Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

AUDIOVERARBEITUNG MIT PYTHON

Projektarbeit im Modul Programmierparadigmen

23. Juni 2025

Ilyas Ouhmid und Leon Weiss

Dozent: Prof. Panitz
Studiengang Angewandte Informatik
Hochschule **RheinMain**

GLIEDERUNG

1. Grundlagen des Klangs

2. Audiosynthese: Klänge am Computer erzeugen

3. Audioanalyse: Die Sprache des Klangs verstehen

4. Zusammenfassung

GRUNDLAGEN DES KLANGS

WAS IST KLANG? DIE PHYSIKALISCHE GRUNDLAGE

→ Klang ist die Veränderung des Luftdrucks über die Zeit.

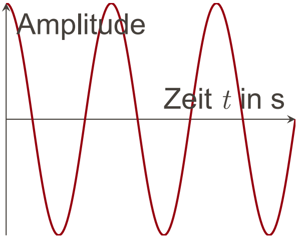
→ Unser Trommelfell nimmt diese Druckschwankungen als Schwingungen wahr.

AMPLITUDE UND FREQUENZ

→ **Amplitude:** Die Stärke der Schwingung, die wir als **Lautstärke** empfinden.

→ **Frequenz:** Die Anzahl der Schwingungen pro Sekunde (in Hertz), die wir als **Tonhöhe** wahrnehmen.

→ Das menschliche Ohr kann Frequenzen zwischen ca. 20 und 20.000 Hertz wahrnehmen.



VOM ANALOGEN SCHALL ZUM DIGITALEN SCHALL

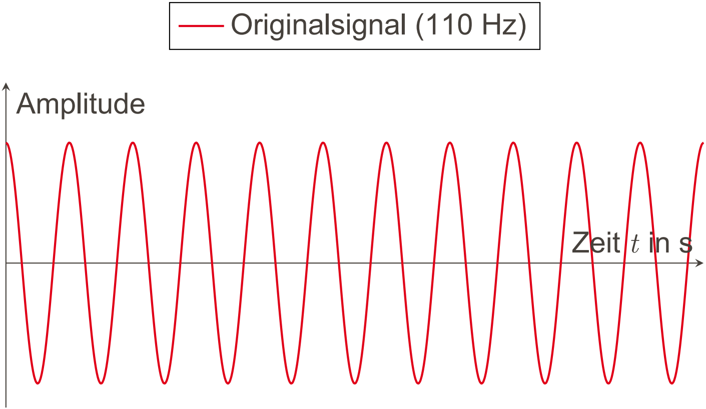
→ Computer können keine kontinuierlichen, analogen Signale speichern, sie arbeiten in einer **diskreten Welt**

→ Schall wird daher als eine Folge von Messwerten (Samples) des Luftdrucks dargestellt

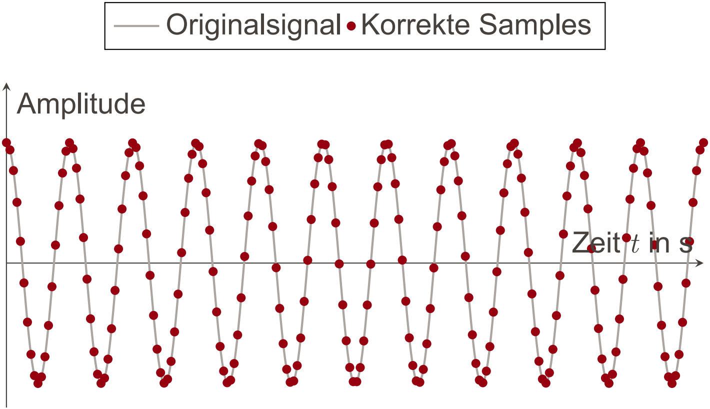
→ **Abtastrate:** Gibt an, wie oft pro Sekunde ein Sample genommen wird. (Der CD-Standard ist 44.100 Hz)

→ **Problem:** Eine zu niedrige Abtastrate kann Schwingungen nicht korrekt erfassen und zu falschen Messergebnissen führen.

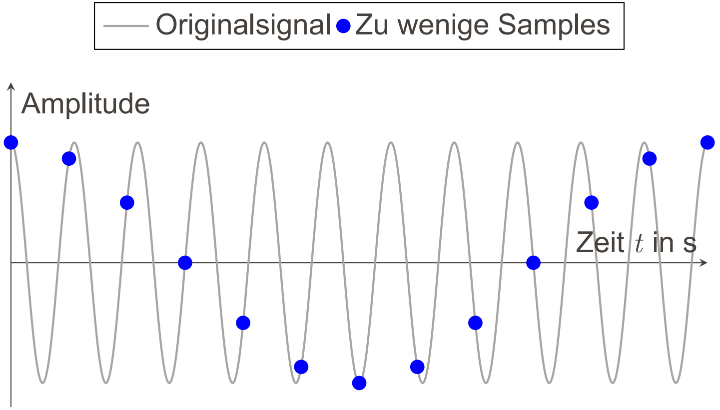
DAS NYQUIST-SHANNON-ABTASTTHEOREM (1/5)



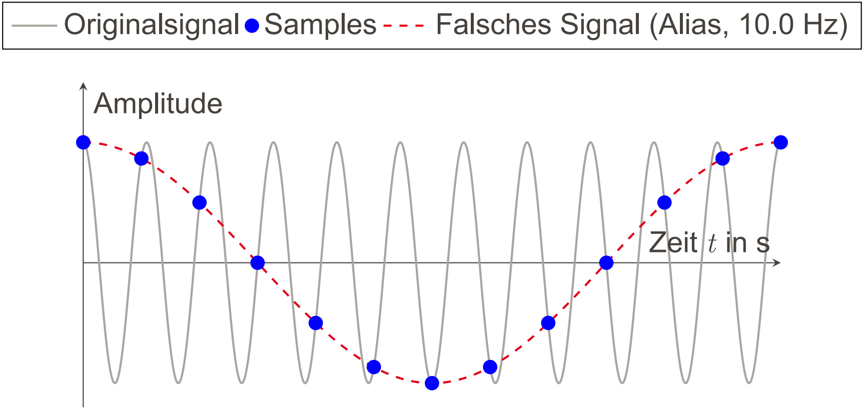
DAS NYQUIST-SHANNON-ABTASTTHEOREM (2/5)



DAS NYQUIST-SHANNON-ABTASTTHEOREM (3/5)



DAS NYQUIST-SHANNON-ABTASTTHEOREM (4/5)



DAS NYQUIST-SHANNON-ABTASTTHEOREM (5/5)

→ **Frage:** Wie oft müssen wir messen, um keine wichtigen Informationen zu verlieren?

→ **Antwort:** ”Die Abtastrate f_s muss mehr als doppelt so hoch sein wie die höchste im Signal enthaltene Frequenz f_{max}.

→ **Formel:**

$$f_s > 2 \cdot f_{max}$$

DIE VERLETZUNG DES THEOREMS: DER ALIASING-EFFEKT

→ **Frage:** Was passiert, wenn wir die Regel verletzen?

→ **Beobachtung:** Die wenigen Messpunkte können die schnelle Schwingung nicht korrekt erfassen. Es entsteht ein Trugbild: eine scheinbar viel langsamere Schwingung

→ **Fachbegriff:** Diesen Effekt nennt man **Aliasing**.

DIE KONSEQUENZ FÜR DIE PRAXIS: WARUM 44.100 HZ?

→ Das menschliche Gehör reicht bis etwa 20.000 Hz ($f_{max} \approx 20.000$ Hz).

→ Nach Nyquist-Shannon benötigen wir also: $f_s > 2 \cdot 20.000$ Hz, also $f_s > 40.000$ Hz.

→ **Fazit:** Die Rate von 44.100 Hz wurde gewählt, um das gesamte menschliche Hörspektrum abzutasten. So wird Aliasing im hörbaren Bereich vermieden.

AUDIOSYNTHESE

EIN EINFACHER TON IN PYTHON

→ Ein Ton wird als eine Liste von Zahlen repräsentiert, die eine mathematische Schwingung (z.B. Sinus) beschreiben.

```
import math
kammertonA = [10000*math.sin(2*440*math.pi*x/44100)
for x in range(0,5*44100)]
```

KLANGFARBE DURCH OBERTÖNE

→ Klänge von echten Instrumenten bestehen aus einer Grundschwingung und vielen **Obertönen**.

→ Dieses Frequenzgemisch bestimmt die **Klangfarbe.**

→ Wir erzeugen komplexere Klänge durch die Addition von Schwingungen.

```
import math
kammertonA = [10000*math.sin(2*440*math.pi*x/44100)
for x in range(0,5*44100)]
```

AUFGABE 1: SIMULATION EINES ZUPFINSTRUMENTS

→ **Ziel:** Einen Klang simulieren, der ausklingt

→ **Realisierung:**

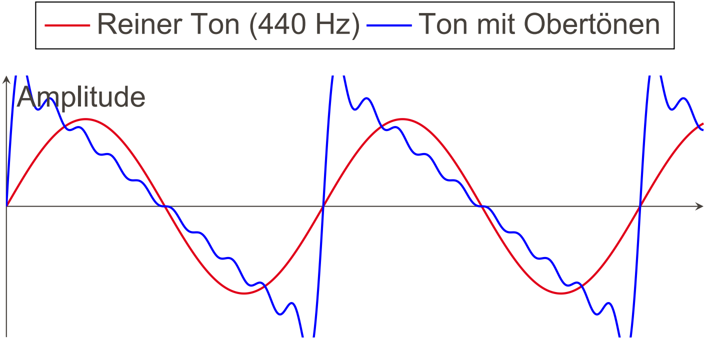
  → **Komplexe Klangfarbe:** Überlagerung von 10 Sinus-Funktionen (Grundton + 9 Obertöne)

  → **Amplitudenhüllkurve:** Die Amplitude wird alle 5000 Samples halbiert, um das Ausklingen zu simulieren

ZUPFINSTRUMENT IN PYTHON

```
def pluggedTime( t, wv):
   samples = []
   sample_rate = 44100
   initial_amplitude = 10000
   for x_n in range(t):
      current_amplitude = initial_amplitude / (2 **
          (x_n // 5000))
      x_in_formula = wv * x_n / sample_rate
      sum = 0
      for i in range(1, 11):
         sum += (1 / i) * math.sin(2 * math.pi *
             x_in_formula * i)
      sample_value = current_amplitude * sum
      samples.append(sample_value)
   return samples
```

VISUALISIERUNG: KLANGFARBE DES ZUPFINSTRUMENTS



AUFGABE 2 & 3: MELODIEN UND AKKORDE

→ **Melodien:** Eine Sequenz von Tönen, die durch das Aneinanderreihen der Sample-Listen erzeugt wird

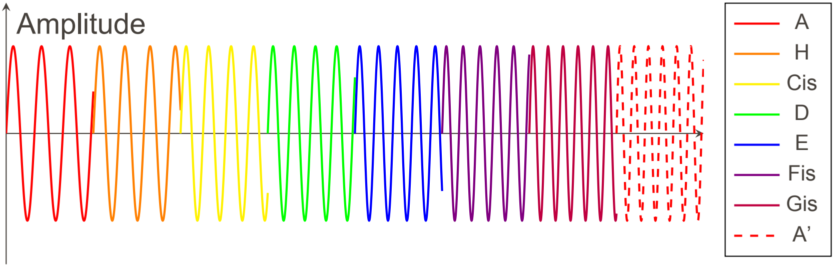
→ **Akkorde:** Gleichzeitig erklingende Töne, die durch die elementweise Addition der Sample-Listen realisiert werden

→ **Arpeggio:** Ein zeitversetzter Einsatz der Töne wird durch das Voranstellen von Nullen in den Sample-Listen der späteren Töne erreicht.

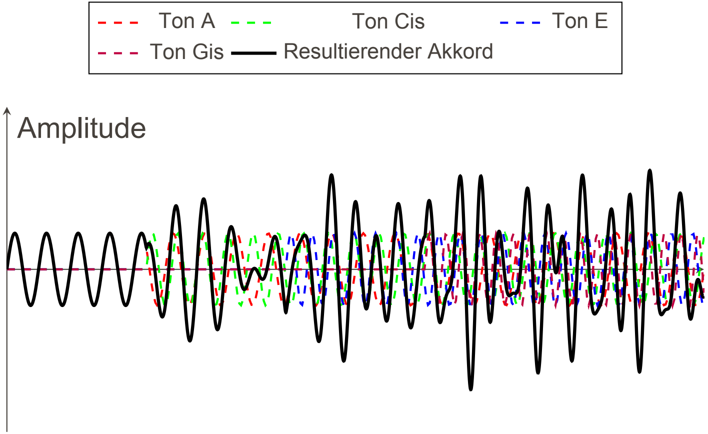
MELODIEN UND AKKORDE IN PYTHON

```
def scale():
    lists = [pluggedH(a), pluggedH(b), pluggedH(cs),
        pluggedH(d), pluggedH(e), pluggedH(fs),
        pluggedH(gs), pluggedH(aP)]
    return list(itertools.chain.from_iterable(lists))

def maj7():
    cs_versetzt = 2000 * [0.0] + cs_ton
    e_versetzt = 4000 * [0.0] + e_ton
    gs_versetzt = 6000 * [0.0] + gs_ton

    return [sum(werte) for werte in
        itertools.zip_longest(a_ton, cs_versetzt,
        e_versetzt, gs_versetzt, fillvalue=0.0)]
```

VISUALISIERUNG: A-DUR-TONLEITER



VISUALISIERUNG: A-MAJ7-AKKORD (ARPEGGIO)



SPEICHERN DER AUDIODATEIEN

→ Die generierte Liste von Fließkommazahlen muss für die WAV-Datei in 16-Bit-Integer (numpy.int16) konvertiert werden.

→ Die Funktion scipy.io.wavfile.write übernimmt das Schreiben.

→ **Problem:** Bei der Addition von Tönen (Akkorde) kann der Wertebereich von int16 überschritten werden.

→ **Lösung:** Die writeWav-Methode verwendet Normalisierung: Alle Werte werden um einen Faktor skaliert, sodass der höchste Wert genau dem Maximum von int16 entspricht

AUDIOANALYSE

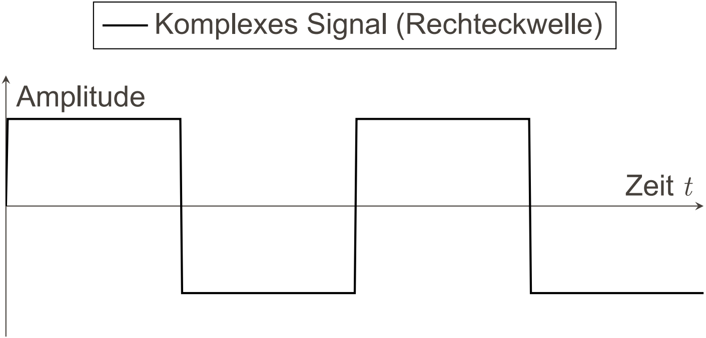
VOM SIGNAL ZURÜCK ZUR FREQUENZ

→ **Ziel:** Die in den rohen Sample-Werten “versteckten” Frequenzen finden.

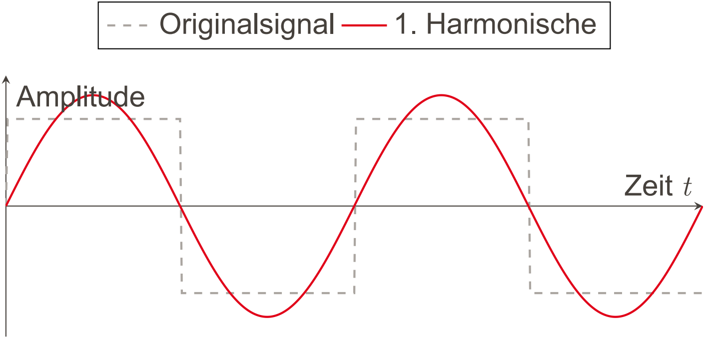
→ **Grundlage:** Der Satz von **Joseph Fourier.** Jede periodische Schwingung lässt sich als eine Summe von Sinus- und Kosinus-Funktionen darstellen.

→ Das bedeutet: Wir können unser komplexes Signal wieder in seine Zutaten zerlegen.

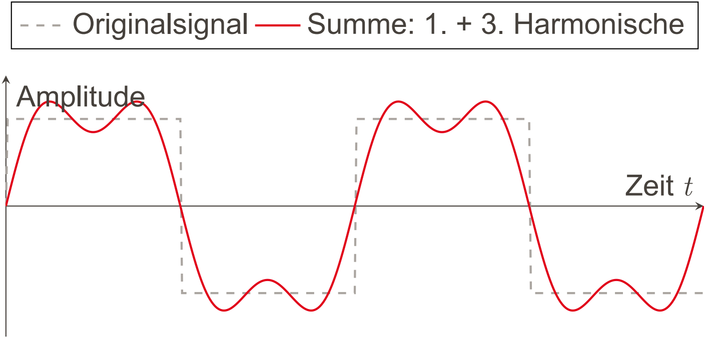
SATZ VON FOURIER (1/5): DAS KOMPLEXE SIGNAL



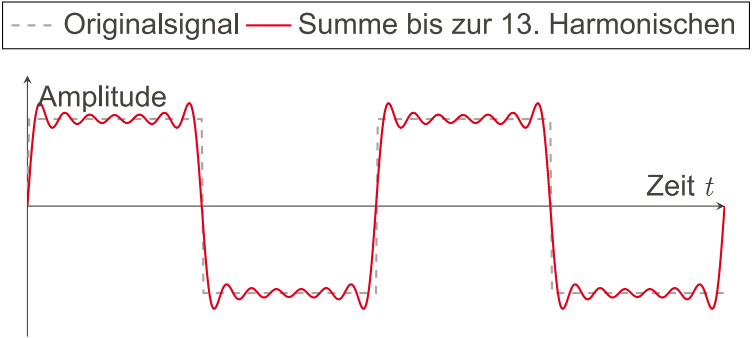
SATZ VON FOURIER (2/5): 1. ANNÄHERUNG (GRUNDTON)



SATZ VON FOURIER (3/5): 2. ANNÄHERUNG

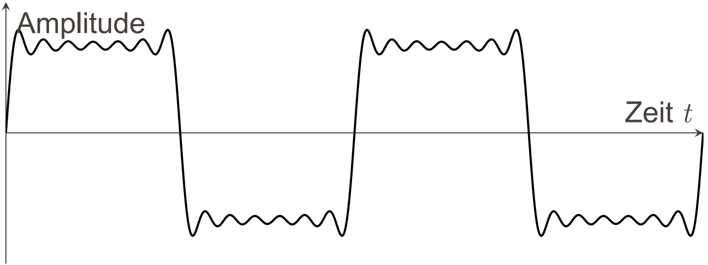


SATZ VON FOURIER (4/5): WEITERE ANNÄHERUNG



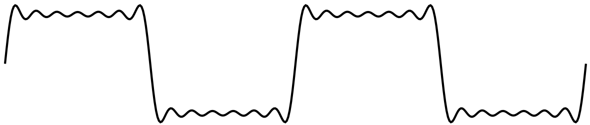
VON DER SYNTHESE ZUR ANALYSE (5/5): DIE ZEITDOMÄNE

Signal in der Zeitdomäne



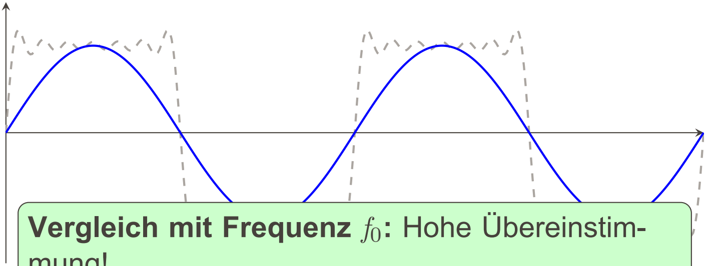
WIE FUNKTIONIERT DIE DFT? (1/4)

Unser Signal



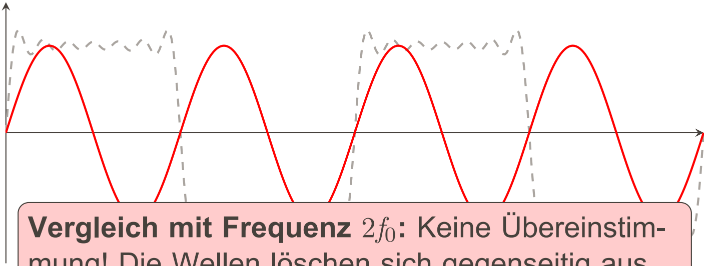
Die DFT vergleicht das Signal mit reinen Sinustönen jeder Frequenz.

WIE FUNKTIONIERT DIE DFT? (2/4)



Vergleich mit Frequenz f_0: Hohe Übereinstimmung!
$\rightarrow$ Die DFT findet eine große Amplitude bei f_0.

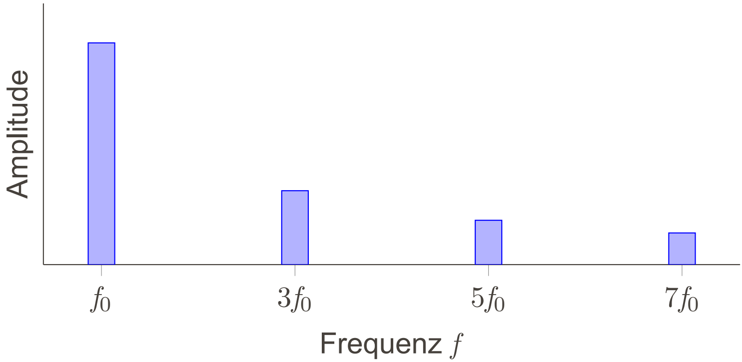
WIE FUNKTIONIERT DIE DFT? (3/4)



Vergleich mit Frequenz $2f_0$: Keine Übereinstimmung! Die Wellen löschen sich gegenseitig aus.
$\rightarrow$ Die DFT findet eine Amplitude von Null bei $2f_0$.

WIE FUNKTIONIERT DIE DFT? (4/4)

Das Ergebnis nach dem Test aller Frequenzen



Dieser Prozess wird für alle relevanten Frequenzen wiederholt und ergibt das finale Frequenzspektrum.

DIE DISKRETE FOURIER-TRANSFORMATION (DFT)

→ Die DFT ist der Algorithmus, der diese Zerlegung für eine diskrete Folge von Samples durchführt

→ **Input:** Eine Liste von Abtastwerten (Zeitdomäne)

→ **Output:** Eine Liste komplexer Zahlen (Frequenzdomäne). Der **Betrag** jeder komplexen Zahl gibt uns die **Amplitude** (Stärke) der jeweiligen Frequenz.

$$\hat{x}[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi kn}{N}}$$

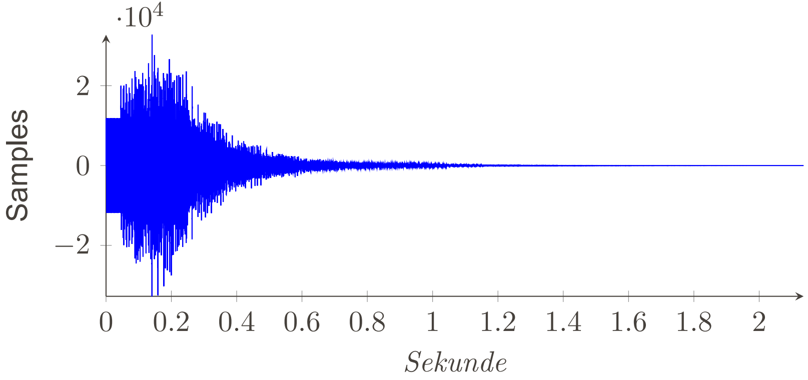
mit N Abtastwerten in einer Sekunden und $\hat{x}[n]$ als n-ten Abtastwerten

DFT IN PYTHON

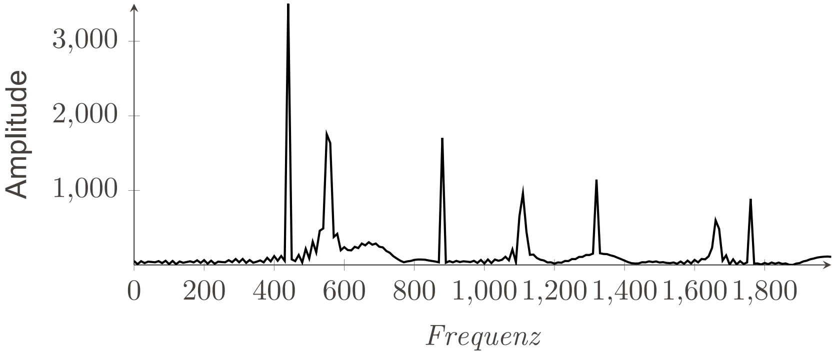
```
def dft(xs):
    N = len(xs)
    x = []

    for k in range(N):
        sum = complex(0,0)
        for n in range(N):
            sum += xs[n] * cmath.exp(-1j * 2 * cmath.pi
                * k * n / N)
        x.append(sum / N)
    return x
```

$$\hat{x}[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi kn}{N}}$$

VISUALISIERUNG: ZEITDOMÄNE



VISUALISIERUNG: FREQUENZDOMÄNE



ZUSAMMENFASSUNG

ZUSAMMENFASSUNG

→ **Erkenntnisse:**

- → **Digitale Repräsentation:** Klang wird als eine Folge von Messwerten (Samples) gespeichert.
- → **Nyquist-Shannon-Theorem:** Die Abtastrate muss mehr als doppelt so hoch sein wie die höchste Frequenz, um Informationsverlust zu vermeiden.
- → **Synthese in Python:** Komplexe Klänge, Melodien und Akkorde werden durch die Überlagerung und Addition mathematischer Schwingungen (z. B. Sinus) erzeugt.
- → **Analyse durch DFT:** Die Diskrete Fourier-Transformation ist der Algorithmus, der die in den Samples “versteckten” Frequenzen eines Signals aufdeckt.
- → **Zeit- vs. Frequenzdomäne:** Die DFT überführt das Signal von der Zeit- in die Frequenzdomäne und zeigt so die Amplitude jeder einzelnen Frequenz an

VIELEN DANK

Offene Fragen?
Falls noch Fragen offengeblieben sind, wollen wir diese gerne noch beantworten.

Ein Entwurf einer Webanwendung zur Audioverarbeitung

Der Quellcode ist unter der MIT-Lizenz verfügbar: https://github.com/leon-weiss/Python-Audio-Processor