PEP 8 für Einsteiger

#	Titel der Konvention	Beschreibung (DE)	Schlechtes Beispiel	Besser nach PEP 8
1	**Variablennamen: snake_case**	Kleinbuchstaben, Wörter durch Unterstriche trennen.	`UserName = "Anna"`	`user_name = "Anna"`
2	**Funktionsnamen: snake_case**	Gleiches Muster wie Variablen; ein Verb sollte erkennbar sein.	`def GetData(): ...`	`def get_data(): ...`
3	**Konstanten: UPPER_CASE**	Nur Großbuchstaben und Unterstriche für unveränderliche Werte.	`piValue = 3.1415`	`PI_VALUE = 3.1415`
4	**Klassennamen: PascalCase**	Jedes Wort beginnt mit Großbuchstaben, keine Unterstriche.	`class userprofile: ...`	`class UserProfile: ...`
5	**Modulnamen: snake_case**	Dateinamen ganz klein, Unterstriche nur wenn nötig.	`DataTools.py`	`data_tools.py`
6	**Paketnamen: lowercase**	Ordnernamen kurz, nur Kleinbuchstaben, keine Unterstriche.	`Utilities/`	`utilities/`
7	**Einfacher Unterstrich für "wegwerf"-Variablen**	`_` signalisiert: „Wert wird nicht benötigt".	`for i, x in enumerate(data): ...` *(und i ungenutzt)*	`for _, x in enumerate(data): ...`
8	**Platzhalterpräfix _internal**	Ein führender Unterstrich kennzeichnet interne Attribute/Funktionen.	`def helper(): ...` (öffentlich importiert)	`def _helper(): ...`
9	**Booleans mit is/has/should**	Namen sollen den Wahr/Falsch-Charakter verdeutlichen.	`enabled = True`	`is_enabled = True`
10	**Sammlungen im Plural**	Listen / Sets / Dicts erhalten einen Pluralnamen.	`user = []`	`users = []`
11	**Iteratoren: i, j, k nur kurzfristig**	Ein-Buchstaben-Namen nur für sehr kurze Schleifen.	`for x in range(10): ...`	`for index in range(10): ...`
12	**Exceptions enden mit *Error***	Eigene Ausnahmen klar als Fehlerklasse erkennbar.	`class ValidationException(Exception): ...`	`class ValidationError(Exception): ...`

#	Titel der Konvention	Beschreibung (DE)	Schlechtes Beispiel	Besser nach PEP 8
13	**Dunder-Methoden reserviert**	Nur für vordefinierte Spezialmethoden benutzen (`__init__`, `__str__`, ...).	`def __custom__(self): ...`	*(Keine frei erfundenen Dunder-Namen)*
14	**Testfunktionen beginnen mit test_**	Erlaubt Test-Runnern die automatische Entdeckung.	`def check_login(): ...`	`def test_login(): ...`
15	**Enum-Mitglieder: UPPER_CASE**	Enum-Konstanten wie normale Konstanten schreiben.	`class Color(Enum): red = 1`	`class Color(Enum): RED = 1`
16	**TypeVars: CapWords + T-Suffix**	Generische Typvariablen enden oft auf `T`.	`T = TypeVar("t")`	`UserT = TypeVar("UserT")`
17	**Numerische Literale lesbar halten**	Unterstriche in großen Zahlen, nicht im Namen.	`million = 1000000`	`million = 1_000_000`
18	**Hauptprogramm-Prüfung: if name**	Immer exakt `if __name__ == "__main__":` schreiben.	`if __name__ == '__main__': ...` (falsche Anführungszeichen)	`if __name__ == "__main__": ...`
19	**Import-Alias sprechend benennen**	Kurze, verbreitete Kürzel (z. B. `np`, `pd`).	`import numpy as n`	`import numpy as np`
20	**Kontextspezifische Präfixe/Suffixe vermeiden**	Keine un-nötigen Typangaben im Namen wie `listUsers`.	`user_list = [...]`	`users = [...]`

PEP 8 Advanced

#	Titel der Konvention	Beschreibung (DE)	Schlechtes Beispiel	Besser nach PEP 8
1	**„Private“ Attribute (_single_leading)**	Ein einzelner Unterstrich kennzeichnet interne API-Teile.	`self.id = 42`	`self._id = 42`
2	**Namens-Mangling (__double_leading)**	Zwei führende Unterstriche lösen Namens-veränderung zur Vermeidung von Konflikten in Subklassen aus.	`self.__counter = 0` (in Base und Child gleich benannt)	`self.__counter = 0` *(nur in Basisklasse; Child benutzt eigenen Namen)*
3	**„Dunder“-Methoden**	Spezialmethoden beginnen und enden mit doppelten Unterstrichen.	`def init(self): ...`	`def __init__(self): ...`
4	**Ausnahme-Klassen enden mit *Error***	Eigene Exceptions sollen auf *Error* enden.	`class ValidationException(Exception): pass`	`class ValidationError(Exception): pass`
5	**Typ-Aliase & TypeVars (CapWords + T-Suffix)**	TypeVars / Aliase sind CamelCase; häufig mit *T* oder *Type*-Suffix.	`Id = int`	`UserIdT = int`
6	**Asynchrone Coroutinen (snake_case)**	Auch bei `async def` bleibt snake_case; Funktionsnamen sollten ein Verb enthalten.	`async def ProcessData(): ...`	`async def process_data(): ...`
7	**Metaklassen enden mit *Meta***	Metaklassen erkennt man am *Meta*-Suffix.	`class Custom(type): ...`	`class CustomMeta(type): ...`