Sonic Piメモ(作成中) 2024.3.28現在 Hiroshi Tachibana CC BY-SA

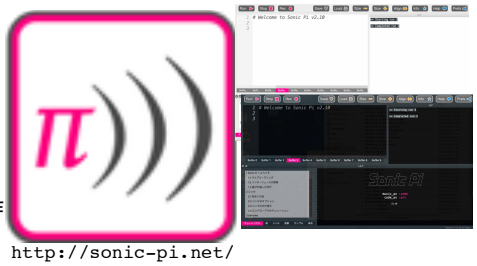
容易なプログラムによってリアルタイムに音を、いろいろな間隔で繰り返し出したり止めたりすることができ、音程や音色を自由に指定でき、多くの効果を追加できる。

Raspberry Pi, Mac, Winで動作. オープンソースでありライセンスは MIT

使用バージョン Sonic-Pi-for-Mac-v4.5.0

Ruby言語(まつもとゆきひろ氏)(独自GPL互換)とSuperCollider(GPL)で作られている。

英University of Cambridge, Computer Laboratory時代のSam Aaron氏作



http://sonic-pi.net/

概要

ステージ上でプログラミングしながら演奏するライブコーディング向け言語。プログラミング入門用にも最適。
多数の音源とエフェクトを持っており、細かなパラメータ調整もできる高機能シンセサイザーでもある。

プログラムを入力してRunボタンにより実行すると音が出ます。 run

プログラムを修正して(いなくても)Runボタンを押すと、そのタイミングで重ねて実行されます。

プログラムは、画面下で選ぶBuffer 0 ～ 9 に入力でき、Bufferを切り替えて重ねてRunすることもできます。 |0| |1| |2| |3|

各バッファにはサイズ制限があります。約200行強。Stopボタンでは、全てが停止します。 stop

ファイルの保存は、Buffer毎に別々に行われ、ファイル名も記憶されておらず上書き保存が容易にできず便利ではありません。 save

ファイルの拡張子は、Rubyと同様の .rb であり中身はテキストファイルです。

全Bufferの現在の状態は、自動保存され、次回起動時に残っています。~/.sonic-pi/store/defaultの中に保存されています。

macでは、ターミナルから、cat ~/.sonic-pi/store/default/w* > SonicPiBuffer.txt 等でまとめて保存できます。

cat ~/.sonic-pi/store/default/*.spi > SonicPi_`date +%Y%m%d`.txt 日付のファイルで保存の場合

Recボタンを押した瞬間から次に押すまでを、WAVファイルに録音することができます。 rec

prefs 画面表示色をダークモード等に変更できる。さらに、［ビジュアル］から半透明化してデスクトップを見せることもできます。

help から、チュートリアル、例、各機能について詳しく知ることができる。 Tutorial Examples Synths Fx Samples Lang

旧版での症状 Boot Errorで起動できなくなった場合は、ターミナルから ps aux|grep Sonic 等で起動していないのに動いている関連Jobを調べ、killすると良い。(macOS) またオーディオ入力に、出力と異なるサンプリングレートのUSBカメラ(マイク付)等が設定されていると、Boot Errorとなることがありシステム環境設定から入力を内臓マイクに変更すれば解決します。(macOS)(SuperColliderによる制限)

チュートリアル
https://sonic-pi.net/tutorial.html
https://gist.github.com/jwinder/e59be201082cca694df9
https://github.com/sonic-pi-net/sonic-pi/tree/dev/etc/doc/tutorial
https://sonic-pi.mehackit.org/index_ja.html
古いマニュアル
https://chuletitaspinguino.files.wordpress.com/2015/12/man_sonic_pi.pdf

プログラム 注意点

文法は、Rubyと同様. 変数の型宣言は必要ない. (Rubyの入門サイトが参考になる) 先頭に $ を付けるとグローバル変数となる.
先頭に @ を付けるとlive_loop間などで変数を共用できる(クラス内で有効なインスタンス変数).
行末に ; 等は必要ない(入れても良い) ;で1行に複数命令の記載ができる。
コロンの位置やカンマ、スペースの有る無し、カッコの種類を間違える等のエラーに要注意.
改行によって次の命令となるので、サンプルプログラムをPDF等からコピペした時に行が繋がっていると、エラーで実行できなくなります。(sleep が行末にくっついてしまう等) 長い行を改行して表示することは可(改行位置は選ぶ)
四則演算は、+-*/ , 累乗・冪乗は** , 平方根はMath.sqrt(2) , 対数はMath.log(a,b) bは底で省略すると自然対数e Math::E

コメント 行中で、# 以下はコメントとなる。

複数行コメントは、 comment do ～ end で可. 文字用ではなく、コードのみをコメント可能. // や /* */ は不可
uncomment do ～ end とunを付け/外しすることで、コメントを解除/設定できる.

または、複数行を選択後、**command + /** で選択行の全ての先頭に ##| をつけることができる. 同操作で解除も可. 要注意:選択時に最終行の次の行の先頭まで選択しないように.

=begin と =end の行で囲んだ部分は注釈行とできる。(Ruby)
stop後は実行されないので、コメント化の代用とできる。

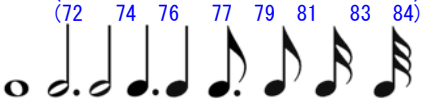
a=1 としてsleep a/2 とすると、a/2は0となってしまう. a=1.0として割り算を行う. 少なくとも分母分子の片方を実数とする必要がある.
i=i+1 と i+=1 は可、i=inc i とも書ける. i++や++iは不可。 1/3.0は、0.3333333333333333
nを12で割った余り(剰余)は、n % 12 割り切れるかどうかは、factor?(10,3) これはfalseを返す、factor?(0,3)はtrue

条件文

```
i=0 if i>10          # 1つの文の場合 ifを後ろに書く
if i>=12 then                # thenは省略可
  elsif i>=24 then           # スペースを入れたelse if は不可. elseの末尾のeは取る.
  else
end
puts 1==2 ? "A" : "B"       # 1が2でないので"B"を返す.       条件 ? がtrueのとき : falseのとき
not equal は、!=    notは! true == !false
```

プリント文 puts (または同 print)によって、変数値や文字等をログに出力できる。
rubyではprintは改行しないが、Sonic Piではputsと同じく改行される。 pは不可.
カンマ , で並べ複数出力可. 変数iの値の出力例 puts "音程は#{i}です."
途中停止 stop (live_loop内のstop文では、そのthreadのみが停止する.)

ド	レ	ミ	ファ	ソ	ラ	シ	ド
:C4	:D4	:E4	:F4	:G4	:A4	:B4	:C5
60	62	64	65	67	69	71	72
(48	50	52	53	55	57	59	60)
(72	74	76	77	79	81	83	84)



4 3 2 1.5 1 0.75 0.5 0.25 0.125 拍

音を出す

音程は、MIDI規格のノートナンバーの数値で中央の"ド"は60または :C4 と表記する。
(C4の前のコロンは必須)(:文字 という書式は、Rubyのシンボル機能で、文字列に類似しており" "で囲う必要がない)
C4の♯,♭は :C♯4(:C♭4),:C♯4 と表記する。大小文字はいずれも可。:Cのみでも使用でき、中央の:C4となる.
play 60 これだけでRunして音を聞くことができる. 大文字で Playは不可.
play :C4 同上(コロンが必須)
休符は、 :R または :r または :rest (sleepを休符として使うことも可)
play 60+12 は、play :C5と同じ。play :C4+12も可.変数に代入して使う場合は:C4.to_iや:C4.to_iや:C4+0.0も有効
範囲は、低音側 0(8.18 Hz)または:C♯0(15.43Hz)～高音側サンプリング周波数44.1kHzの半分となるナイキスト周波数未満の21.096Hzの:E10(136)まで.
複数回のrunを繰り返すと、実行したタイミングで音を重ねることができる。
音の長さは、playの後の sleep 拍 で与える。 一拍は、sleep 1 特別に秒で与えたい場合には sleep rt(1)

速度の指定は、BPM(beat per minute)値で与える.一拍の長さはこの値によって変わる.
5秒を使う音を出すには、
use_bpm 60 (デフォルト) ♩= 60 注:use_bpm=60とすると、use_bpmという変数に値を代入することになりエラーメッセージも出ない
play 60 中央のド
sleep 5 # 5拍
通常の音源によってはdecayが速く、音が消えてしまうので、
play 60, decay:5 または、 play 60,sustain:4,decay:1 など
sleep 5
などとするとわかりやすくなる。逆に速いフレーズでは重なって振り切れるのでdecayやreleaseの値を小さくする等の工夫が必要。
下記で囲うとその部分だけのbpmを変更できる。
with bpm 120 do
end
音符ごとの強さは、amp: (play等の後ろに , で続ける。amp後ろにはコロンが必須で、その後に強さを表す数字を記す。デフォルトは、1)
amp:2, amp:0.5 等、サイン波ではamp:2.5を超えるあたりで音が割れ始める。
アタックAttack、減衰Decay、減衰後の保持Sustain、余韻Releaseは、時間と強さのADSRの各パラメータで設定できる。
多くのsynth音源では、デフォルト値がattack:0, release:1となっているので、この2つを変更すると音色が変わることが多い。音を伸ばすには、sustain:4等とすることが多い。
左右のパンは pan: -1(左端) pan:0(中央)、pan:1(右端)
play 60, amp:2, attack: 0.1, attack_level: 1, decay: 0.2, sustain_level: 0.4,
sustain: 1, release: 0.5, pan: -1 等 (このように途中改行が可能だが , は先頭にできない.)

メインボリューム(音量)を変える set_volume! 0から5 デフォルトは1、現在の音量は、current_volume

変数に日本語も使用できる。 ド=:C4 , 和音1=chord(:C4,:M) 推奨はされていません。

リスト 一次元配列 [:C4, 61, 62] 要素に数字と文字の混在が可.

a= [60, 61, :D4]		変数に代入できる.
play a		全部が同時に鳴る
play a.tick	TICK	先頭から鳴らし、対象要素を一つ進める。先頭の0に戻すには tick_reset
play a.tick(:名前)		名前付きtick。複数のtickがある場合、一度にtickの個数だけ要素が進んでしまうので、それを防ぐために名前を付けて独自のtickとできる. 値の設定は tick_set 5
play a.look	LOOK	tick後に要素を進めずに、再度参照だけをしたい場合には、lookを用いる. tick/4 や loop/4 で要素を4回に1回増やす用途に使用できる。(整数同士の割り算での切り捨て)
play a.tick(step: 2)		2つ進める
tick.reset		リセットして0に戻す
tick_set 10		tickを10にする
要素の指定 [60, 61, 62][1]		は、61 を返す. 要素数を超えるとnilとなる.

puts a[1,2] #(0から数えて)要素1番目から2つ
puts a[1..2] # (0から数えて)要素1番目から2番目まで
puts a[1...2] # (0から数えて)要素1番目から2番目未満まで

a.rotate.first　aの先頭要素を最後に回転させて、先頭を取り出す、a.rotate!ではa自体が変化する.
多次元配列（リスト） a =[0,1,[[2,3],4],5] の場合、a[2][0][1]というように要素を参照すると値は3となる
長さを得るには、[60, 61, 62].length　(3が返る)
配列の結合は、a[0..1]+a[3..4]
　または連結する [1,1].concat([2,2]) は [1,1,2,2] となる。[1,1]+[2,2]と同じ
各要素を繰り返す stretch([1,2],3) は、(ring 1, 1, 1, 2, 2, 2) となる.
[0,1]*2は、[0,1,0,1]となる.
要素毎に連結する puts [1,1,1].zip([2,2]) は、[[1, 2], [1, 2], [1, nil]] 音程と拍数の配列をつなげる等
連続して音符を鳴らす play_pattern_timed [93, 91, 88], 0.1 # 時間を個別指定する [0.5,1,0.5]も可.
（配列のコピーは、b=aで可）　# [0.5,1]の場合、要素が順に繰り返される.
リストの各要素に12を加えて1オクターブ上げるには、[:C3,:E3,:G3].**map**{|i| i+12} は、[:C4,:E4,:G4]
入力を容易にするには、%i[C3 C4 G4 C5 E5 G5 Bf5 C6 D6 E6] は、:と,が付いた配列となる.(例は整数次倍音構成)
　%I(C3 C4 G4) も同様~~だが要素が展開される.~~　rubyの%記法

リング　一次元配列(ring 60, 61, :D4) 文字のringは要素とはならない。リストと比べ先頭と末尾がつながっているのが特徴.

b=(**ring** 60, 61, :D4)	変数に代入できる.
play b	全部が同時に鳴る
play b.tick	先頭から鳴らし、対象要素を一つ進める。最後まで行くと先頭に戻るので、数を気にしなくて良い
play b[1]	2番目の要素を鳴らす。(ring 60, 61, 62)[1]も同様　-1は末尾となる 4は先頭
tick を(単独で)実行すると、全てのringやリストの対象要素が同時に一つ進む。参照だけを行うには **.look** を使う	
(**ring** 60,61,62) は、[60,61,62].**ring** と同じ.	[60,61,62].ring.tick という表記も可
scale、chord で返される値も ring である。	
(ring :a,:b)+(ring :c,:d) は、(ring :a, :b, :c, :d) となる	

(**ramp** 1,2,3)では、tickを続けるとずっと端の3が返される. 要素数を超えた参照は、両端が返される
(ramp 1,2,3)[-100]は1　同[100]は3
その他のメソッド、配列[ ]に対して、ringに対しても可能

a.**shuffle**	シャッフルする. ランダムに並べ替える.
a.pick(n)	ランダムにn個を取り出す(重複あり)
a.**reverse**	逆順のringとする　scale(:C4,:major).reverse
a.mirror	順に進み、逆に戻る.末端は2回鳴る
a.reflect	順に進み、逆に戻る.末端は1回鳴る. ringとした場合、先頭は2回鳴る.
a.first(n)	先頭からn個の要素を取り出す　リストも可　.firstだけでは先頭から1つだけ取り出す
a.last(n)	末尾からn個の要素を取り出す　リストも可　.last 同上
a.take(n)	先頭からn個の要素を取り出す
a.drop(n)	先頭からn個の要素を捨てる　コード F/Aは、chord(:F4,:M,num_octaves:2).drop(1) は、(F,)A,C,F,A,C　F をとる drop(1)
a.drop_last(n)	末尾からn個の要素を捨てる
a.butlast(n) ?	末尾からn個の要素を捨てる　? 動作せず
a.take_last(n)	最後のn要素を取り出す
a.rotate	要素を回転させる [:C,:E,:G]→[:E,:G,:C] = .rotate(1)
a.uniq	重複する要素を省く　リストも可
a.sort	要素をソートする　リストも可
a.push(:C4)	末尾に要素として加える
a.unshift(:B4)	先頭に要素として加える
a.**slice**(2,3)	0から数えて2番目から3つの要素を取り出す
a.**index**(:C4)	:C4が0から数えて何番目の要素かを先頭から検索する
a.rindex(:C4)	:C4が0から数えて何番目の要素かを末尾から検索する
a.scale(n)	aが数値の場合のみn倍する.nは実数が可　ringのみ
a.repeat(n)	n回繰り返す
a.stretch(n)	各要素をn回づつ繰り返す
a.scale(n)	要素が数値の場合、全ての要素をn倍する リングのみ、リストは不可
a.take(n).mirror	結合しての使用も可
ringの連結は、+でも(ring :c,:d).concat((ring :e,:f)) でも可	
a.max	最大値を求める .max(2)では、最大値から2つをリストで返す
a.max{|a,b| a.to_i <=> b.to_i}	数値と音程が混じっている場合に最大値を求めるには
a.min	最小値を求める .min(2)では、最小値から2つをリストで返す
a.tally	aの要素の各個数を統計
a.tally.**collect**{|i| i}	上記をリスト形式で表示
a.to_a	リングをリストにする. chord,scaleは不可　.to_aのaはarray
chord(:C,:M).**map**{|i| i}	リングをリストにする. 出力は、[60,64,67]　scaleも可
文字列のみに対して	
str.lstrip	左側の空白を削除

```
str.rstrip                    右側の空白を削除
str.strip                     両側の空白を削除
str.strip!                    両側の空白を破壊的に削除
str.gsub(" ", "")             全ての空白を削除
```

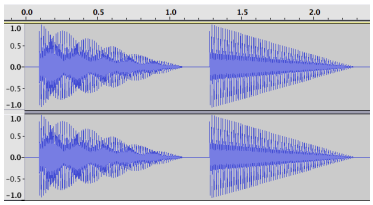
円周率を鳴らす

```
pi="3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 79"
for i in 0...pi.length
  play pi.slice(i, 1).to_i+60,release:0.25
  sleep 0.125
end
```

チューニング、MIDIノート番号と周波数

```
use_tuning :equal             平均律(デフォルト)
use_tuning :just, :C          Cを基音とした純正律
use_tuning :pythagorean,:c 同 ピタゴラス音律
use_tuning :meantone,:c       同 中全音律(ミーントーン)
```

一部分だけ変更するには、with_tuning :just ～ end



平均律(唸りあり) と 純正律 の和音の波形の違い
(rec して Audacity で表示)

```
play chord(:C4,:M)
```

A=442Hzに変更するには、 **set_cent_tuning!** hz_to_midi(**442**) - note(:A4)

```
use_cent_tuning! 1            以降ノート60が60.01の音程となる。 100とすると61の音程となる。
puts midi_to_hz(69)           :Aの周波数を表示 (:A4)も可。 純正律などにチューニングを変更しても反映されない
puts hz_to_midi(440)          =69 周波数をMIDIノート番号に変換
puts pitch_to_ratio(12)       =2 音程差を周波数比に変換
```

音符をMIDI番号に変換　　:C4**.to_i** は、整数60 を返す。　**note(**:C4**)** は、60を返す。
(**midi_notes** :C4,:C5) は、(ring 60, 72) を返す。

音程を音名で表示するには、 puts note_info(60)　は　#<SonicPi::Note :C4> が出力される。

scaleを音名で表示するには、

```
a=scale(:Cs4,:acem_asiran)
a.length.times do
  b=note_info(a.tick).to_s.slice(16..19)
  b=note_info(a.tick).to_s.slice(16..18) if b.slice(b.length-1,b.length)==">"
  puts b
end
```

または、

```
(scale(:Cs4, :acem_asiran)).each do |note|
  note_info_string = note_info(note).to_s
  note_name_match = note_info_string.match(/:¥w+¥d+/)   # ¥はバックスラッシュ
  puts note_name_match[0] if note_name_match
end
# マッチした前と後ろを得る pre_match と post_match もあり.
```

転調　音程を上下する　移調

```
use_transpose 1               半音上げる　0.1等も可
use_octave 1                  1オクターブ上げる
```

一部分だけ転調するには、with_transpose 1 do ～ end

現在の速度は、current_bpm　これを相対的に変えるには

```
use_bpm current_bpm * 1.2     現在の速度の1.2倍の速度にする。
use_bpm_mul 1.2               同上
```

乱数　音程、強さ、他のパラメータなどをランダムにできる。rrand_i , choose , shuffle , one_in , dice

```
use_random_seed 0    乱数の系列:デフォルトでは0に設定されている
use_random_seed Time.new.usec      乱数の系列をCPUの時計のミリ秒を元に与えて毎回変える.
```

乱数の系列を「年月日時分秒ミリ秒」まで与えてできる限り最もランダムにするには、

```
(Time.new.year.to_s+Time.new.month.to_s+Time.new.day.to_s+Time.new.hour.to_s+Time.new.min.to_s+Time.new.sec.to_s+Time.new.usec.to_s).to_i
rand(10)            0以上10未満の実数乱数単独の rand は rand(1)と同じ.
rand_i(10)          0以上10未満の整数乱数単独のrand_iは、0 or 1
rrand(60, 72)       60以上72未満の範囲を指定した実数乱数　ranged rand の略
rrand_i(60, 72)     60以上72以下の範囲を指定した整数乱数
rand_back           乱数の数値を現在の系列の中で一つ前に戻す
if one_in(2)        1/2の確率で実行する.
play 57 if one_in(3)    1/3の確率で鳴らす(if文を後ろに書き、1行で書くことができる) if i==1
  rdist(1,0)  0を中心として±1の乱数を発生させる。(1,-1は含まない)
```

rdistのHELPには、-1 or 1をランダムに出力する方法として rdist(1,0) と書いてありますが、このようには動作しませんでした。
rrand_i(0,1)*2-1 とすることで -1 or 1 をランダムに出力できます。

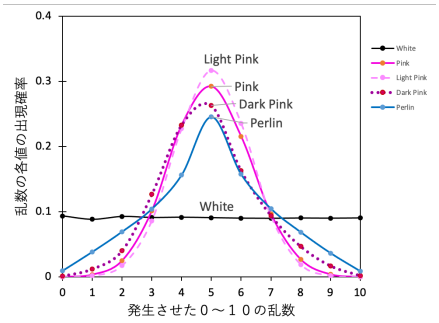


図. 乱数rrand_i(0,10)を100万回発生させたときの分布

他のランダム関連

dice	dice 6 は、1～6 を返す
on_in(n)	1/n の確率 true/false を返す
choose	リストからランダムに選ぶ
shuffle	リストをランダムに並び替える
pick(n)	リストからランダムに n 個の要素取り出す．重複あり

乱数の種類を変えるには、

```
use_random_source :white        デフォルトの完全にランダムなホワイトノイズ
use_random_source :pink         ピンクノイズ、強い雨や滝の音、どのオクターブ域でも大きさが同じ、1/fゆらぎ
use_random_source :dark_pink
use_random_source :light_pink
use_random_source :perlin  パーリンノイズ、CGで炎や雲、稜線などに応用される． アカデミー科学技術賞。分布はpinlに似ているが、並び方は異なる．
```

リストやリングからランダムに一つを選ぶ

```
choose([60, 64, 65, 67, 72])  または [60, 64, 65, 67, 72].choose
choose(0..2)    0,1,2のどれか
```

文字列をや要素をランダムに並べ替える

```
"CDEFG".shuffle または shuffle("CDEFG")
 [:C,:D,:E,:F,:G].shuffle または shuffle([:C,:D,:E,:F,:G])
```

ループの方法

無限ループ

```
loop do
  音
end
```

ループの基本構造は、
何かを do
end

```
                                        # loop{play 60;sleep 0.5} などの書き方も可
                                                # rubyの while文も動作可
```

回数やステップを指定したループ

```
12.times do                 12.times do |i| とすると、ループ内でiを0～11の値として使用できる。
end
(1..12).each |i| も同様
```

または、（推奨はされていません）

```
for i in 0..10              # iを0～10まで．  doを入れるとインデントがずれる．  注：命令一覧にfor文の記載はなし
 音                  .. は <=  # 配列a[i]を使いたい場合に、for i in 0..a.length-1 と書ける(a.count-1でも可)
end                  ...は <   #      同 "..."を使うと、for i in 0...a.length と書ける(a.lengthは含まれない)
ステップを与えるには、  (60..72).step(2).each
```

または i を0~9 に変えるには、

```
10.times |i|
end
```

または、(推奨されています)

```
(0..9).each do |i|
end
```

または、変数 i を使わず _1 を使うこともでき、（推奨はされていません）

```
10.times do
  puts _1
end
```

円周率の文字列から1文字づつ取り出して音にする .each_char

```
pi="3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 79"
# 小数点以下100桁のπ
pi.each_char do |c|
  play scale(:C4,:major,num_octaves:4)[c.to_i],release:0.25
  #play chord(:A3,:M7,num_octaves:4)[c.to_i],release:0.25
  sleep 0.125
end
```

要素番号を使いたい場合には、

```
notes = [:C,:E,:G]
notes.each_with_index do |n, i|
  puts "#{i+1}番目の音は#{n}です"
  play n
  sleep 1
end
```

または、

```
60.upto(72) do |i|          # 下げる場合は、 72.downto(60) do |i|                  # ()がなくても動作可
```

```
 play i; sleep 0.5;  end     # ステップを与えるには、60.step(72,0.5) do |i|
```
ステップを与えて減らすときは　0.5.step(0.02,-0.02) do |i|

```
for i in Range.new(1,10)  というRubyの書き方も可　（推奨はされていません）
```
これも可
```
(0..12).each {     # { } の代わりに do end でも可　（推奨はされていません）
  play _1+60       # _1 に値が入る
  sleep 0.25
}
```
スレッド　音を重ねるには、threadを使いプログラムの同時（並列）実行を行う。 複数のスレッドはRun時に同時スタートする。スレッドは、1回並行実行されて終了する。ライブループは並行実行が回り続ける。
```
in_thread do
 4.times do    # スレッドの内側にループを使うことが多い．指定回数だけ他の音と並列に鳴らして止めたい場合に有効．
 end           # 内側を無限ループとした場合は、live_loopと類似した動作となる．
end
```
再度Runすると、Runしたタイミングで、元の音のタイミングとは関係なく、重ねて実行される。

名前付きスレッドを使うと、再度実行された場合に同じ名前のスレッドは（書き換えたとしても）実行されない。
```
in_thread(name: :名前) do   (の前にスペースがあるとエラー．　2つのコロンにも注意．　間違って音を重ねてしまうことがなくなる
end
```
ライブループ音を重ね、後から置き換えるようなことも行いたい場合には、スレッドを使わずに名前付きlive_loopを使うといい。
```
live_loop :名前 do          Runするごとに同じ名前のlive_loopの音を置き換えることができる。
end                         ループ内のplayをコメント化して再度Runすればそのループの音を消すこともできる．
                            ループ内にsleepがなく、sample等だけの場合、1回で停止する．
                            ライブループ名がCUEとなる
```
cueとsyncでタイミングを合わせる　　cueなしでsyncのみでも使用できる．

スレッド同士での特別な同期を行うには、ループの中にcueとsyncを置く。
```
live_loop :loop1 do
  play :C6, attack: 0.05, release: 0
  sleep 0.5
  cue :cue1        # タイミングを送る側 （名前が:cue1の名前付きcue）
  sleep 0.5        # 時間間隔は、ここで与える
end
loop do            # この外側にthreadのループを置いても問題ない。（変わらない）
  sync :cue1       # タイミングを受ける側． 名前付きsync
  #(または cueなしで、ライブループ名に対してsyncさせるなら sync :loop1 という使い方もある．シンクする位置は変わります）
  sample :drum_heavy_kick
  # サンプルの長さが0.27であり、0.5より短いので、syncでタイミングが合うため sleep はなくても構わない．
end
```
その他の例
```
live_loop :clock do
  sleep 1
end
```
としてcueを出すループだけを用いることもできる。
```
in_thread(name: :th1) do              # 左右にランダムにpan
  loop do
    cue :cue_1
    sleep 0.5
    cue :cue_2
    sleep 0.5
  end
end
live_loop :live1, sync: :cue_1 do                 # auto_sync:1 とすると、最初のループから同期する．
  # sync :cue_1                                   # sync を独立させる使い方は、ver.3から不可となった．
  play :C4, pan: rrand_i(0,1)*2-1,pan_slide: 0.5
end
live_loop :live2, sync: :cue_2 do
  play :C3, pan: rrand_i(0,1)*2-1,pan_slide: 0.5
end
```
指定した時間後に実行を開始するループ at
```
 at 2 do                              # 開始から2拍待った後に3拍目に実行を開始する
 end
 at [1,3,1.1] do           # 全て開始から、1拍待った後、1.1拍後、3拍後、のループを計3回実行する．
 end                       #                                [ ]内は、逆順になっていても可、

 at [1, 2, 3],[{:amp=>1}, {:amp=> 2,:attack=>0.2},{:amp=>0.3}] do |p|
```

```
    sample :drum_cymbal_open, p      # 1拍目は普通、2拍目は立ち上がりをゆっくりで強く、3拍目は弱く
  end
```

関数(メソッド)の定義

```
  define :my_func do
  end
```

関数の使用例

```
  my_func
```

関数に渡す値である引数(ひきすう)は、| | で囲う.

```
  define :my_signal do |n,t,d|      # キーワード引数つきも可   |n:, t:, d:|
    play n, decay: d, release:0
    sleep t
  end                          # 戻り値は、return 式 で与える.
```

上記の引数のある関数の使用例

```
  my_signal :C5, 0.5, 0.1            # 括弧を使った  my_signal( :C5, 0.5, 0.1 )  も可
```

音源の指定

use_synth :beep (正弦波、デフォルト音,:sineと同じ) `puts synth_names , puts synth_names.size`

シンセ音源一覧(MIDIと同じように使える音源. 42種→48種(4.4)→66種(4.5). 内部にあるsynthdefファイル数はもっと多い)

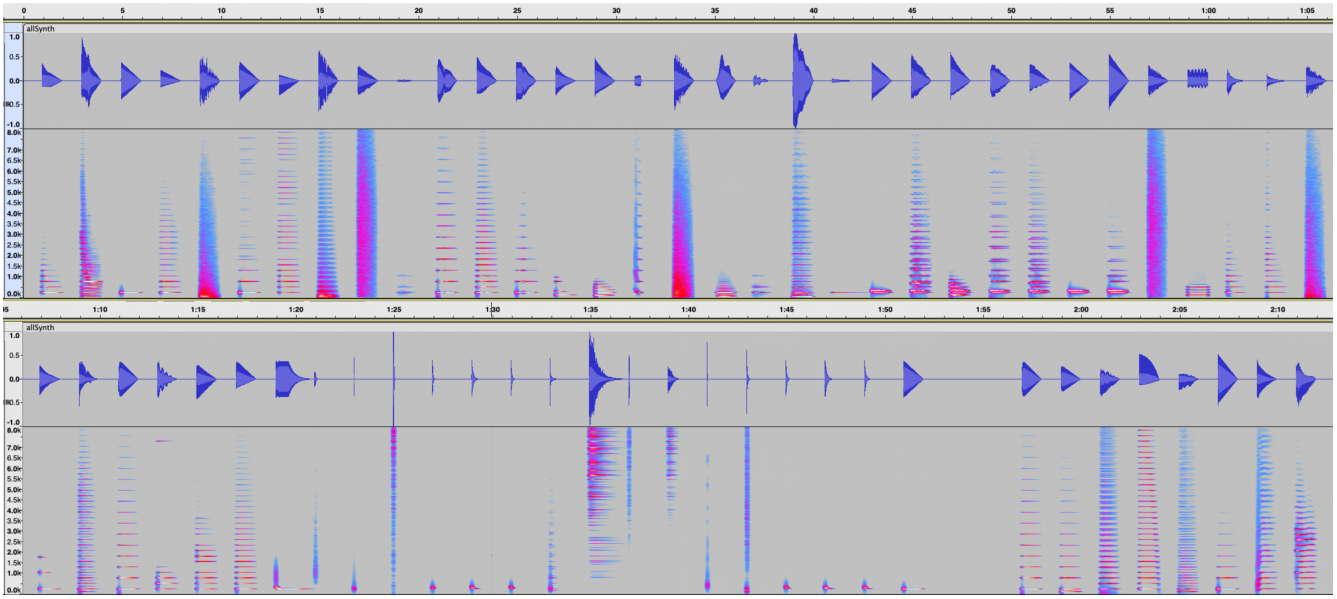
:bass_foundation	:dsaw	:mod_dsaw	:pretty_bell	:sc808_congamid	:sound_in
:bass_highend	:dtri	:mod_fm	:prophet	:sc808_cowbell	:sound_in_stereo
:beep	:dull_bell	:mod_pulse	:pulse	:sc808_cymbal	:square
:blade	:fm	:mod_saw	:rhodey	:sc808_maracas	:subpulse
:bnoise	:gabberkick	:mod_sine	:rodeo	:sc808_open_hihat	:supersaw
:chipbass	:gnoise	:mod_tri	:saw	:sc808_rimshot	:tb303
:chiplead	:growl	:noise	:sc808_bassdrum	:sc808_snare	:tech_saws
:chipnoise	:hollow	:organ_tonewheel	:sc808_clap	:sc808_tomhi	:tri
:cnoise	:hoover	:piano	:sc808_claves	:sc808_tomlo	:winwood_lead
:dark_ambience	:kalimba	:pluck	:sc808_closed_hihat	:sc808_tommid	:zawa
:dpulse	:mod_beep	:pnoise	:sc808_congahi	:sine	
			:sc808_congalo		

全てのシンセの名称をsynth_namesから得て1音だけ聞いてみるプログラム例.　　または、

```
  synth_names.length.times do
    print current_synth
    use_synth synth_names.tick
    play :C4
    sleep 0.5
  end
  print synth_names.length
```

```
synth_names.each do |s|
  puts s
  use_synth s
  play :C4
  sleep 0.5
end
```

```
# 一音だけ臨時に使いたい場合は、 synth :fm, note: 60 で 特定のSynth音を鳴らすことができる.
```



コード音(和音)を出す　　`puts chord_names`

コード名の種類は70、数字で始まる+5や7を直接使うときは'+5'や'7'や'7sus4'とする.文字で始まるM以降は、:M または 'M' とする.

:Mは:major, :mは:minor と同じ

'+5'	'7-10'	:M7	:diminished7	:m13	'm9+5'	:min
'1'	'7-11'	:a	:dom7	:m6	:madd11	:minor
'11'	'7-13'	:add11	:halfdim	'm6*9'	:madd13	:minor7
'11+'	'7-5'	:add13	:halfdiminis	:m7	:madd2	:sus2
'13'	'7-9'	:add2	:hed	'm7+5'	:madd4	:sus4
'5'	'7sus2'	:add4	:i	'm7+5-9'	:madd9	
'6'	'7sus4'	:add9	:i7	'm7+9'	:maj	
'6*9'	'9'	:augmented	:m	'm7-5'	:maj11	
'7'	'9+5'	:dim	'm+5'	'm7-9'	:maj9	
'7+5'	'9sus4'	:dim7	:m11	:m7b5	:major	
'7+5-9'	:M	:diminished	'm11+'	:m9	:major7	

```
play chord(:C4, :M)          コード名を指定してを鳴らす。メジャーコード(ドミソ) play (chord :C4, :M)も可
play chord(:A3, :m,)    マイナーコード Am の例 （ラドミ）   数字コードは'+5' と ' で囲う必要あり
  play chord(:C3, :M, num_octaves: 2)    2オクターブ分のコードを鳴らす(ドミソドミソ) :2を=2と間違えてもエラーは出ない
  play (chord_invert(chord :C4,:M),2)    2回,転回したコードを鳴らす。外側の()は必須.（ドミソをソドミに）-方向の転回も可
          .rotate(n)は、要素を回転させるだけ
  play [:Df3]+chord(:G3,:augmented)      分数コードの基音を加える. Gaug/D♭ ブラックアダーと呼ばれるコード
  play_chord [60,64,67]          音を併記して同時に鳴らす.  [の前にスペースが必須.   play [60,64,67] でも同じ.
```

アルペジオでコードを弾く Em7

```
play_pattern chord(:E3, :m7)          1拍間隔で鳴らす
```

アルペジオでコードを順に弾く時間間隔を与えるには、

```
play_pattern_timed chord(:E3, :m7), 0.25      # 0.25拍間隔で鳴らす
```

play_pattern_timed は、play よりも一つの音の減衰が遅くなります。
play_pattern_timed [:C],[1],sustain:0 と play :C が同等です。
play_pattern_timed [:C],[1] と play :C,sustain:1 が同等です。

コードを転回して鳴らす chord_invert （逆側は、- で可能）

```
i=-1
3.times do
  play_pattern_timed(chord_invert( chord :C4,:M ),i),0.20        # G3 C4 E4
  sleep 0.5                                                      # C4 E4 G4
  i+=1                                                           # E4 G4 C5
end
```

全コードの名称をchord_namesから得て、全コードを聞いてみるプログラム例.

```
chord_names.length.times do
  play_pattern_timed chord(:C4, chord_names.tick),0.1
  sleep 0.5
end
print chord_names.length
```

スケール(音階)の種類 v3 73種→v4 151種 puts scale_names , puts scale_names.length

音程に小数点以下があり、平均律でないものも含まれています。

:aeolian	:dorian	:hirajoshi 平調子	:major_pentatonic	:minor	:shang
:ahirbhairav	:egyptian	:hungarian_minor	:marva	:minor_pentatonic	:spanish
:augmented	:enigmatic	:indian	:melodic_major	:mixolydian	:super_locrian
:augmented2	:gong	:ionian	:melodic_minor	:neapolitan_major	:todi
:bartok	:harmonic_major	:iwato 岩戸調子	:melodic_minor_asc	:neapolitan_minor	:whole
:bhairav	:harmonic_minor	:jiao	:melodic_minor_desc	:octatonic	:whole_tone
:blues_major	:hex_aeolian	:kumoi 雲井調子	:messiaen1	:pelog	:yu
:blues_minor	:hex_dorian	:leading_whole	:messiaen2	:phrygian	:zhi
:chinese	:hex_major6	:locrian	:messiaen3	:prometheus	
:chromatic	:hex_major7	:locrian_major	:messiaen4	:purvi	
:diatonic	:hex_phrygian	:lydian	:messiaen5	:ritusen 律旋	
:diminished	:hex_sus	:lydian_minor	:messiaen6	:romanian_minor	
:diminished2	:hindu	:major	:messiaen7	:scriabin	

v4.5の全151種

:acem_asiran	:buselik	:evcara_3	:hex_phrygian	:indian	:locrian
:acem_kurdi	:buselik_2	:evcara_4	:hex_sus	:ionian	:locrian_major
:acemli_rast	:cargah	:evic	:hicaz	:isfahan	:lydian
:aeolian	:chinese	:evic_2	:hicaz_2	:isfahan_2	:lydian_minor
:ahirbhairav	:chromatic	:ferahfeza	:hicaz_humayun	:iwato	:mahur
:augmented	:diatonic	:ferahfeza_2	:hicaz_humayun_2	:jiao	:major
:augmented2	:diminished	:ferahnak	:hicazkar	:karcigar	:major_pentatonic
:bartok	:diminished2	:gong	:hicazkar_2	:kumoi	:marva
:bayati	:dorian	:gulizar	:hindu	:kurdi	:melodic_major
:bayati_2	:dugah	:harmonic_major	:hirajoshi	:kurdili_hicazkar	:melodic_minor
:bayati_araban	:dugah_2	:harmonic_minor	:hungarian_minor	:kurdili_hicazkar_2	:melodic_minor_asc
:bestenigar	:egyptian	:hex_aeolian	:huseyni	:kurdili_hicazkar_3	:melodic_minor_desc
:bhairav	:enigmatic	:hex_dorian	:huseyni_2	:kurdili_hicazkar_4	:messiaen1
:blues_major	:evcara	:hex_major6	:huzzam	:kurdili_hicazkar_5	:messiaen2
:blues_minor	:evcara_2	:hex_major7	:huzzam_2	:leading_whole	:messiaen3

```
:messiaen4          :neva_2             :saba               :sevkefza_2         :tahir              :zhi
:messiaen5          :nihavend           :scriabin           :sevkefza_3         :tahir_2            :zirguleli_hicaz
:messiaen6          :nihavend_2         :sedaraban          :shang              :todi               :zirguleli_hicaz_2
:messiaen7          :octatonic          :sedaraban_2        :spanish            :ussak              :zirguleli_suznak
:minor              :pelog              :segah              :sultani_yegah      :uzzal              :zirguleli_suznak_2
:minor_pentatonic   :phrygian           :segah_2            :sultani_yegah_2    :uzzal_2            :zirguleli_suznak_3
:mixolydian         :prometheus         :sehnaz             :super_locrian      :whole
:muhayyer           :purvi              :sehnaz_2           :suzidil            :whole_tone
:neapolitan_major   :rast               :sehnaz_3           :suzidil_2          :yegah
:neapolitan_minor   :ritusen            :sehnaz_4           :suznak             :yegah_2
:neva               :romanian_minor     :sevkefza           :suznak_2           :yu
```

SonicPi v3からv4で新たに追加されたScaleはこの78種

```
:acem_asiran        :evcara_2           :hicazkar           :kurdili_hicazkar_4 :segah_2            :suznak_2
:acem_kurdi         :evcara_3           :hicazkar_2         :kurdili_hicazkar_5 :sehnaz             :tahir
:acemli_rast        :evcara_4           :huseyni            :mahur              :sehnaz_2           :tahir_2
:bayati             :evic               :huseyni_2          :muhayyer           :sehnaz_3           :ussak
:bayati_2           :evic_2             :huzzam             :neva               :sehnaz_4           :uzzal
:bayati_araban      :ferahfeza          :huzzam_2           :neva_2             :sevkefza           :uzzal_2
:bestenigar         :ferahfeza_2        :isfahan            :nihavend           :sevkefza_2         :yegah
:buselik            :ferahnak           :isfahan_2          :nihavend_2         :sevkefza_3         :yegah_2
:buselik_2          :gulizar            :karcigar           :rast               :sultani_yegah      :zirguleli_hicaz
:cargah             :hicaz              :kurdi              :saba               :sultani_yegah_2    :zirguleli_hicaz_2
:dugah              :hicaz_2            :kurdili_hicazkar   :sedaraban          :suzidil            :zirguleli_suznak
:dugah_2            :hicaz_humayun      :kurdili_hicazkar_2 :sedaraban_2        :suzidil_2          :zirguleli_suznak_2
:evcara             :hicaz_humayun_2    :kurdili_hicazkar_3 :segah              :suznak             :zirguleli_suznak_3
```

```
play_pattern_timed(scale :C4, :hirajoshi),0.2     # 平調子を聞く.
play_pattern_timed scale(:C4, :hirajoshi),0.2     # 同上

loop do  # ドレミファソラシドの2オクターブを繰り返す。
  play (scale :C, :major , num_octaves: 2).tick    # (ring スケールの音符)と同じ機能となる
  sleep 0.5                    #  .reverse.tick や .shuffle.tick も可
end                            #  .take(3)  先頭から3音取り出す
                               # (scale:C4,:major)[1..2] は (ring 62,64) 2つ目～3つ目の要素. [1..-6]も同じ
```

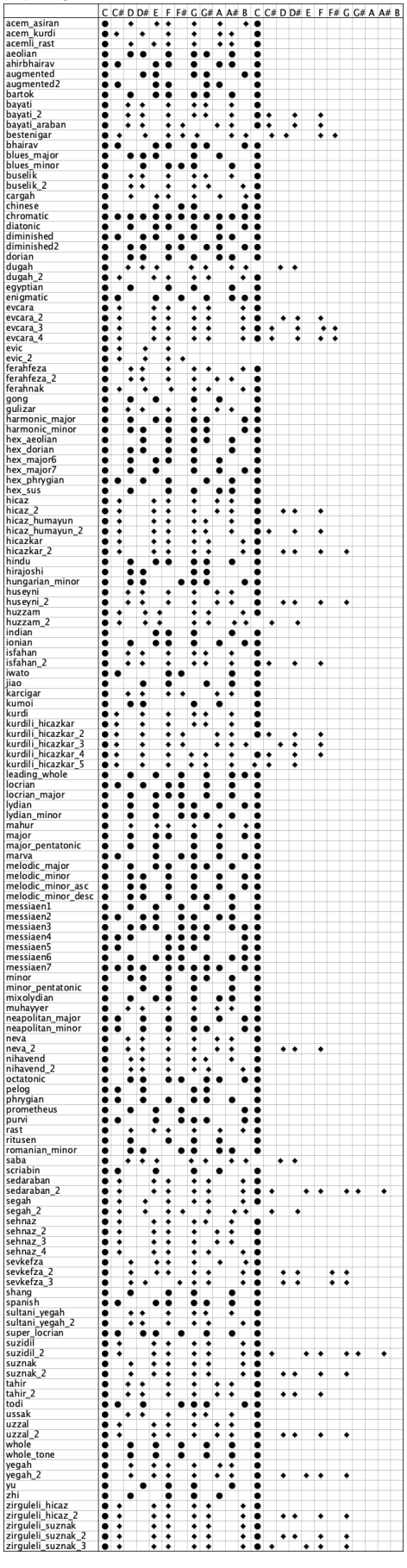
全てのスケールの名称をscale_namesから得て聞いてみるプログラム.

```
  i=0
  scale_names.length.times do
    puts i, scale_names[i], (scale :C4, scale_names[i]).length
    j=0
    (scale :C4, scale_names[i]).length.times do
      play (scale :C4, scale_names[i])[j], release: 0.1
      sleep 0.1
      j=j+1
    end
    sleep 0.5
    i=i+1
  end
  print scale_names.length
```

コードの音程一覧

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
chord name	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A
+5	●				●				●													
1	●																					
11	●				●			●			●				●			●				
11+	●				●			●			●				●				●			
13	●				●			●			●				●			●				●
5	●							●														
6	●				●			●		●												
6*9	●				●			●		●					●							
7	●				●			●			●											
7+5	●				●				●		●											
7+5-9	●				●				●		●			●								
7-10	●				●			●			●					●						
7-11	●				●			●			●						●					
7-13	●				●			●			●										●	
7-5	●				●		●				●											
7-9	●				●			●			●			●								
7sus2	●		●					●			●											
7sus4	●					●		●			●											
9	●				●			●			●				●							
9+5	●										●			●								
9sus4	●					●		●			●				●							
M	●				●			●														
M7	●				●			●				●										
a	●				●				●													
add11	●				●			●										●				
add13	●				●			●														●
add2	●		●		●			●														
add4	●				●	●		●														
add9	●				●			●							●							
augmented	●				●				●													
dim	●			●			●															
dim7	●			●			●			●												
diminished	●			●			●															
diminished7	●			●			●			●												
dom7	●				●			●			●											
halfdim	●			●			●				●											
halfdiminished	●			●			●				●											
i	●			●			●															
i7	●			●			●			●												
m	●			●				●														
m+5	●			●					●													
m11	●			●				●			●				●			●				
m11+	●			●				●			●				●				●			
m13	●			●				●			●				●			●				●
m6	●			●				●		●												
m6*9	●			●				●		●					●							
m7	●			●				●			●											
m7+5	●			●					●		●											
m7+5-9	●			●					●		●			●								
m7+9	●			●				●			●				●							
m7-5	●			●			●				●											
m7-9	●			●				●			●			●								
m7b5	●			●			●				●											
m9	●			●				●			●				●							
m9+5	●										●				●							
madd11	●			●				●										●				
madd13	●			●				●														●
madd2	●		●	●				●														
madd4	●			●		●		●														
madd9	●			●				●							●							
maj	●				●			●														
maj11	●				●			●				●			●			●				
maj9	●				●			●				●			●							
major	●				●			●														
major7	●				●			●				●										
min	●			●				●														
minor	●			●				●														
minor7	●			●				●			●											
sus2	●		●					●														
sus4	●					●		●														

スケールの音程一覧　●は整数（平均律）、◆は実数　横位置は音程に比例



エフェクトの使用

リバーブ

```
  with_fx :reverb do                              # room:1.0,dump:0,mix:0.5とすると強く響く (mix:1.0響きだけ)
    音
  end
```

エコーの例

```
  with_fx :echo, phase: 0.5, decay: 4 do          # 0.5拍毎に音を重ね、4拍で消えるエコー
    音
  end
```

ローパスフィルタ(lpf)は、play 文等の後ろに ,cutoff: 音程の数値 としても多様される。数値でなく :C4 等でもよい.
基音を残し、倍音成分を減らして音色を変えることができるため、初期のシンセからよく用いられてきた。

エフェクトの種類 42種　puts fx_names

:autotuner	:eq	:lpf	:nrhpf	:rbpf	:sound_out
:band_eq	:flanger	:mono	:nrlpf	**:record**	:sound_out_stereo
:bitcrusher	:gverb	:nbpf	:octaver	:reverb	:tanh
:bpf	:hpf	:nhpf	:pan	:rhpf	:tremolo
:compressor	:ixi_techno	:nlpf	:panslicer	:ring_mod	:vowel
:distortion	:krush	:normaliser	:ping_pong	:rlpf	:whammy
:echo	:level	:nrbpf	:pitch_shift	:slicer	:wobble

全てのエフェクトの名称をfx_namesから得て聞いてみるプログラム例.　:recordは、録音用の機能で使い方が異なるので除いています。

```
fx_names.length.times do |i|
  if i!=29 then
    with_fx fx_names[i] do
      play :C4,sustain:1
      sleep 2
    end
  end
end
```

```
# :record の使用例　2 秒録音する場合
with_fx :record, buffer: buffer[:a,2] do
  play 60,sustain:2
end
sleep 3
sample buffer[:a,2], pitch:1, pitch_dis:3
```

録音されたファイルは、下記にできます。 Sonic Pi 終了後も残ります

```
Mac /Users/ユーザ名/.sonic-pi/store/default/cached_samples/a.wav
Win C:/Users/ユーザ名/.sonic-pi/store/default/cached_samples/a.wav
```

サンプリング音源を鳴らす。音はflac形式で内臓されている。191種(v4.4.0-) puts all_sample_names

	:ambi_choir	ドラム	:drum_bass_hard		:elec_soft_kick		:loop_amen		:perc_till
	:ambi_dark_woosh		:drum_bass_soft		:elec_tick		:loop_amen_full	スネア	:sn_dolf
	:ambi_drone		:drum_cowbell		:elec_triangle		:loop_breakbeat		:sn_dub
	:ambi_glass_hum		:drum_cymbal_closed		:elec_twang		:loop_compus		:sn_generic
	:ambi_glass_rub		:drum_cymbal_hard		:elec_twip		:loop_drone_g_97		:sn_zome
	:ambi_haunted_hum		:drum_cymbal_open		:elec_wood		:loop_electric		:tabla_dhec
	:ambi_lunar_land		:drum_cymbal_pedal	グリッチ	:glitch_bass_g		:loop_garzul		:tabla_ghe1
	:ambi_piano		:drum_cymbal_soft		:glitch_perc1		:loop_industrial		:tabla_ghe2
	:ambi_sauna		:drum_heavy_kick		:glitch_perc2		:loop_mehackit1		:tabla_ghe3
	:ambi_soft_buzz		:drum_roll		:glitch_perc3		:loop_mehackit2		:tabla_ghe4
	:ambi_swoosh		:drum_snare_hard		:glitch_perc4		:loop_mika		:tabla_ghe5
	:arovane_beat_a		:drum_snare_soft		:glitch_perc5		:loop_perc1		:tabla_ghe6
	:arovane_beat_b		:drum_splash_hard		:glitch_robot1		:loop_perc2		:tabla_ghe7
	:arovane_beat_c		:drum_splash_soft		:glitch_robot2		:loop_safari		:tabla_ghe8
	:arovane_beat_d		:drum_tom_hi_hard	ギター	:guit_e_fifths		:loop_tabla		:tabla_ke1
	:arovane_beat_e		:drum_tom_hi_soft		:guit_e_slide		:loop_weirdo		:tabla_ke2
	:bass_dnb_f		:drum_tom_lo_hard		:guit_em9	me hack it	:mehackit_phone1		:tabla_ke3
	:bass_drop_c		:drum_tom_lo_soft		:guit_harmonics		:mehackit_phone2		:tabla_na
	:bass_hard_c		:drum_tom_mid_hard		:hat_bdu		:mehackit_phone3		:tabla_na_o
	:bass_hit_c		:drum_tom_mid_soft		:hat_cab		:mehackit_phone4		:tabla_na_s
	:bass_thick_c		:elec_beep		:hat_cats		:mehackit_robot1		:tabla_re
	:bass_trance_c		:elec_bell	ハイハット	:hat_gem		:mehackit_robot2		:tabla_tas1
	:bass_voxy_c		:elec_blip		:hat_gnu		:mehackit_robot3		:tabla_tas2
	:bass_voxy_hit_c		:elec_blip2		:hat_gump		:mehackit_robot4		:tabla_tas3
	:bass_woodsy_c		:elec_blup		:hat_hier		:mehackit_robot5		:tabla_te1
バスドラ	:bd_808		:elec_bong		:hat_mess		:mehackit_robot6		:tabla_te2
	:bd_ada		:elec_chime		:hat_metal		:mehackit_robot7		:tabla_te_m
	:bd_boom		:elec_cymbal		:hat_noiz	その他	:misc_burp		:tabla_te_ne
	:bd_chip		:elec_filt_snare		:hat_psych		:misc_cineboom		:tabla_tun1
	:bd_fat		:elec_flip		:hat_raw		:misc_crow		:tabla_tun2
	:bd_gas		:elec_fuzz_tom		:hat_sci	パーカッション	:perc_bell		:tabla_tun3
	:bd_haus		:elec_hi_snare		:hat_snap		:perc_bell2		:vinyl_backspin
	:bd_klub		:elec_hollow_kick		:hat_star		:perc_door		:vinyl_hiss
	:bd_mehackit		:elec_lo_snare		:hat_tap		:perc_impact1		:vinyl_rewind
	:bd_pure		:elec_mid_snare		:hat_yosh		:perc_impact2		:vinyl_scratch
	:bd_sone		:elec_ping		:hat_zan		:perc_snap		
	:bd_tek		:elec_plip		:hat_zap		:perc_snap2		
	:bd_zome		:elec_pop		:hat_zild		:perc_swash		
	:bd_zum		:elec_snare		:loop_3d_printer		:perc_swoosh		

音の長さだけ伸ばすには、sleep **sample_duration** サンプル名

一拍をサンプルの長さにするには、**use_sample_bpm** サンプル名

```
sample :loop_amen           # 1969年のThe Winstonsの曲「Amen Brother」による(原曲はYoutubeにあり)
```

rate: 2 等でサンプリング速度を変えることによって音程を変えることができる。1がデフォルト、-1で逆再生も可.

synth音源と比較して、音程を変化させたメロディーの演奏にはあまり適さない。

```
sample :guit_harmonics, beat_stretch: 2      # 指定した長さにする．音程も変わる．
sample :guit_harmonics, rate: 2     # 1オクターブ音程を上げる,長さも半分になる rpitch: 12でも1オクターブ上
sample :guit_harmonics, pitch: 12   # 長さを変えずに1オクターブ音程を上げる
sample :ambi_choir, pitch_stretch: 6 # 音程を変えずに長さを変える．最長で元の長さの4倍まで
sample :ambi_choir, pitch: 0.01 , pitch_dis: 0.5 # ランダムにピッチを揺らす. pitch:0 の場合揺れないこともある
puts sample_duration :ambi_choir    # サンプル音の長さ(秒数) を表示
puts all_sample_names               # サンプル音のすべて表示する
puts all_sample_names.length        # サンプル音の総数を表示する
puts sample_groups                  # サンプル音のグループ名を表示する
puts sample_names :drum             # グループdrumのサンプル音名を表示する
```

グループ guit だけを再生するプログラム例。

```
g='guit'
(sample_names g.to_sym).length.times do |i|
  sample g,i
  sleep sample_duration g,i
end
```

サンプリング音をファイルから読み込む。wav, aiff, mp3, flac, ogg が読み込める。m4a(AAC),wmaは不可

startとfinishで開始,終了を指定できる(0～1で指定.全長が1) 内蔵のsampleと同じ操作rate等が可能。

```
sample "/Users/XXXX/my_sound.wav", start:0.5, finish:0.6, rate:2, pan: 0.5
```

逆再生もできる．下記はいずれも全体を逆再生となる.(スクラッチ音等としても利用できる.)

```
sample "/Users/XXXX/my_sound.wav", start:0, finish:1, rate:-1
sample "/Users/XXXX/my_sound.wav", start:1, finish:0, rate: 1
```

全てのサンプル音源を順に名称と長さを表示して聞いてみるプログラムの例．

```
all_sample_names.each do |s|        # all_sample_namesを(sample_names :elec)等に交換可
  sample s
  sleep sample_duration s
end
```

または、

```
all_sample_names.each do |s|
  use_sample_bpm s #1拍をサンプルの長さにする
  sample s
  sleep 1
end
```

または、

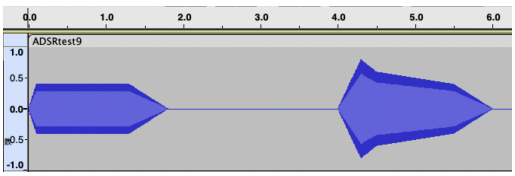
```
all_sample_names.length.times do
  sample all_sample_names.tick
  sleep (sample_duration all_sample_names.look)
end
```

これも同様に、グループ毎に鳴らす

```
for i in 0...sample_groups.length
  for j in 0...(sample_names sample_groups[i]).length
    sample (sample_names sample_groups[i])[j]
    sleep sample (sample_names sample_groups[i])[j].length
  end
end
```

グループbdだけを鳴らす

```
(sample_names :bd).length.times do
  sample (sample_names :bd).tick
  sleep 1
end
```



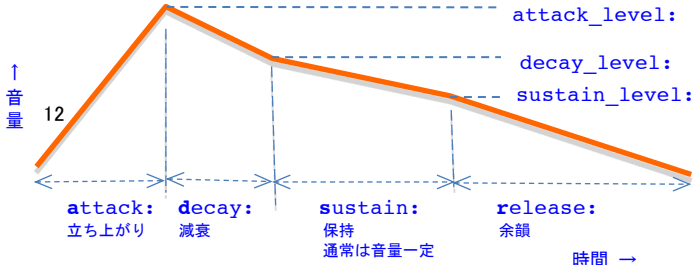
ADSRパラメータの設定 (synth、sample音源共)

多くのシンセサイザーで使われている楽器音を模した音の強度の時間変化の設定

```
play :C4,attack:0.1,decay:0.5,sustain:5,release:0.5
sleep 4
play :C4,attack:0.3,attack_level:2,decay: 0.2, decay_level: 1.5 ,sustain:1,
        sustain_level:1.0, release:0.5, release_level:0.2
```

ピアノの音もattack時間を与えるとバイオリンのようになります．

```
use_synth :piano
```



```
play 60
sleep 1
play 60, attack:0.2
sleep 1
-----
args={ amp:0.5, attack:0.1 , release: 0.3}
play :C4,args   #という使い方もできる
```

ポルタメント（controlと_slide） 音程等を連続的に変化させる.

```
  例:2拍づつ,内,変化時間0.5拍でC5～C6まで変化させる
  s = play scale(:C5,:major).tick, sustain: 16, note_slide: 0.5 # 全体16,変化に要する時間0.5
  sleep 2
  7.times do
    control s, note: scale(:C5,:major).tick
    sleep 2
  end
```

エフェクタのslide 連続変化 エフェクタwobbleの振幅の変化時間を1拍から0.015拍に10拍かけてずらす.

```
  with_fx :wobble, phase: 1, phase_slide: 10 do |e|   # 1拍での変化から10拍かけてずらす
    use_synth :dsaw
    play 50, sustain:5, release:5
    control e, phase: 0.015          # エフェクタwobbleの振幅の時間を1拍から0.015拍に10拍かけてずらす.
  end
```

エフェクタ名の:slicerは前に:が付き、オプションのmix:には後ろに:がつく。
mix:0は、効果0となる。mix:1が効果が最大

```
  with_fx :slicer, mix: 0.5, phase: 0.5 do   # mix: 0.5は、オリジナルとエフェクトが半々となる.
    play 60, sustain: 3                      # phase: 0.5は、0.5拍で揺れの振幅が1回.
    sleep 3
  end
```

density n　ループ内のsleep値の合計の時間内にループをn回 回す

```
a=beat
density 6 do  # この回数の値を変えても
  play 72, release: 0.05
  sleep 1
  play 72, release: 0.05
  sleep 0.5
end
puts beat-a   # 常にsleep値の合計の =1.5 となる回数だけループが回る
```

拍数や時間を示す変数
起動からの実時間を得る **beat**
現在のスレッドの開始からの拍数を得る **vt**, virtual time, 例 play 60 if vt>=4 && vt<=6 # 6～8拍のみで鳴らす
秒数を与える **rt**, real time, sleep rt(1)は、bpmにかかわらず1秒
beatは、Ver4からSonic Piの起動からの時間となりました。スレッドの開始からの時間は下記でも代用可

```
  use_bpm 600
  beat0=beat
  sleep rt(1)  # BPM=600なので1秒=10拍
  puts beat-beat0,vt  # は、拍数=10.0 秒数=1.0
```

等間隔、比例配分の ring を生成する line （区間 a,b を step 等分して、a から step 回繰り返すので b は含まれない）

```
puts line(0,1,steps: 5)                          は、(ring 0.0, 0.2, 0.4, 0.6, 0.8)
puts line(0,1,steps: 5, inclusive: true)         は、(ring 0.0, 0.25, 0.5, 0.75, 1.0)
puts line(0,1,steps: 4).reflect                  は、(ring 0.0, 0.25, 0.5, 0.75, 0.5, 0.25, 0.0)
```

同じ繰り返しのringを作成する **knit**

```
knit 60,3                                        は、(ring 60, 60, 60)
knit :C4,2,:F4,3                                 は、(ring :C4, :C4, :F4, :F4, :F4)
```

オクターブ音を返す**octs**

```
octs :C3,3                                       は、(ring 48, 60, 72)
 play octs :C3,3 は3音が同時に鳴る
```

while～end 文　例:4 回 60 を鳴らして、その後 72 が鳴る

```
  while vt<4
    play 60
    sleep 1
  end
  play 72
```

until文

```
b=0
until b>=4
  play 60
  sleep 1
  b=b+1
end
```

if～end **文**　例:C4 C4 C5 C5 A4 A4 C5 C5 C5...

```
  loop do
    if vt<2                          # virtual time スレッドが開始してからの時間
      play 60
    elsif vt>=4 && vt<6              # else if は、つなげてelseの末尾のeを取り、elsif とする
      play 67
    else
      play 72
    end
    sleep 1
  end
```

case～**when 文**

```
  case rrand_i(1,3)         # 数値、文字等
  when 0
    play 60
  when 1,3          # "," で併記も可
    play 62
  else
    play 72
  end
```

for 文 で**リスト**の要素を一つづつ取り出して使うことができる.

```
  str = [:C4, :E4, :G4]
  for s in str
    play s
    sleep 1
  end
```

break で無限**ループを抜ける**ことができる.

```
i=60
loop do
  play i
  sleep 0.2
  break if i==65
  i=i+1
end
```

density

```
a=beat
density 8 do # この値8を変えても  ループ内総実行時間が、1+0.5になるようにsleep値を調整して8回ループを回す.
  play 72, release: 0.05
  sleep 1
  play 72, release: 0.05
  sleep 0.5
end
puts beat-a   # 常に =1.5
```

use_debug false とすると、ログの出力が減り puts文の出力だけが残るのでデバッグに活用できる.
　画面出力が多場合には、これが律速となってしまう場合もあるので使うこともある。

数学関数等　**Math::sqrt(x)** , Math::log10(x) , Math::sin(x) , Math::atan2(y/x), Math::PI など.

Math.sin(x) , Math.cos(x) 等の使い方も可
累乗は **, ^は排他的論理和 XOR の意味, AND は&&, OR は||, 絶対値は -1.abs, 整数化は.to_i, 実数化は.to_f,
四捨五入は.round, 桁指定の四捨五入は.round(floor=0), 文字列化.to_s, 配列化.to_a, 文字列をシンボル
化 .to_sym, 10 進数を 2 進数に.to_s(2)、10 進数を 8 進数に.to_s(8)

```
puts "aaa".to_sym               :aaa
puts "aaa".class                String
puts "aaa".to_sym.class         Symbol
10進数を2進数のリストに変換する。puts 255.to_s(2).chars.map(&:to_i)    [1, 1, 1, 1, 1, 1, 1, 1]
```

日付時間を取得

Time.new　　現在の時間 2016-09-14 19:36:28 +0900 等
時間の各数値は、Time.new.year , Time.new.month , Time.new.day , Time.new.hour , Time.new.min , Time.new.sec

range(60,70,2) は、60以上70未満を2づつのringを作成(ring 60,62,64,66,68)。inclusive:trueを加えると70以下となる

文字列に変換 .to_s

```
array=["a", 1]
puts array.to_s , array.to_s.length  # 配列を文字列に変換すると[]も含めた文字列となる
["a", 1] 8
```

連番の配列やリングを作る

```
a=(1..53).to_a
b=(1..53).to_a.ring
```

文字列関数は多数あり 参考 http://ref.xaio.jp/ruby/classes/string

```
s="12345"
puts s.length      # returns 5                              文字列の長さ
puts s.index("3")   # returns 2                             文字の位置を検索する
puts s.slice(1, 3)  # 0から数え、1番目から3文字分  returns 234   部分文字列を取り出す
puts s.slice(2..4)  #         2～4  returns 345
puts s.slice(2...4) #         2～4未満  returns 34
```

ローカル変数とグローバル変数

in_thread や live_loop 内で初めて定義した変数はローカル変数となる。
ループの外で定義した変数はグローバル変数となる。ループのカウント変数 i の使い方は要注意。
プログラムは、上から下に実行されてゆくので、全体で使いたい変数は、プログラムの最初に定義するとよい.
注意
以下の.shuffle行がどちらの行にあるかによって音階の内容は異なってきます。

```
use_random_seed 0
notes=(chord :C3, :major, num_octaves: 3).shuffle          # ランダムなCの音階が1回定義される
8.times do                                                 #   上記は9音単位で同じ音が出てこない.
                                  # 行がここにあった場合は、  # ランダムなCの音階がその都度新たに定義される。
  play notes.tick                                          #   上記は全ての音が1/9の確率で出現する.
  sleep 0.25
end
```

後者は、下記と同様の操作となります。(実際に鳴る、乱数として選ばれる音は異なる) 全ての音が1/9の確率で出現する.

```
8.times do
  notes=(chord :C3, :major, num_octaves: 3).choose
  play notes
  sleep 0.25
end
```

整数と実数の混在した計算式での注意 4,5行目の挙動は、他言語と異なる

```
puts 1/2     # = 0
puts 1/2.0   # = 0.5
puts 1.0/2   # = 0.5
puts 1./2    # = 0
puts 1/2.    # このような行があると、プログラム自体がエラーで実行できなくなる。(Rubyに由来)
```

複数の命令をまとめて管理し実行 = -> { } , [ ].map(&:call)

```
c = -> { play :C, sustain: 1 ; sleep 1 }     # cを{ }でまとめた複数の命令とする.
e = -> { play :E, sustain: 1 ; sleep 1 }
g = -> { play :G, release: 2 ; sleep 1 }
```

```
[c,e,g,c].map( &:call )             # 上で定義された[]の中を順に実行する.
[c,e,g,c].map{ |x| x.call }         # 同上
```

キーボードショートカット RUN command(alt)+R, STOP command+S, 行頭へctrl+A, 行末へctrl+E

デバッグ情報の表示　puts **status**
縦棒グラフの表示　**spark** (range 1, 10) または　puts **spark_graph** (range 1, 10)で　 

ファイルから値を１つだけ読み込む
```
a =File.read("/Users/???/test.txt").to_f  # test.txtから値を読み込みfloat型としてaに代入
a =File.read("/Users/???/test.txt").to_s  #                          文字型
```

実行時の遅れ(latency)を減らすため、Runから実行までの時間を与える
```
set_sched_ahead_time! 0.1
use_sched_ahead_time 2
```

Mac版の場合、Sonic Pi.appのパッケージの内容を表示→etc→samplesにflac形式のサンプリング音源ファイルが置いてある。
例については、同→etc→examplesに.rbファイルとして置いてある。
同→etc→synthdefsには、Sonic Piの元となっているSuperColliderのsynthdefファイル等が置いてある。GraphVizの.dotファイルやPDF化したグラフもある。
etc/snippets/fxでは、fxの種類がわかる。
同/etc/doc/cheatsheets/synths.md
同/etc/doc/cheatsheets/fx.md
同/etc/doc/cheatsheets/samples.md
は、テキストファイルであり、それぞれの種類や機能がわかる。

10個のBufferの内容は、下記のテキストファイルに保存されています。
$HOME/.sonic-pi/store/default/workspace_zero.spi ～ workspace_nine.spi

Sonic Piのプログラムが格納されている外部ファイルから実行する方法 (ver.2.11から)
run_file "ファイルのパス/bigFile.rb"
　バッファに入りきれない大きなプログラムも実行できるのが利点

素数を使う　素数階段。一段登ると音程が半音上がる.
```
require 'prime' # 素数階段
a=:C0
use_synth :piano
use_debug false
700.times do |i| # 700以上に増やすとナイキスト周波数を超えて音の意味がなくなります。
  play a
  sleep 0.125
  if Prime.prime?(i+1) then
    puts "MIDI= #{a}  #{midi_to_hz(a)} Hz  素数= #{i+1}"
    a=a+1
  end
end
```

テキストファイル経由でSonic Piを制御する例（→後述）(OSCが使えるようになり不要となった)

with_swingによって1拍目を、with_swingが実行される4回に1回リズムを遅らせることができます。 0.1を-0.1とすると早くなる。 pulse:3 とすると3拍に1回とできる。 offset:2とすると3拍目をずらすことができる
```
live_loop :LL1 do
  with_swing 0.1 do
    sample :drum_tom_lo_soft
  end
  sleep 0.5
end
live_loop :LL2 do
  sample :drum_cymbal_closed
  sleep 0.5
end
```

[Help]の[Lang]の[**spread**] 世界の民族音楽のリズム、ユークリッドリズム、ポリリズムの作成など
Creates a new ring of boolean values which space a given number of accents as evenly as possible throughout a bar. This is an implementation of the process described in 'The Euclidean Algorithm Generates Traditional Musical Rhythms' (Toussaint 2005).
[spread]の例

```
live_loop :euclid_beat do
  sample :elec_bong, amp: 1.5 if (spread 3, 8).tick   # Spread 3 bongs over 8
  # (spread 3, 8) によって (ring true , false, false, true , false, false, true , false)が生成され、tick によって取
    り出される要素が順番に変わる. true のタイミングで sample 音源の elec_bong が鳴らされ、false の場合は鳴らない.

  sample :perc_snap, amp: 0.8 if (spread 7, 11).look  # Spread 7 snaps over 11
  # 前のtickによって同時にこの行のringの要素も進むので、ここではlookのみでtrue/falseを判定し、音を出す.

  sample :bd_haus, amp: 2 if (spread 1, 4).look       # Spread 1 bd over 4
  sleep 0.125
end
```

3つのリズムは、trueを1,falseを0とすると、

```
1 0 0 1 0 0 1 0|1 0 0 1 0 0 1 0|1 0 0 1 0 0 1 0|1 0 0 1 0 0 1 0|
1 0 1 0 1 1 0 1 1 0 1|1 0 1 0 1 1 0 1 1 0 1|1 0 1 0 1 1 0 1 1 0 1|
1 0 0 0|1 0 0 0|1 0 0 0|1 0 0 0|1 0 0 0|1 0 0 0|1 0 0 0|
```

spreadの示す値の例

```
spread(1,8)                (ring true , false, false, false, false, false, false, false)
spread(2,8)                (ring true , false, false, false, true , false, false, false)
spread(3,8)                (ring true , false, false, true , false, false, true , false)
spread(4,8)                (ring true , false, true , false, true , false, true , false)
spread(5,8)                (ring true , false, true , false, true , true , false, true )
spread(6,8)                (ring true , false, true , true , true , false, true , true )
spread(7,8)                (ring true , false, true , true , true , true , true , true )
spread(8,8)                (ring true , true , true , true , true , true , true , true )

spread(3,8)                (ring true , false, false, true , false, false, true , false)
spread(3,8, rotate: 1)     (ring true , false, false, true , false, true , false, false)
spread(3,8, rotate: 2)     (ring true , false, true , false, false, true , false, false)
spread(3,8, rotate: 3)     (ring true , false, false, true , false, false, true , false)

spread(3,8)                (ring true , false, false, true , false, false, true , false)
spread(3,8).rotate(1)      (ring false, false, true , false, false, true , false, true )  # 要素を回転
spread(3,8).rotate(2)      (ring false, true , false, false, true , false, true , false)
spread(3,8).rotate(3)      (ring true , false, false, true , false, true , false, false)
```

詳細は、論文'The **Euclidean Algorithm** Generates Traditional Musical Rhythms'.
(ユークリッド アルゴリズムが伝統的な音楽リズムを生成)
Godfried Toussaint, Renaissance Banff: Mathematics, Music, Art, Culture, 2005
http://cgm.cs.mcgill.ca/~godfried/publications/banff.pdf

helpより

```
(spread 2, 5)     # 13世紀のペルシャのリズム、Khafif-e-ramal.
                  (ring true, false, false, true, false)
(spread 3, 4)     # コロンビアのカンブリアの典型的なパターンと、トリニダードのカリプソのリズム。
                  (ring true, false, true, true)
(spread 3, 5)     # 2番目のオンセットで始まるのは、Khafif-e-ramalという名の13世紀のペルシャのリズムと、ルーマニアの民族舞踊のリズムである。
                  (ring true, false, true, false, true)
(spread 3, 7)     # ブルガリアの民族舞踊で使われるルチェニッツァのリズム
                  (ring true, false, false, true, false, true, false)
(spread 3, 8)     # キューバのトレシージョ・パターン
                  (ring true, false, false, true, false, false, true, false)
(spread 4, 7)     # もう1つのルチェニツァ・ブルガリアの民族舞踊のリズム
                  (ring true, false, true, false, true, false, true)
(spread 4, 9)     # トルコのアクサクのリズム
                  (ring true, false, false, true, false, true, false, true, false)
(spread 4, 11)    # フランク・ザッパがOutside Nowで使用した拍子のパターン
                  (ring true, false, false, true, false, true, false, false, true, false)
(spread 5, 6)     # アラブのポピュラーなリズムであるヨーク・サマイ・パターンは、セカンド・オンセットから始めると得られる.
                  (ring true, false, true, true, true, true)
(spread 5, 7)     # ナワカート・パターン、これもアラブのポピュラーなリズム
                  (ring true, false, true, true, false, true, true)
(spread 5, 8)     # キューバのチンキージョ・パターン.
                  (ring true, false, true, false, true, true, false, true)
(spread 5, 9)     # Agsag-Samaiと呼ばれるアラブのポピュラーなリズム.
                  (ring true, false, true, false, true, false, true, false, true)
(spread 5, 11)    # ムソルグスキーが『展覧会の絵』で使用した拍子パターン
                  (ring true, false, false, true, false, true, false, true, false, true, false)
(spread 5, 12)    # 南アフリカの子供の歌のヴェンダ拍手のパターン.
                  (ring true, false, false, true, false, true, false, false, true, false, true, false)
(spread 5, 16)    # ブラジルのボサノバ・リズム・ネックレス.
```

```
                    (ring true, false, false, false, true, false, false, true, false, false, true, false, false, true,
                    false, false)
(spread 7, 8)       # ベンディール(枠太鼓)で演奏される典型的なリズム (frame drum)
                    (ring true, false, true, true, true, true, true)
(spread 7, 12)      # 西アフリカの一般的な鈴のパターン.
                    (ring true, false, true, false, true, false, true, true, false, true, false, true)
(spread 7, 16)      # ブラジルのサンバ・リズム・ネックレス.
                    (ring true, false, false, true, false, true, false, true, false, false, true, false, true, false,
                    true, false)
(spread 9, 16)      # 中央アフリカ共和国で使われているリズム・ネックレス.
                    (ring true, false, true, false, true, false, true, false, true, true, false, true, false, true,
                    false, true)
(spread 11, 24)     # 中央アフリカのアカ・ピグミー族のリズム・ネックレス.
                    (ring true, false, false, true, false, true, false, true, false, true, false, true, false, false,
                    true, false, true, false, true, false, true, false, true, false)
(spread 13, 24)     # サンガ上部のアカ・ピグミーのリズム・ネックレス
                    (ring true, false, true, false, true, false, true, false, true, false, true, false, true, true,
                    false, true, false, true, false, true, false, true, false, true)
```

WikipediaのEuclidean rhythmには、オープンソース・ミュージック・ハードウェア・プロジェクトのMIDIPal, Gridsなどへのリンクもあり.
https://en.wikipedia.org/wiki/Euclidean_rhythm

LireOfficeのPresentationのサウンド(ライセンスはGPL)をサンプリング音源として利用する

```
s=["apert.wav","apert2.wav","applause.wav","beam.wav","beam2.wav","cow.wav","curve.wav","drama.wav","ex
plos.wav","falling.wav","glasses.wav","gong.wav","horse.wav","kling.wav","kongas.wav","laser.wav","left
.wav","nature1.wav","nature2.wav","ok.wav","pluck.wav","roll.wav","romans.wav","soft.wav","space.wav","
space2.wav","space3.wav","sparcle.wav","strom.wav","theetone.wav","top.wav","train.wav","untie.wav","up
s.wav","wallewal.wav"]
# (macOSでのLibreOffice ver7.5.1.2の場合、バージョンによってパスは異なる)
path="/Applications/LibreOffice.app/Contents/Resources/gallery/sounds/"
s.length.times do |i|
  sample path+s[i]
  sleep sample_duration path+s[i]   # サンプルごとの長さをsleep値とする
  sleep 0.1
end
```

同上　ファイル名を指定しなくても可能な方法(パスのみを与えてファイル名は自動取得できる. 数は取得できない？)

参考 https://github.com/samaaron/sonic-pi/blob/master/etc/doc/tutorial/03.7-Sample-Packs.md

```
path="/Applications/LibreOffice.app/Contents/Resources/gallery/sounds/"
i=0
loop do
  sample path,i
  sleep sample_duration path,i
  i=i+1
end
```

変数値の出力方法

```
5.downto(1) do |i|
  puts "#{i}回目"
end
```

forループのステップを与える

```
for i in (0..12).step(1/12.0)
  play 60+i
  sleep 0.1
end
```

to_a Arrayに変換. grepで検索

```
puts all_sample_names.to_a.grep(/closed/)
[:drum_cymbal_closed]
```

0から1までを0.25刻みでそれぞれの区間を２つづつの配列とする。

```
puts (0..1).step(1/4.0).each_cons(2).to_a
[[0.0, 0.25], [0.25, 0.5], [0.5, 0.75], [0.75, 1.0]]

puts (0..1).step(1/4.0).each_cons(3).to_a
[[0.0, 0.25, 0.5], [0.25, 0.5, 0.75], [0.5, 0.75, 1.0]]

puts (0...1).step(1/4.0).each_cons(2).to_a
[[0.0, 0.25], [0.25, 0.5], [0.5, 0.75]]
```

```
puts (0..1).step(1/4.0).to_a
[0.0, 0.25, 0.5, 0.75, 1.0]
```

macOSのsayコマンドでジャベラせる。 [rec]では録音できず、soundflower(Intel)やBlackHole(M1以降)で録音可

```
system('say Hello world')
```

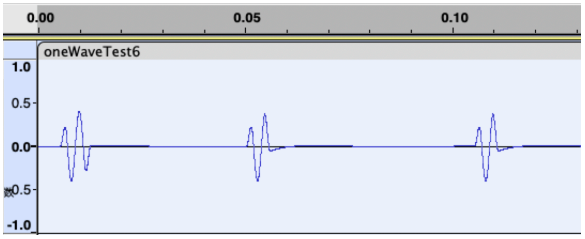
チューニングの違いを聞き、スコープで見る　 唸りが速い方がより音程が異なっています。

```
use_synth_defaults sustain:4, release:1
#平均律と純正律
8.times do
  a=[:c,:d,:e,:f,:g,:a,:b,:c5]
  use_tuning :equal
  play a.tick,pan:-1
  use_tuning :just
  play a.look,pan:1
  sleep 5
end
sleep 2

#平均律とピタゴラス
8.times do
  a=[:c,:d,:e,:f,:g,:a,:b,:c5]
  use_tuning :equal
  play a.tick,pan:-1
  use_tuning :pythagorean
  play a.look,pan:1
  sleep 5
end
sleep 2

#平均律とミーントーン
8.times do
  a=[:c,:d,:e,:f,:g,:a,:b,:c5]
  use_tuning :equal
  play a.tick,pan:-1
  use_tuning :meantone
  play a.look,pan:1
  sleep 5
end
```

play 時間の限界 :sine, :beep の場合

```
play 60,attack:0,release:0,sustain:0.0030
sleep 0.05
play 60,attack:0,release:0,sustain:0.0029
sleep 0.05
play 60,attack:0,release:0,sustain:0.0028
sleep 0.05
```

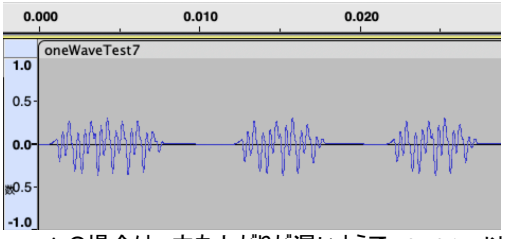
Audacity



0.0030sと0.0029sは異なるが、0.0029sと0.0028sの波形は同じなので、0.0029sまでが有効な最小時間

0.0001sや0.00001sとしても波形は変わりませんでした。なお、時間をattackまたはreleaseで与えても波形は同じでした。

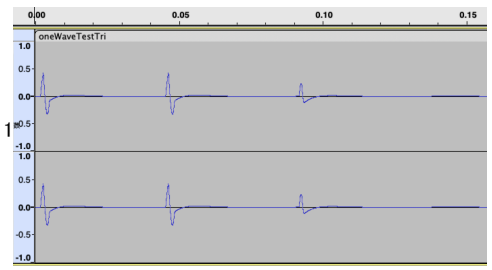
:zawaの場合も同様でした。



:triの場合は、立ち上がりが遅いようで、0.017以下では音が出ず、0.018または0.019が限界でした。

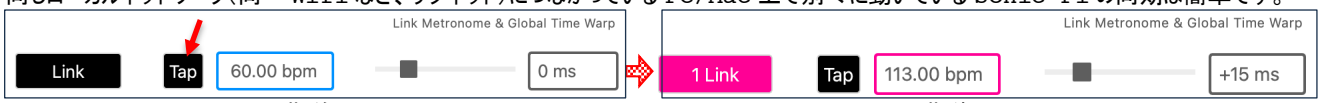
0.017s以下では音がなくなりました。

```
use_synth :tri
play 60,attack:0,release:0,sustain:0.020
sleep 0.05
```



```
play 60,attack:0,release:0,sustain:0.019
sleep 0.05
play 60,attack:0,release:0,sustain:0.018
sleep 0.05
play 60,attack:0,release:0,sustain:0.017
sleep 0.05
```

同じローカルネットワーク(同一 Wifi など、サブネット)につながっている PC/Mac 上で別々に動いている Sonic Pi の同期は簡単です。



同期前　　同期後

2台の PC/Mac で Sonic Pi を起動して live_loop や無限ループ等を動かしておきます。
例
```
loop do
  sample :drum_cymbal_closed
  sleep 1
end
```
音色はそれぞれで異なった方がわかりやすいでしょう。
2台のそれぞれでウィンドウ右側中央にある[Link Metronome & Global Time Warp]の黒い**[Link]**ボタンを押すと、もう一台とリンクし、赤い**[1 Link]**となり同期が開始します。
青枠で囲まれた bpm 値を変えると、もう一台で動いている Sonic Pi の bpm も同じ速度となり、ループの速度が同期します。
[Tap]をマウスクリックして bpm を与えることもできます。
このとき、各 Sonic Pi でのビートのタイミングは、ずれていることが殆どなので、どちらかでスライダーを操作して ms 単位で合わせることができます。
3台の場合にも、同様に**[2 Link]**となり3台が同期します。
live_loop を書き換えて再度[run]しても同期は続いていますので、ライブコーディングでも活用できます。

大文字を小文字に変換する
```
"Sonic Pi".downcase
```
小文字を大文字に変換する
```
"Sonic Pi".upcase
```
大文字と小文字を入れ替える
```
"Sonic Pi".swapcase
```
先頭の小文字を大文字にする
```
"Sonic Pi".capitalize
```
動作せずエラーとなる。!をつけることによって元の文字列を破壊的に置き換える。

任意の英文を音として鳴らしてみましょう。
```
str="Welcome to Sonic Pi".upcase  # 任意の文字列を全て大文字にする
abc="ABCDEFGHIJKLMNOPQRSTUVWXYZ"  # 音の順番を与える
len=str.length
len.times do |i|                  # 文字列 str の長さまで繰り返す
  str1=str.slice(i,1)             # 文字列 str の先頭から1文字づつを取り出し str1 に格納する
  idx=abc.index(str1)             # str1 の文字が文字列 abc の何番目かを idx に格納する
  puts i,str1,abc.index(str1)
  play scale(:C4,:major_pentatonic ,num_octaves:6)[idx] if idx != nil  #スケールの idx 番目の音を鳴らす。abc に存在しない文字の場合は音を鳴らさないので休符となる。
  sleep 0.25
end
```

全角文字も入れた場合
```
str="歓迎 Welcome to Sonic Pi".upcase
abc="ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"  # 26 文字+10
str.each_char do |str1a|
  str1= str1a.chr
  if str1.bytesize==1
    idx=abc.index(str1)           # 半角文字が abc の何番目か
  elsif str1.bytesize==3
```

```
    idx=str1.ord % abc.length    # 全角文字のコードを abc の長さで割った余り
  end
  play scale(:C3,:major_pentatonic ,num_octaves:8)[idx] if idx != nil
  sleep 0.25
end

無限に上昇する音階 Shapard tone
use_synth :sine #:blade
namp=(:C10-:C1)+1
amp=(0.3..0.05).step(-(0.3-0.05)/namp).to_a
define :shepard do
  (note(:C1)..note(:C10)).each do |n|
    play n,attack:0.1,release:0.1,amp: amp[n-:C1.to_i]
    sleep 0.1 #rrand(0.1,0.15)
  end
end

40.times do
  in_thread do
    shepard
  end
  sleep 0.5 #rrand(0.8,1.2)
end
```

MIDI ノート番号と音名と周波数　（平均律、A4=440Hz の場合）

$$周波数 = 440 \times \left(2^{\frac{1}{12}}\right)^{MIDI番号\ -\ 69}$$

チューニング　　周波数を知りたい音

Sonic Pi で各音の周波数を表示するには、puts *440*(2**(1.0/12.0))**(:A4*-69)

低音側

MIDI	音名	周波数
0	C−1	8.18
1	Cs−1	8.66
2	D−1	9.18
3	Ef−1	9.72
4	E−1	10.30
5	F−1	10.91
6	Fs−1	11.56
7	G−1	12.25
8	Af−1	12.98
9	A−1	13.75
10	Bf−1	14.57
11	B−1	15.43
12	C0	16.35
13	Cs0	17.32
14	D0	18.35
15	Ef0	19.45
16	E0	20.60
17	F0	21.83
18	Fs0	23.12
19	G0	24.50
20	Af0	25.96
21	A0	27.50
22	Bf0	29.14
23	B0	30.87
24	C1	32.70
25	Cs1	34.65
26	D1	36.71
27	Ef1	38.89
28	E1	41.20
29	F1	43.65
30	Fs1	46.25
31	G1	49.00
32	Af1	51.91
33	A1	55.00
34	Bf1	58.27
35	B1	61.74

88 鍵ピアノ

MIDI	音名	周波数	
36	C2	65.41	
37	Cs2	69.30	
38	D2	73.42	
39	Ef2	77.78	
40	E2	82.41	
41	F2	87.31	
42	Fs2	92.50	
43	G2	98.00	
44	Af2	103.83	
45	A2	110.00	
46	Bf2	116.54	
47	B2	123.47	
48	C3	130.81	
49	Cs3	138.59	
50	D3	146.83	
51	Ef3	155.56	
52	E3	164.81	
53	F3	174.61	
54	Fs3	185.00	
55	G3	196.00	
56	Af3	207.65	
57	A3	220.00	
58	Bf3	233.08	
59	B3	246.94	
60	**C4**	**261.63**	中央のド
61	Cs4	277.18	
62	D4	293.66	
63	Ef4	311.13	
64	E4	329.63	
65	F4	349.23	
66	Fs4	369.99	
67	G4	392.00	
68	Af4	415.30	
69	**A4**	**440.00**	時報や音叉のラ
70	Bf4	466.16	
71	B4	493.88	

MIDI	音名	周波数
72	C5	523.25
73	Cs5	554.37
74	D5	587.33
75	Ef5	622.25
76	E5	659.26
77	F5	698.46
78	Fs5	739.99
79	G5	783.99
80	Af5	830.61
81	A5	880.00
82	Bf5	932.33
83	B5	987.77
84	C6	1046.50
85	Cs6	1108.73
86	D6	1174.66
87	Ef6	1244.51
88	E6	1318.51
89	F6	1396.91
90	Fs6	1479.98
91	G6	1567.98
92	Af6	1661.22
93	A6	1760.00
94	Bf6	1864.66
95	B6	1975.53
96	C7	2093.00
97	Cs7	2217.46
98	D7	2349.32
99	Ef7	2489.02
100	E7	2637.02
101	F7	2793.83
102	Fs7	2959.96
103	G7	3135.96
104	Af7	3322.44
105	A7	3520.00
106	Bf7	3729.31
107	B7	3951.07

MIDI	音名	周波数
108	C8	4186.01
109	Cs8	4434.92
110	D8	4698.64
111	Ef8	4978.03
112	E8	5274.04
113	F8	5587.65
114	Fs8	5919.91
115	G8	6271.93
116	Af8	6644.88
117	A8	7040.00
118	Bf8	7458.62
119	B8	7902.13
120	C9	8372.02
121	Cs9	8869.84
122	D9	9397.27
123	Ef9	9956.06
124	E9	10548.08
125	F9	11175.30
126	Fs9	11839.82
127	G9	12543.85
128	Af9	13289.75
129	A9	14080.00
130	Bf9	14917.24
131	B9	15804.27
132	C10	16744.04
133	Cs10	17739.69
134	D10	18794.55
135	Ef10	19912.13
136	**E10**	**21096.16**

88 鍵ピアノ

高音側

サンプリング周波数 44.1kHz で表せる限界のナイキスト周波数 22.05kHz