Ein kurzer Überblick über Sonic Pi,

einem Programm, das Sam Aaron geschrieben hat, so dass Du damit Musik machen kannst.
http://sonic-pi.net/

Alles getestet mit Sonic Pi 2.11. Geschrieben von Falko für Kids4it: https://www.kids4it.de/

Die ersten Töne

Töne gleichzeitig spielen

```
play 60
play 64
play 67
```

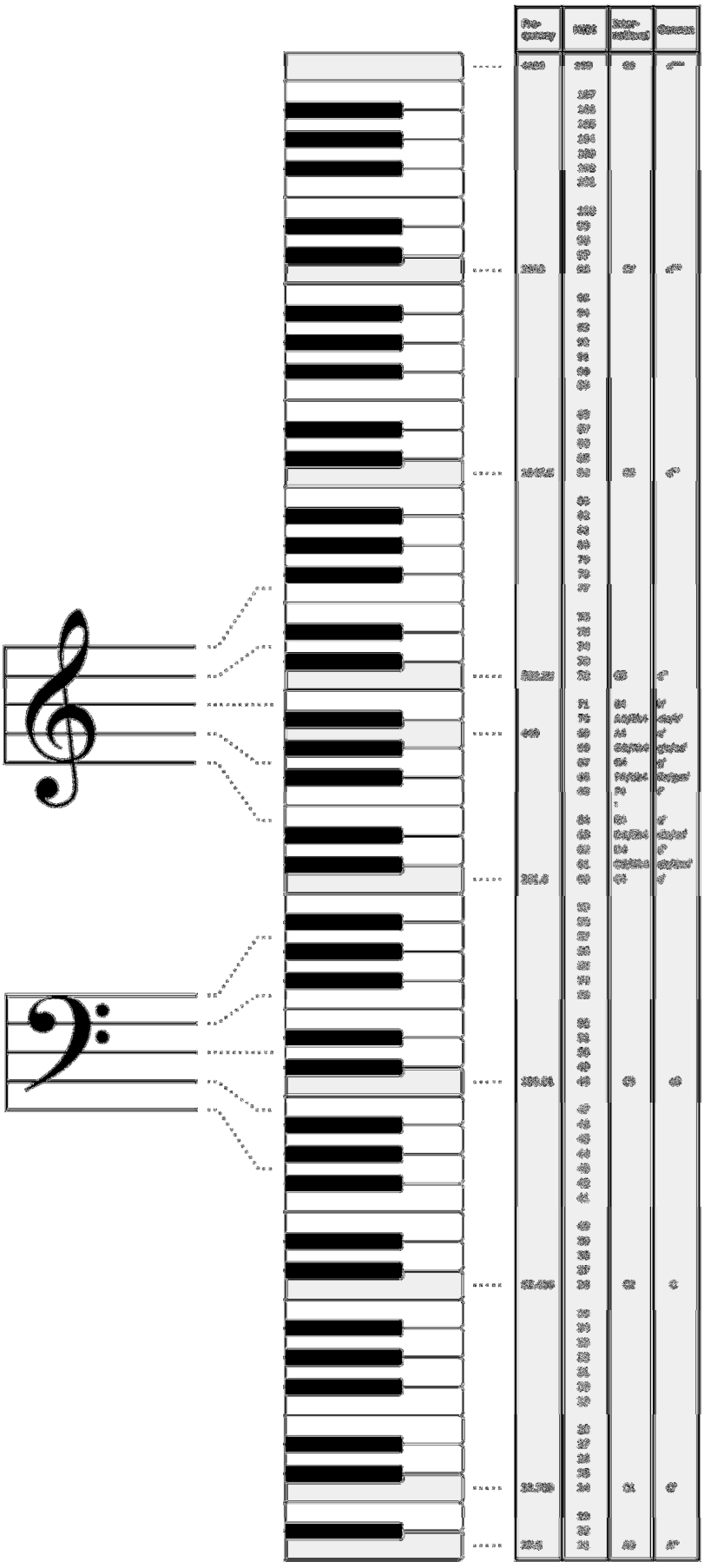
Töne nacheinander spielen

```
play 60
sleep 1
play 64
sleep 1
play 67
```

Statt Zahlen kannst Du auch Noten angeben, immer mit Doppelpunkt davor. Eine Zahl danach gibt die Oktave an. (Englische Schreibweise, also kein H sondern B.) Ohne Zahl ist die 4. Oktave gemeint.

```
play :c
play :c4
play :db3 # Des: Das b steht für „einen Halbton niedriger“.
play :ds3 # Dis: Das s steht für „einen Halbton höher“
play :r       # Eine Pause, manchmal braucht man das.
```

Zur Übersicht über die Noten hilft Dir vielleicht das Bild auf der nächsten Seite.
(Es stammt aus der Wikipedia,
https://commons.wikimedia.org/wiki/File:NoteNamesFrequenciesAndMidiNumbers_V3.svg)


Frequency
MIDI
Scientific notation
German

Kommentare

Manchmal möchtest Du vielleicht einfach etwas in das Programm hinein schreiben, als Notiz für Dich selbst. Damit das Sonic Pi nicht durcheinander bringt, musst Du ein # davor schreiben. Man nennt das einen Kommentar. Sonic Pi ignoriert dann den Rest der Zeile. Kommentare sind auch hilfreich, wenn irgend etwas nicht funktioniert, Du aber nicht weißt, was. Dann kannst Du alle verdächtigen Zeilen mit einem # versehen, solange bis es läuft. Danach kannst Du dann Zeile für Zeile wieder freigeben, indem Du das # weg nimmst, und hoffentlich irgendwann bemerkst, welche Zeile das Problem erzeugt.

```
play 60
# play 70
```

Die zweite Zeile wird ignoriert.

Schleifen

Dem Computer wird nie langweilig, er wiederholt Deine Musik, so oft Du möchtest.

```
10.times do
  play 60
  sleep 1
end
```

Oder so lange, bis Du **Stop** drückst:

```
loop do
  play 60
  sleep 1
end
```

Namen, Variablen

Bestimmt möchtest Du nicht immer das gleiche spielen. Probier mal Folgendes:

```
meineNote = 60
13.times do
  play meineNote
  sleep 1
  meineNote = meineNote + 1
end
```

In der vorletzten Zeile wird der Wert von `meineNote` eins größer gemacht. (Namen müssen mit einem Buchstaben anfangen, und dann können Buchstaben und Ziffern kommen, z.B. `n`, `Hallo`, `m1`)

Ausgabe von Wörtern oder Zahlen

Der Befehl `puts` lässt Sonic Pi Wörter oder Zahlen schreiben, und zwar im sogenannte Protokoll, auf der rechten Seite. Wörter musst Du immer in Anführungsstriche setzen. (Einfache oder doppelte, egal.)

```
puts 17
puts "Hallo"
puts 'Hallo'
```

Das ist sehr hilfreich, wenn einmal etwas nicht so funktioniert, wie Du es erwartest. Manchmal will man nämlich bei einem Namen wissen, was er bedeutet. Das kannst Du dann so machen:

```
n = 60
13.times do
  puts n
  play n
  sleep 1
  n = n + 1
end
```

Hier wird jetzt jedes Mal, wenn die Schleife durchlaufen wird, im Protokoll geschrieben, wie groß n ist, so dass Du Dir sicher sein kannst, welche Note gespielt wird.

Andere Töne – Synthesizer

```
play 60
sleep 1
use_synth :piano
play 60  # Dies ist jetzt ein Klavier-Ton.
```

Probiere andere Synths aus, z.B.: `:pulse :prophet :dsaw :fm :tb303`
In der Hilfe sind alle Synthesizer aufgeführt, ihre Beschreibung ist allerdings auf Englisch.

Ganz andere Töne - Samples

Sonic Pi kommt mit einer ganzen Menge von aufgenommenen Geräuschen, zum Beispiel dem Ruf einer Krähe. Die Namen der Geräusche beginnen immer mit einem Doppelpunkt.

```
sample :misc_crow
```

Diese Geräusche, sogenannte Samples, sind in der Hilfe aufgeführt, aber ich empfehle, einfach die Autovervollständigung zu benutzen und alles aus zu probieren. Gib sample ein, dann den Doppelpunkt, und Du bekommst eine Liste (Schlagzeug beginnt mit `:drum`.. oder `:bd`..).
Cool ist, dass man die Geschwindigkeit ändern kann, mit der die Geräusche abgespielt werden. Sie klingen dann ziemlich anders. Hier unsere Krähe mit halbem Tempo (das klingt schon bedrohlicher):

```
sample :misc_crow, rate: 0.5
```

Ein paar der Samples sind etwas Besonderes, nämlich Schlagzeugschleifen. Das musst Du Dir anhören!

```
loop do
  sample :loop_tabla
  sleep sample_duration :loop_tabla
end
```

Die Trommeln spielen genau so, dass sie am Ende wieder neu anfangen. Sehr hilfreich ist dabei sample_duration – damit gibt der sleep-Befehl dem Sample genau so viel Zeit wie es braucht.

Tickende Ringe

Manchmal hat man mehrere Töne, die man nacheinander spielen will. Dafür, und noch für viel mehr, sind Ringe gut. Ringe sind so etwas wie Behälter, in die man etwas hineintun kann.
Sie heißen Ringe, weil nach dem letzten Ding gewissermaßen wieder das erste kommt. Das ist ein bisschen wie bei einer Uhr, da kommt auch nach 12 wieder die 1.

(Das Bild kommt wieder aus der Wikipedia, https://de.wikipedia.org/wiki/Datei:Dresden-Schlachthof-Uhr.jpg)

Solch einen Ring könntest Du so machen:

```
(ring 1,2,3,4,5,6,7,8,9,10,11,12)
```

Weil Uhren ticken, gibt es nun den Befehl tick, mit dem man immer das nächste Ding aus dem Ring bekommt. Die Schreibweise mit dem Punkt ist neu, aber Du gewöhnst Dich bestimmt schnell daran.
Diese Schleife schreibt unaufhörlich die Zahlen von 1 bis 12 ins Protokoll. (Also so lange, bis Du Stop drückst.)

```
loop do
  puts (ring 1,2,3,4,5,6,7,8,9,10,11,12).tick
  sleep 0.5
end
```

Die Pause, also den sleep-Befehl, habe ich eingefügt, damit Du mitlesen kannst. Wenn Du die Pause zu kurz machst, dann funktioniert die Anzeige nicht mehr richtig. 0.01 geht bei auf meinem Computer noch, bei 0.001 kommt Sonic Pi nach einer Weile durcheinander.
Wenn wir damit Musik machen wollen, können wir also zum Beispiel eine Tonleiter in einen Ring tun:

```
loop do
  play (ring :c4, :d4, :e4, :f4, :g4, :a4, :b4, :c5).tick
  sleep 0.5
end
```

Oder, dasselbe mit Zahlen geschrieben:

```
loop do
  play (ring 60, 62, 64, 65, 67, 69, 71, 72).tick
  sleep 0.5
end
```

(Wie Du merkst, müssen in einem Ring nicht 12 Dinge drin sein, es ist egal wie viele. Wichtig ist, dass jedes Tick einen Schritt weiter geht, und am Ende wieder von vorn anfängt.)
Ein wichtige Sache: tick geht in ALLEN Ringen gleichzeitig einen Schritt weiter, look schaut nur in die Ringe hinein. Das musst Du beachten, wenn Du mehrere Ringe benutzt. Hier zum Beispiel habe ich zwei Ringe. Der erste Ring enthält die Töne, der zweite die Zeiten. Dabei gehören der erste Ton und die erste Zeit zusammen. So funktioniert es richtig:

```
loop do
  play (ring 60, 65, 67, 72, 79).tick
  sleep (ring 1, 0.5, 1, 0.5).look
end
```

Dabei geht das erste tick auf die erste Note, also 60, und das look in der nächsten Zeile auf die erste Zeit, also 1. Beim nächsten tick kommt die 65 und beim look die dazugehörige Zeit 0.5.

FALSCH
Wenn Du aus Versehen auch in der zweiten Zeile tick benutzt, bekommst Du zur 60 schon die 0.5 und das nächste tick gibt Dir dann schon die 67. Es wird also immer etwas übersprungen. Das wollen wir ja nicht.

```
loop do
  play (ring 60, 65, 67, 72).tick
  sleep (ring 1, 0.5, 1, 0.5).look
end
```

Ringe kann man sehr oft verwenden. Je besser Du sie kennst, desto öfter wird Dir auffallen, dass sie sich irgendwo anwenden lassen. Deshalb habe ich hier einiges gesammelt, was man mit ihnen machen kann. Vieles davon wird, wie tick und look, mit einem Punkt hinter dem Ring geschrieben. Dabei kann man mehreres hintereinander schreiben (wie im Beispiel zum Rückwärts-Spielen, ein paar Zeilen weiter unten).
Wie mache ich mir einen Ring?

```
(ring :c4, :d4, :e4, :f4, :g4, :a4, :b4, :c5)
```

Ich möchte außerdem `scala`, `chord` und `spread` empfehlen, die werden später noch etwas ausführlicher behandelt.

Du kannst einen Ring rückwärts gehen lassen:

```
(ring :c4, :d4, :e4, :f4, :g4, :a4, :b4, :c5).reverse
```

Den benutzt Du dann so

```
loop do
  play (ring :c4, :d4, :e4, :f4, :g4, :a4, :b4, :c5).reverse.tick
  sleep 0.5
end
```

Du kannst auch einem Ring einen Namen geben, und ihn so zu einer Variablen Machen:

```
meinRing = (ring :c4, :d4, :e4, :f4, :g4, :a4, :b4, :c5)
```

Den Namen benutzt Du dann wie den Ring selbst, z.B. `meinRing.reverse.tick`
Die Länge eines Rings ist die Zahl der Dinge, die drin sind: `meinRing.length`
Durchmischen: `meinRing.shuffle`
Wähle ein zufälliges Ding aus dem Ring: `meinRing.choose`
Mehr findest Du in Kapitel 8.4 und 8.5 in der Hilfe.

Zufall

Wusstest Du, dass man zur Musik auch Würfel benutzen kann? Du hast ja gesehen, dass für die Noten in Sonic Pi Zahlen stehen. Die können wir auch erwürfeln. Oder wir lassen das den Sonic Pi machen.
„Erwürfelt“ Zahlen von 1 bis 6, wie ein Würfel:

```
rrand_i(1,6)
```

Probier mal dies (nicht alle Noten wirst Du hören können):

```
loop do
  play rrand_i(1,127)
  sleep 0.3
end
```

`rrand_i` gibt immer ganze Zahlen, wenn Du auch Komma-Zahlen haben möchtest, nimm `rrand`.
`rand` erzeugt eine Zahl zwischen 0 und 1.
Auch der Ring-Befehl `choose` benutzt Zufall (siehe oben).

Töne verändern

Was ist ein Ton, und was kann ich damit machen?
Sonic Pi kann Töne auf verschiedene Weise verändern. Zum Beispiel kannst Du mit release: angeben, wie lang der Ton gespielt werden soll.

```
play 65, release: 0.5
play 60, release: 10
```

Da geht noch viel mehr. Alles über Zeit-Einstellungen von Tönen steht im Tutorial in Kapitel 2.4 „Dauer und Hüllkurven“.

Außerdem kannst Du, wenn Du einen Stereo-Kopfhörer oder Stereo-Lautsprecher hast, mit pan: angeben, von welcher Seite der Ton kommen soll. -1 ist links, 1 ist rechts, 0 ist beide. Hier wechseln bei jedem Ton die Lautsprecher:

```
loop do
  play rrand_i(30,80), release: 0.2, pan: -1
  sleep 0.3
  play rrand_i(30,80), release: 0.2, pan: 1
  sleep 0.3
end
```

Effekte

Vielleicht möchtest Du die Töne noch etwas beeindruckender machen, dazu gibt es Effekte. Effekte können ziemlich kompliziert sein, wir probieren hier deshalb ein paar einfache aus. Effekte werden immer mit do und end um das herum geschrieben, worauf sie wirken sollen. Manche Effekte brauchen noch ein paar zusätzliche Werte.
Hall, wie in einem großen Raum. Vergleiche einmal:

```
loop do
  sample :loop_mika
  sleep sample_duration :loop_mika
end
```

mit

```
with_fx :reverb, room: 0.99 do
  loop do
    sample :loop_mika
    sleep sample_duration :loop_mika
  end
end
```

Room darf zwischen 0 und 1 sein, und beeinflusst die Größe des Raums.

Wackel-Effekt, :flanger. Ein schönes Spielzeug.

```
with_fx :flanger, mix: 0.8, phase: 0.1  do
  loop do
    play 50, amp: 0.5
    sleep 1
  end
end
```

Noch ein Wackeln – wobble.

```
with_fx :wobble, phase: 6 do
  live_loop :wob do
    sample :drum_heavy_kick
    sleep 0.5
  end
end
```

Das kling wie in Scheiben geschnitten

```
with_fx :slicer do
```

```
  12.times do
    play 60, sustain: 1.5
    sleep 2
  end
end
```

Bedingungen, Tests

Wenn Sonic Pi eine Klimaanlage wäre, kannst Du mit dem Befehl if sagen: Wenn es kühler ist als 16° Celsius, stelle die Heizung an. Oder wenn es ein Roboter wäre: Wenn alles in Ordnung ist, lasse die grüne Lampe leuchten, ansonsten die rote. Die Voraussetzung, also „wenn es kühler ist als 16° Celsius“ oder „wenn alles in Ordnung ist“, nennt man auch Bedingung. Bedingungen sind entweder wahr oder falsch, deswegen weiß if immer, was es tun soll. Wenn eine Bedingung wahr ist, sagt man auch, sie ist „erfüllt“.
Im nächsten Beispiel rand > 0.2, schaut Sonic Pi also nach, ob eine frisch gewürfelte Zufallszahl (zwischen 0 und 1) größer ist als 0.2. Meist wird das so sein. Dann macht if das, was als nächstes kommt. Wenn nicht, dann guckt es, ob else kommt, und macht das, was dann kommt – bis zum nächsten end. Die Seite mit dem else kannst Du auch weglassen.

```
loop do
  if rand > 0.2
    sample :bd_fat
  else
    sample :misc_crow
  end
  sleep 1
end
```

Die folgende Methode, eine Bedingung anzugeben, unter der ein Sample gespielt werden soll, ist schön kurz, deswegen erwähne ich sie extra. Du kannst nämlich statt

```
loop do
  if rand > 0.2
    sample :bd_fat
  end
  sleep 1
end
```

auch schreiben

```
loop do
  sample :bd_fat, on: rand > 0.2
  sleep 1
end
```

Dabei sind also if .. do .. end ersetzt durch, on: in derselben Zeile.

Live Loops

Ein loop, also eine Schleife, hört ja nie auf. Wenn Du folgendes schreibst, wird das Fingerschnipsen :perc_snap niemals ausgeführt.

```
loop do
  sample :bd_fat, on: rand > 0.2
```

```
  sleep 1
end
sample :perc_snap
```

Das ist natürlich nicht schön. Deshalb gibt es dafür mehrere Lösungen, die im Tutorial beschrieben sind. Die beste davon ist der live_loop, und wenn Du den kennst, brauchst Du Dir die andere, den Thread erst mal nicht anzuschaun.

Neu am `live_loop` ist, dass er einen Namen braucht. Es kann nämlich mehrere davon geben, und manchmal müssen sie miteinander „reden“. Den Namen, im ersten Beispiel :loop1, kannst Du Dir frei ausdenken, er muss mit : beginnen, und dann kommt mindestens ein Buchstabe, danach kann es mit einem Mix aus Buchstaben und Zahlen weitergehen.

```
live_loop :loop1 do
  sample :bd_fat, on: rand > 0.2
  sleep 1
end
sample :perc_snap
```

Hier startet der `live_loop`, und sofort auch das, was danach kommt.
Du kannst auch mehrere Live Loops gleichzeitig laufen lassen. Schreib sie einfach nacheinander. Achte darauf, dass sie unterschiedliche Namen haben (sonst wird nur der letzte mit dem selben Namen gestartet).
Manchmal wird Dir auffallen, dass sie nicht richtig miteinander Musik machen. Etwa so, als würden mehrere Leute klatschen, aber nicht gemeinsam im Rhythmus, sondern durcheinander.
Mehrere Live Loops gemeinsam klatschen zu lassen, nennt man Synchronisation. Dazu schaust Du als erstes nach, ob die Live Loops unterschiedlich schnell sind (vergleiche die sleep-Zeiten). Der schnellste Live Loop muss dann den anderen den Takt vorgeben.
Bei den folgenden zwei Live Loops ist der erste, mit dem Namen :x1, der schnellere. Der zweite Live Loop, mit dem Namen :x2 ist erst fertig, wenn :x1 fast viermal gelaufen ist. Der Befehl sync :x1 bringt den zweiten Live Loop nun dazu, so lange mit dem nächsten Durchlauf zu warten, bis :x1 auch einen neuen Durchlauf beginnt. So sind beide schön im Rhythmus.

```
live_loop :x1 do
  sample :perc_snap
  sleep 0.25
end

live_loop :x2 do
  sync :x1
  sample :bd_fat
  sleep 0.9
end
```

Probier aus, die Zeile mit dem sync auszukommentieren, also schreib ein # davor. Dann starte neu, und vergleiche.

Musik

Wir haben bis jetzt allerlei Technik behandelt. Um Dich bei der Musik zu unterstützen, bietet Sonic Pi noch ein viele Helfer, von denen ich ein paar vorstellen möchte.
So kannst Du Akkorde spielen, sie werden mit chord gebaut.

```
play chord(:c3, :minor)
```

Oder mit mehr Tönen:

```
play chord(:c3, :minor, num_octaves: 3)
```

Du kannst die Töne auch nacheinander erklingen lassen:

```
play_pattern chord(:c3, :major, num_octaves: 3)
```

Übrigens heißt :major Dur, und :minor Moll.
Ein Akkord ist übrigens ein Ring. Du kannst Dir also auch selbst Akkorde bauen.

Sehr hilfreich ist auch, dass Du Dir mit dem Befehl scala Tonleitern bauen kannst. Dazu hat Sonic Pi eine ganze Sammlung verschiedener Tonleitern mitbekommen. Eine Tonleiter in Sonic Pi ist ein Ring, Du kannst also die Töne mit tick nacheinander spielen.

```
live_loop :plop do
  play scale(:c3, :major, num_octaves: 2).tick
  sleep 1
end
```

Wenn Du `num_octaves:` weglässt, wird nur eine Oktave gespielt. Es gibt wie gesagt viele Tonleitern. Die bekanntesten sind `:major` (Dur), und `:minor` (Moll). Empfehlenswert sind auch `:minor_pentatonic` und `:major_pentatonic`, aber noch besser ist es, alle einmal auszuprobieren.
Zum Abschluss noch etwas über Rhythmen
In einem Live Loop kannst Du sehr schön Rhythmen erzeugen. Hier zum Beispiel gibt ein Fingerschnipsen (`:per_snap`) den Grundrhytmus an, und eine Trommel klingt einige der Schläge mit.
Gesteuert wird das über den Ring (ring true, true, false, true) – immer wenn true dran ist (Englisch für wahr), schlägt die Trommel, bei false (Englisch für falsch), schlägt sie nicht.

```
live_loop :beat do
  sample :perc_snap, amp: 0.1
  sample :elec_bong, amp: 0.6, on: (ring true, true, false, true).tick
  sleep 0.25
end
```

Vor ein paar Jahren haben Musikforscher eine Methode entwickelt, viele schöne Rhythmen zu erzeugen, und in Sonic Pi gibt es dafür den Befehl spread. Du musst dazu spread zwei Zahlen mitgeben. Die erste sagt, wie oft die Trommel schlagen soll, die zweite, auf wie viele Schläge des Grundrhythmus dies verteilt warden soll.

```
live_loop :beat do
  sample :perc_snap, amp: 0.1
  sample :elec_bong, amp: 0.6, on: (spread 5, 8).tick
  sleep 0.25
end
```

Du kannst die Zahlen, hier 5 und 8, ein bisschen verändern und experimentieren, welche Rhythmen Dir gefallen.