Coding to Make Music with Sonic Pi

Coding & STEAM 2019 – Coding and the Creative Arts

# Getting Started

In this activity, you will learn about *Sonic Pi* and the following **Computational Concepts**:

* *Sequences*
* *Loops*
* *Conditionals*
* *Parallelism*

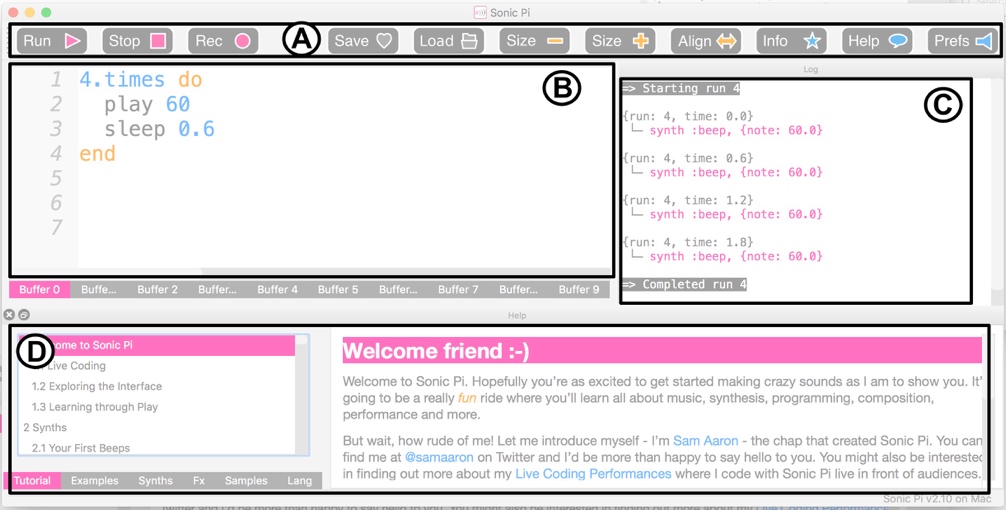
*Sonic Pi* is a program that can be used to create music by writing code. This program is a musical instrument, educational tool and a Coding environment, all in one.

This tutorial includes parts from a few existing different Sonic Pi tutorials, produced by Dr Sam Aaron and members of the Raspberry Pi Foundation. These tutorials are available here: <https://goo.gl/CzE8sK> and <https://goo.gl/Mwb4RP>

To open *Sonic Pi* on your lab computer, go to the *Start Menu*, then *All Apps*, and look for *Sonic Pi*. The program may take 30 seconds to 1 minute to open.

# The Sonic Pi Interface

Once *Sonic Pi* opens, you should see an interface like the one below:



**A** is a bar with all the different controls. The pink controls are for controlling the playback of your program. The grey controls allow you to save and load your program. The yellow controls allow you to adjust the size of the program’s text. The blue controls allow you see information about *Sonic Pi*, the help documentation and change the program’s preferences.

**B** is the **Code Editor**. This is the area where you’ll write your code and compose/perform music.

**C** is the **Log Viewer**. When your song is playing, this will show all of the notes being played. This is useful when trying to figure out what’s going in your program and fixing errors (debugging).

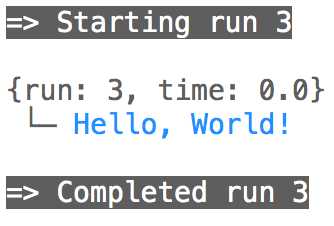
**D** is the **Help Viewer.** This area contains help and information about all aspects of *Sonic Pi*, and includes a series of tutorials and example music.

# Your First Ruby Program

Many tutorials that introduce Coding start with a program called “Hello, World!”. In *Sonic Pi*, you can create this program by typing the following code into the Code Editor:



After typing this, and clicking the **Run** button, you should see a message like this in the *Log Viewer* (on the right side of the screen):



Congratulations! You just wrote a program in *Sonic Pi*.

*Sonic Pi* uses a subset of the *Ruby* programming language. This means that, although not all *Ruby’s* features are available in *Sonic Pi*, writing *Sonic Pi* code is writing *Ruby* code. So, when you are learning how to use *Sonic Pi*, you are learning *Ruby*.

If you found an online tutorial for *Ruby*, the steps in the tutorial would likely work in *Sonic Pi*. The real strength of *Sonic Pi*, however, is as a musical instrument.

Next, remove the line you wrote in the *Code Editor* and write this instead:



After typing this, and clicking the *Run* button, you should see hear a note play.

What’s happening here? The **play** command tells *Sonic Pi* that it needs to play a note. The 60 is the note that should be played. *Sonic Pi* uses numbers, as well as note names, for representing notes. 60 is a note that is called Middle C.  
  
What happens when you change the 60 to a different number? Try numbers like 30 and 90. What do you notice?

As the number increases, the pitch of the note gets higher. Conversely, when you decrease the number the note becomes lower in pitch.

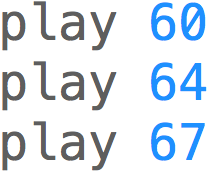
In the next section, we will create and play our first song, while learning some programming concepts along the way.

# Your First Songs

## Sequences

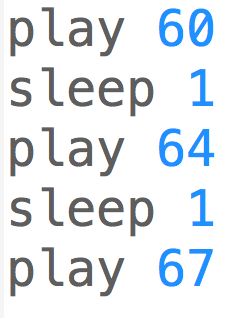
**Sequences** are an essential *Computational Concept*. When you run a program, the computer will follow each instruction you give it in order, as a sequence, from top to bottom. We’ll investigate *sequences* in this section, by writing a melody.

Replace the code you have written in the Code Editor, with this below:



When you click *Run*, you should hear 3 notes play at the same time. Is this what you expected? You may have expected 3 notes to play separately, one after the other. In *Sonic Pi*, we must write code that will affect how long the notes last. To do this, we use the **sleep** command.

Next, change your code so it looks like the code shown below:



Now when you run the program, you should hear a melody playing.

The **sleep** command tells Sonic Pi to wait a certain amount of time before playing the next note. The 1 used in the example above, is not 1 second, it means 1 beat. If you are familiar with musical notation, you may know of bpm (beats per minute). The default bpm in Sonic Pi is 120, so when you **sleep** 1 beat this means the program waits for around half a second, before playing the next note.

If you put the following code above the melody code, and run the program again, it should sound different:



The sleeping will take longer, as the beats per minute is lower. Conversely, if you change the 60 to 240 and run the program again, the melody will play more quickly because the sleeping will take less time.

## MIDI and Notation

When you are writing numbers (e.g. 60 and 64), you are using numbers that represent MIDI notes. MIDI is a technology standard that is used for computer generated (electronic) music and instruments like synthesizers.

You may be familiar with musical notation. You don’t need to know any musical notation to complete this tutorial, but this section may be a little confusing if you don’t.

The image below shows how the MIDI notes map to notes on the treble and bass clefs. For example, Middle C = 60 in MIDI. In Sonic Pi this can be played by using **:C** and **:C4**, as well.



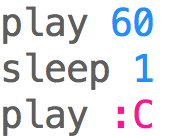
We used **sleep** in the last section, to leave time between notes. By using **sleep** 1, the note played for 1 beat, before the next note played. In Sonic Pi, we can make notes play different durations, by changing the number we put after the **sleep** command. For example, see the images below with the different note lengths:



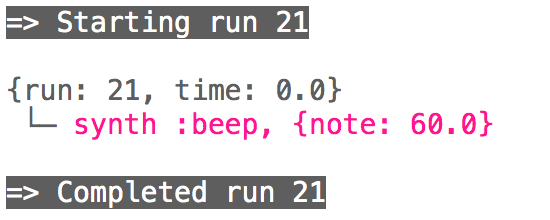
In the next section, we’ll look at how we could write a song in *Sonic Pi*, by mapping musical notation to code.

*Variables* are another *Computational Concept*. We will not go into much detail about these in this tutorial, but we will look at some variables that are built-in to Sonic Pi.

Sonic Pi allows you use to use variables, named after different notes, instead of writing a MIDI number. For example, the two play lines in the code below will play the same note twice:



When you use a name (for example: **:C)** for a note, when the song is playing, you will be able to see the MIDI number this represents in the *Log Viewer.* For example, in the image below, you can see that the note being played is 60, even though the code said “**play :C**”:



In the following sections, we will use a combination of MIDI numbers (e.g. 60) and the note names (e.g. C).

## Loops

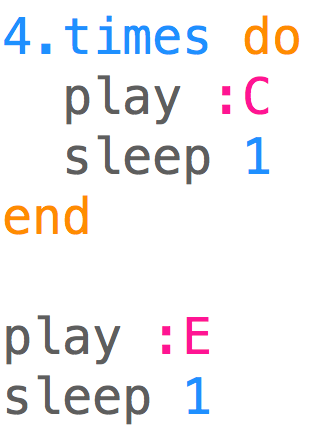
*Loops* are another *Computational Concept*. In Coding, we can use *Loops* to repeat instructions however many times we like.

For example, to repeatedly play the same note over and over, we would use a *Loop*. Write this code and run it:



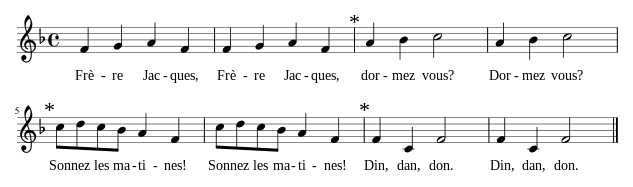
When you *Run* the code above, you will hear the same note (C) play repeatedly. The code between **do** and **end** will run over and over again (in the *Loop*). The E note will never be played. This is because we have not said how many times the *Loop* should run, or when the *Loop* should stop.

Let’s change the code so that the C note 4 times and then plays 1 E note. We would change the program to look like the code below:



After running the code above, you will hear 5 notes – 4 Cs, followed by an E.

You may know of some songs that have repeating melodies. An example of a song that does have repeating melodies is the nursery rhyme, *Frère Jacques*. The notation for this song is shown below.

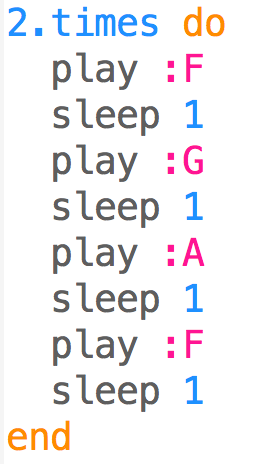


You may notice patterns of notes that repeat themselves. For example, the 2nd bar is a repeat of the 1st bar, the 4th bar is a repeat of the 3rd bar, and so on.

The first bar of this song could be written in Sonic Pi as:



Now, if we put this code in a loop that runs two times, like below, we get the first two bars of Frère Jacques:



If you’re familiar with musical notation, you may be able to write the rest of the song. If you were to program the rest of the song, how would you use loops?

The full song is written out in code here: <https://goo.gl/49WtFt>

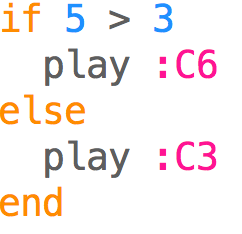
If you copy, paste and run this code in Sonic Pi, you will hear the whole song.

In the next section, we’ll look at *Conditionals* and *Randomisation*.

## Conditionals & Randomisation

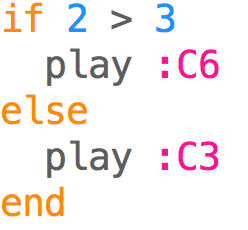
*Conditionals* are another *Computational Concept*. When you use *Conditionals*, you can run different code based on different conditions. To demonstrate *Conditionals*, we will combine *Conditionals* with *Randomisation* to play a song.

*If/else statements* are an example of *Conditionals* in Coding. There is an *if and else statement* in the code below:



When you run the code, you should hear a high note (C6). The if statement checks if the condition (in this case, if 5 is greater than 3), and runs the code in the if block if it’s true. The line “**play :C3**” does not run at all in the example above.

Now, change the 5 to a 2, so it looks like the code below:



When running this code, you’ll hear a low note (C3) play instead. The if statement checks if the condition (in this case, if 2 is greater than 3), and runs the code in the if block if it’s true. In the example above, the code in the else block (the line “**play :C3**”) is run, because 2 is not greater than 3.

Now we’ll use *Randomisation* to create a melody. *Randomisation* is used in many different types of programs, especially ones where there is chance involved. Two examples could include a Blackjack game where *Randomisation* would be used to shuffle a deck of cards and videogames where *Randomisation* could be used to determine where and when enemy characters appear on the screen and how they behave. Inthe arcade game *Pacman*, for example, *Randomisation* could be used to randomly choose which direction one of the ghosts chasing *Pacman* would move next.

There are few different ways to use *Randomisation* in *Sonic Pi*. In this activity, we will focus on creating melodies that involve generating random note numbers with the *rrand\_i* command and simulating a coin toss with the *one\_in* command.

It is important to note that the way random numbers are created in recent versions of *Sonic Pi* is different to the way random numbers in other Coding languages (for example: *Scratch* or *Python*). *Sonic Pi* uses *reliable Randomisation*. This means that when you create a random number in *Sonic Pi*, it will be predetermined and consequently the “random” number will be the same each time you run the program.

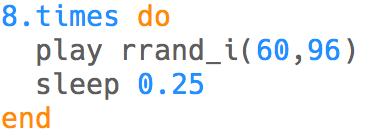
For example, type the following code (which creates and prints a random number) into *Sonic Pi* and run it:



In the *Log Viewer,* you will see that the random number created is *0.75006103515625*. The *rand* function creates a random number between 0 and 1. If you leave the code the same and run it again, you will notice that the random number is *0.75006103515625* again. As you can see, this is not actually random, as we are receiving the same number each time.

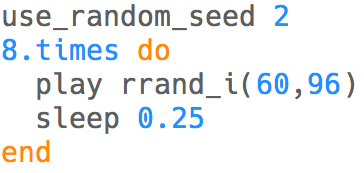
This may seem strange, as you might expect the *Randomisation* to create a new random number each time you run the program. This approach might not be ideal for some uses of *Randomisation*. For example, imagine a *Blackjack* game where you always knew the order of the cards in the deck! However, as explained in more detail in the *Sonic Pi* manual (<https://goo.gl/jzbjp1>), this approach to R*andomisation* is ideal for sharing songs. When you share your code with someone else and you have used *Randomisation* to create the melodies, it is guaranteed to sound the same on your computer as it sounds on theirs.

Each time we use a random command in a *Loop* in *Sonic Pi*, we will get a new “random number”. For example, to create a melody that plays 8 random notes we can write the following code:



This code will play 8 random notes, each a quarter of a beat apart. The *rrand\_i*command creates a random integer (whole number) within a range. There are two parameters to the *rrand\_i* command (which are 60 and 96 in the example above). These tell the *rrandi* command the range in which the random number will come from. The first is the lower bound of the range (for example: 60 in the above code) and the second is the upper bound of the range (for example: 96 in the code above). This means that the random number created will be a whole number between 60 and 96.

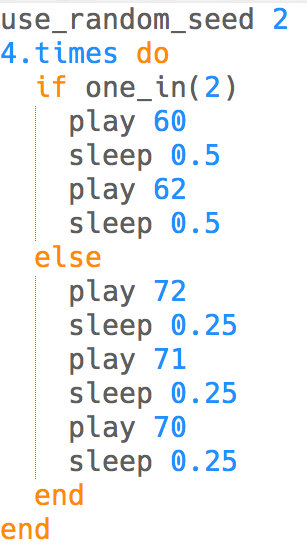
If you run the code above a few times, your melody with 8 notes will play. You will notice that the melody always sounds the same. However, there is a way to change the random notes that play using a command named *use\_random\_seed*. The *seed* is a name of a number that affects how the random number is created. To use this command, change your code to look like the below example and run the code again:



You will notice that the melody sounds different now. Change the seed (2 in the code above) and try running the code again a few times, notice how the melody changes when the seed changes. If you were composing a song in *Sonic Pi* you might experiment with different random seeds until you found one that created a melody that you liked.

Next, we’ll combine the use of *Conditionals* and *Randomisation*. We will simulate a coin toss by using the *one\_in* command. The *one\_in* command is given one input, which is the number of possible outcomes. The *one\_in* command will give us a result of either *true* or *false* randomly with an appropriate probability. For example, for *one\_in(2*), there is a 50% chance that the result of the command will be *true* and for *one\_in(10)* there is a 10% chance that the result of the command will be *true*.

In a coin toss there are 2 outcomes (*heads* or *tails*), so will we use the command *one\_in(2)*. We will have two melodies that are chosen randomly in a *Loop* that runs 4 times. If the *one\_in(2)* is *true* (let’s say that this is the same as getting tails), we will play one melody and if *one\_in(2)* is *false*, we will play another melody. This is shown in the code below:



The first melody has 2 notes that ascend in pitch and the second melody has 3 notes that descend in pitch. When you run this code, the following will happen four times:

* The *one\_in(2)* commandresults in either: *true* or *false*
* If the result is *true*, the ascending pitch melody will play
* If the result is *false*, the descending pitch melody will play

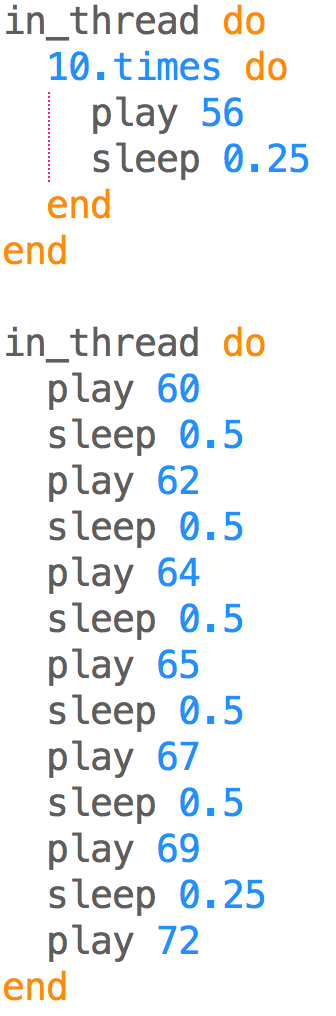
Like the example given above, where we played the melody with 8 notes, every time you run this code it will sound the same. However, if you adjust the *seed* value (2 in the code above) and run the code again you will hear a different melody.

In the next section, you will learn about *Parallelism*.

## Parallelism

*Parallelism* allow you to run two or more *Sequences* of code at the same time. In *Sonic Pi*, you can run sequences in parallel by using *threads*. In Sonic Pi, using threads can allow you to play a bass line and melody at the same time.

The code below shows an example of the use of *threads*. When you run the code, you will hear the main melody, and a simple bass line play at the same time. That is, the music in the first **in\_thread** block will play at the same time as the music in the second **in\_thread** block.



Congratulations! You’ve learned how to make simple songs in *Sonic Pi*, using *Computational Concepts*. In the next section, we explore some of the extra features that *Sonic Pi* offers.

# Extra Features

In this section, we will explore some extra features of *Sonic Pi* briefly. The *Help Viewer* in *Sonic Pi* will provide more in-depth content and examples about these topics, if you are interested in finding out more.

## Samples

*Sonic Pi* can be used to play samples, which can then be used as part of your compositions. For example, you can play a drum loop sample, by writing the following code:



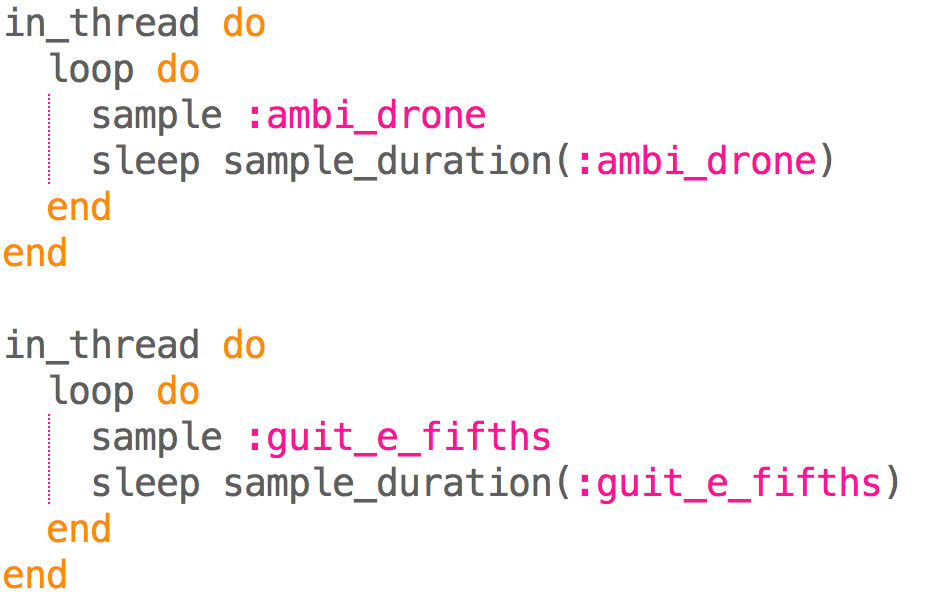
We can speed the sample up, by changing the rate:



Or slow it down, by decreasing the rate:



Samples can be used to compose different songs, and can be run in threads. The code below plays two samples at the same time:



Samples have different lengths. In the example above we used the **sample\_duration** command to make *Sonic Pi* sleep for the appropriate time. There are many different samples available in *Sonic Pi*, and you can even use your own custom ones.

## Synths

In *Sonic Pi*, you can change the synth used for playing notes. There a variety of different synths available, that can make very different sounds. The default synth is called “beep”. One way of changing which synth is being used, is by using the **use\_synth** command. For example, the code below plays the E note, using the dsaw synth:



For a full list of different synths available, see *Sonic Pi’s* *Help Viewer*.

## Lists

A *List* is a type of data structure, common in many Coding languages (both *Blocks* and *Text* languages). One of the useful ways they can be used in *Sonic P* is to play multiple notes. A *List* can have 1 or more elements, which are separated by commas and these must be wrapped in brackets. An example of a *List* of notes (as MIDI numbers) is shown below:



You can put a command in front of a *List* to play some notes. If you put **play** before the *List* and run the code, what happens? It plays all of the notes together (as a chord). Now try with **play\_pattern** instead. You’ll notice that with **play**, all the notes are played at the same time. When using **play\_pattern**, each of the 3 notes will play for a beat, before the next one plays.

The use of *Lists* with commands like **play\_pattern** can make writing code for songs shorter. For example, the first bar of *Frère Jacques* could be written using the following code:



See this code for *Frère Jacques*, using *Lists*: <https://goo.gl/Eq9NjP> This has the full song, written in a much more compact way than we had before.

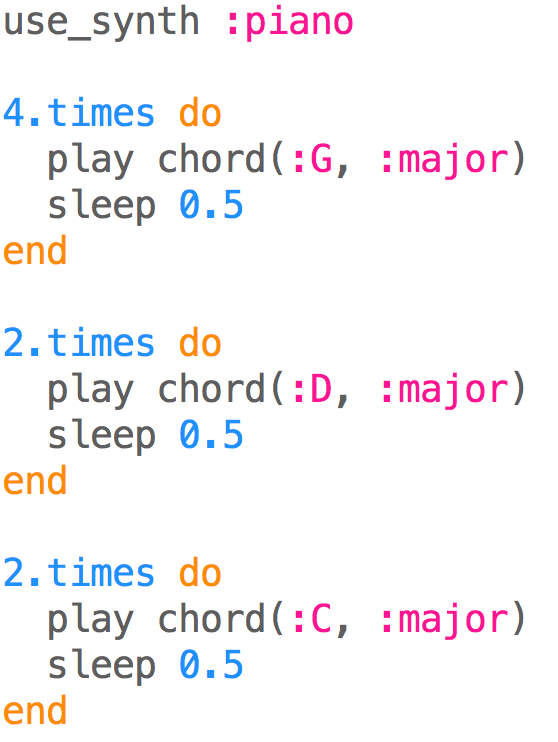
## Chords

You can also write music in Sonic Pi by using chords. In the *Lists* section we wrote a *List* of notes (52, 56, 59), which also happen to be the notes that make up the E major chord.

We could play the E major chord by writing and running the following code:



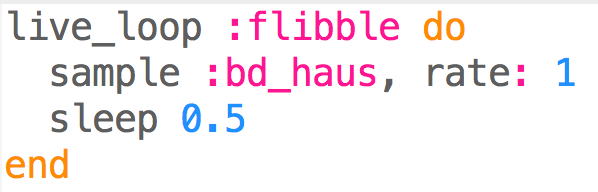
We can write chord progressions now, like in the code below:



# Live Coding

One of the main purposes of *Sonic Pi*, is for it to be used for *Live Coding*. *Live Coding* mixes live music and coding, by allowing you to change your music (code) without stopping the music completely. You can view some *Live Coding* examples and performances on the *Sonic Pi* website.

To hear an example of *Live Coding* in action, write the following code:



Now adjust the time the song sleeps (0.5) to another number (e.g. 0.2) and run the code again. The music won’t stop, and the speed of the drum will increase. Try adjusting this number again, running the code again and see how the music changes.

Live loops allow you to use *Sonic Pi* like you would with a live instrument (for example: a guitar). *Sonic Pi* has been used with these live loops, to perform real music gigs by students and professionals, all over the world.

Good work, you have now completed this tutorial! Now that you are finished, you may want to look at the *Help Viewer* in *Sonic Pi* or experiment with your own compositions.

# Acknowledgements

The following images have been included in this tutorial:

* MIDI notation, from <http://www.theoreticallycorrect.com/Helmholtz-Pitch-Numbering/>
* Note durations, from, <http://sonic-pi.mehackit.org/>
* Frère Jacques notation, from <https://en.wikipedia.org/wiki/File:Fr%C3%A8re_Jacques.svg>