LotusRoot 概説

池田 (2018-03-18)

LotusRoot は音高、持続などのデータ配列から楽譜断片を作成する Ruby プログラムである。
具体的には、オープンソースの記譜ソフト LilyPond のための記譜スクリプトを作成する。

必要ソフトウェア

LotusRoot による楽譜の作成には Ruby（2.2 以降）と LilyPond（2.18.2 以降）のインストールが必要である。
また、必須ではないが LilyPond 用エディタ Frescobaldi のインストールを推奨する。

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

使用条件

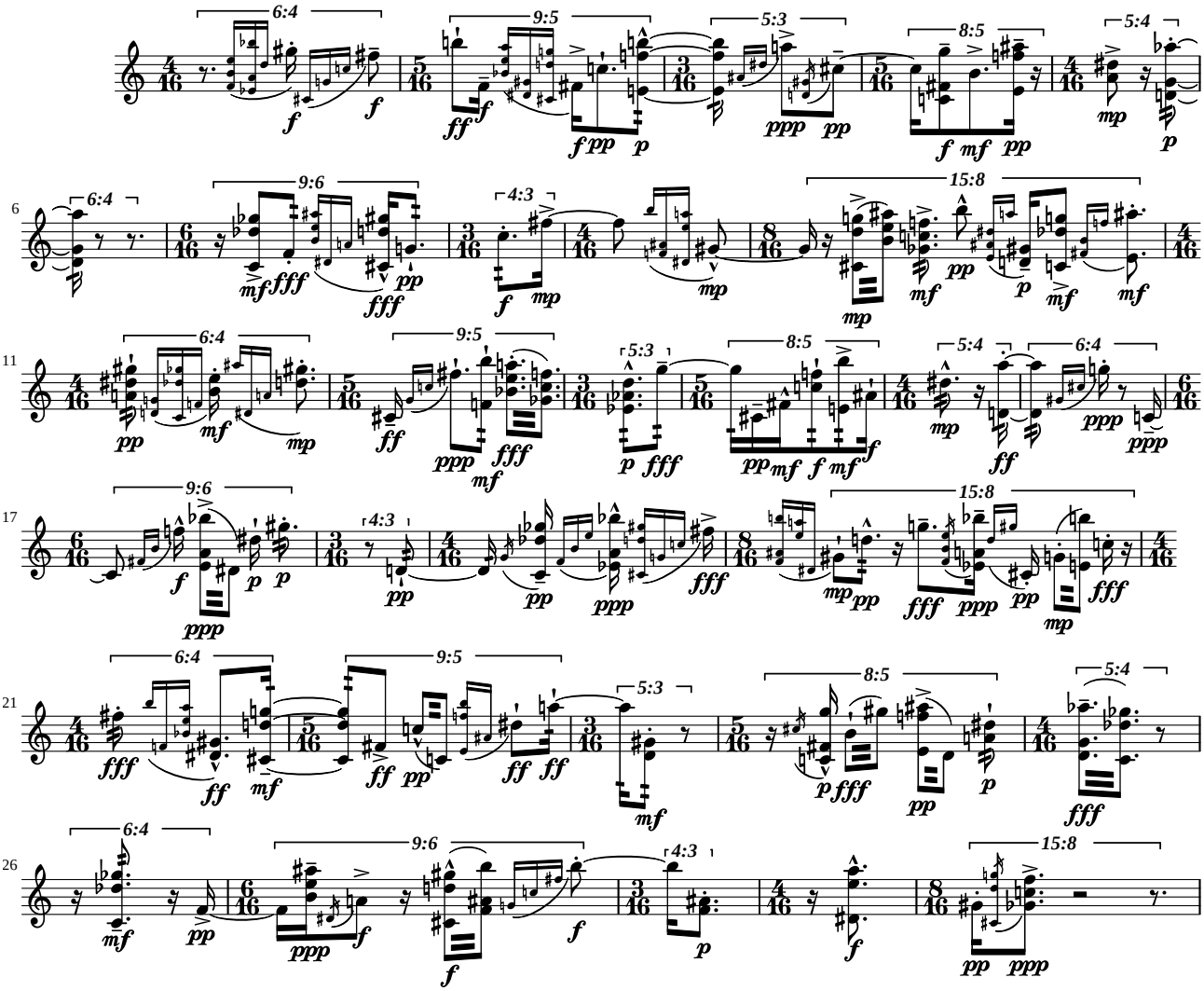
LotusRoot は MIT ライセンスにより公開される。詳細は LICENSE.txt を参照。
日本語訳　https://osdn.jp/projects/opensource/wiki/licenses%2FMIT_license

使用例

コマンドプロンプトから以下を実行する（Windows の場合）。

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

（出力例）



使用方法

以下、サンプルファイルに沿って説明する。全ての機能については README.txt を参照。

00_input.rb

プログラム使用には LotusRoot.rb をロードする。

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

LotusRoot へ渡される基本データは次の 4 種である。

Durations	持続
Elements	音符や休符、持続などを表す記号
Tuplets	連符
Pitches	音高

Elements は文字列の配列で書かれ、その他は整数、小数、有理数など数値の配列で書かれる。
Elements 以外の配列は繰り返し参照される。そのため配列の要素数が 1 でも機能する（同じ内容の繰り返しになる）。
Elements は省略することが出来ない。言い換えれば Elements の数に応じて音符が配置される。

ここでは、Elements が主に次の 3 種からなることを示している。

`"@"`	音符の立ち上がり
`"="`	音符の持続
`"r!"`	休符

例えばこの記譜における最初の音符は、1 つの立ち上がりと 13 の持続によって記述される。
Durations は、具体的には各 Element の長さを意味する。ここでは全ての Elements が長さ 1 なので、この記譜の最初の音符は合計で長さ 14 の持続となる。
Tuplets は整数の場合、4 分音符の分割数を表す。従ってこの記譜の最小音価は 16 分音符であり、この記譜の最初の音符は、16 分音符 14 個分の持続となる。

`"@"`と`"r!"`には LilyPond コマンドを含めることが出来る。例えば`"@\\trill"`と記述することで LilyPond の音符表記の後に「`\trill`」を付加できる。\記号を表現する為、さらにエスケープ記号の\が必要であることに注意。

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

この行は Score オブジェクトの作成とデータ入力を行なう。
基本データの配列名は何でもよいが、仮に `dur, elm, tpl, pch` とする。

```
sco.gen
```

この行は記譜スクリプトの生成を行なう。

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

この行は記譜スクリプトをテキストファイルに出力する。

出力されたファイルは test.ly の次の行で読み込む。

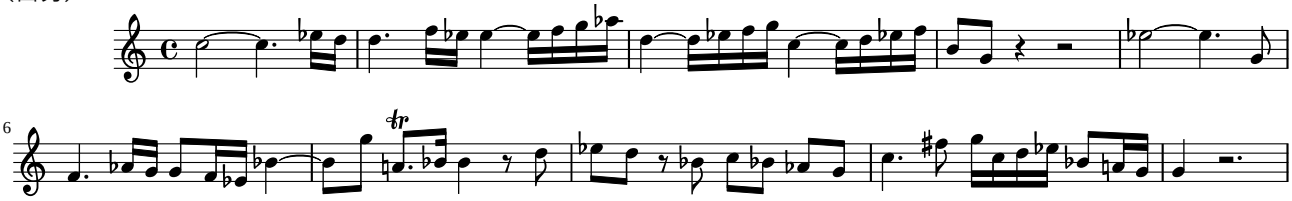
```
\include "sco.txt"
```

これらのファイル名が一致していればファイル名は何でもよい。

```
sco.print
```

この行はコマンドプロンプトに記譜スクリプトを表示する。
その他の行は記譜スクリプトを調整するためのオプションである。詳細は後述。

（出力）



 # 01_input.rb

00_input.rb と同じ記譜内容だが Elements と Durations が異なる。

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

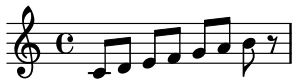
このデータを入力した場合、LotusRoot はこれを内部で以下のように展開する。

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

 # 02_pitch.rb

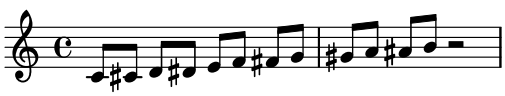
Pitches は整数によって半音階を表し、小数または有理数で 4 分音または 8 分音を表す。
和音の記述には配列を用いる。

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

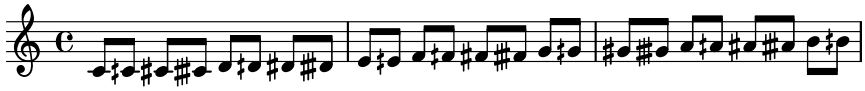


以後、#でコメントアウトされた行を 1 行ずつコメント解除しながら実行してみる。

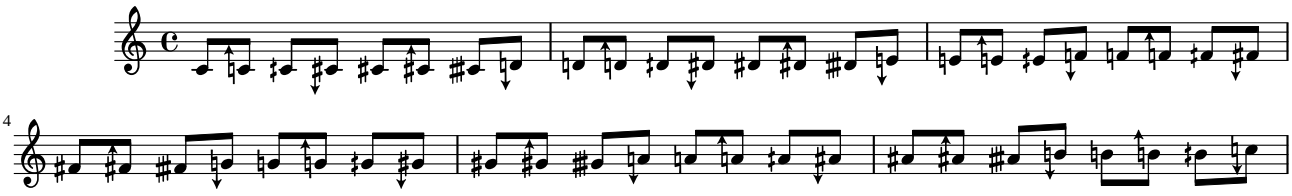
```
pch = [*0..11]
```



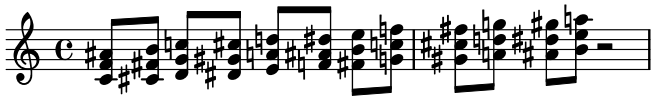
```
pch = [*0..23].map{|e| Rational(e, 2)}
```



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



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



ここで用いられた Ruby 記法について説明する。
[*0..11]は範囲オブジェクトを配列に変換する。
map メソッドは配列の内容を変換する。書式は次の通り。配列 array の各要素 *item* に対して *block* を実行する。

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

以下は pch の実際の内容をコマンドプロンプトに表示する。

```
p pch
```

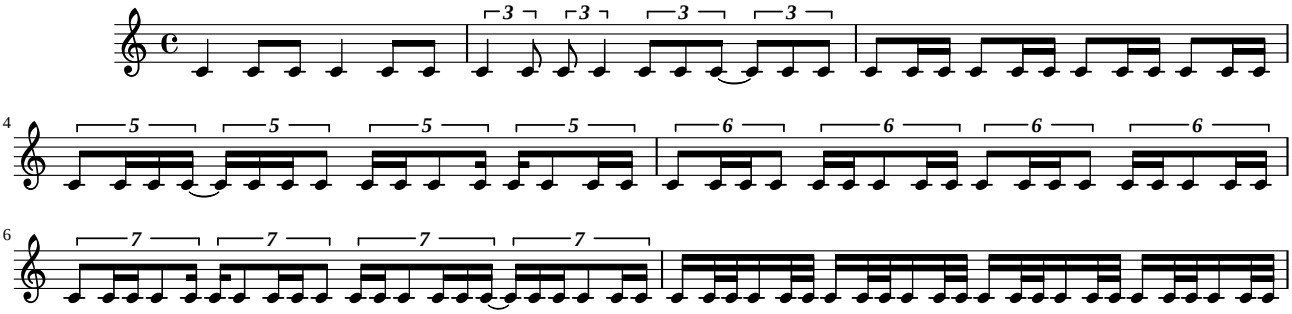
以下は pch に対する map だが、pch の要素数と同じ回数"@"を繰り返すためだけで、pch の内容は無視している。

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

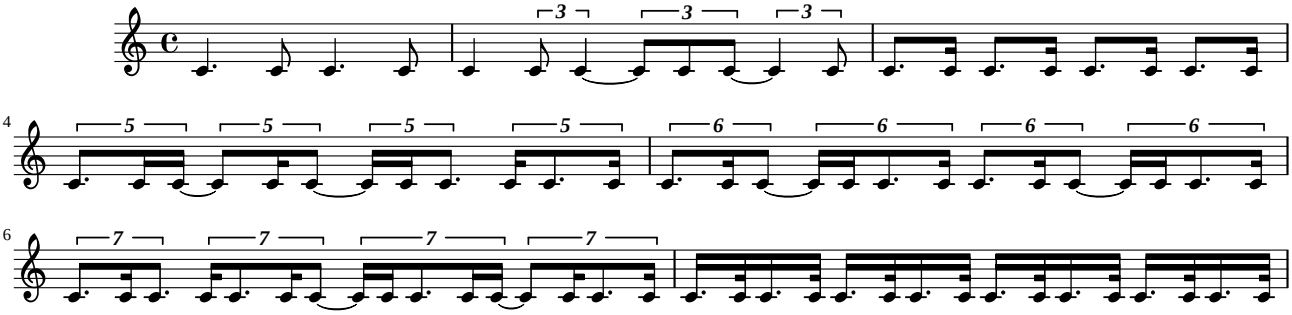
次に LotusRoot のオプションについて説明する。
pitchShift オプションは入力された Pitches 全体に指定の数値を加算する、つまり移高を行なう。
accMode オプションは臨時記号の種類を選択する。0 は（微分音を含め）シャープ系、1 はフラット系で統一される。2 と 3 はそれぞれシャープ系とフラット系から 3/4 音表記を除く。

 # 03_duration.rb

Durations は、具体的には Tuplets で定義される最小音価の何個分であるかによって、持続を定義する。
Durations の値が同じでも連符によって結果が異なる。この場合、Durations は`[2, 1, 1]`を繰り返している。



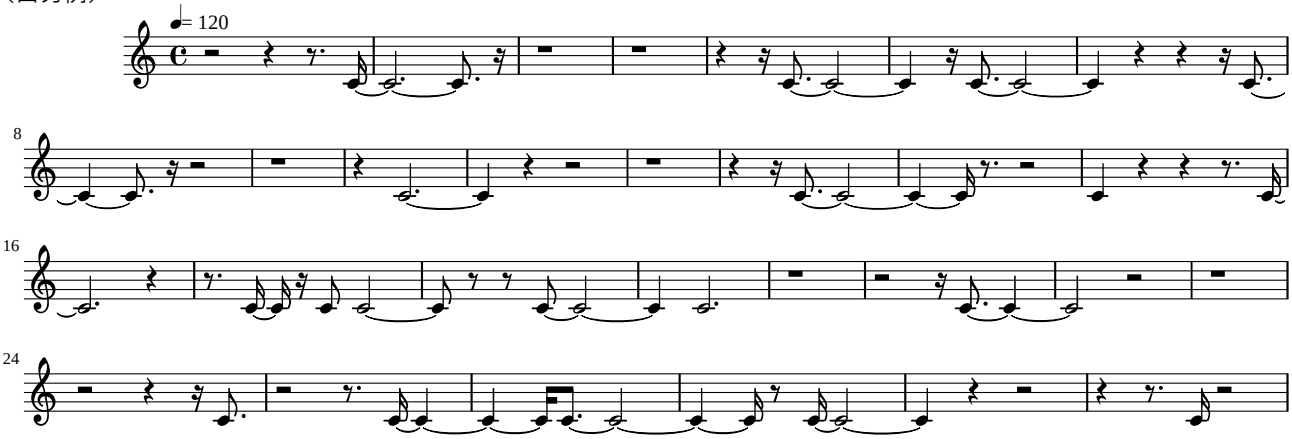
Durations を`[3, 1]`に変更すると、第 2 小節 1 拍目は最小音価 1/3 の 3 個分で 4 分音符となる。



尚、LotusRoot 内部では全ての音価は有理数として処理される。

 # 04_element.rb

（出力例）

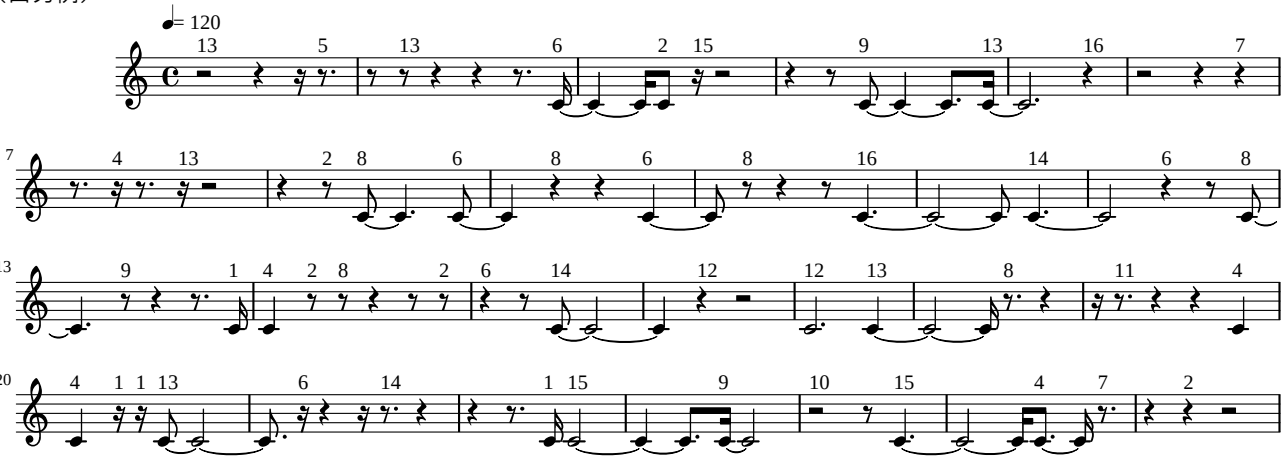


ここではランダムな持続に対して`"r!"`と`"@"`のいずれかをランダムに当てはめている。
Elements には LilyPond コマンドを付記できる。

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

この行は Durations の値を音符の上にマークアップする方法を示している。`#{}`は文字列の中で変数を評価する Ruby 記法である。

（出力例）

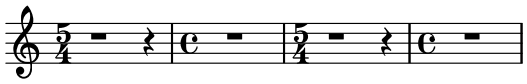


"TMP4;120;"はテンポ記号を挿入するLotusRootのコマンドである。

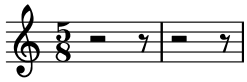
 # 05_metre.rb

拍子はデフォルトで4/4であるが、metreオプションで指定できる。配列で記述し（Tuplets等と同様に）繰り返し参照される。

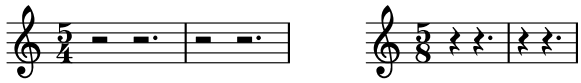
配列の要素を整数nで記述した場合は、n/4拍子として解釈する。



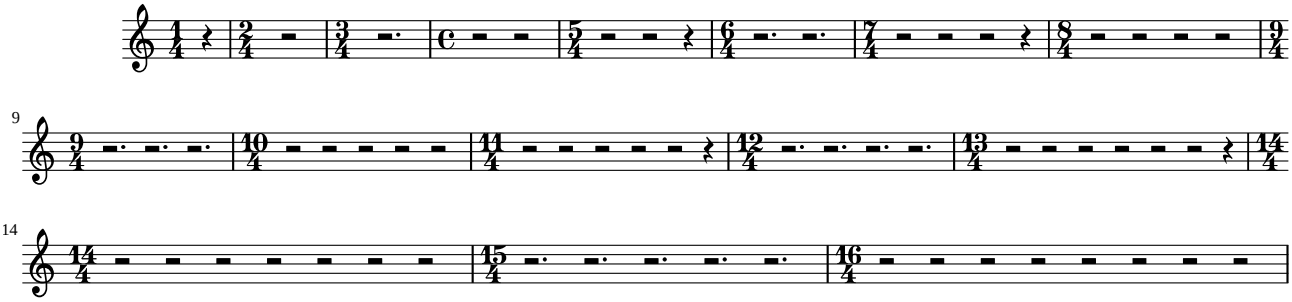
[[beat], unit]の形で記述した場合はbeatが分子、unitが分母となる。



beatは整数の配列であり、1つの整数でもよいが、整数の組み合わせで明示的に拍構造を定義することもできる。



unitは4分音符を1とした整数または有理数である。1/2rはRuby記法で1/2を意味し、この場合は8分音符を意味する。尚、簡易な（明示的でない）記法の場合は自動的に以下のように拍構造を分割する。

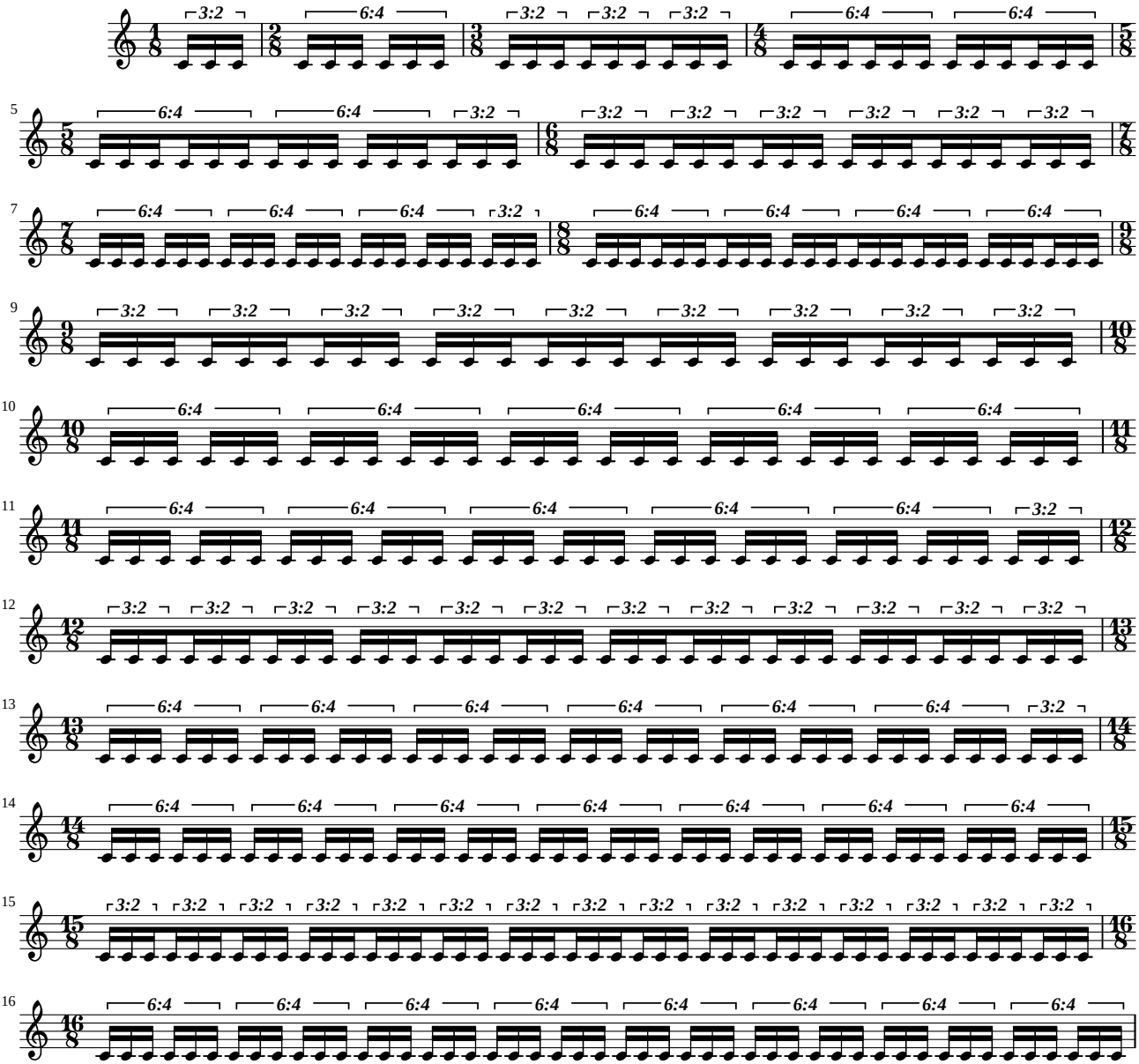


 # 06_tuplet.rb

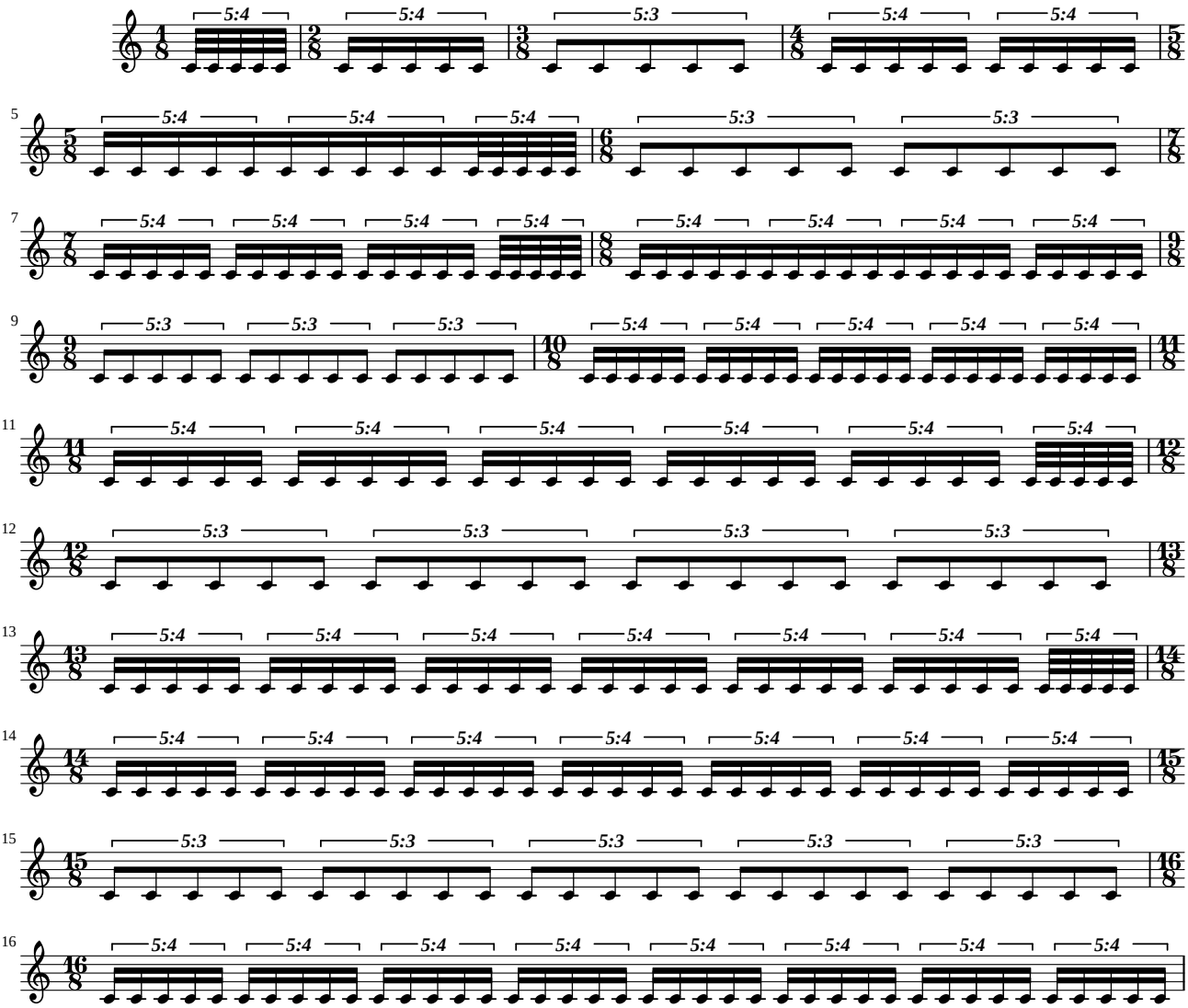
Tuplets を整数の配列で記述すると、拍の分割数で連符を定義する。この場合、`metre` で定義される拍子によって挙動が変わる。

拍は基本的に 4 分音符だが、ある小節が 4 分音符で割り切れない場合、
・連符側が分割できる場合は、分割した連符で小節を充填する(`tpl = [6]`)
・連符側が分割できない場合は、拍構造に従って分割する(`tpl = [5]`)
これは音価を変えず処理に整合性を持たせるためである。`tpl = [4]`とした場合はどの小節も 16 分音符単位で表記される。

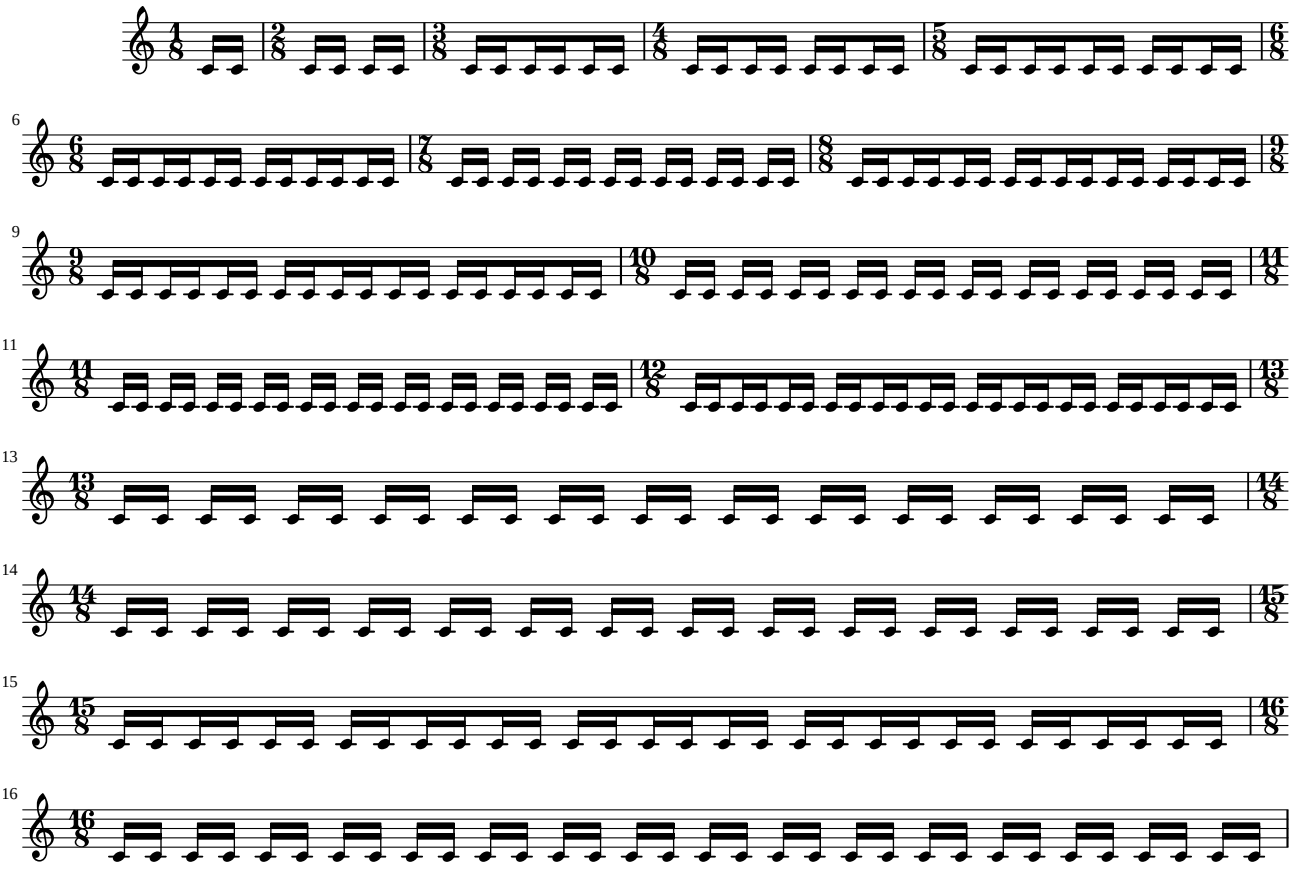
```
tpl = [6]
```



```
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

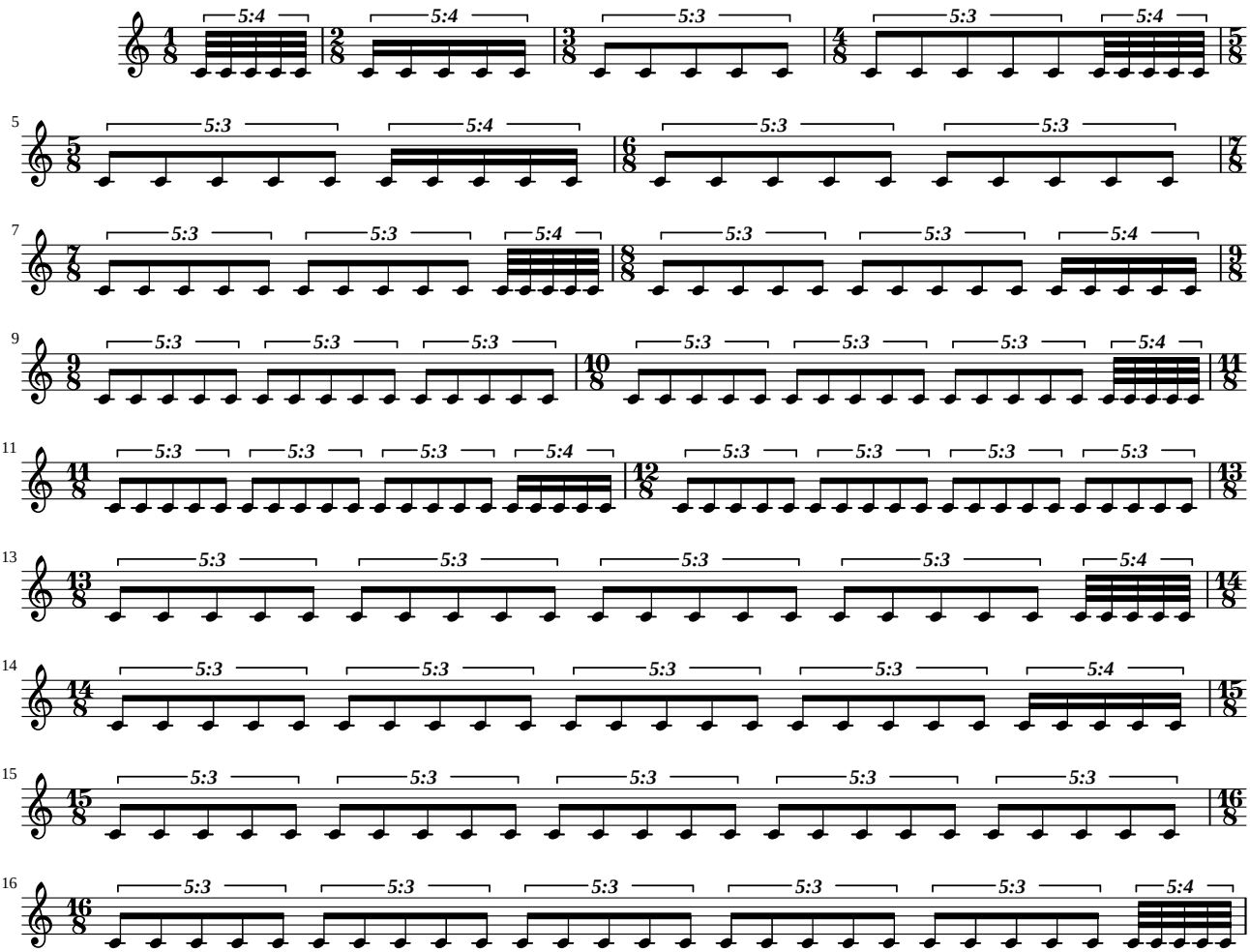
```
tpl = [4]
```



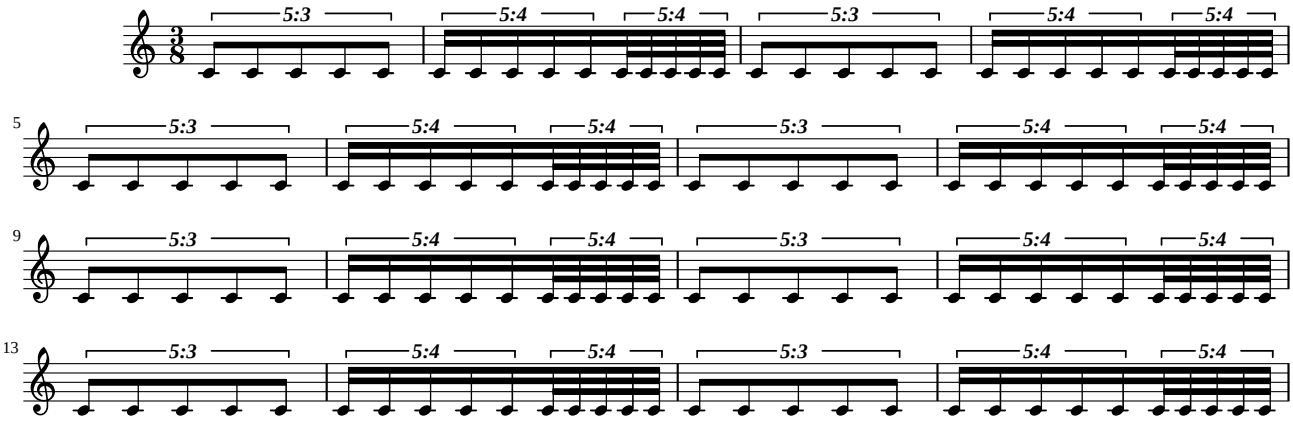
=begin、=end は複数行コメントの Ruby 記法であり、削除または各行を次のようにコメントアウトすることでコメントを無効化できる。

```
# =begin
      （この部分が実行される）
# =end
```

明示的に拍構造が定義された小節に対しては、拍構造に従って拍を分割する。



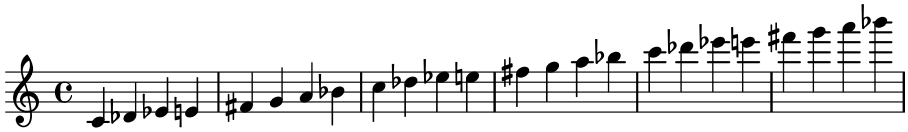
明示的に連符を定義するには[n, d, u]の形で記述する。この場合、n は分子（拍の分割数）、d は分母（拍の長さ＝n の個数）、u は 4 分音符を 1 とした単位音価である。小節を逸脱するような連符を定義するとエラーになる。



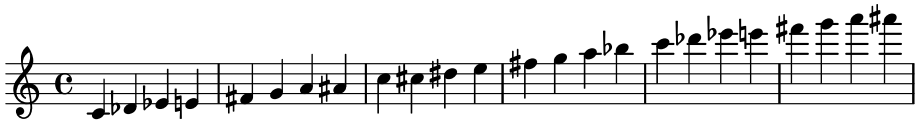
fracTuplet オプションは連符を比率表記にする。
finalBar オプションは指定された小節数で記譜スクリプトを打ち切る。

 # 07_altNoteName.rb

altNoteName オプションは Pitches の値に対して直接音名を指定する。主に臨時記号の指定に用いる。



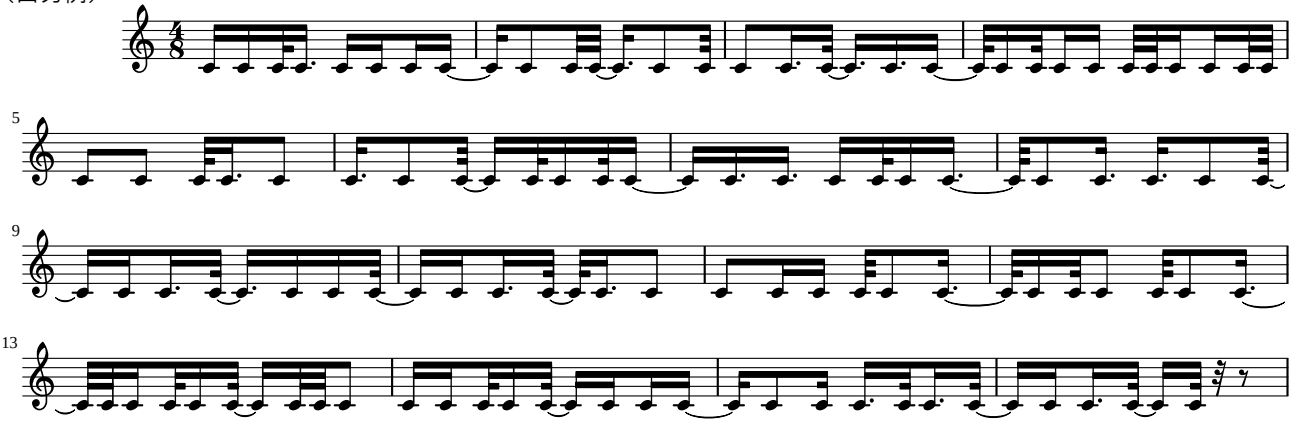
この場合、altNoteName の値は pitchShift オプションによる変更後の Pitches に対する指定となるので注意を要する。
音高の指定が 0~11 の場合は全てのオクターブに適用される。12 以上の値を含むと複数オクターブの繰り返しとなる。



 # 08_tidyTuplet.rb

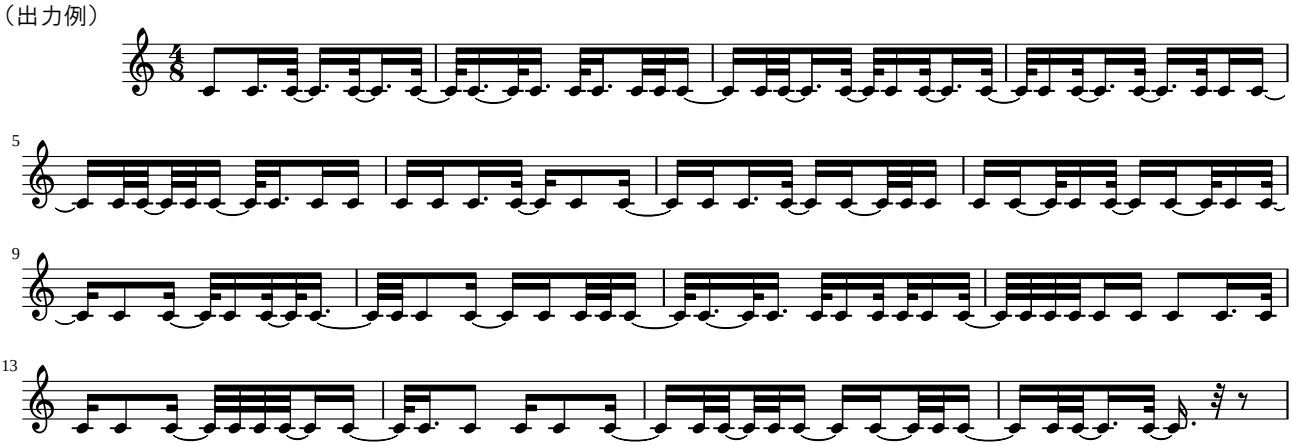
LotusRoot のデフォルトの出力では連符が読みにくい場合がある。

（出力例）



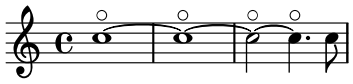
`tidyTuplet` オプションは連符のグルーピングにより可読性の向上を試みる。
（現状では 6 連符と 32 分音符にのみ対応。）

（出力例）

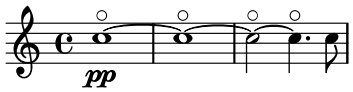


 # 09_tiednotes.rb

タイで連結された全ての音符に対してマークアップを行なうには"@="を用いる。



"#A"と"A#"で囲まれたコマンドは、タイで連結された音符の先頭に適用される。

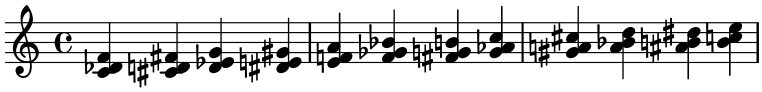


"#Z"と"Z#"で囲まれたコマンドは、タイで連結された音符の末尾に適用される。

尚、"#A..A#"と"#Z..Z#"は"@="以外の Elements では正常に機能しない。

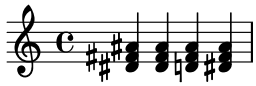
 # 10_autoChordAcc.rb

autoChordAcc = 0とすることで、和音毎に臨時記号の種類を自動的に選択する。
シャープとフラットのいずれかに統一し、混在は行なわない。

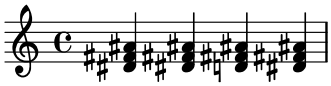


 # 11_reptChordAcc.rb

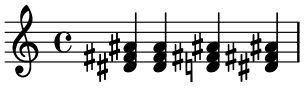
臨時記号の付け方について、LilyPond では\accidentalStyle で設定を行なうが（config.ly を参照）、和音については別途工夫を要する。



reptChordAcc = 0の場合、前後の和音で連続している音高全てに臨時記号を付する。

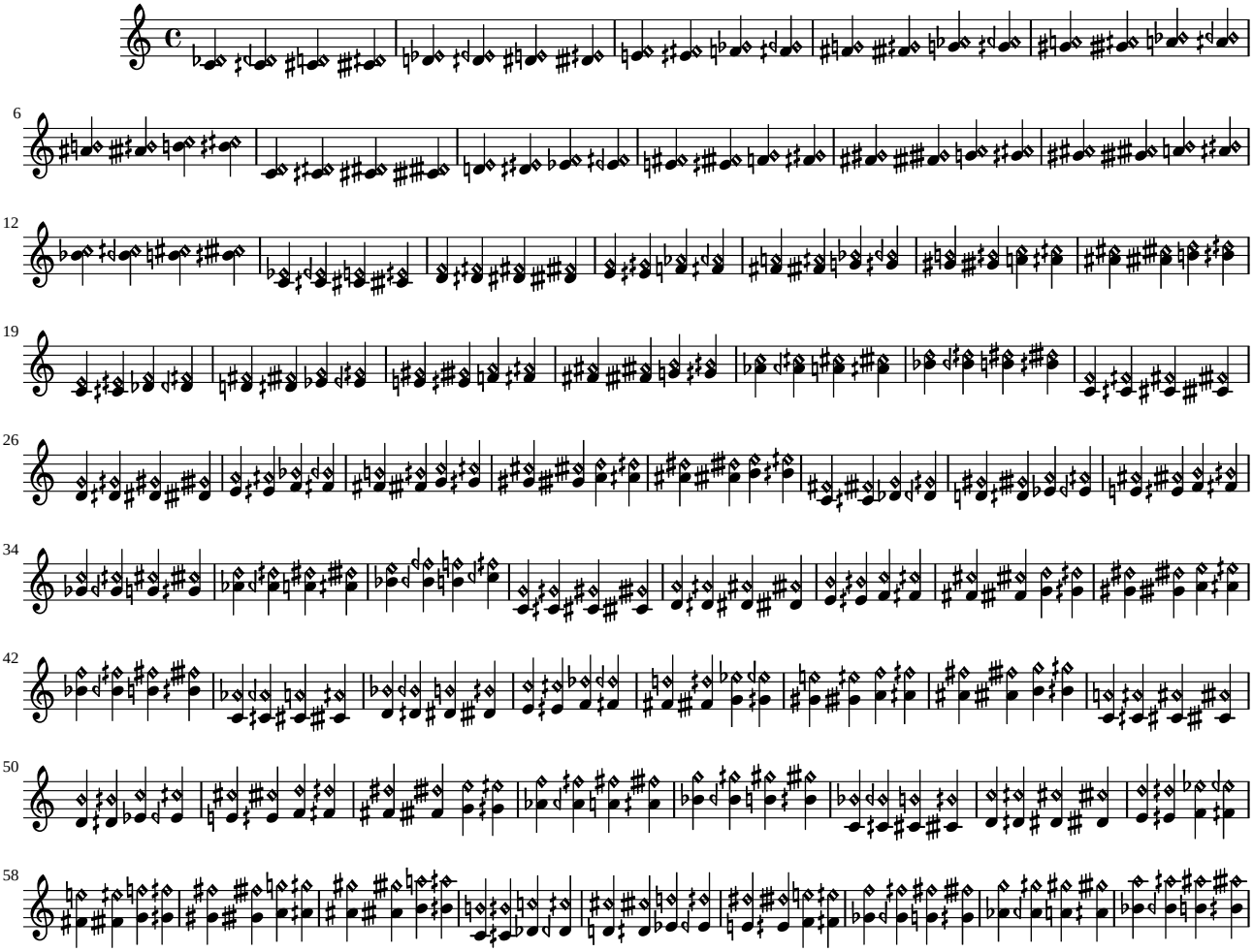


reptChordAcc = 1の場合、同じ和音の連続については上記の処理を省略する。



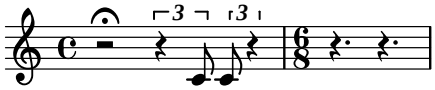
 # 12_artharm.rb

この例では textReplace を用いてテキスト置換を行い、人工ハーモニクスを書く方法を示している。
正規表現を含むテキスト置換の書式は Ruby の gsub メソッドと同じである。
autoChordAcc = 1とすることで、二和音の度数を揃える。

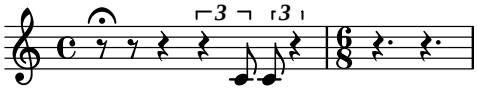


 # 13_rest.rb

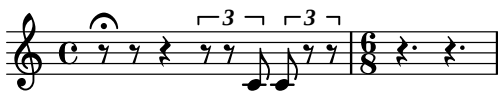
他の文字を含まない"r!"または"s!"は自動的に結合される。
入力データが小節の途中で終わっている場合、最後の小節は自動的に休符で埋められる。



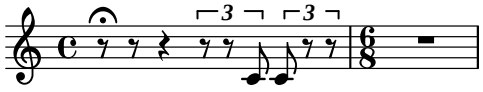
次は休符を分割する方法の一例である。



avoidRest オプションは、指定した音価の休符を分割、除外する。但し Tuplet の最小音価など分割できない場合は無視される。

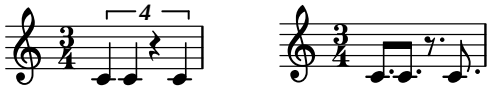


休符は拍構造に従って分割される。wholeBarRest オプションは、小節単位の全休符（LilyPond では大文字の R で記述）を使用する。

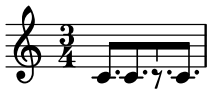


 # 14_misc.rb

dotDuplet オプションは、3 拍 2 連符を付点音符に書き換える。



beamOverRest オプションは、連符毎に休符を跨いで連桁を繋ぐ。



（以降は test.ly ではエラーになる。）
namedMusic オプションは、記譜スクリプトの変数名を指定する。
noMusBracket オプションは、括弧{　}なしの記譜スクリプトのみを出力する。