Natürlich, ich mache den Lernzettel jetzt noch ausführlicher und genauer – speziell zu **Shortcuts (Key Bindings) in prompt_toolkit**, wie du sie definierst, gruppierst, priorisierst, und wie du nebenläufig Prozesse während der Eingabe laufen lässt. Dabei erkläre ich alles Schritt für Schritt mit Beispielen und Kommentaren, damit du es sofort verstehst und selbst programmieren kannst.

🧠 Ausführlicher Lernzettel: prompt_toolkit Key Bindings (Shortcuts) & Nebenläufigkeit während Eingabe

---

1. Key Bindings / Shortcuts in prompt_toolkit – Grundlagen & Definition

---

1.1 Was sind Key Bindings?

- Key Bindings sind Tastenkommandos (Shortcuts), die du programmierst, damit bei Tastendruck bestimmte Funktionen ausgelöst werden.
- Sie verbessern die Bedienbarkeit deines CLI-Programms und machen es interaktiv.

1.2 Wie definiert man Key Bindings in prompt_toolkit?

- Du erstellst ein `KeyBindings()` Objekt.
- Du „hängst“ Funktionen an bestimmte Tastenkombinationen, indem du einen Dekorator `@kb.add()` verwendest.
- Diese Bindings werden dann einer `PromptSession` übergeben.

1.3 Beispiel: Einfacher Shortcut für Ctrl+X

```
from prompt_toolkit import PromptSession
from prompt_toolkit.key_binding import KeyBindings

# Erstelle KeyBindings-Objekt
kb = KeyBindings()

# Dekorator registriert Ctrl+X
@kb.add('c-x')
def _(event):
    print("Du hast Ctrl+X gedrückt!")

session = PromptSession(key_bindings=kb)
```

```
# Eingabeaufforderung mit Shortcut-Unterstützung
text = session.prompt("Eingabe: ")
print(f"Eingabe war: {text}")
```

Erklärung:

- `@kb.add('c-x')` heißt: Wenn Ctrl+X gedrückt wird, führe die folgende Funktion aus.
- Die Funktion erhält ein `event`-Objekt (z.B. um Kontext abzurufen).
- `PromptSession` weiß nun, welche Tastenkombinationen es überwachen soll.

1.4 Was ist `event`?

- `event` enthält wichtige Infos zur Tasteneingabe.
- `event.app` ist die Anwendung selbst.
- Du kannst `event.app.exit()` aufrufen, um z.B. die Eingabe abzubrechen.
- Beispiel:

```
@kb.add('c-c')  # Ctrl+C zum Abbrechen
def _(event):
    print("Eingabe abgebrochen!")
    event.app.exit()  # Beendet die Prompt-Session
```

1.5 Welche Tastenkombinationen kannst du benutzen?

- `c-` steht für Ctrl, z.B. `c-c` = Ctrl+C
- `s-` steht für Shift
- `a-` steht für Alt, z.B. `a-b` = Alt+B
- Du kannst mehrere Tasten kombinieren, z.B. `c-s-x` (Ctrl+Shift+X)
- Auch einzelne Tasten wie `tab`, `enter`, `escape` sind erlaubt.

---

2. Key Bindings Gruppen (Groups) & Prioritäten

2.1 Warum Gruppen?

- Du kannst verschiedene Tastenkombinationen logisch gruppieren.
- So hast du z.B. eine Gruppe für Navigation, eine für Bearbeitung, eine für Spezialfunktionen.
- Gruppen helfen auch bei Konfliktlösungen und modularem Code.

2.2 Wie erstellt man Gruppen?

- Verwende `KeyBindings()` pro Gruppe.
- Am Ende kannst du sie mit `merge_key_bindings` zusammenführen.

2.3 Beispiel: Zwei Gruppen – Navigation und Bearbeitung

```
from prompt_toolkit.key_binding import KeyBindings, merge_key_bindings
from prompt_toolkit import PromptSession

# Navigation-Shortcuts
nav_bindings = KeyBindings()

@nav_bindings.add('c-n')  # Ctrl+N
def _(event):
    print("Navigation: Nächster Eintrag")

@nav_bindings.add('c-p')  # Ctrl+P
def _(event):
    print("Navigation: Vorheriger Eintrag")

# Bearbeitungs-Shortcuts
edit_bindings = KeyBindings()

@edit_bindings.add('c-s')  # Ctrl+S
def _(event):
    print("Bearbeitung: Speichern")

@edit_bindings.add('c-z')  # Ctrl+Z
def _(event):
    print("Bearbeitung: Rückgängig")

# Alle Bindings zusammenführen
all_bindings = merge_key_bindings([nav_bindings, edit_bindings])

session = PromptSession(key_bindings=all_bindings)
text = session.prompt("Eingabe: ")
print(f"Deine Eingabe: {text}")
```

Erklärung:

- Zwei `KeyBindings`-Objekte definiert (nav und edit).
- Mit `merge_key_bindings` kombinierst du sie zu einem einzigen.
- So bleiben deine Shortcuts sauber getrennt und übersichtlich.

2.4 Prioritäten & Konflikte

- Wenn zwei Gruppen die gleiche Taste binden, gewinnt die zuletzt geladene.
- Nutze also sinnvoll Reihenfolge oder vermeide Überschneidungen.

---

3. Erweiterte Funktionen in Key Bindings

3.1 Zugriff auf Eingabetext, Cursorposition & mehr

- `event.current_buffer` liefert den Puffer mit dem aktuellen Text.
- Du kannst Text lesen, ändern oder Cursor bewegen.

```
@kb.add('c-l')  # Ctrl+L leert die Eingabe
def _(event):
    buff = event.current_buffer
    buff.text = ""  # Text löschen
```

3.2 Eingabe abbrechen oder Programm beenden

```
@kb.add('c-c')  # Ctrl+C bricht ab
def _(event):
    print("Programm wird beendet...")
    event.app.exit()  # Beendet prompt_toolkit sauber
```

3.3 Eingabe abschicken (Enter)

- Standardmäßig ist Enter schon gebunden (führt Eingabe aus).
- Du kannst eigene Aktionen auf Enter binden, z.B. Validierung.

---

4. Nebenläufigkeit (Concurrency) während prompt_toolkit läuft

---

4.1 Problem

- Du willst Eingaben machen, aber gleichzeitig andere Dinge im Hintergrund laufen lassen.
- Wichtig: Prompt blockiert nicht — Hintergrund muss nebenher aktiv sein.

4.2 Lösung 1: Threads mit prompt_toolkit

```
import threading
import time
from prompt_toolkit import PromptSession

def background():
    while True:
        print("Hintergrund läuft...")
        time.sleep(3)

thread = threading.Thread(target=background, daemon=True)
```

```
thread.start()

session = PromptSession()
text = session.prompt("Schreibe etwas: ")
print(f"Du hast geschrieben: {text}")
```

Erklärung:

- Der Thread läuft parallel, prompt wartet auf Eingabe.
- `daemon=True`: Thread endet automatisch mit Programm.

4.3 Lösung 2: Asyncio mit prompt_toolkit (moderner)

```
import asyncio
from prompt_toolkit import PromptSession

async def background():
    while True:
        print("Async Hintergrund läuft...")
        await asyncio.sleep(3)

async def main():
    asyncio.create_task(background())
    session = PromptSession()
    text = await session.prompt_async("Schreibe was: ")
    print(f"Du hast: {text}")

asyncio.run(main())
```

Erklärung:

- Nutzt `asyncio` Event Loop, modern und performant.
- `prompt_async` ermöglicht async Eingabe, während background läuft.

4.4 Vorsicht beim Drucken während Eingabe

- Wenn du während der Eingabe printest, kann die Zeile verwischen.
- prompt_toolkit hat Methoden, um Statusleisten zu zeigen oder Ausgaben zu puffern.

---

5. Zusammenfassung & Best Practices

Thema	Wichtigste Punkte
Key Bindings definieren	`KeyBindings()`, `@kb.add('c-x')`
Event-Objekt nutzen	Zugriff auf Buffer, App (`event.current_buffer`, `event.app.exit()`)
Gruppen erstellen	Mehrere `KeyBindings()` mit `merge_key_bindings()` kombinieren
Nebenläufigkeit	Threads (`threading.Thread`) oder Asyncio

Thema	Wichtigste Punkte
	(`asyncio.create_task()`)
Eingabe nicht blockieren	prompt_toolkit ist async-freundlich, nutze `prompt_async` bei Bedarf
Print während Eingabe vorsichtig	Verwende Statusleisten statt `print` für saubere UI

---

6. Was du jetzt machen kannst

- Erstelle eigene Tastenkombinationen für dein Programm.
- Organisiere sie modular in Gruppen (z.B. Navigation, Aktionen).
- Starte einen Hintergrund-Thread, der z.B. Emails prüft, während prompt auf Eingabe wartet.
- Nutze `prompt_async` bei Bedarf für moderne async-Programme.
- Vermeide Print-Störungen mit Statusleisten oder Puffern.

Wenn du möchtest, kann ich dir auch noch zeigen, wie man in prompt_toolkit z.B. **Statusleisten** oder **Popup-Menüs** parallel zur Eingabe anzeigt — das ist der nächste Schritt, wenn du die Basics drauf hast!

Falls du noch Fragen hast, sag gern Bescheid!