
Making MIDI Music with Python

— An Introduction to Music Theory —

What we'll cover

Music theory

Notes and rhythms as strings

Notes and rhythms as Python objects

Using Pyknon to generate MIDI files

whoami

Evan Palmer

BM in Music Education, Portland State University (“Choir Nerd”)

Full Stack Web Development, PDX Code Guild

Likes: Python, Django, Javascript, open source, and pop hits of the 15th c.

You'll need...

A way to play MIDI files

Pyknon (github.com/palmerrev/pyknon)

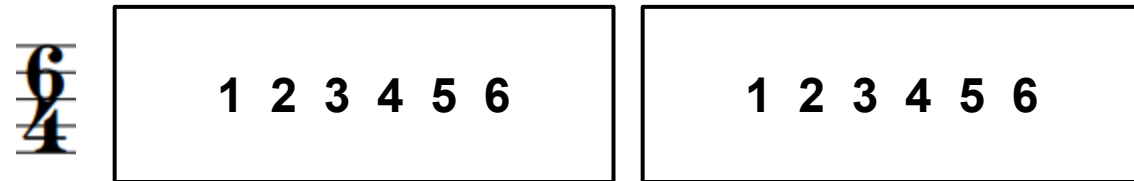
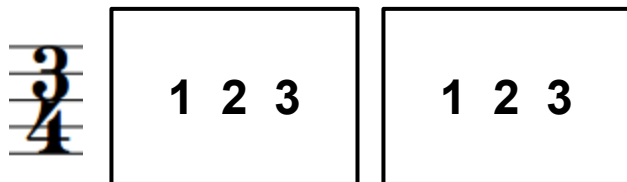
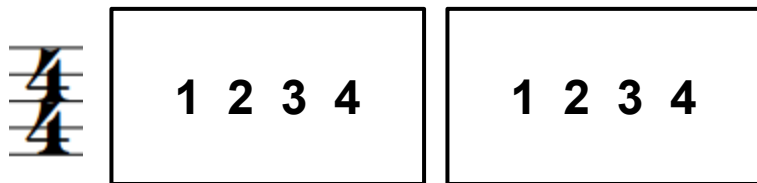
Extra goodies (github.com/palmerrev/pydx15-music)

Music Theory







Rhythm and Meter






“the beat”



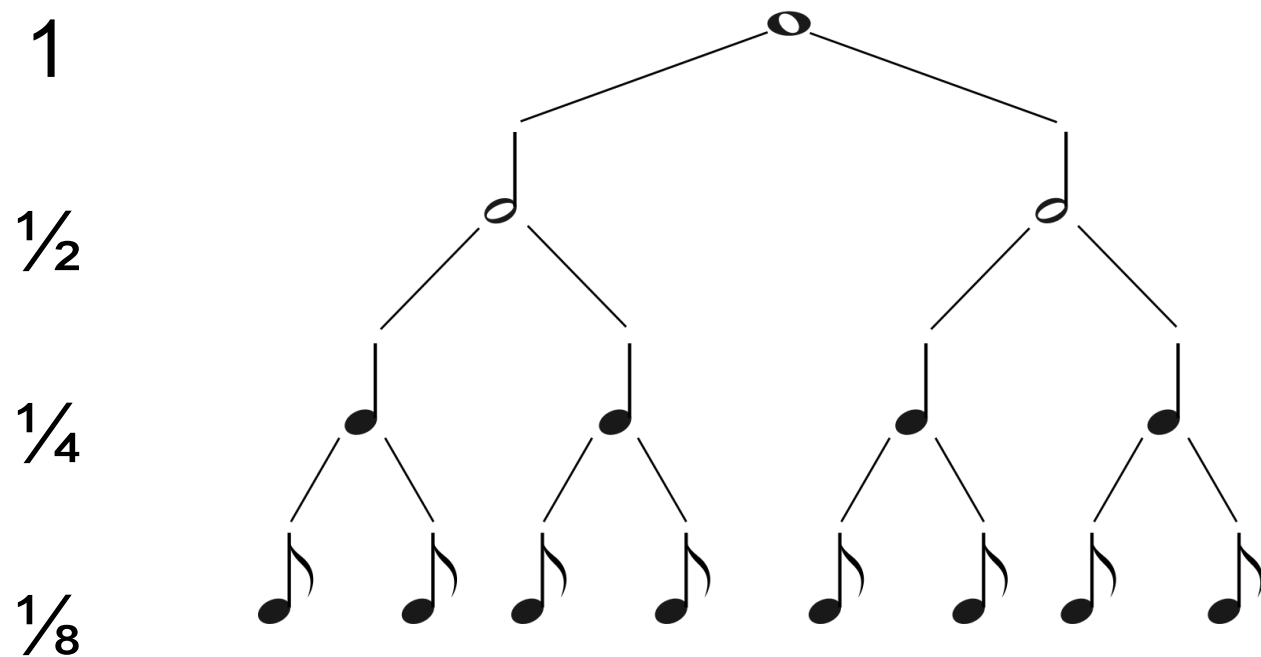
Rhythm and Meter, detail

1	2	3	4
			
1/4	1/4	1/4	1/4
.25	.25	.25	.25

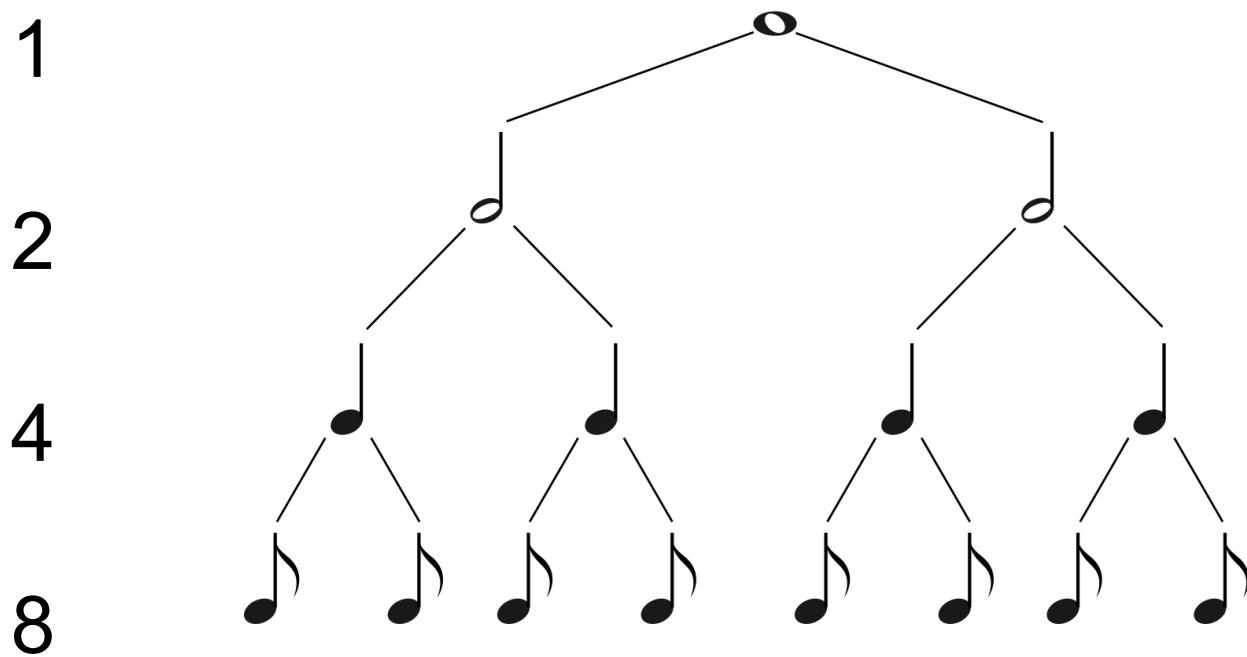
Notes of different lengths

	whole note	1
	half note	$\frac{1}{2}$
	quarter note	$\frac{1}{4}$
	eighth note	$\frac{1}{8}$
	sixteenth note	$\frac{1}{16}$

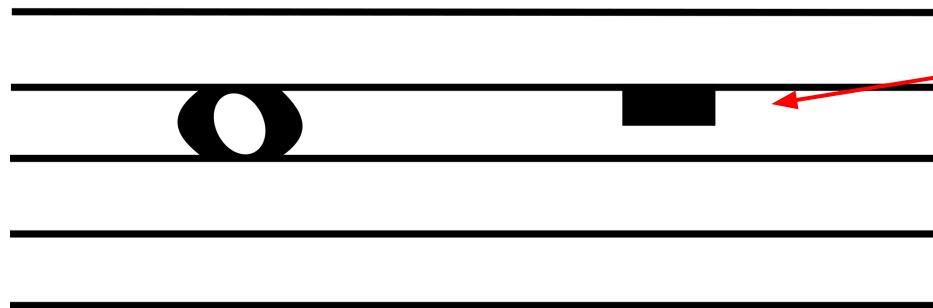
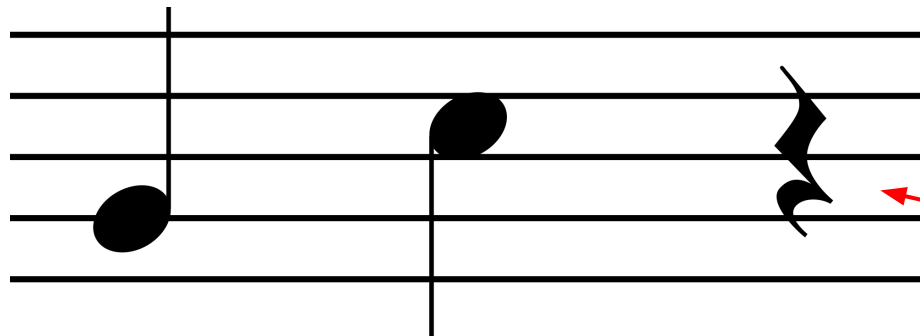
Notes of different lengths



Notes of different lengths



Silence (Rest)



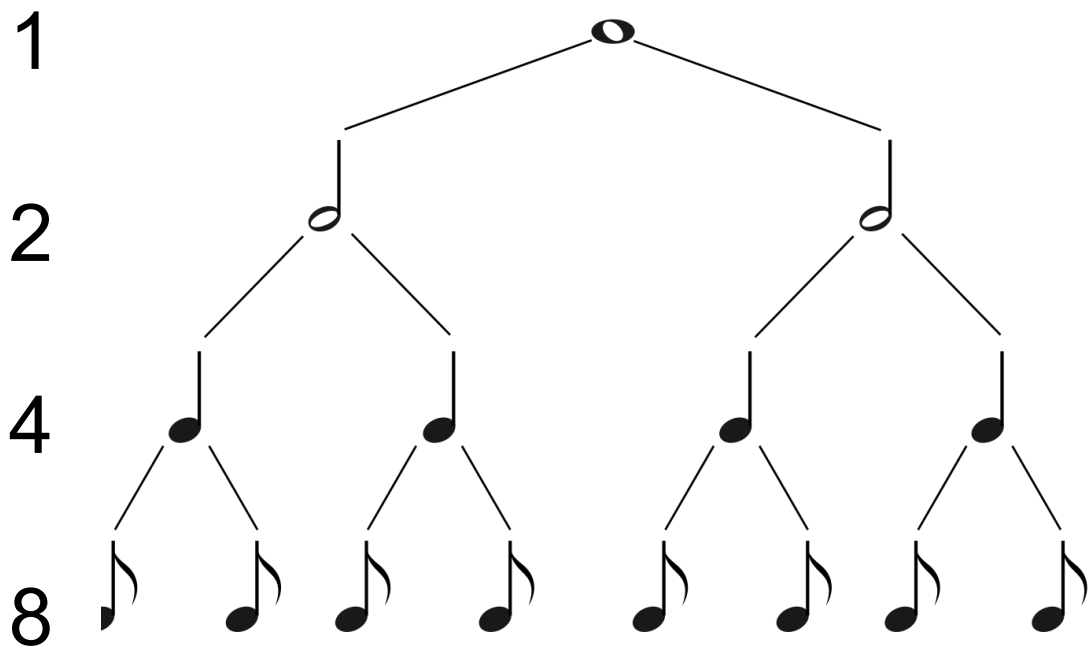
Rhythm in code

```
class RhythmSeq:
    '''A sequence of rhythmic durations'''
    # implementation here
    pass

# "r" represents a rest
RhythmSeq("4 4 4 4") # four quarter notes
RhythmSeq("4 4 8 8 4")

RhythmSeq("4 8 8 4 4")
RhythmSeq("r4 4 4 r4")
```

MORE Notes of different lengths



1.  $1 + \frac{1}{2}$

2.  $\frac{1}{2} + \frac{1}{4}$

4.  $\frac{1}{4} + \frac{1}{8}$

8.  $\frac{1}{8} + \frac{1}{16}$

MORE Notes of different lengths

1.

dotted rhythm example

```
NoteSeq("C4.' G8, C4.' G8, C8' G, C' E G2")
```

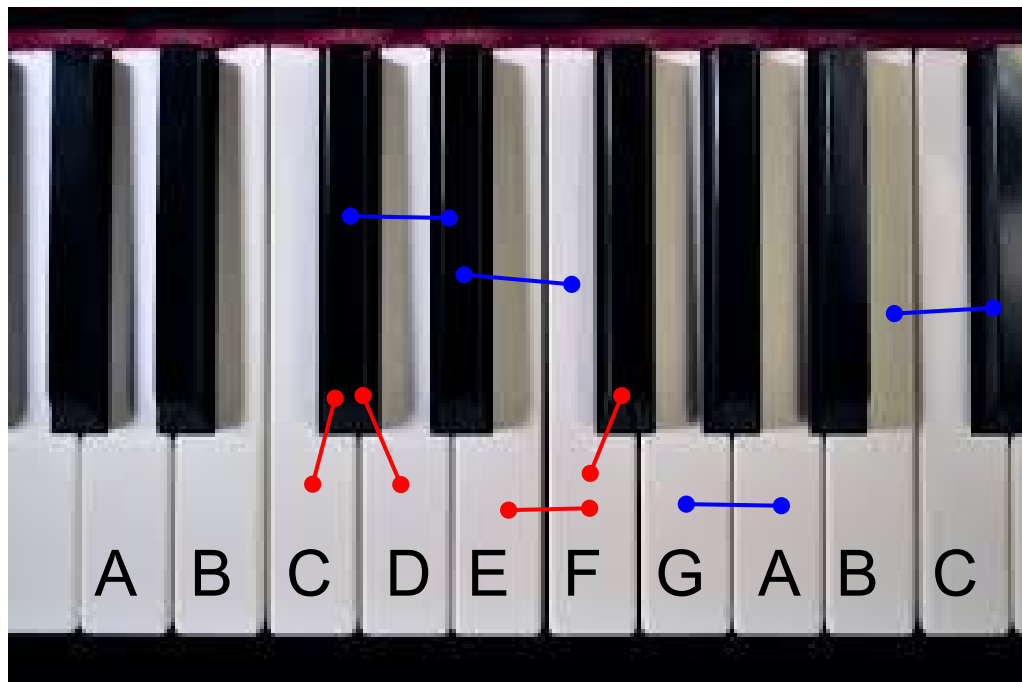
2.



4.

8.

Organizing Frequencies: Notes and Pitches

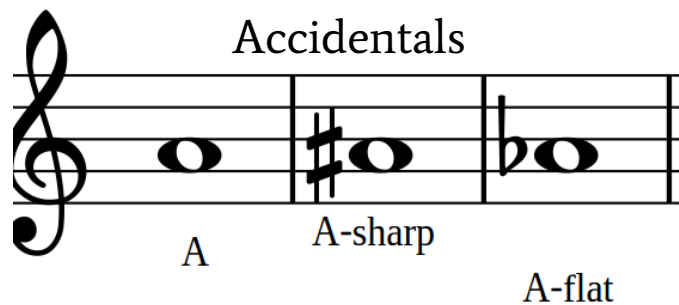


Simple note sequences in code

```
class NoteSeq:
    '''A sequence of notes with rhythms'''
    # implementation here
    pass

# letter-name + rhythm
NoteSeq("C4 D4 E4 D4 C2")
# can include rests
NoteSeq("C4 r4 E4 D4 C2")
# same duration is applied until changed
NoteSeq("C4 r E D2 C") == NoteSeq("C4 r4 E4 D2 C2")
```


Getting all of the notes



sharp = # (“A#”)

flat = b (“Ab”)



Organizing Frequencies: Notes and Pitches

Pyknon default octave = 5

MIDI pitches 0 - 127
(lowest C = 0)

Central C is in the fifth
octave
(MIDI pitch 60)



Organizing Frequencies: Notes and Pitches

Pyknon default octave = 5

Adjust the octave with upticks
and downticks (apostrophes and
commas)

`"C' ' C# ' ' D' ' ...": +1 octave`

`"C' C#' D' ...": default (5)`

`"C C# D ... ": default (5)`

`"C, C#, D, ...": -1 octave`

`"C,, C#,, D,, ...": -2 octaves`

Organizing Frequencies: Notes / Pitches

```
# name/value + rhythm + octave
```

```
NoteSeq("C4 C, C C' C'")
```

```
# octave 5 4 4 5 6
```

```
# same octave and duration applied until changed
```

```
NoteSeq("C#4' ' D E A' C#'")
```

```
# octave 6 6 6 5 6
```

Notes and Rests as Python objects

```
class Rest
```

```
    dur: the duration as a floating point number
```

```
        (quarter is 0.25 since  $1/4 = 0.25$ )
```

```
    stretch_dur(factor): multiplies the duration by factor
```

```
        and returns a new Rest with the new duration
```

Notes and Rests as Python objects

```
class Note
```

```
    value: integer value of a note from 0 to 11 (C to B)
```

```
    octave: octave value where central octave is 5
```

```
    midi_number: MIDI value for the pitch. Read-only.
```

```
    dur: rhythmic value as a floating-point number
```

```
    volume: MIDI volume value from 0 to 127
```

```
    verbose: returns a string <note_name>, <octave>, <dur>
```

Notes and Rests as Python objects

```
class Note
```

Defaults:

```
Note(value=0, octave=5, dur=0.25, volume=100)
```

```
# Note() == moderately loud middle-C quarter note
```

Shorthand:

```
Note("<note_name><dur><octave>")
```

```
Note("C4'")
```

Notes and Rests as Python objects

```
class Note
```

Other Methods:

transposition: moves notes up/down by n half-steps

and returns a new Note

inversion: takes a Note's value as distance from an index note (default is C), moves it to the other side of the index note, (e.g. D -> A#)

Notes and Rests as Python objects

```
class Note
```

Other Methods:

`harmonize`: Harmonize a single note in the context of a scale. Not very useful by itself, but it's used by `NoteSeq`.

`stretch_dur(factor)`: Multiplies the duration by *factor* and returns new `Note` with the resulting duration.

Notes and Rests as Python objects

```
class NoteSeq
```

- A list-like object that can hold Note and Rest objects
 - supports slicing, append, and insert
 - concatenation with the + operator
 - repetition with the * operator

Can be instantiated with a string, list of objects, or read from a file.

Notes and Rests as Python objects

```
class NoteSeq
```

```
    MORE METHODS:
```

```
        retrograde, transposition, transposition_startswith,  
        inversion, inversion_startswith, rotate, stretch_dur,  
        stretch_interval, harmonize
```

Generating MIDI files

```
from __future__ import division # python 2
```

```
def demo():
```

```
→ notes1 = NoteSeq("D4 F#8 A Bb4")
```

```
→ notes2 = NoteSeq([Note(2, dur=1/4), Note(6, dur=1/8),  
                    Note(9, dur=1/8), Note(10, dur=1/4)])
```

```
    midi = Midi(number_tracks=2, tempo=90)
```

```
    midi.seq_notes(notes1, track=0)
```

```
    midi.seq_notes(notes2, track=1)
```

```
    midi.write("midi/demo.mid")
```

same notes!



Generating MIDI files

```
from __future__ import division # python 2
```

```
def demo():  
    notes1 = NoteSeq("D4 F#8 A Bb4")  
    notes2 = NoteSeq([Note(2, dur=1/4), Note(6, dur=1/8),  
                      Note(9, dur=1/8), Note(10, dur=1/4)])  
    → midi = Midi(number_tracks=2, tempo=90)  
    midi.seq_notes(notes1, track=0)  
    midi.seq_notes(notes2, track=1)  
    midi.write("midi/demo.mid")
```

Generating MIDI files

```
from __future__ import division # python 2
```

```
def demo():  
    notes1 = NoteSeq("D4 F#8 A Bb4")  
    notes2 = NoteSeq([Note(2, dur=1/4), Note(6, dur=1/8),  
                      Note(9, dur=1/8), Note(10, dur=1/4)])  
    midi = Midi(number_tracks=2, tempo=90)  
    → midi.seq_notes(notes1, track=0)  
    → midi.seq_notes(notes2, track=1)  
    midi.write("midi/demo.mid")
```

Generating MIDI files

```
from __future__ import division # python 2
```

```
def demo():  
    notes1 = NoteSeq("D4 F#8 A Bb4")  
    notes2 = NoteSeq([Note(2, dur=1/4), Note(6, dur=1/8),  
                      Note(9, dur=1/8), Note(10, dur=1/4)])  
    midi = Midi(number_tracks=2, tempo=90)  
    midi.seq_notes(notes1, track=0)  
    midi.seq_notes(notes2, track=1)  
    → midi.write("midi/demo.mid")
```

Generating MIDI files

Midi.instrument: instrument 0 (piano) to 127

- search “MIDI instrument codes” for details
- support may vary depending on MIDI player

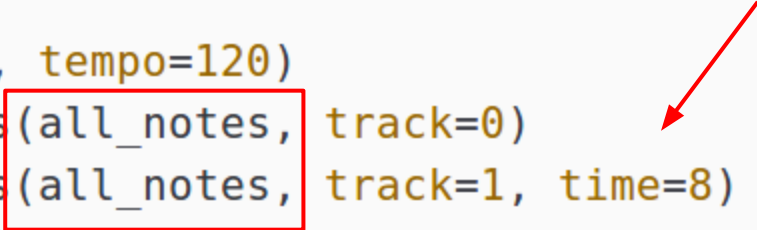
```
Midi.seq_chords([NoteSeq("C E G")])
```

```
Midi.seq_notes(time=0) ← badly named parameter
```


Generating MIDI files

```
# "Frere Jacques", a round in two tracks
filename = "frere-jacques-two-track.mid"
fj_notes1 = NoteSeq("C4' D E C C D E C E F G2 E4 F G2")
fj_notes2 = NoteSeq(
    "G8 A G F E4 C G8 A G F E4 C C G, C2' C4 G, C2'"
)
all_notes = fj_notes1 + fj_notes2

fj_midi = Midi(2, tempo=120)
fj_midi.seq_notes(all_notes, track=0)
fj_midi.seq_notes(all_notes, track=1, time=8)
fj_midi.write(filename)
```



Resources

Used in this talk:

“Music for Geeks and Nerds” by Pedro Kroger (creator of Pyknon)

- chapter on Pyknon available for FREEEEEE at github.com/kroger/pyknon

“The Complete Idiot’s Guide to Music Theory”, 2nd ed. by Michael Miller

teoria.com (FREEEEEEEE!)

Also good:

musictheory.net (FREEEEEEEE!), [PythonInMusic](https://pythoninmusic.com/) wiki (WIKIIIIIII!)

Thank you!

twitter: @__evanpalmer__

linkedIn: /in/evanpalmer1

github: palmerev

slides and extras: github.com/palmerev/pydx15-music

