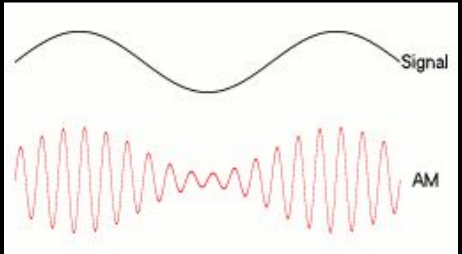
This won’t normalize your signal:

➔ Does not prevent clipping !

Amplitude Modulation

Earliest method to transmit audio in radio broadcast

Definition: varying the amplitude of a high frequency signal, the carrier signal, as a function of a lower frequency signal, the modulating signal (commonly the one containing the information to be transmitted).



Carrier signal

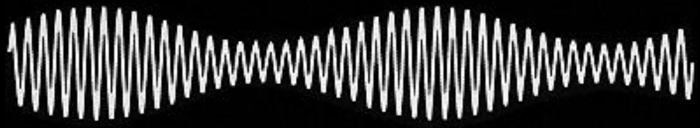
high frequency

$$x_p(t) = A_p \cos(\omega_p t)$$

Modulating signal

low frequency

$$x_m(t) = A_m \cos(\omega_m t)$$



Output: $$y(t) = x_p(t) + k x_p(t) x_m(t)$$

Amplitude Modulation Synthesis

Stereo Oscillator

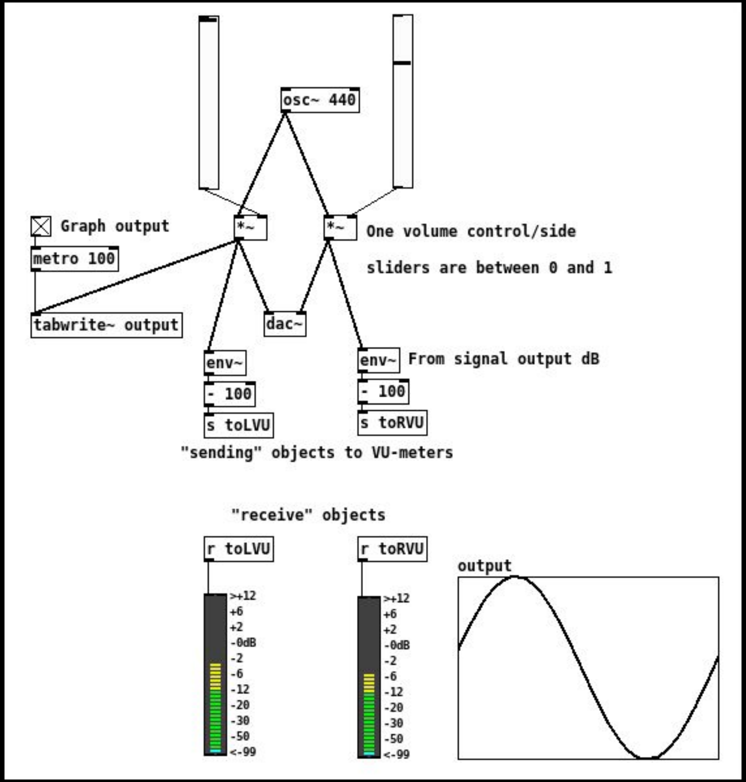
- This patch is stereo: how would you do this ?
- We want to visualize the output dB: which object?

Introducing the wireless connexion in Pd:

➔ Use a “sending” object and a “receive” object formalized as [s *] and [r *]

- To use the VU-meter we need [env~] which take a signal and output its RMS amplitude in dB.

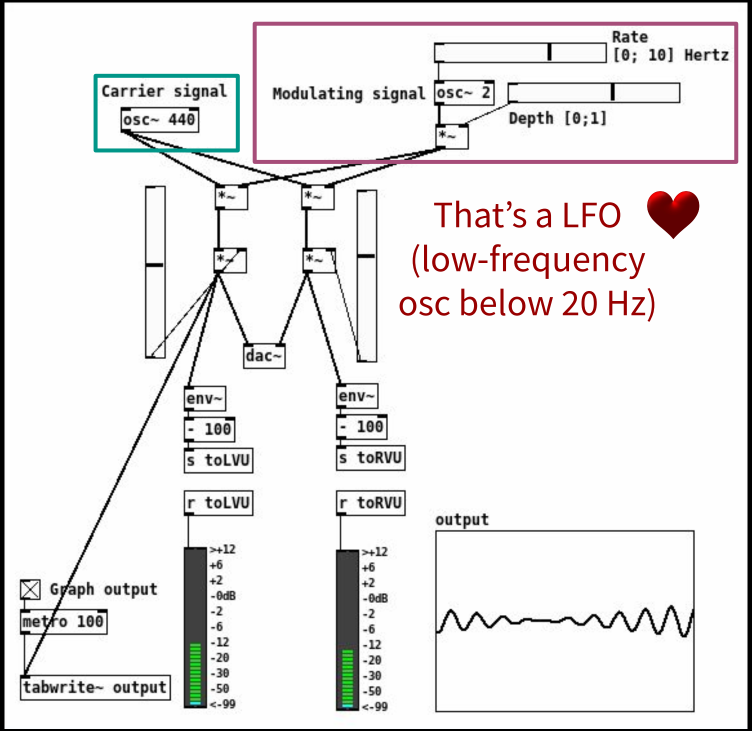
➔ *First step*: building the stereo oscillator



Amplitude Modulation Synthesis

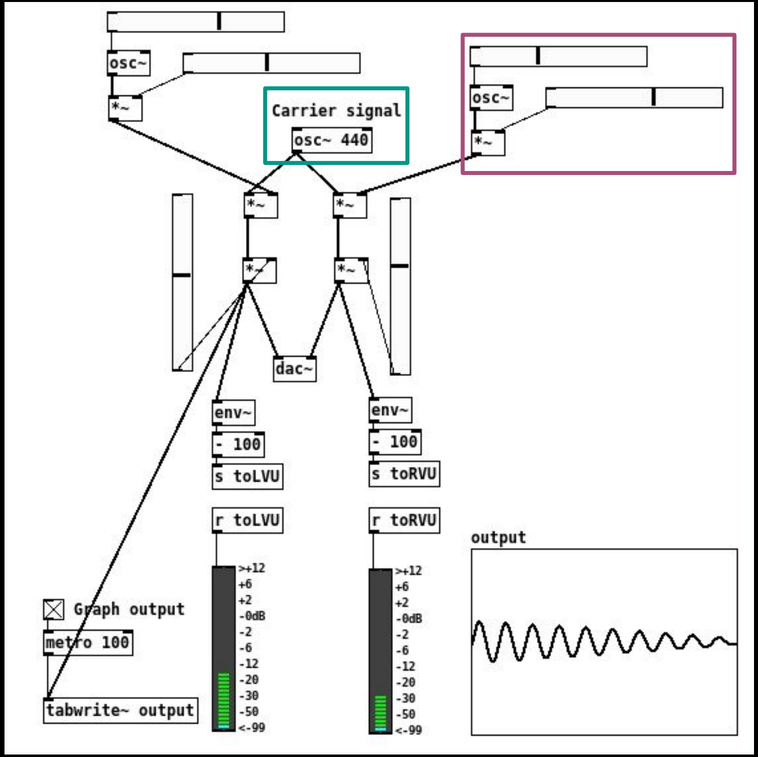
Using one oscillator to modulate the gain of an other one

- According to your intuition, how would you patch an AM?



That’s a LFO (low-frequency osc below 20 Hz)

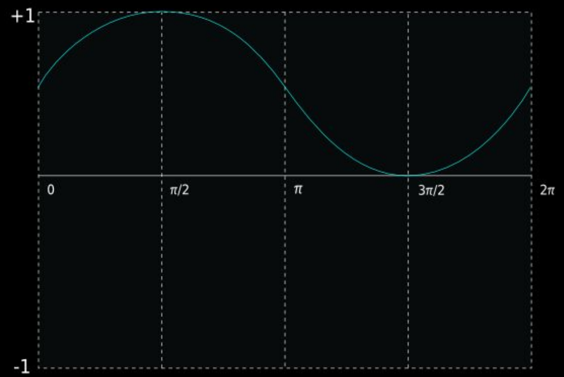
- Let’s create a tremolo from this: adding AM on both sides



Ring Modulation

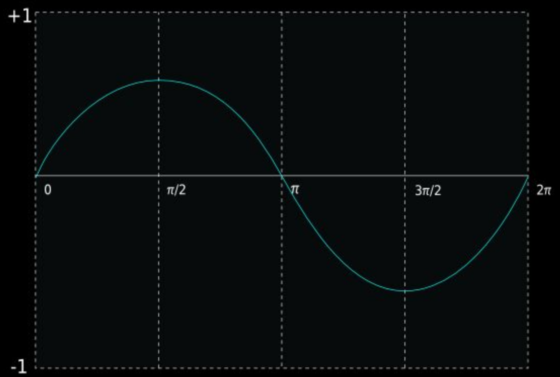
Difference between AM & RM

Unipolar signal



DC offset

Bipolar signal



Mean amplitude around 0

$R(t)$ ring modulation output

$A(t)$ amplitude modulation output

$C(t)$ carrier signal

$M(t)$ modulation signal

- **RM**: multiplication of two bipolar signals by each others $R(t) = C(t) \times M(t)$

➔ The frequency of the carrier signal is not present in the resulting sound.

- **AM**: M is a unipolar modulator, typically between 0 and 1. $A(t) = C(t) \times (M(t) + 1)$

➔ The carrier frequency is preserved.

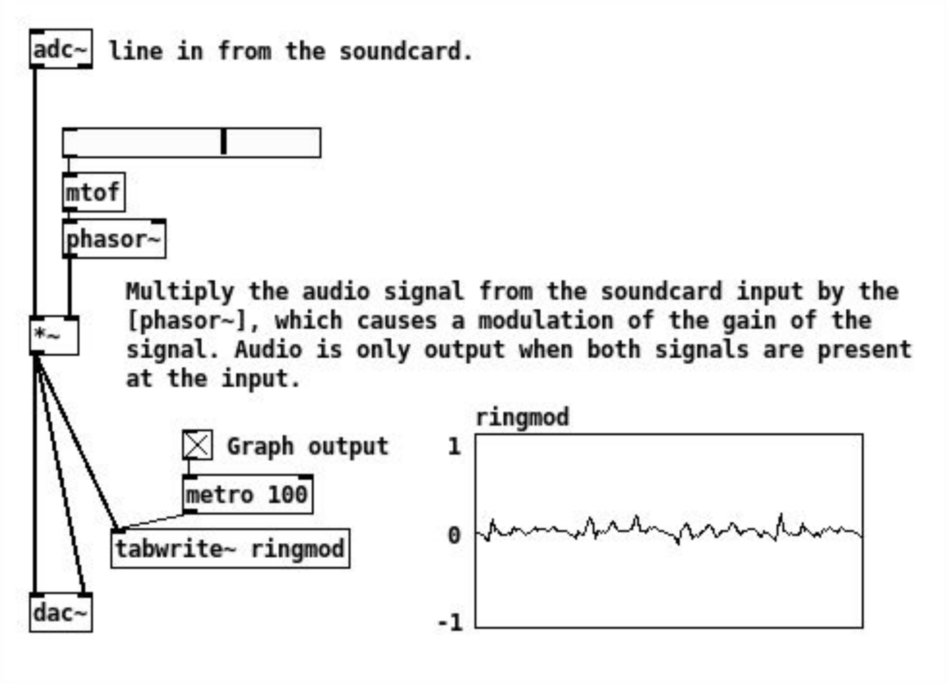
Ring Modulation

Making Alien's voice with ring modulation

Doctor Who - Cyberman voice

Which object will we use to catch our voice ?

➔ [adc~]



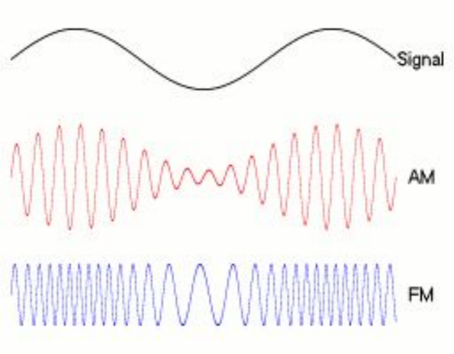
Frequency Modulation

By John Chowning

Stria (1977)

Definition: the information contained in the modulating signal is carried by varying the frequency of the carrier signal.

- Generally more robust than AM to transmit messages (less noise).
- Instable compare to AM regarding synthesis.
- Gives “natural” (& beautiful) sounds.

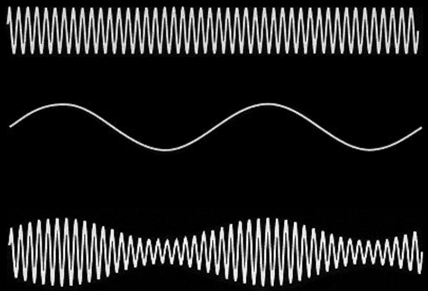


Difference between AM and FM

Mathematical intuition: sinusoid modulated by another sinusoid.

With: f_c carrier frequency, f_m modulating frequency and I_m modulation index, then:

output: $y(t) = \sin(2\pi f_c t + I_m \sin(2\pi f_m t))$



Carrier signal f_c

Modulating signal f_m

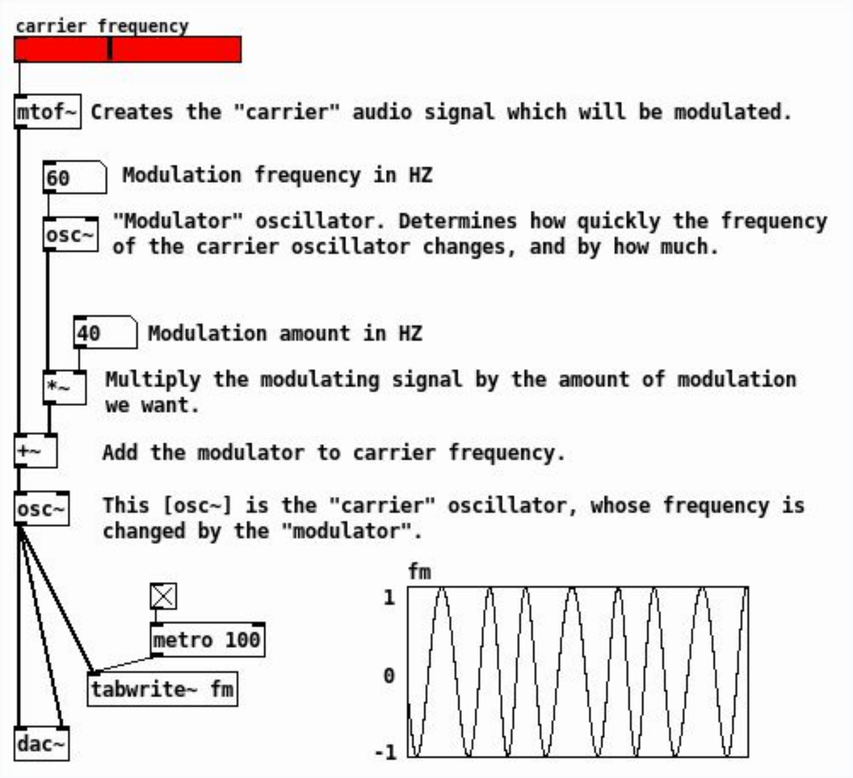
Output

Frequency Modulation

Simple fm patch

You can add colors on your objects to make your patch simpler

- For a very small amount of modulation: vibrato
- For a greater amount of modulation: glissando, or sweeping.

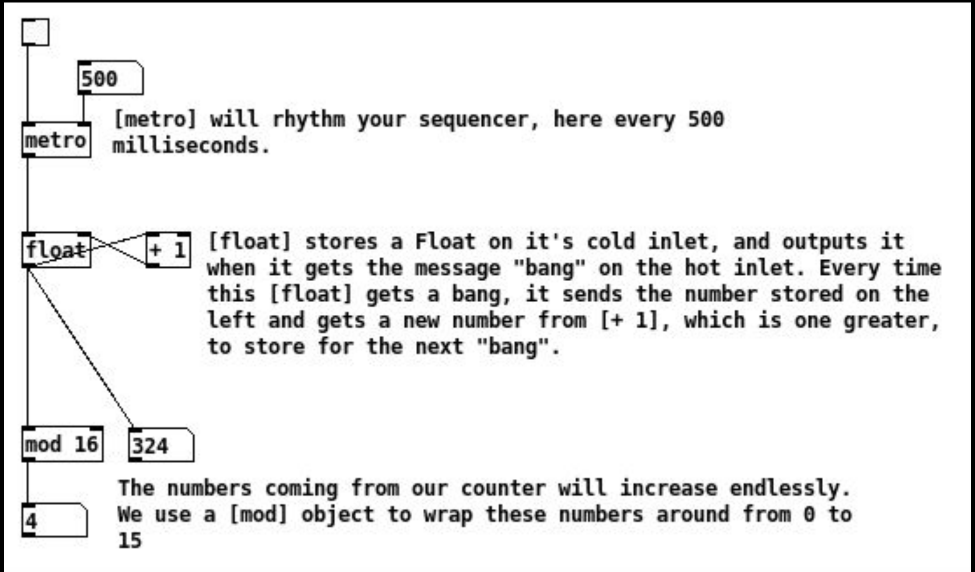


Step Sequencer

First Step: the counter

__Definition__: MIDI-based tool that divides a measure of music into a predetermined number of note value called steps.

➔ We first need a *counter*!



How would you do it ? We want 16 step sequencer

➔ You will probably need [mod *] which wrap number around given value.

➔ we want as output an increasing number modulo 16.

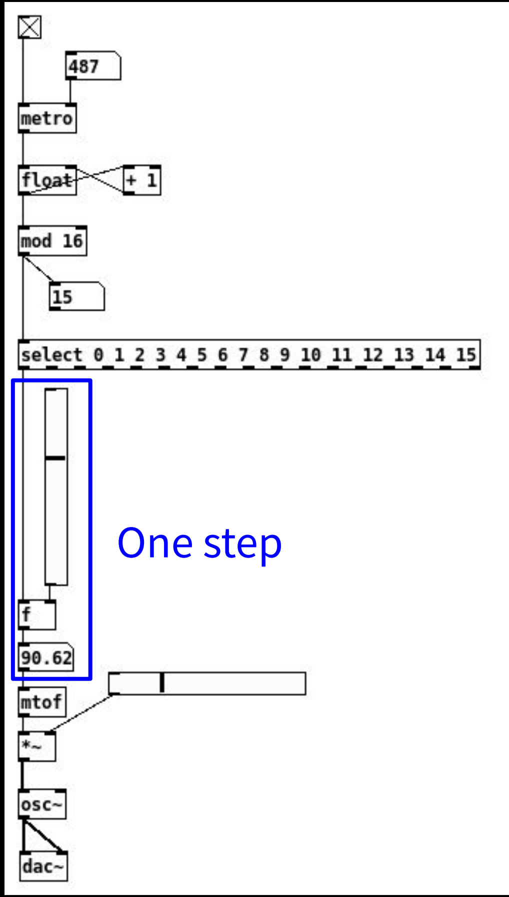
Step Sequencer

Second Step: the step sequencer

We will trigger a bang step by step using [select *] which compare numbers and send a bang if matching to the message.

When you are happy with one of your step, just copy paste 15 times

You can change osc, add enveloppes, configure your patch so that it plays harmonically, plays samples instead of notes...



Read a sound file

Reminder: Common sound file formats:

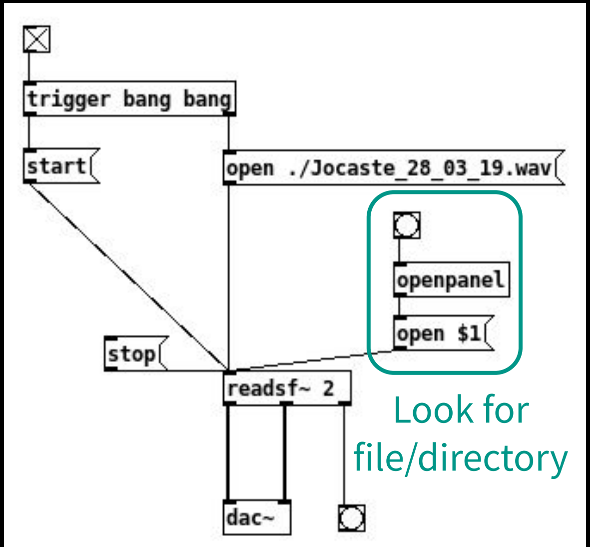
- No compression: .wav or .aiff
- Compression but no quality loss: .flac
- Compression and quality loss: .mp3

➔ Pd objects depending on the format

[readsf~] for .wav

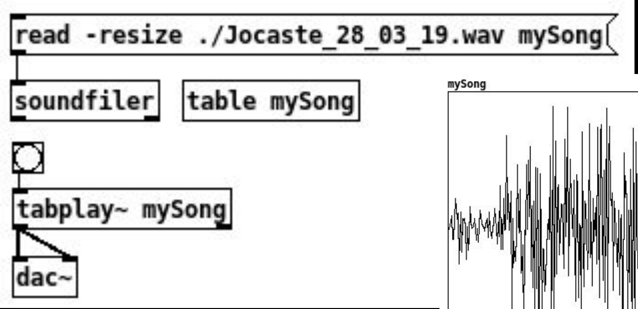
other formats not included in Pd Vanilla

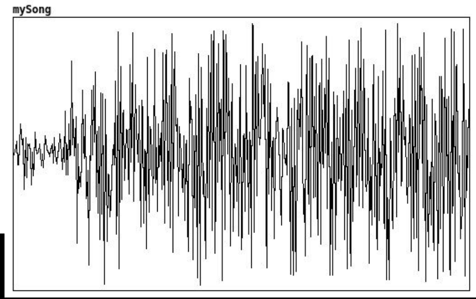
- ◆ Don’t use space or special characters for the name of your sound files
- ◆ Try to put your music in the same folder as your patch



How would you loop the song ?

[read[and [soundfiler] to “edit” the song

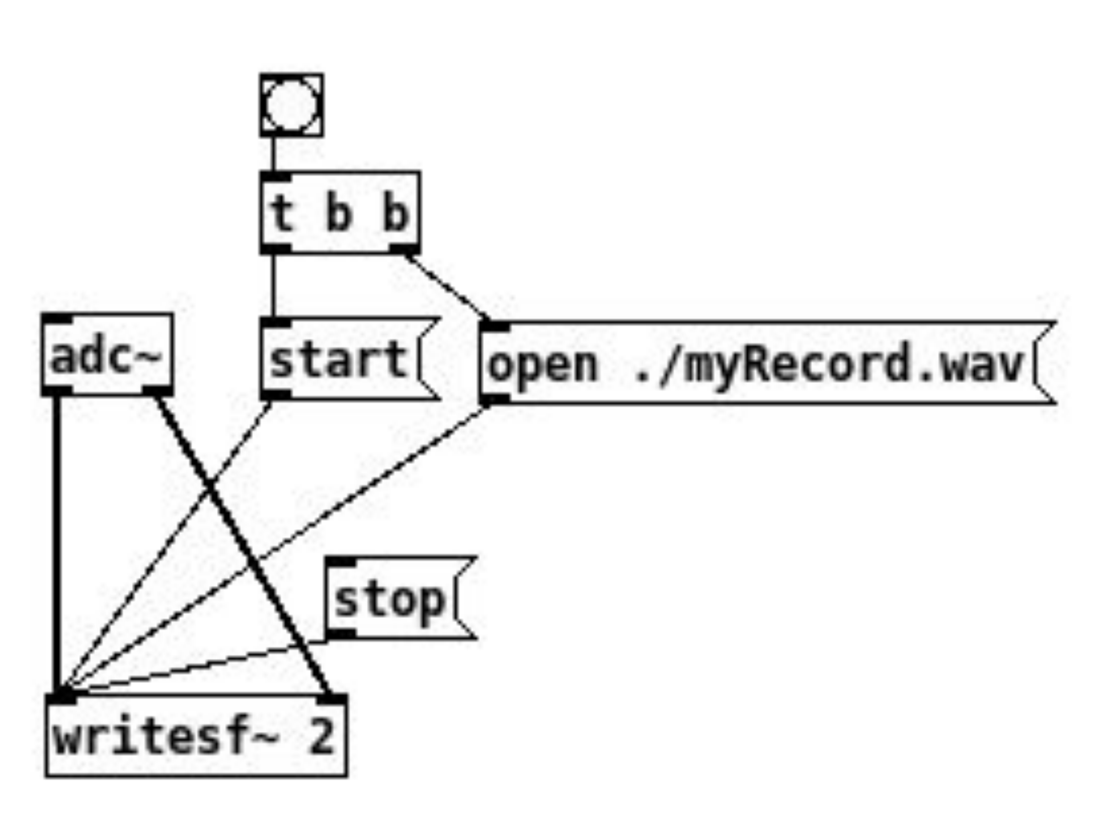




Seek help

Write a sound file

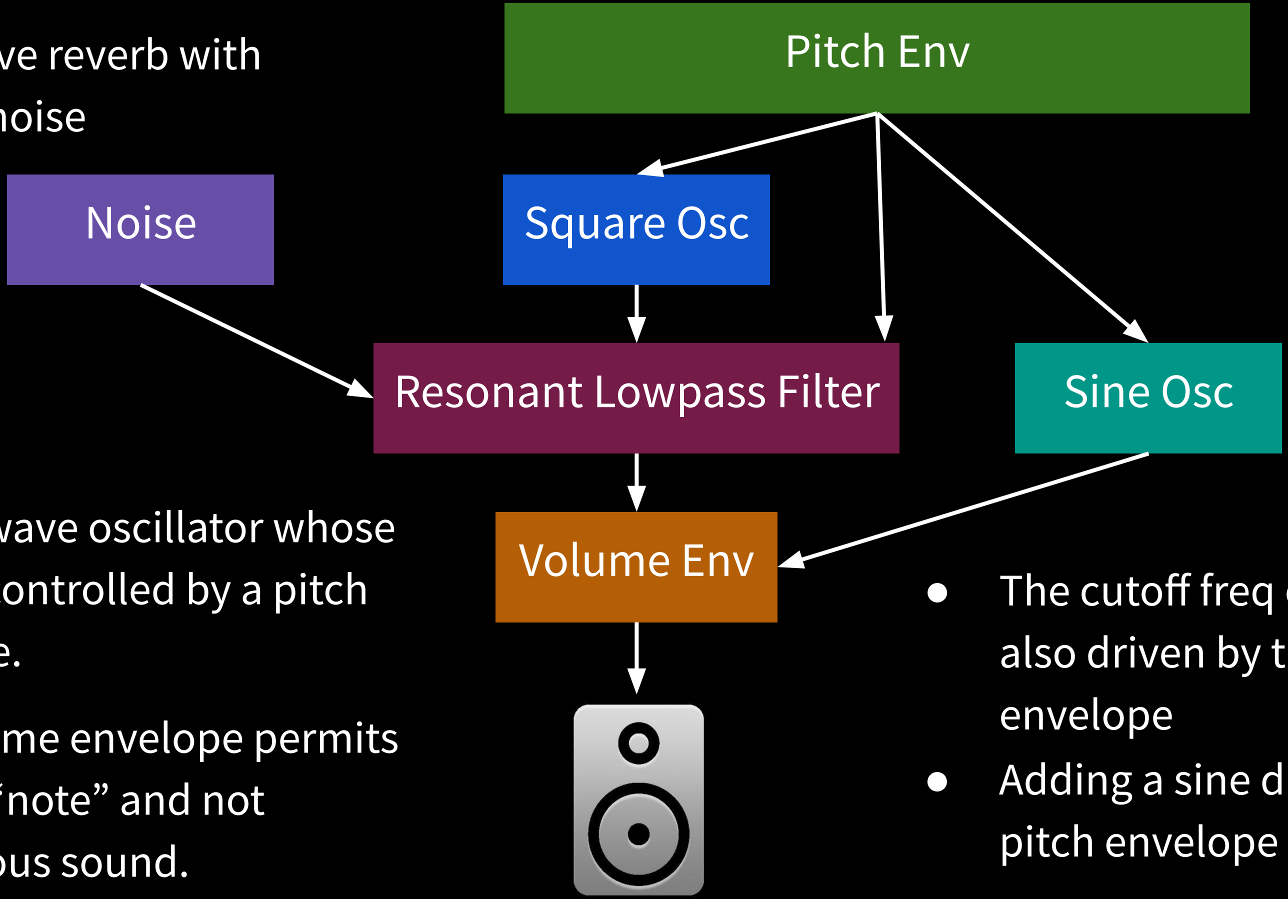
How to record in Pd ?



Kick Drum ❤

General Approach

- Percussive reverb with a white noise



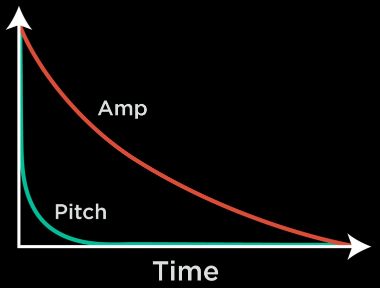
- Square wave oscillator whose pitch is controlled by a pitch envelope.
- The volume envelope permits to have “note” and not continuous sound.

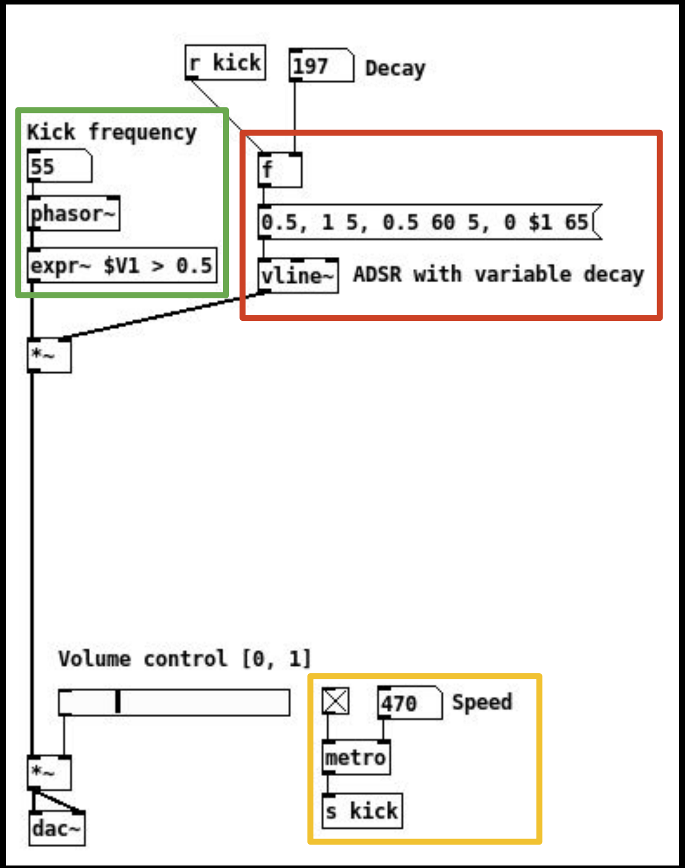
- The cutoff freq of the lowpass is also driven by the pitch envelope
- Adding a sine driven by the pitch envelope

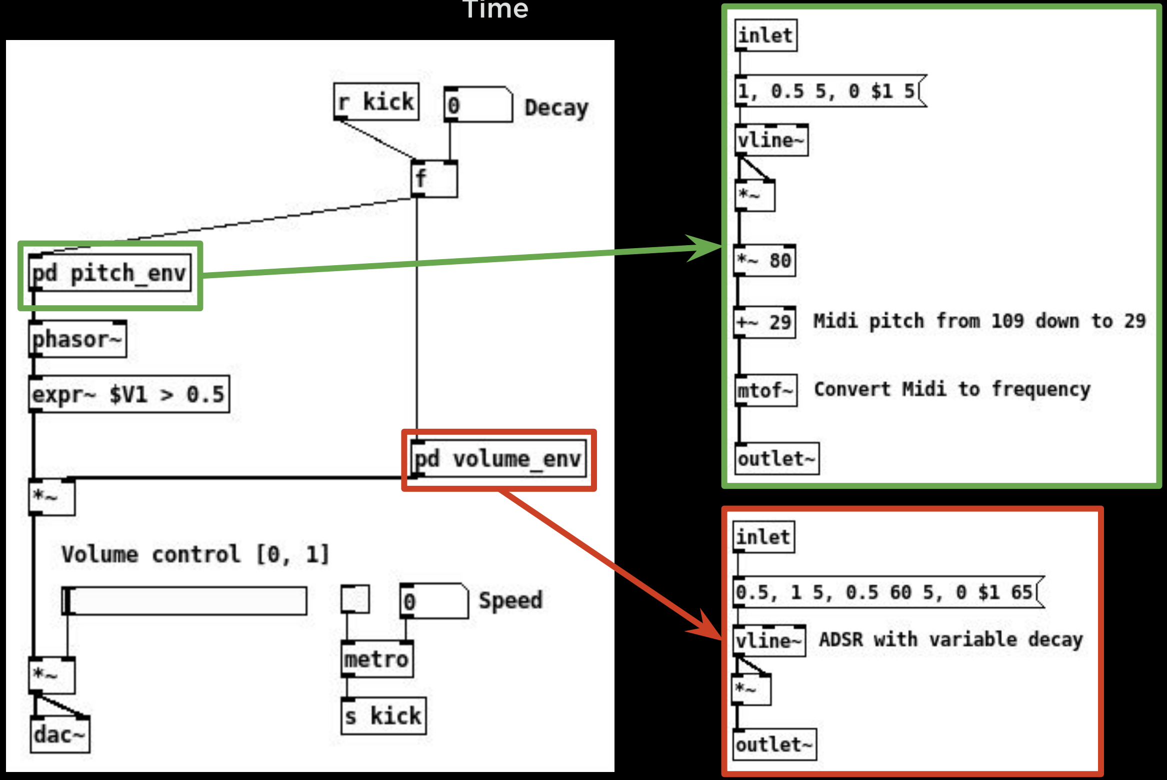
Kick Drum ❤

Part 1:

Phasor and Enveloppes

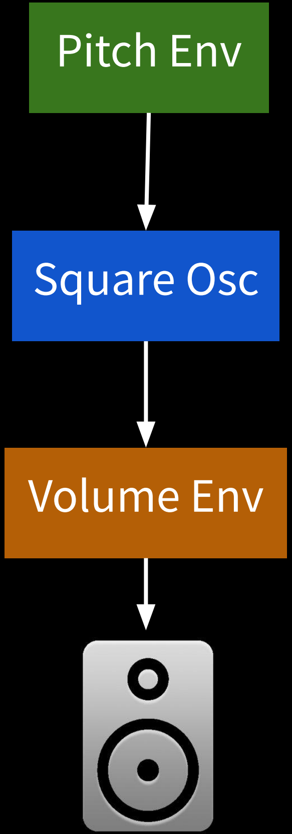






Kick Drum ❤

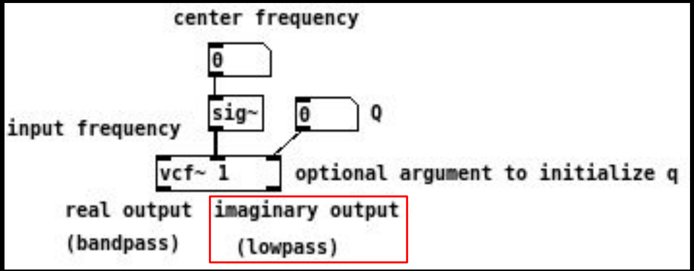
First step achievements



Next: adding a resonant low-pass filter, cutoff driven by our pitch enveloppe

➔ Which object do we need ?

VCF:

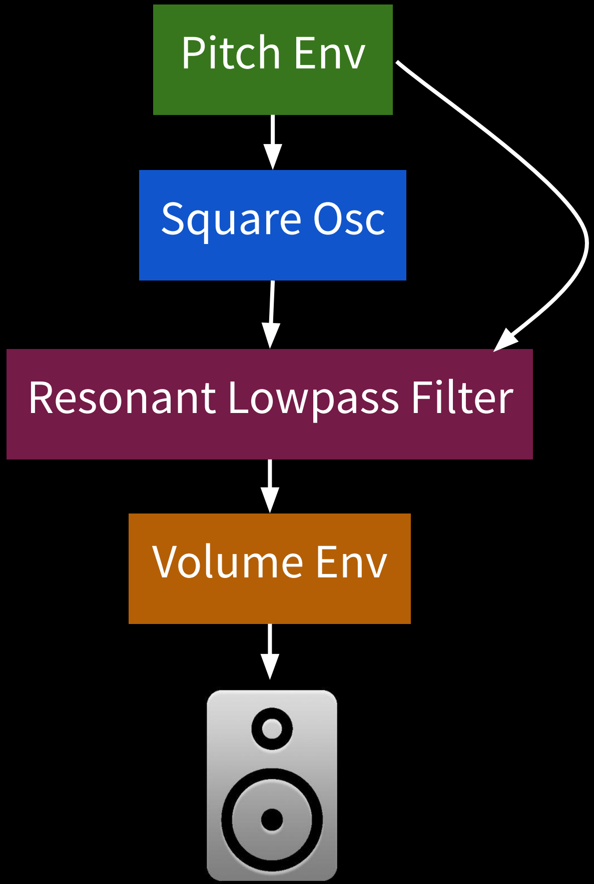


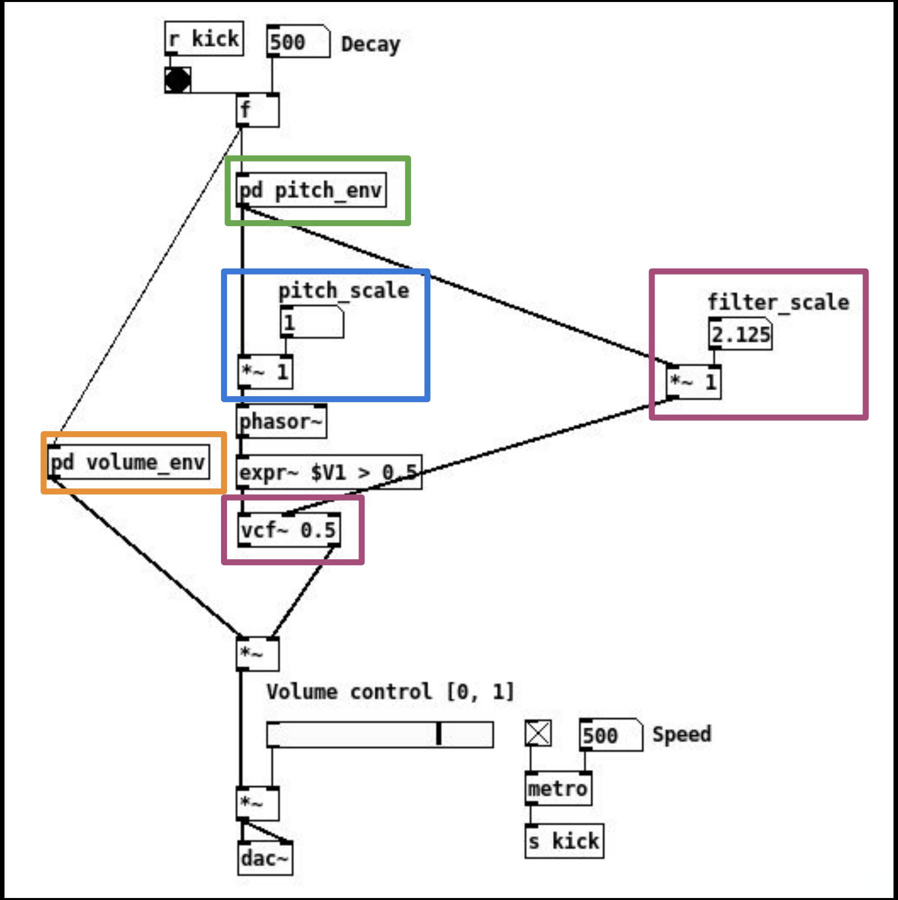
Where will we plug our pitch env ?

First, let’s add a pitch scale, then the vcf with a filter scale.

Kick Drum ❤

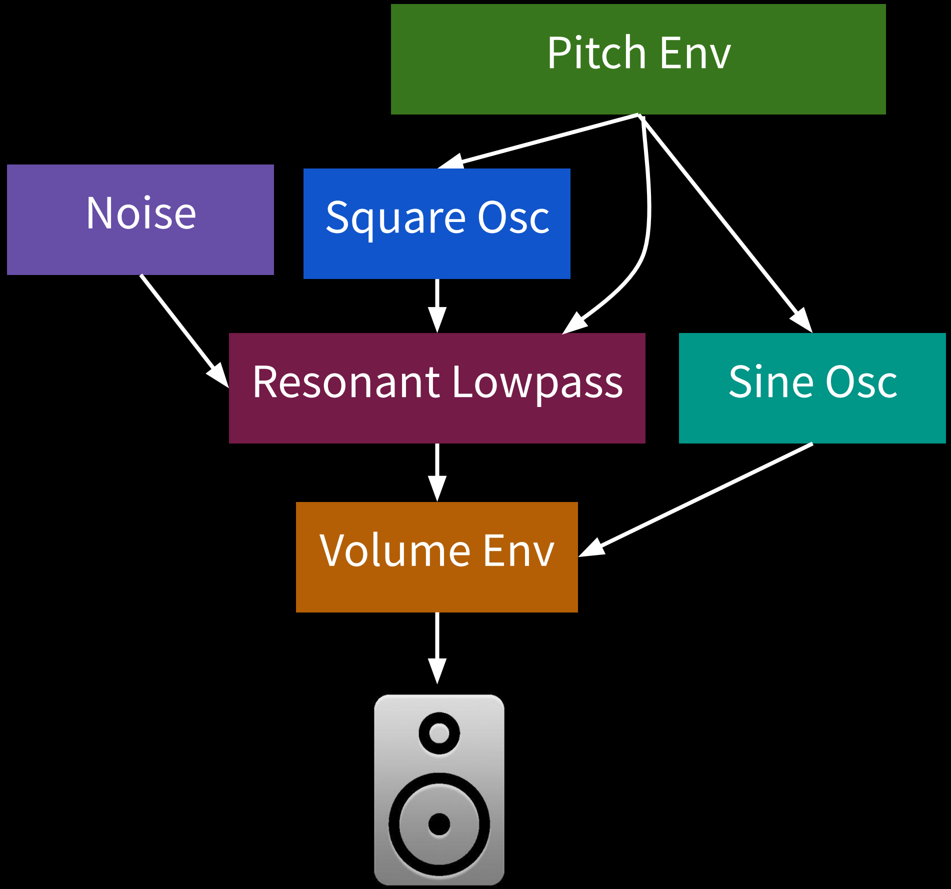
Part 2: Resonant low-pass filter and scales control

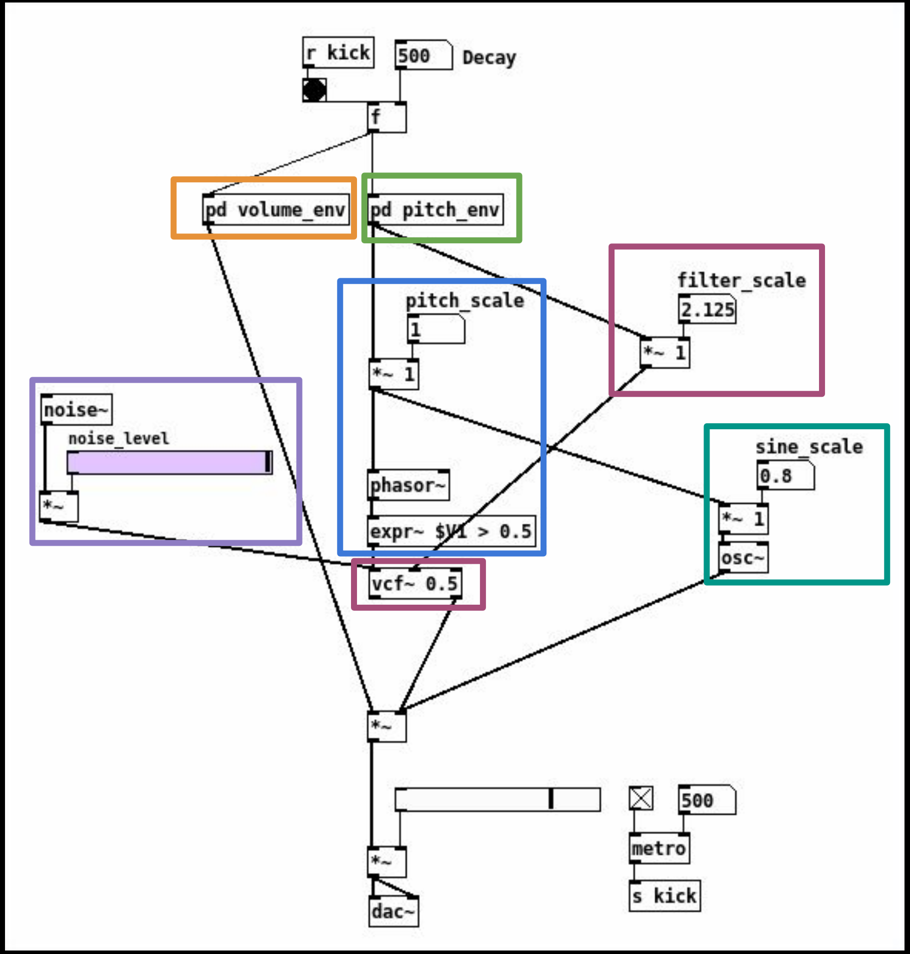




Kick Drum ❤

Part 3: Noise and sine



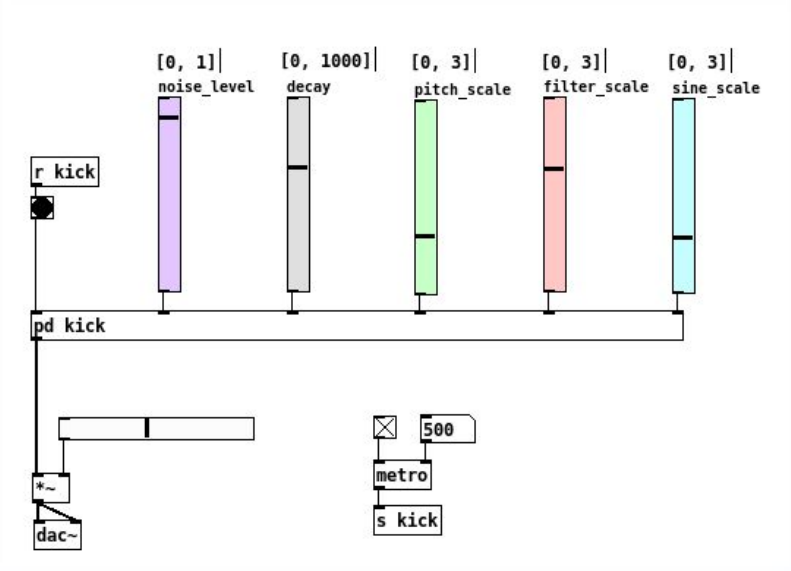


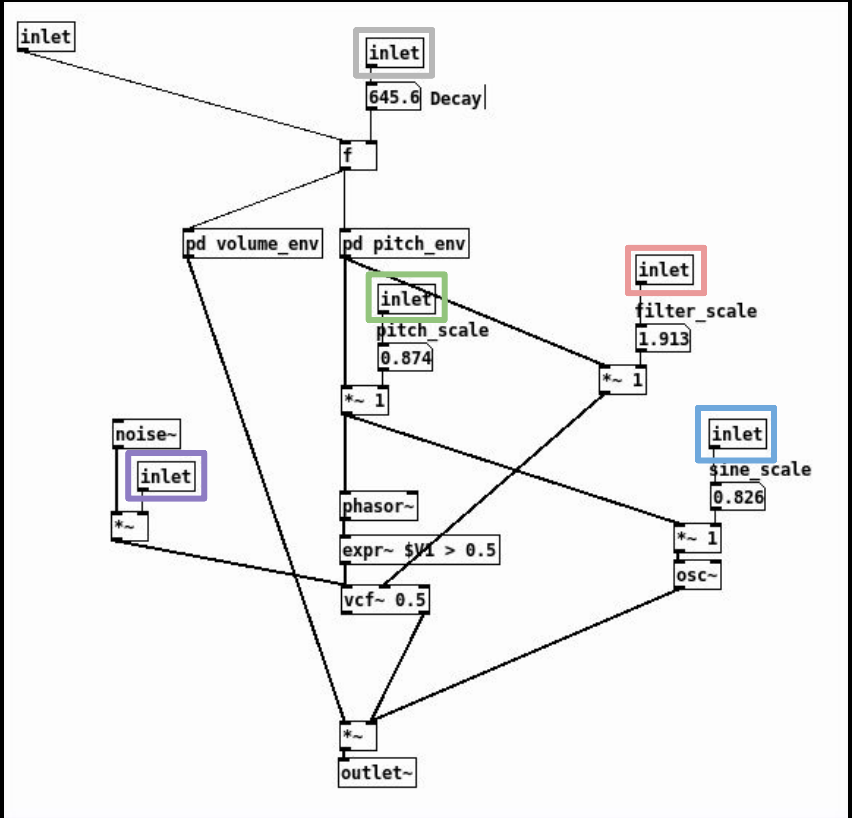
Kick Drum ❤

Part 4: Clean it up

How to simply subpatch:

- put your inlets and outlets appropriately
- cut your boxes with them
- plug respectfully with the order



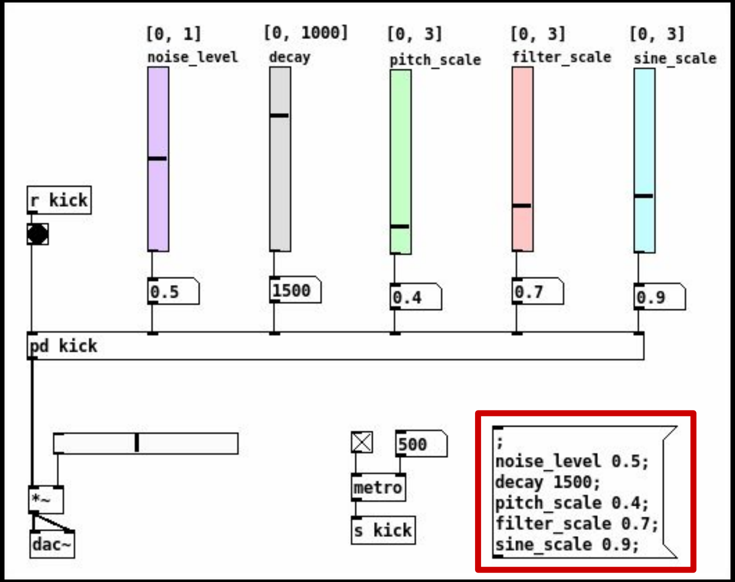


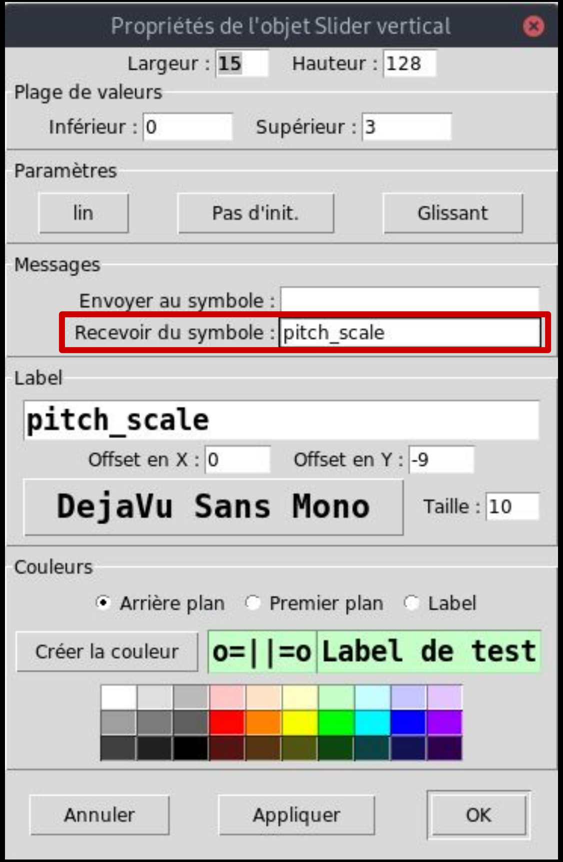
Kick Drum ❤

Part 4: Add preset

We will use messages to build presets.

- First add to all the concern sliders a label in “receive symbol”
- Then write a message beginning with ; followed by all the label you want to drive





Externals love ❤

- So what is going on **inside** a given box ?
- We can even go deeper in Pure Data objects
- Possibility to ***define your own*** **boxes** :)
- The overall system defines **PD *externals***
- Here we will code in C but still talk about *objects*
- PD provides a set of *includes* and *specs*
- Simple SDK with a clear notation
- Entirely dynamic linking / Runtime class loading
- Everything defined as a C struct
- Then simply a set of functions

Externals love

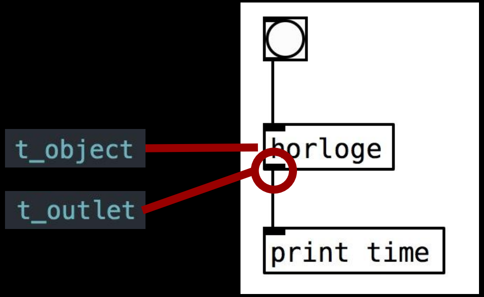
```c
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# include "m_pd.h"

static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object          x_obj;
    t_outlet          *h_out;
}
t_horloge;
```

1. We need to **include the** PD header definitions
2. Then define the **class of our object**
3. **This object reference** is mandatory (cf. later)
4. Need to **manually define inlets and outlets**

All objects have a default left-most hot inlet



Here we want to code a simple object

=> **bang** prints time

Externals love

```c
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# include "m_pd.h"

static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object          x_obj;
    t_outlet          *h_out;
}
t_horloge;

/* Q.2 - Chargement en mémoire des objets de type horloge */
void                horloge_setup(void);
/* Q.3 - Création d'un nouvel objet horloge */
void                *horloge_new(void);
/* Q.4 - Comportement de l'objet en cas de message bang */
void                horloge_bang(t_horloge *x);

#endif
```

1. We need to **include the** PD header definitions
2. Then define the **class of our object**
3. This **object reference** is mandatory (cf. later)
4. Need to **manually define inlets and outlets**

3 minimal functions to code

1. What happens at runtime (**once**)
2. Define **object creation** (add box)
3. **One method per message**

(here we code what happens when a bang is received)

Externals love

1. Runtime function (*_setup)

```
static t_class       *horloge_class;
typedef struct       _horloge
{
    t_object           x_obj;
    t_outlet           *h_out;
}
t_horloge;
```

Reminder of the data structure

Class creation method **class_new**

```
void              horloge_setup(void)
{
  horloge_class = class_new(gensym("horloge"),
                            (t_newmethod)horloge_new,
                            0, sizeof(t_horloge),
                            CLASS_DEFAULT, 0);
  class_addbang(horloge_class, horloge_bang);
}
```

Name of the object

Method to call for each new object

Size / malloc options

*Add the behavior for bang with **class_addbang***

*Later we will also use **class_addmethod** (messages)*

Externals love ❤

2. Box creation function (*_new)

```c
static t_class       *horloge_class;
typedef struct       _horloge
{
    t_object            x_obj;
    t_outlet            *h_out;
}
t_horloge;
```

Reminder of the data structure

```c
void                *horloge_new(void)
{
   t_horloge      *h;

   h = (t_horloge *)pd_new(horloge_class);
   h->h_out = outlet_new(&h->x_obj, &s_symbol);
   return (void *)h;
}
```

Instanciation method **pd_new**

Returned object (void *)

Create the (symbol) outlet and store in x_obj !

Return the created object

Externals love ❤

3. Message handling function (*_bang)

```c
static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object          x_obj;
    t_outlet          *h_out;
}
t_horloge;
```

Reminder of the data structure

Specific object instance

```c
void                horloge_bang(t_horloge *x)
{
  time_t            rawtime;
  struct tm         *timeinfo;

  time(&rawtime);
  timeinfo = localtime(&rawtime);
  outlet_symbol(x->h_out, gensym(asctime(timeinfo)));
}
```

Need to write symbols to a given symbol table

Write information to a specific outlet