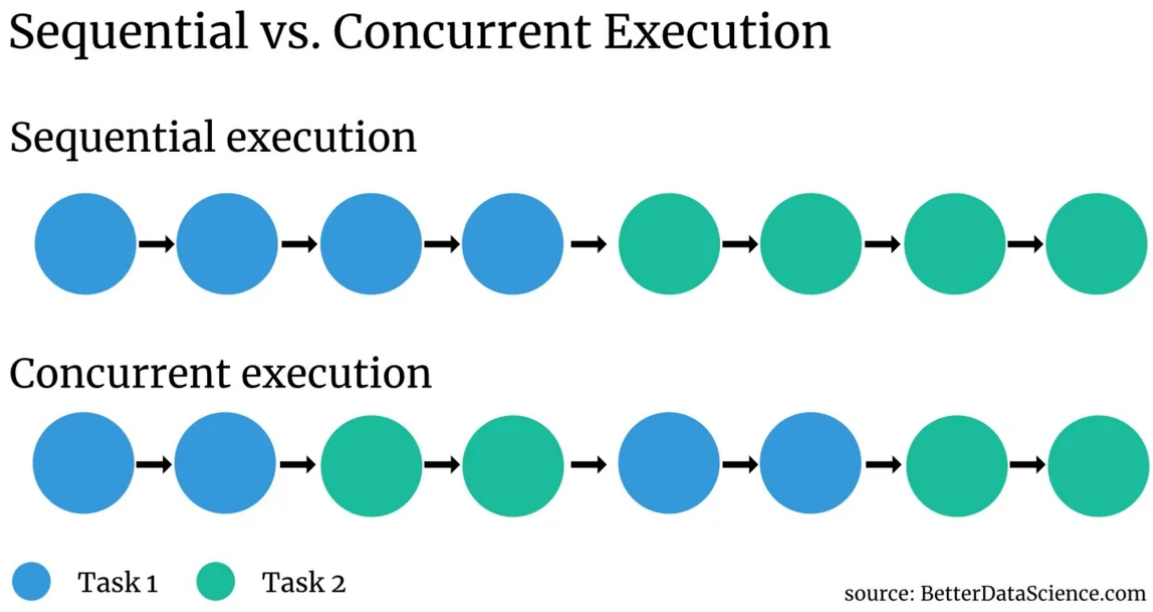
Многозадачность-3: asyncio

Асинхронное программирование

Привет!

В этом уроке мы разберём асинхронное программирование. Представьте, что пишете код в синхронной манере: вызываете функцию, присваиваете значение переменной и печатаете в цикле определённое значение. Цикл блокирует выполнение программы, поэтому необходимо завершить код прежде, чем двигаться дальше. Аналогичная ситуация происходит при десяти последовательных HTTP-запросах. Обработка не начинается даже после того, как они завершаются. Справиться с блокировкой кода можно двумя способами:

- многопоточность,
- параллелизм.



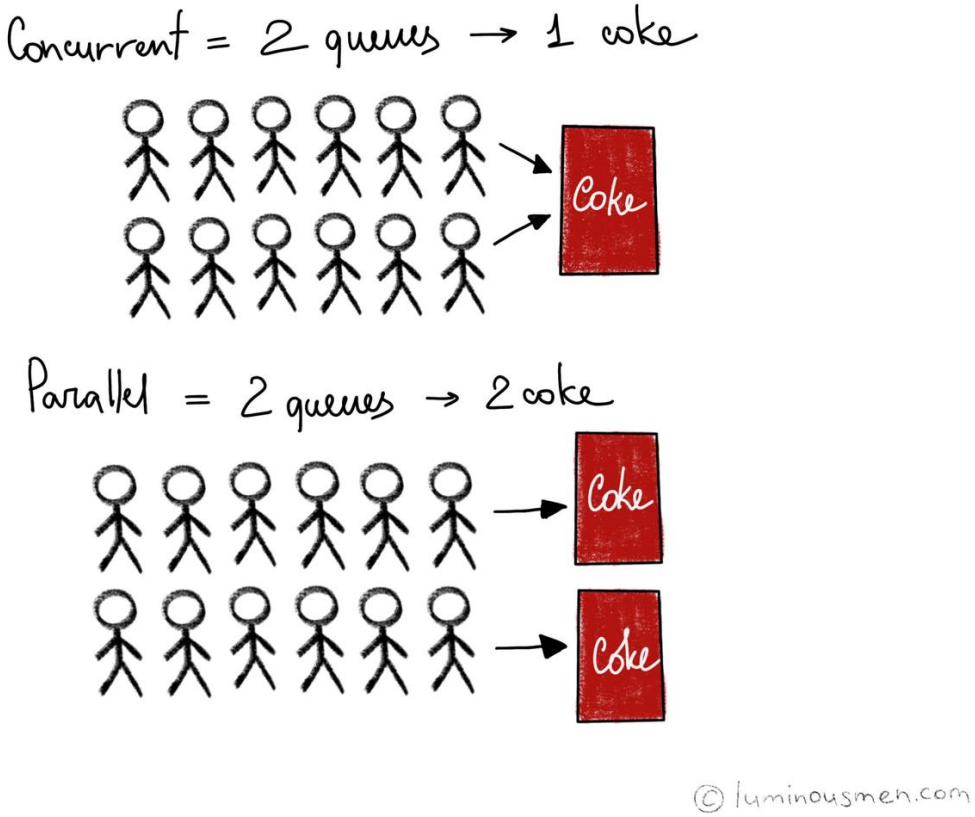
Многопоточность
Создаём десяток тредов для запросов. Переключаемся на следующий тред сразу после того, как он перейдёт к ожиданию ответа от сервера. И так далее.

Этот способ отлично подходит для задачи IO-bound и работы с Python или GIL. Однако в этом способе есть недостатки.

- За управление тредами полностью отвечает планировщик системы. Чтобы приостановить текущий тред и запустить новый, системе нужно совершить сменить контекст. Это ресурсоёмкая операция, которая занимает время.

- С тредами код выглядит монструозно. В нём появляется много join и риск получить deadlock.

Параллелизм
Мы запускаем десять процессов, каждый из которых делает запрос и возвращает что-то. Запросы можно выполнить за один раз или за несколько раз с небольшими задержками в зависимости от количества ядер на процессоре.



В чём сходство этих способов
Запустив первого воркера или набор инструкций, можно не дожидаться конца работы, а перейти к следующей пачке задач. Задача выполняется в асинхронной манере без блокировок.

При асинхронности выполнение одной задачи начинается до окончания работы другой. Запущенная задача не блокируется, а уходит на задний план. Как только одна из таких задач выполнена, внимание переключается на неё.

Многопоточность — это частный случай асинхронности, при котором одновременно выполняется несколько задач. Это может быть лишь видимость, но может — настоящее одновременное выполнение. Во второй ситуации многопоточность превращается в параллелизм, её частный случай.

Многопоточность и параллелизм — асинхронные концепты. При этом сама асинхронность остаётся в стороне.

Зачем нужно асинхронное программирование

- Писать асинхронный код в Python можно, как синхронный с несколькими исключениями.
- Треды легче процессов, но тяжелее нативных объектов питона. Не нужно переключать контекст на процессоре, ведь один тред самостоятельно переходит между задачами. Так производительность для IO-bound задач увеличивается.

Трудности асинхронной работы

Взаимодействовать с памятью в асинхронных приложениях сложнее. Оба треда, воркера или обе корутины могут изменять состояние глобальных объектов конкурентно. Нужно думать о синхронизации между воркерам при использовании тредов и при асинхронном программировании. С процессами работать проще.

	Процесс	Поток	Нативный объект
Память	Много	Меньше	Совсем чуть-чуть
Время переключения между объектами	Долго	Быстрее	Почти мгновенно
Время создания	Долго	Быстрее	Совсем чуть-чуть

В этом уроке вы узнали, что такое асинхронное программирование и какую задачу оно решает. На следующем занятии разберётесь, как писать асинхронный код на Python. Поймёте, почему он превосходит треды.