4. Music Maker
MicroPython for micro:bit

1. Music on the micro:bit

If you connect headphones or a loudspeaker to one of the micro:bit's input / output pins, and write code to turn the pin on and off very quickly, you will hear a tone. Put some tones together and you are making music!

In this project, you will use the micro:bit `music` module to experiment with ready-made tunes and compose your own music. Then you will write a program that let's you play along with ***every song ever written*** – well, lots of songs anyway.

> Before starting to code, ask your mentor to show you how to connect headphones or a loudspeaker to your micro:bit.
>
> ***Careful***: connecting it the wrong way could damage the micro:bit.

2. Start a new MicroPython program

Open the Mu editor and click **New** to create a new program file. Write a comment to say what the program does and add **import** statements to include the Python features for the micro:bit and the music module.

> The yellow highlight shows what code to add.

```
# Experiments with micro:bit music

from microbit import *
import music
```

Save the file with a name you will remember. Save the file every few minutes as you work through the project.

3. Choose a tune

The micro:bit has lots of built-in tunes that you can use for projects. They include short tunes (like ring tones) and complete songs.

To try out the tunes, add just one line of code.

> The yellow highlight shows what code to add. Do not change the other code.

```
# Experiments with micro:bit music

from microbit import *
import music

music.play(music.BLUES, loop=True)
```

The **music.play()** statement plays one of the built-in tunes, in this case **music.BLUES**. The **loop=True** argument means the tune will play over and over again.

Flash the code onto the micro:bit and check that you can hear music.

Try out some of the other built-in tunes. To see what's available, follow these instructions ***exactly***.

In the **`music.play()`** statement, delete **`.BLUES`** so your code looks like this.

Now type **`.`** afrer **`music`**, as if you are about to type in a new tune name. The Mu editor will pop up a list of options. Click on any of the options in CAPITAL LETTERS.

Then save your file and flash the micro:bit again. Check that you can hear the new tune.

4. Get composing

When you get bored with the built-in tunes, try writing some music of your own. Change your program like this.

```
# Experiments with micro:bit music

from microbit import *
import music

three_blind_mice = ['e4:4', 'd4:4', 'c4:6', 'r:2']

music.play(three_blind_mice, loop=True)
```

Each note in the tune is a string of characters, like **`'e4:4'`**. We put the notes into a list (in this case, we call it **`three_blind_mice`**) so that we can pass it easily to **`music.play()`**.

Here is how the note string works.

- The first letter is the name of the note – in our tune, we use the notes C, D and E.
- The next number sets how high or low the note is – for example, **`'c1'`** is the same C note as **`'c4'`** but played lower.
- The number after the **`:`** sets how long the note lasts – for example, **`'c4:6'`** will last longer than **`'d4:4'`**. If you don't include a number, MicroPython assumes you mean **`':4'`**. We call this the default value.

Flash the code onto the micro:bit and check you can here the first line of Three Blind Mice.

> Experiment with adding different notes to the tune list. Ask your mentor for help if you get stuck. Can you work out what the note `'r:2'` sounds like?

5. Lots of notes

For the rest of this project, we will be using lots of different notes. Instead of typing them over and over again in the code, we will set up a list that has all the possible notes in it – a bit like having a piano keyboard that we can choose from.

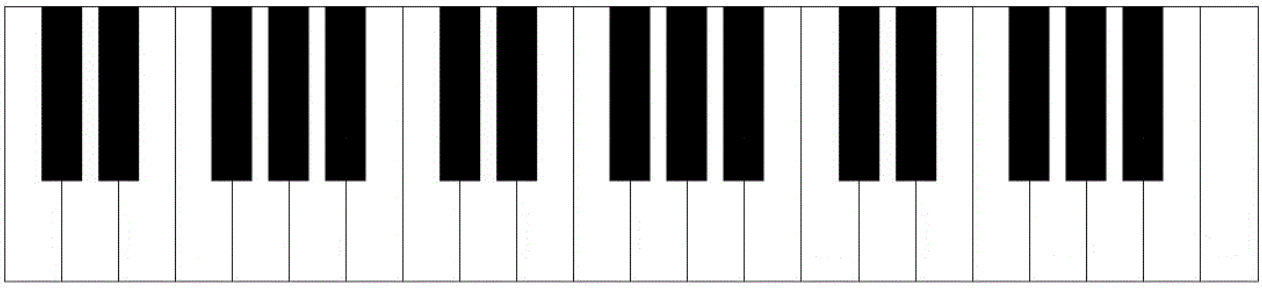
> Your mentor should have the list in a file ready for you to use. Ask them to help you copy and paste it into your program.

Change your code so it looks like this.

```
# Experiments with micro:bit music

from microbit import *
import music

NOTES = ['c3',  'c#3', 'd3',  'd#3',
         'e3',  'f3',  'f#3', 'g3',
         'g#3', 'a3',  'a#3', 'b3',
         'c4',  'c#4', 'd4',  'd#4',
         'e4',  'f4',  'f#4', 'g4',
         'g#4', 'a4',  'a#4', 'b4',
         'c5',  'c#5', 'd5',  'd#5',
         'e5',  'f5',  'f#5', 'g5',
         'g#5', 'a5',  'a#5', 'b5']

for note in NOTES:
    music.play(note)
```

Flash the new code to the micro:bit and listen. It won't sound very exciting but you should check that you hear lots of notes that start low and get higher, and that there are no gaps between them.

If you wanted to play these notes on a piano, this is how much of the keyboard you would need.



Our **NOTES** list includes all these keys, including the sharps and flats on the black keys.

Now that we have a keyboard, let's play something on it. We will show you how to get the code working first, then explain what's going on.

Change your code like this.

```
# Experiments with micro:bit music

from microbit import *
import music

NOTES = ['c3',  'c#3', 'd3',  'd#3',
         'e3',  'f3',  'f#3', 'g3',
         'g#3', 'a3',  'a#3', 'b3',
         'c4',  'c#4', 'd4',  'd#4',
         'e4',  'f4',  'f#4', 'g4',
         'g#4', 'a4',  'a#4', 'b4',
         'c5',  'c#5', 'd5',  'd#5',
         'e5',  'f5',  'f#5', 'g5',
         'g#5', 'a5',  'a#5', 'b5']

MAJOR_CHORD_INTERVALS = [0, 4, 7, 12]
MINOR_CHORD_INTERVALS = [0, 3, 7, 12]


# Main program

root_note_index = NOTES.index('f3')

for interval in MAJOR_CHORD_INTERVALS:
    music.play(NOTES[root_note_index + interval])

sleep(1000)

for interval in MINOR_CHORD_INTERVALS:
    music.play(NOTES[root_note_index + interval])
```

Flash the code to the micro:bit and listen to the output. You should here four notes, then a gap, then four more notes. If you want to listen to them again, press the reset button on the back of the micro:bit.

Now let's look at what the program is doing.

The first four notes are from what musicians call a ***major chord***. In Western music, people usually think that major chords sound happy and positive.

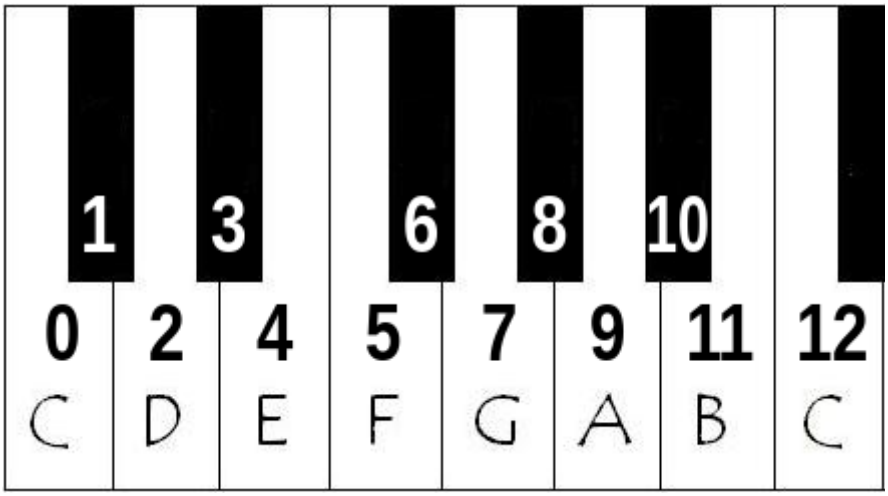
The next four notes are from a ***minor chord***. Compared with a major chord, people often think that minor chords sound sad.

> Music affects each of us in different ways. What did you think when you heard each of the chords?

So what makes a chord major or minor? In the code, you can see a list of numbers that we have called **`MAJOR_CHORD_INTERVALS`**.

```
MAJOR_CHORD_INTERVALS = [0, 4, 7, 12]
```

Intervals show far apart two notes are on a keyboard. Take a look at this close-up of part of our keyboard.



If we start with C, then E is 4 notes away and G is 7 notes away – we count both the black and white keys.

If we start with F, then A is 4 notes away (9 – 5 = 4) and C is 7 notes away (12 – 5 = 7).

The rules of music say that, whatever note we start on, if we add 4, 7 and 12 to it, we will get the other notes we need for a major chord. That's why our list has **`[0, 4, 7, 12]`** in it.

For a minor chord, the intervals have one note different.

```
MINOR_CHORD_INTERVALS = [0, 3, 7, 12]
```

Let's see how the program uses the chord interval lists.

The main program starts by setting which note we are going to use as the root note for the chords. **`NOTES.index('f3')`** looks up the note **`'f3'`** in **`NOTES`** and finds its index – that is its position in the list where the first item is 0, the second is 1, and so on. We name this value **`root_note_index`**.

To play the notes in a chord, **`for interval in MAJOR_CHORD_INTERVALS:`** loops through each interval in turn. The code **`NOTES[root_note_index + interval]`** adds the interval to the root note to give the index of the next note in the chord, and uses that index to look up the actual note string in **`NOTES`**. It is that note string that is passed to **`music.play()`** and we hear the note played.

> Phew, that was a lot of complex music and coding stuff!
>
> Why do we do it this way? Well, using intervals instead of note names makes it easy to use the same code to play major or minor chords based on ***any note on the keyboard***. And it also means we can easily add other types of chords.

6. Simplify the code

Did you notice that, to play the notes in each of the two chords, we used two chunks of similar code?

```
for interval in MAJOR_CHORD_INTERVALS:
    music.play(NOTES[root_note_index + interval])
```

```
for interval in MINOR_CHORD_INTERVALS:
    music.play(NOTES[root_note_index + interval])
```

And we defined our chord intervals using two similar lines of code.

```
MAJOR_CHORD_INTERVALS = [0, 4, 7, 12]
MINOR_CHORD_INTERVALS = [0, 3, 7, 12]
```

If we want to add more chords, or different types of chords, to our program, we will end up with lots of duplicate code.

So, let's define our own function in Python to play the notes of ***any*** chord.

Change your program so it looks like this.

```
# Experiments with micro:bit music

from microbit import *
import music

NOTES = ['c3',  'c#3', 'd3',  'd#3',
         'e3',  'f3',  'f#3', 'g3',
         'g#3', 'a3',  'a#3', 'b3',
         'c4',  'c#4', 'd4',  'd#4',
         'e4',  'f4',  'f#4', 'g4',
         'g#4', 'a4',  'a#4', 'b4',
         'c5',  'c#5', 'd5',  'd#5',
         'e5',  'f5',  'f#5', 'g5',
         'g#5', 'a5',  'a#5', 'b5']

CHORD_INTERVALS = {'major': [0, 4, 7, 4],
                   'minor': [0, 3, 7, 3]}


def play_arpeggio(root_note_index, chord='major'):
    for interval in CHORD_INTERVALS[chord]:
        music.play(NOTES[root_note_index + interval])


# Main program

play_arpeggio(NOTES.index('c4'))

sleep(1000)

play_arpeggio(NOTES.index('e4'), 'minor')
```

To help with writing our function, we are using a Python ***dictionary*** *to* define our chord intervals. Like a dictionary for a human language, it let's us look something up and find out information about it.

So **`CHORD_INTERVALS['major']`** will give the value **`[0, 4, 7, 4]`**, while **`CHORD_INTERVALS['minor']`** will give **`[0, 3, 7, 3]`**. To add other chords, we just add another entry to the dictionary.

To define our function, we use the keyword **`def`**, then give the function a name and (in brackets) show what values we can pass into the function. Our function is called **`play_arpeggio`** and we can pass two values to it:

- **`root_note_index`** selects the starting note from our **`NOTES`** list.
- **`chord`** selects which type of code to play. Putting **`chord='major'`** tells the function to assume a major chord if we don't give a value when calling the function.

To call the function, we put statements like `play_arpeggio(NOTES.index('c4'))` into our main program. Remember, if you want to play a major chord, you don't need to put the second value into the function call.

Flash the code to the micro:bit and listen to the output. You should here four notes, then a gap, then four more notes.

7. Four chords go a long way

It so happens that many pop songs have been written using just four chords: three major chords and one minor chord. Whichever note you choose on the keyboard (so whichever key you are playing in), the four chords are always the same distances apart.

Let's change the program so that it plays these four chords. We will use the micro:bit's accelerometer to control which chord to play, by tipping the micro:bit down in different directions.

Here are details of these four amazing chords.

- The first chord is called the `I` chord (that's capital Roman numeral for 1). It is a major chord ***starting on*** our root note.
- The next chord is the `V` chord (capital Roman numeral for 5). It is a major chord ***seven notes up*** from the root note.
- The next chord is the `vi` chord (lower case Roman numerals for 6). It is a minor chord ***nine notes up*** from the root note.
- The last chord is the `IV` chord (capital Roman numerals for 4). It is a major chord ***five notes up*** from the root note.

Our `play_arpeggio()` function makes it easy to play each of these chords by just adding a number to our root note index when we call the function. We will use the x and y values from the accelerometer to detect which corner of the micro:bit is tilted down.

Change the main program so it looks like this.

> The **...** symbol means there is code that we are not showing, just to keep the length of the worksheet down. DO NOT DELETE THE CODE THAT IS NOT SHOWN.

```
…

# Main program

root = NOTES.index('c4')
music.set_tempo(bpm=210)

while True:
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    if x < 0:
        if y > 0:
            # "Front left down" plays I chord
            play_arpeggio(root)
        else:
            # "Back left down" plays V chord
            play_arpeggio(root + 7)
    else:
        if y < 0:
            # "Back right down" plays vi chord
            play_arpeggio(root + 9, 'minor')
        else:
            # "Front right down" plays IV chord
            play_arpeggio(root + 5)
```

Flash the code to the micro:bit and listen to the output while you tilt it towards each corner in turn. You should here the four notes of each chord being played quickly.

Try playing these two sequences of tilts.

- FRONT LEFT – BACK LEFT – FRONT RIGHT – BACK LEFT – repeat
- FRONT LEFT – BACK LEFT – BACK RIGHT - FRONT RIGHT – repeat

Can you find other sequences that sound good?

8. Add some graphics

Our program sounds good but it would look better if it used the micro:bit display while it is playing music.

Change the main program so that it shows an instruction to the user at the start, and then points to the tilted corner while it's playing.

```
…

# Main program

display.scroll("Tilt to play music", 75)

root = NOTES.index('c4')
music.set_tempo(bpm=210)

while True:
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    if x < 0:
        if y > 0:
            # "Front left down" plays I chord
            display.show(Image.ARROW_SW)
            play_arpeggio(root, 'major')
        else:
            # "Back left down" plays V chord
            display.show(Image.ARROW_NW)
            play_arpeggio(root + 7, 'major')
    else:
        if y < 0:
            # "Back right down" plays vi chord
            display.show(Image.ARROW_NE)
            play_arpeggio(root + 9, 'minor')
        else:
            # "Front right down" plays IV chord
            display.show(Image.ARROW_SE)
            play_arpeggio(root + 5, 'major')
```

Flash the code to the micro:bit and listen to the output while you tilt it towards each corner in turn. Check that the display points to the correct corner.

9. Congratulations

Well done! You have written your first MicroPython music program. You have learned how to use built-in tunes, write your own tunes and chord sequences, and use the accelerometer to control your musical creation.

If you need to check anything, the complete listing is shown on the next page.

Here are some ideas for follow-up experiments.

- Change the root note by putting a different note into the statement `root = NOTES.index('c4')`. Play some sequences and see how it sounds.

- Add some code inside the `while True:` loop to check if Button A is pressed. If it is pressed, set the root note to `'g4'`, else set it to `'c4'`. Now while you are playing a sequence, you can change key by pressing the button, and change back by releasing it. See how this sounds.

```
# Experiments with micro:bit music

from microbit import *
import music

NOTES = ['c3',  'c#3', 'd3',  'd#3',
         'e3',  'f3',  'f#3', 'g3',
         'g#3', 'a3',  'a#3', 'b3',
         'c4',  'c#4', 'd4',  'd#4',
         'e4',  'f4',  'f#4', 'g4',
         'g#4', 'a4',  'a#4', 'b4',
         'c5',  'c#5', 'd5',  'd#5',
         'e5',  'f5',  'f#5', 'g5',
         'g#5', 'a5',  'a#5', 'b5']

CHORD_INTERVALS = {'major': [0, 4, 7, 4],
                   'minor': [0, 3, 7, 3]}


def play_arpeggio(root_note_index, chord='major'):
    for interval in CHORD_INTERVALS[chord]:
        music.play(NOTES[root_note_index + interval])


# Main program

display.scroll("Tilt to play music", 75)

root = NOTES.index('c4')
music.set_tempo(bpm=210)

while True:
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    if x < 0:
        if y > 0:
            # "Front left down" plays I chord
            display.show(Image.ARROW_SW)
            play_arpeggio(root, 'major')
        else:
            # "Back left down" plays V chord
            display.show(Image.ARROW_NW)
            play_arpeggio(root + 7, 'major')
    else:
        if y < 0:
            # "Back right down" plays vi chord
            display.show(Image.ARROW_NE)
            play_arpeggio(root + 9, 'minor')
        else:
            # "Front right down" plays IV chord
            display.show(Image.ARROW_SE)
            play_arpeggio(root + 5, 'major')
```