PythonEuterpea: Basic Musical Structures

Field names for classes below are accessed with the names shown in the examples. For example, if x is a Note, one would access its pitch with `x.pitch`.

Make a single note: `x = Note(pitch, dur, vol, params)`
  pitch is a required argument. Default values: dur=0.25, vol=100, params=None
Make a single rest: `x = Rest(dur, params)`
  dur is a required argument. Default values: params=None

The params argument above is completely optional in both cases and can be any type. Currently, params are not addressed in any of the other functions in PythonEuterpea quickly but are available for possible use in future development.

Put two things in sequence: `x = Seq(left, right)`
Put two things in parallel: `x = Par(top, bottom)`

Insert a modifier node: `x = Modify(mod, tree)`
Possible modifiers:

`Tempo(value)`	Symbolic scaling factor (does not alter tree)
`Instrument(value)`	Assign an instrument by name or patch number.

Outer music wrapper (optional): `x = Music(tree, bpm)`
The bpm argument is 120 beats per minute by default. If set otherwise, it provides a global tempo context for interpreting the rest of the tree. `Music` is usually intended to be a tree root, not a subtree of another structure. Most situations don't actually require use of this class.

Write a MIDI file: `musicToMidi(filename, musicValue)`
musicValue can be: `Note, Rest, Seq, Par, Modify,` or `Music`

Duration constants
These are based on a 4/4 metrical structure and assume there is some global bpm. Unless specified via a Tempo modifier or with the bpm field of Music, the assumed tempo is 120bpm.

```
WN = 1.0          # whole note = one measure in 4/4
DHN = 0.75        # dotted half
HN = 0.5          # half note
DQN = 0.375       # dotted quarter
QN = 0.25         # quarter note
DEN = 0.1875      # dotted eighth
EN = 0.125        # eighth note
DSN = 0.09375     # dotted sixteenth
SN = 0.0625       # sixteenth note
DTN = 0.046875    # dotted thirtysecond
TN = 0.03125      # thirtysecond note
```

Python Euterpea: Creating Music with Pitch Lists

Functions for making music structures from numbers and lists of numbers

Data structure note: the following functions do not use the Music class as a wrapper. The returned results will be at the level of Seq and Par. Default values for arguments are: defDur=0.25 and defVol=100.

Make a single note: `pitchToNote(pitchNumber)`

Sequential composition of a list of pitch numbers (melody):
```
pitchListToMusic(pitchList, defDur, defVol)
```

Parallel composition of a list of pitch numbers (chord):
```
pitchListToChord(pitchList, defDur, defVol)
```

Convert nested lists of pitch numbers into a chord progression (each inner list is a chord):
```
chordListToMusic(pitchList)
```

Sequentially compost pairs of pitch and duration of the format [(p1,d1), (p2,d2), …]:
```
pdPairsToMusic(pitchDurPairList, defVol)
```

Parallel composition for pairs of pitch and duration (same format as above):
```
pdPairsToChord(pitchDurPairList, defVol)
```

Getting a pitch list back from a music value:
```
getPitches(musicStructure)
```
Note: getPitches() is really only a reversal of pitchListToMusic. As a result, the order of the returned list is only meaningful if the input structure is a melody (all Seq constructors and Notes/Rests as leaves) or a single chord (all Par constructors and Notes/Rests as leaves). Any other structural information in the tree is lost when converting back to a list representation. So, in a mixed tree with both Seq and Par constructors, you will know what pitches are in the tree, but won't retain information on their timing. Duration information is similarly lost.

PythonEuterpea: Manipulating Music

The following functions work over musical trees and are intended for use with the Note, Rest, Seq, Par, Modify, and Music constructor.

Get the overall duration: x = dur(value)

Put a list of music structures together in parallel*: `x = chord(listOfValues)`

Put a list of music structures together in sequence*: `x = line(listOfValue)`

In-place map a function over Note leaves: `mMap(functionForNotes, musicStructure)`

In-place map a function over Notes and Rests: `mMapAll(function, musicStructure)`

In-place transpose by amt number of halfsteps: `transpose(musicStructure, amt)`

Temporally reverse a musical structure (write it backwards): `reverse(musicStructure)`

Musical inversion (over pitches only): `invert(musicStructure)`

Musical inversion (over pitches only) around a particular pitch:
`invertAt(musicStructure, referencePitch)`

Repeat a phrase n times:** `times(musicStructure, n)`

Only preserve the first n measures: `cut(musicStructure, n)`

Remove the first n measures: `remove(musicStructure, n)`

Change all volumes to the same value: `setVolume(musicStructure, vol)`

Scale all volumes by a factor : `setVolume(musicStructure, vol)`
`Warning: this does not convert volumes back to an integer!`

Scale all volumes by a factor and round to int : `setVolumeInt(musicStructure, vol)`

Add a constant to all volumes: `adjustVolumes(musicStructure, vol)`

* These functions return a new pointer but do NOT deepcopy the arguments. Any pointers to inner structures will remain unaffected.

** These functions deep copy all arguments before using them.