ChatGPT

Lern-PDF: Alles über @dataclass für Audio-Einstellungen in deinem Python-GUI-Projekt

---

🎧 Warum eine `AudioSettings`-Dataklasse?

In deinem Audiokonvertierungsprogramm hast du mehrere Einstellungen wie:

- Bitrate (z. B. 192)
- Kanäle (Mono, Stereo)
- Abtastrate (z. B. 44100 Hz)
- Format (MP3, WAV, AAC ...)

Statt alle diese Werte einzeln in Variablen zu speichern, erstellst du **eine Datenklasse**, die alle Infos übersichtlich enthält. So wird dein Code einfacher, kürzer und sauberer.

Was du brauchst

Du brauchst wieder:

```
from dataclasses import dataclass
```

Wenn du Standardwerte willst:

```
from dataclasses import dataclass, field
```

Beispiel: Eine AudioSettings Klasse

```
@dataclass
class AudioSettings:
    format: str
    bitrate: int
    channels: str
    sample_rate: int
```

Beispiel zur Nutzung:

```
settings = AudioSettings(format="MP3", bitrate=192, channels="Stereo",
sample_rate=44100)
```

Dann kannst du z. B. schreiben:

```
convert_audio(file_path, output_path, settings)
```

Struktur mit GUI

1. Du holst alle Werte aus den Comboboxen und Eingabefeldern
2. Du wandelst sie in passende Typen um (z. B. int)
3. Du gibst sie an deine Datenklasse weiter

Beispiel (vereinfacht):

```
audio_settings = AudioSettings(
    format=self.audio_format_combobox.get(),
    bitrate=int(self.audio_bitrate_combobox.get()),
    channels=self.audio_channels_combobox.get(),
    sample_rate=int(self.audio_samplerate_combobox.get())
)
```

⚡ Tipps zur Umsetzung in deinem Projekt

Wo erstellen?

Erstelle eine Datei `settings.py` und schreibe dort deine Datenklassen für Audio & Bild rein.

🔪 Keine GUI in der Datenklasse!

Die Klasse enthält **nur Werte** – keine `.get()`, keine Comboboxen, keine tkinter-Dinge.

Für jede Datei neue Settings?

Wenn der Nutzer mehrere Dateien auswählt: ja. Sonst reicht eine Instanz.

Erweiterung mit Standardwerten

```
@dataclass
class AudioSettings:
    format: str = "MP3"
```

```
    bitrate: int = 192
    channels: str = "Stereo"
    sample_rate: int = 44100
```

Dann kannst du auch einfach:

```
settings = AudioSettings()
```

Beispiel: Wert aus "Max(320)" extrahieren

```
def extract_bitrate(text: str) -> int:
    if "(" in text and ")" in text:
        start = text.find("(") + 1
        end = text.find(")")
        return int(text[start:end])
    return int(text)
```

🏋️ Vergleich: Vorher vs. Nachher

Vorher:

```
convert_audio(path, out, format, bitrate, channels, rate)
```

Nachher:

```
settings = AudioSettings(...)
convert_audio(path, out, settings)
```

Im Konverter:

```
def convert_audio(file, out, settings):
    if settings.format == "MP3":
        # konvertieren mit settings.bitrate, etc.
```

Bonus: Fähigkeiten von @dataclass

- Automatische `__init__` Methode
- Automatische `__repr__`: z. B. `print(settings)`
- Vergleichbar mit `==`

Erweiterungsideen für dich

Feature	Typ	Beispiel
normalize_audio	bool	True
remove_silence	bool	False
fade_in/fade_out	int	2 (Sekunden)
custom_metadata	str	"Autor: Max"

Dann z. B.:

```
@dataclass
class AudioSettings:
    format: str
    bitrate: int
    channels: str
    sample_rate: int
    normalize: bool = False
    fade_in: int = 0
```

Fazit

Mit dieser Anleitung kannst du **offline** deine `AudioSettings`-Klasse:

- erstellen
- mit GUI-Werten belegen
- an Konvertierungsfunktionen weitergeben
- erweitern & debuggen

Wenn du auch noch Hilfe für "Projektstruktur", "Tooltips speichern", oder "globale GUI → Settings-Verbindung" brauchst – sag einfach Bescheid!