ChatGPT

Lern-PDF: Alles über @dataclass in Python – Einfach erklärt für dein Projekt

---

Was ist eine Dataclass?

Eine `@dataclass` ist eine praktische Funktion in Python, mit der du **schnell und einfach Klassen erstellen kannst**, die vor allem **Daten speichern**. Du musst dich nicht mehr um Konstruktoren (`__init__`) oder `__repr__`-Methoden kümmert – das erledigt Python automatisch für dich.

Vorteile:

- Spart Zeit
- Erzeugt automatisch wichtige Methoden
- Macht Code klarer und strukturierter

Wie benutzt man eine Dataclass?

1. Import:

```
from dataclasses import dataclass
```

2. Klasse definieren:

```
@dataclass
class Person:
    name: str
    age: int
```

3. Objekt erstellen:

```
person = Person("Anna", 30)
print(person)  # Ausgabe: Person(name='Anna', age=30)
```

Du bekommst automatisch: - `__init__` → Konstruktor - `__repr__` → besser lesbarer `print()` - `__eq__` → Vergleich mit `==`

📦 Typen und Standardwerte

Häufige Typen:

- `str` → Text
- `int` → Ganzzahl
- `float` → Kommazahl
- `bool` → Wahr/Falsch
- `list`, `dict` → Listen und Wörterbücher

Mit Standardwerten:

```
@dataclass
class Item:
    name: str
    quantity: int = 1
```

Weitere Funktionen und Extras

`asdict()` – Objekt in Dictionary umwandeln

```
from dataclasses import asdict

item = Item("Apfel", 3)
print(asdict(item))  # {'name': 'Apfel', 'quantity': 3}
```

`replace()` – Kopieren und ändern

```
from dataclasses import replace

item2 = replace(item, quantity=5)
```

`field()` – für Listen und komplexe Standardwerte

```
from dataclasses import field

@dataclass
class Einkauf:
    artikel: list[str] = field(default_factory=list)
```

Tipps für dein Projekt

1. Trenne GUI und Logik

- In der GUI liest du z. B. Combobox-Werte mit `.get()`
- Danach erstellst du ein Dataclass-Objekt mit diesen Werten
- Das Objekt gibst du an andere Funktionen weiter

2. Erstelle eine Datei `settings.py`

Dort speicherst du alle Dataclasses, z. B.:

```
@dataclass
class ImageSettings:
    format: str
    quality: int = 95
    optimize: bool = False
```

Dann importieren in deiner GUI:

```
from settings import ImageSettings
```

3. Werte vorbereiten:

Falls Combobox-Werte so aussehen wie `"Max(9)"`, nutze:

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

Extra-Tipps für dein Programm

Verwende eine Dataclass für jeden Einstellungsbereich

- `ImageSettings` für Bildoptionen (Format, Qualität, Kompression, usw.)
- `AudioSettings` für Audio (Bitrate, Codec, Sample Rate, usw.)

Halte GUI und Daten getrennt

- Lese Daten **in der GUI** (Comboboxen usw.)
- Wandle sie **sofort** in Dataclass-Objekte um
- Gib diese weiter an deine Konvertier-Funktionen

Validierung zentral

- Wenn du Daten über `@dataclass` speicherst, kannst du später leicht Prüfungen einbauen (z. B. `assert 0 <= quality <= 100`)

Gruppieren für Mehrfach-Konvertierung

- Du kannst z. B. eine Liste von `ImageSettings`-Objekten anlegen und alle Dateien automatisch durchlaufen

---

Weitere Tipps und Nutzungsideen mit Dataclass

Mehrere Konfigurationen gleichzeitig speichern

Wenn du mehrere Bilder oder Audios gleichzeitig bearbeiten willst:

```
all_image_settings: list[ImageSettings] = []
```

Verschachtelte Dataclasses verwenden

Du kannst in einer Dataclass eine andere als Typ angeben:

```
@dataclass
class AdvancedSettings:
    image: ImageSettings
    audio: AudioSettings
```

Du kannst Methoden in Dataclasses nutzen

```
@dataclass
class ImageSettings:
    format: str
    quality: int

    def summary(self) -> str:
        return f"{self.format.upper()} mit Qualität {self.quality}"
```

Vergleich leicht gemacht

```
img1 = ImageSettings(".jpg", 90)
img2 = ImageSettings(".jpg", 90)
print(img1 == img2)  # True
```

Automatisches Debugging

Einfach mit `print(einstellungen)` bekommst du sofort alle Werte angezeigt:

```
ImageSettings(format='.jpg', quality=90, optimize=False)
```

Optional: Validierung (später möglich)

Zwar ist `@dataclass` eher für einfache Daten gedacht, aber du kannst auch einfache Validierung einbauen, z. B.:

```
@dataclass
class ImageSettings:
    quality: int

    def __post_init__(self):
        if not 0 <= self.quality <= 100:
            raise ValueError("Qualität muss zwischen 0 und 100 liegen")
```

Diese Methode `__post_init__` wird direkt nach dem Erzeugen aufgerufen.

Zusammenfassung – Warum `@dataclass`?

Vorteil	Beschreibung
Schnell	Du sparst dir viel Code
Klar	Alle zusammengehörigen Werte in einem Objekt
Flexibel	Du kannst jederzeit neue Felder hinzufügen
Besser lesbar	`print(objekt)` zeigt die Inhalte direkt an
Einfach nutzbar	Übergabe an Funktionen ohne viele Argumente

Merkliste fürs Erstellen deiner eigenen Dataclass

1. `@dataclass` davor schreiben
2. Felder mit Typen angeben
3. Optional Standardwerte setzen
4. Werte aus der GUI lesen und übergeben
5. Nur in einer Datei definieren und überall importieren
6. `asdict()` nutzen, wenn du ein Dictionary brauchst

Mit diesem Wissen kannst du ab jetzt **jede Einstellung und Konfiguration** in deinem Projekt übersichtlich und sauber mit `@dataclass` speichern und weitergeben – ohne Chaos mit 10 einzelnen Variablen.

Wenn du später willst, kannst du auch Validierung, verschachtelte Dataclasses oder automatische Konvertierung dazunehmen – aber das hier reicht völlig für dein aktuelles Projekt.

Viel Erfolg beim Umsetzen deines Programms!