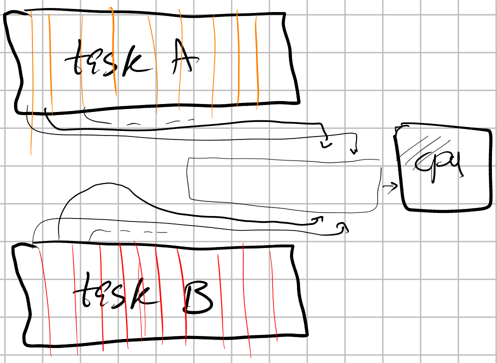
- Async IO

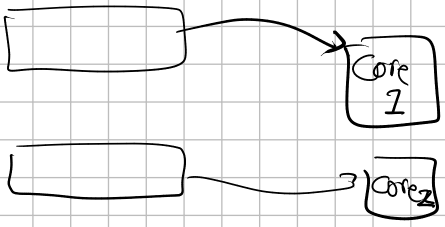
IO - Bound → a lot of waiting on input/output (I/O) to complete

CPU - Bound → computer's cores continually working hard from start to finish.

- multithreading



- multiprocessing



- in multithreading who mange the threads?

$a = 2$
$b = 3 + a$

the os can stop the execution here

what if we (the programmers) mange it instead?

Simultaneous chess exhibit v. Judit Polgár, 1992

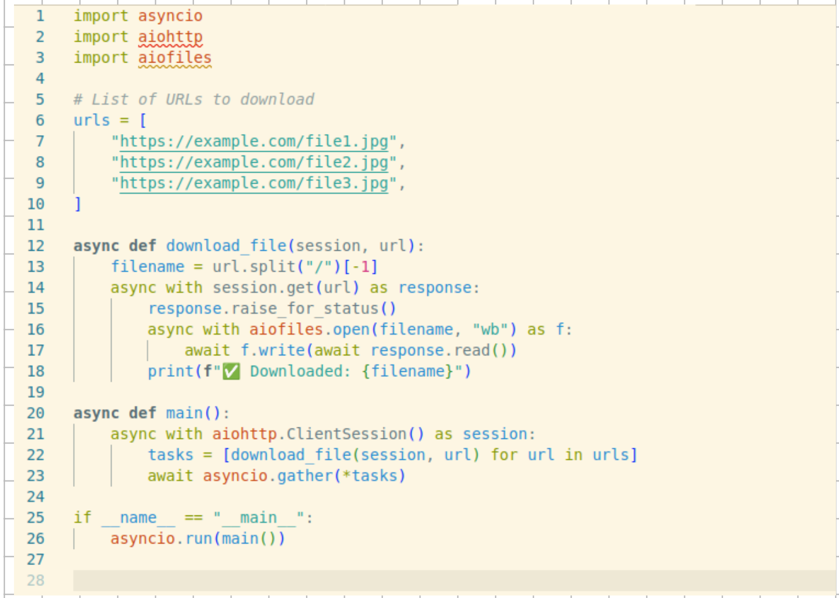
- 4 opponents
- Judit makes each chess move in 5 seconds
- Opponents each take 55 seconds to make a move
- Games average 30 pair-moves (60 moves total)

Synchronous version: Judit plays one game at a time, never two at the same time,
until the game is complete. Each game takes (55 + 5) * 30 == 1800 seconds, or 30 minutes.
The entire exhibition takes 24 * 30 == 720 minutes, or 12 hours.

Asynchronous version: Judit moves from table to table, making one move at each table.
She leaves the table and lets the opponent make their next move during the wait time.
One move on all 24 games takes Judit 24 * 5 == 120 seconds, or 2 minutes. The entire exhibition
is now cut down to 120 * 30 == 3600 seconds, or just 1 hour.

```python
import time

def count():
    print("One")
    time.sleep(1)
    print("Two")
    time.sleep(1)

def main():
    for _ in range(3):
        count()

if __name__ == "__main__":
    start = time.perf_counter()
    main()
    elapsed = time.perf_counter() - start
    print(f"{__file__} executed in {elapsed:0.2f} seconds.")
```

```
$ python countsync.py
One
Two
One
Two
One
Two
countsync.py executed in 6.03 seconds.
```

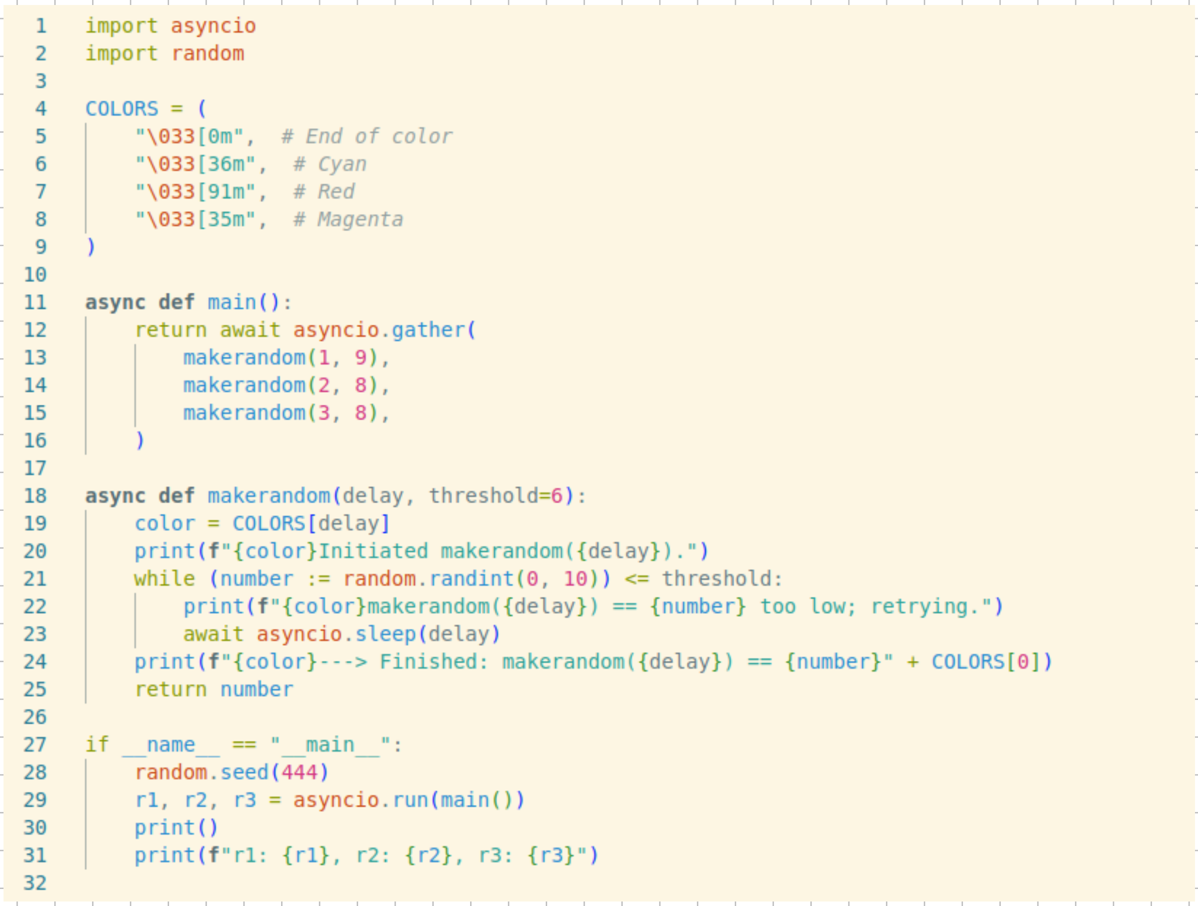


```python
import asyncio
import aiohttp
import aiofiles

# List of URLs to download
urls = [
    "https://example.com/file1.jpg",
    "https://example.com/file2.jpg",
    "https://example.com/file3.jpg",
]

async def download_file(session, url):
    filename = url.split("/")[-1]
    async with session.get(url) as response:
        response.raise_for_status()
        async with aiofiles.open(filename, "wb") as f:
            await f.write(await response.read())
        print(f"✅ Downloaded: {filename}")

async def main():
    async with aiohttp.ClientSession() as session:
        tasks = [download_file(session, url) for url in urls]
        await asyncio.gather(*tasks)

if __name__ == "__main__":
    asyncio.run(main())

```

aiohttp handles async HTTP requests.

aiofiles writes the downloaded content asynchronously.

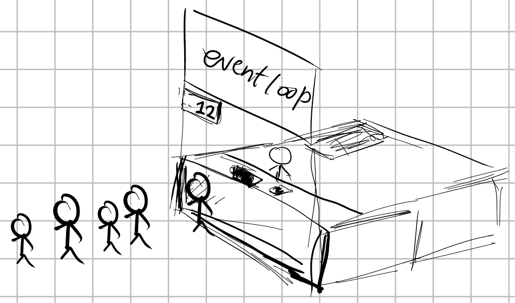
asyncio.gather() runs all download tasks concurrently.

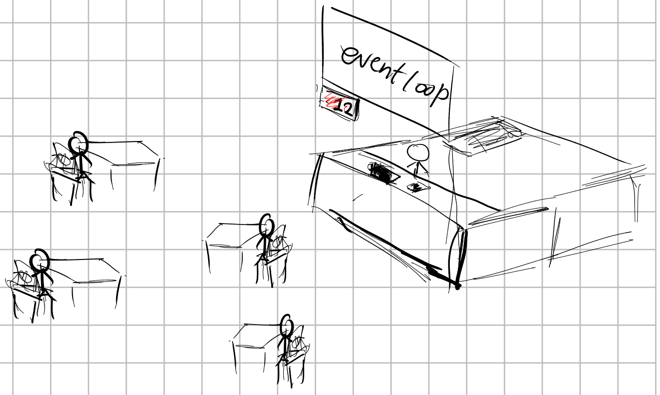


```python
import asyncio
import random

COLORS = (
    "\033[0m",   # End of color
    "\033[36m",  # Cyan
    "\033[91m",  # Red
    "\033[35m",  # Magenta
)

async def main():
    return await asyncio.gather(
        makerandom(1, 9),
        makerandom(2, 8),
        makerandom(3, 8),
    )

async def makerandom(delay, threshold=6):
    color = COLORS[delay]
    print(f"{color}Initiated makerandom({delay}).")
    while (number := random.randint(0, 10)) <= threshold:
        print(f"{color}makerandom({delay}) == {number} too low; retrying.")
        await asyncio.sleep(delay)
    print(f"{color}---> Finished: makerandom({delay}) == {number}" + COLORS[0])
    return number

if __name__ == "__main__":
    random.seed(444)
    r1, r2, r3 = asyncio.run(main())
    print()
    print(f"r1: {r1}, r2: {r2}, r3: {r3}")

```

```python
import asyncio

async def count():
    print("One")
    await asyncio.sleep(1)
    print("Two")
    await asyncio.sleep(1)

async def main():
    await asyncio.gather(count(), count(), count())

if __name__ == "__main__":
    import time

    start = time.perf_counter()
    asyncio.run(main())
    elapsed = time.perf_counter() - start
    print(f"{__file__} executed in {elapsed:0.2f} seconds.")
```

```
$ python countasync.py
One
One
One
Two
Two
Two
countasync.py executed in 2.00 seconds.
```





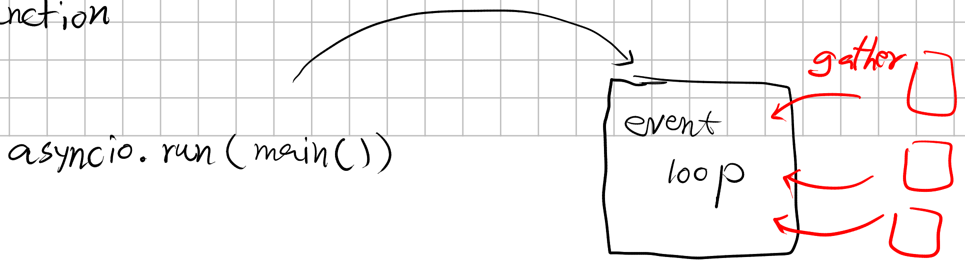
Coroutines

```python
import asyncio

async def count():
    print("One")
    await asyncio.sleep(1)
    print("Two")
    await asyncio.sleep(1)

async def main():
    await asyncio.gather(count(), count(), count())

if __name__ == "__main__":
    import time

    start = time.perf_counter()
    asyncio.run(main())
    elapsed = time.perf_counter() - start
    print(f"{__file__} executed in {elapsed:0.2f} seconds.")
```

wait for the result

start the event loop

☑ Coroutine Function


asyncio. run (main())
event loop
gather