
Sonic Pi
Sam Aaron
&
The Core Team
Developed at the
University of Cambridge Computer Laboratory
with kind support from the Raspberry Pi Foundation
Version 2.6

日本語版

Part 2

Part 2

5 章　プログラミングの構造

これまでの章で、みなさんは play や sample コマンドを使ってサウンドを作り出したり、sleep を使ってシンプルなメロディーやリズムを作曲して、音楽制作（サウンドプログラミング）の基礎を学ぶことができました。

Sonic Pi のコードの世界で他にどんなことができるか興味が出てきたことでしょう…

それでは、プログラミングの基礎となるループや条件文、ファンクション（関数)やスレッドなどに進みましょう。音楽的のアイデアを実現させるをびっくりするほど強力なツールをになることでしょう。

それでは、やってみましょう。

5.1　ブロック

Sonic Pi でよく見る構造をブロック(block)といいます。ブロックでは、沢山のコードを一つの塊として、便利に扱うことが出来ます。

たとえば synth や sample では、その後ろのパラメータによって、変更した行の音を変えることができました。 しかし、時には数行分のコードに重要な変更をしたくなるでしょう。

たとえばループをする場合、5 回のうち 1 回だけリバーブ（残響音）を加えたい場合を考えてみます。

まず下のコードをみてください。

```
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
```

コードの塊を使って何かをしようとするときに、コードのブロックの始まりと終わりを Sonic Pi に伝える必要があります。その際には、do を始まりとして、end を終わりとして使用します。

```
do
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
end
```

しかし、これではまだ完璧ではないので、実行されません（動かしてみてもエラーメッセージが出るだけです）。実行したい始まりと終わりの命令の伝達が Sonic Pi へ完了していないからです。do の前に特別なコードをすこし書くことによって、このブロックを Sonic Pi に教えます。このチュートリアルの後半でこれらの特別なコードを紹介してきます。

ひとまず、みなさんが特別なコードを使ってブロックを動かしたい場合、do と end でコードのブロックをまとめることが重要であることを覚えておいてください。

5.2 イテレーション（反復）とループ

以前、play と sample のブロックを使って様々な音を作り出せること勉強しました。また、sleep を使うことによって、音の発生をコントロールする方法も学びました。

これらの基本的なブロックを使用することで多くの楽しさがあることを発見してもらえたのではないかと思います。しかし、音楽を構成するコードのパワーを使い始めることで、その楽しさの次元はまた新しい段階に向かうでしょう。次のいくつかのセクションでは、パワフルな新しいツールをいくつか探っていきます。はじめはまず「イテレーション（反復）とループ」を学びます。

リピート

何回か繰り返しを行うためにはどのようにコードを書いたらよいでしょう？例えばこのようなコードです。

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
```

これを 3 回繰り返したい場合、どうしたら良いでしょう？単純に考えればコピーして貼り付けを 3 回繰り返せば可能です。

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25

play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25

play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
```

ちょっと長過ぎますよね。もしサンプルの :elec_plip を変更させたい場合、どうしたら良いでしょうか？3 ヶ所全部の :elec_blup をひとつひとつ変えなくてはいけません。さらに重要なことですが、繰り返しが 50 回とか 1000 回になったとしたら、どうしたら良いでしょう？変更したいコードがすごくたくさんになってしまいます。

イテレーション（反復）

イテレーション（反復）とは、一定の処理を繰り返すことです。実際、コードの繰り返しは、これを 3 回繰り返すと口で言うことぐらい簡単にできます。さあ、はじめましょう。先ほどのコードブロックを思い出してください。3 回繰り返したいコードのブロックに「始まり」と「終わり」を指定してください。その際、特別なコードである 3.times を使いましょう。同じコードを 3 回繰り返して書く代わりに、3.times を書くことで、とても簡単に出来るようになります。その時にコードの最終行に end を書き入れることも忘れないようにしましょう。

```
3.times do
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
end
```

コピーと貼付けたりを繰り返すより、ずっと美しいコードになったと思いませんか？このようにブロックを使って沢山の繰り返しの構造を作ることが出来るのです。

```
4.times do
play 50
sleep 0.5
end

8.times do
```

```
play 55, release: 0.2
sleep 0.25
end

4.times do
play 50
sleep 0.5
end
```

イテレーション（反復）のネスティング（入れ子）

イテレーション（反復）の中に、さらなる繰り返しの機能を入れることによって面白いパターンを作ることが出来ます。例えば、

```
4.times do
sample :drum_heavy_kick
2.times do
 sample :elec_blip2, rate: 2
 sleep 0.25
end

sample :elec_snare
4.times do
 sample :drum_tom_mid_soft
 sleep 0.125
end
end
```

ループ（終わりのない繰り返し）

もしたくさんの繰り返しをしたい場合、1000.times do のようにすごく大きな数値を使うことになってしまいます。この場合は、おそらくアーメン・ブレイクのサンプルを無限に繰り返す（stop ボタンを押すまで）機能がほしいと思うでしょう。さあ、アーメン・ブレイクのサンプルを無限にループさせてみましょう。

```
loop do
sample :loop_amen
sleep sample_duration :loop_amen
end
```

loop について知っておかなくてはならない重要な点は、これはコードの中でブラックホールのように動いてしまう点です。一度 loop 機能が動いてしまうと、stop ボタンを押さない限り、永遠に再生されるということです。つまり、これは loop の後にあるコードは、いつまでたっても聞くことが出来ないということを意味しています。例えば、下の例で言うと、シンバルの音は loop の外にあるため、いつまでも再生されることはありません:

```
loop do
play 50
sleep 1
end

sample :drum_cymbal_open
```

さあ、イテレーション（反復）とループのコーディング方法を理解することができましたね!

5.3 条件文

やってみたいと思うかもしれないことのひとつに、ランダムに音を出すこと（前章参照）だけでなく、いくつかのコードをランダムに決め、それを実行させていくということも出てくると思います。例えば、ドラムとシンバルをランダムに鳴らしたい場合、if という言葉を使うとこれを実現できるようになります。

コイントス

それではコインをトス（投げる）してみましょう。もしもコインが表であればドラムを鳴らし、裏であればシンバルを鳴らします。簡単ですね。コイントスの機能は one_in という機能（ランダムネスのセクションで

紹介します）によって実行されます。2 つのうちの 1 つというように条件を細かく指定するときには one_in(2)と記述すると、ドラムを鳴らすコードかシンバルを鳴らす 2 つのコードのうち、どちらかが決定されるようになります。

```
loop do

if one_in(2)
 sample :drum_heavy_kick
else
 sample :drum_cymbal_closed
end
sleep 0.5
end
```

if は 3 つの役割を持っていることに注目しましょう。

- 条件付け
- はじめの選択によって実行されるコード（条件が正しかった場合）
- 二つ目の選択として実行されるコード（条件が間違っていた場合）

典型的なプログラム言語では、yes、はいという意味を true（正しい）で表現し、no、いいえは false（誤り）と表記します。そのため、 one_in に対する明確な回答として、true か false かを選択できる質問が必要となります。

はじめの選択では if と else の間に挟まれたコードが実行され、そして 2 番目の選択では else と end の間のコードが実行されるということに注目しましょう。これは複数行のコードを作る do/end のブロックにとても似ていますね。例えば:

```
loop do

if one_in(2)
 sample :drum_heavy_kick
 sleep 0.5
else
 sample :drum_cymbal_closed
```

```
 sleep 0.25
end

end
```

上記のように、sleep 0.5 と sleep 0.25 の、異なる休符時間を持つ場合、その時間は選択に応じることになります。

Simple if

時には、任意のコードを 1 行だけ実行したいときもあるでしょう。これも if とその後ろに条件を記述することで可能です。例えば:

```
use_synth :dsaw

loop do
play 50, amp: 0.3, release: 2
play 53, amp: 0.3, release: 2 if one_in(2)
play 57, amp: 0.3, release: 2 if one_in(3)
play 60, amp: 0.3, release: 2 if one_in(4)
sleep 1.5
end
```

上のコードでは、それぞれの音符が持つ再生の確率によって、異なるコードの和音を奏でるでしょう。

5.4 スレッド

それでは、強烈なベースラインとかっこいいビートを作った場合、どのようにしてそれらを同時に実行したらいいでしょう？一つの方法は、手動で同時に鳴らす事です—まず、いくつかのベースを演奏し、その後にドラム、またその後にベースというように…ですが、すぐに沢山の楽器を処理することが難しいことに気づくでしょう。

もし、Sonic Pi が自動的に合奏出来るとしたら？ thread(スレッド)と呼ばれる特別な命令によってそれが可能となります。

無限のループ

このシンプルな例で、強烈なベースラインとかっこいいビートを想像してみてください。

```
loop do
sample :drum_heavy_kick
sleep 1
end

loop do
use_synth :fm
play 40, release: 0.2
sleep 0.5
end
```

ループはプログラムのブラックホールのようだとすでにお話しました。一度ループを入力すると、stop ボタンを押さない限り、そこから抜け出せなくなります。では、どうしたら同時にふたつのループを演奏することが出来るでしょう？ 私たちは、同時にそれらのコードをスタートさせたいと Sonic Pi に伝えなくてはいけません。これがスレッドを使った解決方法なのです。

スレッドを使った解決方法

```
in_thread do
loop do
 sample :drum_heavy_kick
 sleep 1
end
end

loop do
use_synth :fm
play 40, release: 0.2
```

```
sleep 0.5
end
```

はじめのループの do/end ブロックを in_thread で囲むことで、次にくる do/end ブロックをぴったりと同時にループさせるよう Sonic Pi に命令することができます。

それではドラムとベースラインを同時に鳴らすことに挑戦してみましょう！

そして、初めにもう一つの音を追加したい場合、

```
in_thread do

loop do
 sample :drum_heavy_kick
 sleep 1
end
end

loop do
use_synth :fm
play 40, release: 0.2
sleep 0.5
end

loop do
use_synth :zawa
play 52, release: 2.5, phase: 2, amp: 0.5
sleep 2
end
```

前と同じ問題が出てきましたね。 in_thread によって最初の繰り返しと 2 つ目の繰り返しが同時に演奏されています。しかし 3 番目の繰り返しが演奏されません。ですので以下のように、もう一つのスレッドが必要となります。

```
in_thread do
loop do
 sample :drum_heavy_kick
 sleep 1
end
end

in_thread do
loop do
 use_synth :fm
 play 40, release: 0.2
 sleep 0.5
end
end

loop do
use_synth :zawa
play 52, release: 2.5, phase: 2, amp: 0.5
sleep 2
end
```

スレッドとして実行する

驚かせるかもしれませんが、Run ボタンを押すということは、実際にはコードを実行するための新しいスレッドを作っていることになります。そのため、複数回 Run ボタンを押すことは、互いの上に音を階層化することになります。Run それ自体がスレッドであるために、音を自動的に紡ぎ合わせることになるのです。

スコープ

Sonic Pi をマスターしようとするとき、スレッドが、音楽のために構築されたブロックの中で最も重要であることに気がつくでしょう。重要な役割りの一つとして、他のスレッドから現在のセッティングを分離する

ことがあります。どういうことかというと、例えば use_synth を使ってシンセの種類を変更する時、 現在のスレッド中にあるシンセだけを変更します。他のどのスレッドも変更しません。 そのことを確認してみましょう:

```
play 50
sleep 1

in_thread do
use_synth :tb303
play 50
end

sleep 1
play 50
```

真ん中の音だけがほかのものと違うことに注目してみましょう。
use_synth は含まれているスレッドだけに影響し、外のスレッドには影響しません。

インヘリタンス（継承機能）

in_thread を使って新しいスレッドを作ると、そのスレッドには現在のスレッドの全ての設定が自動的に継承されます。ではその機能を見てみましょう。

```
use_synth :tb303

play 50
sleep 1

in_thread do
play 55
end
```

2 番目の音符は、別のスレッドから再生されたにもかかわらず :tb303 シンセで演奏されることに注目してください。use_* 関数を使ったいかなる設定も、同様に作用するでしょう。

スレッドが作成されると、彼らの親から（前述の）すべての設定は継承しますが、スレッド以降の変更を共有することはありません。

スレッドの命名

最後に、スレッドに名前つける機能を覚えましょう。

```
in_thread(name: :bass) do
loop do
 use_synth :prophet
 play chord(:e2, :m7).choose, release: 0.6
 sleep 0.5
end
end

in_thread(name: :drums) do
loop do
 sample :elec_snare
 sleep 1
end
end
```

このコードが実行された時のログ画面を見てみましょう。レポートの中にスレッドの名前が表示されることがわかります。

```
[Run 36, Time 4.0, Thread :bass]
|- synth :prophet, {release: 0.6, note: 47}
```

1 つのスレッドには 1 つの名前

名前付きのスレッドについて知っておくべき最後のことは、名前の 付いた 1 つのスレッドだけが同時に実行されることです。では下記を見てみましょう。次のコードを考えてみてください。

```
in_thread do
loop do
 sample :loop_amen
 sleep sample_duration :loop_amen
end
end
```

ワークスペースに上記のスレッドを張り付けて、Run ボタンを押します。数回押してみましょう。複数のアーメン・ブレイクが不協和音として反復されるでしょう。もういいですよね。stop ボタンを押しましょう。

この動作はこれまで何度も見てきました。Run ボタンを押すと、一番上のレイヤーにあるサウンドが鳴ります。このためループが含まれている場合、Run ボタンを 3 回続けて押すと、3 つのレイヤーのループが一斉に実行されます。

ただし、名前付きのスレッドでそれは異なります。

```
in_thread(name: :amen) do
loop do
 sample :loop_amen
 sleep sample_duration :loop_amen
end
end
```

このコードで Run ボタン複数回、押してみてください。一つのアーメン・ブレイクのループのみが聞こえるでしょう。そして下記のテキストがログ画面に現れます。

```
==> Skipping thread creation: thread with name :amen already exists.
```

Sonic Pi は、:amen という名前があるスレッドが既に存在するため、別のスレッドを作成しませんと伝えています。この動作はすぐに必要でないように思えますが、ライブコーディングを始めると、非常に便利なものになるでしょう。

5.5 ファンクション（関数）

一度、膨大なコードを書き始めると、その構造をより簡単かつ整理された状態で把握出来るように、構築し、まとめたいと感じることになるでしょう。ファンクション（関数）はそんなことをとても効果的に成し遂げる方法です。この関数を使うとコードのまとまりに名前をつけることも可能になります。早速、見ていきましょう。

関数の定義

```
define :foo do
play 50
sleep 1
play 55
sleep 2
end
```

ここでは、foo と呼ばれている新しい関数を見ていきます。これは以前から出てきている do/end ブロックと、define という魔法の言葉とともに機能します。しかし、bar や baz や、main_section や lead_riff のようなある程度の意味を関数が呼び出せれば、foo を呼び出す必要はありませんでした。

使用する場合は、関数の名前に:(コロン)を付加することを忘れないでください。

関数の呼び出し

いったん一つの関数を定義すると、ただ名前を書くことでそれを呼びだすことができます:

```
define :foo do
play 50
sleep 1
play 55
sleep 0.5
end
```

```
foo

sleep 1

2.times do
foo
end
```

foo は、イテレーション(繰り返し)ブロック内 play や sample で書かれたコード内で、foo を使うことができます。 この関数は、自身の楽曲の中で、自分自身を表現し、新しい意味ある言葉を作り出すには非常に優れた方法です。

領域を超えた関数の利用

これまでのところ、Run ボタンを押すたびに、Sonic Pi は白紙の状態から実行されました。 これは一つのワークスペースのみが再生されていたからです。 これまでは、別のワークスペースまたは別のスレッド内のコードを参照することはできなかったからです。 しかし、関数はそれを変えることができます。関数を定義すると、 Sonic Pi はそれを覚えることができます。では、ちょっとやってみましょう。 まずワークスペースにあるすべてのコードを消して、foo と名づけたものに変更します。

```
foo
```

Run ボタンを押して、関数の foo が再生されることを確認してください。では、コードはどこにいったのでしょう？Sonic Pi は、なにを実行するにかどうに知ったのでしょうか？Sonic Pi は、ワークスペースから消された関数を覚えていたのです。 この動作は define もしくは defonce を使用した時に機能します。

変数化された関数

最小値と最大値の情報を渡すためには rrand を使用する、面白い方法をご紹介しましょう。

```
define :my_player do |n|
play n
end

my_player 80
sleep 0.5
my_player 90
```

あまり面白くないのですが、ポイントを説明します。play を my_player という関数として定義します。

この変数は、define で括られた do/end ブロックの do の後に記述する必要があります。変数は、垂直のバー |で囲み、複数の変数を扱う場合はカンマ ,で分割し、変数の名前はどんな言葉でもつけることが出来ます。

この魔法は、define を使い do/end ブロック内で行われます。また、実行されるための値のような変数名を使うことも出来ます。 この例で言うと、n という関数になります。コードが起動した際のひとつの約束ごととして、変数はその領域に記憶されている実際の数値に置き換えられます。あなたが関数を呼び出した際は、この数値を関数に置き換えて実行することが出来るのです。この my_player 80 というのは、音階 80 の音を鳴らすということです。関数の定義の中で、n はすぐに 80 に置き換えられます。 そして play n は play 80 となるのです。また次に my_player 90 という関数を呼び出す際には、 n はすぐさま 90 に置き換えられ、play n は play 90 として再生されます。

それではさらにおもしろい例を見てみましょう:

```
define :chord_player do |root, repeats|
repeats.times do
 play chord(root, :minor), release: 0.3
 sleep 0.5
end
end

chord_player :e3, 2
```

```
sleep 0.5
chord_player :a3, 3
chord_player :g3, 4
sleep 0.5
chord_player :e3, 3
```

ここでは repeats.times do という行の中で一つの数値のように repeats が使われます。 また、play を呼び出した際の音階の名前と同様に root が使われています。

関数によって、高度な表現と沢山の構造を簡単に読み込ませることが出来るということがわかりましたね！

5.6 変数値

コードを書いていく中で役に立つことは、名前を作成することです。Sonic Pi はこれをとても簡単に作ることができます。あなたが使用したいと思う名前に続けて、等号のイコール(=)を書き、そのあとに覚えておきたい命令を書きます。

```
sample_name = :loop_amen
```

ここで変数値 sample_name は、:loop_amen として記憶されます。sample_name を使うときはどこでも :loop_amen を使ったことになるのです。例:

```
sample_name = :loop_amen
sample sample_name
```

Sonic Pi では、変数を使う 3 つの理由があります。意味の伝達、繰り返しの操作、そして結果の獲得です。

意味の伝達

コードを書くとき、コンピュータが理解し実行させるため、どのようにその内容を伝えて動作させるのかを考えるだけであれば、それは簡単な事ですね。しかし、覚えておかなくてはいけない大事なことは、コンピュータがコードを読むということだけではないのです。他の人もそれを読み、何が起こっているか理解をしようとします。あなた自身も将来、自分の書いたコードを見返して、どんなことをしたのか？と理解しようとする時が来るかもしれません。たぶん確実にあなたにも他の人にも、そういったことが起こるのです!

あなたのコードがどのように動いているのか。他人が理解をするためにコメントを書く（前章で確認できます）という方法があります。もう一つの方法として、理解しやすい変数名を使うという方法があります。

```
sleep 1.7533
```

なぜ 1.7533 という数値を使ったのでしょう？その数値はどこから来たのか？それは何を意味しているのか？という疑問がわきます。しかし、次のコードを見てみましょう。

```
loop_amen_duration = 1.7533
sleep loop_amen_duration
```

こう書くとすぐに 1.7533 がサンプル音源 :loop_amen の長さであるということがわかりますね。もちろん、下記のように一行に書くことも可能です。

```
sleep sample_duration(:loop_amen)
```

どちらを用いたとしても、コードの意味がよりわかりやすいものになりました。

繰り返しの操作

コードの中では沢山の繰り返しが頻繁に見られます。また、もし何かを変更したいときは、膨大な場所を変更する必要も出てきます。次のコードをみて下さい。

```
sample :loop_amen
sleep sample_duration(:loop_amen)
sample :loop_amen, rate: 0.5
sleep sample_duration(:loop_amen, rate: 0.5)
sample :loop_amen
sleep sample_duration(:loop_amen)
```

:loop_amen によって沢山の事ができるのです！もし:loop_garzul のような他のサンプルの繰り返しによる音が聞きたい場合はどうしましょう？そうするにはすべての :loop_amen を探しだして :loop_garzul に変更する必要があります。そんな変更ができる沢山の時間があればいいんですが…もし仮に君がステージの上で演奏している最中だったらどうしよう？特にみんなのダンスを止めないために、変更するための優雅な時間はないかもしれません。

下記のようなコードを書いたとして、

```
sample_name = :loop_amen
sample sample_name
sleep sample_duration(sample_name)
sample sample_name, rate: 0.5
sleep sample_duration(sample_name, rate: 0.5)
sample sample_name
sleep sample_duration(sample_name)
```

そして、これは試しの前述のものと同様のものです。これも sample_name = :loop_amen を sample_name = :loop_garzul の一行の書き換えによって変更する機能を備えています。つまりこのように変数の魔法によって多くのポイントを変更させることができるのです。

結果の獲得

最後に、変数を使う優れた理由は、そのコードの結果の獲得をするということです。例えば、サンプル音源の長さを使って何かを行いたい時など。

```
sd = sample_duration(:loop_amen)
```

上記のように書くことで、今、:loop_amen というサンプルの長さが必要な時、どこにでも sd を使うことが出来るようになります。 おそらくもっと重要なのは、変数は、play や sample をキャプチャすることができることです。

```
s = play 50, release: 8
```

またこのように書くことで s が変数として記憶され、シンセをコントロールすることを許容するようになります。

```
s = play 50, release: 8
sleep 2
control s, note: 62
```

また、後の章ではもっと詳しくシンセをコントロールすることも学びます。

5.7 スレッドの同期

一度、たくさんの関数やスレッドを同時に使用するすごく高度なライブコーディングを行えるようになると、すごく簡単に、スレッド内でミスをしてしまうことに気がつくと思います。でもそれは全然大したことではないのです。Run ボタンを押すことで、簡単にスレッドを再スタートさせることができるのですから。しかし、スレッドを再スタートさせるときは、元々のスレッドとは調子がずれる事になります。

継続時間

すでに取り上げましたが、新しいスレッドが in_thread として作られると、それまでの親となるスレッドのセッティングが全て踏襲されます。これは現在の時間を含んでいます。つまり、スレッドはスタートするときには常に他のスレッドと同期していることを意味しています。

しかし、一つのスレッドを実行させた時は、それは独自の時間で再生されるので、実行中の他のスレッドと同期していることはありません。

Cue と Sync

Sonic Pi は、cue と sync という関数を使ってこの問題の解決方法を提供します。 cue は、他のすべてのスレッドに向けてビートのメッセージを送ることができます。初期設定では、他のスレッドは、これらのメッセージを関係付けず、無視します。しかし、sync 関数を使えば、あなたは簡単に関連付けをすることができるのです。

sync という機能は、一定の時間実行中のスレッド止める sleep 関数と非常に似ていることに気づくことが重要です。しかし、sleep では、どのくらい休止さえるかを決めることが出来ましたが、sync ではそれを決めることが出来ず、sync は、長さに関わらず次の cue が出てくるまで待つのです。

もうちょっと詳しくみていきましょう:

```
in_thread do
loop do
 cue :tick
 sleep 1
end
end

in_thread do
loop do
 sync :tick
 sample :drum_heavy_kick
end
end
```

ここではふたつのスレッドが使われています。ひとつはサウンドはなりませんが メトロノームのような機能として :tick を使って一秒ごとにビートの情報を送っています。2 つ目のスレッドは tick の情報と同期し、そのビートの情報を受け取った際に、cue のスレッドの時間に情報を引き継ぎ、起動を続けるのです。

この結果、:drum_heavy_kick のサンプル音源は、厳密に同時のスタートでないふたつのスレッドであったとしても、他のスレッドが :tick からの情報をきっちり送っているので、正確にビートを刻んだ音が聞けるはずです。

```
in_thread do
loop do
 cue :tick
 sleep 1
end
end

sleep(0.3)

in_thread do
loop do
 sync :tick
 sample :drum_heavy_kick
end
end
```

このちょっとやっかいな sleep(0.3)は一つ目のスレッドのフレーズから外れてしまう 2 つ目のスレッドを作り出しています。しかしながら、cue や sync を使えば、自動的にスレッドを同期させ、タイミングがずれてしまうようなアクシデントから回避させてくれます。

Cue の名前

cue には、:tick 以外の好きな名前を付けられます。その際には、その名前で、他のスレッドと確実に同期させる必要があります。もし違った場合、永遠に（もしくはストップボタンを押すまで）それは cue の情報を待ち続けることになります。

それでは cue の名前をつけて実行してみましょう。

```
in_thread do
loop do
```

```
 cue [:foo, :bar, :baz].choose
 sleep 0.5
end
end

in_thread do
loop do
 sync :foo
 sample :elec_beep
end
end

in_thread do
loop do
 sync :bar
 sample :elec_flip
end
end

in_thread do
loop do
 sync :baz
 sample :elec_blup
end
end
```

ここではメインの cue でビートメッセージをランダムに :foo と :bar、:baz に送るよう設定しています。それから 3 つのスレッドがそれぞれ独自に同期して、それぞれに設定されたサンプルを再生します。この関係付けられた結果から、cue のスレッドによってそれぞれが同期しながら 0.5 秒刻みに sync スレッドの音がランダムに再生され、そしてサンプルが再生されるのです。

逆に、sync スレッドを次の cue まで居座り続けさせるように、スレッドに設定しても、もちろんそれは動作します。

6 章　スタジオエフェクト

Sonic Pi には、作ったサウンドに簡単にスタジオ·エフェクトを追加できるという最もやりがいのある楽しい側面があります。たとえば、部分的にリバーブを追加したり、エコーやディストーション(歪み)、ワブルベース（ベース音にフィルターLFO を掛け、断続的な音にすること）を加えることができます。

Sonic Pi には、エフェクト（FX）を追加する非常に簡単で強力な方法をがあります。それも、あなたが作った音にディストーション(歪み)を通し、その後エコー、さらにリバーブの効果を連動させ、また、シンセやサンプルにパラメータを与えるのと同様の方法でエフェクトユニットのパラメータを個別に制御することができます。そして、実行されている間でも、パラメータを変更することが可能です。だから、例えば、トラックのいたるところでベースのリバーブを強くするということができるのです。

ギターエフェクター

もし、この話が少し複雑に聞こえる場合でも、心配は無用です。少し触れてみれば、すぐに理解することができることでしょう。まずは、ギターのエフェクターのようなものをイメージしてください。購入可能なギターエフェクターには多くの種類がありますが、リバーブ、ディストーションなど幾つかを数珠繋ぎに追加することができます。ギタリストは、自分のギターにエフェクターの一つ（ディストーションなど）を接続し、そして別のケーブルでリバーブエフェクターに繋ぎます。そしてリバーブエフェクターの出口はギターアンプに繋ぐことができるわけです。

ギター -> ディストーション -> リバーブ -> アンプ

これをエフェクトチェーンと呼びます。Sonic Pi は正にこれをサポートしています。さらに、ディストーション、リバーブ、エコーなどのエフェクターの多くは、どのくらい効果を加えるのか、制御できるようなダイヤルやスライダを持っていて、Sonic Pi もこの種の制御をサポートしているということです。ついに、あなたはギタリストがエフェクターを

使いながらギターを演奏する姿を想像することができるでしょう。ただ、Sonic Pi では、それをコントロールするために他の人を必要としません。コンピュータが代役を務めているのです。

さあ、エフェクトを探究していきましょう！

6.1 エフェクトの追加

このセクションでは、2 つのエフェクト「リバーブとエコー」を見ていきます。これらをどのように制御するのか、チェーン接続するかを見ていきます。

Sonic Pi のエフェクトシステムは、ブロックを使用します。

まだセクション 5.1 を読んでいなければ、一度戻って目を通してください。

リバーブ

リバーブを使用する場合、この様に特殊なコード with_fx :reverb を下記のようにブロックに書きます。

```
with_fx :reverb do
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
end
```

早速コードを実行してリバーブを聞いてみましょう。いいでしょ！サウンド全体が残響効果で素晴らしくなります。

さあ、次はブロックの外にコードを書いて何が起こるか見てみましょう。

```
with_fx :reverb do
play 50
```

```
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
end

sleep 1
play 55
```

どうして最後の play 55 にリバーブが適用されないのでしょうか。 これははは do/end ブロックの外に書かれているため、リバーブエフェクトが適応されないのです。 do/end ブロックの前で音を生成する場合も同様にリバーブは適用されません。

```
play 55
sleep 1

with_fx :reverb do
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
end

sleep 1
play 55
```

エコー

選択できるエフェクト（FX）は他にもたくさんあります。エコーはどうでしょうか?

```
with_fx :echo do
play 50
sleep 0.5
```

```
sample :elec_plip
sleep 0.5
play 62
end
```

Sonic Pi のエフェクトブロックの強力な側面の一つは、既に play と sample で見てきたように、パラメータと同様の数値が渡されることです。たとえば、楽しいエコーのパラメータは、phase:で表され、エコーの長さを表しています。

下記のコードで音の響きをゆっくりにしてみましょう。

```
with_fx :echo, phase: 0.5 do
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
end
```

エコーの遅れを早くしてみましょう。

```
with_fx :echo, phase: 0.125 do
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
end
```

エコーが８秒間の時間でフェードアウトする様に、decay: を設定してみましょう。

```
with_fx :echo, phase: 0.5, decay: 8 do
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
```

```
play 62
end
```

エフェクトの入れ子

ブロック効果の中で最も強力な側面の一つは、入れ子ができるということです。 これはとても簡単にエフェクト同士を連結することができます。たとえば、あなたはどのようにエコーとリバーブの両方をコードに適応させますか？ その答えは簡単です。一つの効果の内側にもう一つを配置するだけです。

```
with_fx :reverb do
with_fx :echo, phase: 0.5, decay: 8 do
 play 50
 sleep 0.5
 sample :elec_blup
 sleep 0.5
 play 62
end
end
```

オーディオの流れについて内側から追ってみましょう。play 50 を含む do/end block のコード全ては、最初にエコーエフェクトに送られ、その音がリバーブエフェクトへと順に送られていきます。

クレイジーなサウンドを得るために非常に深く入れ子を使用することができます。しかし、一度にたくさんのエフェクトを入れ子で実行すると多くのリソースを必要とするため、注意してください。そのため、特にラズベリーパイのような低電力供給のプラットフォームではエフェクトの使用を節約することも必要です。

エフェクターの発見

Sonic Pi は、あなたが演奏するためにたくさんのエフェクト（FX）を搭載しています。どういった効果が使えるのかを見つけるには、ヘルプシ

ステム画面の下にある FX ボタンをクリックし、利用可能なリストを見てみてください。ここに私のお気に入りを記載しておきます。

- wobble,
- reverb,
- echo,
- distortion,
- slicer

さあ、エフェクトを追加して驚きの新しいサウンドの虜となってください！

6.2 エフェクトの実践

外観は一見シンプルですが、 エフェクトの中身は、実際には非常に複雑なモンスターです。そのシンプルさは、しばしば、この機能を使いこなそうとする人を誘惑します。もしもあなたが強力なマシンを持っている場合は問題は無いかもしれませんが、私のようにラズベリーパイで動作させる場合、確実にビートを刻み続けるようにしたいのであれば、どのくらいの負荷をかけているのか注意する必要があります。

このコードを考えてみましょう。

```
loop do
with_fx :reverb do
 play 60, release: 0.1
 sleep 0.125
end
end
```

このコードは非常に短いリリースタイムで音符 60 を演奏する短いコードです。そして、リバーブを加えたいために、コードをリバーブで囲んでいます。問題なさそうにはみえますが…。

コードが何をするかを見ていきましょう。まず、loop は、内側のすべてが永遠に繰り返すことを意味します。次に、with_fx ブロックがあります。これはループが繰り返される度に、新しいリバーブエフェクトを作成することを意味します。ギターの弦を弾くたびに別々のリバーブエフ

ェクターを用意して演奏しているようなものです。こんなことができたらかっこいいかもしれませんが、それはいつもあなたが望むものとは限らないでしょう。たとえば、次のコードは、ラズベリーパイで実行させるためにはとても労力を要します。リバーブは、with_fx によって制御され、停止か削除されるまで、生成され続け、スムーズな演奏に必要とする大切な CPU パワーを奪うことになりかねません。

ギタリストのように、すべての音が 1 つのリバーブエフェクターを通る従来の方法と同様のことをするには、どうしたら良いでしょうか？

以下、サンプルです。

```
with_fx :reverb do
loop do
 play 60, release: 0.1
 sleep 0.125
end
end
```

with_fx ブロックの内部に loop を配置します。この方法では、ループ内すべての音符が再生にリバーブは一回だけ生成されます。このコードは効率的で、ラズベリーパイでも正常に動作します。

loop 内の繰り返しの上位に with_fx を使うことで折り合いを付けます。

```
loop do
with_fx :reverb do
 16.times do
   play 60, release: 0.1
   sleep 0.125
 end
end
end
```

loop の内側の内容を with_fx に引き出すことで、16 の音程ごとに新しいリバーブを作成しています。

だから、間違いというではなく、全ては可能性であることを覚えておいてください。これらの異なるアプローチからは、異なるサウンド、また

異なる特徴をもたらします。あなたのプラットフォームに応じた制約の中で、最も適切に機能するアプローチを使いわけながら演奏を心がけましょう。

7 章 サウンドのコントロール

これまでの章では、どのようにシンセやサンプルを扱い、アンプ（増幅）、パン、エンベロープなどのパラメータを変更するのかを見てきました。個々のサウンドには、元来、音の継続時間を設定するパラメータが備わっています。

もしも演奏中にギターの弦を歪めビブラートさせるように、パラメータを変更できたなら、それってクールではないでしょうか？

あなたは幸運です-このセクションでは、まさしくそれをどのように行うのかを紹介します。

7.1 演奏中のシンセ制御

これまでは、新しいサウンドとエフェクトを引き起こす方法だけ見てきましたが、Sonic Pi は、演奏中の音を操り、コントロールする機能を備えています。シンセを扱うためには、変数を用います。

```
s = play 60, release: 5
```

これはローカル変数 s が音符 60 の演奏を処理することを表しています。このローカル変数は、例えば関数といった他の機能からアクセスすることはできません。s を 一度、用意すれば control 関数を介して制御することができます。

```
s = play 60, release: 5
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
```

```
control s, note: 72
```

演奏している間に、1 つのシンセのみを呼び出し、3 回ピッチを変更します。ここで注目すべき点は、4 つの異なるシンセを呼び出していないということです。

標準的なパラメータ（変数）は、control へ渡すことができます。そして、amp: 、cutoff: あるいは pan: のようなパラメータを制御することができます。

制御不可能なパラメータ

一度シンセが再生されると、一部のパラメータは制御することができなくなります。ADSR エンベロープ・パラメータがこれに該当します。どのパラメータが制御可能かどうかは、ヘルプシステムのコメントを参照してください。「設定を変更することは出来ません」というコメントが記載されている場合、シンセを開始した後、パラメータを制御することはできません。

7.2 エフェクト(FX)の制御

エフェクト（FX）も、少々異なる方法ですが制御する事が出来ます。

```
with_fx :reverb do |r|
play 50
sleep 0.5
control r, mix: 0.7
play 55
sleep 1
control r, mix: 0.9
sleep 1
play 62
end
```

変数を使用する代わりに、do/end ブロックのゴールポストパラメータを使用します。do/end ブロックに対して、”|”と”|”の間にエフェクトを

実行するための特別な名前を指定する必要があります。この動作は、パラメーター化された関数を使用する場合と同じです。

さあ、シンセやエフェクトをコントロールしてみよう！

7.3 パラメータのスライド

シンセやエフェクトの引数を探求しながら、_slide で終わるパラメータが多くあることに気づいたかもしれません。それらを呼び出しても、何の効果を示さなかった可能性があります。これは通常のパラメータではなく、前章で紹介したように、シンセを制御するときのみ動作をする特別なパラメータです。

次の例を考えてみましょう。

```
s = play 60, release: 5
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

ここでは、各 control の呼び出し後、すぐにシンセのピッチの変更を聞くことができますが、変化する間にピッチがスライドさせたくなるかもしれません。その場合、スライドを追加するために note: パラメータを制御するように、シンセの note_slide パラメータを追加する必要があります。

```
s = play 60, release: 5, note_slide: 1
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

control の呼び出しの間の音程が滑らかに繋がっていることがわかるはずです。いい感じではないでしょうか。note_slide: 0.2 よりも短いスライド時間を使うことで、スライドをスピードアップすることができますし、もっと長い時間をつかってテンポを遅くすることが出来ます。

ネバネバするスライド

一度、実行しているシンセの _slidee パラメータを設定したら、それは記憶され、対応するパラメータがスライドする度に使用されます。 スライドを停止するためには、次の control を呼び出す前に 0 に_slide 値を設定する必要があります。

エフェクト・パラメータのスライド

また、エフェクト(FX)パラメータをスライドさせることも可能です。

```
with_fx :wobble, phase: 1, phase_slide: 5 do |e|
use_synth :dsaw
play 50, release: 5
control e, phase: 0.025
end
```

さあ、操作に従って、滑らかな変調を楽しみましょう。

8 章 データ構造

プログラマーのツールキットで非常に便利なツールは、データ構造です。

時には、複数の要素を使用して表現したい場合があるでしょう。たとえば、次々に演奏される連続した音符が便利だと気付くこともあるでしょう。プログラミング言語は、こういったことを正確に行うためのデータ構造を持っています。

プログラマが利用できるデータ構造はとてもエキサイティングで魅力的なもので、人々は常に新しい発明をしています。しかし、今の私たちは非常に単純なリストとしてのデータ構造に考慮する必要があります。

より詳細にそれを見てみましょう。まずは、その基本的な形式を学習し、続いて、リストはどのようにスケール(音階)や和音を表現するために使用されるのかをみていきましょう。

8.1 リスト

このセクションでは、非常に有効なデータ構造であるリストについてみていきます。すでに、その大まかな方法には、音符をリストからランダムに選ぶことを学んだランダムの章で、触れました。

```
play choose([50, 55, 62])
```

ここでは、和音やスケール(音階)を表現するためにリストの使い方について見ていきます。最初に、和音を演奏する方法をおさらいしてみましょう。sleep を使用しない場合、すべての音が同時に演奏されることを思い出してください。

```
play 52
play 55
play 59
```

このコードを表現する別の方法を見てみましょう。

リストの実行

1 つの方法として、[52, 55, 59]のように、すべての音符をリストにすることです。使いやすい play 機能は、リストになった音符をどのように演奏するかをすぐに理解します。試してみてください。

```
play [52, 55, 59]
```

おー、これは読みやすいですね。音符リストの演奏は、通常のパラメータの使用を邪魔しません。

```
play [52, 55, 59], amp: 0.3
```

もちろん、MIDI の数値の代わりに、元来の音符の名前を使うこともできます。

```
play [:E3, :G3, :B3]
```

多少の音楽理論の知識を持っている方なら、幸運なことに 3 オクターブで E マイナーのコードが演奏されたことがわかるでしょう。

リストへのアクセス

もう一つの非常に便利なリストの機能として、リストから情報を取得する機能です。これは少し奇妙に聞こえるかもしれませんが、本の 23 ページを開いてくださいと誰かに頼まれることよりも簡単なことです。リストの場合は、インデックス 23 の要素は何ですか？と尋ねればいいのです。唯一、奇妙なことは、プログラミングのインデックスは通常、1 ではなく 0 から開始されることです。

リストのインデックスは 1,2,3 と数える代わりに、0,1,2 と数えていきます。

それではもう少し詳細にみてみましょう。

```
[52, 55, 59]
```

これは特に難しいことは何もありません。リストの 2 番目の要素は何でしょうか？そう、もちろん 55 です。簡単ですね。それでは、同様にコンピュータが答えることができるかどうかを見てみましょう。

```
puts [52, 55, 59][1]
```

以前にこのようなものを見たことがない場合、少し奇妙に見えるかもしれませんが、大丈夫。私を信頼してください。難しいことはありません。上記には、puts という命令、リストの 52, 55, 59 、インデックスが[1] の 3 つの部分があります。まずはじめに、Sonic Pi へログの中から答えを出力させるため puts を命令します。次にリストを与えています。そして最後に第 2 の要素を問い合わせています。インデックスは角括弧で囲

む必要があり、カウントは 0 で始まるので、2 番目の要素のインデックスは 1 となります。下記、見てみましょう。

```
# indexes:  0   1   2
        [52, 55, 59]
```

コード puts [52, 55, 59][1] を実行してみてください。ログに 55 とポップアップされるでしょう。インデックス 1 を別のインデックスに変え、さらに長い list を作って、次のコードでどのように使うのかを考えてみてください。番号の連続が、音楽の構造として表現されるのではないでしょうか。

8.2 和音

Sonic Pi は、和音名のリストを返す機能を内蔵しています。実際に試してみましょう。

```
play chord(:E3, :minor)
```

さあ、本当に動きましたね。そのままのリストよりも美しく見えますね。そして他の人にとっても読みやすいでしょう。では、どんな和音を、Sonic Pi はサポートしているのでしょうか。沢山あります。下記のようないくつかのコードを試してみましょう。

- chord(:E3, :m7)
- chord(:E3, :minor)
- chord(:E3, :dim7)
- chord(:E3, :dom7)

アルペジオ

play_pattern 関数で、簡単に和音からアルペジオ(和音を構成する音を一音ずつ順番に弾いていく奏法)に変更して演奏を行うことができます。

```
play_pattern chord(:E3, :m7)
```

でも、あまり楽しくないかもしれませんね。非常にゆっくり再生されているので。play_pattern は、リスト内の各音符を再生する度に sleep 1 を呼び出して再生します。play_pattern_timed 関数を使用することで、独自のタイミングと速度を指定することがきます。

```
play_pattern_timed chord(:E3, :m7), 0.25
```

時間のリストは、時間周期として取り扱うことができるでしょう。

```
play_pattern_timed chord(:E3, :m13), [0.25, 0.5]
```

下記と同等です。

```
play 52
sleep 0.25
play 55
sleep 0.5
play 59
sleep 0.25
play 62
sleep 0.5
play 66
sleep 0.25
play 69
sleep 0.5
play 73
```

どちらの書き方を好みますか？

8.3 スケール(音階)

Sonic Pi では、広いの音階を演奏出来ます。C3 のメジャースケールの再生はどのように行うのでしょうか？

```
play_pattern_timed scale(:c3, :major), 0.125, release: 0.1
```

さらに多くのオクターブを実行することができます。

```
play_pattern_timed scale(:c3, :major, num_octaves: 3), 0.125, release: 0.1
```

ペンタトニックスケール(オクターブに 5 つの音が含まれる音階のこと)のすべての音符ではどうでしょう?

```
play_pattern_timed scale(:c3, :major_pentatonic, num_octaves: 3), 0.125, release: 0.1
```

ランダム音階

和音とスケールは、ランダムな選択を強いるには素晴らしい制約です。コード E3 マイナーからランダムに音符を再生するには、この例を実行してみてください。

```
use_synth :tb303
loop do
play choose(chord(:E3, :minor)), release: 0.3, cutoff: rrand(60, 120)
sleep 0.25
end
```

異なる和音名やカットオフの範囲を試してみましょう。

コードとスケールの発見

Sonic Pi によってサポートされているスケールや和音を検索するには、チュートリアルの最も左にある Lang ボタンをクリックし、API リストのスケールまたは和音を選びます。メインパネルの情報に、スケールやコードの長いリストが現れるまで、下にスクロールしてください（あなたがどれを見るかに依拠します）。

間違いということはなく、それは可能性だということを思い出して、楽しんでください。

8.4 リング

標準のリストの中で面白い効果にリングがあります。いくつかのプログラミングを理解していれば、リングバッファやリングアレイを見たことがあるかもしれません。ここでは、まさに ring をみていきます。ーそれは短く、簡単です。

リストに関する前章では、インデックスを使用してリストから要素を取り出す方法を説明しました。

```
puts [52, 55, 59][1]
```

もしインデックス 100 を取り出そうとしたら、何が起こるでしょう？まあ、3 つしか要素がないリストで、インデックス 100 の要素はありません。だから Sonic Pi は空を意味する nil を返すでしょう。

ここでは、現在のビート（拍子）が継続的に増加するような counter を考えてみましょう。まずはカウンタとリストを作成します。

```
counter = 0
notes = [52, 55, 59]
```

これで、リストの音符にアクセスするためのカウンタを使用することができます。

```
puts notes[counter]
```

素晴らしい、52 を取得しました。では、カウンタを増やして別の音符を取得してみましょう。

```
counter = (inc counter)
puts notes[counter]
```

すごいでしょう、 55 を取得し、それを再び繰り返すならば 59 を得ます。さらに繰り返す場合は、リスト内の数が不足し nil になるでしょう。では、ちょうどループのはじめに戻り、再びリストを先頭から再生したい場合はどうしたらよいでしょうか？そのためにリングを使います。

リングの作成

2 つの方法の内、どちらの方法でリングを作成することができます。どちらもパラメータとしての環(円環構造を持つリスト)要素である ring 関数を利用します。

```
(ring 52, 55, 59)
```

また、正常のリストに、.ring メッセージを送り、リングに変換することができます。

```
[52, 55, 59].ring
```

リングのインデックス化

いったんリングを取得したら、インデックスがマイナスもしくはリングの数値より大きい場合を除いて、通常のリストを使用する場合とまったく同じ方法で使用出来ます。そして、リングの 1 つ要素として括弧でくくられて表示されます。

```
(ring 52, 55, 59)[0] #=> 52

(ring 52, 55, 59)[1] #=> 55
(ring 52, 55, 59)[2] #=> 59
(ring 52, 55, 59)[3] #=> 52
(ring 52, 55, 59)[-1] #=> 59
```

リングを使用する

現在のビート（拍子）の値を表すために変数を使用しているとしましょう。その変数は、現在表されているビート値に関わらず、インデックスとして音符を演奏するためや、リリースタイム（放出時間）、リングに格納されている有用な値として使うことができます。

スケールとコードはリング

知っておくと役立つこととして、scale（音階）と chord (和音)によって返されたリストもリングであり、任意のインデックスでそれらにアクセスすることを可能にします。

リングのコンストラクタ(構成子)

加えて、リングを構成する多くの機能があります。

- range は始点、終点とステップサイズを指定します。
- bools は簡単に 1 と 0 を使用するためのブール値を扱うことができます。
- knit は一連の繰り返される値のために構成することを可能にします。

詳細については、それら個々のドキュメンテーションを見てください。

9 章　ライブコーディング

Sonic Pi の中で一番エキサイティングな特徴の 1 つが、ライブでギターをかき鳴らすのと同じように、ライブで音楽を演奏しながら、コードを書き替え、音を変化させることが出来ることです。

この方法で優れているのは、演奏中たくさんの反応をもらうことが出来ることです。シンプルなループを走らせ、そのサウンドが完璧になるまで調整つつ。

ですが、一番の利点は、Sonic Pi でステージ上でライブ出来ることです。このセクションでは、コードを変化させながら力強いパフォーマンスに繋げることができる Live Coding の基本を学んでいきます。 しっかりついてきてくださいね。

9.1 ライブコーディング

これまで、実際に楽みながら演奏する方法を十分に学んできました。この章では、 これまでのすべての章から、どの様に作曲を開始し、ライブパフォーマンスに繋げるかを紹介していきます。そのために 3 つの要素が必要になります。

- チェック要素 1：音を作るコードを書く能力
- チェック要素 2：ファンクション(関数)を作成する能力
- チェック要素 3：(名前付き) スレッドを使う能力

よーし、始めましょう。最初の音をライブコーディングしましょう。 まず演奏したいコードに含まれた関数が必要です。簡単なところから始めましょう。スレッドでその関数を呼び出すループもほしいところです。

```
define :my_loop do
play 50
sleep 1
end

in_thread(name: :looper) do
loop do
 my_loop
end
end
```

もしそれが少し複雑に見える場合は、ファンクション（関数）とスレッドのセクションに戻って復習してください。既にこれらが頭に焼き付けていれならば、複雑ではないでしょう。

ここで定義されているのは、単に play 50 を演奏し、ビートのために sleep を 1 秒実行するファンクション(関数)です。そして、 my_loop を呼び出し、繰り返しを実行する :looper という名前のスレッドを定義します。このコードを実行すると、音符 50 を何度も何度も繰り返します。

チェンジアアップ（変化）させる

これから楽しみが始まります。コードを実行しながら、50 を別の数値 55 に変更し、もう一度 Run ボタンを押してみましょう。すごい迫力！変化しましたね！ライブ！

スレッドそれぞれに別の名前を使うことによって、新しいレイヤは追加されせんでした。 そのうえ、そのファンクション（関数）が再定義されることより、音が変化しました。:my_loop に新しい定義を与えると、 :looper スレッドが新しい定義付けを繰り返します。

スリープ時間や音符を変更し、再び試してみてください。どのように use_synth を追加するにはどうすればよいのでしょうか。その場合は、次のように変更します。

```
define :my_loop do
use_synth :tb303
play 50, release: 0.3
sleep 0.25
end
```

かなり面白くなってきましたが、それをさらに盛り上げることができます。何度も何度も同じ音を再生する代わりに、和音を弾いてみましょう。

```
define :my_loop do
use_synth :tb303
play chord(:e3, :minor), release: 0.3
sleep 0.5
end
```

和音からランダムな音階を演奏するにはどうすればいいでしょう。

```
define :my_loop do
use_synth :tb303
play choose(chord(:e3, :minor)), release: 0.3
sleep 0.25
end
```

またはランダムなカットオフ値を使用してみましょう。

```
define :my_loop do
use_synth :tb303
play choose(chord(:e3, :minor)), release: 0.2, cutoff: rrand(60,
130)
sleep 0.25
end
```

最後に、ドラムを追加してみましょう。

```
define :my_loop do
use_synth :tb303
sample :drum_bass_hard, rate: rrand(0.5, 2)
play choose(chord(:e3, :minor)), release: 0.2, cutoff: rrand(60,
130)
sleep 0.25
end
```

どんどんおもしろくなってきましたね!

しかしながら、ファンクション（関数）とスレッドを使ったライブコーディングにステップアップする前に、一息ついて、Sonic Pi で永遠にコードを変える live_loop について、次の章を読んでいきましょう。

9.2 ライブループ

このチュートリアルの中でもこの章は、最も重要です。 もしたった 1 つの章だけを読むのであれば、この章でしょう。前のセクションでライブコーディングの基礎を読んでいるのであれば、 live_loop は、演奏するための簡単な方法で、あまり多くを記述することはありません。

前のセクションを読んでいない場合、live_loop は Sonic Pi でジャム(即興演奏)するための最良の方法でしょう。

では演奏してみます。まず、新しいワークスペースに次のように書いてください。

```
live_loop :foo do
play 60
sleep 1
end
```

Run ボタンを押してください。基本的なビープ音が毎秒鳴ります。 これでは楽しくないのですが、まだ Stop は押さないでください。 もう一度 60 を 65 へ変更し実行してください。

わー! 調子を崩さずに自動的に変化しました。 これがライブコーディングです。

いっそうベースのように変えてみましょう？演奏したままコードを更新しましょう。

```
live_loop :foo do
use_synth :prophet
play :e1, release: 8
sleep 8
end
```

ここで、Run ボタンを押してください。
カットオフを動かしてみましょう。

```
live_loop :foo do
use_synth :prophet
play :e1, release: 8, cutoff: rrand(70, 130)
sleep 8
end
```

Run ボタンをもう一度押しましょう
いくつかのドラムを追加してみましょう。

```
live_loop :foo do
sample :loop_garzul
use_synth :prophet
play :e1, release: 8, cutoff: rrand(70, 130)
sleep 8
end
```

e1 から c1 に音符を変更してみましょう。

```
live_loop :foo do
sample :loop_garzul
use_synth :prophet
play :c1, release: 8, cutoff: rrand(70, 130)
sleep 8
end
```

このあとは私の言うことを聞くのを止めて、自由に演奏してみましょう! 楽しんでください!

9.3 マルチ・ライブループ

次のライブループを考えていきましょう。

```
live_loop :foo do
play 50
sleep 1
end
```

なぜ': foo'という名前をつけるのか疑問をもつかもしれません。この名前は、このライブループが他のすべてのライブループと異なっていることを示すために重要です。

同じ名前で実行中の 2 つのライブループが存在することはできません。

これは、複数同時にライブループを実行したい場合、それぞれに異なる名前を付ける必要があることを意味します。

```
live_loop :foo do
use_synth :prophet
play :c1, release: 8, cutoff: rrand(70, 130)
sleep 8
end

live_loop :bar do
sample :bd_haus
```

```
sleep 0.5
end
```

各ライブループを個別に変更して更新できます。そしてすべてちゃんと動作します。

ライブループの同期

既に気づいているかもしれませんが、ライブループは、以前にみてきたスレッドの cue のメカニズムを使用して自動的に動作します。ライブループがループするたびに、それが新しい名前を持つライブループの cue event を生成します。 従って、cue をきっかけに何も停止せずにサウンドをループに同期させることができます。

この同期のとれたコードを考えてみましょう。

```
live_loop :foo do
play :e4, release: 0.5
sleep 0.4
end

live_loop :bar do
sample :bd_haus
sleep 1
end
```

それを停止することなくタイミングを修正して同期することができるかを見てみましょう。まずは、foo ループ内の sleep の要素 1 を 0.5 に変えてみましょう。

```
live_loop :foo do

play :e4, release: 0.5
sleep 0.5
end

live_loop :bar do
sample :bd_haus
```

```
sleep 1
end
```

まだ終了することはできません。調子がまったく合っていないことに気付くでしょう。これはループがずれているからです。同期するように、修理していきましょう。

```
live_loop :foo do
play :e4, release: 0.5
sleep 0.5
end

live_loop :bar do
sync :foo
sample :bd_haus
sleep 1
end
```

うわー、すべてを停止することなくすべてが完璧に同期しました。いまから、ライブループを使用したライブコーディングを始めましょう！

10 章 不可欠な知識

このセクションではあなたのソニックパイの経験を最大限に活用するために、非常に便利で、実際のところ不可欠な知識を紹介します。

用意した多くのキーボードのショートカットを活用する利点と、作品を共有する方法、そして Sonic Pi でパフォーマンスするためのいくつかのヒントを見ていきます。

10.1 ショートカットの使用

Sonic Pi はコーディング環境としての楽器です。したがって、あなたが観客の前でライブ演奏している場合は特に、 ショートカットを使うことで、はるかに効率的で自然に Sonic Pi の演奏を行うことができます。

Sonic Pi の多くの機能は、キーボードを介して制御することができます。Sonic Pi の演奏や作業により慣れるために、ショートカットをもっと使いこないましょう。

個人的にはブラインドタッチの学習検討をお勧めします。マウスを使う必要があるときに、動作が遅れ、僕はいつもイライラしてしまいます。だから、普段からこれらのショートカットのすべてを使っています！

ショートカットを学べば、効果的にキーボードを使用でき、あなたは、間もなく、プロのようなライブコーディングができるようになるでしょう。

しかし、一度にすべてを憶えようとしないで、まずはあなたが最も使うものを試しながら、実践の中で、さらに追加しながら憶えていくとよいでしょう。

プラットフォームを超えた一貫性

クラリネットを習っているところを想像してみてください。すべてのクラリネットは、指使いや操作の全てが一貫していることを前提として作られていますが、もしそうでなかったら、別のクラリネットに乗り換えるには大変な時間を費やしてしまうだろうし、一回だけの制作のためにそれを使用することについて戸惑ってしまうでしょう。

残念なことに、3 つの主要なオペレーティングシステム（Linux、Mac OS X および Windows の場合）では、カット＆ペーストなどの操作のための独自の基準が用意されています。Sonic Pi は、これらの基準に準拠します。しかしながらプラットフォームの基準に対応することよりも、優先事項として、Sonic Pi 内はプラットフォームとして一貫した配置がなされています。これは Raspberry Pi で演奏したり、ショートカットを学ぶ時、Mac や PC に乗り換えても、違和感無く同じように操作できることを意味しています。

Control と Meta

一貫性への考え方のひとつにショートカットの命名があります。Sonic Pi において、主要な 2 つの組み合わせキーを当てはめるために、Control

キーと Meta キーを用います。すべてのプラットフォームでは、Control キーは同じです。しかし、Linux と Windows では、実際の Meta キーは Alt キーで、Mac の Meta メタキーはコマンド⌘キーです。一貫性のために、Meta を使いますが、あなたのオペレーティングシステムにおける適切なキー配置であることを憶えておいてください。

短縮

シンプルで読みやすくするため、Control キー＋その他のキーは「C-」、Meta キー＋その他のキーは「M-」の略語を使います。例えば、もし Meta キーと' r 'を同時に押さえるショートカットの場合、「M-r」と表記します。 以下は、有用ないくつかのショートカットです。

停止と開始

マウスを使う代わりに、M-r でコードを実行することができます。同様に、M-s で停止することができます。

ナビゲーション

ナビゲーションのショートカットを使わないの操作は実にもどかしいです。そのため、ショートカットを学ぶことに時間を費やすことを強くお勧めします。マウスやキーボードの矢印キーを手で移動させるより、ブラインドタッチを学ぶことで、これらのショートカットをさらに有効に活用することができます。

文章の先頭に移動するには C-a、文章の末尾に移動するには C-e、１行上は C-p、１行下がるには C-n、一文字進むには C-f、そして、一文字戻るには C-b。C-k でカーソルからラインの末尾まで全ての文字を消去する事ができます。

コードの整形

コードを自動整形するには M-m を押します。

ヘルプシステム

ヘルプシステムに切り替えるには、M-i を押します。しかし、何かを見つける場合、もっとはるかに便利なショートカットは、カーソル下の単語検索し、ドキュメントを表示する C-i です。簡単でしょう！

完全なリストについては、次の 10.2 ショートカット一覧表を見てみましょう。

10.2 ショートカット一覧表

以下は、Sonic Pi で利用可能な主なショートカットをまとめたものです。動機と背景については、セクション 10.1 を参照してください。

規定

このリストでは下記の規定に準拠します。 (Windows と Linux の Meta キー は Alt キー 、そして Mac は Cmd キー):

- C-a は Control キーを押さえながら a キー、両方を同時にを押した後に離す事を意味しています。
- M-r は Meta キーを押さえながら r キー、両方を同時にした後に離す事を意味しています。
- M-Z は Meta キーを押さえながら Shift キー、そして最後に z キー全てを同時に押した後に離す事を意味しています。
- C-M-f は Control キーを押さえながら Meta キー、そして最後に f キー全てを同時に押した後に離す事を意味しています。

主なアプリケーションの操作

- M-r - コードを実行

- M-s - コードを停止
- M-i - ヘルプシステムを表示
- M-p - 設定を表示

選択／コピー／張り付け

- M-a - 全てを選択
- M-c - 選択個箇所をコピー
- M-x - 選択個箇所をカット
- M-v - エディターへ張り付け

文字の操作

- M-+ - 文字サイズを大きく
- M-- - 文字サイズを小さく
- M-m - テキストを整形
- C-g - エスケープ

ナビゲーション

- C-a - 行の冒頭に移動
- C-e - 行の末尾に移動
- C-p -1 行前に移動
- C-n - 次行に移動
- C-f - 一文字進む
- C-b - 一文字戻る
- C-M-f - 一単語進む
- C-M-b - 一単語戻る
- C-h - 前の文字を削除
- C-d - 次の文字を削除
- C-l - 真ん中へ移動

削除

- C-h - 前の文字を削除
- C-d - 次の文字を削除

エディタの機能

- C-i - カーソル以下の文字を表示
- M-z - アンドゥ（行った操作を取り消し、元の状態に戻る）
- M-Z - リドゥ（一旦取り消した操作をやり直す）
- C-k - カーソルから行の末尾まで全ての文字を消去する

10.3 共有

Sonic Pi は、お互いに共有し学習するものです。

いったん、どのように音楽をコード化するかを学んだら、作曲したコードを共有することは電子メールを送信するのと同じくらい簡単なことです。あなたの作品から学び、さらに新しいマッシュアップで部分を使えるように他の人とコードを共有しましょう。

もしあなたの作品を共有するためのよい方法が見当たらなければ、あなたの音楽を SoundCloud へ、そしてコードを GitHub に置くことをお勧めします。その方法で、あなたの作品は、容易にたくさんの人に届けることができるのです。

コードを GitHub へ

GitHub は、コードを共有し作業するためのサイトです。コードの共有や共同作業のためにプロの開発者と同じくアーティストも使用しています。新しい作品のコード（あるいは未完の作品）を共有する最も簡単な方法は、この GitHub で Gist を作成することです。Gist 機能では、コードをアップロードすることで、簡単に他の人が参照、コピーし共有することが出来ます。

音を SoundCloud へ

作品を共有するもう一つの重要な方法は、音を録音し SoundCloud にアップロードすることです。作品をアップロードしたら、他のユーザーがコメントしあなたの作品について話し合うことができます。また、トラックの詳細にコードを参照するための Gist へリンクを貼ることをお勧めします。

作品を記録するには、ツールバーの Rec ボタンを押すと、すぐに録音を開始します。もしコードが再生中でなければ、開始するために Run を押してください。録音が完了したら、点滅している Rec ボタンを押すと、ファイル名を入力するよう求められます。WAV ファイルとして保存された録音は、無料のソフトウエア（例えば、 Audacity を試してみてください）の任意の設定によって編集したり、MP3 に変換することができます。

希望

私はみなさんが作品を共有し、Sonic Pi での新しいトリックや動きをお互いに教えあってもらえることを心から願っています。あなたが何を魅せてくれるか本当に楽しみにしています。

10.4 パフォーマンス

Sonic Pi の中で最もエキサイティングな側面の一つは、楽器としてコードを使うことができるということです。これは、コードをライブで書き込むことが、音楽を演奏する新しい方法とみなすことができることを意味します。

我々は、これをライブコーディングと呼んでいます。

画面を表示しよう

コードをライブするとき、観客にあなたの操作画面を表示することをお勧めします。そうでなければ、ギターを指や弦を隠しながら演奏するようなものです。私は家で練習するときには、Raspberry Pi と小型プロジェ

クターでリビングルームの壁に投影しています。自分のテレビや学校や職場のプロジェクターを使ってみましょう。挑戦してみてください、とても楽しいですよ。

バンドを結成しよう

絶対に 1 人で遊ばないでください - ライブコーディングバンドを結成しましょう！他の人とのセッションはとても楽しいものです。一人はビートを担当し、他の BGM や環境音など、どんな面白い音の組み合わせがつくれるか一緒に試しましょう。

TOPLAP

ライブコーディングは新しいものではありません。一部の人々は、自ら構築した特別なシステムを使用して、ここ数年でライブコーディングに取り組んできました。他のライブコーダーやシステムについての詳細を知るには TOPLAP という絶好の場所があります。

Algorave

ライブコーディングの世界を探求するためのもう一つの偉大な情報があるのは Algorave です。ここでは、クラブシーンでのライブコーディングに特化した情報を見つけることができます。

11 章 Minecraft Pi(マインクラフトパイ)

Sonic Pi は現在、 Minecraft Pi と対話するためのシンプルな API をサポートしています。Minecraft の特別版は、Raspberry Pi の Linux ベースのオペレーティングシステム Raspbian にデフォルトでインストールされています。

ライブラリは不要

Minecraft Pi は、とっても簡単に扱えるよう設計されています。必要なことは、Minecraft Pi を起動して世界を創造するだけです。その後、play や synth を扱うように mc_*関数を自由に使います。何かのライブラリをインストールしたり、インポートする必要はありません。箱から出して動かしてみましょう。

自動接続

The Minecraft Pi API は Minecraft Pi アプリケーションへの接続を可能にします。あなたは何も心配をしなくてもよいということです。Minecraft Pi を起動せずに、Minecraft Pi API を使おうとした場合には、Sonic Pi はこれを丁寧に教えてくれます。同様に、live_loop を実行する一方で、もしも、Minecraft Pi を閉じてしまっても、live_loop の接続を停止し、接続できてないことを丁寧に伝えてくれます。再接続するために、再び Minecraft Pi を起動して、Sonic Pi が自動検出して、再接続を試みます。

ライブコーディングのデザイン

Minecraft Pi API は live_loop 内でシームレスに動作するように設計されています。これは、Sonic Pi の音に変更を加え、Minecraft Pi の世界の変更と同期させることが可能であることを意味します。インスタントな Minecraft ベースのミュージックビデオです！ Minecraft Pi はアルファ版のソフトウェアであり、わずかに不安定であることに注意してください。何か問題が発生した場合は、単純に Minecraft Pi を再起動し、以前と同様に作業を続けましょう。Sonic Pi の自動接続機能が対応します。

Raspberry Pi 2.0 が必要

Sonic Pi と Minecraft の両方を同時に実行したい場合、特に Sonic Jam Pi のサウンド機能を使用したい場合は Raspberry Pi 2.0 を使用することをお勧めします。

API サポート

現段階では、Sonic Jam Pi は、次のセクション 11.1 から詳述されている基本ブロックとプレイヤーの操作をサポートしています。 世界の中のプレイヤーの相互作用で発せられるイベントのコールバックのサポートは、 将来的なリリースバージョンで予定されています。

11.1 Minecraft Pi API の基礎

Sonic Pi は現在、下記の Minecraft Pi の基本インタラクションをサポートしています:

- チャットメッセージの表示
- ユーザーの位置設定
- ユーザーの位置情報の取得
- 指定した座標にブロックタイプを設定
- 指定した座標のブロック タイプを取得

これら、それぞれを順番に見てみましょう。

チャットメッセージの表示

それでは Sonic Pi での Minecraft Pi の制御が、どれだけ簡単か見てみましょう。まずはじめに Minecraft Pi と Sonic Pi が同時に起動しているかを確認し、 Minecraft の世界に入ることが出来ることを確認してください。

新しい Sonic Pi のワークスペースで、次のコードを入力してください：

```
mc_message "Hello from Sonic Jam Pi"
```

Run ボタンを押すと、Minecraft ウィンドウにあなたのメッセージが表示されます。おめでとう、あなたは最初の Minecraft コードを書きました！簡単でしたね。

ユーザーの位置の設定

これから、小さな魔法を試してみましょう。どこかに瞬間移動してみましょう！以下を試してください。

```
mc_teleport 50, 50, 50
```

Run を押すとボーン！新しい場所へあなたの座標が移動しました。たいていは、乾燥した土地、または水に落ちたか、空中のどこかでしょう。50, 50, 50、これらの数字は何でしょう?これらは瞬間移動しようとしている場所の座標です。Minecraft をプログラムするにあたり、座標がどのように動作するのか、本当に重要であるので、短い時間でみていきましょう。

座標

いくつかの宝の場所が、大きな X 印でマーキングされた海賊の地図を想像してみてください。X の正確な位置は、左から右へ向かってどれくらい離れているか、下から上へ向かってどのくらい離れているか、この二つの数字で記述することができます。例えば、横へ 10cm、上に 8cm。これら２つの数字 10 と 8 が座標です。他に隠された宝物の在処も別の２つの数字で容易に記述できることが想像できますね。おそらく、2 つ横切り、9 つ上には大きな金脈がありそうです…

さて、 Minecraft の中では 2 つの数字では十分ではありません。また、私たちがどれだけ高い所にいるのかを知る必要があります。したがって、3 つの数字が必要になります。

- どのくらい右端から、左へ - x
- どのくらい手前から世界の奥へ - z
- どのくらい高くあがったか - y

通常、 x, y, z でこれらの座標を記述します。

現在の座標を知る

座標を使って遊んでみましょう。Minecraft のマップで素敵な場所に移動した後、 Sonic Pi に切り替え、次を入力してください:

```
puts mc_location
```

Run ボタンを押すと、ログ画面に、現在位置の座標が表示されます。それらを書き留め、前方に移動して再び挑戦してください。座標がどのように変化するかに注目しましょう！

これを正確に繰り返す練習をお勧めします。座標移動を繰り返して、ビットの世界を移動しましょう。座標がどのように変化するか感触を得られるまで、これを行いましょう。これを調整する方法が理解できれば、Minecraft の API を使用したプログラミングはほぼ完了します。

さあ、構築しましょう。

現在位置をどのように知り、座標を利用して瞬間移動する方法を知っているあなたは、Minecraft 内に、コードでなにかを構築し始めるための必要な道具をすべて持っています。座標 40, 50, 60 にガラスのブロックを作りたかったら、それはとっても簡単ですね:

```
mc_set_block :glass, 40, 50, 60
```

ハハ、本当に簡単でしたね。あなた handywork を表示するには ちょっと近くに瞬間移動してみましょう:

```
mc_teleport 35, 50, 60
```

振り向くと、あなたのガラスのブロックが表示されるはずです！それをダイヤモンドに変更してみましょう:

```
mc_set_block :diamond, 40, 50, 60
```

もし正しい方角で見れば、それがあなたの目の前で変更される可能性があります！ これはエキサイティングな何かの始まりです…

ブロックを調べる

少し複雑なことに移る前に、もう一つお伝えします。座標の集合を与え、特定のブロックの種類が何であるかを Minecraft に訪ねることができま

す。 では、先ほど作成したダイヤモンドブロックでそれを試してみましょう:

```
puts mc_get_block 40, 50, 60
```

イェイ! それは :diamond(ダイヤモンド）ですね。ガラスに戻して、 もう一度尋ねてみましょう。:glass を示しましたか？信じています(^o^)

使用可能なブロックタイプ

Minecraft Pi コーディングへ暴れに行く前に、利用可能な ブロックタイプの便利なリストを示します:

:air	空気
:stone	石
:grass	草
:dirt	汚れ
:cobblestone	石畳
:wood_plank	木の板
:sapling	苗木
:bedrock	岩盤
:water_flowing	水流
:water	水
:water_stationary	静止した水
:lava_flowing	溶岩流
:lava	溶岩
:lava_stationary	固まった溶岩
:sand	砂
:gravel	砂利
:gold_ore	金の鉱石
:iron_ore	鉄鉱石
:coal_ore	石炭鉱石
:wood	木材
:leaves	葉
:glass	ガラス
:lapis	ラピス
:lapis_lazuli_block	ラピスラズリブロック

:sandstone	砂岩
:bed	ベッド
:cobweb	クモの巣
:grass_tall	背の高い草
:flower_yellow	黄色い花
:flower_cyan	シアン色の花
:mushroom_brown	茶色いキノコ
:mushroom_red	赤いキノコ
:gold_block	金のブロック
:gold	金
:iron_block	鉄のブロック
:iron	鉄
:stone_slab_double	石板(ダブル)
:stone_slab	石板
:brick	レンガ
:brick_block	レンガブロック
:tnt	
:bookshelf	本棚
:moss_stone	苔石
:obsidian	黒曜石
:torch	トーチ
:fire	火
:stairs_wood	木の階段
:chest	チェスト
:diamond_ore	ダイアモンドの鉱石
:diamond_block	ダイアモンドのブロック
:diamond	ダイアモンド
:crafting_table	作業テーブル
:farmland	農地
:furnace_inactive	廃炉
:furnace_active	炉
:door_wood	木のドア
:ladder	はしご
:stairs_cobblestone	石畳の階段

:door_iron	鉄のドア
:redstone_ore	レッドストーン鉱石
:snow	雪
:ice	氷
:snow_block	雪のブロック
:cactus	サボテン
:clay	粘土
:sugar_cane	サトウキビ
:fence	フェンズ
:glowstone_block	光る石のブロック
:bedrock_invisible	目に見えない岩盤
:stone_brick	石レンガ
:glass_pane	ガラス板
:melon	メロン
:fence_gate	フェンスゲート
:glowing_obsidian	輝く黒曜石
:nether_reactor_core	原子炉コア

12 章 おわりに

これで Sonic Pi 入門のチュートリアルを終了します。何かを学べたのではないでしょうか。すべてを理解していなくても、心配は無用です。とにかく時間を使って演奏し楽しんでください。質問があれば、気軽にチュートリアルを読み直してみてください。

もし、チュートリアルでカバーされていない疑問がある場合は、Sonic Pi forums を開き、質問をしてください。そこでは、誰かが親身に手を貸してくれるでしょう。

最後に、このヘルプシステムの他の部分をより深くチェックすることをお勧めします。このチュートリアルではカバーしていない機能がいくつもあるので、新たな発見が待っているでしょう。

遊んで、楽しんで、コードを共有して、そしてスクリーンを見せながら友人のために演奏してしてください。そして、思い出してください。

間違いはない、あるのはただ可能性だけ

Sam Aaron