Sardine: a Modular Python Live Coding Environment

Raphaël Forment
Université Jean Monnet (Saint Étienne, ECLLA)
raphael.forment@gmail.com

ABSTRACT

Sardine is a live coding environment and library for Python 3.10+ focusing on the modularity and extensibility of its base components (clocks, parser, *handlers*). Sardine has been designed to be easily integrated with existing *live-coding* environments as both a tool for experimentation and demonstration of various live coding techniques : temporal recursion, patterning, integration in various hardware and software setups. Although the tool is still in active development, it has already been used in multiple performances and algoraves. This paper is dedicated to the introduction of the **Sardine** system and the explanation of the main guidelines currently followed by contributors to the project. It will also present the preliminary results of our work through practical realizations that served as objectives and experimental validations for the first stages of development.

1 Introduction

The Python language, with its large collection of libraries and modules, makes for a suitable environment for implementing a live coding library. The popularity of Python ensures good user support and many options for customization and integration into different text editors and running environments. The possibility to enjoy a powerful ecosystem of libraries focused on input/output, network communication or data processing makes it a prime target for the implementation of a library focused on experimentation in digital interactive media. Moreover, thanks to its lightweight syntax, Python can be read by programmers coming from different domains with a minimal adaptation time, making it a great platform for sharing information, collaborating and demonstrating the implementation of a given feature.

Sardine is born out of a curiosity for similar Python-based live-coding libraries such as FoxDot, Isobar or the more recent TidalVortex. It is also best described as an attempt to re-create and bring the concept of temporal recursion and TidalCycles patterning to Python.

2 Methodology and objectives: a framework for exploring live-coding in Python

3 Sardine implementation

Sardine is implemented and distributed as two Python modules: **Fishery** and **Sardine**.

3.1 Clock and Scheduling System

3.2 Sardine Pattern Language

A small patterning language has been developed for **Sardine** using the Lark parsing toolkit. Defined as a LALR parser, the syntax of the language is best described as a list-based calculator capable of dealing with basic MIDI note definition, custom chance operators and other composition tools.

3.3 Players and Handlers

Description of the event based system. How to define an handler, what is an hadler, etc...

Demo of the **SuperDirt** handler, etc...

4 Sardine usage

Basic facts about the usage of Sardine in various text editing environments + how to install and handle a Sardine installation.

4.1 Algorave and performance

Zorba, Lorient, example code taken from performances.

4.2 Controlling Legacy MIDI Synthesizers

Rémi Georges usage of Sardine: controlling legacy synthesizers along with TidalCycles, etc…

4.3 Usage of Sardine at the IIL Laboratory

Projects involving the Magnetic Resonator Piano, Boids, etc…

5 Project directions

5.1 Packaging and distribution

Distribution and release for Python 3.11 with updated C++ dependencies whenever possible. Distribution on Pypi when it'll be bug free, etc…

5.2 Opening up for collaboration

Documenting, section about the website and integration of the Sardinopedia.

5.3 Creation and performance

6 Conclusion

Call for contributors, etc…

7 Acknowledgments

I warmly thank my thesis supervisors Laurent Pottier and Alain Bonardi for their support and advice in the creation of this tool. I thank the doctoral school *3LA* from the University of Lyon for the funding it provided to this research. I extend my thanks to the musicians and friends who allowed me to take **Sardine** on stage and to present it to a wider audience these few last months: the **Cookie Collective**, Rémi Georges, etc…

Welcome to the markdown (aka commonmark) template for the International Conference on Live Coding 2023.

This document is a guide to using markdown for the conference, and is itself written in markdown. For full understanding, refer to iclc2023.txt to see the source of this document, and iclc2023.pdf to see the typeset output. Use of this template is currently only recommended for those familiar with commandline tools.

We suggest you take a copy of this template (iclc2023.txt), and use it as a starting point for your ICLC paper.

Preparing your submission using markdown will enable us to make proceedings available both in PDF files suitable for print, and in HTML suitable for the web. This is useful for making sure your paper is fully accessible, via Internet search, and with assistive technology such as screen readers for blind people. We recommend taking a straightforward approach to formatting your document.

If you do not wish to use markdown, please do not be discouraged from submitting your paper. There is also a word document template available from the conference website.

8 Learning and using markdown

We are happy to answer any questions you have about markdown in connection with your conference submission.

If you have questions, you can email us directly: iclc@creativecodingutrecht.nl

8.1 Running pandoc

Pandoc is software which turns text written in markdown into a beautiful looking document, complete with references. You will need to run it to create PDF documents of your paper for checking and uploading for peer review.

You may download pandoc for all major operating systems (including MS Windows, Apple Mac OS and GNU/Linux) from the following website: http://pandoc.org

As an alternative to the above downloads, on OS X only, the homebrew package manager can be used to install pandoc: http://brew.sh/

If you use homebrew to install on OS X you will need to install the pandoc package as follows:

```
brew update
brew install pandoc
```

To produce PDF files you will need to have LaTeX installed, as well as pandoc. See the pandoc website for installation instructions: http://pandoc.org/installing.html. LaTeX is used internally, you will not have to edit any LaTeX documents.

To render your markdown source as HTML, open a terminal window, change into the folder where the template is and run the following command:

```
pandoc --template=pandoc/iclc.html --citeproc --number-sections iclc2023.md -o iclc2023.html
```

To produce a PDF document, make sure you have LaTeX installed (see above), and run the following:

```
pandoc --template=pandoc/iclc.latex --citeproc --number-sections iclc2023.md -o iclc2023.pdf
```

For a higher quality output, add the option `--latex-engine=xelatex` to the above. You will need the Inconsolata and Linux Libertine opentype fonts installed.

An example Makefile is also provided to run these commands for you. If you have *make* installed, you can use it to build the pdf files.

8.2 Bibliographic references

Pandoc accepts bibliographic databases in a range of formats, so make sure you have the right extension on your file.

Table 1: Supported bibliography formats with file extension.

Format	File extension
MODS	.mods
BibLaTeX	.bib
BibTeX	.bibtex
RIS	.ris
EndNote	.enl
EndNote XML	.xml
ISI	.wos
MEDLINE	.medline
Copac	.copac

Figure 1: *A descriptive caption should be given for all figures, understandable without reference to the rest of the article.*

Format	File extension
JSON citeproc	.json

Authors may be referenced in two ways; inline, e.g. Schwitters (1932) wrote the Ursonate sound poem, or in parenthesis, e.g. Ursonate is a sound poem (Schwitters 1932). Multiple references should be grouped together like so (Schwitters 1932; Miller 1956; Greenewalt 1946).

The pandoc command given in the above section will automatically render your references according to Chicago author-date style.

At the head of the markdown source file for this template, you will see an entry for "bibliography" that points to the file references.bib. Here you'll find examples of bibliography entries in BibLaTex format, including examples for articles, books, book chapters and items from conference proceedings.

8.3 Code

We have chosen a single column layout to better support code examples without having to break lines. The following shows how to include a code example with syntax highlighting:

```
d1 $ every 3 (iter 4) $ brak $ "bd [sn [[sn bd] sn]]*1/3"
```

For more information please visit this page: [https://hackage.haskell.org/package/skylighting]

8.4 Figures

Images should be included as figures, with captions provided and formatted as shown in Figure 1. Be prepared for the page layout and image size to be changed during the editing and layout process, and consider this when referring to the figures in the text.

9 Conclusion

We look forward to receiving your completed papers. Submission is through the online peer review system at https://iclc2023.creativecodingutrecht.nl/openconf.php only. Do not send papers directly by e-mail. In all cases, please submit a PDF version of your paper for peer review. At a later stage in preparing the proceedings, we may ask for the markdown or Word versions of your paper.

9.1 Acknowledgments

At the end of the Conclusions, acknowledgements to people, projects, funding agencies, etc. can be included after the second-level heading "Acknowledgments".

References

10 Greenewalt, Mary H. 1946. *Nourathar, the Fine Art of Light Color Playing*. Philadelphia. Pa. Westbrook.

Miller, G. A. 1956. "The Magical Number Seven Plus or Minus Two: Some Limits on Our Capacity for Processing Information." *Psychological Review* 63 (2): 81–97.

Schwitters, Kurt. 1932. "Ursonate." *Merz* 24.