2025/11/27 21:45 1/3 LU03.A10 - Timer und API-Call

LU03.A10 - Timer und API-Call

In dieser Übung werden Sie die Kraft der Asynchronität in Python erleben. Ihre Aufgabe ist es, einen asynchronen Timer zu erstellen, der alle 3 Sekunden einen API-Aufruf ausführt. Gleichzeitig sollte ein separater asynchroner Prozess eine andere Aufgabe durchführen, ohne durch den API-Aufruf unterbrochen zu werden.

Detaillierte Aufgabenstellung

API-Aufruf: Ihr Programm sollte alle 3 Sekunden einen asynchronen Aufruf an die folgende URL machen: https://postman-echo.com/delay/3. Nachdem eine Antwort vom Server erhalten wurde, sollte diese in der Konsole ausgegeben werden.

Zweite Aufgabe: Parallel zum API-Aufruf sollte Ihr Programm eine andere asynchrone Aufgabe durchführen. Diese Aufgabe sollte darin bestehen, jede Sekunde eine Zahl auszugeben, die ständig um 1 erhöht wird (ein asynchroner Timer).

Hinweis: Verwenden Sie asyncio in Kombination mit einer Bibliothek wie httpx für asynchrone HTTP-Anfragen.

Erwartetes Verhalten

Wenn Sie Ihr Programm ausführen, sollten Sie sehen, dass der Timer jede Sekunde hochzählt. Alle 3 Sekunden wird Ihr Programm eine Pause einlegen, um auf die Antwort des API-Aufrufs zu warten. Nachdem die Antwort erhalten wurde, wird sie in der Konsole angezeigt, und der Timer fährt ohne Unterbrechung fort.

Beispiel

Ein möglicher Output könnte wie folgt aussehen:

```
0
1
2
3
API Response: <Response [200 0K]>
4
```

Vorlage

```
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.
   #TODO: Hier die Daten verarbeiten
async def fetch data from api(callback):
   Diese Funktion ruft asynchron alle 3 Sekunden eine API
('https://run.mocky.io/v3/685db531-06e7-4d66-bbf6-99de9f2feab3?mocky-delay=3
000ms') 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.
   # TODO: Hier in einer Endlosschleife die API aufrufen und die Daten an
die Callback-Funktion übergeben
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.
   #TODO: Hier den Timer implementieren
async def main():
```

https://wiki.bzz.ch/ Printed on 2025/11/27 21:45

2025/11/27 21:45 3/3 LU03.A10 - Timer und API-Call

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())



From:

https://wiki.bzz.ch/ - BZZ - Modulwiki

Permanent link:

https://wiki.bzz.ch/modul/m323/learningunits/lu03/aufgaben/timer?rev=1764102250

Last update: 2025/11/25 21:24

