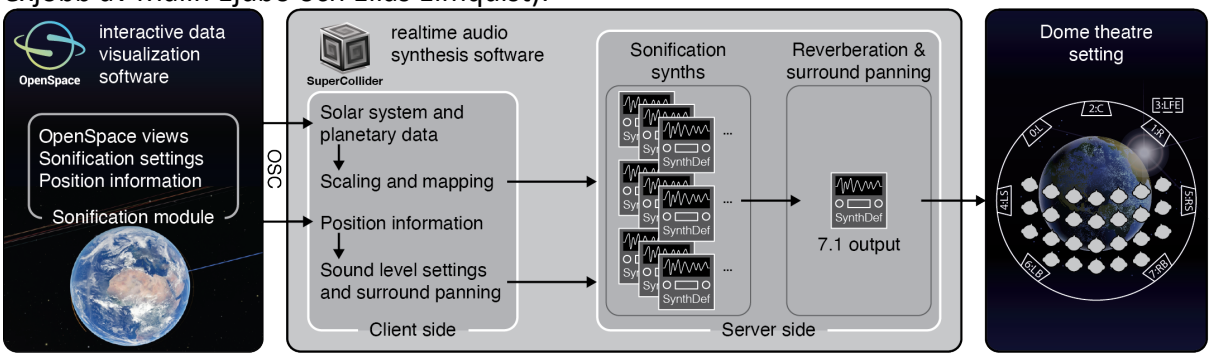
SuperCollider 1 – Ljudsyntes

I denna laboration ska ni testa på ljudprogrammeringsmiljön SuperCollider (SC). SC är en realtidssyntesmiljö med stora möjligheter att koda ljudsyntes och signalbehandling. SC finns för Mac, Linux och Windows och kan laddas ned här:
https://supercollider.github.io/

Förutom att bekanta er med miljön i SC går laborationen ut på att koda och experimentera med olika syntesmetoder.

SC är en kodmiljö som består av två sidor, en serversida som innehåller synthdefinitioner som skapar ljud och en klientsida som exempelvis skickar noter till synthdefinitionerna för att förändra tonhöjden på syntarna. Här följer ett exempel på struktur där SC används för att sonifiera (ljudsätta) OpenSpace, information sänds från OpenSpace via OSC (OpenSound Control) till SC där infon tas emot på klientsidan, skalas och sedan skickas till synthdefinitionerna på servern, och sedan skickar servern ut ljudet i 7.1 surround (från ett exjobb av Malin Ejdbo och Elias Elmquist):



Inför laborationen

Förberedelse – 1. Ladda ned och installera

https://supercollider.github.io/download

Förberedelse – 2. Bekanta er med SC

När SC startas skapas ett nytt tomt ”projekt”. Till höger (oftast) om detta ligger Help browser där det går att söka hjälp om SC, och nedanför detta finns Post window. Nedanför Post window visas om kodtolken är aktiv, och en del info om Servern som CPU-belastning etc.

Det är möjligt att skriva ut saker till Post window, och det görs med `postln` (post line). Genom `postln("Hello world");` så skriver SC ut Hello world i Post window. Kodraden exekveras genom att stå på kodraden med markören och klicka på <Shift> + <Enter>. Ett annat sätt att skriva detta på är `("Hello world").postln;` vilket resulterar i samma output.

En variabel deklareras med `var x;` och flera variabler kan deklareras i samtidigt med `var x, y, z;`. Alla variabler måste deklareras först i en funktion eller i en synthdefinition. Globala variabler deklareras utanför funktionerna och det finns ”superglobala” variabler typ s (server) och w (window), och det finns environment-variabler som deklareras med `~`.

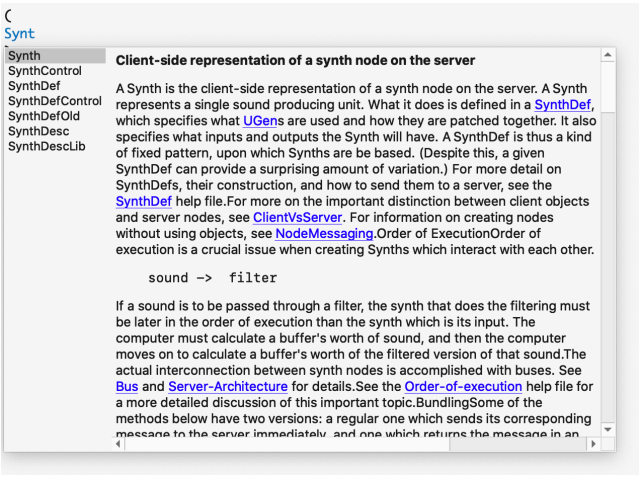
Skapa en variabel som innehåller strängen ”Hello world” och skriv sedan ut variabeln. Eftersom <Shift> + <Enter> exekverar en rad i taget så måste båda raderna markeras för att innehållet i variabeln ska skrivas ut. Därför är det bra att sätta parenteser kring koden, en

startparentes innan koden och sedan en slutparentes efter koden. Då går det att använda <Ctrl> + <Enter> i Windows eller <Cmd> + <Enter> i Mac för att exekvera all kod i parentesen. Med <Shift> + <Enter> körs fortfarande bara den/de rader som är markerade. Med <Ctrl> + <.> i Windows och <Cmd> + <.> stoppas exekveringen.

```
(
var x = ("Hello world");
x.postln;
)
```

Med `//` kommenteras en rad i taget i koden, och med `/*` .. `*/` kommenteras flera rader.

När man skriver kod och vissa keywords dyker det upp rutor så att det är enkelt att välja rätt nyckelord och få tips om användning och fortsättning på koden:



Det händer att kodtolken i SC rasar, under menyn Language kan man både avsluta den och/eller starta om den.

Att skriva ut till Post window görs på klientsidan. Det går inte att skriva ut på samma sätt från serversidan. Än så länge har inte servern startats, detta måste göras manuellt antingen genom att välja Server -> Boot Server eller genom att exekvera `s.boot;`. I menyn Server kan man också starta om servern eller döda alla servrar som är i gång, det går också att stänga av servern genom att exekvera `s.quit;`. När servern bootats visas i Post window de ljuddevices som är tillgängliga i datorn. När kodtolken startas om dödas också alla servrar som är igång.

Med `s.scope;` skapas ett oscilloskop vilket kan vara till hjälp i laborationerna i den här kursen.

Läs mer här:
https://doc.sccode.org/Classes/Server.html

Förberedelse – 3. Mer om SC

Testa på SC och bekanta er med gränssnittet i SC. Följande är några tips för att komma igång mer än vad den korta introduktionen ger i denna text.

Eli Fieeldsteel har en väldigt bra Youtube-kanal med många exempel och tips:
https://youtu.be/yRzsOOiJ_p4

Det finns också ett antal tutorials här:
http://doc.sccode.org/Tutorials/Getting-Started/00-Getting-Started-With-SC.html

Och här finns info om SC, om strukturen och klasser:
http://doc.sccode.org/

Uppgiften

I den här laborationen ska ni skapa en synthdefinition, arbeta med denna och testa olika syntesmetoder.

Skapa en sinusvåg

Börja med att skapa en synthdefinition enligt: `SynthDef.new(name, ugenGraphFunc, rates, prependArgs, variants, metadata)`

Läs mer här:
https://doc.sccode.org/Classes/SynthDef.html

Ni behöver inte skriva `.new` när ni skapar er synthdef. Man ger en synthdef ett namn antingen genom `\name` eller med `"name"`. Efter namnet ska ni skriva själva den ljudalstrande funktionen (ugenGraphFunc). Den funktionen skrivs inom måsvingar `{ .. }`. Normalt avslutar man synthdefinitionen med att säga att den ska läggas till i servern med `.add` så att man sedan från klientsidan ska kalla på den och starta dess ljud, men i detta fall vill vi att synthdefen ska spela direkt genom `.play`. Er kod borde likna detta:

```
SynthDef(\firstSynth, {
    // här är funktionen för att skapa ljud
}).play;
```

Varje synthdef har normalt en utgång som ljudet ska gå ut genom. En sådan skapas med `Out` och den får antingen audio rate eller control rate genom `.ar` eller `.kr`. Detta är ett sätt för SC att spara på beräkningsprestanda genom att ge kontrollsignaler som ofta är långsammare än audiosignaler en lägre samplingsfrekvens. Normalt går ljudet till ljuddevice 0, men det går också att skicka ljudet till olika buffrar i SC. Använd i detta fall audio rate och ljuddevice 0.

Läs mer här:
http://doc.sccode.org/Classes/Out.html

Nu ska vi skapa en sinusoscillator och skicka ljudet till utgången. Det görs med `SinOsc`. Oscillatorn ska vara i audio rate och få en frekvens, exempelvis 220Hz. Er kod bör likna detta:

```
SynthDef(\firstSynth, {
    Out.ar(0, SinOsc.ar(220));
}).play;
```

Läs mer om SinOsc här:
https://doc.sccode.org/Classes/SinOsc.html

I detta fall så skapar funktion en utgångsbuss (device 0) och skickar ut en sinusoscillator i 220Hz till den utgången. Inte så snyggt kodmässigt, det är bättre att lägga oscillatorn i en variabel och sedan skicka variabeln till utgången, för sannolikheten är stor att vi vill bearbeta ljudet från oscillatorn på olika sätt innan den skickas ut.

Sätt synthdefen inom parenteser så att ni enkelt kan exekvera hela koden, och stoppa in koden så att ni får se oscilloskopet också. Tänk på att servern måste vara igång för att ni ska höra något. Ljudet stoppas sedan när exekveringen avslutas.

Ljudet spelas (troligtvis) bara upp i vänster kanal, då ljuddevice 0 nog är i stereo (kanal 0 (vänster) och 1 (höger)) men det skickas bara en ljudkanal till utgången. Detta kan åtgärdas genom att skicka en array till utgången och lägga samma ljudkälla i båda posterna i arrayen. I

det här exemplet har jag stoppat oscillatorn i variabeln output:

```
Out.ar(0, [output, output]);
```

Läs mer om arrayer här:
https://doc.sccode.org/Classes/Array.html

Eller genom följande funktion som säger att output ska användas två gånger:

```
Out.ar(0, {output}!2);
```

Avslutningsvis skulle ni kunna använda `FreqScope` som är en frekvensanalysator för att se övertonsserien i ljudet ni skapar. Den skapas med `FreqScope.new(width, height, busNum, scopeColor, bgColor, server)`, exempelvis:

```
FreqScope(400, 200, 0, server: s);
```

I det exemplet används en bredd på 400 och en höjd på 200, och den ska visa ljudet på busnummer (dvs audio device) 0. Den ska också köras på server `s`. Eftersom de tre första värdena är i den ordning som `FreqScope` definieras i behövs de inte deklareras mer, men sedan hoppar koden till det sista argumentet och då används `server:` för att definiera vilket argument som avses. Det innebär att `scopeColor` och `bgColor` (i detta fall) är körs enligt default inställningen.

Läs mer här:
https://doc.sccode.org/Classes/FreqScope.html

Titta och lyssna på ljudet. Testa olika frekvenser. Hur låter det, hur ser det ut, och varför?

_____

_____

_____

Det går att ändra fas på sinusvågen, vad händer då? Låter det annorlunda?

_____

_____

_____

Det är också möjligt att multiplicera ljudet från oscillatorn med ett annat värde i inputargumenten till `SinOsc`, exempelvis med 0.5 för att halvera ljudvolymen. Och, avslutningsvis kan man addera ett värde till sinusvågen. Detta är ungefär som att justera DC-offseten av ett ljud, man manipulerar alltså värdet kring vilket ljudet svänger. Testa att addera olika värden, vad händer? Och förändras ljudet?

_____

_____

_____

Skapa andra vågformer

En sinuston är inte kanske så intressant att lyssna på, testa därför att skapa andra vågformer. Andra vågformer skapas exempelvis med `LFTri` som skapar en triangelvåg, `LFSaw` som skapar en sågtandsvåg, och `LFPulse`. `LFPulse` är något annorlunda i syntesen än `LFTri` och `LFSaw` då den behöver ett argument för pulsbredden (`width:` mellan 0 och 1, där en ren fyrkantsvåg har en pulsbredd på 0.5).

Läs mer här:
https://doc.sccode.org/Classes/LFTri.html
https://doc.sccode.org/Classes/LFSaw.html
https://doc.sccode.org/Classes/LFPulse.html

Experimentera med ljuden, på vilket sätt skiljer de sig åt? Vad händer med övertonsserien mellan vågformerna? Hur påverkar pulsbredden ljudet?

_____

_____

_____

Vad händer om pulsbredden är 0 respektive 1? Varför?

_____

_____

_____

Skapa flera ljud samtidigt

Gå tillbaka till sinusoscillatorn och skapa en kopia på den koden (men i ett annat variabelnamn). Summera dessa två vågformer i en ny variabel och skicka denna till utgången. Skapa en ny variabel (jag kallade den för detune) för att stämma isär de två tonerna något, ta den ena oscillatorns frekvens +detune och den andres frekvens -detune, och börja med ett värde på 1Hz.

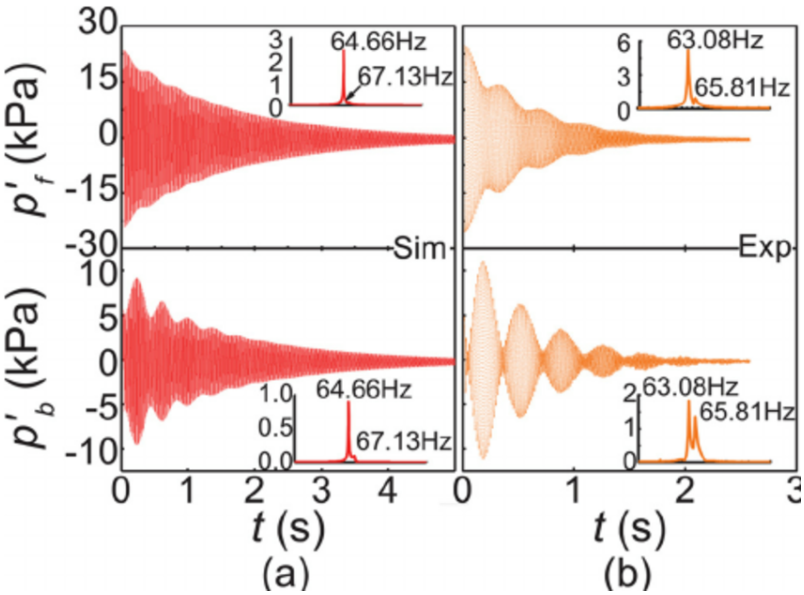
Hur låter det? På vilket sätt förändras ljudet när ni ändrar värdet på detune (testa från 0, 0.1, 1 och uppåt)?

_____

_____

_____

Läs mer om beating och interferens mellan två ljud här:
https://en.wikipedia.org/wiki/Beat_(acoustics)



Genom att halvera frekvensen skapas en ton som är en oktav lägre, genom att multiplicera med två skapas en ton som är en oktav högre, och genom att ta 1.5 skapas (nästan) en kvint. Testa olika värden, och läs mer om musikaliska intervall här:
https://en.wikipedia.org/wiki/Interval_(music)

Den tidigare tonen på oscillator 1 hade 220Hz vilket är ett A. Sätt den nya tonen i 329.63Hz vilket är ett E. Nu kommer två toner, ett kvintintervall, att spela upp.

Läs mer om MIDI-notnummer och tonhöjd och frekvens här:
https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies

Om båda oscillatorerna är i default amplitud så finns risk för att ljudet överstyr, att det blir distorsion. Det skapar nya övertoner, som för all del kan vara trevliga och intressanta men i detta exempel vill vi undvika dem.

Hur låter det? Är det en ren tvåtonsklang eller låter det vasst och fräsigt?

_____

_____

_____

Problemet går att åtgärda antingen med argument `mul:` i `SinOsc`, eller när ljuden summeras (mixas) genom att till exempel dela summan med 2.

Jämför övertonsserien och ljudets kvalitet när det överstyr (distar) och när det inte gör det. Vad händer med ljudet?

_____

_____

_____

Testa att modulera frekvensen

I stället för att summera de två ljuden ska ni ändra utgången till att bara spela upp den andra oscillatorn. Den första oscillatorn ska ändras till en låg frekvens (exempelvis 1Hz) och ska användas för att modulera frekvensen av den andra oscillatorn. Gör detta genom att summera den första oscillatorn med frekvensargumentet för den andra oscillatorn. När oscillatorn ska användas som kontrollkälla kan den ändras till control rate i stället för audio rate. Exempelvis så här:

```
var sin2 = SinOsc.ar(329.63 + sin1);
```

Nu kommer tonhöjden i oscillator 2 att variera något, som ett långsamt vibrato. En oscillator i SC svänger normalt kring 0 mellan -1 och 1. Det innebär att frekvensen på oscillator 2 kommer svänga mellan 328.63 och 330.63.

Hur låter det? Testa att ändra frekvens på oscillator 1, hur blir det då?

_____

_____

_____

Effekten är inte så stor eftersom signalen är så liten från oscillator 1. Men, genom att använda `range` så kan vi förändra detta omfång. Range är en klass som förändrar ett omfång till att vara mellan två specifika värden, exempelvis `range(-10, 10)`.

Läs mer här:
https://doc.sccode.org/Classes/Range.html

Hur låter det nu? Storleken på effekten är ju kopplat till frekvensen på oscillatorn vi lyssnar på och storleken på omfånget, testa att ändra omfånget. Att ändra frekvensen på den första oscillatorn till att gå från lågfrekvent kontrollsignal till hörbar audiosignal kan också skapa intressanta ljud.

_____

_____

_____

Testa att modulera frekvensen med de andra vågformerna också. Vad händer, vad blir skillnaden?

_____

_____

_____

Testa att modulera amplituden

På liknande sätt går det att modulera amplituden. I stället för att summera två frekvenser i den andra oscillatorn ska man göra om signalen från den första oscillatorn till en kontrollsignal som svänger kring 0.5 mellan 0 och 1, där 0 är inget ljud och 1 är max ljud. Där efter multiplicerar man ljudet ut från den andra oscillatorn med den nya kontrollsignalen. Detta skapar då en tremoloeffekt där ljudvolymen fluktuerar.

Sätt frekvensen på den första oscillatorn till låg (1Hz), och ändra dess `range` till att vara mellan 0 och 1. Multiplicera sedan kontrollsignalen med signalen från den andra oscillatorn.

Hur låter det? Vad händer om frekvensen på den första oscillatorn (kontrolloscillatorn i detta fall) ökas?

_____

_____

_____

Läs mer här:
https://www.soundonsound.com/techniques/amplitude-modulation

Experimentera med olika inställningar på `range`. Exempelvis `[0.9, 1]` vilket ger en lätt tremoloeffekt.

Pulsbreddsmodulation

När man använder fyrkants-/pulsvåg kan man förändra pulsbredden. Detta är ett bra sätt att göra ett enkelt ljud med bara en oscillator att låta ”större”. När pulsbredden förändras, moduleras, till att vara bred till smalare och så tillbaka igen så skapas en övertonsserie som kontinuerligt förändras över tid, och det låter som om två instrument spelar samtidigt.

Genom att skapa en till sinusoscillator, kalla den för `pwm`, med låg frekvens och sedan använda `pwm` som input för `width:` så kommer övertonsserien att förändras.

Hur låter det? Varför?

_____

_____

_____

Vid en pulsbredd på 0 respektive 1 så är ljudet helt tyst. Varför blir det så? Använd `range` för att förändra omfånget på `pwm`. Hur låter det då?

_____

_____

_____

Testa ringmodulation

På samma sätt som när amplituden modulerades går det att skapa ringmodulation. Men, i detta fall ska ljudsignalen som modulerar den andra svänga mellan -1 och 1 som vanligt. Ringmodulation skapar en frekvensmix vilket ger summan och differensen mellan de två tonerna.

Hur låter det? Vad händer med övertonsserien?

_____

_____

_____

Läs mer här:
https://en.wikipedia.org/wiki/Ring_modulation

Om de två tonerna är samma ton (i samma oktav), vad händer då? Varför? Om de är samma ton men i olika oktav vad händer då?

_____

_____

_____

Testa enkel FM-syntes

I det förra exemplet modulerades frekvensen av oscillator 2 kring sin grundfrekvens. FM-syntes påminner om detta men där frekvensen av oscillator 2 multiplernas med frekvensen i oscillator 1. Grundfrekvensen hos oscillator 2 kommer då att vara samma som oscillator 1. Om oscillator 1 används utan förändring med `range`, oscillator 1 multipliceras med frekvensen (329.63Hz) på oscillator 2 och vi lyssnar på ljudet från oscillator 2. Hur låter det? På vilket sätt skiljer det sig från ljudet från bara oscillator 2 utan frekvensmodulation (frekvensmultiplikation)?

_____

_____

_____

Läs mer om FM-syntes här:
https://www.musictech.net/guides/essential-guide/how-fm-synthesis-works/

Vad händer om frekvensen på oscillator 2 ändras?

_____

_____

_____

Vad händer om ni ökar omfånget på oscillator 1 till att bara ha positiva värden? Eller till att ha ett större omfång än 2?

_____

_____

_____

Testa att lägga till distorsion

Att använda distorsion, exempelvis genom att överstyra eller klippa signalen, tillför nya övertoner. En enkel sinusvåg förändras gradvis till en fyrkantsvåg. I SC finns olika sätt att medvetet överstyra signalen. Här följer några olika exempel:

`Clip.ar(in: 0.0, lo: 0.0, hi: 1.0)`, där det enklaste är att skriva `.clip(-0.5, 0.5);` för att klippa ert summerade ljud till att svänga mellan -0.5 och 0.5. Vågformen som är lägre än -0.5 och högre än 0.5 klipps alltså bort.

Ett annat sätt är att använda `.softclip`. Då görs en mjuk klippning så att ljudet svänger mellan -0.5 och 0.5, men med en ganska stor skillnad i ljudet jämfört mot `clip`.

Experimentera med dessa två metoder och lyssna och jämför resultatet. Vilken låter bäst, vilket får flest övertoner, vad beror dessa skillnader på? Vad händer med ljudvolymen?

_____

_____

_____

Läs mer om distorsion här:
https://en.wikipedia.org/wiki/Distortion_(music)

Kontrollera en synthdef från klientsidan

Låt oss vidareutveckla vår kod en del och förändra syntens tonhöjd från klientsidan. Börja med att ändra `.play` i synthdefinitionen till `.add`. Om ni exekverar koden nu ska synten inte låta. Om den gör det, boota om tolken.

Sedan ska ni förbereda synthdefinitionen att ta emot argument för frekvens. Detta görs antingen med `arg inputFrequency;` eller med `| inputFrequency |`, där `inputFrequency` är ett namn ni bestämmer. Det går också att deklarera ett värde för argumentet, typ `arg inputFrequency = 220;`.

Sedan behöver ni skapa en environment-variabel (alltså en variabel med `~` i början av variabelnamnet) där synthdefinitionen registeras så att ni kan nå den enkelt på klientsidan i kod utanför koden som exekveras inom parenteser. Det gör ni med `Synth` enligt `Synth.new(defName, args, target, addAction: 'addToHead')` och efteråt lägger ni till

`.register`. Er kod bör likna:
```
~myFirstSynth = Synth.new(\firstSynth).register;
```

Läs mer här:
https://doc.sccode.org/Classes/Synth.html

Genom att använda `.set` skickar ni argument till synthdefinitionen på servern. Exempelvis så här:
```
~myFirstSynth.set(\inputFrequency, 440);
```

Detta exempel kan utvecklas mer genom att skapa en array med MIDI-noter, exempelvis så här i A moll:
```
var notes = [57, 60, 64, 69];
```

Sedan ska en `fork` skapas. En `fork` skapar en ny process som snurrar parallellt med andra processer i SC. En `fork` skrivs genom:

```
fork({
    // kod som ska köras
});
```

I `fork` ska en `loop` användas. En `loop` skrivs genom:

```
loop({
    // kod som ska loopas
});
```

Läs mer här:
https://doc.sccode.org/Reference/loop.html

Med en `for`-loop inuti `loop` kan vi gå igenom varje not i arrayen och skicka den till synthdefinitionen på servern och vänta 0.3 sekunder. Kolla på denna kod:
```
for (0, notes.size-1, { arg index;
    ~myFirstSynth.set(\inputFrequency, notes[index]);
    0.3.wait;
});
```

Läs mer här om kontrollstrukturer i SC:
http://danielnouri.org/docs/SuperColliderHelp/Language/Control-Structures.html
och om wait:
https://doc.sccode.org/Classes/Condition.html

Detta skapar en enkel sequencer som snurrar igenom noterna och börjar om igen, ett arpeggio. Men, frekvensen är väldigt låg och det hörs knappt att tonerna byts. Anledningen till detta är att vi nu inte skickar frekvens i Hz utan MIDI-notnummer. MIDI-informationen måste göras om till frekvens i synthdefinitionen, exempelvis när frekvensargumentet ges till oscillatorn, genom `.midicps` där cps står för cykler per sekund.

Vidare läsning

Om subtraktiv ljudsyntes:
https://en.wikipedia.org/wiki/Subtractive_synthesis

Om additiv ljudsyntes (FM-syntes):
https://en.wikipedia.org/wiki/Additive_synthesis