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.

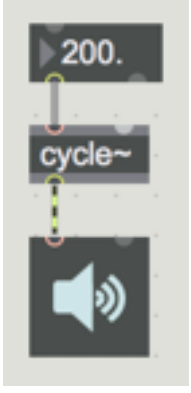
The [cycle~] object produces a **sine wave**.

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

Let's link them up!

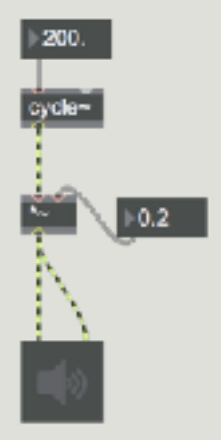
We don't hear anything. Why might that be?

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.

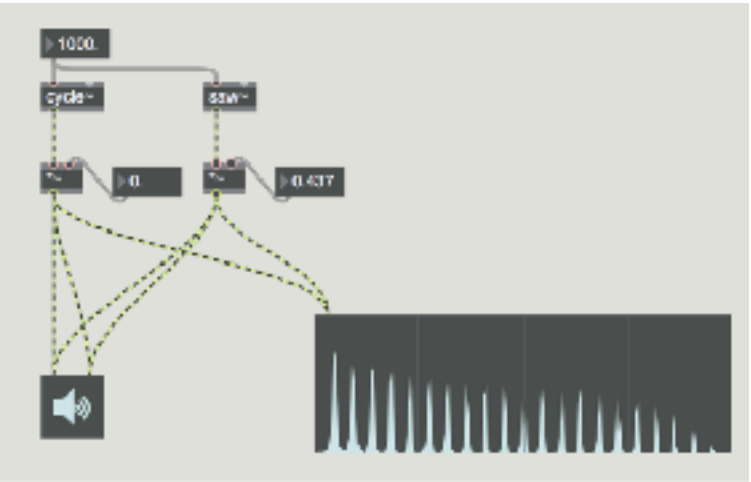


We only hear sound out of the left speaker. Speakers normally have two outputs (one for each ear), but we could theoretically have more - for example, movie theaters have six speakers (5.1 - five speakers plus a sub. Later on, when talking about experimental music, can touch on Stockhausen and his pioneering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

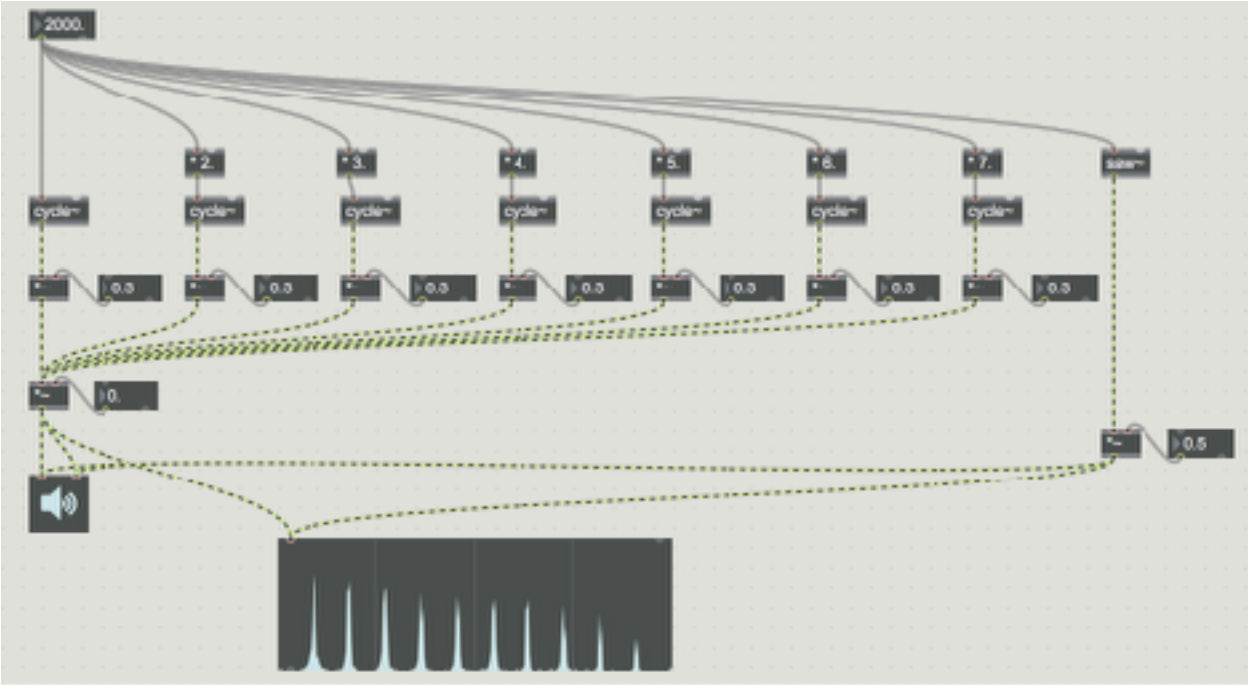
We now have sound! But it's quite loud, so we need to scale our **amplitude**. How might we do this? We'll again use the float object, as well as a multiplication object [*~]



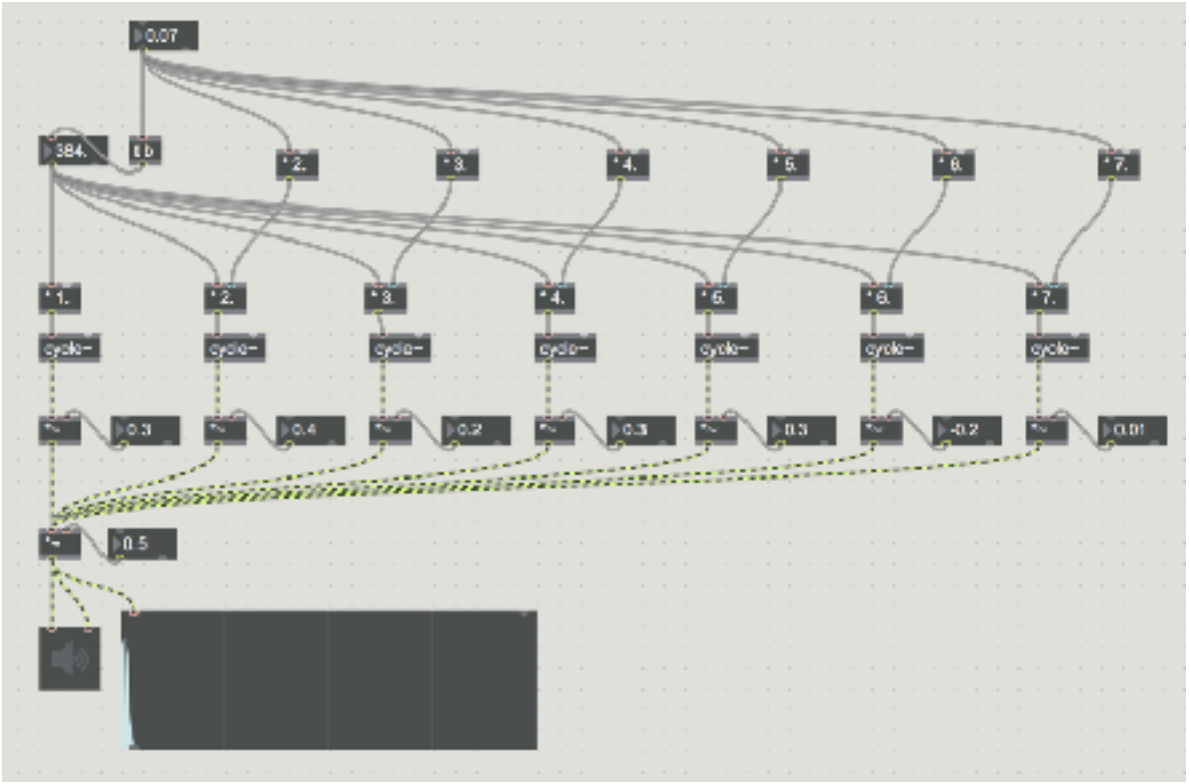
Theoretically, we now have the tools available to make *any possible sound*. This is because all sounds are created by an infinite number of sine waves of varying amplitude and frequency. To illustrate this, let's compare two different waveforms - our **sine wave** and a **sawtooth wave**. Using a **spectroscope** we can see the **frequency spectrum** of any sound (this is what we used to help with our mixing when using Ableton's EQ8).
PATCH: additive_synthesis_ex0



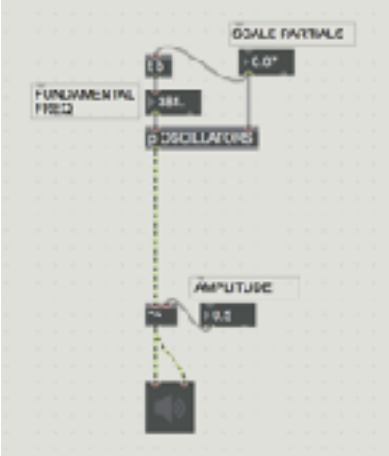
A sawtooth wave has **energy** in the frequency spectrum at integer multiples of the **fundamental frequency**. We call these **partials**. We could even use a large number of sonewaves to recreate the sound of a sawtooth wave. Let's start to do this. **PATCH: additive_synthesis_ex1**



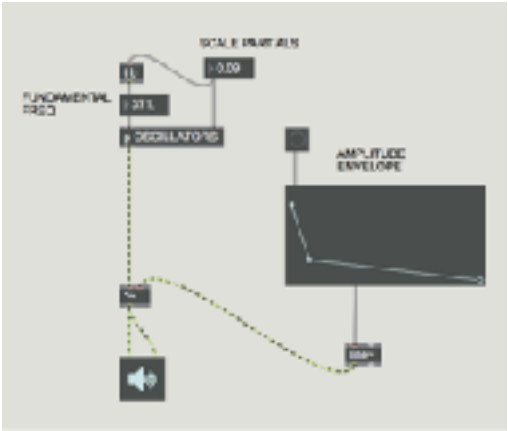
Notice how similar both of these sound. Any sound, no matter how complex, can be made by adding sonewaves together at varying energy for different frequencies. We call this **additive synthesis.** We can experiment with how the fundamental frequency and each partial's amplitude is scaled to create different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