BBC micro:bit
Buzzer With MicroPython

Introduction

You will need a buzzer or speaker to use the examples on this page. A buzzer might cost you less than 50p to buy from an online store. You need one that works with 3V3 logic. Most of the cheaper buzzers work with 2V-5V and are fine. One of these will look something like this,

Connecting

It has two metal pins at the bottom. One of the pins is labelled **+**. You connect this to **pin 0** on your micro:bit. You can do this with the breadboard if you have the suitable connector. If not, use some aligator clips.

Connect the pin that is labelled **-** to GND.

Programming - Pitch

You can set the pitch of the buzzer sound with the **music.pitch()** method. This example uses that statement two ways. In the first example, the frequency 440hz is played for 1000 milliseconds. After waiting a second, the same tone is played with the length set to -1. This plays until it is stopped. You can't tell the difference when listening to the output from this program.

```
from microbit import *
import music
while True:
    music.pitch(440,1000)
    sleep(1000)
```

BBC Microbit

Collapse All Expand All

+ Block Editor - The Basics
+ Block Editor - Components
+ Kodu - micro:bit Worlds
+ JavaScript Blocks
+ JavaScript Blocks - Exercises
+ Blocks - Bit:Bot
+ Blocks - Bit:Commander
+ MicroPython - Starting Off
+ MicroPython - Examples
+ MicroPython - Components
+ MicroPython - Breakout Boards
+ MicroPython - Exercises
+ MicroPython - Pi Accessories
+ MicroPython - Bit:Bot
+ MicroPython - Bit:Commander
+ MicroPython - Projects
+ MicroPython - Visual Basic
+ Other - Odds & Ends

```
    music.pitch(440,-1)
    sleep(1000)
    music.stop()
    sleep(1000)
```

You can take advantage of the continuous tone to make a not-so musical instrument using the accelerometer readings to determine the pitch of the note being played. Pressing the A button controls whether or not a note is played.

```
from microbit import *
import music

while True:
    if button_a.is_pressed():
        music.pitch(accelerometer.get_x(),-1)
    else:
        music.stop()
    sleep(10)
```

Programming - Built-In Tunes

MicroPython has some built-in constants that can be played whenever you need. This program plays them all for you, one at a time, so that you can enjoy them at your leisure.

```
from microbit import *
import music
built_in_tunes = [music.DADADADUM, music.ENTERTAINER, music.PRELUDE,
music.ODE, music.NYAN, music.RINGTONE, music.FUNK, music.BLUES,
music.BIRTHDAY, music.WEDDING, music.FUNERAL, music.PUNCHLINE,
music.PYTHON, music.BADDY, music.CHASE, music.BA_DING,
music.WAWAWAWAA, music.JUMP_UP, music.JUMP_DOWN, music.POWER_UP,
music.POWER_DOWN]
while True:
    for tune in built_in_tunes:
        music.play(tune)
        sleep(5000)
```

Study the **music.play()** statement in the following program. The argument that is set to false makes the tune play in the background. Playing the tune does not block anything else that we want to do at the same time. The last argument, set to true, makes the tune play in a loop.

```
from microbit import *
import music
import random
built_in_images = [Image.HEART, Image.HEART_SMALL, Image.HAPPY, Image.SMILE, Image.SAD,
Image.CONFUSED, Image.ANGRY, Image.ASLEEP, Image.SURPRISED, Image.SILLY, Image.FABULOUS,
Image.MEH, Image.YES, Image.NO, Image.CLOCK12, Image.CLOCK11, Image.CLOCK10, Image.CLOCK9,
Image.CLOCK8, Image.CLOCK7, Image.CLOCK6, Image.CLOCK5, Image.CLOCK4, Image.CLOCK3,
Image.CLOCK2, Image.CLOCK1, Image.ARROW_N, Image.ARROW_NE, Image.ARROW_E, Image.ARROW_SE,
Image.ARROW_S, Image.ARROW_SW, Image.ARROW_W, Image.ARROW_NW, Image.TRIANGLE,
```

```
Image.TRIANGLE_LEFT, Image.CHESSBOARD, Image.DIAMOND, Image.DIAMOND_SMALL, Image.SQUARE,
Image.SQUARE_SMALL, Image.RABBIT, Image.COW, Image.MUSIC_CROTCHET, Image.MUSIC_QUAVER,
Image.MUSIC_QUAVERS, Image.PITCHFORK, Image.XMAS, Image.PACMAN, Image.TARGET, Image.TSHIRT,
Image.ROLLERSKATE, Image.DUCK, Image.HOUSE, Image.TORTOISE,Image.BUTTERFLY,
Image.STICKFIGURE, Image.GHOST, Image.SWORD, Image.GIRAFFE, Image.SKULL,
Image.UMBRELLA, Image.SNAKE]

music.play(music.PYTHON, pin0, False, True)
display.show(built_in_images, delay=1000,loop=True)
```

The display statement has a delay set that doesn't match the tempo of this tune. In the next section, you will come across some statements that might help you work out how to match synchronise the two things better than you can by trial and improvement.

Programming - Playing Melodies

In order to make your own music with MicroPython, you first need to get your head around its musical notation, the way it represents musical information.

Tempo

The length of individual notes is calculated from the tempo you set. The tempo is a number of **beats per minute (bpm)**. In order to divide a beat into notes, we use a value called **ticks**. By default, it takes 4 of these to make a beat.

You define a note with a letter and optional sharp symbol followed by a number to indicate its octave. Middle C is octave 4. The you write a colon, followed by the number of ticks the note should be played for. You can use an R to indicate a rest. For example,

C4:4

This means the note C from octave 4, or middle C, played for 4 ticks.

The following program plays you a tune, specified using this notation. After the first note, you can leave out the octave or duration until you need to change it. This tune was converted from code for a different microcontroller and so it was easier to leave the durations in.

```
from microbit import *
import music

music.set_tempo(bpm=240, ticks=4)

tune = [
'F#4:4','G4:4','A4:4','B4:12','A4:4','B4:4','E5:4',
'D5:4','B4:4','A4:4','G4:4','E4:12','G4:4','B4:4',
'C5:4','D5:12','E5:4','D5:4','B4:4','G4:4','B4:4',
'A4:16','R:4','F#4:4','G4:4','A4:4','B4:12','A4:4',
'B4:4','E5:4','D5:4','B4:4','A4:4','G4:4','E4:12',
'F#4:4','G4:4','A4:4','B4:12','C5:4','B4:4','A4:4',
'G4:4','A4:4','G4:16','R:4','D5:4','E5:4','F#5:4',
```

```
'G5:12','F#5:4','F#5:4','E5:4','D5:4','E5:4','D5:4',
'B4:4','G4:12','D5:4','E5:4','F#5:4','G5:12','F#5:4',
'F#5:4','E5:4','D5:4','B4:4','A4:16','R:4','D5:4',
'D5:4','D5:4','B5:12','A5:4','A5:4','G5:4','E5:4',
'G5:4','D5:4','B4:4','G4:12','F#4:4','G4:4','A4:4',
'B4:4','E5:4','D5:4','B4:4','A4:4','G4:4','E4:4',
'F#4:4','G4:16','R:4']

music.play(tune)
```

There is also a **music.get_tempo()** function that will return for you a tuple consisting of the bpm and ticks.

Challenges

1. Improve the way that the accelerometer and buzzer combination works. Try to work out how to make the negative values result in sounds too. You might also want to map the values on to a smaller range of frequencies, preferably actual notes.
2. Use the LED Matrix to make an animation with sound effects. Use the built-in tunes to give you ideas. Start by trying to create images to illustrate one of the tunes of your choice. Then connect a few of them together.
3. Use some of the built-in tunes or your own compositions as background or other sounds for a game you have made.
4. Make your micro:bit play scales to you according to your pre-programmed or in-program choices.
5. Use sheet music or your own composition to play the tune of your choice.
6. Turn your micro:bit into a metronome. Allow the user to choose a value for the bpm and then think about a ticking animation to go along with what you are doing. Think carefully and you can time your animation to go with the tempo of the beats.