

PROGRAMMING MUSIC AND SYNTHESIZERS ON-THE-FLY WITH PHARO



Domenico Cipriani - ADC25

Inria



AGENDA

1. Foundations [45 minutes]
2. Rhythm and Melodies [75 minutes]
3. Sound Design with Phausto [75 minutes]
4. Performance [20 minutes]
5. Visual Interface and Control [25 minutes]



PART 1: FOUNDATIONS



Domenico Cipriani - ADC25

Inria



What is live coding?

- Live coding means writing and modifying code in real time to create music, visuals, or performances.
- The code itself becomes part of the live performance, often projected for the audience to see.
- It's about improvisation and creativity — composing and changing sounds or visuals on the fly.
- Common in electronic music and audiovisual art, using languages like SuperCollider, TidalCycles, or Sonic Pi.
- Focuses on process over perfection — the act of coding *is* the art.

Pharo, Coypu, Phausto

PHARO

Free, open-source, general-purpose
language + cross-platform ID

COYPU

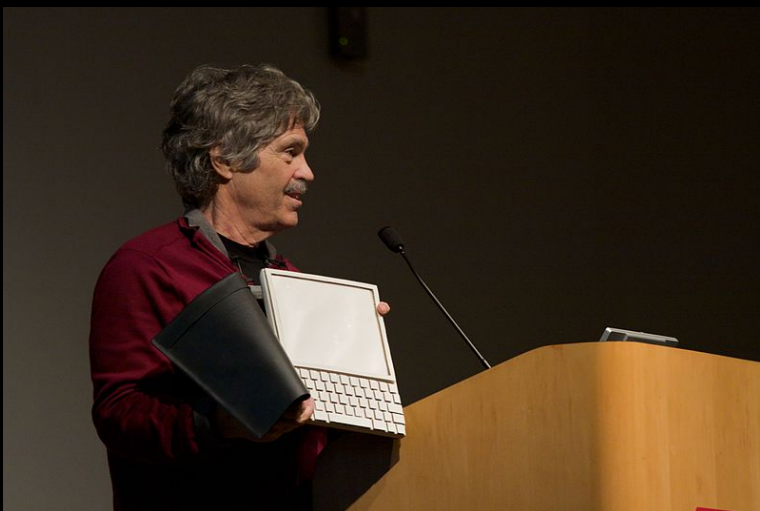
API and Domain Specific Language for
programming music on-the-fly

PHAUSTO

Library and API for DSP programming that
enables sound generation within Pharo

Smalltalk

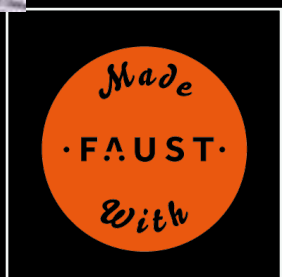
- A pure **Object-Oriented** Programming language developed at the *Learning Research Group* at Xerox PARC in the 1970s by Alan **Kay**, **Dan Ingalls**, **Adele Goldberg**, **Ted Kaeheler**.
- Designed for educational use following principles of *Constructionism*.
- Deeply influenced by **Simula**, developed by **Ole-Johan Dahl** and **Kristen Nygaard** in the 1960s at the Norwegian Computing Center in Oslo.
- Not just a language, also an IDE users can inspect and modify.
- Programs written in Smalltalk are compiled into byte code and interpreted by a virtual machine.
- Smalltalk has influenced *Objective C*, *Ruby*, *SuperCollider*.



Domenico Cipriani - ADC25



Inria



Install Pharo

Download the Pharo Launcher to download a Pharo Image:

<https://pharo.org/download>

A Pharo image is an object space + Pharo Core Libraries + the virtual machine



The Pharo IDE

The screenshot displays the Pharo 14.0 IDE interface with the title bar "Pharo 14.0 - ROBOTCOYPU.image". The interface is divided into several panes:

- Transcript:** Shows a list of messages sent to objects, including "myArgv:" and "create DSP WitName: Destroyed libContext 2025-10-10T10:05:59 INIT EXECUTED".
- Inspector:** Displays the state of a "SequencerMono" object with variables like "self", "seqKey", "extraParams", "notes", "durations", "noteIndex", and "gates".
- Playground:** Contains a code editor with the following code:


```
1 PortMidi traceAllDevices .
2
3 mout := MIDISender new.
4
5 mout openWithDevice: 4.
6
7
8 p := Performance uniqueInstance .
9 p performer: PerformerMIDI new
10
11
12 '9090' hexBeat midiCh: 1 to: #kick; ccn: 20 ccv: 1 .
13 #downbeats asRhythm midiCh: 1 to: #kick.
14 '0202' hexBeat midiCh: 2 to: #snare.
15 #semiquavers asRhythm midiCh: 5 to: #ch.
16 '60/64' asDirtNotes midiCh: 5 to: #vox.
17 '64/ 16 , 72/16, 67 /16, 69 /16' asDirtNotes midiCh: 4 to: #lead.
18 '60/4 , 60/4 , 48/56' asDirtNotes midiCh: 3 to: #bass.
19
20 p mute: #lead.
21 p solo: #lead.
22 p freq: 144 bpm.
23 p playFor: 64 bars.
24 p stop.
```
- Process Browser:** Lists running processes such as "(80) DelaySemaphoreSched", "(70) Callback queue", "(60) Low Space Watcher", "(60) SDL2 Event loop", "(50) Finalization Process", "(40) WatchDog (f05cf024-a4)", "(40) ThreadSafeTranscriptw", "(40) Morphic UI Process", "(40) Morphic UI Process", "(31) Calypso update", "(30) 819888384", and "(10) Idle Process".
- Method: GreyHoleDW>>asBox:** Shows the implementation of the "asBox" method for the "GreyHoleDW" object.
- Instance of PortMidiLibrary class did not understand #traceAllDevices:** Displays an error message and a "Create method" button.
- PortMidiLibrary traceAllDevices:** Shows the implementation of the "traceAllDevices" method for the "PortMidiLibrary" class.

Domenico Cipriani - ADC25

Inria



Pharo syntax on a postcard



method name parameter

exampleWithNumber: x

pragma comment

<syntaxOn: #postcard> "A "complete" Pharo syntax"

local variable binary message unary message

boolean literals nil literal block

true & false not & (nil isNil)

keyword message

ifFalse: [self perform: #add: with: x].

assignment pseudo variables

y := thisContext stack size + super size.

instance variable integer literals byte array

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

array generated at runtime literal array

{ -42 . #(\$a #a #'I'm' 'a' 1.0 1.23e2 3.14s2 1) }

local block variable symbols character string floating point scaled decimal

do: [:each |

block parameter global variable

| var |

var := Transcript

keyword message cascade

show: each class name;

show: each printString].

keyword message

^ x < y

return instruction

other method definition examples:

- unary
- + binaryMessageArgument
- keyword: arg
- keyword: arg1 withTwo: arg2

PLACE
STAMP
HERE

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').
4. Multiple messages with the same precedence are evaluated **from left to right**.

<https://www.pharo.org>

Installing Coypu and Phausto

Go to the *Coypu* GitHub repositories:

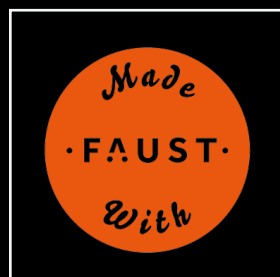
<https://github.com/lucretiomsp/Coypu>

Copy the *Metacello* script into your *Playground*.

```
Metacello new  
  baseline: 'Coypu';  
  repository: 'github://lucretiomsp/coypu:master';  
  load
```

Select all the text and evaluate (CMD/CTRL + D).

Coypu already comes together with *Phausto*

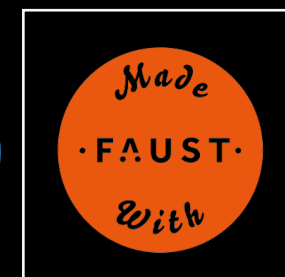


PART 1: FOUNDATIONS



Domenico Cipriani - ADC25

Inria

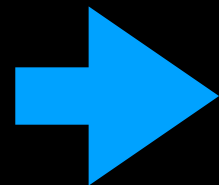


Coypu's anatomy



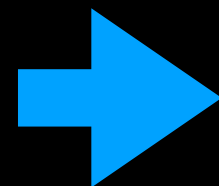
Principles

- **Iconicity**



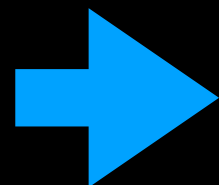
Written code should resemble what we hear
16 upbeats

- **Economy**



The less we type, the better
#(60 63 67) + 16

- **Polysemy**



Many ways to do the same thing

16 downbeats.

#downbeats asRhythm.

'60 , ~ , ~ , ~ , 60 , ~ , ~ , ~ , 60 , ~ , ~ , ~ , 60 , ~ , ~ , ~' asDirtNotes.
#(1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0) asSeq.



Connecting with audio servers



Creating rhythms

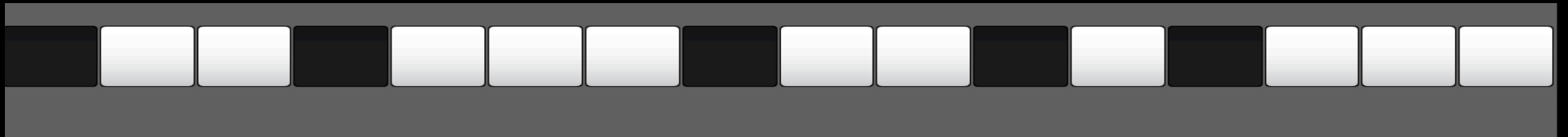


A rhythm can be represented as an array of 0s and 1s, where each 1 represents a trigger.



BINARY	100010001000
HEXADECIMAL	8888
PHARO	1000100010001000
COYPU	#downbeats asRhythm
COYPU	16 downbeats
COYPU	'8888' hexBeat

Creating rhythms



- `#(1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0)`
- `#rumba` asRhythm
- `'9128'` hexBeat



- `#(1 0 1 0 1 0 1 0 1 0 1 1 1)`
- 12 quavers, 4 semiquavers
- `'AAAF'` hexBeat

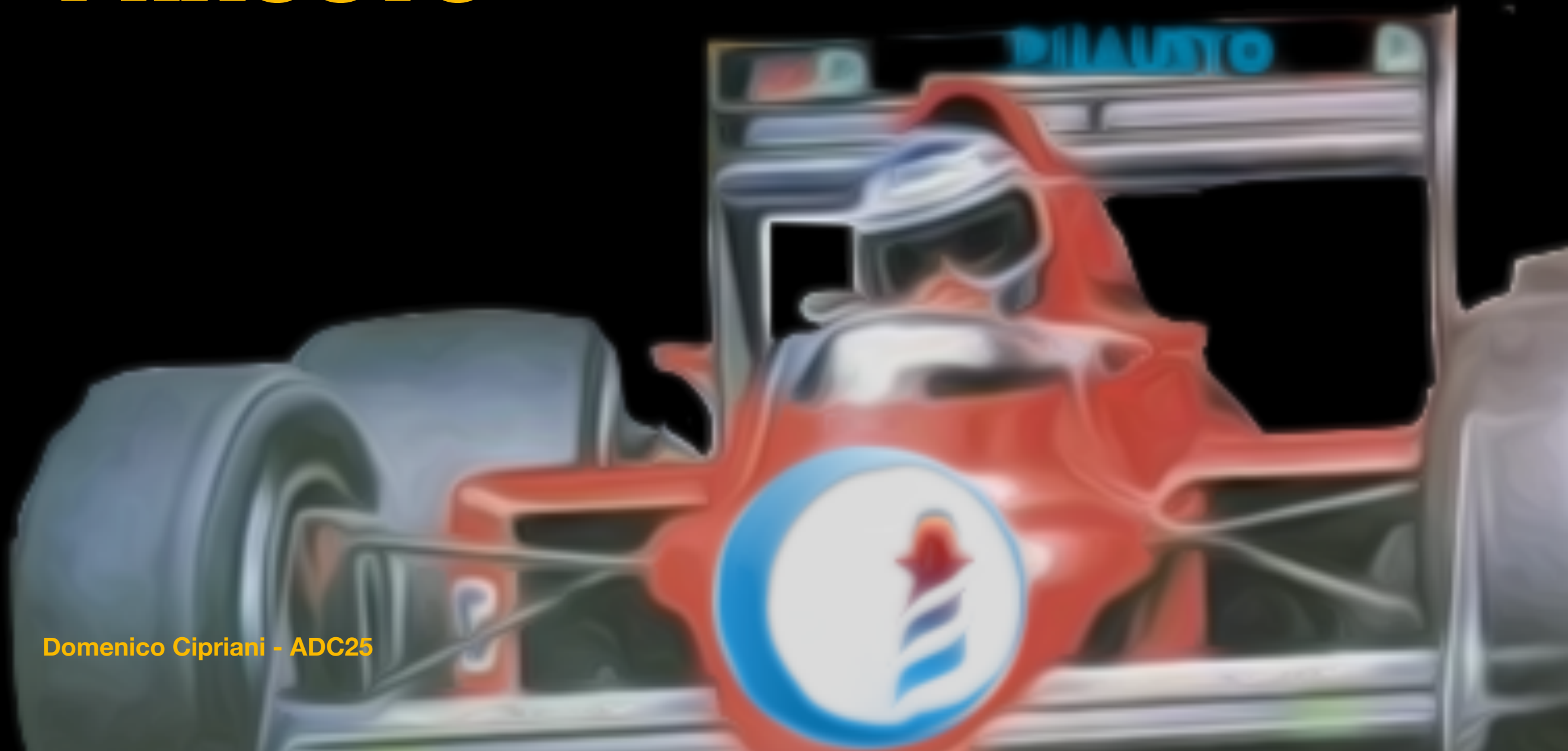
The 'musky' notation

Domenico Cipriani - ADC25

Inria



PART 3: SOUND DESIGN WITH PHAUSTO



Domenico Cipriani - ADC25

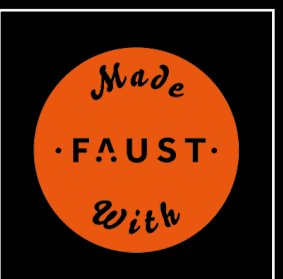
Start your engine

Domenico Cipriani - ADC25

Inria



Let's build a synth



Drive Phausto with Coypu



Build your own library

Domenico Cipriani - ADC25

Inria



PART 4: PERFORMANCE



Domenico Cipriani - ADC25

Inria



PART 5: VISUAL INTERFACE

