Max/MSP exercises 2a

Ex.X

There are various types of message that Max understands.

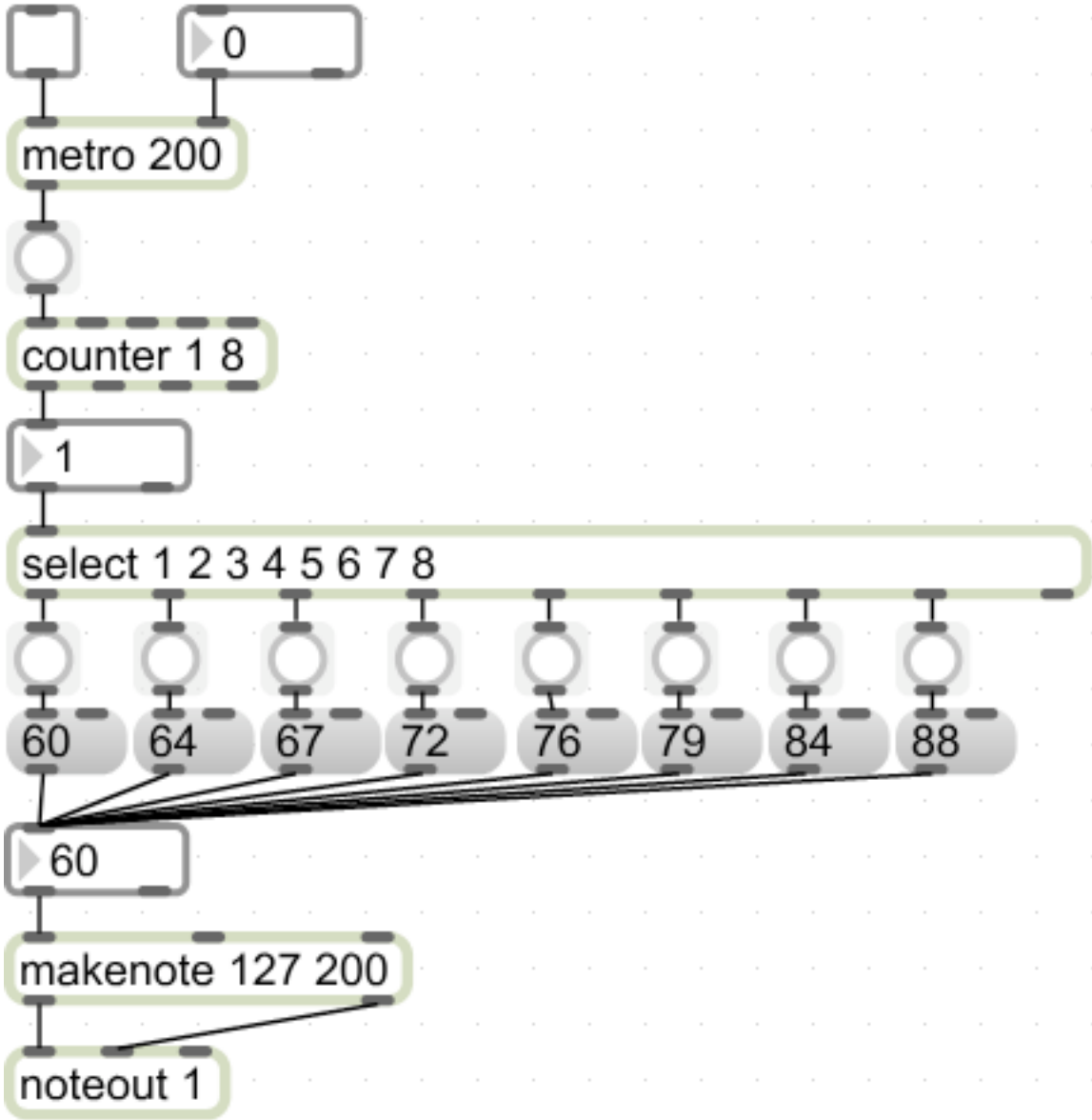
type	abbr	definition
integer/int	i or 0	a whole number (e.g. 1, 2, 3, 4
float	f or 0	a number with a decimal point (e.g. 1., 2.5, 3.141592654)
symbol	s	a word or series of letters (e.g. foo, dog, adsofh)
list	l	a series of numbers/symbols grouped as a single message (e.g. dog 1 3.5 seventeen, 1.2407584 17 x blob)
bang	b	a message that says 'do it', 'carry out that function' etc.

Max deals with these message types differently, so you will need to be aware of what each is, which objects use which data type, and how each is abbreviated in Max.

Ex. 1

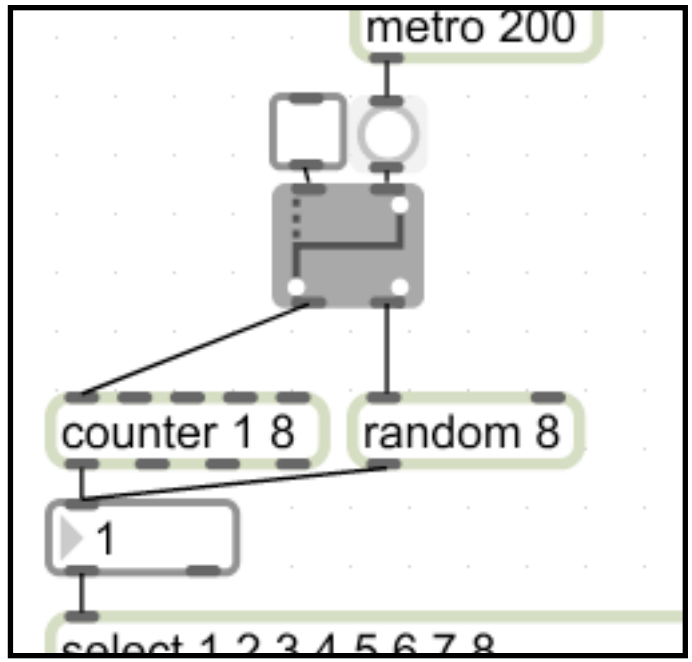
1. Copy this patch. It's very similar to the one we made last week but this one plays a sequence of notes on the piano synth setting of your computer's internal synth.

There is actually an awful lot you can do with just these simple objects in Max. Exploring them will help you significantly to get your head around the logic of building sequences of events over time.
We will also use this patch as a starting point to investigate some new objects.



Ex.2

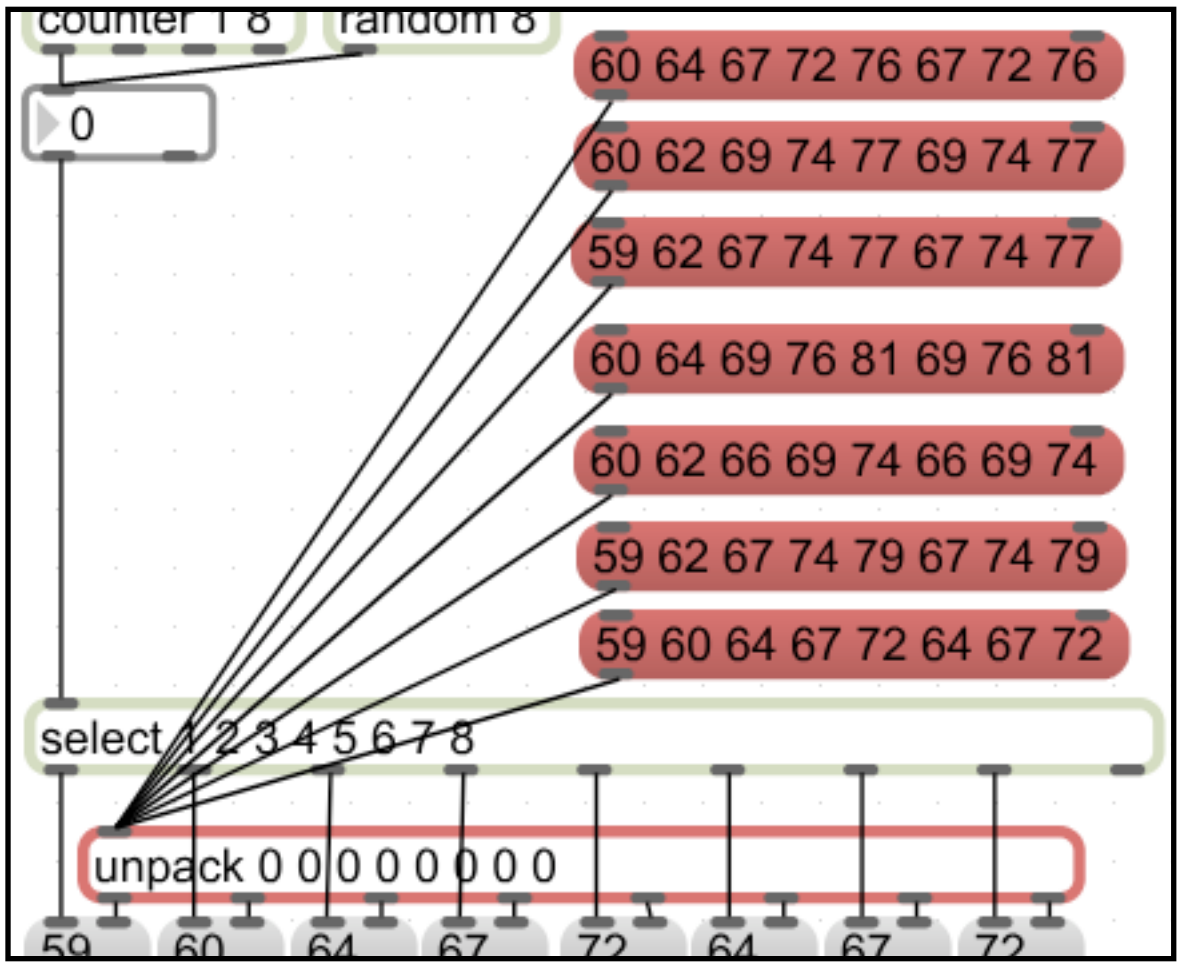
1. Modify the patch as shown. Lock the patch and switch on the [metro] again. Use the [toggle] to switch between [counter] and [random] 'modes'. What is the range of numbers that [random] is outputting? (allow yourself a few seconds to figure this out: it's important).



2. What do you need to do to the *output* of [random] to ensure that all numbers in the [select] object will be accessible? (Clue: it involves a [+] object and the simplest of maths)

Ex.3

1. Change the [random] object back to a Modify the patch as follows (the new objects are in red). Lock the patch and press the message boxes in turn.



To speed things up a bit, copy the following lists of numbers into each of the message boxes

1 60 64 67 72 76 67 72 76

2 60 62 69 74 77 69 74 77

3 59 62 67 74 77 67 74 77

4 60 64 69 76 81 69 76 81

5 60 62 66 69 74 66 69 74

6 59 62 67 74 79 67 74 79

7 59 60 64 67 72 64 67 72

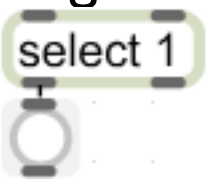
Make sure you understand how this is working, then move on.

Ex.3 (cont)

You might have recognised that when in [counter] mode these are the pitches for the first few bars of a Bach Prelude. We'll say that each series of eight notes is a 'bar'. Each bar (i.e. each sequence of notes in the [message box]es) needs to be played twice each to play the piece correctly.

At the moment you would have to click the [message box]es manually in order to get the note sequences to change in the right order and at the right time. But surely we can get Max to do this for us...

It turns out that we can, simply by using [select] and [counter] objects.

2. First we need to get Max to recognise the first beat of each 'bar', because this is when we need to trigger each [message box]. Insert the following in the patch so that the button will flash on the first beat of each sequence. 

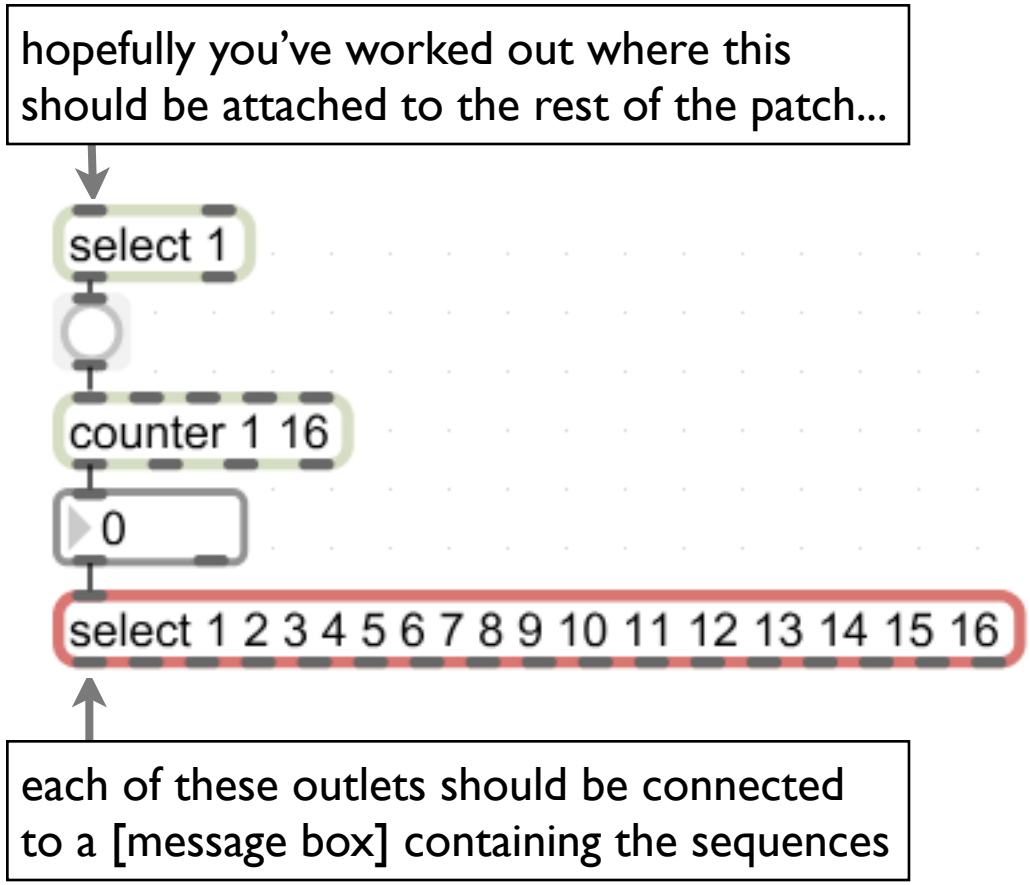
3. Now we need to register a count for each bar so that we can tell Max when to trigger each [message box]. Use 

for this.

Ex.3 (cont)

4. Now we can use a further [select] object to use these numbers to trigger the [message box]es in turn:

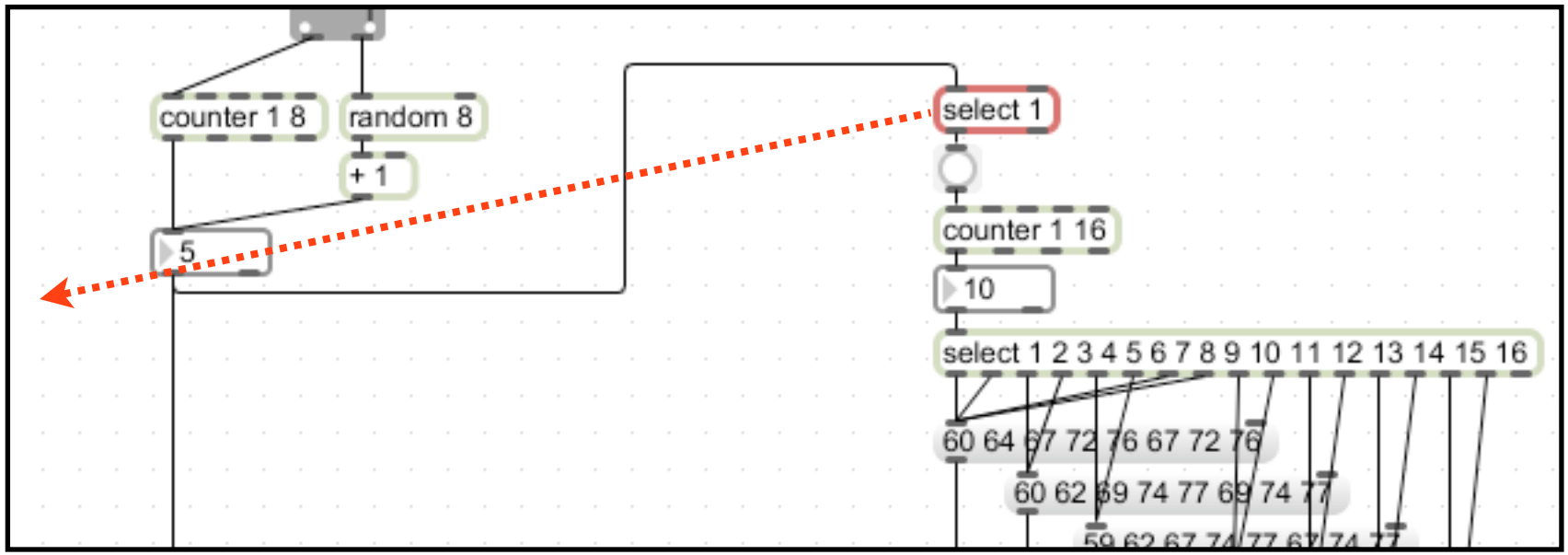
5. Referring to the lists on p5, the sequences should play in order as follows (notice there are (conveniently) 16 numbers below...):
1, 1, 2, 2, 3, 3, 1, 1, 4, 4, 5, 5, 6, 6, 7, 7



Hopefully this demonstrates that you can, via hierarchies of [select]s and [counter]s, articulate a larger-scale structure. There are more elegant ways, of course (check out the [coll] and [table] objects -- we'll look at these later) but these will do for now.

Ex.3 (cont)

6. Download the patch 'Prelude.maxpat' from the MUST1002 website. What happens if you move the [select 1] object as shown below (apart from making the patch ugly)? And why does the behaviour change as it does?



7. Add a means of getting the patch to trigger these sequences in random order.

Ex.4

1. Download the patch 'blue.maxpat' from the MUST1002 website. This is very similar to what you've seen in the last exercise. There are a couple of extra bits (what is the [%] object doing here, for example?) so look at the patch carefully and see if you can figure out how it works. Modify it if that helps.

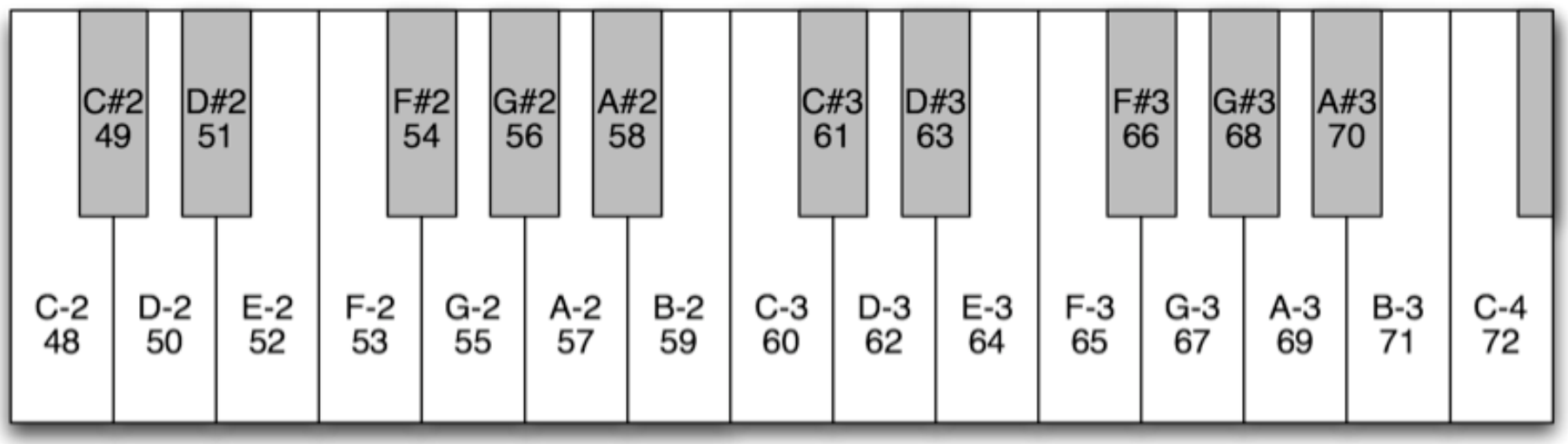
2. Download the patch 'prelude2.maxpat' from the MUST1002 website. Here we're back to the Bach Prelude, but this patch uses the [coll] object to store the note sequences. This is a *very* powerful object. Use the 'help' resources (press 'alt' and click on the object) to figure out how it works (we will look at this object in more detail later on).

Ex.5 (a challenge!)

Build your own: AUTO-ACCOMPANIST
This instrument is designed to play a concordant chord for each diatonic note from C-2 to C-3.

1. Download and open the patch 'aa.maxpat' from the MUST1002 website. Make sure you understand how it works.

2. Use a [select] object to recognise the MIDI note number for each diatonic note of the C-2 to C-3 scale. You will need to know the MIDI value for each of these notes (use the keyboard below to help you with this).



Ex.5 (cont)

3. Using three [message box]es, and referring to the exercise above, create message boxes that would play the following chords:

a) C-3, E-3, G-3, C-4; (again, you will need to work out the MIDI values for each of these)

b) D-3, F-3, G-3, B-3;

c) C-3, F-3, A-3, C-4.

Connect these to the [makenote] object (left-hand inlet) and test them.

4. Connect the patch so that:

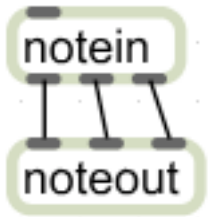
a) notes C-2, E-2, G-2 and C-3 trigger chord a);

b) notes D-2, F-2 and B-2 trigger chord b);

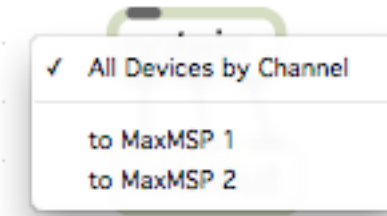
c) note A-2 triggers chord c).

Ex.6

1. Let's explore some objects in Max that will allow you to communicate with MIDI ins and outs on your computer. Copy this routine:



2. Lock the patch and double-click on the [note-in] object. Check that 'keystation' appears in the list (it doesn't here because I don't have an Oxygen8 at home) and choose it. If it isn't there, go to the Audio/MIDI setup for your computer and establish whether the keyboard is recognised there.



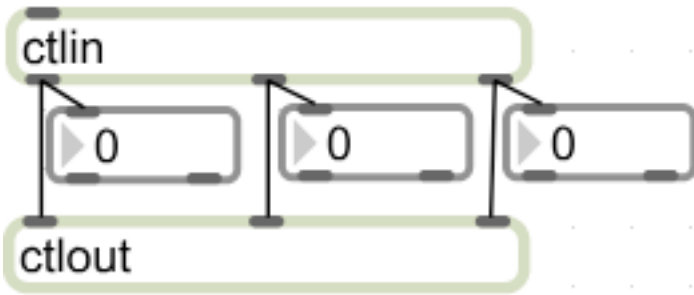
3. Assuming you have a MIDI keyboard, try playing notes on it. You should find that what you play is routed straight to the Mac's onboard synth (i.e. you'll get a piano sound).

4. Add number boxes to the outlets of [notein]. What do the numbers represent (if you're not sure, hover the cursor over the outlets)?

Ex.6 (cont)

5. Using some of the maths objects you've come across, modify the patch so that it is possible to...:
   - transpose each note on the keyboard by one octave;
   - invert the velocity so that hitting the keyboard harder makes it quieter (this is a bit tricky)

6. Copy the following into the same patch. Now move the modulation wheel while hitting notes. What happens?

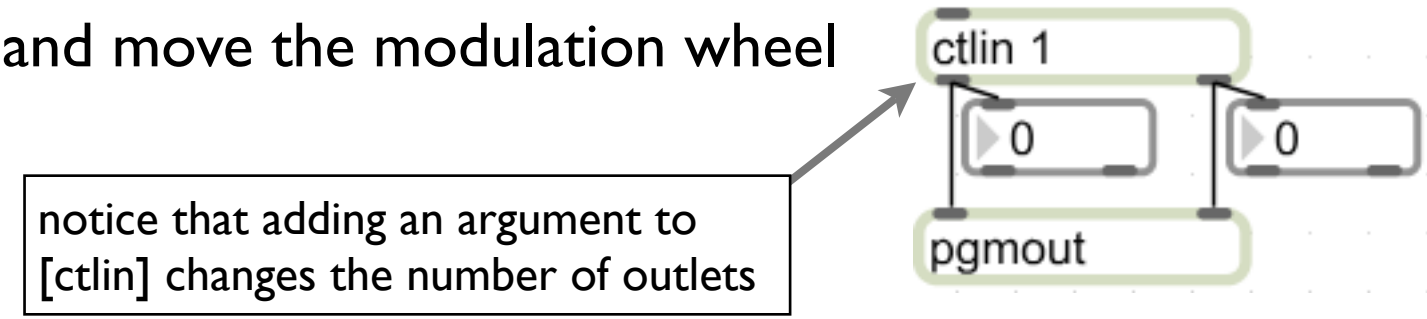


7. Hover your cursor over each of the outlets of the [ctlin] object and note what they are outputting. Move each of the controller pots on the keyboard. You will notice that the middle outlet registers the controller number for each of the pots (the modulation wheel is *always* controller number 1), while the left-hand outlet gives the value for that controller.

Ex.6 (cont)

8. Choose one of the controller pots on the keyboard and move it. Note the controller number. Now write that controller number as an argument into the [ctlin] object. You should now find that only the pot that you first moved will be recognised by that [ctlin] object. Make further [ctlin] objects with different arguments to be controlled by different pots.

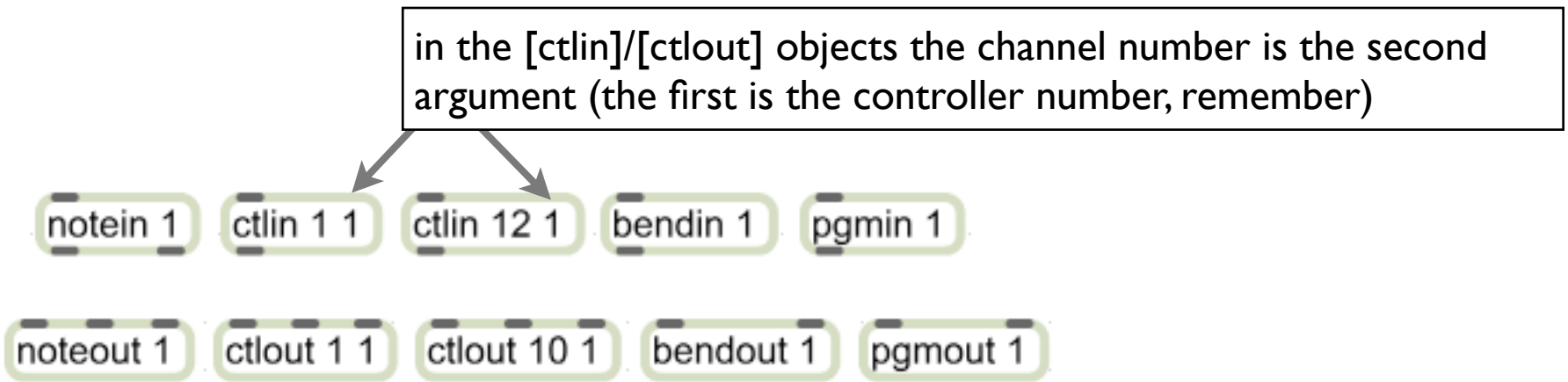
9. Copy the following and move the modulation wheel while hitting notes:



This should demonstrate, once again, that we are simply dealing with numbers here, and that these numbers can be mapped and remapped to perform different tasks.

Ex.6 (cont)

N.B. Different MIDI input and output objects ([notein]/[noteout], [bendin]/[bendout] etc) will all take a 'channel' argument. All of the following input objects will receive data from MIDI channel 1. Meanwhile all of the output channels will send data to the same instrument on the external MIDI synth. By default different channels on the output synth tend to be routed to different instruments. Channel 10 is usually dedicated to General Percussion sounds (though this can be changed), which is what we were using in Wk1.



Ex.7

1. Returning to the prelude.maxpat patch, try getting the modulation wheel to control the speed of your [metro] object.

You will notice that simply connecting a [ctlin] object to the right-hand inlet of [metro] produces unsatisfactory results. So insert a [scale] object into your patch, figure out how it works (use its 'help' patch) and it to scale the MIDI data to useful values for [metro] (e.g. 50 - 1000).

2. We could also get the instrument timbre to change each time [metro] triggers a note by sending new values to a [pgmout] object. Try this using

- a [counter] object (this would need to count from 1 to 127)
- a [random] object (this will need to work within a range of 1 to 127)