# LU03.A10 - Timer and API call

In this exercise you will experience the power of asynchrony in Python. Your task is to create an asynchronous timer that executes an API call every 3 seconds. At the same time, a separate asynchronous process should perform another task without being interrupted by the API call.

## Detailed task

**API call:** Your programme should make an asynchronous call to the following URL every 3 seconds: https://run.mocky.io/v3/685db531-06e7-4d66-bbf6-99de9f2feab3?mocky-delay=3000ms . Once a response has been received from the server, this should be displayed in the console.

**Second task:** Parallel to the API call, your programme should perform another asynchronous task. This task should consist of outputting a number every second that is constantly incremented by 1 (an asynchronous timer).

**Note:** Use asyncio in combination with a library such as httpx for asynchronous HTTP requests.

## Expected behaviour

When you run your programme, you should see the timer count up every second. Every 3 seconds your programme will pause to wait for the API call response. After the response is received, it will be displayed in the console and the timer will continue without interruption.

## Example

A possible output could look like this:

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

...
```

## Template

```
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():
    """
    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())
```

---