Simple Music Note Transposition With Transposy

Dustin Chung

Project: Feb 2019 — Writeup: Feb 2019

Overview

Transposy is a simple utility written in Python that musically transposes inputs of notes. It has the ability to transpose by number of steps, to transpose relative to a given starting note, and to return all transposition combinations that avoid undesired notes. It does not, however, take into account octaves, note timings, and key signatures.

1 Introduction

Transposition is a very important skill in music: by changing the absolute pitches of the notes in a melody without changing their intervals, the tune becomes accessible to different instruments where certain notes may not be accessible. For example, a certain song may be written at too high a pitch for a singer to sing well. Using transposition, the pitch of the song can be lowered.

The method of transposition discussed in this document works by stepping up or down semitones repeatedly. If all notes in a given input are stepped in the same direction with the same interval, then it can be considered transposed.

2 Methods & Implementation

A Python class was written, called Transposer. It initializes with the input of notes, which must be entered in the format of an array of arrays, with each inner array containing one or more musical notes in their letter notation. Single-membered inner arrays correspond to single notes, whilst arrays with more than one note correspond to chords.

2.1 Initialization

2.1.1 Rules for input of notes

The letter notation is given as a string and follows simple rules:

- All notes to be expressed in capital letters.
- All non-natural notes are expressed as sharps (e.g. Bb = A#)
- All notes are expressed by themselves (e.g. NOT ["AB"], but ["A", "B"])

2.1.2 Initializer inputs

As outlined above, an array of arrays must be provided to the Transposer constructor. The inner arrays can be on any size, so long as there are no deeper array levels within them. The table below shows examples of valid and invalid inputs.

Example Input	Valid?	Reason for Rejection
[[“A”], [“B”], [“C”], [“D”]]	Yes	
[[“A”], [“B”,“C”], [“D”]]	Yes	
[[“A”, “B”, “C”, “D”]]	Yes	
[“A”, [“B”], [“C”], [“D”]]	No	All inner elements must be in an array (”A” to [”A”])
[[“A”], [], [False], [2]]	No	All notes must be expressed as strings.
“A”	No	Input must be array of arrays ([[”A”]])

Table 1: Examples of valid and invalid inputs for the initializer.

3 Use, Design, and Methods

The Transposer class contains a hash (Python dict) that holds all possible notes in the chromatic scale as keys. The values of these keys is an array containing two elements: the note one semitone higher, and lower than the corresponding key note:

```
keymap = {"C":["B", "C#"], "C#":["C", "D"], "D":["C#", "D#"], "D#
    ":["D", "E"], "E":["D#", "F"], "F":["E", "F#"], "F#":["F", "G"
    ], "G":["F#", "G#"], "G#":["G", "A"], "A":["G#", "A#"], "A#":[
    "A", "B"], "B":["A#", "C"]}
```

This approach of mapping out keys relative to each allows us to traverse all the notes in the chromatic scale with some efficiency, through repeated access of the keymap hash.

3.0.1 The __step function

The private **__step** method is the main workhorse of the Transposer class. It takes a note as an argument, as well as a boolean which denotes the direction in which to operate. It then returns the next semitone down/up:

```
def __step(self, note, bool_step_up):
neighbours = self.keymap[f"{note}"]

if bool_step_up:
return neighbours[1]
else:
return neighbours[0]
```

This takes advantage of the standardization given to the keymap hash, whereby the first element of any associated array is a step down, and the second element is a step up. This method is the foundation on which other methods are built. For the following sections that describe the transposy methods, assume a preliminary setup as described in the code block below:

```
import transposy
notes = [["A", "B"], ["C#"], ["D"]]
t = transposy.Transposer(notes)
```

This will allow for easier following of the examples.

3.0.2 The transpose_by function

The first utility of transposy is the **transpose_by** method. This method takes an integer and boolean as arguments, which dictate the number of semitones to step and the direction to step in respectively. To achieve this, it calls the **__step** method repeatedly for each note that was initialized, storing the result each time and using that for the next loop iteration.

```
t.transpose_by(1) # [["A#", "C"], ["D"], ["D#"]]
t.transpose_by(4, False) # [["F", "G"], ["A"], ["A#"]]
```

3.0.3 The start_on_note function

This method takes a note as an argument, and transposes the first note in the initial input to that note. It then calculates the number of steps that were taken to achieve that transposition, and then calls the **__step** method that many times on all the other notes. Since transposy does not take into account octaves, this method always transposes upwards.

```
t.start_on_note("D") # [["D", "E"], ["F#"], ["G"]]
```

3.0.4 The avoid function

When called with an array of notes as an argument, this method returns an array of arrays, each containing the results of a transposition that does not contain any of the notes in the argument. This allows the user to see all possible transpositions that avoid notes of their choosing.

```
t.avoid(["C", "B"])
# Returns:
# [
#         [["C#", "D#"], ["F"], ["F#"]],
#          [["D", "E"], ["F#"], ["G"]],
#          [["D#", "F"], ["G"], ["G#"]],
#          [["E", "F#"], ["G#"], ["A"]],
#          [["F", "G"], ["A"], ["A#"]]
# ]
```

4 Conclusion

A simple Python class was devised that allowed musical notes to be transposed through different methods, such as direct transposition, setting the starting note and transposing from there, and getting all possible transpositions that avoid a set of undesired input notes. Limitations of the program include it not accounting for octaves, musical timing, and key signatures. It is the opinion of the author that this simple program will find use with amateur musicians who do not have the skill of transposition fully learned.