13. Функционалдық ағындар

- Операциялардың асинхронды орындалуын қолдайтын деректер ағындарымен

жұмыс істеуге арналған кітапхананы енгізу.

Асинхронные операции - это операции, которые могут выполняться независимо от основного потока выполнения программы. В отличие от синхронных операций, которые выполняются последовательно и блокируют основной поток, асинхронные операции позволяют продолжать выполнение других задач во время ожидания завершения операции. Это позволяет улучшить отзывчивость приложения и эффективно использовать ресурсы. Асинхронные операции часто используются в сетевом вводе-выводе, обработке событий и параллельных вычислениях. В языке программирования Python асинхронные операции обычно реализуются с использованием асинхронного программирования и механизмов, таких как asyncio.

Асинхронды операциялар-бұл бағдарламаның негізгі орындалу ағынына тәуелсіз орындалатын операциялар. Дәйекті түрде орындалатын және негізгі ағынды блоктайтын синхронды операциялардан айырмашылығы, асинхронды операциялар операцияның аяқталуын күту кезінде басқа тапсырмаларды жалғастыруға мүмкіндік береді. Бұл қолданбаның жауаптылығын жақсартуға және ресурстарды тиімді пайдалануға мүмкіндік береді. Асинхронды операциялар көбінесе желілік енгізу-шығару, оқиғаларды өңдеу және параллельді есептеулерде қолданылады. Python бағдарламалау тілінде асинхронды операциялар әдетте асинхронды бағдарламалау және asyncio сияқты механизмдер арқылы жүзеге асырылады.

import asyncio #модуль

# Функция для удаления знаков препинания из текста

async def remove\_punctuation(text): #текстты аргумент ретынде каблдайт

await asyncio.sleep(1) # функцияны 1 секка задерживает чтоб имитацию асинъ функц

return text.translate(str.maketrans('', '', string.punctuation))

#текстты кайтарады транслэйт методы аркыыл

# Функция для подсчета слов в тексте

async def count\_words(text):

await asyncio.sleep(1) # Эмуляция асинхронной операции

return len(text.split()) #болинип турган соз санын кайтарады методом сплит

# Функция для обращения порядка слов в тексте

async def reverse\_words(text):

await asyncio.sleep(1) # Эмуляция асинхронной операции

return ' '.join(text.split()[::-1]) #кері ретпен турган создерді кайтарады

# бул функция устидеги 3 функты текстке последовательно колданады

async def process\_text(text, functions):

for function in functions: # functions тагы арбир функция ушин

text = await function(text) #ағымдағы функты await комегімен текстке колданамыз, онын аякталуын куте отыра

return text #текстты кайтьарамыз

# Пример использования

async def main():

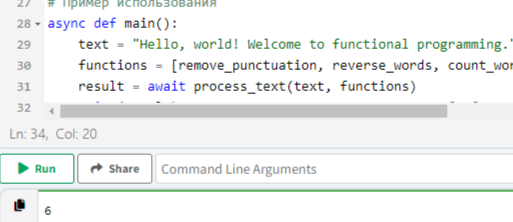
text = "Hello, world! Welcome to functional programming."

functions = [remove\_punctuation, reverse\_words, count\_words]

result = await process\_text(text, functions)

print(result) # Выводит количество слов после обработки

asyncio.run(main())



Этот код демонстрирует асинхронную обработку текста с использованием библиотеки asyncio в Python. Давайте разберем его по строкам:

1. `import asyncio`: Эта строка импортирует модуль asyncio, который предоставляет инструменты для написания асинхронного кода.

2. Определение асинхронной функции `remove\_punctuation(text)`: Эта функция удаляет знаки препинания из текста. Она асинхронная, что означает, что может быть приостановлена, пока выполняется асинхронная операция, такая как `asyncio.sleep(1)`, которая эмулирует задержку в 1 секунду.

3. Определение асинхронной функции `count\_words(text)`: Эта функция подсчитывает количество слов в тексте. Она также асинхронная и использует `asyncio.sleep(1)`.

4. Определение асинхронной функции `reverse\_words(text)`: Эта функция обращает порядок слов в тексте. Она также является асинхронной и использует `asyncio.sleep(1)`.

5. Определение асинхронной функции `process\_text(text, functions)`: Эта функция принимает текст и список функций обработки текста и последовательно применяет эти функции к тексту. Она также является асинхронной.

6. `async def main()`: Эта асинхронная функция представляет точку входа в программу. Внутри нее создается текст, определяются функции обработки текста и вызывается функция `process\_text` для обработки текста с использованием этих функций.

7. `asyncio.run(main())`: Это основной вызов, который запускает асинхронную функцию `main()` и управляет циклом выполнения asyncio.

Когда этот код запущен, он обрабатывает текст с помощью функций обработки текста (удаление знаков препинания, обращение порядка слов и подсчет слов) и выводит результат (количество слов после обработки).

**7-зертханалық жұмыс: Функционалдық принциптерді қолдану арқылы модульдік код құру**

**1.** Бағдарламалаудағы модульдік дегеніміз не және ол қандай артықшылықтар

береді?

Бағдарламалаудағы модульдік-бұл бағдарлама жеке компоненттерге (модульдерге) бөлінетін тәсіл, олардың әрқайсысы белгілі бір функционалдылыққа жауап береді. Модульділіктің артықшылықтарына мыналар жатады:

Модульдік бағдарламалау бағдарламаны жеке компоненттерге (модульдерге) бөлуге мүмкіндік береді, бұл кодты қолдауды, қайта пайдалануды және тексеруді жеңілдетеді. Бұл оқуды, масштабтауды жақсартады және жобаны жақсырақ ұйымдастыруды қамтамасыз етеді.

**2.** Python-да функция құрамын қалай жүзеге асырасыз және ол қандай есептер үшін қолайлы?

Python-да функцияны "def" кілт сөзі арқылы анықтауға болады, содан кейін функция Атауы және жақшадағы параметрлер тізімі. Ол кодты ұйымдастыруға құрылымдық және қайта пайдалануға болатын тәсілді қамтамасыз етеді, сонымен қатар кодты бірнеше рет шақыруға болатын жеке блоктарға орау мүмкіндігінің арқасында әртүрлі тапсырмаларды шешуді жеңілдетеді.`.

**3**. Зертханада қандай асинхронды бағдарламалау әдістерін қолдандыңыз?

- Мақсаты: Студенттердің асинхронды бағдарламалаудың әртүрлі тәсілдері және

оларды пайдалану туралы түсінігі.

Зертханада мен келесі асинхронды бағдарламалау әдістерін қолдандым:

(1. Python-дағы асинхронды функциялар (async/await): бұл тәсіл параллель орындалуы мүмкін асинхронды функцияларды құруға мүмкіндік береді және басқа асинхронды операциялардың нәтижелерін бұғаттаусыз күтеді.

*Сіз қазіргі уақытта кодта асинхронды бағдарламалауды қолданасыз. Бұл кодта "async io" модулінің көмегімен асинхронды операциялардың эмуляциясы жүреді. Асинхронды функциялар мәтінді өңдеу, тексерулер жүргізу және асинхронды операцияларды орындау үшін қолданылады. 'Async io функциясы.run (main ())` басқа асинхронды функцияларды орындау үшін негізгі асинхронды функцияны іске қосады.*

**4**. Асинхронды бағдарламалау Python тіліндегі функционалдық стильмен қалай

интеграцияланады?

Python-дағы асинхронды бағдарламалау деректерді өңдеу үшін жоғары деңгейлі функциялар мен генераторлар сияқты функционалды компоненттерді қолдану арқылы функционалды стильмен біріктіріледі. Бұл компоненттер асинхронды операцияларды талғампаздықпен өңдеуге және асинхронды қосымшалардағы мәліметтер ағынын басқаруға мүмкіндік береді. `Functools` және `tools` сияқты функционалды бағдарламалауға арналған кітапханалар деректерді өңдеуді жеңілдету үшін асинхронды қосымшаларда да қолданыла алады. Сонымен қатар, асинхронды бағдарламалауды функционалды композиция және каррирлеу сияқты функционалды шаблондармен біріктіру модульдік және икемді асинхронды қосымшаларды құруға мүмкіндік береді.

5. Модульдік бағдарламалауда қандай мәселелер туындауы мүмкін және оларды

шешу жолы?

Модульдік бағдарламалауда Модульдер арасындағы тәуелділікке, қолдау мен масштабтаудың күрделілігіне қатысты мәселелер туындауы мүмкін. Бұл мәселелерді модуль интерфейстерін нақты анықтау, қателерді анықтау үшін модульдік сынақтарды пайдалану және қолдау мен кеңейтуді жеңілдету үшін кодты құрылымдау арқылы шешуге болады.

**6**. Функция құрамының кодты қайта пайдалану мүмкіндігін арттырудағы рөлі

қандай?

- Мақсаты: Оқушылар модульдік және қайта пайдалануға болатын кодты құру

үшін функция құрамының маңыздылығын түсінеді.

Кодты қайта пайдалану мүмкіндігін арттырудағы функция құрамының рөлі келесідей:

Состав функции играет ключевую роль в повышении возможности повторного использования кода, поскольку он позволяет упаковать определенную логику или операцию в отдельную именованную единицу. Это способствует модульности кода, позволяя использовать функцию в разных частях программы или даже в различных проектах без необходимости переписывать ее с нуля. Кроме того, использование параметров в составе функции позволяет настраивать ее поведение для конкретных потребностей, делая код более гибким и адаптируемым. Такой подход к разработке программ облегчает поддержку и обновление кода, а также сокращает время разработки нового функционала.

**7**. Асинхронды бағдарламалау практикалық емес мысал келтіре аласыз ба?

Конечно, вот еще один пример: создание асинхронного микросервиса для обработки запросов пользователей в реальном времени. В таком микро сервисе асинхронные операции позволяют эффективно управлять множеством запросов одновременно без необходимости создания большого количества потоков или процессов. Это особенно полезно при работе с сетевыми приложениями, где непредсказуемая нагрузка может привести к блокировке потоков, если используется синхронный подход. Асинхронное программирование также облегчает интеграцию с другими сервисами, такими как базы данных или внешние API, поскольку не блокирует основной поток выполнения приложения, что позволяет эффективно использовать ресурсы и обрабатывать большие объемы данных.

**8**. Асинхронды қолданбаларда күйді қалай басқарасыз?

- Мақсаты: Асинхронды және функционалдық жүйелердегі мемлекеттік басқару

әдістерін білуді бағалау.

Асинхронды қосымшалардағы күй күй объектілері, оқиғалар және асинхронды хабарлама кезектері сияқты арнайы механизмдер арқылы басқарылады. Python-да asyncio сияқты құралдар күйді басқару үшін жиі қолданылады.

**9**. Функционалдық үлгілер дегеніміз не және олар сіздің зертханалық

жұмысыңызда қалай қолданылды?

- Мақсаты: Студенттер бағдарламалық жасақтаманы әзірлеуде функционалдық

үлгілерді пайдалануды түсінеді.

Функционалды үлгілер-бұл белгілі бір мәселелерді шешуге арналған типтік құрылымдарды немесе шешімдерді сипаттайтын дизайн үлгілері. Оларды қателерді өңдеу, деректерді манипуляциялау немесе кодты ұйымдастыру сияқты әртүрлі мақсаттарда пайдалануға болады. Біздің жұмысымызда біз кодты әзірлеу процесін жеңілдету және стандарттау, оқуды жақсарту және техникалық қызмет көрсету үшін функционалды үлгілерді қолдандық.

Функциональные уровни представляют собой компоненты программного кода, которые выполняют определенные задачи или операции. В данном примере использованы асинхронные функции для обработки текста, такие как удаление знаков препинания, подсчет слов и обращение порядка слов в тексте. Эти функции объединяются в единый процесс обработки текста с помощью функции `process\_text`, которая последовательно применяет каждую функцию из списка к исходному тексту. Результат выводится на экран в конце процесса. Такой подход демонстрирует использование функционального программирования для организации обработки данных в асинхронном режиме, что может быть полезно в приложениях, требующих асинхронной обработки текстовой информации, например, веб-скрапинг, обработка больших объемов данных и др.

**10**. Қандай Python құралдары мен кітапханалары асинхронды бағдарламалауға

қолдау көрсетеді?

- Мақсаты: Python тілінде асинхронды бағдарламалау үшін қолжетімді құралдар

туралы студенттердің білімін тексеру.

Асинхронды бағдарламалауға арналған кейбір Python құралдары мен кітапханаларына asyncio, aiohttp, aiofiles және asyncpg кіреді. Бұл құралдар асинхронды қосымшаларды тиімді құруға және асинхронды енгізу-шығару операцияларын өңдеуге мүмкіндік беретін Python-да асинхронды операцияларды қолдайды.

**11**. Модульдік тәсіл тестілеу және жөндеу бағдарламаларына қалай әсер етеді?

- Мақсаты: Модульдіктің тестілеу және жөндеу процестеріне әсерін түсіну.

Модульный подход улучшает программы тестирования и отладки, поскольку позволяет изолировать отдельные компоненты приложения для тестирования, что делает процесс более прозрачным, уменьшает зависимости и упрощает обнаружение и устранение ошибок.

Модульдік тәсіл тестілеу және күйін келтіру бағдарламаларын жақсартады, өйткені ол тестілеу үшін қосымшаның жеке компоненттерін оқшаулауға мүмкіндік береді, бұл процесті мөлдір етеді, тәуелділікті азайтады және қателерді табуды және жоюды жеңілдетеді.

**12**. Функционалдық стильде асинхронияны жүзеге асыру кезінде қандай

қиындықтарға тап болдыңыз?

Функционалды стильде асинхронды бағдарламалауды жүзеге асыруда белгілі бір қиындықтар туындайды. Біріншіден, функционалдық принциптер функциялардың тазалығын және жанама әсерлердің болмауын болжайды, бұл асинхронды операциялармен жұмыс істеу кезінде байқау қиын болуы мүмкін, өйткені олар көбінесе күйге ие және сыртқы ресурстармен әрекеттесе алады. Екіншіден, асинхронды функциялар кері байланыс тудыруы немесе корутиндерді қолдануы мүмкін, бұл бағдарламаның логикасын қиындатып, оны аз оқуға мүмкіндік береді. Сонымен қатар, асинхронды операцияларды басқару қателерді өңдеуге және ресурстарды басқаруға ерекше назар аударуды қажет етеді, бұл кодтың күрделілігінің артуына және оны қолдаудың қиындауына әкелуі мүмкін.

**13**. Таза асинхронды кодты жазуға функционалдық бағдарламалау қалай ықпал

ететінін түсіндіре аласыз ба?

Асинхронды кодтағы функционалды бағдарламалау кодтың таза және құрылымдалған ұйымдастырылуына ықпал етеді. Жаһандық күйді пайдаланудың және айнымалыларды тікелей өзгертудің орнына функциялар деректерді параметрлер мен қайтарылатын мәндер арқылы басқарады. Бұл кодты жақсы оқуға және түсінуге ықпал етеді. Сонымен қатар, функционалды тәсіл жанама әсерлерді азайтуға мүмкіндік береді, бұл кодты болжамды және оңай тексеруге мүмкіндік береді. Нәтижесінде, асинхронды кодтағы функционалды бағдарламалау неғұрлым сенімді және қолдау көрсетілетін қосымшаларды құруға ықпал етеді.

**14**. Асинхронды функцияларды құру кезінде қандай негізгі шақырулар болды?

- Мақсаты: Асинхронды функция құрамымен байланысты қиындықтарды және олардың шешімдерін түсіну.

Основные вызовы при создании асинхронных функций включают управление состоянием, координацию параллельных операций, обработку ошибок и решение проблем синхронизации. Асинхронды функцияларды құрудағы негізгі қоңырауларға күйді басқару, параллель операцияларды үйлестіру, қателерді өңдеу және синхрондау мәселелерін шешу кіреді.

**15**. Модульдік асинхронды қолданбаның архитектурасын қалай жақсартуға болады?

Для улучшения архитектуры модульного асинхронного приложения можно использовать четкие интерфейсы между модулями, разделение ответственностей, уменьшение зависимостей между компонентами, а также применение шаблонов проектирования, таких как Dependency Injection и Observer.