Project one:
Setting up a minor dj tool in max/MSP using your laptop or desktop keyboard as a controller.

Through my many years as a professional electronic musician and DJ, i spent a lot of time playing live by either just clicking play and having a basic fail safe set or just letting go of the controls and don't really touch anything during the live set.
This got so incredibly boring that i decided to create some tools for me to use when i was out playing, even though there where professional tools already that could do these things for me, they always contained more than I needed and the building process of such a tool was destined to be a learning and meaningful experience.

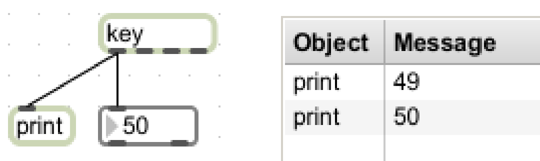
I decided to build a DJ tool, which basic functions was to play two or more tracks at the same time, which could be pitched and manually synced / beatmatched.
In this way I would have a tool that was my own, my own way of playing my tracks and I wasn't using the regular DJ setup which was usually frowned upon by DJ's and other live acts, as a non live act thing to do.

When building a tool or coding anything, it's important to have an idea before you get started, and / or have a slight idea problem solving process. If you start with nothing and just go into blindness, you will sooner or later experience that your code or patch is super messy and takes more time to clean up and keep track of than actually working with it.
Therefore it's very important that it's clear what we will create, why and what it should do in the end and what not to do. A homemade piece of software, or any professonal piece of software for that matter should only function as intended, of course anything can be used as not intended often with cool benefits in performance, but this makes any kind of code / program buggy – and if this tool is to work in a live situation, not crash and so on, buggy is not acceptable (as if it ever was).

Here we will build this tool and along this way expand it on the go to show how already working parts of a piece of software can be expanded easily.
For this we will use max/MSP, which can be aquired at www.cycling74.com – a very usefull GUI based tool, if you don't have or want to use max/MSP, you can always use PD, Pure Data, which can be aquired at Puredata.org, but PD use slightly different, though free.
Max/MSP is also a newly integrated part of abletons LIVE software, so it will never harm to get into max/MSP.

First we need to open max/MSP. If you don't know max/MSP before hand, you should turn to the chapter on max/MSP first, or if you like, use it as a learning by doing experience by following this guide.

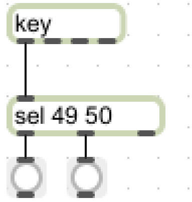
In max/MSP, create a new open patch. A blank piece of paper.
Press N for new object or if you like, double click (max 5 or later) – to get the list of colorful icons up to choose from, and create an object named key.
This object has several outputs and one is ascii number, which you can connect to the object print and here quite easily see the ascii number for any key pressed on your keyboard.
So routing the ascii signal from any key pressed can now be used as buttons to toggle anything you like.



Object	Message
print	49
print	50

Here you can see, that by pressing the keys >1< and >2<, we get the responce 49 and 50.

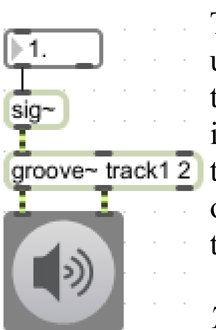
Using the object *sel* or *select*, followed by the ascii number, will route the key pressed out through a specific output, instead of just starting everything at the same time.
So connecting this to something which will play, will of course cause it to play.



Now we would like something to interact with and setup a buffer~ object and a groove~ object. The ~ meaning that the object is treating audio or audiofiles as in and / or output instead of just number / midi messaging. The buffer can store an audio file, and the groove~ will playback what is in the buffer, if they are named the same. So let's name them buffer~ track1 2 and groove~ track1 2. (Track1 is the name of the buffer, and the number 2 after the name is the amount of audio channels in the buffer – as well in the plaback object groove~ if we are certain that the music which we input have two channels / stereo / dual-mono etc.) Most objects can be controlled by so-called messages sent to them, and these are easily made by pressing >>m<<, and type whatever message you like, or you can hoover the mouse over the input of an object and get a list of inputs which the object can receive, in this case the message >>replace<< sent into a buffer~ object, will replace the current audio file stashed inside it with a new one. Opening a window in which to choose the file to open.

Replace changes the file loaded in buffer~

Groove~ needs a control signal to play, and straight playback is the number 1, but altering this number will cause in a pitch change. 2 meaning double the playback speed and 0.5 meaning half the playback speed..



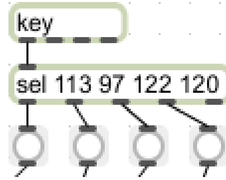
This is a signal code, being sent at the speed of the sampling rate of your sound, usually 44,1 kHz or 48 kHz. So if the input is 1 then it will be 44,1 kHz * 1, giving the same playback as intended originally in the audio file. Max/MSP works out by itself, if the number 2 is sent and then doubling the playback speed, even though there are not double as many samples in the file. The number 1 must be sent to the object sig~ and then into groove~. The sig~ object just repeats the number given at the current samplerate as mentioned before.

The number 1 sent from sig~ to groove~ results in playback at normal speed.

Instead of just sending a message with our number 1 into the sig~ object, using a variable number as a number box, float or integer box is more flexible. The float object for more precision. Remember, you can always press ALT or command (on a mac) – and double click any object to get a very useful reference patch opened, which will quite well describe what any object does if used as intended.

So creating a float box by pressing >>F<<, and routing the signal of buttons >>Q<< and >>A<< into adding or substracting a given value from the original number, we now have a controlled pitch value. And letting >>Z<< and >>X<< control start and stop, by letting X drop the pitch value to 0

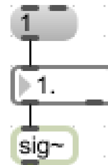
and Z put the value back to the initial number. Try it out by creating a key object and find the keycodes of these keys, or do it the easy way and copy it from the figure here:



The sel object routing out the signal, when keys Q,A,X and Z is pressed.

Q pitches up, A Pitches down, X stops playback and Z resumes. If we route it in this order.
To do this, we need to be able to add a certain value to the current pitch, so we need to store the original number and add something to it. The reason for storing the number is of many reasons and one of them is that if we need to pause the playback, then we can return to the pitch we came from instead of returning to normal playback pitch and having to manually remember at which pitch we where, just two seconds before having to go live and into the mix.

Storing a value, can be done with the object float. Not to be mistaken for a float value object.
Let's create a message, which starts normal playback, by making a message which sends the value 1 into our number box controlling the sig~ object.



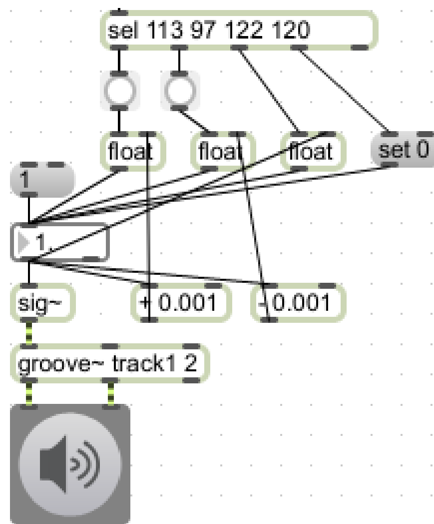
This message of 1, can now be clicked with the mouse when the patch is locked, to start playback – it can be linked to any input if you like.

Now, we shall add and substract a value of 0.001 to the value sent to the sig~ object. We do this by sending the original number in a + 0.001 object's left inlet. Note than most objects also do the math or functions, whenever information is received. + 0.001, means that whatever we send into the left inlet is instantly added this value. - 0.001 would of course substract the value from the number. So making both objects already, wil give us the possibility to both increase and decrease our pitch control number. Sending the result of the new float value to a float objects right inlet, will store the value inside the float object. If the left inlet receives a signal, from our pressed key Q or A, the value will be sent out through it's output, which we can then route to our pitch control value. In this way we make sure that our value is stored and ready to be used whenever we need it, but will never stack or overflow, or even be called, if we do not press the key.

Now connecting our sel output, from the key inputs into the float object, we can now control this. Adding an extra output from our pitch control number into a third float object will let us store the current value instead of a slightly increased or decreased value, this will allow us to add the functions of the extra two keys Z and X into the patch but connection Z to that float objects left inlet and X to a message >>set 0<<, set meaning that the number 0 is sent to the object but no further, a great way to control the signal path. This will allow our playback to stop whenever X is pressed and to resume whenever Z is pressed. Their values are 122 and 120.

Already now a patch can look a bit messy, so encapsulating some of the objects is a good idea, also aligning or rerouting the patch cords can be done by pressing ctrl-shift-A or Y.

This allows us to playback the audio inside our buffer and we can increase and decrease the pitch, pause and resume playnack, just by using these four keys. Q,A,Z and X.

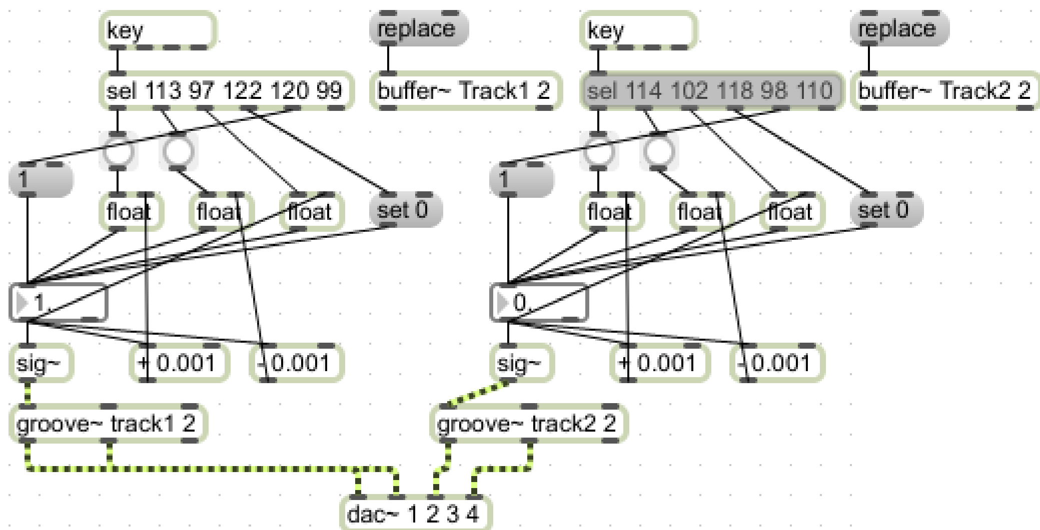


Basically all we need to do now, is dublicate the patch and make sure that our key input is different for this one, and that our buffer~ and groove~ objects are named track2 2 in their extension instead of track1.
But the little message of 1 which is out there to the left, could become of use, because what if you are already pitched up or down and you don't have time to press the Q or A key 25 times, then having a key which resets the pitch, might be a good idea. A little help from an experienced player, would be to place the reset key away from the other pitch keys, so let's choose key C for this.
Ascii value of C is 99, so sel 99 and dragging from the newly created outlet of our sel object over to the message box containing our number 1, we can now reset the pitch.

The setup works fine so far, but when we add track2 to the patch, we shall need more outputs, taking that you have more than just a stereo out on your computer of course.

Our current output is the grey icon with two inlets. This object is actually named ezdac~ and doesn't provide any ability to output more than stereo.
There is another output, dac~ (which stands for Digital to Analog Conversion – simply put from digital computer to real world audio)

this object can be added the extension of as many as you like outputs, let's just name them 1 2 3 and 4. so dac~ 1 2 3 4, gives us an object with four audio inlets, which internally in max/MSP can be routed to whatever output you desire, in our case probably the first four outputs of your soundcard. So let's connect the two outputs of track1 to the first two inlets instead. And our track2 to the third and fourth inlet. Creating a new key and sel object to control the control inputs of track2 can be a good idea, but they could also just be added to the other key and sel object, but to make it less visually confusing we'll do it the other way here. And use keys next to the other set, so R,F,V,B and N.



So now we have a fully functional little DJ tool, without too many gadgets and gizmos connected to it. Connect the four outputs of your soundcard and cue up some music in the track1 and track2 buffers. Play them and mix them, any way you like.

But playing on your laptop, with the keyboard can get alittle messy, and a little annoying at some point when the guys up on the stage do nothing but stare into the laptop to make sure that they press the right buttons.

And we aren't really done with adding gizmos.