# Асинхронне програмування

**№ уроку:** 5 **Курс:** Python Advanced

**Засоби навчання:** PyCharm

## Огляд, мета та призначення уроку

Вивчити основи асинхронності, завдання для її застосування. Розібратися з поняттям співпрограми/корутини та ключовими словами async/await. Зрозуміти призначення циклу подій (Event Loop). Розглянути приклади роботи з модулем Asyncio.

## Вивчивши матеріал цього заняття, учень зможе:

- Розуміти загальну схему роботи асинхронності та її особливостей.
- Створювати асинхронні програми, використовуючи async/await/yield from.
- Використовувати Event Loop для запуску власних програм.
- Використовувати модуль asyncio для створення та запуску співпрограм.

## Зміст уроку

1. Основні поняття асинхронності.
2. Співпрограми/корутини та ключові слова async/await.
3. Модуль Asyncio та запуск циклу подій.
4. Запуск співпрограм у циклі подій. Приклади та різні варіанти.
5. Приклади сторонніх бібліотек і фреймворків: aiohttp, gevent і tornado.

## Резюме

У кожній програмі рядки коду виконуються по черзі. Наприклад, якщо у вас є рядок коду, який запитує щось із сервера, то це означає, що ваша програма не робить нічого під час очікування відповіді. У деяких випадках це допустимо, але в багатьох ні. Одним із розв'язків цієї проблеми є потоки (*threads*).

Потоки дають можливість вашій програмі виконувати низку завдань одночасно. У потоків є недоліки: багатопотокові програми є складнішими та схильні до помилок, наявні проблема стану гонитви (*race condition*), взаємне (*deadlock*) й активне (*livelock*) блокування та вичерпання ресурсів (*resource starvation*).

Хоча асинхронне програмування розв'язує проблеми потоків, воно було розроблено з іншою метою – для перемикання контексту процесора. Коли ви маєте кілька потоків, **кожне ядро процесора може запускати тільки один потік за раз**. Щоби всі потоки/процеси могли спільно використовувати ресурси, процесор дуже часто перемикає контекст. Щоби спростити роботу, процесор із довільною періодичністю зберігає всю контекстну інформацію потоку та перемикається на інший потік.

**Асинхронне програмування** – це потокове оброблення програмного забезпечення/користувацького простору, де застосунок, а не процесор, управляє потоками та перемиканням контексту. В асинхронному програмуванні контекст перемикається лише в заданих точках перемикання, а не з періодичністю, яка визначена CPU.

Уявіть секретаря, який не марнує час. Він має п'ять завдань, які виконує одночасно: відповідає на телефонні дзвінки, приймає відвідувачів, намагається забронювати квитки на літак, контролює графіки зустрічей та заповнює документи. Тепер уявіть, що такі завдання, як-от контроль графіків зустрічей, приймання телефонних дзвінків і відвідувачів, повторюються не часто та розподілені в часі. Велику частину часу секретар розмовляє телефоном з авіакомпанією, водночас заповнюючи документи. Коли надійде телефонний дзвінок, він поставить розмову з авіакомпанією на паузу, відповість на дзвінок, а потім повернеться до розмови з авіакомпанією. У будь-який час, коли нове завдання вимагатиме уваги



CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

секретаря, заповнення документів буде відкладено, оскільки воно не є критичним. Секретар, який виконує кілька завдань одночасно, перемикає контекст у потрібний час. Він **асинхронний**.

Для запуску асинхронних програм є спеціальний пул завдань, куди можна складати співпрограми. Вони будуть послідовно перемикатися між собою. Такий пул завдань називається циклом подій (EventLoop). С співпрограми призначені для запуску в циклі подій.

Починаючи з версії 3.5, до Python додали нові ключові слова async/await, які повністю замінюють yield/fr om. Однак для сумісності був доданий декоратор asyncio.coroutine, який можна використовувати якраз д ля підтримки yield/from.

**async/await**

Бібліотека Asyncio є досить потужною, тому Python-спільнота вирішила зробити її стандартною бібліотек ою. До синтаксису також додали ключове слово async. Ключові слова призначені для чіткішого позначен ня асинхронного коду. Тому тепер методи не плутаються із генераторами (yield/from). Ключове слово asy nc йде до def, щоби показати, що метод є асинхронним. Ключове слово await показує, що ви очікуєте зав ершення співпрограми. Приклад ключовими словами async/await:import asyncio

```
async def async_worker(number, divider):
 """
 замість yield/yield from використовуємо синтаксис async/await.
 """
 print('Worker {} started'.format(number))
 await asyncio.sleep(2)
 print(number / divider)
 return number / divider

async def gather_worker():
# виконання декількох завдань та отримання результату від кожної з них

# у результаті виконання ми отримаємо список результатів у тому ж порядку,

# у якому передавали співпрограми.


 result = await asyncio.gather(
 async_worker(50, 10),
 async_worker(60, 10),
 async_worker(70, 10),
 async_worker(80, 10),
 async_worker(90, 10),
 )
 print(result)

event_loop = asyncio.get_event_loop()
task_list = [
 # async_worker(30, 10),
 # asyncio.ensure_future(async_worker(30, 10)),
 event_loop.create_task(gather_worker())
]
tasks = asyncio.wait(task_list)
event_loop.run_until_complete(tasks)
event_loop.close()
```

**Бібліотека Asyncio** – це не єдиний спосіб писати асинхронні програми: можна використовувати функції зворотного виклику (tornado) або зелені потоки (gevent).

## Закріплення матеріалу

- Що таке асинхронність?
- Хто перемикає керування між завданнями в багатопотоковому програмуванні?
- Хто перемикає керування між завданнями в співпрограмах?
- Як GIL впливає на роботу асинхронних програм?
- Що таке співпрограма та що пов'язує її з генераторами?
- Що таке цикл подій, навіщо він потрібний?
- Навіщо потрібне ключове слово async?
- Навіщо потрібне ключове слово await?
- Починаючи з якої версії мови Python він став підтримувати в синтаксисі слова async і await?
- Яку співпрограму можна використовувати, щоби заснути на N секунд і перейти до іншого завдання?
- Навіщо потрібна бібліотека aiohttp?

## Самостійна діяльність учня

Завдання 1
Створіть співпрограму, яка отримує контент із зазначених посилань і логує хід виконання в спеціальний файл, використовуючи стандартну бібліотеку urllib, а потім проробіть те саме з бібліотекою aiohttp. Кроки, які мають бути залоговані: початок запиту до адреси X, відповідь для адреси X отримано зі статусом 200. Перевірте хід виконання програми на > 3 ресурсах і перегляньте послідовність запису логів в обох варіантах і порівняйте результати. Для двох видів завдань використовуйте різні файли для логування, щоби порівняти отриманий результат.

Завдання 2
Розробіть сокет-сервер на основі бібліотеки asyncio.

## Рекомендовані ресурси

Офіційний сайт Python – asyncio
https://docs.python.org/3.11/library/asyncio.html
Офіційний сайт AIOHHTTP
https://aiohttp.readthedocs.io/en/stable/



CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com