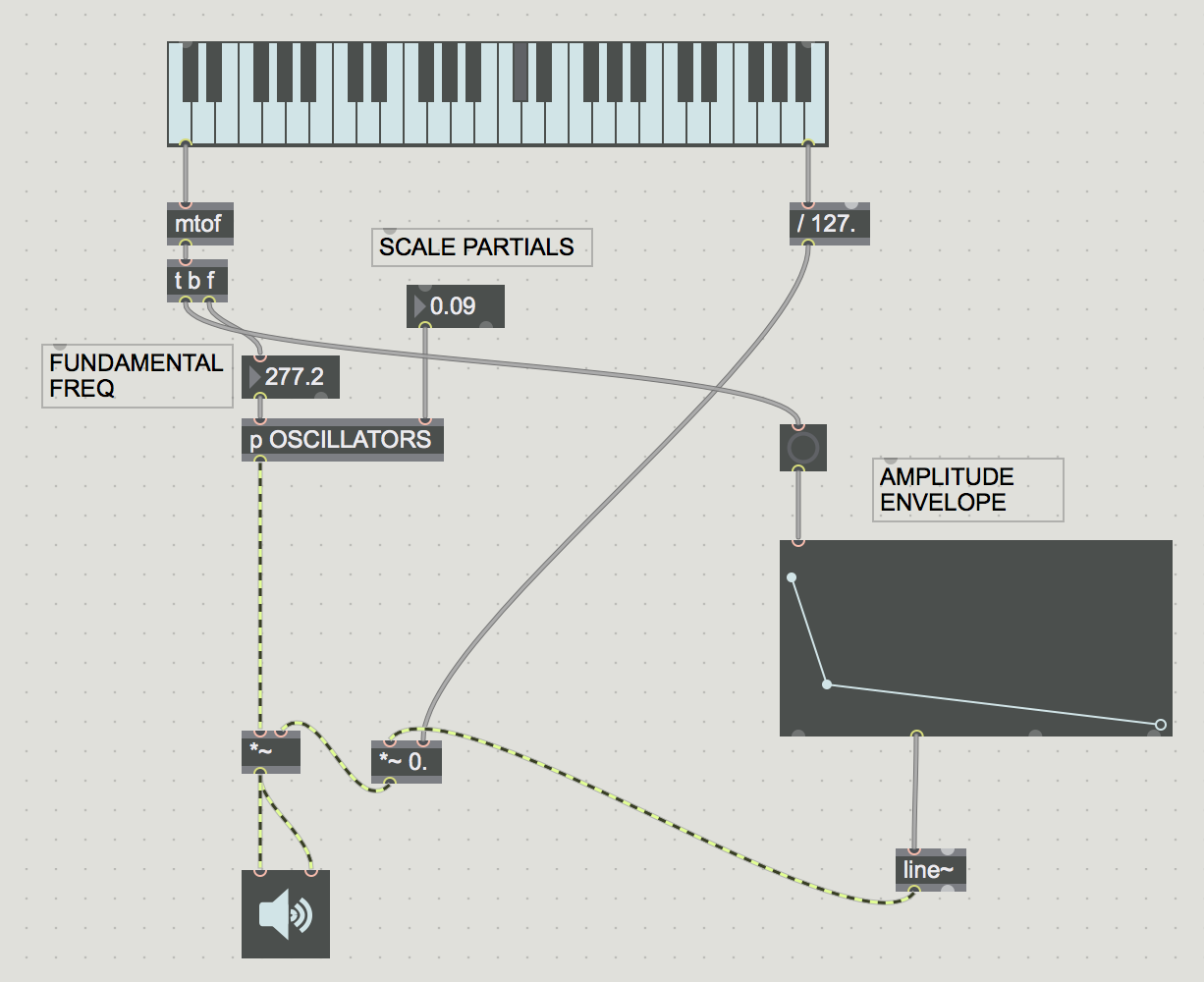
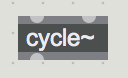
INTRO TO MAX/MSP &

ADDITIVE SYNTHESIS

# OBJECTS & ADDITIVE SYNTHESIS

MaxMSP is composed of **‘objects’**. Each object has a specific function which we can link to one another. Let’s start from the basics.

Welcome to the [cycle~] object.

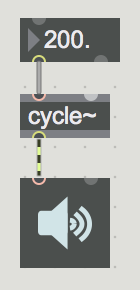
The [cycle~] object produces a **sine wave**.

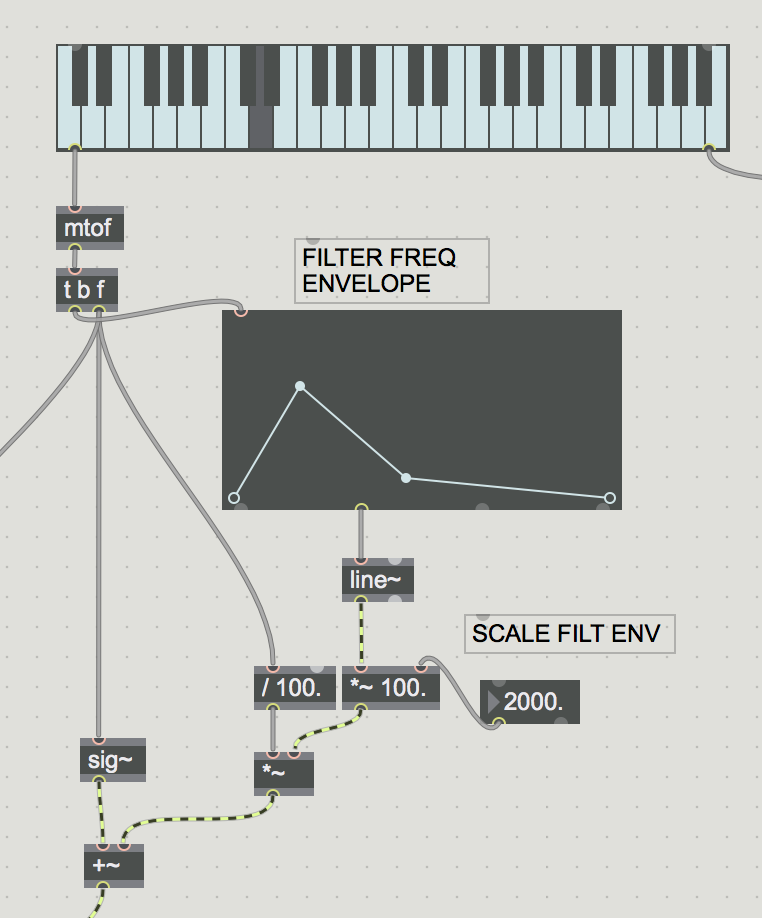
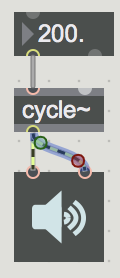
This is the [ezdac~] object. This will be our **output**.

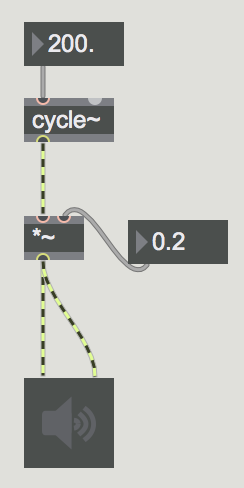
Let’s link them up!

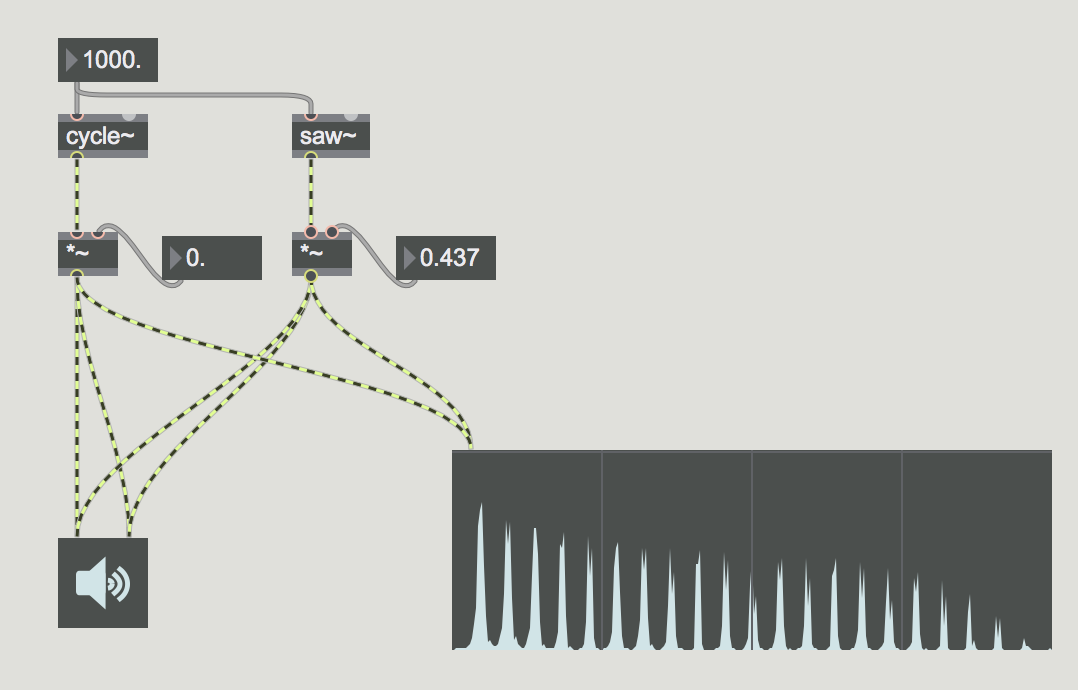
We don’t hear anything. Why might that be?

We need to provide a **frequency** to our sine wave, as well as turn on our output (like flipping on the lights, even if they’re plugged in you still need to flip the switch). Let’s add a **float object** for our frequency.



We only hear sound out of the left speaker. Speakers normally have two outputs (one for each ear), but we could theoretically have more - for example, movie theaters have six speakers (5.1 - five speakers plus a sub. Later on, when talking about experimental music, can touch on Stockhausen and his pioneering work in multiphonic pieces and performances.)

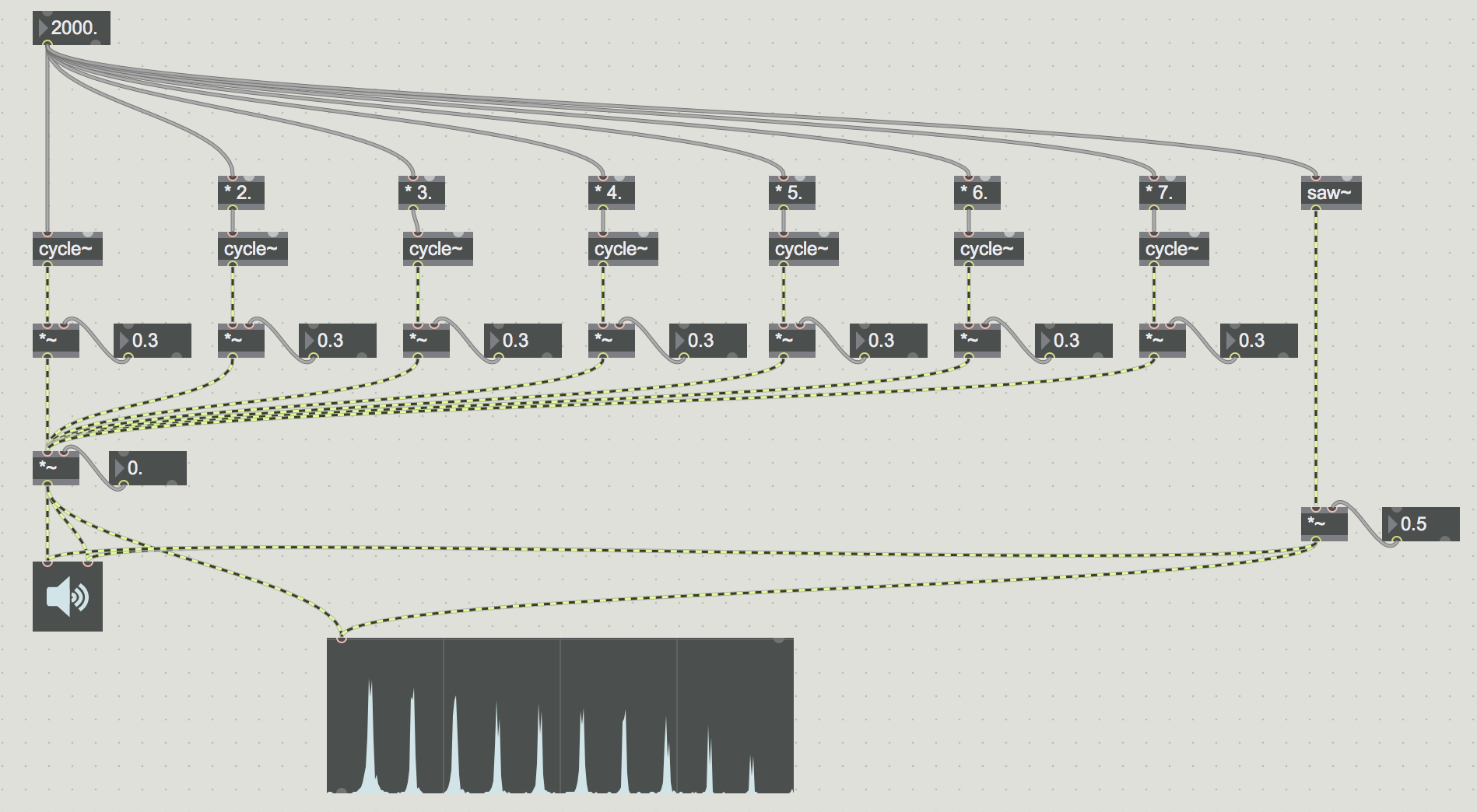
We now have sound! But it’s quite loud, so we need to scale our **amplitude**. How might we do this? We’ll again use the float object, as well as a multiplication object [\*~]



Theoretically, we now have the tools available to make *any possible sound*. This is because all sounds are created by an infinite number of sine waves of varying amplitude and frequency. To illustrate this, let’s compare two different waveforms - our **sine wave** and a **sawtooth wave**. Using a **spectroscope** we can see the **frequency spectrum** of any sound (this is what we used to help with our mixing when using Ableton).

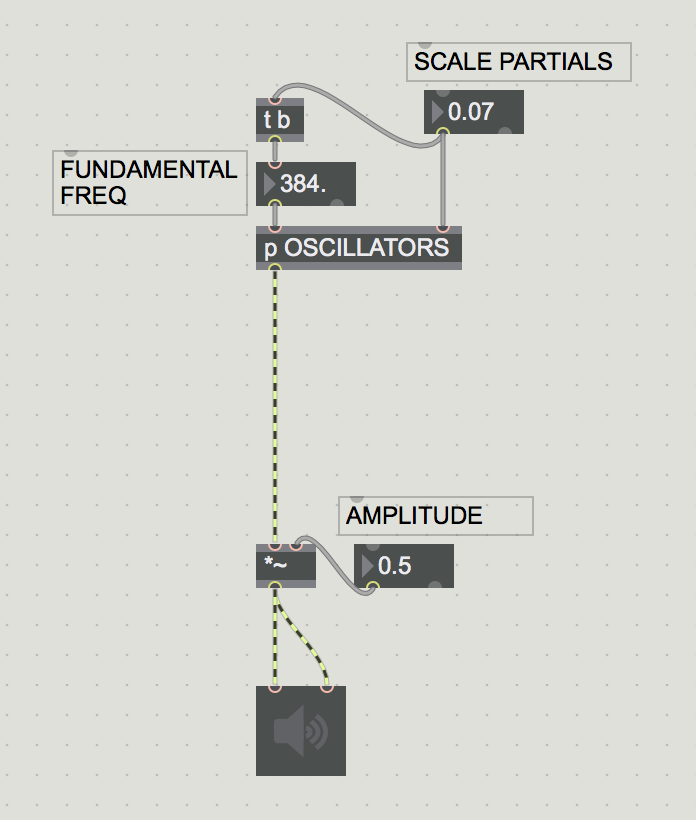
**PATCH: additive\_synthesis\_ex0**

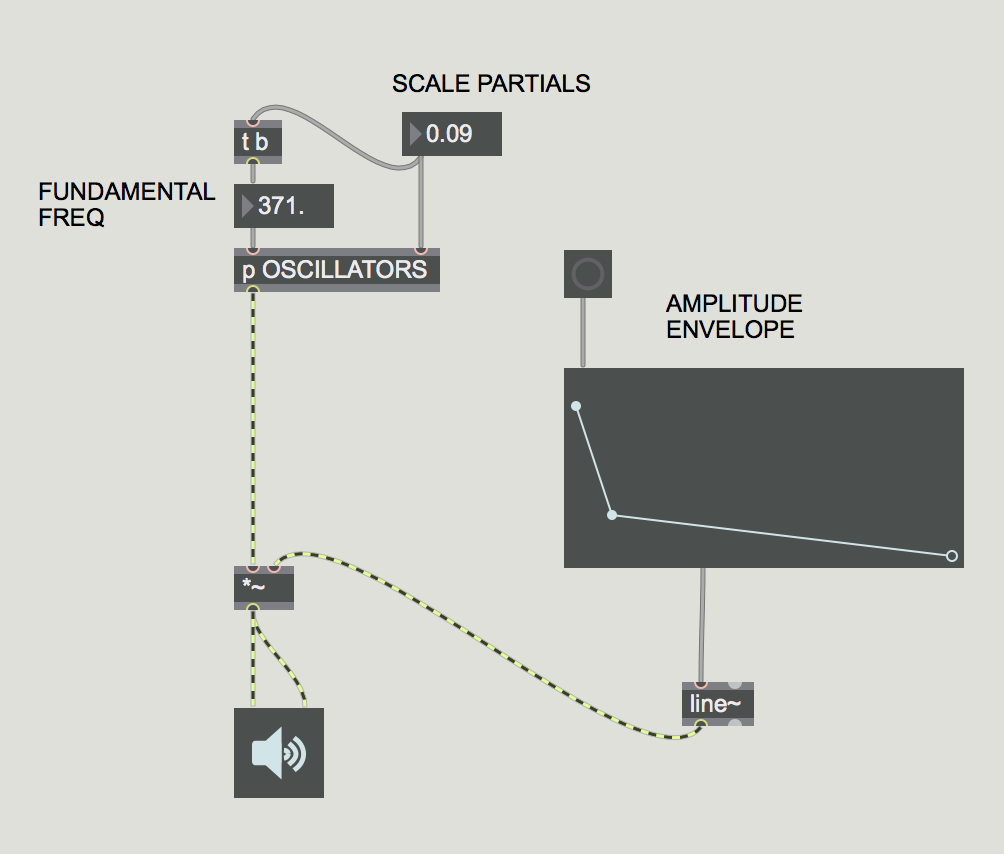
A sawtooth wave has **energy** in the frequency spectrum at integer multiples of the **fundamental frequency**. We call these **partials**. We could even use a large number of sine waves to recreate the sound of a sawtooth wave. Let’s start to do this.

**PATCH: additive\_synthesis\_ex1**

Notice how similar both of these sound. Any sound, no matter how complex, can be made by adding sine waves together at varying energy for different frequencies. We call this **additive synthesis.** We can experiment with how the fundamental frequency and each partial’s amplitude is scaled to create different **timbres**. **PATCH: additive\_synthesis\_ex2**

Let’s **encapsulate** our oscillators by selecting them and hitting shift+command+E. This will help keep our patch clean, and make the routing a bit clearer. **PATCH: additive\_synthesis\_ex3**

Being continuously blasted by sine tones isn’t very many people’s idea of a good time. Rather than the sound continuously playing, we can use **envelopes** to control the amplitude of the sound. Let’s take a look at the **function object**. This object allows us to create our own envelope, which we can connect to our amplitude using [line~].**PATCH: additive\_synthesis\_ex4**

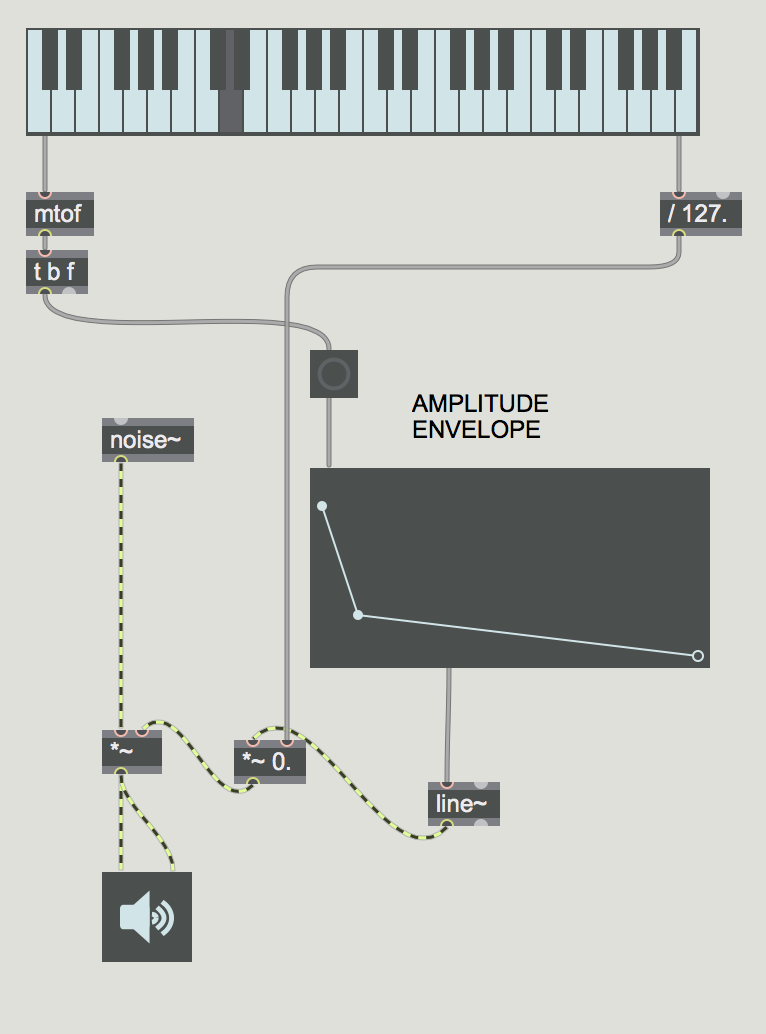


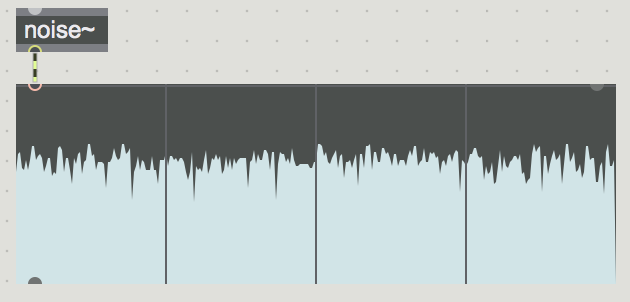
Lastly, let’s use a keyboard to control our fundamental frequency. We can use the **kslider** object to accomplish this. The **mtof** object converts MIDI note values (0-127; recall our Ableton lesson) to frequencies. The height of our mouse on the kslider controls the velocity for our note. How could we link this to our amplitude? **PATCH: additive\_synthesis\_ex5**

SUBTRACTIVE SYNTHESIS

In additive synthesis, we layered sine tones to create unique timbres. **Subtractive synthesis** works in the opposite direction, beginning from a rich frequency spectrum and using **filters** to sculpt the frequency spectrum into a unique timbre.

# FILTERED NOISE

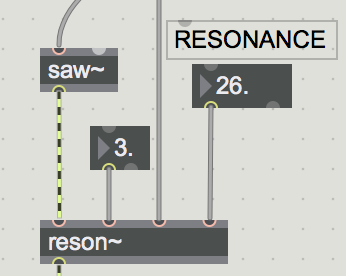
What kind of sound has the richest possible frequency spectrum? That’s right! **White noise** has the richest possible frequency spectrum, with even energy distribution throughout the spectrum. Let’s use this as the basis for our first foray into subtractive synthesis. We can begin by first opening up our patch from last time (additive\_synthesis\_ex5) and deleting our oscillators. This will leave us with our keyboard and amplitude envelope. In place of our oscillators, let’s put a **[noise~]** object. **PATCH: subtractive\_synth\_ex0**

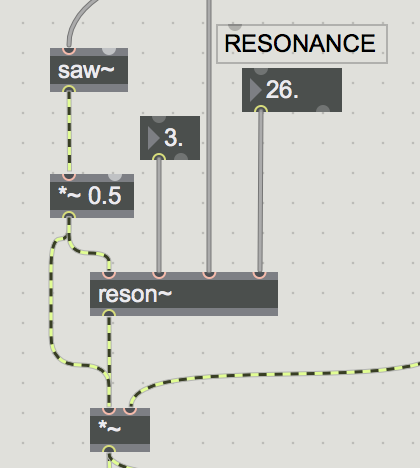
What does it sound like when we play a key? There is definitely a percussive element to it. Subtractive synthesis is frequently used for percussion synthesis, since many percussion sounds are quite noisy - noisy in the sense that they have widespread energy distribution in their frequency spectrum, not noisy in the sense of a construction site or rock ’n’ roll. This kind of noise can be seen when we connect the spectroscope~ object to our output.

What if we wanted to add pitch? This is where we use filtering to subtract from our frequency spectrum. Let’s use a bandpass filter with the **[reson~]** object. We can now pitch our noise, and play with the resonance of our filter for less or more noise in our sound. We also should boost our signal’s gain, since when we cut out that much energy from the frequency spectrum, the overall amplitude of our signal is greatly decreased. **PATCH: subtractive\_synth\_ex1**

# SAW & DRY/WET

Let’s take a look at **[saw~]**. The sawtooth waveform has even energy in all of its partials, giving it a very bright, harsh sound - perfect for subtractive synthesis. Let’s replace our noise~ object with a saw~ object and see what kind of aesthetic is produced. **PATCH: subtractive\_synth\_ex2a**





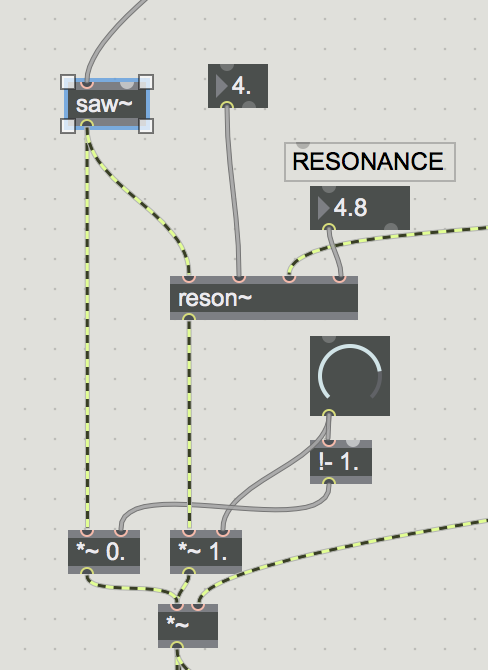
This sounds very much like a sine wave. Why is that? That’s right, we are only isolating the fundamental frequency through our reson~ object. Let’s brightness the sound up by mixing in the unfiltered sound with our filtered sound. How might we do this?

**PATCH: subtractive\_synth\_ex2b**

We can simply halve the signal then add it back to itself. This will insure that our signal doesn’t **clip** (exceed 1.0). We can hear that the sound is now **brighter** (has more energy in the upper partials).

We can make our sound more dynamic by creating an envelope for our filter. How might we do this?

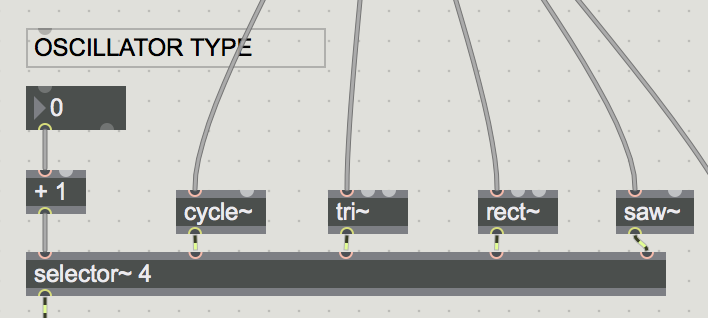
We have to add (or subtract) a value from our frequency. However, frequency is scaled logarithmically, so the value we add or subtract shouldn’t be fixed but rather proportional to our input frequency. Therefore, we can divide our input frequency by 100, then multiply that by a value according to a new function object. Let’s try to code that. **PATCH: subtractive\_synth\_ex3**

We can change the gain of the **dry** signal versus the **wet** signal to make our signal more or less filtered. If we wanted this to be controlled by a signal knob, how could we do that?

The **[!- 1.]** object subtracts the input from 1, so if we use this to control the gain for the dry signal, and the original value of the dial (0-1.) to control the gain for the filter, there will always be a combined value of 1. for their mixed signal. Max/MSP requires a bit of logic in this way - figuring out how basic arithmetic can be strung together to create a functioning, logical and elegant signal flow. **PATCH: subtractive\_synth\_ex4**

Right now we are using the most elementary of synthesis building blocks, but a synthesizer can use a large number of objects to create very lush, nuanced and evolving sounds. Let’s take a quick look at other filters and oscillators offered by Max.

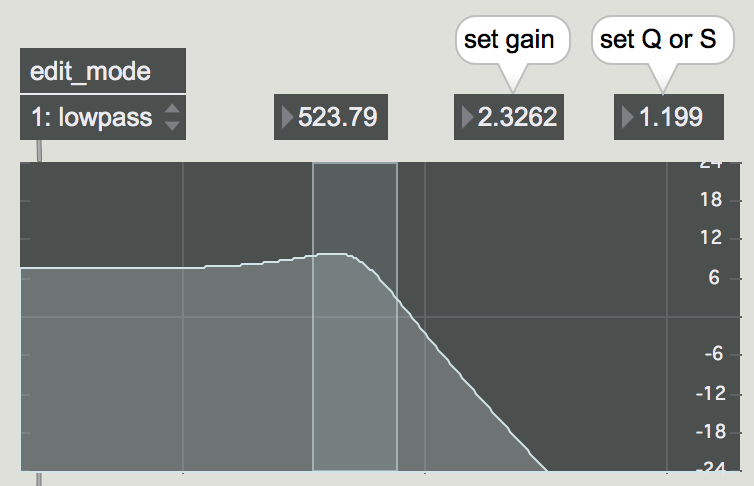
# MORE FILTERS AND OSCILLATORS

Let’s try out different oscillators. We can do this by using a **[selector~]** object to move through different signal inputs.

The four basic waveforms are the **sine wave, triangle wave, square wave,** and **sawtooth wave,** shown in this patch as the **[cycle~], [tri~], [rect~] and [saw~]** objects. These objects are specially designed to be used as audio rate oscillators, since other objects, such as **[triangle~]** and **[phasor~]** which produce the same waveform have not been **antialiased** (meaning they have harmonic noise in their signal since their harmonics extend above the 22.1kHz Nyquist frequency (more on this next lesson) - however, those objects produce their own aesthetic and could also be used if one preferred their sound).

While these are the most basic waveforms for subtractive synthesis, they are far from the full extent of what’s possible with synthesis. When we look at sampling in two weeks, we can look at using sample fragments as **wavetables**, meaning using an array of values that are played back very quickly to produce completely unique and individual timbres.

There are many filter types. One common way to move through different kinds of filters is to use the **biquad** object. This object allows you to make any filter type and graphically control it with the **filtergraph** object. Combining our oscillators with the biquad and filtergraph already makes for a significantly more flexible synthesizer from our first round. Play around with the settings for this. How could we expand on it? **PATCH: subtractive\_synth\_ex5**

****

FM SYNTHESIS

One of the beautiful things about Max/MSP is the ability to explore connections - what happens when this object links to another object? How would it sound for this to connect to that? These possibilities can make Max a creative exploration of strange, unnatural, signal flows.

Peculiar things start to happen when signals get brought together. For this lesson, we’ll look at what happens when a signal controls another signal’s frequency in a common synthesis method called **FM Synthesis**.

# LFO Modulation

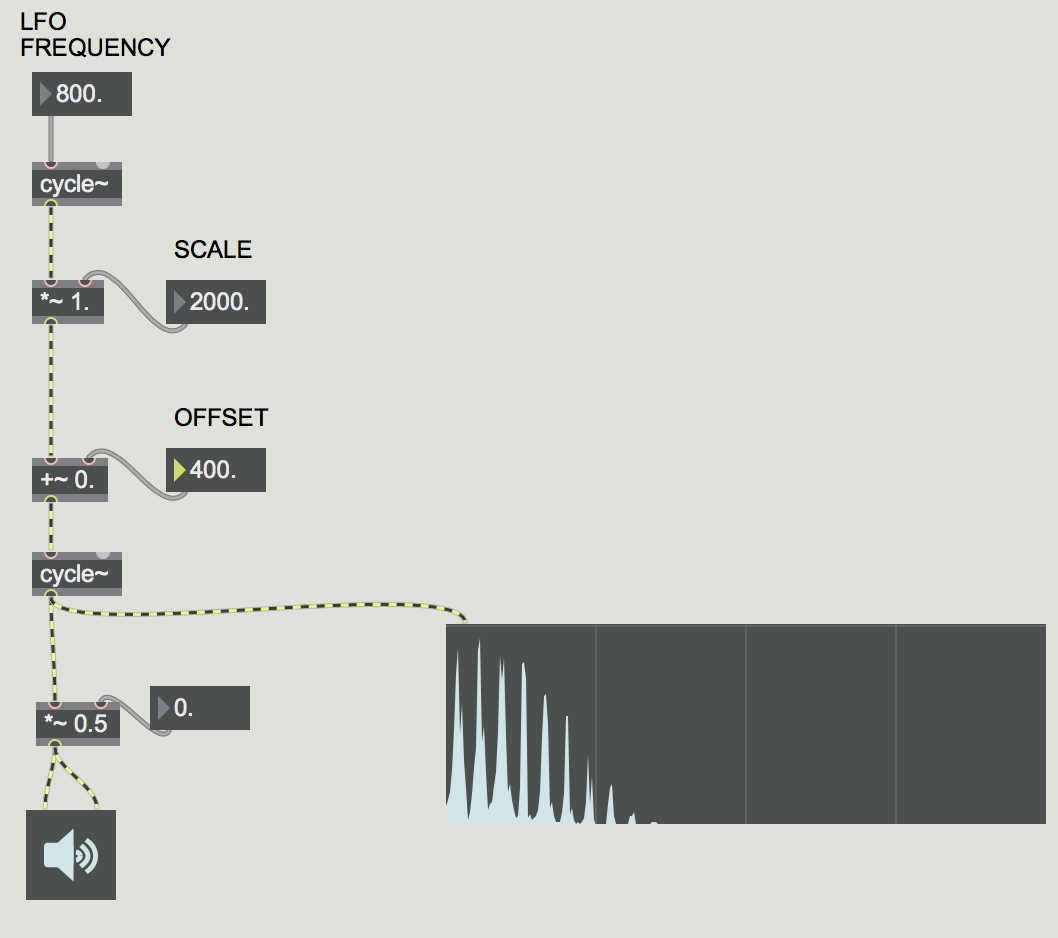
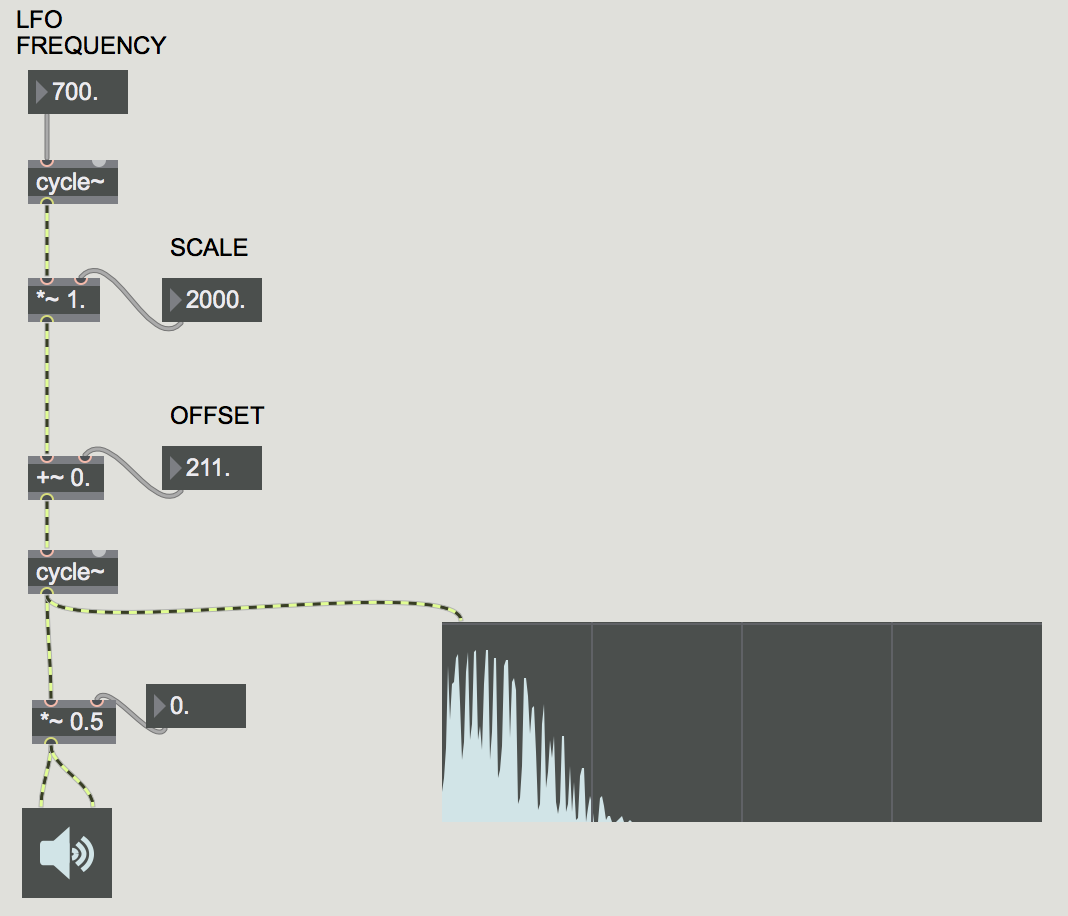
Recalling the behavior of our Auto Filter effect in Ableton, we had an LFO (low frequency oscillator) that could control the filter cutoff at a subaudible rate. Let’s create our own LFO to control the frequency of a sine tone. How might we do this?

We have one sine wave acting as our LFO. We then scale this LFO and offset its value, and connect that to another sine wave. Play around with these values, try out different values. Don’t be afraid of extremes!

**PATCH: fmsynth\_ex0**

Note: that this schema can be applied to any signal value - amplitude, filter parameters, effects, sequencers, etc. We will explore this later on.

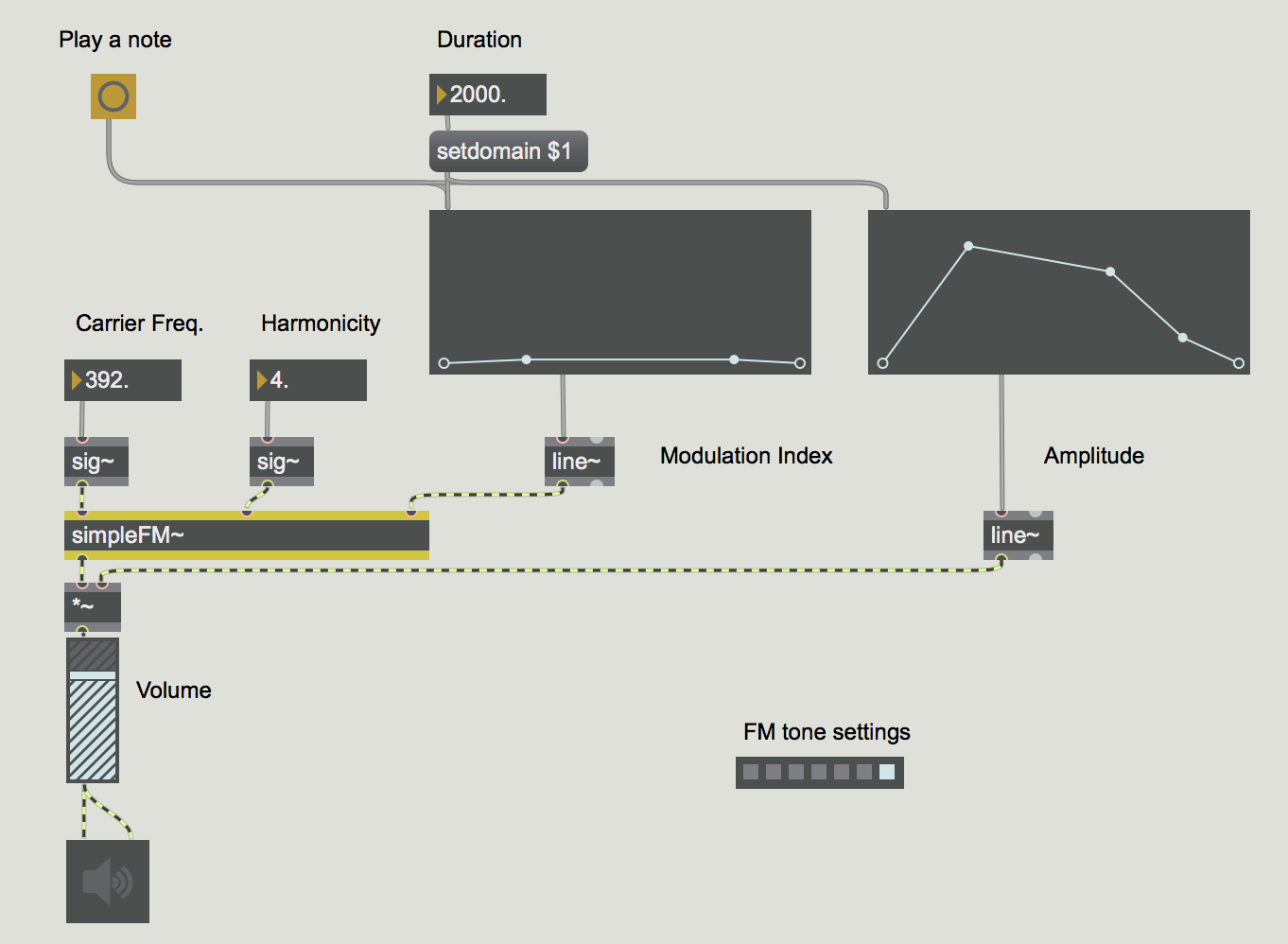
When our LFO rate exceeds 20Hz (thus entering the audible range of 20Hz to 20kHz), the quality of the sound begins to change. The sound is no longer ‘wobbly’ but rather *textured*. This comes from harmonic patterning that is unique to FM synthesis. Let’s take a look at a spectroscope~ to get a better sense for the oscillator’s timbral behavior.

Exploring a bit, we find that when the LFO is at a frequency which is an integer multiple of the ‘Offset’ frequency, that the spectrum resembles that of a square or sawtooth wave. At non-integer ratios, the spectrum becomes more ‘noisy’. **PATCH: fmsynth\_ex1**

# FM Synthesis

This system of relationships between oscillators has been formalized, and there’s a specific lexicon that has emerged around it, primarily based on the work of John Chowning at Stanford in the late 1970’s. The oscillator which is producing sound is known as the **carrier**, and the oscillator modulating the frequency is known as the **modulator**. The ratio between their frequencies is termed the **harmonicity ratio**, while the depth of the modulation is known as the **modulation index**. In the Documentation for Max, there is a wonderful example of their relationship, as well as an object called **simpleFM~** that encapsulates this functionality.

To make Max extra challenging, they’ve obscured how to get to the tutorials in recent releases of Max. If you right click the cycle~ object and open its reference, then scroll to the bottom of the page and open one of the Tutorial options, the documentation will open. You can hit the home button in the top left to get to the opening page. The tutorial we’re looking for is in MSP, titled ‘FM Synthesis”. Let’s take a look at this patch. **PATCH: fm\_tutorial**

****

We can save this as our own patch and start making modifications. Bring over our keyboard from the other patches to make the patch more playable. How would we connect everything together? How could we map velocity to modulation index?

We can also make our presets using the **preset** object. The preset object (the grey squares where it says ‘FM tone settings’) will store all number box and function values, allowing you to quickly jump back to settings you enjoyed. Scale the preset box to make it bigger and allow for more presets.

# Combining It All Together

The real majesty and mayhem of Max lies in the ability to tear apart and recombine *everything*. What if we used our oscillator selector from the subtractive synthesis example as our carrier or modulator oscillator within FM? How would that sound? What if our pitch determined our oscillator type? What if we layered an FM, subtractive and additive synth together? *What if a carrier modulated itself??* The possibilities are endless. Let’s play with these ideas individually for a bit before we conclude.