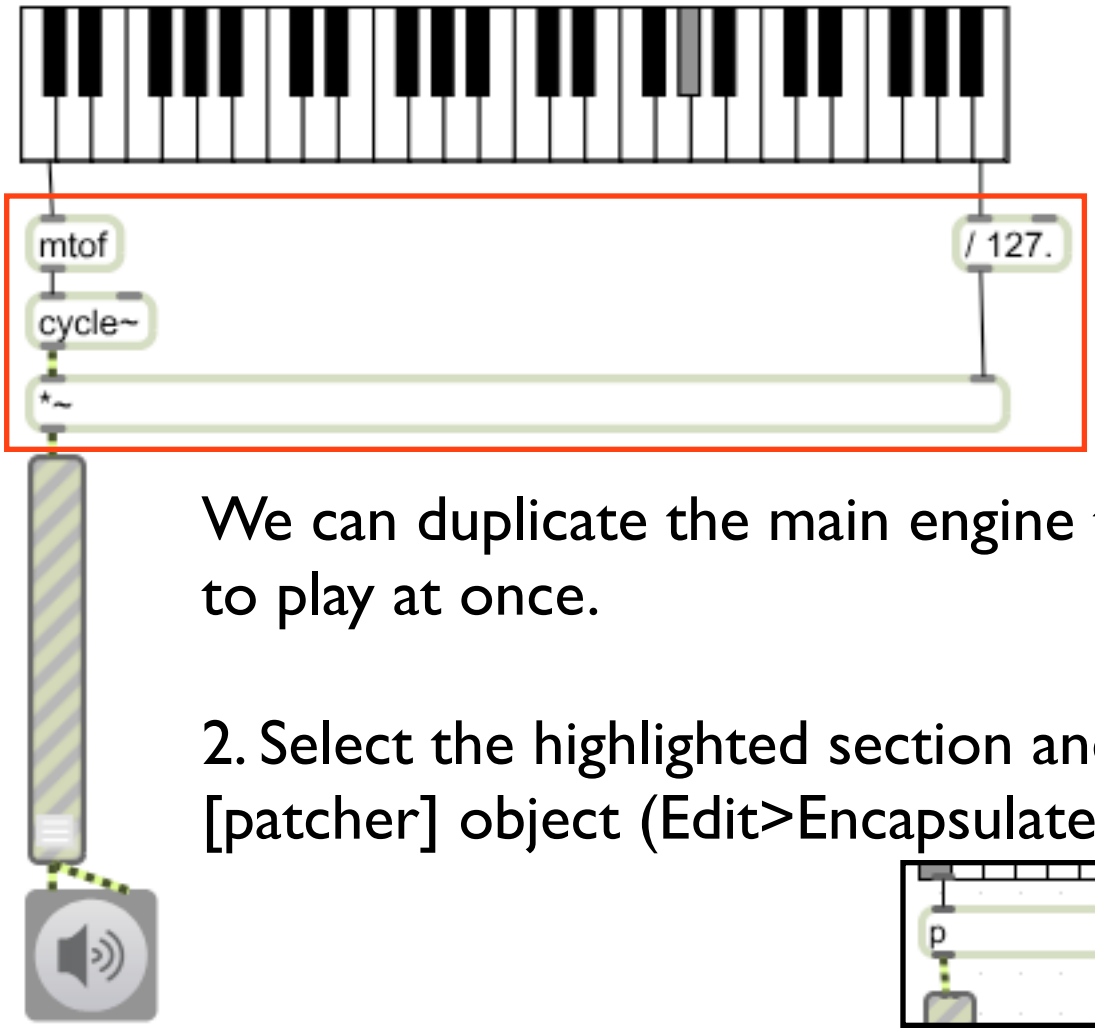
Max/MSP exercises 7a

Ex. 1

Thus far we've been working monophonically. So now we'll look at ways in which you might work with multiple voices.

1. First we'll make a very simple synthesis patch. Copy the following.



We can duplicate the main engine to permit more than one sound to play at once.

2. Select the highlighted section and Encapsulate it within a new [patcher] object (Edit>Encapsulate).

Interlude

If you have investigated the [kslider] object while exploring the MIDI capabilities of Max, you'll know that it has two modes: monophonic and polyphonic (you can choose between them in the Inspector Window).

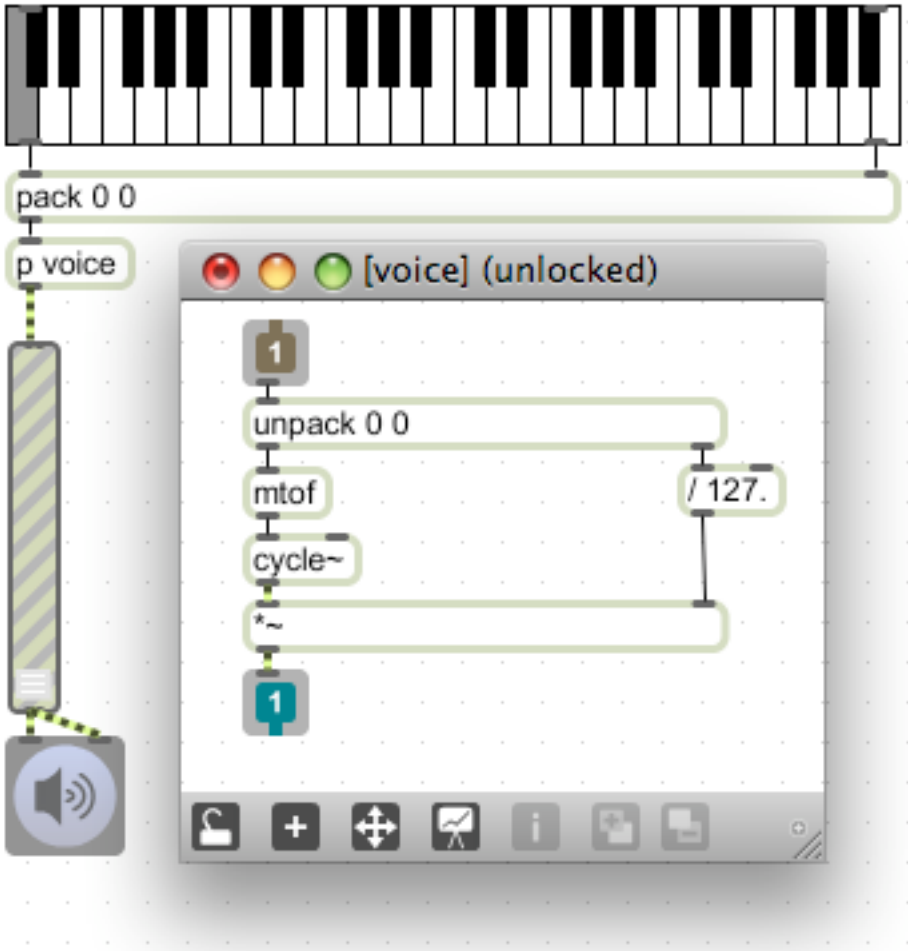
Polyphonic mode allows you not only to have several notes selected at once, but also to both trigger and *release* those notes (released notes send the note number and a velocity of 0).

We'll need our keyboard to be in polyphonic mode for this series of exercises.

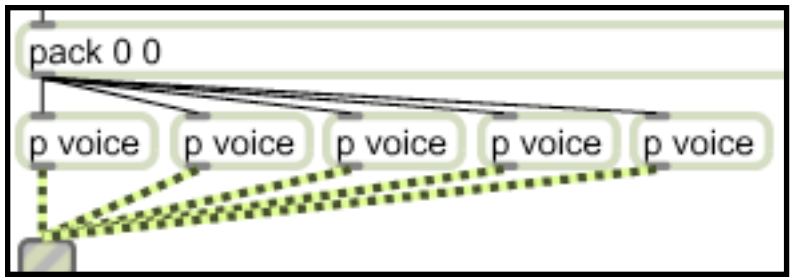
Ex.1 (cont)

Encapsulating the main engine of the patch makes things neater. But we could also arrange things to ensure efficiency when we start to add more voices.

3. Modify the patch as shown (i.e. add the [pack] object):



4. We can now duplicate the 'voice' patcher to give us several voices. What is wrong with the following means of doing so?

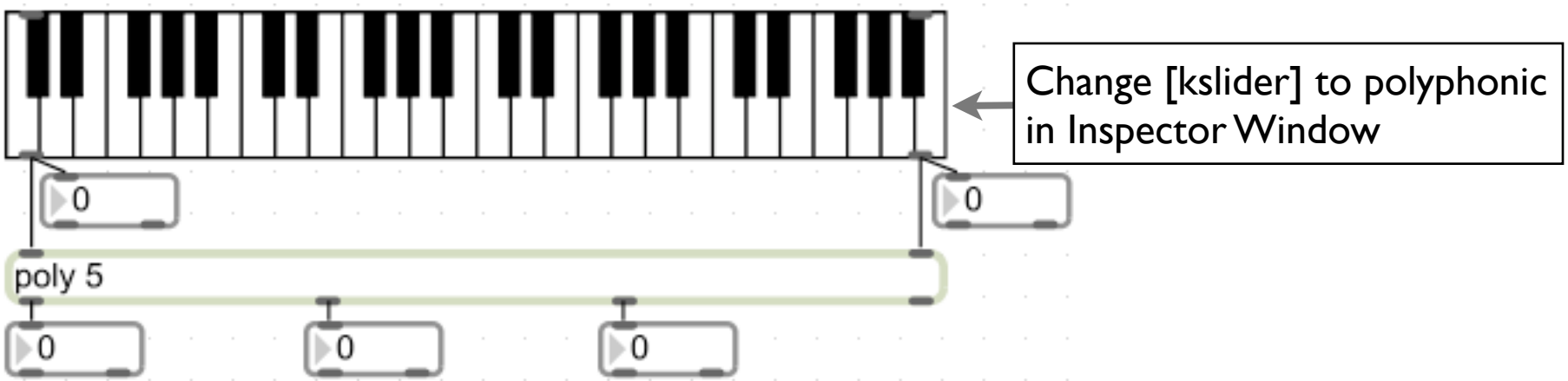


Ex.1 (cont)

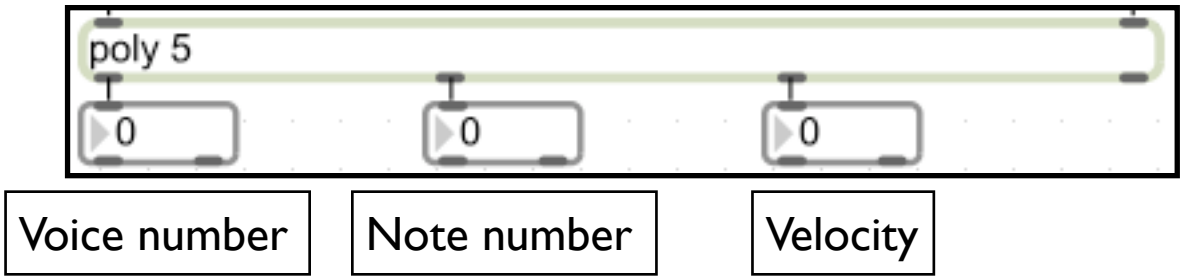
Answer: You just get a louder note, because several voices are playing the same note.

So simply adding more voices isn't enough to ensure useful polyphony. You need to be able to allocate individual notes to these voices from the keyboard (i.e. each note gets its own voice). For this we can use the [poly] object.

5. Copy the following:



Ex.1 (cont)



The outlets for [poly] are shown above. Notice that note number and velocity are through-put to outlets 2 & 3 of [poly]. Outlet 1 sends the voice number. This is what we're interested in at the moment.

6. Do the following in order:

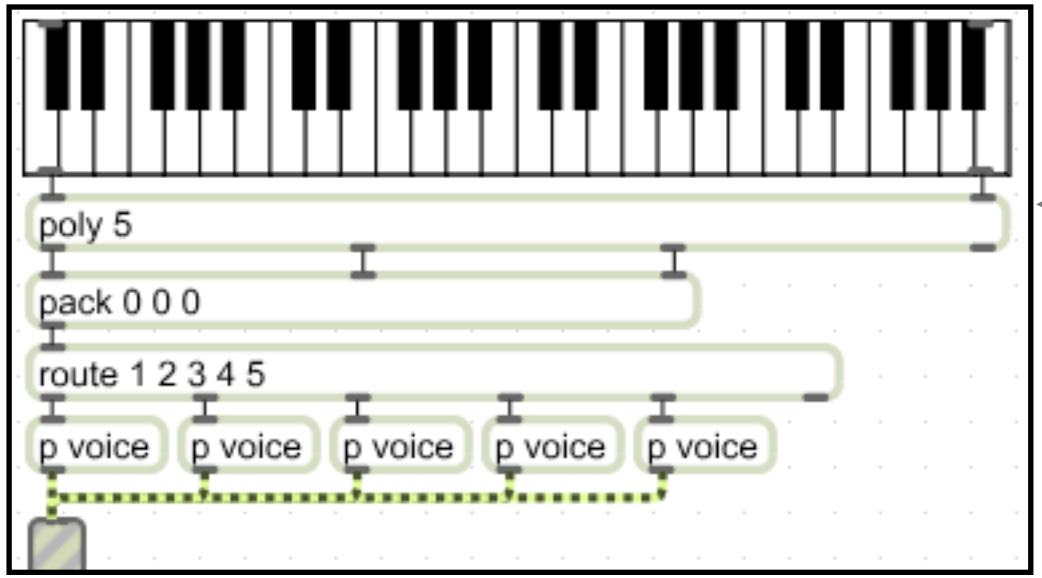
- Press a note. Outlet 1 sends a voice number of 1.
- Press another note (the original will stay selected). This is given voice number 2.
- Press another note. This is given voice number 3.
- Press the first note again ('releasing' it). This remembers voice number 1, allying it with a velocity of 0 (note-off).
- Press another note. This is given voice number 1 again (it has been reallocated to the newly-selected note).
- Press another note. This is given voice number 4.

[poly] therefore allocates note-ons to voices and, crucially, remembers which notes were allocated to which voices. When a note is released, it sends a note-off to the relevant voice number, freeing up the voice again.

Ex.1 (cont)

[poly]'s functionality makes it very useful for allocating note-ons and note-offs to the [patcher] voices we've just created.

7. Returning to the previous patch, modify it as follows:



Change [kslider] to polyphonic in Inspector Window

Here we use [poly] in conjunction with the [route] object (check the help file to see what route does) to allocate the note-ons and note-offs to each voice.

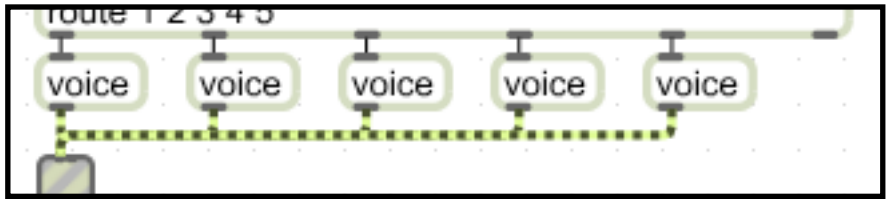
Ex. 2

So you've now created a polyphonic synth with 5 voices. It's a bit basic, though, so you might want to add some extra functionality to the voices. Problem is, you'd have to change every one of the [patcher] voices independently at the moment. When working with duplicated routines like this, there's an easier way forward...:

1. Make a folder on your desktop labelled myPolyPatch. Save your patch to it as 'myPolyPatch.maxpat'.

2. Double-click one of your voice [patcher]s. It will open in a separate window. Go to File>Save as and save the patch as 'voice.maxpat'. Make sure you save it to the same place as your host patch.

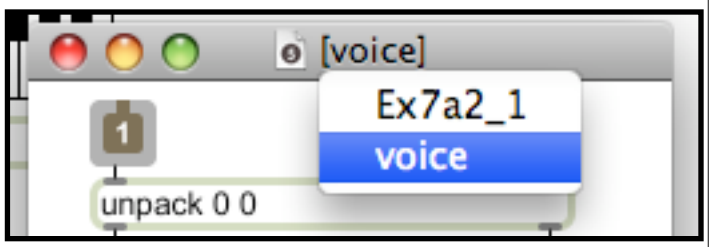
3. Close the 'voice' patch. Now, modify myPolyPatch as follows:



Ex. 2 (cont)

What you've done is to make an abstraction. This is a patch that you can call from another patch which behaves exactly like a normal Max object. The great advantage with this, particularly in the present situation, is that you can have multiple instances of the same abstraction in the same patch. When you modify the patch, all instances are updated. This means we can increase the functionality of all of the voices simultaneously.

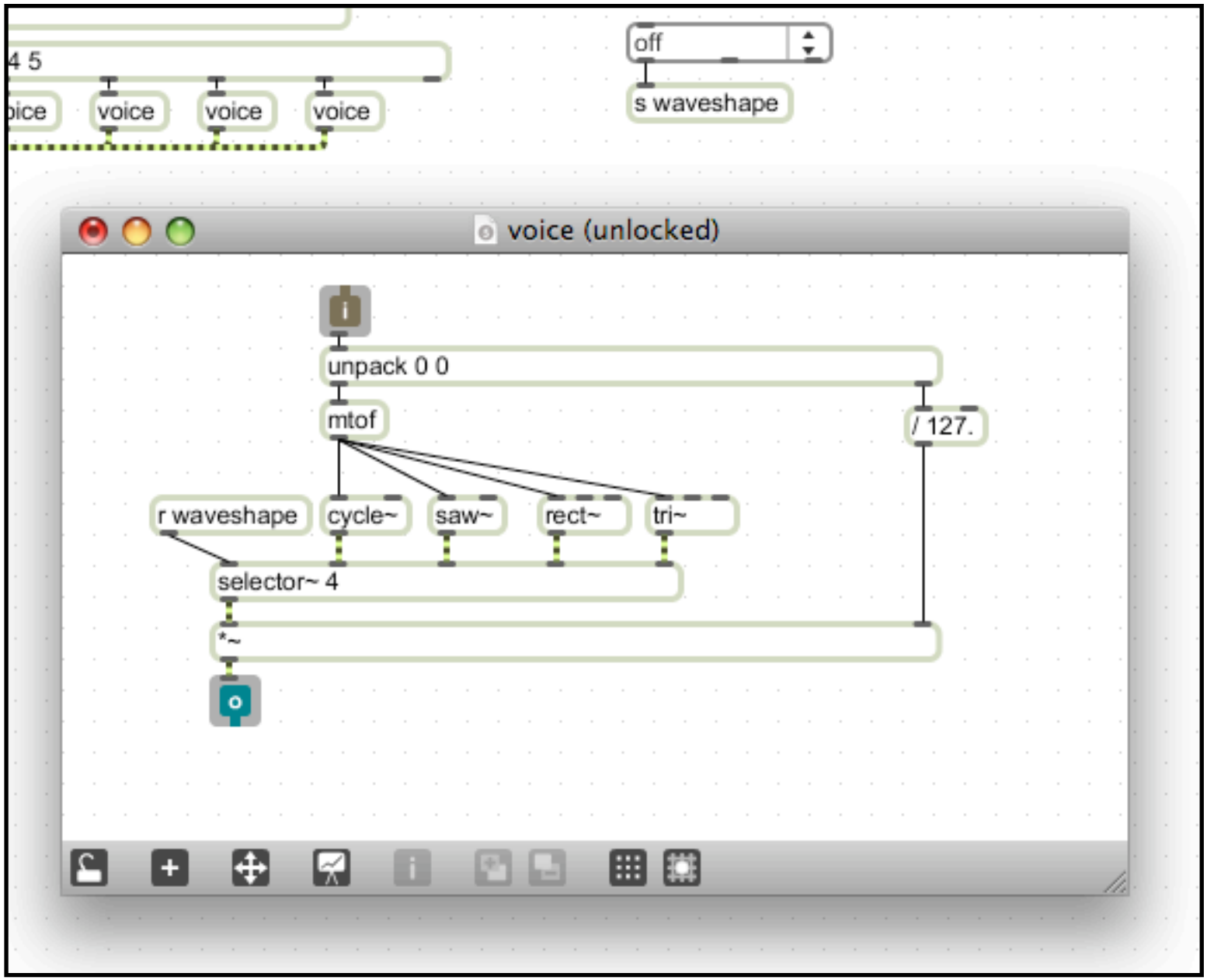
4. Open the 'voice' patch. You can do this in two ways:
• open the file from the Finder(Mac)/Explorer(Win).
• double-click the 'voice' abstraction in the patch, then option/alt-click the title bar of the window and select 'voice' from the drop-down menu that appears.



5. Modify the 'voice' patch so that it offers a variety of different oscillator types, e.g. sine, square, triangle, sawtooth waves (refer to Exercises5a p10 if you need reminding how to do this). Save the patch to update the abstraction.
How might you permit choosing between oscillator types from the main patch? (Hint: [send]/[receive] objects?)

Ex. 2 (cont)

Solution:



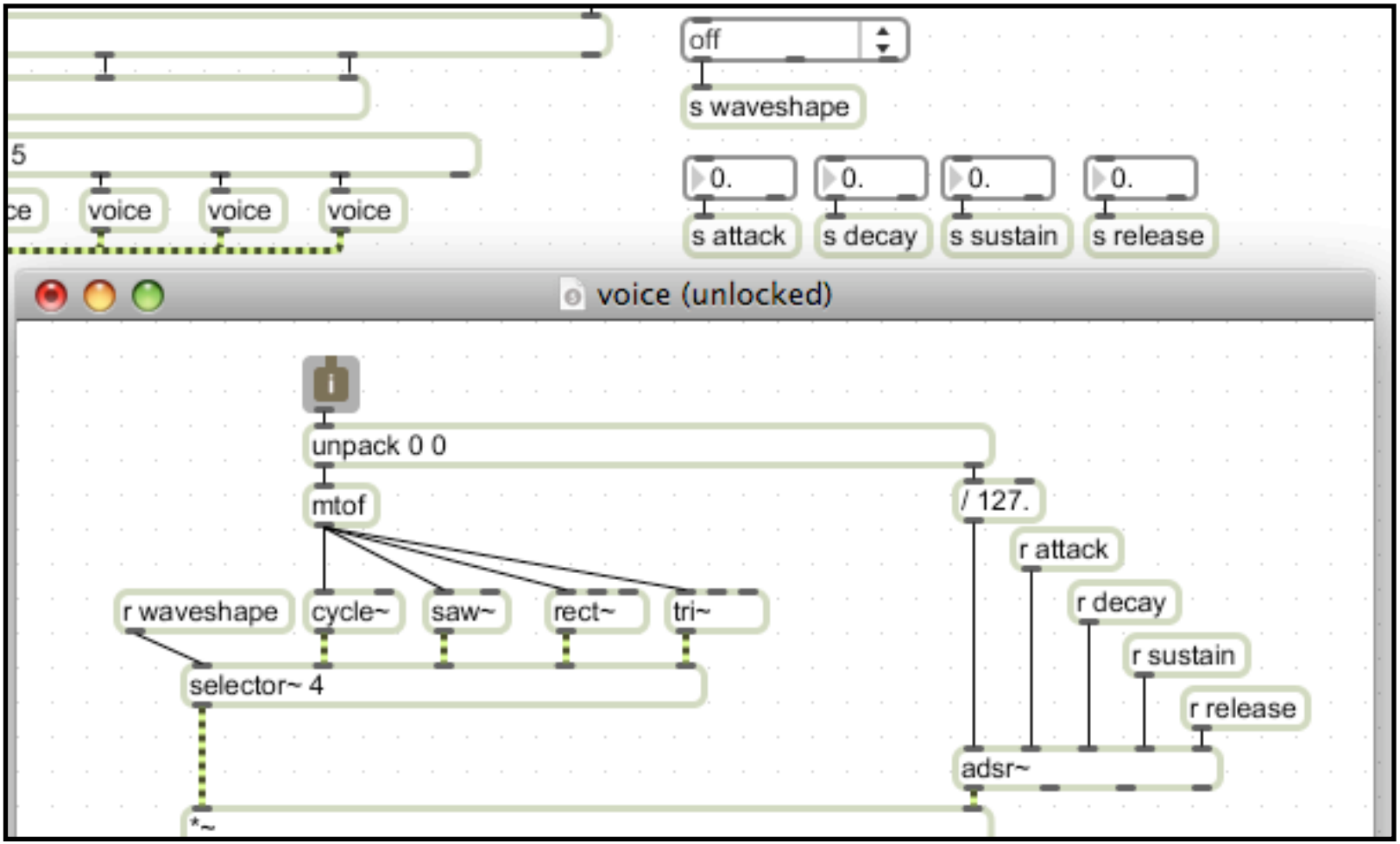
Ex. 2 (cont)

You could, of course, add other synthesis types as discussed in previous exercises, including additive, subtractive, AM, FM and wavetable. See Ex.5 for some extra info on this...

You might also want to include some enveloping functionality within the 'voice' patch. For this you can use the adsr~ object, which will once again be controlled from the main patch.

Ex. 2 (cont)

6 Modify the 'voice' and main patches as shown:



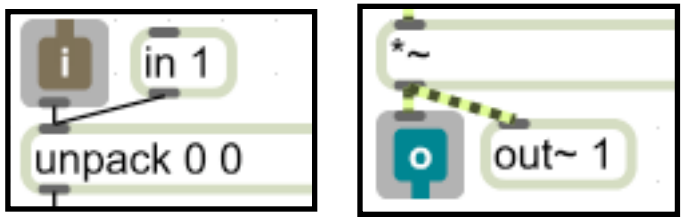
You might, of course, want to investigate prettier interface controls for the envelope parameters (e.g. sliders or dials). Remember also that while attack, decay and release parameters are all durations (ms), sustain refers to level, and is here a factor of the attack amplitude (how hard you hit the note to begin with), given as a number between 0 & 1 (e.g. 0.5 = 50% original volume).

Ex. 3

There's one more thing we can do to make this polyphony even more efficient. That is to use the [poly~] object (not to be confused with [poly]).

The [poly~] object provides a neat (though sometimes a bit confusing) combination of the [poly] object and an abstraction, combining voice allocation with multiple versions of a patch within a single object. For now, though, we'll use a very simple implementation that replaces our multiple 'voice' objects with a single one. First, we must make a few small alterations to our 'voice' abstraction.

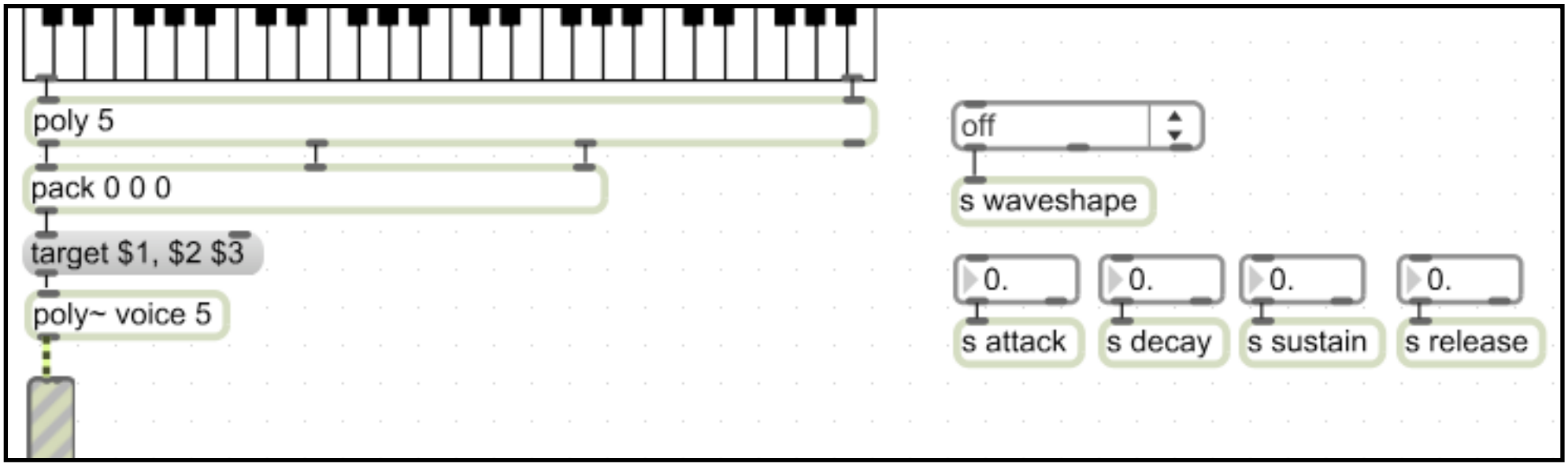
1. Modify the 'voice' abstraction by adding [in] and [out~] objects as shown:



[poly~] uses [in]/[in~] and [out]/[out~] (notice you need to differentiate between control/signal connections) objects to denote input and output rather than [inlet]s and [outlet]s.

Ex. 3 (cont)

2. Now modify the main patch as shown below:



Much remains the same except for the [poly~], which contains multiple instances of 'voice' within a single object (the argument to [poly~] declares the number of instances, so you could have many more than 5), and the 'target' message.

The target message chooses the [poly~] instance number to which any subsequent message is sent; note number and velocity are then sent to that instance. The $1, $2 and $3 parse the values from the [poly] object. Hence 'target 2, 60 127' would make instance 2 of [poly~] play middle C at 127 velocity.

Ex. 3 (cont)

3. Explore the [poly~] help file for more information on this object. In particular, find out:

- how to open the patcher windows for each instance of [poly~]
- another way of getting [poly~] to allocate notes
- how to mute a [poly~] instance

You can investigate an alternative way of making [poly~] allocate voices by looking at the MUST2004 [poly~] introduction at http://www.mti.dmu.ac.uk/must/MUST2004/supplemental/poly%7E.pdf.

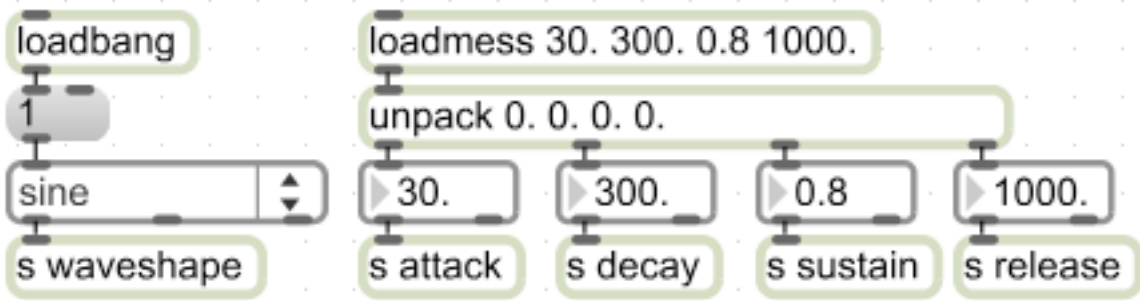
Ex. 3 (cont)

Finally, it is always a good idea to add a means of initiating a patch upon load to minimise the amount of work that a user must do to make the patch sound. In this example we have several objects which default to a 0 value if the patch is loaded from file.

4. Close the main patch (saving it if necessary).

5. Reopen the patch, noticing that the envelope and timbre parameters all begin at 0 or off.

6. Modify the patch as follows:

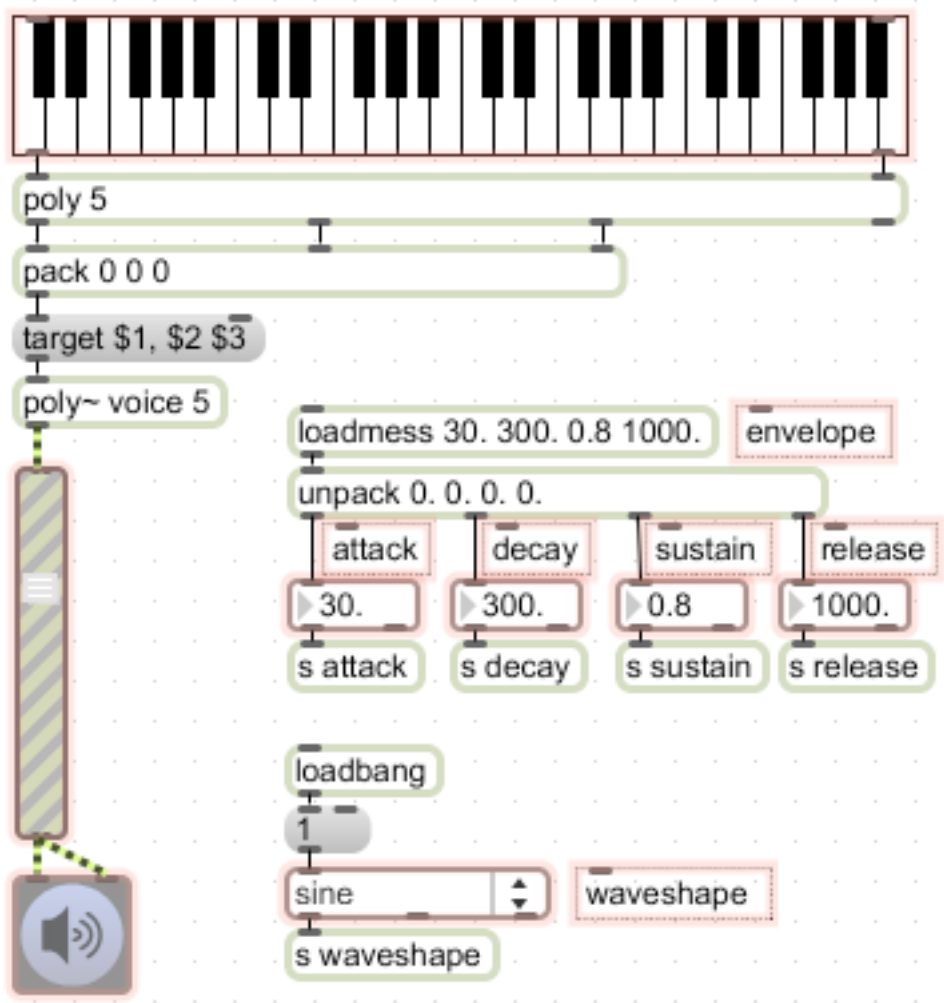


7. Save the patch, close it, and ropen. You should find that [loadbang] and [loadmess] have sent a bang/message upon startup which initiates the objects as required.

Ex. 4

Much of the contents of the patch it is unnecessary for us to see. Max offers a 'Presentation Mode' that enables us to show only certain objects that are essential for our interface.
It is also good practice to label parts of the patch to show what different controls do.

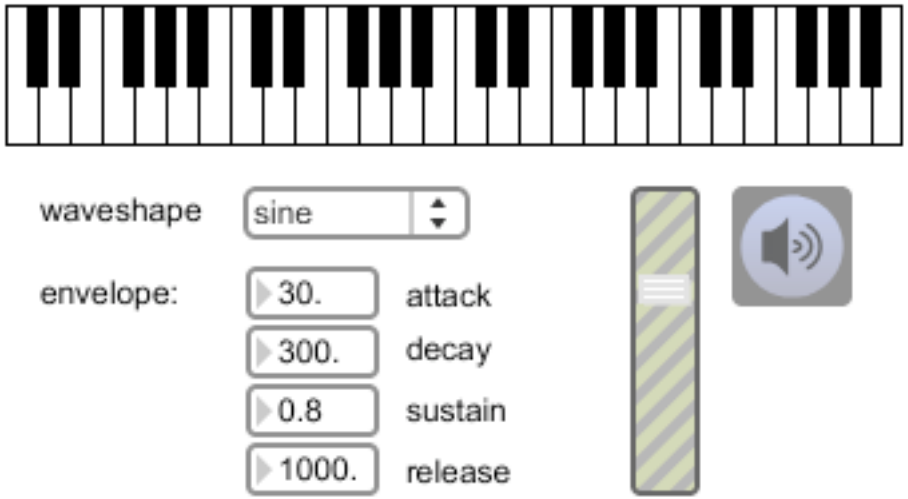
1. Label the patch as per the diagram shown. Then select the elements necessary for your interface and go to Object>Add to Presentation. When you have done so, a pink glow will surround the objects you've chosen.



Ex. 4 (cont)

2. Click on the Presentation Mode toggle at the bottom of the screen: 

3. You can now arrange the interface elements to your satisfaction:



4. To ensure that the patch opens up in Presentation Mode when you load it, go to the patch's Inspector Window (View>Patcher Inspector) and tick the 'Open in Presentation' checkbox.
Test this by saving the patch, then closing and opening it again.

Ex. 5

On p.11, it is suggested that you can integrate other synthesis types into an abstraction beyond the simple ones used for demonstration. One of those is additive. The examples we've used previously to introduce additive synthesis involved the use of [function] editors. It is necessary to be able to control the [function] editors in the 'voice' patch from the main patch.

1. Explore the patch savingFunctionData.maxpat on the MUST1002 website. Try using this to implement a polyphonic version of the additive synthesis model we explored in Exercises5a (this is quite a challenge, but if you master it you'll certainly prove yourself to be getting to grips with MaxMSP).