What is Live Coding?

SOUNDS WEIRD
Learning Platform
for Sound Experiments

Live coding is a way of creating art by using programming. It is comparable to playing a musical instrument. You need to practice live coding, you need to get to know it as an instrument, deal with it, and find your own ways of playing it and expressing yourself. In live coding programming is both creating a tool and playing on it at the same time. Thor Magnusson, an artist and researcher in this field, commented quite humorously when he writes in his excellent book **Sonic Writing** that 'the **whole point** of live coding is to create and redefine the instrument during play' (Thor Magnusson: **Sonic Writing**, 2019, p. 245). The computer screens of the live coders are often projected on stage, so the aesthetics of the code they write become an integral part of live coding performances. There are dozens of different computer languages for creating music or images by live coding. Among the most famous are SuperCollider, Pure Data, TidalCycles, fluxus, Sonic Pi and hydra. More are still being created, and some artists are creating their own live coding environments. At this point, we should mention TOPLAP, an organization that unites the world of live coding, and organizes events.

Probably the most common manifestation of live coding is the creation of dance music at parties, which is referred to as Algorave. Artists create new original music on their laptops right on stage, and VJs experiment with generated visuals. Algoraves leave out of the standard cycle of so-called producers who produce tracks, because the tracks are created in real time.

Of course, among the live coding community there are also artists who with the help of live coding experiment and mix styles. Their concerts are often an eclectic mix of the known and the unknown (examples are Atsushi Tadokoro, Reinick Bell, Akihiro Kubota, Fero Király, and Alex McLean).

An advantage of live coding is that almost all languages used for live coding are free software, or open source, which in practice means they are available for free. The open source philosophy is one of the best things that has happened in the computer world. You become a truly free computer user when you use free software, starting with the operating system. Do you know Linux? For artists there is, for example, Ubuntu Studio. You don't need to buy an expensive computer for live coding, a second-hand laptop is all you need.

Good tips for beginners

In my experience, it is advantageous to use the US keyboard layout when programming because specific characters that are often used are easily available on it: * ~ # $ % < > { } [ ] ( ) _ ;.

Write with ten fingers

In live coding, the fluency of typing on the computer keyboard is a decisive factor. Typing with all ten fingers not only makes input more efficient, but is also good for the hands, because the load from typing is evenly distributed on all fingers, which can prevent unpleasant hand diseases (for instance carpal tunnel syndrome). I'm not saying that you are at risk of this when live coding, but movement stereotypes are not good for our body. Therefore, if you find yourself writing a lot with stiff hands, it will be a good time to think about whether to solve that. There are many tutorials and sites to learn

typing with ten fingers. I first used the keyboard program klavaro, where I learned to type, and then I trained my speed using Monkeytype.com, and keybr.com.

Something from history

You probably all use the QWERTY or QWERTZ keyboard layout. This invention dates back to the 19th century, and the layout was chosen for typing speed, and prevented the arms of the mechanical typewriter to not jam against each other. Despite the fact that we no longer use mechanical typewriters, modern keyboards, where letters and characters could be laid out more efficiently, according to what is most often used, still use the QWERTY layout. The cure for this has been invented, and there are several modern layouts such as Colemak, workman, and dvorak that are more efficient and save hand movements. You just have to force yourself to start using them.

How to lead a group of students learning live coding

The following points summarize some of my observations acquired by teaching the module **Music in the Digital Space – How to Live Code**.

1. The base of the module is to learn the students to understand the principles of programming, how the syntax of a language and individual functions work, and to encourage them to independently search for information on the internet.
2. The next step is to turn this knowledge into experience through live playing and improvisation. During the course, the students and myself invented ways of playing together by using game rules and procedures, which helped us a lot to improve.
3. Public performances are an important milestone for every student who invests more energy than usual in the preparation.
4. No less important is breaking away from mainstream musical aesthetics, finding joy in improvisation, and exploring for new musical and sound territories.
5. When new participants enter in the course of the workshop, it is necessary to diversify the learning process. It presents a good opportunity to offer more experienced students the role of a teacher, which encourages them to verbalize and sort what they have learned from a different point of view.

Below are some game rules one can use to structure playing together.

- Area limitation. At Estuary, you only use one playing slot. Students take turns, with each student making only one change per round. This is a very good game for teaching each other, as students have plenty of time to watch the changes made by their classmates.
- Limiting music style. We define in advance what kind of music the students are going to create (slow, fast, techno, fake, very variable, static, ambient, without beats, noise, drones, et cetera).
- Limiting sounds. This game teaches to focus in more detail on sound quality and work with it. You determine in advance the set of samples that will be used. For instance: 1. everyone chooses their own set of sounds, 2. they all use the same set of sounds.

Estuary

This chapter contains basic information about the Estuary programming environment and the languages used in the module. More information in English can be found on the website of Estuary. The text comes from the Estuary Reference Guide.

Estuary is a platform for collaboration and learning through live coding. It enables you to experiment with sound, music, and visuals in a web browser. Estuary brings together a curated collection of live coding languages in a single environment, without the requirement to install software (other than a web browser), and with support for networked ensembles (whether in the same room or distributed around the world). Estuary is free and open source software, released under the terms of the GNU Public License (version 3).

Some of the livecoding languages available within Estuary are:

- TidalCycles: for making patterns of musical events (created/maintained by Alex McLean)
- Punctual: for synthesizing audio and/or video from the same notation (created/maintained by David Ogborn)
- CineCer0: for videos and typography (created/maintained by the Estuary development team)
- Hydra: for video synthesis (created/maintained by Olivia Jack)

Some additional features of Estuary are:

- interfaces for collaboration and communication in networked ensembles
- built-in tutorials and reference materials
- text localization to an expanding set of natural languages
- visual customization via themes (described by CSS)

MiniTidal

A live coding language that allows you to make musical patterns with text, describing sequences and ways of transforming and combining them, exploring complex interactions between simple parts.

First Steps

This tutorial will cover some of the basics of making music with MiniTidal. MiniTidal is a subset of TidalCycles that supports most typical TidalCycles operations (but not all), and everything shown here (and anything that works with MiniTidal) is easy to transfer to the standalone TidalCycles. Lets make some sound!
Copy the code example below to the Estuary and then click Eval to “evaluate” it.

```
s “bd cp”
```

In the example above, we've specified a pattern of samples (specifically 'bd' and 'cp') that fill up a 'cycle' (which can be thought of sort of like musical bars if you like). Everything that appears within the quotes ("") divides a cycle into equal parts: the "bd" sample gets the first half of the cycle, and the "cp" gets the second half. If we put a third element into our pattern we get a different rhythm - each sample gets 1/3rd of a cycle:

```
s "bd cp sn"
```

A "~" placed in a Tidal pattern has a special designation as a 'rest' (silence). So we can get rid of the 'sn' in the above example but preserve the rhythm:

```
s "bd cp ~"
```

You can try to make your own patterns by adding more names of samples (or more ~ for silences) between the quotation marks.

Patterns within Patterns

Sometimes we want to subdivide a part of a cycle - for instance if we want 2 samples to play in the last half of a cycle instead of just one:

```
s "bd [cp sn]"
```

Or if we want two samples to play at the same time we can enclose them in square brackets with a comma between them:

```
s "bd [cp,sn]"
```

Both ideas together:

```
s "bd [cp,sn casio]"
```

More Fun Stuff with Tidal

Here are various additional examples to tinker with. You can find more examples of Tidal(Cycles) usage at tidalcycles.org (just note that the examples there all have things like "d1 $" in front of them - in MiniTidal/Estuary you need to leave off that initial d1 $):

```
fast 2 $ s "bd cp"
jux (fast 2) $ s "bd cp"
s "bd? sn? hh? cp?"
every 4 (fast 4) $ s "bd cp"
s "arpy*8" # note "0 2 4 5 7 9 11 12"
s "glitch" # n (irand 8)
stack [s "bd cp",s "arpy*4" # note "[0,4,7]" ]
```

For more comprehensive documentation of TidalCycles visit https://tidalcycles.org/docs/

CineCer0

CineCer0 (pronounced “sin–ay–ser-oh”) is a language for displaying and transforming videos and text in the browser. It can be used, for example, in the performance of live coded cinema, kinetic typography, VJ-ing, etc. Originally inspired by the CineVivo project, and created specifically for the Estuary platform during the SSHRC-funded research project “Platforms and practices for networked, language- neutral live coding”. CineCer0 features an economical Haskell-like notation and a strongly declarative syntax.

Examples:

```
circleMask 0.5 $ vol 0.5 $ video “videos/cootes/branches.mov”
setSize 0.8 $ image “images/hogweed.mov”
rgb 1 0 1 $ size 6 $ setPosY (-0.6) $ text “This is a text”
```

Play Video

Evaluate the word “video” + the URL of a video inside quotation marks “”.
The URL for videos must point directly to a video file that the web browser can play (not a URL of a video streaming service). Often, such URLs would end in “.mov”, “.mp4” or “.m4v”.
The following examples are using the two different methods to import/play videos. Later on, we will modify the appearance (position, size, blur, brightness, etc). To erase the video: erase everything and evaluate the empty code-box. To change the video: replace the URL and evaluate again.

```
video “https://upload.wikimedia.org/wikipedia/commons/transcoded/9/9e/
Carpenter_bee_on_Abelia_flowers.webm/Carpenter_bee_on_Abelia_flowers.
webm.480p.webm”
video “https://cdn.videvo.net/videvo_files/video/free/2020-05/small_
watermarked/3d_ocean_1590675653_preview.webm”
```

Play Image

Evaluate the word “image” + the URL of a video inside quotation marks “”.
The URL for images must point directly to an image file that the web browser can play. Often, such URLs would end in “.jpg”, “.jpeg” or “.png”.
The following examples are using the two different methods to import/play images. Later on, we will modify the appearance (position, size, blur, brightness, etc). To erase the image: erase everything and evaluate the empty code-box. To change the video: replace the URL and evaluate again.

```
image “https://upload.wikimedia.org/wikipedia/commons/thumb/3/38/
Edinburgh_Union_Canal_SR.jpg/1024px-Edinburgh_Union_Canal_SR.jpg”
image “https://upload.wikimedia.org/wikipedia/commons/f/fe/Royal_emblem_
of_Paramara.jpg”
```

Visualize Text

Evaluate the word “text” + the text you want to visualize inside quotation marks “”. The string of text you evaluate can be as short or long as you want. Later on, we will check how to modify the appearance (position, font size, colour, font type, etc). To erase the text: erase everything and evaluate the empty code-box. To change the text: replace it and evaluate again.

```
text “Hello word”
text “This is my text - Este es mi texto”
```

Change position (axis X,Y) of video, image and text

The anchor point for both video and text is at the centre of these objects. By default, both coordinates (x,y) are in position “0,0” = the centre of the screen.

```
video “https://github.com/jac307/videoTextures/blob/master/mariposa/20.mov?raw=true”
text “This is my text. Este es mi texto. Ceci est mon texte. Este é o meu texto. Das ist mein Text”
```

Functions to change position: “setPosX #”, “setPosY #” and “setCoord # #” --to modify both values. Values go from “(-1)” (top/left-corner) to “1” (bottom/right-corner). Negative numbers must be inside “()”.

```
setPosX 0.5 $ image “https://upload.wikimedia.org/wikipedia/commons/thumb/a/a5/Red-flowers-at-cerro-pelon.jpg/800px-Red-flowers-at-cerro-pelon.jpg”
setCoord 0.2 (-0.5) $ text “This is my text. Este es mi texto. Ceci est mon texte. Este é o meu texto. Das ist mein Text”
```

Video/Image: modify size

The default values for video/image-size are: “width=1” and “height=1”. These plays the video/image at its natural aspect-ratio, with the video-height fitting the screen-height.

```
video “https://upload.wikimedia.org/wikipedia/commons/9/9a/Time_Lapse_of_New_York_City.ogv”
```

Functions to change size: “setWidth #”, “setHeight #”, and “setSize #” --requires just one value (affects proportionally width and height). Values bellow “1” make the video smaller (can down up to “0”). Values above “1” makes the video bigger.

```
setHeight 4.1 $ video “https://upload.wikimedia.org/wikipedia/commons/9/9a/Time_Lapse_of_New_York_City.ogv”
setSize 0.8 $ image “https://upload.wikimedia.org/wikipedia/commons/thumb/b/ba/Tricyrtis_hirta_-_blossom_side_%28aka%29.jpg/640px-Tricyrtis_hirta_-_blossom_side_%28aka%29.jpg”
```

Text: modify font-size

By default, the value of font-size is “1”. This is equivalent to the normal size of the editor’s text in a session of Estuary.

```
text “My text. Mi texto. Mon texte. Meu texto. Mein Text”
```

Function to change font size: “size 3 #”. Values bellow “1” make the text smaller (can down up to “0”). Values above “1” makes the text bigger.

```
size 2 $ text “My text. Mi texto. Mon texte. Meu texto. Mein Text”
size 4 $ text “My text. Mi texto. Mon texte. Meu texto. Mein Text”
```

Hydra

Hydra is a live coding platform for visual synthesis based on analog video synthesizers. Hydra is an open-source project developed by Olivia Jack. You can access a standalone version of Hydra (separate from Estuary) as well as further documentation and examples here: https://hydra.ojack.xyz/.

Examples:

```
shape([3,4,5,6].smooth()).repeat(20).kaleid().modulateScale(o0,0.2,0.5).
out()
osc(10, -5, [0.4,0.9].smooth()).modulateRotate(o0, [1.5,2.5,3.5].fast(3).
smooth()).out(o0)
```

More comprehensive documentation for this course is available at https://www.ferokiraly.com/estuary/en/

Fero Király