Sonic Pi

((((Sonic π))))
Starter – label the Sonic Pi interface
www.raspberrypi.org/learning/sonic-pi-taster
Starter

Play
Stop
Save
Workspace tabs
Output panel
Programming panel
Error panel
Playing Notes

- Type Play 60 into Sonic Pi and press the Run button.
- You should information about the sound played in the log panel.
Playing Notes

• Type Play 60 into Sonic Pi and press the Run button
• You should information about the sound played in the log panel
Check syntax

• Syntax is the way that you write the code
• Check out what happens if you type in and run the following

• What can you conclude?
Play command

• Type in the following code

```
1  play  60
2  play  67
3  play  69
```

• What happens?
Sleep command

• Type in the following code

```plaintext
1  play 60
2  sleep 0.5
3   play 67
4  sleep 0.5
5  play 69
6  sleep 0.5
```

• What happens now?
• What do the numbers represent?
• What does the sleep 0.5 do?
First tune

• Type in the following code

```
1 play 60
2 sleep 0.5
3 play 60
4 sleep 0.5
5 play 67
6 sleep 0.5
7 play 67
8 sleep 0.5
9 play 69
10 sleep 0.5
11 play 69
12 sleep 0.5
13 play 67
```

• This is the first part of “Twinkle Twinkle Little Star”

• What would the code look like if you carried on to write the rest of the tune?
Lists

• Type in the following code

```
 play_pattern [60, 60, 67, 67, 69, 69, 67]
```

• This is called a list. How does it know when a new number or note in the list starts and the previous one ends?
• The play_pattern command instructs the program to play the numbers or notes one at a time
• What do you notice about the speed of the tune?
Changing tempo or bmp

• Type in the following code

```plaintext
use_bpm 100
play_pattern [60, 60, 67, 67, 69, 69, 67]
```

• *use_bmp* sets the tempo or speed of the tune
• Can you guess what *bmp* stands for?
• Can you guess what the number 100 is?
• Experiment with the *use_bmp* number. What do you notice? Can you explain this? How would you make the notes play every second? Every half a second?
Repeats or Loops

• Type in the following code

```plaintext
1 use_bpm 100
2 play_pattern [60, 60, 67, 67, 69, 69, 67]
3 sleep 0.5
4 play_pattern [60, 60, 67, 67, 69, 69, 67]
```

• What does the code do?
• How could we make it more efficient?
• Remember Scratch’ or Hour of Code?
Repeats or Loops

• Change your code

```ruby
use bmp 100
2.times do
  play_pattern [60,60,67,67,69,69,67]
sleep 0.5
end
```

• Explain 2.times do
• Why is it important include ‘end’
• Get the tune to play 3 times
• Get the tune to play 4 times with 1 second between each loop
You may have noticed that some of my code is indented (moved to the right by 2 spaces)

This is good practice when you have a loop. It makes it easier to see that this block of code is being repeated.

```ruby
use bmp 100
2.times do
  play_pattern [60, 60, 67, 67, 69, 69, 67]
sleep 0.5
end
```
• Musical notes represented by numbers are called midi notes
• A midi tune can be made to repeat many, many times even using a short piece of code

```ruby
use bmp 100
1000.times do
  play_pattern [60,60,67,67,69,69,67]
sleep 0.5
end
```

• Short code loads quickly and allows the computer to use its resources for other things
• Can use think of a use for midi tunes?
Create a tune

- Write the rest of “Twinkle Twinkle Little Star”

<table>
<thead>
<tr>
<th>Note Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
</tr>
<tr>
<td>60</td>
</tr>
</tbody>
</table>

C C G G A A G
F F E E D D C
G G F F E E D
G G F F E E D
C C G G A A D
F F E E D D C
Create a tune

- Did it look a bit like this?

```ruby
use_bpm 100
play_pattern [60, 60, 67, 67, 69, 69, 67]
sleep 1
play_pattern [65, 65, 64, 64, 62, 62, 60]
2.times do
  play_pattern [67, 67, 65, 65, 64, 64, 62]
end
sleep 1
play_pattern [60, 60, 67, 67, 69, 69, 67]
sleep 1
play_pattern [65, 65, 64, 64, 62, 62, 60]
```
Using variables

- It is time consuming looking up the note numbers each time you want to create a tune.
- You can add variables and give the program this information at the start.

<table>
<thead>
<tr>
<th>Note Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
</tr>
<tr>
<td>60</td>
</tr>
</tbody>
</table>
Using variables

• Type in

A=69
B=71
C=60
D=62
E=64
F=65
G=68
Using variables

• Now replace the numbers in the tune with notes

```ruby
use_bpm 100
sleep 1
  play_pattern [F,F,E,E,D,D,C]
2.times do
  play_pattern [G,G,F,F,E,E,D]
end
sleep 1
sleep 1
play_pattern [F,F,E,E,D,D,C]
```
Using variables

• However some notes sound too high.
• Can you identify which notes sound too high?
Using variables

- Did you identify that the notes which are too high are A and G?
- This is because of octaves. High octaves = High notes. You are using octave 3. A and G need to be an octave lower and use octave 2
- Identify the new numbers and change your program
Create a tune

• Try these songs

EDCDEEE
DDDEEE
EDCDEEE
EDDEDCC

GGAGCB
GGAGDC
GGGECBA
FFECDCC
CCGGAAD
FFEEDDC

EEE
EEE
EGCDE
FFF
FFEE
EDDEDCC
Using variables

• However some notes sound too high.
• Did you get it right?

\[
\begin{align*}
A &= 57 \\
B &= 71 \\
C &= 60 \\
D &= 62 \\
E &= 64 \\
F &= 65 \\
G &= 55 \\
\end{align*}
\]
Synths

• So far you have used the default synthesizer called pretty_bell
• You can change the synthesizer using the use_synth command followed by the name of the synth

```ruby
use_bpm 100
use synth :fm
2.times do
  play_pattern [C,G,F,F,E,E,D]
sleep 1
end
```
Synths

• Try other synths. When you type in use_synth the program pops up with a list of available synths to choose from (this is called intellisense)
Synths

• Try writing code which plays a tune with one synth e.g. `pretty_bell` and then plays the tune again using another synth e.g. `fm`

```plaintext
use_bpm 100
use synth :pretty_bell
play_pattern [C,G,F,F,E,E,D]
sleep 1
use synth :fm
play_pattern [C,G,F,F,E,E,D]
sleep 1
```
Synths

• Try writing code which plays a tune \textit{twice} with one synth e.g. \texttt{pretty\_bell}, speeds up, and then plays the tune again \textit{three times} using another synth e.g. \texttt{fm}

• Answers are on the next slide
Create a surprise

• You can use the rand function to generate a number between 0 and 1
• Half of the time this number will be less than 0.5 or < 0.5
• Half of the time this number will be greater than 0.5 or > 0.5
• You can use ‘if rand < 0.5’ to play a note e.g. 60
• If the rand number is not < 0.5 then you can use ‘else’ to play another note e.g. 65
Create a surprise

use synth :fm
10.times do
    if rand < 0.5
        play 60
    else
        play 69
    end
end

Do needs an end statement

If needs an end statement

Here it is

Here it is
Spooky Music

use_synth :pretty_bell
10.times do
  play 60 + rand(10)
  sleep 0.5
end

Rand (10) creates a random number between 0 and 10

The note 60 has a number, from 0 to 10 added to it. This randomly creates a note from 60 to 70.

Sequence repeats 10 times
Sorting and Shuffling

- You sort a list into order using `.sort`

```python
play_pattern = [65, 60, 67, 61, 69, 64, 63, 62, 66, 68].sort
```

- You can shuffle a list using `.shuffle`

```python
play_pattern = [69, 68, 67, 66, 65, 64, 63, 62, 61, 60].shuffle
```
Playing two musical sequences at once

The ‘in_thread do’ command plays at the same time as other sequences. This allows you to have two melodies playing at the same time.
Type in sample and the intellisense menu pops up. Choose a sample and play it.
Samples

You can alter the speed of the sample by adding a rate command.

Rate 0.5 = half speed, Rate 2 = twice speed, Rate -1 = reverse

```python
sample :loop_amen, rate: 1
sleep 3
sample :loop_amen, rate: 0.5
sleep 3
sample :loop_amen, rate: 1.5
sleep 3
sample :loop_amen, rate: -1
```

Try other samples and rates
Fade In and Fade Out

You can fade in a sample using: attack 1
This fades in the first second of the sample
```
sample :loop_amen, attack: 1
sleep 3
```

You can fade out a sample using: release 1
This fades out the last second of the sample
```
sample :loop_amen, release: 1
sleep 3
```

Or combine these commands

```
sample :loop_amen, attack: 1, release: 1
```
Random Between

You can use the rrand (50,100) function. This is short for ranged random. This would give a random note numbered between 50 and 100. You can also do this for the sleep command

```ruby
20.times do
  sample :perc_bell, rate: (rrand 0.125, 1.5)
  sleep rrand(0.2, 2)
end
```
Infinite beats

‘loop do’ and be used to create a loop which goes on forever

```ruby
loop do
  sample :perc_bell
  sleep 1
end
```
‘loop do’ and be used to create a loop which goes on forever

```ruby
loop do
  sample :perc_bell
  sleep 1
end
```
Sonic Pi has built in chords
I have
• set the beats per minute
• set the sample sound
• created a 3 times loop

I have then used
play chord
with the chord letter e.g. :A and
chord type e.g. : M (major), m (minor), m7 (minor 7)

Should be the opening to Coldplay Fix You
Chords

Why not add a percussion track

• loop do will loop forever (needs and end at the end of the loop)

• I have added an in_thread do so that the tune plays at the same time

• I need an extra end, to end the in_thread do