Music21 Quick Start

This quick start guide shows you how to install and use the `music21` package. This guide is for Python developers familiar with musical notation and terminology.

The `music21` package lets you programmatically create musical notation and analyze music. Learn more about `music21` from What is Music21 in the package documentation.

Follow this tutorial to learn how to perform the following actions using `music21`:

- Install the Music21 Package
- Add Notes to a Staff
- Set the Key and Time Signature
- Play Back Your Music
- Analyze Your Music

Learn more about the modules and methods in this guide from the API Documentation section.

Install the Music21 Package

Run the following command to install `music21` and its dependencies on this notebook:

```
!pip install music21
```

```
Collecting music21
  Downloading music21-9.3.0-py3-none-any.whl.metadata (5.0 kB)
Collecting chardet (from music21)
  Downloading chardet-5.2.0-py3-none-any.whl.metadata (3.4 kB)
Requirement already satisfied: joblib in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from music21) (1.4.2)
Collecting jsonpickle (from music21)
  Using cached jsonpickle-3.3.0-py3-none-any.whl.metadata (8.3 kB)
Collecting matplotlib (from music21)
  Downloading matplotlib-3.9.2-cp312-cp312-macosx_10_12_x86_64.whl.metadata (11 kB)
Collecting more-itertools (from music21)
  Downloading more_itertools-10.5.0-py3-none-any.whl.metadata (36 kB)
Requirement already satisfied: numpy in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from music21) (2.1.1)
Requirement already satisfied: requests in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from music21) (2.32.3)
Requirement already satisfied: webcolors>=1.5 in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from music21) (24.8.0)
Collecting contourpy>=1.0.1 (from matplotlib->music21)
  Downloading contourpy-1.3.0-cp312-cp312-macosx_10_9_x86_64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib->music21)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib->music21)
  Downloading fonttools-4.54.1-cp312-cp312-macosx_10_13_universal2.whl.metadata (163 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib->music21)
  Downloading kiwisolver-1.4.7-cp312-cp312-macosx_10_9_x86_64.whl.metadata (6.3 kB)
Requirement already satisfied: packaging>=20.0 in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from matplotlib->music21) (24.1)
Collecting pillow>=8 (from matplotlib->music21)
  Downloading pillow-11.0.0-cp312-cp312-macosx_10_13_x86_64.whl.metadata (9.1 kB)
Collecting pyparsing>=2.3.1 (from matplotlib->music21)
  Downloading pyparsing-3.2.0-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from matplotlib->music21) (2.9.0.post0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from requests->music21) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from requests->music21) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from requests->music21) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from requests->music21) (2024.8.30)
Requirement already satisfied: six>=1.5 in /Users/christophercho/.pyenv/versions/3.12.4/lib/python3.12/site-packages (from python-dateutil>=2.7->matplotlib->music21) (1.16.0)
```

```
Downloading music21-9.3.0-py3-none-any.whl (22.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22.9/22.9 MB 39.0 MB/s eta 0:00:00 MB/s eta 0:00:0101
Downloading chardet-5.2.0-py3-none-any.whl (199 kB)
Using cached jsonpickle-3.3.0-py3-none-any.whl (42 kB)
Downloading matplotlib-3.9.2-cp312-cp312-macosx_10_12_x86_64.whl (7.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.9/7.9 MB 35.1 MB/s eta 0:00:00m eta 0:00:01
Downloading more_itertools-10.5.0-py3-none-any.whl (60 kB)
Downloading contourpy-1.3.0-cp312-cp312-macosx_10_9_x86_64.whl (267 kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.54.1-cp312-cp312-macosx_10_13_universal2.whl (2.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.8/2.8 MB 27.3 MB/s eta 0:00:00
Downloading kiwisolver-1.4.7-cp312-cp312-macosx_10_9_x86_64.whl (65 kB)
Downloading pillow-11.0.0-cp312-cp312-macosx_10_13_x86_64.whl (3.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 28.5 MB/s eta 0:00:00
Downloading pyparsing-3.2.0-py3-none-any.whl (106 kB)
Installing collected packages: pyparsing, pillow, more-itertools, kiwisolver, jsonpickle, fonttools, cycler, contourpy, chardet, matplotlib, music21
Successfully installed chardet-5.2.0 contourpy-1.3.0 cycler-0.12.1 fonttools-4.54.1 jsonpickle-3.3.0 kiwisolver-1.4.7 matplotlib-3.9.2 more-itertools-10.5.0 music21-9.3.0 pillow-11.0.0 pyparsing-3.

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
```

Import the Music21 Package Functions

Add the following import statement to access the functions in the `music21` package:

```
from music21 import *
```

Add Notes to a Staff

To add a sequence of notes, create a **Stream**. A Stream is a container that stores sequential music data, including the following types:

- Notes
- Time signatures
- Key signatures
- Clefs
- Measures

To create a Stream object, call the `Stream()` method from the `stream` module.

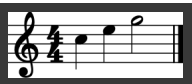
To add notes, append **Note** objects to the Stream sequentially. A Note defines the attributes of a musical note. To create a Note, specify the pitch and duration in the constructor `Note()` from the `note` module.

To view the Stream container as musical notation, call the `show()` method on the Stream object.

Run the following code to add a sequence of notes to a Stream and display the musical notation:

```
noteStream = stream.Stream()
noteStream.append(note.Note("C5", type='quarter'))
noteStream.append(note.Note("E5", type='quarter'))
noteStream.append(note.Note("G5", type='half'))

noteStream.show()
```



Optional: Add a Chord to a Staff

To add a chord, append a **Chord** object to the Stream. A Chord consists of a group of two or more musical notes that belong to the same beat. It shares a base class with Note. To construct a chord, specify the pitches in an array and the duration in the constructor `Chord()` from the `chord` module and add the Chord to your Stream.

Run the following code to add a sample chord to your Stream and display it:

```
gMajorChord = chord.Chord(['G4','B4','D5'], type='whole')
noteStream.append(gMajorChord)

noteStream.show()
```

Set the Key and Time Signature

To set the musical key, append a **Key** object to the Stream. A Key represents the information related to the key signature and can include the associated scale. To construct a Key, specify the name of the key in the `Key()` method from the `key` package, using capitalization to represent major or minor mode. For example, the value `'e'` represents the key of *E minor*, and the value `'E'` represents the key of *E major*.

Note: You can use **KeySignature** instead of a Key, but it omits the relationship between the key and the tonic in a diatonic scale. You must use a Key instead of a KeySignature for harmonic analysis.

To set the time signature, append a **TimeSignature** object to the Stream. A TimeSignature contains a value that represents the music's meter. To construct a TimeSignature, specify the time value in the `TimeSignature()` method from the `meter` package. For example, a value of `'2/4'` sets the length of a measure to *two quarter notes*.

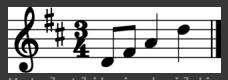
Run the following code to clear your Stream, set the key and time signatuures, and add a few notes:

```
# clear() removes all information from the stream
noteStream.clear()

noteStream.append(meter.TimeSignature('3/4'))
noteStream.append(key.Key('D'))
noteStream.append(clef.TrebleClef())

noteStream.append(note.Note('D4', type='eighth'))
noteStream.append(note.Note('F#4', type='eighth'))
noteStream.append(note.Note('A4', type='quarter'))
noteStream.append(note.Note('D5', type='quarter'))

noteStream.show()
```



```
Matplotlib is building the font cache; this may take a moment.
```

Play Back Your Music

To render a widget that lets you play your Stream using the MIDI protocol, pass the value `'midi'` to the `show()` method. This call generates a MIDI representation of the notes in the Stream.

To learn about additional output options, see the Stream.show() API documentation.

Run the following code to create a widget that plays your Stream in MIDI format:

```
noteStream.show('midi')
```

Analyze Your Music

To get the harmonic function of a chord, pass the Chord and Key to the `romanNumeralFromChord()` method in the `roman` module.

Roman numeral analysis describes the chord characteristics such as the scale degree, inversion, and interval spacing. To learn more about this type of analysis, see Roman numeral analysis on Wikipedia.

Run the following code to create a sample chord progression, print the Roman numeral analysis, and display the musical notation:

```
chordProgressionStream = stream.Stream()

fMajorKey = key.Key('F')

# Set the time signature, key, and clef
chordProgressionStream.append(meter.TimeSignature('4/4'))
chordProgressionStream.append(key.Key('F'))
chordProgressionStream.append(clef.TrebleClef())

# Create a circle of fourths chord progression
chords = [
    chord.Chord(['F4','A4','C5'], type='quarter'),
    chord.Chord(['B-4','D5','F5'], type='quarter'),
    chord.Chord(['E4','G4','B-4'], type='quarter'),
    chord.Chord(['A4','C5','E5'], type='quarter'),
    chord.Chord(['D4','F4','A4'], type='quarter'),
    chord.Chord(['G4','B-4','D5'], type='quarter'),
    chord.Chord(['C5','E5','G5'], type='quarter'),
    chord.Chord(['F4','A4','C5'], type='quarter'),
]

# Add the chords
print('Roman numeral analysis of the chords:\n')
for c in chords:
    chordProgressionStream.append(c)

# Print the roman numeral analysis of the chords in the Stream
for element in chordProgressionStream:
    if isinstance(element, chord.Chord):
        print(roman.romanNumeralFromChord(element, fMajorKey))
```

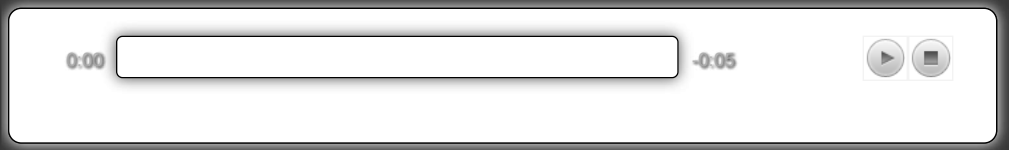
```
chordProgressionStream.show()
```

```
Roman numeral analysis of the chords:

<music21.roman.RomanNumeral I in F major>
<music21.roman.RomanNumeral IV in F major>
<music21.roman.RomanNumeral viio in F major>
<music21.roman.RomanNumeral iii in F major>
<music21.roman.RomanNumeral vi in F major>
<music21.roman.RomanNumeral ii in F major>
<music21.roman.RomanNumeral V in F major>
<music21.roman.RomanNumeral I in F major>
```

Run the following code to near the chord progression:

```
chordProgressionStream.show('midi')
```



Summary

After completing this quick start guide, you should have an understanding of the following concepts:

- Relationship between the referenced music data classes
- Creation of notes and chords
- Generation of musical notation and playback
- Harmonic analysis of chords

To continue learning about this package, read the Music21 package documentation and search for other online resources such as this Music21 YouTube Playlist.

API Documentation

To learn more about the modules and methods mentioned in this guide, see the following API documentation:

Object Type	Module Name	Method Name and Documentation Link
Stream	`stream`	`Stream()`
Note	`note`	`Key()`
Pitch	`pitch`	`Pitch()`
Duration	`duration`	`Duration()`
Chord	`chord`	`Chord()`
Clef	`clef`	`TrebleClef()`
Key	`key`	`Key()`
TimeSignature	`meter`	`TimeSignature()`
RomanNumeral	`roman`	`romanNumeralFromChord()`