Table of Contents

- User's Guide, Chapter 55: Advanced Meter Topics
  - Objects for Organizing Hierarchical Partitions
  - Creating and Editing MeterTerminal Objects
  - Creating and Editing MeterSequence Objects
  - Advanced Time Signature Configuration
  - Configuring Display
  - Configuring Beam
  - Configuring Beat
  - Annotating Found Notes with Beat Count
  - Using Beat Depth to Provide Metrical Analysis
  - Configuring Accent
  - Applying Articulations Based on Accent

Table of Contents

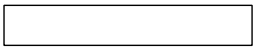
- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals

- User’s Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
- User’s Guide, Chapter 20: Examples 2
- User’s Guide, Chapter 21: Ordering and Sorting of Stream Elements
- User’s Guide, Chapter 22: Graphing and plotting
- User’s Guide, Chapter 23: Roman Numeral Analysis
- User’s Guide, Chapter 24: Configuring Environment Settings
- User’s Guide, Chapter 25: Post-Tonal Tools (1)
- User’s Guide, Chapter 26: Stream Iteration and Filtering
- User’s Guide, Chapter 27: Grace Notes
- User’s Guide, Chapter 28: Lyric Searching
- User’s Guide, Chapter 29: Spanners 1 (Slurs)
- User’s Guide, Chapter 30: Examples 3
- User’s Guide, Chapter 31: Clefs, Ties, and Beams
- User’s Guide, Chapter 32: Articulations
- User’s Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
- User’s Guide, Chapter 53: Advanced Corpus and Metadata Searching
- User’s Guide, Chapter 54: Extending Converter with New Formats
- User’s Guide, Chapter 55: Advanced Meter Topics
- User’s Guide, Chapter 58: Understanding Sites and Contexts
- User’s Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Quick search



This Page

- Show Source

User’s Guide, Chapter 55: Advanced Meter Topics¶

Objects for Organizing Hierarchical Partitions¶

Hierarchical metrical structures can be described as a type of fractional, space-preserving tree structure. With such a structure we partition and divide a single duration into one or more parts, where each part is a fraction of the whole. Each part can, in turn, be similarly divided. The objects for configuring this structure are the MeterTerminal and the MeterSequence objects.

MeterTerminal and the MeterSequence objects are for advanced configuration. For basic data access about common meters, see the discussion of TimeSignature, below.

Creating and Editing MeterTerminal Objects¶

A MeterTerminal is a node of the metrical tree structure, defined as a duration expressed as a fraction of a whole note. Thus, 1/4 is 1 quarter length (QL) duration; 3/8 is 1.5 QL; 3/16 is 0.75 QL. For this model, denominators are limited to $n = 2$:superscript:`x`, for x between 1 and 7 (e.g. 1/1 to 1/128).

MeterTerminals can additionally store a weight, or a numerical value that can be interpreted in a variety of different ways.

The following examples in the Python interpreter demonstrate creating a MeterTerminal and accessing the `numerator` and `denominator` attributes. The `duration` attribute stores a `Duration` object.

```
from music21 import *
mt = meter.MeterTerminal('3/4')
mt
```

```
 <music21.meter.core.MeterTerminal 3/4>
```

```
mt.numerator, mt.denominator
```

```
 (3, 4)
```

```
mt.duration.quarterLength
```

```
 3.0
```

A MeterTerminal can be broken into an ordered sequence of MeterTerminal objects that sum to the same duration. This new object, to be discussed below, is the MeterSequence. A MeterTerminal can be broken into these duration-preserving components with the `subdivide()` method. An argument for subdivision can be given as a desired number of equal-valued components, a list of numerators assuming equal-denominators, or a list of string fraction representations.

```
mt.subdivide(3)
```

```
 <music21.meter.core.MeterSequence {1/4+1/4+1/4}>
```

```
mt.subdivide([3,3])
```

```
 <music21.meter.core.MeterSequence {3/8+3/8}>
```

```
mt.subdivide(['1/4','4/8'])
```

```
 <music21.meter.core.MeterSequence {1/4+4/8}>
```

Creating and Editing MeterSequence Objects¶

A MeterSequence object is a sub-class of a MeterTerminal. Like a MeterTerminal, a MeterSequence has a `numerator`, a `denominator`, and a `duration` attribute. A MeterSequence, however, can be a hierarchical tree or sub-tree, containing an ordered sequence of MeterTerminal and/or MeterSequence objects.

The ordered collection of MeterTerminal and/or MeterSequence objects can be accessed like Python lists.

MeterSequence objects, like MeterTerminal objects, store a weight that by default is the sum of constituent weights.

The `partition()` and `subdivide()` methods can be used to configure the nested hierarchical structure.

The `partition()` method replaces existing MeterTerminal or MeterSequence objects in place with a new arrangement, specified as a desired number of equal-valued components, a list of numerators assuming equal-denominators, or a list of string fraction representations.

The `subdivide()` method returns a new MeterSequence (leaving the source MeterSequence unchanged) with an arrangement of MeterTerminals as specified by an argument in the same form as for the `partition()` method.

Note that MeterTerminal objects cannot be partitioned in place. A common way to convert a MeterTerminal into a MeterSequence is to reassign the returned MeterSequence from the `subdivide()` method to the position occupied by the MeterTerminal.

The following example creates and partitions a MeterSequence by re-assigning subdivisions to MeterTerminal objects. The use of Python list-like index access is also demonstrated.

```
ms = meter.MeterSequence('3/4')
ms

 <music21.meter.core.MeterSequence {3/4}>

ms.partition([3,3])
ms

 <music21.meter.core.MeterSequence {3/8+3/8}>

ms[0]

 <music21.meter.core.MeterTerminal 3/8>

ms[0] = ms[0].subdivide([3,3])
ms[0]

 <music21.meter.core.MeterSequence {3/16+3/16}>

ms

 <music21.meter.core.MeterSequence {{3/16+3/16}+3/8}>

ms[1] = ms[1].subdivide([1,1,1])
ms[1][0]

 <music21.meter.core.MeterTerminal 1/8>

ms[1]

 <music21.meter.core.MeterSequence {1/8+1/8+1/8}>

ms

 <music21.meter.core.MeterSequence {{3/16+3/16}+{1/8+1/8+1/8}}>
```

The resulting structure can be graphically displayed with the following diagram:

```
# 3/8 divisions
```

Numerous MeterSequence attributes provide convenient ways to access information about, or new objects from, the nested tree structure. The `depth` attribute returns the depth count at any node within the tree structure; the `flat` property returns a new, flat MeterSequence constructed from all the lowest-level MeterTerminal objects (all leaf nodes).

```
ms.depth
```

```
 2
```

```
ms[0].depth
```

```
 1
```

```
ms.flat
```

```
 <music21.meter.core.MeterSequence {3/16+3/16+1/8+1/8+1/8}>
```

Numerous methods provide ways to access levels (slices) of the hierarchical structure, or all nodes found at a desired hierarchical level. As all components preserve the duration of their container, all levels have the same total duration. The `getLevel()` method returns, for a given depth, a new, flat MeterSequence. The `getLevelSpan()` method returns, for a given depth, the time span of each node as a list of start and end values.

```
ms.getLevel(0)
```

```
 <music21.meter.core.MeterSequence {3/8+3/8}>
```

```
ms.getLevel(1)
```

```
 <music21.meter.core.MeterSequence {3/16+3/16+1/8+1/8+1/8}>
```

```
ms.getLevelSpan(1)
```

```
 [(0.0, 0.75), (0.75, 1.5), (1.5, 2.0), (2.0, 2.5), (2.5, 3.0)]
```

```
ms[1].getLevelSpan(1)
```

```
 [(0.0, 0.5), (0.5, 1.0), (1.0, 1.5)]
```

Finally, numerous methods provide ways to find and access the relevant nodes (the MeterTerminal or MeterSequence objects) active given a quarter length position into the tree structure. The `offsetToIndex()` method returns, for a given QL, the index of the active node. The `offsetToSpan()` method returns, for a given QL, the span of the active node. The `offsetToDepth()` method returns, for a given QL, the maximum depth at this position.

```
ms.offsetToIndex(2.5)
```

```
 1
```

```
ms.offsetToSpan(2.5)
```

```
 (1.5, 3.0)
```

```
ms.offsetToDepth(.5)
```

```
 2
```

```
ms[0].offsetToDepth(.5)
```

```
 1
```

```
ms.getLevel(1).offsetToSpan(.5)
```

```
 (0, 0.75)
```

Advanced Time Signature Configuration¶

The music21 `TimeSignature` object contains four parallel MeterSequence objects, each assigned to the attributes `displaySequence`, `beatSequence`, `beamSequence`, `accentSequence`. The following displays a graphical realization of these four MeterSequence objects.

```
# four MeterSequence objects
```

The TimeSignature provides a model of all common hierarchical structures contained within a bar. Common meters are initialized with expected defaults; however, full MeterSequence customization is available.

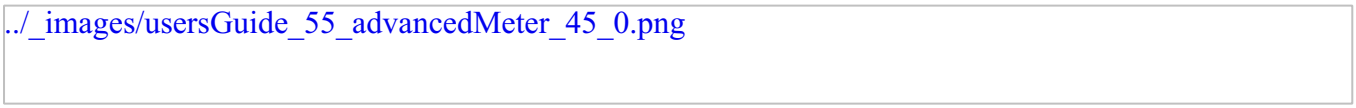
Configuring Display¶

The TimeSignature `displaySequence` MeterSequence employs the highest-level partitions to configure the displayed time signature symbol. If more than one partition is given, those partitions will be interpreted as additive meter components. If partitions have a common denominator, a summed numerator (over a single denominator) can be displayed by setting the TimeSignature `summedNumerator` attribute to True. Lower-level subdivisions of the TimeSignature MeterSequence are not employed.

Note that a new MeterSequence instance can be assigned to the `displaySequence` attribute with a duration and/or partitioning completely independent from the `beatSequence`, `beamSequence`, and `accentSequence` MeterSequences.

The following example demonstrates setting the display MeterSequence for a TimeSignature. NOTE that there is currently a bug in the first one that is showing 5/16 instead of 5/8. We hope to fix this soon.

```
from music21 import stream, note
ts1 = meter.TimeSignature('5/8') # assumes two partitions
ts1.displaySequence.partition(['3/16', '1/8', '5/16'])
ts2 = meter.TimeSignature('5/8') # assumes two partitions
ts2.displaySequence.partition(['2/8', '3/8'])
ts2.summedNumerator = True
s = stream.Stream()
for ts in [ts1, ts2]:
    m = stream.Measure()
    m.timeSignature = ts
    n = note.Note('b')
    n.quarterLength = 0.5
    m.repeatAppend(n, 5)
    s.append(m)

s.show()
```



Configuring Beam¶

The TimeSignature `beamSequence` MeterSequence employs the complete hierarchical structure to configure the single or multi-level beaming of a bar. The outer-most partitions can specify one or more top-level partitions. Lower-level partitions subdivide beam-groups, providing the appropriate beam-breaks when sufficiently small durations are employed.

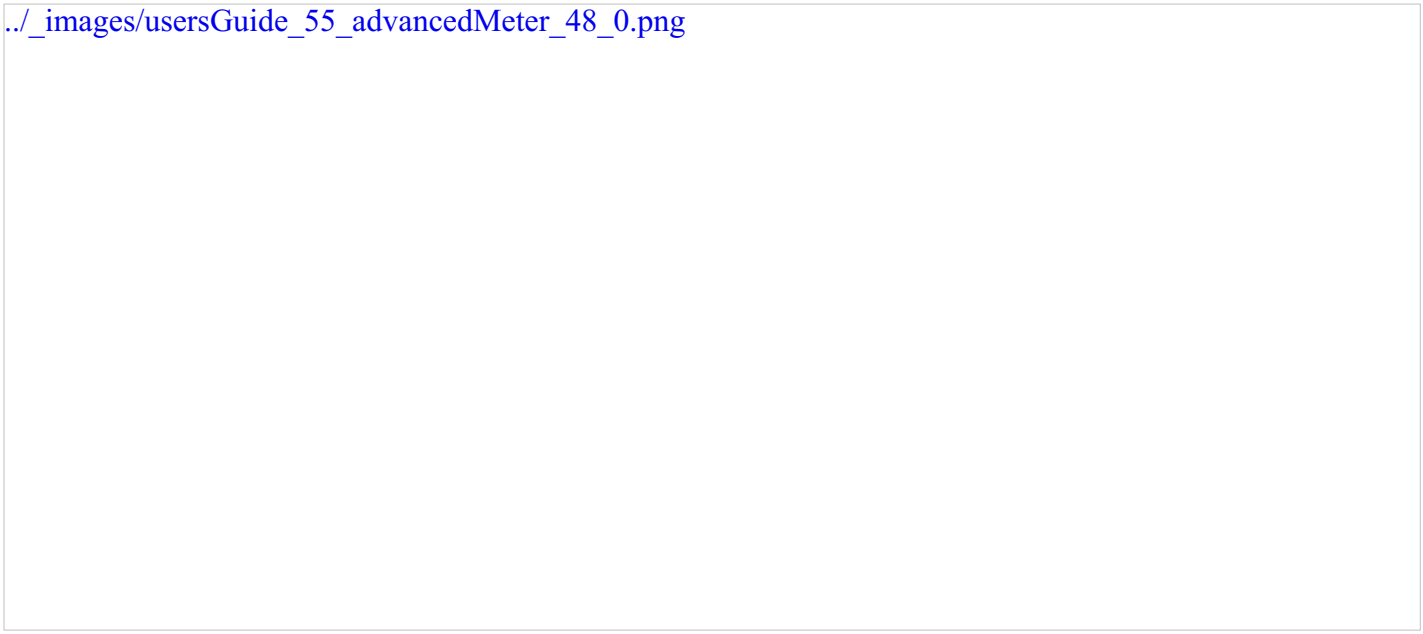
The `beamSequence` MeterSequence is generally used to create and configure `Beams` objects stored in `Note` objects. The TimeSignature `getBeams()` method, given a list of `Duration` objects, returns a list of `Beams` objects based on the TimeSignature `beamSequence` MeterSequence.

Many users may find the Stream `makeBeams()` method the most convenient way to apply beams to a Measure or Stream of Note objects. This method returns a new Stream with created and configured Beams.

The following example beams a bar of 3/4 in four different ways. The diversity and complexity of beaming is offered here to illustrate the flexibility of this model.

```
ts1 = meter.TimeSignature('3/4')
ts1.beamSequence.partition(1)
ts1.beamSequence[0] = ts1.beamSequence[0].subdivide(['3/8', '5/32', '4/32', '3/32'])
```

```
ts2 = meter.TimeSignature('3/4')
ts2.beamSequence.partition(3)

ts3 = meter.TimeSignature('3/4')
ts3.beamSequence.partition(3)

for i in range(len(ts3.beamSequence)):
    ts3.beamSequence[i] = ts3.beamSequence[i].subdivide(2)

ts4 = meter.TimeSignature('3/4')
ts4.beamSequence.partition(['3/8', '3/8'])
for i in range(len(ts4.beamSequence)):
    ts4.beamSequence[i] = ts4.beamSequence[i].subdivide(['6/32', '6/32'])
    for j in range(len(ts4.beamSequence[i])):
        ts4.beamSequence[i][j] = ts4.beamSequence[i][j].subdivide(2)

s = stream.Stream()
for ts in [ts1, ts2, ts3, ts4]:
    m = stream.Measure()
    m.timeSignature = ts
    n = note.Note('b')
    n.quarterLength = 0.125
    m.repeatAppend(n, 24)
    s.append(m.makeBeams())

s.show()
```



The following is a fractional grid representation of the four beam partitions created.

```
# four beam partitions
```

Configuring Beat¶

The TimeSignature `beatSequence` MeterSequence employs the hierarchical structure to define the beats and beat divisions of a bar. The outer-most partitions can specify one or more top level beats. Inner partitions can specify the beat division partitions. For most common meters, beats and beat divisions are pre-configured by default.

In the following example, a simple and a compound meter is created, and the default beat partitions are examined. The `getLevel()` method can be used to show the beat and background beat partitions. The timeSignature `beatDuration`, `beat`, and `beatCountName` properties can be used to return commonly needed

beat information. The TimeSignature `beatDivisionCount`, and `beatDivisionCountName` properties can be used to return commonly needed beat division information. These descriptors can be combined to return a string representation of the TimeSignature classification with `classification` property.

```
ts = meter.TimeSignature('3/4')
ts.beatSequence.getLevel(0)
```

```
 <music21.meter.core.MeterSequence {1/4+1/4+1/4}>
```

```
ts.beatSequence.getLevel(1)
```

```
 <music21.meter.core.MeterSequence {1/8+1/8+1/8+1/8+1/8+1/8}>
```

```
ts.beatDuration
```

```
 <music21.duration.Duration 1.0>
```

```
ts.beatCount
```

```
 3
```

```
ts.beatCountName
```

```
 'Triple'
```

```
ts.beatDivisionCount
```

```
 2
```

```
ts.beatDivisionCountName
```

```
 'Simple'
```

```
ts.classification
```

```
 'Simple Triple'
```

```
ts = meter.TimeSignature('12/16')
ts.beatSequence.getLevel(0)
```

```
 <music21.meter.core.MeterSequence {3/16+3/16+3/16+3/16}>
```

```
ts.beatSequence.getLevel(1)
```

```
 <music21.meter.core.MeterSequence {1/16+1/16+1/16+1/16+1/16+1/16+1/16+1/16+1/16+1/16+1/16+1/16
```

```
ts.beatDuration
```

```
 <music21.duration.Duration 0.75>
```

```
ts.beatCount
```

```
 4
```

```
ts.beatCountName
```

```
 'Quadruple'
```

```
ts.beatDivisionCount
```

```
 3
```

```
ts.beatDivisionCountName
```

```
 'Compound'
```

```
ts.classification
```

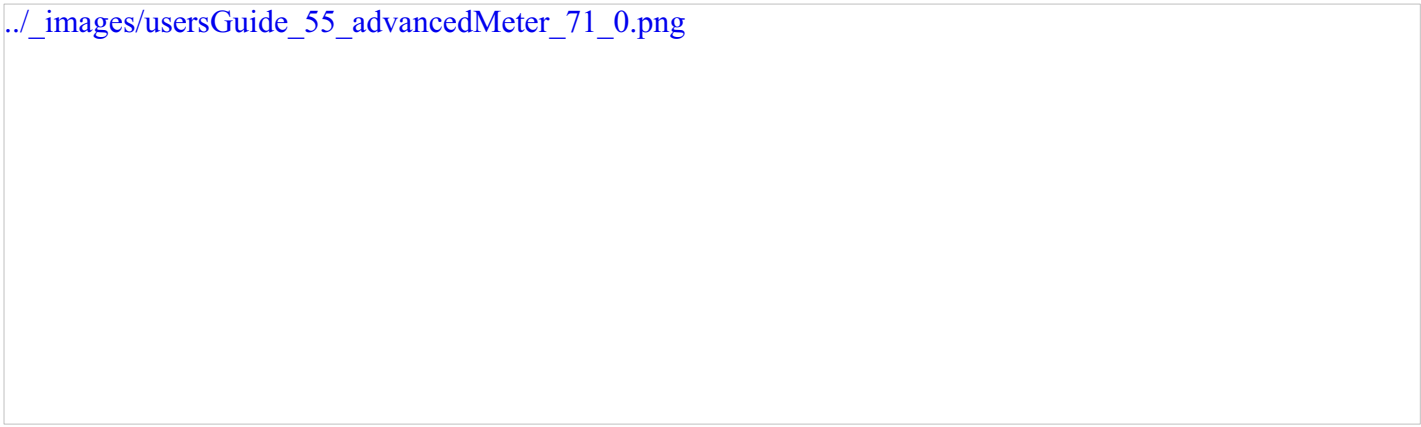
```
 'Compound Quadruple'
```

Annotating Found Notes with Beat Count¶

The `getBeat()` method returns the currently active beat given a quarter length position into the TimeSignature.

In the following example, all leading tones, or C#s, are collected into a new Stream and displayed with annotations for part, measure, and beat.

```
score = corpus.parse('bach/bwv366.xml')
ts = score[meter.TimeSignature].first()
ts.beatSequence.partition(3)

found = stream.Stream()
offsetQL = 0
for part in score.parts:
    found.insert(offsetQL, part[clef.Clef].first())
    for i in range(len(part.getElementsByClass(stream.Measure))):
        m = part[stream.Measure][i]
        for n in m.notesAndRests:
            if n.name == 'C#':
                n.addLyric(f'{part.id[0]}, m. {m.number}')
                n.addLyric(f'beat {ts.getBeat(n.offset)}')
                found.insert(offsetQL, n)
                offsetQL += 4

found.show()
```



Using Beat Depth to Provide Metrical Analysis¶

Another application of the `beatSequence` MeterSequence is to define the hierarchical depth active for a given note found within the TimeSignature.

The `getBeatDepth()` method, when set with the optional parameter `align` to “quantize”, shows the number of hierarchical levels that start at or before that point. This value is described by Lerdahl and Jackendoff as metrical analysis.

In the following example, `beatSequence` MeterSequence is partitioned first into one subdivision, and then each subsequent subdivision into two, down to four layers of partitioning.

The number of hierarchical levels, found with the `getBeatDepth()` method, is appended to each note with the `addLyric()` method.

```
score = corpus.parse('bach/bwv281.xml')
partBass = score.getElementById('Bass')
ts = partBass[meter.TimeSignature].first()
ts.beatSequence.partition(1)
for h in range(len(ts.beatSequence)):
    ts.beatSequence[h] = ts.beatSequence[h].subdivide(2)
    for i in range(len(ts.beatSequence[h])):
        ts.beatSequence[h][i] = ts.beatSequence[h][i].subdivide(2)
        for j in range(len(ts.beatSequence[h][i])):
            ts.beatSequence[h][i][j] = ts.beatSequence[h][i][j].subdivide(2)
```

```
for m in partBass.getElementsByClass(stream.Measure):
    for n in m.notesAndRests:
        for i in range(ts.getBeatDepth(n.offset)):
            n.addLyric('*')

partBass.measures(0, 7).show()
```

Alternatively, this type of annotation can be applied to a Stream using the `labelBeatDepth()` function.

Configuring Accent¶

The TimeSignature `accentSequence` MeterSequence defines one or more levels of hierarchical accent levels, where quantitative accent value is encoded in MeterTerminal or MeterSequence with a number assigned to the `weight` attribute.

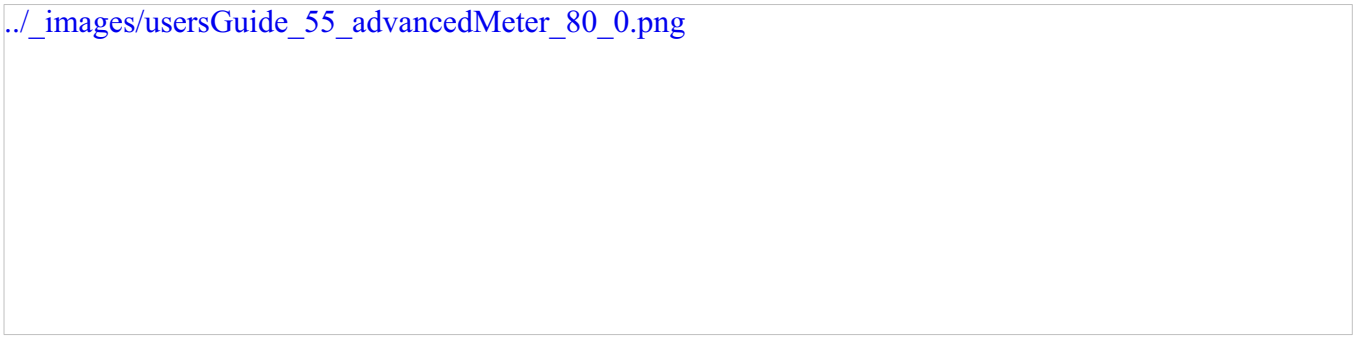
Applying Articulations Based on Accent¶

The `getAccentWeight()` method returns the currently active accent weight given a quarter length position into the TimeSignature. Combined with the `getBeatProgress()` method, Notes that start on particular beat can be isolated and examined.

The following example extracts the Bass line of a Bach chorale in 3/4 and, after repartitioning the beat and accent attributes, applies accents to reflect a meter of 6/8.

```
score = corpus.parse('bach/bwv366.xml')
partBass = score.getElementById('Bass')
ts = partBass[meter.TimeSignature].first()
ts.beatSequence.partition(['3/8', '3/8'])
ts.accentSequence.partition(['3/8', '3/8'])
ts.setAccentWeight([1, .5])
for m in partBass[stream.Measure]:
```

```
    lastBeat = None
    for n in m.notesAndRests:
        beat, progress = ts.getBeatProgress(n.offset)
        if beat != lastBeat and progress == 0:
            if n.tie != None and n.tie.type == 'stop':
                continue
            if ts.getAccentWeight(n.offset) == 1:
                mark = articulations.StrongAccent()
            elif ts.getAccentWeight(n.offset) == .5:
                mark = articulations.Accent()
            n.articulations.append(mark)
            lastBeat = beat
        m = m.sorted()

partBass.measures(0, 8).show()
```



This is probably more depth to the concept of meter than anyone would ever want! But hope that it has whetted your appetite for jucier concepts still to come!

The next chapters are not yet complete, so let's jump head to Chapter 58: Understanding Sites and Contexts

Navigation

Table of Contents

- User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - Starting `music21`

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes
  - User's Guide, Chapter 28: Lyric Searching

    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - The Pitch object
  - Carving time with Duration objects
  - Back to Notes

Table of Contents

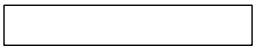
- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering

- User’s Guide, Chapter 27: Grace Notes
- User’s Guide, Chapter 28: Lyric Searching
- User’s Guide, Chapter 29: Spanners 1 (Slurs)
- User’s Guide, Chapter 30: Examples 3
- User’s Guide, Chapter 31: Clefs, Ties, and Beams
- User’s Guide, Chapter 32: Articulations
- User’s Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
- User’s Guide, Chapter 53: Advanced Corpus and Metadata Searching
- User’s Guide, Chapter 54: Extending Converter with New Formats
- User’s Guide, Chapter 55: Advanced Meter Topics
- User’s Guide, Chapter 58: Understanding Sites and Contexts
- User’s Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Quick search



This Page

- Show Source

User's Guide, Chapter 3: Pitches, Durations, and Notes again¶

Now that you've made a couple of `Note` objects, it's time to dig a little deeper into what makes a `Note` really a `Note`, namely, music21.pitch, and music21.duration objects.

The Pitch object¶

Since we've already covered `Note` objects, `Pitch` objects will be a breeze. Just like how the `Note` object is

found in the `note` module, the `Pitch` object is found in the `pitch` module.

Let's create a `Pitch`. Like we did with `Notes`, just class the class with a note name, such as `B` with the optional symbols for sharp or flat, (# or - respectively).

You can put an octave number after the name (4 = low treble clef), but you don't have to:

```
from music21 import *
p1 = pitch.Pitch('b-4')
```

Here we'll use a more abstract variable name, `p1` for our first `Pitch`, just in case we change the pitch later (via `.transpose()` or something else).

Just like we saw with `Notes` there are a lot of attributes (a.k.a. properties; we'll use the term interchangeably for a bit before we talk about the difference) and methods that describe and change pitches. The first three will be old hat from `Note` objects:

```
p1.octave
```

```
 4
```

```
p1.pitchClass
```

```
 10
```

```
p1.name
```

```
 'B-'
```

```
p1.accidental.alter
```

```
 -1.0
```

Here are two more that you can use. The first is pretty self-explanatory. The second gives the value of the `Pitch` in the older, "MIDI" representation that is still in use today. It's a number between 0 and 127 where middle C (C4) is 60 and C#4/Db4 is 61, B3 is 59, etc.

```
p1.nameWithOctave
```

```
 'B-4'
```

```
p1.midi
```

```
 70
```

Most of these attributes can be changed (they are "settable properties" in Python speak).

When an attribute is set, the `Pitch` object changes whatever is necessary to reflect the new value:

```
p1.name = 'd#'
p1.octave = 3
p1.nameWithOctave
```

```
 'D#3'
```

And our familiar `.transpose()` method also appears on `Pitch` as well. Remember that `p1` is now a `D#`:

```
p2 = p1.transpose('M7')
p2
```

```
 <music21.pitch.Pitch C##4>
```

Notice that at the command line, just printing the variable name gives you the representation `<music21.pitch.Pitch C##4>`. You can also get this by typing `repr(p2)`.

So, there's really nothing new about `Pitch` objects that you didn't already know from learning about `Notes`.

So why the two different objects? It turns out, they are so similar because actually every `Note` object has a `Pitch` object inside it (like the monster in *Alien* but more benign). Everything that we did with the `note.Note` object, we could do with the `note.Note.pitch` object instead:

```
csharp = note.Note('C#4')
csharp.name
```

```
'C#'
```

```
csharp.pitch.name
```

```
'C#'
```

```
csharp.octave
```

```
4
```

```
csharp.pitch.octave
```

```
4
```

But pitch objects have a lot more to offer for more technical working, for instance, `Pitch` objects know their names in Spanish:

```
csharp.pitch.spanish
```

```
'do sostenido'
```

Notes don't:

```
csharp.spanish
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-16-a0f0e2502262> in <module>
----> 1 csharp.spanish


AttributeError: 'Note' object has no attribute 'spanish'
```

Here are some other things you can do with Pitch objects. Get the sharp printed nicely:

```
print(csharp.pitch.unicodeName)
```

```
C♯
```

Get some enharmonics – these are methods, so we add `()` to them:

```
print( csharp.pitch.getEnharmonic() )
print( csharp.pitch.getLowerEnharmonic() )
```

```
D-4
B##3
```

By the way, you know how we said that you shouldn't have a variable named `pitch` *because there's already a module named* `pitch`*. You might wonder why* `Note` *objects can have an attribute named* `pitch` *without causing any problems. It's because the* `.pitch` *attribute is always attached to a* `Note` *, so it's never used without a prefix of some sort (in this case,* `csharp.pitch` *), and that's enough to prevent any trouble.*

So far, it looks like `Pitch` objects can do everything `Note` objects can do and more. So why do we need `Note` objects? It's because they also have `Duration` attributes, as we'll see in the next section. Without a `Duration` attribute, you cannot put an object into a Measure or show it on your screen.

Carving time with Duration objects¶

For a `Note` to occupy musical space, it has to last a certain amount of time. We call that time the `Note`'s `Duration`. `Duration` objects are ubiquitous in music21. Nearly all objects have, or can have, a `Duration`. A `Duration` object can represent just about any time span.

`Duration` objects are best used when they're attached to something else, like a `Note` or a `Rest`, but for now, let's look at what we can do with them on their own.

`Duration` objects reside in the `duration` module. When you create a `Duration` object, you can say what type of duration you want it to be when you create it.

Here we'll create the duration of a half note:

```
halfDuration = duration.Duration('half')
```

The string "half" is called the "type" of the `Duration`. Music21 `Durations` use the common American duration types: "whole", "half", "quarter", "eighth", "16th", "32nd", "64th". Note that for durations shorter than an eighth note, we use numbers instead of spelling out the whole name of the `Duration` type. Music21 also supports less commonly used types such as "breve" (2 whole notes), "longa" (4 whole notes), and "maxima" (8 whole notes) and on the other side, "128th", "256th", etc. down to "2048th" notes. (Some of these very long and very short notes can't be displayed in many musical notation systems, but it's good to know that we're ready when they are).

The other standard way of creating a `Duration` is by passing it a number when it is created. That number represents how many quarter notes long it is. So we could have created our half note `Duration` by saying `2` or `2.0`. But we can also create `Durations` that aren't exactly "whole", "half", "quarter", etc. Let's create a dotted quarter note, which is 1.5 quarter notes long:

```
dottedQuarter = duration.Duration(1.5)
```

As with the `Pitch` and `Note` objects we've already seen, there are a bunch of attributes that `Duration` objects have. The most important one is `.quarterLength`. The `quarterLength` of our `dottedQuarter` variable is of course 1.5: we set it to be. But just as importantly, the `halfDuration` object also has its quarterLength set:

```
dottedQuarter.quarterLength
```

```
1.5
```

```
halfDuration.quarterLength
```

```
2.0
```

The `.type` attribute tells you what general type of `Duration` you have:

```
halfDuration.type
```

```
'half'
```

```
dottedQuarter.type
```

```
'quarter'
```

The `type` attribute cannot be everything that describes the `Duration`, there has to be some place where music21 keeps track of the fact that the `dottedQuarter` variable has a dot (otherwise it wouldn't have a `quarterLength` of 1.5). You'll find the attribute called `.dots`:

```
halfDuration.dots
```

```
0
```

```
dottedQuarter.dots
```

```
1
```

The attributes of `dots`, `type`, and `quarterLength` are actually special attributes called "properties". A

property is an attribute that is smart in some way. Let's change the number of dots on our `dottedQuarter` object and see what happens to the `quarterLength` property:

```
dottedQuarter.dots = 2
dottedQuarter.quarterLength
```

```
1.75
```

```
dottedQuarter.dots = 3
dottedQuarter.quarterLength
```

```
1.875
```

```
dottedQuarter.dots = 4
dottedQuarter.quarterLength
```

```
1.9375
```

Or let's change the `quarterLength` of the dottedQuarter and see what happens to the `type` and `dots`:

```
dottedQuarter.quarterLength = 0.25
dottedQuarter.type
```

```
'16th'
```

```
dottedQuarter.dots
```

```
0
```

QuarterLengths are so important to music21 that we'll sometimes abbreviate them as `qL` or `qLs`. Almost everything that is measured in music21 is measured in `qLs`.

Music21 can also deal with other `quarterLengths` such as 0.8, which is 4/5ths of a quarter note, or 1/3, which is an eighth note triplet. (For this reason, users compiling spreadsheets or other text-based output should expect to find the occasional `quarterLength` expressed as a `Fraction`.)

(You can go ahead and make a triplet or other `Tuplet`, but we'll get to triplets, including tips for manipulating a `Fraction` in Chapter 19).

Back to Notes¶

So now you can see the advantage of working with `Note` objects: they have both a `.pitch` attribute, which contains a `Pitch` object, and a `.duration` attribute, which contains a `Duration` object. The default `Pitch` for a `Note` is `C` (meaning `C4`) and the default `Duration` is 1.0, or a quarter note.

```
n1 = note.Note()
n1.pitch
```

```
<music21.pitch.Pitch C4>
```

```
n1.duration
```

```
<music21.duration.Duration 1.0>
```

But we can play around with them:

```
n1.pitch.nameWithOctave = 'E-5'
n1.duration.quarterLength = 3.0
```

and then the other properties change accordingly:

```
n1.duration.type
```

```
'half'
```

```
n1.duration.dots
```

```
 1

n1.pitch.name

 'E-'

n1.pitch.accidental

 <music21.pitch.Accidental flat>

n1.octave

 5
```

We already said that some of the attributes of `Pitch` can also be called on the `Note` object itself. The same is true for the most important attributes of `Duration`:

```
n1.name

 'E-'

n1.quarterLength

 3.0
```

Let's change the quarterLength back to 1.0 for now:

```
n1.quarterLength = 1.0
```

`Notes` can do things that neither `Pitch` or `Duration` objects can do. For instance, they can have lyrics. Let's add some lyrics to `Notes`. You can easily set `Lyric` objects just by setting the `lyric` property. (For reference, the `lyric` attribute is actually an attribute of `GeneralNote`, which is a "base class" from which the `Note` class "inherits". In other words, the `Note` class gains the `lyric` attribute from `GeneralNote`. But that's not too important.)

```
otherNote = note.Note("F6")
otherNote.lyric = "I'm the Queen of the Night!"
```

But let's do something more complex. Here I add multiple lyrics to `n1` using the `Note's` `addLyric()` method. And instead of adding a simple String, I'll add as a lyric the name of the note itself and its `pitchClassString`.

```
n1.addLyric(n1.nameWithOctave)
n1.addLyric(n1.pitch.pitchClassString)
```

Finally, lets put the `quarterLength` of the note as a string with a preface "QL:":

```
n1.addLyric(f'QL: {n1.quarterLength}')
```

The placement of an "f" before the string `f'QL: {n1.quarterLength}'` says to substitute anything in `{}` with its actual value as a string. (Remember that `.quarterLength` is not a string, but a float).

As it should be becoming clear, we can always check our work with the `show()` method.

```
n1.show()
```

If we now edit the `quarterLength` property we can still change the `Note`'s `Duration`. But because we already set the lyric to show "`QL: 1.0`, it won't be changed when we `.show()` it again in the following example.

```
n1.quarterLength = 6.25
n1.show()
```

There are many more things we can do with a `Note` object, but I'm itching to look at what happens when we put multiple `Notes` together in a row. And to do that we'll need to learn a bit about the topic of Chapter 4: Streams.

Navigation

Table of Contents

- User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
  - Customizing Plots
  - Axis (and allies)
  - Customizing Data Points
  - Graph Primitives
  - Embedding in Apps: Selecting the matplotlib Backend

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings

    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Previous topic

List of Works Found in the *music21* Corpus

Next topic

User's Guide: Table of Contents

Table of Contents

- About *music21*
- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes
  - User's Guide, Chapter 28: Lyric Searching
  - User's Guide, Chapter 29: Spanners 1 (Slurs)
  - User's Guide, Chapter 30: Examples 3
  - User's Guide, Chapter 31: Clefs, Ties, and Beams
  - User's Guide, Chapter 32: Articulations
  - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
  - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching

    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - Sites and the storing of elements
  - Derivations
  - Context attributes
    - Methods on `Music21Objects`
  - .getOffsetBySite and .setOffsetBySite
  - getContextByClass()
  - Splitting methods
  - Showing and Writing

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements

    - User's Guide, Chapter 22: Graphing and plotting
    - User's Guide, Chapter 23: Roman Numeral Analysis
    - User's Guide, Chapter 24: Configuring Environment Settings
    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 58: Understanding Sites and Contexts
  - Working with Sites
  - From weakness I get strength of memory

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes

    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes
  - User's Guide, Chapter 28: Lyric Searching
  - User's Guide, Chapter 29: Spanners 1 (Slurs)
  - User's Guide, Chapter 30: Examples 3
  - User's Guide, Chapter 31: Clefs, Ties, and Beams
  - User's Guide, Chapter 32: Articulations
  - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)

    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 29: Spanners 1 (Slurs)
  - Slurs
  - Spanners in Streams
  - Querying spanners
  - Getting spanners from Music21Objects
  - Manipulating spanners
  - Changing the look of spanners

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis

    - User's Guide, Chapter 24: Configuring Environment Settings
    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 25: Post-Tonal Tools (1)
  - Pitches as Pitch Classes
  - Chords as Forte Set Classes
  - Creating and Processing Twelve-Tone Matrices

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering

    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 61: TimespanTrees and Verticalities
  - So what *is* the point of a tree?
  - TimeSpans vs Music21Objects
  - `makeElement`: the guts of Chordify

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering

    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 9: Chordify
  - Using Chordify to Annotate Intervals
  - Chordify and advanced scores

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes

    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 27: Grace Notes
  - Basic Graces
  - Multiple Grace Notes
  - Stealing time
  - Chordify and Grace Notes
  - Appogiature

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings

    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes
  - User's Guide, Chapter 28: Lyric Searching
  - User's Guide, Chapter 29: Spanners 1 (Slurs)
  - User's Guide, Chapter 30: Examples 3
  - User's Guide, Chapter 31: Clefs, Ties, and Beams
  - User's Guide, Chapter 32: Articulations
  - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)

    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- About *music21*
- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes
  - User's Guide, Chapter 28: Lyric Searching
  - User's Guide, Chapter 29: Spanners 1 (Slurs)
  - User's Guide, Chapter 30: Examples 3
  - User's Guide, Chapter 31: Clefs, Ties, and Beams
  - User's Guide, Chapter 32: Articulations
  - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)

    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
  - The Default Local Corpus
  - Creating multiple corpus repositories via local corpora
  - Inspecting metadata bundle search results
  - Manipulating multiple metadata bundles
  - Getting a metadata bundle
  - Creating persistent metadata bundles

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis

    - User's Guide, Chapter 24: Configuring Environment Settings
    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 7: Chords
  - Displaying Chords
  - More ways of creating chords; Chords and Streams
  - Post-tonal chords (in brief)

Table of Contents

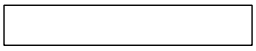
- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering

- User’s Guide, Chapter 27: Grace Notes
- User’s Guide, Chapter 28: Lyric Searching
- User’s Guide, Chapter 29: Spanners 1 (Slurs)
- User’s Guide, Chapter 30: Examples 3
- User’s Guide, Chapter 31: Clefs, Ties, and Beams
- User’s Guide, Chapter 32: Articulations
- User’s Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
- User’s Guide, Chapter 53: Advanced Corpus and Metadata Searching
- User’s Guide, Chapter 54: Extending Converter with New Formats
- User’s Guide, Chapter 55: Advanced Meter Topics
- User’s Guide, Chapter 58: Understanding Sites and Contexts
- User’s Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Quick search



This Page

- Show Source

User's Guide, Chapter 7: Chords¶

Chords, as the name might suggest, are objects that combine multiple `Pitch` objects on a single stem. They can be found in the music21.chord module. The most general way to create a `Chord` object is by passing in a list of pitch names you want in the chord:

```
from music21 import *
cMinor = chord.Chord(["C4","G4","E-5"])
```

`Note` and `Chord` objects, since both are subclasses of the `GeneralNote` object share many features in common:

```
cMinor.duration.type = 'half'
cMinor.quarterLength
```

```
 2.0
```

But since a `Chord` contains many pitches, it does not have a `.pitch` attribute:

```
cMinor.pitch
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-3-e8314cc72a18> in <module>
----> 1 cMinor.pitch


AttributeError: 'Chord' object has no attribute 'pitch'
```

Instead it has a `.pitches` attribute which returns a Tuple of pitches in the Chord.

```
cMinor.pitches
```

```
 (<music21.pitch.Pitch C4>, <music21.pitch.Pitch G4>, <music21.pitch.Pitch E-5>)
```

A little bit more about Python. `.pitches` returns a tuple, what's that? A tuple is like a list, except that unlike a list which has square brackets around it, a tuple has parentheses around it:

```
baroqueTuple = ('Strozzi', 'Bach', 'Handel', 'Telemann')
classicalList = ['Mozart', 'Haydn', 'Saint-George', 'Beethoven']
```

```
baroqueTuple
```

```
 ('Strozzi', 'Bach', 'Handel', 'Telemann')
```

```
classicalList
```

```
 ['Mozart', 'Haydn', 'Saint-George', 'Beethoven']
```

Both tuples and lists can find members by accessing them with numbers in square brackets:

```
baroqueTuple[0]
```

```
 'Strozzi'
```

```
classicalList[0]
```

```
 'Mozart'
```

But the biggest difference between the two is that you can manipulate a list, but not a tuple. If we try to add someone to the classicalList, using `.append` it works great.

```
classicalList.append('Ella Fitzgerald') # she's a classic to me...
```

```
classicalList
```

```
 ['Mozart', 'Haydn', 'Saint-George', 'Beethoven', 'Ella Fitzgerald']
```

```
classicalList.remove('Mozart')
```

```
classicalList
```

```
 ['Haydn', 'Saint-George', 'Beethoven', 'Ella Fitzgerald']
```

But a tuple can't be changed:

```
baroqueTuple.append('Miles Davis')
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-17-fd7e965db659> in <module>
----> 1 baroqueTuple.append('Miles Davis')


AttributeError: 'tuple' object has no attribute 'append'
```

In this case, that's a shame, because Miles belongs with Bach! So we shouldn't have used a tuple there.

But in the class of `.pitches` it makes sense that it returns a tuple, since changing the result `.pitches` separate from the Chord could be ambiguous – did you mean to change the result of `.pitches` or to change the pitches in the chord itself? `music21` will often return tuples in places where manipulating the result could cause headaches or bugs down the line. In any case, it's easy to turn a tuple into a list or vice versa:

```
baroqueList = list(baroqueTuple)
baroqueList.append('Miles Davis')
baroqueList
```

```
 ['Strozzi', 'Bach', 'Handel', 'Telemann', 'Miles Davis']
```

```
classicalTuple = tuple(classicalList)
classicalTuple
```

```
 ('Haydn', 'Saint-George', 'Beethoven', 'Ella Fitzgerald')
```

Okay, back to chord, where we left off:

```
cMinor.pitches
```

```
 (<music21.pitch.Pitch C4>, <music21.pitch.Pitch G4>, <music21.pitch.Pitch E-5>)
```

But you already knew what pitches were in the `Chord` since you just created it! What else can you do with it?

How about determining if it is a `major` or a `minor` triad?

```
cMinor.isMajorTriad()
```

```
 False
```

```
cMinor.isMinorTriad()
```

```
 True
```

You can also figure out if it is in inversion or not:

```
cMinor.inversion()
```

```
 0
```

Chords in root position have inversion of 0. But consider this other chord:

```
cMajor = chord.Chord(["E3","C4","G4"])
cMajor.inversion()
```

```
 1
```

With this chord, two other methods become important:

```
cMajor.root()
```

```
 <music21.pitch.Pitch C4>
```

```
cMajor.bass()
```

```
 <music21.pitch.Pitch E3>
```

You can find the third and fifth of the `Chord` with .third and .fifth. Note that these properties do not have `()` after them. This was a mistake in how we created `music21` and hopefully this will all be fixed and consistent soon:

```
cMajor.third
```

```
 <music21.pitch.Pitch E3>
```

```
cMajor.fifth
```

```
 <music21.pitch.Pitch G4>
```

There is also a .seventh property, but it won't do anything here:

```
cMajor.seventh
```

The result of that is `None` which we can test like so…

```
cMajor.seventh is None
```

```
 True
```

We can append or remove notes from a chord, just like in a set:

```
dMaj = chord.Chord('D4 F#4')
dMaj.add('A5')
dMaj
```

```
 <music21.chord.Chord D4 F#4 A5>
```

```
dMaj.remove('D4')
dMaj
```

```
 <music21.chord.Chord F#4 A5>
```

```
dMaj.add(pitch.Pitch('D3'))
dMaj.add(note.Note('F#5'))
dMaj
```

```
 <music21.chord.Chord D3 F#4 F#5 A5>
```

Displaying Chords¶

We can display the `Chord` object just like any `Note` (Don't worry if this isn't working for you yet…we'll get this set up in Chapter 8)

```
cMinor.show()
```



```
cMajor.show()
```


These chords are a bit "spacey", so let's get `c` in `closedPosition()`:

```
cClosed = cMinor.closedPosition()
cClosed.show()
```

Notice that `cMinor` is unchanged. The closed position chord is only `cClosed`:

```
cMinor.show()
```

If we wanted to change the Chord object itself, we call `.closedPosition(inPlace=True)` which alters the original. Since the original is altered, we don't put `x = ...` in front of it.

```
cMajor.closedPosition(inPlace=True)
cMajor.show()
```

There is also a method, `semiClosedPosition()` which acts like `.closedPosition` except that if there is already a pitch at that step (i.e., D-flat and D-sharp are both step "D"), then the note is moved up an octave. This is useful for displaying complex, post tonal chords in the most compact form possible:

```
c1 = chord.Chord(['C4', 'E5', 'C#6', 'E-7', 'G8', 'C9', 'E#9'])
c2 = c1.semiClosedPosition()
c2.show()
```

We can get the `common_name` of each of these Chords:

```
cn1 = cMinor.commonName
print(cn1)
```

```
 minor triad
```

```
print(cMajor.commonName)
```

```
 major triad
```

More complex chords have less common "commonNames". Here's one that the American composer Elliott Carter liked a lot.

```
elliottCarterChord = chord.Chord(['C4','D-4','E4','F#4'])
elliottCarterChord.commonName
```

```
 'all-interval tetrachord'
```

```
elliottCarterChord.show()
```

More ways of creating chords; Chords and Streams¶

There are other ways of creating a Chord if you'd like. One way is from a bunch of already created `Note`

objects:

```
d = note.Note('D4')
fSharp = note.Note('F#4')
a = note.Note('A5')
dMajor = chord.Chord([d, fSharp, a])

dMajor.show()
```

Or we can pass a string with note names separated by spaces:

```
e7 = chord.Chord("E4 G#4 B4 D5")
e7.show()
```

The octaves are optional, especially if everything is within an octave:

```
es = chord.Chord("E- G B-")
es.show()
```

But you will definitely want them if a chord crosses the boundary of an octave (between B and C). Unless you love 6-4 chords, this is probably not what you want:

```
fMajor = chord.Chord("F A C")
fMajor.show()
```

Notice that because C sorts before F and A that the chord is in second inversion, or 64. We can figure out the inversion of a `Chord` like so:

```
print(fMajor.inversion(), fMajor.inversionName())
```

```
 2 64
```

In addition to .commonName, there are a few other "name" properties that might be interesting:

```
fMajor.fullName
```

```
 'Chord {F | A | C} Quarter'
```

```
fMajor.pitchedCommonName
```

```
 'F-major triad'
```

Like `Note` objects, we can put `Chord` objects inside a `Stream`:

```
stream1 = stream.Stream()
stream1.append(cMinor)
stream1.append(fMajor)
stream1.append(es)
stream1.show()
```

We can mix and match `Notes`, `Rests`, and `Chords`:

```
rest1 = note.Rest()
rest1.quarterLength = 0.5
noteASharp = note.Note('A#5')
noteASharp.quarterLength = 1.5

stream2 = stream.Stream()
stream2.append(cMinor)
stream2.append(rest1)
stream2.append(noteASharp)
stream2.show()
```

Post-tonal chords (in brief)¶

There are a lot of methods for dealing with post-tonal aspects of chords. If you're not interested in twentieth century music, go ahead and skip to the next chapter, but, here are some fun things.

The `intervalVector` of a chord is a list of the number of `[semitones, whole-tones, minor-thirds/augmented-seconds, major-thirds, perfect fourths, and tritones]` in the chord or inversion. A minor triad, for instance, has one minor third (C to E-flat), one major third (E-flat to G), and one perfect fourth (G to C above, since octave does not matter):

```
cMinor.intervalVector
```

```
 [0, 0, 1, 1, 1, 0]
```

A major triad has the same interval vector:

```
cMajor.intervalVector
```

```
 [0, 0, 1, 1, 1, 0]
```

The elliottCarterChord is unique in that it has an `.intervalVector` of all 1's:

```
elliottCarterChord.intervalVector
```

```
 [1, 1, 1, 1, 1, 1]
```

Well, it's almost unique: there is another chord with the same `.intervalVector`. That Chord is called its Z-relation or Z-pair.

```
elliottCarterChord.hasZRelation
```

```
 True
```

```
otherECChord = elliottCarterChord.getZRelation()
otherECChord
```

```
 <music21.chord.Chord C D- E- G>
```

```
otherECChord.show()
```

```
otherECChord.intervalVector
```

```
 [1, 1, 1, 1, 1, 1]
```

The other post-tonal tools you might be interested in are given below. We'll return to them in a later chapter, but here are three important ones:

```
print(elliottCarterChord.primeForm)
```

```
 [0, 1, 4, 6]
```

```
print(elliottCarterChord.normalOrder)
```

```
 [0, 1, 4, 6]
```

```
print(elliottCarterChord.forteClass)
```

```
 4-15A
```

If you really only care about semitones, you can create a chord just with the pitchClasses:

```
oddChord = chord.Chord([1, 3, 7, 9, 10])
oddChord.show()
```

There's a little problem with the A and A# being on the same space that makes it hard to read. Let's flip the A# to Bb:

```
oddChord.pitches[-1].getHigherEnharmonic(inPlace=True)
oddChord.show()
```

If you use pitchClasses above 11, then they are treated as MIDI numbers, where 60 = MiddleC, 72 = C5, etc. Enharmonic spelling is chosen automatically.

```
midiChordType = chord.Chord([60, 65, 70, 75])
midiChordType.show()
```

Okay, so now you've learned the basics (and more!) of Notes and Chords. If you haven't been able to see them on your own, Chapter 8: Installing MusicXML Readers will fix it. It's also going to cover the basic file formats of `music21`.

Navigation

Table of Contents

- User's Guide, Chapter 26: Stream Iteration and Filtering
  - Filtering elements in iteration
    - Filter Shortcuts
    - Custom Filters
  - Recursive and Offset Iterators
  - From Iterator to Stream

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings

- User's Guide, Chapter 25: Post-Tonal Tools (1)
- User's Guide, Chapter 26: Stream Iteration and Filtering
- User's Guide, Chapter 27: Grace Notes
- User's Guide, Chapter 28: Lyric Searching
- User's Guide, Chapter 29: Spanners 1 (Slurs)
- User's Guide, Chapter 30: Examples 3
- User's Guide, Chapter 31: Clefs, Ties, and Beams
- User's Guide, Chapter 32: Articulations
- User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
- User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
- User's Guide, Chapter 54: Extending Converter with New Formats
- User's Guide, Chapter 55: Advanced Meter Topics
- User's Guide, Chapter 58: Understanding Sites and Contexts
- User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 30: Examples 3
  - Roman Numeral Analysis on a Melody

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes
  - User's Guide, Chapter 28: Lyric Searching

- User’s Guide, Chapter 29: Spanners 1 (Slurs)
- User’s Guide, Chapter 30: Examples 3
- User’s Guide, Chapter 31: Clefs, Ties, and Beams
- User’s Guide, Chapter 32: Articulations
- User’s Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
- User’s Guide, Chapter 53: Advanced Corpus and Metadata Searching
- User’s Guide, Chapter 54: Extending Converter with New Formats
- User’s Guide, Chapter 55: Advanced Meter Topics
- User’s Guide, Chapter 58: Understanding Sites and Contexts
- User’s Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Quick search

This Page

- Show Source

User's Guide, Chapter 30: Examples 3¶

Since the last set of examples in Chapter 20, we've learned about sorting streams, graphs and plots, roman numerals, environment, post-tonal tools, iterators, grace notes, and spanners. Now let's put as many of these together as we can to analyze music computationally.

The corpus of `music21` includes the over 1050 songs, reels, jigs, from *Ryan's Mammoth Collection of Fiddle Tunes,* thanks to John Chambers and others. The collection was originally published in 1883 by Blake Howe after William Bradbury Ryan's work. Let's use these, which are called "ryansMammoth" in the corpus, for some examples.

(Note that some of the tunes in this corpus are based on uncredited tunes either by African-American composers or adapted from such tunes ["Minstrel Songs"], and for these reasons were attacked in *Dwight's Journal* as too "trashy" (1855, p. 118) for cosmopolitan New England tastes. The work of finding the 19th-century Black compositional geniuses behind the original tunes continues, and I hope we can supplement the adpated melodies in the corpus with originals in the near future. In the meantime, I hope that the computational studies of these works can stick a finger in the face of John Sullivan Dwight, if his myopic vision of American creativity could ever see it.)

First we'll load `music21` and search the corpus for these tunes:

```
from music21 import *
ryans = corpus.search('ryansMammoth')
ryans
```

```
 <music21.metadata.bundles.MetadataBundle {1059 entries}>
```

Let's look at one of these that I know is called 'Highland Regiment':

```
highland = ryans.search('Highland Regiment')
highland
```

```
 <music21.metadata.bundles.MetadataBundle {1 entry}>
```

```
highlandParsed = highland[0].parse()
highlandParsed.measures(0, 8).show()
```

Let's take our knowledge of spanners and see if the last note of a slur is generally higher or lower than the first note. We'll use a `RecursiveIterator` with a `ClassFilter` of 'Slur' to find them all:

```
highlandIterator = highlandParsed.recurse()
highlandSlurs = highlandIterator.getElementsByClass('Slur')

higher = 0
lower = 0
same = 0 # could happen for slurs longer than 2 notes

for sl in highlandSlurs:
    firstNote = sl.getFirst()
    lastNote = sl.getLast()
    psDiff = lastNote.pitch.ps - firstNote.pitch.ps
    if psDiff > 0:
        higher += 1
    elif psDiff < 0:
        lower += 1
    else:
        same += 1
```

```
(higher, lower, same)

 (19, 30, 0)
```

Hmmm… it looks like of the 49 slurs in this piece, more of them end lower than higher. Let's do this on a sample of the first 20 pieces in the collection. Let's augment our slur counting function a little bit, and make it safe in case a slur begins on a Chord, by taking the average pitch value of all the notes in the Chord, introduce a Counter object from Python's `collections` module, and go to it:

```
from statistics import mean
from collections import Counter

totalCounter = Counter()

def countOneSlur(sl, totalCounter):
    firstNote = sl.getFirst()
    lastNote = sl.getLast()
    if not hasattr(firstNote, 'pitches'):
        return
    if not hasattr(lastNote, 'pitches'):
        return

    firstNotePs = mean(p.ps for p in firstNote.pitches)
    lastNotePs = mean(p.ps for p in lastNote.pitches)
    psDiff = lastNotePs - firstNotePs
    if psDiff > 0:
        totalCounter['higher'] += 1
    elif psDiff < 0:
        totalCounter['lower'] += 1
    else:
        totalCounter['same'] += 1
```

Now let's make a function that takes in an object from `corpus.search` and parses it and runs each slur through `countOneSlur`.

```
def runOneScore(scCorpusSearchObject, totalCounter):
    scParsed = scCorpusSearchObject.parse()
    for sl in scParsed.recurse().getElementsByClass('Slur'):
        countOneSlur(sl, totalCounter)
```

Always important to test to make sure we haven't broken anything. This should give the same answer as before:

```
runOneScore(highland[0], totalCounter)

totalCounter

 Counter({'lower': 30, 'higher': 19})
```

It works as before, though there's no "same" entry in `totalCounter` since we did not encounter such a case. Let's reset our counter and run the first 20 pieces through the process:

```
totalCounter = Counter()

stop = 20
for piece in ryans:
    runOneScore(piece, totalCounter)
    stop = stop - 1
    if stop == 0:
        break

totalCounter

 Counter({'lower': 90, 'higher': 80, 'same': 11})
```

Hmmm… this still shows a few more "lower" cases than "higher" but not very much of a difference. In fact, the entire difference of 10 can be attributed to the first test example, since "Highland Regiment" is in this test

set. Maybe the notion that slurs more often end on lower notes than higher notes can be rejected, but I think we'd like to run the whole data set first. Simply remove the "stop" and "break" variables from above and the results will come back in a minute or two. I ran it and got these results:

```
Counter({'higher': 3321, 'lower': 3637, 'same': 425})
```

Again, it's about 10% more times that a slur ends below the first note than above – that's the same ratio as we saw for the first ten pieces, but it might be quite statistically significant given the much larger dataset. After all, if you flip a coin three times, it's not that unusual to get all heads or all tails, but if you flip a coin one thousand times, even 550 heads and 450 tails is quite significant as an indicator of bias in the coin.

So let's write a little test program that treats `'higher'` and `'lower'` as if they are coin flips and see how often we get results outside the range (in either direction) as our counter results. `numBiased` takes in a `Counter` object, and a number of trials to run, flips a coin as many times as the higher and lower values and prints out the fraction of the flip trials that lie outside the range. Optionally, it can print out the exceptional results along the way:

```
import random

def numBiased(inCounter, trials=100, printData=False):
    chanceTrials = 0
    totalSimulations = inCounter['higher'] + inCounter['lower']
    maxValue = max([inCounter['higher'], inCounter['lower']])
    minValue = min([inCounter['higher'], inCounter['lower']])
    for trialNum in range(trials):
        randCounter = Counter()
        for flipNum in range(totalSimulations):
            if random.randint(0, 1) == 1:
                outcome = 'higher'
            else:
                outcome = 'lower'
            randCounter[outcome] += 1
        if (randCounter['higher'] < maxValue
                and randCounter['higher'] > minValue):
            chanceTrials += 1
        elif printData:
            print(randCounter)
    return chanceTrials / trials
```

We'll run it first on the data from "Highland Regiment", printing out the intermediate values along the way:

```
numBiased(Counter({'higher': 19, 'lower': 30}), printData=True)
```

```
 Counter({'higher': 30, 'lower': 19})
 Counter({'higher': 31, 'lower': 18})
 Counter({'higher': 31, 'lower': 18})
 Counter({'lower': 32, 'higher': 17})
 Counter({'lower': 30, 'higher': 19})
 Counter({'lower': 30, 'higher': 19})
 Counter({'lower': 31, 'higher': 18})
 Counter({'lower': 31, 'higher': 18})
 Counter({'lower': 30, 'higher': 19})
 Counter({'higher': 30, 'lower': 19})
 Counter({'lower': 30, 'higher': 19})
 Counter({'higher': 30, 'lower': 19})
 Counter({'higher': 31, 'lower': 18})
 Counter({'lower': 31, 'higher': 18})
 Counter({'lower': 31, 'higher': 18})

 0.85
```

Most of the time, the results are within the range of Highland Regiment, but this is below the 0.95 threshold that would allow us to say that there's something happening here. The results for the first ten pieces are even closer to a null hypothesis:

```
numBiased(Counter({'higher': 80, 'lower': 90, 'same': 11}))
```

```
0.46
```

But, as I noted above, much larger trials may be much more significant. We will run it on the `Counter` object for the whole piece and see what the results are:

```
totalCounter = Counter({'higher': 3321, 'lower': 3637, 'same': 425})
numBiased(totalCounter)
```

```
1.0
```

Those are very significant results! There wasn't a single case where coins were flipped about 7000 times and we got fewer than 3321 or more than 3637 heads or tails! Thus we can definitely say that in this collection of fiddle tunes, you're more likely to slur down than up – the size of the results is small, but the significance is high.

Roman Numeral Analysis on a Melody¶

The majority of the pieces in *Ryan's Mammoth Collection* are single-voice pieces, which might make Roman Numeral analysis difficult, but actually it can be even more interesting this way. Let's take a tune from close to MIT's home, a reel called "The Boston":

```
bostonMD = ryans.search('The Boston -- Reel')
bostonMD
```

```
<music21.metadata.bundles.MetadataBundle {1 entry}>
```

```
boston = bostonMD[0].parse()
boston.measures(0, 9).show()
```

Now let's create a chord from each beat in the piece (skipping the pickup notes), aggregating all the pitches from within that beat. First, we'll create a new `Part` object that has the same framework of `Measures` as the original, along with time signatures, etc., but no notes. We'll use the `.template()` method but tell it not to fill it with rests.

```
bostonPart = boston.parts[0]
outPart = bostonPart.template(fillWithRests=False)
```

Like we've done before, we'll start from the smallest part and work outwards. Since we will be working with RomanNumerals, we'll have to know what key we are in; so we can analyze the piece; hopefully this will be F major.

```
pieceKey = boston.analyze('key')
pieceKey
```

```
 <music21.key.Key of F major>
```

Now let's make a routine that turns a list of Notes into a RomanNumeral object:

```
def notesToRoman(notes):
    uniquePitches = set(n.pitch for n in notes)
    ch = chord.Chord(list(uniquePitches))
    return roman.romanNumeralFromChord(ch, pieceKey)
```

Let's test it with the notes of measure 1:

```
noteIterator = bostonPart.measure(1).notes
rn1 = notesToRoman(noteIterator)
rn1
```

```
 <music21.roman.RomanNumeral I in F major>
```

Great, this is exactly what we're looking for. Now let's go into each measure that is full and analyze it separately.

Now we'll go into each measure that is full, and analyze it separately. We'll get everything from offset 0 to 1, then 1 to 2:

```
inMeasures = list(bostonPart[stream.Measure])
outMeasures = list(outPart[stream.Measure])

for i in range(14):
    inMeasure = inMeasures[i]
    if inMeasure.duration.quarterLength != 2.0:
        continue
    outMeasure = outMeasures[i]

    for beatStart in (0, 1):
        beatNotes = inMeasure.getElementsByOffset(beatStart,
                                                  beatStart + 1,
                                                  includeEndBoundary=False
                                                 ).getElementsByClass(note.NotRest)
        beatRN = notesToRoman(beatNotes)
        beatRN.lyric = beatRN.figure
        outMeasure.insert(beatStart, beatRN)

outPart.show()
```

Well, that's a great way to earn a C-minus in any music analysis class! Yes, all of these chords are correct, but only a few are useful for analysis. So let's do two things: (1) filter out all notes that are on a very weak position (the second or fourth sixteenth note) and are a passing or neighbor tone or something like that, and

(2) not show the figure for anything with the functionality of a "iii" chord or less. First let's redefine `notesToRoman` to do just that.

```
def notesToRoman(notes):
    goodPitches = set()
    for n in notes:
        if (n.offset * 2) == int(n.offset * 2):
            goodPitches.add(n.pitch)
        else:
            nPrev = n.previous('Note')
            nNext = n.next('Note')
            if nPrev is None or nNext is None:
                continue
            prevInterval = interval.Interval(n, nPrev)
            nextInterval = interval.Interval(n, nNext)

            if (prevInterval.generic.undirected in (1, 2)
                and nextInterval.generic.undirected in (1, 2)):
                pass
            else:
                goodPitches.add(n.pitch)
    ch = chord.Chord(list(goodPitches))
    return roman.romanNumeralFromChord(ch, pieceKey)
```

Now we can figure out that `functionalityScore` minimum:

```
iii = roman.RomanNumeral('iii')
iii.functionalityScore
```

```
 15
```

And let's re-run it:

```
outPart = bostonPart.template(fillWithRests=False)

inMeasures = list(bostonPart[stream.Measure])
outMeasures = list(outPart[stream.Measure])

for i in range(14):
    inMeasure = inMeasures[i]
    if inMeasure.duration.quarterLength != 2.0:
        continue
    outMeasure = outMeasures[i]

    for beatStart in (0, 1):
        beatNotes = inMeasure.getElementsByOffset(beatStart,
                                                  beatStart + 1,
                                                  includeEndBoundary=False
                                                 ).getElementsByClass(note.NotRest)
        beatRN = notesToRoman(beatNotes)
        if beatRN.functionalityScore > 15:
            beatRN.lyric = beatRN.figure
        outMeasure.insert(beatStart, beatRN)

outPart.show()
```

It's still not great, but we get a much better sense from this that the diminished chord on vii in root position is often being used in place of V as a dominant in this piece. Let's run the routine one more time and this time put the lyrics directly back in the score:

```
inMeasures = list(bostonPart[stream.Measure])

for i in range(14):
    inMeasure = inMeasures[i]
    if inMeasure.duration.quarterLength != 2.0:
        continue

    for beatStart in (0, 1):
        beatNotes = inMeasure.getElementsByOffset(beatStart,
                                                  beatStart + 1,
                                                  includeEndBoundary=False
                                                 ).getElementsByClass(note.NotRest)
        beatRN = notesToRoman(beatNotes)
        if beatRN.functionalityScore > 15:
            beatNotes[0].lyric = beatRN.figure

bostonPart.measures(0, 13).show()
```

This is looking pretty decent, but have no fears, we'll find more sophisticated score reduction tools in `music21` in the chapters to come.

Let's visualize this whole piece as a piano roll:

```
bostonPart.plot('pianoroll')
```

Here the centrality of the low F and the C5 stand out, along with an avoidance of the low tonic during the middle section, from measures 10-15, which I presume emphasizes the dominant harmony, but due to the presence of B-flats, is not *in* the dominant.

```
bostonPart.measures(10, 15).analyze('key')
```

```
 <music21.key.Key of F major>
```

It's clear that this piece is not chromatic at all, as a histogram of the pitch classes will show well:

```
bostonPart.plot('histogram', 'pitchClass')
```

Are there particular pitches – perhaps exploiting the open strings – that appear more often in this repertory? We can stitch a bunch of pieces together and see. We'll use an `Opus` Stream, which is a Stream that can hold other scores:

```
manyScores = stream.Opus()

for i in range(50):
    sc = ryans[i].parse()
    manyScores.insert(0, sc)

manyScores.plot('histogram', 'pitchClass')
```

It appears Boston Reel was “wicked queer” for using F major. Many more pieces emphasize notes of the open strings on the violin, with A, D, and E being the top three most used pitches (with G about tied with B).

Well, we didn’t get to everything since the last set of examples, but we got through a lot, and I hope it gives some sense of what you could do with your own repertory. Let’s now dial back the difficulty for the next section and look at some fundamental objects we’ve missed in Chapter 31: Clefs, Ties, and Beams.

Navigation

Table of Contents

- User's Guide, Chapter 10: Examples 1
  - Where do Chords move to?

Table of Contents

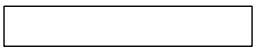
- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes
  - User's Guide, Chapter 28: Lyric Searching

- User’s Guide, Chapter 29: Spanners 1 (Slurs)
- User’s Guide, Chapter 30: Examples 3
- User’s Guide, Chapter 31: Clefs, Ties, and Beams
- User’s Guide, Chapter 32: Articulations
- User’s Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
- User’s Guide, Chapter 53: Advanced Corpus and Metadata Searching
- User’s Guide, Chapter 54: Extending Converter with New Formats
- User’s Guide, Chapter 55: Advanced Meter Topics
- User’s Guide, Chapter 58: Understanding Sites and Contexts
- User’s Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Quick search



This Page

- Show Source

User’s Guide, Chapter 10: Examples 1¶

Well, that’s long enough to go without doing some actual work! Let’s see what is possible to do so far.

Where do Chords move to?¶

Let’s take our favorite work so far, Bach’s BWV 66.6, and see where each chord goes.

We’ll begin, as always, with importing `music21` and parsing the work from the corpus.

```
from music21 import *
s = corpus.parse('bwv66.6')
```

Now let's chordify it.

```
sChords = s.chordify()
sChords
```

```
 <music21.stream.Part Soprano>
```

`sChords` is a Part containing no Chords! It has measures, which contain Chords. We're going to want to see all the chords, so let's flatten it with `.flatten()`.

```
sFlat = sChords.flatten()
sFlat
```

```
 <music21.stream.Part Soprano_flat>
```

Inside `sChords` is not just chords, but also the time signatures, etc., so let's create a Stream that only has the chords in sChords. And we'll get rid of all internal barlines, measures, etc. We'll use `.getElementsByClass('Chord')` for that:

```
sOnlyChords = sFlat.getElementsByClass('Chord')
sOnlyChords
```

```
 <music21.stream.iterator.StreamIterator for Part:Soprano_flat @:0>
```

We are eventually going to want to display each chord, so we'll create a new `Stream` to do so. We'll make it a `Part` object:

```
displayPart = stream.Part(id='displayPart')
displayPart
```

```
 <music21.stream.Part displayPart>
```

Now we're going to want to look at each chord and the following chord, so we want to go from the first chord, index `0`, to the second to last chord, index `len(sOnlyChords) - 1`, and then get each chord and the next chord. We will do that with the following sequence of commands

```
for i in range(0, len(sOnlyChords) - 1):
    thisChord = sOnlyChords[i]
    nextChord = sOnlyChords[i + 1]
    # do something with thisChord and nextChord here
```

Python hint: `range(min, max)` or `range(max)` is a "generator" function that yields every number starting with `min` (or zero if no min is given) to one less than the maximum number. `range()` is a really useful function, and we will use it all the time to do the same thing over a whole bunch of items. For instance:

```
for n in range(5):
    print(n)
```

```
 0
 1
 2
 3
 4
```

Then for each chord pair, we can create a new measure and append it to `displayPart` only if the first chord of each group is a triad or a seventh. For the purposes of making this example short, let's do it only if the first chord's root is "A". And we can put them all in closed position.

We can create a new function to do all this and call it `appendChordPairs()`

```
def appendChordPairs(thisChord, nextChord):
    if ((thisChord.isTriad() is True or
```

```
            thisChord.isSeventh() is True) and
                thisChord.root().name == 'A'):
        closePositionThisChord = thisChord.closedPosition(forceOctave=4)
        closePositionNextChord = nextChord.closedPosition(forceOctave=4)

        m = stream.Measure()
        m.append(closePositionThisChord)
        m.append(closePositionNextChord)
        displayPart.append(m)
```

Okay, so I think we have this set up. Let's replace "`# do something`" with `appendChordPairs()`:

```
for i in range(len(sOnlyChords) - 1):
    thisChord = sOnlyChords[i]
    nextChord = sOnlyChords[i + 1]
    appendChordPairs(thisChord, nextChord)
```

Do we have it? Let's see if `displayPart` has anything in it?

```
len(displayPart)
```

```
 7
```

Only seven pairs of chords, well, we did limit it to chords built on `A`. Let's see it!

```
displayPart.show()
```

Learning from the last chapter, we can label the chords with Roman Numerals in the key of A. Obviously, the first chord is always going to be "I", but where does "I" move to? That's more interesting. We'll cheat and use an analysis technique we'll later see in Chapter 23.

```
keyA = key.Key('A')
for c in displayPart.recurse().getElementsByClass('Chord'):
    rn = roman.romanNumeralFromChord(c, keyA)
    c.addLyric(str(rn.figure))

displayPart.show()
```

Well, it's basically everything that I expected, except for that `III6` chord! I smell a modulation happening here. Let's make all the pitches of that Chord pink so we can find them later. And we'll softly introduce a new concept, the `derivation.chain()` (see Chapter 13) to make sure that everything that this note comes from is also pink.

```
for c in displayPart.recurse().getElementsByClass('Chord'):
    if c.lyric == 'III6':
        c.style.color = 'pink'
        for x in c.derivation.chain():
            x.style.color = 'pink'

displayPart.show()
```

Now we can find this chord more easily:

```
sChords.show()
```

And we can analyze each Measure's key to show that indeed it is part of a modulation to f# minor:

```
for m in sChords.getElementsByClass('Measure'):
    k = m.analyze('key')
    print(m.number, k)
```

```
 0 E major
 1 E major
 2 A major
 3 f# minor
 4 E major
 5 A major
 6 f# minor
 7 C# major
 8 F# major
 9 b minor
```

This is a brief summary of some things we can do so far. I'd like to use other pieces besides this Chorale, so let's move on to Chapter 11: Corpus Searching to learn more.

Navigation

- index
- modules |
- next |

Table of Contents

- User's Guide, Chapter 2: Notes
  - Creating and working with Notes
    - (Advanced digression):

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering
  - User's Guide, Chapter 27: Grace Notes

    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 31: Clefs, Ties, and Beams
  - Automatic Clef Generation
  - Ties
    - Ties and chords
    - Making and Stripping Ties from a Stream
  - Beams
  - Beams the easy way

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis

    - User's Guide, Chapter 24: Configuring Environment Settings
    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - Attributes and Properties
    - `id`
    - Groups
    - ActiveSite
    - offset
    - priority
    - classSortOrder

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting

    - User's Guide, Chapter 23: Roman Numeral Analysis
    - User's Guide, Chapter 24: Configuring Environment Settings
    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 4: Lists, Streams (I) and Output
  - Working with multiple objects via Lists
  - Introduction to Streams
  - Creating simple Streams
  - Accessing Streams
    - Separating out elements by class with `.getElementsByClass()`
    - Separating out elements by offset with `.getElementsByOffset()`
  - More Stream Features
  - Streams within Streams

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements

    - User's Guide, Chapter 22: Graphing and plotting
    - User's Guide, Chapter 23: Roman Numeral Analysis
    - User's Guide, Chapter 24: Configuring Environment Settings
    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - Parsing Files in General
  - File types available to `music21`
  - Getting and working with MusicXML Files
  - Getting and working with Humdrum Files
  - Getting and working with ABC Files
    - ABC Opus files
    - ABC single-part Opus files
  - Parsing Musedata Files
  - Parsing MIDI Files
  - Conclusion

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)

    - User's Guide, Chapter 20: Examples 2
    - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
    - User's Guide, Chapter 22: Graphing and plotting
    - User's Guide, Chapter 23: Roman Numeral Analysis
    - User's Guide, Chapter 24: Configuring Environment Settings
    - User's Guide, Chapter 25: Post-Tonal Tools (1)
    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 11: Corpus Searching
  - Types of corpora
  - Local Corpus
  - Simple searches of the corpus
  - Metadata search fields

Table of Contents

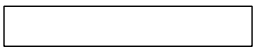
- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)

- User’s Guide, Chapter 26: Stream Iteration and Filtering
- User’s Guide, Chapter 27: Grace Notes
- User’s Guide, Chapter 28: Lyric Searching
- User’s Guide, Chapter 29: Spanners 1 (Slurs)
- User’s Guide, Chapter 30: Examples 3
- User’s Guide, Chapter 31: Clefs, Ties, and Beams
- User’s Guide, Chapter 32: Articulations
- User’s Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
- User’s Guide, Chapter 53: Advanced Corpus and Metadata Searching
- User’s Guide, Chapter 54: Extending Converter with New Formats
- User’s Guide, Chapter 55: Advanced Meter Topics
- User’s Guide, Chapter 58: Understanding Sites and Contexts
- User’s Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**
- Installation
- Developer Reference
- Documentation and tests in progress

Quick search



This Page

- Show Source

User's Guide, Chapter 11: Corpus Searching¶

One of music21's important features is its capability to help users examine large bodies of musical works, or *corpora*.

Music21 comes with a substantial corpus called the *core* corpus. When you download music21 you can immediately start working with the files in the corpus directory, including the complete chorales of Bach, many Haydn and Beethoven string quartets, three books of madrigals by Monteverdi, thousands of folk songs

from the Essen and various ABC databases, and many more.

To load a file from the corpus, simply call *corpus.parse* and assign that file to a variable:

```
from music21 import *
bach = corpus.parse('bach/bwv66.6')
```

The `music21` core corpus comes with many thousands of works. All of them (or at least all the collections) are listed on the Corpus Reference.

Users can also build their own corpora to index and quickly search their own collections on disk including multiple local corpora, for different projects, that can be accessed individually.

This user's guide will cover more about the corpus's basic features. This chapter focuses on music21's tools for extracting useful metadata - titles, locations, composers names, the key signatures used in each piece, total durations, ambitus (range) and so forth.

This metadata is collected in *metadata bundles* for each corpus. The *corpus* module has tools to search these bundles and persist them on disk for later research.

Types of corpora¶

Music21 works with three categories of *corpora*, made explicit via the `corpus.Corpus` abstract class.

The first category is the *core* corpus, a large collection of musical works packaged with most music21 installations, including many works from the common practice era, and inumerable folk songs, in a variety of formats:

```
coreCorpus = corpus.corpora.CoreCorpus()
len(coreCorpus.getPaths())
```

```
 3193
```

Note

If you've installed a "no corpus" version of music21, you can still access the *core* corpus with a little work. Download the *core* corpus from music21's website, and install it on your system somewhere. Then, teach music21 where you installed it like this:

```
>>> coreCorpus = corpus.corpora.CoreCorpus()
>>> coreCorpus.manualCoreCorpusPath = 'path/to/core/corpus'
```

Local Corpus¶

`Music21` also can have one or more *local* corpora–bodies of works provided and configured by individual music21 users for their own research. They will be covered in Chapter 53. Anyone wanting to use them can jump ahead immediately to that chapter, but for now we'll continue with searching in the core corpus.

```
localCorpus = corpus.corpora.LocalCorpus()
```

You can add and remove paths from a *local* corpus with the `addPath()` and `removePath()` methods:

```
localCorpus.addPath('~/Desktop')
localCorpus.directoryPaths
```

```
 ('/Users/myke/Desktop',)
```

Currently, after adding paths to a corpus, you'll need to rebuild the cache.

```
corpus.cacheMetadata()
```

We hope that this won't be necessary in the future.

To remove a path, use the `removePath()` method.

```
localCorpus.removePath('~/Desktop')
```

```
 /Users/cuthbert/git/music21base/music21/corpus/corpora.py: WARNING: local metadata cache:
 /Users/cuthbert/git/music21base/music21/corpus/corpora.py: WARNING: cache: filename: /Use
 metadata.bundles: WARNING: MetadataBundle Modification Time: 1686436276.496933
 metadata.bundles: WARNING: Skipped 0 sources already in cache.
 /Users/cuthbert/git/music21base/music21/corpus/corpora.py: WARNING: cache: writing time: (

 /Users/cuthbert/git/music21base/music21/corpus/corpora.py: WARNING: cache: filename: /Use
```

By default, a call to `corpus.parse` or `corpus.search` will look for files in any corpus, core or local.

Simple searches of the corpus¶

When you search the corpus, music21 examines each metadata object in the metadata bundle for the whole corpus and attempts to match your search string against the contents of the various search fields saved in that metadata object.

You can use `corpus.search()` to search the metadata associated with all known corpora, *core*, *virtual* and even each *local* corpus:

```
sixEight = corpus.search('6/8')
sixEight
```

```
 <music21.metadata.bundles.MetadataBundle {2164 entries}>
```

To work with all those pieces, you can parse treat the MetadataBundle like a list and call `.parse()` on any element:

```
myPiece = sixEight[0].parse()
myPiece.metadata.title
```

```
 "I'll Touzle your Kurchy."
```

This will return a `music21.stream.Score` object which you can work with like any other stream. Or if you just want to see it, there's a convenience `.show()` method you can call directly on a MetadataEntry.

You can also search against a single `Corpus` instance, like this one which ignores anything in your local corpus:

```
corpus.corpora.CoreCorpus().search('6/8')
```

```
 <music21.metadata.bundles.MetadataBundle {2164 entries}>
```

Because the result of every metadata search is also a metadata bundle, you can search your search results to do more complex searches. Remember that `bachBundle` is a collection of all works where the composer is Bach. Here we will limit to those pieces in 3/4 time:

```
bachBundle = corpus.search('bach', 'composer')
bachBundle
```

```
 <music21.metadata.bundles.MetadataBundle {363 entries}>
```

```
bachBundle.search('3/4')
```

```
 <music21.metadata.bundles.MetadataBundle {40 entries}>
```

Metadata search fields¶

When you search metadata bundles, you can search either through every search field in every metadata instance, or through a single, specific search field. As we mentioned above, searching for “bach” as a composer renders different results from searching for the word “bach” in general:

```
corpus.search('bach', 'composer')
```

```
 <music21.metadata.bundles.MetadataBundle {363 entries}>
```

```
corpus.search('bach', 'title')
```

```
 <music21.metadata.bundles.MetadataBundle {20 entries}>
```

```
corpus.search('bach')
```

```
 <music21.metadata.bundles.MetadataBundle {564 entries}>
```

So what fields can we actually search through? You can find out like this (in v2, replace `corpus.manager` with `corpus.corpora.Corpus`):

```
for field in corpus.manager.listSearchFields():
    print(field)
```

```
 abstract
 accessRights
 accompanyingMaterialWriter
 actNumber
 adapter
 afterwordAuthor
 alternativeTitle
 ambitus
 analyst
 annotator
 arranger
 associatedWork
 attributedComposer
 audience
 bibliographicCitation
 calligrapher
 collaborator
 collectionDesignation
 collotyper
 commentaryAuthor
 commission
 commissionedBy
 compiler
 composer
 composerAlias
 composerCorporate
 conceptor
 conductor
 conformsTo
 copyright
 corpusFilePath
 countryOfComposition
 date
 dateAccepted
 dateAvailable
 dateCopyrighted
 dateCreated
 dateFirstPublished
 dateIssued
 dateModified
 dateSubmitted
 dateValid
 dedicatedTo
 dedication
 description
 dialogAuthor
 distributor
```

editor
educationLevel
electronicEditor
electronicEncoder
electronicPublisher
electronicReleaseDate
engraver
etcher
extent
fileFormat
fileNumber
filePath
firstPublisher
format
groupTitle
hasFormat
hasPart
hasVersion
identifier
illuminator
illustrator
instructionalMethod
instrumentalist
introductionAuthor
isFormatOf
isPartOf
isReferencedBy
isReplacedBy
isRequiredBy
isVersionOf
keySignatureFirst
keySignatures
language
librettist
license
lithographer
localeOfComposition
lyricist
manuscriptAccessAcknowledgement
manuscriptLocation
manuscriptSourceName
medium
metalEngraver
movementName
movementNumber
musician
noteCount
number
numberOfParts
opusNumber
orchestrator
originalDocumentOwner
originalEditor
otherContributor
otherDate
parentTitle
pitchHighest
pitchLowest
placeFirstPublished
platemaker
popularTitle
printmaker
producer
proofreader
provenance
publicationTitle
publisher
publishersCatalogNumber
quarterLength

```
 quotationsAuthor
 references
 relation
 replaces
 requires
 responsibleParty
 rightsHolder
 sceneNumber
 scholarlyCatalogAbbreviation
 scholarlyCatalogName
 scribe
 singer
 software
 source
 sourcePath
 subject
 suspectedComposer
 tableOfContents
 tempoFirst
 tempos
 textLanguage
 textOriginalLanguage
 timeSignatureFirst
 timeSignatures
 title
 transcriber
 translator
 type
 volume
 volumeNumber
 woodCutter
 woodEngraver
```

This field has grown now that the development team is seeing how useful this searching method can be! Now that we know what all the search fields are, we can search through some of the more obscure corners of the *core* corpus:

```
corpus.search('taiwan', 'locale')
```

```
 <music21.metadata.bundles.MetadataBundle {27 entries}>
```

What if you are not searching for an exact match? If you're searching for short pieces, you probably don't want to find pieces with exactly 1 note then union that set with pieces with exactly 2 notes, etc. Or for pieces from the 19th century, you won't want to search for 1801, 1802, etc. What you can do is set up a "predicate callable" which is a function (either a full python `def` statement or a short `lambda` function) to filter the results. Each piece will be checked against your predicate and only those that return true. Here we'll search for pieces with between 400 and 500 notes, only in the `core` corpus:

```
predicate = lambda x: 400 < x < 500
corpus.corpora.CoreCorpus().search(predicate, 'noteCount')
```

```
 <music21.metadata.bundles.MetadataBundle {213 entries}>
```

You can also pass in compiled regular expressions into the search. In this case we will use a regular expression likely to find Handel and Haydn and perhaps not much else:

```
import re
haydnOrHandel = re.compile(r'ha.d.*', re.IGNORECASE)
corpus.search(haydnOrHandel)
```

```
 <music21.metadata.bundles.MetadataBundle {186 entries}>
```

Unfortunately this really wasn't a good search, since we also got folk songs with the title of "Shandy". Best to use a '*^*' search to match at the beginning of the word only:

```
haydnOrHandel = re.compile(r'^ha.d.*', re.IGNORECASE)
corpus.search(haydnOrHandel)
```

```
<music21.metadata.bundles.MetadataBundle {15 entries}>
```

We've now gone fairly high level in our searching. We will return to the lowest level in Chapter 12: The Music21Object

Navigation

Table of Contents

- User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - Lists of Lists
  - Functions and Recursion
  - Wrapup

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)
  - User's Guide, Chapter 26: Stream Iteration and Filtering

    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress

Table of Contents

- User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - Classes and Subclasses
  - Accessing Scores, Parts, Measures, and Notes
  - Recursion in Streams
  - Flattening a Stream

Table of Contents

- About *music21*

- User's Guide
  - User's Guide: Table of Contents
  - User's Guide, Chapter 1: Installing and Getting Started with `music21`
  - User's Guide, Chapter 2: Notes
  - User's Guide, Chapter 3: Pitches, Durations, and Notes again
  - User's Guide, Chapter 4: Lists, Streams (I) and Output
  - User's Guide, Chapter 5: Lists of Lists, Functions, and Recursion
  - User's Guide, Chapter 6: Streams (II): Hierarchies, Recursion, and Flattening
  - User's Guide, Chapter 7: Chords
  - User's Guide, Chapter 8: Installing MusicXML Readers and File Formats (1)
  - User's Guide, Chapter 9: Chordify
  - User's Guide, Chapter 10: Examples 1
  - User's Guide, Chapter 11: Corpus Searching
  - User's Guide, Chapter 12: Getting Back to Basics: The Music21Object
  - User's Guide, Chapter 13: More Music21Object Attributes and Properties
  - User's Guide: Chapter 14: Time Signatures and Beats
  - User's Guide, Chapter 15: Keys and KeySignatures
  - User's Guide, Chapter 16: TinyNotation
  - User's Guide: Chapter 17: Derivations
  - User's Guide: Chapter 18: Intervals
  - User's Guide, Chapter 19: Advanced Durations (Complex and Tuplets)
  - User's Guide, Chapter 20: Examples 2
  - User's Guide, Chapter 21: Ordering and Sorting of Stream Elements
  - User's Guide, Chapter 22: Graphing and plotting
  - User's Guide, Chapter 23: Roman Numeral Analysis
  - User's Guide, Chapter 24: Configuring Environment Settings
  - User's Guide, Chapter 25: Post-Tonal Tools (1)

    - User's Guide, Chapter 26: Stream Iteration and Filtering
    - User's Guide, Chapter 27: Grace Notes
    - User's Guide, Chapter 28: Lyric Searching
    - User's Guide, Chapter 29: Spanners 1 (Slurs)
    - User's Guide, Chapter 30: Examples 3
    - User's Guide, Chapter 31: Clefs, Ties, and Beams
    - User's Guide, Chapter 32: Articulations
    - User's Guide, Chapter 44: Advanced Graphing (Axes, Plots, and Graphs)
    - User's Guide, Chapter 53: Advanced Corpus and Metadata Searching
    - User's Guide, Chapter 54: Extending Converter with New Formats
    - User's Guide, Chapter 55: Advanced Meter Topics
    - User's Guide, Chapter 58: Understanding Sites and Contexts
    - User's Guide, Chapter 61: TimespanTrees and Verticalities

- **Module Reference**

- Installation

- Developer Reference

- Documentation and tests in progress