Queue

Очередь — базовая структура данных, позволяющая реализовать принцип работы FIFO (first in first out). Может быть реализована поверх массива или связного списка.

Пример. Есть начальник и 3 подчиненных. Начальник создает задачи, а работники разбирают и выполняют их по мере возможности. Короутины worker-ы master будут обмениваться данными друг с другомчерез **asyncio.Queue**:

```
import asyncio
import random


async def worker(idx: int, queue: asyncio.Queue):
    while True:
        task_id = await queue.get()
        print(f"worker-{idx} started task-{task_id}")
        await asyncio.sleep(random.randint(1, 3))
        print(f"worker-{idx} finished task-{task_id}")


async def master(queue: asyncio.Queue):
    task_id = 1
    while True:
        await asyncio.sleep(1)
        print(f"master gave task-{task_id}")
        await queue.put(task_id)
        task_id += 1


async def main():
    queue = asyncio.Queue()

    coros = [
        master(queue),
        worker(1, queue),
        worker(2, queue),
        worker(3, queue),
    ]

    await asyncio.gather(*coros)


asyncio.run(main())
```

```
master gave task-1
worker-1 started task-1
master gave task-2
worker-1 finished task-1
worker-1 started task-2
master gave task-3
worker-1 finished task-2
worker-1 started task-3
master gave task-4
worker-2 started task-4
```

Мы можем ограничить количество объектов в очереди при ее инициализации. Это может понадобиться, если мы беспокоимся за рост ресурсов, потребляемых нашим приложением.

```
Queue(maxsize=N)
```

Кроме обычной очереди, в библиотеке есть очередь с приоритетом — **PriorityQueue.** Она нужна, если мы хотим обработать какой-то тип событий раньше, чем остальные.

Пример. У нас в очереди много задач и появилась необходимость выполнить что-то прямо сейчас

Если вам интересен механизм работы очереди с приоритетом, вы можете изучить его здесь.