COMPUTERGESTÜTZTE MUSIKFORSCHUNG 1

Institut für Musikinformatik und Musikwissenschaft
Wintersemester 2025–26

Christophe Weis
christophe.weis@stud.hfm.eu

Woche 03
28.10.2025

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

07.

Einige nützliche Python-Tools

Listen sortieren

- Python-Listen **sortieren** mit dem *key*-Parameter:

  Beispiel 1:

```
my_list = [0, 3, -2, 1]
sorted_list1 = sorted(my_list)
sorted_list2 = sorted(my_list, key=lambda number: number**2)
print(sorted_list1) → [-2, 0, 1, 3]
print(sorted_list2) → [0, 1, -2, 3]
```

  Beispiel 2:

```
my_list = [('First', 1), ('Fourth', 4), ('Second', 2), ('Fifth', 5), ('Third', 3)]

sorted_list = sorted(my_list, key = lambda tuple_in_my_list: tuple_in_my_list[1])

print(sorted_list)
→ [('First', 1), ('Second', 2), ('Third', 3), ('Fourth', 4), ('Fifth', 5)]
```

List-Comprehension

- **List-Comprehension** in Python:

```
my_list = [0, 1, 2, 3, 4, 5]

squares = [number**2 for number in my_list]

print(squares)
→ [0 ,1, 4, 9, 16, 25]
```

Elemente in Listen zählen

- Elemente in Python-Listen **zählen**:

```
from collections import Counter

my_list = ['a', 'b', 'b', 'b', 'c', 'c']

counted_list = Counter(my_list)
print(counted_list)
→ Counter({'b': 3, 'c': 2, 'a': 1})

counted_items = list(counted_list.items())
print(counted_items)
→ [('a', 1), ('b', 3), ('c', 2)]
```

Listen kombinieren

- Zwei Python-Listen **aneinanderreihen**:

```
[1, 2, 3] + ['a', 'b']
→ [1, 2, 3, 'a', 'b']
```

Strings vergleichen

- **Strings vergleichen** und *.upper*- und *.lower*-Methoden:

```
my_string1 = 'Abc'
my_string2 = 'aBc'

my_string1 == my_string2
→  False

print(my_string1.upper()) → 'ABC'
print(my_string1.lower()) → 'abc'

my_string1.upper() == my_string2.upper()
→ True
```

08.

Computergestützte Fugenanalyse I

Challenges der Fugenanalyse

- Allgemein gilt in der Musikanalyse:

  Es gibt nicht immer nur die eine „korrekte“ Analyse.
  Musik lebt von Uneindeutigkeiten und auch Musikwissenschaftler*innen können sich uneinig sein – oder plädieren dafür, verschiedene, differenzierte Standpunkte einzunehmen.

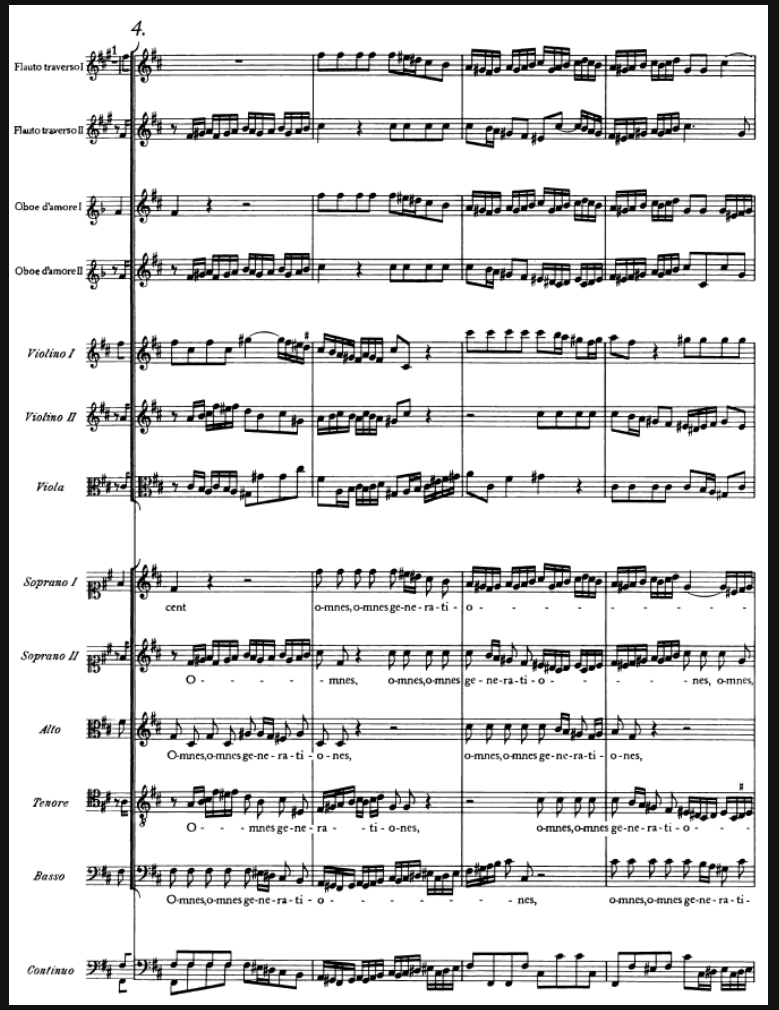
- Spezifische Herausforderungen in der computergestützten Fugenanalyse:

  - Erkennen des Fugenthemas an sich
    - Das ist eine Frage für eine spätere Woche… 😟
    - Insbesondere müssen wir uns dazu noch überlegen, wie wir die Ähnlichkeit zwischen 2 Melodien, Objekten, Dingen, … bestimmen!

  - Erkennen von realen/tonalen Beantwortungen des Fugenthemas
  - Wiedererkennen von Fugenthemen in verschiedenen Tonarten
  - Berücksichtigung von möglichen melodischen/rhythmischen Mutationen des Fugenthemas
  - Uneindeutige Enden von Fugenthemen
  - Erkennen von möglichen Kontrasubjekten

Analyse einer Chorfuge

- Aufgabe:
  Bestimme, wie oft das Fugenthema in der ersten Sopranstimme der 5-stimmigen Chorfuge *Omnes Generationes* aus dem Magnificat von J. S. Bach vorkommt.



Analyse einer Chorfuge

- Aufgabe:
  Bestimme, wie oft das Fugenthema in der ersten Sopranstimme der 5-stimmigen Chorfuge *Omnes Generationes* aus dem Magnificat von J. S. Bach vorkommt.

- Hinweis:
  Stellt man das Fugenthema als **Intervall-Pattern** dar, genügt es in diesem Fall, die Fuge auf die Intervall-Patterns [0, 0, 0, 0, -1] und [0, 0, 0, 0, -2], die die Intervalle zu Beginn des Fugenthemas darstellen, abzusuchen.

- Lösung:
  Siehe Notebook der Woche 03.