Hier ist ein ausführliches Infoblatt zum Thema **Python-Klassen**, perfekt geeignet für deine Offline-Bibliothek und deinen Programmieralltag:

Infoblatt: Python-Klassen – Struktur, Nutzen & Modularität

1. Was sind Klassen?

- Klassen sind **Baupläne für Objekte**.
- Sie definieren **Eigenschaften (Attribute)** und **Verhalten (Methoden)** von Objekten.
- Ein Objekt (Instanz) ist ein konkretes Exemplar einer Klasse.

---

2. Warum Klassen verwenden?

- **Strukturierung:** Organisiere deinen Code in logische Einheiten (z.B. GUI, Dateiverwaltung).
- **Wiederverwendbarkeit:** Du kannst Klassen mehrfach nutzen und erweitern (Vererbung).
- **Kapselung:** Verberge Details (Variablen, Hilfsfunktionen) in der Klasse, damit dein Code sauber bleibt.
- **Erleichtert Wartung & Erweiterung:** Ändere eine Klasse, alle Instanzen profitieren.

---

3. Grundaufbau einer Klasse

```
class KlasseName:
    def __init__(self, parameter1, parameter2):
        self.attribut1 = parameter1
        self.attribut2 = parameter2

    def methode1(self):
        # Funktionalität der Methode
        print(self.attribut1)

# Objekt erzeugen
objekt = KlasseName("Hallo", 123)
objekt.methode1()  # Ausgabe: Hallo
```

- `__init__` = Konstruktor (wird aufgerufen beim Erzeugen eines Objekts).
- `self` = Referenz auf das aktuelle Objekt (muss bei Methoden immer als erstes Argument stehen).
- Attribute und Methoden gehören zur Klasse/Objekt.

---

4. Beispiel: GUI-Klasse für dein Programm

```
import tkinter as tk

class MainGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("BiltonConvert")

    def start(self):
        self.root.mainloop()

# Instanz erzeugen und GUI starten
app = MainGUI()
app.start()
```

- GUI-Elemente und Logik sind in einer Klasse gebündelt.
- So hast du Kontrolle über das Fenster über Methoden.

---

5. Klassen in anderen Dateien (Modularität)

- Für große Programme: Teile deinen Code auf mehrere Dateien auf (Module).
- Beispiel:
    - `gui.py` enthält die GUI-Klasse
    - `filehandler.py` enthält Klassen/Funktionen zum Datei-Upload
    - `converter.py` enthält Klassen zum Konvertieren von Dateien

Importieren von Klassen aus anderen Dateien:

```
# In gui.py
class MainGUI:
    pass

# In main.py
from gui import MainGUI

app = MainGUI()
app.start()
```

- So bleibt dein Code übersichtlich und modular.

---

6. Klassenattribute vs. Instanzattribute

- **Instanzattribute:**
  Gehören nur zum jeweiligen Objekt, z.B. `self.name`.
- **Klassenattribute:**
  Werden von allen Objekten der Klasse geteilt.

```
class MyClass:
    klassen_attribut = 0  # Klassenattribut
```

```
    def __init__(self):
        self.instanz_attribut = 1  # Instanzattribut
```

---

7. Vererbung (Klassen erweitern)

- Du kannst eine Klasse von einer anderen erben, um Funktionalitäten wiederzuverwenden oder zu erweitern.

```
class BaseConverter:
    def convert(self):
        print("Basis-Konvertierung")

class AdvancedConverter(BaseConverter):
    def convert(self):
        print("Erweiterte Konvertierung")

ac = AdvancedConverter()
ac.convert()  # Ausgabe: Erweiterte Konvertierung
```

---

8. Tipps für deine Programmstruktur mit Klassen

- Lege GUI, Logik, Dateiverwaltung in **separate Klassen**.
- Nutze Module (andere Dateien), um große Programme übersichtlich zu halten.
- Vermeide zu große Klassen – lieber mehrere kleine, spezialisierte Klassen.
- Methoden sollen immer nur eine Aufgabe erfüllen.

---

9. Kurze Übersicht wichtige Begriffe

Begriff	Erklärung
Klasse	Bauplan für Objekte
Objekt	Instanz einer Klasse
Attribut	Eigenschaft eines Objekts
Methode	Funktion innerhalb einer Klasse
`__init__`	Konstruktor, Initialisierungsmethode
`self`	Verweis auf das aktuelle Objekt
Vererbung	Ableiten einer Klasse von einer anderen

Wenn du möchtest, kann ich dir auch Beispielprojekte oder Vorlagen für typische Klassen in deinem Programm schreiben, die du direkt verwenden kannst.

Sag Bescheid!