Software Design

Audible-Plot

Daniel Birket

March 30, 2023

Abstract

This document describes the software design for audible-plot.

1 Classes

1.1 Class Pitch

This class represents a pitch or frequency and converts between three pitch and frequency formats:

- Integer MIDI note number, 0 to 127
- String note name and octave: "C1" to "C9" including sharps and flats like "F#4" and "Eb3"
- Frequency, in Hertz

The internal representation of the pitch will be a MIDI number between 12 and 127, stored as a ‘float‘ instead of ‘integer‘. The class must extend the class "object", not "float".

The frequency corresponding to any pitch can be computed with the formula:

$f = 440 * 2^{\text{m-69}}$ Where:

- f: frequency in hertz
- m: MIDI note number, as a float

or in python:

```
freq = 440.0 * 2.0^(midi - 69.0)*
```

In the other direction, the float midi note may be calculated by inverting the same equation:

$m = log_2(f/440) + 69$

or in python:

```
midi = math.log2(freq / 440.0) + 69.0
```

When converting from a string to a MIDI note, the string should be of the format "C0" or "C#0", with the capital note letter name as the first character, the digit of the octave as the last digit and an optional "#", "" (Unicode "MUSIC SHARP SIGN"), or "+" (PLUS), or a "b", "" (Unicode "MUSIC FLAT SIGN"), or "-" (HYPHEN) character between them.

The octave digit must be 0 to 9. Note that some of octave 0 is below the range of human hearing. The base MIDI number of each octave is:

Octave	MIDI
0	12
1	24
2	36
3	48
4	60
5	72
6	84
7	96
8	108
9	120

The base MIDI note of each octave may be computed as:

$base = 12 * (octave + 1)$

The octave of a MIDI note may be computed as:

$octave = int((midi - 12)/12)$

This formula produces:

MIDI	Octave
12	0
23	0
24	1
35	1
36	2
119	8
120	9

The note letter must be capital A to G. These notes map to an MIDI note offset from the base of the octave as follows:

Note	Offset
C	0
D	2
E	4
F	5
G	7
A	9
B	11

The accidental (sharp or flat) must be one of #, , or + for sharp or b, , or - for flat. These characters add or subtract one from the MIDI note number. Double sharps and double flats, if implemented, add or subtract 2 midi notes. Unusual musical combinations, like "E#4" or "Cb4", which are the same as "F4" and "B3", respectively, are permitted.

The MIDI number is the sum of the octave base, the note offset and the +1 or -1 for a sharp or flat, if any

If the string is not of this format, then the conversion should check if the string is a valid integer or float number in the range greater than or equal to 12 and less than 128 and interpret a number in that range as a float MIDI number.

If the string is empty, the pitch should initialize to "A4", a 440 Hz 'A' note.

Finally, if the string is a valid integer or float greater than or equal to 128 and less than or equal to 22000, it should interpret the value as a frequency.

If the initializer is already an integer or float, it should perform the same range checks.

If the string is neither a valid note name or a number in one of the valid ranges for a MIDI note or an audible frequency, it should raise a ValueError.

If the initializer is not a string, integer or float, or another Pitch, it should raise a TypeError.

Except for the init initializer, which detects the format of the pitch, Pitch should provide a property() interface to note, midi and freq fields, with getter and setter methods and doc strings.

When converting from a float MIDI number to a note string, the program will return a 2-item tuple. The first item of the tuple must be a note string with note letter, optional sharp sign (using the shift-3 sharp not the unicode "MUSIC SHARP SIGN", and the octave digit. The note string represents the integer part of the float MIDI number. The second item of the tuple must be the fractional part of the MIDI number, a float number greater than or equal to 0.0 and less than 1.0.

The init initializer should also accept this tuple.

The class Pitch must have the following interface:

```
from typing import Any

class Pitch(object):

    def __init__(self, value : Any = None) -> None:
        pass
        if isinstance(value, tuple) or isinstance(value, str):
            self.note = value
        elif isinstance(value, float) or isinstance(value, int):
            if value < 128.0:
                self.midi = value
            else:
                self.freq = value
        else:
            raise TypeError("value must be tuple, string, float or int.")

        # if isinstance(value, int) or isinstance(value, float)
        midi = 0

    # Getter and Setter of midi (same as internal _midi)
    @property
    def midi(self) -> float:
        "float MIDI number of pitch."
        return self._midi
```

```
@midi.setter
def midi(self, m: float) -> None:
    assert isinstance(m, int) or isinstance(m, float)
    # assert valid 12 to 127 midi number here
    self._midi = float(m)

@midi.deleter
def midi(self) -> None:
    self._midi = 69.0         # "A4", 440 Hz 'A'


# Getter and Setter of frequency
@property
def freq(self) -> float:
    "float frequence of pitch in hertz."
    return 0.0

@freq.setter
def freq(self, f: float) -> None:
    assert isinstance(f, int) or isinstance(f, float)
    # assert audible frequency range here
    self._midi = 0.0

@freq.deleter
def freq(self) -> None:
    del self.midi


# Getter and Setter of note string
@property
def note(self) -> str:
    "note string of pitch."
    return "A4"

@note.setter
def note(self, n) -> None:
    if isinstance(n, tuple):
        note_str, note_bend = n # unpack tuple
    elif isinstance(n, str):
        note_str = n
        note_bend = 0.0
    assert isinstance(note_str, str), "Note string must be a string like 'A4'."
    assert isinstance(note_bend, float), "Note bend must be a float like 0.0."
    assert 0.0 <= note_bend < 1.0, "Note bend must 0.0 or between 0.0 and 1.0."

    self._midi = 0.0
```

```
    @note.deleter
    def note(self) -> None:
        del self.midi
```

1.2 Class ScaleToPitch

This class implements scaling between a range of data values and a range of pitches (not frequencies). Frequency doubles for each octave of increase of pitch, but people perceive pitch (musical notes) as a linear increase, not an exponential increase, so it is more appropriate to scale data to pitch than to frequency.