Project 5 - Making Music

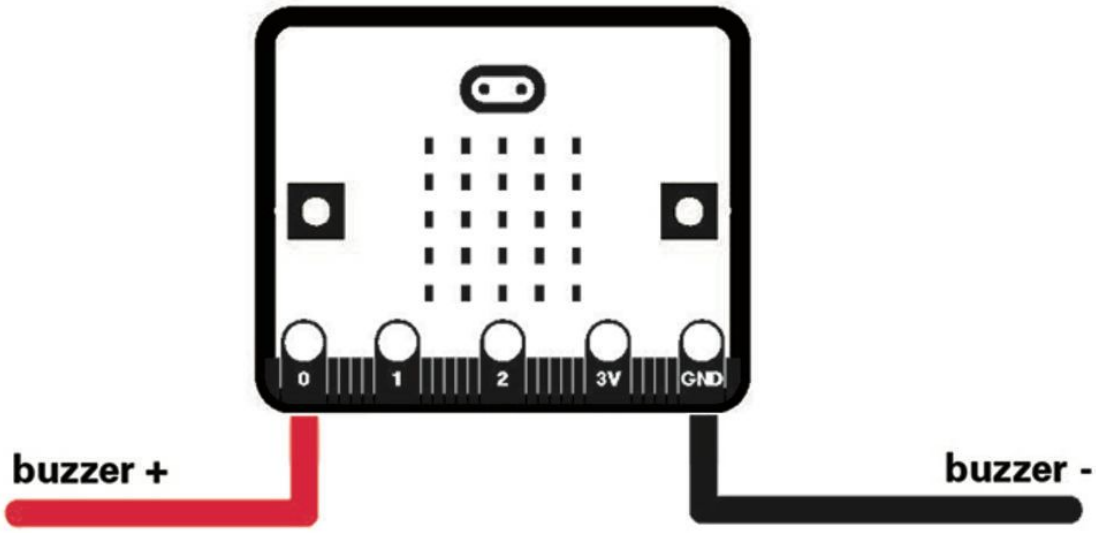
Supplies you should have:

1 micro:bit
1 buzzer
2 alligator clips

For this project, we're going to connect a small buzzer to the the micro:bit and use it to generate some sounds.

Taking one of your alligator clips, connect the **black** wire to the part of the board marked 'GND'.

With the second alligator clip, attach the end of the other wire to the part of the board marked '0'.



Beeps

Now let's write some code. In your Mu editor, type or copy this code and then flash it to the micro:bit by clicking on the 'Flash' button.

```
from microbit import *

pin0.write_digital(1)
```

As soon as your code was flashed to the micro:bit, you should have heard a single beep. 'pin0' is the part of the micro:bit marked with the number 0. Activating it with the `write_digital()` function causes the buzzer to emit a beep.

Remember that 1 is "on" and 0 is "off", so if we called this, it would *deactivate* pin 0:

```
pin0.write_digital(0)
```

Music

Let's try some music. Put this code in your Mu editor and flash it to the micro:bit:

```
import music

music.play(music.NYAN)
```

Did you hear the Nyan-cat theme play?

There are lots of other tunes built into the micro:bit. To see what they are, click on the "REPL" button and type these commands into the text area at the bottom:

```
>>> import music
>>> dir(music)
```

You should see a list with the names of some tunes in UPPER CASE letters:

```
['DADADADUM', 'ENTERTAINER', 'PRELUDE', 'ODE', 'NYAN', 'RINGTONE', 'FUNK',
'BLUES', 'BIRTHDAY', 'WEDDING', 'FUNERAL', 'PUNCHLINE', 'PYTHON', 'BADDY',
'CHASE', 'BA_DING', 'WAWAWAWAA', 'JUMP_UP', 'JUMP_DOWN', 'POWER_UP', ...]
```

Try changing the tune that's played. Which one is your favorite?

New Tunes

Now let's try creating our own melodies!

In music, every note has:

1. a **name** that starts with a letter (like C# or F)
2. an **octave** (how high or low the note is played)
3. a **duration** (how long it lasts)

In micropython, the version of Python we're using to write code for the micro:bit, notes are represented by simple letters. Octaves are represented by a number 0 through 8. Durations are also expressed as numbers (the higher the value, the longer the note lasts).

Each note is expressed as a string of characters, like this:

```
NOTE[octave][:duration]
```

For example, "A1:4" refers to the note named A in octave number 1 to be played for a duration of 4.

To create a melody for the micro:bit, we need to make a list of notes. For example, here's how to make MicroPython play opening of the song "Frere Jaques":

```
import music

tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
"E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]

music.play(tune)
```

The `music.play()` function plays each note in the list, in order. What happens if you change the letter values of some of the notes? What about the duration of some of the notes? Can you write your own music to play on the micro:bit?

Metronome

Okay, let's try one more musical example. Let's turn our micro:bit into a *metronome*.

A metronome is a device that marks exact time by emitting a regular, repeated click or beep - musicians use it to practice playing at an even speed.

```
from microbit import *
import music

tempo = 120
duration = 20
pitch = 1760

while True:
    if button_a.is_pressed() and button_b.is_pressed():
        tempo = 120
    elif button_a.is_pressed():
        tempo += 10
    elif button_b.is_pressed():
        tempo -= 10

    display.show(Image.HEART)
    music.pitch(pitch, duration)
    display.clear()

    wait = ((60 / tempo) * 1000) - duration
    sleep(int(wait))
```

This is a pretty complicated example - let's walk through what the code is doing.

At the very beginning, we're setting three values:

1) tempo (or how fast a thing happens) (120 beats per minute, or two beats per second)
2) duration (or how long a thing happens)
3) pitch (the 'highness' or 'lowness' of a sound)

The part that says "`while True:`" is important. This puts the micro:bit into an *infinite loop*.

Remember when we talked about `while` loops and how it's bad to get stuck in an infinite loop? Well this is a case where it's actually useful - it means the micro:bit is constantly checking for what to do.

In this case, it will always beep at the tempo you set, until you unplug it from its power supply.

Here are some other new things: `button_a.is_pressed()` and `button_b.is_pressed()` are both functions that come with the microbit library.

We get to use them because we wrote `from microbit import *` at the top of our code. These functions return `True` if a button is pressed.

Inside our `while` loop, we're doing a few things:

1) If buttons **A** and **B** are pressed at the same time, we set the tempo to the original number, 120
2) If only button **A** is pressed, we add 10 to the tempo
3) If only button **B** is pressed, we subtract 10 from the tempo

Every time the micro:bit beeps, we are going to display an image to go along with it - in this case, we're using a heart since the beeps are a little like a heartbeat.

`music.pitch()` is how we tell the micro:bit to beep, and then `display.clear()` removes the heart from the screen right after the beep.

```
display.show(Image.HEART)
music.pitch(pitch, duration)
display.clear()
```

Finally, we need to know how often to beep.

```
wait = ((60 / tempo) * 1000) - duration
sleep(int(wait))
```

Using the value of `tempo` (which we can change by pressing buttons), we're just doing a little math to tell the micro:bit how long to pause between beeps.

See if you can make some of these changes to the code:

- Suppose you want to make a Hip Hop song. That music genre often has a tempo of about 90 beats per minute. Can you set the default tempo of the metronome to that?

- What if you're not sure what tempo you want to work at? Can you set the metronome to a random tempo between 90 and 140 BPM if you press one or both of the buttons?

- Maybe you don't want to accidentally set your metronome to be too fast or two slow; can you prevent the tempo from going below 80 or above 180? (**Hint:** Notice how the code checks for the A *and* B buttons being pressed? Python can use 'and' to combine lots of different checks, including the value of variables.)

For more projects you can do with your micro:bit, check out these resources:

- **The MicroPython guide to BBC micro:bit**
  https://www.microbit.co.uk/python-guide

- **MicroPython/micro:bit documentation:**
  http://microbit-micropython.readthedocs.io/en/latest/

- **Micro:bit projects on Instructables:**
  http://www.instructables.com/howto/microbit/