LU03.L03 - Task Scheduler

Simpel: Alle Tasks haben identische Verzögerungen

```
import time
def task1():
    print('Task 1 executed!')

def task2():
    print('Task 2 executed!')

def task_scheduler(tasks, delay):
    for task in tasks:
        task()
        time.sleep(delay)

if __name__ == '__main__':
    tasks = [task1, task2]
    delay = 2
    task_scheduler(tasks, delay)
```

Expert: Jeder Task hat eine unterschiedliche Verzögerung

```
def task_scheduler_expert(tasks, delays):
    for task, delay in zip(tasks, delays):
        task()
        time.sleep(delay)

if __name__ == '__main__':
    tasks = [task1, task2]
    delays = [2,3]
    task_scheduler_expert(tasks, delays)
```

Erklärung

- **Schritt 1:** Die `task_scheduler`-Funktion wird so geändert, dass sie eine Liste von Verzögerungen akzeptiert, wobei jede Verzögerung einem bestimmten Task entspricht.
- **Schritt 2:** Die `zip`-Funktion wird verwendet, um die Tasks und Verzögerungen in der `for`-Schleife zu verbinden. Die `zip`-Funktion verbindet die beiden Listen (Tasks und Verzögerungen) so, dass sie als Paare verarbeitet werden können. Für jeden Durchlauf der Schleife wird ein Task und die entsprechende Verzögerung aus den beiden Listen entnommen und ausgeführt.
- **Schritt 3:** Der Scheduler wird mit den Tasks und den individuellen Verzögerungen aufgerufen.

Diese Musterlösungen zeigen die Flexibilität von First-Class

Functions in Python und wie man verschiedene Verzögerungen in der Ausführung von Code hinzufügen kann.

Exkurs zur ZIP-Funktion

Die `zip`-Funktion kombiniert die Elemente mehrerer Iterables (wie Listen oder Tupel) und gibt einen Iterator von Tupeln zurück, wobei das erste Element in jedem Argument das erste Element im ersten Tupel usw. ist. Im obigen Beispiel würde die Verwendung von `zip` mit den Listen `tasks` und `delays` zu einem Iterator führen, der die folgenden Tupel enthält:

```
(<function task1 at 0x10044d1b0>, 1)
(<function task2 at 0x10044d900>, 3)
```

Wenn du diesen Iterator in eine Liste umwandeln möchtest, kannst du die `list`-Funktion verwenden. Zum Beispiel:

```
tasks = [task1, task2]
delays = [1, 3]

zipped_list = list(zip(tasks, delays))
# zipped_list wäre nun: [(<function task1 at 0x10044d1b0>,
1), (<function task2 at 0x10044d900>, 3)]
```

Natürlich kann `zip` auch für andere Datentypen verwendet werden:

```
# Definieren von zwei Listen
names = ['Alice', 'Bob', 'Charlie']
ages = [25, 30, 22]

# Verknüfen von zwei Listen in eine neue Liste von Tuples
zipped_list = list(zip(names, ages)) # zipped_list wäre nun:
[('Alice', 25), ('Bob', 30), ('Charlie', 22)]

# Verwenden der zip-Funktion, um die Listen zu kombinieren
und direkt mit dem Iterator zu interieren
for name, age in zip(names, ages):
    print(f'{name} is {age} years old')
```

Führt zur Ausgabe:

```
Alice is 25 years old
Bob is 30 years old
Charlie is 22 years old
```