LISTening Exercises

Christopher Dalziel

November–December 2024

1 Introduction

Debugging systems can be difficult, as errors are very often subtle and not easily spotted. This can be especially true when writing programs using lists and counters, mistakes and misunderstandings about list indexing can cause code to fail in ways that are very non-obvious.

These exercises are designed to have code which will take some time to understand, and you will possibly need to refer to the documentation for relevant concepts. Links to the documentation will be provided where useful. Don't worry if you don't have any musical experience; anything it might be useful (or interesting!) to know has been provided to you. All that's required of you musically is the ability to listen.

2 Getting Started

All of the code needed for this exercise is provided in separate folders, with a folder dedicated to each snippet. All code should be run in the Python micro:bit online editor.

To start an exercise you should open the associated folder and select all of the python files within. The files can be opened in the editor by opening the project panel using the project button in the bottom left, and then dragging the files over the newly open panel.

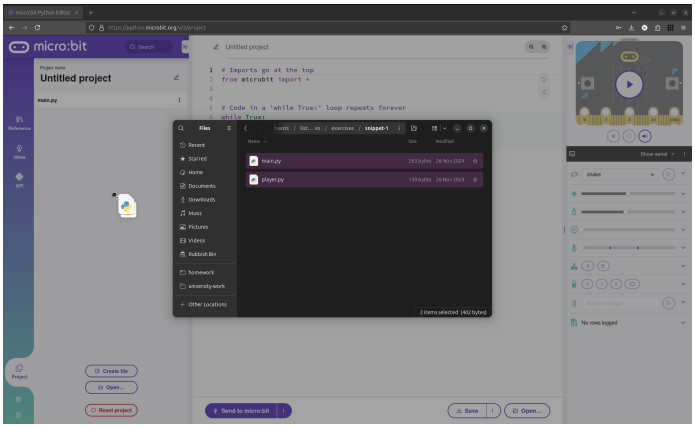

Figure 1: Dragging the files to open them in the editor

By hitting confirm you will have opened the files in the editor and are ready to begin. When you are ready to save your changes you can open the three dots beside the save button and click "Save Python script".

3 Background

3.1 Music Notes

In (western) music theory there are twelve notes:

- the **natural** notes, labelled A through G;
- and then some of those notes labelled with "**accidentals**" - sharps ($\sharp$) and flats ($\flat$)

A	A#/B♭	B	C	C♯/D♭	D	D♯/E♭	E	F	F♯/G♭	G	G♯/A♭

Figure 2: All twelve music notes, called the **chromatic** scale

4 Scales, Chords, and List Indexing

A **scale** is a collection of music notes, which are typically selected to sound good together. The most well known scales are the major and minor scales

For this section we will be using the C major scale, which consists of the following notes:

$$C,\ D,\ E,\ F,\ G,\ A,\ B,\ C$$

Note that the second C is played at a higher pitch than the first, so that the scale can be repeated at a higher pitch

4.1 Snippet 1 - Power Chords

A **chord** is a collection of notes played either close together or all at once. The simplest kind of chord is the so called "power" chord, which consists of a **root** (the *first* note), the *fifth*, and an **octave** (the *eighth* note).

When played in sequence, these notes rise in pitch and should sound pleasing to the human ear.

Finish the following snippet using list indexing and concatenation to play the three notes of a C major power chord in order:

```
c_major = ["c", "d", "e", "f", "g", "a", "b", "c"]

# Todo: finish c_major_power
c_major_power = c_major[0] + ???

# Play the notes in c_major_power
for note in c_major_power:
    # Play the given note at the 4th octave
    play(note, 4)
```

When you've finished this exercise, what do you notice is wrong with the provided code when you run it?

4.2 Snippet 2 - Playing Scales

An **octave** is the gap between a given note (say C_4) and that same note at a higher or lower pitch (C_3 and C_5). In the last exercise the `play()` function defaulted to playing the fourth octave.

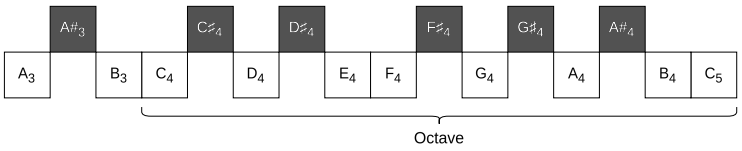

Figure 3: An octave (from C_4 to C_5)

The following code should play through a C major scale, starting from C_4 and ending at C_5. Implement the conditional to increment the octave.

```
c_major = ["c", "d", "e", "f", "g", "a", "b", "c"]
octave = 4

# Play each note in the c_major scale begining at C 4
for index, note in enumerate(c_major):
    # Todo: implement conditional
    if ??? and ???:
        octave += 1
    play(note, octave)
```

> If you've never seen the function `enumerate()` before, have a look at the documentation.

Looking at the code you've written and at Figure 3, what could we change about the scale we input that would make our code incorrect?

5 Constructing Scales with the Circle of Fifths

The **circle of fifths** is an important concept in classical music theory.

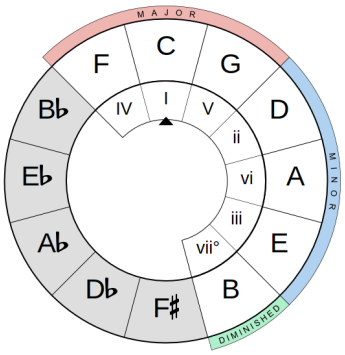

Figure 4: An interactive circle of fifths set to C major

Its main use for our purposes is telling us what notes are in a scale. On the example above we can see that the C major scale contains the notes A through G with no sharp or flat notes

Our version of the circle will be implemented as a list. We would represent Figure 6 as follows, with the root note C (indicated on the circle with a small triangle) as the first element of the list:

Figure 5: List of fifths set to C major

5.1 Snippet 3 - Changing Roots

Thusfar we've been using C major as our scale of choice, but suppose we wanted to play a piece in A major instead - how would we know what notes were in the scale?

The answer is that we could "rotate" the notes in the circle of fifths so that the little triangle was pointing at A, which would be a rotation by 3 notes clockwise ($C \to G \to D \to A$).

You can try this on the interactive circle of fifths by setting the "tonic" column on the left to A, it should look like the following (without having been literally rotated - that's editing magic):

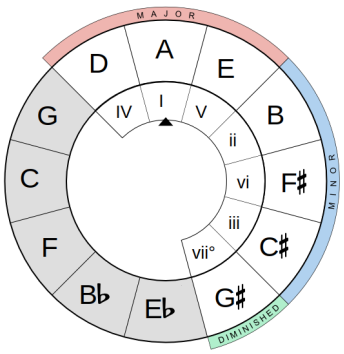

Figure 6: The circle of fifths set to A major

We can implement this idea for our lists using "cycling", an operation which changes the order of our list so that the first n elements (where n is a whole number) get moved to the end of the list.

$$\texttt{cycle}([1,2,3,4,5,6,7,8],3) \rightarrow [4,5,6,7,8,1,2,3]$$

This code snippet should take a list, and return that list with a new first element such that it has been cycled to the left by n steps. Use list slicing to complete the cycle function.

```
list_of_fifths = ["c", "g", "d", "a", "e", "b", "f#", "c#",
↪   "g#", "d#", "a#", "f"]

# Cycles n elements through the provided list
def cycle(list, n):
    # Todo: get the first n elements of list
    start = ???
    # Todo: get the rest of the list
    end = ???

    cycled = end + start
    return cycled
```

How does this cycling function work if we give a negative number for n? Is this what you expected? How can we interpret this in terms of rotation?

5.2 Snippet 4 - Major Construction

When we have the circle of fifths rotated to sit on a root, the notes of the major scale always sit in the same way relative to the root. We can therefore construct a major scale by rotating the circle to the root and taking the notes in those positions.

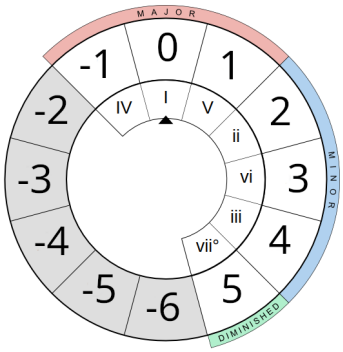

Figure 7: An indexed circle of fifths for major scales

Using this information, complete the following snippet using the `index()` method of lists, list slicing, and negative indexing.

```
list_of_fifths = ["c", "g", "d", "a", "e", "b", "f#", "c#",
 ↪   "g#", "d#", "a#", "f"]

# Get the major scale for the provided root
def get_major_scale(root):
    # Check that the input is a valid music note
    if not is_music_note(root):
        raise ValueError("Invalid root note")

    # Todo: get the list of fifths set to the root
    cycled = cycle(list_of_fifths, ???)

    # Todo: Get the first 6 elements of the list
    forwards = cycled[:???]
    # Todo: Get the last element of the list
    backwards = cycled[???]

    # Put the notes in alphabetical order
    notes = sorted(forwards + backwards)
```

```
    # Return the notes, starting at the root
    return cycled(notes, ???)
```

5.3 Snippet 5 - Relatively Minor

Other scales can be constructed by a similar process (you can see this on the interactive circle by changing the "mode" column). For example, a minor scale would be from -4 to 2 on Figure 7 instead of -1 to 5.

A major scale's **relative minor** is a minor scale that contains the same notes as that major scale, but has a different root that makes it "feel" minor.

Converting to minor from major (as shown above) involves subtracting three from the indices on each side, so to get the relative minor we take our root and see what note is three positions to the right.

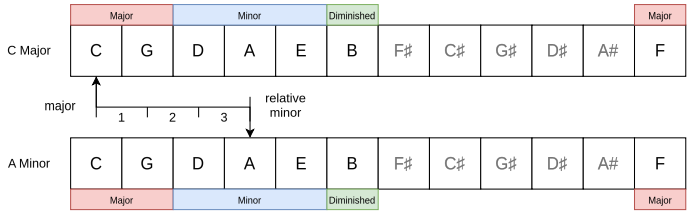

Figure 8: The relative minor of C major is A minor

The following snippet contains a function which should obtain the relative minor, but it throws an error for certain inputs. What inputs does it break for, how might we fix this?

```
# Get the relative minor for the provided root note
def get_relative_minor(root):
    # Check that the input is a valid music note
    if not is_music_note(root):
        raise ValueError("Invalid root note")

    # The relative minor is three ahead of the given note
    rm_idx = list_of_fifths.index(root) + 3
    relative_minor = list_of_fifths[rm_idx]

    return relative_minor
```

6 Putting it all Together

That's the end of the lesson! Congratulations for getting this far. If you've still got some time, there's one final activity that you can do - if you've completed the exercises thus-far this one shouldn't be too difficult :)

Using what you've done so far, write a program that plays any requested major scale followed by its relative minor scale. Feel free to copy and edit code from the previous exercises to complete this task.