ircam Centre Pompidou

pd~

Introduction to ~Pure Data~

Ninon Devis: ninon.devis@ircam.fr

Made with love for ATIAM

What’s PD?

- Developed by Miller Puckette @Ircam (1988) => Max MSP
- Open-source visual programming language
- Can even run on Raspberry Pi & smartphones
- Two major components:

Starting

Go to www.puredata.info for precompiled version

(GNU/Linux, Mac OS X & Windows compatible)

➔ Open Pd!

The first opened window is a console which displays:

- errors of your patches
- messages when using the “print” object

Open a new patch

Ctrl/Cmd + N

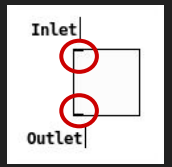
Basics

- *Unlock mode*: enable edition of patch
- *Lock mode*: run the graphical objects

Ctrl/Cmd + E

- *Toggle*: acting as a switch on/off

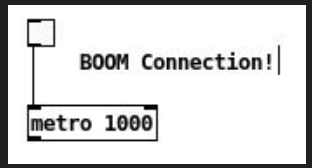
Shift + Ctrl + T



- Metronome: create an object,

  then write “metro 1000” inside

Ctrl + Shift + 1



- *Bang*: acting as a *trigger*

Ctrl/Cmd + B

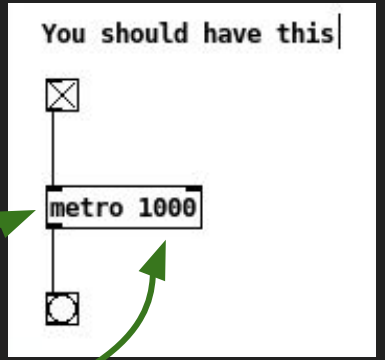
Always patch from *up to down* and *right to left*:
that’s how the signal flows in PD!

Basics

5 Types of Pd boxes

Objects

Ctrl + Shift + 1



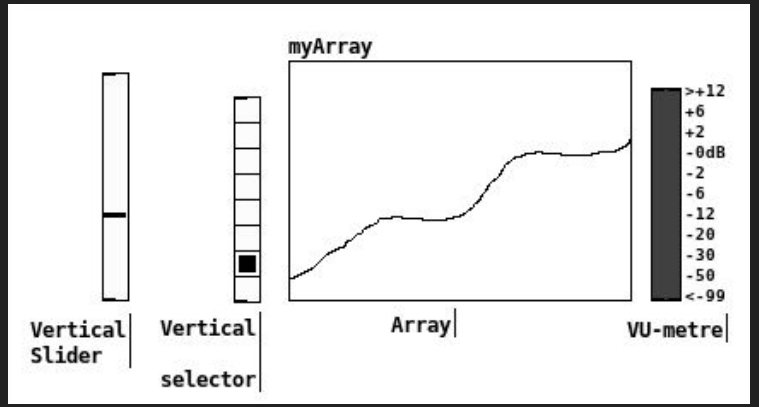
Your first patch \o/

Objects in Pd:

- convention [name_object] in this course
- function: defined by the first string
- argument(s): following item(s)

Examples:

Graphical ones:



Kind of complete list of Pd objects:

http://blazicek.net/list_of_pure_data_objects.html

Basics

5 Types of Pd boxes

Messages: Ctrl + Shift + 2

can contain *any types* of data and send it through its outlet.
Can be clicked in lock mode to send the message.

Numbers: Ctrl + Shift + 3

store *integers* and *floats*. Value can be varied by sliding in lock mode.

Symbols: Ctrl + Shift + 4

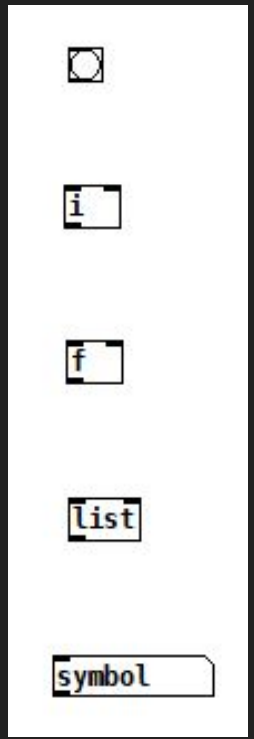
similar to numbers but use string instead of numbers

Comments: Ctrl + Shift + 5

pretty obvious. They are as necessary as usual programming comment. Highly recommended to use them.

Speaking about data types...

- **Bang:** in the bang object
- **Integer:** in the [i] object
- **Float:** in the [f] object
- **List:** in the [list] object
- **Symbol:** in the symbol box



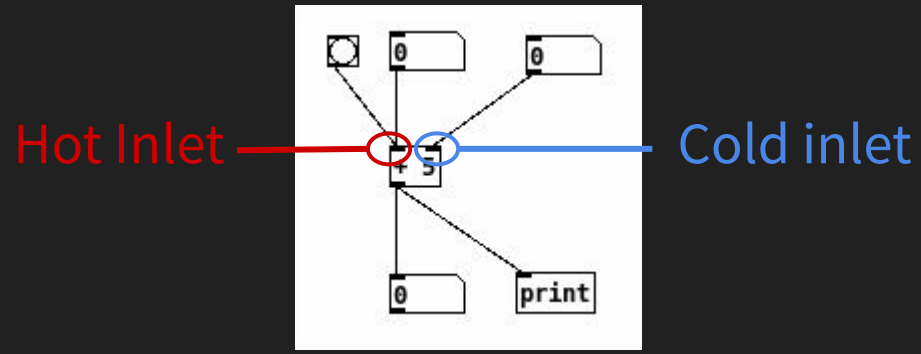
Basics

Hot & Cold Inlets

Most important and complex concept of Pd coming

- Hot Inlet: the left-most. A message sent into a hot inlet *triggered its execution*.
- Cold inlet: the remaining ones. Useful to change the argument of the object, store it into the object but *do not trigger a calculation*.

A bang is used to triggered the cold inlet storage



Basics

Signal flows *up to down* and *right to left*

Can be avoided by using:

- [trigger]
- [route]

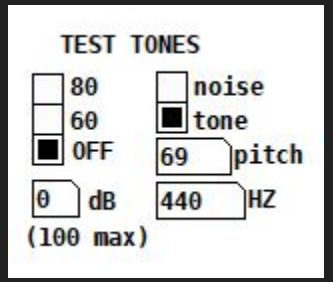
} Find Help ! See examples

Pd has been made to learn ***by yourself***

- Objects help are in patch form
- You can try, modify, copy/paste them…till you get it !

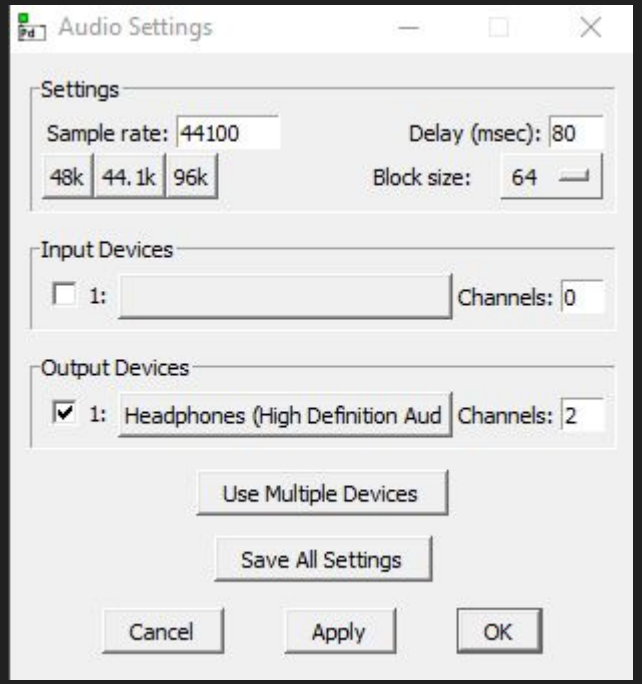
Let’s check the sound

➔ Media > Test Audio and MIDI



If you don’t have any sound:

➔ Media > Audio Settings

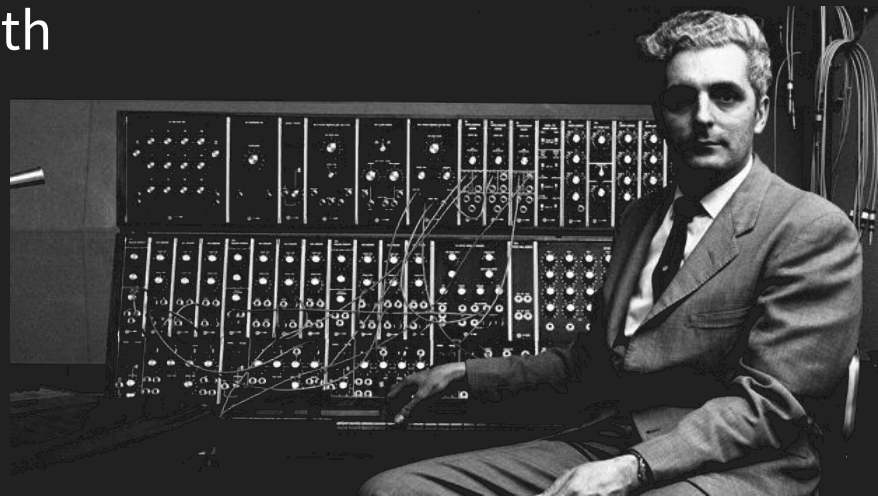


Synthesizer

One of the most fundamental instrument in electronic music ❤️

Originally based on a modular architecture with:

- **Oscillators:** generate the tones
- **Filter:** emphasizes or remove certain frequencies
- **Amplifier:** controls the gain of the synth

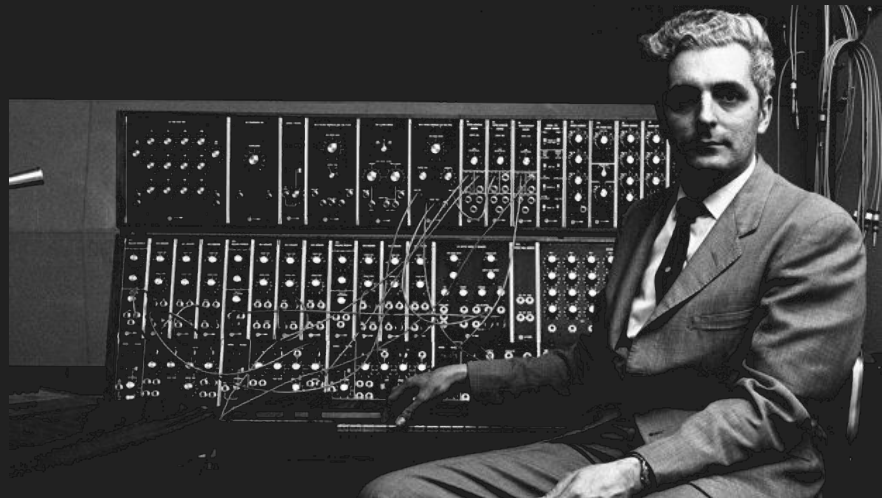


Synthesizer

And some modulation modules:

- **LFO (low frequency oscillator):** modulates either the frequency or gain of the oscillator(s) or frequency of the filters
- **Envelope generator:** controls changes in frequency or gain over the note

Let’s create a simplified Minimoog with Pd ❤️



Oscillators & frequencies

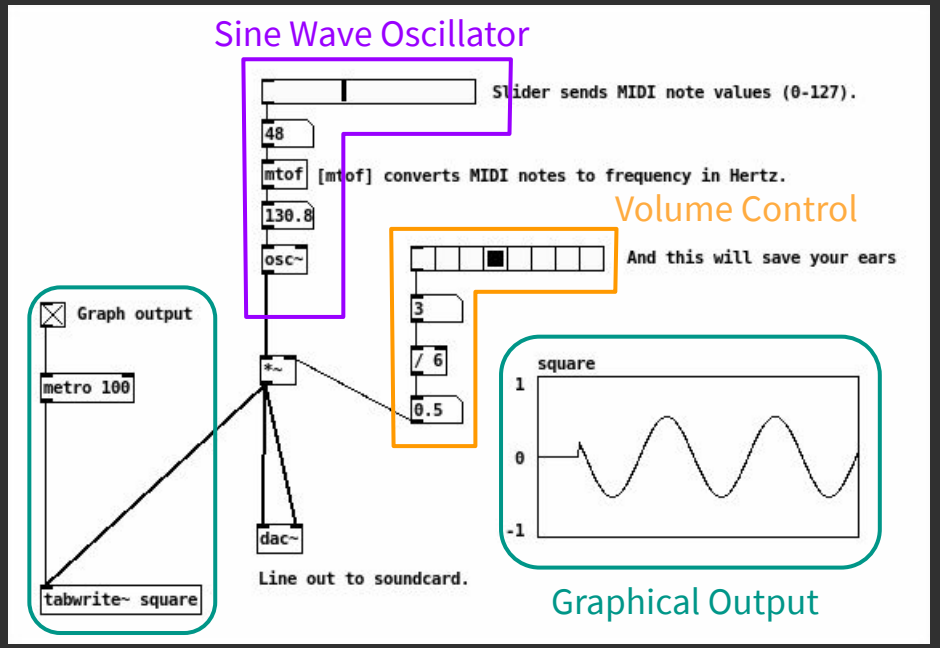
you need:

Mac users:
⌘ instead CTRL

slider	⇧+Ctrl+J
numbers	⇧+Ctrl+3
objects	⇧+Ctrl+1
toggle	⇧+Ctrl+T
selector	⇧+Ctrl+I
comments	⇧+Ctrl+5
table	⇧+Ctrl+A

Audio signal = stream of numbers between -1 and 1

Sine Wave Oscillator



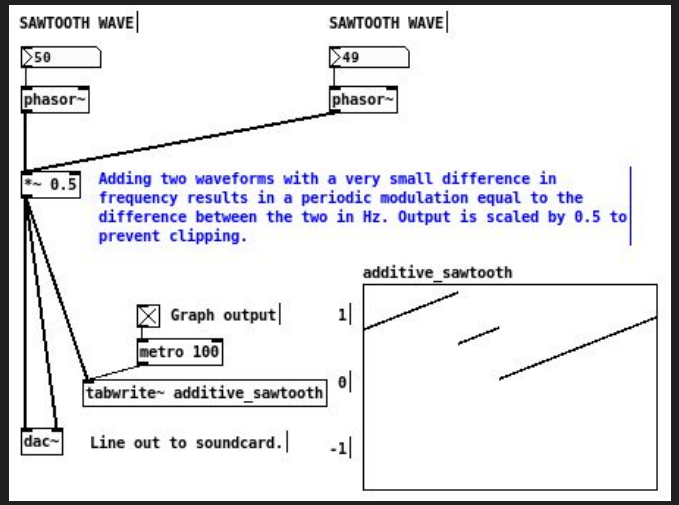
➔ Typical signature: ~ for *audio objects*

Additive Synthesis

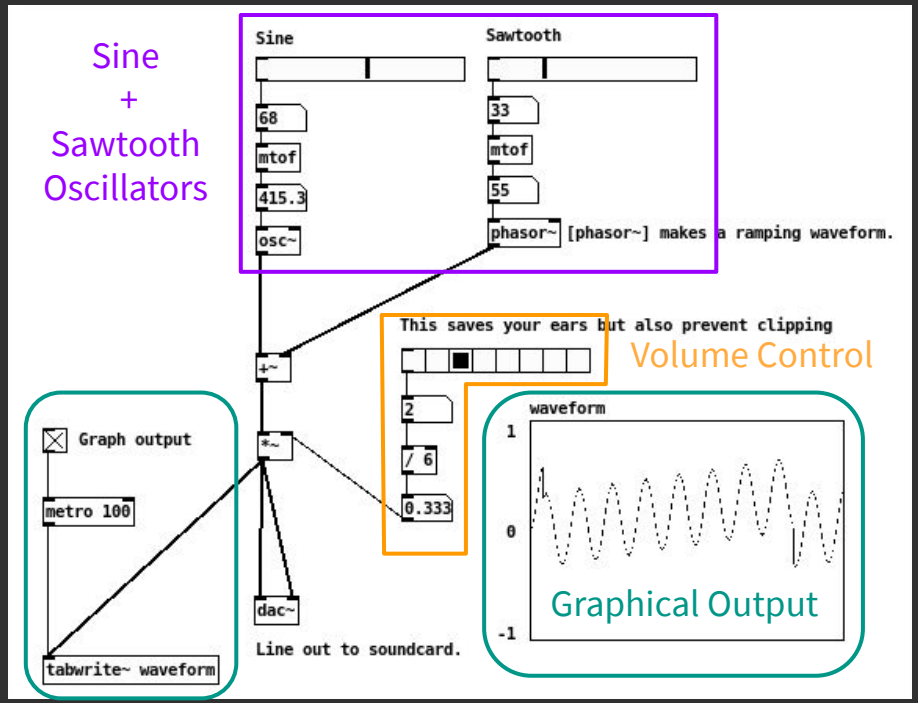
Notice from your patch:

- difference between *number/message* and *audio* cable

Beating frequency for fat bass



Sine + Sawtooth



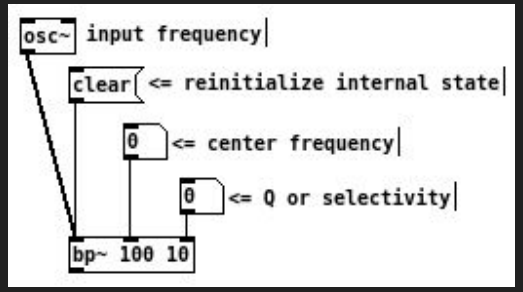
Filters

Easy to manipulate, three classic types:

Low pass: [lop~]

High pass: [hip~]

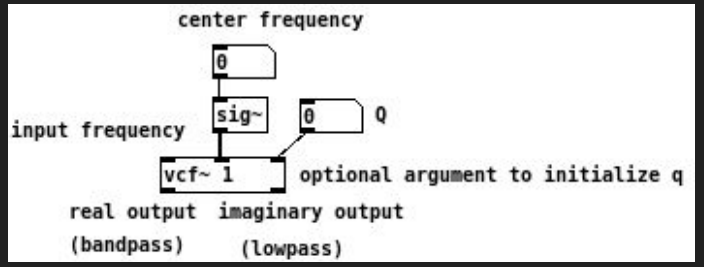
Band pass: [bp~]



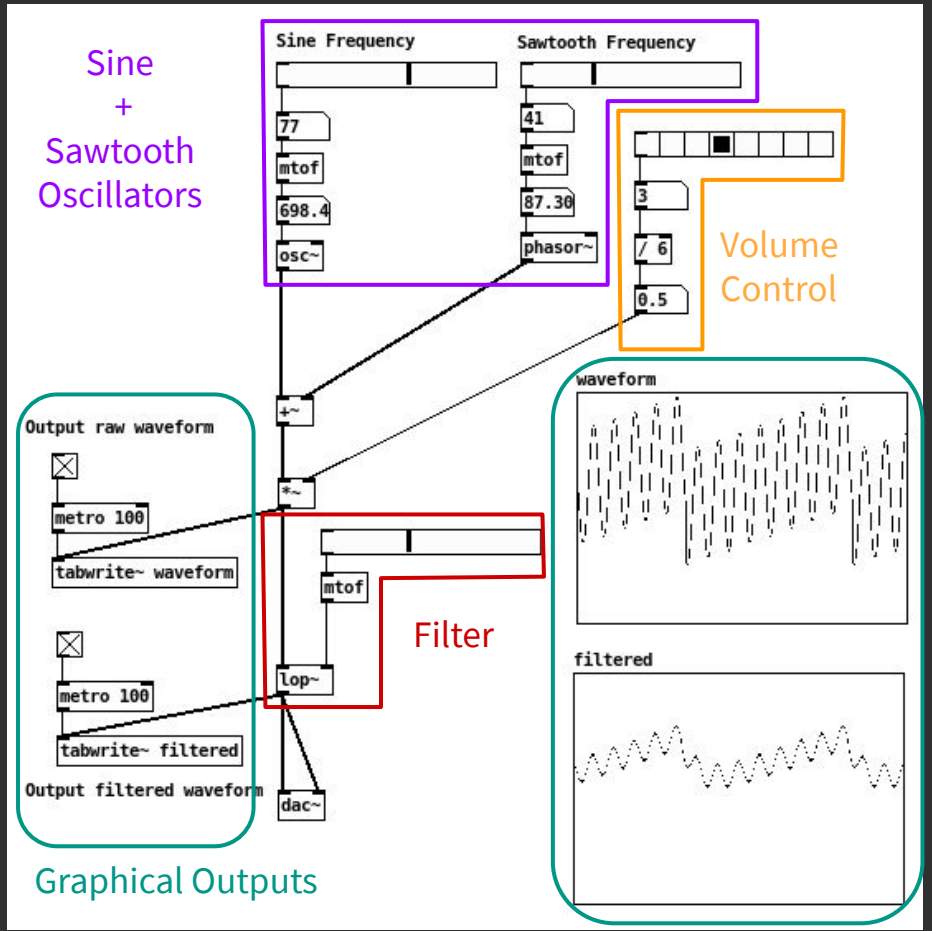
And a ***VCF*** (voltage controlled filter):

[vcf~] : resonant bp and lp that take audio signal to set center frequency

- Can change continuously in time !



Low Pass Filter



Amplifiers

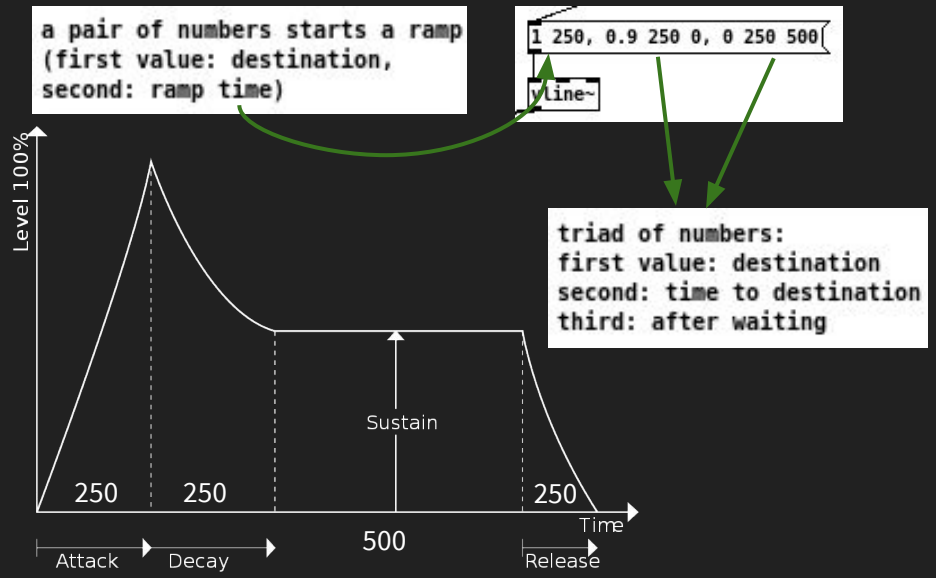
Controls the gain of the synth

You may want to have a look on:

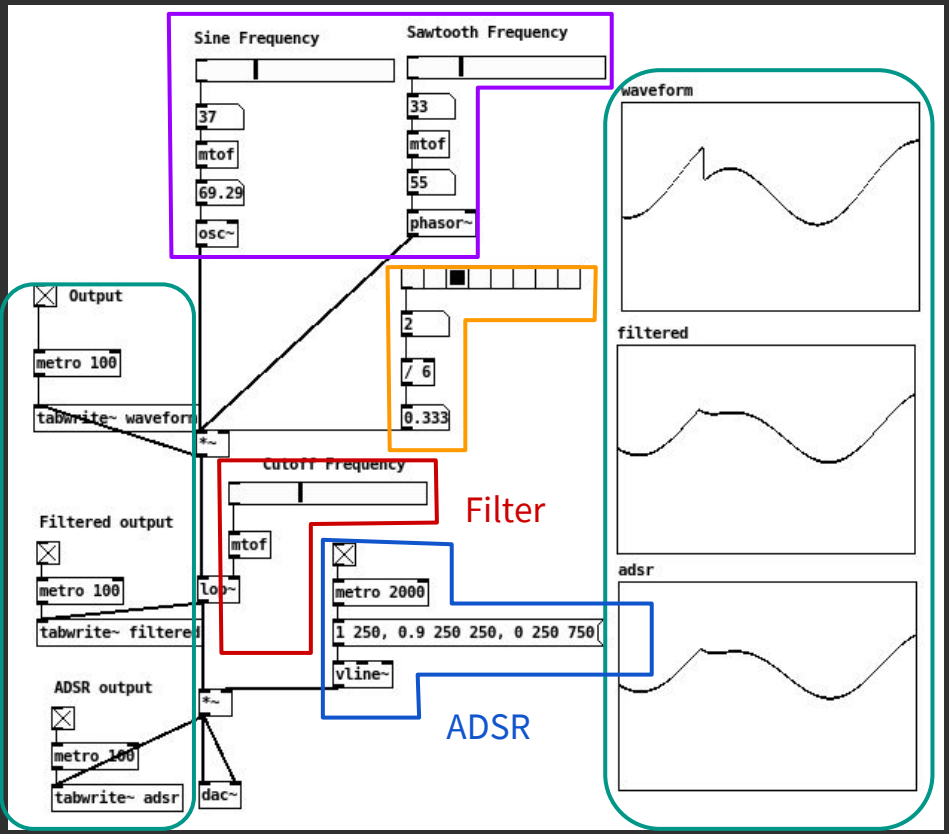
[line~] [tabread4~] [vline~]

Let’s go for a classic ADSR with **[vline~]**

➔ Generates an audio ramp



ADSR

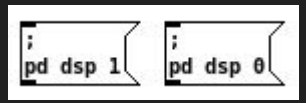


Keyboard Control

Object [key] return ASCII value
➔ can be treated as MIDI note

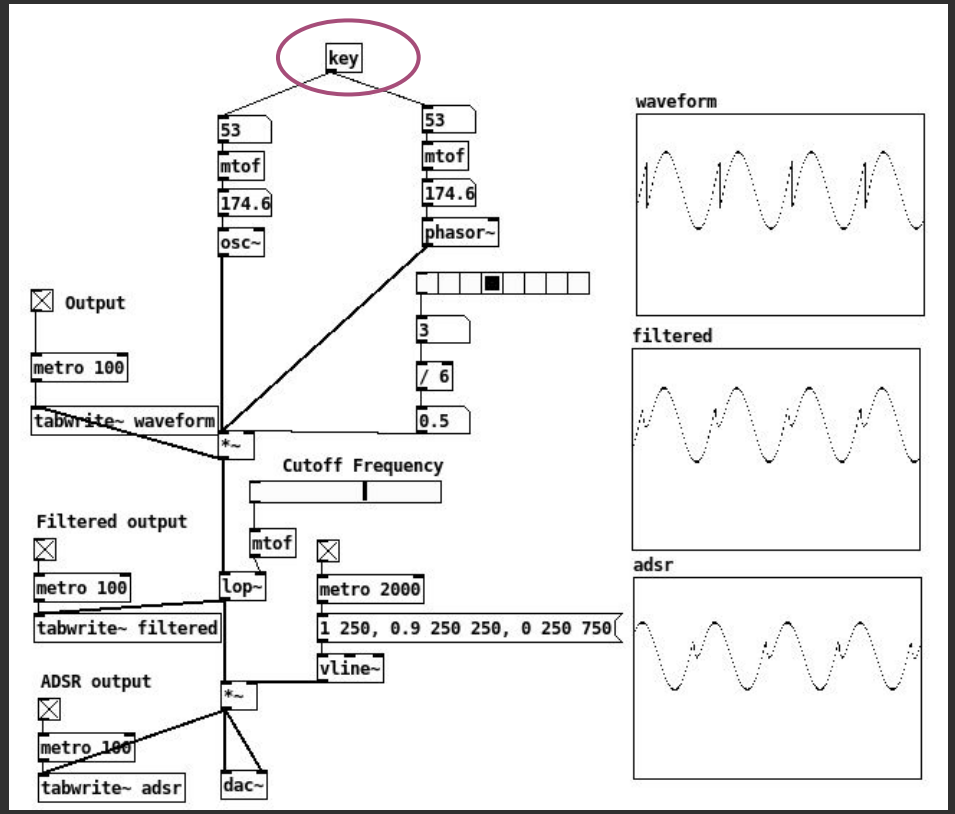
For those who want to use a MIDI keyboard look at [notein] object

DSP control on/off:



Message box, not an object

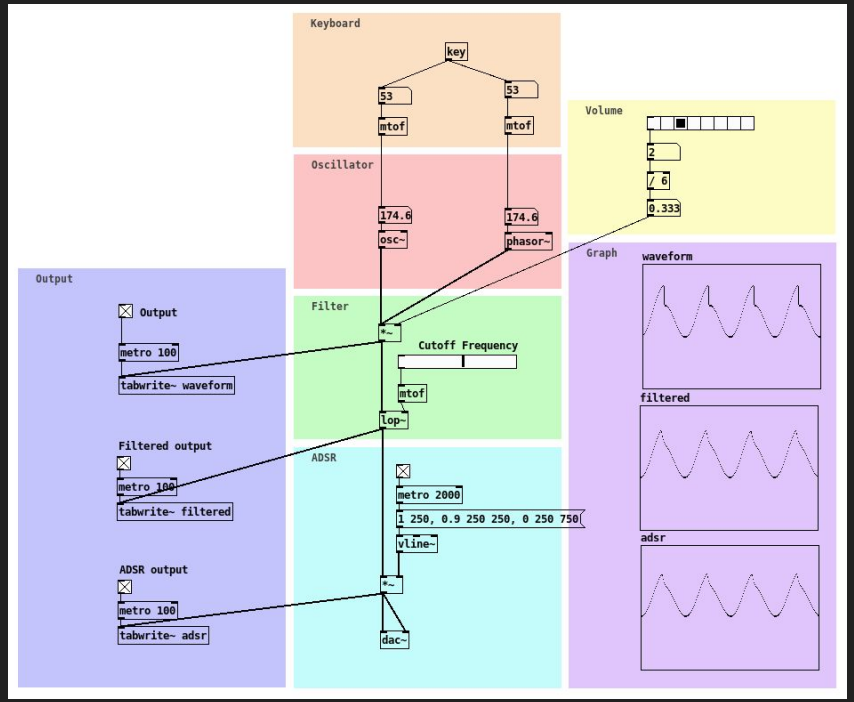
Keyboard plays notes



Subpatches

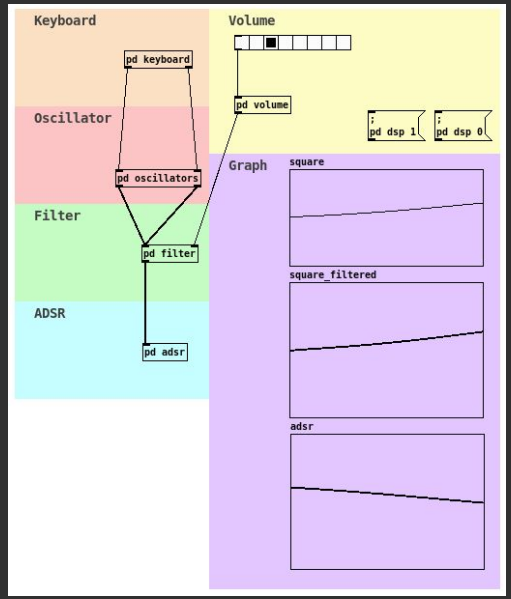
lightness with:

- colors: ⇧+Ctrl+C
- subpatches/abstractions
  [pd name_subpatch]



`pd name_subpatch` ➔ your subpatch appears!

- Add the piece of code you want
- Add as many [inlet] and [outlet] you need: this will add the inlet/outlet on the original [pd name_subpatch]
- Plug your subpatches together



N.B. be aware of [inlet] and [inlet~]

Project

Now you can already manage half of the project =)

Patch me some monophonic synths with the following features:

- at least one triangle oscillator
- at least one voltage controlled filter
- one capable of playing notes with keyboard so that the note lasts only the time the key is pressed
- with a delay of 1 second on your signal
- which plays melodies from random pitch and duration

Please remember: ¼ of the final grade is on the comprehensibility of your patches: usage of abstraction, comments, organization…

You will need to look by yourself the necessary objects, we have not seen all of them!