DSP MADE ACCESSIBLE: FAST PLUG-INS DEVELOPMENT WITH PHAUSTO AND Cmajor

Domenico Cipriani - 2024

pharo.org

INTRO

- Languages such as **C++** , **FAUST , Cmajor** are extremely powerful l but they can also be intimidating for beginners or overly complex for prototyping simpler instruments and effects.

- Using **Phausto** together with **Cmajor** can bridges this gap, combining technical power with creative simplicity, and helping sound artists to bring their audio ideas to life faster.

pharo.org

WHAT IS PHAUSTO?

- **Phausto** is a multi-platform library and API that enables the programming Digital Signal Processors (DSPs) and sound generation in **Pharo**

- The audio is generated through FFI calls to a *dynamic engine* that computes audio signal by leveraging the power on an embedded **FAUST** compiler.

- Phausto has been developed with three main goals:
    1. To allow sound artists and musician to program synthesisers and effects and compose music with Pharo;
    2. To teach DSP programming to beginners and offer a fast prototyping platform for musician and audio developers, thanks to its Cmajor and C++ exporters
    3. To enrich Pharo applications with sound;

Domenico Cipriani - 2024

pharo.org

Becoming a Programmer

- I began programming in 2017 (thanks to Cristian Vogel and **Symbolic Sound *Kyma***).
- I have been developing *Coypu* in **Pharo** and programming music-on-the fly since 2020.

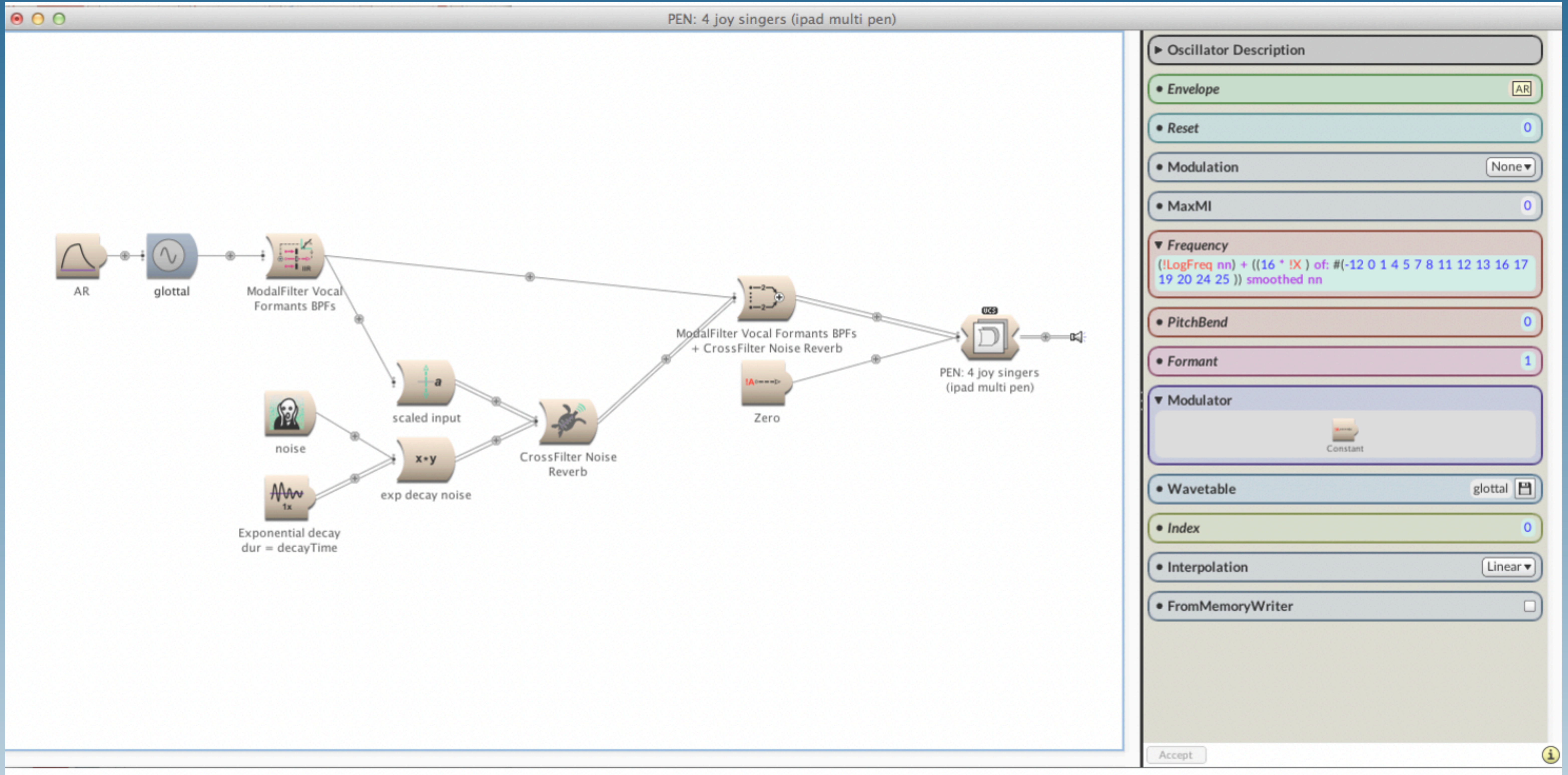
- Audio plug-ins developer for soft computing since 2020, thanks to **JUCE**
- Sponsored by the **Pharo Association** since April 2024.

Domenico Cipriani - 2024

pharo.org

SYMBOLIC SOUND KYMA

- Music programming language and IDE written in Smalltalk created by Carla Scaletti and Kurt J. Hebel at Urbana Champaign, Illinois.


PEN: 4 joy singers (ipad multi pen)
AR
glottal
ModalFilter Vocal Formants BPFs
scaled input
noise
x•y
exp decay noise
Exponential decay dur = decayTime
CrossFilter Noise Reverb
ModalFilter Vocal Formants BPFs + CrossFilter Noise Reverb
Zero
PEN: 4 joy singers (ipad multi pen)
Accept

▶ Oscillator Description
- *Envelope* AR
- *Reset* 0
- Modulation None
- MaxMI 0

▼ *Frequency*

```
(!LogFreq nn) + ((16 * !X ) of: #(-12 0 1 4 5 7 8 11 12 13 16 17
19 20 24 25 )) smoothed nn
```

- *PitchBend* 0
- *Formant* 1

▼ Modulator
Constant

- Wavetable glottal
- *Index* 0
- Interpolation Linear
- FromMemoryWriter ☐

- The Smalltalk code is compiled on an external DSPs called Capybara, Paca(rana), Pacamara (Ristretto)

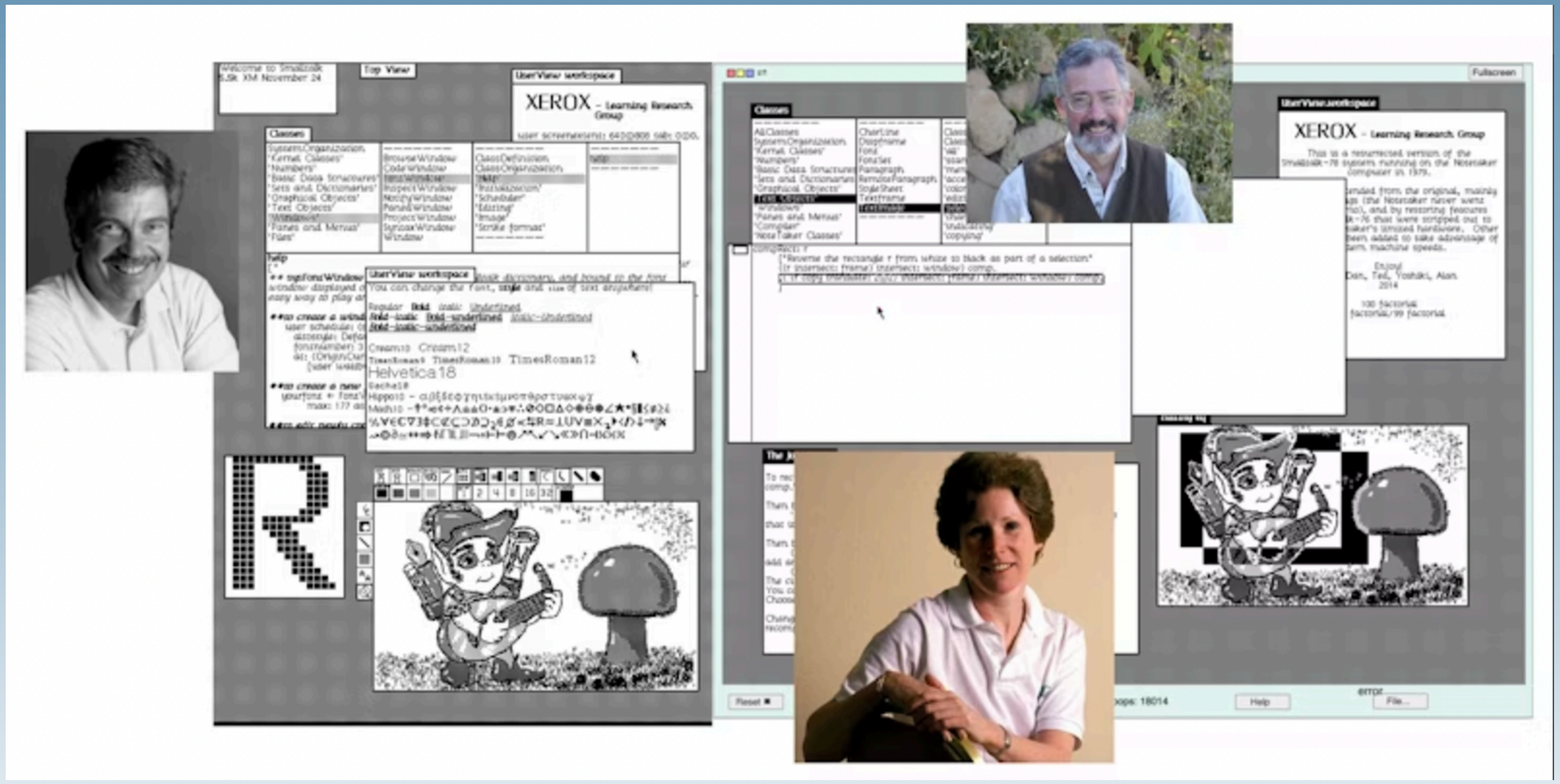
- “The Holy Grail of sound design”

Domenico Cipriani - 2024

pharo.org

WHAT IS SMALLTALK?

- Smalltalk was created at Xerox Parc in 1972 by by the Learning Research Group (LRG) scientists, including Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Diana Merry.
- Smalltalk was designed as a purely object-oriented language for teaching programming to children, emerging from Alan Kay's vision of the "Dynabook" - a personal computer intended for young learners.


XEROX - Learning Research Group
XEROX - Learning Research Group

Smalltalk-80's release marked its commercial er
revolutionary concepts:

- First practical graphical user interface (GUI)
- Model-View-Controller (MVC) pattern
- Integrated development environment (IDE)
- Live programming environment

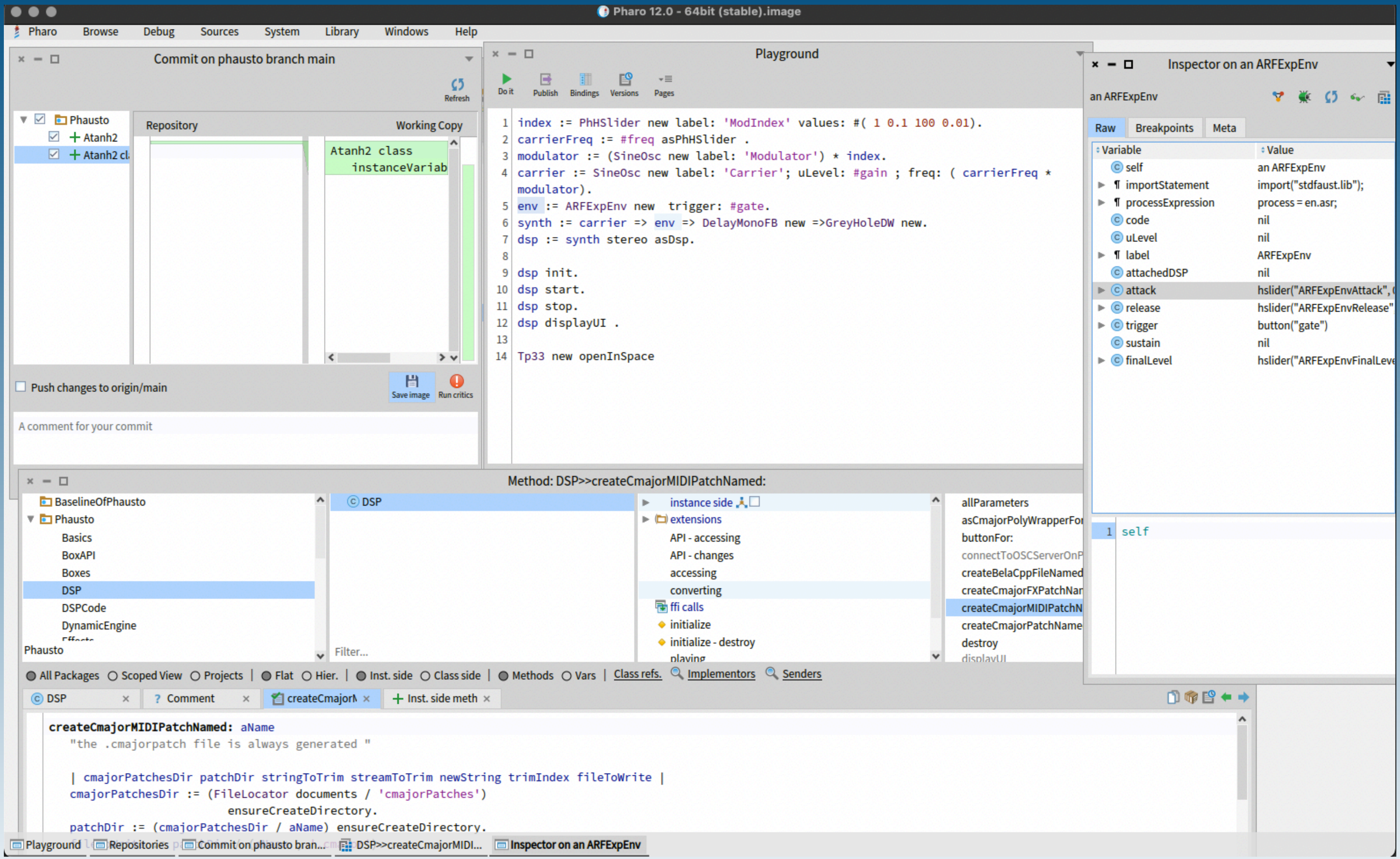
Domenico Cipriani - 2024

pharo.org

WHAT IS PHARO?

- **Pharo** is a pure object-oriented, dynamically typed, and reflective language; its syntax fits in a postcard and it comes with a platform-independent IDE.
- Created by Stéphane Ducasse and Marcus Denker at Inria in Lille, it originated as a fork of **Squeak**, the free and open-source implementation of **Smalltalk**.
- **Pharo** is developed by an international community of open-source developers, coordinated and maintained by the *Pharo consortium*.
- It comes with a non-viral MIT license!
- **Pharo** comes with Integrated *Git* support and with with an integrated framework for *SUnit Tests*

pharo.org



```
index := PhHSlider new label: 'ModIndex' values: #( 1 0.1 100 0.01).
carrierFreq := #freq asPhHSlider .
modulator := (SineOsc new label: 'Modulator') * index.
carrier := SineOsc new label: 'Carrier'; uLevel: #gain ; freq: ( carrierFreq *
modulator).
env := ARFExpEnv new  trigger: #gate.
synth := carrier => env => DelayMonoFB new =>GreyHoleDW new.
dsp := synth stereo asDsp.

dsp init.
dsp start.
dsp stop.
dsp displayUI .

Tp33 new openInSpace
```

Variable	Value
self	an ARFExpEnv
importStatement	import("stdfaust.lib");
processExpression	process = en.asr;
code	nil
uLevel	nil
label	ARFExpEnv
attachedDSP	nil
attack	hslider("ARFExpEnvAttack",
release	hslider("ARFExpEnvRelease"
trigger	button("gate")
sustain	nil
finalLevel	hslider("ARFExpEnvFinalLeve

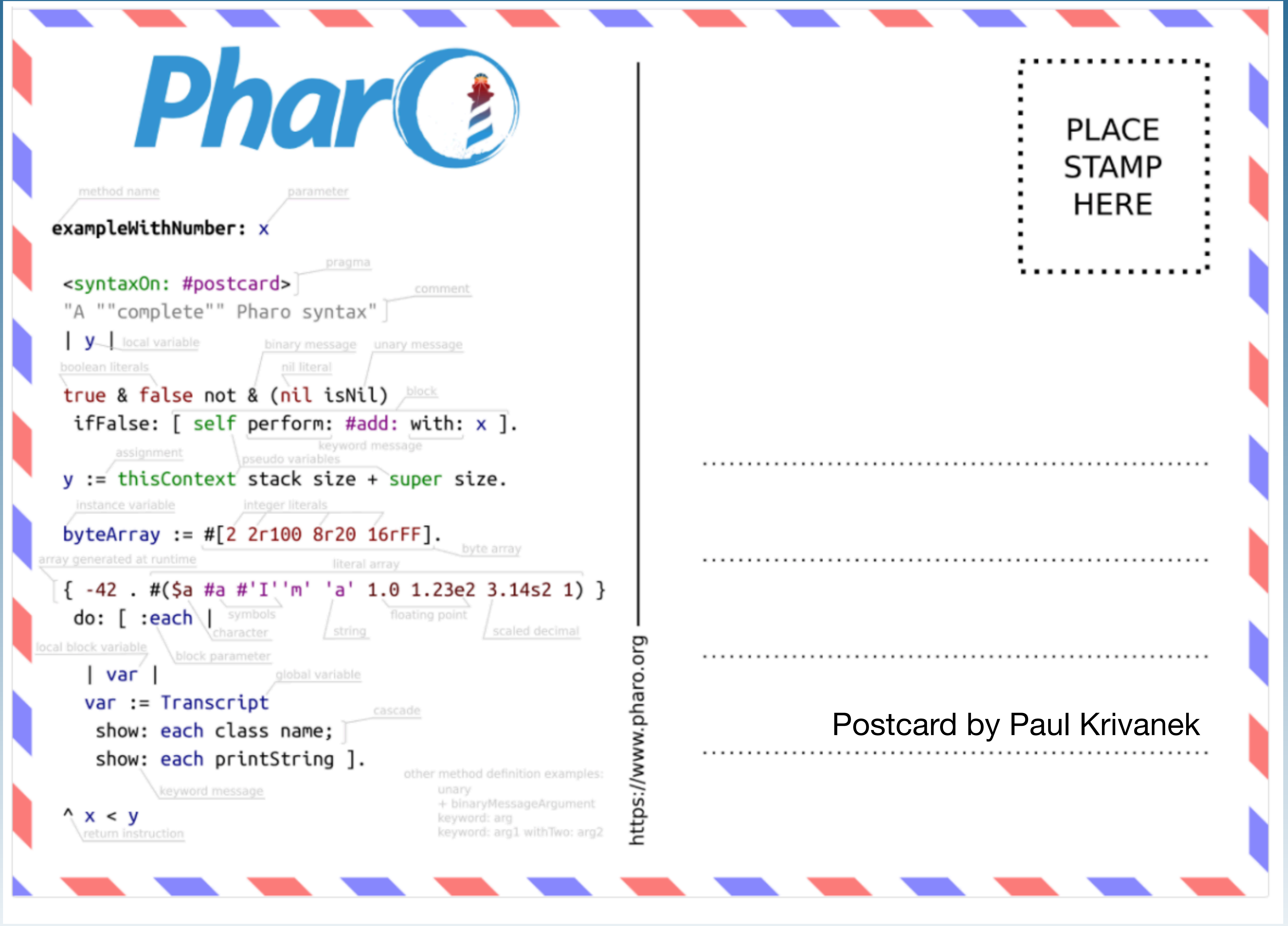
```
self
```

```
createCmajorMIDIPatchNamed: aName
    "the .cmajorpatch file is always generated "

    | cmajorPatchesDir patchDir stringToTrim streamToTrim newString trimIndex fileToWrite |
    cmajorPatchesDir := (FileLocator documents / 'cmajorPatches')
                            ensureCreateDirectory.
    patchDir := (cmajorPatchesDir / aName) ensureCreateDirectory.
```

Domenico Cipriani - 2024

SYNTAX FIT A POSTCARD



```
exampleWithNumber: x

 <syntaxOn: #postcard>
 "A ""complete"" Pharo syntax"
 | y |
 true & false not & (nil isNil)
  ifFalse: [ self perform: #add: with: x ].

 y := thisContext stack size + super size.

 byteArray := #[2 2r100 8r20 16rFF].

 { -42 . #($a #a #'I''m' 'a' 1.0 1.23e2 3.14s2 1) }
  do: [ :each |
    | var |
    var := Transcript
     show: each class name;
     show: each printString ].

 ^ x < y
```

Pharo
method name
parameter
pragma
comment
local variable
binary message
unary message
boolean literals
nil literal
block
keyword message
assignment
pseudo variables
instance variable
integer literals
byte array
array generated at runtime
literal array
symbols
character
string
floating point
scaled decimal
local block variable
block parameter
global variable
cascade
keyword message
return instruction
other method definition examples:
unary
+ binaryMessageArgument
keyword: arg
keyword: arg1 withTwo: arg2
https://www.pharo.org
PLACE STAMP HERE
Postcard by Paul Krivanek

- All **Pharo** syntax fit on a Postcard!

Rule 1: Everything is an Object
Rule 2: Every Class has a superclass
Rule 4: Everything happens by sending messages
Rule 5: Method lookup follows inheritance chain
Rule 6 : Classe are Objects too and they follow the same rules

Precedence rules:
1. Unary message (3 factorial)
2. Binary messages (3 + 5)
3. Keyword messages (Transcript show: ‘Hello’)

When multiple messages of the same precedence appear, Smalltalk evaluates them **from left to right**.

Domenico Cipriani - 2024

Postcard by Pavel Krivanek

pharo.org

WHAT IS FAUST?

1. **FAUST** is a purely functional programming language. It is considered state-of-the-art in the research and development of time-domain algorithms that can be represented as block diagrams, such as virtual analog synthesisers, filters, waveguide physical models, and reverbs.
2. **FAUST** standard libraries offer a ready-to-use, extensive collection of sound generators, physical models, DSP helper functions, and effects, all resulting from cutting-edge audio research supported by a large community.
3. **FAUST** architecture and its *C-box-API* enable embedding its compiler inside Pharo

WHAT IS IN FAUST LIBRARIES?

Oscillators	Basic Oscillators, Wave-Table-Based Oscillators, Low Frequency Oscillators, Alias-Suppressed Oscillators, Impulse Trains, Filter-Based Oscillators, Waveguide-Resonator-Based Oscillators, Casio CZ Oscillators, PolyBLEP-Based Oscillators
Filters	Basic Filters, Comb Filters, Ladder Filters, Digital Filter Sections Specified as Analog Filter Sections, Simple Resonator Filters, Butterworth Filters, Special Filter-Bank Delay-Equalizing Allpass Filters, Parametric Equalizers (Shelf, Peaking), State Variable Filters (SVF), …
Envelopes	A collection of linear and exponential envelope generators
Effects	Reverbs, delays, flangers, choruses, pitch shifters, mixers and saturators
Physical Modeling	String Instruments, Bowed String Instruments, Wind Instrument, Exciters, Modal Percussions, Vocal Synthesis
Analysis Tools	Amplitude tracking, spectrum-analysers, Fast Fourier transform
Dynamics processor	Compressors, limiters, expanders

WHY Cmajor ?

- We can easily export our DSP to a Cmajor plug-in thanks to the Faust compiler.
- We can use the plug-in we created we the Cmajor wrapper plug-in: https://github.com/cmajor-lang/cmajor/releases
- Cmajor allows simple procedural DSP code to be easily composed into graph structures.
- It makes impossible to write code that can crash or break real-time safety rules.
- It can be very easily learned by anyone who’s familiar with C/C++, javascript or other C-style languages.

Domenico Cipriani - 2024

pharo.org

LEARN PHARO

- The **Pharo** MOOC: https://mooc.pharo.org/ (7 weeks).
- Advanced OOP Design and Development with **Pharo**: https://advanced-design-mooc.pharo.org/ (10 modules)
- Its powerful reflection and inspection capabilities, allowing you to explore and understand the system interactively.".
- Free books! https://books.pharo.org/

Domenico Cipriani - 2024

pharo.org

INSTALL PHAUSTO

- First, download the **Pharo** launcher: https://pharo.org/download
- The *Pharo Launcher* is a tool allowing you to easily download Pharo core images.
- Download the packed *librariesBundle* for your platform from the Phausto repo, https://github.com/lucretiomsp/phausto
- Open a Playground (CMD +OW), then copy and evaluate (CMD+D) this script.

```
Metacello new
    baseline: 'Phausto';
    repository: 'github://lucretiomsp/phausto:main';
    load
```

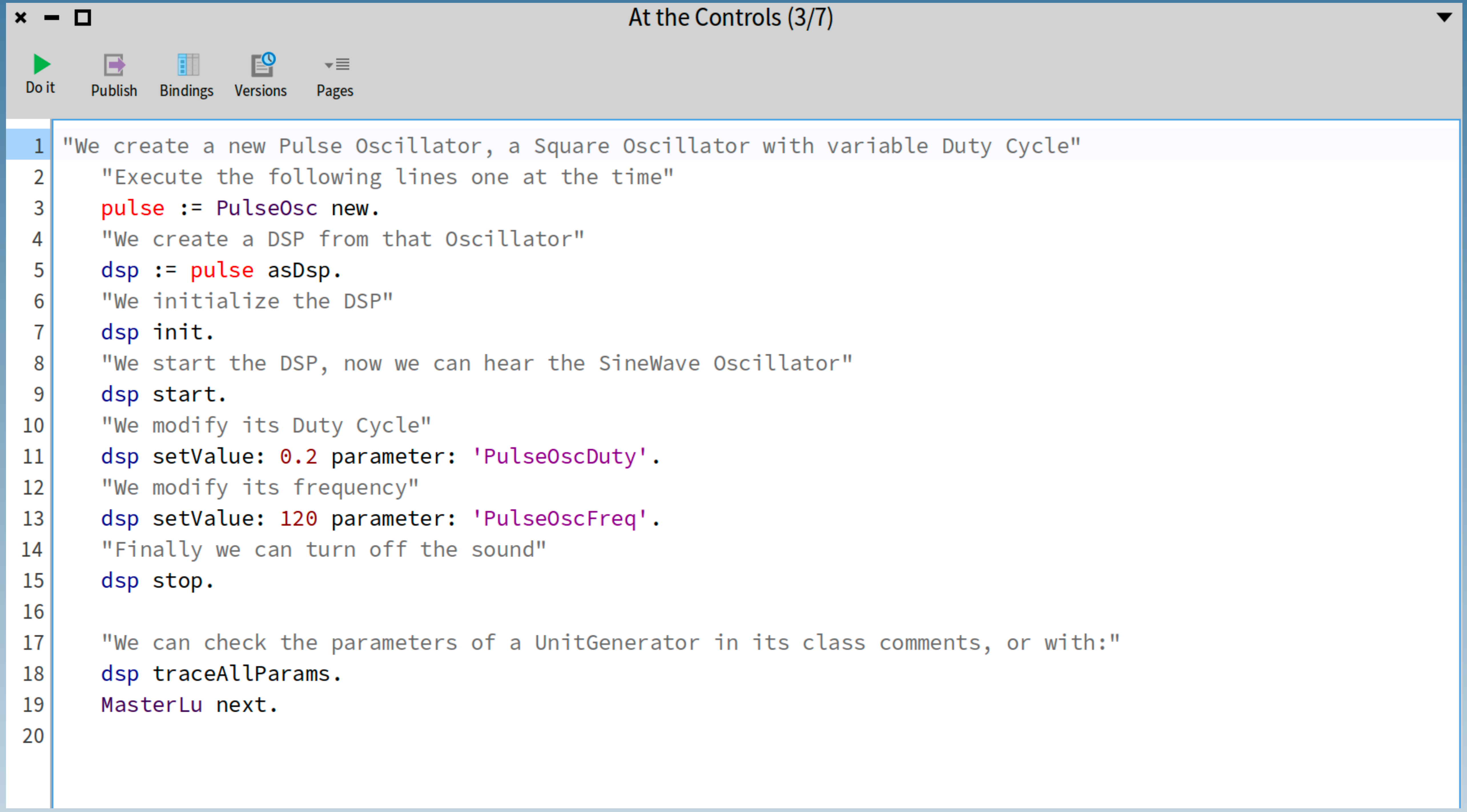
Domenico Cipriani - 2024

pharo.org

LEARN PHAUSTO

- Open a Playground and evaluate: MasterLu go.

  - The semantics of Phausto align closely with Faust.
  - We strive to keep parameters names identical whenever possible.
  - We support parallel, sequential, split and merge composition but wit Pharo syntax



```
"We create a new Pulse Oscillator, a Square Oscillator with variable Duty Cycle"
   "Execute the following lines one at the time"
   pulse := PulseOsc new.
   "We create a DSP from that Oscillator"
   dsp := pulse asDsp.
   "We initialize the DSP"
   dsp init.
   "We start the DSP, now we can hear the SineWave Oscillator"
   dsp start.
   "We modify its Duty Cycle"
   dsp setValue: 0.2 parameter: 'PulseOscDuty'.
   "We modify its frequency"
   dsp setValue: 120 parameter: 'PulseOscFreq'.
   "Finally we can turn off the sound"
   dsp stop.

   "We can check the parameters of a UnitGenerator in its class comments, or with:"
   dsp traceAllParams.
   MasterLu next.
```

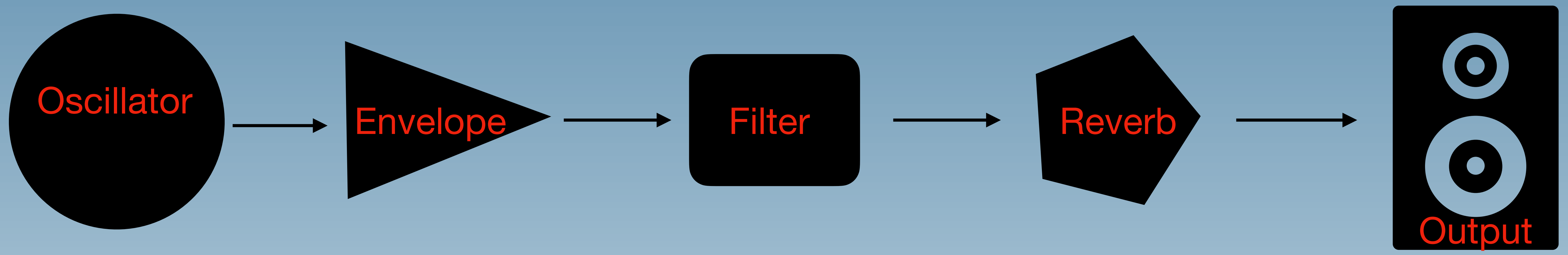
- Or visit: https://lucretiomsp.github.io/musicwithpharo/

Domenico Cipriani - 2024

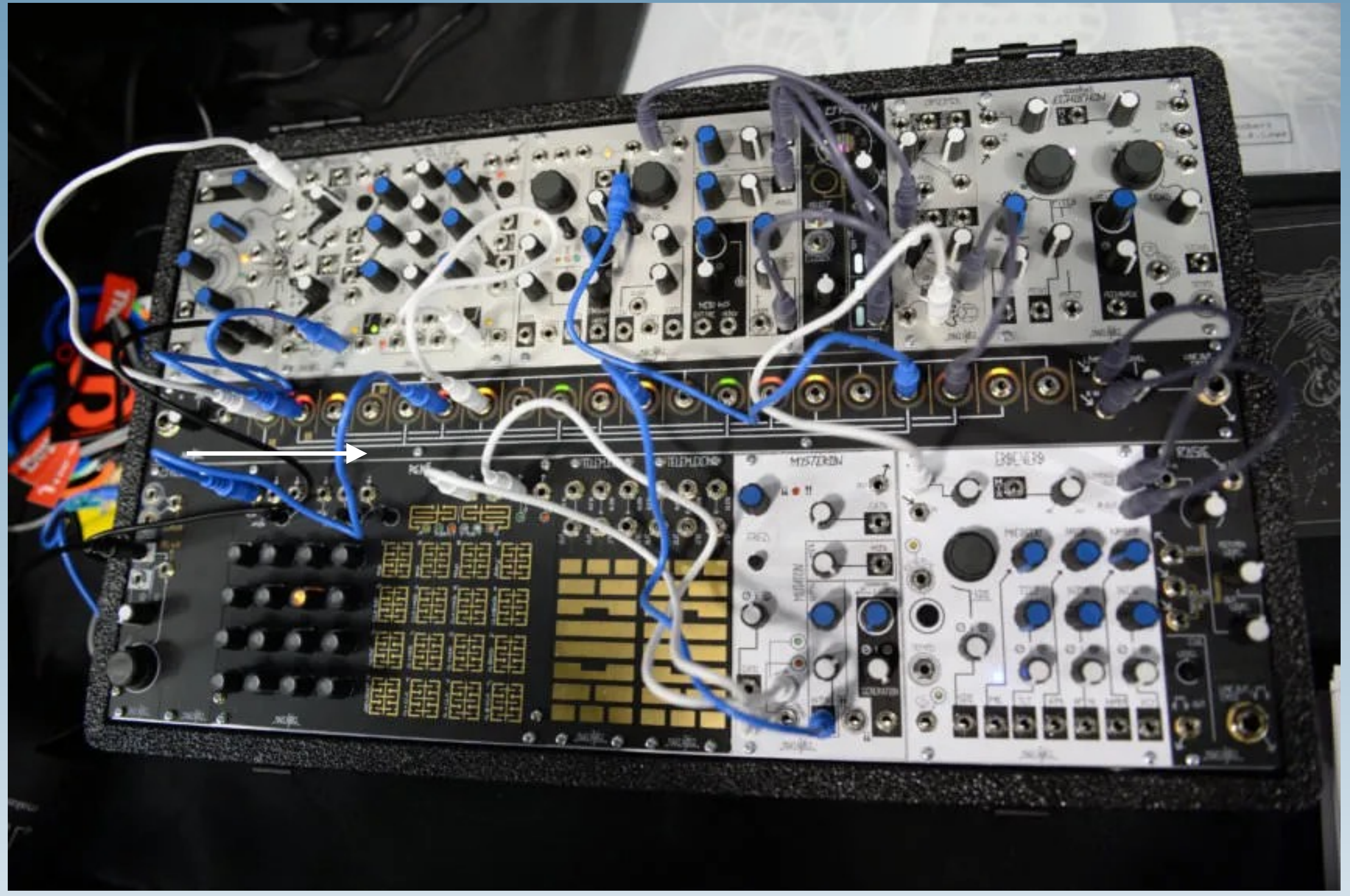
pharo.org

MODULAR DSP PROGRAMMING

- Phausto offers an approach to develop and design synthesisers and effect that is inspired by modular synthesiser patching.
- In Phausto, we connect Unit Generators setting their members value or using the **ChucK** operator **=>** *(That we kindly borrowed from our Chuckian friends)*
- Phausto organises and implement sthe functions and the semantics of FAUST standard library into *Unit Generators* subclasses drawing deep inspiration from the ChucK programming language.



```
synth  := PulseOsc new => ADSREnv new =>  ResonLp new =>  SatRev new.
dsp  := synth  new stereo asDsp.
dsp init.
dsp start.
```

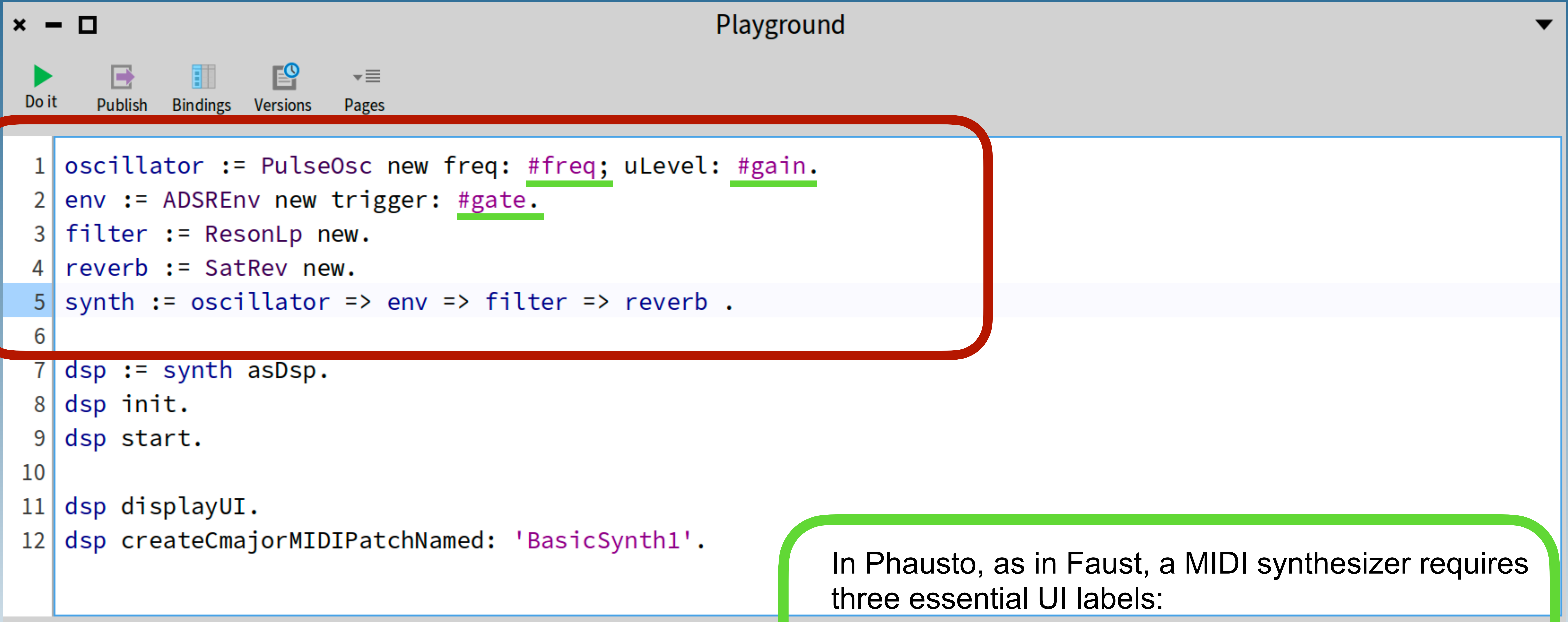


The concept of Unit Generator (UGens) as basic building blocks for signal processing algorithms was first developed by Max Matthews and John E.Muller for the Music III program n 1960.

Domenico Cipriani - 2024

pharo.org

CREATE A Cmajor PATCH



```
oscillator := PulseOsc new freq: #freq; uLevel: #gain.
env := ADSREnv new trigger: #gate.
filter := ResonLp new.
reverb := SatRev new.
synth := oscillator => env => filter => reverb .

dsp := synth asDsp.
dsp init.
dsp start.

dsp displayUI.
dsp createCmajorMIDIPatchNamed: 'BasicSynth1'.
```

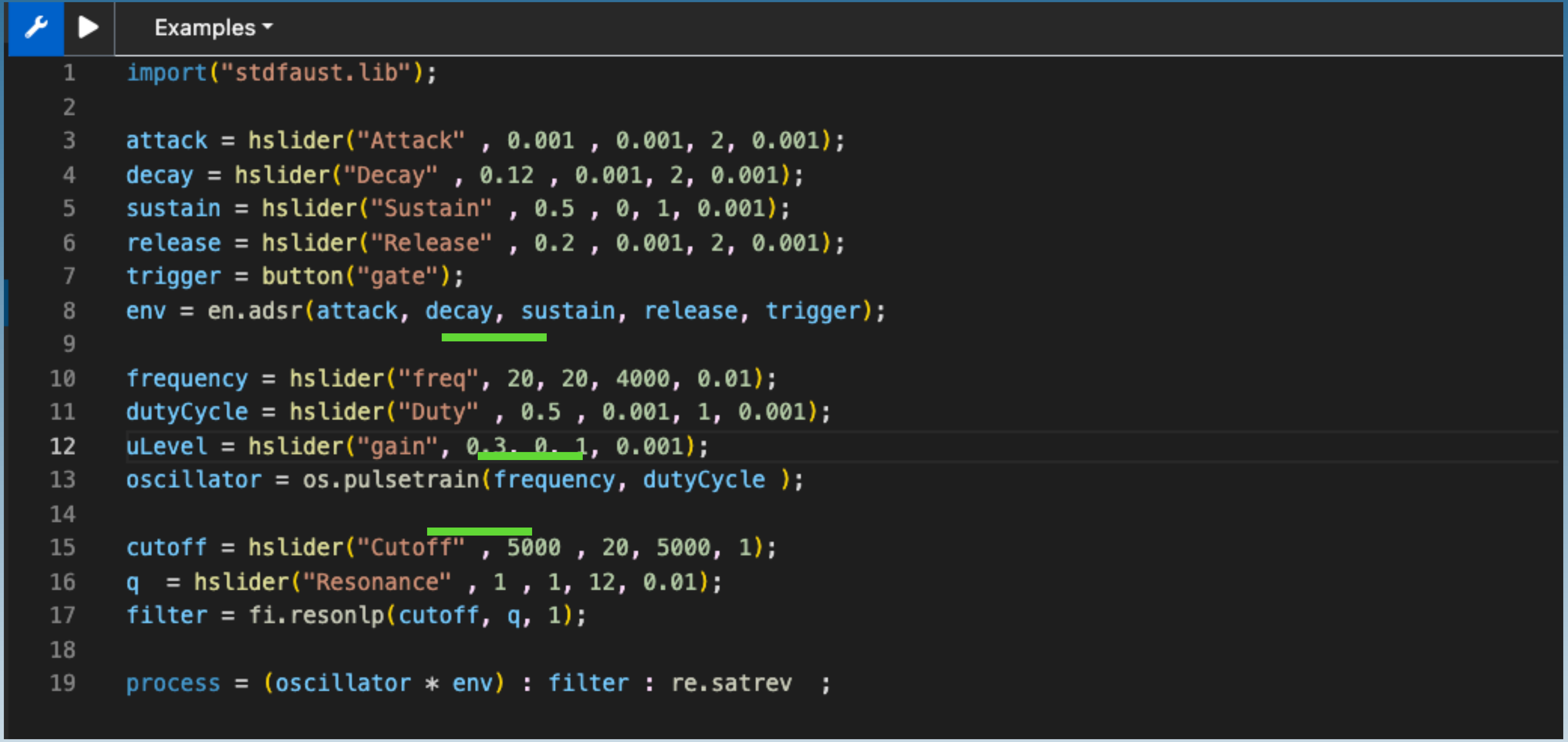
In Phausto, as in Faust, a MIDI synthesizer requires three essential UI labels:

- **freq**: Controls the oscillator's frequency, typically linked to MIDI note-on messages.
- **gate**: Manages note-on and note-off events to trigger sound.
- **gain**: Adjusts the output volume.

Domenico Cipriani - 2024

pharo.org

(Same DSP written in Faust)



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

attack = hslider("Attack" , 0.001 , 0.001, 2, 0.001);
decay = hslider("Decay" , 0.12 , 0.001, 2, 0.001);
sustain = hslider("Sustain" , 0.5 , 0, 1, 0.001);
release = hslider("Release" , 0.2 , 0.001, 2, 0.001);
trigger = button("gate");
env = en.adsr(attack, decay, sustain, release, trigger);

frequency = hslider("freq", 20, 20, 4000, 0.01);
dutyCycle = hslider("Duty" , 0.5 , 0.001, 1, 0.001);
uLevel = hslider("gain", 0.3, 0, 1, 0.001);
oscillator = os.pulsetrain(frequency, dutyCycle );

cutoff = hslider("Cutoff" , 5000 , 20, 5000, 1);
q  = hslider("Resonance" , 1 , 1, 12, 0.01);
filter = fi.resonlp(cutoff, q, 1);

process = (oscillator * env) : filter : re.satrev  ;
```

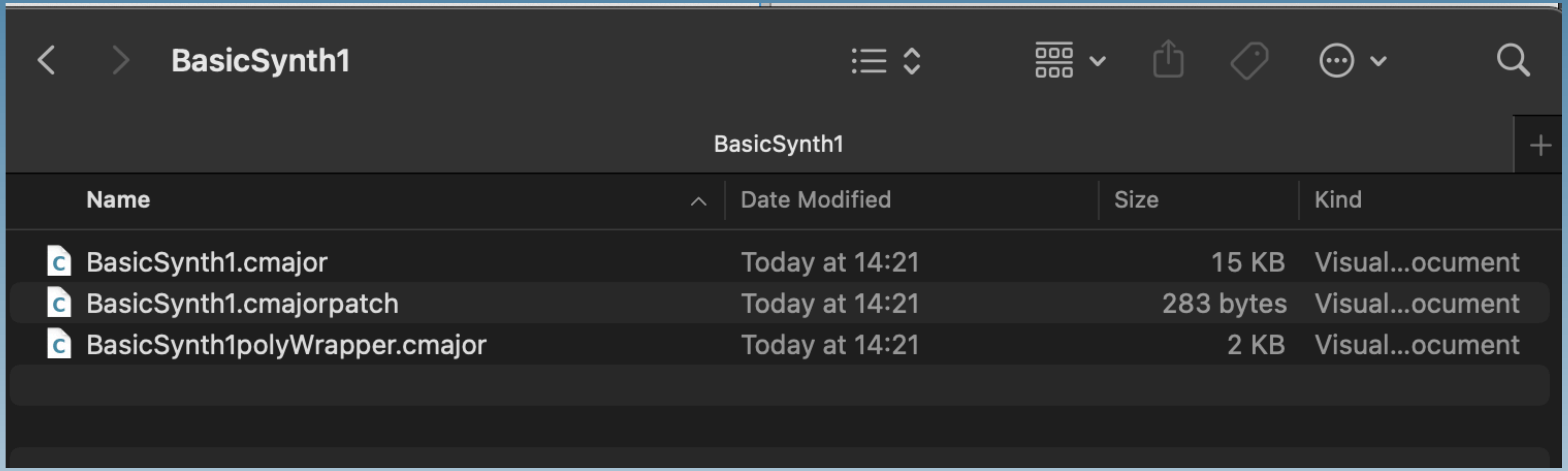
Phausto code is much shorter, as all *Unit Generators* come with a user interface with default parameters

Domenico Cipriani - 2024

CREATE A Cmajor PATCH

- dsp createCmajorMIDIPatchNamed: 'BasicSynth1'.

Creates a new folder inside documents/cmajorPatches/



Name	Date Modified	Size	Kind
BasicSynth1.cmajor	Today at 14:21	15 KB	Visual...ocument
BasicSynth1.cmajorpatch	Today at 14:21	283 bytes	Visual...ocument
BasicSynth1polyWrapper.cmajor	Today at 14:21	2 KB	Visual...ocument

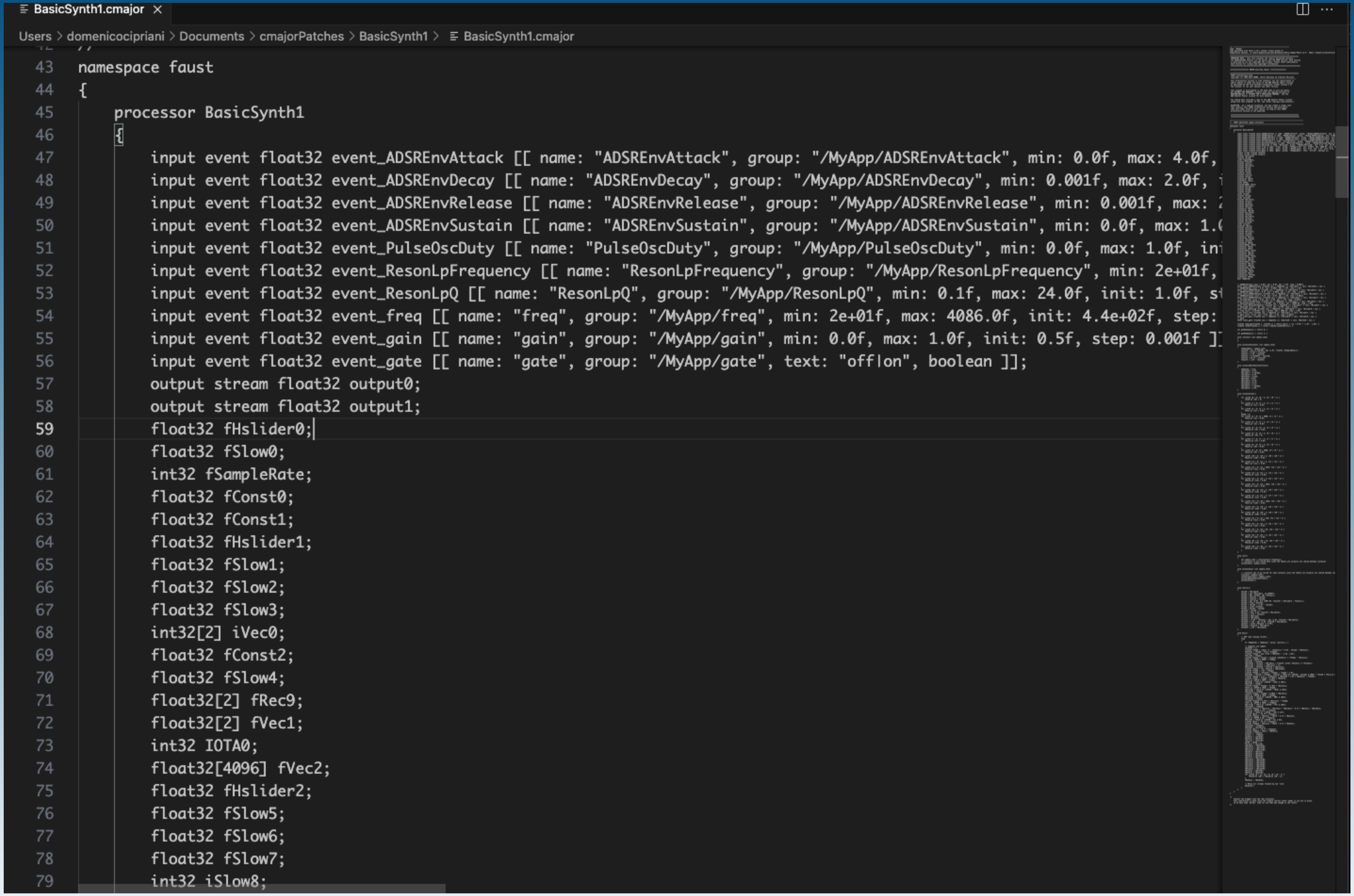
The folder contains a *.cmajor* file, a *.cmajorpatch* file and a *polyWrapper.cmajor* file (thanks Cesare Ferrari for the help!), which handles polyphony.

- dsp createCmajorFXPatchNamed: 'BasicEffect'.

Creates a Cmajor patch to be used on an audio track; the folder does not contains the *polyWrapper.*

Domenico Cipriani - 2024

pharo.org


BasicSynth1.cmajor
Users > domenicocipriani > Documents > cmajorPatches > BasicSynth1 > BasicSynth1.cmajor

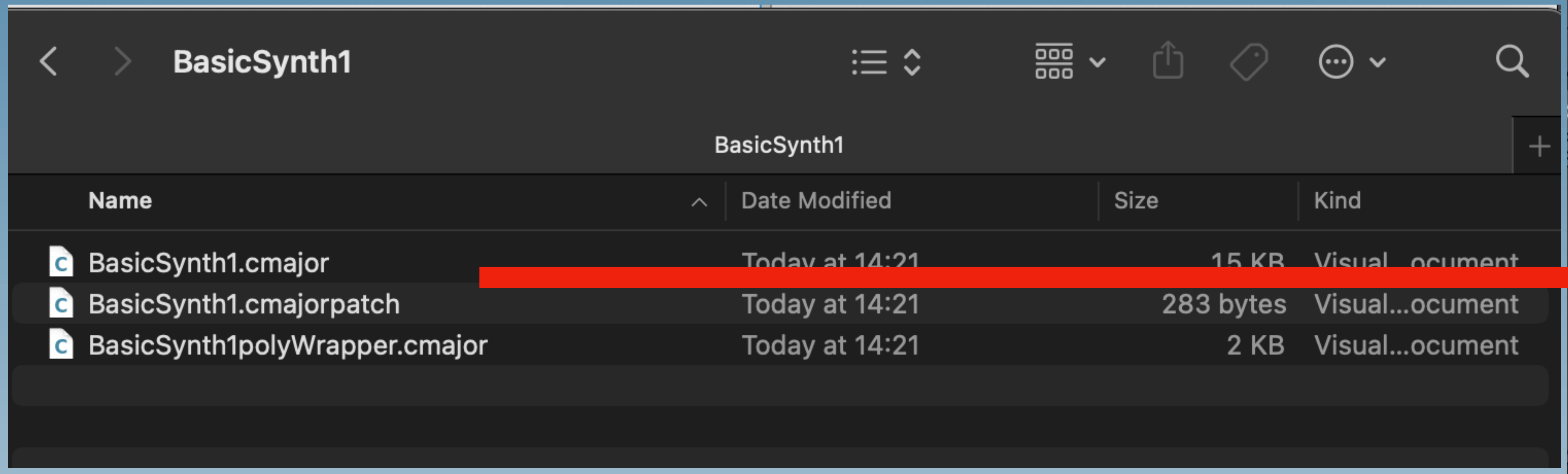
```
namespace faust
{
    processor BasicSynth1
    {
        input event float32 event_ADSREnvAttack [[ name: "ADSREnvAttack", group: "/MyApp/ADSREnvAttack", min: 0.0f, max: 4.0f,
        input event float32 event_ADSREnvDecay [[ name: "ADSREnvDecay", group: "/MyApp/ADSREnvDecay", min: 0.001f, max: 2.0f, i
        input event float32 event_ADSREnvRelease [[ name: "ADSREnvRelease", group: "/MyApp/ADSREnvRelease", min: 0.001f, max:
        input event float32 event_ADSREnvSustain [[ name: "ADSREnvSustain", group: "/MyApp/ADSREnvSustain", min: 0.0f, max: 1.
        input event float32 event_PulseOscDuty [[ name: "PulseOscDuty", group: "/MyApp/PulseOscDuty", min: 0.0f, max: 1.0f, in
        input event float32 event_ResonLpFrequency [[ name: "ResonLpFrequency", group: "/MyApp/ResonLpFrequency", min: 2e+01f,
        input event float32 event_ResonLpQ [[ name: "ResonLpQ", group: "/MyApp/ResonLpQ", min: 0.1f, max: 24.0f, init: 1.0f, s
        input event float32 event_freq [[ name: "freq", group: "/MyApp/freq", min: 2e+01f, max: 4086.0f, init: 4.4e+02f, step:
        input event float32 event_gain [[ name: "gain", group: "/MyApp/gain", min: 0.0f, max: 1.0f, init: 0.5f, step: 0.001f ]
        input event float32 event_gate [[ name: "gate", group: "/MyApp/gate", text: "off|on", boolean ]];
        output stream float32 output0;
        output stream float32 output1;
        float32 fHslider0;
        float32 fSlow0;
        int32 fSampleRate;
        float32 fConst0;
        float32 fConst1;
        float32 fHslider1;
        float32 fSlow1;
        float32 fSlow2;
        float32 fSlow3;
        int32[2] iVec0;
        float32 fConst2;
        float32 fSlow4;
        float32[2] fRec9;
        float32[2] fVec1;
        int32 IOTA0;
        float32[4096] fVec2;
        float32 fHslider2;
        float32 fSlow5;
        float32 fSlow6;
        float32 fSlow7;
        int32 iSlow8;
```

A short sample of the ~400 lines of **Cmajor** code generated by the **Faust** compiler

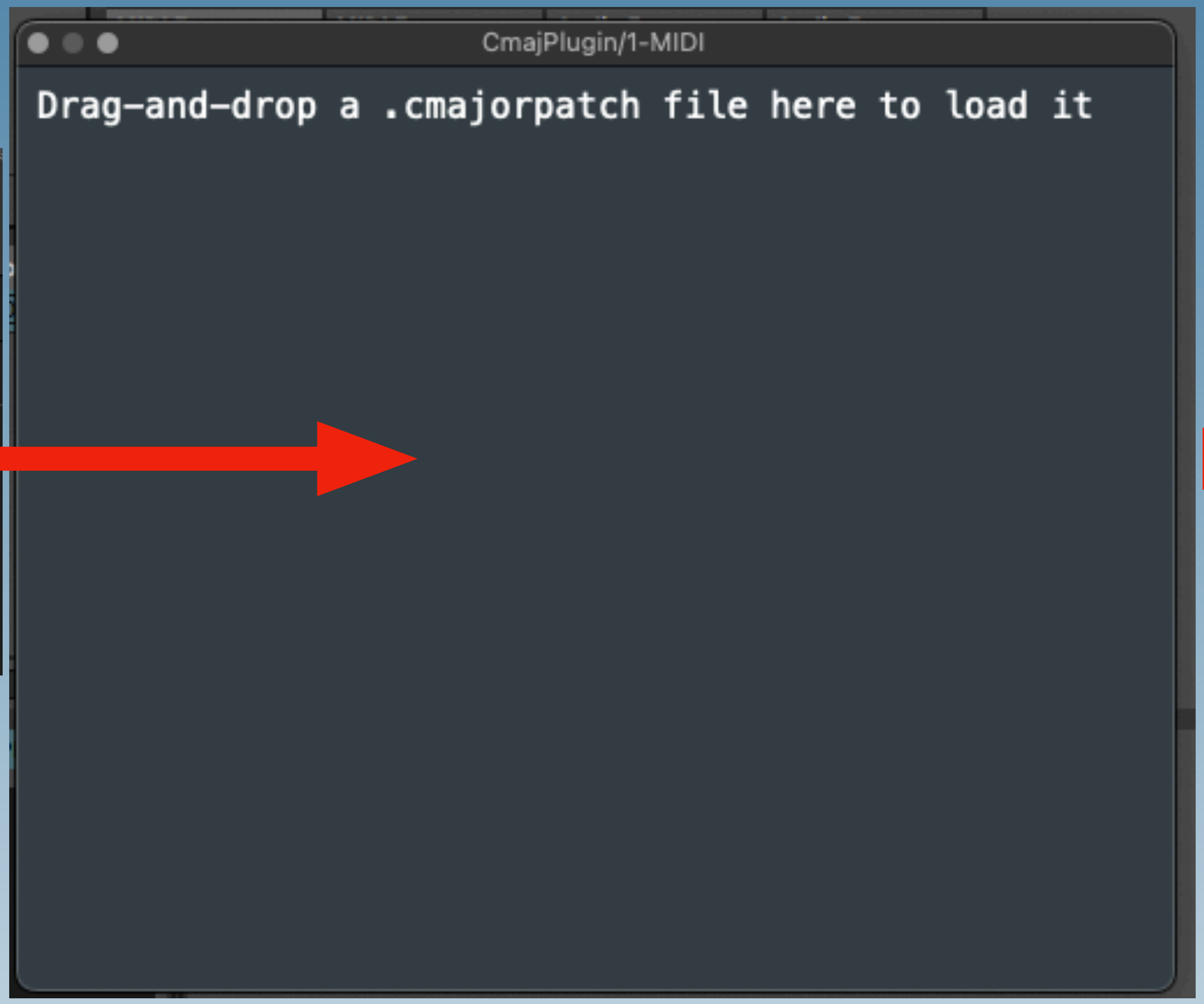
Domenico Cipriani - 2024

pharo.org

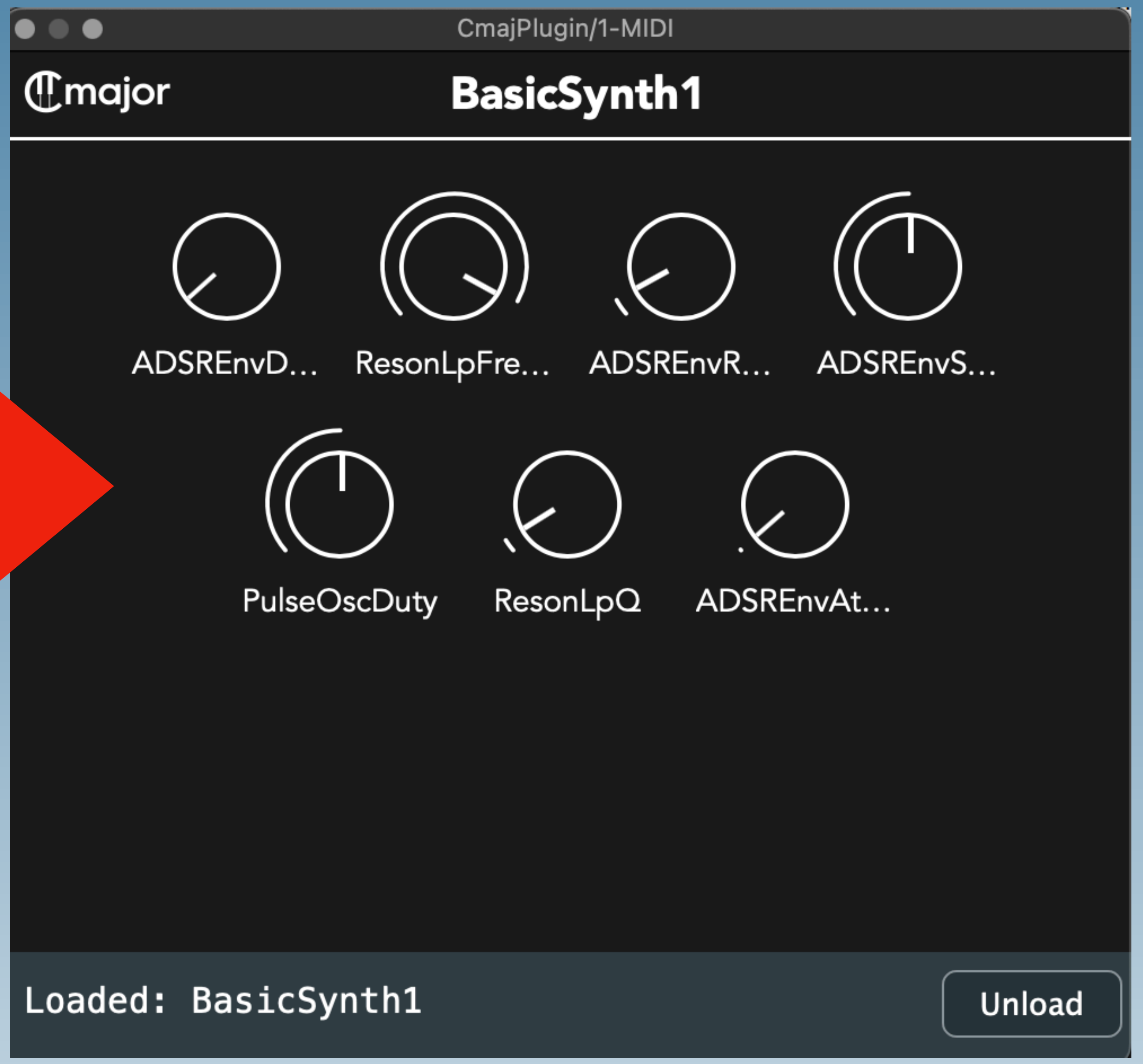
USE A Cmajor PATCH IN A DAW



Name	Date Modified	Size	Kind
BasicSynth1.cmajor	Today at 14:21	15 KB	Visual...ocument
BasicSynth1.cmajorpatch	Today at 14:21	283 bytes	Visual...ocument
BasicSynth1polyWrapper.cmajor	Today at 14:21	2 KB	Visual...ocument



Drag-and-drop a .cmajorpatch file here to load it

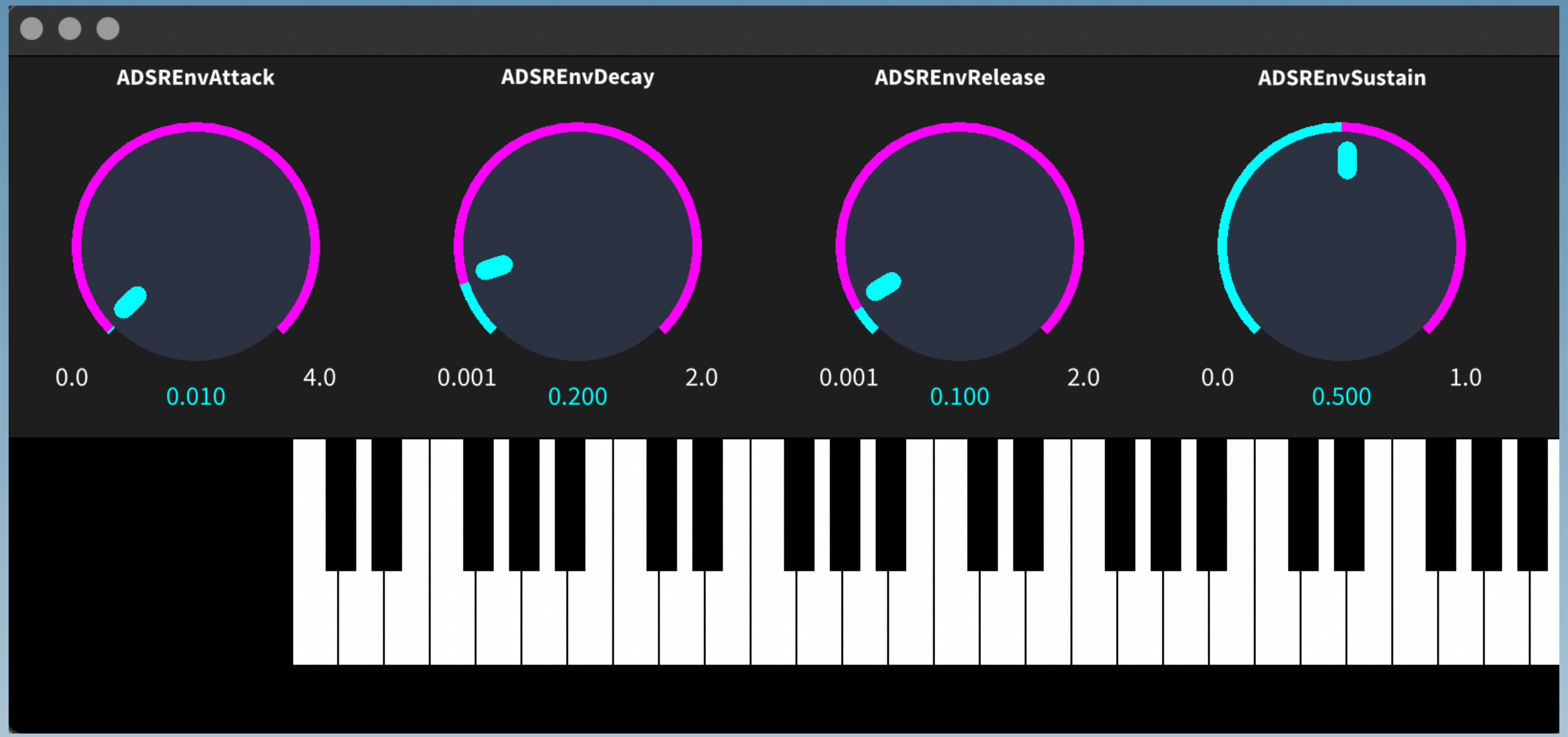


```
Loaded: BasicSynth1
```

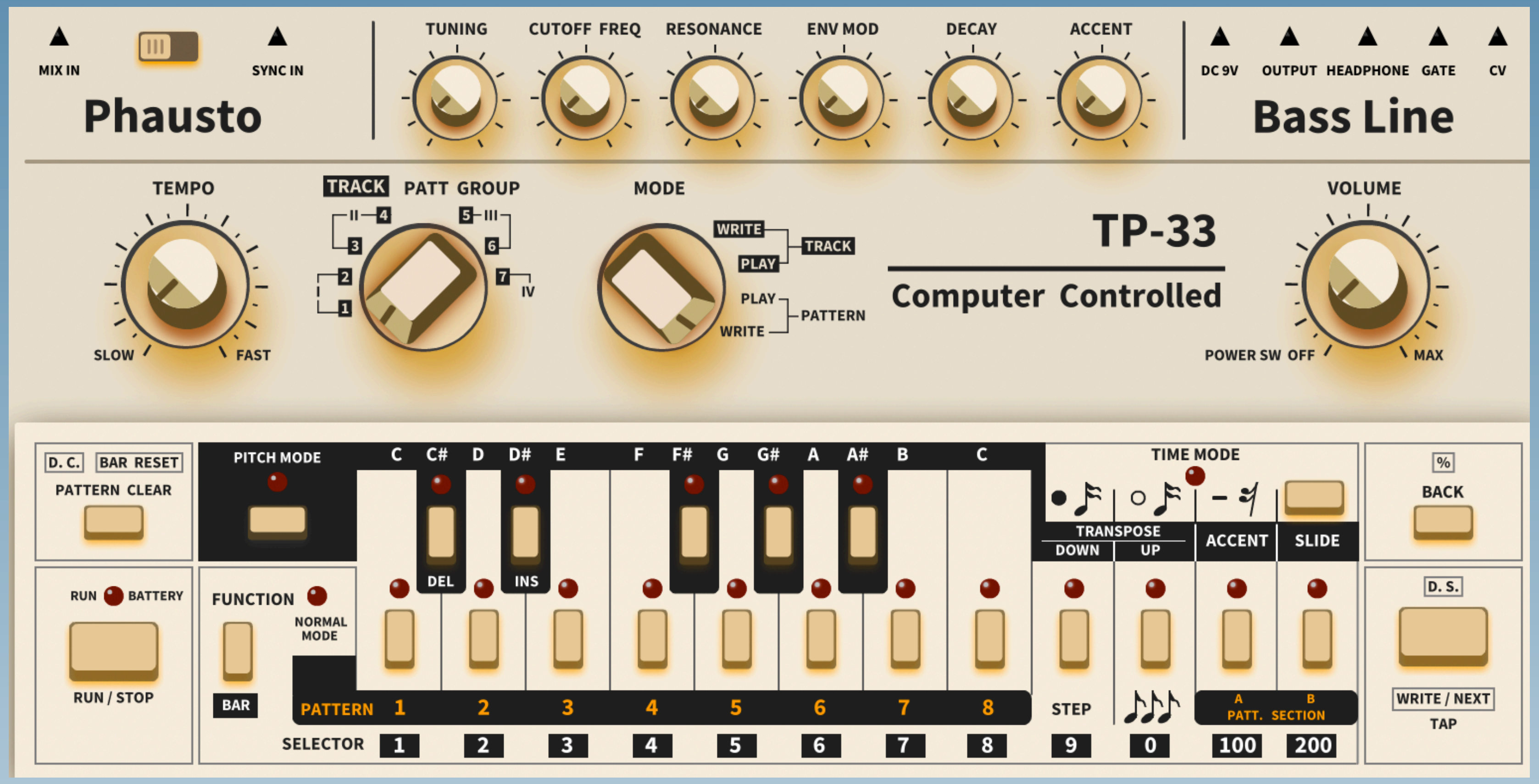
Domenico Cipriani - 2024

WHAT'S NEXT?

- Porting all the functions from the **Faust** standard library (65% covered at the moment).
- ‘*Auto-smoothing*’ on UI parameters
- Implement more *Toolkit* objects, i.e.higher level blocks to construct synthesiser and effects.
- Export a UI designed in **Pharo** with **Bloc** to a **Javascript** file.



ADSREnvAttack 0.0 0.010 4.0
ADSREnvDecay 0.001 0.200 2.0
ADSREnvRelease 0.001 0.100 2.0
ADSREnvSustain 0.0 0.500 1.0



Bloc is a low level UI infrastructure and framework for Pharo

- Additional (video) tutorials and a comprehensive booklet on **Phausto**.
- Encouraging sound artists and producers to use plug-ins from Cmajor patches by developing a series of low cost synths and effects by them available to a ready-to-install Cmajor plug-in wrapper from the **SoftComputing** Bandcamp

Domenico Cipriani - 2024

pharo.org