Würgeschlange 3 - Lesson 11

Tobias Maschek, Viktor Reusch
https://github.com/jemx/wise1920-python

mit Materialien von Felix Döring, Felix Wittwer https://github.com/fsr/python-lessons

21. Januar 2020

Python-Kurs

Gliederung

1. Async/Await
2. multiprocessing
3. subprocess

Async/Await

Async/Await

- **Problem:** blocking operation, d. h. warten auf andere Ressource (z. B. HTTP-Request)
- **Idee:** Währenddessen etwas berechnen/weitere Ressourcen anfragen
- **Lösung:** Asynchrone Programm-Ausführung
- event loop: führt *async* Funktionen sequentiell in globaler Schleife aus
- *async* Funktionen in event loop registrieren
- Bei blocking operation warten *await*
- Unterdessen andre Funktion in event loop ausführen
- Wenn Ressource verfügbar pausierte Funktion fortsetzen
- Sinnvoll bei Netzwerk-Ressourcen: häufige Verwendung in *JavaScript*

Hey!

```python
import asyncio
from datetime import datetime

async def sleep2sec(m):
    await asyncio.sleep(2)

async def main():
    start = datetime.now()
    await sleep2sec("first")
    await sleep2sec("second")
    print(f"program took {datetime.now() - start}")


asyncio.run(main())  # program took 0:00:04.001479
```

Hey! Wake Up!

```python
import asyncio
from datetime import datetime

async def sleep2sec(m):
    await asyncio.sleep(2)

async def main():
    start = datetime.now()
    s1 = sleep2sec("first")
    s2 = sleep2sec("second")
    await s1
    await s2
    print(f"program took {datetime.now() - start}")


asyncio.run(main())  # program took 0:00:04.000503
```

WAKE UP FASTER!!!111elf

```python
import asyncio
from datetime import datetime

async def sleep2sec(m):
    await asyncio.sleep(2)

async def main():
    start = datetime.now()
    s1 = asyncio.create_task(sleep2sec("first"))
    s2 = asyncio.create_task(sleep2sec("second"))
    await s1
    await s2
    print(f"program took {datetime.now() - start}")


asyncio.run(main())  # program took 0:00:02.000283
```

HTTP-Beispiel — Vorbereitung

```python
import asyncio

async def print_header_line(host, path):
    reader, writer = await asyncio.open_connection(host, 443, ssl=True)
    query = f"GET {path} HTTP/1.0\r\nHost: {host}\r\n\r\n"
    writer.write(query.encode('latin-1'))
    line = await reader.readline()
    line = line.decode('latin1').strip()
    writer.close()
    return line
```

HTTP-Beispiel — Sequenziell

```python
import asyncio

# ...

async def run_all():
    coroutines = (
        print_header_line("httpbin.org", f"/status/{status}")
        for status in [200, 418]
    )
    for coroutine in coroutines:
        print(await coroutine)

asyncio.run(run_all())

# HTTP/1.1 200 OK
# HTTP/1.1 418 I'M A TEAPOT
# total time: 1.120 seconds
```

HTTP-Beispiel — Asynchron

```python
import asyncio

# ...

async def run_all():
    tasks = [
        asyncio.create_task(
            print_header_line("httpbin.org", f"/status/{status}")
        ) for status in [200, 418]
    ]
    for task in tasks:
        print(await task)

asyncio.run(run_all())

# HTTP/1.1 200 OK
# HTTP/1.1 418 I'M A TEAPOT
# total time: 0.608 seconds
```

multiprocessing

So viele Würgeschlangen!

- Wie nutze ich mehrere Kerne und Hyper-Threading aus?
- Starten rechenintensiver Python-Funktionen Unterprozessen
- Auch bei gleichzeitiger Ausführung (Client und Server gleichzeitig)
- *multiprocessing* startet gewünschte Python-Funktion in neuem Prozess
- Erzeugt viel Overhead: nur nutzen, wenn nötig/sinnvoll

Dokumentation

Multithreading?

- Threads sind parallele Programmabläufe in einem Prozess
- Threads haben gegenseitigen Zugriff auf Hauptspeicher: **Fehleranfällig**
- **Problem:** global interpreter loc (GIL)
- Vereinfacht Speicherverwaltung von cPython (C-Unterbau für Python-Funktionen)
- Objekte können von threads nicht gleichzeitig bearbeitet werden

Sequenziell

```python
def factorial(n):
    acc = 1
    for i in range(2, n + 1):
        acc *= i
    #print(f"{n}! = {acc}")

for n in range(20000, 20010):
    factorial(n)
# total time: 42.439 seconds
```

Parallel

```python
from multiprocessing import Process

def factorial(n):
    acc = 1
    for i in range(2, n + 1):
        acc *= i
    #print(f"{n}! = {acc}")

if __name__ == "__main__":  # main required for multiprocessing
    processes = (Process(target=factorial, args=(n, )) for n in range(30000,
    30100))
    for p in processes:
        p.start() # start p in parallel
    for p in processes:
        p.join() # wait for p to finish
    # total time: 12.851 seconds
```

Queue

```python
from multiprocessing import Process, Queue

def factorial(n, queue: Queue):
    acc = 1
    for i in range(2, n + 1):
        acc *= i
    queue.put((n, acc))

if __name__ == "__main__":
    q = Queue()
    p = Process(target=factorial, args=(10, q))
    p.start()
    p.join()
    print(q.get())  # (10, 3628800)
```

Aufgabe 11-1

Überprüft ob drei gegebene Zahlen Primzahlen sind.

- Die Zahlen können (und werden) sehr groß. Sinnvoll ist es die drei Zahlen **gleichzeitig** zu überprüfen.
- Nutzt dafür multiprocessing.

subprocess

Prozesse Starten

- Wie ruft man andere Programme von Python auf?
- *subprocess*-Modul nutzen
- *run*-Funktion bekommt eine Parameter-Liste übergeben
- Erstes Element in Liste ist das aufzurufende Programm
- Weiteren Element sind Programm-Parameter
- Ausgabe des Aufgerufenen Programms wird direkt ausgegeben

Dokumentation

Beispiel — 1

```python
import subprocess

# open calculator in Windows
subprocess.run(["calc"])

# capture python version to process.stdout instead of printing
process = subprocess.run(
    ["python", "--version"],
    capture_output=True, text=True  # use str instead of bytes
)
print(process.returncode) # 0
print(process.stdout) # Python 3.8.1
```

Beispiel — 2

```
import subprocess

# enter 2 + 2 into the opened subprocess
subprocess.run(["python", "-i"], input="2 + 2", text=True)
# Python 3.8.1 [...]
# Type "help", "copyright", "credits" or "license" for more information.
# >>> ...
# 4
# >>>

# list the directory content in Windows
# execute subprocess in directory mytest
subprocess.call(["dir"], shell=True, cwd="mytest")
# [...]
# 27.08.2019  13:30                18 .gitignore
# 27.08.2019  13:33                18 src
# [...]
```