ChatGPT

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

---

Warum eine `ImageSettings`-Dataklasse?

In deinem Bildkonvertierungsprogramm gibst du viele Werte ein, z. B.:

- Format: PNG, JPEG, BMP ...
- Qualität: z. B. 90 bei JPEG
- Kompressionslevel: z. B. 6 bei PNG
- Transparenz behalten: Ja/Nein
- Farbmodus: RGB, L, CMYK

Statt jeden Wert einzeln in Variablen zu speichern, nutzt du **eine Datenklasse**, die alle Infos übersichtlich enthält. Das macht deinen Code einfacher, sauberer und leichter erweiterbar.

Was du brauchst

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

📚 Beispiel: Eine ImageSettings-Klasse

```
@dataclass
class ImageSettings:
    format: str
    quality: int = 100
    compression_level: int = 0
    transparency: bool = False
    color_mode: str = "RGB"
```

Verwendung:

```
settings = ImageSettings(format="PNG", compression_level=6)
```

Dann kannst du in deiner Konvertierung schreiben:

```
convert_image(file_path, output_path, settings)
```

Struktur mit deiner GUI

1. Du holst alle Werte aus deinen Comboboxen, Eingabefeldern usw.
2. Du wandelst sie ggf. um (z. B. aus "Max(9)" wird 9)
3. Du gibst sie an deine `ImageSettings`-Instanz weiter

Beispiel:

```
image_settings = ImageSettings(
    format=self.format_combobox.get(),
    quality=extract_number(self.jpeg_quality_combobox.get()),
    compression_level=extract_number(self.png_compression_combobox.get()),
    transparency=self.transparency_checkbox.get(),
    color_mode=self.color_mode_combobox.get()
)
```

---

extract_number(): Die Hilfsfunktion

Wenn du in der Combobox Werte wie "Max(9)" hast:

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

Erweiterung: Wenn du auch Floats oder andere Formate brauchst, kannst du es erweitern.

Tipps für dein Projekt

📚 Wo speichern?

Erstelle `settings.py` und speichere `ImageSettings` und `AudioSettings` dort. Kein GUI-Code!

Kein tkinter in deiner Datenklasse!

`@dataclass` speichert **nur Werte**. Du holst `.get()`-Werte **vorher** in deiner GUI.

Pro Datei eine Instanz?

Wenn du pro Datei unterschiedliche Einstellungen brauchst: ja. Sonst reicht eine gemeinsame Instanz.

Erweiterungsideen für dich

Feature	Typ	Beispiel
resize_width	int	800
resize_height	int	600
keep_aspect_ratio	bool	True
sharpen_level	float	1.2
add_watermark	bool	False
convert_to_grayscale	bool	False

Dann z. B.:

```
@dataclass
class ImageSettings:
    format: str
    quality: int = 100
    compression_level: int = 0
    transparency: bool = False
    color_mode: str = "RGB"
    resize_width: int = 0
    resize_height: int = 0
    keep_aspect_ratio: bool = True
```

📊 Vorher vs. Nachher

Vorher:

```
convert_image(file, out, format, qual, comp, trans, color)
```

Nachher:

```
settings = ImageSettings(...)
convert_image(file, out, settings)
```

Und im Konverter:

```python
def convert_image(file, out, settings):
    if settings.format == "JPEG":
        compress_jpeg(file, out, quality=settings.quality)
```

📊 Vorteile von @dataclass (Wiederholung)

- Automatisch `__init__()` generiert
- Automatisch `__repr__()` für Debugging
- Vergleichbar mit `==`
- Optional: Standardwerte

Fazit

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

- erstellen
- mit GUI-Werten belegen
- weitergeben
- modular erweitern

Wenn du später noch Tooltips, Validierung oder dynamische Vorschauen brauchst: Ich helfe dir gerne!