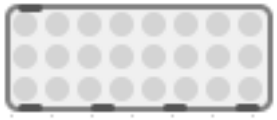
Useful Objects

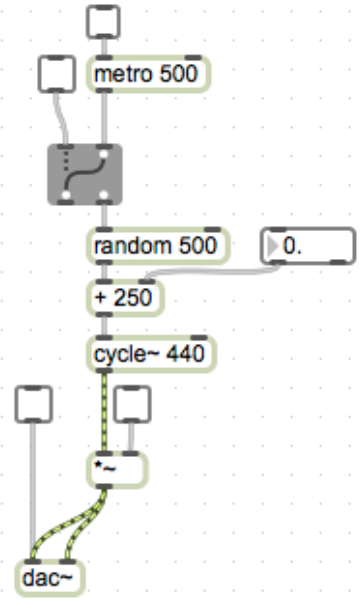
Electronic Music II

Spring 2013

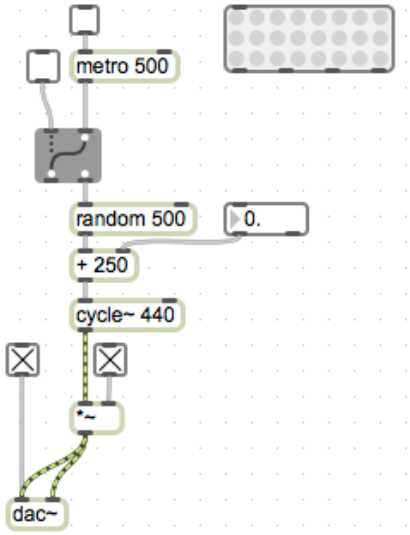
1. `preset`
   a. The `preset` object is used to store and recreate the settings of patcher objects such as toggles, sliders, number boxes, etc.
   b. Create a blank object and type "preset". The resulting object will resemble the following:



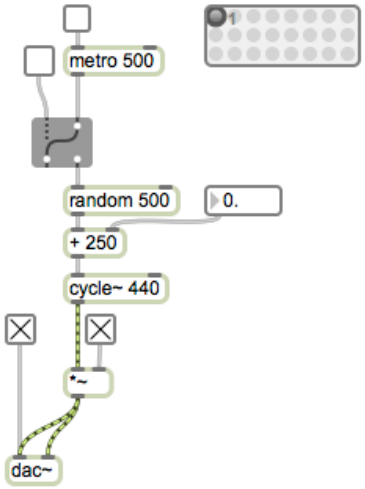
   c. Each circle (termed "bubbles" by Max) can store the states of each applicable object in the patch. Consider the following setup:



   d. There are several ways in which this patch could operate:
      i. Produce a steady sine tone
      ii. Produce a sine tone at a frequency determined randomly every 500 ms.
      iii. Produce a random sine tone between x and $499+x$ Hz, determined every 500 ms.
   e. To coordinate the many objects that need to be set for each way of operating, I can use presets.
   f. First, make the necessary changes in the settings of each object. For example, to reproduce mode i. listed above, only the lower two toggle objects need to be checked.

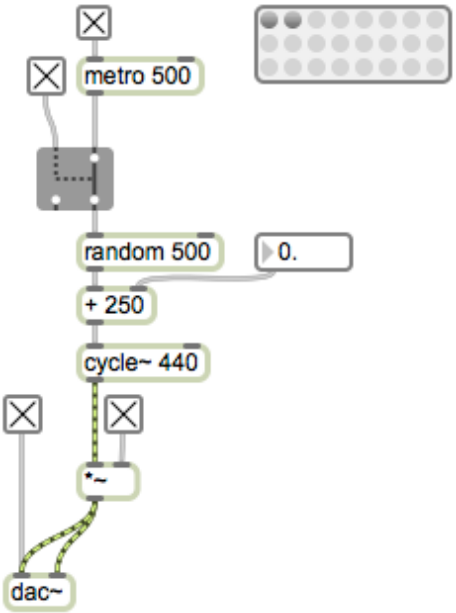


g. To store the present setting, lock the patch and Shift-click on a bubble. The bubble will darken, as shown below.

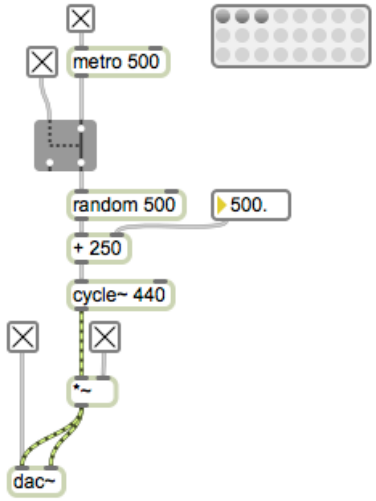


Notice that the bubbles are numbered. The numbers appear when you mouse over them.

h. Now, I will recreate mode ii. of the patch, as listed above, and store it in the next preset setting.



i. To recreate mode iii., I will set the value of the float box to 500, and store this in the third preset setting.



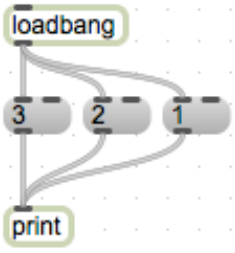
j. Now, thanks to the `preset` object I can switch between these three modes with one click, while the patch is running.

2. `loadbang`

a. The `loadbang` object will trigger a bang as soon as the patch is opened.

b. Some considerations and caveats of `loadbang`'s behavior:

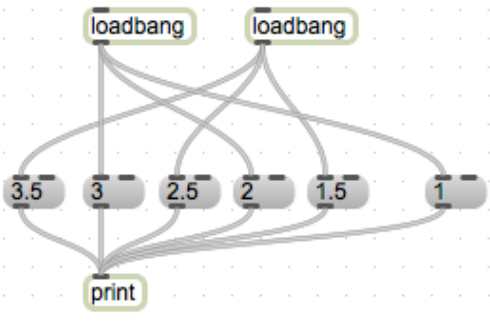
i. The bang from each **individual** `loadbang` will act in the usual **right to left** sequence of Max's execution order.



Object	Message
print	1
print	2
print	3

ii. The execution order of **multiple** `loadbang` objects follows the **order in which they were created**.

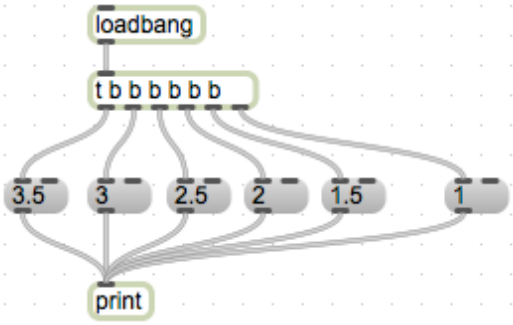
iii. So, adding the following to our setup:



We would intuitively think that, working right to left as usual, in the Max window we would see 1, 1.5, 2, 2.5, etc. But, since the `loadbang` connected to 1.5, 2.5, 3.5 was created **after** the `loadbang` connected to 1, 2, 3, we get the following output:

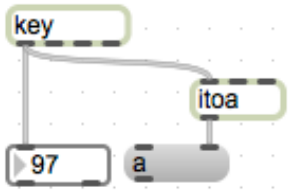
print	1
print	2
print	3
print	1.500000
print	2.500000
print	3.500000

iv. A solution to this would be to use one `loadbang` and a `trigger`:



print	1
print	1.500000
print	2
print	2.500000
print	3
print	3.500000

3. `counter`
   a. The `counter` object outputs sequential numbers, based on its creation arguments or input to its inlets.
   b. Created with **no** arguments, the object will count up by 1, starting at 0.
   c. Created with **one** argument, the object will count from 0 to the **argument value**. Keep in mind that this is distinct from an object like `random`, which outputs from 0 to *n*-1; `counter` outputs from 0 to *n*.
   d. Created with **two** arguments, the object will count from the lower value to the higher value by 1. Keep in mind, that even if the higher value is entered first, `counter` will begin at the lower value.
   e. Created with **three** arguments, the behavior of `counter` becomes divergent; the first of the three arguments will set a loop 'mode':
      i. 0 – `counter` will count from the lower value to the higher value by 1, then return to the lower value.
      ii. 1 – `counter` will count from higher to lower, and return to the higher value.
      iii. 2 – `counter` will count from lower, to higher, then back down to lower.
   f. A bang to the leftmost inlet will advance the `counter`.
   g. Sending a value to the rightmost inlet will reset the maximum value of the object.
   h. Explore the help file for information on `counter`'s more specialized inlets and outlets.
4. `key`
   a. `key` responds to typing on the keyboard, and reports the ASCII value of the last key pressed.



Notice that I have also added the *itoa* object, which translates numbers into their equivalent ASCII

characters, to reflect in the patch what key was just pressed.

b. Using the numerical output of `key` and a `select` object, we can connect specific keystrokes with a trigger inside the patch.