Super! Hier ist der **aktualisierte Lernzettel zu `concurrent.futures`**, jetzt mit:

- **einfachen, kommentierten Codebeispielen zu jedem Punkt**
- **Dummy-Beispielen** (keine Mail- oder Bash-Beispiele, nur verständliche Testfunktionen)
- einem **Extra-Abschnitt**, wie du Prozesse sauber beendest oder einzeln steuerst

---

🧠 Lernzettel: `concurrent.futures` in Python

Ziel: Paralleles und schnelles Ausführen mehrerer Aufgaben

Jetzt mit ausführlichen Beispielen für Anfänger

🟨 Was ist parallele Programmierung?

Statt Aufgaben nacheinander auszuführen, willst du mehrere Dinge gleichzeitig starten.
z. B.: Während etwas 5 Sekunden rechnet, soll das Programm weiterlaufen.

🔹 **Beispiel:**

```
# Beispiel: normale Ausführung, linear (nicht parallel)
import time

def aufgabe():
    print("Starte Aufgabe...")
    time.sleep(3)  # Tut so, als ob hier etwas 3 Sekunden dauert
    print("Aufgabe fertig.")

aufgabe()
print("Danach weiter...")  # Diese Zeile wird erst nach 3 Sekunden ausgeführt
```

➡️ Lösung: Mit paralleler Programmierung kannst du während der 3 Sekunden andere Dinge tun.

🧵 Unterschied: Thread vs. Prozess

Begriff	Thread	Prozess
Speicher	Teilt Speicher mit Hauptprozess	Eigenen Speicher
Geschwindigkeit	Schneller zu starten	Langsamer, aber unabhängig
Einsatz	Warten auf Dinge (E-Mails, Web)	Viel Rechnen (z. B. Mathe)

🔹 **Beispiel: Unterschied zeigen**

```
from concurrent.futures import ThreadPoolExecutor, ProcessPoolExecutor
import time

def rechen_aufgabe(x):
    time.sleep(2)
```

```
    return x * x

# Mit Threads (gut bei Warten)
with ThreadPoolExecutor() as executor:
    zukunft = executor.submit(rechen_aufgabe, 5)
    print("Warte auf Ergebnis...")
    print(zukunft.result())

# Mit Prozessen (gut bei CPU-Last)
with ProcessPoolExecutor() as executor:
    zukunft = executor.submit(rechen_aufgabe, 5)
    print("Warte auf Ergebnis (Prozess)...")
    print(zukunft.result())
```

📦 Was ist `concurrent.futures`?

Das Modul gibt dir eine einfache Art, Aufgaben **parallel** zu starten – ohne viel Technik drumherum.

- 🔹 **Beispiel: einfache Nutzung**

```
from concurrent.futures import ThreadPoolExecutor

def dummy():
    print("Ich laufe jetzt im Hintergrund!")

with ThreadPoolExecutor() as executor:
    executor.submit(dummy)  # Startet dummy() im Hintergrund
```

🔧 Die 3 wichtigsten Funktionen

`submit()` – einzelne Aufgaben starten

```
from concurrent.futures import ThreadPoolExecutor

def warte_funktion(name):
    print(f"{name} gestartet.")
    return f"{name} ist fertig."

with ThreadPoolExecutor() as executor:
    future = executor.submit(warte_funktion, "Test1")
    ergebnis = future.result()  # .result() wartet, bis die Aufgabe fertig ist
    print("Ergebnis:", ergebnis)
```

🟡 Verwende `submit()`, wenn du einzelne Aufgaben mit eigenen Parametern starten willst.

`map()` – viele Aufgaben gleichzeitig starten

```
from concurrent.futures import ThreadPoolExecutor

def quadrat(x):
    return x * x
```

```
werte = [1, 2, 3, 4]

with ThreadPoolExecutor() as executor:
    ergebnisse = executor.map(quadrat, werte)

for r in ergebnisse:
    print("Quadrat:", r)
```

🟡 Verwende `map()`, wenn du **die gleiche Funktion auf viele Werte** anwenden willst.

`as_completed()` – Ergebnisse zurückbekommen, sobald fertig

```
from concurrent.futures import ThreadPoolExecutor, as_completed
import time

def verzögert(i):
    time.sleep(i)
    return f"Erledigt nach {i} Sekunden"

with ThreadPoolExecutor() as executor:
    futures = [executor.submit(verzögert, i) for i in [3, 1, 2]]

    for future in as_completed(futures):
        print(future.result())  # Gibt Ergebnisse aus, sobald sie fertig sind
```

➡️ Hier siehst du: **Die schnellste Aufgabe kommt zuerst zurück**, egal in welcher Reihenfolge sie gestartet wurde.

🔁 Nutzung in `while True`-Schleife (wie bei dir)

- 🔹 **Beispiel:**

```
import time
from concurrent.futures import ThreadPoolExecutor

def verarbeite_irgendwas():
    print("Aufgabe gestartet")
    time.sleep(2)
    print("Aufgabe fertig")

with ThreadPoolExecutor() as executor:
    while True:
        # Beispiel: alle 5 Sekunden Aufgabe starten
        zukunft = executor.submit(verarbeite_irgendwas)
        print("Warte auf nächste Schleife...")
        time.sleep(5)
```

➡️ Hier läuft die Funktion parallel, aber die Schleife wartet trotzdem 5 Sekunden, bevor sie erneut startet.

🔍 `Future`-Objekte

Ein `Future` ist ein **Platzhalter für ein späteres Ergebnis**.

- **Beispiel:**

```
from concurrent.futures import ThreadPoolExecutor
import time

def hallo():
    time.sleep(1)
    return "Hallo!"

with ThreadPoolExecutor() as executor:
    future = executor.submit(hallo)
    print("... Funktion läuft im Hintergrund ...")
    print(future.result())  # Wartet und gibt dann das Ergebnis
```

🚫 Vorsicht bei `result()`

Wenn du direkt nach `submit()` gleich `result()` aufrufst, **wartest du wieder**.

- **Richtig:**

```
future = executor.submit(aufgabe)
# Keine .result() direkt!
```

- **Falsch (blockierend):**

```
future = executor.submit(aufgabe)
future.result()  # Das blockiert → keine echte Parallelität mehr
```

🛠️ executor.shutdown() richtig verwenden

- **Beispiel:**

```
from concurrent.futures import ThreadPoolExecutor
import time

def aufgabe():
    print("Start")
    time.sleep(1)
    print("Fertig")

executor = ThreadPoolExecutor()
executor.submit(aufgabe)
executor.shutdown(wait=True)  # wartet, bis alle Aufgaben fertig sind
```

🧪 Übungsidee: Mehrere Dummy-Funktionen gleichzeitig

- Schreibe zwei Funktionen mit `sleep()`

- Starte beide mit `submit()`
- Lass die Hauptschleife `print()` ausgeben, während die anderen Aufgaben laufen

---

🧹 Extra: Einen Prozess beenden, während andere weiterlaufen

Du kannst `Future.cancel()` verwenden, um einzelne Aufgaben **abzubrechen**, **wenn sie noch nicht laufen**:

🔹 **Beispiel:**

```
from concurrent.futures import ThreadPoolExecutor
import time

def warten():
    time.sleep(5)
    return "Fertig"

with ThreadPoolExecutor() as executor:
    f1 = executor.submit(warten)
    f2 = executor.submit(warten)

    abgebrochen = f2.cancel()  # Nur erfolgreich, wenn noch nicht gestartet
    print("Aufgabe 2 abgebrochen?", abgebrochen)

    print("Ergebnis Aufgabe 1:", f1.result())
```

> ✅ Wenn du einzelne Aufgaben später gezielt beenden oder kontrollieren willst, musst du dafür `Future`-Objekte im Griff haben.
> Das geht **nicht**, wenn sie schon laufen – dafür brauchst du spezielle Prozesskontrolle (z. B. mit `multiprocessing` oder `signal`).

🧭 Fazit & Merksätze

- Nutze `ThreadPoolExecutor` für alles, was **nicht rechnen muss** (wie E-Mails abrufen, Bash starten).
- Starte Funktionen mit `submit()`, **nicht blockieren!**
- Kontrolliere Rückgaben mit `.result()` **später** oder nutze `as_completed()`
- Halte deinen Code einfach und gut geloggt, um zu sehen, was parallel passiert.

Wenn du willst, kann ich dir zu jedem dieser Beispiele auch eine **kleine Übungsaufgabe** geben – sag einfach Bescheid!