Sehr gern. Hier kommt ein **erweiterter und besonders gründlicher Lernzettel zu `AutoSuggest` von `prompt_toolkit`**. Ziel ist, dass du alles wirklich **verstehst**, es **sicher anwenden** kannst und weißt, **wann es sich lohnt**, welche **Vorteile** es bringt und wie du es **effektiv mit anderen Features kombinierst**.

📘 Lernzettel: `AutoSuggest` in `prompt_toolkit`

> Vollständig. Gründlich. Praxisnah.

🧠 Was ist `AutoSuggest`?

`AutoSuggest` ist ein Feature der `prompt_toolkit`-Bibliothek, das **dem Benutzer beim Tippen automatisch Vorschläge anzeigt** – direkt **inline** (in grauer Schrift).

➡️ **Ziel**: Schnelle, fehlerfreie und komfortable Eingabe.

Vergleichbar mit:

- Google-Suche → Vorschläge beim Tippen
- Shell / Terminal mit Autovervollständigung

🛠️ Wie funktioniert das?

`prompt_toolkit.prompt()` nimmt ein Argument namens `auto_suggest`. Dort übergibst du ein Objekt, das von der Basisklasse `AutoSuggest` abgeleitet ist.

📌 Signatur:

```
prompt(..., auto_suggest=AutoSuggestObject)
```

Das System überprüft **während des Tippens** laufend die bisherige Eingabe (`document.text_before_cursor`) und ruft die Methode `get_suggestion()` auf.

Wenn ein Vorschlag passt, wird er **grau eingefügt** und kann durch `[→]`, `[Tab]` oder `[Enter]` übernommen werden.

⚙️ Vorgefertigte Variante: `AutoSuggestFromHistory`

Diese Klasse ist **sofort einsatzbereit** und speichert **alle bisherigen Eingaben**.

Wenn du also beim ersten Mal `"filter_name=From"` eingegeben hast, wird dieser Text beim nächsten Mal vorgeschlagen, sobald du `"fil"` tippst.

✅ Vorteile:

- Kein Aufwand
- Sofortige Produktivitätssteigerung
- Merkt sich alles automatisch
- **Perfekt für interaktive Tools mit häufig wiederholten Eingaben**

Beispiel:

```
from prompt_toolkit import prompt
from prompt_toolkit.auto_suggest import AutoSuggestFromHistory

text = prompt("Filter: ", auto_suggest=AutoSuggestFromHistory())
```

🔧 Eigene Vorschläge: Eigene `AutoSuggest`-Klasse

Wenn du **intelligentere, inhaltlich gesteuerte Vorschläge** willst, schreibst du eine eigene Klasse.

Struktur:

```
from prompt_toolkit.auto_suggest import AutoSuggest, Suggestion

class MySuggest(AutoSuggest):
    def get_suggestion(self, buffer, document):
        text = document.text_before_cursor
        if text.startswith("in"):
            return Suggestion("inSubject")
        return None
```

Erklärung:

- `buffer`: der Eingabepuffer (kompletter Zustand)
- `document`: enthält `text_before_cursor` (aktueller Eingabetext)
- Rückgabe: `Suggestion(text)` oder `None`

🧪 Tipps zur Erstellung eigener Vorschläge

Frage	Empfehlung
Was soll vorgeschlagen werden?	Liste mit Regeln, Filternamen, Kommandos etc.
Wann soll etwas vorgeschlagen werden?	Nach Präfix, z. B. `"in"` → `"inSubject"`
Wie viele Vorschläge?	**Nur einen!** → `AutoSuggest` schlägt nur einen Wert vor
Mehrere Optionen?	Dann besser mit `Completer` kombinieren

💡 Wann ist `AutoSuggest` sinnvoll?

Sehr sinnvoll bei:

- Eingaben, die sich oft wiederholen
- Konfiguration von Filtern / Regeln
- Kommandozeilen-Tools mit vielen Optionen
- Eingabefeldern mit freien Werten, z. B.:
    - E-Mail-Betreffe
    - Bedingungen
    - Namen von Regeln
    - Regex-Muster

Nicht sinnvoll bei:

- Reinen Menüs (da besser `Completer`)
- Wenn Vorschläge zwingend korrekt sein müssen
- Bei sicherheitskritischer Eingabe (z. B. Passwörter)

🔁 Unterschiede zu `Completer` (z. B. `WordCompleter`)

Merkmal	`AutoSuggest`	`Completer`
Trigger	Automatisch beim Tippen	Manuell durch `[Tab]`
Vorschlagstyp	**Ein einziger Vorschlag** inline	**Mehrere Vorschläge** zur Auswahl
Darstellung	Grau und direkt im Eingabefeld	Liste oder Menü
Rückgabeklasse	`Suggestion`	`Completion`
Mehrere Vorschläge möglich?	❌ Nein	✅ Ja
UX-Typ	Komfort-Vervollständigung	Navigierbare Auswahl
Kombinierbar?	✅ Sehr gut mit `Completer` kombinierbar	✅ Ja

⚡ Best Practices zur Verwendung

✅ Nutze `AutoSuggestFromHistory`, wenn:

- dein CLI **häufige, freie Eingaben** verarbeitet
- du keine feste Vorschlagslogik brauchst

✅ Nutze eine eigene Klasse, wenn:

- du Vorschläge abhängig vom **Eingabetext** steuern willst
- du z. B. bei `"in"` → `"inSubject"` vorschlagen willst

✅ Kombiniere mit `Completer`, wenn:

- der Nutzer auch **auswählen** können soll (Tab-Menü)
- du sowohl **frei tippen** als auch **Werte vorgeben** willst

---

🧠 Typische Beispiele aus der Praxis

Anwendung	AutoSuggest-Vorteil
Filterdefinitionen	Merkt sich benutzerdefinierte Regeln
Feldnamen eingeben	Vorschlag: `subject_contains`, `from_address`, …
Regex-Muster	Erneute Verwendung gespeicherter Patterns
Regelgruppen konfigurieren	Schnelles Wiederholen häufig genutzter Begriffe
Custom Rule Language	Intelligente Vorschläge auf „in“, „is“, „contains“

✅ Fazit: Wann ist `AutoSuggest` lohnenswert?

Situation	Empfehlung
Interaktive CLI mit freier Eingabe	**Ja**
Regeln / Filter / Konfiguration	**Ja**
Autovervollständigung aus Verlauf	**Ja**
Auswahl aus fixen Optionen	**Besser mit Completer**
Wiederverwendbare Eingaben	**Perfekt geeignet**
UX verbessern ohne eigene Listen	**Sehr sinnvoll**

Wenn du willst, kann ich dir als Nächstes erklären:

- Wie du `AutoSuggest` und `Completer` **in einem prompt** einsetzt
- Wie du aus Nutzereingaben ein eigenes **Auto-Lernsystem für Vorschläge** machst (z. B. per Datei speichern)
- Wie du `AutoSuggest` mit Filtern, Regelsets oder AI kombinierst

Sag mir einfach Bescheid.