LU03.A03 - Task Scheduler

Simpel: Alle Tasks haben identische Verzögerungen

Erstelle einen Task-Scheduler, der eine Liste von Funktionen mit einer festen Verzögerung zwischen jedem Task ausführt.

Vorgehen

- Definiere zwei oder mehr einfache Funktionen, die nur eine Ausgabe wie `print('Task 1 executed!')` ausführen.
- Schreibe eine Funktion namens `task_scheduler`, die eine Liste von Funktionen und eine feste Zeitverzögerung als Argumente akzeptiert. Dieser Taskscheduler führt die Tasks in der Liste dann mit der Zeitverzögerung aus.
- Benutze die `time.sleep`-Methode, um die Verzögerung zwischen den einzelnen Tasks zu implementieren.

```
def task1():
    """
    Eine einfache Funktion, die eine Meldung ausgibt, um zu signalisieren,
dass Task 1 ausgeführt wurde.
    """
    pass


def task2():
    """
    Eine einfache Funktion, die eine Meldung ausgibt, um zu signalisieren,
dass Task 2 ausgeführt wurde.
    """
    pass


def task_scheduler(tasks, delay):
    """
    Führt eine Liste von Tasks mit einer festen Zeitverzögerung zwischen
jedem Task aus.

    Args:
    tasks (list): Eine Liste von Funktionen, die ausgeführt werden sollen.
    delay (int): Die Zeitverzögerung in Sekunden zwischen den einzelnen
Tasks.

    Returns:
    None
```

```
    """
    pass

def task_scheduler_expert(tasks, delays):
    """
    Führt eine Liste von Tasks mit unterschiedlichen Zeitverzögerungen
zwischen jedem Task aus.

    Args:
    tasks (list): Eine Liste von Funktionen, die ausgeführt werden sollen.
    delays (list): Eine Liste von Zeitverzögerungen in Sekunden für die
jeweiligen Tasks.

    Returns:
    None
    """
    pass

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

Expert: Jeder Task hat eine unterschiedliche Verzögerung

Erweitere den Task-Scheduler, damit er für jeden Task eine individuelle Verzögerung hat.

Aufgabenstellung

- Modifiziere die `task_scheduler`-Funktion, so dass sie zwei Listen als Argumente akzeptiert: eine mit den Tasks und eine mit den Verzögerungen. Nennen die Funktion `task_scheduler_expert`
- Stelle sicher, dass die Verzögerungen individuell für jeden Task angewendet werden.

```
# define your tasks here

def task_scheduler(tasks, delay):
    # This function should be done by now
    pass

def task_scheduler_expert(tasks, delays):
```

```
    # your code goes here
    pass

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

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
```
