COMPUTERGESTÜTZTE MUSIKFORSCHUNG 1

Institut für Musikinformatik und Musikwissenschaft
Wintersemester 2025–26

Christophe Weis
christophe.weis@stud.hfm.eu

Woche 02
21.10.2025

Organisation

wöchentlich, Di. 14.30–16.00, K10 Raum 309

Modul **Music Processing**

- **BA MI (HF)/MW (EF), wiss. Schwerpunkt:** Pflicht (4. Semester)
- **BA MI (HF)/MW (EF), künstl. Schwerpunkt:** Wahlpflicht (6. Semester)
- **BA MW (HF)/MI (EF):** Pflicht (4. Semester) – reduzierter Arbeitsaufwand
- **BA MI/MW (KF):** Pflicht (4. Semester)
- **BA:** Wahlfach

Projektarbeit

- eine selbstständige praktische Arbeit aus den Bereichen Musikkodierung, symbolbasierte Musikverarbeitung und –analyse mit Dokumentation (ca. 5000 Zeichen)

Übungen

- Tutorin: Joanna Friedrich-Sroka
- wöchentlich, Di. 11.15–12.45, K10 Raum 309

06.

music21
Objekte & Streams

Grundlagen

- Ausführliche music21-Documentation: https://www.music21.org/music21docs/index.html

- Installation & Upgrade:
  - `pip install music21`
  - `pip install music21 --upgrade`

- Python-Bibliothek importieren:
  - `import music21 as m21`
  - oder: `from music21 import *`

- Beispiel für die Konfiguration der Kompatibilität mit *MuseScore:*

```
us = m21.environment.UserSettings()
us['musescoreDirectPNGPath'] = "C:/Program Files/MuseScore 4/bin/MuseScore4.exe"
us['musicxmlPath'] = "C:/Program Files/MuseScore 4/bin/MuseScore4.exe"
```

Grundlagen

- Laden von lokalen Dateien:

  ```
  my_score = m21.converter.parse("Dateipfad")
  ```

  → Die Methode `.parse` liefert die Datei als **Score-Stream** (falls das Datei-Format unterstützt wird).

- Anzeige von music21-Objekten und Ausgabe von Notentext:

  ```
  some_object.show()
  some_object.show('text')
  some_object.show('musicXML')
  ```

- Exportieren von in music21 erstellten oder bearbeiteten Score-Streams:

  ```
  my_score.write("musicxml", "gewünschter Dateipfad")
  my_score.write("midi", "gewünschter Dateipfad")
  ```

music21-Korpus

- Die music21-Library enthält einen Korpus von symbolischen Datensätzen von Musikstücken (MusicXML und einige andere Formate).

- Überblick über alle im Korpus enthaltenen Werke:
  https://www.music21.org/music21docs/about/referenceCorpus.html

- Zum Laden eines Datensatzes:

```
bach_corpus = m21.corpus.search('bach', fileExtensions='xml')
```

- Werke eines Datensatzes können (wie externe Dateien) mithilfe der Methode `.parse` in einen **Score-Stream** geladen werden:

```
first_piece = bach_corpus[0]
first_piece_score = first_piece.parse()
```

- Ebenso können einzelne Stücke aus dem Korpus herausgelesen werden:

```
some_piece_score = m21.corpus.parse('schumann_clara/opus17/movement3.xml')
```

Objekte und Attribute

- Implementierung typischer musikalischer Konzepte als **Objekte** innerhalb von **Modulen**
- Auflistung aller Module: https://www.music21.org/music21docs/moduleReference/index.html

- *Beispiel:*
  Das **Modul** `m21.note` enthält die **Klassen** `m21.note.Note` und `m21.note.Rest`
  → erlauben es, „`m21.note.Note`“- und „`m21.note.Rest`“-Objekte zu erstellen

- Erstellung eines „`m21.note.Note`“-Objekts:

```
middle_c = m21.note.Note('C4')
middle_c → <music21.note.Note C>
```

- Verwendung von **Attributen** und **Subattributen**:

```
middle_c.name → 'C'
```

- Manche Attribute erzeugen neue Objekte (mit weiteren Attributen):

```
middle_c.pitch → <music21.pitch.Pitch C4>
middle_c.pitch.frequency → 261.6255653005985
```

Streams

- **Streams** funktionieren in music21 als **Container** für music21-Objekte und funktionieren **ähnlich wie Listen**.
- Positionen von Elementen innerhalb von Streams werden zusätzlich mithilfe des **zeitlichen Offsets seit Anfang des Streams** definiert.
- *Beispiel:*

```
c_major = m21.key.Key('C')
note1 = m21.note.Note('C4')
note2 = m21.note.Note('D4')

my_stream = m21.stream.Stream()
my_stream.append(CMajor)
my_stream.append(note1)
my_stream.append(note2)

my_stream.show('text')  →  {0.0} <music21.key.Key of C major>
                           {0.0} <music21.note.Note C>
                           {1.0} <music21.note.Note D>
```

Streams

- `len(my_stream)` → `3`
  `my_stream[1]` → `<music21.key.Key of C major>`
  `my_stream[-1]` → `<music21.note.Note D>`

- Die Methode `.getElementsByClass` erlaubt es, alle Elemente einer bestimmten Klasse aus Streams herauslesen:

```
my_stream.getElementsByClass(m21.note.Note).show('text')

→   {0.0} <music21.note.Note C>
    {1.0} <music21.note.Note D>

for thing in my_stream.getElementsByClass(m21.note.Note):
    print(thing.pitch)

→   C4
    D4
```

Substreams

- Streams können weitere Streams enthalten:

```
note3 = m21.note.Note('E4')

another_stream = m21.stream.Stream()
another_stream.append(myStream)
another_stream.append(note3)

len(another_stream) → 2

another_stream.getElementsByClass(m21.note.Note).show('Text')
→       {0.0} <music21.stream.Stream 0x1c49c0a0dd0>
        {0.0} <music21.key.Key of C major>
        {0.0} <music21.note.Note C>
        {1.0} <music21.note.Note D>
        {2.0} <music21.note.Note E>
```

- Je nach hierarchischer Position innerhalb eines Systems von ineinander verschachtelten Streams, definiert eine Stream eine bestimmte Stream-Subklasse (**Score**, **Part**, **Measure** oder **Voice**).

Flattening & Recursion

- **Flattening** erlaubt es, eine Verschachtelung von Streams in einen einzigen Stream umzuwandeln
  → der resultierende Stream enthält alle Elemente, die keine Stream-Subklassen sind.

```
another_stream.flatten().show('text')
→   {0.0} <music21.key.Key of C major>
    {0.0} <music21.note.Note C>
    {1.0} <music21.note.Note D>
    {2.0} <music21.note.Note E>
```

- **Recursion** erlaubt es, über alle Elemente eines Streams und dessen Substreams zu iterieren.

```
another_stream.recurse().show('text')
→   {0.0} <music21.key.Key of C major>
    {0.0} <music21.note.Note C>
    {1.0} <music21.note.Note D>
    {2.0} <music21.note.Note E>

len(another_stream.recurse().getElementsByClass(m21.note.Note))
→   3
```

Flattening & Recursion

- **Flattening** erlaubt es, eine Verschachtelung von Streams in einen einzigen Stream umzuwandeln
  → der resultierende Stream enthält alle Elemente, die keine Stream-Subklassen sind.

```
another_stream.flatten().show('text')
→    {0.0} <music21.key.Key of C major>
     {0.0} <music21.note.Note C>
     {1.0} <music21.note.Note D>
     {2.0} <music21.note.Note E>
```

→ **Nützlich z. B. um die Offsets aller Objekte in Bezug auf den Anfang eines Stückes zu bestimmen.**

- **Recursion** erlaubt es, über alle Elemente eines Streams und dessen Substreams zu iterieren.

```
another_stream.recurse().show('text')
→    {0.0} <music21.key.Key of C major>
     {0.0} <music21.note.Note C>
     {1.0} <music21.note.Note D>
     {2.0} <music21.note.Note E>

len(another_stream.recurse().getElementsByClass(m21.note.Note))
→    3
```

→ **Nützlich für Anwendungen, bei denen Takte und Stimmen „nicht verloren gehen sollen“ (z. B. wenn man die Anzahl an Takten zählen will).**