concurrent.futures

Jan Popko

Python Advanced

Business Trends Academy (BTA) GmbH
bta BUSINESS TRENDS ACADEMY
Nestorstraße 36
D-10709 Berlin
Tel.: +49 (0) 30 894 087 57
Fax: +49 (0) 30 895 429 94
Geschäftsführer:
Gabriele Fleischmann-Hahn
Maxi-Marlen Fleischmann
Hauptsitz des Unternehmens:
Nestorstraße 36, D-10709 Berlin
HRB 115251 B / Amtsgericht Berlin Charlottenburg
Steuer-Nr. 27/248/31179

Modul concurrent.futures

- mit diesem Modul kann man Threads und Prozesse erstellen
- beides hat das selbe Interface

```
import concurrent.futures as cf

def func(x):
    *does something
    return *something

if __name__ == "__main__":

    Executor = cf.ProcessPoolExecutor()
    future = executor.submit(func, args)
```

Gleicher Syntax mit: `ThreadPoolExecutor`
Mögliche Executor Objekte: `submit, map`

Argumente für Executors

`max_workers`

Threads:

-Anzahl der maximalen Threads

- bei None oder 0: Anzahl der Prozessoren*5

Prozesse:

- Anzahl der maximalen Prozesse
- bei None oder 0: Anzahl der Prozessoren

`thread_name_prefix`

Threads:

- alle erstellten Threads werden diesen Präfix+Integer haben
- ähnlich wie Thread.name

Executor Objekte

`.submit(func, *args)`

- Funktion, Argumente
- returns ein Future Objekt

`.map(func, *iterables, timeout = None)`

- Funktion, Iterables,
- timeout = x: Gibt TimeoutError wenn das resultat von `.__next__()` nicht nach x Sekunden bereit ist, bei None gibt es kein Timeout.
- returns ein Iterable

`.shutdown(wait = True)`

- wenn wait True ist, dann wartet das Programm bis alle Ressourcen im Zusammenhang mit dem Executor frei sind, wenn wait = False ist, dann wird das Programm sofort weiter laufen
- ähnlich der join() Funktion bei Threads/Prozessen

Future Objekte

`.cancel()`
- versucht die Future abzubrechen
- True wenn erfolgreich, False wenn nicht

`.cancelled()`
- True, wenn die Future erfolgreich abgebrochen wurde

`.running()`
- True , wenn die Future gerade läuft und nicht gecancelled werden kann

`.done()`
- True , wenn die Future erfolgreich gecancelled wurde oder fertig ist

`.result(timeout=None)`
- gibt den Wert der Future wieder
- Programm wartet bis es ein result gibt (ähnlich wie join)
- wenn der Vorgang nicht komplett ist, vor dem Wert des timeouts: TimeoutError
- bei timeout = None – endlos Zeit

Mögliche Ausgabe des Resultats

```
import concurrent.futures as cf

def func(x):
    return x * x

if __name__ == "__main__":
    with cf.ProcessPoolExecutor() as executor:
        future = executor.submit(func, 10)
    print(future.result())


    with cf.ProcessPoolExecutor() as executor:
        future = executor.map(func, range(10))
    for elem in future:
        print(elem)
```