An overview of LotusRoot

Takumi Ikeda (2021-03-23)

LotusRoot is a Ruby program that generates musical notation from arrays of data such as pitches and durations.

Specifically, the program generates notation scripts for LilyPond, a text-based music engraver.

Requirement

LotusRoot requires Ruby (2.2 or later) and LilyPond (2.18.2 or later) to be installed.
Also, installing Frescobaldi is recommended, an editor for LilyPond, although it is not required.

Ruby	https://ruby-lang.org
LilyPond	http://lilypond.org
Frescobaldi	http://frescobaldi.org

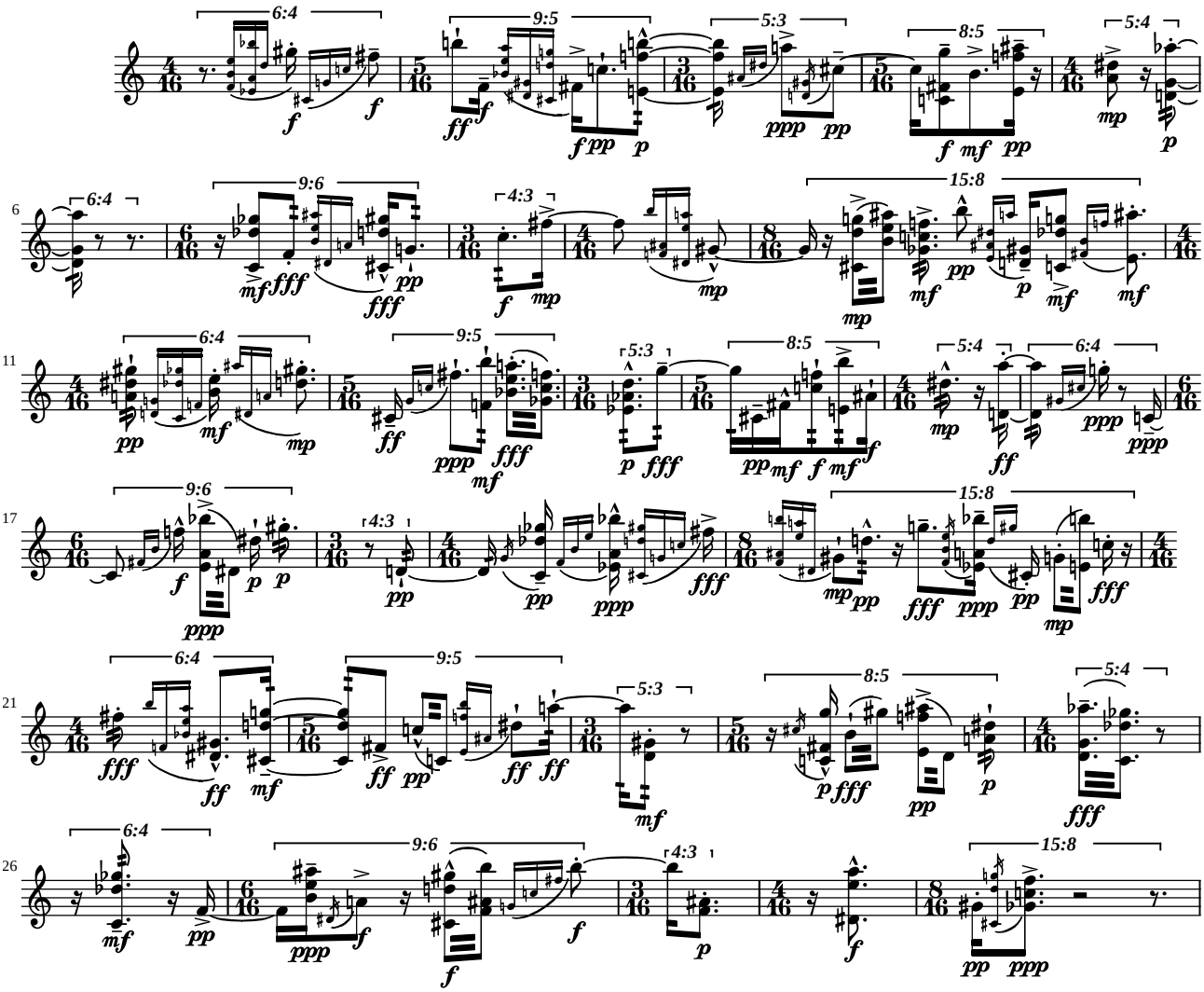
Terms of use

How to use

Run the following commands from cmd.exe (for Windows).

```
cd <path>\LotusRoot\doc
ruby test_mess.rb
lilypond test.ly
```

(output example)



Usage

The following is an explanation, along with sample files. Refer to README.txt for all functions.

00_input.rb

To use LotusRoot, load LotusRoot.rb in your Ruby program.

```
require_relative 'bin/LotusRoot'
```

There are four basic types of data given to LotusRoot.

- *Durations*
- *Elements* (Symbols for notes, rests, sustain and so on.)
- *Tuplets*
- *Pitches*

Elements are written as an array of strings, while others are written as an array of numbers, including integers, floats, and rational numbers.
Arrays other than *Elements* are referenced repeatedly. Therefore, even if the array size is 1, it will still work (the same content will be repeated).
Elements cannot be omitted. In other words, notes and rests are placed according to the number of *Elements*.

This example shows that *Elements* is mainly composed of three types of symbols.

`"@"`	The "attack" of a note
`"="`	The "sustain" of a note
`"r!"`	Rest

For example, the first note in this notation is described by one attack and 13 sustains.
Durations define the length of each *Element*. In this case, all *Elements* are of length 1, so the first note of this notation will have a total duration of 14.
Tuplets represent the number of quarter note divisions when integers write them. Thus, the minimum note value in this notation is a sixteenth note, and the first note of this notation is the duration of 14 sixteenth notes.

`"@"` and `"r!"` can contain any string, such as the LilyPond command. For example, `"@\\trill"` will add `"\trill"` after the LilyPond script. Note that an additional backslash is required to represent the backslash.

```
sco = Score.new(dur, elm, tpl, pch)
```

The above line is used to create the Score object and input the basic data.
The name of the basic data arrays can be anything, but let us say `dur`, `elm`, `tpl`, and `pch`.

```
sco.gen
```

The above line generates the notation script.

```
sco.export("sco.txt")
```

The above line exports the notation script as a text file.

The output file will be read in the next line of test.ly.

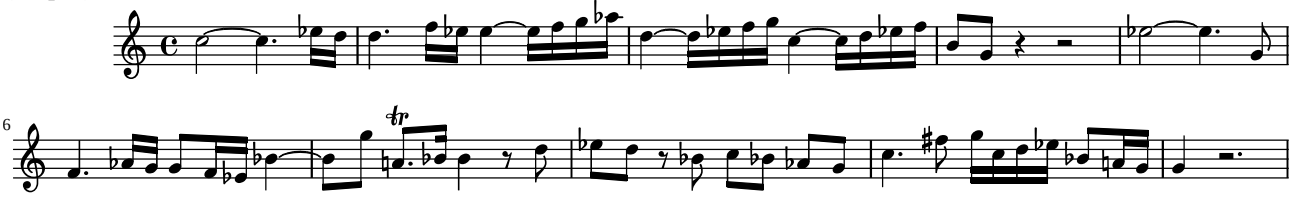
```
\include "sco.txt"
```

The file name can be anything, as long as these file names match.

```
sco.print
```

The above lines will display the notation script at the command prompt.
The other lines are options for adjusting the notation script. See below for details.

(output)



 # 01_input.rb

The musical notation is the same as 00_input.rb, but *Element*s and *Durations* are different.

```
elm = ["@"]
dur = [3]
```

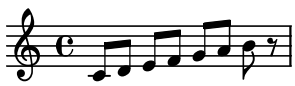
When this data is input, LotusRoot will expand it internally as follows.

```
elm = ["@", "=", "="]
```

 # 02_pitch.rb

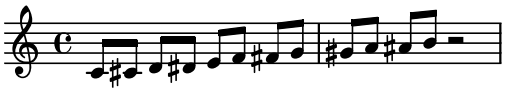
Pitches represent chromatic tones by integers, and quarter tones or eighth tones by floats or rational numbers.
Chords are described using nested arrays.

```
pch = [0, 2, 4, 5, 7, 9, 11]
```

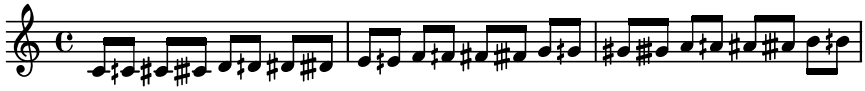


After that, try to run the program while uncommenting the commented-out lines one by one.

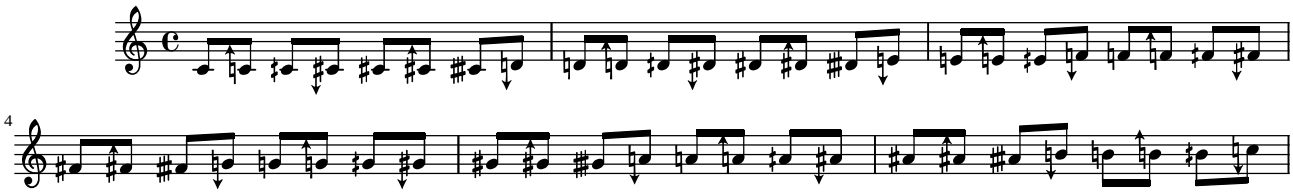
```
pch = [*0..11]
```



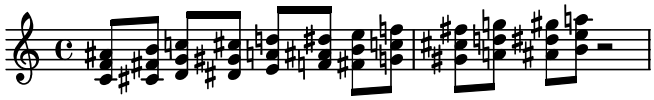
```
pch = [*0..23].map{|e| Rational(e, 2)}
```



```
pch = [*0..47].map{|e| Rational(e, 4)}
```



```
pch = [*0..11].map{|e| [0,5,10].map{|f| e+f}}
```



For Ruby beginners, we will discuss the Ruby notation used here.

`[*0..11]` converts Range objects `0..11` to Array`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]`.

The `map` method converts the contents of an array. The format is as follows. It executes a *`block`* for each content *`item`* in the `array`.

```
array.map{|item| block }
```

The following will output the contents of `pch` to the command prompt.

```
p pch
```

The following is a `map` to `pch`, but only to repeat `"@"` the same number of times as the size of `pch`, ignoring the contents of `pch`.

```
elm = pch.map{"@"}
```

Now, we will discuss LotusRoot's options.

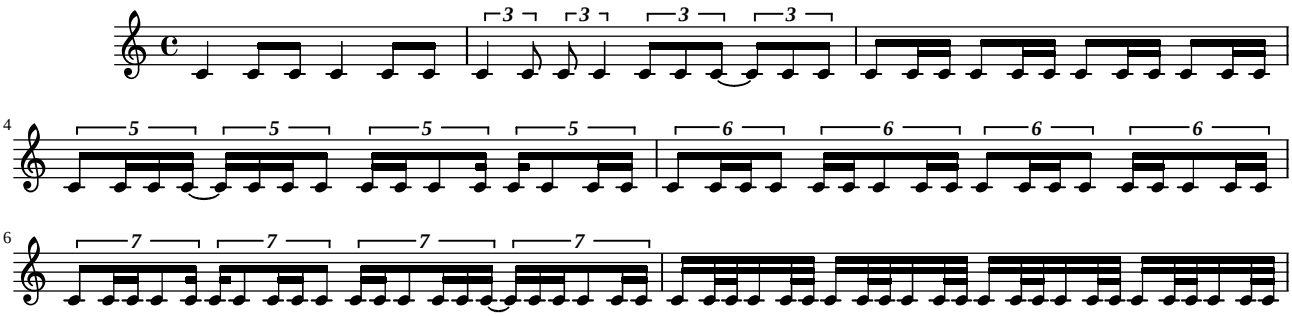
The `pitchShift` option adds a specified number to the entire *Pitches*. In other words, it transposes the *Pitches*.

The `accMode` option selects the type of accidentals: `0` for sharps (including microtones), `1` for flats. `2` and `3` for sharps and flats, but they avoid 3/4 tone notation.

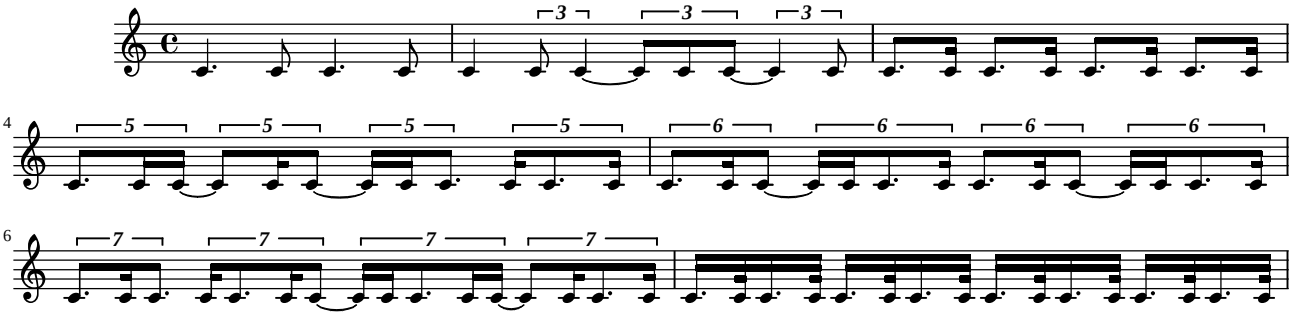
 # 03_duration.rb

Durations define each *Element*'s durations, specifically by how many of the minimum note values defined in *Tuplets*.

Even if the value of *Durations* is the same, the result differs depending on the *Tuplets*. In this case, *Durations* is repeating `[2, 1, 1]`.



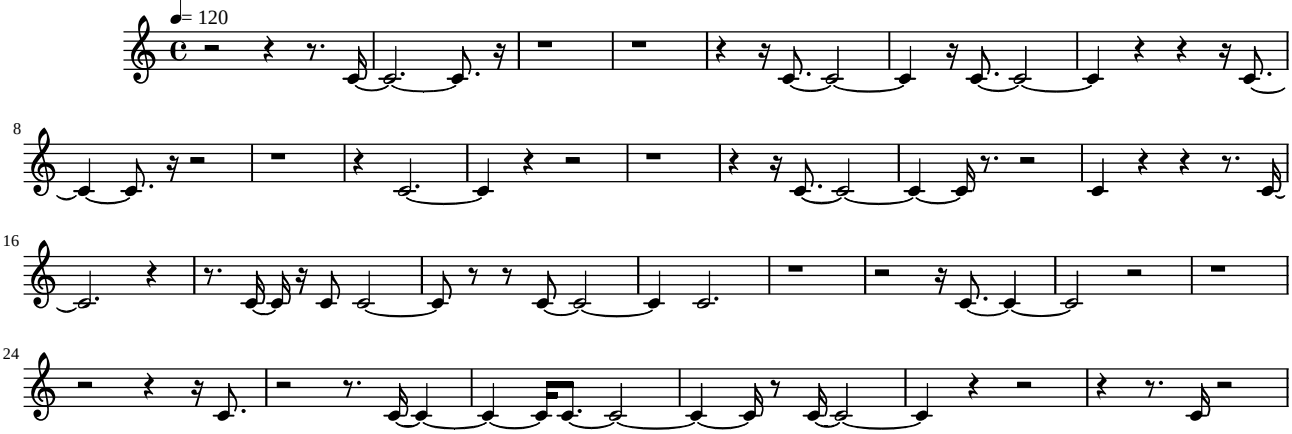
If we change *Durations* to `[3, 1]`, the second bar's first beat is for three of the minimum note value 1/3, resulting in a quarter note.



Besides, LotusRoot internally processes all note values as rational numbers.

 # 04_element.rb

(example)

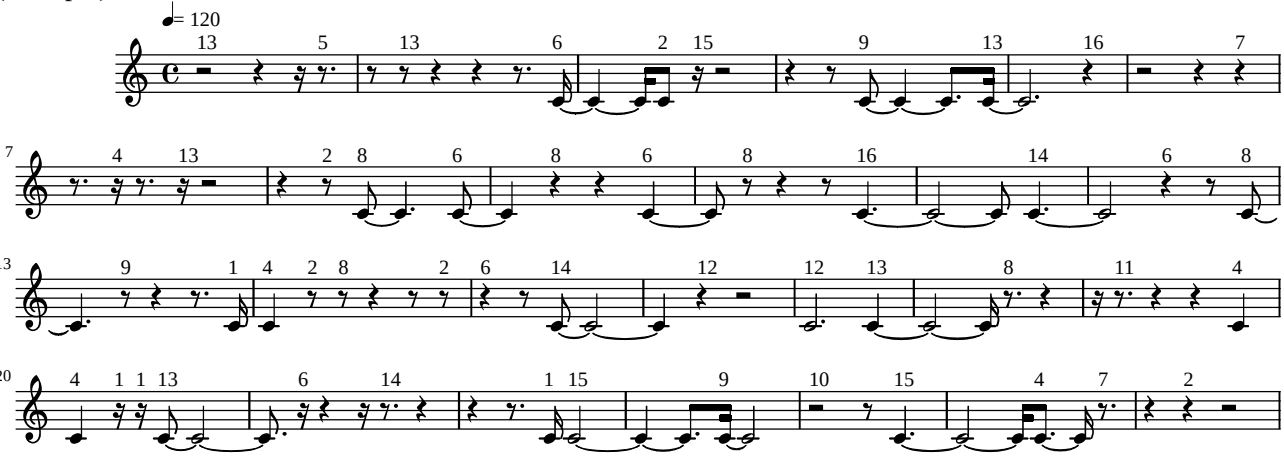


In this example, either `"r!"` or `"@"` is randomly applied to the random durations.

```
elm = elm.zip(dur).map{|e,d| e+"^\\markup{#{d}}"}
```

This line shows how to display the values of *Durations* above the notes. `#{}` is a Ruby notation that evaluates variables in a string.

(example)

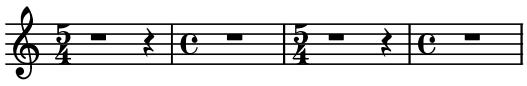


`TMP4;120;` is LotusRoot's command to insert the metronome mark.

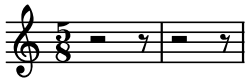
 # 05_metre.rb

The default time signature is 4/4, but it can be specified with the `metre` option. It is written as an array (like *Tuplets*) and is repeatedly referenced.

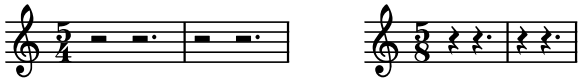
If an element is described by an integer N, it shall be interpreted as N/4 metre.



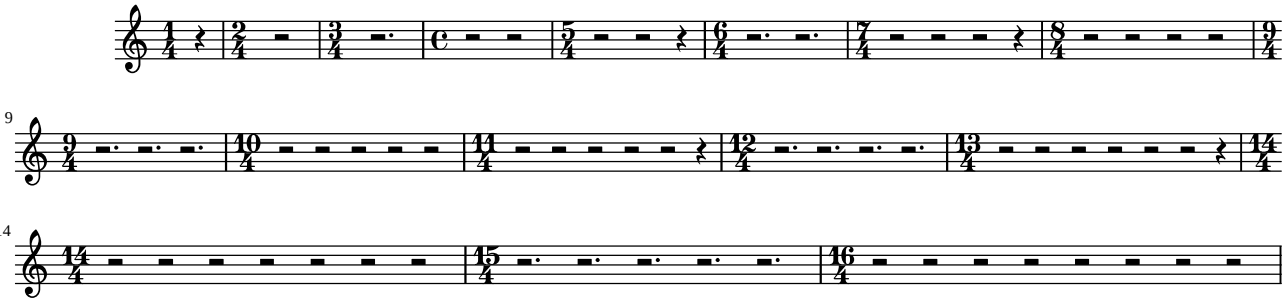
When written in the form `[[beat], unit]`, `[beat]` will be the numerator and `unit` will be the denominator.



A `[beat]` is an array of integers, a single integer or a combination of integers to define the beat structure explicitly.



A `unit` is an integer or rational number with a quarter note as 1. `1/2r` means a rational number 1/2 in Ruby notation, which in this case means the eighth note. In the case of simple (non-explicit) notation, the beat structure is automatically divided as follows.



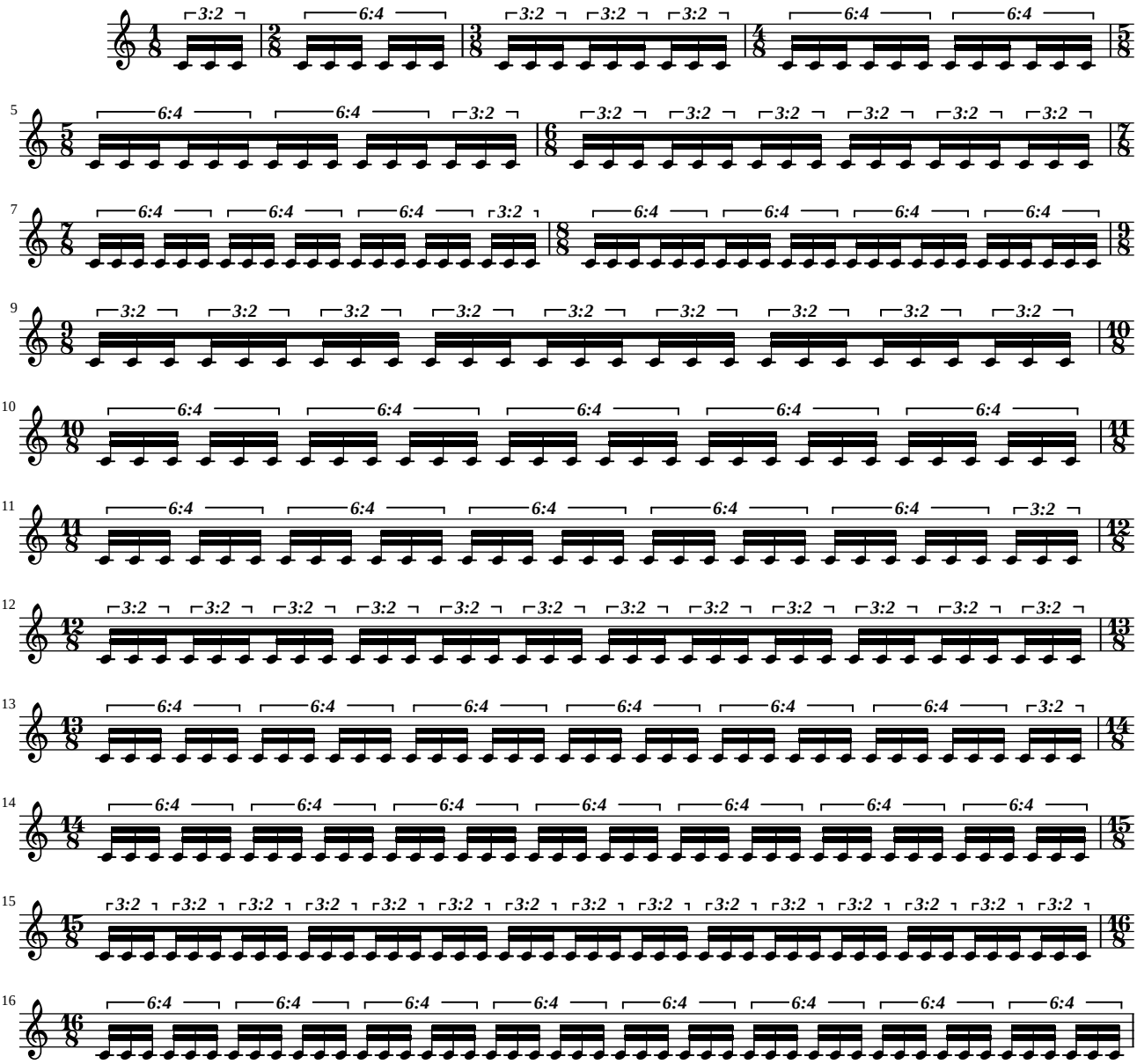
 # 06_tuplet.rb

Tuplets defines a series of tuplets. When written as an integer N, a beat is divided into N. In this case, the behaviour changes depending on the time signature defined in the `metre` option.

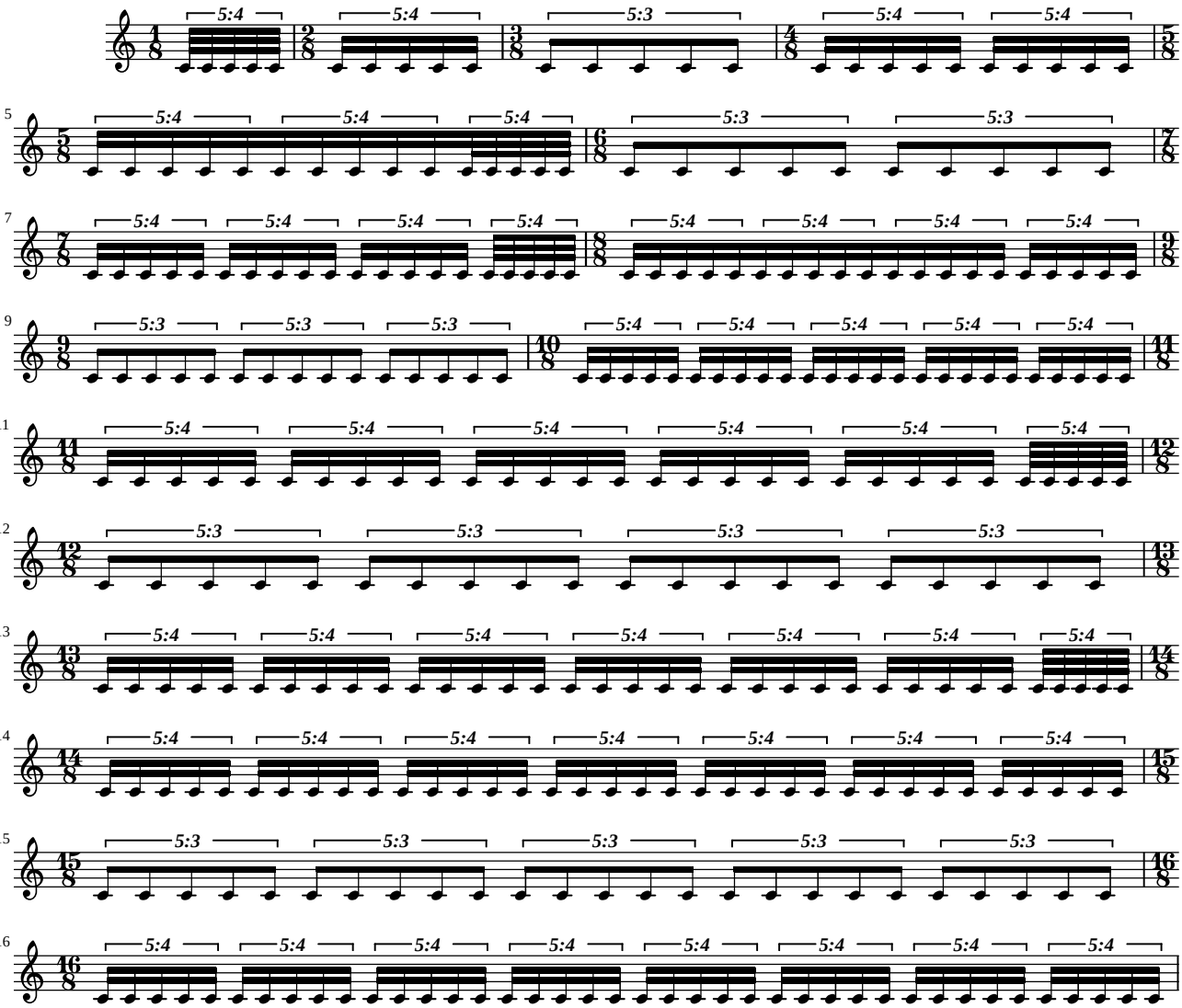
A beat is basically a quarter note, but if a bar is not divisible by a quarter note,
- If the tuplet can be split, fills the bar with the split tuplets (`tpl = [6]`).
- If the tuplets cannot be split, divides the beat according to the beat structure (`tpl = [5]`).

This is so that the processing can be done without changing the note value as much as possible.
If `tpl = [4]`, every bar will be written in 16th note units.

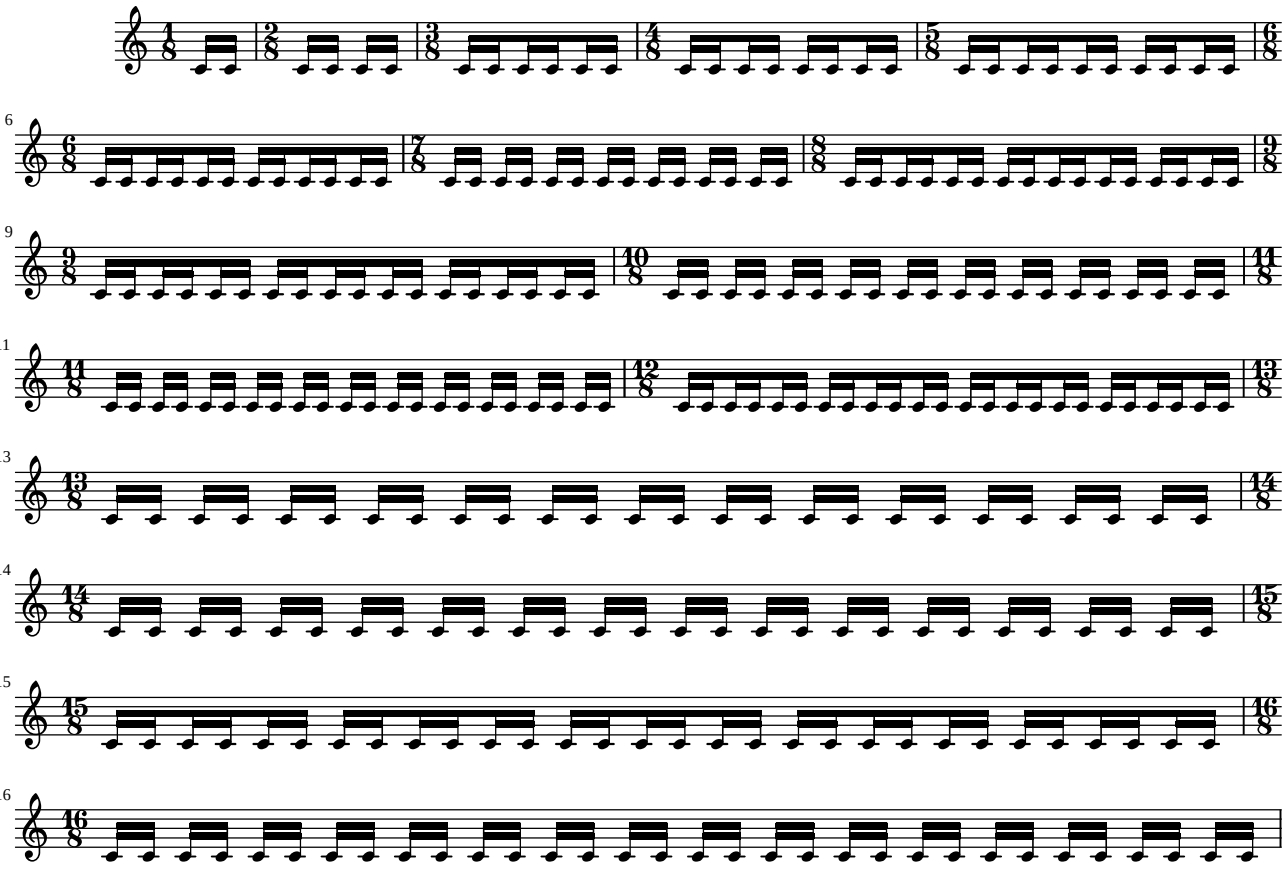
`tpl = [6]`



```
tpl = [5]
```


1 5:4 | 2 5:4 | 3 5:3 | 4 5:4 5:4 | 5
5 5:4 5:4 5:4 | 6 5:3 5:3 | 7
7 5:4 5:4 5:4 5:4 | 8 5:4 5:4 5:4 5:4 | 9
9 5:3 5:3 5:3 | 10 5:4 5:4 5:4 5:4 5:4 | 11
11 5:4 5:4 5:4 5:4 5:4 5:4 | 12
12 5:3 5:3 5:3 5:3 | 13
13 5:4 5:4 5:4 5:4 5:4 5:4 5:4 | 14
14 5:4 5:4 5:4 5:4 5:4 5:4 5:4 | 15
15 5:3 5:3 5:3 5:3 5:3 | 16
16 5:4 5:4 5:4 5:4 5:4 5:4 5:4 5:4

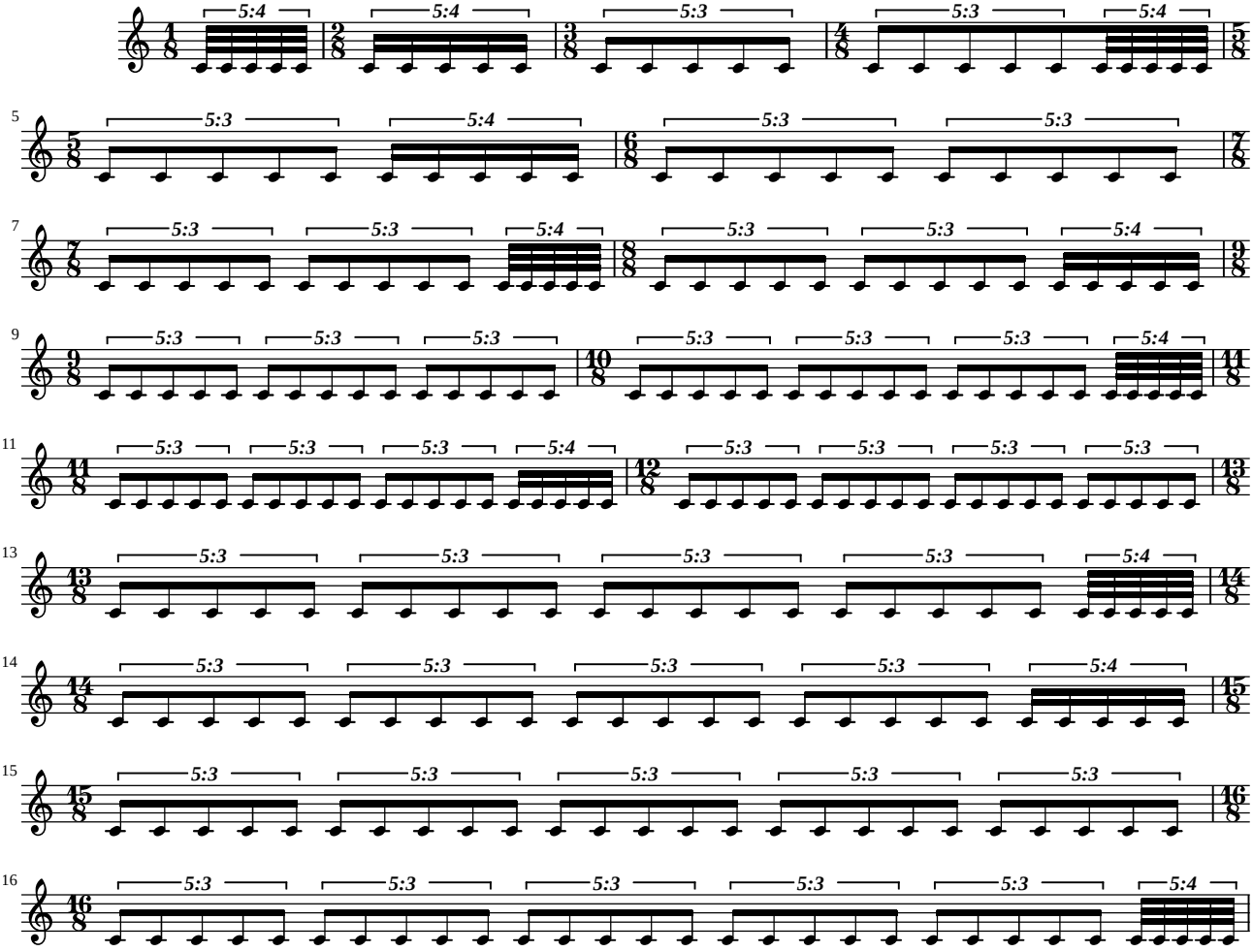
```
tpl = [4]
```



`=begin` and `=end` is Ruby notation for multi-line comments, and comments can be disabled by deleting or commenting out each line as follows.

```
# =begin
      (This part will be executed.)
# =end
```

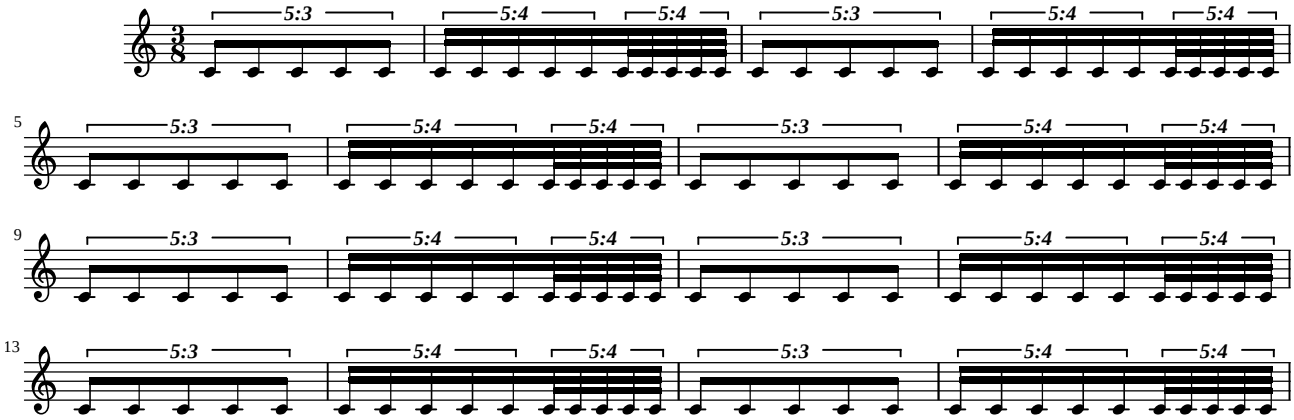
For bars with explicitly defined beat structure, the beats are divided according to the beat structure.



To explicitly define tuplets, write them in the form `[N, D, U]`.

In this case, `N` is the numerator (the number of divisions of the duration), `D` is the denominator (the length of the duration before the divisions), and `U` is the note value unit with a quarter note as 1.

Defining a tuplet that deviates from the bar will result in an error.

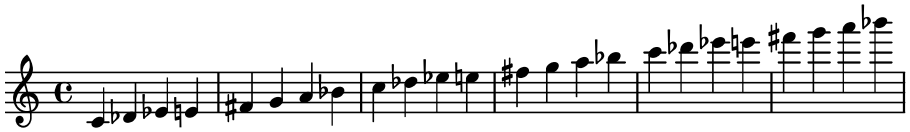


The `fracTuplet` option renders tuplets in ratio notation.

The `finalBar` option terminates the notation script at the specified number of bars.

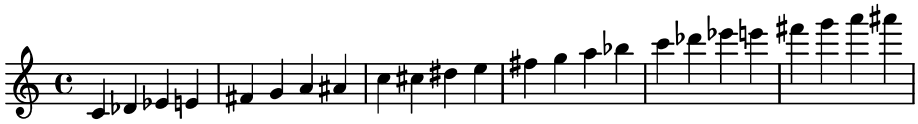
 # 07_altNoteName.rb

The `altNoteName` option specifies note names for the value of *Pitches* directly. It is mainly used to specify accidentals.



Note that in this case, the value of `altNoteName` is specified for the pitches after the `pitchShift` option has changed.

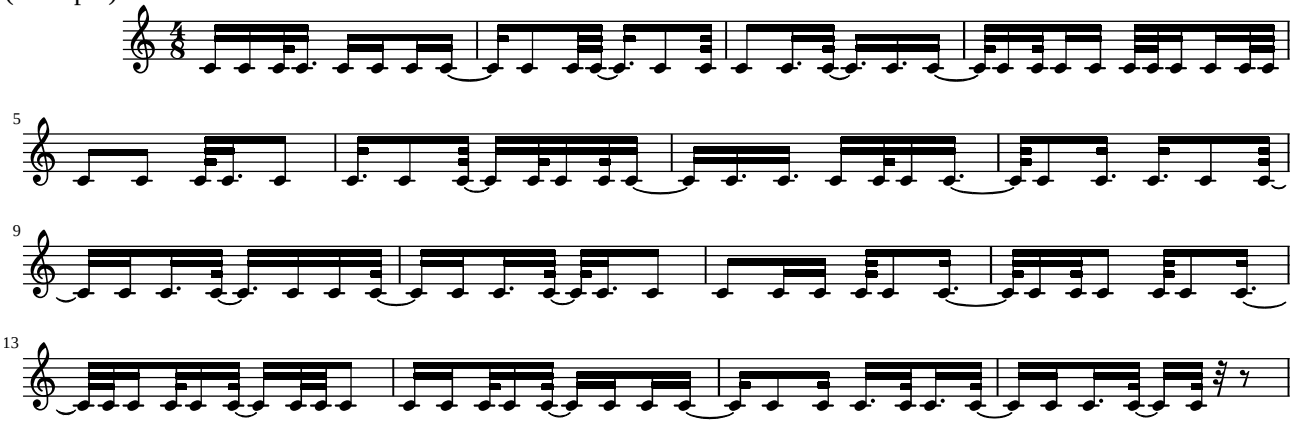
If the pitch specification value is 0~11, it will be applied to all octaves; if it contains a value of 12 or more, it will be repeated for multiple octaves.



 # 08_tidyTuplet.rb

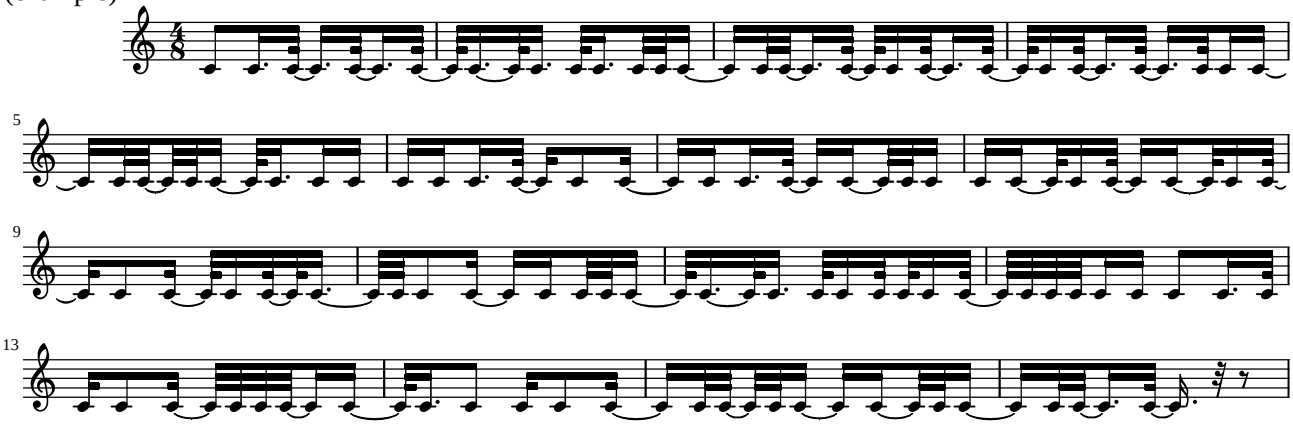
The default output of LotusRoot may be difficult to read in some cases.

(example)



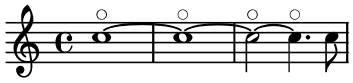
The `tidyTuplet` option attempts to improve readability by grouping consecutive notes (currently, only sextuplets and 32nd notes are supported).

(example)

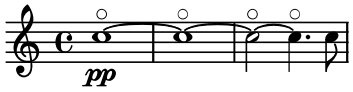


 # 09_tiednotes.rb

Use `"@="` to markup all notes connected by tie.



The commands enclosed in `#A` and `A#` are applied to the notes' beginning concatenated in tie.

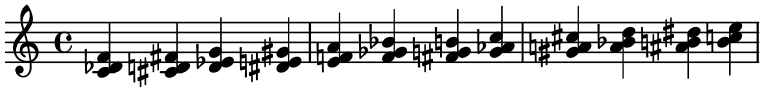


The commands enclosed in `#Z` and `Z#` are applied to the notes' end concatenated in tie.

Note that `#A..A#` and `#Z..Z#` do not work properly with *Elements* other than `"@="`.

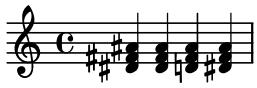
 # 10_autoChordAcc.rb

By setting `autoChordAcc = 0`, the type of accidentals is automatically selected for each chord. Standardizes on either sharps or flats and does not mix both.

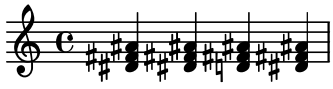


 # 11_reptChordAcc.rb

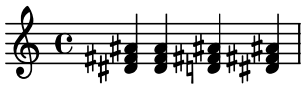
LilyPond uses the `\accidentalStyle` setting (see config.ly) for adding accidentals, but the chords require some trickery.



If `reptChordAcc = 0`, all successive notes in the preceding and following chords are marked with accidentals.



If `reptChordAcc = 1`, the above process is omitted for the sequence of the same chords.

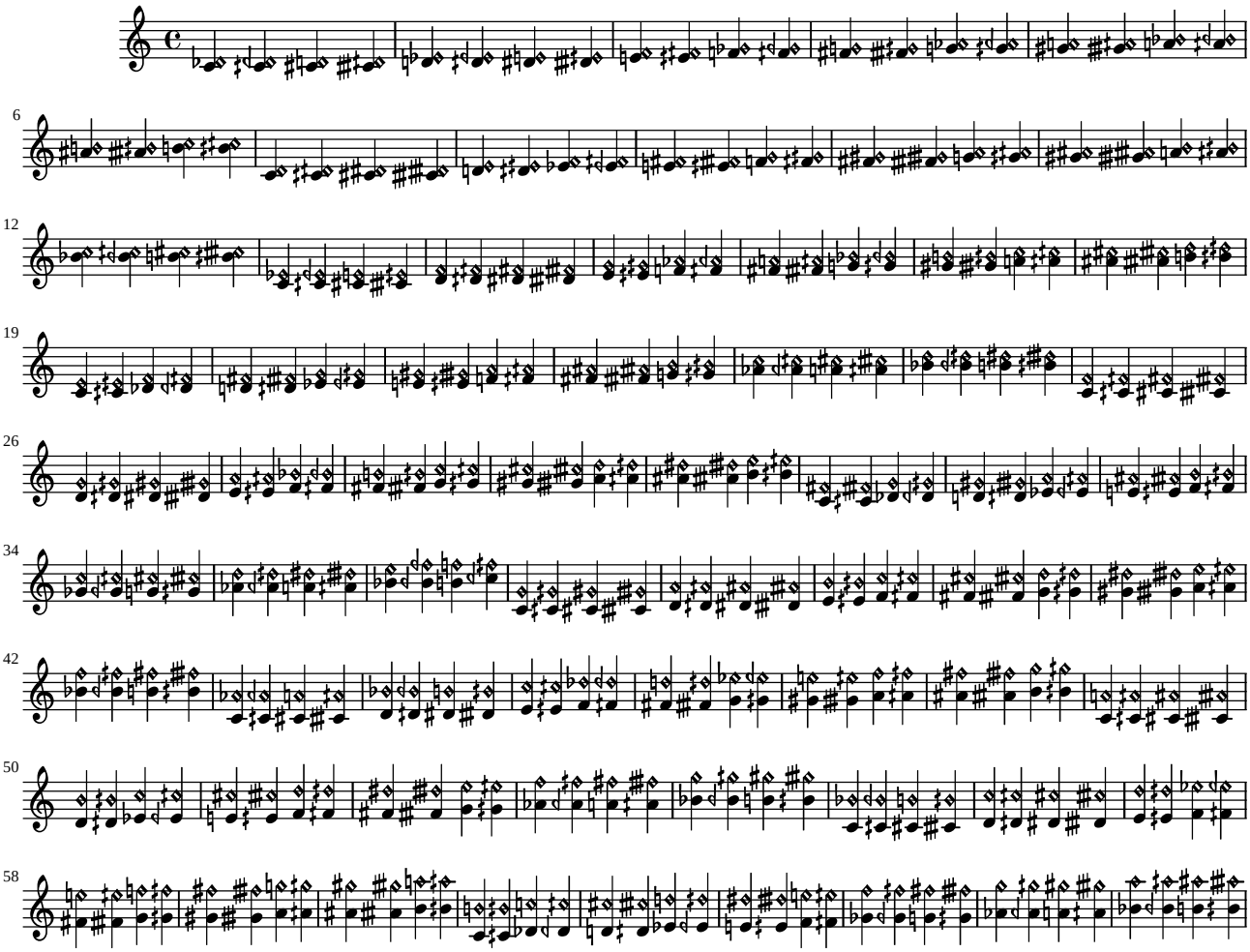


 # 12_artharm.rb

This example shows how to use the `textReplace` option to perform text replacement and write artificial harmonics.

The format of text replacement with regular expressions is the same as Ruby's `gsub` method.

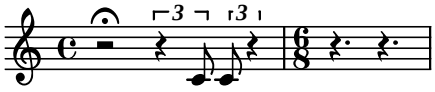
It aligns the degrees of the two notes chords by setting `autoChordAcc = 1`.



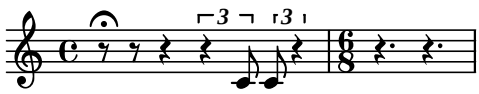
 # 13_rest.rb

`"r!"` or `"s!"` that does not contain any other characters will be automatically concatenated.

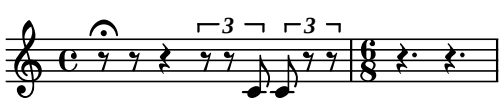
If the input data ends in the middle of a bar, the last bar will automatically be filled with rests.



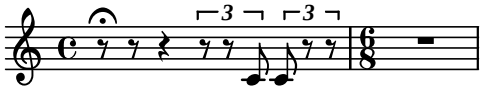
The following is an example of how to split a rest.



The `avoidRest` option splits rests of the specified note value and eliminates them. However, if the rest cannot be split, such as the minimum note value of a *Tuplet*, it is ignored.

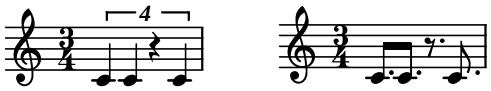


The rests are divided according to the beat structure.
The `wholeBarRest` option engraves whole bar rests (written with an uppercase `R` in LilyPond).

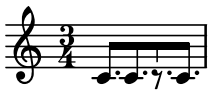


 # 14_misc.rb

The `dotDuplet` option rewrites tuplets like 2:3, as dotted notes.



The `beamOverRest` option connects a beam over rest.



The following will result in an error in test.ly.
The `namedMusic` option specifies the variable name of the notation script.
The `noMusBracket` option will output only the notation script without parentheses `{ }`.