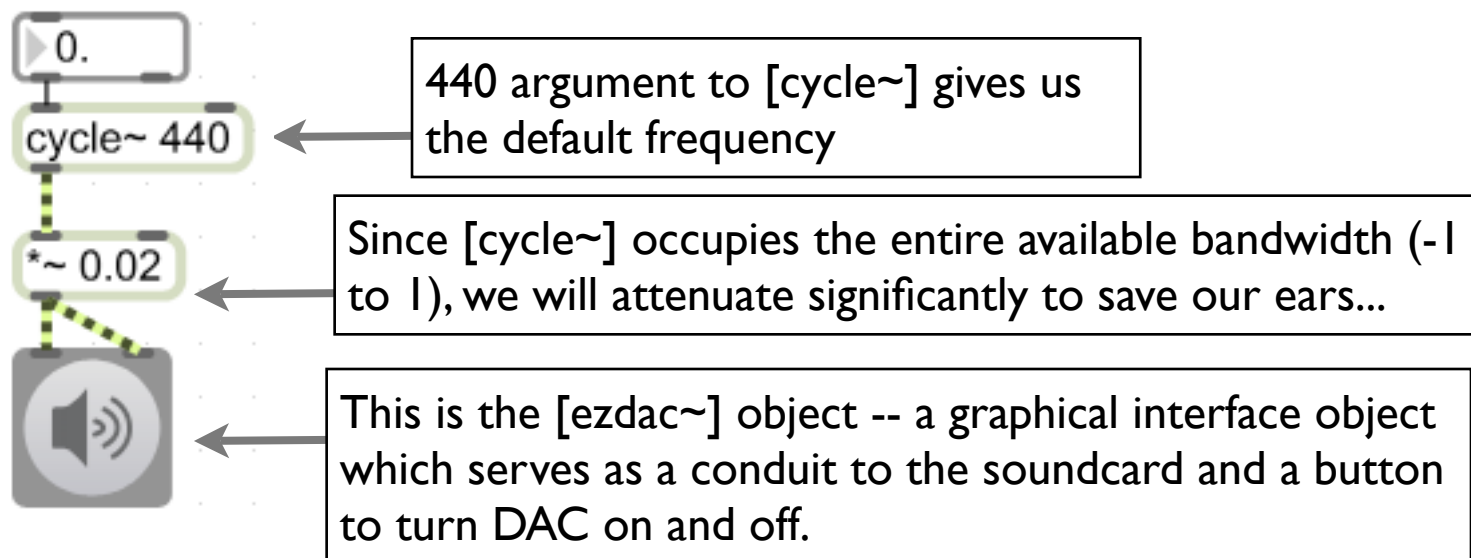


Max/MSP exercises 5a

Ex. 1

Now that we've looked at some basic sampling objects in MSP, we should check out some means of doing synthesis.

1. Copy the following:



This gives us a sine tone oscillator with a starting pitch of 440Hz.

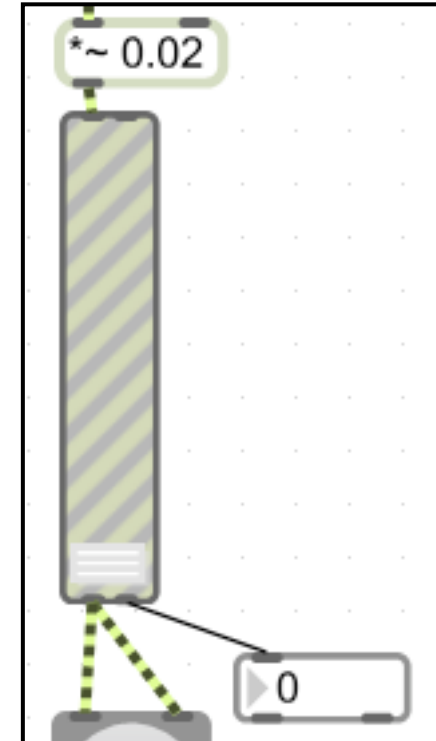
2. Mess about with the `[float]` number box. You'll notice that this controls the frequency of `[cycle~]`.

Ex. I (cont)

A useful object to know in MSP is the [gain~] object which is designed to control amplitude.

3. Modify the previous patch as per the following:

The [gain~] object performs exactly the same function as the [*~] object in the previous patch (i.e. multiplying each sample that it throughputs in order to amplify or attenuate it). It could in theory replace our [*~] object in the original patch. It remains for the moment as we'll use it for something else later, but it also continues to serve as a useful 'limiter' for the signal, even if we crank up the [gain~] fader.



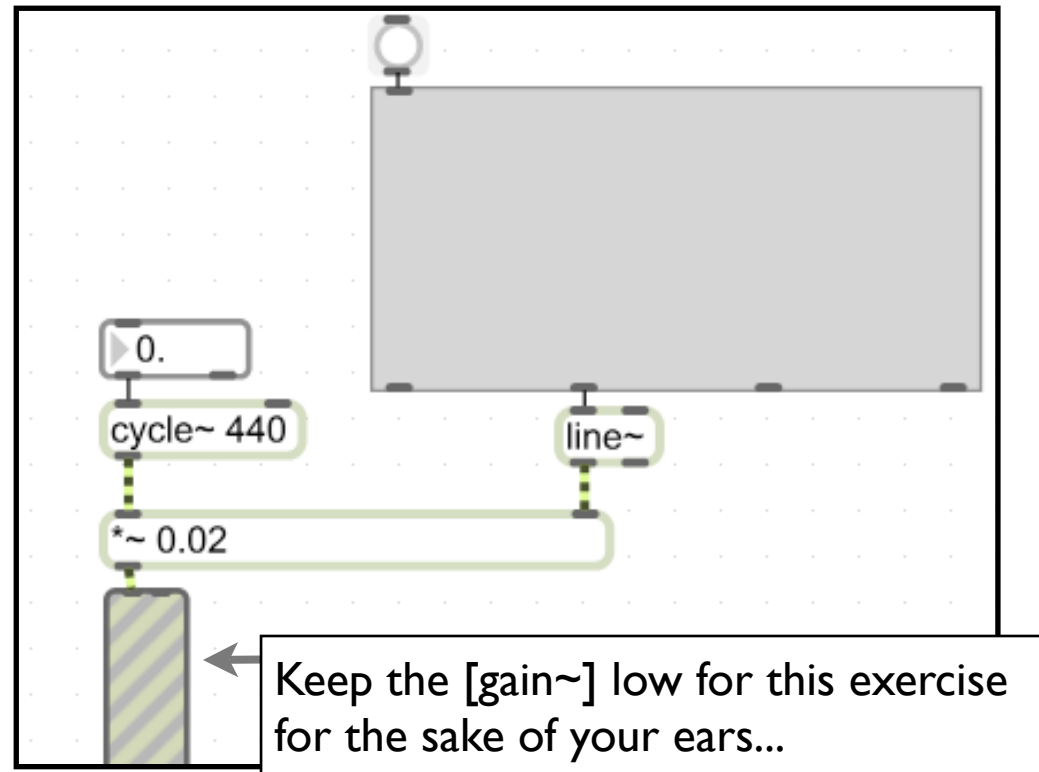
4. Explore the [gain~] fader. Notice the output to the [number] box. Check the [gain~] help file to see why its range is 0 to 158.

5. Using the [*~] object that we've left in the patch, add an envelope generating function to the patch (check Ex.4 in the Exercises 4a pdf from last time).

Ex. I (cont)

Solution to previous exercise:

You could add a similar engine to control pitch in the same way.



6. Try to figure out how to do this (solution overleaf).

Things to know/think about:

- [cycle~] will take a signal input (so you can connect a [line~] to it)
- in order to work within the frequency domain, you'll need to change the 'Lo and Hi Display Range' of your [function] object in the inspector window.
- you'll want to trigger both envelopes at the same time.

Ex. I (cont)

Solution to previous exercise:

The image shows a Pure Data patch on the left and a function inspector window on the right. The patch consists of a signal source (a sine wave) connected to a 'line~' object. This is followed by a 'cycle~ 440' object, then a '*~ 0.02' object, and finally a 'line~' object. The output of the patch is connected to a 'line~' object. The function inspector window is titled 'function inspector' and has tabs for 'All', 'Appearance', and 'Behavior'. The 'All' tab is selected, showing a list of settings and their values. A callout box points to the 'Lo and Hi Display Range (...)' setting, which is set to '100. 1000.'.

Setting	Value
Background Color	
Border Color	
Line Color	
Point Color	
Sustain Indication Color	
Text Color	
Annotation	
Hint	
Scripting Name	
Allows Adding Points by Cl...	<input checked="" type="checkbox"/>
Allows Changing Sustain b...	<input checked="" type="checkbox"/>
Allows Moving Points by C...	<input checked="" type="checkbox"/>
Auto Sustain	<input type="checkbox"/>
Display Legend at Top of ...	<input checked="" type="checkbox"/>
Hi Domain Display Value (...)	1000.
Lo and Hi Display Range (...)	100. 1000.
Output mode	Normal

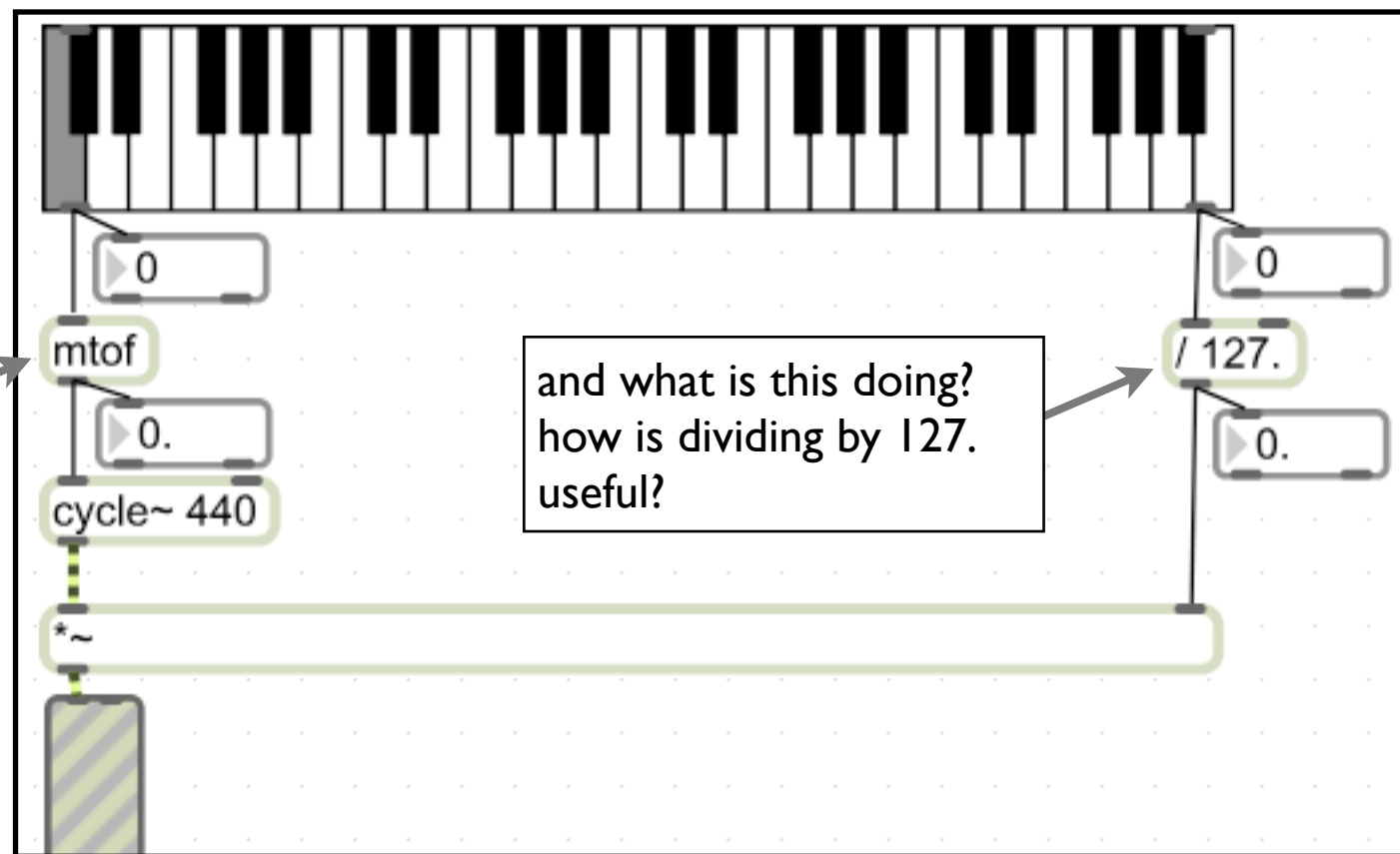
our 'note' can be anywhere between 100 and 1000Hz (you might choose different limits to your range)

Ex.2

Instead of a Hz being used to control our pitch, perhaps we'd like to use a keyboard.

1. Copy the following:

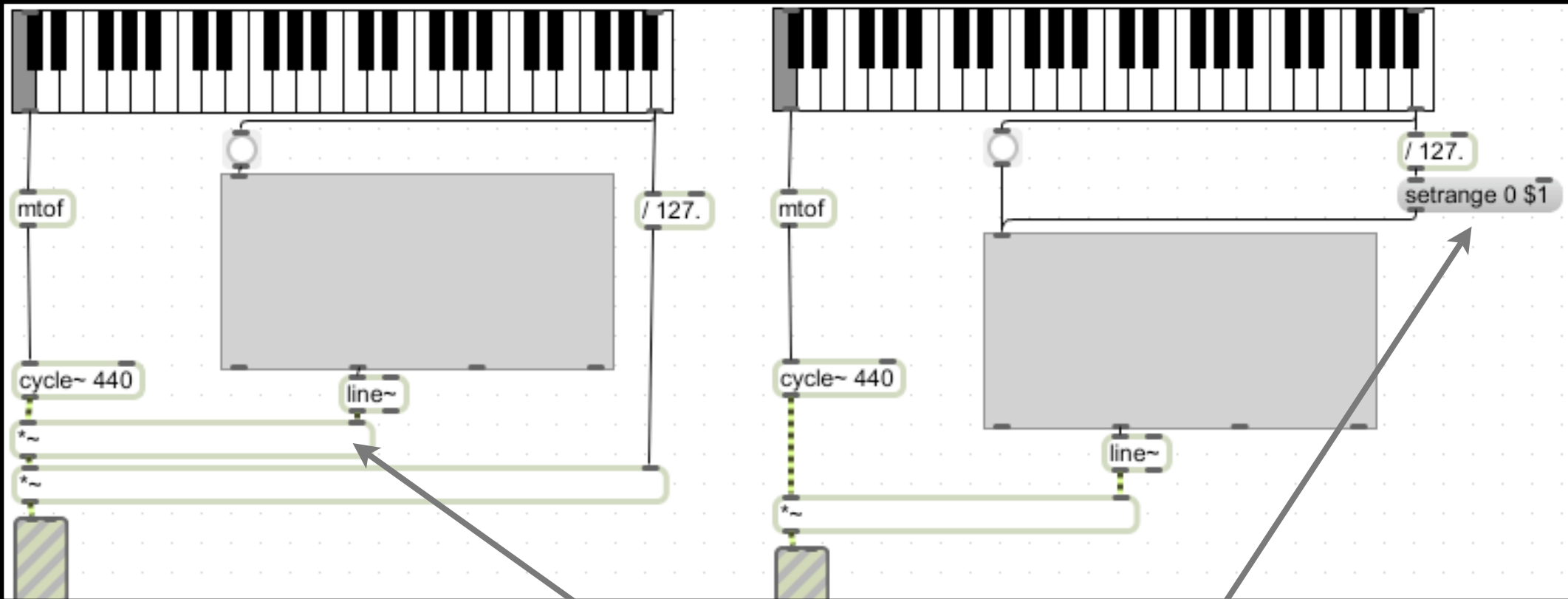
what does [mtof] do?
check it's 'help' window
to find out.



2. Have a think about how you might modify this patch to give an envelope whose peak amplitude is determined by how hard you hit the MIDI keyboard.

Ex.2 (cont)

There are two possible solutions as follows:



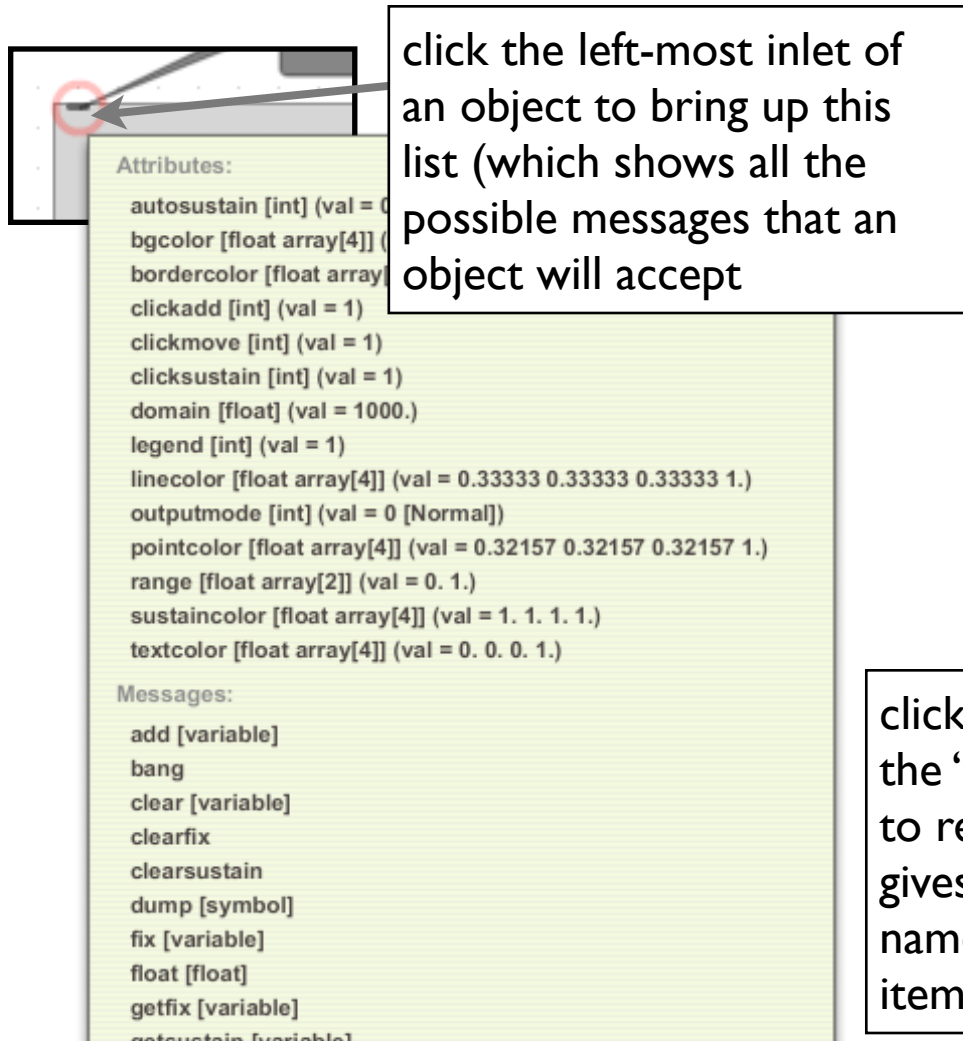
Which is more efficient?

note the two `[*~]` objects which, with the `[gain~]` object, make three objects dealing with amplitude.

remember that the message object throughputs data

Interlude

The 'setrange' message in the second solution on the last page is useful in enabling us to change the 'attributes' of an object without having to use the 'inspector window'. To find out what attributes you can change for an object, you can do either of the following:



A screenshot of a Pure Data object's attribute list. The list is titled 'Attributes:' and contains various attributes with their current values. Below the attributes is a 'Messages:' section with a list of messages. A red circle highlights the left-most inlet of the object, and an arrow points from a text box to it.

Attributes:

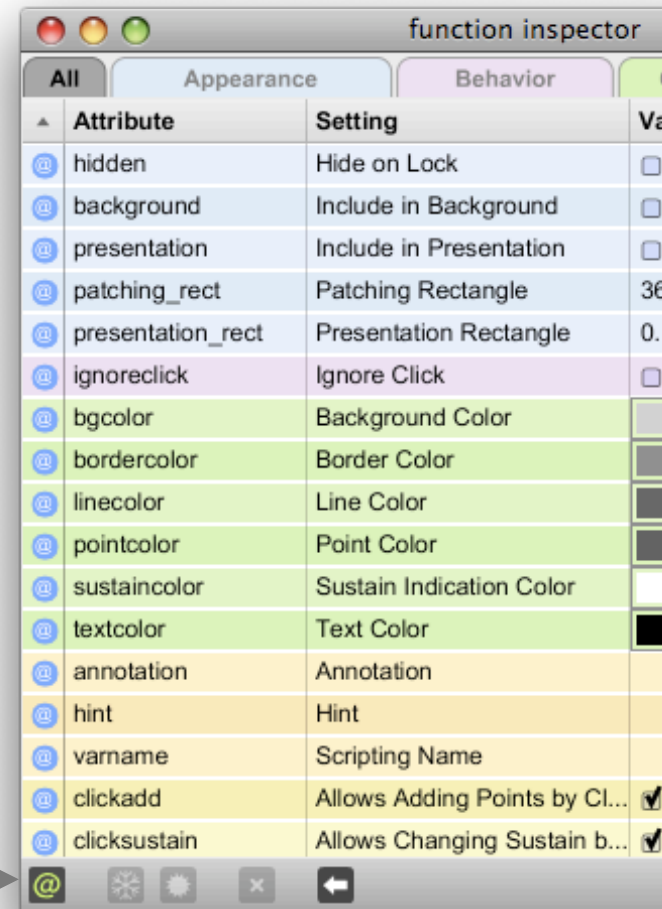
- autosustain [int] (val = 0)
- bgcolor [float array[4]] (val = 0. 0. 0. 0.)
- bordercolor [float array[4]] (val = 0. 0. 0. 0.)
- clickadd [int] (val = 1)
- clickmove [int] (val = 1)
- clicksustain [int] (val = 1)
- domain [float] (val = 1000.)
- legend [int] (val = 1)
- linecolor [float array[4]] (val = 0.33333 0.33333 0.33333 1.)
- outputmode [int] (val = 0 [Normal])
- pointcolor [float array[4]] (val = 0.32157 0.32157 0.32157 1.)
- range [float array[2]] (val = 0. 1.)
- sustaincolor [float array[4]] (val = 1. 1. 1. 1.)
- textcolor [float array[4]] (val = 0. 0. 0. 1.)

Messages:

- add [variable]
- bang
- clear [variable]
- clearfix
- clearsustain
- dump [symbol]
- fix [variable]
- float [float]
- getfix [variable]
- getsustain [variable]

click the left-most inlet of an object to bring up this list (which shows all the possible messages that an object will accept)

click the @ button in the 'inspector window' to reveal a list which gives you attribute names for the various items in the inspector



A screenshot of the 'function inspector' window. The window has tabs for 'All', 'Appearance', and 'Behavior'. The 'All' tab is selected, showing a table of attributes and settings. A blue '@' button is highlighted in the bottom left corner of the window, with an arrow pointing from a text box to it.

Attribute	Setting	Value
hidden	Hide on Lock	<input type="checkbox"/>
background	Include in Background	<input type="checkbox"/>
presentation	Include in Presentation	<input type="checkbox"/>
patching_rect	Patching Rectangle	36
presentation_rect	Presentation Rectangle	0.
ignoreclick	Ignore Click	<input type="checkbox"/>
bgcolor	Background Color	
bordercolor	Border Color	
linecolor	Line Color	
pointcolor	Point Color	
sustaincolor	Sustain Indication Color	
textcolor	Text Color	
annotation	Annotation	
hint	Hint	
varname	Scripting Name	
clickadd	Allows Adding Points by Cl...	<input checked="" type="checkbox"/>
clicksustain	Allows Changing Sustain b...	<input checked="" type="checkbox"/>

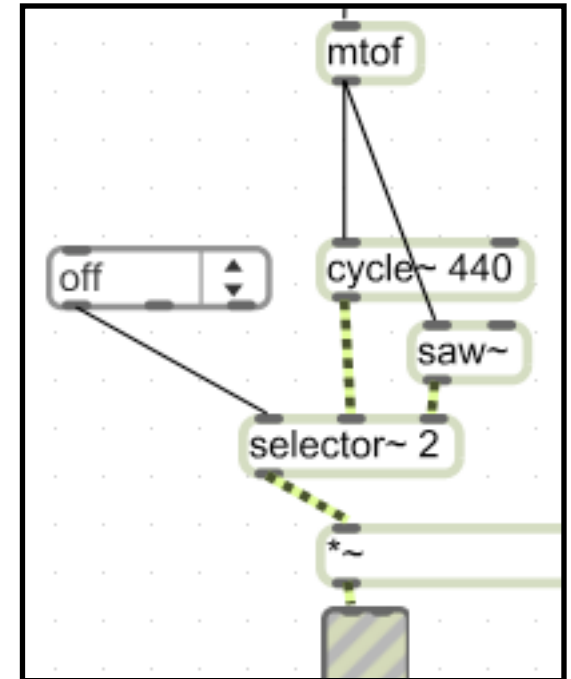
Ex.2 (cont)

So far we have a monophonic synth but with only one tone.

3. Add the following to your patch:
(You'll need to add 'off, sine, saw' to the [umenu]'s 'inspector window')

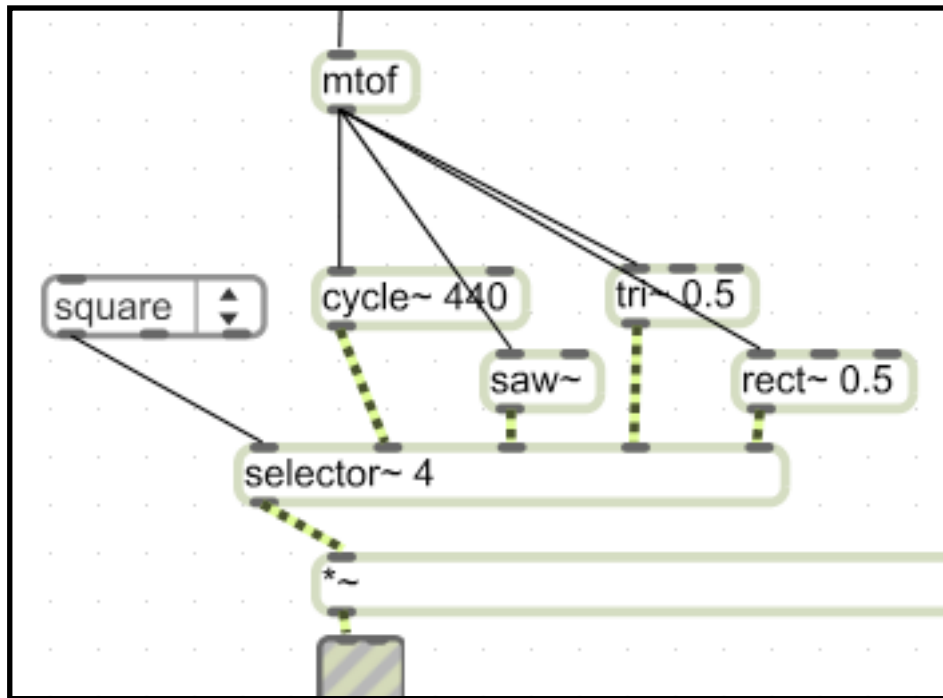
4. Using the same principle, add the following triangle and square wave oscillators and modify the [selector~] and [umenu] objects accordingly:

- [tri~] (give a 0.5 argument)
- [rect~] (give a 0.5 argument)



Ex.2 (cont)

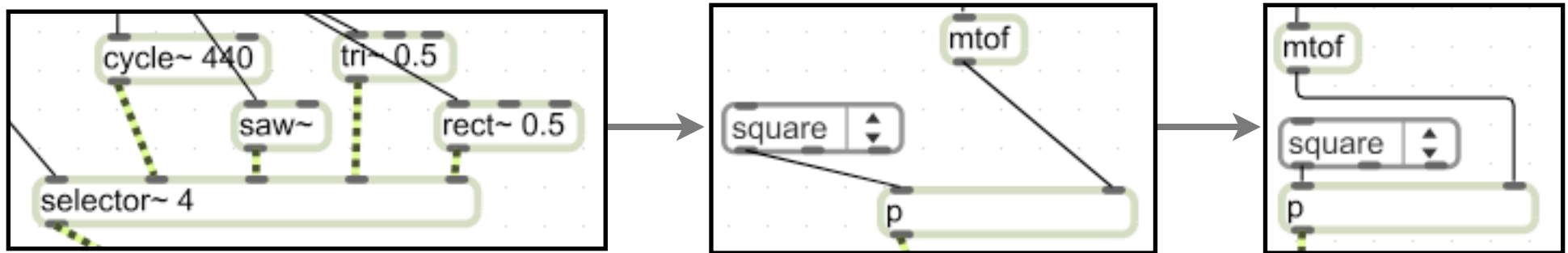
Solution:



Ex.3

Using this model we can make a basic additive synth. But first, to remove a bit of clutter:

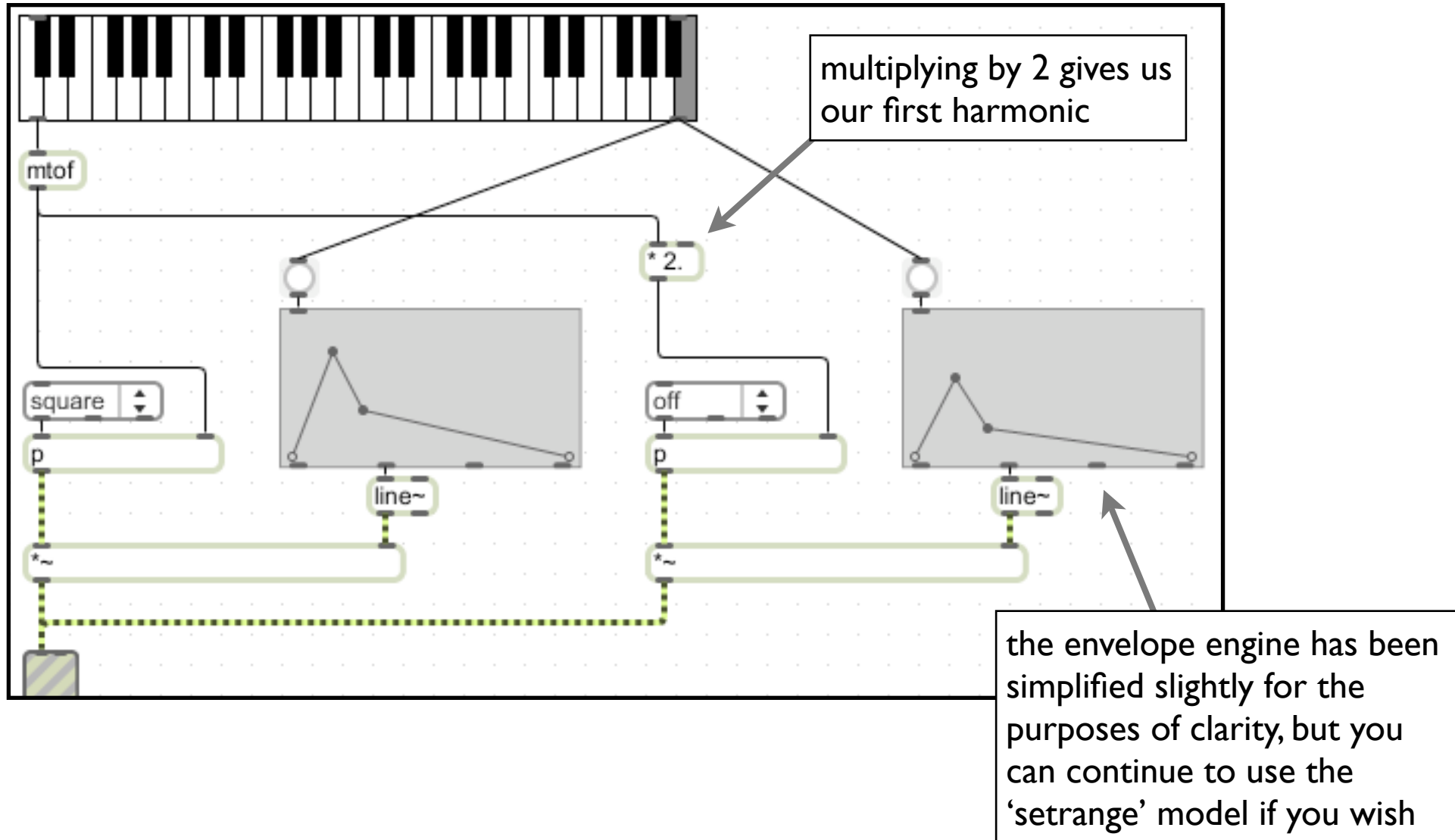
1. Select the following items (note that the [umenu] is excluded), then go to Edit>Encapsulate to put them in their own sub-patcher.



Notice how useful this can be for tidying things up. We've done this because we want to duplicate the set of oscillators we've just made.

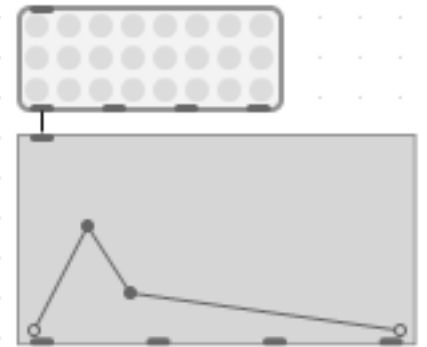
Ex.3 (cont)

2. Now duplicate the oscillator and envelope parts of the patch as follows, noting the highlighted changes:



Ex.3 (cont)

3. Use this model to create 4 more partials (so you have a fundamental and 5 harmonics) which continue to adhere to the harmonic series (ie. x_3 , x_4 , x_5 , x_6).
4. Investigate the [preset] object. This allows you to save and recall the state of any object to which it is attached. Connect its left-hand outlet to any graphical interface object whose state you want to save.



Open the patch Ex5a3_4 to see a solution.