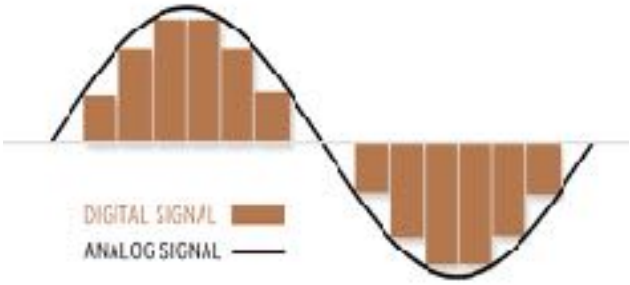
TIME IN MAX/MSP

Some of you may be wondering - why is it Max/MSP? What is Max? What is MSP? This brings our attention to the issue of *time* in digital audio. How is sound represented digitally? How do we control this sound? Let's start by taking a look at **digital signal processing** or **DSP**.

DSP

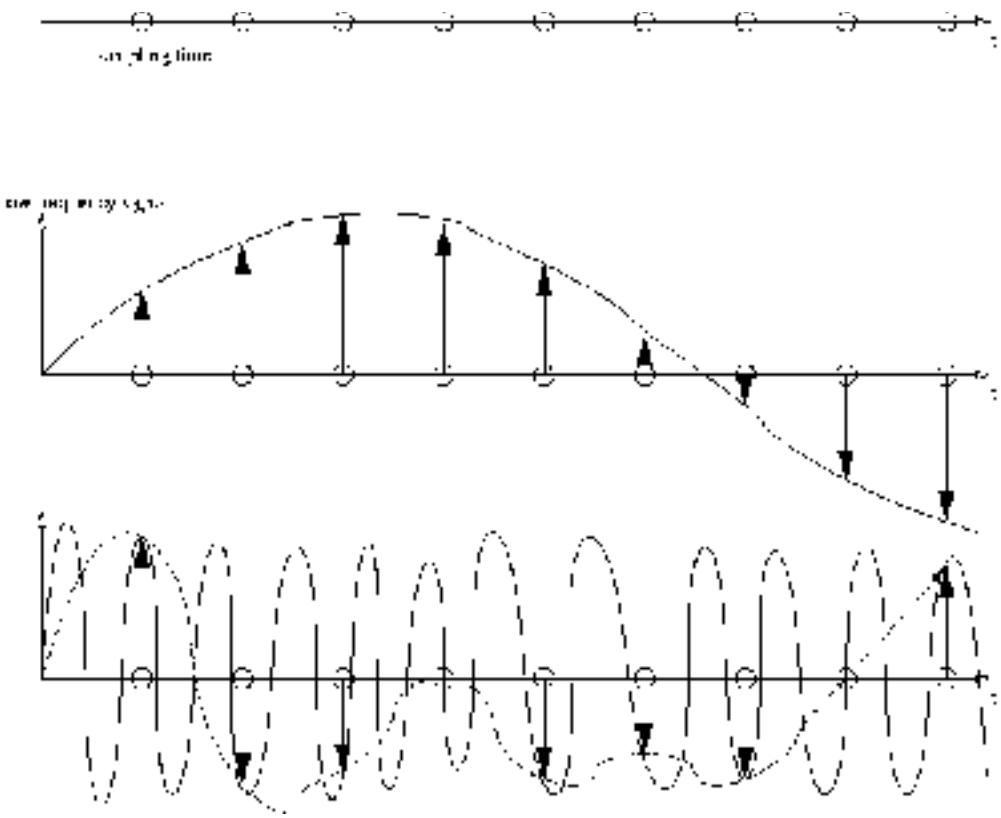
DSP refers to having a signal (a continuous stream of values) which is transformed within a computer. Audio is its own special kind of signal. Before we can understand how this signal is processed digital, let's understand the key differences between **digital** sound and **physical** sound.

When we think of the physical world in the perceptible, Newtonian sense, (excluding quantum mechanics), it is a **continuous** space, meaning it does not start and stop. Between this screen and your face, there are an infinite number of points. We can measure this distance in centimeters or feet, but the space itself does not stop at any point. The same is true for audio, with the harmonics of a sound in the frequency spectrum extending infinitely, becoming increasingly less However, within the digital realm, resources are limited - your computer has a certain size, there's a certain amount of electricity it's supplied with, etc. and therefore, the information within a computer must be **discrete**, or limited.



Within audio, sound that is represented physically is termed **analog**, since it provides an analog to the behavior of the sound wave in physical space. For example, the grooves on a vinyl record are the actual waveform, which is then amplified and sent out the speakers, thereby vibrating the air in the exact pattern is it's etched on the LP. However, given the discrete nature of digital sound, its behavior is slightly different. Digital audio is represented as a series of numbers, which indicated the amplitude of our sound wave. But how quickly to playback these values? This is known as the **sample rate**, that is to say, the sample rate is the speed at which values for our sound wave are played back. Since our ears can only hear between 20Hz and 20kHz, digital sound limits its information to this range by having a sample rate of 44.1 kHz. According to the **Nyquist theorem**, sound (including all harmonics) can only be played back without distortion when the sample rate is twice as high as your highest frequency. So, with a

sample rate of 44.1 kHz, what's the highest frequency that can be represented? Let's take a look at this as a graph.



As we can see, when we have the same sample rate but a higher frequency sound wave, the values from the sound wave are misrepresented. This is the same phenomenon that causes the wheels on cars in videos to appear to be spinning backward: https://www.youtube.com/watch?v=B8EMI3_0TO0 As audio is played back from the computer to our speakers, it is sent through a **digital to audio converter (DAC)**, which smoothes the sound out and translates the digital values into electricity, which then is sent to our speaker.

MSP

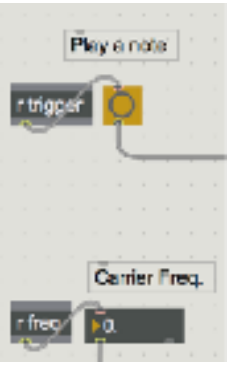
So what does this have to do with anything? Well, it's good to know how audio is functioning inside the computer so that one is more knowledgable about the differences between different kinds of audio technology, and what certain terminology means. It also is a key differentiation between different parts of Max/MSP. The **MSP** portion of the title refers to everything that operates at **audiorate**, that is data which is cent continuously at our sample rate (typically 44.1 kHz, although this can be changed inside Option->Audio Status). We know an object is an MSP object when it has the tilde '~' at the end of its name. What objects have we worked with that are audiorate?

MAX

Not all information within our Max/MSP programs have to be sent at a continuous rate of 44,100 times per second. That would be extremely demanding on our computer. The **Max scheduler** controls non-audio information, and operates at its fastest at about 1000 Hz (still quite fast). The Max scheduler is useful for controlling the parameters of our audio. For example, MIDI would operate according to the Max scheduler, but the synthesizer it controls is oscillating at audio rate. What objects have we used that are Max objects?

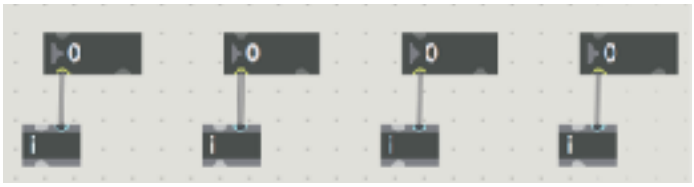
Max provides very useful **GUI** objects which we can use to create instruments, interfaces, and generative media machines. In line with our theme of time, today we will look at how to create a basic sequencer to control the synthesizers we built last class.

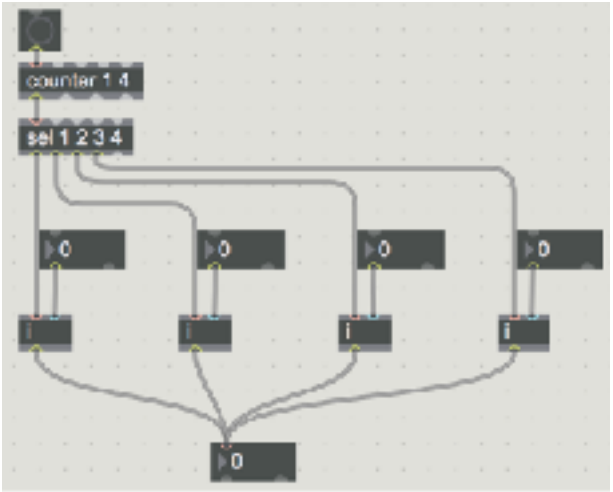
Basic Sequencer



What is a sequencer? How does it work? A sequencer rhythmically plays back stored information to control a synthesizer or sampler. We can use the **send** and **receive** objects to send information between Max patches. Let's open up our FM patch from last class and create a [receive freq] object and [receive trig] object. Connect these to the 'Carrier Freq' number box and the 'Play a note' **bang** object. This 'bang' object simply tells Max to *do* something. As we continue to use it more, it's necessity will become obvious. Note - [send] and [receive] can be shortened to [s] and [r].

Let's open a blank patch and create four **int** objects. These objects stored integer (non-decimal) values. Create four int boxes (by hitting 'i') and connect this to the right inlet of the int object.



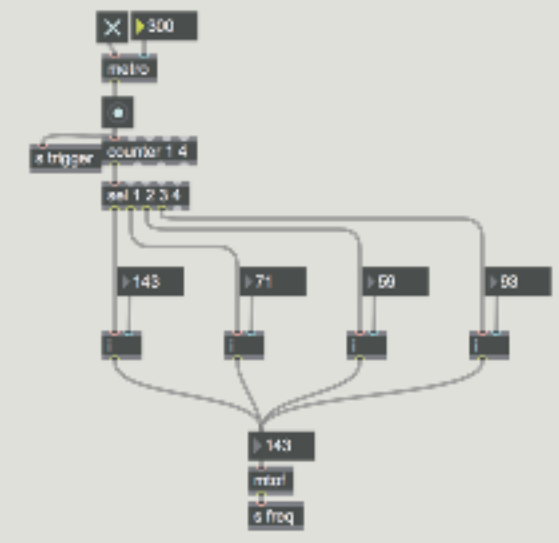


Above this, let's create a **counter** object, with arguments of 1 and 4. The counter object counts every time it receives a bang. The arguments determine the minimum and maximum for its counting; it loops back to the minimum value when it exceeds the maximum value. Let's connect our counter to a **select** object (sel for short) with the arguments 1, 2, 3 and 4. Connect the first four outlets of the select object to the left inlet of our int objects. Lastly, connect a bang object to your counter, and all of your int objects to an int box.

Start putting values into your int boxes (that are connected to the int objects) and clicking your bang. Notice how the int box at the bottom changes. What values is it returning? Why? What's the logic that is taking place here?

Let's now make our sequencer work on its own by adding a **metro** object. The metro object outputs a bang every X number of milliseconds. You can change the millisecond rate of the metro by adding an integer box to the right inlet. We can turn our metro on/off by the **toggle** object, which sends out a 0 or 1 depending on whether the box is checked. Let's then add an [mtof] object to convert midi notes to frequency from our int box at the bottom, and then send the bangs from our metro and the value from the motif object to our send objects (freq and trigger). Turn on your DAC by clicking the speaker symbol in our synthesizer patch.

PATCH: basic_sequencer.maxpat

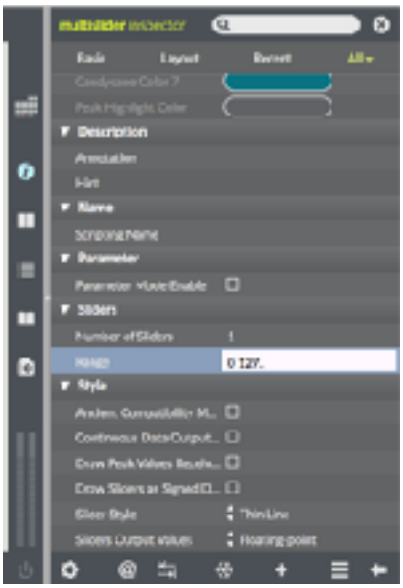


Congratulations! You just made your first sequencer. However, it's quite limited. What are you some ways we may want to expand on our sequencer?

GUI-BASED SEQUENCER

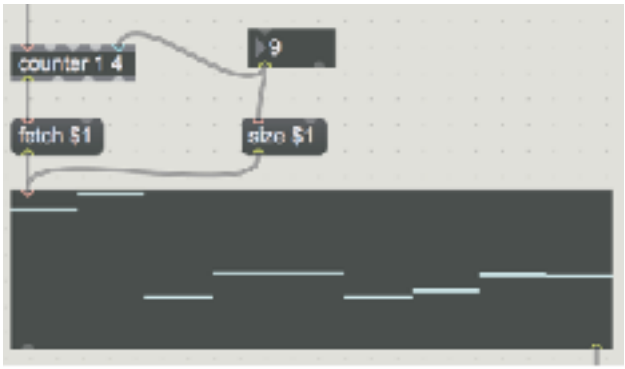
One fabulously useful way to create sequencers in Max/MSP is sue of the **multislider** object. This object allows us to graphically interact with our data, thereby making interacting with our patch much quicker and our patch becoming much more visually compelling. Let's take our patch and delete the select object and our int boxes and objects. Create a multi slider object and stretch it out. We will then use the fetch $1 **message** to grab the value of a given slider. Messages in Max control parameters within our objects. The '$1' indicates a variable for that message. If we connect the counter to our fetch message, this tells the multi slider to fetch the value for slider 1 when the fetch value is 1, slider 2 when the fetch value is 2, etc.

Connect a **float box** (by hitting F) to the bottom right outlet of the mutlislider. Now turn on your metro and change values in our multislider. As you can see, we only have one slider. What are its minimum and maximum values? What values would make sense for minimum and maximum? What is the range for MIDI? That's right, 0-127, so let's change the minimum maximum values for our sliders. We can do this by clicking our object and going into the inspector window and changing the Range values (we could also do this by a message, but the



inspector window is quicker, and we only need to change it once). Note, our inspector window also has many options for modifying the GUI of our multislider.

We could also change the number of sliders in our multi slider object in the inspector window. But what if we wanted a dynamic number of sliders? Create a message **size $**1 and connect an int box to it. Connect the same int box to the last inlet of our counter. What does that inlet do? Why do we have to connect this int box to both our multislider object as well as to our counter object?



Try playing around a bit with the values of your sliders, see what patterns you can come up. Try duplicating this mutlislider sequencer, but connecting the output to the metro. Now you have one sequencer controlling the frequency, and another sequencer controlling the original sequencer. Owing to Max's immense flexibility, this paradigm for sequencing can be used to control *anything*, including preset selection, any synthesis parameter (harmonicity, modulation index), MIDI info (velocity, duration, octave), etc. The only limit is your imagination.

PATCH: mutlislider_sequencer.maxpat

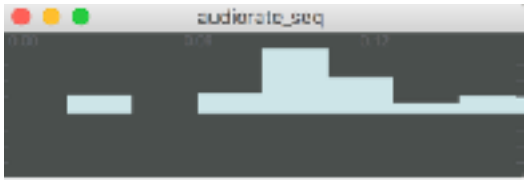
AUDIORATE SEQUENCER

For control information, the Max scheduler fulfills pretty much any purpose we have. There are many layers of complexity we could add to our sequencer (changing direction, using basic mathematic operations to offset it, sequencing these basic operations, dividing our pitch control between octave and pitch, playing with phasing sequencers). However, having an **audio rate sequencer** can lead to some very interesting and experimental directions.

To create our audio rate sequencer, lets use the **peek~** object and the **buffer~** object. A buffer object stores sample values, while the peek object can write into a buffer object. Let's then use a mutlislider, this time with values from 0.-1. to write into our peek object. We can do this using a counter to provide the index to our peek object, and using **iter** to read through our multi slider values one by one. Before we do that, we first have to set our counter's value back to 0. For our buffer object, use the same int box to control the size of your multi slider as well as the message **sizeinsamps $**1 connected to out buffer. It's OK to not understand everything I just said, it'll make sense in due time.

By double clicking our buffer object, we can see its values. Notice how they change in parallel with our changes in the multislider object.

PATCH: audiorate_seq1

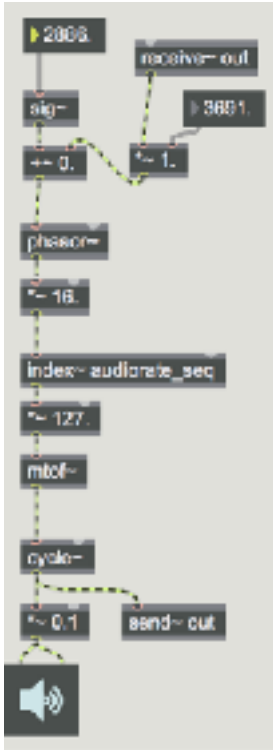


Let's use these values to control the frequency of a sine wave. We can use **phasor~** (which produces a signal moving from 0. to 1.) to read through our **index~** object (read the value of a buffer by index). We can then scale this value and connect it to **mtof~**, the MSP version of our familiar mtof object. Scale the output from your sine to protect your ears.

PATCH: audiorate_seq2

Notice how *everything* we are using is an MSP object (except for our number boxes). What's exciting about this is that anything can be connected to anything. What would happen if we connect our output to our input? This technique is called **feedback**, and normally Max/MSP doesn't allow it since it can make things explode (not literally, but it can cause your computer to freeze). We can get around this by using the audio rate **send~** and **receive~** objects. Try adding these together and playing around with values.

PATCH: audiorate_seq3



Try playing around with these values. Try adding in other audio rate objects (saw~, biquad~, reson~, rect~, etc) and include them in the signal flow. Don't be afraid to connect things together (being sure to use send~ and receive~ for feedback). Creating a place to scale your feedback values can create very dynamic, bizarre, noisy, hypnotizing patches. You can then use your audio rate feedback sequencer as a *synthesizer itself* by controlling its frequency with your slider object or a with a more traditional sequencer. Enjoy the noise!

PATCH: audiorate_seq4