Asynchronous programming

- CPU-bound work normally done by synchronous blocks
  - Exception: work can be parallelised
- Example:

```
N = sqrt(M)
K = N + 5
```

- Distributed applications often require going “out”
  - Remote calls (e.g. web services)
  - I/O (e.g. remote storage)
- During these calls CPU is idle
  - Waiting for result of network/IO
  - We can do better

Asynchronous programming

- Synchronous (**blocking**) call

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

synchronous 

synchronous 

Example: cooking pasta

- Requires several steps

1. Boiling water (5 min)
2. Cook pasta on water (5 min)
3. Cook tomato sauce (10 min)
4. Make salad (10 min)

- (Synchronous) total cooking time = 30 min

Example: cooking pasta

- Let’s try to optimize some steps

1. Boiling water (5 min)
   - At the same time, make salad (10 min)
2. Cook pasta on water (5 min)
   - At the same time, cook tomato sauce (10 min)

- (Asynchronous) total cooking time = **20 min** (33.3% decrease)

Python async calls

- Library asyncio
- **Coroutines** (async/await)

```
async def mycoroutine(args):
    # function body
    await call()
```

- Coroutines can be **suspended** and **resumed** at given points.
- **await** tells the coordinator (the **event loop**) that execution may be suspended and something else can be done in the meanwhile.
- Coroutines **must be awaited**.

The event loop

- Simply calling a coroutine does not start it.
- Scheduling and subsequent execution can be started with asyncio.run(mycoroutine)
- Event loop starts the execution of a coroutine.
- Coroutine runs until await or return is found.
- Execution is handed over to event loop.
- Event loop decides what to run next, and when.

Multithreading vs. async/await

- We can also implement non-blocking IO with multi-threading
- Main differences:
  - Multi-threading is handled by the OS. Threads are limited in number.
  - Programmer does not have control of suspend/resume
  - More complicated than async/await
- Async/await is a **cooperative** mechanism. Each task must inform the event loop about when it is going to wait (**await**)

Exercise

- Git Repo: https://github.com/IMC-UAS-Krems/DS_Examples
- Open file misc/sync.py in IDE
- Create a new script async.py where the synchronous program is converted to an async/await program using asyncio.

Questions

1. How long did it take to run the async program?
2. How much improvement (in %) could you achieve with respect to the sync program?
3. Why did the running time improve?