**Sonic Pi Handout:**

Sonic Pi is a program developed by Sam Aaron as an instrument to simultaneously teach programming and music. It comes pre-installed on every NOOBS operating system for Raspberry Pi. The user can directly program sounds, rhythms, and samples, and as a result, entire songs can be created in Sonic Pi, with enough diligence and effort. This short guide is a brief introduction to the necessities of using the software. More information can be found in the SonicPi tutorial, available within the software (and the user is highly encouraged to look at this documentation).

**Essential commands**:

* play: plays whatever note is specified, constrained to whatever options are specified.

e.g.

play 60, amp: … , sustain: … , attack: … , release: … , decay: …

“60” is middle C in this case. However, this is kind of unintuitive when making music. You can specify a note and an octave if you wish! So 60 translates to:

play :C

play :Fs6 or play :Bb5 (F# sixth octave, B flat fifth octave)

* sleep: tells the program how many beats to rest for.

e.g.

sleep 1

* loop do … end: used to loop sections of code / music

e.g.

loop do

#some code here

end

* in\_thread: creates a new thread (which can also be named), so that one can loop a certain section, but continue to more parts of the code.

e.g.

in\_thread (name: :thread\_name) do

loop do

#some code here

end

end

**Lists:**

There is a data structure in programming called a “list.” It is exactly what the name implies and is crucial in the SonicPi software. You can create one like so:

[0.25, 1.5, 1, 6,..., #however many things]

Basically you just list things out in between the []. Typically they need to be of a certain type, like how in the previous example, they’re all numbers (but this is not a strict rule and varies depending on use of the list).

**Some other useful commands**:

use\_bpm: Specify at beginning of code to tell program what beat per minute to play at

e.g.

use\_bpm 120

use\_synth: this allows the user to select a different synth, meaning the notes you play take on a different sound (this correlates physically to different sounds waves, like a sinusoid verses a square wave). The auto-complete in the software will show the user the available synths, and you are of course invited to try out all of them!

e.g.

use\_synth :whichever\_synth

play\_patterened\_sequence: this allows one to play lists containing sequences of notes or chords (allowing for arpeggiation of said chords)

e.g.

play\_patterened\_sequence [list of notes], [circle of times], #other options

play\_patterened\_sequence chord(:chord, :chord\_options)

play\_chord: plays a specified chord

e.g.

play\_chord (:chord, :chord\_type)

**Some things about the previous commands**:

* You’ll notice play has several options following it. These are all explained in-depth in the SonicPi tutorial under the section labeled “Duration With Envelopes.” These options extend to any command that makes sound, and can be specified in any order.
* For in\_thread, the (name: :thread\_name) is optional. Using this command makes your thread a “named thread.”
* Even if the user continually reruns their code, a named thread will only run one time. This can be used to avoid a sort of cacophony.
* In the play\_patterened\_sequence command, you can specify the time intervals between notes played in a sequence via a list. This is called the “circle of times,” because if you have fewer intervals specified than notes, the program will just cycle to the beginning of the interval list.

**Functions:**

You can create functions which, when called upon in the main body of code, will perform whatever task you’ve programmed. This can be extremely useful when you want to do the same thing multiple times, as you just have to call the function rather than repasting blocks of code.

Creating a function looks like this:

function\_name do |arg1, arg2, … ,|

#function code

end

Calling a function looks like this:

#some previous code

function\_name arg1, arg2…

#some following code

The |arg1, arg2, … ,| is a way of passing external variables to the function. It is not required to have this list, and the provided “arg” names are just placeholder names. You could call your variables whatever you like, e.g. |attack\_lvl, sustain\_lvl, amp\_lvl|.