Algorithmic composition using patterns

Notam, january 2020

About me

- Name: Mads Kjeldgaard
- Occupation: Composer and developer
- Work: The Norwegian Center for Technology and Art (Notam)

Notam

- Development for art projects (hardware, software, tech and artistic guidance)
- Communities / meetups (SC meetup among others), see website notam.no
- Studios / 3D sound / VR / Visuals / Electronics
- Courses

My practice

- Computer music / livecoding
- Concrete music
- Cybernetic / systemic music

Contact info

- mail: mail@madskjeldgaard.dk
- web: madskjeldgaard.dk
- github: github.com/madskjeldgaard
- work: notam.no

About algorithmic composition

What is an algorithm?

An algorithm is a process that takes something as an input, computes on it, and then outputs the result.

"A recipe is a good example of an algorithm because says what must be done, step by step. It takes inputs (ingredients) and produces an output (the completed dish)." - from Wikipedia

In music, we can crudely think of the input as *parameters* and the output as *sound*

When composing with algorithms ...

... We define the *conditions* for a composition, rather than the specificities of a composition

Algorithmic time

When writing music using algorithms, you are forced to reconsider compositional time in your work

Algorithmic time: Nonlinearity

The most immediate consequence is an escape* from the linear timeline we experience in a DAW

* You can never escape time

Algorithmic time: On the verge

"algorithms are on the verge of time, in so far as they are on the verge between constancy and change, on the one hand, and between concrete and abstract temporality, on the other." - Julian Rohrhuber, Algorithmic music and the Philosophy of Time

Algorithmic time: SuperCollider and time

SuperCollider and Time

(Ircam) - A

nice technical introduction to SuperCollider's idea of time by the

creator of SuperCollider

Design

Short history of SuperCollider

SC was designed by James McCartney as closed source proprietary software

Version 1 came out in 1996 based on a Max
object
called Pyrite. Cost 250$+shipping and could only run on PowerMacs.

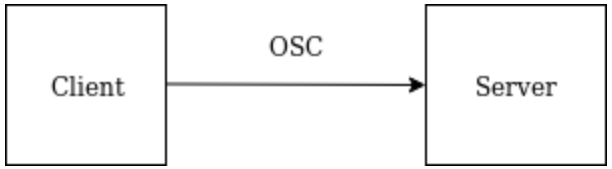
Became free open source software in 2002 and is now cross platform.

Overview

When you download SuperCollider, you get an application that consists of 3 separate programs:

1. The IDE, a smart text editor
2. The SuperCollider language / client (**sclang**)
3. The SuperCollider sound server (**scsynth**)

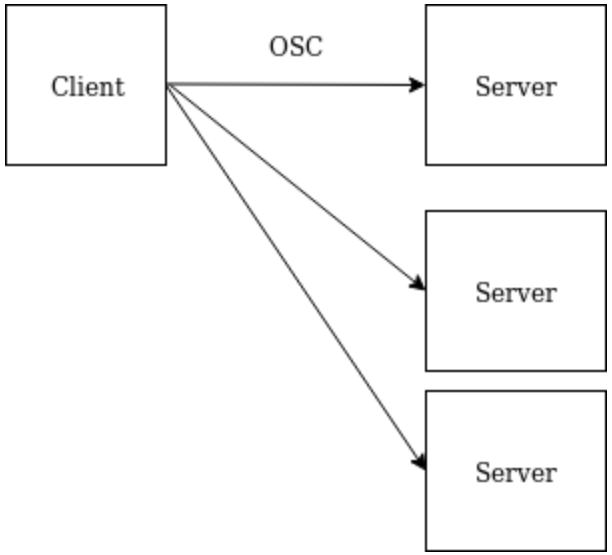
Architecture



The client (language and interpreter) communicates with the server (signal processing)

This happens over the network using Open Sound Control

Multiple servers



This modular / networked design means one client can control many servers

Consequences of this modular design

Each of SuperCollider's components are replacable

IDE <---> Atom, Vim, or Visual Studio

language <---> Python, CLisp, Javascript

server <---> Max/MSP, Ableton Live, Reaper

Extending SuperCollider

The functionality of SuperCollider can be extended using external packages

These are called Quarks and can be installed using SuperCollider itself

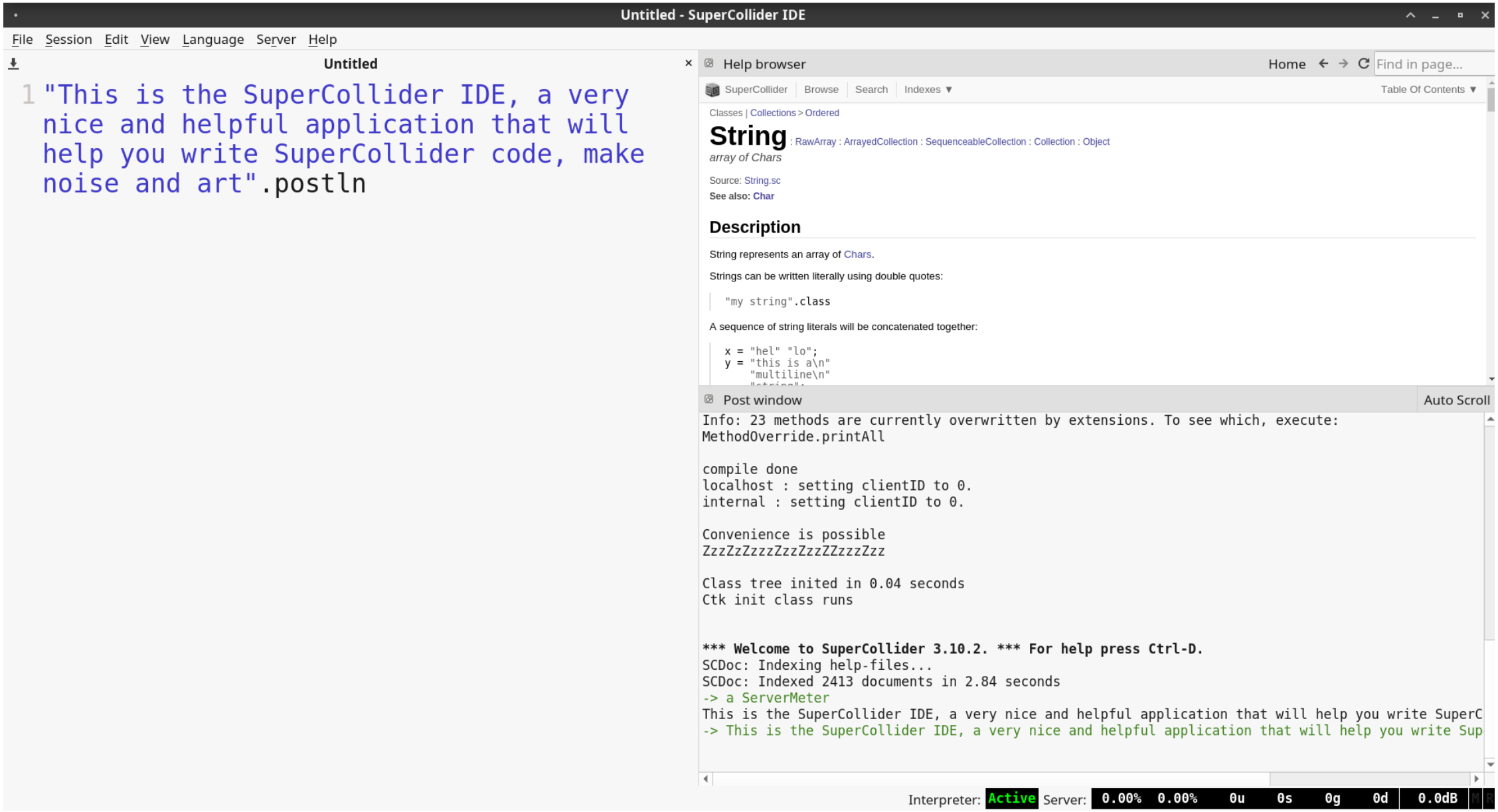
```
// Install packages via GUI (does not contain all packages)
Quarks.gui;

// Install package outside of gui using URL
Quarks.install("https://github.com/madskjeldgaard/KModules");
```

SC Plugins

SC3 Plugins is a
collection of user contributed code, mostly for making sound

The plugins are quite essential (and of varying quality / maintenance)

IDE


Untitled - SuperCollider IDE
File Session Edit View Language Server Help
Untitled
1 "This is the SuperCollider IDE, a very nice and helpful application that will help you write SuperCollider code, make noise and art".postln
Help browser
Home
Find in page...
SuperCollider Browse Search Indexes ▼
Table Of Contents ▼
Classes | Collections > Ordered
String : RawArray : ArrayedCollection : SequenceableCollection : Collection : Object
array of Chars
Source: String.sc
See also: Char
Description
String represents an array of Chars.
Strings can be written literally using double quotes:
"my string".class
A sequence of string literals will be concatenated together:
x = "hel" "lo";
y = "this is a\n"
"multiline\n"
Post window
Auto Scroll
Info: 23 methods are currently overwritten by extensions. To see which, execute:
MethodOverride.printAll
compile done
localhost : setting clientID to 0.
internal : setting clientID to 0.
Convenience is possible
ZzzZzZzzzZzzZzzZZzzzZzz
Class tree inited in 0.04 seconds
Ctk init class runs
*** Welcome to SuperCollider 3.10.2. *** For help press Ctrl-D.
SCDoc: Indexing help-files...
SCDoc: Indexed 2413 documents in 2.84 seconds
-> a ServerMeter
This is the SuperCollider IDE, a very nice and helpful application that will help you write SuperC
-> This is the SuperCollider IDE, a very nice and helpful application that will help you write Sup
Interpreter: Active Server: 0.00% 0.00% 0u 0s 0g 0d 0.0dB

Important keyboard shortcuts

- Open help file for thing under cursor: **Ctrl/cmd + d**
- Evaluate code block: **Ctrl/cmd + enter**
- Stop all running code: **Ctrl/cmd + .**
- Start audio server: **Ctrl/cmd + b**
- Recompile: **Ctrl/cmd + shift + l**
- Clear post window: **Ctrl/cmd + shift + p**

The IDE as a calculator

SuperCollider is an interpreted language

This means we can "live code" it without waiting for it to compile

A good example of this is using it as a calculator

Autocompletion

Start typing and see a menu pop up with suggestions (and help files)

The status line

Shows information about system usage

Right click to see server options + volume slider

About patterns

From the Pattern help file:

"[The Pattern] classes form a rich and concise score language for music"

In other words:

Patterns are used to sequence and compose music

Abstracting the composition process

the conditions for a composition vs. a fixed composition

It's just data

Easily transpose, stretch and warp the composition

Duration is not an issue

Composing a 4 bar loop is not necessarily any more or less work than a 4 hour one

Guides in the help system

Patterns are pretty well documented in the help system:

- A practical guide
- Understanding Streams, Events and Patterns

Event patterns

Like pressing the key of a piano

What data does that involve?

- Duration of key press
- Pitch of the key
- Sustain (are you holding the foot pedal?)
- etc. etc.

What an Event looks like

```
// See the post window when evaluating these
().play; // Default event
(freq:999).play;
(freq:123, sustain: 8).play;
```

Changing the default synth

The default synth sucks

You can change it by defining a new synth called \default

More info on my website

Introducing the allmighty Pbind

Arguably the most important pattern class in SuperCollider

Pbind data

Pbind simply consists of a list of key/value pairs

Keys correspond to Synth arguments

Most often, keys correspond to a Synth's arguments.

Example: If a SynthDef has the argument cutoff, we can access that argument in a Pbind using \cutoff.

Some keys are special

dur

\dur is used in most SynthDef's to specify the duration of a note/event.

Make sure this key never gets the value 0.

stretch

\stretch is used to stretch or shrink the timing of a Pbind

When does a Pbind end?

If one of the keys of a Pbind are supplied with a fixed length value pattern, the one running out of values first, will make the Pbind end.

Livecoding: Pdef

Livecoding patterns is easy. All you have to do is wrap your event pattern (Pbind) in a Pdef:

```
Pdef('myCoolPattern', Pbind(...)).play;
```

What this means

The Pdef has a name ('myCoolPattern') which is a kind of data slot accessible throughout your system

Everytime you evaluate this code, it overwrites that data slot (maintaining only one copy)

Value patterns

The building blocks of compositions

- List patterns
- Random patterns
- Envelope patterns
- Rests
- Data sharing between event parameters
- Patterns in patterns

List patterns

See all of them here

Pseq: Classic sequencer

```
// Play values 1 then 2 then 3
Pseq([1,2,3]);

// 4 to the floor
Pseq([1,1,1,1]);
```

Testing value patterns: asStream

You will see the .asStream method a lot in the documentation for value patterns.

```
// Pattern
p = Pseq([1,2,3]);

// Convert to stream
p = p.asStream;

// See what values the pattern produces
p.next; // 1, 2, 3, nil
```

Random value patterns: Pwhite and Pbrown

```
// (Pseudo) random values
Pwhite(lo: 0.0, hi: 1.0, length: inf);

// Drunk walk
Pbrown(lo: 0.0, hi: 1.0, step: 0.125, length: inf);
```

Random sequence patterns: Prand and Pxrand

```
// Randomly choose from a list
Prand([1,2,3],inf);

// Randomly choose from a list (no repeating elements)
Pxrand([1,2,3],inf);
```

Probability: Pwrand

Choose items in a list depending on probability

```
// 50/50 chance of either 1 or 10
Pwrand([1, 10], [0.5, 0.5])

// 25% chance of 1, 25% change of 3, 50% chance of 7
Pwrand([1, 3, 7], [0.25, 0.25, 0.5])

// 30% chance of 3, 40% change of 2, 30% chance of 5
Pwrand([4, 2, 5], [0.3, 0.4, 0.3])
```

Envelope pattern: Pseg

```
// Linear envelope from 1 to 5 in 4 beats
Pseg( levels: [1, 5], durs: 4, curves: \linear);

// Exponential envelope from 10 to 10000 in 8 beats
Pseg( levels: [10, 10000], durs: 8, curves: \exp);
```

Rest

Skip/sleep a pattern using Rest. If used in the \dur key of a Pbind, the value in the parenthesis is the sleep time

```
// One beat, two beats, rest 1 beat, 3 beats
Pbind(\dur, Pseq([1,2,Rest(1),3])).play;
```

Pkey: Share data between event keys

Using Pkey we can make an event's parameters interact with eachother

```
// The higher the scale degree
// ... the shorter the sound
Pbind(
    \degree, Pwhite(1,10),
    \dur, 1 / Pkey(\degree)
).play
```

More info about data sharing in patterns:

here

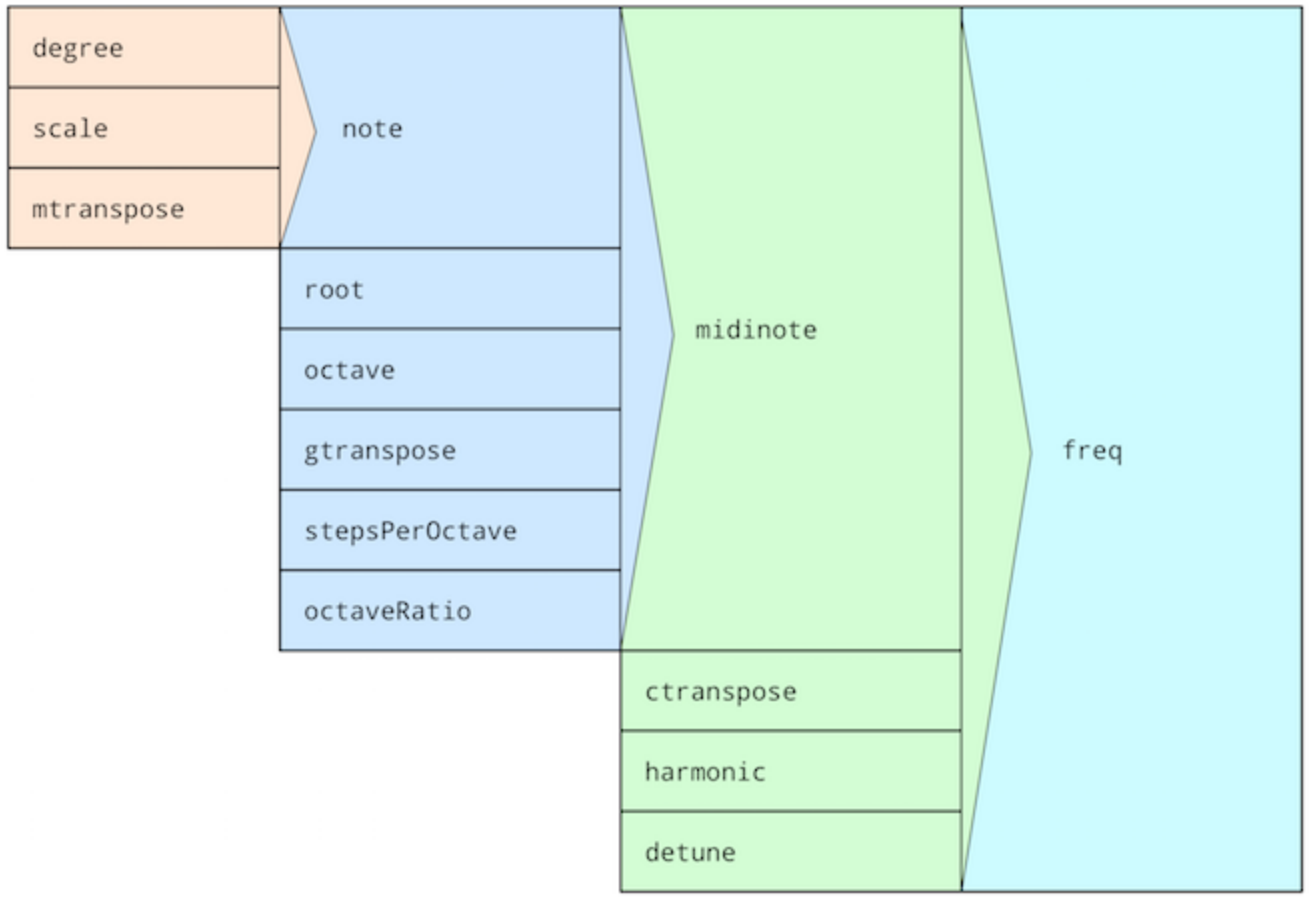
patterns in patterns: The computer music inception

You can put patterns in almost all parts of patterns.

This may lead to interesting results:

```
// A sequence with 3 random values at the end
Pseq([1,2,Pwhite(1,10,3)]);

// An exponential envelope of random length
Pseg(levels: [10, 10000], durs: Pwhite(1,10), curves: \exp);
```

Working with pitches and Pbinds


degree
scale
mtranspose
note
root
octave
gtranspose
stepsPerOctave
octaveRatio
midinote
ctranspose
harmonic
detune
freq

Changing scales

```
// Use the \scale key, pass in a Scale object
Pbind(\scale, Scale.minor, \degree, Pseq((1..10))).play;
Pbind(\scale, Scale.major, \degree, Pseq((1..10))).play;
Pbind(\scale, Scale.bhairav, \degree, Pseq((1..10))).play;
```

Available scales

```
// See all available scales
Scale.directory.postln
```

Changing root note

```
// Use the \root key to transpose root note (halftones)
Pbind(\root, 0, \degree, Pseq((1..10))).play;
Pbind(\root, 1, \degree, Pseq((1..10))).play;
Pbind(\root, 2, \degree, Pseq((1..10))).play;
```

Changing octaves

```
// Use the \octave key
Pbind(\octave, Pseq([2,4,5],inf), \degree, Pseq((1..10))).play;
Pbind(\octave, Pwhite(3,6), \degree, Pseq((1..10))).play;
Pbind(\octave, 7, \degree, Pseq((1..10))).play;
```

Playing chords

```
// Add an array of numbers to the degree parameter
// to play several synths at the same time (as a chord)
Pbind(\degree, [0,2,5] + Pseq([2,4,5],inf), \dur, 0.25).play;
```

Changing tempo

The tempo of patterns are controlled by the TempoClock class You can either create your own TempoClock or modify the default clock like below

```
TempoClock.default.tempo_(0.5) // Half tempo
TempoClock.default.tempo_(0.25) // quarter tempo
TempoClock.default.tempo_(1) // normal tempo
```

Learning resources

Videos

Tutorials by Eli Fieldsteel covering a range of subjects: SuperCollider Tutorials

Books

E-books

- A gentle introduction to SuperCollider
- Thor Magnussons Scoring Sound

Paper books

- Introduction to SuperCollider, Andrea Valle
- The SuperCollider Book

Community

- scsynth.org
- sccode.org
- Slack
- Lurk
- Mailing list
- Telegram
- Telegram ES
- Facebook

Awesome SuperCollider

A curated list of SuperCollider stuff

Find inspiration and (a lot more) more resources here:

Awesome
Supercollider

Learning to code: Advice

- Practice 5 minutes every day
- Set yourself goals: Make (small) projects
- Use the community