# Max and Pitch

Some of this appears in the introduction. It's important, so review it again.

## Representation of Pitches in Max

In the Max environment, pitches are necessarily represented as numbers, typically by the MIDI code required to produce that pitch on a synthesizer. We must begin with and return to this representation, but for the actual manipulation of pitch data other methods are desirable, methods that are reflective of the phenomena of octave and key.

A common first step is to translate the midi pitch number (mpn) into two numbers, representing Pitch Class (pc) and octave (oct) this is done with the formulas[1]:

oct = mpn / 12
pc = mpn % 12

The eventual reconstruction of the mpn is done by

mpn = 12*oct + pc

In this system pc can take the values 0 - 11, in which 0 represents a C.

| C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Oct ranges from 0 to 10. Middle C, which is called C3 in the MIDI literature and C4 by most musicians, is octave 5 under this formula.

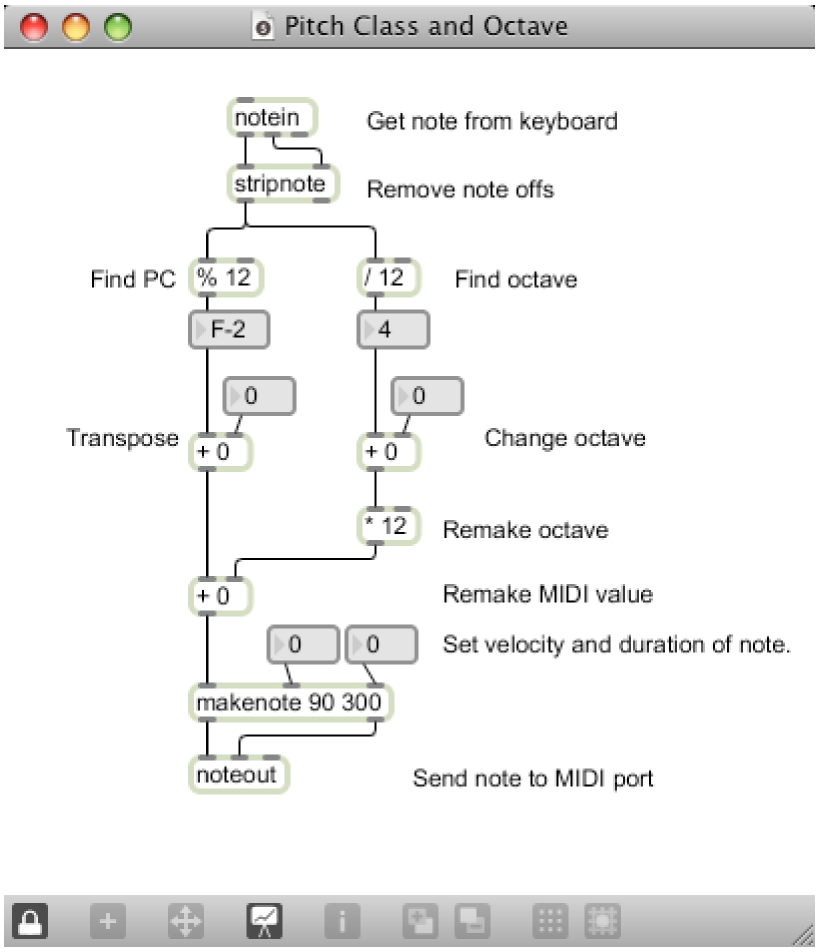
Once the pc is split from its octave, a variety of manipulations can be undertaken. For instance, to transpose, you add the appropriate number of half steps. To go up a fifth (7 steps) from D (pc=2)

new pitch = (old pitch + steps) % 12

gives 9 (A) as the answer. The rem 12 is necessary to keep the answer within the range of 0 to 11.

To transpose down, you add the 12's complement (12-n) of the number. Down a fifth starts with the complement of 7 which is 5. A fifth below D comes out (2+5)%12 or 7 (G).

[1] The / function is the integer divide, which throws away the remainder. The % operator is the rem function, which divides by its argument and returns the remainder. Thus 5 % 2 = 1. This is often confused with modulo, which is counting around a clock face. This gives different answers when negative numbers are used: -10 mod 12 is 2 whereas -10 % 12 is 10. Music really requires modulo, but it's not included in many computer languages. We avoid problems by keeping all pitch classes positive.

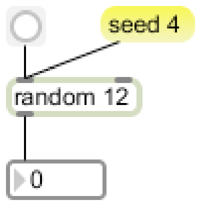


This patcher illustrates the principles of extracting and manipulating pitch classes.

## Generating Pitches

The notein object is not the only way to create notes in Max. Here are some strategies to play with:

The **random** object will create apparently random numbers. The numbers are actually the result of a mathematical formula applied to the last number produced. The values change in a way that is not easily predicted, and if the numbers are studied over a long period of time any one is just as likely as any other.



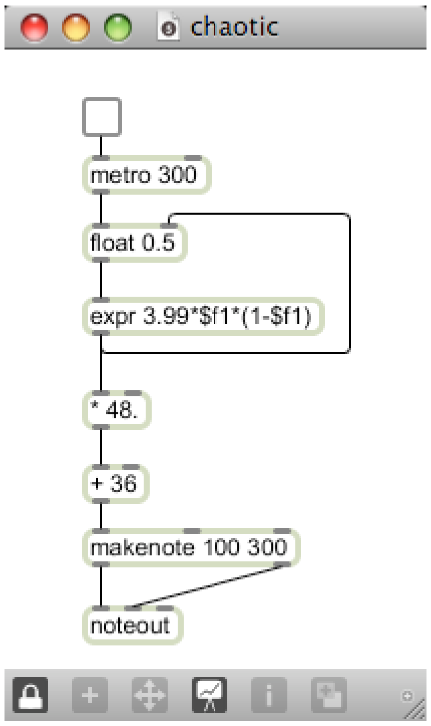
You can get repeatable sequences with the seed function. The message "seed n" (with n any number but 0) will restart the formula with the seed value. Seed 0 uses a number based on the time the computer has been running, which will not be predictable. Seed 0 is the default.

The **urn** object selects from a series of numbers in a random order. (Like dealing cards) When all have been output, the right outlet bangs to tell you. A clear message will start the dealing again.

The **decide** object just gives 0 or 1, like flipping a coin. Decide performs this task better than random 2 because the random algorithm has a slight preference for odd values.

The **drunk** object executes the "random walk" procedure. In this, the output is a random distance from the last output. The noise is "Brownian", and is sometimes interesting, but pitches repeat a lot.

There are a lot of ways to generate **fractal** note patterns. There's no object per se, but **expr** allows you to use any of the classic fractal formulas. Here's a patcher using one:



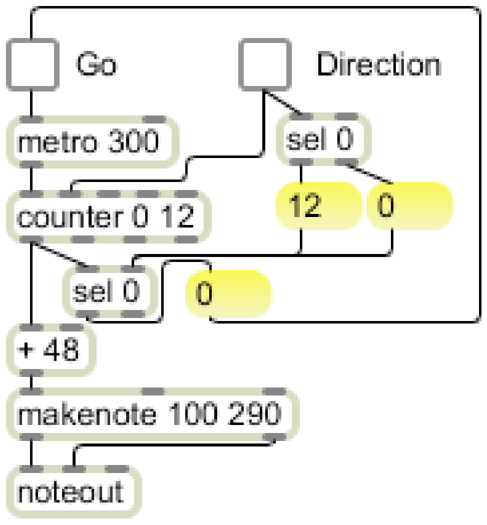
This will always play the same pattern of notes, but that pattern defies description. The pattern you get depends on the contents of the float object. If you seed it with a random number between 0 and 1 each time the patch is loaded, the results will always come out different.

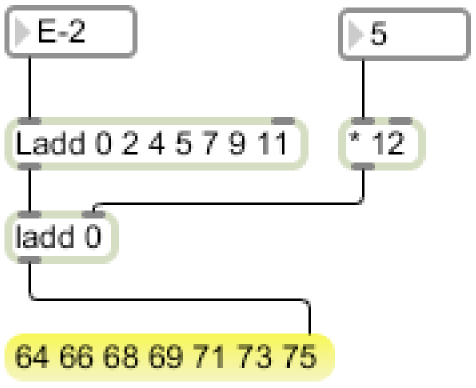
We multiply by 48 and add 36 to get the outputs into interesting octaves.

For more about this kind of process, see the essay Max & Chaos.

## Generating Scales

You can make chromatic scales with the **counter** object. It has inlets for direction as well as beginning and ending values so you can make very complex arabesques. The patcher shown just gives the basics, a scale up or down. The select objects are there because a long standing bug in counter makes it difficult to detect the end of the count. Instead of using the overflow and underflow outlets, you have to detect the last number of the count series and use that to turn the metro off.





A major scale is represented by the numbers 0 2 4 5 7 9 11. If we keep these in a list, they are easy to manipulate with Lobjects. For instance, we can shift the scale to any octave just by adding a multiple of 12. We can transpose it to any key by adding the pitch class that represents the key desired.

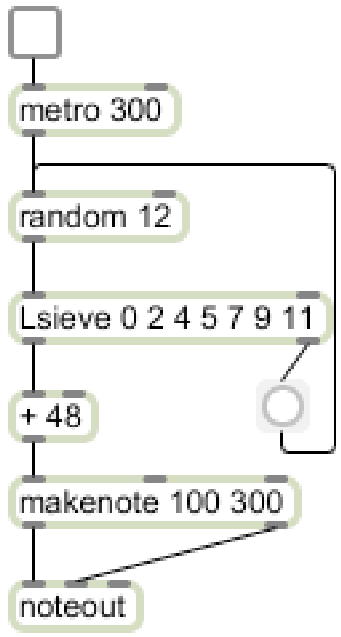
**Scale Degrees**
In theory class, we learned to refer to notes of a scale as first, third, seventh etc. with the added modification of major or minor. There's not a lot of call for this in computer music, but if you wish, you can use a coll to retrieve the numerical equivalent of any label. The contents would look like this:

```
do, 0;
re, 2;
mi, 4;
fa, 5; etc.
```

This can work either way.

## Processing Random Pitches

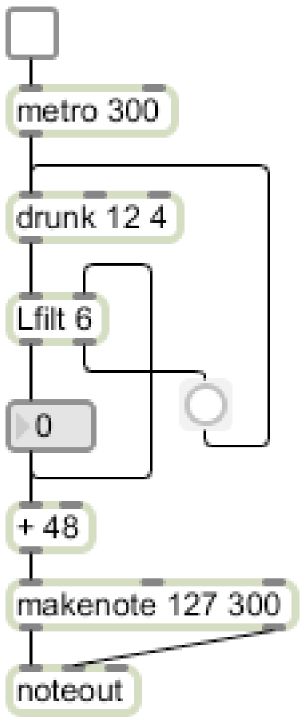
Randomness is most effective when tamed by some musical rules. A very powerful rule type is the "sieve", which lets some notes through, but rejects others. The **Lsieve** object does this handily. Lsieve 0 2 4 5 7 9 11 will only allow the notes of C major through.



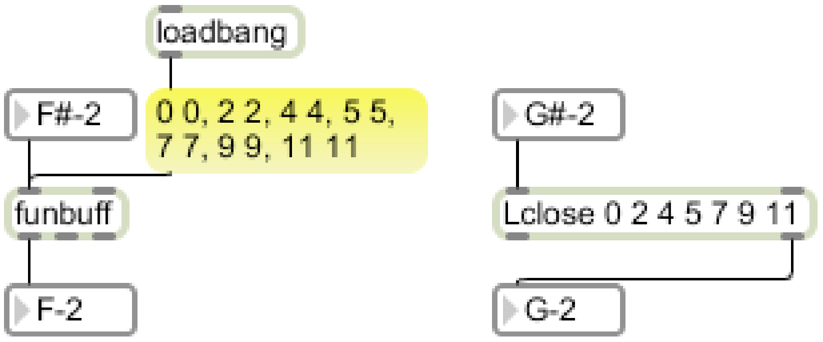
Any note that fails the test in this patcher will cause the random object to try again, because failed values fall out the right outlet. You need to be very careful when using this type of feedback process. If Lsieve were to reject everything random puts out (for instance if it had all values higher than the range of the random object) the patcher would go into an endless loop and a stack overflow would occur. A safer approach would be to omit the feedback (leaving holes in the stream of notes) or to trigger some other process to create a note.

The secret to generating an interesting piece with sieves is to make the sieves change in some way. The values accepted by Lsieve can be changed by sending a list in the right inlet.

The Lfilt object has a complimentary action. It will reject whatever is in its argument list. The values to reject can be changed by sending a list to the right inlet. What does this patch do?



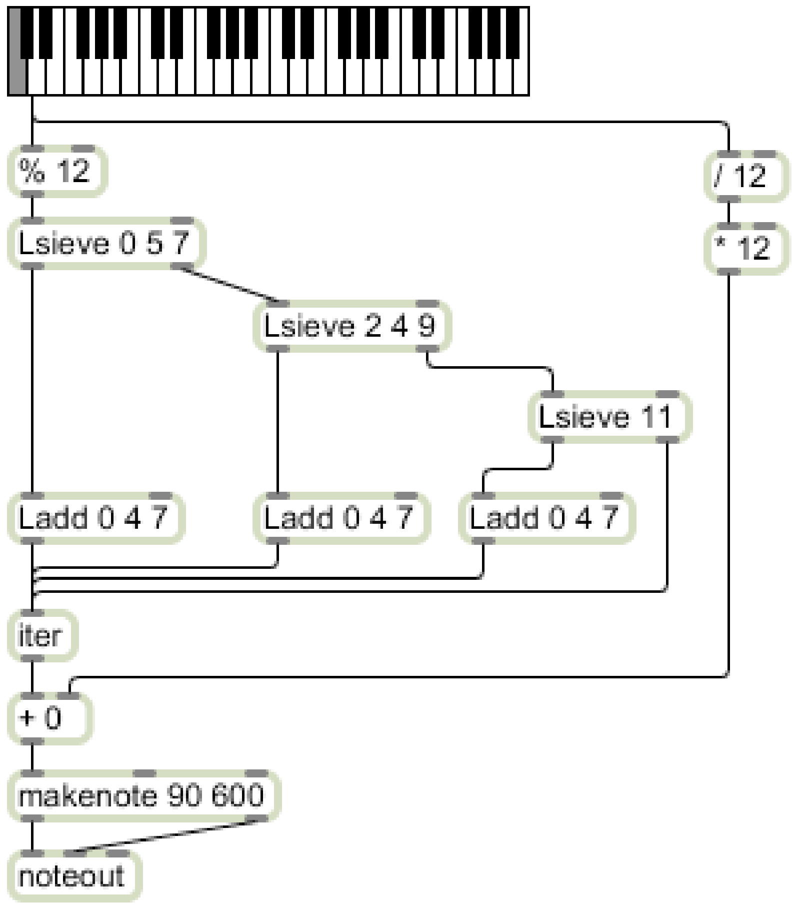
The Lsieve and Lfilt objects work by throwing data away. Another approach to the constraint problem is to change unwanted data somehow. A simple way to do this is with the funbuff object. The funbuff stores a series of pairs (lists with two members). Once the pair has been input, funbuff will produce second value in the pair any time the first comes in. If an input value is not in the funbuff, the next lower input value that is in there will be used. This patcher will keep everything in C major:



The Lclose object performs a similar function.

## Chords

Playing block chords is easy in Max, all you have to do is send the individual notes to makenote at the same time. Of course, all actions in Max are really carried out one at a time, but this can occur so fast that chords sound simultaneous to our ears. The main difficulty is figuring out whether to send a major or minor chord for each scale degree. Here is a basic approach:

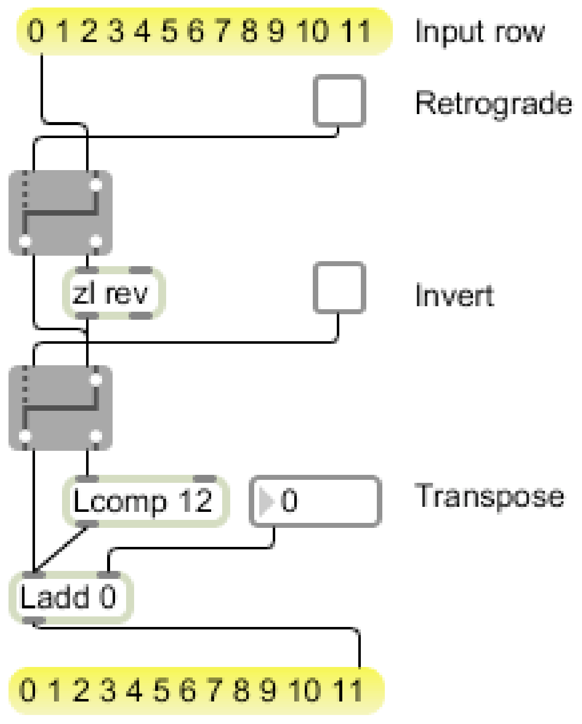


The Lsieve objects sort the scale notes according to the chords they should have in Cmajor. A "wrong" note is played, but doesn't get a chord.

The Ladd objects create the chord as a list. The pitch class input is added to each member of the initialized list. The iter object turns this list into three individual pitches.

All else is as described earlier. There are more approaches to chord building in the Max & Chords tutorial.

## Serial Manipulations



Here is a patcher that demonstrates how to do traditional row manipulations. Zl rev reverses lists. The graphic gate is used here to turn it on or off. I used the graphic gate rather than switch or gate because the latter are turned off when a patcher is first opened.

Lcomp 12 will provide the inversion of the row. There is an Linvert object, but the math definition of inversion is not the same as that in music. Musical inversions are produced by subtracting the pitch class from a constant (12), which is really the complement operation.

Ladd provides any desired transposition.