



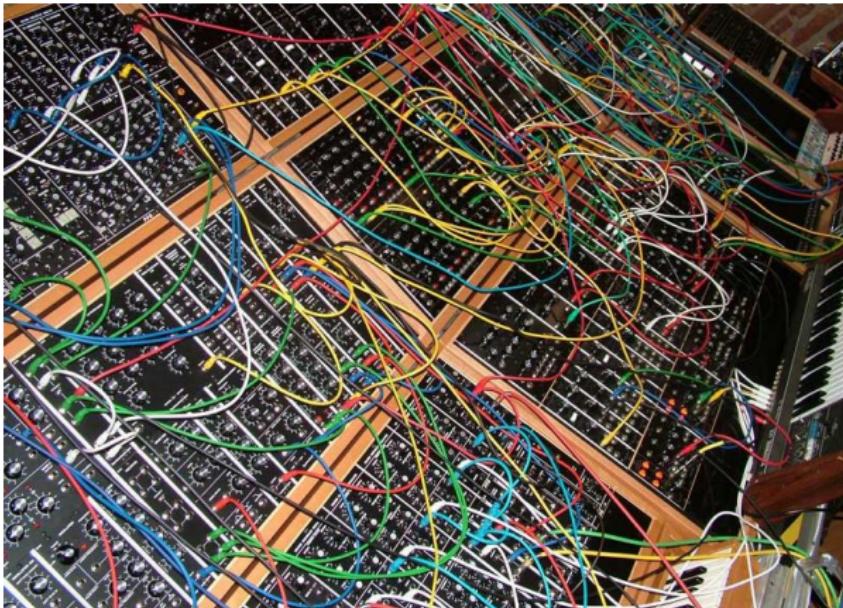
## FAUST Tutorial for Functional Programmers

Y. Orlarey, S. Letz, D. Fober, R. Michon

ICFP 2017 / FARM 2017

# What is Faust ?

# What is Faust?



A programming language (DSL) to build electronic music instruments

# Some Music Programming Languages

- 4CED
- Adagio
- AML
- AMPLE
- Antescofo
- Arctic
- Autoklang
- Bang
- Canon
- ChANT
- **Chuck**
- CLCE
- CMIX
- Cmusic
- CMUSIC
- Common Lisp Music
- Common Music
- Common Music Notation
- **Csound**
- CyberBand
- DARMS
- DCMP
- DMIX
- Elody
- EsAC
- Euterpea
- Extempore
- **Faust**
- Flavors Band
- Fluxus
- FOIL
- FORMES
- FORMULA
- Fugue
- Gibber
- GROOVE
- GUIDO
- HARP
- Haskore
- HMSL
- INV
- invokator
- KERN
- Keynote
- Kyma
- LOCO
- LPC
- Mars
- MASC
- **Max**
- MidiLisp
- MidiLogo
- MODE
- MOM
- Moxc
- MSX
- MUS10
- MUS8
- MUSCMP
- MuseData
- MusES
- MUSIC 10
- MUSIC 11
- MUSIC 360
- MUSIC 4B
- MUSIC 4BF
- MUSIC 4F
- MUSIC 6
- MCL
- **MUSIC III/IV/V**
- MusicLogo
- Music1000
- MUSIC7
- Musictex
- MUSIGOL
- MusicXML
- Musixtex
- NIFF
- NOTELIST
- Nyquist
- OPAL
- OpenMusic
- Organum1
- Outperform
- Overtone
- PE
- Patchwork
- PILE
- Pla
- PLACOMP
- PLAY1
- PLAY2
- PMX
- POCO
- POD6
- POD7
- PROD
- **Puredata**
- PWGL
- Ravel
- SALIERI
- SCORE
- ScoreFile
- SCRIPT
- SIREN
- SMDL
- SMOKE
- SSP
- SSSP
- ST
- **SuperCollider**
- Symbolic Composer
- Tidal

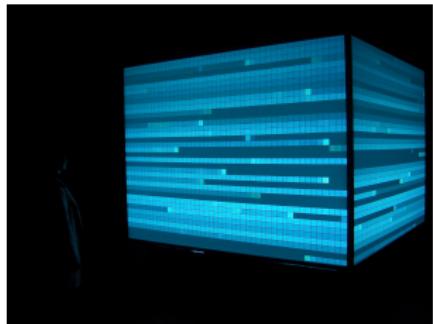
# Brief Overview to Faust

- Faust offers end-users a high-level alternative to C to develop audio applications for a large variety of platforms.
- The role of the Faust compiler is to synthesize the most efficient implementations for the target language (C, C++, LLVM, Javascript, etc.).
- Faust is used on stage for concerts and artistic productions, for education and research, for open sources projects and commercial applications :

# What Is Faust Used For ?

# Artistic Applications

Sonik Cube (Trafik/Orlarey), Smartfaust (Gracia), etc.



## Open-Source projects

Guitarix: Hermann Meyer



# WebAudio Applications

YC20 Emulator

Thanks to the HTML5/WebAudio API and Asm.js it is now possible to run synthesizers and audio effects from a simple web page !



# Sound Spatialization

Ambitools: Pierre Lecomte, CNAM

**Ambitools** (Faust Award 2016), 3-D sound spatialization using Ambisonic techniques.



# Medical Applications

Brain Stethoscope: Chris Chafe, CCRMA-Stanford

**Brain stethoscope** turns seizures into music in hopes of giving the listener an empathetic and intuitive understanding of the neural activity.



# Simulation Applications

Stanford Car Simulator: Romain Michon, CCRMA-Stanford

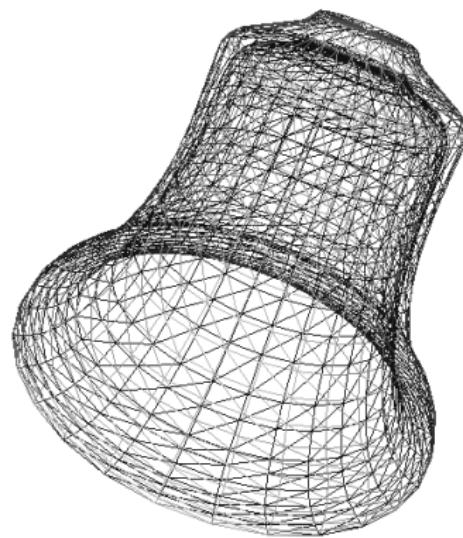
**Stanford Car Simulator**, simulation of the sound of a car engine in Faust.



# Simulation Applications

Bell simulations: Romain Michon and Chris Chafe, CCRMA-Stanford

CAD description of a bell turned into a procedural audio simulation in Faust



# Hearing Aids Applications

Soniccloud, USA

**Soniccloud:** every cell phone call can be perfectly calibrated for an individual's unique Hearing Fingerprint – across 10 sonic dimensions.

*The team at SonicCloud has had an outstanding experience working with Faust. Specifically, we have been able to optimize code to run our DSP algorithms in real-time without having to hand-optimize C/C++ code or write assembler.*  
*(Soniccloud)*



## THE PHONE CALL RE-INVENTED

SonicCloud's app brings clarity and crispness to every call...based on the way you hear.



Watch Video



## Musical Instruments

Moforte, USA

**Moforte** (USA) designs musical instruments for iPad and iOS using Faust



# Exercice 1: Djembe

# Exercise 1: Djembe

## Faust Online Compiler:

- <https://faust.grame.fr/onlinecompiler>

## Faust Libraries Documentation:

- <http://faust.grame.fr/libraries.html>

## Faust Code:

```
import("stdfaust.lib");

process = button("play")
          : pm.djembe(330,0.8,0.5,1);
```

# Faust Signals and Time Model

# Faust Signals and Time Model

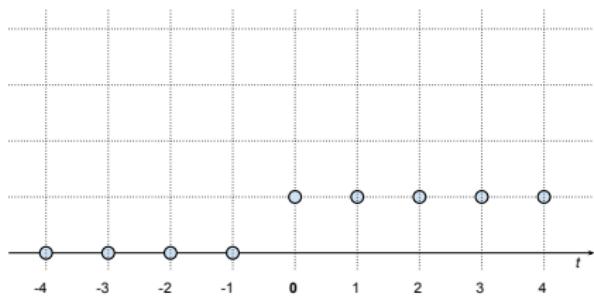
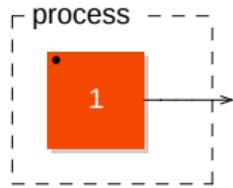
Faust programs operate on periodically sampled *signals*.

- A *signal*  $s \in \mathbb{S}$  is a *time to sample* function.
- Two kinds of signals:  $\mathbb{S} = \mathbb{S}_{\mathbb{Z}} \cup \mathbb{S}_{\mathbb{R}}$ 
  - ▶ Integer signals:  $\mathbb{S}_{\mathbb{Z}} = \mathbb{Z} \rightarrow \mathbb{Z}$
  - ▶ Floating-point signals:  $\mathbb{S}_{\mathbb{R}} = \mathbb{Z} \rightarrow \mathbb{R}$
- The value of a Faust signal is always 0 before time 0 :
  - ▶  $\forall s \in \mathbb{S}, s(t < 0) = 0$
- A Faust program denotes a *signal processor*  $p \in \mathbb{P}$ , a (continuous) function that maps a group of  $n$  input *signals* to a group of  $m$  output *signals* :
  - ▶  $\mathbb{P} = \mathbb{S}^n \rightarrow \mathbb{S}^m$

# Faust Signals and Time Model

Example, a "constant" signal

```
process = 1;
```



$$y(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}$$

# Primitive Signal Processors

# Faust Primitives

## Arithmetic operations

Syntax	Type	Description
+	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	addition: $y(t) = x_1(t) + x_2(t)$
-	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	subtraction: $y(t) = x_1(t) - x_2(t)$
*	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	multiplication: $y(t) = x_1(t) * x_2(t)$
$\wedge$	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	power: $y(t) = x_1(t)^{x_2(t)}$
/	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	division: $y(t) = x_1(t)/x_2(t)$
%	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	modulo: $y(t) = x_1(t)\%x_2(t)$
int	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cast into an int signal: $y(t) = (\text{int})x(t)$
float	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cast into an float signal: $y(t) = (\text{float})x(t)$

# Faust Primitives

## Bitwise operations

Syntax	Type	Description
&	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical AND: $y(t) = x_1(t) \& x_2(t)$
	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical OR: $y(t) = x_1(t)   x_2(t)$
xor	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical XOR: $y(t) = x_1(t) \wedge x_2(t)$
<<	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arith. shift left: $y(t) = x_1(t) << x_2(t)$
>>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arith. shift right: $y(t) = x_1(t) >> x_2(t)$

# Faust Primitives

## Comparison operations

Syntax	Type	Description
<	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	less than: $y(t) = x_1(t) < x_2(t)$
$\leq$	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	less or equal: $y(t) = x_1(t) \leq x_2(t)$
>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	greater than: $y(t) = x_1(t) > x_2(t)$
$\geq$	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	greater or equal: $y(t) = x_1(t) \geq x_2(t)$
$\equiv$	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	equal: $y(t) = x_1(t) \equiv x_2(t)$
$\neq$	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	different: $y(t) = x_1(t) \neq x_2(t)$

# Faust Primitives

## Trigonometric functions

Syntax	Type	Description
acos	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	arc cosine: $y(t) = \text{acosf}(x(t))$
asin	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	arc sine: $y(t) = \text{asinf}(x(t))$
atan	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	arc tangent: $y(t) = \text{atanf}(x(t))$
atan2	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arc tangent of 2 signals: $y(t) = \text{atan2f}(x_1(t), x_2(t))$
cos	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cosine: $y(t) = \text{cosf}(x(t))$
sin	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	sine: $y(t) = \text{sinf}(x(t))$
tan	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	tangent: $y(t) = \text{tanf}(x(t))$

# Faust Primitives

## Other Math operations

Syntax	Type	Description
exp	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	base-e exponential: $y(t) = \text{expf}(x(t))$
log	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	base-e logarithm: $y(t) = \text{logf}(x(t))$
log10	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	base-10 logarithm: $y(t) = \text{log10f}(x(t))$
pow	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	power: $y(t) = \text{powf}(x_1(t), x_2(t))$
sqrt	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	square root: $y(t) = \text{sqrtf}(x(t))$
abs	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	absolute value (int): $y(t) = \text{abs}(x(t))$ absolute value (float): $y(t) = \text{fabsf}(x(t))$
min	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	minimum: $y(t) = \text{min}(x_1(t), x_2(t))$
max	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	maximum: $y(t) = \text{max}(x_1(t), x_2(t))$
fmod	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	float modulo: $y(t) = \text{fmodf}(x_1(t), x_2(t))$
remainder	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	float remainder: $y(t) = \text{remainderf}(x_1(t), x_2(t))$
floor	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	largest int $\leq$ : $y(t) = \text{floorf}(x(t))$
ceil	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	smallest int $\geq$ : $y(t) = \text{ceilf}(x(t))$
rint	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	closest int: $y(t) = \text{rintf}(x(t))$

# Faust Primitives

## Delays and Tables

Syntax	Type	Description
mem	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	1-sample delay: $y(t+1) = x(t), y(0) = 0$
@	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	delay: $y(t+x_2(t)) = x_1(t), y(t < x_2(t)) = 0$
rdtable	$\mathbb{S}^3 \rightarrow \mathbb{S}^1$	read-only table: $y(t) = T[r(t)]$
rwtable	$\mathbb{S}^5 \rightarrow \mathbb{S}^1$	read-write table: $T[w(t)] = c(t); y(t) = T[r(t)]$
select2	$\mathbb{S}^3 \rightarrow \mathbb{S}^1$	select between 2 signals: $T[] = \{x_0(t), x_1(t)\}; y(t) = T[s(t)]$
select3	$\mathbb{S}^4 \rightarrow \mathbb{S}^1$	select between 3 signals: $T[] = \{x_0(t), x_1(t), x_2(t)\}; y(t) = T[s(t)]$

# Faust Primitives

## User Interface Primitives

Syntax	Example
button( <i>str</i> )	button("play")
checkbox( <i>str</i> )	checkbox("mute")
vslider( <i>str, cur, min, max, inc</i> )	vslider("vol",50,0,100,1)
hslider( <i>str, cur, min, max, inc</i> )	hslider("vol",0.5,0,1,0.01)
nentry( <i>str, cur, min, max, inc</i> )	nentry("freq",440,0,8000,1)
vgroup( <i>str, block-diagram</i> )	vgroup("reverb", ...)
hgroup( <i>str, block-diagram</i> )	hgroup("mixer", ...)
tgroup( <i>str, block-diagram</i> )	vgroup("parametric", ...)
vbargraph( <i>str, min, max</i> )	vbargraph("input",0,100)
hbargraph( <i>str, min, max</i> )	hbargraph("signal",0,1.0)

# Exercice 2: Adding rhythm and sliders to the Djembe

## Exercise 2: Adding rhythm to the Djembe

### Faust Code:

```
import("stdfaust.lib");

process = button("play"), // try checkbox("play")
           ba.pulsen(100, 4800) : *
: pm.djembe(330,0.8,0.5,1);
```

## Exercise 3: Adding sliders to the Djembe

### Faust Code:

```
import("stdfaust.lib");

process = checkbox("play"),
          ba.pulsen(100, 4800) : *
: pm.djembe(
    hslider("freq", 300, 100, 1000, 1),
    hslider("position", 0.8, 0, 1, 0.1),
    hslider("sharpness", 0.5, 0, 1, 0.1),
    hslider("gain", 0.5, 0, 1, 0.1)
);
```

## Exercise 4: Adding an echo to the Djembe

### Faust Code:

```
import("stdfaust.lib");

echo = +~(@(22100):*(hslider("fb",0,0,1,0.01)));

process = checkbox("play"),
          ba.pulsen(100, 4800) : *
: pm.djembe(
          hslider("freq", 300, 100, 1000, 1),
          hslider("position", 0.8, 0, 1, 0.1),
          hslider("sharpness", 0.5, 0, 1, 0.1),
          hslider("gain", 0.5, 0, 1, 0.1)
)
: echo;
```

# Programming by composition

# Programming by Composition

Block-Diagram Algebra

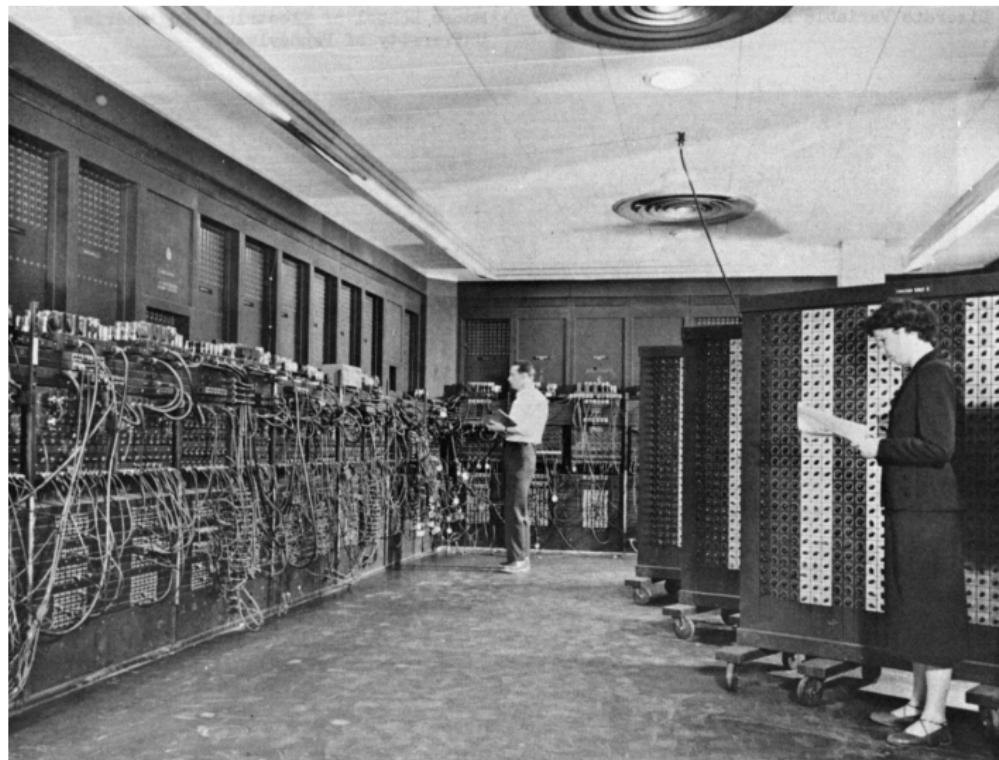
Programming by patching is familiar to musicians :



# Programming by Composition

## Block-Diagram Algebra

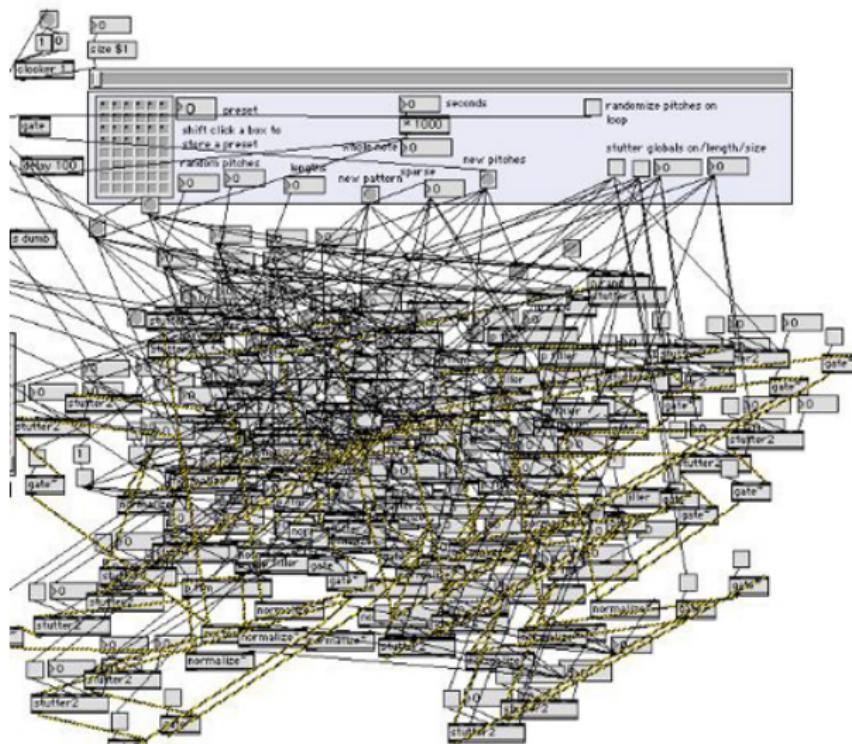
Programming by patching, the ENIAC computer :



# Programming by Composition

## Block-Diagram Algebra

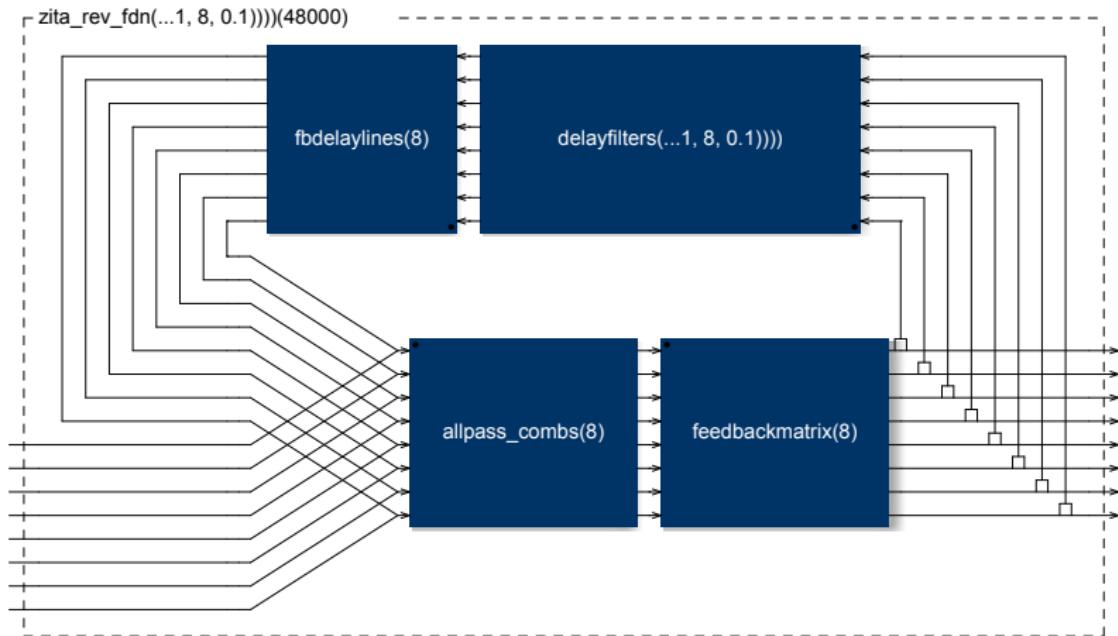
Block-diagrams are widely used in Visual Programming Languages like Max/MSP:



# Programming by Composition

## Block-Diagram Algebra

Faust allows structured block-diagrams, here part of the zita reverb.



# Programming by Composition

## Composition Operations

- $(A <: B)$  split composition (associative, priority 1)
- $(A :> B)$  merge composition (associative, priority 1)
- $(A : B)$  sequential composition (associative, priority 2)
- $(A, B)$  parallel composition (associative, priority 3)
- $(A \sim B)$  recursive composition (priority 4)

# Programming by Composition

Same Expression in Lambda-Calculus, FP and Faust

Lambda-Calculus

$\lambda x. \lambda y. (x+y, x*y) \ 2 \ 3$

FP/FL (John Backus)

$[+, *] : <2, 3>$

Faust

$2, 3 <: +, *$

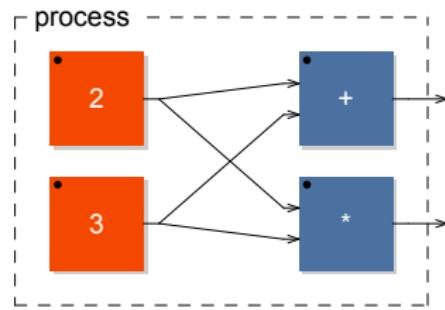


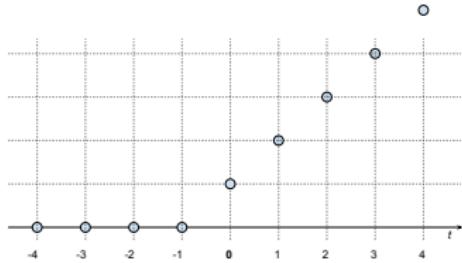
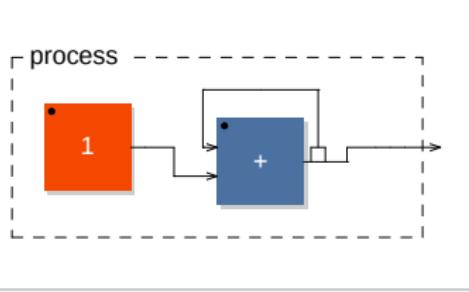
Figure: block-diagram of  
 $2, 3 <: +, *$

# Programming by Composition

## A Very Simple Example

```
process = 1 : +^_;
```

$$y(t) = \begin{cases} 0 & t < 0 \\ 1 + y(t - 1) & t \geq 0 \end{cases}$$



# Programming by Composition

## Parallel Composition (associative, priority 3)

The *parallel composition*  $(A, B)$  is probably the simplest one. It places the two block-diagrams one on top of the other, without connections.

$$(A, B) : (\mathbb{S}^n \rightarrow \mathbb{S}^m) \rightarrow (\mathbb{S}^{n'} \rightarrow \mathbb{S}^{m'}) \rightarrow (\mathbb{S}^{n+n'} \rightarrow \mathbb{S}^{m+m'})$$

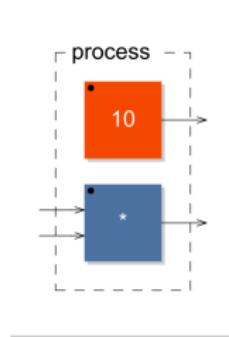


Figure: Example of parallel composition  $(10, *)$

# Programming by Composition

## Sequential Composition (associative, priority 2)

The *sequential composition* ( $A:B$ ) connects the outputs of  $A$  to the corresponding inputs of  $B$ .

$$(A:B) : (\mathbb{S}^n \rightarrow \mathbb{S}^m) \rightarrow (\mathbb{S}^m \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$$

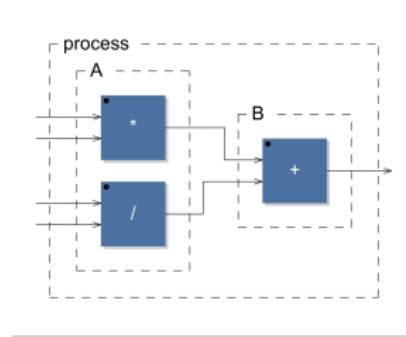


Figure: Example of sequential composition  $((*,/):+)$

# Programming by Composition

## Split Composition (associative, priority 1)

The *split composition* ( $A <: B$ ) operator is used to distribute the outputs of  $A$  to the inputs of  $B$

$$(A <: B) : (\mathbb{S}^n \rightarrow \mathbb{S}^m) \rightarrow (\mathbb{S}^{k \cdot m} \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$$

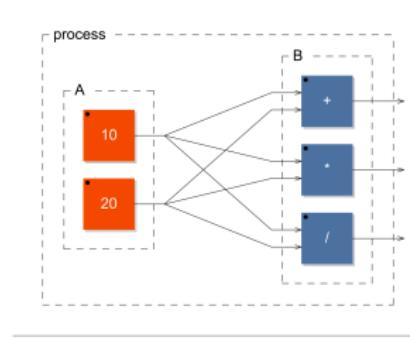


Figure: example of split composition  $((10, 20) <: (+, *, /))$

# Programming by Composition

## Merge Composition (associative, priority 1)

The *merge composition* ( $A :> B$ ) is used to connect several outputs of  $A$  to the same inputs of  $B$ . Signals connected to the same input are added.

$$(A :> B) : (\mathbb{S}^n \rightarrow \mathbb{S}^{k \cdot m}) \rightarrow (\mathbb{S}^m \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$$

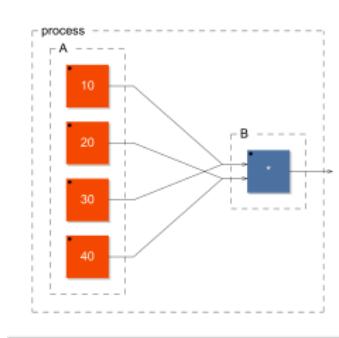


Figure: example of merge composition  $((10, 20, 30, 40) :> *)$

# Programming by Composition

## Recursive Composition (priority 4)

The *recursive composition* ( $A \sim B$ ) is used to create cycles in the block-diagram in order to express recursive computations.

$$(A \sim B) : (\mathbb{S}^{n+n'} \rightarrow \mathbb{S}^{m+m'}) \rightarrow (\mathbb{S}^{m'} \rightarrow \mathbb{S}^{n'}) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^{m+m'})$$

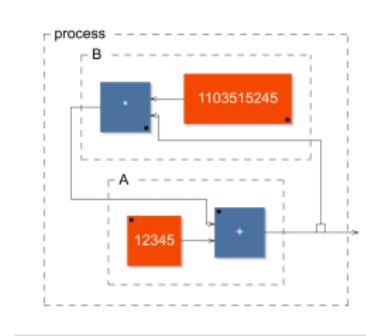
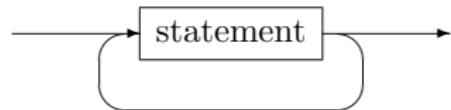


Figure: example of recursive composition  $+(12345) \sim *(1103515245)$

# Faust Program

# Faust Program

*program*



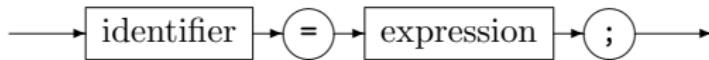
- A Faust program is essentially a list of *statements*. These statements can be :
  - ▶ metadata *declarations*,
  - ▶ file *imports*
  - ▶ *definitions*
- Example :

```
declare name      "noise";
declare copyright "(c)GRAME\u20222006";
import("music.lib");
process = noise * vslider("volume", 0, 0, 1, 0.1);
```

# Definitions

## Simple Definitions

*definition*



- A *definition* associates an identifier with an expression it stands for.

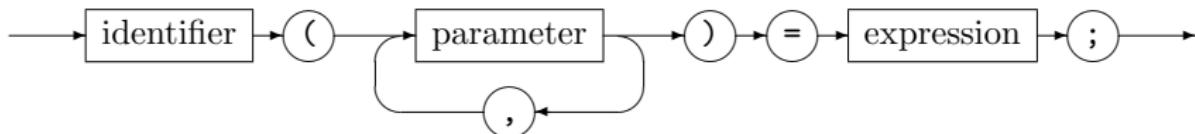
Example :

```
random = +(12345) ~ *(1103515245);
```

# Definitions

## Functions' definitions

*definition*



- Definitions with formal parameters correspond to functions' definitions.

Example :

```
linear2db(x) = 20*log10(x);
```

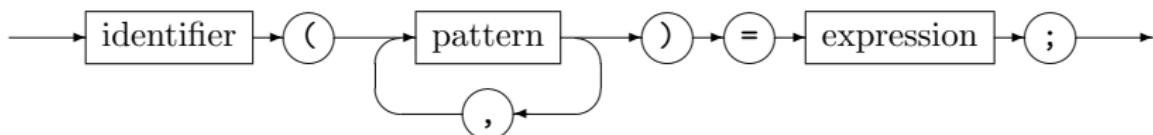
- Alternative notation using a *lambda-abstraction*:

```
linear2db = \x.(20*log10(x));
```

# Definitions

## Pattern Matching Definitions

*definition*



- Formal parameters can also be full expressions representing patterns.

Example :

```
duplicate(1,exp) = exp;
duplicate(n,exp) = exp, duplicate(n-1,exp);
```

- Alternative notation :

```
duplicate = case {
    (1,exp) => exp;
    (n,exp) => duplicate(n-1,exp);
};
```

# Statement

## Import file

*fileimport*

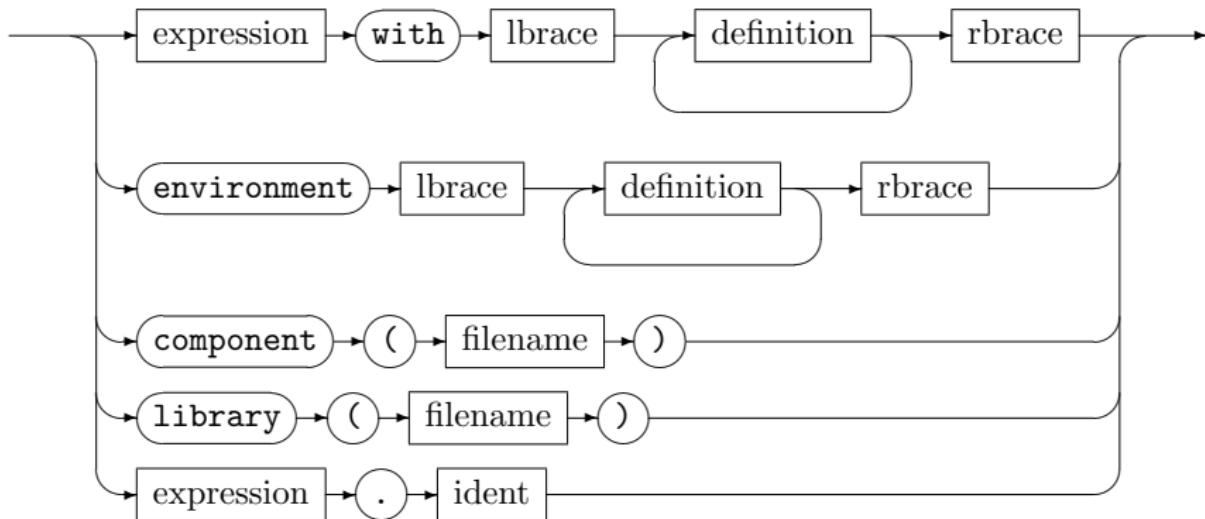


- allows to import definitions from other source files.
- for example `import("math.lib");` imports the definitions from "math.lib" file, a set of additional mathematical functions provided as foreign functions.

# Expressions

## Environments

*envexp*

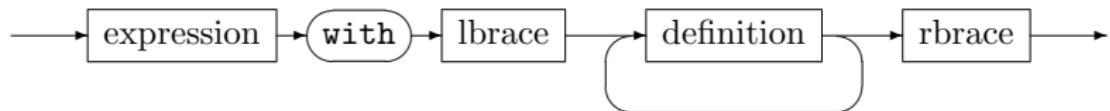


- Each Faust expression has an associated *lexical environment*

# Environments

## With Expression

*withexpression*



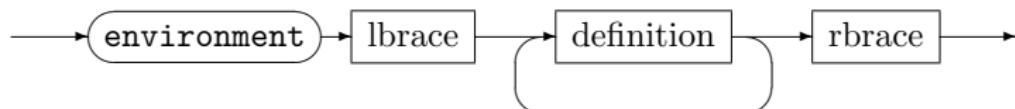
- With expression allows to specify a *local environment*, a private list of definitions that will be used to evaluate the left hand expression
- example pink noise filter :

```
pink = f : + ~ g with {  
    f(x) = 0.04957526213389*x  
        - 0.06305581334498*x@1  
        + 0.01483220320740*x@2;  
    g(x) = 1.80116083982126*x  
        - 0.80257737639225*x@1;  
};
```

# Environments

## Environment

*environment*



- an **environment** is used to group together related definitions :

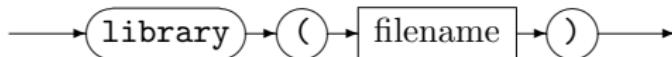
```
constant = environment {
    pi = 3.14159;
    e = 2,718 ;
    ....
};
```

- definitions of an environment can be easily accessed : `constant.pi`

# Environments

## Library

*library*

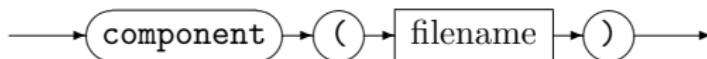


- allows to create an environment by reading the definitions from a file.
- example : `library("filter.lib")`
- definitions are accessed like this : `library("filter.lib").smooth`

# Environments

## Component

*component*



- allows to reuse a full Faust program as a simple expression.
- example :

```
component("osc.dsp") <: component("freeverb.dsp")
```

- equivalence between :

```
component("freeverb.dsp")
```

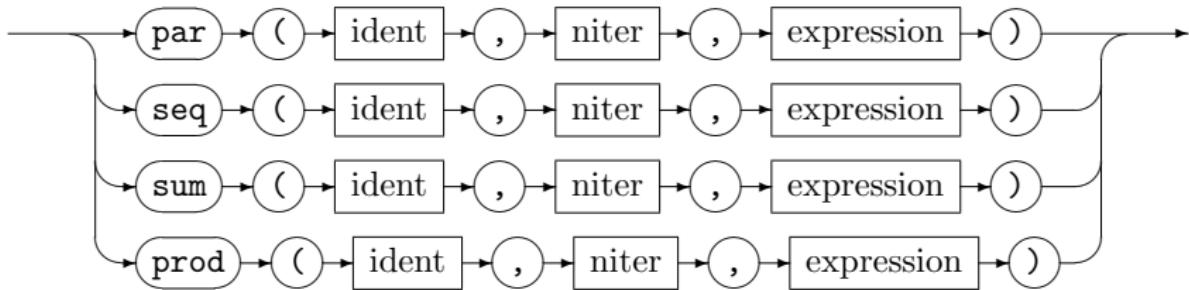
and

```
library("freeverb.dsp").process
```

# Expressions

## Iterations

*diagiteration*



- Iterations are analog to `for(...)` loops
- provide a convenient way to automate some complex block-diagram constructions.

# Expressions

## Iterations

The following example shows the use of `seq` to create a 10-bands filter:

```
process = seq(i, 10,
              vgroup("band_{%i",
                     bandfilter( 1000*(1+i) )
              )
            );
```

# Exercise 5: Djembe automation

# Exercise 5: Djembe automation

## Faust Code:

```
import("stdfaust.lib");

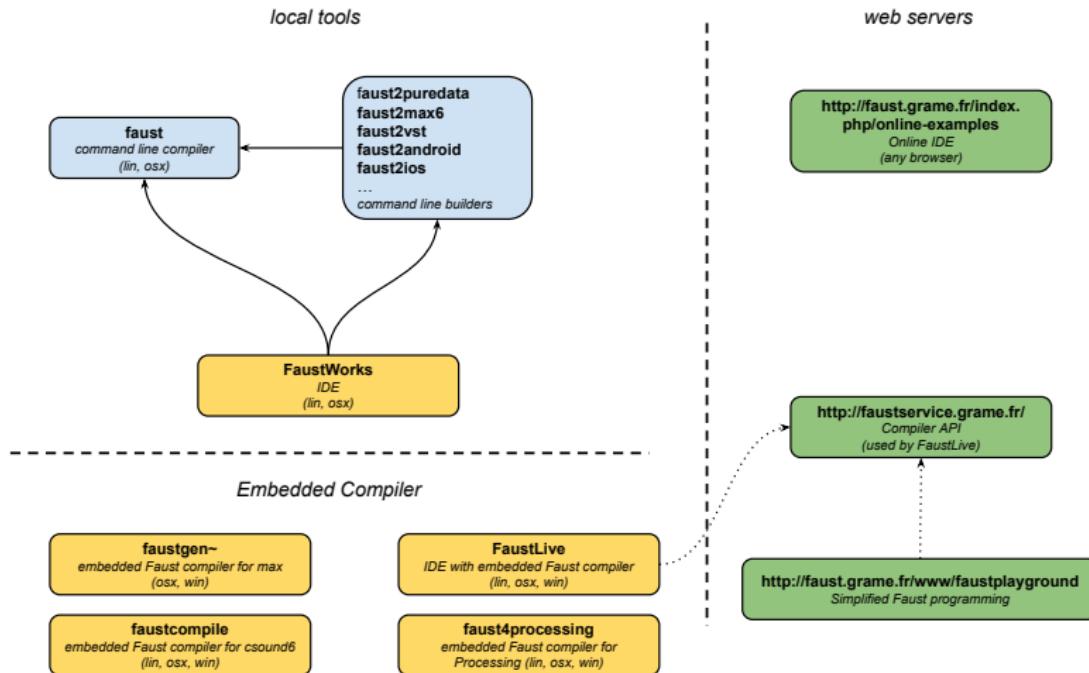
saw(f) = f/ma.SR : (+,1:fmod)^_ ;

process = checkbox("play"),
          ba.pulsen(100, 4800) : *
: pm.djembe(
    hslider("freq", 300, 100, 1000, 1),
    saw(hslider("fpos", 1, 0.05, 20, 0.01)),
    saw(hslider("fsharp", 1, 0.05, 20, 0.01)),
    saw(hslider("fgain", 1, 0.05, 20, 0.01))
)
;
```

# Faust Ecosystem

# Faust Ecosystem

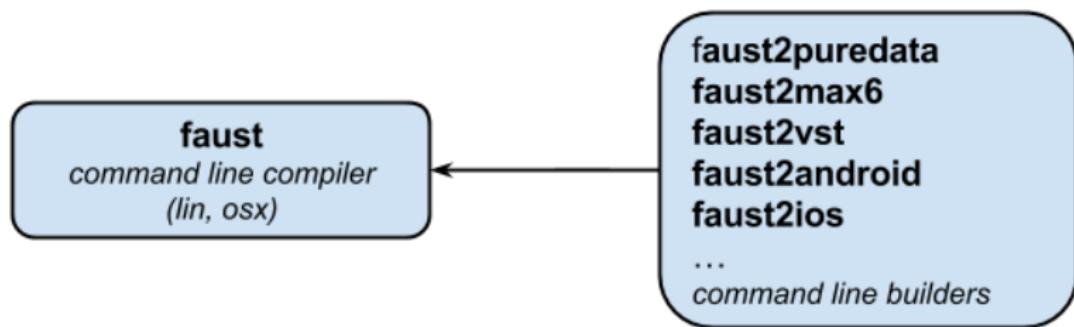
## Overview



# Faust Ecosystem

## Command-line Tools

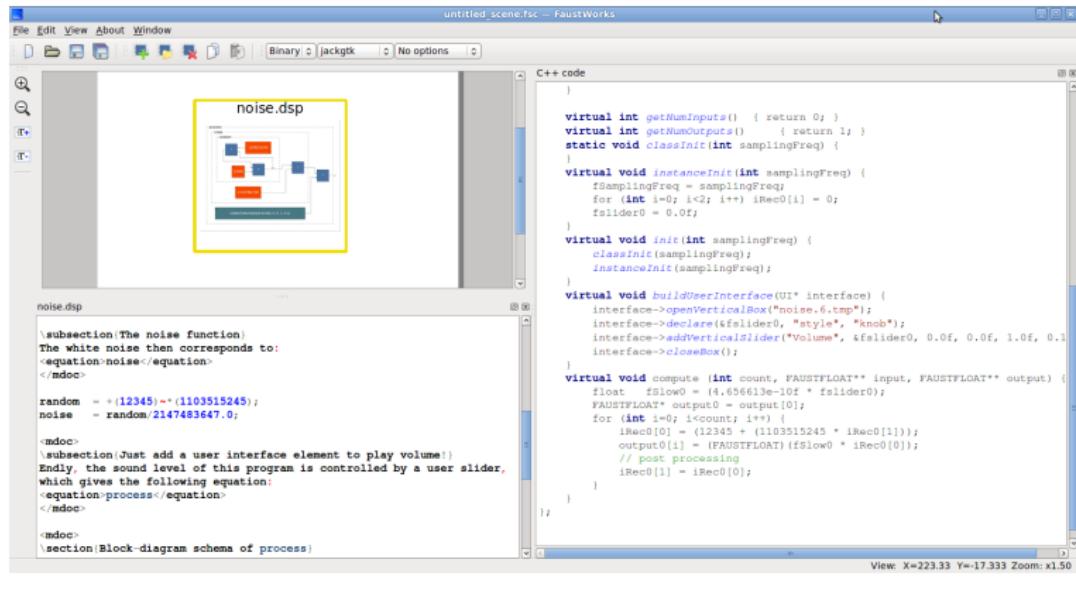
Simplify the compilation workflow : full automated process to build Android and iOS applications, VST plugins, Max/MSP externals, Csound opcodes, etc.



# Faust Ecosystem

## FaustWorks

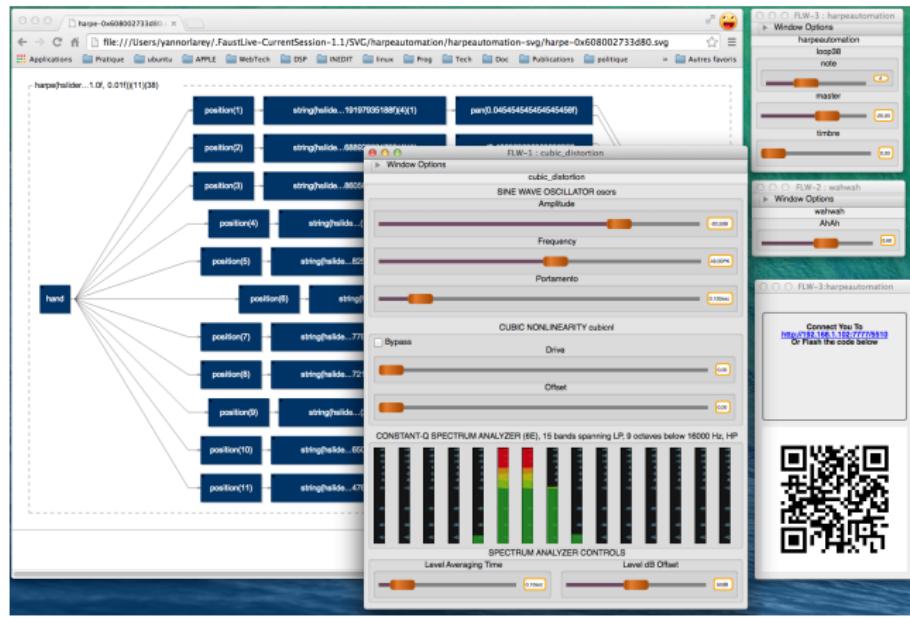
FaustWorks can simplify Faust learning in particular by providing "realtime" code and diagram generation:



# Faust Ecosystem

## FaustLive

FaustLive speeds up the Edit/Compile/Run. It provides advanced cooperation features :

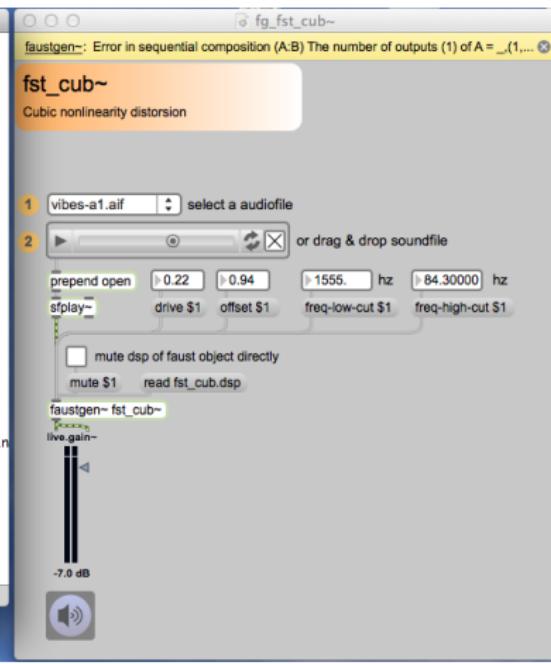


# Faust Ecosystem

## Faustgen

Faustgen speeds up the Edit/Compile/Run within the Max framework:

```
DSP code : fst_cub~  
declare name      "fst_cub~";  
declare version   "1.33";  
//declare author   "Julius O. Smith ";  
declare license   "STK-4.3";  
  
import("math.lib");  
import("music.lib");  
import("effect.lib");  
import("filter.lib");  
  
----- cubicnl(drive,offset) -----  
// Cubic nonlinearity distortion  
//  
// USAGE: cubicnl(drive,offset), where  
//   drive = distortion amount, between 0 and 1  
//   offset = constant added before nonlinearity to give even harmonics  
//           Note: offset can introduce a nonzero mean - feed  
//                 cubicnl output to dcblocker to remove this.  
//  
// REFERENCES:  
//   https://ccrma.stanford.edu/~jos/pasp/Cubic_Soft_Clipper.html  
//   https://ccrma.stanford.edu/~jos/pasp/Nonlinear_Distortion.html  
  
// direct from effect Lib and combine with a filter. A dc remover will be in  
  
sf1 = vslider("freq-low-cut",130,20,1000,1):smooth(0.99);  
sf2 = vslider("freq-high-cut",5000,1000,10000,1):smooth(0.99);  
  
drive = vslider("drive", 0, 0, 1, 0.01);  
offset = vslider("offset", 0, 0, 1, 0.01);  
  
process = cubicnl(drive,offset) * snakerhn(sf1,sf2) * mvecho-  
Insertion Point Line: 1
```



# Faust Ecosystem

## Faust4processing

Faust4processing provides an embedded Faust compiler for Processing:

The image shows two windows side-by-side. On the left is a 'Java Examples' browser window with a tree view of available examples. On the right is the Processing IDE window displaying a sketch named 'osc'. The sketch code is as follows:

```
void setup()
{
    // prepare the user interface
    size(480, 120);
    cp5 = new ControlP5(this);
    System.out.println("Working Directory = " + System.getProperty("user.dir"));

    // Define and compile osc signal processor
    osc = new FaustProcessing(this, "osc",
        "-----\n"
        "----- Sinusoidal Oscillator\n"
        "-----\n"
        "\n"
        "import(\"music.lib\");\n"
        "\n"
        "db2linear1(x) = pow(10.0, x/20.0);\n"
        "\n"
        "smooth(c) = *(1-c) : +~*(c);\n"
        "vol = hslider(\"volume [unit:dB]\", -20, -96, 0, 0.1) : db2linear :\n"
        "freq = hslider(\"freq [unit:Hz]\", 1000, 20, 24000, 1);\n"
        "\n"
        "process = vgroup(\"Oscillator\", osc(freq) * vol);\n"
    );
}

// build osc's user interface
int nbr=osc.getParamsCount();
size(600, 50 + nbr*25 + 45);

cp5 = new ControlP5(this);

for (int i=0; i<nbr; i++) {
    cp5.addSlider(osc.get
        .setPosition(50, i*25+50)
        .setSize(300, 20)
        .setRange(osc.getParamMin(i), osc.getParamMax(i))
        .setValue(osc.getParamValue(i))
}
```

The Processing IDE window shows the sketch running. A slider labeled 'volume' is set to -20.00, and a slider labeled 'freq' is set to 1000.00. The output window shows the text 'OSC'.

# Faust Ecosystem

## PMIX (Oliver Larkin)

PMIX speeds up the Edit/Compile/Run within VST host:

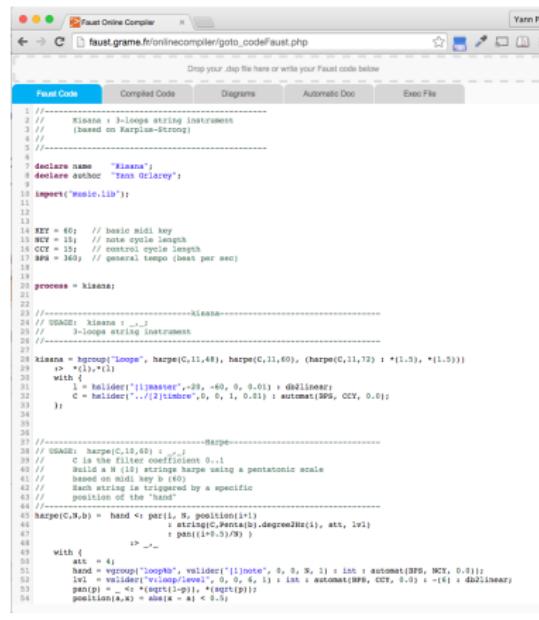
The screenshot shows the pMix application window. On the left, a patch diagram illustrates a signal flow from 'Midi Input' and 'Audio Input' through a 'Kisana' module (with controls for level, note, master, timbre) and a 'WhaWah' module to an 'Audio Output'. On the right, the 'Code Editor' tab is active, displaying Faust code for a 'Kisana' instrument. The code includes declarations for name ('Kisana'), author ('Yann Orlarey'), imports ('music.lib'), and various parameters like KEY, NCY, CCY, and RPS. Below the code editor is a 'Process' graph visualization showing complex data flow and processing blocks.

```
1 //-
2 // Kisana : 3-loops string instrument
3 // (based on Karpplus-Strong)
4 //
5 //-
6
7 declare name      "Kisana";
8 declare author    "Yann Orlarey";
9
10 import("music.lib");
11
12
13
14 KEY = 60; // basic midi key
15 NCY = 15; // note cycle length
16 CCY = 15; // control cycle length
17 RPS = 360. // general tempo (beats per sec)
```

# Faust Ecosystem

## Online compiler

The Online compiler can be used from a web browser to compile Faust programs for a variety of systems, including the Web.  
<http://faust.grame.fr/onlinecompiler>



A screenshot of a web-based online compiler for Faust code. The interface includes a toolbar at the top with icons for file operations, a search bar, and a tab labeled 'Yann Pro'. Below the toolbar is a header with tabs: 'Faust Code' (which is selected), 'Compiled Code', 'Diagrams', 'Automatic Doc', and 'Exec File'. The main area contains the Faust code for 'Kiasma', which is a 3-loops string instrument. The code includes declarations for name, author, imports, parameters (NCF, CCF, HPP, process), and a class definition for Kiasma. It also includes sections for 'kiasma' and 'Harpe', each with its own class definitions and methods. The code uses various Faust constructs like 'hgroup', 'harpe', 'automat', and 'db2linear'.

```
1 //> Kiasma : 3-loops string instrument
2 //> (based on Karpin-strong)
3 //
4 //
5 //
6
7 declare name "Kiasma";
8 declare author "Yann Orlarey";
9
10 import("music.lib");
11
12
13 NCF = 40; // basic midi key
14 NCF = 15; // note cycle length
15 CCF = 15; // control cycle length
16 HPP = 360; // general tempo (beat per sec)
17
18 process = kiasma;
19
20
21
22
23 //-----kiasma-----
24 // USAGE: kiasma 1 .. 2
25 // 3-loops string instrument
26
27
28 kiasma = hgroup("Loops", harpe(C,11,48), harpe(G,11,40), (harpe(G,11,72) : *(1.5), *(1.5)));
29 v0 = *(1.5)*{1}
30 with {
31   1 = halider("1@master", -29, -60, 0, 0.01) : db2linear;
32   2 = halider("../2@timbre", 0, 0, 1, 0.81) : automat(HPP, CCF, 0.01);
33 }
34
35
36
37 //-----Harpe-----
38 // USAGE: harpe(C,11,48)
39 // C is the filter coefficient 0..1
40 // Build a N (12) strings harpe using a pentatonic scale
41 // Each string is triggered by a specific
42 // position of the "hand"
43
44
45 harpe(C,N,b) = hand <-> partl(N, position(i=1)
46   : stringq(Penta(b).degree2Hzz(i), att, lvl)
47   : pm((i=1)*5)/9);
48
49 with {
50   i = 4;
51   hand = vgroup("loopA", validator("linote", 0, 0, N, 1) : int : automat(HPP, NCF, 0.0));
52   lvl = validator("volump", level, 0, 0, 6, 1) : int : automat(HPP, CCF, 0.0) : -(6) : db2linear;
53   pan(p) = -c * (sgcrit1-p0); *sgcrit(p);
54   position(n,i) = min(n-i) < 0;
55 }
```

# Faust Ecosystem

## Faust playground

The Web as a gigantic Lego box to reuse and recompose audio applications. <http://faust.gramme.fr/faustplayground>

The screenshot shows the FaustPlayGround interface. At the top, there's a navigation bar with tabs: LIBRARY, LOAD, EDIT, SAVE, EXPORT, and HELP. Below the navigation bar, a message says "The application is called: Patch". A note below it states: "Only alphabet letters and numbers are accepted. Spaces, apostrophes and accents are automatically replaced. The name cannot start with a number. It must be between 1 and 50 characters." To the right, there's a "Choose export" section with a text input containing "http://faustservice.gramme.fr", a "Refresh server" button, and dropdown menus set to "android" for both "Target" and "Platform". There are also "ECHO" and "BRIGHTNESS" sliders. A "Download" button is next to a QR code. In the center, there's a "Patch" area with a title "KOSANA". This area contains a microphone icon, three green knobs labeled "1 (NOTE 48)", "2 (NOTE 80)", and "3 (NOTE 72)" with values "5.0 PK", "5.0 PK", and "0.0 PK" respectively, a "BRIGHTNESS" slider at 1.00, a "RESONANCE" slider at 4.00, and a "FEEDBACK" slider with a red dot. A pen icon is also present. On the right side of the patch, there's a speaker icon with a green dot. The bottom of the screen has a footer with the text "Extension of the WebAudio Playground by Chris Wilson" and a small GRAME logo. The bottom right corner features standard browser navigation icons.

Thanks! Questions?