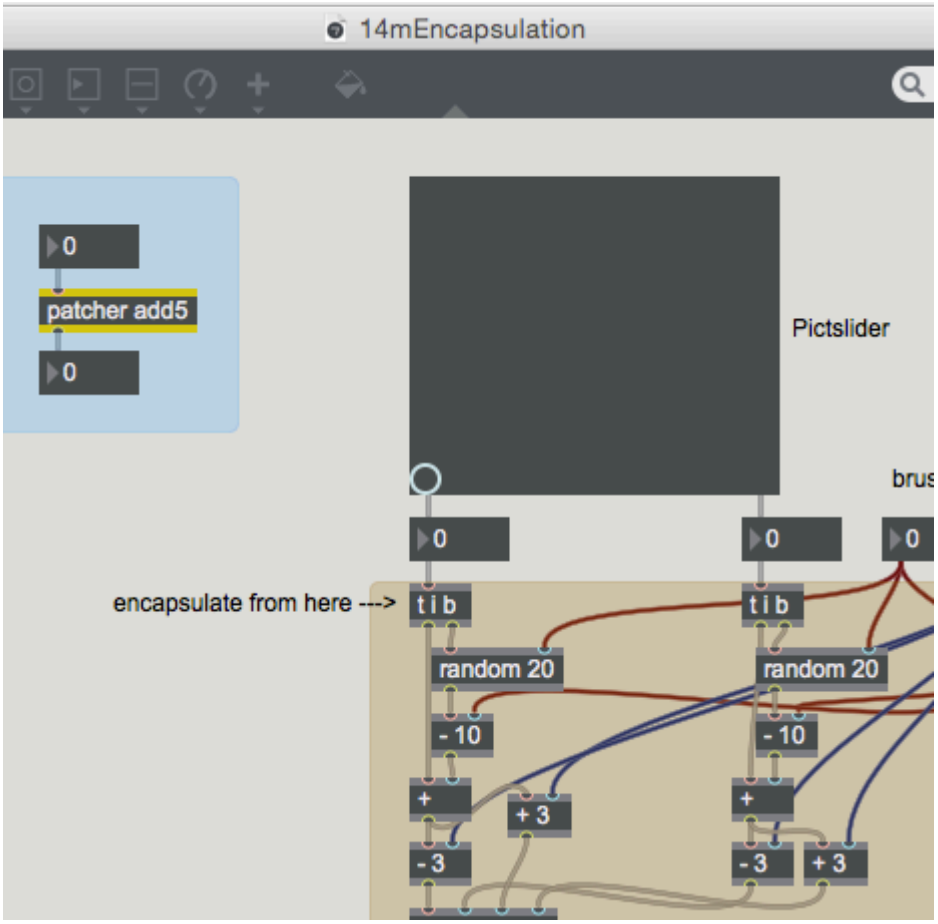
Kontrolloppgave til øvelsen *Encapsulation*

Vi hopper rett til øvelsen *Encapsulation*. Øvelsene *Movie Playback* og *Movie Sequencing* benytter seg av *imovie*-objektet til å spille av Apple Quictime videoer. Dette er et gammelt objekt, og det meste av arbeidet med video i Max foregår nå med tillegget *Jitter*. Arbeid med video og *Jitter* ligger utenfor innholdet av dette kurset.

I øvelsen *Encapsulation* skal vi organisere patcher med å kapsle de inn i såkalte subpatcher. Vi skal også se på hvordan vi skal bruke den såkalte *Encapsulate*-funksjonen.

Gå gjennom øvelsen *Encapsulation* 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.

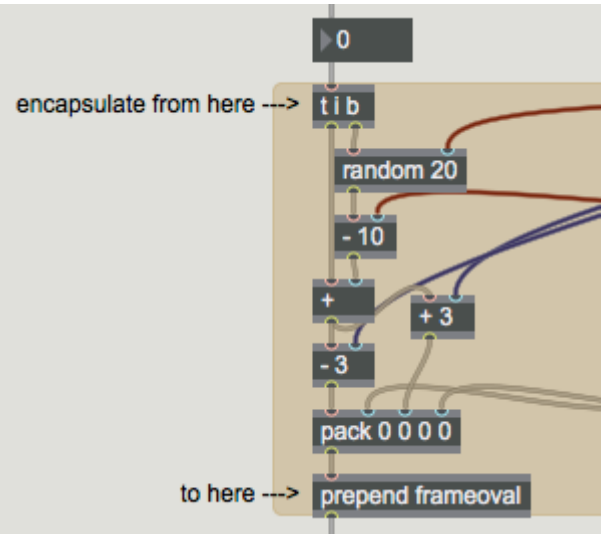


Gå igjennom de ulike delene av patchen og vær sikker på at du skjønner hvordan alle delene fungerer. Når alt er klart, gå videre til oppgaven.

Oppgave 12

Lagre patchen *«14mEncapsulation»* under et nytt navn, f.eks. *«my_tutorial_14»*.

12.1 «Kapsle inn» den delen som er markert med *«Encapsulate from here -> to here»* ved hjelp av *Encapsulate*-funksjonen fra en av Maxmenyene.



12.2 Lag en subpatch med navn «calculation». Subpatchen skal kunne takle flyttall og inneholde følgende regnestykke:

$$\frac{(X * 10) + 5}{2}$$

Viktige objekter og funksjoner:

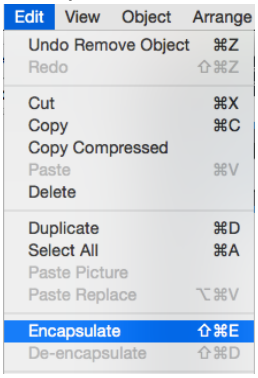
patcher

Inlet

Outlet

Pitchslider

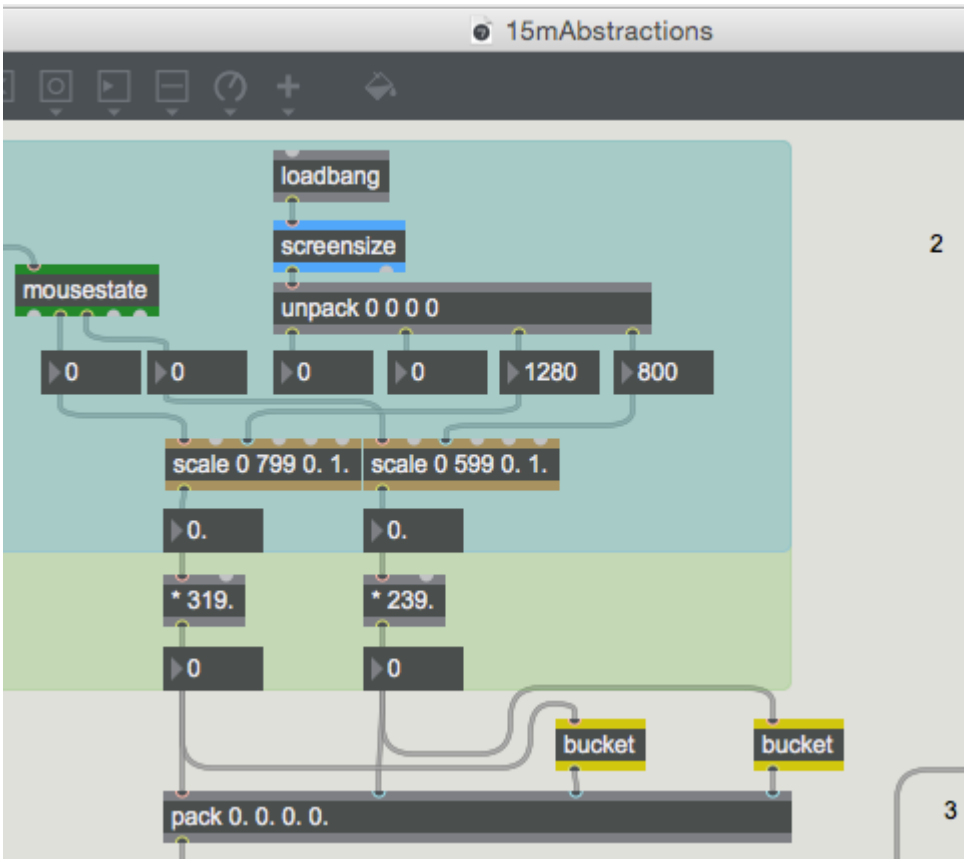
Encapsulate



Kontrolloppgave til øvelsen *Abstractions*

I denne øvelsen skal vi jobbe med såkalte abstraksjoner *(abstractions)*. En abstraksjon er beslektet med en subpatch, men i stedet for å lagre koden inne i patchen, lagrer vi koden som en egen patch vi kan kalle på samme måten som et normalt objekt.

Gå gjennom øvelsen *Abstractions* 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 13

Ta utgangspunkt i subpatchen vi kalte for «regnestykke» i forrige tutorial, og lagre den som «*calculation2*».

I motsetning til forrige regnestykke:

$$\frac{(X*10)+5}{2}$$

skal denne abstraksjonen ha to ekstra variabler (y og z):

$$\frac{(X*Y)+Z}{2}$$

Disse variablene skal du kunne definere som argumenter til abstraksjonen når du kaller den opp. Se på den delen av øvelsen som heter «Using an abstraction with replaceable arguments» for å skjønne hvordan du skal gjøre dette.

Kall denne abstraksjonen opp fra en annen patch.

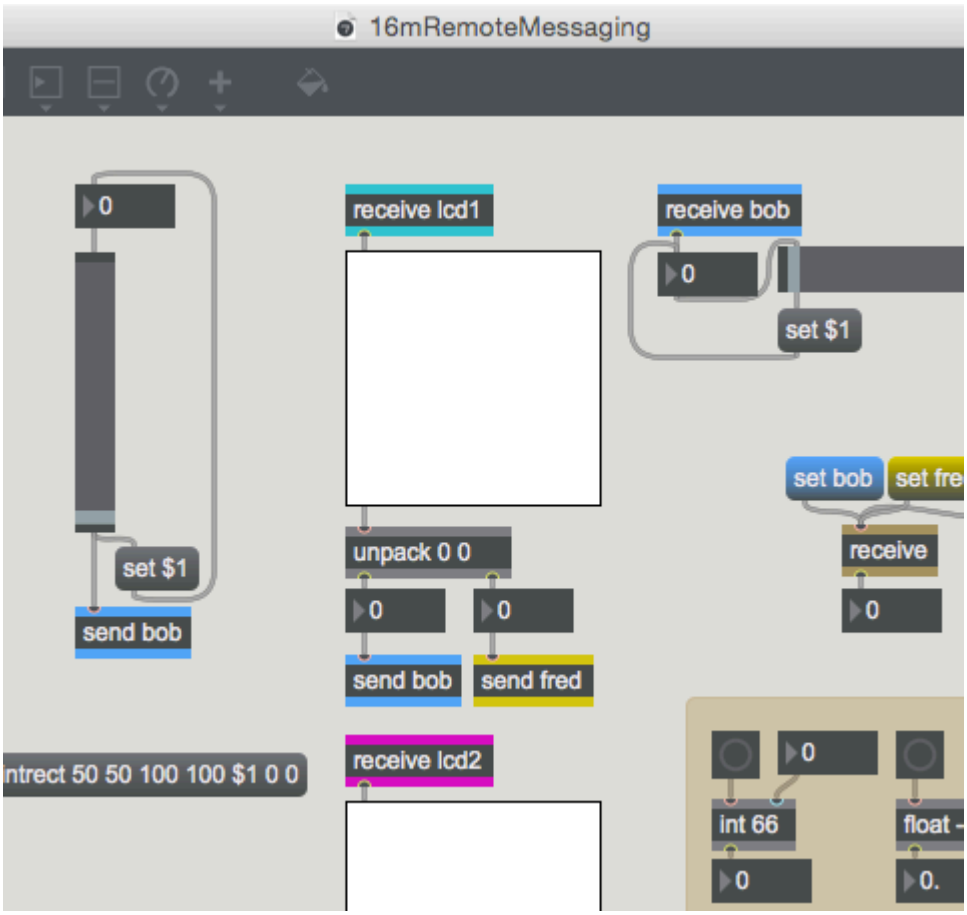
Nye objekter introdusert i denne øvelsen:

bucket

Kontrolloppgave til øvelsen *Remote Messaging*

I denne øvelsen skal vi jobbe med å sende data rundt i patchen uten å benytte patchkabler.

Gå gjennom øvelsen *Remote Messaging* 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 14

Denne oppgaven består av syv deloppgaver.

14.1 Koble to nummerbokser sammen med *send* og *receive*.

14.2 Legg til et ekstra *receive*-objekt. I likhet med det første *receive*-objektet, skal denne også motta tallene som blir sendt til *send*-objektet. Men: dette objektet skal bare sende ut tall til utgangen når den er aktivert med et et *bang*.

14.3 Deloppgave 14.2 kan løses på to måter. En av de er ved hjelp av et *value*-objekt. Vis begge måtene.

14.4 Lag en prosedyre som sender og mottar teksten «test_hello».

14.5 Legg til et objekt der du kan bytte mellom å sende til to ulike *receive*-mottakere.

14.6 Sett også opp et objekt der du kan bytte mellom å motta fra to ulike *send*-objekter.

14.7 Opprett et *message*-objekt som sender direkte til et av *receive*-mottakerene uten å koble patchtråder fra *message* videre til andre objekter. Om dette skulle være uklart se den delen av øvelsen som har overskriften *Working with ; and forward*.

Vær nøye med å merke de ulike delene av patchen med *comment*-objekter. Denne patchen inneholder såpass mange ulike deler at det er viktig å holde oversikten.

Nye objekter introdusert i denne øvelsen:

send

receive

forward

int

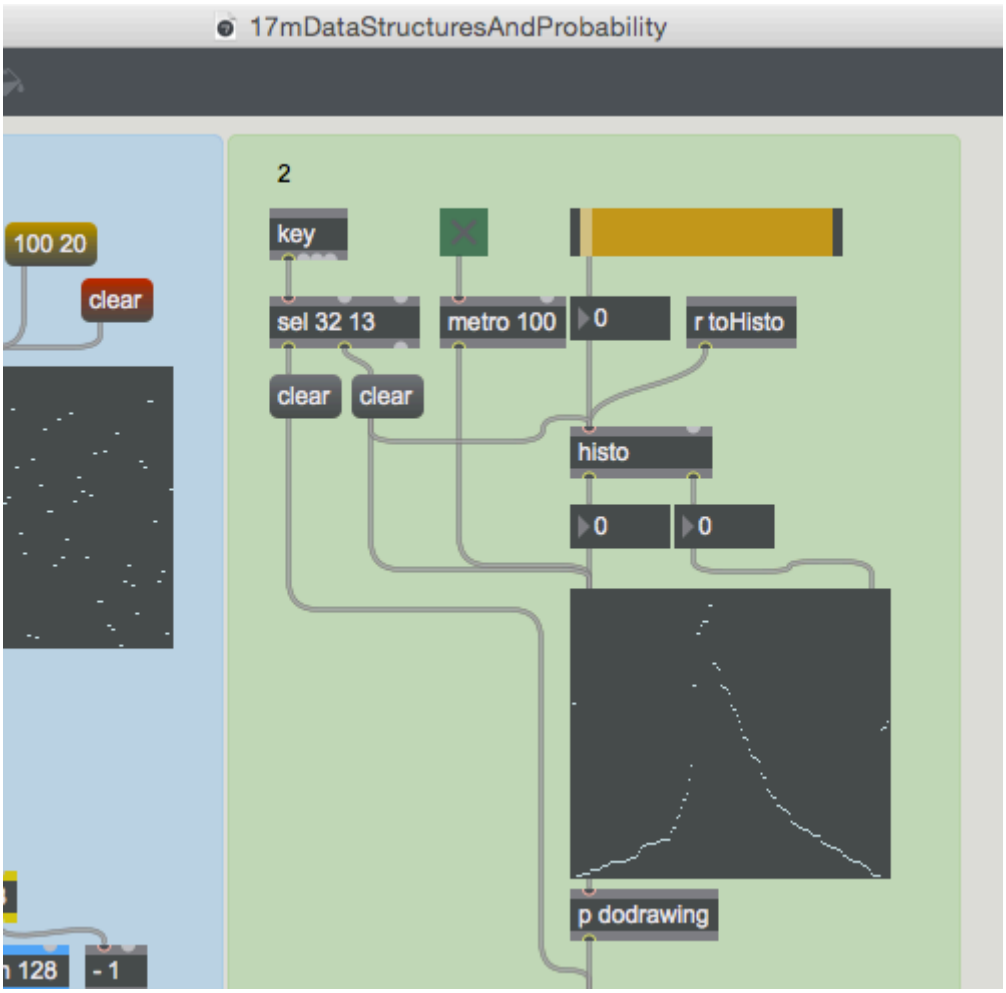
float

Value

Kontrolloppgave til øvelsen *Data Structures and Probability*

I denne øvelsen skal vi jobbe med tabeller og sannsynlighetsberegning. Sannsynlighetsberegning er en teknikk som bl.a. ble mye brukt av den gresk-franske komponisten Iannis Xenakis.

Gå gjennom øvelsen *Data Structures and Probability* 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 15

15.1

Bruk *table*-objektet, sett opp en tabell som tar tall fra 0 til 100 inn, og gir ut tall fra 0 til 200 og tilbake til 0. Lagre dataene i tabellen sammen med patchen slik at de er der neste gang du åpner patchen. Data inn og ut av tabellen leveres med en *slider*.

15.2

Gjør det samme en gang til, men denne gangen bruker du en tabell med grafisk grensesnitt.

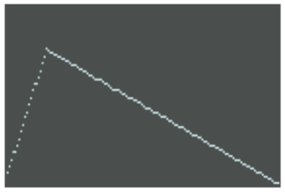
15.3

Send ut en stor mengde tall som representerer Normalfordelingen, Gausskurven (På Engelsk: Gaussian (or normal) distribution) og visualiser dette i et *itable*-objekt. Hint: Programmeringen for Gausskurven finnes allerede inne i eksempelpatchen til øvelsen.

Nye objekter introdusert i denne øvelsen:

table

itable



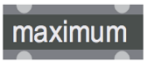
uzi

swap

histo

minimum

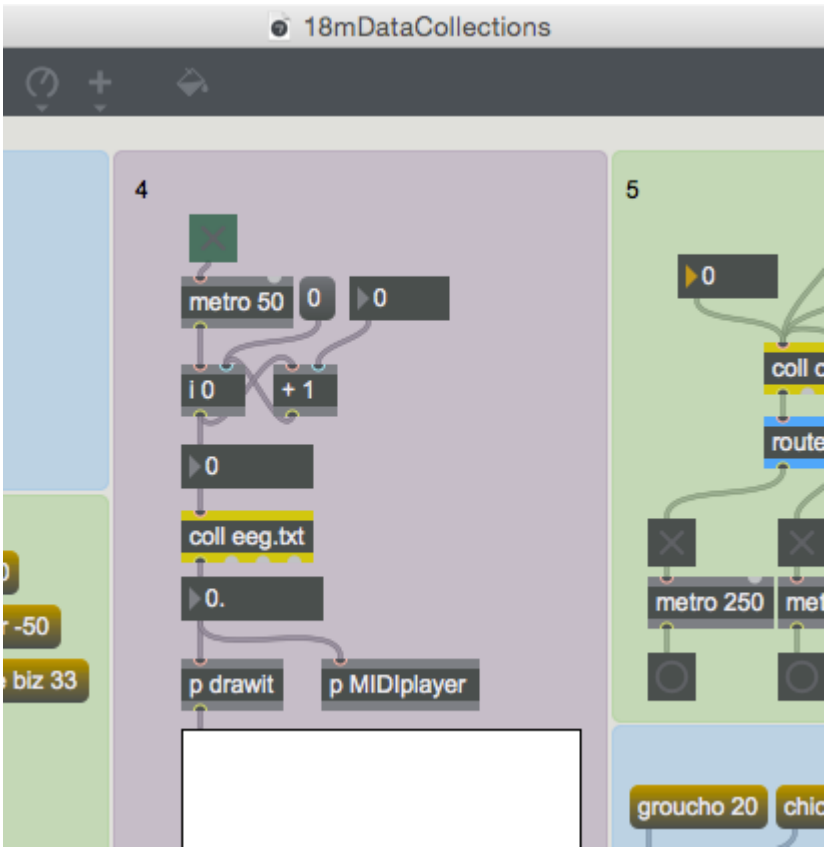
maximum



Kontrolloppgave til øvelsen *Data Collections*

I denne øvelsen skal vi lære hvordan vi kan arbeide med lister, hvordan vi kan lagre de på ulike måter og hvordan de kan dirigeres til ulike steder i patchen.

Gå gjennom øvelsen *Data Collections* 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 16

I denne oppgaven skal vi som sagt arbeide med lister. Lag tre forskjellige lister.

16.1 Lagre 5 forskjellige tall på adresse 0-4.

I den samme listen skal du også lagre 2 forskjellige tall på to forskjellige navneadresser, f.eks. «one» og «two». Hvis du åpner patchen igjen skal tallene fortsatt være lagret i listen. På adresse 5-7 skal du lagre 3 forskjellige tekstbeskjeder. Skriv ut alle data i listen til Maxvinduet. Når du åpner patchen igjen skal informasjonen fortsatt være lagret i listen.

16.2 Lag en tekstfil som du lagrer i samme mappen som patchen. Kall den f.eks. «*twelvenumbers.txt*». Tekstfilen skal inneholde en liste med tolv ulike tall. Skriv ut alle data i listen til Maxvinduet.

16.3 Lag en liste med 16 tall der de ulike tallene rutes til 4 forskjellige steder i henhold til adresser som blir oppgitt i listen. Skriv ut alle data i listen til Maxvinduet.

Nye objekter introdusert i denne øvelsen:

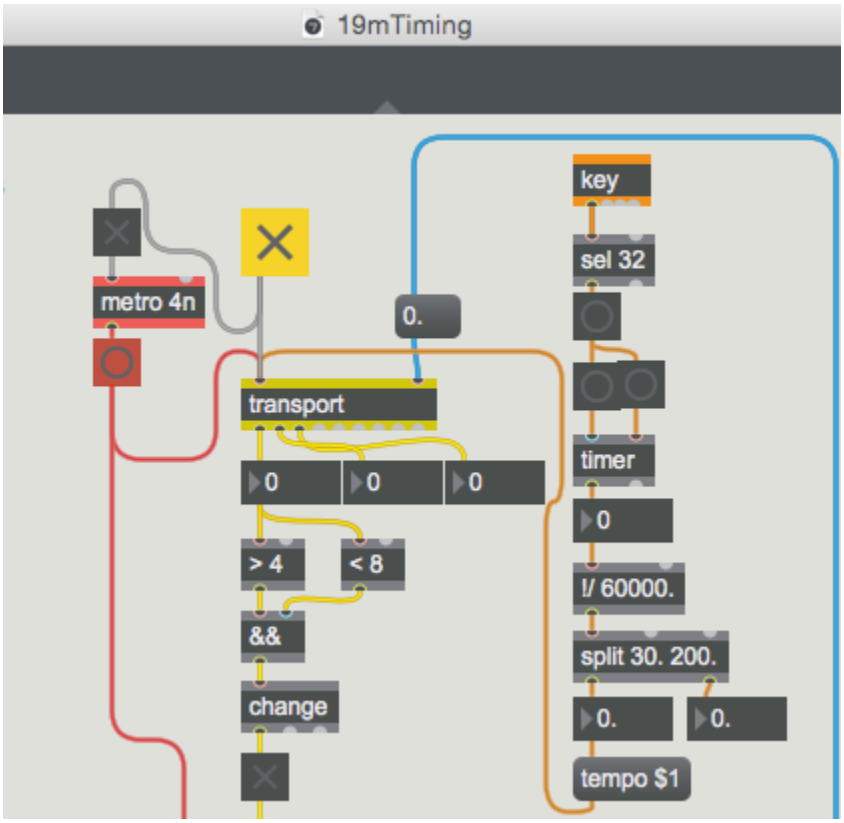
coll

route

Kontrolloppgave til øvelsen *Timing*

I denne øvelsen skal vi lære hvordan vi kan jobbe med tidsaspekter, forsinkelser og tradisjonelle musikalske verdier for tid, slik som tempo, takt og slag.

Gå gjennom øvelsen *Timing* 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 17

Lag først fire små enkle rutiner:

17.1 Forsink et bang med ett sekund, og skriv denne *bang*-beskjeden ut i Maxvinduet.

17.2 Forsink et tall med to sekunder, og skriv ut tallet i en nummerboks.

17.3 Lag en rutine som måler tiden som har gått mellom to *bang*-beskjeder.

17.4 Lag en «stoppeklokke» som måler tiden fra start til stopp. Resultatet skal vises i sekunder med tre desimaler.

17.5 Lag deretter en rutine som viser takter og slag når den blir startet. Gjør det også mulig å forandre tempo. Hver gang du starter rutinen skal denne delen av patchen starte fra første slag i første takt. Ved takt fire slag to skal teksten «bar four beat two» skrives ut til Maxvinduet.

Nye objekter introdusert i denne øvelsen:

pipe

delay

clocker

timer

>

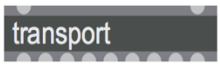
<

==

!=

&&

transport



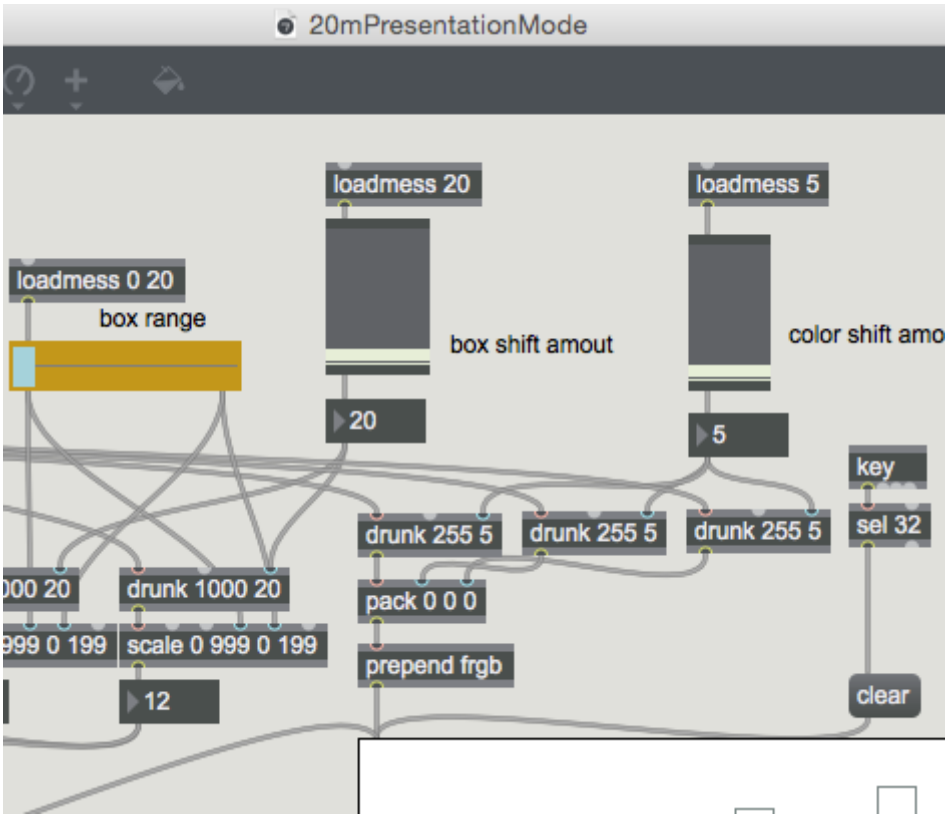
timepoint

split

Kontrolloppgave til øvelsen *Presentation Mode*

Presentation mode gjør det enklere å lage brukervennlige grensesnitt. I denne oppgaven skal vi bruke det vi lagde i forrige oppgave, men forbedre brukergrensesnittet slik at det blir mer oversiktlig.

Gå gjennom øvelsen *Presentation Mode* 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 18

Ta utgangspunkt i del to fra forrige oppgave, altså den delen som viser takter og slag.

I *Presentation mode* skal du vise et grensesnitt som kun inneholder følgende:

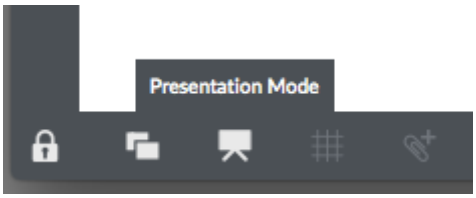
start/stopp, takt, slag og tempo.

Husk å konfigurer patchen slik at den vises i *Presentation mode* når du starter den. Dette gjør du ved å gå til *View*-menyen og velger *Patcher Inspector*. Gå deretter til *Presentation attribute* og velg *Open*.

Til slutt, lukk *Patcher Inspector* og lagre patchen.

Nye objekter introdusert i denne øvelsen:

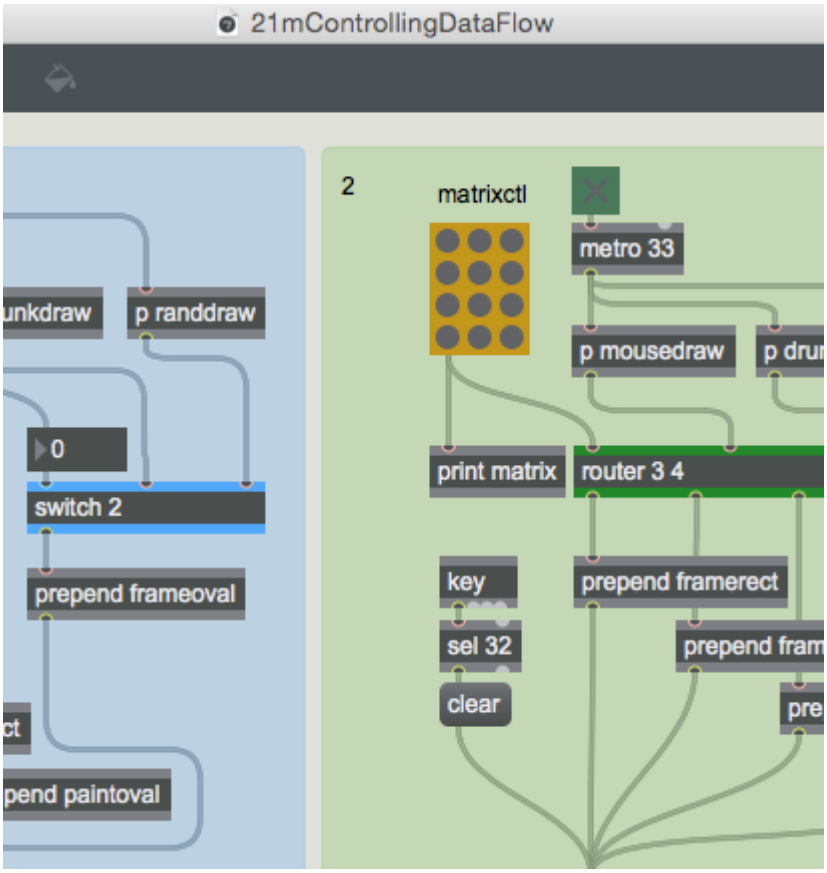
Presentation mode


Presentation Mode

Kontrolloppgave til øvelsen *Controlling Data Flow*

Denne øvelsen ser på hvordan man kan dirigere datastrømmene i en patch. Øvelsen ser også på hvordan man kan lage grensesnitt for å kontrollerer datastrømmer.

Gå gjennom øvelsen *Controlling Data Flow* 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 19

Lag en patch som har tre deler. Det skal gå heltall gjennom hver av delene, men tallene skal dirigeres på ulike måter.

19.1 Del 1 skal ha fire innganger og en utgang. Inngang velges med trykknapper.

19.2 Del 2 skal ha en inngang og fire utganger. Utgang velges med trykknapper.

19.3 Del 3 skal ha to innganger og fire utganger. Et felles grafisk grensesnitt velger både inn- og utganger.

Nye objekter introdusert i denne øvelsen:

gate

switch

router

radiogroup

matrixctrl

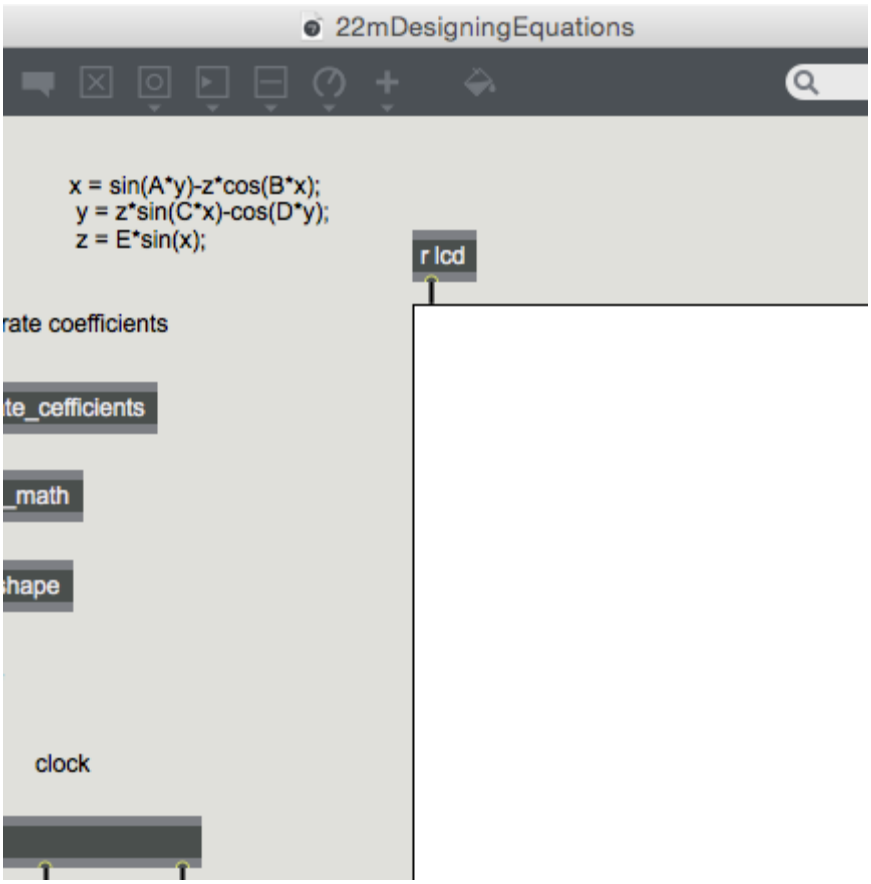
gswitch

gswitch2

Kontrolloppgave til øvelsen *Designing Equations*

I denne øvelsen skal vi se på skaleringer, lagring av tall og ulike måter å lage matematiske ligninger på. Objektet *expr* gjør det mulig å skrive matematiske ligninger direkte inn i en Max patch, og kan derfor være en viktig problemløser.

Gå gjennom øvelsen *Designing Equations* 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 20

Lag følgende enkle regnestykke:

$$\frac{y}{2}$$

«y» er et tall som sendes inn i regnestykket under oppstart. Koble et *bang*-objekt til patchen for å trigge en ny utregning. Første gang utregningen blir trigget, vil resultatet være y/2. Andre gang vil resultatet være (y/2)/2. Tredje gang vil resultatet være ((y/2)/2)/2, osv. Med andre ord må «y» byttes ut med resultatet fra forrige utregning.

Nye objekter introdusert i denne øvelsen:

expr

value

Kontrolloppgave til øvelsen *Basics — Getting MIDI input and output*

Denne øvelsen inneholder noen svært grunnleggende eksempler på hvordan man får MIDI inn og ut av en patch. MIDI (Musical Instrument Digital Interface) er en standard protokoll for kommunikasjon mellom musikk og lydutstyr som ble etablert i 1982. Dette er altså en over 30 år gammel datastandard som fortsatt er i bruk, noe som i seg selv er ganske bemerkelsesverdig i teknologisk sammenheng. Riktignok er ikke den fysiske delen av MIDI-standarden så mye i bruk lenger (kabler, MIDI-grensesnitt, fysisk lydutstyr etc. er i stor grad erstattet av datamaskiner og USB-grensesnittet). Men konseptet med MIDI er fortsatt viktig i mye musikkproduksjonsprogramvare. Den lave oppløsningen gjør MIDI derimot lite egnet til annet enn enkle musikkgrensesnitt som f.eks. klavertastatur, trommemaskiner og begrensede miksergrensesnitt. For mer komplekse applikasjoner kan kommunikasjonsprotokoller som Open Sound Control (OSC) eller sensorgrensesnitt som f.eks. Arduino være et alternativ. MIDI er allikevel en standard som er greit å kjenne til og som gjør det enkelt og greit å koble opp enkelt musikkutstyr. Skal du ha et klaviatur eller noen enkle skruknotter så finnes det ikke noen enklere måte å gjøre det på.

En annen grunn til at vi bruker MIDI her er at det er enkelt å teste ut ulike typer datastrømmer med den innebygde MIDI-delen i Max. Med den innebygde MIDI-delen kan vi f.eks. spille av en rekke tall som f.eks. enkle MIDI-noter med den innebygde pianolyden. Dette låter selvsagt helt forferdelig, men klangkvaliteten er ikke poenget her, snarere å illustrere ulike strukturer og prosesser med enkle lyder. Dette kommer vi tilbake til i senere øvelser.

Gå gjennom øvelsen *Basics — Getting MIDI input and output* 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 21

21.1
Lag en enkel prosedyre som tar noter inn fra klaviaturet og spiller de ut igjen.

21.2

For hver note som trykkes ned skal det spilles av en akkord på tre toner. Denne akkorden bygger på tonen som trykkes ned. Avstanden mellom tone 1 og tone 2 er en liten ters og avstanden mellom tone 1 og tone 3 er en ren kvint.

Viktige objekter:

Gswitch2

midiinfo

ctlout

ctlin

noteout

notein

midiout

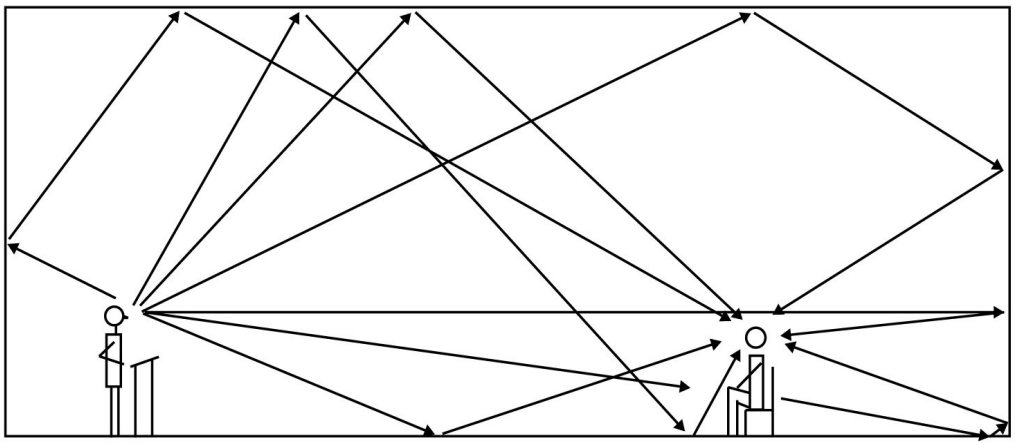
midiin

BEAP — granulærsyntese og romklang

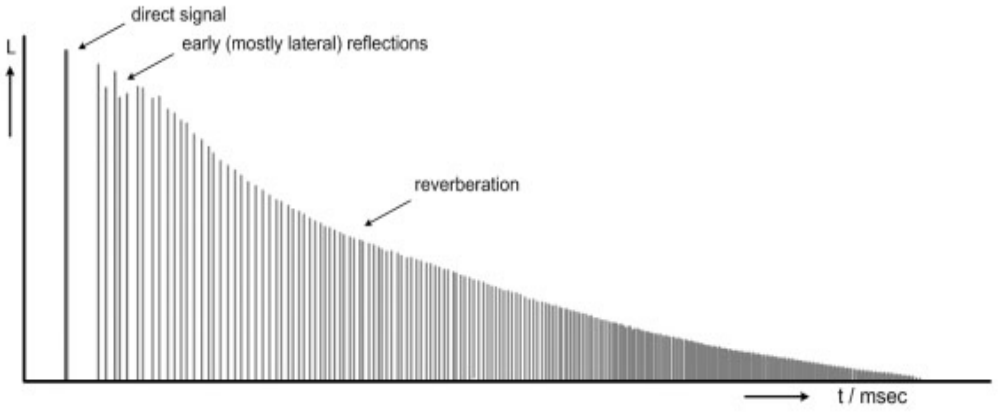
I denne øvelsen skal vi se nærmere på kombinasjonen av granulær syntese og eksperimentell bruk av romklang.

La oss fortsette med granulærsyntesepatchen som vi lagde i forrige BEAP-øvelse. I feltet helt til høyre kan du sette hvor ofte vi skal starte et nytt korn. Forandre det fra 5 til 50 millisekunder. Forandre kornlengden (*Duration*) til 0 og det tilfeldige avviket på lengden (*Random*) til 2%. Nå er lydkornene ganske spredte i tid.

La oss se nærmere på hvordan en romklang fungerer. I et akustisk rom reflekteres og forsinkes lyden i både tak og vegger. Dette kan enkelt fremstilles slik:



Om man fremstiller dette som en rekke forsinkelser over tid, ser det slik ut:



Først treffer det direkte signalet øret, deretter kommer en del tidlige refleksjoner, og så de sene refleksjonene som utgjør etterklangen (*reverbration*).

For å simulere dette bruker vi en stor mengde digitale forsinkelser av lyden. Vi skal ikke gå inn på detaljene i dette nå. Om du senere har lyst til å vite hvordan dette fungerer kan du åpne BEAP-patchen for å se hva den består av.

Legg til en romklang i patchen. Gå til *Effects*-kolonnenog velg *Reverb 2*-patchen.
Koble *Left* og *Right* utgang fra *Granular* til *Reverb 2 Input*. Koble *L* og *R* utgang fra romklangen til BEAP Stereo output patchen. Se skjermbilde av hele patchen nederst på siden om det er noe du lurer på med hensyn til signalgangen.

Still inn romklangen slik:

Parameteret *Mix* innstilt på 100% gjør at vi kun hører romklangen og ingenting av direktelyden. *Size* bestemmer størrelsen på rommet. En lav *Size*-verdi bruker korte forsinkelser og vil gi følelsen av et lite rom. Om du setter *Size* til 5.5 simulerer du et veldig lite rom. *Decay* varierer mellom 0 og 100. I denne BEAP patchen er 0 maks etterklang og 100 ingen etterklang, noe som for så vidt er litt lite intuitivt. Sett *Decay* verdien til 0 for maksimal etterklang. *Damping* legger inn et filter i romklangens signalgang for å simulere demping av etterklangen i et akustisk rom. Sett *Damping* til 5.4 kHz for litt dempning. *Diffusion* bestemmer hvor «utvasket» etterklangs vil oppfattes, lavere verdier vil beholde lyden av separate ekko i etterklangen, mens høyere verdier vil smøre lyden ut mer som i en god konsertsal. Sett den til 0.2. Ved å stille *Size*- og *Decay*-parametrene inn med såpass ekstreme verdier fungerer romklangen mer som en resonant effekt enn en simulering av et faktisk akustisk rom. I vår patch vil de korte lydkornene eksiterere denne resonante effekten.

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.

Hele patchen