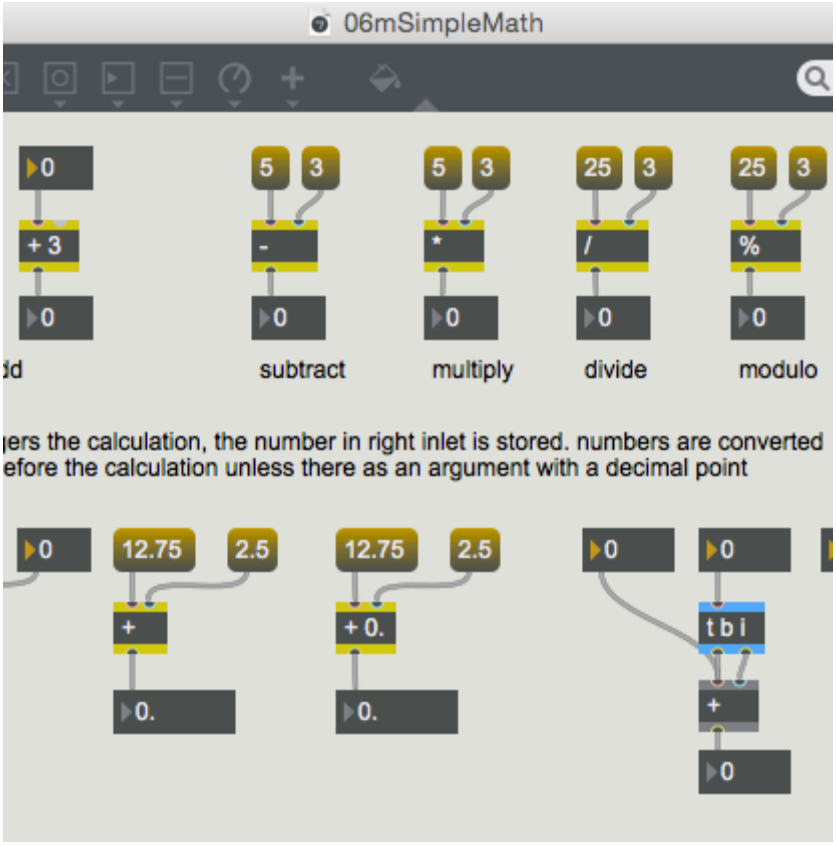
Kontrolloppgave til øvelsen *Simple Math in Max*

En av delene i denne øvelsen heter «Everybody loves Math». Dette stemmer kanskje ikke helt for alle som driver med lyd, musikk og kunst, men det er allikevel veldig praktisk å kunne beherske en del grunnleggende matematikk for å kunne lage små programmer. Øvelsen går igjennom ulike grunnleggende matematiske objekter.

Gå gjennom øvelsen *Simple Math* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 6

6.1 Gjør regnestykket 10.75 - 1.15. Husk å ta hensyn til flyttallene.

6.2 Lag en liten prosedyre der du setter et fast tall som blir multiplisert med et annet tall. Bruk et *button*-objekt for å trigge multiplikasjonen på nytt.

6.3 Lag et regnestykke som gjør om fra Fahrenheit til Celsius. Husk å benytt flyttall for at utregningen skal bli nøyaktig.

6.4 Vanligvis blir +-objektet bare trigget fra venstre inngang. Lag en liten prosedyre der det kan trigges både fra venstre og høyre inngang. Prosedyren skal også kunne ta flyttall.

6.5 Lag en prosedyre med en verdi som øker med +1 for hver gang du trykker på et *button*-objekt.

6.6 Lag en liten prosedyre som deler heltallet 9 på heltallet 2. I tillegg skal et objekt gi ut resten av divisjonen (hint: se på modulo objektet).

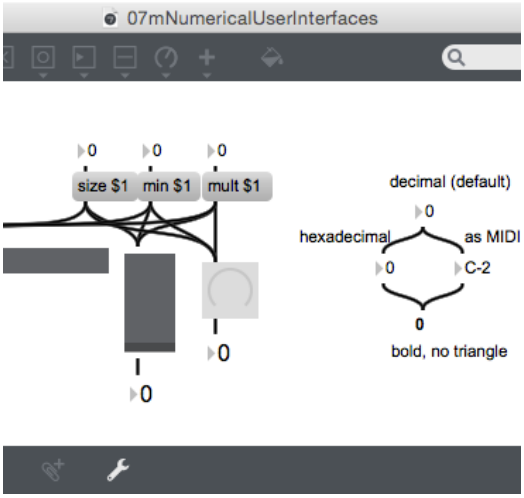
Nye objekter introdusert i denne øvelsen:

trigger	modulo	divisjon	pluss	minus	multiplikasjon
					

Kontrolloppgave til øvelsen *Numerical User Interfaces*

Denne øvelsen gir en enkel innføring i numeriske brukergrensesnitt. Mye av informasjonen i en vanlig Maxpatch representeres som tall, enten det gjelder lyd, musikk, video eller interaksjonsdata. Slidere, skruknapper og nummerbokser er noen av de objektene som kan gi rask og oversiktlig informasjon over hva som skjer i patchen.

Gå gjennom øvelsen *Numerical User Interfaces* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 7

7.1

Lag en liten prosedyre som gjør om fra note (tonehøyde og oktav) til MIDI-notenummer. Denne oppgaven skal kun løses ved hjelp av innstillingene til *number*-objektet. For å få til dette må du inn i «inspector».

7.2

Lag et brukergrensesnitt med to *slider*-objekter.
Minimumsverdi, totalstørrelse og multiplikasjonsfaktor for begge *slider*-objektene skal settes med skruknapper.

Slider nr. 2 skal ha:

en minimumsverdi som er halvparten så stor som slider nr. 1.

en totalstørrelse som er dobbelt så stor som slider nr. 1.

en multiplikasjonsfaktor som er tre ganger så stor som slider nr. 1.

Nye objekter introdusert i denne øvelsen:

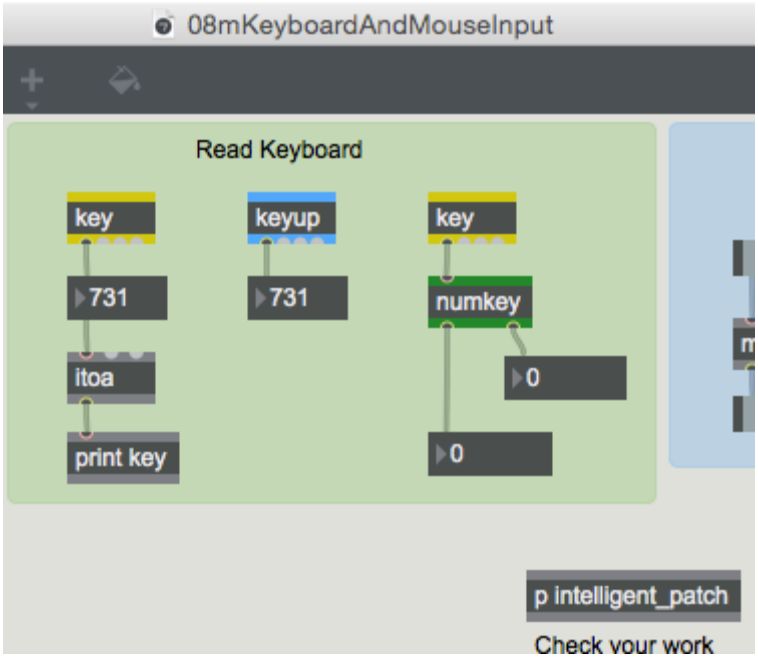
Slider

dial

Kontrolloppgave til øvelsen *Keyboard and Mouse Input*

I mange tilfeller styrer man store installasjoner eller viktige konsertavviklinger med museknappen eller mellomrom-tasten som eneste grensesnitt. Dette kan kanskje virke litt skjørt i forhold til et stort miksebord eller andre store mekaniske innretninger, men de enkleste løsningene er ofte de beste og da kan det være bra å vite hvordan man bruker tastaturet til å interagere med datamaskinen på en enkel måte.

Gå gjennom øvelsen *Keyboard and Mouse Input* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 8

8.1

Lag et grafisk grensesnitt med to *slider*-objekter som representerer musens bevegelser over skjermen. Et horisontalt *slider*-objekt representerer de horisontale bevegelsene til musen, og et vertikalt *slider*-objekt representerer de vertikale bevegelsene.

Et horisontalt *slider*-objekt får du ved å ta tak i nederste høyre hjørne på det vertikale *slider*-objektet og dra det til siden slik at det blir horisontalt.

8.2

slider-objektene skal kun være aktive når man trykker ned musen. Husk at *mousestate*-objektet må ha et *loadbang*-objekt for å aktiveres ved start. For å lese mer om hvordan du kan bruke *loadbang*-objektet, ta en titt på hjelpefilen til *mousestate*-objektet.

s*lider*-objektene skal også være aktive når man trykker ned mellomrom-tasten (spacebar). For å få dette til må du bruke *select*-objektet som vi enda ikke har gjennomgått. For å lese mer om hvordan du kan bruke *select*-objektet, ta en titt på hjelpefilen til *key*-objektet.

8.3

En liten nøtt: den vertikale slideren skal bevege seg opp når musen beveger seg opp, og likeledes ned når musen beveger seg ned. Dette innebærer en liten problemløsning.

Nye objekter introdusert i denne øvelsen:

numkey

mousefilter

mousestate

modifiers

keyup

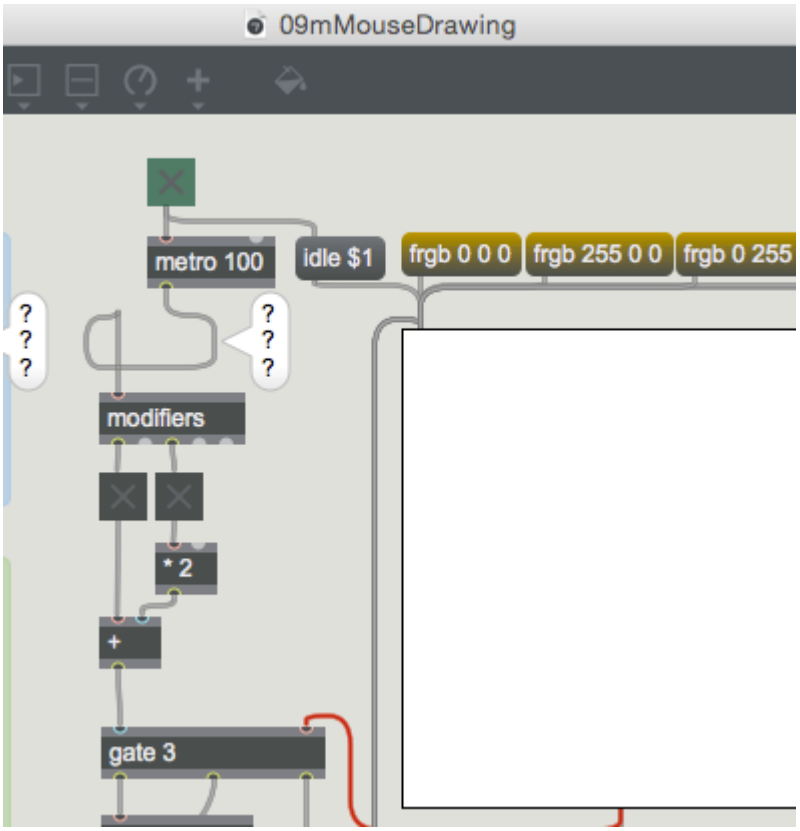
key

itoa

Kontrolloppgave til øvelsen *Mouse Drawing*

I denne øvelsen fortsetter vi med mus- og tastaturgrensesnitt, men i mer komplekse former. Vi skal se på *lcd*-objektet og hvordan vi arbeider med lister.

Gå gjennom øvelsen *Mouse Drawing* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 9

I likhet med mye annen programmering, er det sjeldent vi har behov for å finne opp hjulet på nytt i maxprogrammeringen. Derimot ser man vanligvis på hva andre har gjort, laster ned deler av en programmeringskode fra nettet, modifiserer koden og resirkulere den til eget bruk. Denne oppgaven dreier seg egentlig om å forandre på og resirkulere den tidligere øvelsespatchen *09mMouseDrawing*.

9.1

Lag en patch som tegner:

1. rektangel
2. sirkel
3. fylt rektangel

Rektangelet skal være lite, sirkelen skal være mellomstor og det fylte rektangelet skal være størst.

9.2

- For å tegne rektangelet skal Shift-tasten være nede.

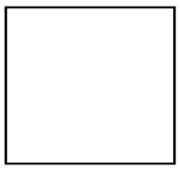
- For å tegne sirkelen skal Option/Alt-tasten være nede.
- For å tegne det fylte rektangelet skal både shift og Option/Alt være nede.

9.3

Tegningene skal slettes hver gang du trykker ned mellomromstasten.

Nye objekter introdusert i denne øvelsen:

lcd



prepend

unpack

pack

gate

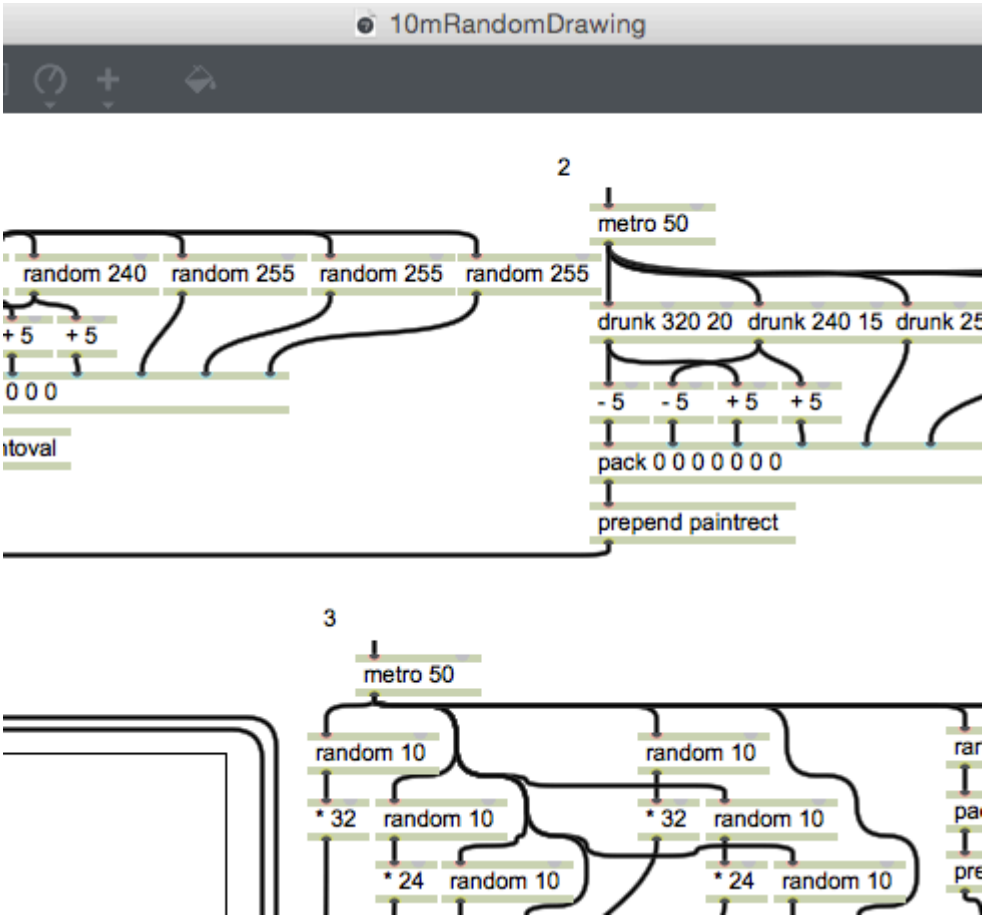
Kontrolloppgave til øvelsen *Random Drawing*

I denne øvelsen fortsetter vi med å tegne til *lcd*-objektet, men nå med hjelp av en tilfeldighetsgenerator. Tilfeldigheter er faktisk viktige i noe som er så tilsynelatende rigid som programmering. Tilfeldigheter hjelper oss med å ta valg, simulere «menneskelige» tendenser og naturfenomener, eller rett og slett gi oss et tallmateriale å arbeide med.

Det finnes mange ulike former for tilfeldigheter og de tilfeldighetene som finnes inne i et dataprogram er faktisk ikke tilfeldige, men bare algoritmer (dataprogrammer, regnestykker) som oppfattes tilfeldige. Dette blir kalt pseudotilfeldighet.

Man snakker også om ulike grader av kvalitet på tilfeldigheter. For å få ekte tilfeldighet kan man måle atmosfærisk støy og oversette dette til data. Nettsiden *www.random.org* tilbyr denne tjenesten. Dette er imidlertid litt tungvint i normal programmering. Da er det enklere å f.eks. bruke objektet *random*. Skal man derimot lage et dataprogram for å velge tall til lottotrekningen er nok ikke *random*-objektet i Max godt nok.

Gå gjennom øvelsen *Random Drawing* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 10

10.1

Lag et grensesnitt med 8 åtte horisontale slidere som viser tilfeldige tall mellom 0-127 ti ganger i sekundet. De fire øverste skal vise helt tilfeldige tall, mens de fire nederste skal vise tilfeldige tall som beveger seg med små skritt innenfor et begrenset område.

10.2

Lag en prosedyre som skriver 12 tilfeldige tall fra 1 til 12 til Maxvinduet hver gang du trykker på en knapp. Tallene skal ikke repeteres før alle tolv er brukt opp. For å unngå repetisjon må du bruke en annen tilfeldighetsgenerator enn *random*. (Se på hjelpefilen til *random* for å få tips om andre tilfeldighetsgeneratorer.) For å få en rekke på 12 tall bruker du objektet *uzi*. Åpne hjelpefilen til *uzi* for å forstå funksjonaliteten.

Nye objekter introdusert i denne øvelsen:

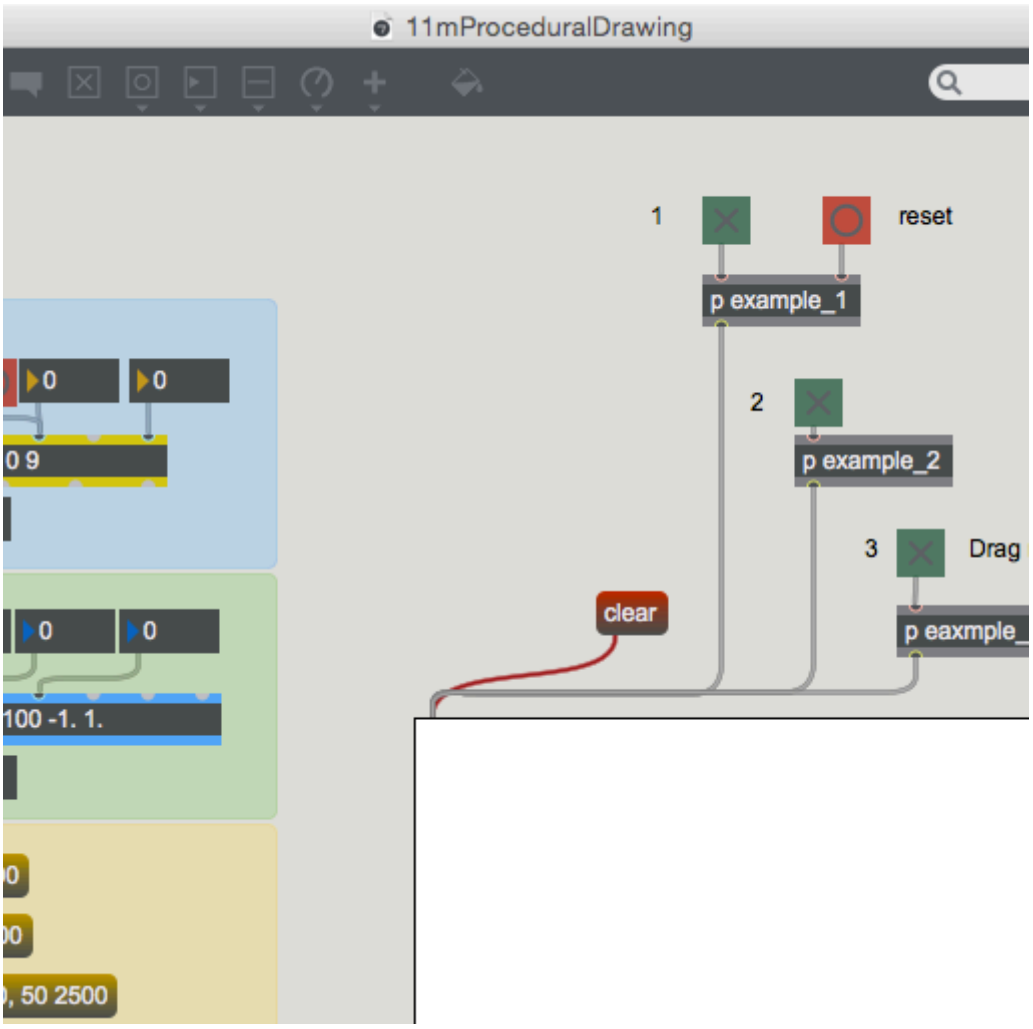
random

drunk

Kontrolloppgave til øvelsen *Procedural Drawing*

Denne øvelsen er en av de mer komplekse og inneholder både grafiske grensesnitt, trigonometri og ulike prosedyrer. Studer den nøye og bryt den ned i små biter for å forstå den bedre.

Gå gjennom øvelsen *Procedural Drawing* og den tilhørende øvelsespatchen. Om det er noen punkter du er usikker på, gå tilbake og repeter det som er uklart. Om det er et objekt du ikke er helt sikker på, les referansen eller hjelpefilen til objektet.



Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 11

Først fire enklere oppgaver:

11.1 Lag en prosedyre som teller fra -5 til +2 med et skritt hver gang du trykker ned en knapp.

11.2 Lag en prosedyre som teller fra 0 til 100 og tilbale til 0. Det skal komme nye tall for hvert 50. millisekund.

11.3 Bruk en slider med tall som går fra 0-127. Skaler disse tallene slik at de går fra -10.0 til +20.0 (flyttall).

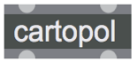
11.4 Lag en prosedyre som teller fra 128 til 256 på 2 sekunder. Når du trykker på en annen knapp skal verdien gå fra der du er til 500 på 1,5 sekund. Når du trykker på en siste knapp skal verdien gå fra der du er til 50 på 1 sekund.

11.5 En litt vanskeligere oppgave:

Øvelsespatchen *11mProceduralDrawing* demonstrerer mange funksjoner. La oss fokusere på den delen som er merket med «*1*» øverst i midten av patchen. Denne bruker en beskjed som heter *paintoval* til å skrive ovaler til *lcd*-objektet. Studer dokumentasjonsmaterialet og hjelpematerialet for *lcd*-objektet og les dokumentasjonen for *paintoval*. I stedet for en patch som lager en rekke av sirkler med gradvis skiftende farge, lag en patch som lager sirkler med tilfeldig valgt farge og tilfeldig valgt plassering.

Nye objekter introdusert i denne øvelsen:

cartopol

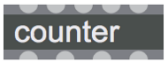


sin

line

scale

counter

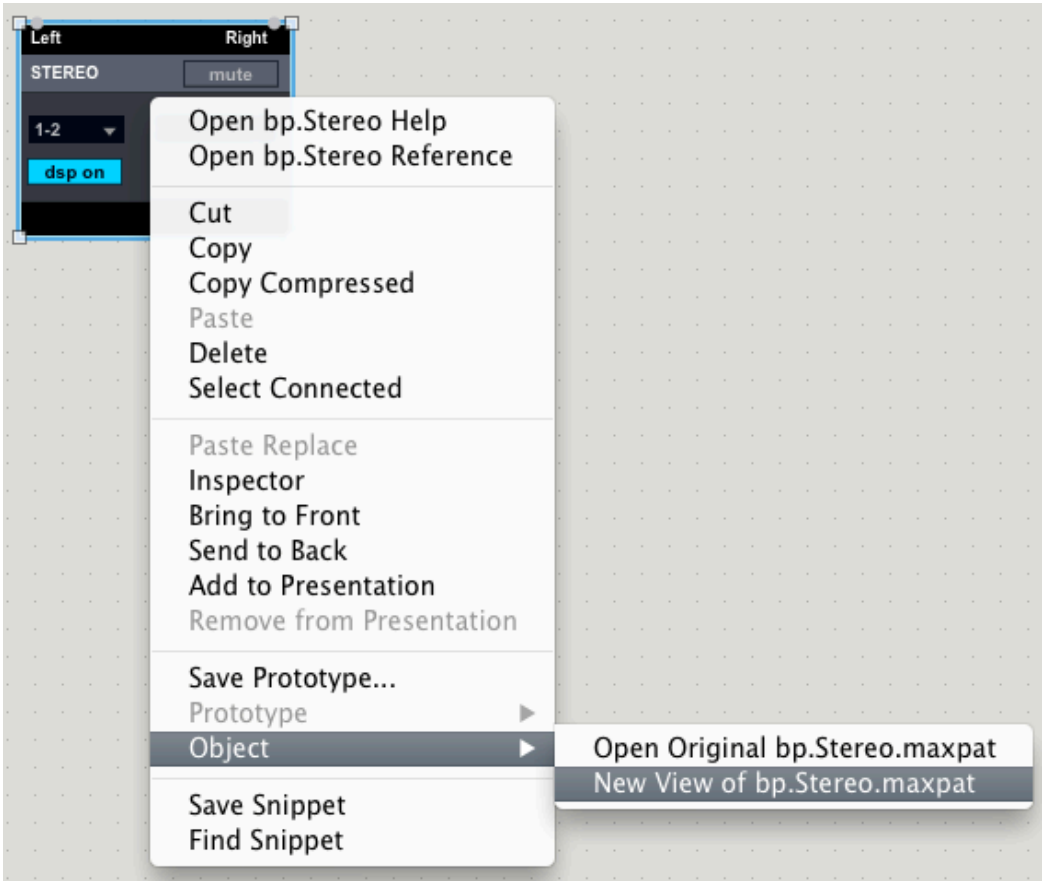


BEAP — mer granulærsyntese

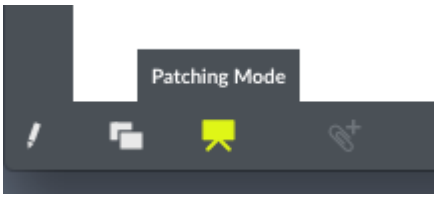
I denne øvelsen skal vi se nærmere på hvordan BEAP-patchene fungerer og gå videre med granulærsyntesen.

Med hjelp av innkapsling (*encapsulation*), kan vi plassere deler av patchen vår inne i en såkalt subpatch ved hjelp av *patcher*-objektet. Vi skal ikke se så inngående på dette nå, men heller se nærmere på dette i øvelsen *Encapsulation — Patchers inside of patchers*. En spesiell variant av innkapsling er *bpatcher*-objektet. Vi går gjennom dette i øvelsen *Bpatchers — Working with inlined patchers*, men kort fortalt kan vi si at en bpatcher er en subpatch der deler av patchen er synlig. På denne måten kan vi lage komplekse patcher med et enkelt brukergrensesnitt. Det er slik BEAP-patchene fungerer. De ser ut som enkle lydbehandlingsobjekter men egentlig er de komplekse patcher innkapslet i et *bpatcher*-objekt.

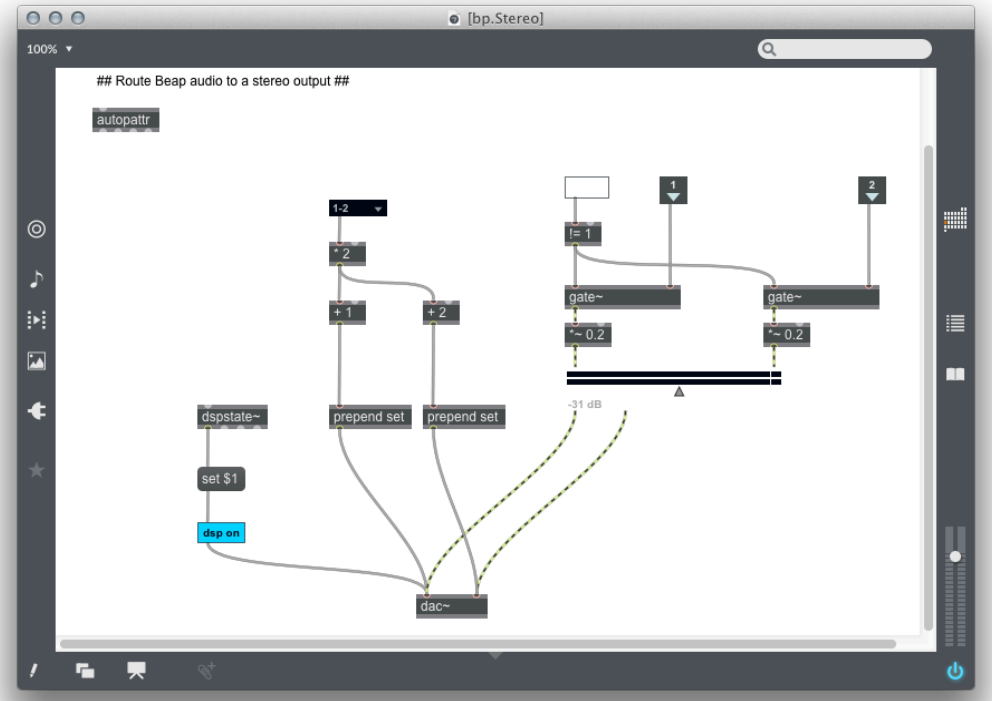
La oss ta BEAP Stereo Output patchen som et eksempel. Om du høyreklikker (eller ctrl-klikker) på BEAP stereo output patchen kan du velge «New View of bp.Stereo.maxpat»:



Når vi får opp denne patchen er den lagret i *Presentation Mode*. *Presentation Mode* er en enkel måte å lage oversiktlige grensesnitt på, og vi skal se nærmere på dette i øvelsen *Presentation Mode — Creating a presentation interface for a patcher*. Inntil videre skrur vi av Presentation Mode med å velge *Patching Mode* nederst til venstre i vinduet:



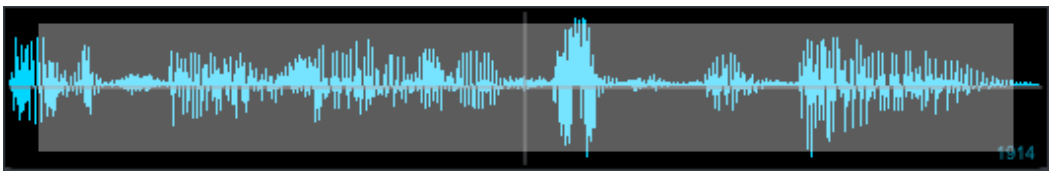
Vi kan nå se hva som egentlig skjuler seg på innsiden av BEAP stereo output patchen:



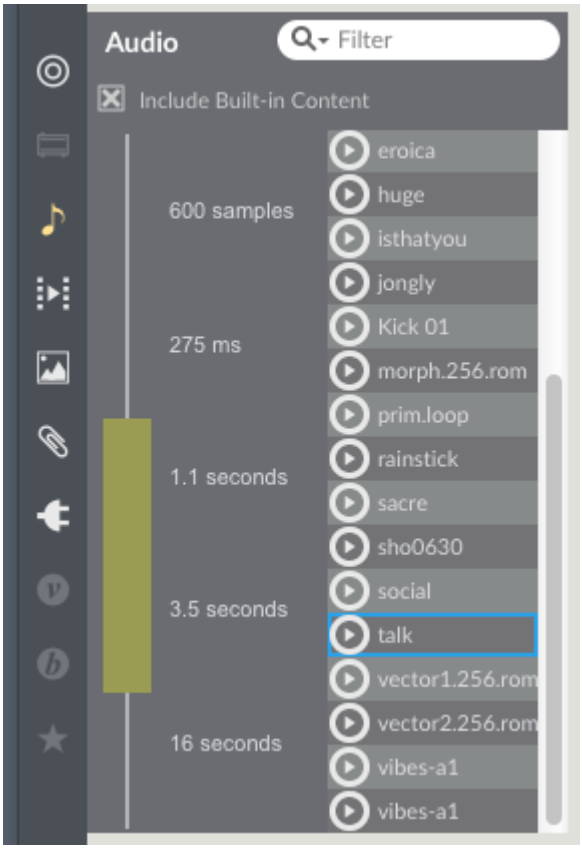
Som vi sa i forrige forelesning er hovedpoenget med BEAP-øvelsene å undersøke interessante lydbearbeidingsmetoder i Max på en enkel og morsom måte. Det er altså ikke meningen at du skal forstå alt som skjer på innsiden av en BEAP-patch. Etter hvert som du blir flinkere i Max og MSP kan du gå tilbake til disse BEAP-patchene og studere dem for å få innsikt i interessante lydbearbeidingsmetoder. Det finnes også hjelpefiler for BEAP-patchene. Disse får du opp på vanlig måte gjennom å velge BEAP-patchen og så velge hjelpefilen i *Help*-menyen, eller bare holde ned *alt*-knappen og trykke på BEAP-patchen.

La oss fortsette med granulærsyntese-patchen som vi lagde i forrige BEAP-øvelse.

Her er det flere mulige innstillinger. Med *mute*-knappen øverst til høyre kan man skru lyden til denne BEAP-patchen av og på. Midt i BEAP-patchen vises lydfilen som brukes i granulærsyntesen:



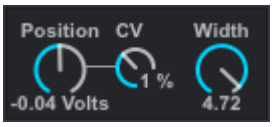
Du kan dra og slippe lydfiler over dette vinduet for å skifte ut lydene. Dette kan du enten gjøre direkte fra en mappe på datamaskinen din eller fra *audio*-menyen til venstre i Max patcher vinduet. I denne øvelsen skal du bruke lydfilen «talk» fra *audio*-menyen til venstre i Max patcher vinduet:



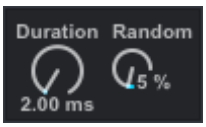
Helt til venstre i BEAP-patchen styrer du tonehøyden på lydavspillingen med «Offset». Et lite tips: om du holder nede *cmd*-knappen (cmd eller ⌘ i OSX, ctrl i Windows) samtidig som du drar musen opp og ned kan du stille inn verdiene med høyere presisjon.

Random styrer graden av tilfeldig avvik fra den tonehøyden du har valgt. *CV2* styrer graden av ekstern modulasjon. Dette skal vi ikke se på i denne øvelsen.

I det neste feltet kan du velge *Position*. Dette bestemmer hvor i lydfilen du spiller fra. *Width* bestemmer graden av tilfeldig valgt posisjon.



I granulær syntese blir lyden hakket opp i små biter som deretter blir satt sammen igjen. *Duration* bestemmer hvilken varighet disse bitene skal ha. *Random* bestemmer hvor stor grad av tilfeldighet det skal være i lengden på disse bitene.



Pan bestemmer hvor lyden skal komme fra i stereobildet, og *Random* bestemmer med hvor stor grad av tilfeldighet lydbitene skal plasseres mellom venstre og høyre kanal i stereobildet.

I det siste feltet kan du sette to verdier:

I det nederste feltet kan du sette hvor mange «stemmer» du vil at granulærsyntesen din skal ha. Granulærsyntese fungerer ved å hakke lyden opp i små biter som man setter sammen igjen. Du kan ha store skyer med 30-40 slike biter, eller stemmer, eller du kan ha bare noen få. Minimumsantall i denne BEAP patchen er 8. Jo flere stemmer du har jo tettere blir klangen, men da bruker du også mer datakraft.

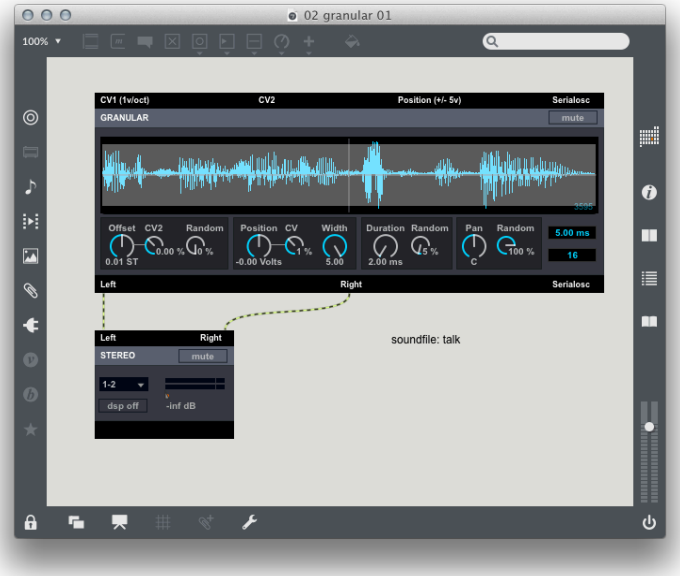
I det øverste feltet kan du sette hvor ofte vi skal starte et nytt korn. I dette eksempelet er det satt til 5 millisekunder.

Sky av korn

La oss nå bruke denne granulærsyntesepatchen til å lage en sky av korte lydkorn. Husk at vi i denne øvelsen skal bruke lydfilen «talk» fra *audio*-menyen til venstre i Max patcher vinduet.

Sett *Duration* til 2 ms for korte lydkorn, og *Random* til 5% for små avvik. La *Position* stå på 0, men sett *Width* til maksimumsverdien 5 slik at lydkornene sprer seg over hele lydfilen. Set *Pan* til C (sentrum) men sett *Random* til maksimumsverdien 100 slik at lyden hopper tilfeldig mellom venstre og høyre kanal. La de andre verdiene være som de er. Du har nå en sky av korte lydkorn.

Husk at du må velge «Autosave Snapshot» for å lagre innstillingene, og «Embed Snapshot in Parent» for at «Granular»-patchen skal huske hvilken lydfil du har valgt (dette er forklart i BEAP oppgave nr. 1).



Hele patchen