LU03.L10 - Timer und API-Call

Einführung

Das bereitgestellte Programm demonstriert die Kraft der Asynchronität in Python. Es verwendet `asyncio` und `httpx` für asynchrone Operationen und führt zwei parallele Aufgaben aus: Einen API-Aufruf alle 3 Sekunden und einen Timer, der jede Sekunde hochzählt.

Komponenten-Erklärung

1. **Callback-Funktion `api_response_callback`**:
   - Diese Funktion wird als Callback verwendet und gibt die Daten aus, die von der API empfangen wurden.
   - Es nimmt die `response_data` als Argument und gibt sie direkt in der Konsole aus.
2. **Asynchrone Funktion `fetch_data_from_api`**:
   - Diese Funktion ruft alle 3 Sekunden asynchron eine API auf.
   - Sie verwendet den `httpx.AsyncClient`, um eine asynchrone HTTP-Anfrage zu senden.
   - Nach Erhalt der Antwort von der API wird der bereitgestellte Callback mit den Daten aufgerufen.
   - Die Funktion pausiert dann für 3 Sekunden, bevor sie erneut eine Anfrage an die API sendet.
3. **Asynchrone Funktion `async_timer`**:
   - Diese Funktion fungiert als asynchroner Timer.
   - Sie verwendet `asyncio.sleep` für die Verzögerung von einer Sekunde.
   - In jeder Iteration ihrer Schleife gibt sie einen Zähler aus und erhöht ihn dann um eins.
4. **Asynchrone Funktion `main`**:
   - Dies ist die Hauptfunktion, die beide oben genannten asynchronen Funktionen parallel ausführt.
   - Sie verwendet `asyncio.create_task`, um sowohl den API-Aufruf als auch den Timer als separate, gleichzeitig laufende Aufgaben zu starten.
   - Sie wartet dann darauf, dass beide Aufgaben abgeschlossen sind.
5. **Programmeinstiegspunkt `if __name__ == '__main__'`**:
   - Dies ist der Einstiegspunkt des Programms.
   - Er ruft die `main`-Funktion mit `asyncio.run` auf, was den Beginn der asynchronen Ausführung markiert.

Zusammenfassung

Das Programm demonstriert, wie man asynchrone Operationen in Python effizient handhabt. Es zeigt, wie man gleichzeitig einen asynchronen API-Aufruf und einen asynchronen Timer ausführt, ohne dass der eine den anderen blockiert. Das Ergebnis ist eine nahtlose Ausführung beider Aufgaben, wobei der Timer jede Sekunde hochzählt und alle 3 Sekunden eine API-Antwort ausgegeben wird.

```
import asyncio
import httpx
```

```
def api_response_callback(response_data):
    """
    Callback-Funktion, die aufgerufen wird, nachdem die API-Antwort
empfangen wurde.

    Args:
    - response_data: Die Daten, die von der API empfangen wurden.

    Returns:
    - None, da die Daten direkt in der Konsole ausgegeben werden.
    """
    print(f"API Response: {response_data}")


async def fetch_data_from_api(callback):
    """
    Diese Funktion ruft asynchron alle 3 Sekunden eine API
('https://hub.dummyapis.com/delay?seconds=3') auf, die eine
    Verzögerung von 3 Sekunden simuliert. Nachdem die Daten von der API
abgerufen wurden, wird der bereitgestellte
    Callback mit den Daten aufgerufen.

    Args:
    - callback: Die Callback-Funktion, die aufgerufen wird, nachdem die API-
Daten empfangen wurden.

    Returns:
    - None, da die Daten an die Callback-Funktion weitergegeben werden.
    """
    while True:
        async with httpx.AsyncClient() as client:
            response = await
client.get('https://hub.dummyapis.com/delay?seconds=3')
        callback(response)
        await asyncio.sleep(3)


async def async_timer():
    """
    Diese Funktion fungiert als asynchroner Timer, der jede Sekunde
hochzählt und den aktuellen Wert ausgibt.
    Sie verwendet `asyncio.sleep` für die Verzögerung und führt eine endlose
Schleife aus, die den Zähler jede Sekunde erhöht.

    Returns:
    - None, da der Zählerstand direkt in der Konsole ausgegeben wird.
    """
```

```
    count = 0
    while True:
        print(count)
        count += 1
        await asyncio.sleep(1)


async def main():
    """
    Hauptfunktion, die beide asynchrone Funktionen, `fetch_data_from_api`
und `async_timer`, parallel ausführt.
    Sie verwendet `asyncio.create_task` um die beiden Funktionen als
separate, gleichzeitig laufende Tasks zu starten.

    Returns:
    - None, da alle Ausgaben direkt in den jeweiligen Funktionen erfolgen.
    """

    api_task =
asyncio.create_task(fetch_data_from_api(api_response_callback))
    timer_task = asyncio.create_task(async_timer())

    await api_task
    await timer_task


if __name__ == '__main__':
    asyncio.run(main())
```
