Pure Data

❖ Intermediate course ❖

pd~

Ninon Devis

https://ninon-io.github.io/

Made with love for ATIAM

Table of Content

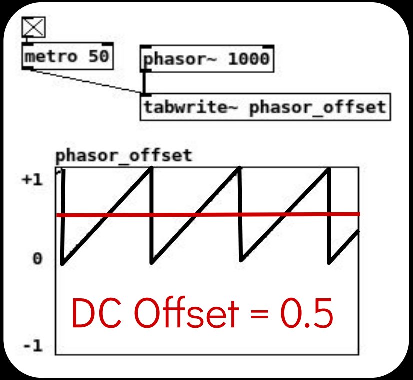
01

Processing

Signal normalization & various synthesis methods

NORMALIZING THE SIGNAL

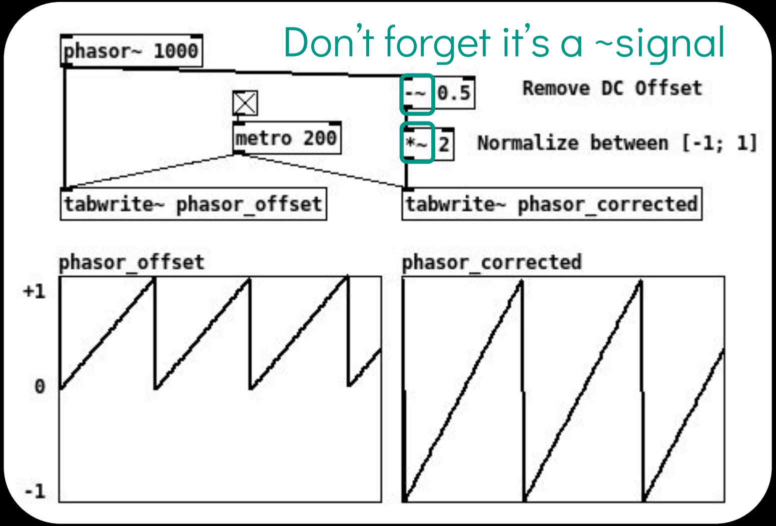
How to normalize this signal ?



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].

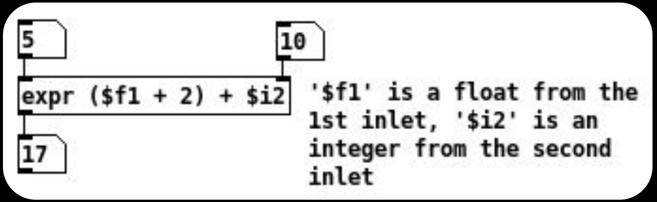
NORMALIZING THE SIGNAL

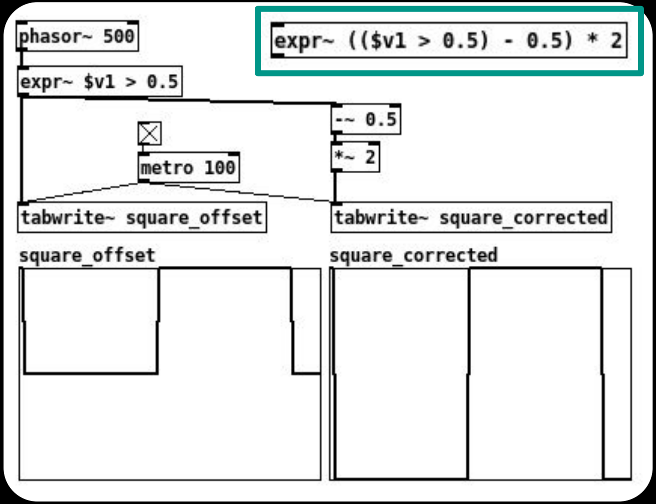
How to convert a sawtooth into a square?

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

- 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

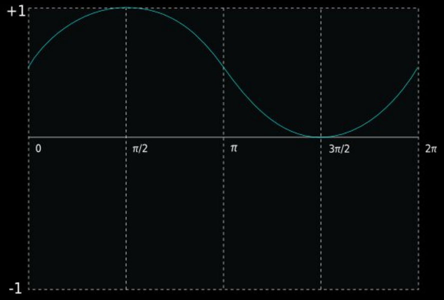
MODULATION

Multiplying audio signal

Difference between AM & RM

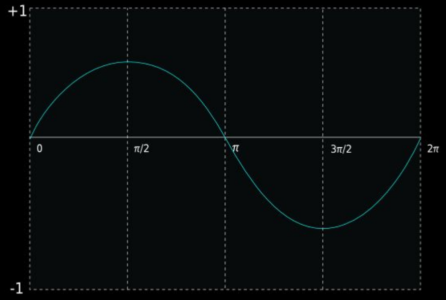
Modulation is typical in synthesis as it enriches the character of the sound and adds variance in timbre over time.

Unipolar signal



DC offset

Bipolar signal



Mean amplitude around 0

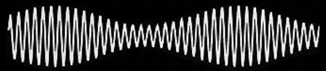
- ❖ Ring Modulation: multiplication of two bipolar signals by each others
- ➔ The frequency of the carrier signal is not present in the resulting sound.
- ❖ Amplitude Modulation: M is a unipolar modulator, typically between 0 and 1.
- ➔ The carrier frequency is preserved.

Reminder: Example for AM

Carrier signal

Modulating signal

Output



$R(t)$ ring modulation output

$A(t)$ amplitude modulation output

$C(t)$ carrier signal

$M(t)$ modulation signal

$$R(t) = C(t) \times M(t)$$

$$A(t) = C(t) \times (M(t) + 1)$$

Ring Modulation

$$R(t) = C(t) \times M(t)$$

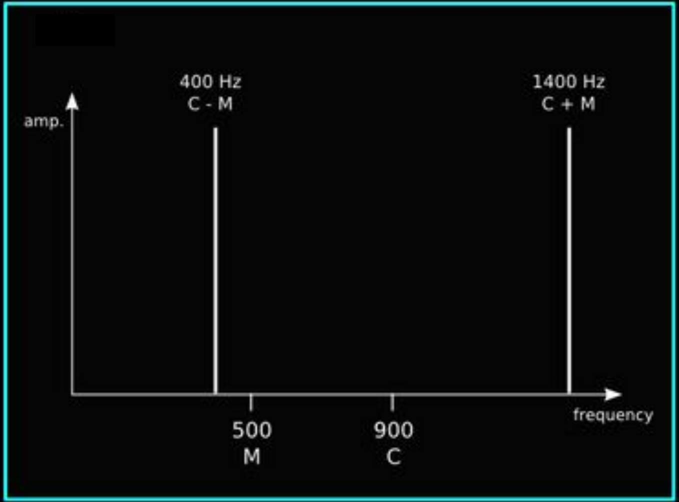
We multiply 2 bipolar signals by each other resulting:

$$sin(\alpha n + \phi)sin(\beta n + \xi) = \frac{1}{2}(cos(\boxed{(\alpha+\beta)}n+(\phi+\xi))+cos(\boxed{(\alpha-\beta)}n+(\phi-\xi)))$$

We obtain two partials, one at the sum of the two original frequencies and one at their difference.

- This shifts the component frequencies of a sound

Doctor Who
Cyberman voice



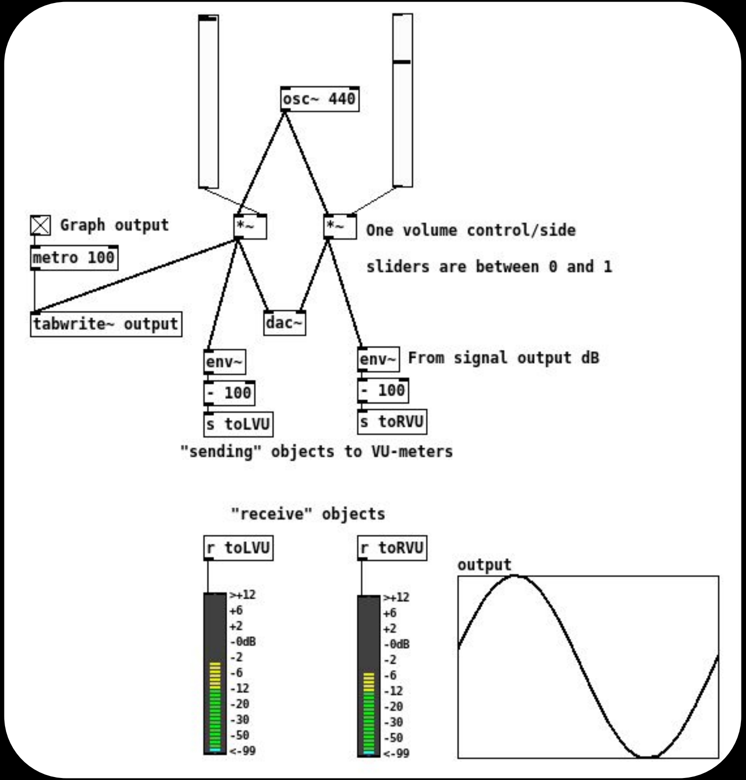
STEREO PATCH

- 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 as [s *] and [r *]

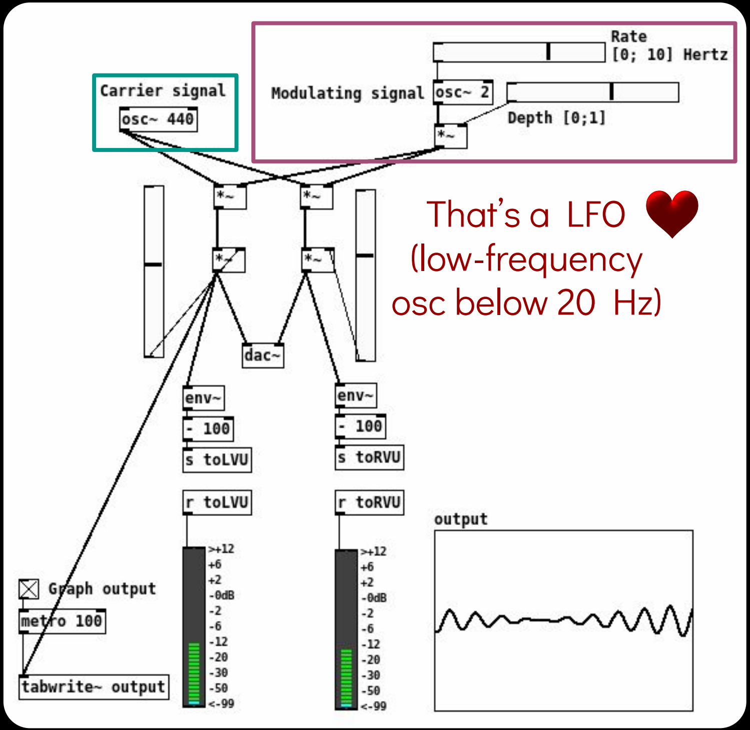
- To use the VU-meter we need [env~] which take a signal and output its RMS amplitude in dB.

Stereo Oscillator



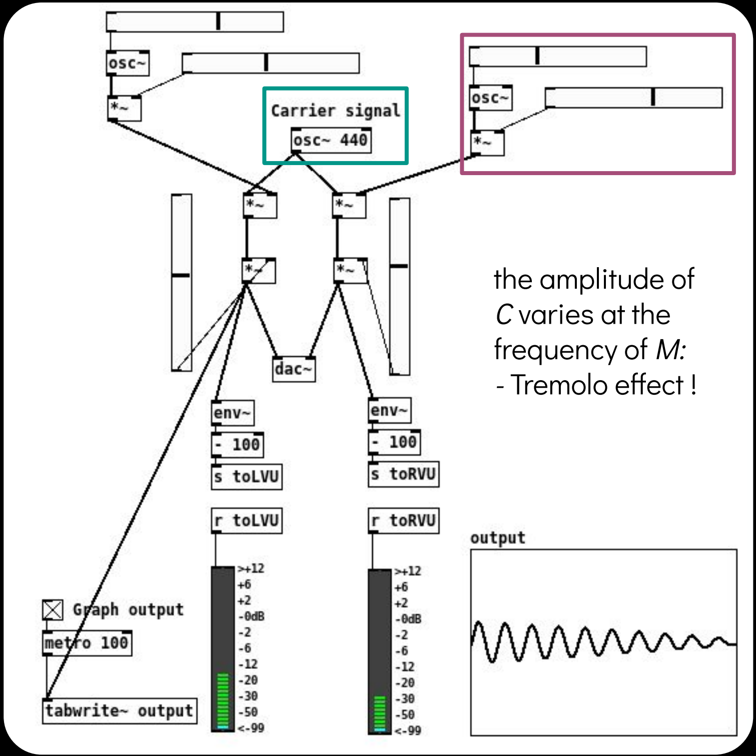
RM SYNTHESIS Using one oscillator to modulate the gain of an other one

- According to your intuition, how would you patch a RM?



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

- Let's create a tremolo from this: adding RM on both sides

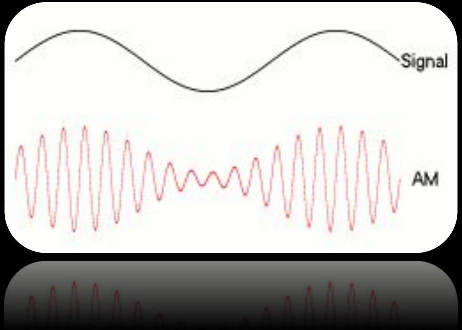


the amplitude of *C* varies at the frequency of *M*:
- Tremolo effect !

Amplitude Modulation

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).



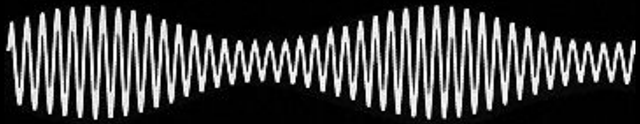
Earliest method to transmit audio in radio broadcast

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

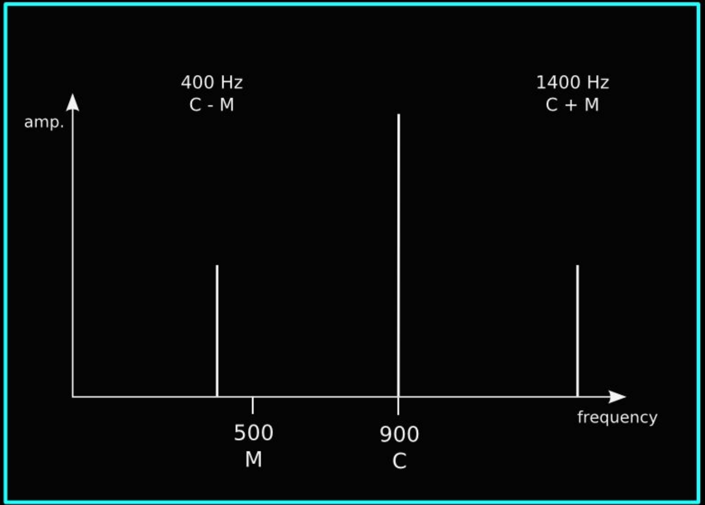
$$A(t) = C(t) \times (M(t) + 1)$$

The modulator M is unipolar, typically set between 0 and 1.

$$\frac{A_c A_m}{4}\left[\sin\left(2\pi\boxed{(f_c - f_m)}t + \frac{\pi}{2}\right) + \sin\left(2\pi\boxed{(f_c + f_m)}t - \frac{\pi}{2}\right)\right] + \frac{A_c}{2}\sin\boxed{(2\pi f_c t)}$$

The carrier frequency is preserved and the sidebands generated are at *half* the amplitude of the carrier amplitude.

Having the carrier frequency, it is then possible to demodulate the signal in order to access the information hold by the carrier using a pass band.



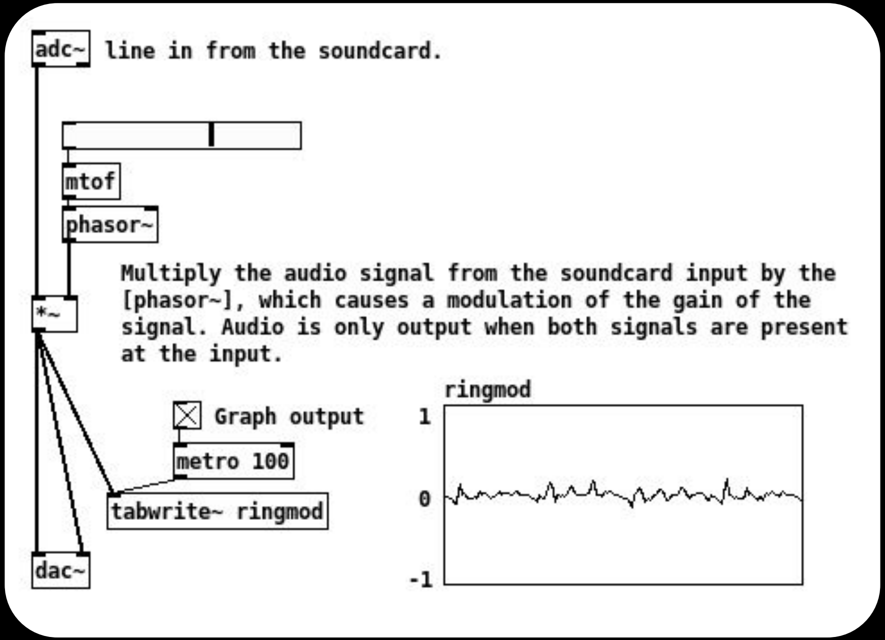
AM WITH COMPLEX SIGNAL

Making Alien's voice with amplitude modulation

Which oscillator would you choose ?

Which object will we use to catch our voice ?

➔ [adc~]



Frequency Modulation

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.

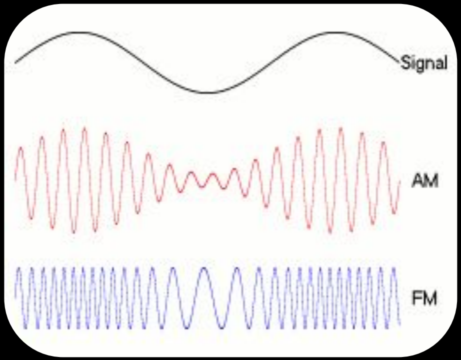
Mathematical intuition:

sinusoid modulated by another sinusoid.

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

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

Stria - John Chowning (1977)



Difference between AM and FM

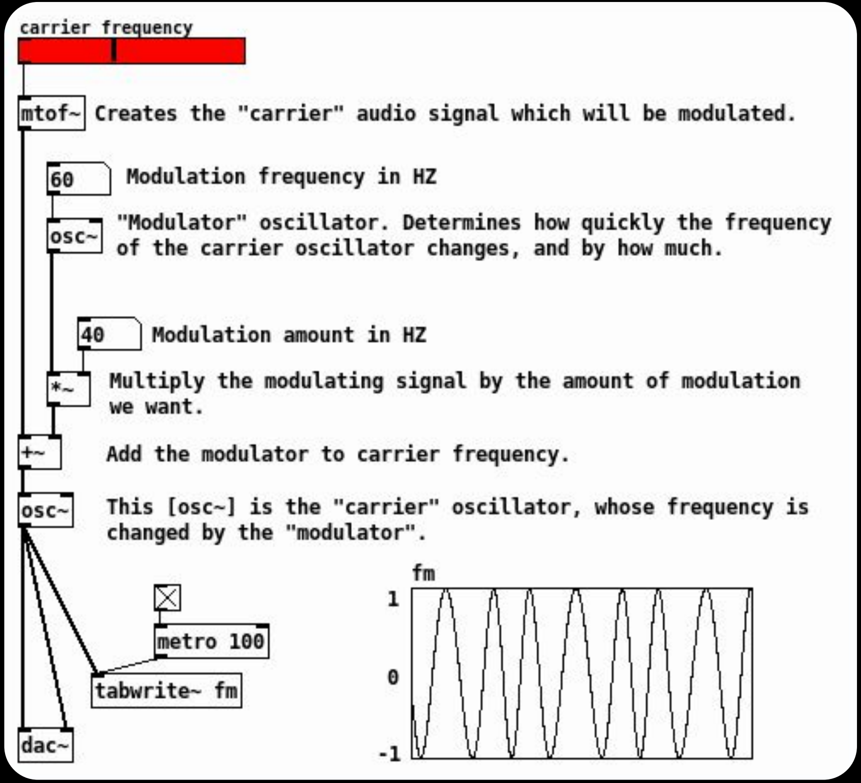
Carrier signal f_c

Modulating signal f_m

Frequency Modulation

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

Simple fm patch



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

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

02

Miscellaneous

Step sequencer, sound files & Kicks

THE COUNTER

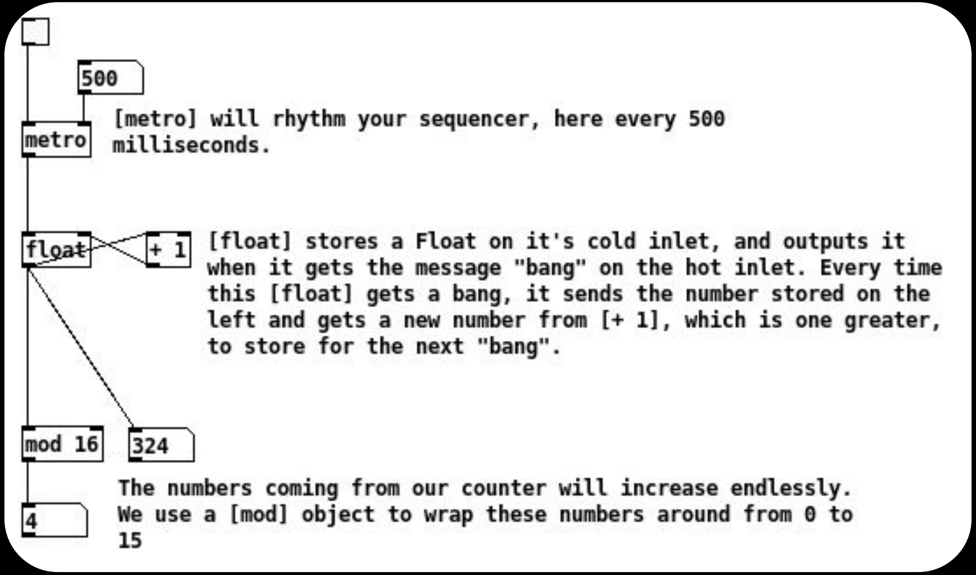
Definition of step sequencer:

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 steps sequencer

- You will probably need [mod *] which wrap number around given value.
- we want as output an increasing number modulo 16.

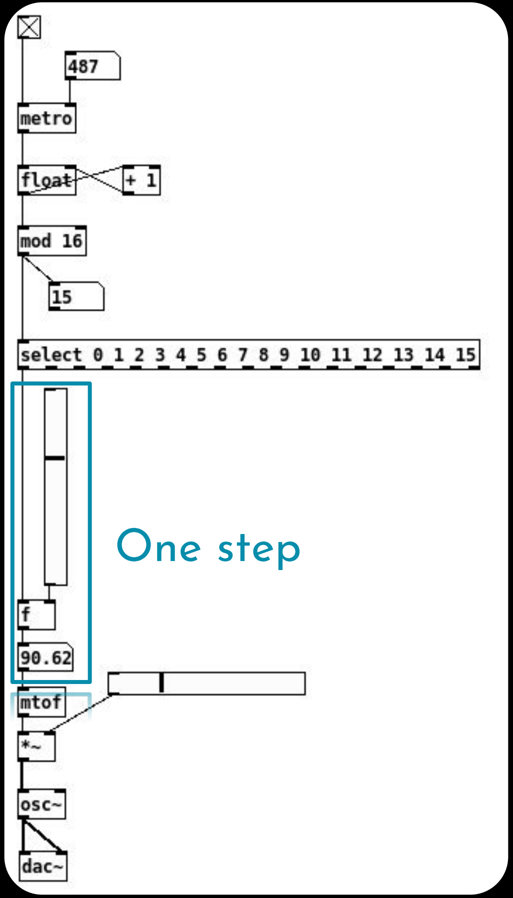


The Step Sequencer

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 FILES

Reminder about files formats

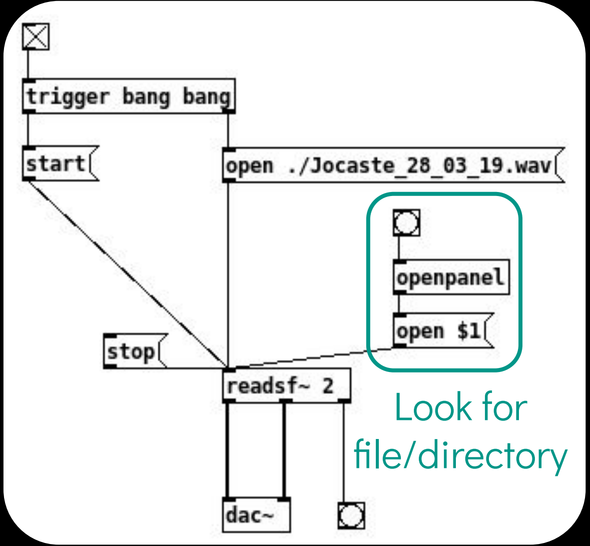
- No compression: .wav or .aiff
- Compression but no quality loss: .flac
- Compression and quality loss: .mp3

➔ Pd objects depending on the format

[readsf~] for .wav

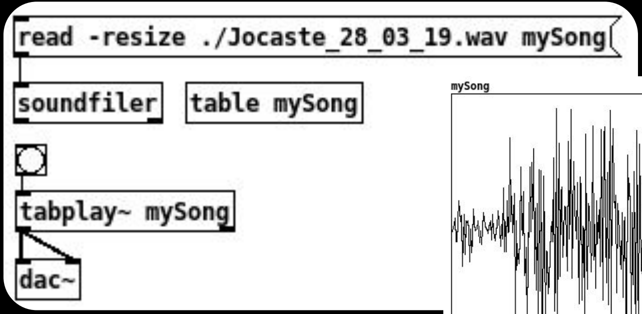
other formats not included in Pd Vanilla

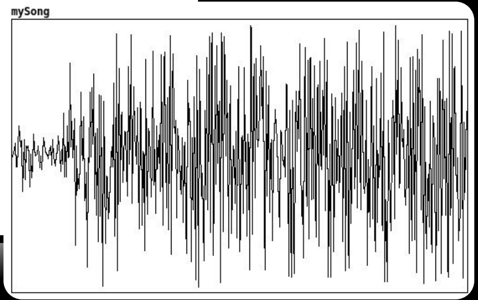
- Don't use *space* or *special characters* for the name of your sound files
- Try to put your music in the *same folder* as your patch



How would you loop the song ?

[read[and [soundfiler] to "edit" the song



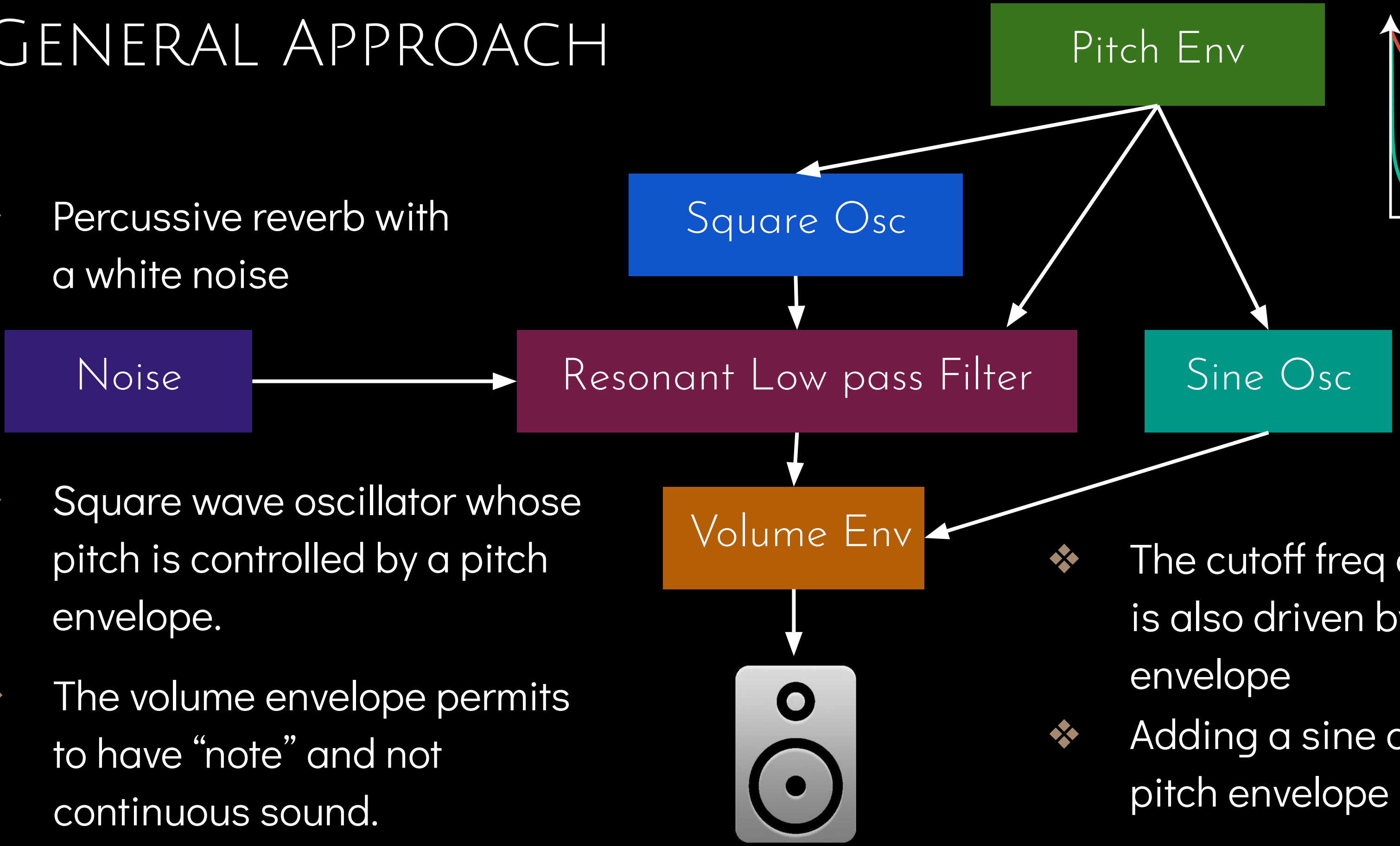


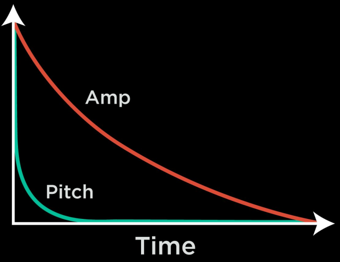
Seek help

KICK

GENERAL APPROACH

❖ Percussive reverb with a white noise





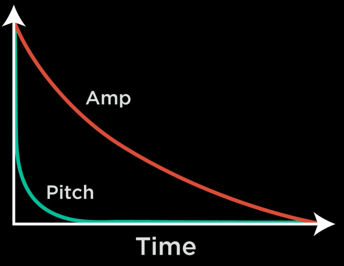
❖ Square wave oscillator whose pitch is controlled by a pitch envelope.

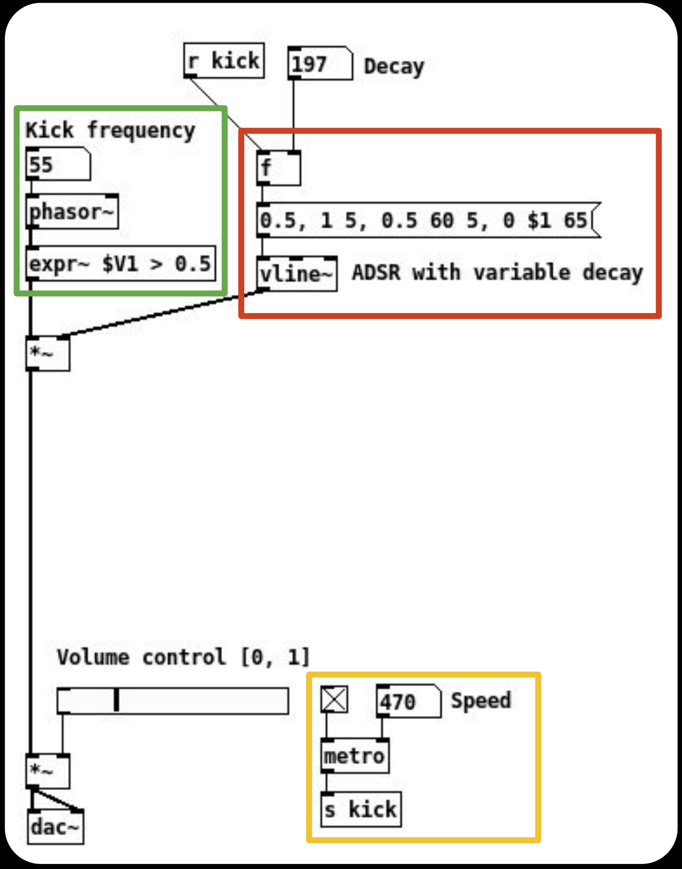
❖ The volume envelope permits to have “note” and not continuous sound.

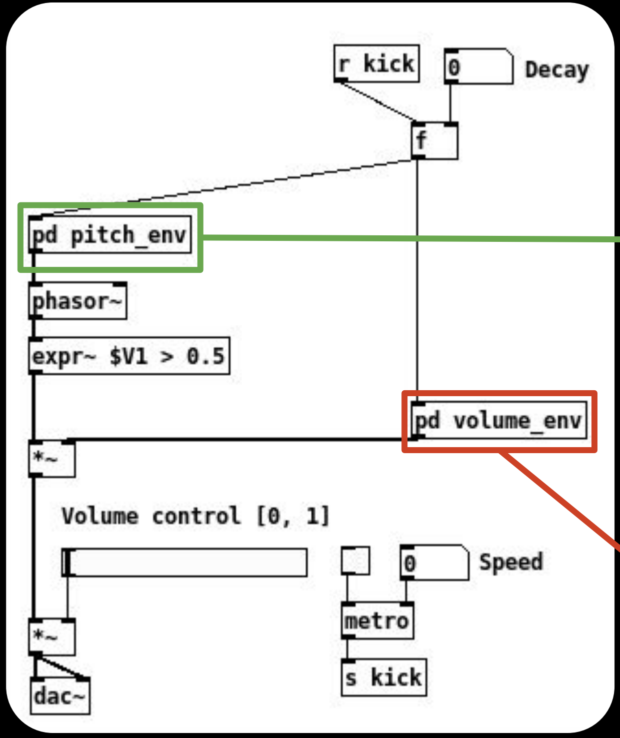
❖ The cutoff freq of the lowpass is also driven by the pitch envelope

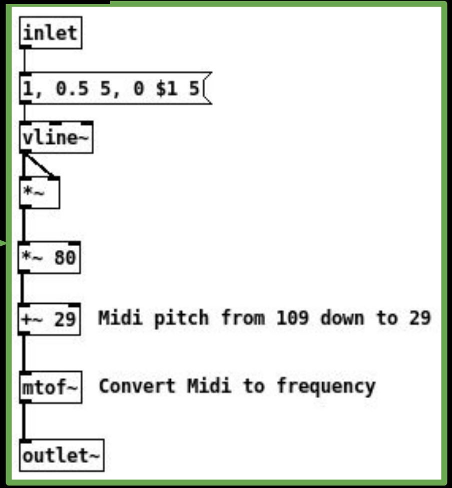
❖ Adding a sine driven by the pitch envelope

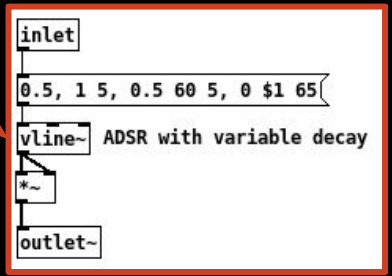
Phasor and Envelope



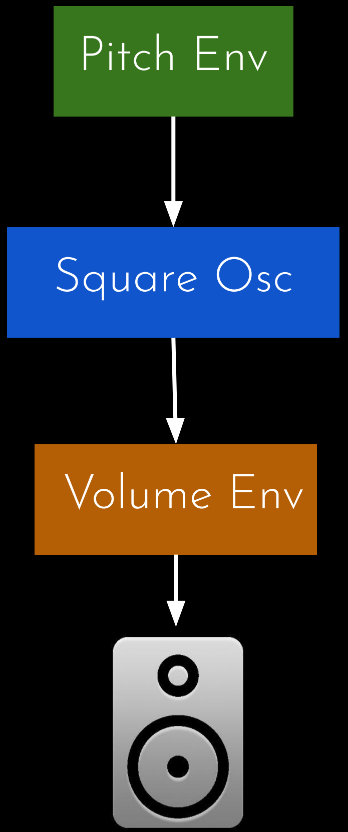








First Step Achievement

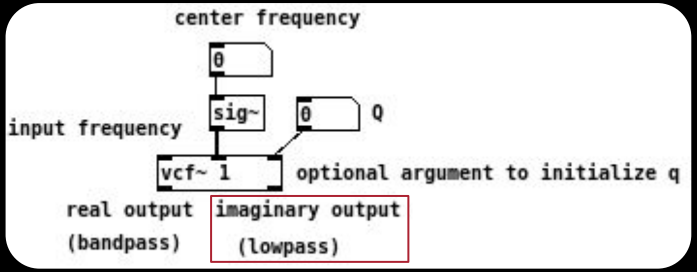


Next :

adding a resonant low-pass filter,
With the cutoff driven by our pitch enveloppe.

- Which object do we need ?

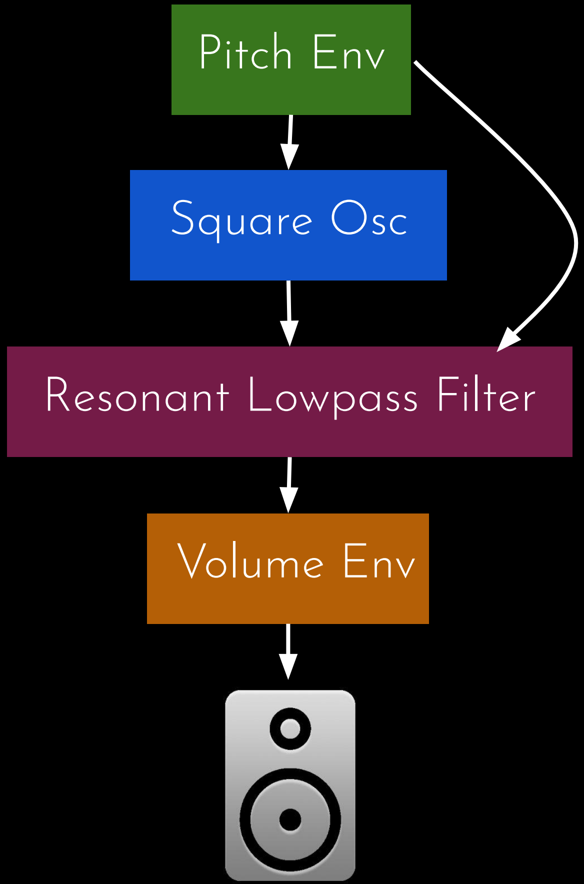
VCF :

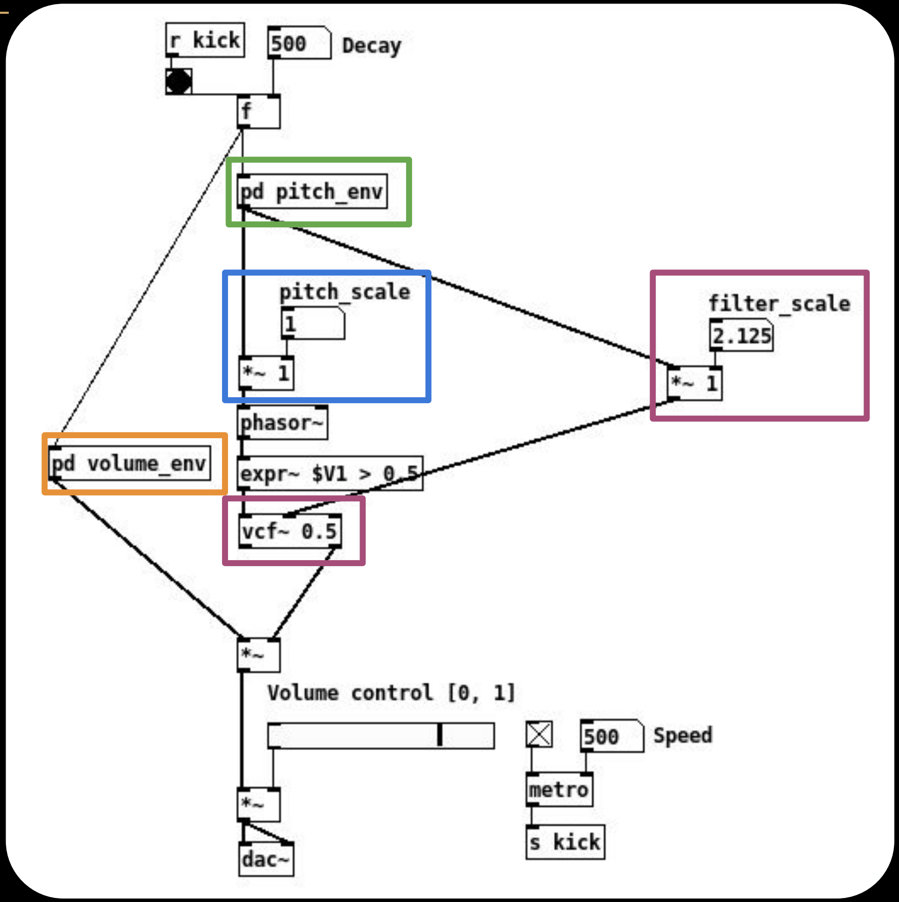


Where will we plug our pitch env ?

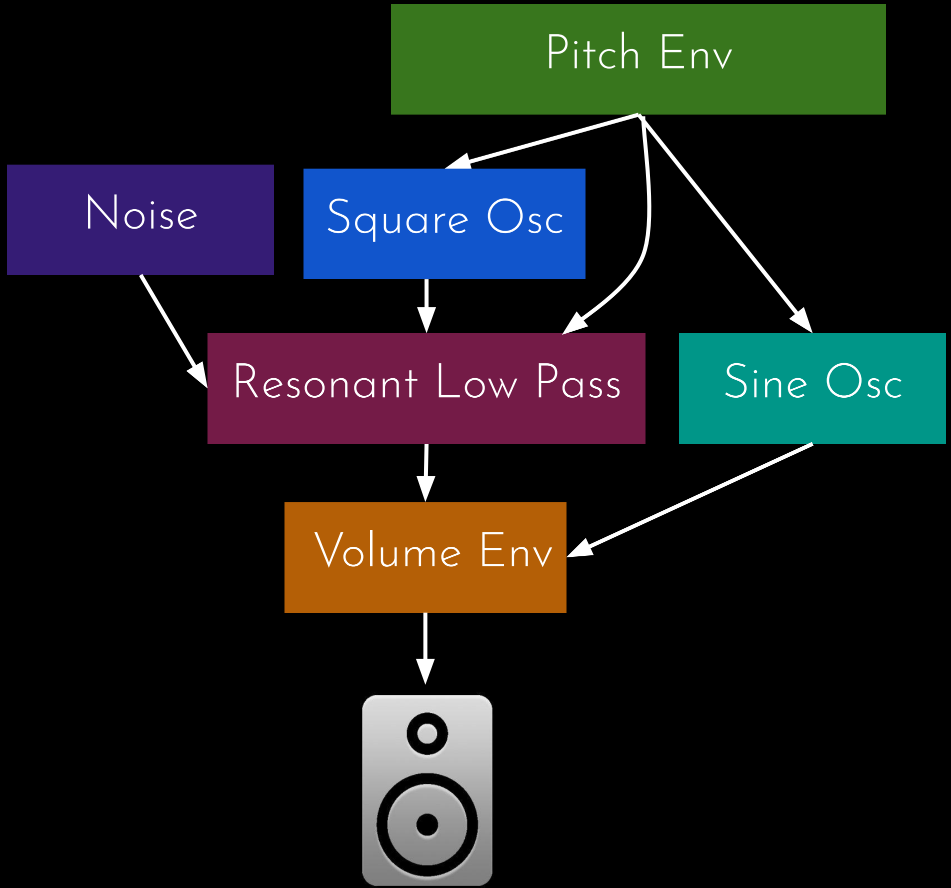
First, let's add a pitch scale, then the vcf with a filter scale.

Resonant Low Pass Filter & Scales

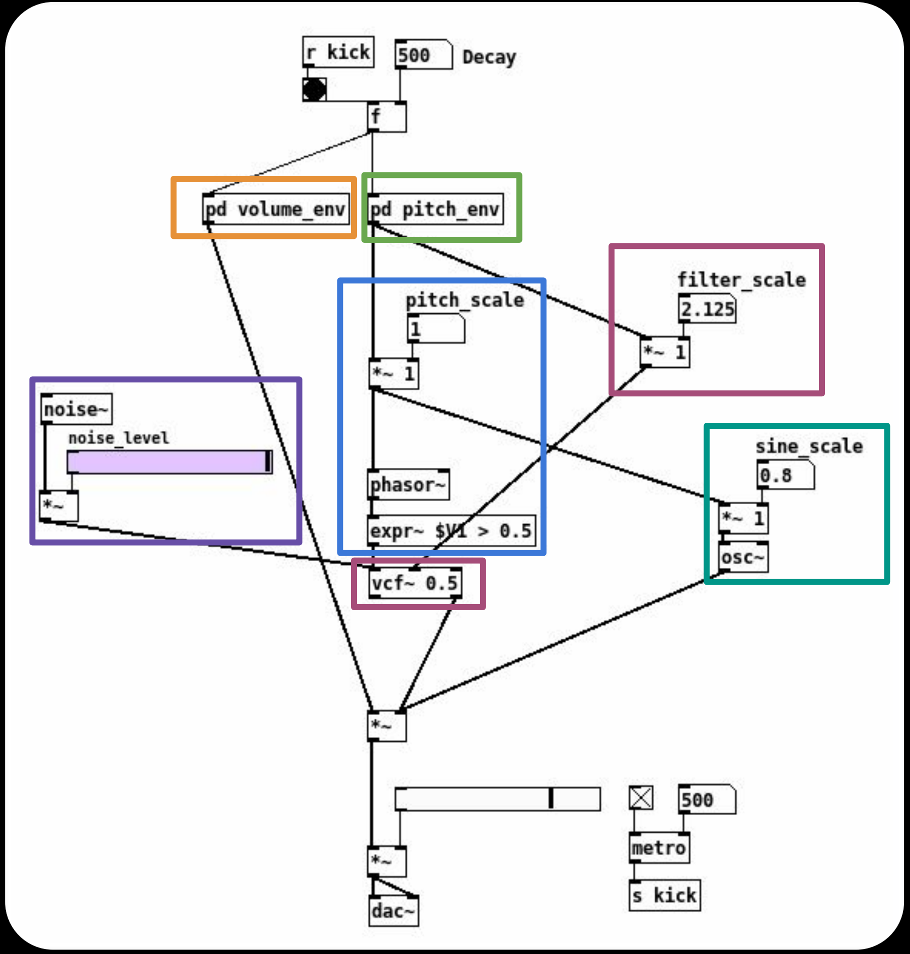




Noise & Sine



Pitch Env
Noise
Square Osc
Resonant Low Pass
Sine Osc
Volume Env

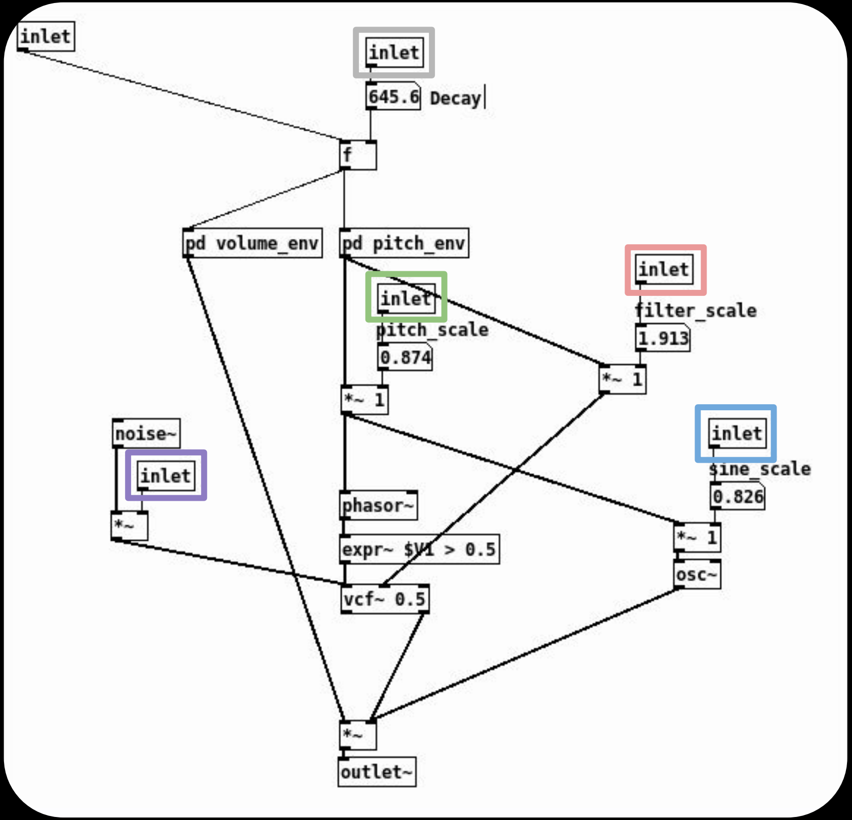


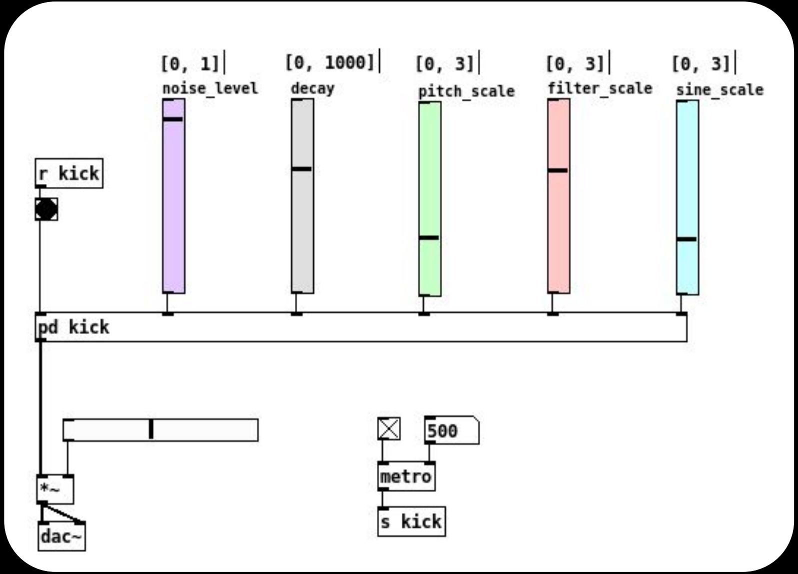
r kick
500
Decay
f
pd volume_env
pd pitch_env
pitch_scale
1
*~ 1
filter_scale
2.125
*~ 1
noise~
noise_level
*~
phasor~
expr~ $V1 > 0.5
vcf~ 0.5
sine_scale
0.8
*~ 1
osc~
*~
500
metro
s kick
*~
dac~

Clean it Up

How to simply subpatch:

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

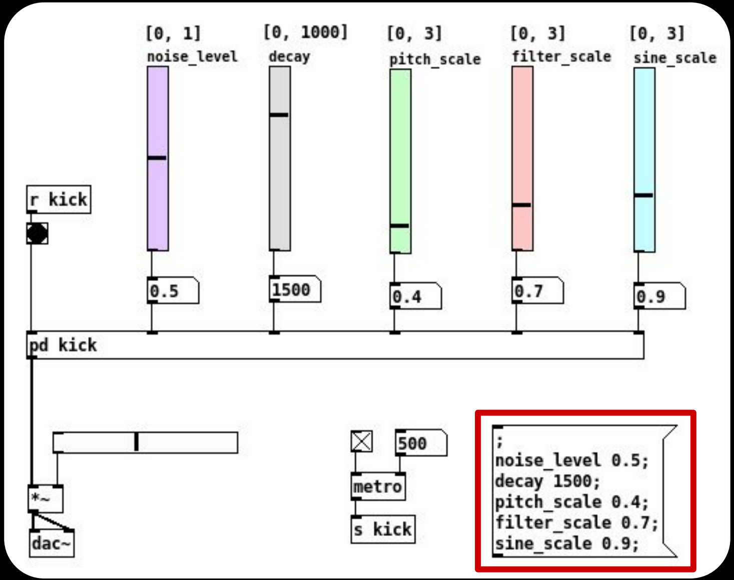


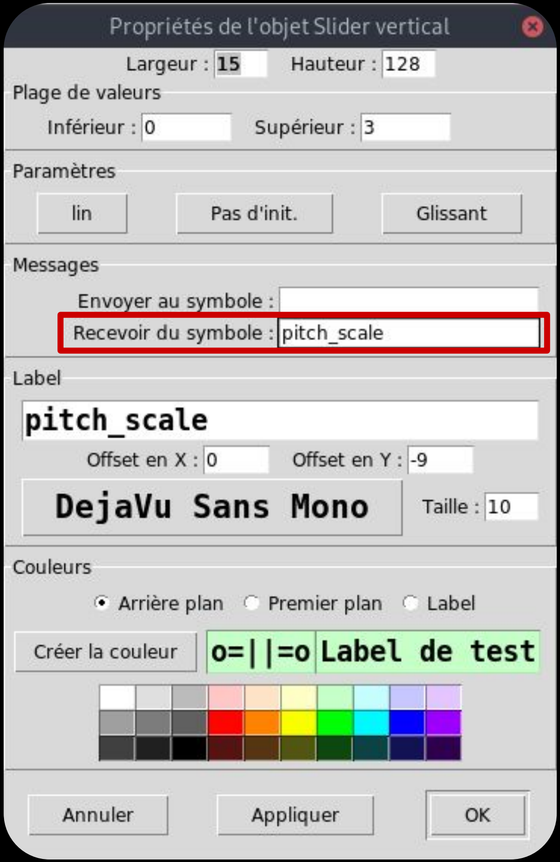


ADD PRESETS

Messages boxes to add 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





03

EXTERNALS

Create your own PD boxes

Externals love

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

Externals love

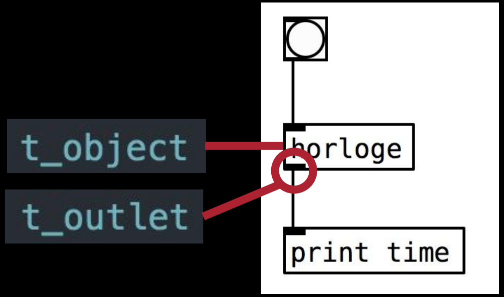
```c
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# 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**

This **object reference** is mandatory (cf. later)

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**

This **object reference** is mandatory (cf. later)

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

Reminder of the data structure

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

1. Runtime function (*_setup)

- Global explanation of the object
- Explains its name, types and functions
- Mimics a class system

Class creation method **class_new**

```c
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

Reminder of the data structure

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

2. Box creation function (*_new)

- Function called when we create a box
- Similar to an object *constructor*
- Explain all initialization stuff

```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;
}
```

Instantiation method **pd_new**

Returned object (void *)

Create the (symbol) outlet and store in x_obj !

Return the created object

Externals love

Reminder of the data structure

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

3. Message handling (*_bang)

- Function called when we receive a message
- Here specific example of a *bang*
- Beware of the processing time !

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)));
}
```

Just get the current time

Need to write symbols to a given symbol table

Write information to a specific outlet

Externals love

What happens for signal stuff ?

```c
/** Q.5 - Fonction centrale effectuant le calcul */
t_int            *myfft_tilde_perform(t_int *w);
/** Q.4 - Ajout de l'objet myfft~ à l'arbre de traitement DSP */
void             myfft_tilde_dsp(t_myfft_tilde *x, t_signal **sp);
/** Q.3 - Libération de la mémoire de l'objet myfft~ */
void             myfft_tilde_free(t_myfft_tilde *x);
```

1. Our own perform function
2. The DSP call (block_size dependent)
3. Memory liberation

```c
void                    myfft_tilde_setup(void)
{
  myfft_tilde_class = class_new(gensym("myfft~"),
                                (t_newmethod)myfft_tilde_new,
                                0, sizeof(t_myfft_tilde),
                                CLASS_DEFAULT,
                                A_DEFFLOAT, 0);
  class_addmethod(myfft_tilde_class,(t_method)myfft_tilde_dsp, gensym("dsp"), 0);
  CLASS_MAINSIGNALIN(myfft_tilde_class, t_myfft_tilde, f);
}
```

Similar class setup method

Need to add the DSP function

```c
void                    myfft_tilde_dsp(t_myfft_tilde *x, t_signal **sp)
{
  dsp_add(myfft_tilde_perform, 4, x, sp[0]->s_vec, sp[1]->s_vec, sp[0]->s_n);
}
```

DSP call fills the rightful buffers